

PostGIS 3.7.0rc2 手册

Contents

1	介绍	1
1.1	What Is PostGIS?	1
1.2	What Is in the PostGIS Distribution?	1
1.3	The PostGIS Project Family	2
1.4	How to Use This Manual	2
1.5	Standards Support	3
1.6	Project and Community	3
2	PostGIS 安装	4
2.1	简短版本	4
2.2	从源代码和安装	4
2.2.1	取源代码	5
2.2.2	安装要求	5
2.2.3	建配置	7
2.2.4	建	8
2.2.5	建和部署 PostGIS 展	8
2.2.6		10
2.2.7	安装	13
2.3	常	13
3	PostGIS 管理	15
3.1	性能	15
3.1.1		15
3.1.2	行	16
3.2	配置格支持	16
3.3	建空数据	17
3.3.1	使用展 (EXTENSION) 用空数据	17
3.3.2	不通展 (EXTENSION) 的方式用空数据 (不建)	17
3.4	升空数据	18
3.4.1	升	18
3.4.1.1	使用 EXTENSION 升	18
3.4.1.2	不使用 EXTENSION 升	20
3.4.1.3	硬升	20

4 数据管理	22
4.1 空几何数据模型	22
4.1.1 OGC 几何类型	22
4.1.1.1 点 (Point)	23
4.1.1.2 线串 (LineString)	23
4.1.1.3 线性环 (LinearRing)	23
4.1.1.4 多边形 (Polygon)	23
4.1.1.5 多点 (MultiPoint)	23
4.1.1.6 多线串 (MultiLineString)	23
4.1.1.7 多面 (MultiPolygon)	24
4.1.1.8 几何集合 (GeometryCollection)	24
4.1.1.9 多面体曲面 (PolyhedralSurface)	24
4.1.1.10 三角形 (Triangle)	24
4.1.1.11 TIN	24
4.1.2 SQL/MM 第 3 部分-曲线	25
4.1.2.1 圆弧串 (CircularString)	25
4.1.2.2 复合曲线 (CompoundCurve)	25
4.1.2.3 曲线多边形 (CurvePolygon)	25
4.1.2.4 多曲线 (MultiCurve)	26
4.1.2.5 多曲面 (MultiSurface)	26
4.1.3 WKT 和 WKB	26
4.2 几何数据类型	27
4.2.1 PostGIS EWKB 和 EWKT	28
4.3 地理数据类型	29
4.3.1 创建地理表	29
4.3.2 使用地理表	31
4.3.3 如何使用地理数据类型	32
4.3.4 地理高程常见问题解答	33
4.4 几何有效性	33
4.4.1 几何	33
4.4.2 有效的几何形状	35
4.4.3 有效性管理	40
4.5 空参考系	42
4.5.1 SPATIAL_REF_SYS 表	43
4.5.2 使用自定义空参考系	44
4.6 空表	44
4.6.1 创建空表	44
4.6.2 GEOMETRY_COLUMNS 表	45
4.6.3 手动注册几何列	46

4.7	加空数据	48
4.7.1	使用 SQL 加数据	49
4.7.2	使用 Shapefile 加器	49
4.8	提取空数据	51
4.8.1	使用 SQL 提取数据	51
4.8.2	使用 Shapefile 文件程序	52
4.9	空索引	53
4.9.1	GiST 索引	53
4.9.2	BRIN 索引	54
4.9.3	SP-GiST 索引	55
4.9.4	索引的使用	56
5	空	57
5.1	空关系的确定	57
5.1.1	度展九交模型 (Dimensionally Extended 9-Intersection Model)	57
5.1.2	命名空关系	60
5.1.3	一般空关系	61
5.2	使用空索引	63
5.3	空 SQL 示例	64
6	性能化技巧	67
6.1	大几何形状的小表	67
6.1.1	描述	67
6.1.2	解决法	67
6.2	几何索引聚	68
6.3	避免度	68
7	PostGIS 参考手册	69
7.1	PostGIS Geometry/Geography/Box 数据类型	69
7.1.1	box2d	69
7.1.2	box3d	70
7.1.3	geometry	70
7.1.4	geometry_dump	71
7.1.5	geography	71
7.2	表管理功能	71
7.2.1	AddGeometryColumn	71
7.2.2	DropGeometryColumn	74
7.2.3	DropGeometryTable	75
7.2.4	Find_SRID	76
7.2.5	Populate_Geometry_Columns	76

7.2.6	UpdateGeometrySRID	78
7.3	几何构造函数	79
7.3.1	ST_Collect	79
7.3.2	ST_LineFromMultiPoint	83
7.3.3	ST_MakeEnvelope	84
7.3.4	ST_MakeLine	85
7.3.5	ST_MakePoint	89
7.3.6	ST_MakePointM	91
7.3.7	ST_MakePolygon	93
7.3.8	ST_Point	97
7.3.9	ST_PointZ	98
7.3.10	ST_PointM	99
7.3.11	ST_PointZM	99
7.3.12	ST_Polygon	100
7.3.13	ST_TileEnvelope	102
7.3.14	ST_HexagonGrid	104
7.3.15	ST_Hexagon	109
7.3.16	ST_SquareGrid	111
7.3.17	ST_Square	112
7.3.18	ST_Letters	113
7.3.19	ST_MakeNurbsCurve	115
7.4	几何函数器	118
7.4.1	GeometryType	118
7.4.2	ST_Boundary	120
7.4.3	ST_BoundingDiagonal	126
7.4.4	ST_CoordDim	127
7.4.5	ST_Dimension	128
7.4.6	ST_Dump	130
7.4.7	ST_DumpPoints	134
7.4.8	ST_DumpSegments	139
7.4.9	ST_DumpRings	142
7.4.10	ST_EndPoint	144
7.4.11	ST_Envelope	148
7.4.12	ST_ExteriorRing	152
7.4.13	ST_GeometryN	155
7.4.14	ST_GeometryType	159
7.4.15	ST_HasArc	160
7.4.16	ST_InteriorRingN	161
7.4.17	ST_NumCurves	163

7.4.18ST_CurveN	164
7.4.19ST_IsClosed	166
7.4.20ST_IsCollection	170
7.4.21ST_IsEmpty	172
7.4.22ST_IsPolygonCCW	174
7.4.23ST_IsPolygonCW	175
7.4.24ST_IsRing	176
7.4.25ST_IsSimple	178
7.4.26ST_M	180
7.4.27ST_MemSize	181
7.4.28ST_NDims	182
7.4.29ST_NPoints	183
7.4.30ST_NRings	184
7.4.31ST_NumGeometries	185
7.4.32ST_NumInteriorRings	186
7.4.33ST_NumInteriorRing	187
7.4.34ST_NumPatches	188
7.4.35ST_NumPoints	188
7.4.36ST_PatchN	189
7.4.37ST_PointN	191
7.4.38ST_Points	194
7.4.39ST_StartPoint	195
7.4.40ST_StartM	200
7.4.41ST_EndM	203
7.4.42ST_SetStartM	206
7.4.43ST_SetEndM	209
7.4.44ST_Summary	212
7.4.45ST_X	214
7.4.46ST_Y	214
7.4.47ST_Z	215
7.4.48ST_ZMFlag	216
7.4.49ST_HasZ	217
7.4.50ST_HasM	218
7.4.51ST_ControlPoints	220
7.4.52ST_Degree	221
7.4.53ST_Weights	223
7.4.54ST_Knots	225
7.4.55ST_NumControlPoints	227
7.4.56ST_NurbsCurveIsRational	228

7.4.57	ST_IsNurbsCurve	230
7.4.58	ST_Evaluate	232
7.4.59	ST_NurbsToLineString	233
7.5	几何函数	235
7.5.1	ST_AddPoint	235
7.5.2	ST_CollectionExtract	237
7.5.3	ST_CollectionHomogenize	239
7.5.4	ST_CurveToLine	243
7.5.5	ST_Scroll	247
7.5.6	ST_FlipCoordinates	248
7.5.7	ST_Force2D	249
7.5.8	ST_Force3D	251
7.5.9	ST_Force3DZ	253
7.5.10	ST_Force3DM	255
7.5.11	ST_Force4D	257
7.5.12	ST_ForceCollection	259
7.5.13	ST_ForceCurve	262
7.5.14	ST_ForcePolygonCCW	263
7.5.15	ST_ForcePolygonCW	264
7.5.16	ST_ForceSFS	266
7.5.17	ST_ForceRHR	267
7.5.18	ST_LineExtend	268
7.5.19	ST_LineToCurve	269
7.5.20	ST_Multi	271
7.5.21	ST_Normalize	272
7.5.22	ST_Project	273
7.5.23	ST_QuantizeCoordinates	275
7.5.24	ST_RemovePoint	277
7.5.25	ST_RemoveRepeatedPoints	277
7.5.26	ST_RemoveIrrelevantPointsForView	281
7.5.27	ST_RemoveSmallParts	285
7.5.28	ST_Reverse	287
7.5.29	ST_Segmentize	288
7.5.30	ST_SetPoint	291
7.5.31	ST_ShiftLongitude	294
7.5.32	ST_WrapX	296
7.5.33	ST_SnapToGrid	297
7.5.34	ST_Snap	300
7.5.35	ST_SwapOrdinates	304

7.6 几何有效性	305
7.6.1 ST_IsValid	305
7.6.2 ST_IsValidDetail	308
7.6.3 ST_IsValidReason	310
7.6.4 ST_MakeValid	312
7.7 空参考系功能	316
7.7.1 ST_InverseTransformPipeline	316
7.7.2 ST_SetSRID	318
7.7.3 ST_SRID	320
7.7.4 ST_Transform	320
7.7.5 ST_TransformPipeline	325
7.7.6 postgis_srs_codes	328
7.7.7 postgis_srs	329
7.7.8 postgis_srs_all	330
7.7.9 postgis_srs_search	330
7.8 几何入	332
7.8.1 已知文本 (WKT)	332
7.8.1.1 ST_BdPolyFromText	332
7.8.1.2 ST_BdMPolyFromText	332
7.8.1.3 ST_GeogFromText	333
7.8.1.4 ST_GeographyFromText	334
7.8.1.5 ST_GeomCollFromText	335
7.8.1.6 ST_GeomFromEWKT	335
7.8.1.7 ST_GeomFromMARC21	338
7.8.1.8 ST_GeometryFromText	342
7.8.1.9 ST_GeomFromText	343
7.8.1.10 ST_LineFromText	348
7.8.1.11 ST_MLineFromText	349
7.8.1.12 ST_MPointFromText	350
7.8.1.13 ST_MPolyFromText	350
7.8.1.14 ST_PointFromText	353
7.8.1.15 ST_PolygonFromText	355
7.8.1.16 ST_WKTToSQL	356
7.8.2 已知的二进制文件 (WKB)	357
7.8.2.1 ST_GeogFromWKB	357
7.8.2.2 ST_GeomFromEWKB	358
7.8.2.3 ST_GeomFromWKB	360
7.8.2.4 ST_LineFromWKB	362
7.8.2.5 ST_LinestringFromWKB	363

7.8.2.6	ST_PointFromWKB	365
7.8.2.7	ST_WKBToSQL	366
7.8.3	其它格式	367
7.8.3.1	ST_Box2dFromGeoHash	367
7.8.3.2	ST_GeomFromGeoHash	368
7.8.3.3	ST_GeomFromGML	369
7.8.3.4	ST_GeomFromGeoJSON	374
7.8.3.5	ST_GeomFromKML	376
7.8.3.6	ST_GeomFromTWKB	377
7.8.3.7	ST_GMLToSQL	379
7.8.3.8	ST_LineFromEncodedPolyline	379
7.8.3.9	ST_PointFromGeoHash	381
7.8.3.10	ST_FromFlatGeobufToTable	383
7.8.3.11	ST_FromFlatGeobuf	384
7.9	几何输出	384
7.9.1	已知文本 (WKT)	384
7.9.1.1	ST_AsEWKT	384
7.9.1.2	ST_AsText	386
7.9.2	已知的二进制文件 (WKB)	388
7.9.2.1	ST_AsBinary	388
7.9.2.2	ST_AsEWKB	390
7.9.2.3	ST_AsHEXEWKB	391
7.9.3	其它格式	392
7.9.3.1	ST_AsEncodedPolyline	392
7.9.3.2	ST_AsFlatGeobuf	394
7.9.3.3	ST_AsGeobuf	394
7.9.3.4	ST_AsGeoJSON	395
7.9.3.5	ST_AsGML	397
7.9.3.6	ST_AsKML	400
7.9.3.7	ST_AsLatLonText	402
7.9.3.8	ST_AsMARC21	404
7.9.3.9	ST_AsMVTGeom	406
7.9.3.10	ST_AsMVT	408
7.9.3.11	ST_AsSVG	409
7.9.3.12	ST_AsTWKB	412
7.9.3.13	ST_AsX3D	413
7.9.3.14	ST_GeoHash	417
7.10	算符	419
7.10.1	边界算符	419

7.10.1.1	419
7.10.1.2(geometry,box2df)	420
7.10.1.3(box2df,geometry)	421
7.10.1.4(box2df,box2df)	422
7.10.1.5	423
7.10.1.6(geometry,gidx)	425
7.10.1.7(gidx,geometry)	426
7.10.1.8(gidx,gidx)	427
7.10.1.9<	428
7.10.1.10<	429
7.10.1.11>	430
7.10.1.12<	432
7.10.1.13<	433
7.10.1.14	434
7.10.1.15>	437
7.10.1.16@	439
7.10.1.17@(geometry,box2df)	440
7.10.1.18@ (box2df,geometry)	441
7.10.1.19@ (box2df,box2df)	442
7.10.1.20>	442
7.10.1.21>	444
7.10.1.22	445
7.10.1.23(geometry,box2df)	447
7.10.1.24(box2df,geometry)	447
7.10.1.25(box2df,box2df)	448
7.10.1.26=	449
7.10.2距离运算符	450
7.10.2.1<->	450
7.10.2.2 =	452
7.10.2.3<#>	453
7.10.2.4<->>	454
7.11空关系	455
7.11.1拓扑关系	455
7.11.1.1ST_3DIntersects	455
7.11.1.2ST_Contains	457
7.11.1.3ST_ContainsProperly	468
7.11.1.4ST_CoveredBy	471
7.11.1.5ST_Covers	473
7.11.1.6ST_Crosses	475

7.11.1.7ST_Disjoint	479
7.11.1.8ST_Equals	481
7.11.1.9ST_Intersects	483
7.11.1.10ST_LineCrossingDirection	486
7.11.1.11ST_OrderingEquals	489
7.11.1.12ST_Overlaps	492
7.11.1.13ST_Relate	498
7.11.1.14ST_RelateMatch	501
7.11.1.15ST_Touches	503
7.11.1.16ST_Within	510
7.11.2距离关系	513
7.11.2.1ST_3DDWithin	513
7.11.2.2ST_3DDFullyWithin	514
7.11.2.3ST_DFullyWithin	516
7.11.2.4ST_DWithin	517
7.11.2.5ST_PointInsideCircle	520
7.12测量函数	521
7.12.1ST_Area	521
7.12.2ST_Azimuth	524
7.12.3ST_Angle	527
7.12.4ST_ClosestPoint	529
7.12.5ST_3DClosestPoint	531
7.12.6ST_Distance	534
7.12.7ST_3DDistance	536
7.12.8ST_DistanceSphere	538
7.12.9ST_DistanceSpheroid	539
7.12.10ST_FrechetDistance	541
7.12.11ST_HausdorffDistance	542
7.12.12ST_Length	544
7.12.13ST_Length2D	547
7.12.14ST_3DLength	547
7.12.15ST_LengthSpheroid	549
7.12.16ST_LongestLine	551
7.12.17ST_3DLongestLine	554
7.12.18ST_MaxDistance	557
7.12.19ST_3DMaxDistance	560
7.12.20ST_MinimumClearance	561
7.12.21ST_MinimumClearanceLine	562
7.12.22ST_Perimeter	563

7.12.2	ST_Perimeter2D	567
7.12.2	ST_3DPerimeter	568
7.12.2	ST_ShortestLine	569
7.12.2	ST_3DShortestLine	571
7.13	加函数	574
7.13.1	ST_ClipByBox2D	574
7.13.2	ST_Difference	575
7.13.3	ST_Intersection	577
7.13.4	ST_MemUnion	581
7.13.5	ST_Node	581
7.13.6	ST_Split	583
7.13.7	ST_Subdivide	586
7.13.8	ST_SymDifference	589
7.13.9	ST_UnaryUnion	591
7.13.1	ST_Union	592
7.14	几何处理	597
7.14.1	ST_Buffer	597
7.14.2	ST_BuildArea	606
7.14.3	ST_Centroid	609
7.14.4	ST_ChaikinSmoothing	614
7.14.5	ST_CatmullRomSmoothing	618
7.14.6	ST_ConcaveHull	620
7.14.7	ST_ConvexHull	624
7.14.8	ST_DelaunayTriangles	626
7.14.9	ST_FilterByM	629
7.14.1	ST_GeneratePoints	631
7.14.1	ST_GeometricMedian	632
7.14.1	ST_LineMerge	634
7.14.1	ST_MaximumInscribedCircle	639
7.14.1	ST_LargestEmptyCircle	641
7.14.1	ST_MinimumBoundingCircle	643
7.14.1	ST_MinimumBoundingRadius	645
7.14.1	ST_OrientedEnvelope	646
7.14.1	ST_OffsetCurve	647
7.14.1	ST_PointOnSurface	654
7.14.2	ST_Polygonize	659
7.14.2	ST_ReducePrecision	662
7.14.2	ST_SharedPaths	666
7.14.2	ST_Simplify	668

7.14.25	ST_SimplifyPreserveTopology	671
7.14.25	ST_SimplifyPolygonHull	673
7.14.26	ST_SimplifyVW	677
7.14.27	ST_SetEffectiveArea	680
7.14.28	ST_TriangulatePolygon	682
7.14.29	ST_VoronoiLines	685
7.14.30	ST_VoronoiPolygons	686
7.15	覆盖范围函数	688
7.15.1	ST_CoverageInvalidEdges	688
7.15.2	ST_CoverageClean	690
7.15.3	ST_CoverageSimplify	691
7.15.4	ST_CoverageEdges	693
7.15.5	ST_CoverageUnion	694
7.16	仿射变换函数	696
7.16.1	ST_Affine	696
7.16.2	ST_Rotate	698
7.16.3	ST_RotateX	701
7.16.4	ST_RotateY	702
7.16.5	ST_RotateZ	703
7.16.6	ST_Scale	705
7.16.7	ST_Translate	709
7.16.8	ST_TransScale	712
7.17	聚类函数	714
7.17.1	ST_ClusterDBSCAN	714
7.17.2	ST_ClusterIntersecting	716
7.17.3	ST_ClusterIntersectingWin	717
7.17.4	ST_ClusterRelateWin	719
7.17.5	ST_MinimumSpanningTree	720
7.17.6	ST_ClusterKMeans	722
7.17.7	ST_ClusterWithin	726
7.17.8	ST_ClusterWithinWin	727
7.18	边界框函数	729
7.18.1	Box2D	729
7.18.2	Box3D	731
7.18.3	ST_EstimatedExtent	733
7.18.4	ST_Expand	734
7.18.5	ST_Extent	736
7.18.6	ST_3DExtent	738
7.18.7	ST_MakeBox2D	739

7.18.8ST_3DMakeBox	740
7.18.9ST_XMax	740
7.18.10ST_XMin	743
7.18.11ST_YMax	745
7.18.12ST_YMin	747
7.18.13ST_ZMax	750
7.18.14ST_ZMin	752
7.18.15ST_XSize	755
7.18.16ST_YSize	756
7.18.17ST_ZSize	757
7.18.18ST_MMin	759
7.18.19ST_MMax	759
7.18.20ST_MSize	760
7.19☒性参考	761
7.19.1ST_LineInterpolatePoint	761
7.19.2ST_3DLineInterpolatePoint	765
7.19.3ST_LineInterpolatePoints	766
7.19.4ST_LineLocatePoint	768
7.19.5ST_LineSubstring	771
7.19.6ST_LocateAlong	777
7.19.7ST_LocateBetween	778
7.19.8ST_LocateBetweenElevations	780
7.19.9ST_InterpolatePoint	782
7.19.10ST_3DInterpolatePoint	783
7.19.11ST_AddMeasure	785
7.20☒迹函数	788
7.20.1ST_IsValidTrajectory	788
7.20.2ST_ClosestPointOfApproach	789
7.20.3ST_DistanceCPA	791
7.20.4ST_CPAWithin	792
7.21版本函数	793
7.21.1PostGIS_Extensions_Upgrade	793
7.21.2PostGIS_Full_Version	794
7.21.3PostGIS_GEOS_Version	795
7.21.4PostGIS_GEOS_Compiled_Version	795
7.21.5PostGIS_Liblwgeom_Version	796
7.21.6PostGIS_LibXML_Version	797
7.21.7PostGIS_LibJSON_Version	797
7.21.8PostGIS_Lib_Build_Date	798

7.21.9	PostGIS_Lib_Version	798
7.21.10	PostGIS_PROJ_Version	799
7.21.11	PostGIS_PROJ_Compiled_Version	799
7.21.12	PostGIS_Wagyu_Version	800
7.21.13	PostGIS_Scripts_Build_Date	801
7.21.14	PostGIS_Scripts_Installed	801
7.21.15	PostGIS_Scripts_Released	802
7.21.16	PostGIS_Version	803
7.22	大一自定义量 (GUCs)	803
7.22.1	postgis.gdal_datapath	803
7.22.2	postgis.gdal_enabled_drivers	804
7.22.3	postgis.enable_outdb_rasters	805
7.22.4	postgis.gdal_vsi_options	807
7.22.5	postgis.gdal_cpl_debug	807
7.23	故障排除函数	807
7.23.1	PostGIS_AddBBox	807
7.23.2	PostGIS_DropBBox	808
7.23.3	PostGIS_HasBBox	809
8	SFCGAL 函数参考	811
8.1	SFCGAL 管理函数	811
8.1.1	postgis_sfcgal_version	811
8.1.2	postgis_sfcgal_full_version	811
8.2	SFCGAL 器和置器	812
8.2.1	CG_ForceLHR	812
8.2.2	CG_IsPlanar	813
8.2.3	CG_IsSolid	813
8.2.4	CG_MakeSolid	815
8.2.5	CG_Orientation	815
8.2.6	CG_Area	816
8.2.7	CG_3DArea	817
8.2.8	CG_Volume	818
8.3	SFCGAL 理和关系函数	819
8.3.1	CG_Intersection	819
8.3.2	CG_Intersects	821
8.3.3	CG_3DIntersects	823
8.3.4	CG_Difference	824
8.3.5	CG_3DDifference	825
8.3.6	CG_Distance	827

8.3.7 CG_3DDistance	828
8.3.8 CG_3DConvexHull	829
8.3.9 CG_3DIntersection	832
8.3.10CG_Union	835
8.3.11CG_3DUnion	836
8.3.12CG_AlphaShape	838
8.3.13CG_ApproxConvexPartition	841
8.3.14CG_ApproximateMedialAxis	842
8.3.15CG_GenerateFlatRoof	845
8.3.16CG_GenerateHippedRoof	846
8.3.17CG_GenerateGableRoof	847
8.3.18CG_GenerateSkillionRoof	848
8.3.19CG_GenerateRoof	850
8.3.20CG_PolygonRepair	851
8.3.21CG_ConstrainedDelaunayTriangles	854
8.3.22CG_Extrude	856
8.3.23CG_ExtrudeStraightSkeleton	858
8.3.24CG_GreeneApproxConvexPartition	860
8.3.25CG_MinkowskiSum	861
8.3.26CG_OptimalAlphaShape	863
8.3.27CG_OptimalConvexPartition	865
8.3.28CG_StraightSkeleton	867
8.3.29CG_Tessellate	869
8.3.30CG_Triangulate	871
8.3.31CG_Visibility	873
8.3.32CG_YMonotonePartition	875
8.3.33CG_StraightSkeletonPartition	876
8.3.34CG_3DBuffer	878
8.3.35CG_Rotate	881
8.3.36CG_2DRotate	882
8.3.37CG_3DRotate	883
8.3.38CG_RotateX	884
8.3.39CG_RotateY	885
8.3.40CG_RotateZ	886
8.3.41CG_Scale	887
8.3.42CG_3DScale	888
8.3.43CG_3DScaleAroundCenter	889
8.3.44CG_Translate	890
8.3.45CG_3DTranslate	891

8.3.46	CG_Simplify	892
8.3.47	CG_3DAlphaWrapping	897
8.3.48	CG_NurbsCurveInterpolate	898
8.3.49	CG_NurbsCurveApproximate	900
8.3.50	CG_NurbsCurveDerivative	903
8.4	Deprecated SFCGAL Function Aliases	906
8.4.1	ST_3DIntersection	906
8.4.2	ST_3DDifference	907
8.4.3	ST_3DUnion	907
8.4.4	ST_Tessellate	907
8.4.5	ST_3DArea	908
8.4.6	ST_Extrude	908
8.4.7	ST_ForceLHR	909
8.4.8	ST_Orientation	909
8.4.9	ST_MinkowskiSum	910
8.4.10	ST_StraightSkeleton	910
8.4.11	ST_ApproximateMedialAxis	910
8.4.12	ST_IsPlanar	911
8.4.13	ST_Volume	911
8.4.14	ST_MakeSolid	912
8.4.15	ST_IsSolid	912
8.4.16	ST_ConstrainedDelaunayTriangles	913
8.4.17	ST_3DConvexHull	913
8.4.18	ST_AlphaShape	913
8.4.19	ST_OptimalAlphaShape	914
9	拓扑	915
9.1	Building Faces from Linework	915
9.2	拓扑基表	917
9.3	拓扑型	917
9.3.1	getfaceedges_returntype	917
9.3.2	TopoGeometry	918
9.3.3	validatetopology_returntype	918
9.4	拓扑域	919
9.4.1	TopoElement	919
9.4.2	TopoElementArray	920
9.5	拓扑和拓扑几何管理	921
9.5.1	AddTopoGeometryColumn	921
9.5.2	RenameTopoGeometryColumn	922

9.5.3	DropTopology	922
9.5.4	RenameTopology	923
9.5.5	DropTopoGeometryColumn	924
9.5.6	FixCorruptTopoGeometryColumn	924
9.5.7	Populate_Topology_Layer	925
9.5.8	TopologySummary	926
9.5.9	ValidateTopology	927
9.5.10	ValidateTopologyRelation	929
9.5.11	ValidateTopologyPrecision	929
9.5.12	MakeTopologyPrecise	930
9.5.13	FindTopology	931
9.5.14	FindLayer	932
9.5.15	TotalTopologySize	932
9.5.16	UpgradeTopology	933
9.6	拓扑管理	934
9.7	拓扑造器	934
9.7.1	CreateTopology	934
9.7.2	CopyTopology	935
9.7.3	ST_InitTopoGeo	936
9.7.4	ST_CreateTopoGeo	936
9.7.5	TopoGeo_AddPoint	937
9.7.6	TopoGeo_AddLineString	938
9.7.7	TopoGeo_AddPolygon	938
9.7.8	TopoGeo_LoadGeometry	939
9.8	拓扑器	939
9.8.1	ST_AddIsoNode	939
9.8.2	ST_AddIsoEdge	940
9.8.3	ST_AddEdgeNewFaces	941
9.8.4	ST_AddEdgeModFace	941
9.8.5	ST_RemEdgeNewFace	942
9.8.6	ST_RemEdgeModFace	943
9.8.7	ST_ChangeEdgeGeom	943
9.8.8	ST_ModEdgeSplit	944
9.8.9	ST_ModEdgeHeal	945
9.8.10	ST_NewEdgeHeal	945
9.8.11	ST_MoveIsoNode	946
9.8.12	ST_NewEdgesSplit	947
9.8.13	ST_RemoveIsoNode	948
9.8.14	ST_RemoveIsoEdge	948

9.9 拓扑查询器	949
9.9.1 GetEdgeByPoint	949
9.9.2 GetFaceByPoint	950
9.9.3 GetFaceContainingPoint	951
9.9.4 GetNodeByPoint	952
9.9.5 GetTopologyID	953
9.9.6 GetTopologySRID	953
9.9.7 GetTopologyName	954
9.9.8 ST_GetFaceEdges	954
9.9.9 ST_GetFaceGeometry	955
9.9.10GetRingEdges	956
9.9.11GetNodeEdges	957
9.9.12FindVertexSegmentPairsBelowDistance	958
9.10拓扑管理	958
9.10.1Polygonize	958
9.10.2AddNode	959
9.10.3AddEdge	959
9.10.4AddFace	961
9.10.5ST_Simplify	962
9.10.6RemoveUnusedPrimitives	963
9.11拓扑几何构造函数	963
9.11.1CreateTopoGeom	963
9.11.2toTopoGeom	966
9.11.3TopoElementArray_Agg	968
9.11.4TopoElement	968
9.12拓扑几何查询器	969
9.12.1clearTopoGeom	969
9.12.2TopoGeom_addElement	969
9.12.3TopoGeom_remElement	970
9.12.4TopoGeom_addTopoGeom	971
9.12.5toTopoGeom	971
9.13拓扑几何查询器	971
9.13.1GetTopoGeomElementArray	971
9.13.2GetTopoGeomElements	972
9.13.3ST_SRID	972
9.14拓扑几何输出	973
9.14.1AsGML	973
9.14.2AsTopoJSON	975
9.15拓扑空关系	977

9.15.1	Equals	977
9.15.2	Intersects	978
9.16	输入和输出拓扑	979
9.16.1	使用拓扑输出器	979
9.16.2	使用拓扑输入器	979
10	栅格数据管理、栅格和应用程序	980
10.1	添加和创建栅格	980
10.1.1	使用 raster2pgsql 添加栅格	980
10.1.1.1	用法示例	980
10.1.1.2	raster2pgsql 栅格	981
10.1.2	使用 PostGIS 栅格函数创建栅格	983
10.1.3	使用 “out db” 云栅格	983
10.2	Raster Coverages and Raster/Vector Analysis	984
10.3	栅格目录	986
10.3.1	栅格列目录	986
10.3.2	栅格概述	987
10.4	使用 PostGIS Raster 创建自定义应用程序	987
10.4.1	PHP 示例使用 ST_AsPNG 与其他栅格函数行输出	988
10.4.2	ASP.NET C# 示例使用 ST_AsPNG 与其他栅格函数行输出	988
10.4.3	将栅格输出为图像文件的 Java 控制台应用程序	990
10.4.4	使用 PLPython 通过 SQL 输出图像	992
10.4.5	使用 PSQL 输出栅格	992
11	栅格参考	994
11.1	栅格支持数据类型	995
11.1.1	geomval	995
11.1.2	addbandarg	995
11.1.3	rastbandarg	995
11.1.4	raster	996
11.1.5	reclassarg	996
11.1.6	summarystats	997
11.1.7	unionarg	998
11.2	栅格管理	998
11.2.1	AddRasterConstraints	998
11.2.2	DropRasterConstraints	1000
11.2.3	AddOverviewConstraints	1001
11.2.4	DropOverviewConstraints	1002
11.2.5	PostGIS_GDAL_Version	1003

11.2.6	PostGIS_Raster_Lib_Build_Date	1003
11.2.7	PostGIS_Raster_Lib_Version	1004
11.2.8	ST_GDALDrivers	1004
11.2.9	UpdateRasterSRID	1007
11.2.10	ST_CreateOverview	1008
11.3	栅格处理器	1009
11.3.1	ST_AddBand	1009
11.3.2	ST_AsRaster	1011
11.3.3	ST_AsRasterAgg	1014
11.3.4	ST_Band	1015
11.3.5	ST_MakeEmptyCoverage	1019
11.3.6	ST_MakeEmptyRaster	1020
11.3.7	ST_Tile	1022
11.3.8	ST_Retile	1024
11.3.9	ST_FromGDALRaster	1024
11.4	栅格属性器	1025
11.4.1	ST_GeoReference	1025
11.4.2	ST_Height	1026
11.4.3	ST_IsEmpty	1027
11.4.4	ST_MemSize	1028
11.4.5	ST_MetaData	1029
11.4.6	ST_NumBands	1029
11.4.7	ST_PixelHeight	1030
11.4.8	ST_PixelWidth	1031
11.4.9	ST_ScaleX	1033
11.4.10	ST_ScaleY	1034
11.4.11	ST_RasterToWorldCoord	1034
11.4.12	ST_RasterToWorldCoordX	1035
11.4.13	ST_RasterToWorldCoordY	1036
11.4.14	ST_Rotation	1037
11.4.15	ST_SkewX	1038
11.4.16	ST_SkewY	1039
11.4.17	ST_SRID	1040
11.4.18	ST_Summary	1041
11.4.19	ST_UpperLeftX	1042
11.4.20	ST_UpperLeftY	1042
11.4.21	ST_Width	1043
11.4.22	ST_WorldToRasterCoord	1043
11.4.23	ST_WorldToRasterCoordX	1044

11.4.2	\$T_WorldToRasterCoordY	1045
11.5	格波段器	1046
11.5.1	ST_BandMetaData	1046
11.5.2	ST_BandNoDataValue	1047
11.5.3	ST_BandIsNoData	1048
11.5.4	ST_BandPath	1049
11.5.5	ST_BandFileSize	1050
11.5.6	ST_BandFileTimestamp	1051
11.5.7	ST_BandPixelType	1051
11.5.8	ST_MinPossibleValue	1052
11.5.9	ST_HasNoBand	1053
11.6	格像素器和置器	1054
11.6.1	ST_PixelAsPolygon	1054
11.6.2	ST_PixelAsPolygons	1055
11.6.3	ST_PixelAsPoint	1056
11.6.4	ST_PixelAsPoints	1057
11.6.5	ST_PixelAsCentroid	1059
11.6.6	ST_PixelAsCentroids	1060
11.6.7	ST_Value	1062
11.6.8	ST_NearestValue	1065
11.6.9	ST_SetZ	1067
11.6.10	ST_SetM	1069
11.6.11	ST_Neighborhood	1071
11.6.12	ST_SetValue	1073
11.6.13	ST_SetValues	1074
11.6.14	ST_DumpValues	1082
11.6.15	ST_PixelOfValue	1083
11.7	格器	1084
11.7.1	ST_SetGeoReference	1084
11.7.2	ST_SetRotation	1085
11.7.3	ST_SetScale	1086
11.7.4	ST_SetSkew	1087
11.7.5	ST_SetSRID	1089
11.7.6	ST_SetUpperLeft	1090
11.7.7	ST_Resample	1090
11.7.8	ST_Rescale	1092
11.7.9	ST_Reskew	1093
11.7.10	ST_SnapToGrid	1095
11.7.11	ST_Resize	1096

11.7.1	<code>ST_Transform</code>	1097
11.8	栅波段器	1101
11.8.1	<code>ST_SetBandNoDataValue</code>	1101
11.8.2	<code>ST_SetBandIsNoData</code>	1102
11.8.3	<code>ST_SetBandPath</code>	1104
11.8.4	<code>ST_SetBandIndex</code>	1105
11.9	栅波段和分析	1107
11.9.1	<code>ST_Count</code>	1107
11.9.2	<code>ST_CountAgg</code>	1108
11.9.3	<code>ST_Histogram</code>	1109
11.9.4	<code>ST_Quantile</code>	1110
11.9.5	<code>ST_SummaryStats</code>	1112
11.9.6	<code>ST_SummaryStatsAgg</code>	1115
11.9.7	<code>ST_ValueCount</code>	1116
11.10	栅入	1119
11.10.	<code>ST_RastFromWKB</code>	1119
11.10.	<code>ST_RastFromHexWKB</code>	1120
11.11	栅出	1120
11.11.	<code>ST_AsBinary/ST_AsWKB</code>	1120
11.11.	<code>ST_AsHexWKB</code>	1121
11.11.	<code>ST_AsGDALRaster</code>	1122
11.11.	<code>ST_AsJPEG</code>	1123
11.11.	<code>ST_AsPNG</code>	1124
11.11.	<code>ST_AsTIFF</code>	1125
11.12	栅理：代数	1126
11.12.	<code>ST_Clip</code>	1126
11.12.	<code>ST_ColorMap</code>	1131
11.12.	<code>ST_Grayscale</code>	1134
11.12.	<code>ST_Intersection</code>	1138
11.12.	<code>ST_MapAlgebra</code> (callback function version)	1141
11.12.	<code>ST_MapAlgebra</code> (expression version)	1148
11.12.	<code>ST_MapAlgebraExpr</code>	1150
11.12.	<code>ST_MapAlgebraExpr</code>	1154
11.12.	<code>ST_MapAlgebraFct</code>	1161
11.12.	<code>ST_MapAlgebraFct</code>	1165
11.12.	<code>ST_MapAlgebraFctNgb</code>	1171
11.12.	<code>ST_Reclass</code>	1175
11.12.	<code>ST_ReclassExact</code>	1176
11.12.	<code>ST_Union</code>	1178

11.1 配置地理代数回函数	1179
11.13.\$T_Distinct4ma	1179
11.13.\$T_InvDistWeight4ma	1180
11.13.\$T_Max4ma	1182
11.13.\$T_Mean4ma	1183
11.13.\$T_Min4ma	1184
11.13.\$T_MinDist4ma	1185
11.13.\$T_Range4ma	1186
11.13.\$T_StdDev4ma	1187
11.13.\$T_Sum4ma	1189
11.1 栅格地理：DEM（高程）	1190
11.14.\$T_Aspect	1190
11.14.\$T_HillShade	1193
11.14.\$T_Roughness	1194
11.14.\$T_Slope	1195
11.14.\$T_TPI	1197
11.14.\$T_TRI	1198
11.14.\$T_InterpolateRaster	1199
11.14.\$T_Contour	1200
11.1 栅格地理：栅格到几何	1201
11.15.Box3D	1201
11.15.\$T_ConvexHull	1202
11.15.\$T_DumpAsPolygons	1203
11.15.\$T_Envelope	1204
11.15.\$T_MinConvexHull	1204
11.15.\$T_Polygon	1206
11.15.\$T_IntersectionFractions	1208
11.1 栅格算符	1209
11.16.&&	1209
11.16.&<	1209
11.16.&>	1210
11.16.#	1211
11.16.@	1211
11.16.=	1212
11.16.~	1213
11.1 栅格和栅格波段空关系	1213
11.17.\$T_Contains	1213
11.17.\$T_ContainsProperly	1214
11.17.\$T_Covers	1215

11.17.1	<code>\$T_CoveredBy</code>	1216
11.17.2	<code>\$T_Disjoint</code>	1217
11.17.3	<code>\$T_Intersects</code>	1218
11.17.4	<code>\$T_Overlaps</code>	1219
11.17.5	<code>\$T_Touches</code>	1220
11.17.6	<code>\$T_SameAlignment</code>	1221
11.17.7	<code>\$T_NotSameAlignmentReason</code>	1222
11.17.8	<code>\$T_Within</code>	1223
11.17.9	<code>\$T_DWithin</code>	1224
11.17.10	<code>\$T_DFullyWithin</code>	1225
11.18	网格提示	1226
11.18.1	Out-DB 网格	1226
11.18.1.1	包含多个文件的目录	1226
11.18.1.2	最大打开文件数	1226
11.18.1.2.1	整个系统最大打开文件数	1227
11.18.1.2.2	每个进程的最大打开文件数	1227

12 PostGIS 补充

1230

13 PostGIS 特殊函数索引

1231

13.1	PostGIS 聚合函数	1231
13.2	PostGIS 窗口函数	1232
13.3	PostGIS SQL-MM 兼容函数	1232
13.4	PostGIS 地理支持函数	1236
13.5	PostGIS 网格支持函数	1237
13.6	PostGIS 几何/地理/网格输出函数	1242
13.7	PostGIS 边界框函数	1242
13.8	支持 3D 的 PostGIS 函数	1244
13.9	PostGIS 曲线几何支持函数	1249
13.10	PostGIS 多面体曲面支持函数	1252
13.11	PostGIS 函数支持矩阵	1255
13.12	新的、增加的或更改的 PostGIS 函数	1274
13.12.1	PostGIS 新增功能或增强功能 (3.7)	1274
13.12.2	PostGIS 新增功能或增强功能 (3.6)	1276
13.12.3	PostGIS 新增功能或增强功能 (3.5)	1277
13.12.4	PostGIS 新增功能或增强功能 (3.4)	1278
13.12.5	PostGIS 新增功能或增强功能 (3.3)	1280
13.12.6	PostGIS 新增功能或增强功能 (3.2)	1280
13.12.7	PostGIS 新增功能或增强功能 (3.1)	1282

13.12.	PostGIS 新增功能或增⼤功能 (3.0)	1283
13.12.	PostGIS 新增功能或增⼤功能 (2.5)	1284
13.12.	PostGIS 新增功能或增⼤功能 (2.4)	1285
13.12.	PostGIS 新增功能或增⼤功能 (2.3)	1286
13.12.	PostGIS 新增功能或增⼤功能 (2.2)	1288
13.12.	PostGIS 新增功能或增⼤功能 (2.1)	1290
13.12.	PostGIS 新增功能或增⼤功能 (2.0)	1294
13.12.	PostGIS 新增功能或增⼤功能 (1.5)	1302
13.12.	PostGIS 新增功能或增⼤功能 (1.4)	1303
13.12.	PostGIS 新增功能或增⼤功能 (1.3)	1304
14	告⽩	1305
14.1	告⽩件	1305
14.2	告⽩文档	1305
15	Credits and Acknowledgments	1306
15.1	Project Steering Committee	1306
15.2	Core Contributors Present	1306
15.3	Core Contributors Past	1307
15.4	PostGIS Raster Origins	1307
15.5	Other Contributors	1308
A	⾏⾏明	1312
A.1	PostGIS 3.7.0	1312
A.1.1	重大⾏化	1312
A.1.2	新特性	1312
A.1.3	增⼤功能	1313
A.1.4	Bug Fixes	1314

Abstract

PostGIS 是一种扩展功能，适用于 **PostgreSQL** 对象关系数据库系统，它允许将 GIS（地理信息系统）对象存储在数据库中。PostGIS 包含基于 GiST 的 R-Tree 空间索引的支持，以及用于分析和处理 GIS 对象的功能。



本手册的版本是 3.7.0rc2



本作品已得 **创作共用署名-相同方式共享 3.0** 许可。可随意使用这份材料，任何您喜欢的方式，但我要求您将荣誉归于 PostGIS 项目，尽可能提供返回<https://postgis.net>的链接。

Chapter 1



1.1 What Is PostGIS?

PostGIS extends **PostgreSQL** so that geospatial data can be stored, indexed, queried, and processed in the database. It keeps location data beside ordinary relational data, so a spatial query can use the same SQL, transactions, constraints, access controls, backup tools, and application connections as the rest of a PostgreSQL database.

The main `postgis` extension adds:

- geometry for planar coordinate systems and geography for geodetic coordinates on the earth.
- Spatial indexes that let the PostgreSQL planner avoid scanning every row when objects can be filtered by location.
- Functions for measurement, relationships, overlays, geometry construction and editing, coordinate transformation, clustering, and spatial data exchange.
- Support for common geospatial encodings and formats, including WKT, WKB, GeoJSON, GML, KML, FlatGeobuf, Geobuf, and Mapbox Vector Tiles.

This makes PostgreSQL a spatial database: applications can ask both ordinary questions and location-aware questions without moving data to a separate GIS processing system. Desktop GIS, map servers, ETL tools, and application frameworks can connect through normal PostgreSQL interfaces.

1.2 What Is in the PostGIS Distribution?

This manual documents the main PostGIS source distribution. Depending on how PostGIS was built and packaged, it can provide several PostgreSQL extensions and command-line tools:

postgis The core geometry and geography types, spatial indexes, and most spatial functions.

postgis_raster Raster storage, analysis, loading, and export. Raster support uses GDAL and can be omitted from a build.

postgis_topology SQL/MM topology types and functions for models based on shared nodes, edges, and faces.

postgis_sfcgal Additional 2D and 3D processing functions backed by the optional SFCGAL library.

Loaders and utilities `shp2pgsql`, `pgsql2shp`, and `raster2pgsql` move data between common file formats and PostGIS. Other utilities support upgrades, profiling, and development.

The source tree also contains internal libraries such as `liblwgeom` and `libpgcommon`. They are implementation layers shared by PostGIS components, not separately released PostgreSQL extensions. Most users start by enabling the core extension with `CREATE EXTENSION postgis`; see Chapter 2 for available components and build options.

1.3 The PostGIS Project Family

This manual covers the main `postgis/postgis` distribution. The official [PostGIS source organization](#) also hosts or mirrors complementary projects. These projects have their own release cycles, installation instructions, and documentation; installing the core extension does not install them.

- [Address Standardizer](#) parses and normalizes postal addresses. It moved from the main PostGIS source tree to a separate repository for the PostGIS 3.7 development cycle.
- [PostGIS TIGER Geocoder](#) provides United States address geocoding and reverse geocoding using Census TIGER data. It is also maintained separately from the main distribution.
- [H3-pg](#) provides PostgreSQL bindings for the H3 hierarchical hexagonal indexing system and inter-operates with PostGIS.
- [PostGIS Java](#) provides Java bindings for PostGIS types.
- [Introduction to PostGIS](#) is the hands-on workshop maintained by the PostGIS project.

Other PostgreSQL spatial extensions can complement PostGIS while remaining independent projects. For example, [pgRouting](#) adds routing and graph analysis. Use each project's own manual for its functions and support policy.

1.4 How to Use This Manual

The manual is both a guide to the PostGIS data model and the complete reference for the SQL interface. Choose a starting point based on what you are trying to do:

If you are new to spatial SQL Start with the [Introduction to PostGIS workshop](#) for a tutorial, then use Chapter 4 to understand spatial types, coordinate systems, validity, loading, and indexes.

If you are installing or upgrading PostGIS Read Chapter 2 and Chapter 3. Package-specific instructions and supported release information are maintained on the [PostGIS website](#).

If you are writing spatial queries Read Chapter 5 for spatial relationships and index-aware query patterns, and Chapter 6 for common performance problems.

If you are connecting a desktop GIS Use the client's current database documentation for its user interface and supported geometry types. For example, the [QGIS DB Manager lesson](#) covers PostgreSQL connections, table layers, SQL query layers, import, and export. A query layer should return a stable unique identifier and a geometry column with a known SRID and geometry type. Expensive read-only queries can be exposed through a view or materialized view and indexed as appropriate.

If you need a particular function Use Chapter 7 or Chapter 13. Function pages state accepted types, dimensional support, standards conformance, dependency requirements, and version availability.

If you use an optional component See Chapter 9, Chapter 10, Chapter 11, or Chapter 8.

If something goes wrong Collect the versions reported by `PostGIS_Full_Version` and follow Chapter 14. Check Appendix A when behavior differs between release lines.

1.5 Standards Support

This manual marks individual functions as compliant with spatial standards so users can tell which behavior is portable and which behavior is a PostGIS extension. PostGIS implements the **OGC Simple Features for SQL** model, including the geometry type, Well-Known Text (WKT), Well-Known Binary (WKB), and the common spatial accessors, constructors, predicates, and processing functions. PostGIS also implements many **SQL/MM Spatial** types and functions.

PostGIS supports geometry families defined by later standards, including SQL/MM curved geometry and OGC Simple Features Access 1.2 PolyhedralSurface, Triangle, and TIN. Support for these geometry families is function-specific, and each function page documents the supported input types and conformance level.

PostGIS also includes widely used extensions and integrations. Examples of PostGIS extensions include Extended WKT (EWKT), Extended WKB (EWKB), SRID metadata in those extended formats, geography, and raster. PostGIS also supports input and output exchange formats defined by other standards, such as GeoJSON, GML, and KML. These features are documented individually and should not be read as OGC Simple Features or SQL/MM requirements unless the specific function or format says so.

When the implementation intentionally differs from a standard, or when a standard leaves behavior ambiguous, the relevant function page describes the PostGIS behavior. Chapter 13 summarizes SQL/MM compliance and geometry-type support across functions.

1.6 Project and Community

PostGIS began in 2001 as a spatial database research project at Refrations Research. It is now an **OSGeo project** developed by contributors and organizations around the world. The Project Steering Committee coordinates releases, project policy, and the general direction of development. The complete contributor and sponsor record is in Chapter 15 at the end of this manual.

The manual is the authoritative reference for the released SQL interface. The project website and source tree provide the surrounding material:

- **Documentation** routes readers to getting-started instructions, versioned manuals, tips, training material, and the FAQ.
 - **Community** lists mailing lists, chat, events, videos, and other support channels.
 - **Developer documentation** covers development environments, testing, coding style, internals, maintenance, governance, releases, and website work.
-

Chapter 2

PostGIS 安装

本章介绍安装 PostGIS 所需的步骤。

2.1 简短版本

如果所有依赖项都在路径中，请按如下方式操作：

Code

```
tar -xvzf postgis-3.7.0rc2.tar.gz
cd postgis-3.7.0rc2
./configure
make
make install
```

安装 PostGIS 后，您需要使每个数据库可用（Section 3.3）或升级它（Section 3.4）。

2.2 从源代码编译和安装

Note

在许多操作系统都包含 PostgreSQL/PostGIS 的二进制包。在大多数情况下，只有当您想要最前沿的版本或者您是软件包维护者时才需要编译。



This section includes general compilation instructions. If you are compiling for Windows or another operating system, see the [PostGIS getting started guides](#) and the [development environment guides](#).

Pre-Built Packages for various OS are listed in the [PostGIS getting started guides](#)

如果您是 Windows 用户，您可以通过 Stackbuilder 或 [PostGIS Windows 下载站点](#) 获得预构建的版本。我们也有非常前沿的 [Windows 二进制版本](#)，通常每周构建一次，或者在有令人兴奋的新优化生成时构建。您可以使用这些版本来测试正在开发中的 PostGIS 发布。

PostGIS 模块是 PostgreSQL 后台服务器的扩展。在 PostGIS 3.7.0rc2 中，除了模块，必须提供完整的 PostgreSQL 服务器文件。它可以在 PostgreSQL 14 - 19 版本之构建。旧版本的 PostgreSQL 不再支持。

如果您没有安装 PostgreSQL，请参考 PostgreSQL 安装指南。它位于 <https://www.postgresql.org>。

**Note**

关于 GEOS 功能，当您安装 PostgreSQL 时，您可能需要将 PostgreSQL 式样接到标准 C++ ：

Code

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

这是解决与旧开发工具不正常交互的虚假 C++ 异常的一种通用方法。如果你遇到奇怪的错误（例如，后端意外关闭或类似的错误），这是一个技巧。当然，您可能需要重新从您的 PostgreSQL。

下一步是概述 PostGIS 源配置和安装。这些是 Linux 用 C 写的，不适用于 Windows 或 Mac。

2.2.1 获取源代码

从以下站点 <https://download.osgeo.org/postgis/source/postgis-3.7.0rc2.tar.gz> 获取源代码的存档

Code

```
wget https://download.osgeo.org/postgis/source/postgis-3.7.0rc2.tar.gz
tar -xvzf postgis-3.7.0rc2.tar.gz
cd postgis-3.7.0rc2
```

这将在当前工作目录中创建一个名为 postgis-3.7.0rc2 的目录。

或者从 [git](https://git.osgeo.org/gitea/postgis/postgis/) 存储库 <https://git.osgeo.org/gitea/postgis/postgis/> 中查看它。

Code

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

切换到新创建的 postgis 目录以进行安装。

Code

```
./configure
```

2.2.2 安装要求

要构建和使用 PostGIS，您需要：

不可缺少

- PostgreSQL 14 - 19. 需要完整安装 PostgreSQL（包括服务器文件）。可从 <https://www.postgresql.org> 获取 18。
For current release-line support and end-of-life policy, see https://postgis.net/development/versions_eol/
- GNU C 编译器 (gcc)。其他一些 ANSI C 编译器也可以用来构建 PostGIS，但使用 gcc 我们测试得要少得多。
- GNU Make (gmake or make)。对于多系统，GNU make 是 make 的默认版本。通常用 `make -v` 查看版本。其他版本的 make 可能无法正确处理 PostGIS Makefile。

- Proj 投影库。需要 Proj 6.1 及以上版本。Proj 用于 PostGIS 中提供坐标重投影支持。Proj 可以从<https://proj.org/> 下载。
- GEOS geometry library, version 3.10.0 or greater, but GEOS 3.15+ is required to take full advantage of all the new functions and features. GEOS is available for download from <https://libgeos.org> .
- LibXML2. 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>.
- Perl is required to build PostGIS. Perl is available for download from <https://www.perl.org/>.
- The iconv library is required by the shp2pgsql loader for encoding conversion. On GNU/Linux systems it is typically provided by libc; on other platforms you may need a separate **libiconv** package.
- The GNU Multiple Precision Arithmetic Library (libgmp). Used to control precision of math operations in PostGIS Topology. GMP is available for download from <https://gmplib.org/>. It is detected via **pkg-config** and has no fallback, so **pkg-config** must be available for a default build.
- GDAL, version 2.4 or higher, is required for raster support. GDAL 3+ is preferred. <https://gdal.org/download.html>.
- 要使用 PostgreSQL+JIT 运行，需要 LLVM 版本 6 或更高版本。参 <https://trac.osgeo.org/postgis/ticket/4125>。

可选项的

- GDAL (可选) 当您不需要它时才可以将其省略。确保您使用您要使用的程序，如 Section 3.2 中所述。
- JSON-C. JSON-C is currently used to import GeoJSON via the function ST_GeomFromGeoJson. JSON-C is available for download from <https://github.com/json-c/json-c/releases/>. It is enabled by default but optional; if it is not found configure only warns and the build continues. Disable with `--without-json`.
- GNU gettext for loader and documentation translation support. It may be disabled with `--with-gettext=0`. <https://www.gnu.org/software/gettext/>.
- GTK (需要 GTK+2.0, 2.8+)。用于 shp2pgsql-gui, 一个 shapefile loader。它位于 <http://www.gtk.org/>。
- SFCGAL, 1.3.1 or higher is required, 2.3+ is needed to be able to use all functionality. SFCGAL can be used to provide additional 2D and 3D advanced analysis functions to PostGIS cf Chapter 8. <https://sfcgal.gitlab.io/sfcgal/>.
- 要用 ST_AsMVT, 您需要 ProtoBuf-C (运行) 和 ProtoC-C 编译器 (构建)。要用 protobuf-c 的正确最低版本, 需要 pkg-config。参 [protobuf-c](#)。默认情况下, PostGIS 使用 Wagyu 快速估计 MVT 多边形, 但需要 C++11 编译器。使用 CXXFLAGS 使用您用于 PostgreSQL 安装的相同编译器。如果要禁用此功能并改用 GEOS, 在配置期指定它。 `--without-wagyu`。
- CUnit (CUnit)。是回归测试所需要的。 <http://cunit.sourceforge.net/>
- DocBook (xsltproc) 构建文档需要。Docbook 可从 <http://www.docbook.org/> 获取。
- Python 3 is required to run the manual example tester that generates the visual examples which the HTML and PDF documentation embed. The examples are executed against a live PostgreSQL server running the staged PostGIS build via **make -C regress visual-examples**; no Python database driver is required, the tooling talks to the server through the **psql** client. Python 3 is available from <https://www.python.org/> .
- DBLatex (dbratex) 需要以 PDF 格式构建文档。它位于 <http://dbratex.sourceforge.net/> 。
- PDF documentation needs an SVG rasterizer for its high-resolution figure fallbacks. The build prefers rsvg-convert, and can also use GraphicsMagick (`gm convert`) or ImageMagick (`magick` or `convert`).

2.2.3 构建配置

与大多数 Linux 安装一样，第一步是生成一个将用于构建源代码的 Makefile。这是通过 shell 脚本完成的

Code

```
./configure
```

如果未提供任何参数，此命令将自行在系统上构建 PostGIS 源代码所需的文件和库。**./configure** 是一种常用的用法，但它接受一些参数，以防您在非标准位置有必要的库或程序。

以下列表显示了常用参数：有关完整列表，请使用 **--help** 或 **--help=short** 参数。

--with-library-minor-version 从 PostGIS 3.0 开始，默认生成的库文件将不再将次要版本作为文件名的一部分。这意味着所有 PostGIS 3 库都将以 **postgis-3** 结尾。这样做是为了使 **pg_upgrade** 更容易，缺点是您只能在服务器中安装一个版本的 PostGIS 3 系列。要获取文件的旧行号（包括次要版本）：例如 **postgis-3.0** 将此开关添加到您的配置语句中。

--prefix=PREFIX This is the installation prefix for PostGIS command-line utilities. By default, utilities are installed in the executable directory reported by the detected PostgreSQL installation.



Caution

PostgreSQL extension files are installed in the directories reported by **pg_config**, so the server can load them. This includes the PostGIS shared library and SQL extension files. Use **--with-pgconfig=FILE** to choose the PostgreSQL installation that PostGIS builds and installs against.

--with-pgconfig=FILE PostgreSQL 有一个名为 **pg_config** 的实用程序，使 PostGIS 等扩展能够定位 PostgreSQL 安装目录。使用此参数 (**--with-pgconfig=/path/to/pg_config**) 可以手动指定 PostGIS 的特定 PostgreSQL 的安装目录。

--with-gdalconfig=FILE GDAL 是必需的库，提供格式支持 **gdal-config** 所需的功能，以使库安装能够找到 GDAL 装目录。使用此参数 (**--with-gdalconfig=/path/to/gdal-config**) 手动指定 PostGIS 将使用其构建的特定 GDAL 的安装目录。

--with-geosconfig=FILE 作为一个基本的几何库，GEOS 有一个名为 **geos-config** 的实用程序，它会告诉您在安装库时在何处安装 GEOS。使用此参数 (**--with-geosconfig=/path/to/geos-config**) 手动指定要用于 PostGIS 构建的特定 GEOS 的安装目录。

--with-xml2config=FILE LibXML 是运行 GeomFromKML/GML 程序所需的库。通常情况下，如果您已安装 libxml，它将被找到，但如果没有或者您想使用特定版本，您需要指定一个特定的 **xml2-config** 配置文件，以便库安装程序找到 LibXML 安装目录。使用这个参数 (**--with-xml2config=/path/to/xml2-config**) 来手动指定 PostGIS 将使用其构建的特定 LibXML 安装。

--with-projdir=DIR Proj 是 PostGIS 必不可少的投影库。使用此参数 (**--with-projdir=/path/to/projdir**) 手动指定要用于 PostGIS 构建的特定 Proj 的安装目录。

--with-libiconv=DIR iconv 的安装目录。

--with-jsondir=DIR **JSON-C** 是 MIT 许可的 JSON 库，是 PostGIS ST_GeomFromJSON 所必需的。使用此参数 (**--with-jsondir=/path/to/jsondir**) 手动指定要用于 PostGIS 构建的特定 JSON-C 的安装目录。

--with-gui 库数据输入 GUI（需要 GTK+2.0）。此参数将 **shp2pgsql** 构建一个名为 **shp2pgsql-gui** 的图形用户界面。

--without-raster 在没有栅格功能的情况下运行。

--without-topology 不支持 topology 支持。

- with-gettext=no** 默认情况下，会安装 gettext 并使用它来行注释，但如果在添加程序时坏的不兼容行，此命令可以禁用它。使用此功能的配置可以解决的示例，参见<http://trac.osgeo.org/postgis/ticket/748>。注意：关闭此功能不会消除多功能。它用于 GUI 加器中的内部帮助/功能，这些功能尚未实现，用于实现段。
- with-sfcgal=PATH** 默认情况下，如果没有此开关，PostGIS 将不会安装 sfcgal 支持。PATH 是一个可选项参数，允许指定 sfcgal-config 的选项 PATH。
- without-phony-revision** 禁用 postgis_revision.h 更新以匹配 Git 存储库中的当前 HEAD。

**Note**

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

Code

```
./autogen.sh
```

此脚本生成配置脚本。用于自定义 PostGIS 安装。

如果要获取 PostGIS 作为存档文件，无需 **./autogen.sh**，因为配置已经生成。

2.2.4 构建

生成 Makefile 后，构建 PostGIS 就像运行命令一样：

Code

```
make
```

Output

```
[last line of output]
PostGIS was built successfully. Ready to install.
```

所有函数的注释都是根据文档自动生成的。如果你之后想把一些注释安装到自己的空数据库里，需要运行依 docbook 的命令来注入注释。不过，如果是从 tar.gz 包安装，doc 目录里已经包含了 postgis_comments.sql 等注释文件，就不需要再额外生成了；此外，通过 CREATE EXTENSION 安装，这些注释也会一并包含在内。

Code

```
make comments
```

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

构建的 HTML 和 PDF 版本可以在[PostGIS / PostgreSQL 学习指南](#)中找到

Code

```
make cheatsheets
```

2.2.5 构建和部署 PostGIS 扩展

当 PostgreSQL 提供扩展支持，PostGIS 扩展会被自动构建并安装。

如果要从源存储构建，必须先构建函数描述。这些是在安装文档手册构建的。您也可以手动安装：

Code

```
make comments
```

如果从存档文件创建，其中一些注释文件已创建，因此无需创建注释。

如果您在 PostgreSQL 9.1 上运行创建，则在 make 安装过程中自行创建扩展。如果需要，可以从扩展文件生成，或者根据需要在同一服务器上复制文件。

Set PGUSER if you need to override the psql connection user, then test before installing. After installation, run the extension regression tests.

Code

```
cd extensions
cd postgis
make clean
make
export PGUSER=postgres
make check
make install
make check RUNTESTFLAGS=-extension
```

**Note**

make check 使用 psql 来运行测试，并使用 psql 环境变量。常用的可用于覆盖的有 PGUSER、PGPORT 和 PGHOST。参考 [psql 环境变量](#)

扩展文件在同一版本的 PostGIS 中始终是相同的，无论操作系统如何。只要已安装了 PostGIS 二进制文件，就可以将扩展文件从一个操作系统复制到一个操作系统。

如果您想在与开发环境不同的同一服务器上手动安装扩展，除了通常需要的 PostGIS 二进制文件之外，需要将以下文件从 extensions 文件夹复制到 PostgreSQL 安装目录下的 PostgreSQL/share/extension 文件夹中。

- 指定未提供，控制文件 `postgis.control`，`postgis_topology.control` 指示要安装的扩展版本等信息。
- 注意，每个扩展的 `/sql` 文件夹中的所有文件（例如，`extensions/postgis/sql/*.sql` 和 `extensions/postgis_topology/sql/*.sql`）需要复制到 PostgreSQL 的 `share/extension` 文件夹的最上层

完成一步后，您将在 `PgAdmin>` 中看到 `postgis`、`postgis_topology` 作为可用的扩展。

如果使用 psql，可以运行以下查询来检查是否已安装扩展：

Code

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

Output

name	default_version	installed_version
postgis	3.7.0rc2	3.7.0rc2
postgis_raster	3.7.0rc2	3.7.0rc2
postgis_sfcgal	3.7.0rc2	
postgis_topology	3.7.0rc2	

如果您正在数据库安装扩展程序，您将在 `installed_version` 列中看到提及。如果您没有收到任何行，意味着您的服务器上根本没有安装 `postgis` 扩展。PgAdmin III 1.14+ 将在数据库服务器的扩展部分中提供此信息，甚至允许通过右键单击行升或卸。

如果您有有效的扩展，可以使用 pgAdmin 扩展接口或通过运行以下 SQL 将 PostGIS 扩展安装到数据库：

Code

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

在 `psql` 中，你可以使用以下命令来查看你已安装的版本以及它所在的模式。

Code

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

Output

```
List of installed extensions
+-----+-----+
+--[ RECORD 1 ]-----+
| Name      | postgis          |
| Version   | 3.7.0rc2        |
| Schema    | public          |
| Description | PostGIS geometry, geography, and raster... |
+--[ RECORD 2 ]-----+
| Name      | postgis_raster   |
| Version   | 3.7.0rc2        |
| Schema    | public          |
| Description | PostGIS raster types and functions |
+--[ RECORD 3 ]-----+
| Name      | postgis_topology |
| Version   | 3.7.0rc2        |
| Schema    | topology        |
| Description | PostGIS topology spatial types and functions |
+-----+-----+
```

Warning

扩展表 `spatial_ref_sys`、`layer`、`topology` 不能独立备份。只有在备份的 `postgis` 或 `postgis_topology` 扩展才会一起备份，而只会在你备份整个数据库时生成。备份中只会包含那些不是 PostGIS 自己的 `srid` 值，所以不要修改我随 PostGIS 一起提供的条目并指望修改能长期保留——如果可能，提交工单。扩展表的值永远不会出现在备份里，因为它是通过 `CREATE EXTENSION` 建的，在同一扩展版本下默认值是一致的。这些行属于当前 PostgreSQL 扩展模型的固有机制。

如果你在没有使用扩展系数的情况下安装了 3.7.0rc2 版本，你可以通过运行以下命令将其基于扩展的安装方式，以将函数打包到各自的扩展中。在 PostgreSQL 13 中已移除了使用 `unpackaged` 安装方式，因此建议在升级到 PostgreSQL 13 之前切换到扩展构建方式。

Code

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

2.2.6 安装

要安装 PostGIS，运行以下命令

I Code

```
make check
```

此命令使用 `make check` PostgreSQL 数据生成的行各种和回。

**Note**

如果您使用非 PostgreSQL、GEOS 或 Proj 位置配置 PostGIS，您可能需要将它的位置添加到 `LD_LIBRARY_PATH` 环境变量中。

**Caution**

目前，**make check** 依赖于 `PATH` 和 `PGPORT` 环境变量来运行。它不是使用 **--with-pgconfig** 配置参数指定的 PostgreSQL。路径以匹配在配置期到的 PostgreSQL。或者不可避免的麻烦做好。

**Note**

沙箱式建号（非 PostgreSQL 超用）在回周期中，可以通过出 `POSTGIS_REGRESS_DB_OWNER` 来委托数据所有。框架会建立一个的回，并将其所有者指定的角色，同仍然使用限更低的角色行接。如果展的建角色需要与数据所有者不同，可以再配合使用 `POSTGIS_REGRESS_ROLE_EXT_CREATOR`。些量的作用是：在目 PostgreSQL 例允些委托角色安装展的前提下，自化境在不把用号提升超用的情况下，依然能走完整的升路径和展安装流程。

如果成功，`make check` 将生成近 500 个出的。果将似于以下（下面省略了多行）：

I Code

```
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

输出似乎以下内容:

Output

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

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

2.2.7 安装

要安装 PostGIS, 运行以下命令

Code

```
make install
```

会将 PostGIS 安装文件复制到由 **--prefix** 参数指定的相应子目录。以下是得注意的子目录:

- 加程序和程序二进制文件安装在 [prefix]/bin。
- SQL 文件 (例如 postgis.sql) 安装在 [prefix]/share/contrib 中。
- PostGIS 安装在 [prefix]/lib。

如果之前生成了 **make** 注, 运行以下命令来安装些 SQL 文件: postgis_comments.sql, raster_comments.sql

Code

```
make comments-install
```



Note

postgis_comments.sql, raster_comments.sql, topology_comments.sql 成 **xsiltproc** 的外部依赖关系, 因此它与正常的构建和安装分离。

2.3 常见问题

当安装或升级未按期进行, 需要做一些事。

PostgreSQL version mismatch.

确保已安装 PostgreSQL14 或更新版本, 并且你正在使用与正在运行的 PostgreSQL 版本相同的 PostgreSQL 源代码行。当你的 (Linux) 发行版已安装了 PostgreSQL, 或以其他方式安装了 PostgreSQL 但忘了, 可能会出混淆。PostGIS 只能与 PostgreSQL14 或更新版本一起使用, 如果使用旧版本, 可能会出奇且意想不到的消息。要正在运行的 PostgreSQL 版本, 可以使用 **psql** 连接到数据库并运行以下:

Code

```
SELECT version();
```

If you are running an RPM based distribution, you can check for the existence of pre-installed packages using the **rpm** command as follows:

Code

```
rpm -qa | grep postgresql
```

Restoring an upgrade into a database without PostGIS.

如果升级失败，确保原到已安装 PostGIS 的数据。

Code

```
SELECT PostGIS_Full_Version();
```

此外，配置是否正确到 PostgreSQL、Proj4 和 GEOS 的安装位置。

配置的输出用于生成 `postgis_config.h` 文件。POSTGIS_PGSQL_VERSION, POSTGIS_PROJ_VERSION 和 POSTGIS_GEOS_VERSION 量是否已正确置。

Chapter 3

PostGIS 管理

3.1 性能调优

PostGIS 性能的关注点与任何 PostgreSQL 工作负载的关注点非常相似。唯一需要额外考虑的是，几何形状和格式通常很大，因此与内存相关的优化通常对 PostGIS 的影响比其他类型的 PostgreSQL 负载更大。

有关优化 PostgreSQL 的一般信息，请参考 [调整 PostgreSQL 服务器](#)。

可以通过 `ALTER SYSTEM` 命令在服务器端设置 PostgreSQL 配置，这样就不用去改 `postgresql.conf` 或 `postgresql.auto.conf`。

Set `work_mem`, reload the configuration so new connections use the change, and display the current value. Use **SHOW ALL** to list all settings.

Code

```
ALTER SYSTEM SET work_mem = '256MB';
SELECT pg_reload_conf();
SHOW work_mem;
```

除了 Postgres 设置之外，PostGIS 有一些自定义设置，请参考 [Section 7.22](#)。

3.1.1 设置

这些设置在 `postgresql.conf` 中配置：

`constraint_exclusion`

- 默认：分区
- 一般用在分区表场景。默认值“`partition`”的含义是：只有当表位于继承链里时，优化器才会去做约束；否则就不分析，避免白白增加优化器的开销。

`shared_buffers`

- Default: ~128MB
- 将其设置为可用 RAM 的 25% 到 40%。在 Windows 上，您可能无法设置得那么高。

`max_worker_processes` 用来控制服务器可用的后台进程数量上限，限制能参与并行执行的工作进程数量。

- 默认值：8
- 设置系可以支持的最大后台进程数。该参数只能在服务器端设置。

3.1.2 并行

work_mem - 配置用于排序操作和复写的内存大小

- 默认：1-4MB
- 处理大型数据、复写、大量 RAM 并行调整
- 处理多并行或 RAM 低的情况并行下。
- 如果您有大量 RAM 而并行人很少：

Code

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

Maintenance_work_mem - 用于 VACUUM、CREATE INDEX 等的内存大小。

- 默认：16-64MB
- 通常太低 - 占用 I/O，在交叉内存固定象
- 建议在具有/大量 RAM 的服务器上使用 32 MB 到 1GB，但取决于并行数。如果您有大量 RAM 而并行人很少：

Code

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

max_parallel_workers_per_gather

PostgreSQL / PostGIS 的并行机制会根据配置来决定一个并行计划最多能申请多少个并行工作进程。只要把它改成大于 0，像 `ST_Intersects` 关系函数的就能利用多个进程来跑，速度有机会提升到原来的几倍以上。如果你的机器 CPU 富余，建议把配置到你有的处理器数量。同样要把 `max_worker_processes` 也一起提高，至少不小于个数。

- 默认:0
- 配置由每个 Gather 节点的最大工作进程数。并行工作进程来自 `max_worker_processes` 建立的进程池。注意，在并行，请求的工作进程数量可能无法使用。如果出现这种情况，计划将以比预期更少的工作进程行，可能效率低下。将此配置默认 0 将禁用并行行。

3.2 配置栅格支持

如果用了栅格的支持，您可能需要一下下面的内容来正确配置它。

Out-of-db 栅格（就是放在数据外部的 raster）和所有 raster 默认都是禁用的。要重新启用它，需要在服务器环境配置环境变量 `POSTGIS_GDAL_ENABLED_DRIVERS` 和 `POSTGIS_ENABLE_OUTDB_RASTERS`。你也可以采用更跨平台的方式，直接配置的 [Section 7.22](#)。

如果要启用离栅格：

Code

```
POSTGIS_ENABLE_OUTDB_RASTERS=1
```

如果包括任何其它或不包含，会禁用离栅格。

若想启用已安装的 GDAL 库，配置以下环境变量

Code

```
POSTGIS_GDAL_ENABLED_DRIVERS=ENABLE_ALL
```

如果只想用某些，按如下所示置境量：

Code

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



Note

在 Windows 上，不要在程序列表中加上引号

置境量的方式会因操作系而异。如果是在 Ubuntu 或 Debian 上通过 apt-postgresql 安装的 PostgreSQL，推荐的方式是在 `/etc/postgresql/MAJOR/CLUSTER/environment` 文件，其中占位符表示 PostgreSQL 的主版本号和集群名称。

在 Windows 上，如果您作服行，可以通过系量行置，于 Windows 7，您可以通过右键计算机 -> 属性高系置或在源管理器中航至 控制面板\所有控制面板目\系。然后点高系置->高->境量并添加新的系量。

置境量后，您需要重启 PostgreSQL 服才能使更改生效。

3.3 建空数据

3.3.1 使用展 (EXTENSION) 用空数据

如果你已并安装好了 extensions/postgis 模，就可以通过 EXTENSION 机制把一个普通数据成空数据。

postgis 的核心展包括几何、地理、spatial_ref_sys 以及所有函数和注。格 (Raster) 和拓扑 (Topology) 打包独的展模。

在要用空的数据中行以下 SQL 代段：

Code

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

3.3.2 不通展 (EXTENSION) 的方式用空数据 (不建)



Note

通常，当你不能或不想在 PostgreSQL 展的目中安装 PostGIS (例如，在或开期，或在受限境中)，才需要做。

将 PostGIS 对象和函数定义添加到数据库中是通过添加位于构建段指定的 `[prefix]/share/contrib` 中的各种 sql 文件来完成的。

核心 PostGIS 对象（几何和地理类型及其支持函数）位于 `postgis.sql` 脚本中。栅格对象位于 `rtpostgis.sql` 脚本中。拓扑对象位于 `topology.sql` 脚本中。

对于完整的 EPSG 坐标系定义符号集，您可以添加 `spatial_ref_sys.sql` 定义文件并填充 `spatial_ref_sys` 表。这将允许您几何形状进行 `ST_Transform()` 操作。

如果您希望向 PostGIS 函数添加注释，可以在 `postgis_comments.sql` 脚本中找到它们。只需从 **psql** 终端窗口输入 `\dd [function_name]` 即可查看注释。

在终端中运行以下 shell 命令：

Code

```
DB=[yourdatabase]
SCRIPTSDIR=`pg_config --sharedir`/contrib/postgis-3.6/

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

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

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

3.4 升级空数据库

当您需要替换或部署新的 PostGIS 对象定义时，升级有空数据库可能会很棘手。

遗憾的是，并非所有定义都可以在正在运行的数据库中轻松替换，因此卸载/重装可能是最好的选择。

PostGIS 次要版本升级和修复提供升级，主要升级提供硬升级。

在升级 PostGIS 之前备份数据始终是明智的。可以将 `-fc` 标志与 `pg_dump` 一起使用，以始终通过硬升级还原。

3.4.1 升级

如果使用扩展安装数据库，您必须使用扩展模型其运行升级。如果使用旧的 SQL 脚本运行安装，则切换到扩展，因为它不再支持 SQL 脚本。

3.4.1.1 使用 EXTENSION 升级

如果你最初是用 EXTENSION 的方式安装 PostGIS，那升级的时候也必须使用 EXTENSION 的方式。用 EXTENSION 做小版本升级一般都比省心，过程也挺简单。

使用 `PostGIS_Extensions_Upgrade` 函数可以把 PostGIS 升级到你当前已安装的最新版本。

Code

```
SELECT postgis_extensions_upgrade();
```

如果你要升级的数据已经在跑 PostGIS 2.5, 先运行一次 **ALTER EXTENSION**, 然后再用 `postgis_extensions_upgrade()` 可以把旧的 `postgis_raster` 留象合并回主展, 避免残留在外面。

Code

```
ALTER EXTENSION postgis UPDATE;  
SELECT postgis_extensions_upgrade();
```

This second call is needed to rebundle the `postgis_raster` extension.

Code

```
SELECT postgis_extensions_upgrade();
```

如果安装了多个版本的 PostGIS, 并且不想升级到最新版本, 可以指定式版本。行以下操作:

Code

```
ALTER EXTENSION postgis UPDATE TO "3.7.0rc2";  
ALTER EXTENSION postgis_topology UPDATE TO "3.7.0rc2";
```

您可能会看到似于以下内容的通知:

Output

```
No migration path defined for b'...' to 3.7.0rc2
```

在这种情况下, 必备份数据, 按照 Section 3.3.1 中所述生成新数据, 然后将备份原到新数据。

您可能会收到似于以下内容的消息:

Output

```
Version "3.7.0rc2" of extension "postgis" is already installed
```

那么一切都已是最新的, 您可以安全地忽略它。除非您正在从开版本升级到下一个版本 (不会得新的版本号); 在这种情况下, 您可以将 “next” 附加到版本字符串, 下次您需要再次除 “next” 后:

Code

```
ALTER EXTENSION postgis UPDATE TO "3.7.0rc2next";  
ALTER EXTENSION postgis_topology UPDATE TO "3.7.0rc2next";
```

**Note**

如果您最初安装 PostGIS 没有指定版本, 通常可以在恢复之前跳 `postgis` 展的重新安装, 因备份只有 `CREATE EXTENSION postgis`, 因此在恢复期会最新本。

**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.

I Code

```
DROP EXTENSION postgis_raster;
```

3.4.1.2 不使用 EXTENSION 升级

展示是 PostGIS 官方支持的安装方式。如果你在依赖非打包安装，那就需要进行一次 **HARD UPGRADE**，在一个用展示方式建的新库里完成升级，然后把数据迁移去。

3.4.1.3 硬升级

硬升级意味着完全删除/重新加载 PostGIS 中可用的数据。如果 PostGIS 对象的内部状态发生变更或无法升级，则需要硬升级。附录中的 [行说明](#) 指示每个版本是否需要删除/重新加载（硬升级）。

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 in the [PostGIS Windows installation guide](#).

程序如下：

1. 创建要升级的数据库的“自定义格式”备份（我称之为 `olddb`），包括二进制 blob (-b) 和视图 (-v) 输出。用它可以是数据库的所有者，不必是 postgres 超级用户。

I Code

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

2. 在新数据库中全新安装 PostGIS - 我将此数据库称为 `newdb`。请参考 [Section 3.3.2](#) 和 [Section 3.3.1](#) 了解如何执行此操作的说明。

备份中的 `spatial_ref_sys` 将保留，但不会覆盖现有 `spatial_ref_sys`。这是为了确保正式数据集的更正被送到要恢复的数据库。如果要覆盖旧条目，只需在生成 `newdb` 时不要加载 `spatial_ref_sys.sql` 文件。

如果您的数据库确实很旧，或者您知道在备份和函数中使用了过期不推荐使用的函数，您可能需要所有函数和视图等加载 `legacy.sql` 才能正确返回。在确实需要时才执行此操作。如果可能的话，考虑在备份之前升级您的数据库和函数。稍后可以通过加载 `uninstall_legacy.sql` 来删除已弃用的函数。

3. 使用 `postgis_restore` 将备份恢复到新的 `newdb` 数据库中。意外错误（如果有）将由 `psql` 打印到标准输出流。查看一些内容。

I Code

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

在以下情况下可能会生成错误：

1. 您的某些视图或函数使用已弃用的 PostGIS 对象。为了解决这个问题，您可以在恢复之前加载 `legacy.sql` 脚本，或者您必须恢复到仍包含这些对象的 PostGIS 版本，并在移植代换后再次迁移。如果 `legacy.sql` 方式适合您，不要忘记修复您的代码以停止使用已弃用的函数并将其加入 `uninstall_legacy.sql`。
2. 一些在备份文件中的自定义空参考系可能具有无效的 SRID。有效的 SRID 必须大于 0 且小于 999000。在 999000.999999 范围内被保留供内部使用，而大于 999999 的 SRID 根本不能使用。所有具有无效 SRID 的自定义视图将被保留，其中大于 999999 的将被移到保留范围内，但空 `_ref_sys` 表将失去其条件约束，可能会失去其主键（当多个无效的 SRID 映射到相同的保留 SRID 时）。为了解决此问题，您将自定义 SRS 复制到具有有效 SRID（可能在 910000..910999 范围内），将所有表赋予新的 srid（请参考 [UpdateGeometrySRID](#)），删除无效的从 `spatial_ref_sys` 表并重建它：

Code

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

Code

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

如果要升级包含法国 **IGN** 映射的旧数据，可能会超出 SRID 的范围，并在输入数据时遇到以下警告：

Code

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

在这种情况下，您可以执行以下步骤：完全丢弃最初从 SQL `postgis_restore` 中生成的 IGN。因此，执行以下命令：

Code

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

执行以下命令：

Code

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

Create your newdb, activate the required PostGIS extensions, and insert the French IGN spatial reference definitions using [this script](#). After these operations, import your data:

Code

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

Chapter 4

数据管理

4.1 空型数据模型

4.1.1 OGC 几何类型

开放地理空间联盟 (OGC) 定义了 [简单功能抽象模型 \(SFA\)](#)，以提供地理空间数据模型。它定义了几何的基本空型，以及操作和几何以进行空间分析任意的操作。PostGIS 将 OGC 几何模型映射到 PostgreSQL 数据类型 [geometry](#) 和 [geography](#)。

几何是一种抽象类型。几何属于其具体子类型之一。子类型表示各种类型和各种维度的几何形状。其中包括基本类型点、线、多边形和面，以及集合类型多点、多线、多面和多面体。OGC 功能抽象模型 - 第一部分：通用体系结构 [v1.2.1](#) 定义了 [PolyhedralSurface](#)、[Triangle](#) 和 [TIN](#) 添加了子类型。

几何类型在二维笛卡尔平面上进行建模。PolyhedralSurface、Triangle、和 TIN 也可以表示三维空间中的形状。形状的大小和位置由坐标指定。每个坐标都有 X 和 Y 坐标值，用于确定其在平面上的位置。形状由点和线构造，点由两个坐标定义，线由三个坐标定义。

Coordinates may contain optional Z and M ordinate values. The Z ordinate is often used to represent elevation. The M ordinate contains a measure value, which may represent time or distance. If Z or M values are present in a geometry value, they must be defined for each point in the geometry. If a geometry has Z or M ordinates the **coordinate dimension** is 3D; if it has both Z and M the coordinate dimension is 4D. The coordinate dimension is at least 2, because every geometry has at least X and Y coordinates.

几何与空参考系指示它所嵌入的坐标系。空参考系由几何 SRID 编号指定。X 和 Y 的位由空参考系确定。平面参考系中，X 和 Y 坐标通常表示东向和北向，而在大地测量量系中，它表示经度和纬度。SRID 0 表示一个具有无位的无限笛卡尔平面。参见 [Section 4.5](#)。

The geometry **dimension** is a property of geometry types. Point types have dimension 0, linear types have dimension 1, polygonal types have dimension 2, and solid types have dimension 3. Collections have the dimension of the maximum element dimension.

几何可以空。空不包含点（属于原子几何类型）或不包含元素（属于集合）。

几何的一个重要属性是它的空范围或边界框，OGC 模型将其称为包络。它是包含几何坐标的 2 或 3 框。它是在空中表示几何形状并指定几个几何形状是否相互作用的有效方法。

The geometry model allows evaluating topological spatial relationships as described in [Section 5.1.1](#). To support this the concepts of **interior**, **boundary** and **exterior** are defined for each geometry type. Geometries are topologically closed, so they always contain their boundary. The boundary is a geometry of dimension one less than that of the geometry itself. For example, the boundary of a Polygon is its rings (LineStrings), the boundary of a LineString is its endpoints (Points), and the boundary of a Point is empty.

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

4.1.1.1 点 (Point)

点是表示空间中个位置的 0 几何形。

Code

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

4.1.1.2 串 (LineString)

串是由的段序列形成的一。每条段由个点定，一条段的点形成下一条段的起点。OGC 有效的串具有零个或个或多个点，但 PostGIS 也允个点串。串可以与自身交叉（自相交）。如果起点和点相同，串是合的。如果串不自相交，它是的。

Code

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

4.1.1.3 性 (LinearRing)

性是一个既封又的串。第一个点和最后一个点必相等，并且不得自相交。

Code

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

4.1.1.4 多形 (Polygon)

多形是由外部界 (壳) 和零个或多个内部界 (洞) 分隔的二平面区域。每个界都是 性。

Code

```
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 多点 (MultiPoint)

多点是点的集合。

Code

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

4.1.1.6 多 (MultiLineString)

多是串的集合，如果多的每个元素都已合，多合。

Code

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

4.1.1.7 多面 (MultiPolygon)

A MultiPolygon is a collection of non-overlapping, non-adjacent Polygons. Polygons in the collection may touch only at a finite number of points. (Two polygons are adjacent if they share an edge. They touch if they share only points or edges on their boundaries). For more details on MultiPolygon validity, see Section 4.4.

Code

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

4.1.1.8 几何集合 (GeometryCollection)

几何集合是几何的异或 (混合) 集合。

Code

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

4.1.1.9 多面体曲面 (PolyhedralSurface)

多面体曲面是共享一些面的斑或刻面的面集合。每个面片都是一个平面多边形。如果多边形坐标具有 Z 坐标，表面是三面的。

Code

```
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 三角形 (Triangle)

三角形是由三个不同的非共面点定义的多边形。因为三角形是一个多边形，所以它由四个坐标指定，第一和第四个相等。

Code

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

4.1.1.11 TIN

TIN 是一组不重叠的三角形，表示了一个三角不规则网 (Triangulated Irregular Network) 的面。

Code

```
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 第 3 部分-曲线

ISO/IEC 13249-3 SQL 多媒体-空型 标准 (SQL/MM) 展示了 OGC SFA 以定义包含弯曲几何形状的子型。SQL/MM 型支持 XYM、XYZ 和 XYZM。

**Note**

SQL-MM 中的所有浮点比按照指定的公差行，目前 1E-8。

4.1.2.1 弧串 (CircularString)

弧串是基本曲线型，类似于线性世界中的串。每个弧段由三个点指定：起点和点（第一个和第三个）以及弧上的其他点。要指定合，起点和点相同，中点是直径上的相点（即弧的中心）。在弧序列中，前一个弧的点是下一个弧的起点，就像串的一段一样。这意味着弧串必须具有大于 1 的奇数个点。

Code

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

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

4.1.2.2 复合曲线 (CompoundCurve)

复合曲线是一条曲线，可以包含弧段和线性段。意味着除了具有格式良好的件之外，每个件（最后一个除外）的点必须与后件的起点重合。

Code

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

4.1.2.3 曲线多边形 (CurvePolygon)

曲线多边形与多边形相似，因为它有一个外和零个或多个内。区别在于多边形是串，而曲线多边形是复合曲线或弧串。

从 PostGIS 1.4 开始，PostGIS 在支持曲线多边形上的复合曲线。

Code

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

示例：具有由弧串和串成的复合曲线定义的外壳的曲线多边形，以及由弧串定义的孔

Code

```
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 多曲线 (MultiCurve)

多曲线是曲线的集合，可以包括线串、弧串或复合曲线。

Code

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

4.1.2.5 多曲面 (MultiSurface)

多曲面是曲面的集合，可以是（面的）多边形或曲面多边形。

Code

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

4.1.3 WKT 和 WKB

OGC SFA 规范定义了两种准格式，用于表示供外部使用的几何。已知文本（WKT）和已知二进制文件（WKB）。WKT 和 WKB 都包含有关定义几何对象的信息。

已知文本（WKT）提供空数据的准字符表示形式。以下是空几何对象的 WKT 表示形式示例：

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

WKT 输入和输出由函数 `ST_AsText` 和 `ST_GeomFromText` 提供：

Code

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

For example, a statement to create and insert a spatial object from WKT with SRID 312 is:

Code

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

已知二进制（WKB）是空数据二进制数据（字节数）的可移植且准确的表示形式。空对象的 WKB 表示如下所示：

- WKT: POINT(1 1)
WKB: 010100000000000000000000F03F000000000000F03
- WKT: LINESTRING (2 2, 9 9)
WKB: 01020000000200000000000000000000400000000000000040000000000000224000000000000000

WKB 的输入和输出由以下函数 **ST_AsBinary** 和 **ST_GeomFromWKB** 提供：

Code

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

例如，用于创建和插入 WKB 的空对象如下所示：

Code

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

4.2 几何数据类型

PostGIS 通常称 **geometry** 的 PostgreSQL 数据类型来遵循 OGC 要素模型。它使用内部类型代表示所有几何子类型（参 **GeometryType** 和 **ST_GeometryType**）。允许将空要素建模用 **geometry** 类型的列定义的表行。

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

**Note**

PostGIS 遵循 SFA 标准，在空函数前面加上前缀 “ST_”。表示 “空和”，但标准部分尚未开发。相反，它可以解 “空型”。

SFA 标准指定空对象包含空参考系标识符 (SRID)。创建插入数据的空对象需要 SRID（可能默认为 0）。参 **ST_SRID** 和 Section 4.5

为了使几何更加高效，PostGIS 定义了不同类型的空索引。有关更多信息，参 Section 4.9 和 Section 5.2。

4.2.1 PostGIS EWKB 和 EWKT

OGC SFA 规范最初支持 2D 几何形状，并且几何 SRID 不包含在输入/输出表示中。OGC SFA 规范 1.2.1（与 ISO 19125 规范一致）增加了 3D（XYZ）和量（XYM 和 XYZM）坐标的支持，但仍然不包括 SRID。

由于这些限制，PostGIS 定义了扩展的 EWKB 和 EWKT 格式。它提供 3D（XYZ 和 XYM）和 4D（XYZM）坐标支持，并包括 SRID 信息。包含所有几何信息允许 PostGIS 使用 EWKB 作为二进制格式（例如，在数据库文件中）。

EWKB 和 EWKT 用于 PostGIS 数据对象的“规范形式”。对于输入，二进制数据的规范形式是 EWKB，对于文本数据，接受 EWKB 或 EWKT。允许通过使用 `::geometry` 将 HEXEWKB 或 EWKT 中的文本转换为几何来构建几何。对于输出，二进制的规范形式是 EWKB，对于文本，规范形式是 HEXEWKB（十六进制制的 EWKB）。

例如，此程序使用来自 EWKT 文本的制式生成几何图形，并使用 HEXWKB 的规范格式输出它：

Code

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

Output

```
geometry
-----
0101000020040000000000000000000000000000000000000000000000000000
```

PostGIS EWKT 输出与 OGC WKT 有一些不同之处：

- 对于 3DZ 几何形状，省略了 Z 限定符：
OGC: POINT Z (1 2 3)
EWKT: POINT (1 2 3)
- 对于 3DM 几何形状，包括 M 限定符：
OGC: POINT M (1 2 3)
EWKT: POINTM (1 2 3)
- 对于 4D 几何形状，省略了 ZM 限定符：
OGC: POINT ZM (1 2 3 4)
EWKT: POINT (1 2 3 4)

EWKT 避免角度指定度以及 OGC/ISO 格式可能产生的不一致，例如：

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



Caution

PostGIS 扩展格式向上兼容 OGC 格式，因此，每个有效的 OGC WKB/WKT 也是有效的 EWKB/EWKT。但是，将来可能会有所不同，如果 OGC 以与 PostGIS 定义冲突的方式扩展格式。因此，您不应依赖这种兼容性！

空对象的 EWKT 文本表示的示例如下：

- POINT(0 0 0) -- XYZ
- SRID=32632;POINT(0 0) -- XY 与 SRID
- POINTM(0 0 0) -- XYM
- POINT(0 0 0 0) -- XYZM

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

可以使用以下函数使用这些格式的输入和输出：

Code

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

例如，使用 EWKT 创建和插入 PostGIS 空几何对象的语句：

Code

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

4.3 地理数据类型

PostGIS geography 数据类型“地理”坐标（有称“大地”坐标，或“lat/lon”或“lon/lat”）表示的空要素提供本机支持。地理坐标是以角度单位（度）表示的球坐标。

PostGIS 几何类型的基是平面。平面上点之间的最短路径是直线。几何函数（面积、距离、角度、交点等）是使用线性向量和笛卡儿平面计算的。这简化了操作并运行得更快，但对于地球球面以上的数据是不准确的。

PostGIS 地理数据类型基于球面模型。球体上点之间的最短路径是一个大圆弧。地理上的函数（面积、距离、角度、交点等）是使用球体上的弧计算的。通过考虑世界的球体形状，函数可提供更准确的结果。

由于基数学更复杂，因此地理类型定义的函数少于几何类型定义的函数。随着库的推移，随着新算法的添加，地理类型的功能将扩展。作为一种解决方法，可以在几何和地理类型之间来回切换。

与 geometry 类型类似，geography 数据也通过空参考系标识符(SRID)关联到空参考系。spatial_ref_sys 表中定义的任何大地测量（基于弧度）的空参考系都可以使用，也可以按 Section 4.5.2 所述添加自定义的大地测量空参考系。

测量函数返回的单位（例如，ST_Distance, ST_Length, ST_Perimeter, ST_Area）与ST_DWithin参数中给出的距离之单位的空参考系单位米。

4.3.1 创建地理表

您可以使用有地理类型列的 CREATE TABLE SQL 语句创建一个表来存储地理数据。以下示例创建一个表，其中包含存储 WGS84 大地坐标系 (SRID 4326) 中的 2D LineString 的地理列：

Code

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

地理类型支持四个可变的类型修饰符：

- 空类型修饰符用于列中允许的形状的类型和维度。根据它，空类型可以是点、线串、面、多点、多线、多面、几何集合。地理类型不支持曲面、三角形或多面体曲面。通常将后置 Z、M 和 ZM 附加到类型修饰符，可以解决坐标度的约束。例如，“LINESTRINGM”只允许三线串，第三个是 M。同样，“POINTZM”需要四维 (XYZM) 数据。
- SRID 修饰符将空参考系 SRID 限制为特定数字。如果省略，SRID 默认为 4326 (WGS84 大地测量)，并且所有计算均使用 WGS84 行。

以下是生成包含地理列的表的示例：

生成一个具有二维点地理的表，默认情况下 SRID 为 4326 (WGS84 度/度)：

Code

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

生成具有 NAD83 二度二维点地理的表：

Code

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

创建一个包含 3D (XYZ) POINT 和默认 SRID 4326 的表：

Code

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

生成具有二维线串地理的表，SRID 默认为 4326：

Code

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

使用 SRID 4267 (NAD 1927 二度) 创建包含二维多边形地理的表：

Code

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

地理字段在 geography_columns 系表中注册。您可以查看 geography_columns 并看到表已列出：

Code

```
SELECT * FROM geography_columns;
```

创建索引的工作方式与创建几何列相同。PostGIS 将注意到列类型是 GEOGRAPHY，并创建适当的基于球体的索引，而不是用于 GEOMETRY 的常用平面索引。

Code

```
CREATE INDEX global_points_gix ON global_points USING GIST ( location );
```

4.3.2 使用地理表

您可以采用与几何相同的方式将数据插入地理表中。如果几何数据具有 SRID 4326，几何数据将自动转换为地理类型。也可以使用 **EWKT** 和 **EWKB** 格式以指定地理类型。

Code

```
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)');
```

`spatial_ref_sys` 表中列出的任何大地测量（经/纬度）空间参考系都可以指定地理 SRID。如果使用非大地坐标系，则会生成警告。

A NAD83 longitude/latitude geography value:

Code

```
SELECT 'SRID=4269;POINT(-123 34)::geography;
```

Output

```
geography
-----
0101000020AD100000000000000000C05EC000000000000004140
```

A NAD27 longitude/latitude geography value:

Code

```
SELECT 'SRID=4267;POINT(-123 34)::geography;
```

Output

```
geography
-----
0101000020AB100000000000000000C05EC000000000000004140
```

NAD83 UTM zone meters fails because it is a meter-based planar projection.

Code

```
SELECT 'SRID=26910;POINT(-123 34)::geography;
```

Output

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

面积和长度功能以米为单位。因此，距离参数以米为单位（面积以平方米）。

This query finds rows within a 1000 km tolerance:

Code

```
SELECT name FROM global_points WHERE ST_DWithin(location, 'SRID=4326;POINT(-110 29)::geography, 1000000);
```

通过计算一架从西雅图飞往伦敦的飞机（`LINESTRING(-122.33 47.606, 0.0 51.5)`）表示的大圆航线与行程中距离雷克雅未克（`POINT(-21.96 64.15)`）的距离，您可以看到地理学的作用（**限制路**）。

地理类型计算出雷克雅未克与西雅图和伦敦之间的大圆行路径之圆的球体上的真正最短距离 122.235 公里。

Code

```
SELECT ST_Distance('LINESTRING(-122.33 47.606,0.0 51.5)::geography', 'POINT(-21.96 64.15)'::geography);
```

Output

```
st_distance
-----
122235.23815667
```

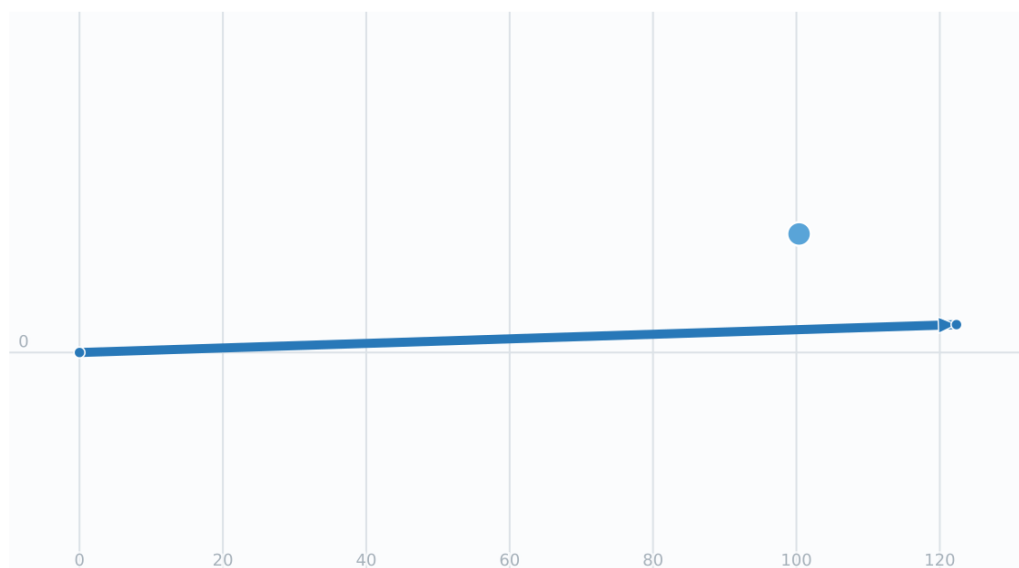
几何类型在平面世界地图上计算雷克雅未克之岛的笛卡儿距离以及西雅图和伦敦之间的直线，这是没有意义的。计算结果的单位是“度”，但它与点之间的真实角度差并不匹配，称其“度”本身是不准确的。

Code

```
SELECT ST_Distance('LINESTRING(-122.33 47.606,0.0 51.5)::geometry', 'POINT(-21.96 64.15)::geometry');
```

Output

```
st_distance
-----
13.342271221453624
```

Figure

■ Code 1 ■ Code 2

4.3.3 何时使用地理数据类型

地理数据类型允许您将数据存储在经度/纬度坐标中，但有代价：在地理上定义的函数比在几何上定义的函数少；定义的那些函数需要更多 CPU 时间来执行。

您的数据由正在构建的应用程序的工作区域确定。您的数据是跨越全球还是大片大区域，是州、县或直辖区的本地数据？

- 如果数据包含在小的区域中，您可能会发现，就可用的性能和功能而言，选择合适的投影并使用 GEOMETRY 是最佳解决方案。

- 如果您的数据是整个地球或大圆，您将能建立一个系，而无需担心地理投影的。保存度/度数据并使用地理中定义的函数。
- 如果您不了解投影，并且不想了解它，并且您准备接受地理中可用功能的限制，那么使用地理可能比使用几何更容易。只需将数据加度/度，然后从那里开始。

有关地理和几何之的支持比，参 Section 13.11。有关地理函数的要列表和明，参 Section 13.4

4.3.4 地理高常解答

1. 你是用球体还是球体算？

By default, all distance and area calculations are done on the spheroid. You should find that the results of calculations in local areas match up well with local planar results in good local projections. Over larger areas, the spheroidal calculations will be more accurate than any calculation done on a projected plane. The difference is that the spheroid takes into account the curvature and non-spherical shape of the Earth, whereas planar calculations assume a flat surface. 所有地理函数都可以使用球体算，方法是将最布参数置“FALSE”。将在一定程度上加快算速度，特别是对于几何形状非常圆的情况。

2. 日期变更和极点呢？

所有算都没有日期变更或极点的概念。由于坐是球体（度/度），因此从算的角度来看，穿日期变更的形状与其他任何形状没有什么不同。

3. 您可以理的最弧是多少？

我使用大弧作点之间的“插”。意味着任何点在上以种方式接起来，具体取决于您沿着大行的方向。我所有的代都假设些点是由沿着大圆的条路径中“短的”一条接起来的。因此，弧度超过 180 度的形状将无法正确建模。

4. 什么算欧洲/俄罗斯/大地理区域的面么慢？

因多形大得离谱。大面不好有个原因一个原因是界框很大，因此无您行什么，索引都向于拉取特征。一个原因是点很多，并且（距离、包含）函数必须至少遍点一次，通常 N（其中 N 是一个要素的点数）。与 GEOMETRY 一样，我建您在具有非常大的多形，但在小区域中行，将几何数据“非范化”小的，以便索引可以有效地子象的各个部分，就不必每次都提取整个象。参 ST_Subdivide 函数文档。因您 * 可以 * 将整个欧洲存在一个多形中并不意味着您 * 。

4.4 几何有效性

PostGIS 符合开放地理空间联盟 (OGC) 的要素规范。规范定义了和有效几何概念。些定义允要素几何模型以一致且明确的方式表示空象，从而支持高效算。（注：OGC SF 和 SQL/MM 规范和有效的定义相同。）

4.4.1 几何

几何是指没有异常几何点（例如自交或自相切）的几何。

作 0 几何象，POINT 本上是。

MULTIPOINT 是的，如果没有个坐（POINT）的坐相同（具有相同的坐）。

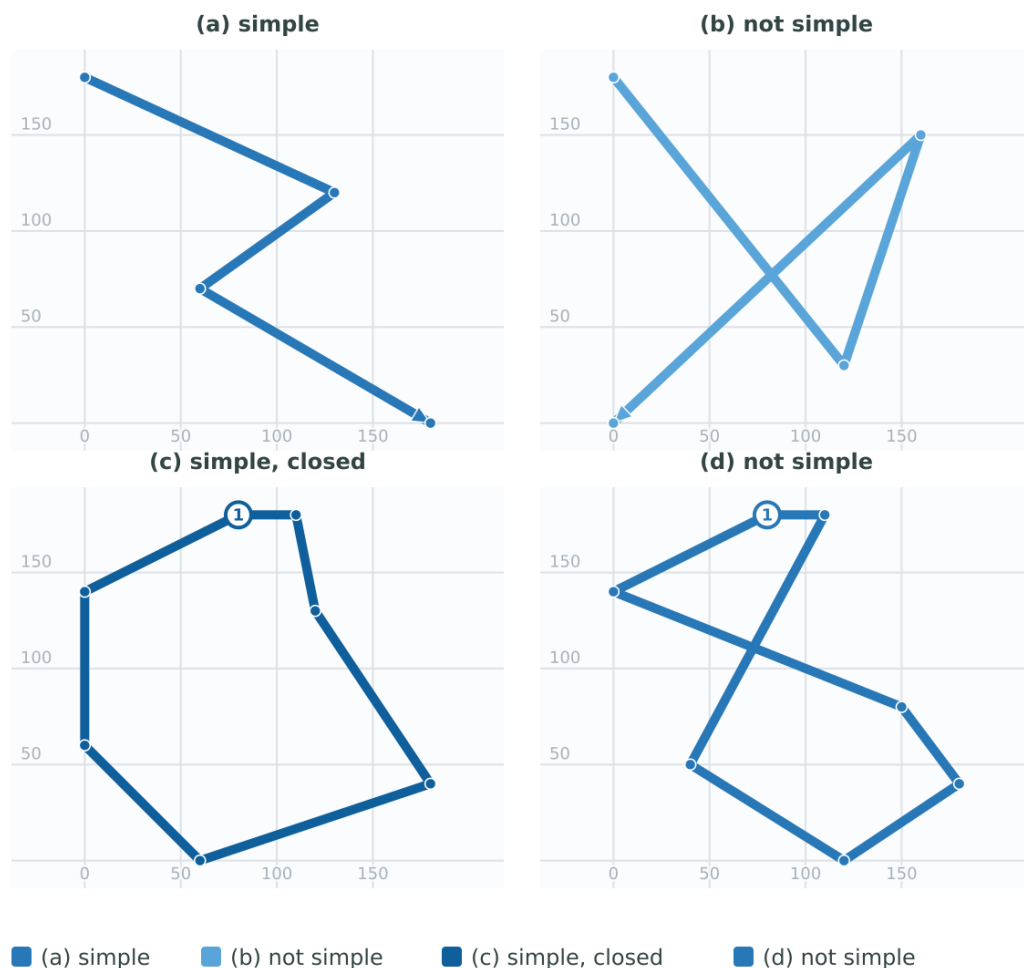
如果 LINESTRING 不次穿同一点（端点除外），很。如果串的端点相同，称合并称性。

Use ST_IsSimple to classify linear geometries. Each labeled result below is returned only when the function agrees with the stated classification; example (c) also verifies that the simple LineString is closed.

Code

```
WITH cases(label, geom) AS (VALUES
  ('a', 'LINESTRING (10 190, 140 130, 70 80, 190 10)::geometry'),
  ('b', 'LINESTRING (10 190, 130 40, 170 160, 10 10)::geometry'),
  ('c', 'LINESTRING (90 190, 120 190, 130 140, 190 50, 70 10, 10 70, 10 150, 90 190):: ←
    geometry'),
  ('d', 'LINESTRING (90 190, 120 190, 50 60, 130 10, 190 50, 160 90, 10 150, 90 190):: ←
    geometry')
)
SELECT
  (SELECT geom FROM cases WHERE label = 'a' AND ST_IsSimple(geom))
  AS "input_(a) simple",
  (SELECT geom FROM cases WHERE label = 'b' AND NOT ST_IsSimple(geom))
  AS "input_(b) not simple",
  (SELECT geom FROM cases
    WHERE label = 'c' AND ST_IsSimple(geom) AND ST_IsClosed(geom))
  AS "input_(c) simple, closed",
  (SELECT geom FROM cases WHERE label = 'd' AND NOT ST_IsSimple(geom))
  AS "input_(d) not simple";
```

Figure



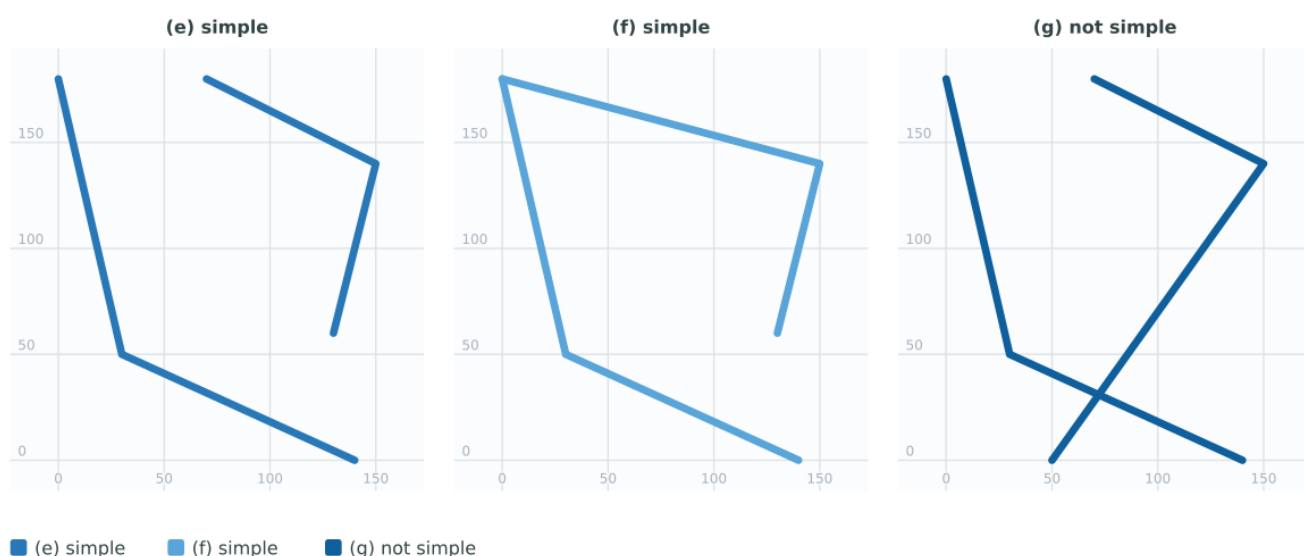
当 **MULTILINESTRING** 的所有元素都很简单并且任意两个元素之唯一的交集生在两个元素界上的点，**MULTILINESTRING** 才是简单的。

The next example applies the same classification to the three MultiLineStrings. In **(f)** the component lines meet only at a shared endpoint, while in **(g)** they cross in their interiors.

Code

```
WITH cases(label, geom) AS (VALUES
  ('e', 'MULTILINESTRING ((30 190, 60 60, 170 10), (100 190, 180 150, 160 70))'::geometry),
  ('f', 'MULTILINESTRING ((30 190, 60 60, 170 10), (30 190, 180 150, 160 70))'::geometry),
  ('g', 'MULTILINESTRING ((30 190, 60 60, 170 10), (100 190, 180 150, 80 10))'::geometry)
)
SELECT
  (SELECT geom FROM cases WHERE label = 'e' AND ST_IsSimple(geom))
  AS "input_(e) simple",
  (SELECT geom FROM cases WHERE label = 'f' AND ST_IsSimple(geom))
  AS "input_(f) simple",
  (SELECT geom FROM cases WHERE label = 'g' AND NOT ST_IsSimple(geom))
  AS "input_(g) not simple";
```

Figure



POLYGON 是由线性环形成的，所以常见的多边形几何是有效的。

一般来说，PostGIS 函数不要求几何参数很有效。有效性主要用作定义几何有效性的基础。有效性也是某些类型的空数据模型的要求（例如，线性网通常不允许交叉）。使用 **ST_UnaryUnion** 可以使多点和线性几何有效。

4.4.2 有效的几何形状

几何有效性主要适用于二维几何形状（POLYGON 和 MULTIPOLYGON）。有效性是由允许多边形几何形状明确地模拟平面区域的基础定义的。

POLYGON 以下条件是有用的：

1. 多边形边界（外壳和内孔）很有效（不交叉或自接触）。因此，多边形不能有切割、尖峰或。这意味着多边形孔必须表示内孔，而不是外孔自接触（所谓的“倒孔”）。
2. 边界不交叉
3. 边界可以在点上接触，但只能作切线（即不在一条线上）
4. 内孔包含在外孔中
5. 多边形内部接触（即不得以将多边形分成多个部分的方式接触）

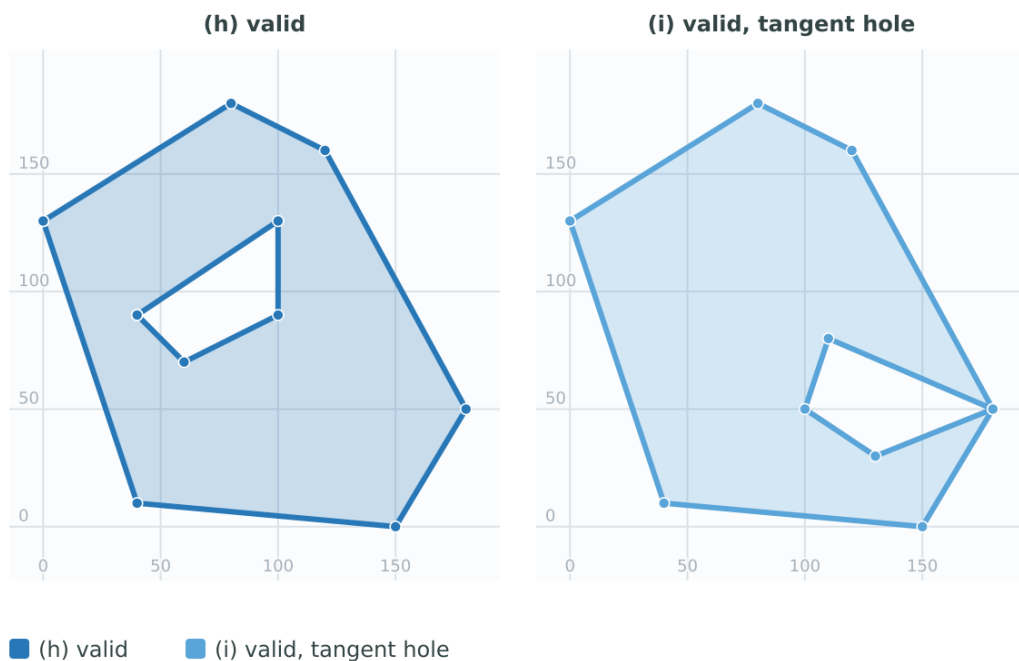
Consecutive repeated points in a polygon boundary ring do not make the ring non-simple. Under the SFS curve model, a curve is interpreted by a continuous parameterization; consecutive equal coordinates are topologically indistinguishable from a single vertex. PostGIS therefore treats them as redundant vertices, so they do not make an otherwise valid polygon invalid.

The following examples use **ST_IsValid** to check the validity rules. Each labeled geometry is returned only if its computed validity agrees with the label.

Code

```
WITH cases(label, geom) AS (VALUES
  ('h', 'POLYGON ((10 140, 90 190, 130 170, 190 60, 160 10, 50 20, 10 140),
    (50 100, 70 80, 110 100, 110 140, 50 100))'::geometry),
  ('i', 'POLYGON ((10 140, 90 190, 130 170, 190 60, 160 10, 50 20, 10 140),
    (190 60, 140 40, 110 60, 120 90, 190 60))'::geometry)
)
SELECT
  (SELECT geom FROM cases WHERE label = 'h' AND ST_IsValid(geom))
  AS "input_(h) valid",
  (SELECT geom FROM cases WHERE label = 'i' AND ST_IsValid(geom))
  AS "input_(i) valid, tangent hole";
```

Figure



Examples **(j-m)** are invalid. They show, respectively, a disconnected interior, crossing rings, a cut line, and a hole outside the shell. The invalidity reason is checked with **ST_IsValidReason**. The disconnected interior in **(j)** can instead be represented as a valid **MULTIPOLYGON**.

Code

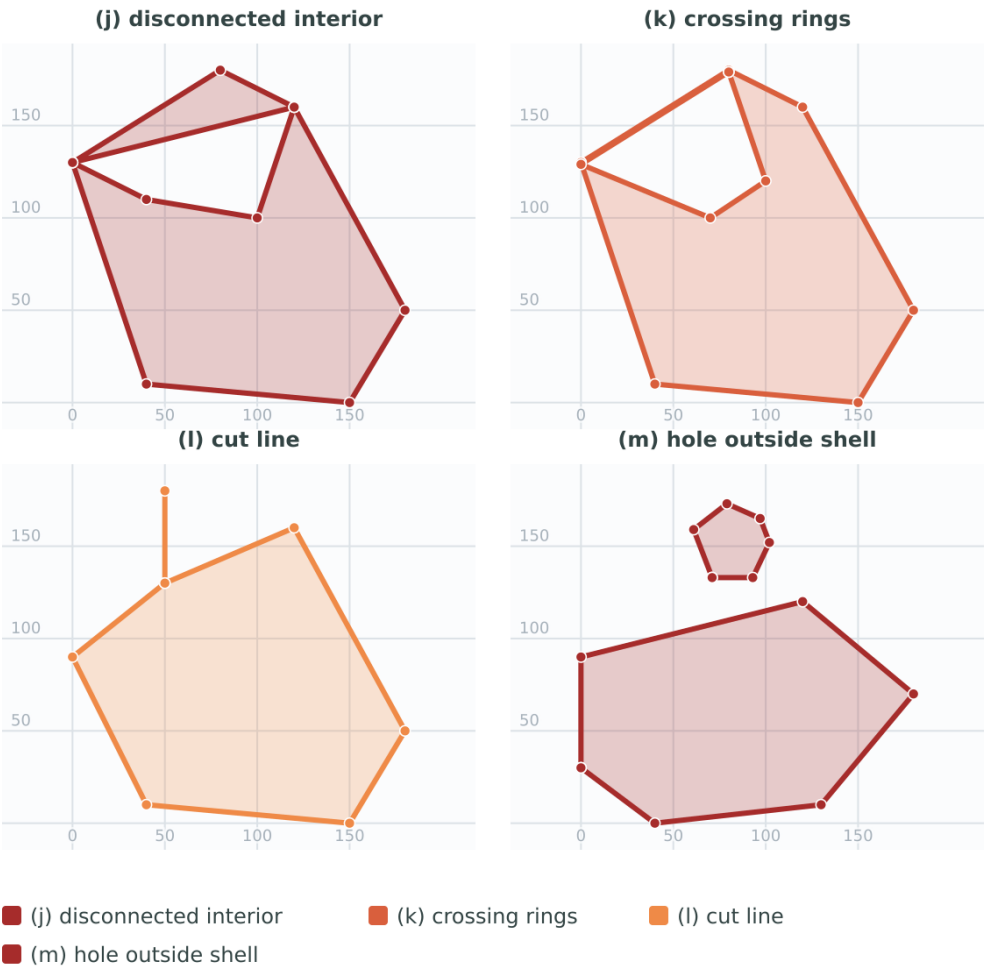
```
WITH cases(label, geom, reason_pattern) AS (VALUES
  ('(j) disconnected interior',
    'POLYGON ((10 140, 90 190, 130 170, 190 60, 160 10, 50 20, 10 140),
      (130 170, 10 140, 50 120, 110 110, 130 170))'::geometry,
    'Interior is disconnected%'),
  ('(k) crossing rings',
    'POLYGON ((10 140, 90 190, 130 170, 190 60, 160 10, 50 20, 10 140),
      (90 189, 10 139, 80 110, 110 130, 90 189))'::geometry,
```

```
'Self-intersection%'),
(' (l) cut line',
 'POLYGON ((10 100, 60 140, 60 190, 60 140, 130 170,
            190 60, 160 10, 50 20, 10 100))'::geometry,
 'Ring Self-intersection%'),
(' (m) hole outside shell',
 'POLYGON ((10 40, 10 100, 130 130, 190 80, 140 20, 50 10, 10 40),
            (81 143, 103 143, 112 162, 107 175, 89 183, 71 169, 81 143))'::geometry,
 'Hole lies outside shell%')
)
SELECT label, geom
FROM cases
WHERE NOT ST_IsValid(geom)
      AND ST_IsValidReason(geom) LIKE reason_pattern
ORDER BY label;
```

Output

label	geom
(j) disconnected interior	POLYGON((10 140,90 190,130 170,190 60,160 10,50 20,10 140),(130 170,10 140,50 120,110 110,130 170)) ↩
(k) crossing rings	POLYGON((10 140,90 190,130 170,190 60,160 10,50 20,10 140),(90 189,10 139,80 110,110 130,90 189)) ↩
(l) cut line	POLYGON((10 100,60 140,60 190,60 140,130 170,190 60,160 10,50 20,10 100))
(m) hole outside shell	POLYGON((10 40,10 100,130 130,190 80,140 20,50 10,10 40),(81 143,103 143,112 162,107 175,89 183,71 169,81 143)) ↩

Figure



MULTIPOLYGON 以下条件是有用的：

1. 其元素 POLYGON 有效
2. 元素不重叠（即其内部不得相交）
3. 元素不在点接触（即不沿接触）

The next example checks the MultiPolygon rules. In **(n)** the polygons touch at only one point. In **(o)** they share an edge, and in **(p)** their interiors overlap.

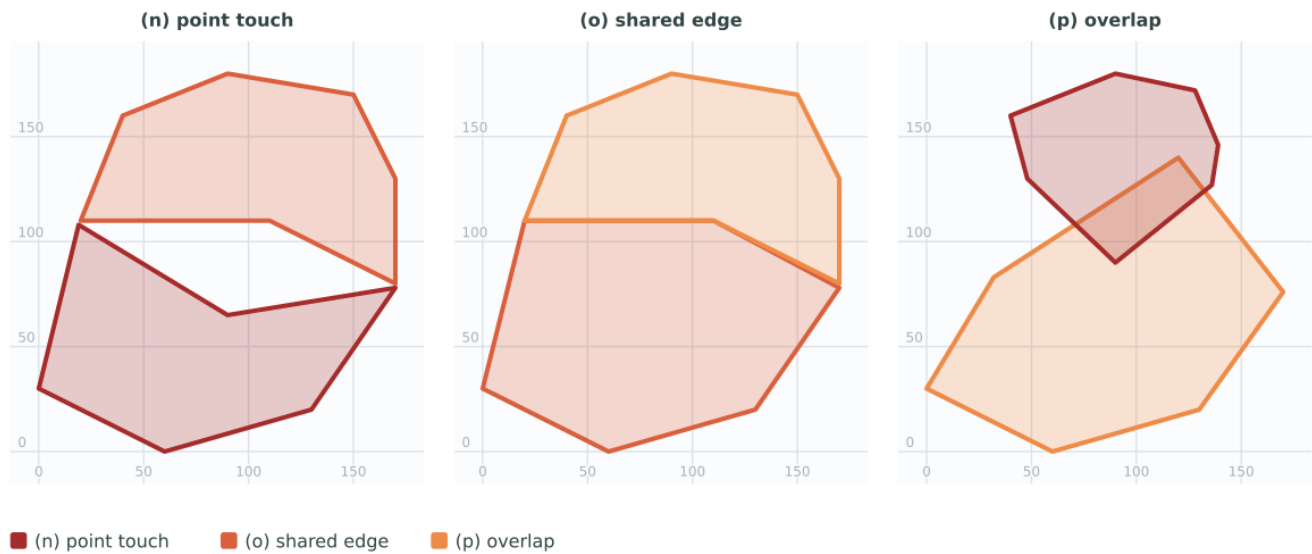
Code

```
WITH cases(label, geom, valid, reason_pattern) AS (VALUES
  ('(n) point touch',
    'MULTIPOLYGON (((10 40, 29 118, 100 75, 180 88, 140 30, 70 10, 10 40)),
      ((50 170, 100 190, 160 180, 180 140, 180 90, 120 120,
        30 120, 50 170)))'::geometry,
    true, NULL),
  ('(o) shared edge',
    'MULTIPOLYGON (((10 40, 30 120, 120 120, 180 88, 140 30, 70 10, 10 40)),
      ((50 170, 100 190, 160 180, 180 140, 180 90, 120 120,
        30 120, 50 170)))'::geometry,
    false, 'Self-intersection%'),
  ('(p) overlap',
    'MULTIPOLYGON (((10 40, 42 93, 130 150, 180 86, 140 30, 70 10, 10 40)),
      ((50 170, 100 190, 138 182, 149 156, 146 137, 100 100,
        58 140, 50 170)))'::geometry,
    false, 'Self-intersection%')
)
SELECT label, geom
FROM cases
WHERE ST_IsValid(geom) = valid
  AND (valid OR ST_IsValidReason(geom) LIKE reason_pattern)
ORDER BY label;
```

Output

label	geom
(n) point touch	MULTIPOLYGON(((10 40,29 118,100 75,180 88,140 30,70 10,10 40)),((50 170,100 190,160 180,180 140,180 90,120 120,30 120,50 170)))
(o) shared edge	MULTIPOLYGON(((10 40,30 120,120 120,180 88,140 30,70 10,10 40)),((50 170,100 190,160 180,180 140,180 90,120 120,30 120,50 170)))
(p) overlap	MULTIPOLYGON(((10 40,42 93,130 150,180 86,140 30,70 10,10 40)),((50 170,100 190,138 182,149 156,146 137,100 100,58 140,50 170)))

Figure



这些意味着有效的多边形几何也很。对于线性几何，唯一的有效性是 **LINESTRING** 必须至少有 2 个点并且具有非零长度（或等效地，至少有 2 个不同的点。注意，非（自相交）是有效的。

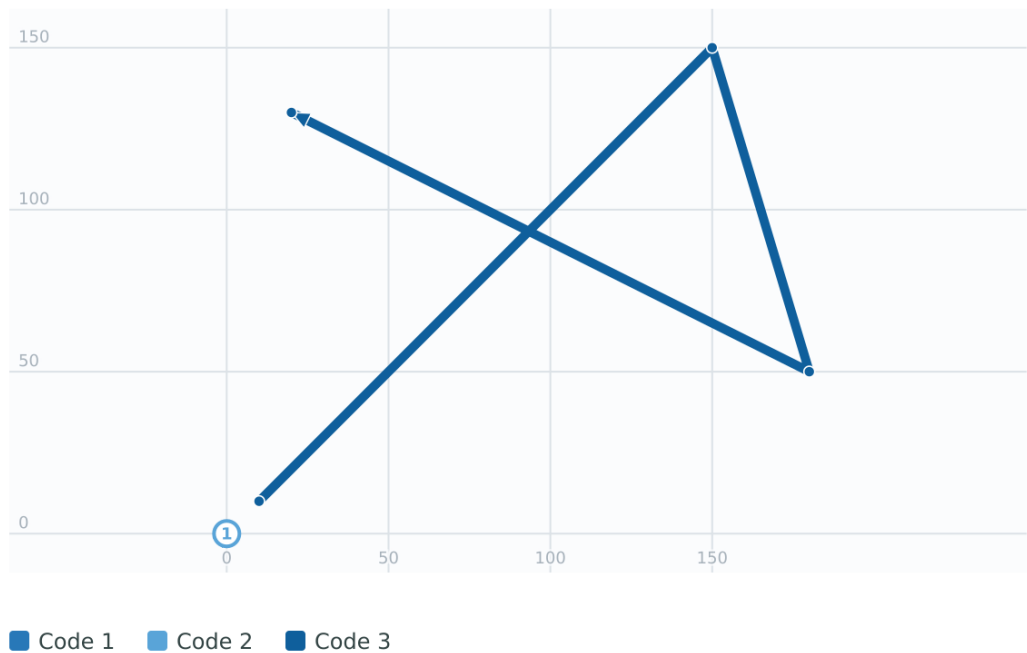
Code

```
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;
```

Output

len_nonzero	len_zero	self_int
t	f	t

Figure



POINT 和 MULTIPOINT 几何形状没有有效性检查。

4.4.3 有效性管理

PostGIS 允许创建和存储有效和无效的几何形状。允许合并或修复无效的几何形状。在某些情况下，OGC 有效性规则比时期更严格（例如零度曲线和有倒孔的多边形。）

PostGIS 提供的多功能都依赖于几何参数有效的假设。例如，计算在多边形外部定义有孔的多边形的面积，或者从非简单边界构造多边形是没有意义的。假设有效的几何输入函数更有效地运行，因为它不需要拓扑正确性。（值得注意的例外是零度曲线和具有反孔的多边形通常可以正确处理。）此外，如果输入有效，大多数 PostGIS 函数都会生成有效的几何输出。这使得 PostGIS 功能可以安全地连接在一起。

如果在调用 PostGIS 函数时遇到意外消息（例如“GEOS Intersection () 抛出了一个错误!”），应首先确认函数参数是否有效。如果无效，应考虑使用以下技巧之一来修复您正在处理的数据。

Note

如果函数报告有效输入错误，那么您可能在 PostGIS 或其使用的库之一中遇到错误，您应将此报告给 PostGIS 项目。如果 PostGIS 函数有效输入返回无效几何形状，情况也是如此。

要检查几何形状是否有效，请使用ST_IsValid函数。执行以下操作：

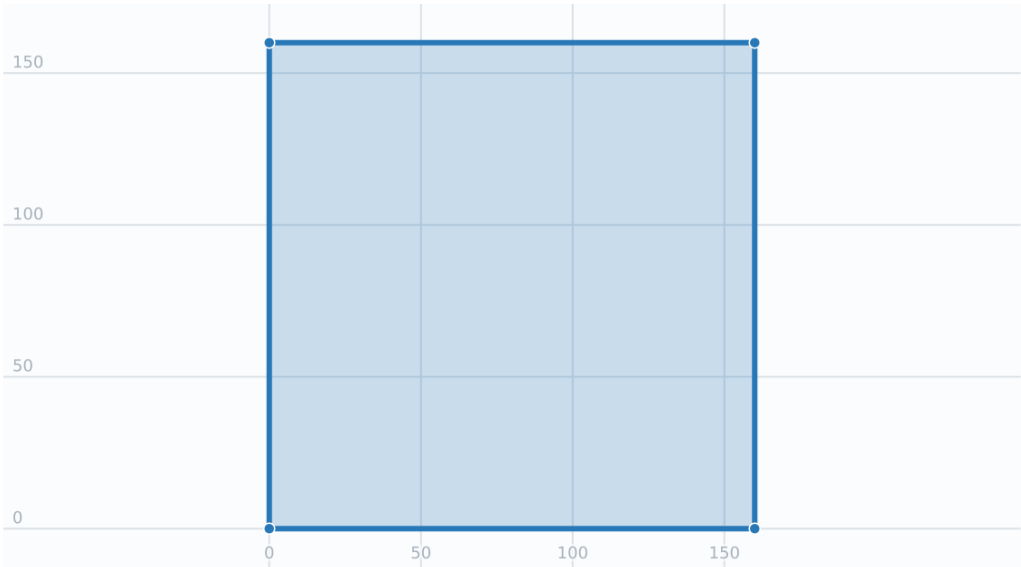
Code

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

Output

```
t
```

Figure



Code

有关几何无效的性☒和位置的信息由ST_IsValidDetail函数确定：

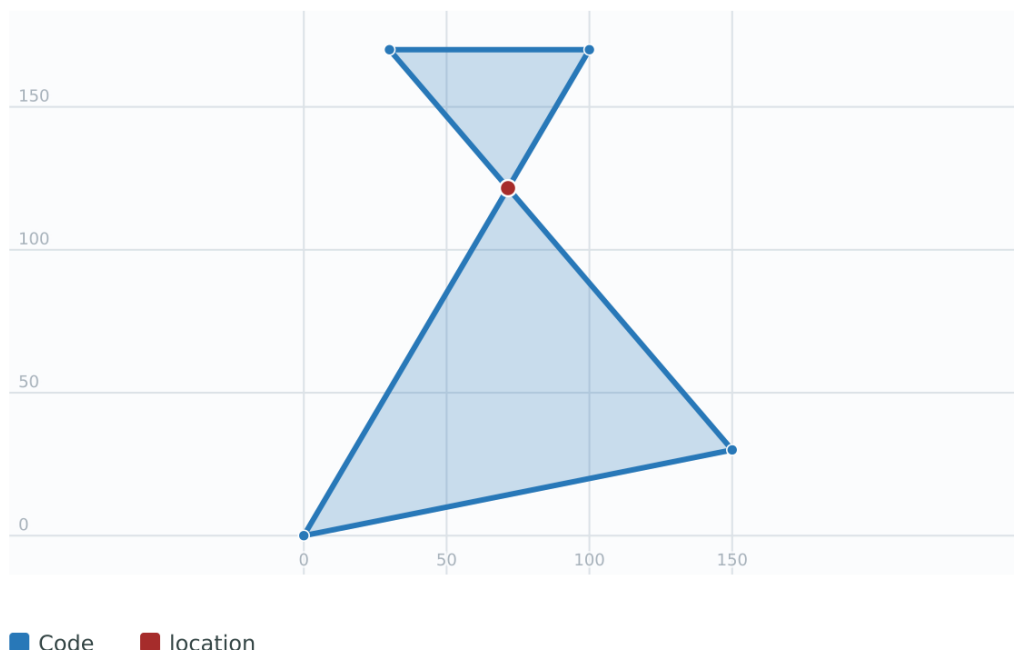
Code

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

Output

valid	reason	location
f	Self-intersection	POINT(91.51162790697674 141.56976744186048)

Figure



在某些情况下，需要自修正无效的几何形状。使用 **ST_MakeValid** 函数来执行此操作。（**ST_MakeValid** 是空函数的一种情况，确允许无效输入！）

修复几何的复杂度需要花费大量 CPU 时间，因此默认情况下，PostGIS 在加载几何时不会修复几何。如果不信任数据源，可以使用约束表来限制几何。执行以下操作：

Code

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

4.5 空参考系

空参考系 (SRS)（也称为**坐标参考系 (CRS)**）定义了如何将几何形状引用到地球表面。SRS 分三种类型：

- 地理 (geodetic) SRS，使用角坐标（经度和纬度），直接映射到地球表面。
- 投影 (projected) SRS 使用数学投影将球形地球的表面“展平”到平面上。它以允许直接测量距离、面积和角度等量的方式分配位置坐标。坐标系是笛卡儿坐标系，意味着它具有定义的原点和一个垂直轴（通常面向北和东）。每个投影的 SRS 使用固定的单位（通常是米或英尺）。投影的 SRS 的适用范围可能会受到限制，以避免变形并适合定义的范围。
- 本地 (local) SRS 是不参考地球表面的笛卡儿坐标系。PostGIS 指定 SRID 为 0。

所使用的空参考系存在许多差异。一般空参考系在欧洲石油行业的 **EPSG database** 中进行了标准化。为了方便起见，PostGIS（和许多空参考系）使用称为 SRID 的整数来引用空参考系。

几何通过其 SRID 与空参考系相关，可由 **ST_SRID** 获取。可以使用 **ST_SetSRID** 指定几何形状的 SRID。某些几何构造函数允许提供 SRID（如 **ST_Point** 和 **ST_MakeEnvelope**）。EWKT 格式支持有前缀 **SRID=n;** 的 SRID。

处理几何形状（例如 **ST_Union** 和 **ST_Intersection** 函数）的空函数要求输入几何形状位于同一空参考系中（具有相同的 SRID）。可以使用 **ST_Transform** 和 **ST_TransformPipeline** 将几何数据从一个空参考系转换到另一个空参考系。从函数返回的几何形状具有与输入几何形状相同的 SRS。

4.5.1 SPATIAL_REF_SYS 表

PostGIS 使用的 SPATIAL_REF_SYS 表是一个符合 OGC 标准的数据表，用于定义可用的空参考系。它包含数字 SRID 和坐系的文本描述。

spatial_ref_sys 表定义：

Code

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

列是：

srid 唯一标识数据内空参考系(SRS) 的整数代码。

auth_name 参考系引用的标准或标准机的名称。例如，“EPSG”是有效的 auth_name。

auth_srid 由 auth_name 中引用的权威机定义的空参考系的 ID。对于 EPSG，它是 EPSG 代码。

srtext 已知空参考系的文本表示形式。WKT SRS 表示的一个示例是：

Code

```
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]
]
```

有关 SRS WKT 的更多信息，请参考 OGC 标准参考系的已知文本表示。

proj4text PostGIS 使用 PROJ 提供坐功能。proj4text 列包含特定 SRID 的 PROJ 坐定义字符串。例如：

Code

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

欲了解更多信息，请参考 [PROJ 网站](#)。spatial_ref_sys.sql 文件包含所有 EPSG 投影的 srtext 和 proj4text 定义。

索引用于定义的空参考系定义，PostGIS 使用以下策略：

- 如果 auth_name 和 auth_srid 存在（非空），使用基于这些条目（如果存在）的 PROJ SRS。
- 如果 srtext 存在，使用它构建 SRS（如果可能）。
- 如果 proj4text 存在，使用它构建 SRS（如果可能）。

4.5.2 用自定义空参考系

PostGIS的 `spatial_ref_sys` 表包含由 **PROJ** 投影管理的 3000 多个最常用的空参考系定义。但有很多坐标系它不包含。如果您具有有关空参考系的所需信息，您可以将 SRS 定义添加到表中。或者，如果您熟悉 PROJ 构造，您可以定义自己的自定义空参考系。记住，大多数空参考系都是区域性的，在其预期范围之外使用没有任何意义。

<http://spatialreference.org/> 可用于寻找不在 PostGIS 核心集中的空参考系

Some commonly used spatial reference systems are: **4326 - WGS 84 Long Lat**, **4269 - NAD 83 Long Lat**, **3395 - WGS 84 World Mercator**, **2163 - US National Atlas Equal Area**, and the 60 WGS84 UTM zones. UTM zones are one of the most ideal for measurement, but only cover 6-degree regions. For WGS84 longitude/latitude data, the UTM zone is `least(floor((longitude + 180) / 6) + 1, 60)`; the EPSG code is `32600 + zone` in the northern hemisphere or `32700 + zone` in the southern hemisphere.

美国各州使用国家平面空参考系（基于米或英尺） - 通常每个州有一个或几个。大多数基于米的数据都在核心集中，但许多基于英尺的数据或 ESRI 构建的数据需要从 spatialreference.org 复制。

您甚至可以定义非基于地球的坐标系，比如 **火星 2000** 个火星坐标系是非平面的（以球度定位），但是您可以将其与 **geography** 类型一起使用，以米而不是度定位得度和近量。

以下是使用未分配的 SRID 和以美国中心的伯特等角投影的 PROJ 定义自定义坐标系的示例：

Code

```
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 空表

4.6.1 建空表

您可以使用 SQL 语句中的 **CREATE TABLE** 语句建立一个用于存储几何数据的表，其中包括一个 **geometry** 类型的列。以下示例建了一个有几何列的表，列存储 BC-Albers 坐标系 (SRID 3005) 中的 2D (XY) LineStrings：

Code

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

geometry 类型属于几个可用的类型修饰符：

- 空类型修饰符限制列中允许的形状和维度类型。它可以是任何受支持的 **几何子类型** (例如 POINT、LINESTRING、POLYGON、MULTIPOINT、MULTILINESTRING、MULTIPOLYGON、GEOMETRYCOLLECTION 等)。修饰符添加后支持坐标数限制：Z、M 和 ZM。例如，修饰符 “LINESTRINGM” 允许具有三个维度的串，并将第三个维度度量。同样，“POINTZM” 需要四维 (XYZM) 数据。
- SRID** 修饰符将空参考系的 SRID 限制特定的数。如果省略，默认为 0。

建具有几何列的表的示例：

建一个表，其中包含具有默认 SRID 的任何类型的几何形：

Code

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

☐ 2D 点☐建具有 SRID 默☐☐的表：

Code

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

☐建一个包含 3D（XYZ）点和☐式 SRID 3005 的表：

Code

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

使用默☐ SRID ☐建具有 4D（XYZM）LINESTRING 几何☐形的表：

Code

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

使用 SRID 4267（NAD 1927 ☐☐度）☐建包含 2D POLYGON 几何☐形的表：

Code

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

一个表可以有多个几何列。☐可以通☐在☐建表☐指定它或使用 **ALTER TABLE**☐句添加它来☐☐。下面的示例演示如何添加列以存☐ 3D LineStrings：

Code

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

4.6.2 GEOMETRY_COLUMNS ☐☐

适用于 *SQL* 的 *OGC* ☐☐要素☐范定☐了用于描述 **GEOMETRY_COLUMNS** ☐☐的元数据表。在 PostGIS 中，**geometry_columns** 是从数据☐的系☐目☐表中☐取的☐☐。☐可确保空☐元数据信息始☐与当前定☐的表或☐☐一致。☐☐☐☐☐：

Code

```
\d geometry_columns
```

Output

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)	

☐些列是：

f_table_catalog, **f_table_schema**, **f_table_name** 包含几何列的要素表的完全限定名称。PostgreSQL 中没有☐似的“目☐”，因此☐列留空。☐于“schema”，使用 PostgreSQL 模式名称（默☐☐ **public**）。

f_geometry_column 要素表中几何列的名称。

coord_dimension The coordinate dimension (2, 3 or 4) of the column. The coordinate dimension is at least 2, representing X and Y coordinates.

srid 用于此表中的坐 $\square$ 几何的空 $\square$ 参考系的 ID。它是 $\square$ 引用 `spatial_ref_sys` 表的外 $\square$ 引用 ($\square$ 参 $\square$ Section 4.5.1)。

type 空 $\square$ $\square$ 象的 $\square$ 型。要将空 $\square$ 列限制 $\square$ $\square$ 一 $\square$ 型, $\square$ 使用以下之一: POINT、LINESTRING、POLYGON、MULTIPOINT、MULTILINESTRING、MULTIPOLYGON、GEOMETRYCOLLECTION 或相 $\square$ 的 XYM 版本 POINTM、LINESTRINGM、POLYGONM、MULTIPOINTM、MULTILINESTRINGM、MULTIPOLYGONM、GEOMETRYCOLLECTIONM。 $\square$ 于异 $\square$ (混合 $\square$ 型) 集合, 您可以使用 “GEOMETRY” 作 $\square$ $\square$ 型。

4.6.3 手 $\square$ 注册几何列

您可能需要 $\square$ $\square$ 做的 $\square$ 种情况是 SQL $\square$ $\square$ 和批量插入。 $\square$ 于批量插入情况, 您可以通过 $\square$ 束列或 $\square$ 行更改表来更正 Geometry_columns 表中的注册。 $\square$ 于 $\square$ $\square$, 您可以使用 CAST 操作来公开。 $\square$ 注意, 如果您的列是基于 typmod 的, $\square$ 建 $\square$ 程将正确注册它, 因此无需 $\square$ 行任何操作。此外, 未 $\square$ 几何 $\square$ 用空 $\square$ 函数的 $\square$ $\square$ 将与基 $\square$ 表几何列注册相同。

For example, suppose you have a view created like this.

I Code

```
CREATE VIEW public.vwmytablemercator AS
SELECT gid, ST_Transform(geom, 3395) As geom, f_name
FROM public.mytable;
```

To register the geometry column correctly, cast the transformed geometry.

I Code

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

I Code

```
DROP VIEW public.vwmytablemercator;
CREATE VIEW public.vwmytablemercator AS
  SELECT gid, ST_Transform(geom, 3395)::geometry(Polygon, 3395) As geom, f_name
  FROM public.mytable;
```

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

I Code

```
SELECT poi.gid, poi.geom, citybounds.city_name
INTO myschema.my_special_pois
FROM poi INNER JOIN citybounds ON ST_Intersects(citybounds.geom, poi.geom);
```

Create a 2D spatial index on the new table.

Code

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

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

Code

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

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

Code

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

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

Code

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

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

Code

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

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

Code

```
\d pois_ny
```

Output

```
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)
```

在 `geometry_columns` 中，☑者都已正确注册

Code

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

Output

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

但是 - 如果我想要创建新的

Code

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

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

The typmod based geom view column registers correctly, but the constraint based one does not.

Output

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

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

Code

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

Output

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

4.7 加空数据

建空表后，即可上空数据到数据。有内置方法可以将空数据放入 PostGIS/PostgreSQL 数据：使用格式化的 SQL 句或使用形状文件加器。

4.7.1 使用 SQL 加载数据

如果空数据可以以文本表示（如 WKT 或 WKB），那么使用 SQL 可能是将数据加载到 PostGIS 的最简单方法。通过使用 `psql` SQL 应用程序加载 SQL INSERT 语句的文本文件，可以将数据批量加载到 PostGIS/PostgreSQL 中。

SQL 加载文件（例如 `roads.sql`）可能看起来像这样：

Code

```
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;
```

SQL 文件可以使用 `psql` 方法加载到 PostgreSQL 中：

Code

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

4.7.2 使用 Shapefile 加载器

`shp2pgsql` 数据加载器将 Shapefile 转换为适合以几何或地理格式插入 PostGIS/PostgreSQL 数据库的 SQL。加载程序有多种通用命令行标志的操作模式。

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

有一个 `shp2pgsql-gui` 图形界面，其中大多数操作是命令行加载程序。对于一次性非脚本加载或者您是 PostGIS 新手，它可能更容易使用。它可以配置 PgAdminIII 的插件。

(**c|a|d|p**) 这些是相互排斥的标志：

- c 创建一个新表并从 Shapefile 文件填充它。这是默认模式
- a 将数据从 Shapefile 文件添加到数据库表中。如果使用此标志加载多个文件，这些文件必须具有相同的属性和相同的数据类型。
- d 在创建包含 shapefile 中的数据的新表之前，先删除数据库表。
- p 它只生成用于表建的 SQL 代码，不添加任何数据。如果要表建与数据加载完全分开，使用此模式。

-? 显示帮助屏幕。

-D 输出数据使用 PostgreSQL 格式。此模式与 -a、-c 和 -d 组合使用。它的加载速度比默认的“插入”SQL 格式快得多。通常将其用于大型数据集。

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

- s [**<FROM_SRID>:**]**<SRID>** 使用指定的 SRID 创建并填充几何表。（可选）指定输入 shapefile 文件使用指定的 FROM_SRID，在这种情况下，几何形状将被重新投影到目标 SRID。
- k 保留属性的大小写（列、架和属性）。注意，Shapefile 文件中的属性都是大写的。
- i 将所有整数限制为 32 位整数。即使在 DBF 中被认为是有效的，也不会生成 64 位的 bigint。
- I 在几何列上生成 GiST 索引。
- u Create the target table as UNLOGGED. This can speed up loading transient staging data, but PostgreSQL does not preserve unlogged table contents after a crash or unclean shutdown.
- m Specify a file containing a set of mappings of (long) column names to 10 character DBF column names. The content of the file is one or more lines of two names separated by white space and no trailing or leading space. For example:
 COLUMNNAME DBFFIELD1
 AVERYLONGCOLUMNNAME DBFFIELD2
- S 生成单一的几何形状而不是多个几何形状。当所有几何形状上都是同一的才会成功（即具有一个壳的 MULTIPOLYGON，或具有一个点的 MULTIPOINT）。
- t **<dimensionality>** 限制输出几何具有指定的维度。使用以下方法表示维度的字符串：2D、3DZ、3DM、4D。如果输入的维度少于指定的维度，输出将用零填充这些维度。如果输入具有指定的更多维度，不需要的维度将被删除。
- w 使输出格式为 WKT 而不是 WKB。重要的是要注意，精度可能会降低，并且可能会输出坐波。
- e 单独行每个语句，而不使用事务。当存在一些生成的不良几何形状时，允许添加大多数好的数据。注意，不能与 -D 标志一起使用，因为“”格式始终使用事务。
- W **<encoding>** 指定输入数据（dbf 文件）的编码。使用，dbf 的所有属性都会从指定的编码 UTF8。生成的 SQL 输出将包含 SET CLIENT_ENCODING to UTF8 命令，以便后端能从 UTF8 重新编码数据配置内部使用的任何编码。
- N **<policy>** 设置 NULL 几何操作策略（插入 *、跳过、中止）
- n -n 输入 DBF 文件。如果您的数据没有相关的 shapefile 文件，它将自切换到此模式并添加 DBF。因此，当您设置了完整的 shapefile 并且只需要属性数据而不需要几何形状时，才需要设置此标志。
- G 地理类型不使用几何类型，而是使用 WGS84 度和度（SRID=4326）（需要度数据）
- T **<tablespace>** 指定新表的表空间。索引仍将使用默认表空间，除非使用 -X 参数。The PostgreSQL 文档如何使用自定义表空间有很好的描述。
- X **<tablespace>** 指定新表上的索引要使用的表空间。它适用于主索引，如果 -I 一起使用，也适用于 GiST 空索引。
- Z 使用，标志将阻止生成 ANALYZE 语句。如果没有 -Z 标志（默认行），将生成 ANALYZE 语句。

使用加载程序创建输入文件并加载它的示例可能会如下所示：

Code

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

和加载可以使用 UNIX 通道一步完成：

Code

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

To load a directory of Shapefiles, run the loader once per file and derive the table name from the file name. On Windows cmd.exe, use a single percent sign interactively and double percent signs in a batch file:

Code

```
REM In a .bat file, use %F. At an interactive cmd.exe prompt, use %F.
for %F in ("C:\data\shapefiles\*.shp") do (
    shp2pgsql -d -D -s 4269 -I -W UTF-8 "%~fF" "myschema.%%~nF" | psql -d roadsdb
)
```

If every Shapefile has the same attributes and data types and should be appended into one table, create the table from the first file and use -a for the remaining files.

4.8 提取空值数据

可以使用 SQL 或 Shapefile 加载程序从数据表中提取空值数据。有关 SQL 的部分介绍了一些可用于空值表行比较和值的函数。

4.8.1 使用 SQL 提取数据

从数据表中提取空值数据的最直接方法是使用 SQL SELECT 语句来定义要提取的数据集并将结果列到可解析的文本文件中：

Code

```
SELECT road_id,road_geom AS geom, road_name FROM roads;
```

Output

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)

有时需要某种限制来减少返回的记录数量。对于基于属性的限制，使用与非空表相同的 SQL 方法。在空值限制的情况下，以下函数很有用：

ST_Intersects 此函数指示两个几何形状是否共享任何空值。

= 它检查两个几何形状在几何上是否相同。例如，如果 “POLYGON((0 0,1 1,1 0,0 0))” 与 “POLYGON((0 0,1 1,1 0,0 0))” 相同（确是如此）。

接下来，您可以在表中使用这些算符。注意，在 SQL 命令行上指定几何形状和框，必须将字符串表示形式传递给几何函数。312 是一个虚拟的空参考系，与我表的数据相匹配。例如：

I Code

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

上面的查询从“ROADS_GEOM”表中返回与查询等效的查询个查询。

要查询道路是否穿过多边形定义的面，查询行以下操作：

I Code

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

最常查询的空查询可能是“基于框架”的查询，由数据查询器和网络制查询器等客户端使用，以查询“地图框架”的数据行。

使用“&&”运算符，可以指定比查询特征是 BOX3D 是几何。但是，如果指定几何形状，其边界框用于比。

使用框架的“BOX3D”象，如下所示：

I Code

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

注意使用 SRID 312 来指定包的投影。

4.8.2 使用 Shapefile 文件程序

pgsql2shp 表查询器连接到数据库并将表（可能由查询定义）查询形状文件。基本方法是：

I Code

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

I Code

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

命令行包括：

- f <filename>** 将输出写入特定文件名。
- h <host>** 要连接到的数据库的主机名。
- p <port>** 要连接到的数据库的端口。
- P <password>** 用于连接到数据库的密码。
- u <user>** 要连接到数据库的用户名。

- g <geometry column> 用于具有多个几何列的表，写入形状文件时要使用的几何列。
- b 使用二进制游标。这将使操作更快，但如果表中的任何非几何属性缺少文本游标，游标操作将不起作用。
- r 原始模式。不要删除 gid 字段或列名称。
- m filename 将符号重新映射十个字符名称。文件的内容是由空格分隔的符号行，并且没有尾随或前空格：VERYLONGSYMBOL SHORTONE ANOTHERVERYLONGSYMBOL SHORTER 等。

4.9 空索引

空索引使得使用空数据存取大型数据集成可能。如果没有索引，搜索功能需要顺序扫描数据中的每条记录。索引通过将数据分成可快速遍历以查找匹配的记录来加快搜索速度。

通常用于属性数据的 B 索引方法对于空数据来不是很有用，因为它只支持二维度的数据存取和索引。如几何形状（具有 2 个或更多维度）之数据需要支持跨所有数据维度的范围的索引方法。PostgreSQL 在空数据管理方面的主要索引之一是它提供了多种适用于多维数据的索引方法：GiST、BRIN 和 SP-GiST 索引。

- **GiST (广义搜索树)** 索引将数据分解成“一维的事物”、“二维的事物”、“内部的事物”，并且可用于多种数据类型，包括 GIS 数据。PostGIS 使用在 GiST 之上实现的 R-Tree 索引来索引空数据。GiST 是最常用、最通用的空索引方法，提供非常好的性能。
- **BRIN (范围索引)** 索引通过表范围的空范围进行操作。搜索是通过扫描范围来完成的。BRIN 适用于某些类型的数据（空排序、更新不频繁或不更新）。但它提供了更快的索引构建和更小的索引大小。
- **SP-GiST (空分区广义搜索树)** 是一种通用索引方法，支持四叉树、k-d 树和基数树 (tries) 等分区搜索。

空索引存储几何形状的边界框。空索引使用索引作为主要过滤器来快速确定一维可能与查询条件匹配的几何形状。大多数空索引需要使用空索引函数的辅助过滤器来提供更具体的空条件。有关使用空索引的更多信息，请参考 Section 5.2。

请参考 [PostGIS Workshop](#) 有关空索引的部分以及 [PostgreSQL 手册](#)。

4.9.1 GiST 索引

GiST 代表“广义搜索树”，是多维数据索引的通用形式。PostGIS 使用在 GiST 之上实现的 R-Tree 索引来索引空数据。GiST 是最常用、最通用的空索引方法，提供非常好的性能。GiST 的其他索引用于加速不适合普通 B 索引的各种多维数据（整数数组、光数据等）的搜索。有关更多信息，请参考 [PostgreSQL 手册](#)。

一旦空数据表超过几千行，您将需要建立索引来加速数据的空搜索（除非您的所有搜索都基于属性，在这种情况下，您将需要在空数据表上建立普通索引）属性字段）。

在“geometry”列上建立 GiST 索引的方法如下：

Code

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

上述方法将始终建立 2D 索引。若要取几何类型的 n 维索引，可以使用以下方法建立一个索引：

Code

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

建立空索引是一项计算密集型工作。它会在建表时阻止表的写入，因此在生产系统上，您可能希望以慢的并行感知方式执行：

Code

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

建立索引后，PostgreSQL 不会收集表上的信息会很有帮助。它用于优化查询：

Code

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

4.9.2 BRIN 索引

BRIN 代表 Block Range Index，是 PostgreSQL 提供了一种通用索引方法。BRIN 是一种有索引，这意味着需要二次索引以确定是否符合搜索条件（所有空索引都是如此）。它提供了更快的索引构建速度和更小的索引尺寸，同时性能仍然合理。BRIN 的主要用途是与表内物理位置相关的列建立索引，以支持超大表的高效索引。除了空索引，BRIN 还可以加速各种属性数据（整数、数字等）的搜索。更多信息参见 [PostgreSQL manual](#)。

一旦空表超过几千行，您将需要建立索引来加速数据的空搜索。只要 GiST 索引的大小不超过数据可用的 RAM 量，并且只要您能承受索引存大小以及写入更新索引的成本，GiST 索引的性能就非常好。否则，对于非常大的表，可以考虑将 BRIN 索引作为替代方案。

BRIN 索引包含一范围的表（称范围）中的行中包含的所有几何形的边界框。使用索引行范围，将范围以查找与范围相交的范围。当数据物理排序以使范围的边界框具有最小重叠（并且理想情况下是互斥的），才是有效的。生成的索引非常小，但取相同数据的性能通常低于 GiST 索引。

构建 BRIN 索引比构建 GiST 索引占用的 CPU 资源要少得多。我经常发现，对于相同数据，BRIN 索引的构建速度比 GiST 索引快十倍。由于 BRIN 索引只每个表范围存一个边界框，因此通常使用的磁盘空间比 GiST 索引少一千倍。

您可以选择在某个范围内索引的数。如果少这个数字，索引会更大，但可能会提供更好的性能。

为了使 BRIN 有效，表数据按物理顺序存，从而最大限度地减少范围重叠量。数据可能已适当排序（例如，如果它是从已按空顺序排序的一个数据集加的）。否则，可以通过按一空数据行排序来完成。一种方法是构建一个按几何排序的新表（在最近的 PostGIS 版本中使用有效的希伯特曲线排序）：

Code

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

或者，可以通过使用 GeoHash 作（范围）索引并索引索引行聚来数据行适当排序：

Code

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

在 geometry 列上构建 BRIN 索引的方法：

Code

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

上述方法构建二索引。若要生成三索引，使用以下方法：

Code

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

您可以使用 4D 运算符取 4D 度索引：

Code

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

在上面的命令中，范围内的页数使用默认值 128。此语句用于指定聚合中某个区域中的页数

Code

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

记住，BRIN 索引存储大量行的一个索引条目。如果您的表存储具有混合页数的几何形状，生成的索引的性能可能会很差。您可以通过存储几何形状页数最少的算符来避免这种性能损失

BRIN 索引支持 geography 数据类型。在 geography 列上创建 BRIN 索引的方法是：

Code

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

上述方法在球体上的地理空间对象上创建 2D 索引。

目前，提供“包容性支持”，这意味着只有 &&、~ 和 @ 算符可用于 2D 情况（用于几何和地理），而算符 &&& 只能用于 3D 几何形状。目前不支持 kNN 搜索。

BRIN 与其他索引类型的一个重要区别是数据不会重新索引。表中空闲数据的更改只需附加到索引的末尾。这将导致索引搜索性能随着索引的推移而降低。可以通过运行 VACUUM 或使用特殊函数 `brin_summarize_new_values(regrange)` 来更新索引。因此，BRIN 可能最适合用于只读或很少更改的数据。有关更多信息，请参考手册。

看一下空闲数据使用 BRIN 的情况：

- 索引创建非常快，索引大小非常小。
- 索引创建比 GiST 慢，但可以接受。
- 要求按空闲顺序对表数据进行排序。
- 需要手动索引。
- 它最适合重复很少或没有重复（例如，点）且静态或不经常更改的大型表。
- 用于返回相关大量数据的查询更有效。

4.9.3 SP-GiST 索引

SP-GiST 代表“空闲分区广泛搜索”，是多维数据类型索引的通用形式，支持分区搜索，例如四叉树、k-d 树和基数树 (tries)。这些数据结构的共同特征是它们将搜索空间重复划分为大小不必相等的分区。除了空闲索引之外，SP-GiST 用于加速多种数据的搜索，例如路径、ip 路由、子字符串搜索等。有关更多信息，请参考 PostgreSQL 手册。

与 GiST 索引的情况一样，SP-GiST 索引是有用的，因为它存储包围空闲对象的边界框。SP-GiST 索引可以被 GiST 索引的替代方案。

一旦 GIS 数据表超过几千行，就可以使用 SP-GiST 索引来加速数据的空闲搜索。在“geometry”列上创建 SP-GiST 索引的方法如下：

Code

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

上面的语句创建了一个二维索引。几何类型的三维索引是使用三叉算符生成的，如下所示：

Code

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

创建索引是一个计算密集型操作。它会在创建表时阻止表的写入，因此在生产系统上，您可能希望以慢的并行感知方式进行操作：

Code

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

建立索引后，PostgreSQL 不收集表上的统计信息会很有帮助。它用于优化计划：

Code

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

SP-GiST 索引可以加速涉及以下运算符的查询：

- 用于二维索引，<<, &<, &>, >>, <<|, &<|, |&>, |>>, &&, @>, <@, 和 ~=
- 用于三维索引，&/&, ~=, @>>, 和 <<@。

目前不支持 kNN 搜索。

4.9.4 索引的使用

通常，索引会无形中加速数据：一旦建立索引，PostgreSQL 规划器就会自行决定何时使用它来提高性能。但在某些情况下，规划器不会使用有索引，因此最慢会使用慢的序列扫描而不是索引。

如果索引未被使用，您可以进行以下几项操作：

- 可以通过索引计划和查看统计信息来估算所需的内容。不正确的连接、忘记的表和错误的表可能会导致多次意外的表扫描。若要获取统计信息，在查询开始时运行 **EXPLAIN**。
- 确保收集有关表中行的数量和分布的统计信息，以便规划器提供更好的信息来做出有关索引使用的决策。**VACUUM ANALYZE** 将统计信息重置。
无论如何，您应定期清理数据。许多 PostgreSQL DBA 定期将 **VACUUM** 作为非高峰 cron 作业运行。
- 如果清理没有帮助，您可以使用命令 **SET ENABLE_SEQSCAN TO OFF;** 限制规划器使用索引信息。通过这种方式，您可以限制规划器是否能生成索引加速扫描。您应使用此命令进行测试；一般来说，规划者比你更了解何时使用索引。测试后，不要忘记运行 **SET ENABLE_SEQSCAN TO ON;** 以便规划器能正常进行其他操作。
- 如果 **SET ENABLE_SEQSCAN TO OFF;** 帮助您的查询运行得更快，您的 PostgreSQL 可能未根据您的硬件进行调整。如果您限制规划器扫描与索引扫描的成本有差异，您可以减小 **RANDOM_PAGE_COST**（位于 `postgresql.conf` 中），或使用 **SET RANDOM_PAGE_COST TO 1.1;**。RANDOM_PAGE_COST 的默认为 4.0。您可以将其设置为 1.1（用于 SSD）或 2.0（用于快速磁盘）。减小值使规划器更有可能使用索引扫描。
- 如果 **SET ENABLE_SEQSCAN TO OFF;** 没有帮助，您可能使用 PostgreSQL 规划器尚无法优化的 SQL 构造。可以以规划器能处理的方式重写查询。例如，具有内联 SELECT 的子查询可能不会生成有效的计划，但可能会使用 LATERAL JOIN 进行重写。

有关更多信息，请参阅 PostgreSQL 手册的规划部分。

Chapter 5

空

空数据存在的理由是在数据内行，通常需要桌面 GIS 功能。有效使用 PostGIS 需要了解些空函数可用、如何在中使用它，并确保适当的索引以提供良好的性能。

5.1 空关系的确定

空关系示个几何形如何相互关。用于几何的基本功能。

5.1.1 度展九交模型 (Dimensionally Extended 9-Intersection Model)

根据OpenGIS SQL 特征范，“比个几何形的基本方法是几何形的内部、界和外部之的相交行交集，并根据‘交集矩’的元素个几何之的关系行分。”

在点集拓扑理中，嵌入二空的几何中的点被分三个集合：

界

几何的界是下一个低度的几何的集合。于数 0 的 POINT，界是空集。LINESTRING 的界是个端点。于 POLYGON，界是外和内条。

内部

几何形的内部是几何形中不在界内的那些点。于 POINT 来，内部就是点本身。LINESTRING 的内部是端点之的点集。于 POLYGON，内部是多形内的面。

外部

几何体的外部是几何体嵌入的空其余部分；句，所有不在几何内部或界上的点。它是一个二非封曲面。

九交模型 (DE-9IM) 通指定每个几何形的上述集合之的 9 个相交的度来描述个几何形之的空关系。交集度可以正式表示 3x3 交集矩。

于几何形 g ，内部、界和外部使用符号 $I(g)$ 、 $B(g)$ 和 $E(g)$ 表示。此外， $dim(s)$ 表示集合 s (域 $\{0, 1, 2, F\}$)：

- 0 => 点集合
- 1 => 集合
- 2 => 面集合
- F => 空集合

使用此表示法， $\Box$ 个几何 a 和 b 的交集矩 $\Box\Box$ ：

	内部 (Interior)	$\Box$ 界 (Boundary)	外部 (Exterior)
内部 (Interior)	$\dim(I(a) \cap I(b))$	$\dim(I(a) \cap B(b))$	$\dim(I(a) \cap E(b))$
$\Box$ 界 (Boundary)	$\dim(B(a) \cap I(b))$	$\dim(B(a) \cap B(b))$	$\dim(B(a) \cap E(b))$
外部 (Exterior)	$\dim(E(a) \cap I(b))$	$\dim(E(a) \cap B(b))$	$\dim(E(a) \cap E(b))$

The following overlapping polygons provide a concrete example. **ST_Relate** computes the matrix 212101212; it matches the overlap pattern $T^*T^{***}T^{**}$ used by **ST_Overlaps**.

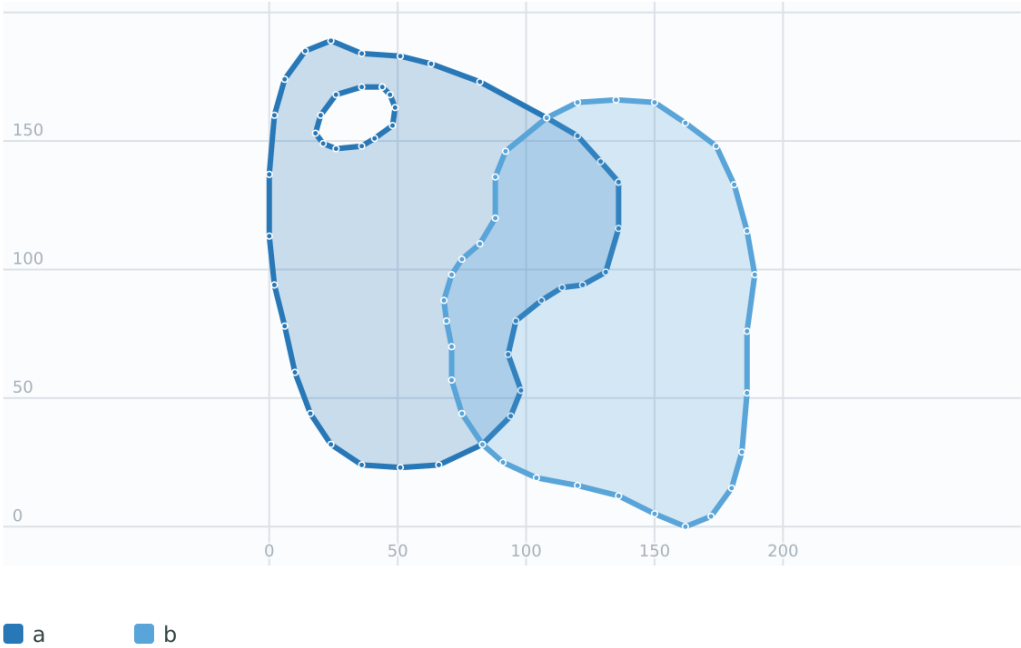
Code

```
WITH example AS (  
  SELECT  
    'POLYGON ((140 140,140 122,135 105,126 100,118 99,110 94,100 86,97 73,102 59,98 49,87 38,70 30,55 29,40 30,28 38,20 50,14 66,10 84,6 100,4 119,4 143,6 166,10 180,18 191,28 195,40 190,55 189,67 186,86 179,112 165,124 158,133 148,140 140),(48 177,40 177,30 174,24 166,22 159,25 155,30 153,40 154,45 157,52 162,53 169,51 174,48 177))'::geometry AS a,  
    'POLYGON ((75 63,79 50,87 38,95 31,108 25,124 22,140 18,154 11,166 6,176 10,184 21,188 35,190 58,190 82,193 104,190 121,185 139,178 154,166 163,154 171,139 172,124 171,112 165,96 152,92 142,92 126,86 116,79 110,75 104,72 94,73 86,75 76,75 63))'::geometry AS b  
) , relation AS (  
  SELECT a, b, ST_Relate(a, b) AS matrix  
  FROM example  
)  
SELECT matrix,  
       ST_RelateMatch(matrix, 'T*T***T**') AS matches_overlap_pattern,  
       ST_Overlaps(a, b) AS overlaps  
FROM relation;
```

Output

matrix	matches_overlap_pattern	overlaps
212101212	t	t
(1 row)		

Figure



The executable query and generated overlay above are the visual source for the relation. The query below renders the nine cells of the same kind of matrix as a 3-by-3 set of examples. The final exterior/exterior cell is clipped to a finite viewport, since the true exterior/exterior intersection is unbounded.

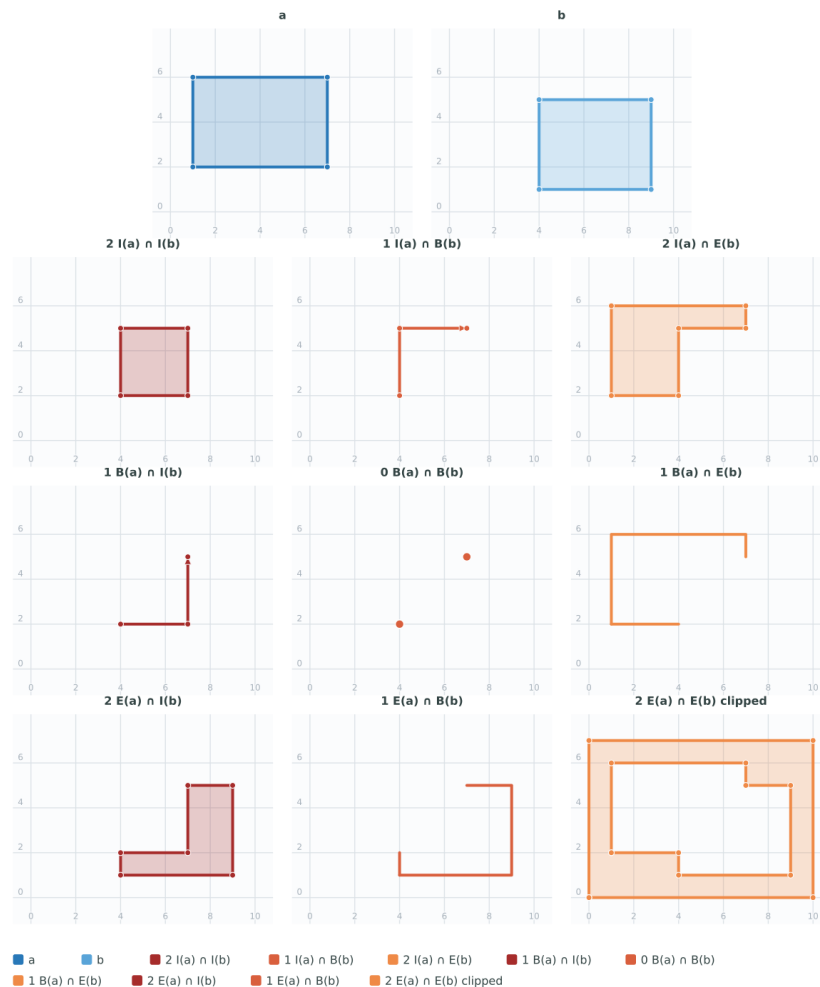
Code

```
WITH example AS (
  SELECT
    ST_MakeEnvelope(0, 0, 6, 4) AS a,
    ST_MakeEnvelope(3, -1, 8, 3) AS b,
    ST_MakeEnvelope(-1, -2, 9, 5) AS viewport
), parts AS (
  SELECT
    a,
    b,
    ST_Boundary(a) AS ba,
    ST_Boundary(b) AS bb,
    viewport
  FROM example
)
SELECT
  ST_Relate(a, b) AS matrix,
  ST_AsText(ST_Normalize(a)) AS input_a,
  ST_AsText(ST_Normalize(b)) AS input_b,
  ST_AsText(ST_Normalize(ST_Intersection(a, b))) AS "2 I(a) b'nb' I(b)",
  ST_AsText(ST_Normalize(ST_Intersection(a, bb))) AS "1 I(a) b'nb' B(b)",
  ST_AsText(ST_Normalize(ST_Difference(a, b))) AS "2 I(a) b'nb' E(b)",
  ST_AsText(ST_Normalize(ST_Intersection(ba, b))) AS "1 B(a) b'nb' I(b)",
  ST_AsText(ST_Normalize(ST_Intersection(ba, bb))) AS "0 B(a) b'nb' B(b)",
  ST_AsText(ST_Normalize(ST_Collect(
    'LINESTRING(0 0,0 4,6 4,6 3)::geometry',
    'LINESTRING(0 0,3 0)::geometry'
  ))) AS "1 B(a) b'nb' E(b)",
  ST_AsText(ST_Normalize(ST_Difference(b, a))) AS "2 E(a) b'nb' I(b)",
  ST_AsText(ST_Normalize(ST_Collect(
    'LINESTRING(3 -1,8 -1,8 3,6 3)::geometry',
    'LINESTRING(3 -1,3 0)::geometry'
  ))) AS "1 E(a) b'nb' B(b)",
  ST_AsText(ST_Normalize(ST_Difference(viewport, ST_Union(a, b))))
  AS "2 E(a) b'nb' E(b) clipped"
FROM parts;
```

Output

```
-[ RECORD 1 ]-----+-----
matrix          | 212101212
input_a         | POLYGON((0 0,0 4,6 4,6 0,0 0))
input_b         | POLYGON((3 -1,3 3,8 3,8 -1,3 -1))
2 I(a) b'nb' I(b) | POLYGON((3 0,3 3,6 3,6 0,3 0))
1 I(a) b'nb' B(b) | LINESTRING(3 0,3 3,6 3)
2 I(a) b'nb' E(b) | POLYGON((0 0,0 4,6 4,6 3,3 3,0 0,0 0))
1 B(a) b'nb' I(b) | LINESTRING(3 0,6 0,6 3)
0 B(a) b'nb' B(b) | MULTIPOINT((6 3),(3 0))
1 B(a) b'nb' E(b) | MULTILINESTRING((0 0,0 4,6 4,6 3),(0 0,3 0))
2 E(a) b'nb' I(b) | POLYGON((3 -1,3 0,6 0,6 3,8 3,8 -1,3 -1))
1 E(a) b'nb' B(b) | MULTILINESTRING((3 -1,8 -1,8 3,6 3),(3 -1,3 0))
2 E(a) b'nb' E(b) clipped | POLYGON((-1 -2,-1 5,9 5,9 -2,-1 -2),(0 0,3 0,3 -1,8 -1,8 3,6 3,6 4,0 4,0 0))
```

Figure



从左到右，从上到下。交集矩形的字符串表示形式“212101212”。

欲了解更多信息，：

- 适用于 SQL 的 OpenGIS 功能规范 (1.1 版本, 第 2.1.13.2)
- 百科：度展九交集模型 (DE-9IM)
- 地理工具：点集理论和 DE-9IM 矩

5.1.2 命名空关系

为了便于确定常的空关系，OGC SFS 定义了一命名空关系。PostGIS 将这些功能提供函数 `ST_Contains`, `ST_Crosses`, `ST_Disjoint`, `ST_Equals`, `ST_Intersects`, `ST_Overlaps`, `ST_Touches`, `ST_Within`。它定义了非准关系 `ST_Covers`, `ST_CoveredBy`, 和 `ST_ContainsProperly`。

空通常用作 SQL WHERE 或 JOIN 子句中的条件。如果空索引可用，命名空会自使用空索引，因此无需使用界框算符 `&&`。例如：

Code

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

有关更多信息和插图，参 [PostGIS Workshop](#)。

5.1.3 一般空☐关系

In some cases the named spatial relationships are insufficient to provide a desired spatial filter condition. These requirements can be expressed by computing the full DE-9IM intersection matrix with **ST_Relate**.

☐了☐☐特定的空☐关系，使用相交矩☐模式。☐是用附加符号 {T,*} 增☐的矩☐表示：

- T => 表示交集的☐度不☐空; 即 {0,1,2}
- * => 任何☐西

Using intersection matrix patterns, specific spatial relationships can be evaluated succinctly. The **ST_Relate** and **ST_RelateMatch** functions can both test these patterns.

These road segments share a line segment. **ST_Crosses** does not find this relationship for linear features, because it only returns true when their interiors intersect at a point. The pattern 1*1***1** tests for a one-dimensional intersection instead.

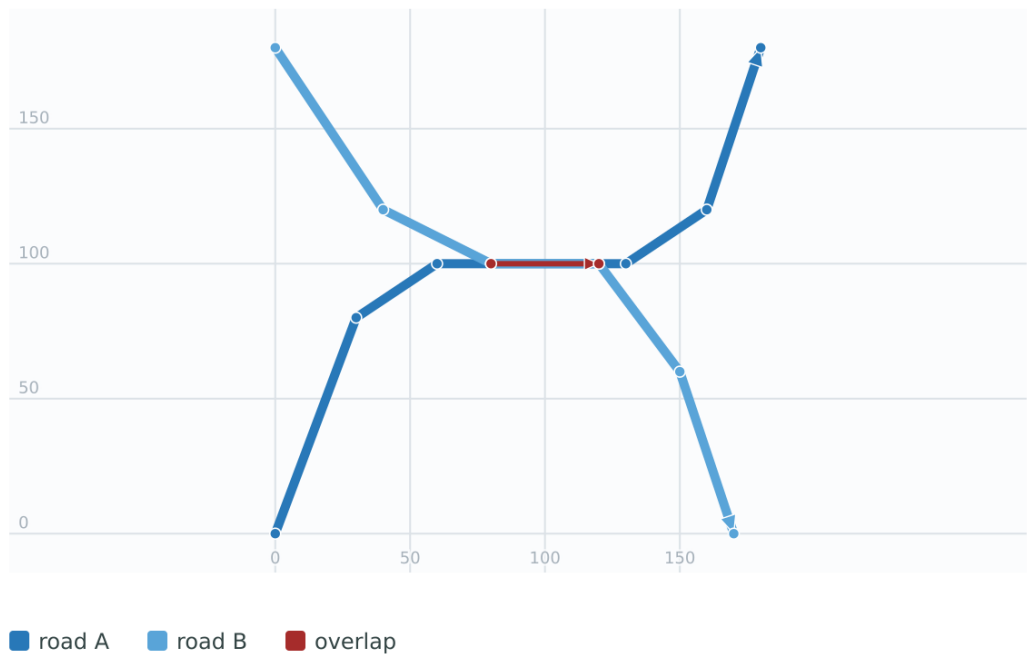
Code

```
WITH roads AS (  
  SELECT  
    'LINESTRING(10 10,40 90,70 110,140 110,170 130,190 190)::geometry AS "road A",  
    'LINESTRING(10 190,50 130,90 110,130 110,160 70,180 10)::geometry AS "road B"  
) , relation AS (  
  SELECT "road A", "road B", ST_Relate("road A", "road B") AS matrix  
  FROM roads  
)  
SELECT  
  ST_Intersection("road A", "road B") AS overlap,  
  matrix,  
  ST_Crosses("road A", "road B") AS "ST_Crosses",  
  ST_RelateMatch(matrix, '1*1***1**') AS "matches 1*1***1**"  
FROM relation;
```

Output

overlap	matrix	ST_Crosses	matches 1*1***1**
LINESTRING(90 110,130 110)	1F1FF0102	f	t
(1 row)			

Figure



The next example finds a wharf which runs from the lake interior onto its shoreline, with one endpoint on the shoreline. The other wharves provide nearby counterexamples in the figure. The pattern 102101FF2 expresses the complete required relationship in one test.

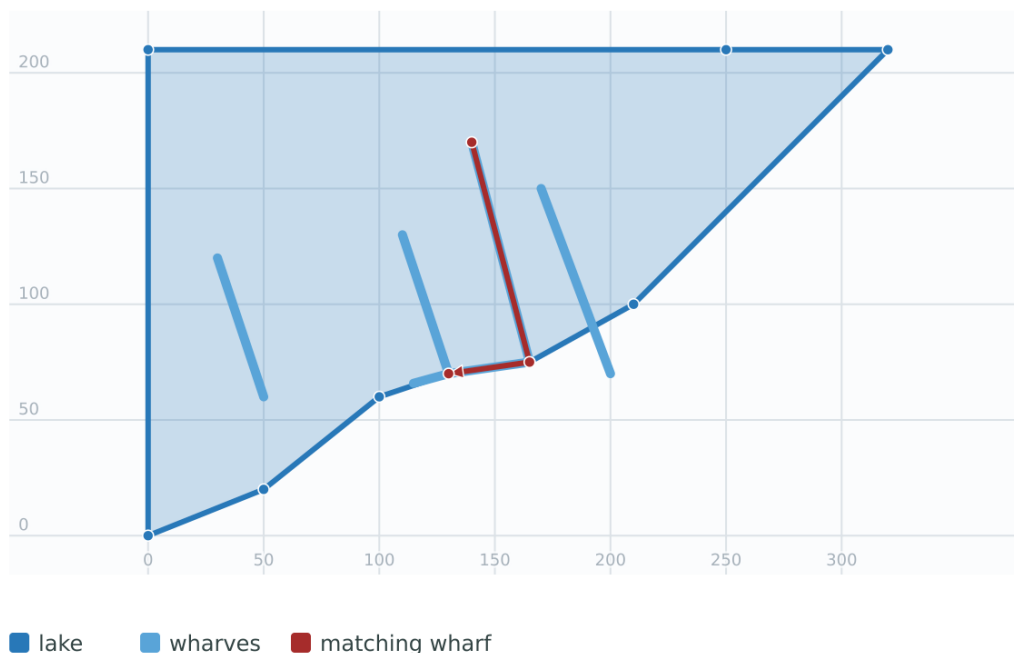
Code

```
WITH data AS (  
  SELECT  
    'POLYGON((-20 10,30 30,80 70,110 80,145 85,190 110,300 220,230 220,-20 220,-20 10))':: geometry AS lake,  
    'MULTILINESTRING((120 180,145 85,110 80),(10 130,30 70),(90 140,110 80,94.81118881118876 75.83496503496498),(150 160,180 80))'::geometry AS wharves  
) , wharf AS (  
  SELECT lake, (item).path[1] AS id, (item).geom  
  FROM data CROSS JOIN LATERAL ST_Dump(wharves) AS item  
) , relation AS (  
  SELECT id, geom, ST_Relate(lake, geom) AS matrix  
  FROM wharf  
)  
SELECT  
  id AS wharf,  
  geom AS "matching wharf",  
  matrix  
FROM relation  
WHERE ST_RelateMatch(matrix, '102101FF2')  
ORDER BY id;
```

Output

wharf	matching wharf	matrix
1	LINSTRING(120 180,145 85,110 80)	102101FF2
(1 row)		

Figure



5.2 使用空索引

使用空条件构建索引，为了获得最佳性能，确保使用空索引（如果存在）非常重要（参看 Section 4.9）。此，必在索引的 `WHERE` 或 `ON` 子句中使用空运算符或索引感知函数。

空运算符包括边界运算符（其中最常用的是 `&&`；有关完整列表，参看 Section 7.10.1）以及最近索引中使用的距离运算符（最常用的是 `<->`；有关完整列表，参看 Section 7.10.2。）

索引感知函数会自动将边界运算符添加到空条件中。索引感知函数包括命名空关系 `ST_Contains`, `ST_ContainsProperly`, `ST_CoveredBy`, `ST_Covers`, `ST_Crosses`, `ST_Intersects`, `ST_Overlaps`, `ST_Touches`, `ST_Within`, `ST_Within`, 和 `ST_3DIntersects` 以及距离 `ST_DWithin`, `ST_DFullyWithin`, `ST_3DDFullyWithin`, 和 `ST_3DDWithin`。）

`ST_Distance` 等函数不使用索引来优化其操作。例如，以下索引在大型表上会非常慢：

Code

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

此索引在 `geom_table` 中距离点 (100000, 200000) 100 个单位以内的所有几何形状。它会很慢，因为它正在计算表中每个点与指定点之间的距离，即执行 `ST_Distance()` 计算（表中的每一行）。

通过使用索引感知函数 `ST_DWithin` 可以大大减少处理的行数：

Code

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

此索引包含相同的几何形状，但采用更有效的方法。通过使用 `ST_DWithin()` 有几何形状的放大边界框的 `&&` 运算符，可以在内部索引。如果 `geom` 存在空索引，索引规划器知道索引可用于在计算距离之前减少感

趣的行数。空索引允许您搜索与边界框展范重的几何，从而搜索可能位于所需距离内的几何。然后行距离算以看果集中的是否包括在内。

有关信息和示例，参[PostGIS Workshop](#)。

5.3 空 SQL 示例

本中的示例使用性道路表和多形城市界表。bc_roads 表的定义是：

Output

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

bc_municipality 表定义如下：

Output

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

1. 道路的度是多少公里？
个可以用非常 SQL 来回答，例如：

Code

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

Output

```
km_roads
-----
70842.1243039643
```

2. 治王子城有多大（以公顷位）？
此将空算（面面）与属性条件（自治市名称）合使用：

Code

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

Output

```
hectares
-----
32657.9103824927
```

3. 按面算，省最大的直市是几个？
此使用空量作排序。有多种方法可以解决此，但最有效的方法如下：

Code

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

Output

name	hectares
TUMBLER RIDGE	155020.02556131

注意，为了回答这个问题，我必须计算每个多边形的面积。如果我经常这样做，那么向表中添加一个区域列是有益的，可以性能建立索引。通常按降序对结果行排序，并使用 PostgreSQL “LIMIT” 命令，我可以轻松地找到最大，而无需使用 MAX() 等聚合函数。

4. 每个城市包含的道路长度是多少？

这是“空连接”的示例，它使用空连接交互（“包含”）作为连接条件（而不是在公共边上连接的通常关系方法）将来自两个表（有连接）的数据集合在一起：

Code

```
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;
```

Output

name	roads_km
SURREY	1539.476
VANCOUVER	1450.331
LANGLEY DISTRICT	833.793
BURNABY	773.769
PRINCE GEORGE	694.376
...	

此查询需要一段时间，因为表中的每条道路都会包含到最终结果中（示例表中大约有 250K 条道路）。对于小的数据集（数百条记录中的数千条），查询可能非常快。

5. 建立一个包含治王子市内所有道路的新表。

这是“覆盖”的一个示例。也就是，我取出一个表并出一个由空连接切割结果成的新表。与上面所示的“空连接”不同，此查询会在上面生成新的几何形状。生成的附加就像增加的耦合，对于更精确的分析工作非常有用：

Code

```
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. 多利州的“道格拉斯街”有多长（公里）？

Code

```
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';
```

Output

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

7. 哪个市政面的孔洞最大？

Code

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

Output

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

Chapter 6

性能优化技巧

6.1 大几何形状的小表

6.1.1 问题描述

当前的 PostgreSQL 版本在索引器上对 TOAST 表存在一些弱点。TOAST 表可以理解成一种用于存放“大数据”的“扩展区”，它用来存放那些无法放入普通数据区的大型字段（比如很长的文本、图片，或者点数量特别多的复杂几何对象等）。详情可参考 [PostgreSQL 官方文档关于 TOAST 的说明](#)。

如果您碰巧有一个具有相当大的几何形状的表格，但它表的行数不是太多（例如包含所有欧洲国家的高分辨率边界的表格），会出现问题。那么表本身很小，但是它使用了大量的 TOAST 空间。在我的示例中，表本身只有 80 行，它使用 3 个数据块，但 TOAST 表使用 8225 个页面。

在，我没有使用几何运算符 `&&` 来索引很少匹配的边界框。由于索引器，表似乎只有 3 块和 80 行。优化程序估计按顺序遍历小表比使用索引更快。然后我决定忽略 GiST 索引。通常，这种做法是正确的。但是，在这种情况下，`&&` 运算符必须从磁盘读取所有几何形状并将其与边界框进行比较，因此必须读取所有 TOAST 页面。

要查看您是否遇到此问题，请使用“`EXPLAIN ANALYZE`” `postgres` 命令。有关更多信息和技术细节，您可以查看 PostgreSQL 性能问题列表上的线程：<http://archives.postgresql.org/pgsql-performance/2005-02/msg00030.php>

请参见 PostGIS 中的新线程 <https://lists.osgeo.org/pipermail/postgis-devel/2017-June/026209.html>

6.1.2 解决方法

PostgreSQL 社区正在通过制作 TOAST 感知索引引用来解决这个问题。目前，有四种解决方法：

第一个解决方法是限制索引器使用索引。发送“`SET enable_seqscan TO off;`”在输出之前发送到服务器。这基本上迫使索引器尽可能避免顺序扫描。所以它像往常一样使用 GiST 索引。但是必须在每个连接上设置此标志，并且它会导致索引器在其他情况下做出错误估计，因此您应该在之后“`SET enable_seqscan TO on;`”。

第二个解决方法是使顺序扫描与索引器一样快。可以通过建立一个“内存”bbox 的附加列并与之匹配来实现。在我的示例中，命令如下：

Code

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

在更改索引以使用 bbox 使用 `&&` 运算符而不是 `geom_column`，例如：

I Code

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

当然，如果您更改或添加行到 `mytable`，必须保持 `bbox` “同步”。最透明的方法是使用触发器，但您也可以修改程序以保持 `bbox` 列最新或在每次修改后行上面的 `UPDATE`。

6.2 几何索引聚

The PostgreSQL `CLUSTER` command physically reorders table rows according to an index. For a mostly read-only table whose queries frequently use the same spatial index, this can reduce I/O and improve cache locality.

PostgreSQL supports clustering on GiST indices, including indices on nullable geometry columns. To cluster a table using a geometry index:

I Code

```
CLUSTER my_table USING my_geom_index;
```

I Output

```
CLUSTER
```

6.3 避免维度

有，您的表中碰巧有 3D 或 4D 数据，但始终使用出 2D 几何形的符合 OpenGIS 的 `ST_AsText()` 或 `ST_AsBinary()` 函数来它。他通内部用 `ST_Force2D()` 函数来一点，会大型几何形来巨大的开。了避免种开，一次性先除些外的度可能是可行的：

I Code

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

注意，如果您使用 `AddGeometryColumn()` 添加几何列，几何度将会受到限制。要它，您需要除束。住更新 `geometry_columns` 表中的条目并随后重新建束。

于大型表，明智的做法是通过 `WHERE` 子句和主或其他可行准将 `UPDATE` 限制表的一部分，然后行行的“`VACUUM;`”，从而将此 `UPDATE` 划分更小的部分。在您的更新之。大大少了磁空的需求。此外，如果您有混合度几何形，通“`WHERE dimension(geom)>2`”限制 `UPDATE` 会跳已已于 2D 中的几何形的重写。

Chapter 7

PostGIS 参考手册

下面列出的功能是 PostGIS 用您可能需要的功能。有一些其他功能是 PostGIS 对象所需的支持功能，但一般用不到。



Note
PostGIS 已开始从有的命名定过渡到以 SQL-MM 中心的定。因此，您所了解和喜的大多数函数都已使用空型 (ST) 前行了重命名。以前的功能仍然可用，但本文档中未列出更新后的功能等效的功能。本文档中未列出的非 ST_ 函数已被弃用，并将在未来版本中除，因此停止使用它。

7.1 PostGIS Geometry/Geography/Box 数据类型

7.1.1 box2d

box2d — 表示二维界框的类型。

描述

box2d 是一种空数据类型，用于表示包围几何形或几何形集合的二维界框。例如，ST_Extent 聚合函数返回一个 box2d 对象。

表示包含 xmin, ymin, xmax, ymax。些是 X 和 Y 范围的最小和最大。

box2d 对象具有类似于 BOX(1 2,5 6) 的文本表示形式。

型制

下表列出了此数据类型允许的自和式制：

到	行
box3d	automatic
geometry	automatic

相关信息

Section 13.7

7.1.2 box3d

box3d — 表示三维矩形框的类型。

描述

box3d 是一种 PostGIS 空数据类型，用于表示包围几何形状或几何形状集合的三维矩形框。例如，[ST_3DExtent](#) 聚合函数返回一个 box3d 对象。

它表示包含 xmin, ymin, zmin, xmax, ymax, zmax。这些是 X、Y 和 Z 范围的最小值和最大值。

box3d 的文本表示如 BOX3D(1 2 3,5 6 5) 所示。

类型限制

下表列出了此数据类型允许的自和式限制：

限制到	行
box	automatic
box2d	automatic
geometry	automatic

相关信息

Section [13.7](#)

7.1.3 geometry

geometry — 表示具有平面坐标系的空要素的类型。

描述

geometry 是一种基本的 PostGIS 空数据类型，用于表示平面（欧几里得）坐标系上的要素。

几何体的所有空操作都使用几何体所在的空参考系的位置。

类型限制

下表列出了此数据类型允许的自和式限制：

限制到	行
box	automatic
box2d	automatic
box3d	automatic
bytea	automatic
geography	automatic
text	automatic

相关信息

Section [4.1](#), Section [13.3](#)

7.1.4 geometry_dump

`geometry_dump` — 用于描述复几何形状部分的复合型。

描述

`geometry_dump` 是一种复合数据类型，包含以下字段：

- `geom` - 表示几何形状的几何型。几何型取决于原始函数。
- `path[]` - 一个整数数组，用于定义几何形状中到 `geom` 件的航路径。路径数从 1 开始（即 `path[1]` 是第一个元素。）

`ST_Dump*` 系列函数使用它作为输出型，将复几何形状分解成其组成部分。

相关信息

Section 13.6

7.1.5 geography

`geography` — 使用大地（球）坐标系表示空特征的型。

描述

`geography` 是一种空数据类型，用于表示大地坐标系中的要素。大地坐标系使用球体地球建模。通过考球体模型，地理型的空操作可提供更准确的结果。

型制

下表列出了此数据类型允许的自和式制：

到	行
<code>geometry</code>	式制

相关信息

Section 4.3, Section 13.4

7.2 表管理功能

7.2.1 AddGeometryColumn

`AddGeometryColumn` — 将 `geometry`（几何）列添加到有表。

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);
```

描述

将几何列添加到已有属性表。schema_name 是表模式的名称。srid 必须是 SPATIAL_REF_SYS 表中条目的整数引用。类型必须是与几何类型匹配的字符串，例如 'POLYGON' 或 'MULTILINESTRING'。如果 schemaname 不存在（或在当前 search_path 中不可见）或指定的 SRID、几何类型或维度无效，将引发错误。

更改：2.0.0 此函数不再更新 geometry_columns，因为 geometry_columns 是从系目录读取的。默认情况下，它也不创建约束，而是使用 PostgreSQL 内置的类型修饰符行。因此，例如，使用此函数创建 wgs84 POINT 列相当于：ALTER TABLE some_table ADD COLUMN geom geometry(Point,4326);

更改：2.0.0 如果您需要约束的旧行，使用默认的 use_typmod，但将其置为 false。

更改：2.0.0 不能再在 Geometry_columns 中手动注册，但是根据几何 Typmod 表几何形创建并在没有包装器函数的情况下使用的将正确注册自身，因为它继承了父表的 Typmod 行。使用其他几何形的几何函数的需要 typmod 几何形，以便某些几何列能在 geometry_columns 中正确注册。参见 Section 4.6.3。



此方法符合了 SQL 1.1 的 OGC 功能规范。



函数支持 3d 并且不会丢失 z-index。



此方法支持形字符串和曲线。

增强：2.0.0 引入了 use_typmod 参数。默认创建 typmod 几何列而不是基于约束。

示例

This example creates a schema to hold data.

Code

```
CREATE SCHEMA my_schema;
```

Create a simple PostgreSQL table in the schema.

Code

```
CREATE TABLE my_schema.my_spatial_table (id serial);
```

Describing the table shows a single id column.

Code

```
\d my_schema.my_spatial_table
```

| Output

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

Add a spatial column to the table.

| Code

```
SELECT AddGeometryColumn('my_schema', 'my_spatial_table', 'geom', 4326, 'POINT', 2);
```

Add a point column using the old constraint-based behavior.

| Code

```
SELECT AddGeometryColumn('my_schema', 'my_spatial_table', 'geom_c', 4326, 'POINT', 2, false ↵);
```

Add a CurvePolygon column using the old constraint-based behavior.

| Code

```
SELECT AddGeometryColumn('my_schema', 'my_spatial_table', 'geomcp_c', 4326, 'CURVEPOLYGON', ↵2, false);
```

| Output

my_schema.my_spatial_table.geomcp_c SRID:4326 TYPE:CURVEPOLYGON DIMS:2

Describe the table again to see the new geometry columns.

| Code

```
\d my_schema.my_spatial_table
```

| Output

Table "my_schema.my_spatial_table"			
Columns:			
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)			

The geometry_columns view also registers the new columns.

| Code

```
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';
```


| **Output**

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

相关信息

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

7.2.2 DropGeometryColumn

DropGeometryColumn — 从空☐表中移除 geometry（几何）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, varchar column_name);

描述

Removes a geometry column from a spatial table. Note that schema_name will need to match the f_table_schema field shown by the geometry_columns view.



Warning
DropGeometryColumn is retained for backward compatibility and is not recommended for new applications. Use the standard **ALTER TABLE ... DROP COLUMN** command instead. The geometry_columns view updates automatically from the system catalogs.

- ☑ 此方法☑☑了 [SQL 1.1](#) 的 [OGC ☑☑功能☑范](#)。
- ☑ 函数支持 3d 并且不会☑失 z-index。
- ☑ 此方法支持☑形字符串和曲☑。

Changed: 2.0.0. Retained for backward compatibility after geometry_columns became a view of the system catalogs.

示例

Use standard SQL to drop the geometry column. The geometry_columns view updates automatically.

Code

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

The legacy compatibility function performs the equivalent operation.

Code

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

Output

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

相关信息

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

7.2.3 DropGeometryTable

DropGeometryTable — 删除表及其在 geometry_columns 中的所有引用。

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);

描述

删除表及其在 geometry_columns 中的所有引用。注意：如果未提供模式，则在模式感知的 postgresql 安装上使用 current_schema()。

**Warning**

DropGeometryTable is retained for backward compatibility and is not recommended for new applications. Use the standard **DROP TABLE** command instead. The geometry_columns view updates automatically from the system catalogs.

Changed: 2.0.0. Retained for backward compatibility after geometry_columns became a view of the system catalogs.

示例

Use standard SQL to drop a table containing geometry columns. The geometry_columns view updates automatically.

Code

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

The legacy compatibility function performs the equivalent operation.

Code

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

Output

```
my_schema.my_spatial_table dropped.
```

相关信息

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

7.2.4 Find_SRID

Find_SRID — 返回 geometry（几何）列中定义的 SRID。

Synopsis

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

描述

通过搜索 GEOMETRY_COLUMNS 表返回指定几何列的整数 SRID。如果几何列尚未正确添加（例如使用 [AddGeometryColumn](#) 函数），此函数将不起作用。

示例**Code**

```
SELECT Find_SRID('public', 'tiger_us_state_2007', 'geom_4269');
```

Output

```
4269
```

相关信息

[ST_SRID](#)

7.2.5 Populate_Geometry_Columns

Populate_Geometry_Columns — 确保几何列由正确类型修饰符定义或具有适当的空值约束。

Synopsis

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

描述

确保几何列具有适当的类型修饰符或空值约束，以确保它在 `geometry_columns` 表中正确注册。默认情况下，会将所有没有类型修饰符的几何列添加到具有类型修饰符的几何列。

除了向后兼容和空值需求（例如表继承，其中每个子表可能具有不同的几何类型），仍然支持旧的约束行。如果您需要旧的行，您需要将新的可参数化 `falseuse_typmod=false`。完成此操作后，将创建不约束行类型的几何列，但将定义 3 个约束。特别是，这意味着属于表的每个几何列至少具有三个约束：

- `enforce_dims_geom` - 确保每个几何形状具有相同的维度（参见 [ST_NDims](#)）
- `enforce_geotype_geom` - 确保每个几何具有相同的类型（参见 [GeometryType](#)）
- `enforce_srid_geom` - 确保所有几何形状位于同一投影中（参见 [ST_SRID](#)）

如果提供了表 `oid`，此函数会确定表中所有几何列的 `srid`、维度和几何类型，并根据需要添加约束。如果成功，会将适当的行插入到 `Geometry_columns` 表中，否则，将捕获异常并引发通知来描述。

如果提供了表的 `oid`，与表 `oid` 一致，此函数会确定表中所有几何形状的 `srid`、维度和类型，将适当的条目插入到 `geometry_columns` 表中，但不会进行任何操作来限制行。

无参数格式是参数格式的包装器，它首先删除并重新填充数据中每个空表和表的 `Geometry_columns` 表，并在适当的情况下向表中添加空值约束。它返回在数据中找到的几何列数以及插入到 `geometry_columns` 表中的列数的摘要。参数格式返回插入到 `geometry_columns` 表中的行数。

可用性：1.4.0

更改：2.0.0 默认情况下，在使用类型修饰符而不是空值约束来约束几何类型。您仍然可以通过使用新的 `use_typmod` 并将其置为 `false` 来使用旧约束行。

增强：2.0.0 引入了 `use_typmod` 可参数，允许控制是否使用 `typmodifiers` 或空值约束建列。

示例

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

Code

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

Output

```
1
```

Describe the table to see the typmod-based geometry column.

Code

```
\d myspatial_table
```

Output

```
Table "public.myspatial_table"

Columns:
  gid integer                                not null default nextval('myspatial_table_gid_seq'::regclass)
  geom geometry(LineString,4326)
```

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

Code

```
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);
```

Output

```
1
```

Describe the table to see the constraint-based geometry column.

Code

```
\d myspatial_table_cs
```

Output

```
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 — 更新几何列中所有要素的 SRID 以及表元数据。

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);
```

描述

更新几何列中所有要素的 SRID，更新 `Geometry_columns` 中的约束和参考。如果列是由类型定义制行的，类型定义将会更改。注意：如果未提供模式，在模式感知的 `pgsql` 安装上使用 `current_schema()`。



函数支持 3d 并且不会丢失 z-index。



此方法支持形字符串和曲线。

示例

将几何形插入道路表，其中 SRID 集已使用 **EWKT 格式**：

Code

```
COPY roads (geom) FROM STDIN;
SRID=4326;LINESTRING(0 0, 10 10)
SRID=4326;LINESTRING(10 10, 15 0)
\.
```

会将路表更改 4326，无论它以前是什么 SRID：

Code

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

前面的示例相当于以下 DDL 句：

Code

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

如果您在添加投影（或将其引入未知），并且您想一次性 Web 墨卡托，您可以使用 DDL 来完成此操作，但没有等效的 PostGIS 管理功能可以一次性完成此操作。

Code

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

相关信息

[UpdateRasterSRID](#), [ST_SetSRID](#), [ST_Transform](#), [ST_GeomFromEWKT](#)

7.3 几何造函数

7.3.1 ST_Collect

`ST_Collect` — 从一个几何形构建 `GeometryCollection` 或 `Multi*` 几何形。

Synopsis

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

描述

将几何图形收集到几何图形集合中。如果是 **Multi*** 或 **GeometryCollection**，具体取决于输入几何图形是否具有相同或不同类型（同型或异型）。输入几何图形在集合中保持不变。

第一种形式：接受两个输入几何图形

第二种形式：接受几何数组

第三种形式：接受几何行集的聚合函数。



Warning

ST_Collect does not dissolve boundaries or resolve overlaps. Collecting overlapping polygons can produce an invalid MultiPolygon. If the result must be a valid polygonal geometry with overlaps merged, use **ST_Union** instead. **ST_Union** also splits linestrings at intersections and may return a single geometry after dissolving boundaries.

If any input geometry is a collection (**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 input collections to their atomic elements (see example below).

可用性：引入了 1.4.0 - **ST_Collect**（几何）。**ST_Collect** 已得到增强，可以更快地处理更多几何图形。



该函数支持 3d 并且不会丢失 z-index。



此方法支持图形字符串和曲线。

示例

Two-input variant.

合并 2D 点。

Code

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

Output

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

Figure



合并 3D 点。

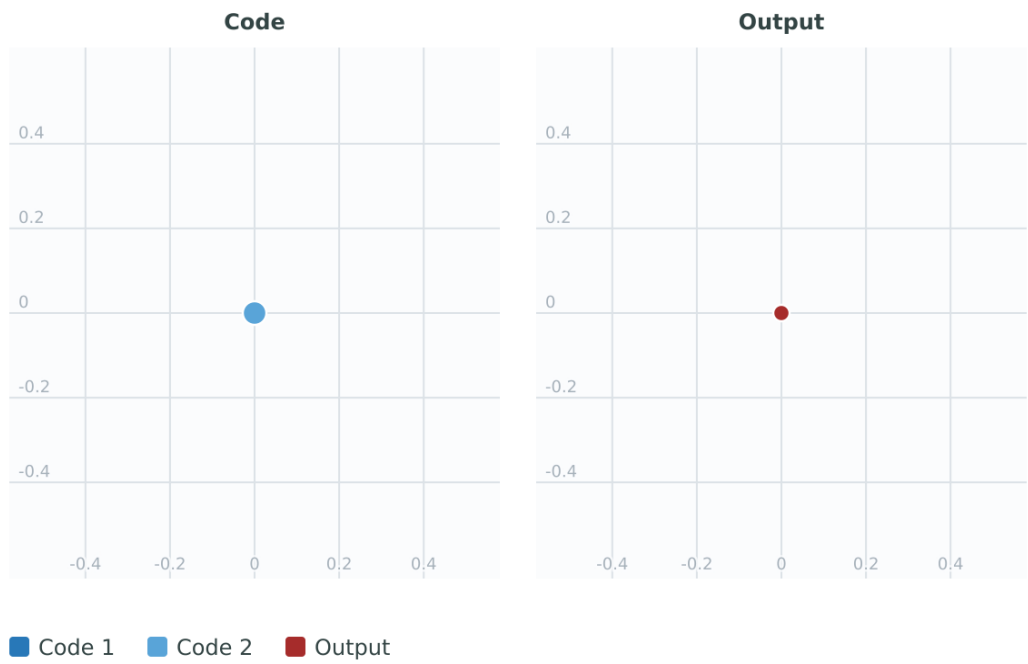
Code

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

Output

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

Figure

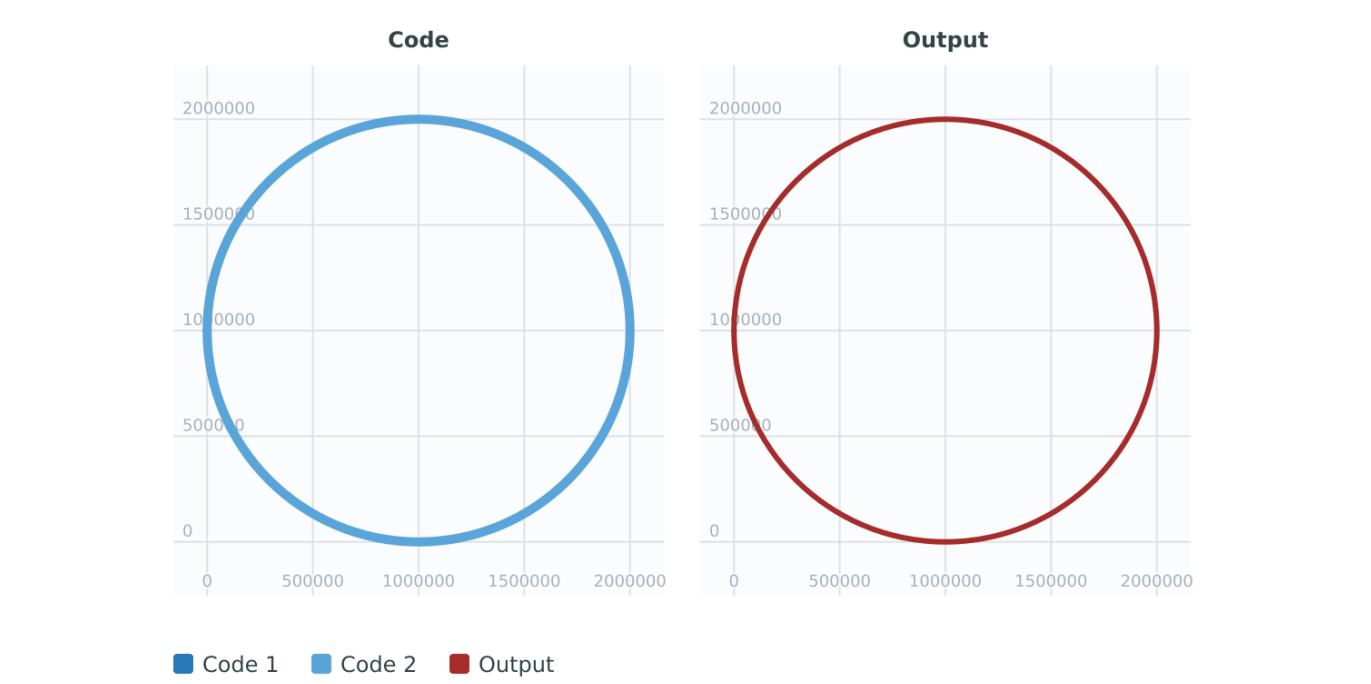


合并曲线。

Code

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

Figure



Array variant.

使用子集的数造函数。

Code

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

使用数造函数来取。

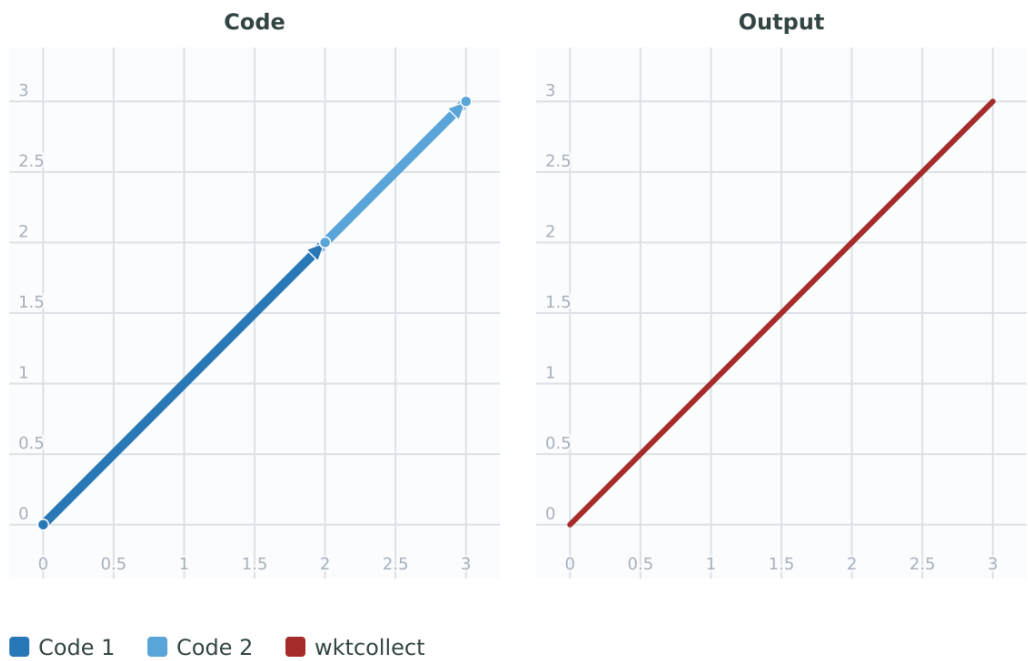
Code

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

Output

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

Figure



Aggregate variant.
通过在表中对几何行分组来生成多个集合。

Code

```
SELECT stusps, ST_Collect(geom) AS geom
FROM (
  SELECT stusps, (ST_Dump(geom)).geom AS geom
  FROM somestatetable
) AS f
GROUP BY stusps;
```

相关信息

[ST_Dump](#), [ST_Union](#)

7.3.2 ST_LineFromMultiPoint

ST_LineFromMultiPoint — 从多点几何构建线串。

Synopsis

geometry **ST_LineFromMultiPoint**(geometry aMultiPoint);

描述

从多点几何形式生成线串。
使用 [ST_MakeLine](#) 从 Point 或 LineString 输入构建。

✓ 函数支持 3d 并且不会丢失 z-index。

示例

从 3D 多点构建 3D 线串

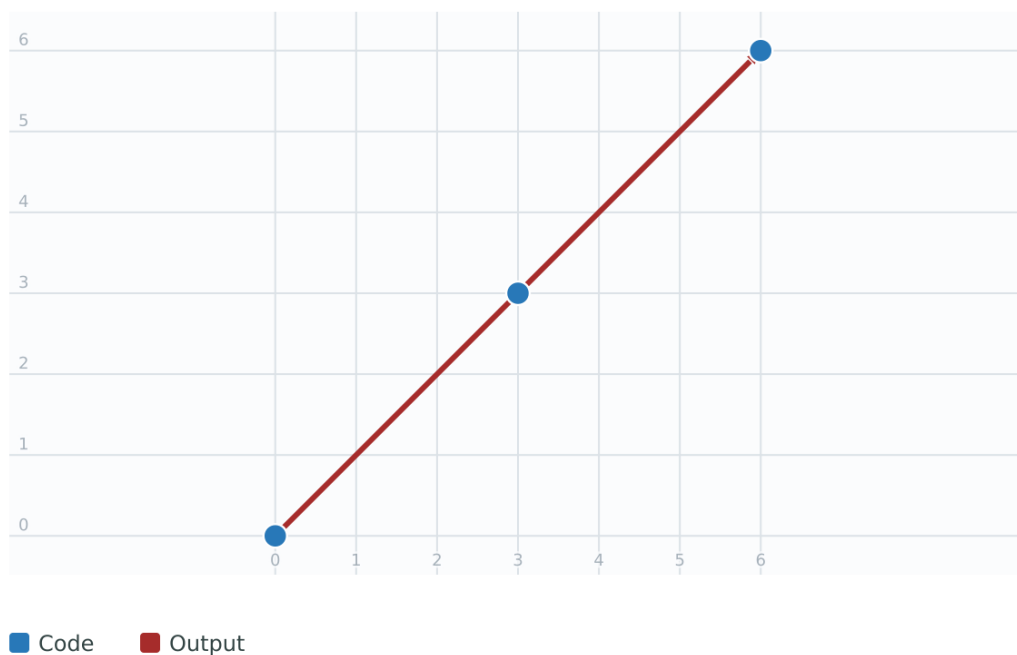
Code

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

Output

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

Figure



相关信息

[ST_AsEWKT](#), [ST_MakeLine](#)

7.3.3 ST_MakeEnvelope

ST_MakeEnvelope — 根据最小和最大坐标构建矩形多边形。

Synopsis

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

描述

根据最小和最大 X 和 Y 生成矩形多边形。输入必须与 SRID 指定的空参考系匹配。如果未指定 SRID，使用未知空参考系 (SRID 0)。

可用性：1.5

增强：2.0：引入了在不指定 SRID 的情况下指定外包矩形的功能。

示例

Building a bounding box polygon.

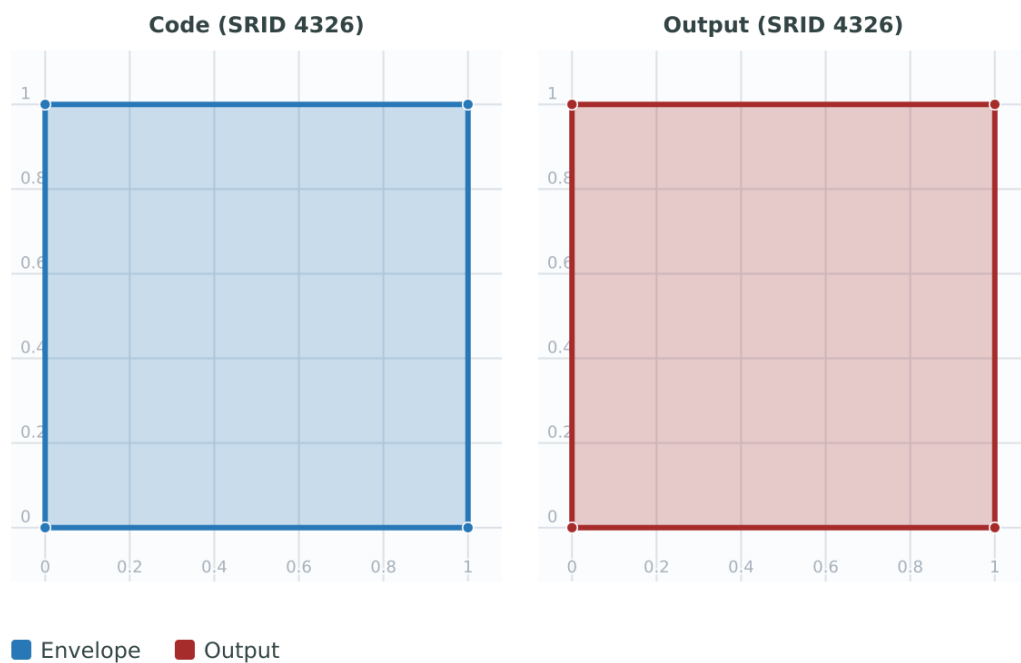
Code

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

Output

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

Figure



相关信息

[ST_MakePoint](#), [ST_MakeLine](#), [ST_MakePolygon](#), [ST_TileEnvelope](#)

7.3.4 ST_MakeLine

`ST_MakeLine` — 从 `Point`, `MultiPoint`, 或 `LineString` geometries 构建 `LineString`。

Synopsis

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

描述

构建包含 Point、MultiPoint 或 LineString 几何形的点的 LineString。其他几何型会致。

第一种形式：接受个入几何形

第二种形式：接受几何数

第三种形式：接受几何行集的聚合函数。为了确保入几何的序，在函数用中使用 **ORDER BY**，或使用有 **ORDER BY** 子句的子。

入 LineString 开重复的点会被折成个点；Point 和 MultiPoint 入中的重复点不会被折。MultiLineString 的各个子段会按照它在集合中的序理。若需要出的 LineString 折重复点，可使用 **ST_RemoveRepeatedPoints**。



函数支持 3d 并且不会失 z-index。

可用性：3.7.0 - 引入了 MultiLineString 入要素的支持

可用性：2.3.0 - 引入了 MultiPoint 入元素的支持

可用性：2.0.0 - 引入了 LineString 入元素的支持

可用性：1.4.0 - 引入了 ST_MakeLine(geomarray)。ST_MakeLine 聚合函数得到增，可以更快地理更多点。

示例

Two-input variant.

生成一条由个点成的。

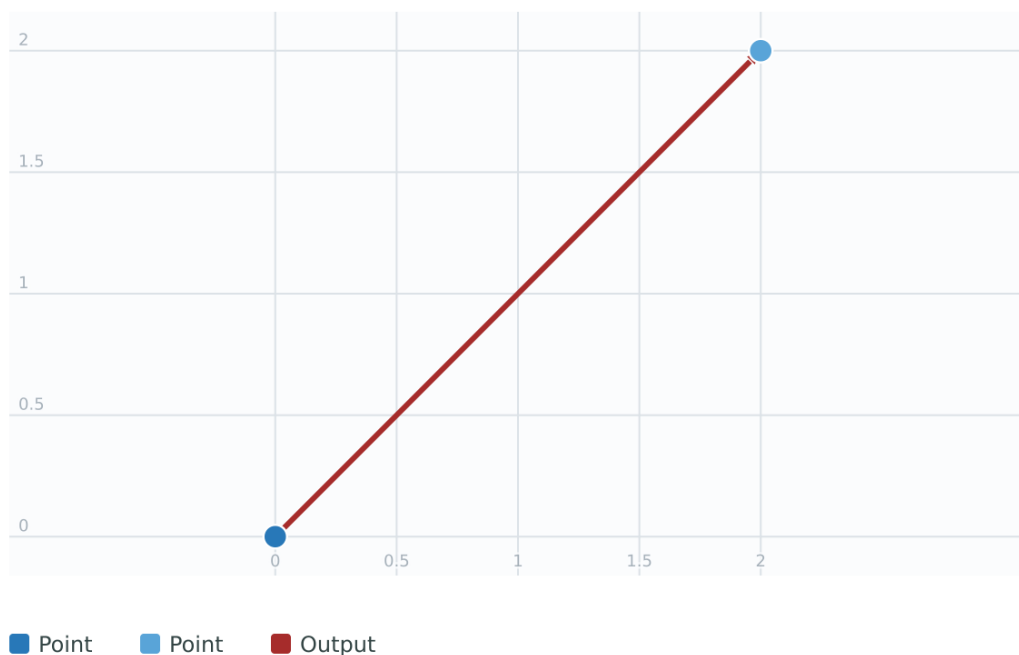
Code

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

Output

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

Figure



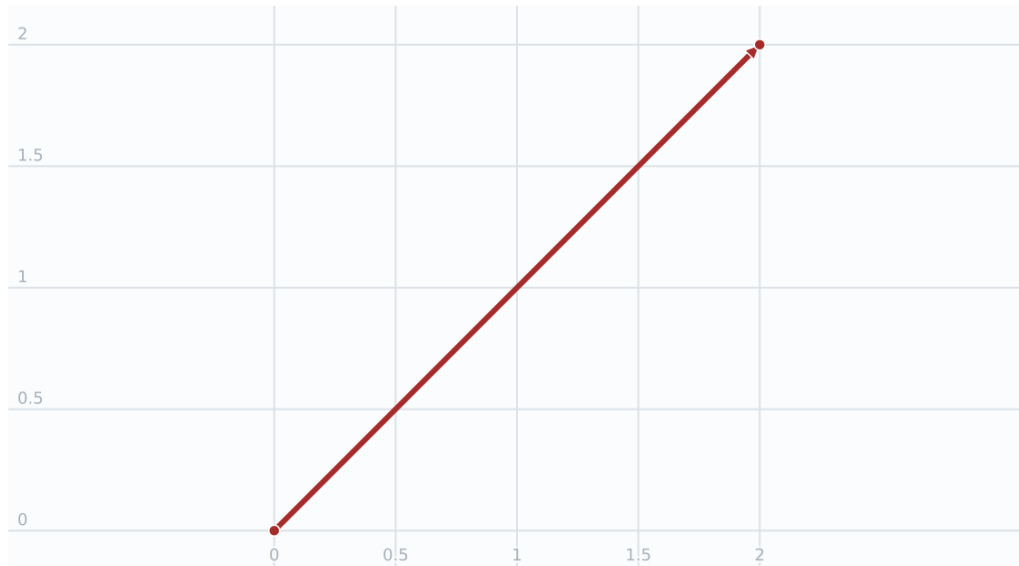
从三个点生成三。

Code

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

Output

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

Figure

■ Output

从 2 个未连接的点串生成一条线。

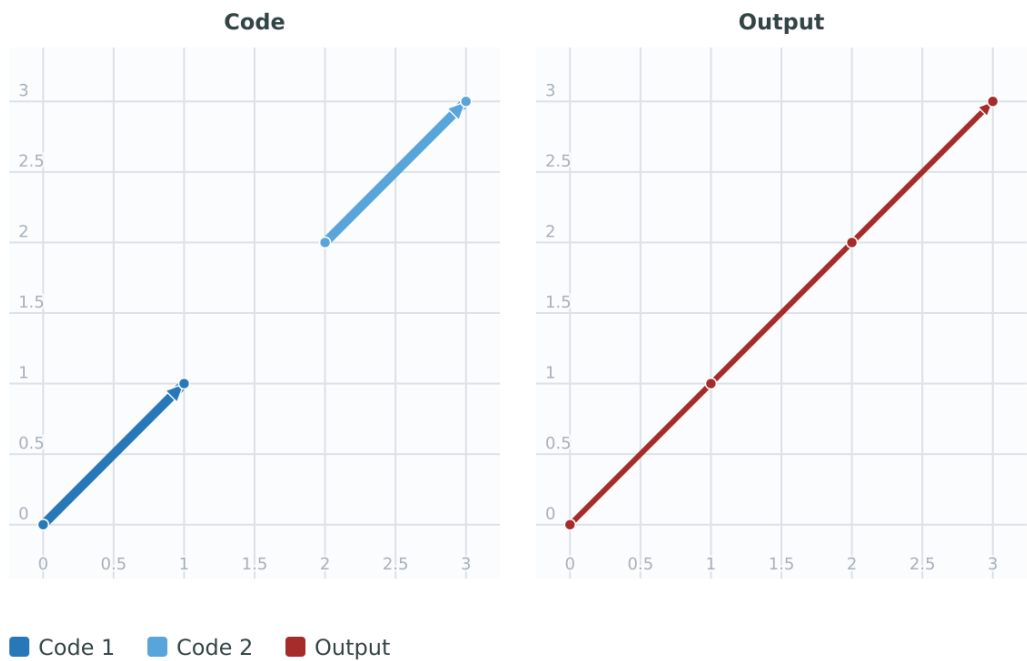
Code

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

Output

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

Figure



Array variant.
从由排序的子形成的数建一条。

Code

```
SELECT ST_MakeLine(ARRAY(SELECT ST_Centroid(geom) FROM visit_locations ORDER BY visit_time) ↵  
);
```

从三点数生成三

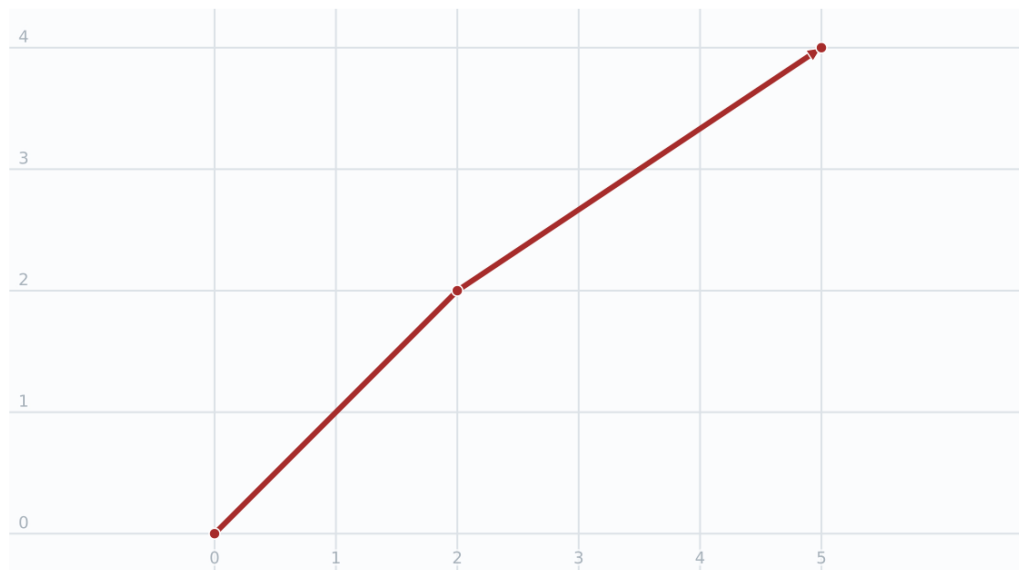
Code

```
SELECT ST_MakeLine(ARRAY[ST_MakePoint(1, 2, 3), ST_MakePoint(3, 4, 5), ST_MakePoint(6, 6, ↵  
6)]);
```

Output

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

Figure



■ Output

Aggregate variant.

此示例从一轨迹中基于行的 GPS 点序列，并为每个轨迹建一条线。结果几何形是由按行顺序的 GPS 迹点成的 LineStrings。

ORDER BY 子句可用于以正确的顺序生成串。

Code

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

当在聚合内部无法保证排序时，可以先在子句中先行排序。不过，有并行化器并不会遵循子句的排序结果，导致输出顺序依然无法保证。

Code

```
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 gps.track_id;
```

相关信息

[ST_RemoveRepeatedPoints](#), [ST_AsEWKT](#), [ST_AsText](#), [ST_GeomFromText](#), [ST_MakePoint](#), [ST_Point](#)

7.3.5 ST_MakePoint

ST_MakePoint — 建 2D、3DZ 或 4D 点。

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);
```

描述

创建一个 2D XY、3D XYZ 或 4D XYZM 的点几何对象。使用 **ST_MakePointM** 来创建有 XYM 坐的点。使用 **ST_SetSRID** 来创建的点指定一个空参考系 (SRID)。

然不符合 OGC 范, **ST_MakePoint** 比 **ST_GeomFromText** 和 **ST_PointFromText** 更快。此外, 它可以使用数坐。



Note

于大地坐, X 是度, Y 是度



Note

可以使用函数 **ST_Point**、**ST_PointZ**、**ST_PointM** 和 **ST_PointZM** 来建具有指定 SRID 的点。



函数支持 3d 并且不会失 z-index。

示例

This example creates a point with an unknown SRID.

Code

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

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

Code

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

This example creates a 3D point with altitude.

Code

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

This example returns the Z coordinate of a point.

Code

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

| Output

1.5

相关信息

[ST_GeomFromText](#), [ST_PointFromText](#), [ST_SetSRID](#), [ST_MakePointM](#), [ST_Point](#), [ST_PointZ](#), [ST_PointM](#), [ST_PointZM](#)

7.3.6 ST_MakePointM

`ST_MakePointM` — 根据 X、Y 和 M 构建一个点。

Synopsis

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

描述

构建一个具有 X、Y 和 M（量）坐的点。使用 [ST_MakePoint](#) 来构建具有 XY、XYZ 或 XYZM 坐的点。使用 [ST_SetSRID](#) 来构建的点指定一个空参考（SRID）。



Note

于大地坐，X 是度，Y 是度



Note

函数 [ST_PointM](#) 和 [ST_PointZM](#) 可用于构建具有 M 和指定 SRID 的点。

示例



Note

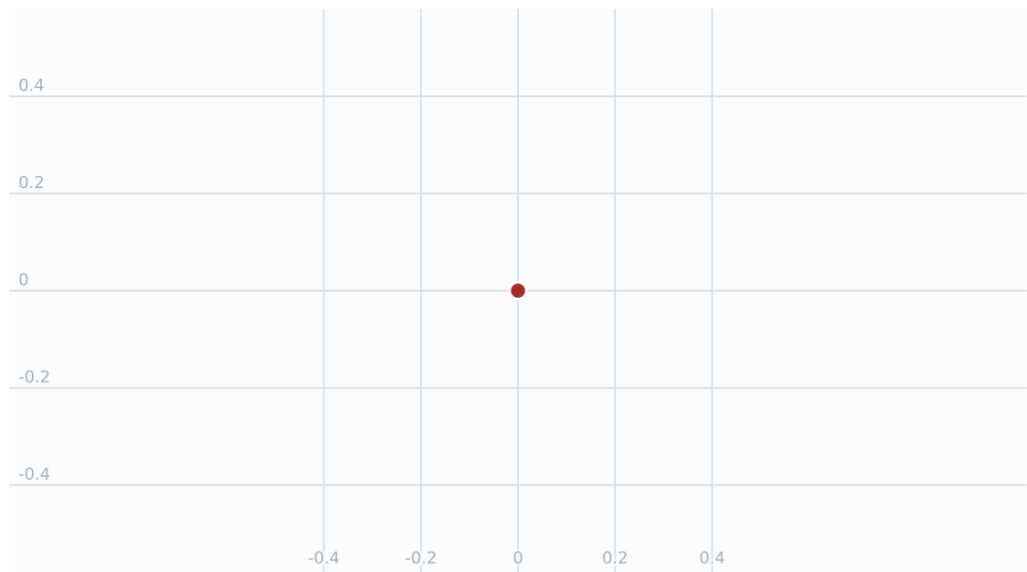
[ST_AsEWKT](#) `ST_AsEWKT` 用于字符串出。是因 [ST_AsText](#) 与 M 不。

生成具有未知 SRID 的点。

| Code

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

Figure



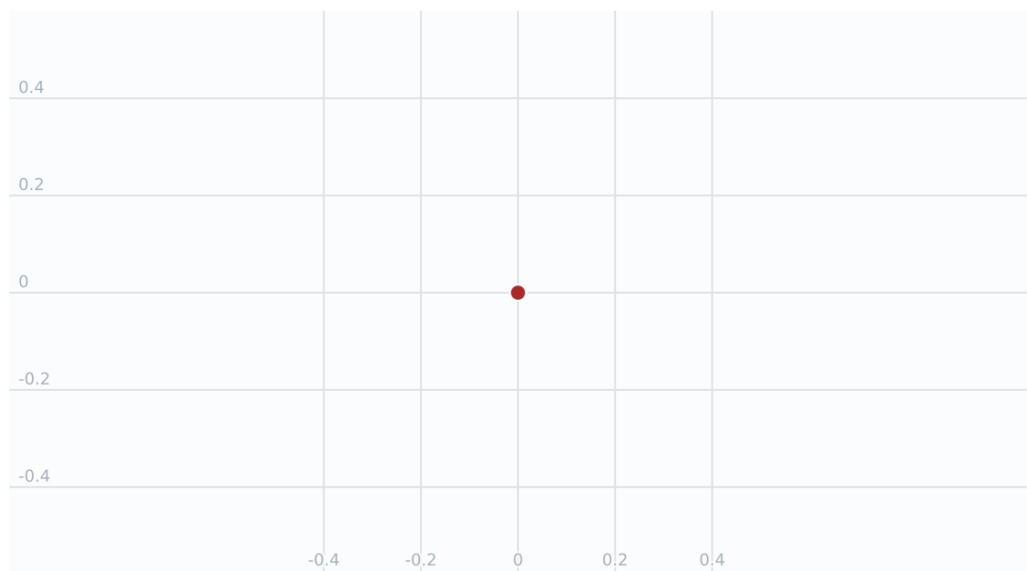
Output

在 WGS 84 地理坐标系中生成具有 M 值的点。

Code

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

Figure



Output

取生成点的 M 值。

Code

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

Output

```
10
```

相关信息

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

7.3.7 ST_MakePolygon

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

Synopsis

geometry **ST_MakePolygon**(geometry linestring);

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

描述

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

Variant 1: Accepts one shell ring.

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

**Note**

此函数不接受多行字符串。使用 [ST_LineMerge](#) 生成☐串。它☐使用 [ST_Dump](#) 来提取☐串。



☐函数支持 3d 并且不会☐失 z-index。

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

示例

Single input variant.

从 2D ☐串生成多☐形。

Code

```
SELECT ST_MakePolygon('LINESTRING(75 29,77 29,77 29,75 29)');
```

使用 [ST_StartPoint](#) 和 [ST_AddPoint](#)☐合开放☐串以生成面。

Code

```
SELECT ST_MakePolygon(ST_AddPoint(foo.open_line, ST_StartPoint(foo.open_line)) )
FROM (
  SELECT 'LINESTRING(75 29,77 29,77 29,75 29)::geometry As open_line) As foo;
```

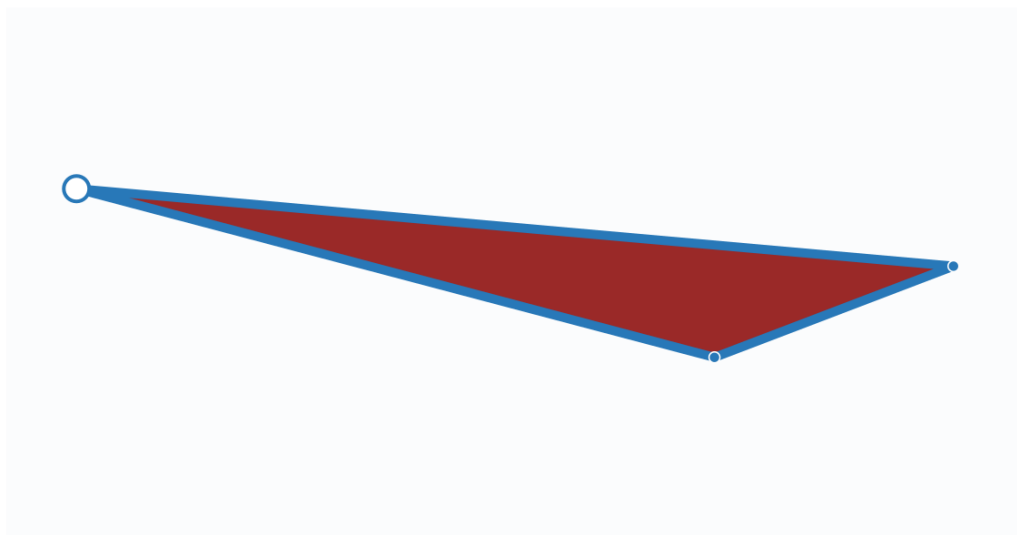
从 3D 线串生成多边形

Code

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

Output

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

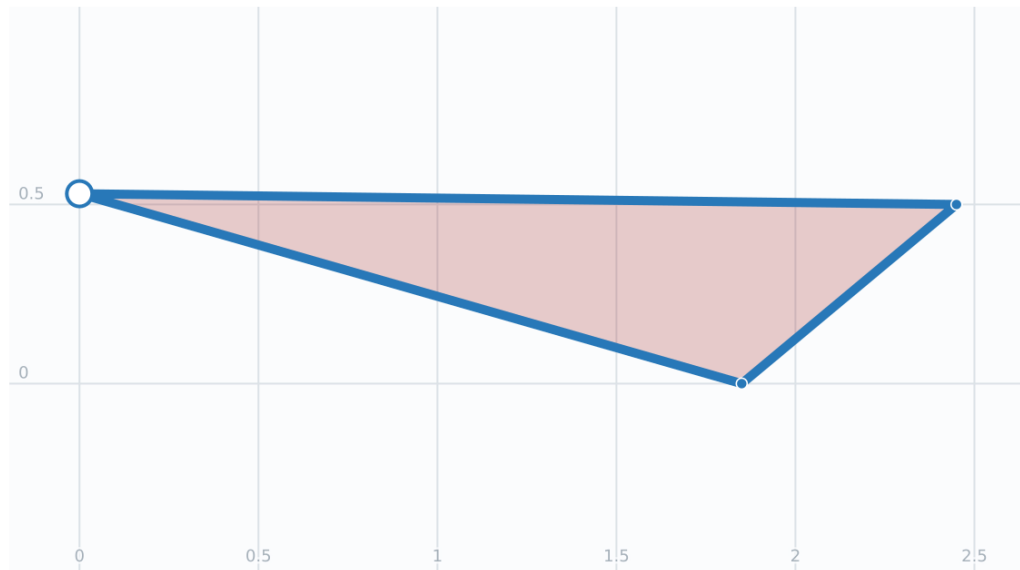
Figure**3D view**

■ Code ■ Output

从具有 M 值的线串生成多边形

Code

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

| Figure

■ Code ■ Output

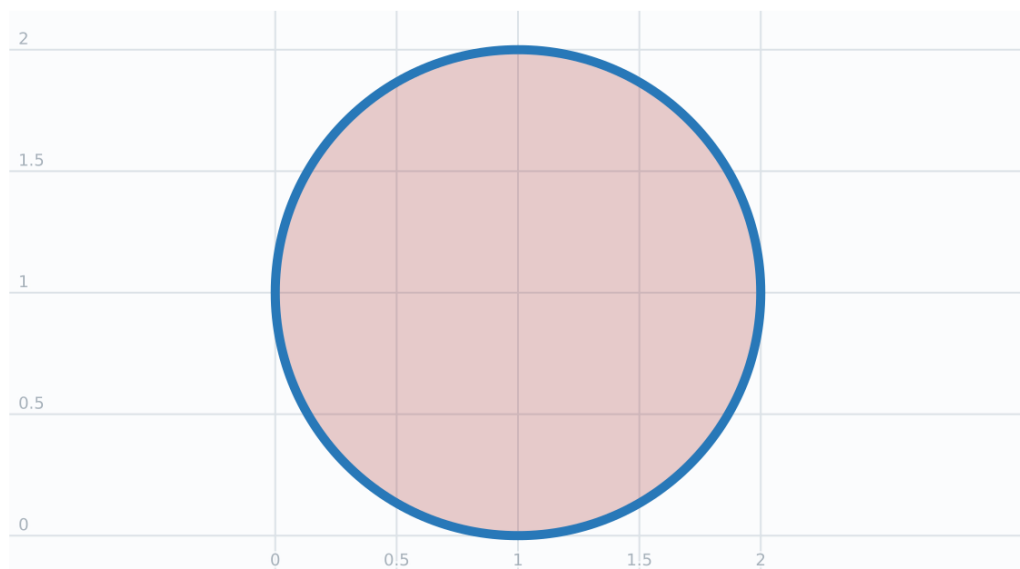
Create a CurvePolygon from a CircularString shell.

| Code

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

| Output

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

| Figure

■ Code ■ Output

Outer shell with inner holes variant.

☐ 建立一个☐有☐外孔的☐形多☐形

Code

```
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;
```

☐ 建立一个☐省份☐界，其中的孔代表湖泊。☐ 入是省份多☐形/多☐形表和水域☐串表。形成湖泊的☐是通☐使用 **ST_IsClosed** 确定的。使用 **ST_Boundary** 提取省份☐条。根据 **ST_MakePolygon** 的要求，通☐使用 **ST_LineMerge** 将☐界☐制☐个 LineString。 (但是，☐注意，如果一个省份☐有多个区域或☐有☐☐，☐将生成无效的多☐形。) 使用 LEFT JOIN 可确保包括所有省份，即使它☐没有湖泊。



Note

使用 CASE ☐造是因☐将空数☐☐☐到 ST_MakePolygon 会☐致 NULL 返回☐。

Code

```
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 AS geom
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;
```

☐ 一种技☐是利用相关子☐☐和 ARRAY() ☐造函数将行集☐☐☐数☐。

Code

```
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;
```

相关信息

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

7.3.8 ST_Point

ST_Point — 创建具有 X、Y 和 SRID 的点。

Synopsis

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

描述

返回具有给定 X 和 Y 坐标的 Point。它是 [ST_MakePoint](#) 的 SQL-MM 等效，采用 X 和 Y。



Note

关于大地坐标，X 是经度，Y 是纬度

增量：3.2.0 srid 作为可选参数被添加。旧的安装需要与 ST_SetSRID 结合以在几何体上设置 srid。



该方法符合了 SQL/MM 规范。SQL-MM 3: 6.1.2

示例

Geometry.

Code

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

创建一个指定了 SRID 的点：

Code

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

指定 SRID 的一种方式：

Code

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

创建 **地理** 点（使用 :: 运算符）：

Code

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

在 Pre-PostGIS 3.2 版本的代码中，使用 CAST：

Code

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

如果点的坐标不在地理坐标系（如 WGS84）中，必须先进行坐标转换，然后再将其投射到地理。在此示例中，马萨诸塞州平面英尺（SRID 2273）上的点将转换为 WGS84（SRID 4326）。

Code

```
SELECT ST_Transform(ST_Point(3637510, 3014852, 2273), 4326)::geography;
```

相关信息

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

7.3.9 ST_PointZ

ST_PointZ — 创建具有 X、Y、Z 和 SRID 的点。

Synopsis

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

描述

生成具有给定 X、Y 和 Z 坐标的点，如果给定，则具有 SRID。

增修：3.2.0 srid 作为可选参数被添加。旧的安装需要与 ST_SetSRID 配合以在几何体上设置 srid。

示例**Code**

```
SELECT ST_PointZ(-71.104, 42.315, 3.4, 4326)
```

Code

```
SELECT ST_PointZ(-71.104, 42.315, 3.4, srid => 4326)
```

Code

```
SELECT ST_PointZ(-71.104, 42.315, 3.4)
```

相关信息

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

7.3.10 ST_PointM

ST_PointM — 创建具有 X、Y、M 和 SRID 的点。

Synopsis

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

描述

生成一个点，点定义了 X、Y 和 M 坐标，如果指定，点具有 SRID。

增：3.2.0 srid 作外的可参数被添加。旧的安装需要与 ST_SetSRID 合以在几何体上 srid。

示例

Code

```
SELECT ST_PointM(-71.104, 42.315, 3.4, 4326)
```

Code

```
SELECT ST_PointM(-71.104, 42.315, 3.4, srid => 4326)
```

Code

```
SELECT ST_PointM(-71.104, 42.315, 3.4)
```

相关信息

[ST_MakePoint](#), [ST_Point](#), [ST_PointZ](#), [ST_PointZM](#)

7.3.11 ST_PointZM

ST_PointZM — 创建具有 X、Y、Z、M 和 SRID 的点。

Synopsis

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

描述

生成一个点，点定义了 X、Y、Z、M 坐标，如果指定，点具有 SRID。

增：3.2.0 srid 作外的可参数被添加。旧的安装需要与 ST_SetSRID 合以在几何体上 srid。

示例

Code

```
SELECT ST_PointZM(-71.104, 42.315, 3.4, 4.5, 4326)
```

Code

```
SELECT ST_PointZM(-71.104, 42.315, 3.4, 4.5, srid => 4326)
```

Code

```
SELECT ST_PointZM(-71.104, 42.315, 3.4, 4.5)
```

相关信息

[ST_MakePoint](#), [ST_Point](#), [ST_PointM](#), [ST_PointZ](#), [ST_SetSRID](#)

7.3.12 ST_Polygon

ST_Polygon — 从具有指定 SRID 的线串建多边形。

Synopsis

geometry **ST_Polygon**(geometry lineString, integer srid);

描述

返回根据给定 LineString 建的多边形，并根据 srid 置空参考系。

ST_Polygon 似于 [ST_MakePolygon](#) 格式 1，但添加了 SRID 置。

要建孔的多边形，使用 [ST_MakePolygon](#) 格式 2，然后使用 [ST_SetSRID](#)。

**Note**

此函数不接受多行字符串。使用 [ST_LineMerge](#) 生成线串。它使用 [ST_Dump](#) 来提取线串。



此方法符合了 [SQL 1.1](#) 的 OGC 功能范。



方法符合了 SQL/MM 范。SQL-MM 3: 8.3.2



函数支持 3d 并且不会失 z-index。

示例

生成二多边形。

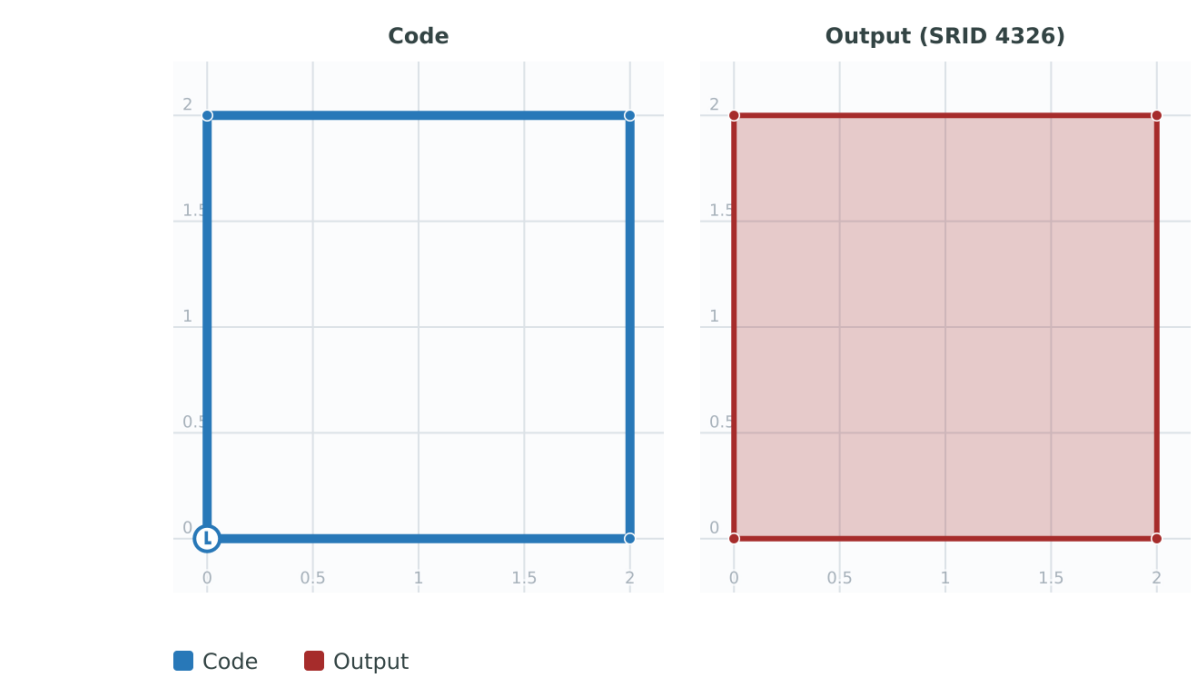
Code

```
SELECT ST_Polygon('LINESTRING(75 29,77 29,77 31,75 31,75 29)::geometry, 4326);
```

Output

```
POLYGON((75 29,77 29,77 31,75 31,75 29))
```

Figure



生成三多形。

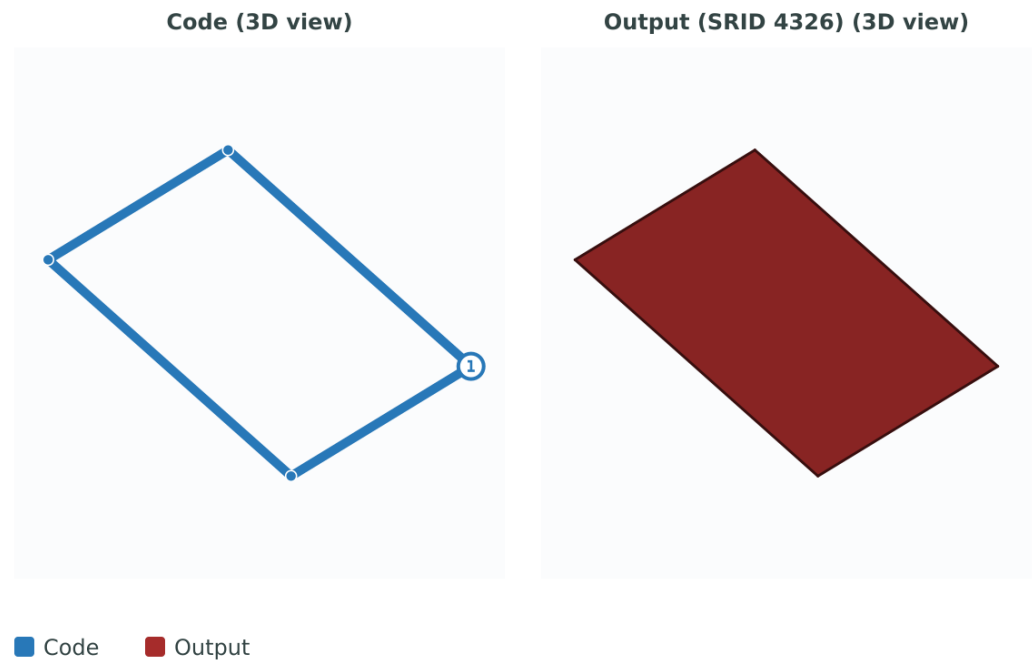
Code

```
SELECT ST_Polygon('LINESTRING(75 29 1,77 29 2,77 31 2,75 31 1,75 29 1)::geometry, 4326);
```

Output

```
SRID=4326;POLYGON((75 29 1,77 29 2,77 31 2,75 31 1,75 29 1))
```

Figure



相关信息

[ST_AsEWKT](#), [ST_AsText](#), [ST_GeomFromEWKT](#), [ST_GeomFromText](#), [ST_LineMerge](#), [ST_MakePolygon](#)

7.3.13 ST_TileEnvelope

ST_TileEnvelope — 使用 XYZ 切片系在 Web Mercator（SRID : 3857）中构建矩形多边形。

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

构建一个矩形多边形，给出 XYZ 切片系中切片的范围。范围由放大 Z 和范围的网格中范围的 XY 索引指定。可用于定义 ST_AsMVTGeom所需的范围边界，以将几何形状放入 MVT 范围坐标系空。

默认情况下，切片范围位于Web Mercator坐标系 (SRID:3857) 中，使用 Web 墨卡托系数的范围 (-20037508.342789、20037508.342789)。是 MVT 最常用的坐标系。可的 bounds 参数可用于在任何坐标系中生成范围。它是一个几何体，具有 SRID 和“范围零”正方形的范围，XYZ 范围系内切在正方形内。

可的 margin 参数可用于将范围展定百分比。例如。margin=0.125 将范围展 12.5%，相当于当范围大小 4096 范围 buffer=512（如 ST_AsMVTGeom中使用的）。用于构建范围缓冲区以包含位于范围区域之外的数据非常有用，但其存在会影范围渲染。例如，城市名称（一个点）可能靠近范围的范围，因此其范围范围呈在范围范围上，即使点位于一个范围的可范围区域中。在范围中使用范围范围将在范围范围中包含城市点。使用范围来小范围。禁止使用小于 -0.5 的范围，因范围会完全消除范围。与 ST_AsMVTGeom 一起使用范围，范围勿指定范围。参范围 ST_AsMVT。

增⬢：添加了 3.1.0 margin 参数。

可用性：3.0.0

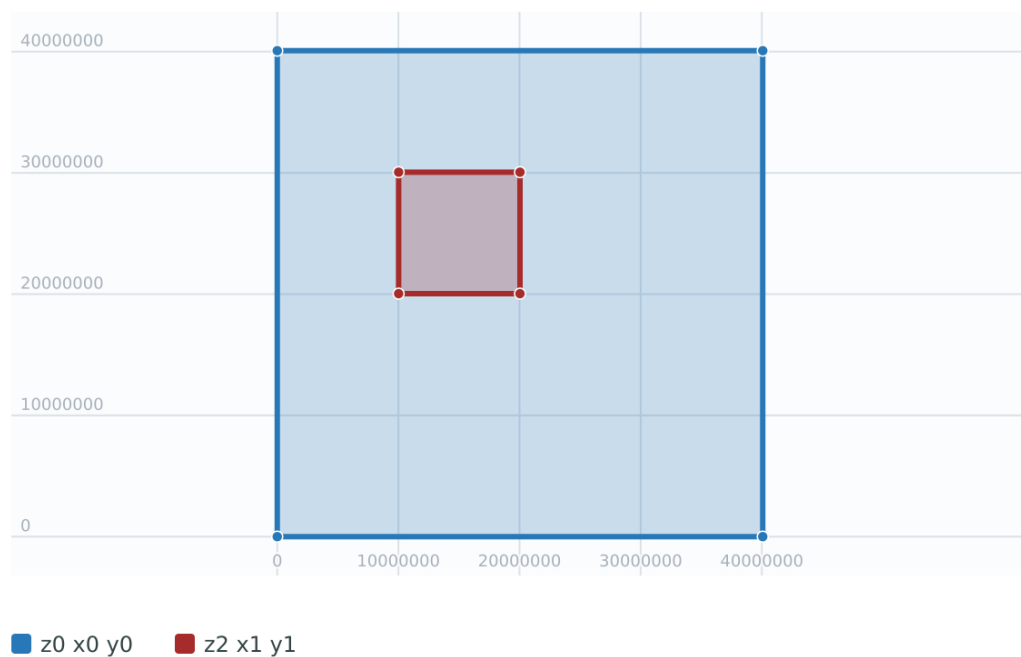
示例

Show tile (2, 1, 1) within the root tile (0, 0, 0).

Code

```
SELECT ST_TileEnvelope(0, 0, 0) AS input_z0_x0_y0, ST_TileEnvelope(2, 1, 1) AS z2_x1_y1;
```

Figure

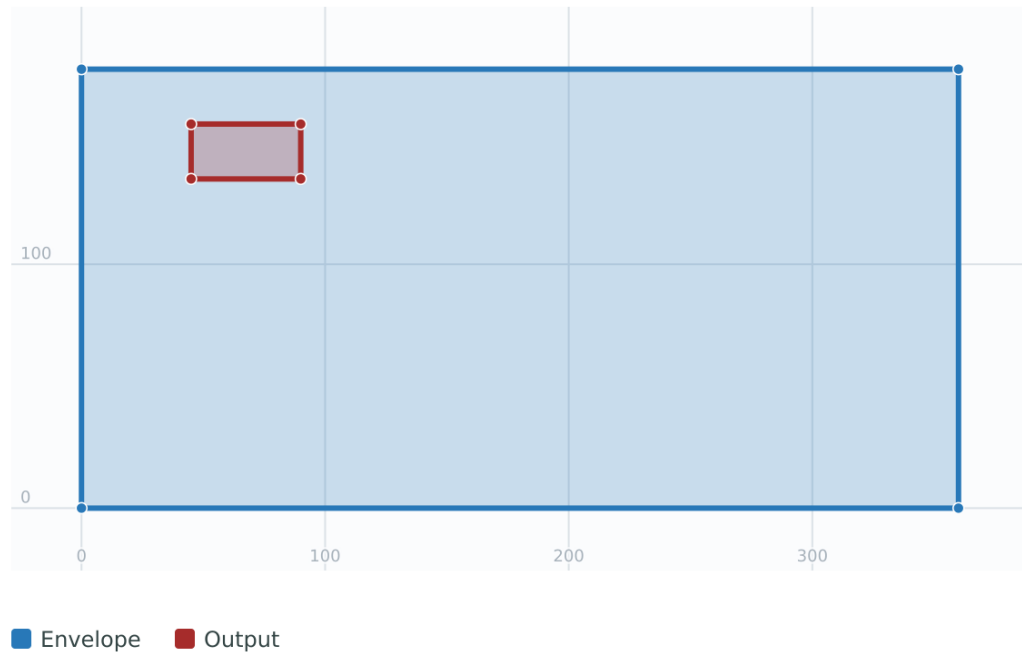


Code

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

Output

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

Figure

相关信息

ST_MakeEnvelope**7.3.14 ST_HexagonGrid**

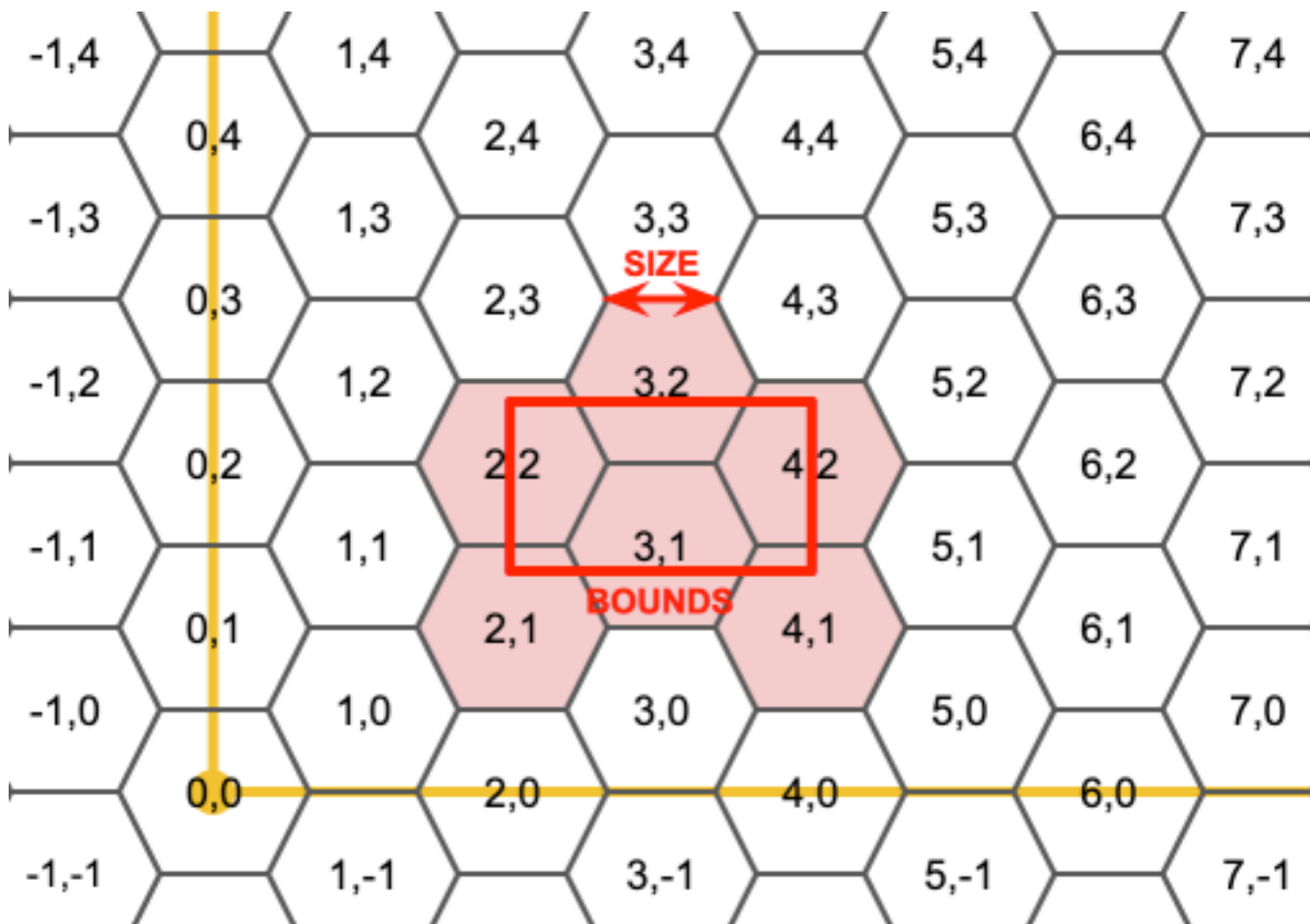
ST_HexagonGrid — 返回一个完全覆盖几何参数边界的六边形和元格索引。

Synopsis

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

描述

Starts with the concept of a hexagon tiling of the plane. (Not a hexagon tiling of the globe; for H3 use the **h3-pg** extension.) 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.



六边形的 SRS 是边界几何体提供的 SRS。

将六边形的尺寸加倍或三倍会生成与原始格网相匹配的新父格网。不幸的是，不可能生成子格网完全适合内部的父六边形格网。

可用性：3.1.0

示例

Unit hexagons covering rectangular bounds.

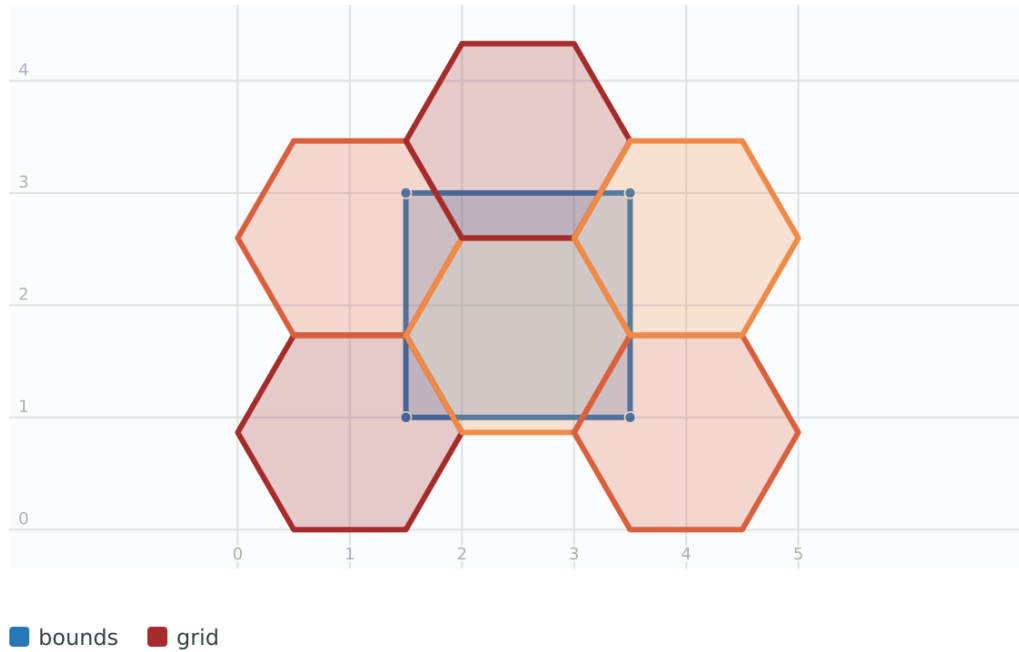
Code

```
WITH bounds AS (
  SELECT ST_MakeEnvelope(2, 1, 4, 3, 3857) AS geom
), grid AS (
  SELECT hex.*
  FROM bounds
  CROSS JOIN LATERAL ST_HexagonGrid(1, geom) AS hex
)
SELECT (SELECT geom FROM bounds) AS input_bounds,
       ST_Collect(geom ORDER BY i, j) AS grid
FROM grid;
```


Output

```
POLYGON((2 1,2 3,4 3,4 1,2 1)) | MULTIPOLYGON(((0.5 0.866,1 0,2 0,2.5 0.866,2 1.732,1 1.732,0.5 0.866)),((0.5 2.598,1 1.732,2 1.732,2.5 2.598,2 3.464,1 3.464,0.5 2.598)),((2 1.732,2.5 0.866,3.5 0.866,4 1.732,3.5 2.598,2.5 2.598,2 1.732)),((2 3.464,2.5 2.598,3.5 2.598,4 3.464,3.5 4.33,2.5 4.33,2 3.464)),((3.5 0.866,4 0,5 0,5.5 0.866,5 1.732,4 1.732,3.5 0.866)),((3.5 2.598,4 1.732,5 1.732,5.5 2.598,5 3.464,4 3.464,3.5 2.598)))
```

Figure

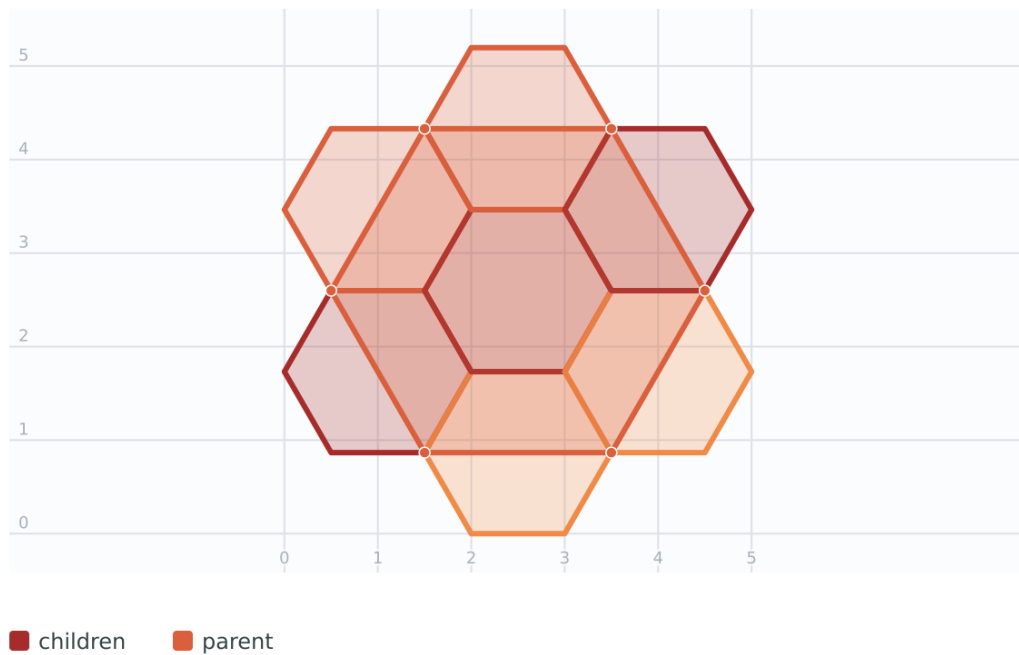


Child tiles that intersect a doubled-size parent hexagon share the tiling origin, but do not nest perfectly inside the parent.

Code

```
WITH parent AS (
  SELECT ST_Hexagon(2, 0, 0) AS geom
), children AS (
  SELECT child.*
  FROM parent
  CROSS JOIN LATERAL ST_HexagonGrid(1, geom) AS child
  WHERE ST_Area(ST_Intersection(child.geom, parent.geom)) > 0
)
SELECT ST_Collect(geom ORDER BY i, j) AS children,
       (SELECT geom FROM parent) AS parent
FROM children;
```

Figure



Counting points in hexagons.

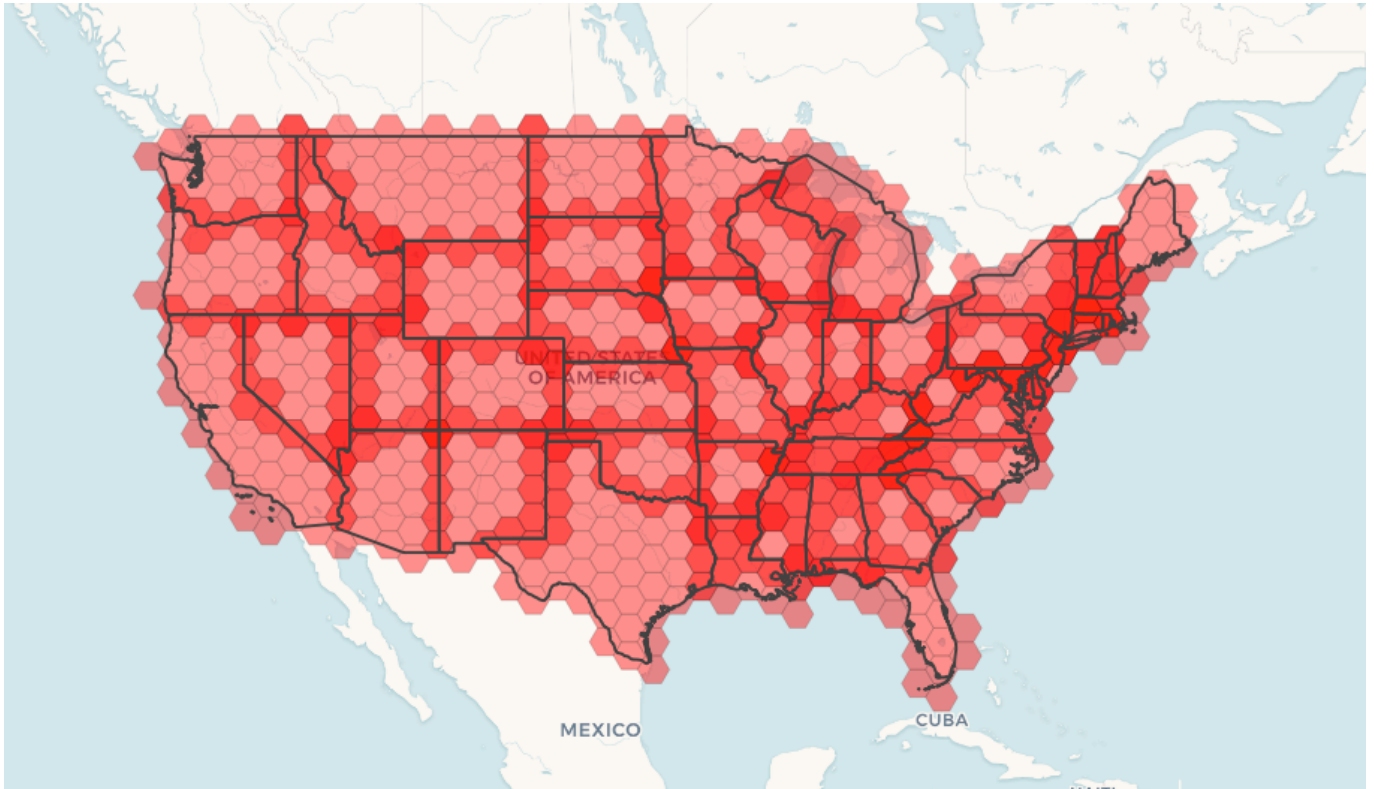
要生成六边形网格，使用点的范围作边界生成六边形网格，然后在空上接到网格。

Code

```
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;
```

Generating hex coverage of polygons.

If a grid is generated for each polygon boundary and filtered to intersecting cells, adjacent regions receive overlapping hexagons along their shared border. The build-time figure below uses two simple regions in place of an external administrative-boundary table, so the example is self-contained.



Code

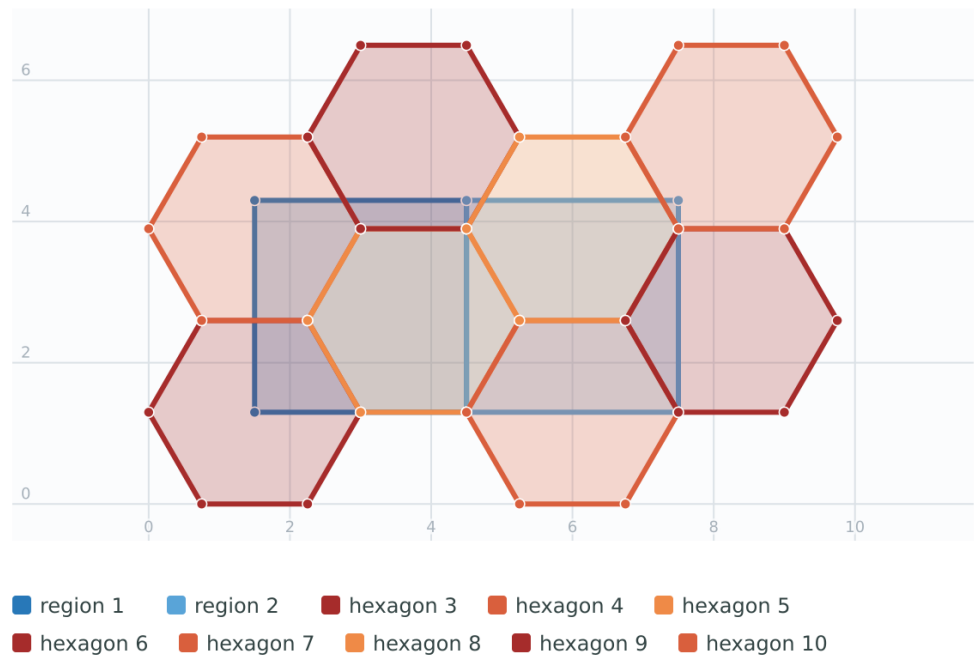
```
WITH admin1(gid, geom) AS (
    VALUES
        (1, ST_MakeEnvelope(0, 0, 3, 3, 3857)),
        (2, ST_MakeEnvelope(3, 0, 6, 3, 3857))
), coverage AS (
    SELECT hex.i, hex.j, hex.geom
    FROM admin1
    CROSS JOIN LATERAL ST_HexagonGrid(1.5, admin1.geom) AS hex
    WHERE ST_Intersects(admin1.geom, hex.geom)
    GROUP BY hex.i, hex.j, hex.geom
), layers AS (
    SELECT 'region' AS kind, gid AS position, geom
    FROM admin1
    UNION ALL
    SELECT
        'hexagon',
        1000 + row_number() OVER (ORDER BY i, j),
        geom
    FROM coverage
)
SELECT
    CASE kind WHEN 'region' THEN geom END AS input_region,
    CASE kind WHEN 'hexagon' THEN geom END AS hexagon
FROM layers
ORDER BY position;
```

Output

```
POLYGON((0 0,0 3,3 3,3 0,0 0)) | null
POLYGON((3 0,3 3,6 3,6 0,3 0)) | null
null | POLYGON((-1.5 0,-0.75 -1.3,0.75 -1.3,1.5 0,0.75 1.3,-0.75 1.3,-1.5 0))
null | POLYGON((-1.5 2.6,-0.75 1.3,0.75 1.3,1.5 2.6,0.75 3.9,-0.75 3.9,-1.5 2.6))
null | POLYGON((0.75 1.3,1.5 0,3 0,3.75 1.3,3 2.6,1.5 2.6,0.75 1.3))
null | POLYGON((0.75 3.9,1.5 2.6,3 2.6,3.75 3.9,3 5.2,1.5 5.2,0.75 3.9))
```

```
null | POLYGON((3 0,3.75 -1.3,5.25 -1.3,6 0,5.25 1.3,3.75 1.3,3 0))
null | POLYGON((3 2.6,3.75 1.3,5.25 1.3,6 2.6,5.25 3.9,3.75 3.9,3 2.6))
null | POLYGON((5.25 1.3,6 0,7.5 0,8.25 1.3,7.5 2.6,6 2.6,5.25 1.3))
null | POLYGON((5.25 3.9,6 2.6,7.5 2.6,8.25 3.9,7.5 5.2,6 5.2,5.25 3.9))
```

Figure



The same pattern applies to a real administrative-boundary table:



Note
当引用 FROM 列表中的先前表时，LATERAL 关键字包含在返回集合的函数中。因此，CROSS JOIN LATERAL、CROSS JOIN 或表的 CROSS JOIN LATERAL 是本示例的等效构造。

Code

```
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)
```

相关信息

[ST_EstimatedExtent](#), [ST_SetSRID](#), [ST_SquareGrid](#), [ST_TileEnvelope](#)

7.3.15 ST_Hexagon

ST_Hexagon — 使用提供的尺寸和六边形网格空间内的元素坐标返回一个六边形。

Synopsis

geometry **ST_Hexagon**(float8 size, integer cell_i, integer cell_j, geometry origin);

描述

使用与 **ST_HexagonGrid** 相同的六边形格网概念，但在所需的单元格坐标处生成一个六边形。（可选）可以指定格网的原点坐标，默认为原点 0,0。

生成的六边形没有设置 SRID，因此使用 **ST_SetSRID** 将 SRID 设置为您期望的。

可用性：3.1.0

示例

Creating a hexagon at the origin.

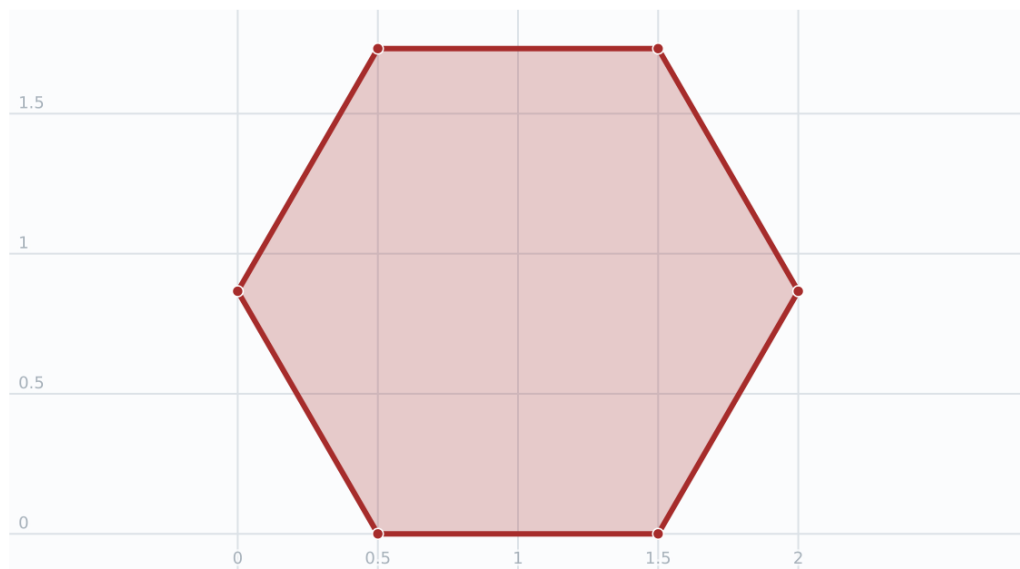
Code

```
SELECT ST_SetSRID(ST_Hexagon(1.0, 0, 0), 3857);
```

Output

```
POLYGON((-1 0,-0.5 -0.8660254037844387,0.5 -0.8660254037844387,1 0,0.5 0.8660254037844387,-0.5 0.8660254037844387,-1 0))
```

Figure



Output

相关信息

ST_TileEnvelope, **ST_HexagonGrid**, **ST_Square**

7.3.16 ST_SquareGrid

ST_SquareGrid — 返回一个完全覆盖几何参数边界的网格正方形和元格索引。

Synopsis

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

描述

从平面正方形格网的概念开始。对于一个给定的平面 SRS 和给定的尺寸，从 SRS 的原点开始，存在一个独特的平面方形格网，Tiling(SRS, Size)。此函数回答了以下问题：给定 Tiling(SRS, Size) 中的哪些网格与给定边界重叠。

输出正方形的 SRS 是边界几何的 SRS。

正方形的加倍或尺寸大小会生成与原始格网完美契合的新父格网。墨卡托中的基准网格地格网只是墨卡托平面中的四个平方网格的。

可用性：3.1.0

示例

Generating a 1 degree grid for a country.

网格将填充国家/地区的整个边界，因此如果您只想接触国家/地区的方格，必须随后使用 ST_Intersects 进行过滤。

Code

```
WITH grid AS (
SELECT (ST_SquareGrid(1, ST_Transform(geom, 4326))).*
FROM admin0 WHERE name = 'Canada'
)
SELECT geom
FROM grid
```

Counting points in squares (using single chopped grid).

要过滤方形格网行点聚合，使用点的范围作为边界生成方形网格，然后在空集上连接到网格。注意，估计范围可能与范围有所不同，因此必须谨慎，至少确保您已分析了表。

Code

```
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;
```

Counting points in squares using set of grid per point.

会生成与第一个示例相同的结果，但由于大量点来会慢

Code

```
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;
```

相关信息

[ST_TileEnvelope](#), [ST_HexagonGrid](#), [ST_EstimatedExtent](#), [ST_SetSRID](#)

7.3.17 ST_Square

ST_Square — 使用提供的尺寸大小和正方形网格空间内的元格坐标返回一个正方形。

Synopsis

geometry **ST_Square**(float8 size, integer cell_i, integer cell_j, geometry origin='POINT(0 0)');

描述

使用与[ST_SquareGrid](#)相同的正方形格网概念，但在所需的元格坐标生成一个正方形。（可选）可以调整平面的原点坐标，默认为原点 0,0。

正方形将使用指定原点的 SRID 生成。如果指定原点的 SRID 不明（默认为空情况），可以使用 [ST_SetSRID](#) 来设置 SRID。

可用性：3.1.0

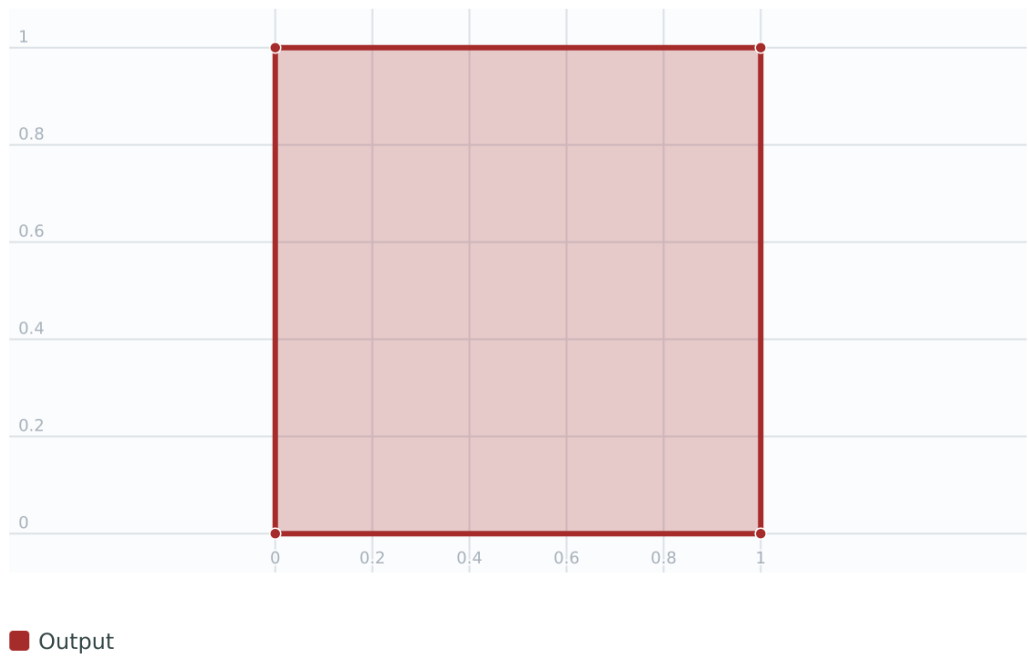
示例

Creating a square at the origin.

Code

```
SELECT ST_SetSRID(ST_Square(1.0, 0, 0), 3857);
```

Figure



相关信息

[ST_TileEnvelope](#), [ST_SquareGrid](#), [ST_Hexagon](#)

7.3.18 ST_Letters

ST_Letters — 返回渲染几何形的入字母，默起始位置位于原点，默文本高度 100。

Synopsis

geometry **ST_Letters**(text letters, json font);

描述

使用内置字体将字符串渲染多形几何体。默文本高度 100.0，即从下行字母底部到大写字母部的距离。默起始位置将基的起点置于原点。覆盖字体涉及入一个 json 映射，以字符，并以 base64 的 TWKB 作字体形状，字体从下行底部到大写部的高度 1000 个位。

默情况下，文本是在原点生成的，因此要重新定位文本并整文本大小，首先用 ST_Scale 函数，然后用 ST_Translate 函数。

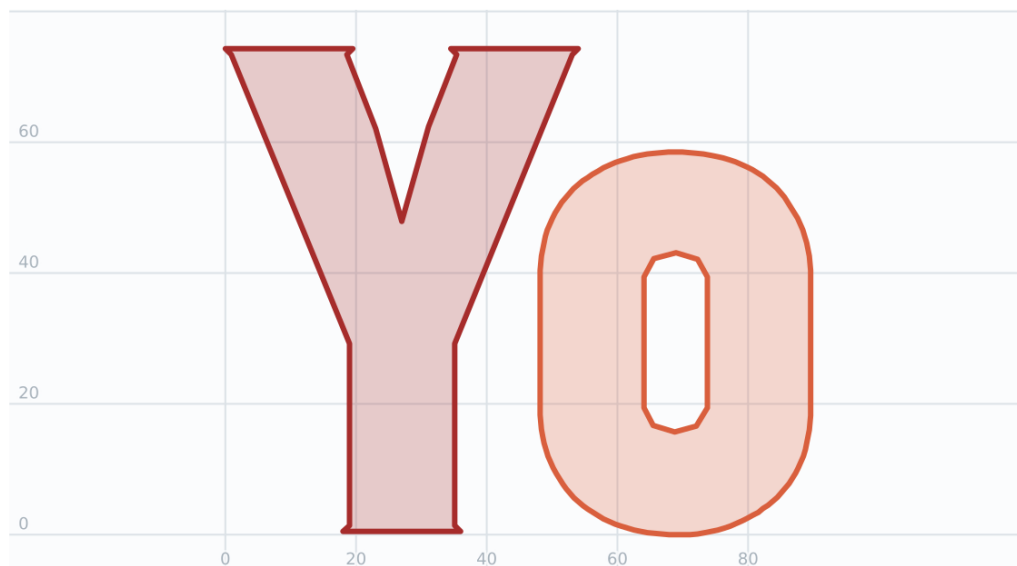
可用性：3.3.0

示例

Generating the word 'Yo'.

Code

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

Figure

■ Output

Generating the word 'PostGIS'.

Code

```
SELECT ST_Letters('PostGIS') AS letters;
```

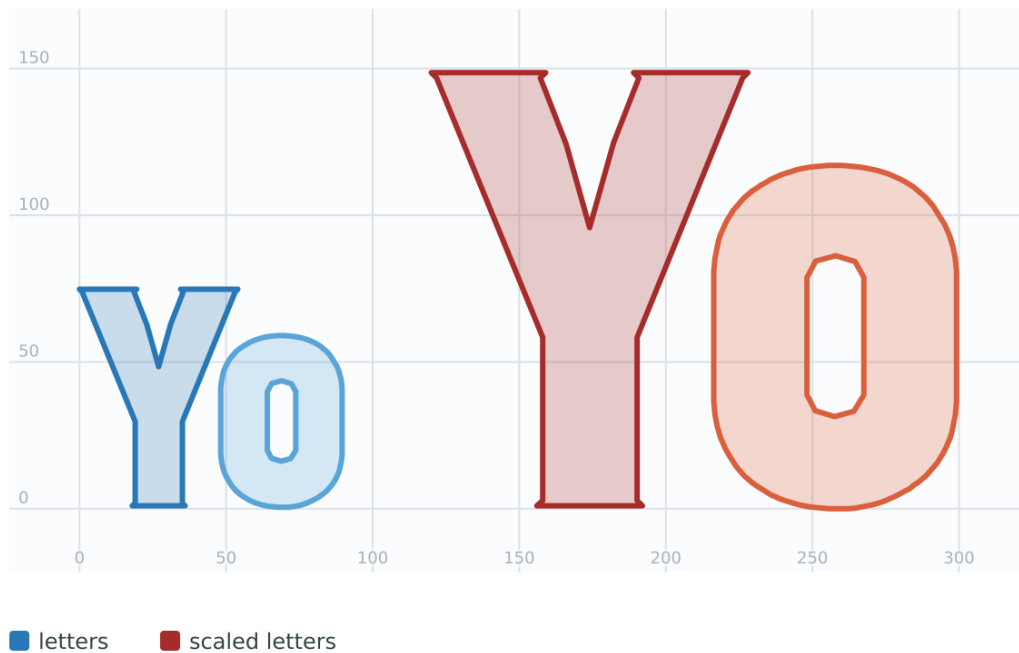
Figure

■ Output

Scaling and moving words.

Code

```
WITH letters AS (  
    SELECT ST_Letters('Yo') AS geom  
)  
SELECT geom AS input_letters,  
       ST_Translate(ST_Scale(geom, 2, 2), 120, 0)  
       AS scaled_letters  
FROM letters;
```

Figure

相关信息

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

7.3.19 ST_MakeNurbsCurve

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

Synopsis

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

描述

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

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

The `control_points` must be a `LINESTRING` geometry defining the control polygon. The number of control points must be at least `degree + 1`.

The optional `weights` array assigns a weight to each control point. All weights must be positive. If `NULL` or omitted, uniform weights (1.0) are used, resulting in a non-rational B-spline.

The optional `knots` array defines the parameter space. If `NULL` or omitted, a uniform clamped knot vector is automatically generated. If provided, the knot vector must have exactly `(npoints + degree + 1)` elements and must be non-decreasing.

Availability: 3.7.0

示例

This example creates a degree-2 NURBS curve with uniform weights.

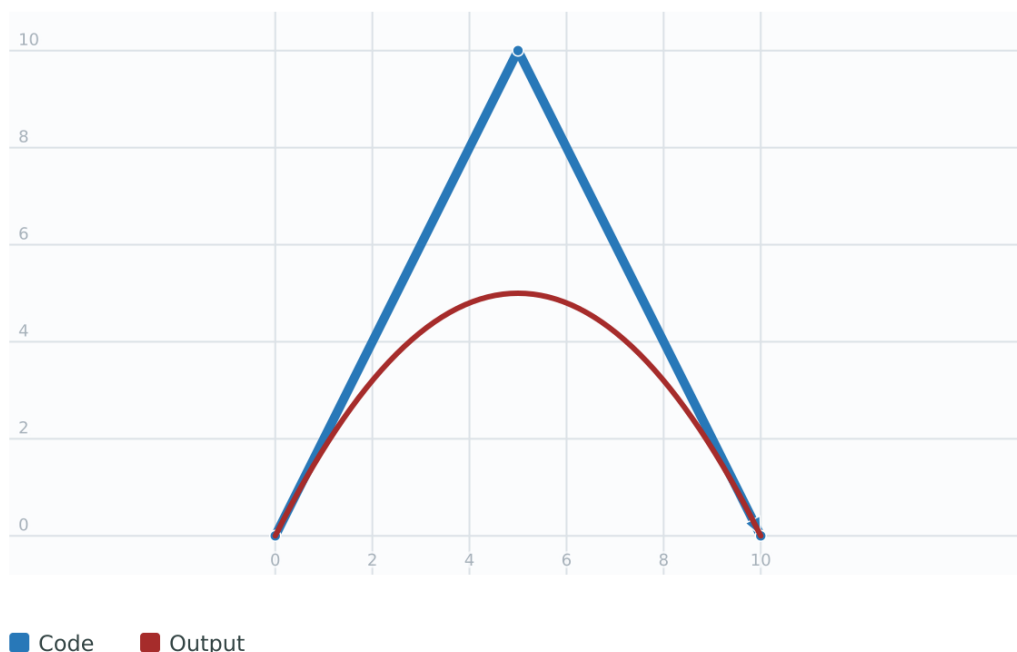
Code

```
SELECT ST_MakeNurbsCurve(  
    2,  
    'LINESTRING(0 0,5 10,10 0)>:::geometry  
);
```

Output

```
NURBS_CURVE(DEGREE 2, CONTROLPOINTS(NURBSPOINT(WEIGHTEDPOINT(0 0),WEIGHT 1), NURBSPOINT(WEIGHTEDPOINT(5 10),WEIGHT 1),  
NURBSPOINT(WEIGHTEDPOINT(10 0),WEIGHT 1)),KNOTS (KNOT(0,3), KNOT(1,3)))
```

Figure



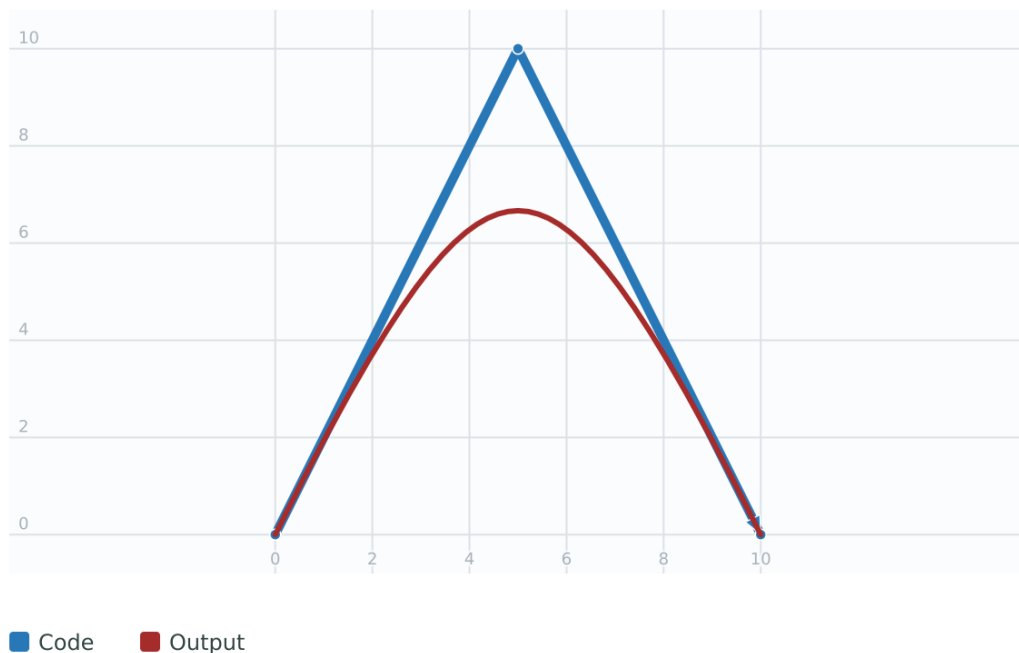
This example creates a rational NURBS curve with custom weights.

Code

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

Output

```
NURBS CURVE(DEGREE 2, CONTROLPOINTS(NURBSPPOINT(WEIGHTEDPOINT(0 0),WEIGHT 1), NURBSPPOINT(WEIGHTEDPOINT(5 10),WEIGHT 2),
NURBSPPOINT(WEIGHTEDPOINT(10 0),WEIGHT 1)),KNOTS (KNOT(0,3), KNOT(1,3)))
```

Figure

This example creates a NURBS curve with a non-uniform knot vector.

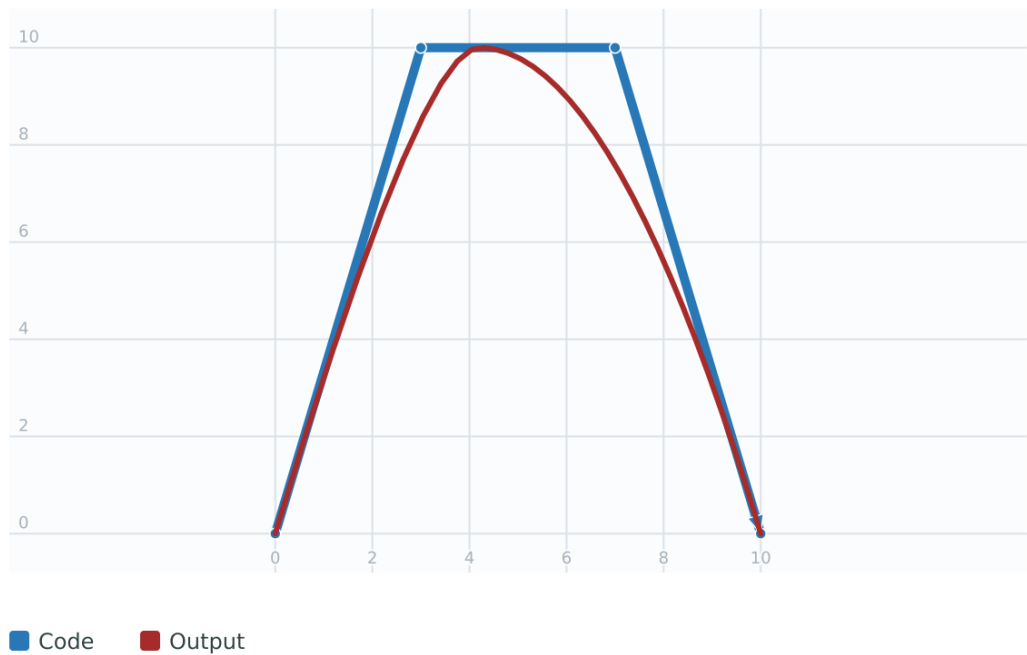
Code

```
SELECT ST_MakeNurbsCurve(
  2,
  'LINESTRING(0 0,3 10,7 10,10 0)::geometry,
  ARRAY[1.0, 1.0, 1.0, 1.0],
  ARRAY[0.0, 0.0, 0.0, 0.3, 1.0, 1.0, 1.0]
);
```

Output

```
NURBS CURVE(DEGREE 2, CONTROLPOINTS(NURBSPPOINT(WEIGHTEDPOINT(0 0),WEIGHT 1), NURBSPPOINT(WEIGHTEDPOINT(3 10),WEIGHT 1),
NURBSPPOINT(WEIGHTEDPOINT(7 10),WEIGHT 1), NURBSPPOINT(WEIGHTEDPOINT(10 0),WEIGHT 1)),KNOTS (KNOT(0,3), KNOT(0.3,1)
, KNOT(1,3)))
```

Figure



相关信息

[ST_NurbsToLineString](#), [ST_Evaluate](#), [ST_ControlPoints](#)

7.4 几何函数

7.4.1 GeometryType

GeometryType — 以文本形式返回几何的类型。

Synopsis

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

描述

以字符串形式返回几何类型，例如 “LINESTRING”、“POLYGON”、“MULTIPOINT” 等。

OGC SPEC s2.1.1.1 - 返回 Geometry 的可实例化子型的名称，Geometry 的可实例化子型的名称以字符串形式返回。



Note
该函数通常返回 “POINTM” 形式的字符串来指示是否具有 M 值。

增强功能：引入了 2.0.0 的多面体曲面、三角形和三角网的支持。

- ✓ 此方法符合了 SQL 1.1 的 OGC 功能规范。
- ✓ 此方法支持几何字符串和曲线。
- ✓ 该函数支持 3d 并且不会丢失 z-index。
- ✓ 该函数支持多面体曲面。
- ✓ 此函数支持三角形和不规则三角网面 (TIN)。

示例

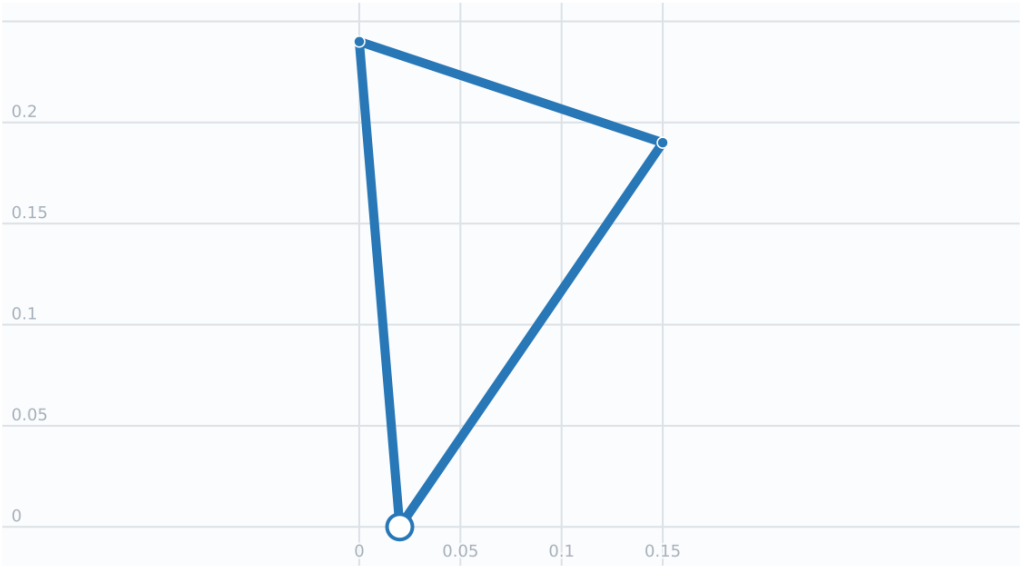
Code

```
SELECT GeometryType('LINESTRING(77.29 29.07,77.42 29.26,77.27 29.31,77.29 29.07)')::geometry ↵
);
```

Output

LINESTRING

Figure



Code

Code

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

Code

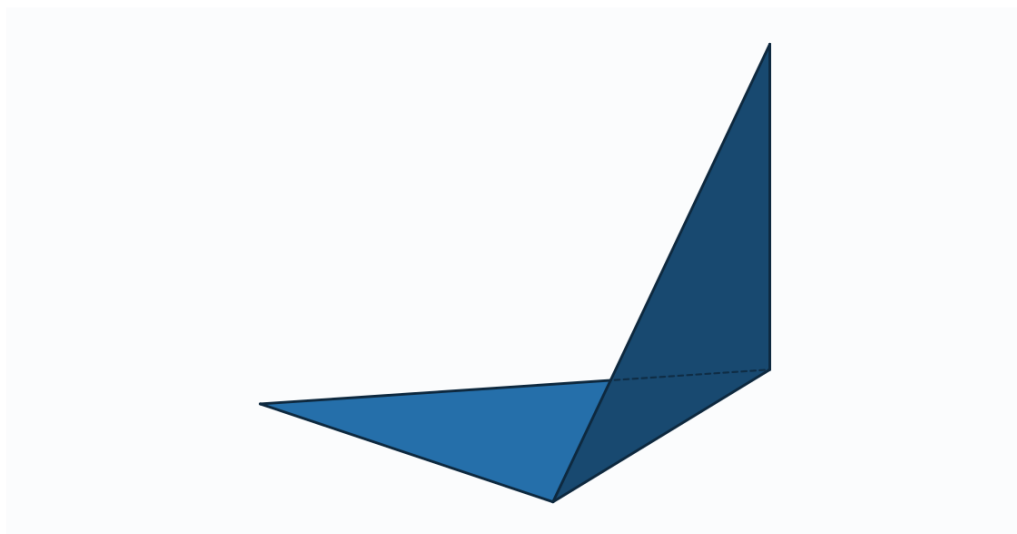
```
SELECT GeometryType(geom) as result
FROM
  (SELECT
    'TIN (((
      0 0 0,
      0 0 1,
      0 1 0,
      0 0 0
    )),(
      0 0 0,
      0 1 0,
      1 1 0,
      0 0 0
    ))
    )'::geometry AS geom
  ) AS g;
```

Output

```
TIN
```

Figure

3D view



■ geom

相关信息

[ST_GeometryType](#)

7.4.2 ST_Boundary

ST_Boundary — 返回几何图形的边界。

Synopsis

geometry **ST_Boundary**(geometry geomA);

描述

返回此 Geometry 的闭合边界的闭合。闭合边界的定义如 OGC SPEC 第 3.12.3.2 节中所述。由于该函数的结果是一个多边形，因此是拓扑封闭的，因此可以使用 OGC SPEC 第 3.12.2 节中定义的特征几何基元来表示生成的边界。它是通过 GEOS 模型实现的。



Note

在 2.0.0 之前，如果与 GEOMETRYCOLLECTION 一起使用，此函数会引发异常。从 2.0.0 开始，它将返回 NULL（不支持的输入）。



此方法实现了 SQL 1.1 的 OGC 多边形功能规范。OGC SPEC s2.1.1.1



该方法实现了 SQL/MM 规范。SQL-MM IEC 13249-3: 5.1.17



该函数支持 3d 并且不会丢失 z-index。

增强：引入了 2.1.0 三角函数支持

更改：3.2.0 支持 TIN，不使用地理，不线性化曲线

示例

LineString with boundary points overlaid.

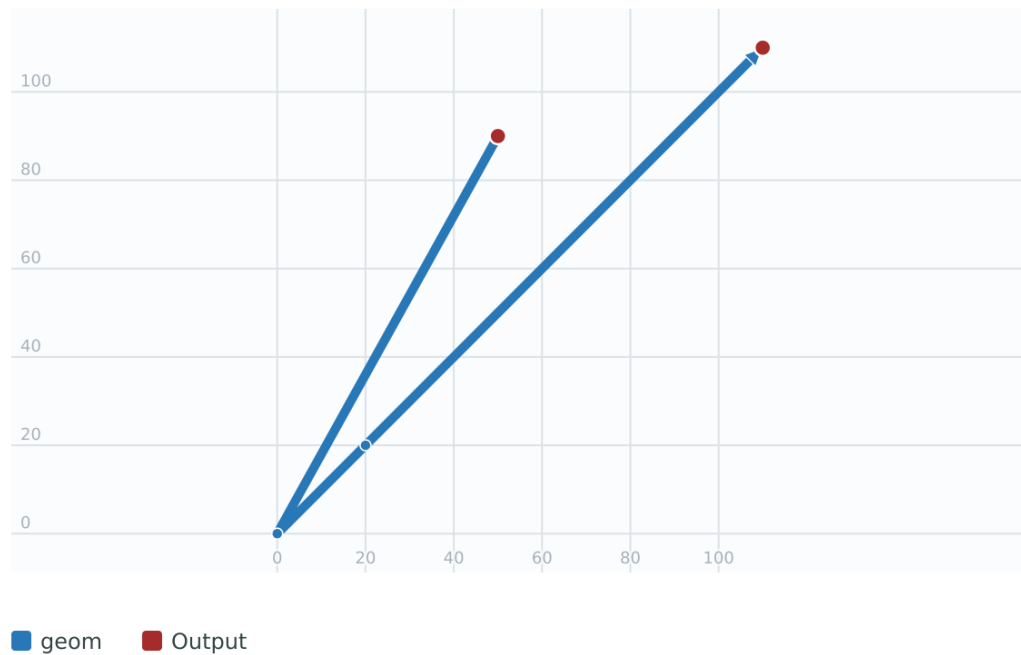
Code

```
SELECT ST_Boundary(geom)
FROM (SELECT 'LINESTRING(100 150,50 60,70 80,160 170)::geometry As geom) As f;
```

Output

```
MULTIPOINT((100 150),(160 170))
```


Figure



Polygon holes with the boundary MultiLineString overlaid.

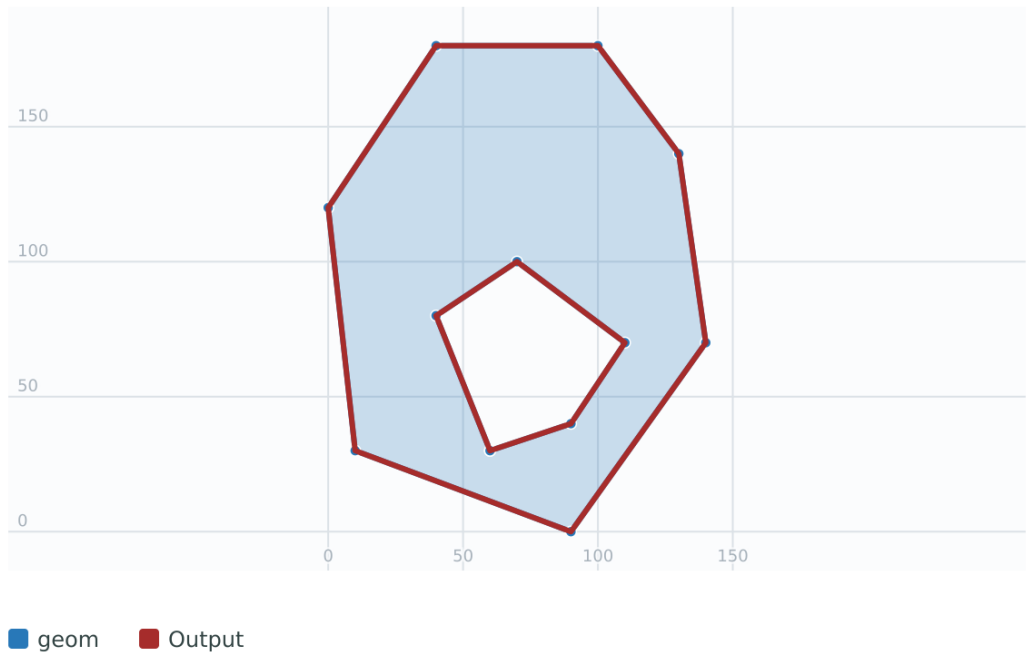
Code

```
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;
```

Output

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

Figure



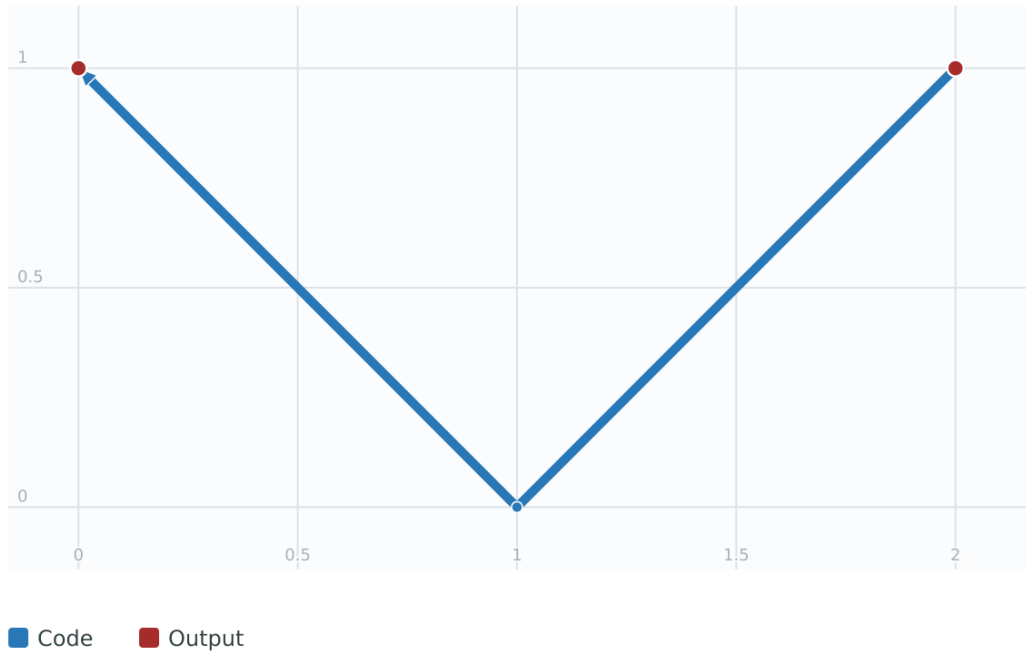
Code

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

Output

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

Figure

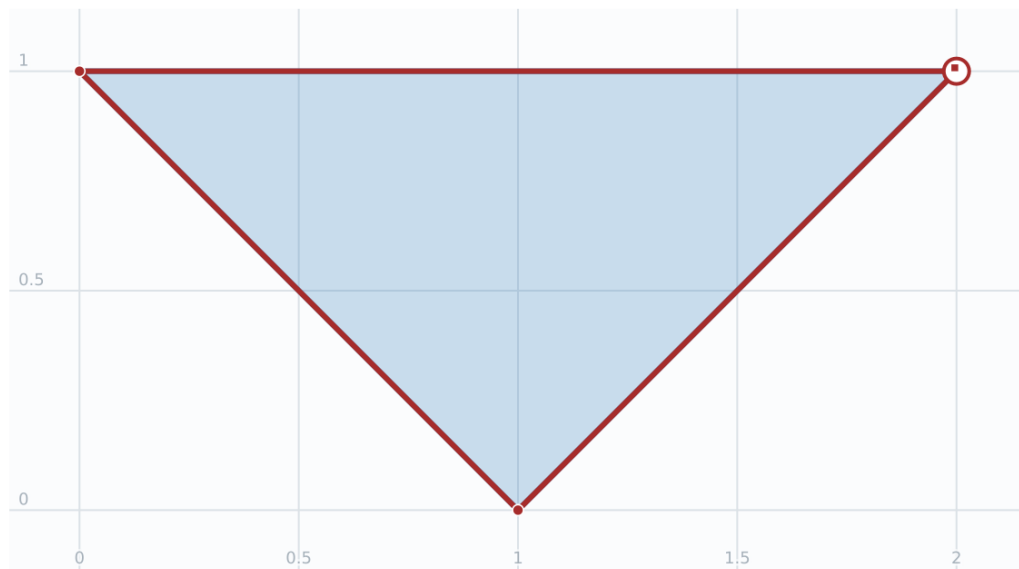


Code

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

Output

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

Figure

■ Code ■ Output

This example uses a 3D polygon.

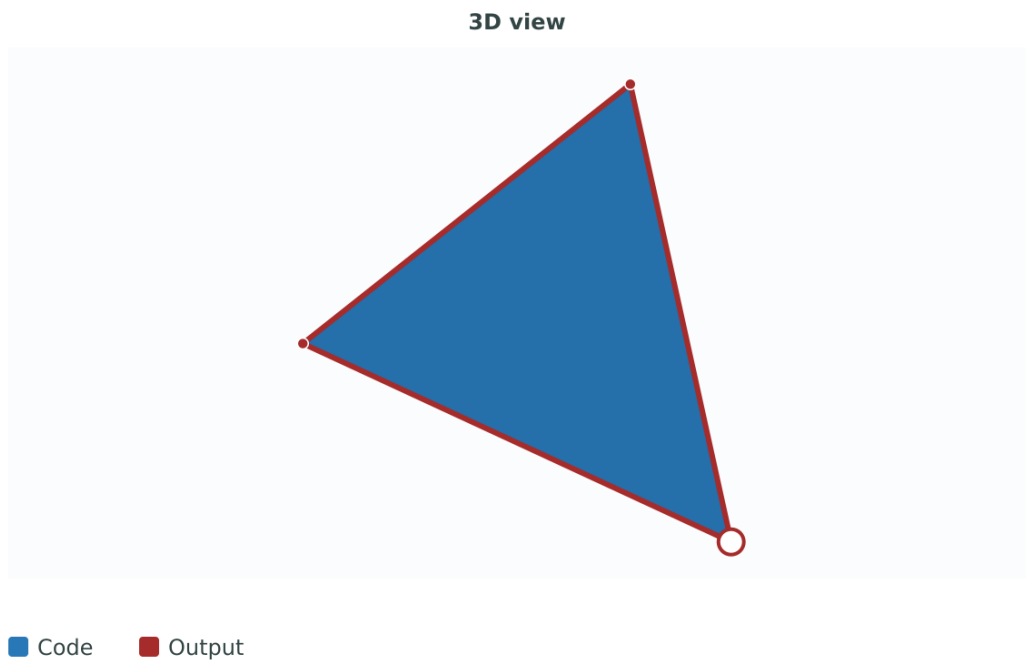
Code

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

Output

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

Figure



This example uses a 3D MultiLineString.

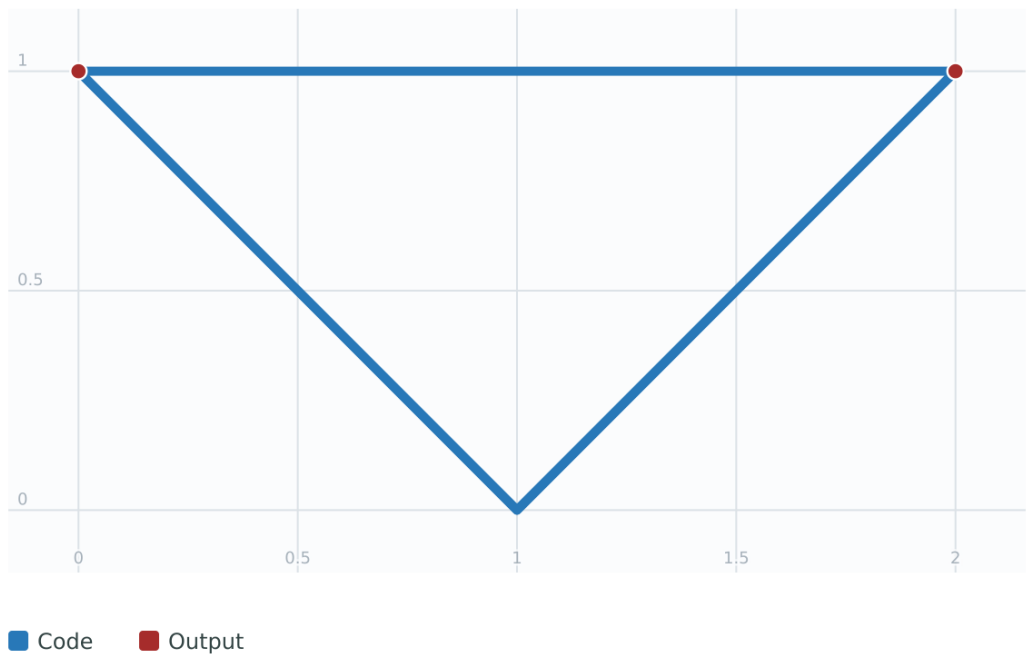
Code

```
SELECT ST_Boundary('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) )');
```

Output

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

Figure



相关信息

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

7.4.3 ST_BoundingDiagonal

ST_BoundingDiagonal — 返回几何图形的边界框的对角线。

Synopsis

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

描述

以字符串形式返回给定几何图形的边界框的对角线。它是一个由点组成的字符串，从最小点开始，到最大点结束。如果输入几何图形为空，则对角线为空。

可选的 `fits` 参数指定是否需要最佳拟合。如果 `false`，则可以接受稍大的边界框的对角线（对于具有多个点的几何图形，计算速度更快）。无论哪种情况，返回的对角线的边界框始终覆盖输入几何体。

返回的几何体保留输入几何体的 SRID 和维度（Z 和 M 存在）。



Note

在退化情况下（即输入中的单个点），返回的字符串将在形式上无效（无内部）。如果在拓扑上仍然有效。

可用性：2.2.0

✓ 函数支持 3d 并且不会丢失 z-index。

✓ 功能支持 M 坐标。

示例

This example shows a polygon together with the diagonal of its bounding box.

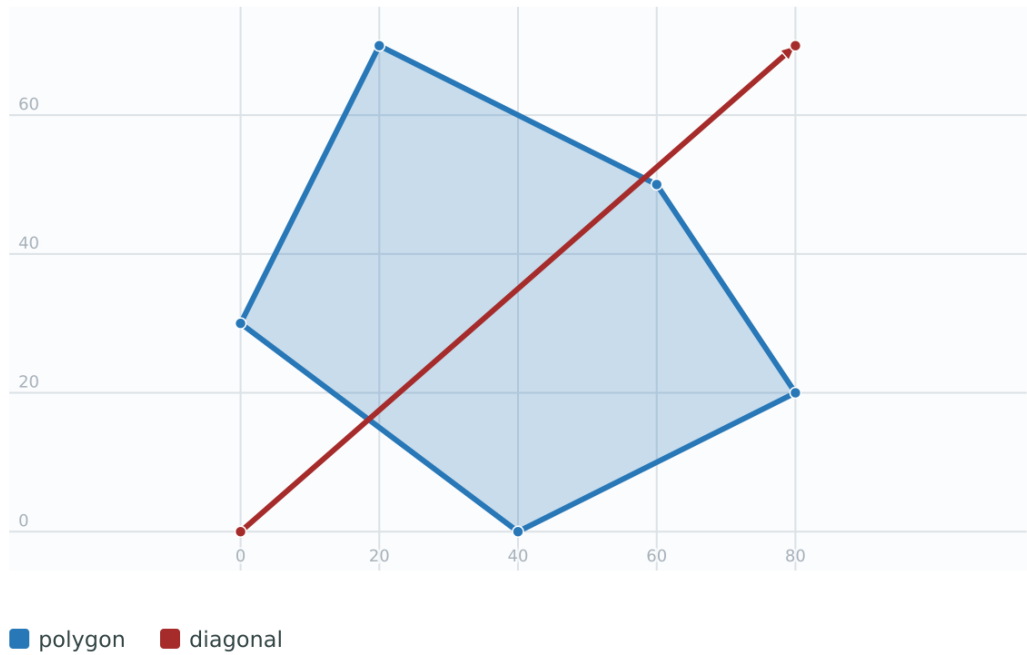
Code

```
WITH input AS (  
  SELECT 'POLYGON((10 20,30 60,70 40,90 10,50 -10,10 20))'::geometry AS geom  
)  
SELECT geom AS input_polygon,  
       ST_BoundingDiagonal(geom) AS diagonal  
FROM input;
```

Output

```
POLYGON((10 20,30 60,70 40,90 10,50 -10,10 20)) | LINESTRING(10 -10,90 60)
```

Figure



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

Code

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

Output

-10

相关信息

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

7.4.4 ST_CoordDim

ST_CoordDim — 返回几何体的坐标维度。

Synopsis

integer **ST_CoordDim**(geometry geomA);

描述

返回 ST_Geometry 的坐标维度。

是 [ST_NDims](#) 的 MM 兼容名

- ✔ 此方法符合了 SQL 1.1 的 OGC 功能规范。
- ✔ 此方法符合了 SQL/MM 规范。SQL-MM 3: 5.1.3
- ✔ 此方法支持弧形字符串和曲线。
- ✔ 此函数支持 3d 并且不会丢失 z-index。
- ✔ 此函数支持多面体曲面。
- ✔ 此函数支持三角形和不规则三角网面 (TIN)。

示例

Code

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

Output

```
3
```

Code

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

Output

```
2
```

相关信息

[ST_NDims](#)

7.4.5 ST_Dimension

ST_Dimension — 返回几何图形的拓扑维数。

Synopsis

integer **ST_Dimension**(geometry g);

描述

返回此 Geometry 对象的拓扑维数，维数必小于或等于坐维度。OGC SPEC s2.1.1.1 - 对于 POINT 返回 0, 对于 LINESTRING 返回 1, 对于 POLYGON 返回 2, 以及 GEOMETRYCOLLECTION 物件的最大尺寸。如果尺寸未知 (例如, 对于空 GEOMETRYCOLLECTION), 返回 0。

- ✔ 此方法符合了 SQL/MM 规范。SQL-MM 3: 5.1.2



Note

A TIN, including a closed TIN surface, is a surface geometry and has dimension 2. Use **CG_MakeSolid** to make the solid representation explicit; the resulting 3D solid has dimension 3. Solid geometry semantics, such as volume, are handled separately by the SQL/MM 3D solid functions (SQL-MM IEC 13249-3: 9.1).

增⏏：2.0.0 引入了多面体曲面支持和 TIN 支持。当⏏定空几何⏏，它不再引⏏异常。



Note

在 2.0.0 之前，提供空几何⏏形会引⏏异常。



⏏函数支持多面体曲面。



此函数支持三角形和不⏏⏏三角网面 (TIN)。

示例

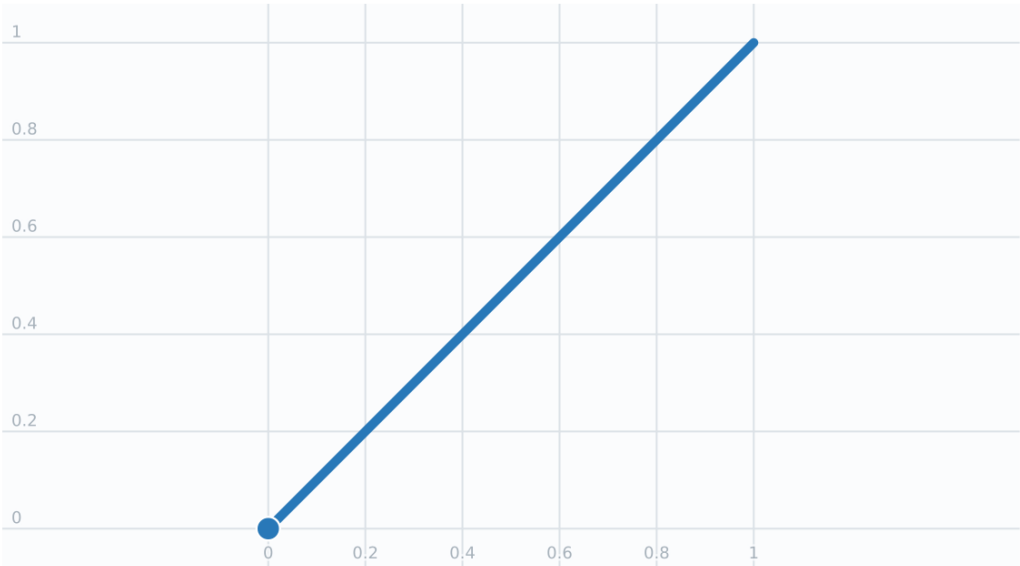
Code

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

Output

1

Figure



■ Code

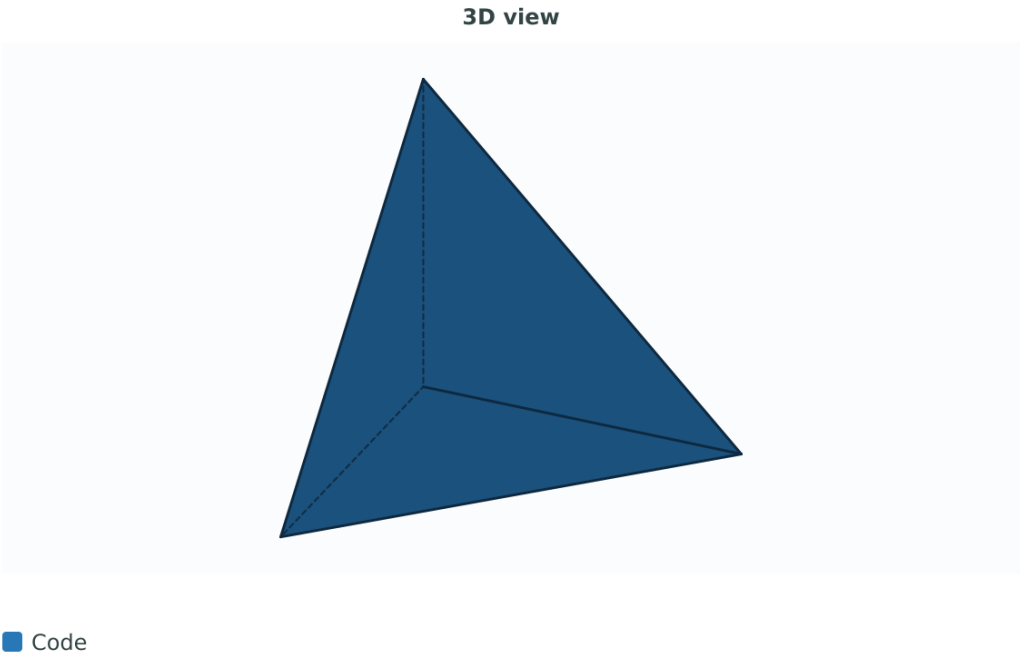
Code

```
SELECT ST_Dimension('TIN Z (((0 0 0,0 0 1,0 1 0,0 0 0)),((0 0 0,0 1 0,1 0 0,0 0 0)),((0 0 0,1 0 0,0 0 1,0 0 0)),((1 0 0,0 1 0,0 0 1,1 0 0)))');
```

Output

2

Figure



相关信息

[ST_NDims](#), [CG_MakeSolid](#), [CG_Volume](#)

7.4.6 ST_Dump

ST_Dump — 返回几何图件的一行 geometry_dump 行。

Synopsis

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

描述

提取几何图件的集合返回函数 (SRF)。它返回一行 **geometry_dump** 行，每行包含一个几何图形 (*geom* 字段) 和一个整数数字 (*path* 字段)。

对于基本几何图型 (POINT、LINESTRING、POLYGON)，返回一个图，其中包含空 *path* 数字，图入几何图形 *geom*。对于集合或多几何体，将每个集合图件返回一条图，并且路径表示图件在集合内的位置。

`ST_Dump` 用于展示几何体很有用。它与 `ST_Collect / GROUP BY` 相反，它新建行。例如，它可用于将 MULTIPOLYGONS 展示为 POLYGONS。

增强功能：引入了 2.0.0 的多面体曲面、三角形和三角网的支持。

可用性：PostGIS 1.0.0RC1。需要 PostgreSQL 7.3 或更高版本。



Note

在 1.3.4 之前，此函数在与包含曲线的几何体形一起使用时崩溃。此问题已在 1.3.4 及更高版本中得到修正



此方法支持体形字符串和曲线。



函数支持多面体曲面。



此函数支持三角形和不规则三角网面 (TIN)。



函数支持 3d 并且不会丢失 z-index。

示例

Dump the components of geometries stored in a table.

Code

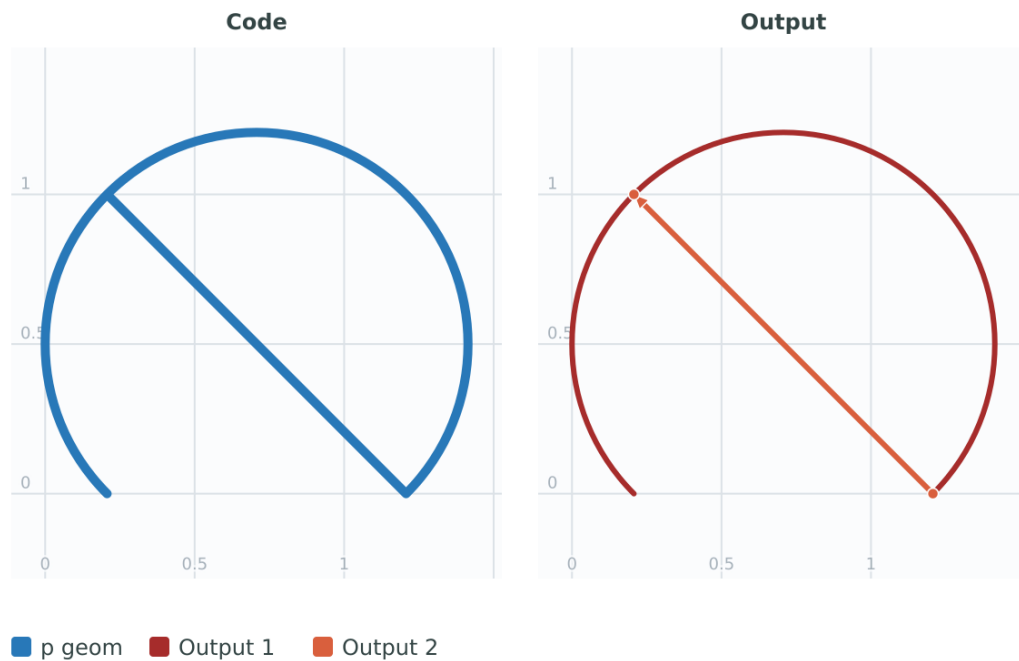
```
SELECT sometable.field1,
       (ST_Dump(sometable.geom)).geom AS geom
FROM sometable;
```

Break a CompoundCurve into its constituent LineStrings and CircularStrings.

Code

```
SELECT a.geom, ST_HasArc(a.geom)
FROM ( SELECT (ST_Dump(p_geom)).geom AS geom
      FROM (SELECT 'COMPOUNDCURVE(CIRCULARSTRING(0 0,1 1,1 0),(1 0,0 1))'::geometry AS
      p_geom) AS b
      ) AS a;
```

Figure

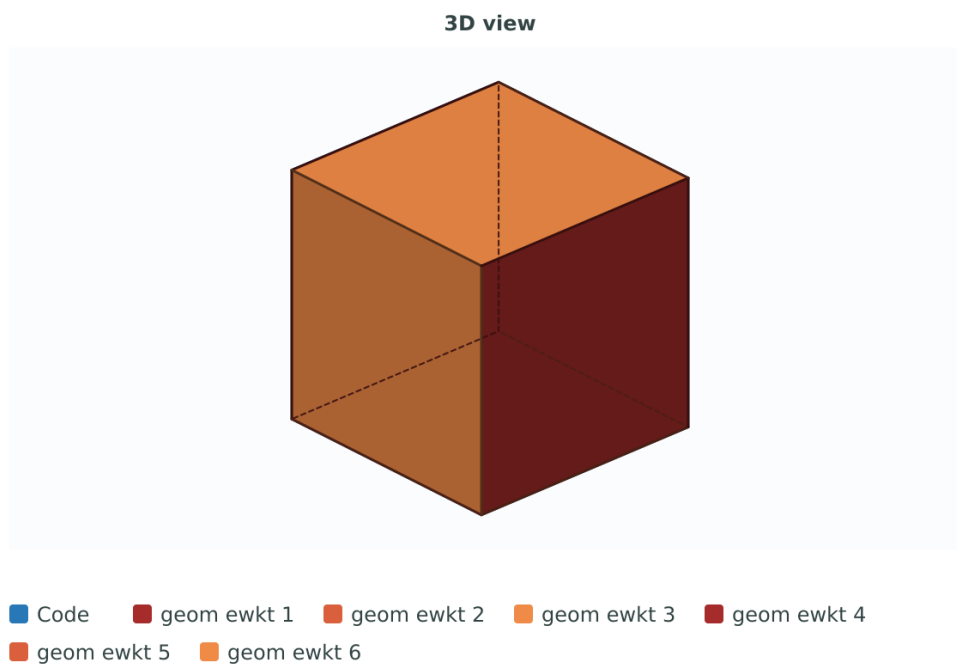


This example breaks a PolyhedralSurface into its faces.

Code

```
SELECT (a.p_geom).path[1] As path,(a.p_geom).geom As geom_ewkt
FROM (SELECT ST_Dump('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;
```

Figure

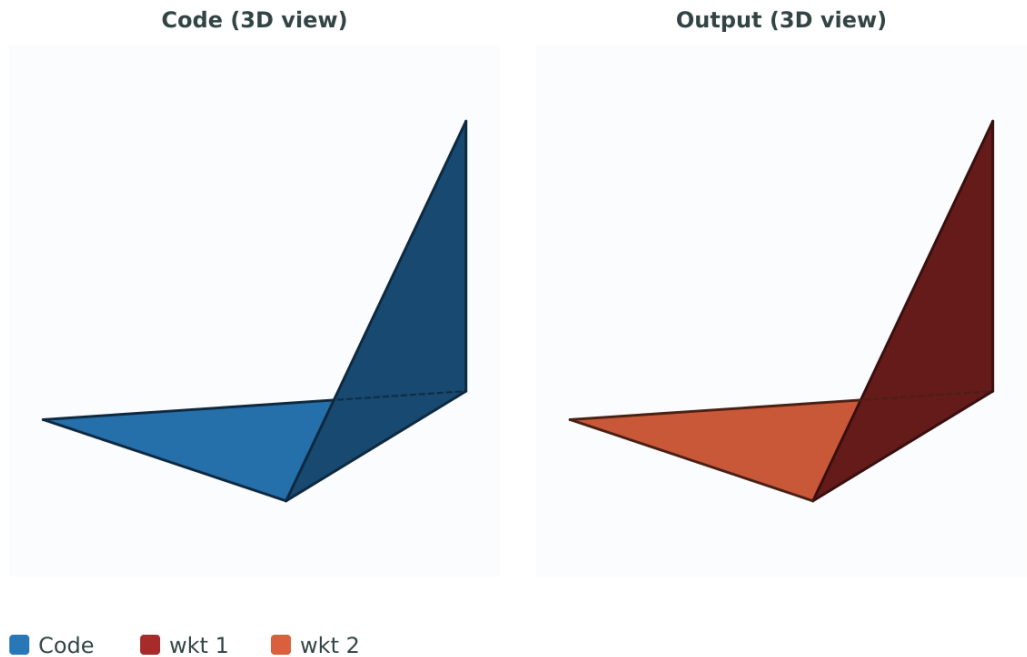


This example uses a TIN geometry.

Code

```
SELECT (g.gdump).path,(g.gdump).geom as wkt
FROM
  (SELECT
    ST_Dump('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;
```

Figure



相关信息

[geometry_dump](#), [Section 13.6](#), [ST_Collect](#), [ST_GeometryN](#)

7.4.7 ST_DumpPoints

ST_DumpPoints — 返回几何体中坐标的一行 geometry_dump 行。

Synopsis

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

描述

提取几何体坐标（点）的集合返回函数 (SRF)。它返回一行 geometry_dump 行，每行包含一个几何体（geom 字段）和一个整数编号（path 字段）。

- geom 字段 POINT 表示所提供几何体的坐标。
- path 字段 (integer[]) 是坐标所提供几何体元素中的坐标位置的索引。索引是从 1 开始的。例如，对于 LINESTRING，路径 {i}，其中 i 是 LINESTRING 中的第 n 个坐标。对于 POLYGON，路径 {i,j}，其中 i 是环号（1 是外环；后面是内环），j 是环中的坐标位置。

要取包含坐标的几何体，使用 ST_Points。

增强：2.1.0 速度更快。重新编译原生 C 语言。

增强功能：引入了 2.0.0 多面体曲面、三角形和三角网的支持。

可用性：1.1.0

- ✔ 此方法支持多边形字符串和曲线。
- ✔ 函数支持多面体曲面。
- ✔ 此函数支持三角形和不规则三角网面 (TIN)。
- ✔ 函数支持 3d 并且不会丢失 z-index。

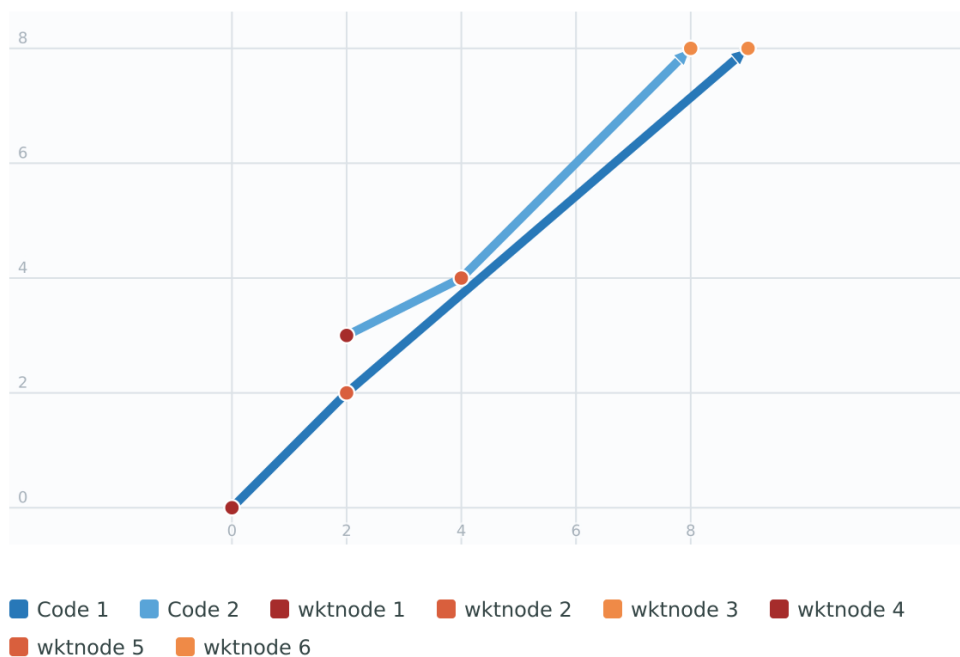
示例

Explode a table of LineStrings into their vertices.

Code

```
SELECT edge_id, (dp).path[1] As index, (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;
```

Figure



Dump every vertex from a mixed GeometryCollection.

Code

```
SELECT path, 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;
```

Output

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)

Collect the dumped vertices to visualize them over the source geometry.

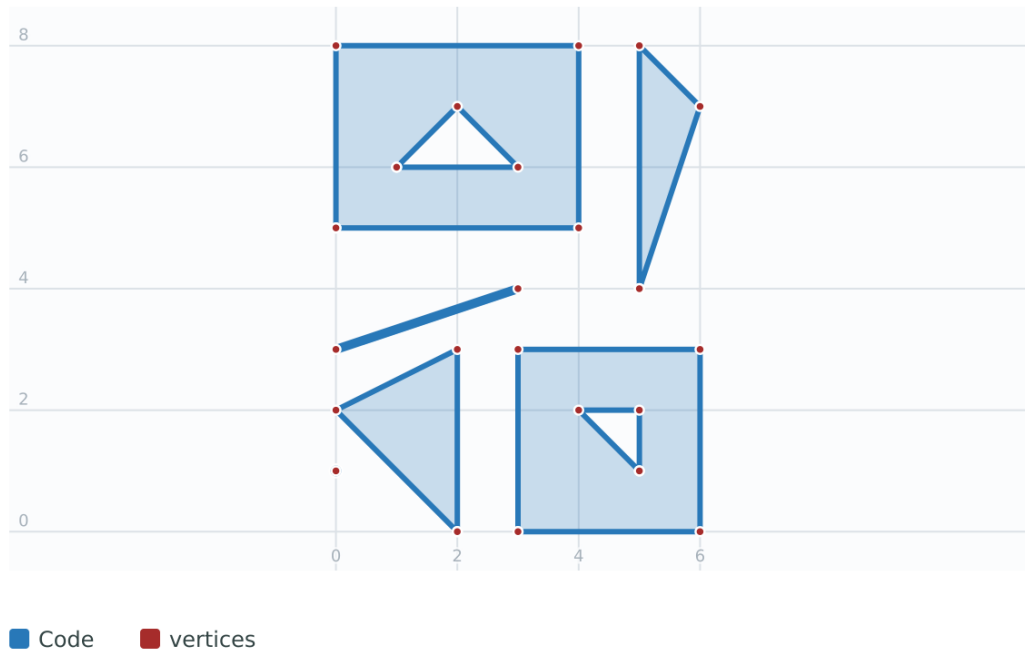
Code

```
WITH source(geom) AS (VALUES (
    '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
    ))
SELECT ST_Collect((dp).geom ORDER BY (dp).path) AS vertices
FROM source
CROSS JOIN LATERAL ST_DumpPoints(source.geom) AS dp;
```

Output

```
MULTIPOINT((0 1),(0 3),(3 4),(2 0),(2 3),(0 2),(2 0),(3 0),(3 3),(6 3),(6 0),(3 0),(5 1),(4 2),(5 2),(5 1),(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))
```

Figure



This example uses a cube represented as a PolyhedralSurface.

Code

```
SELECT (g.gdump).path,(g.gdump).geom as wkt
FROM
  (SELECT
    ST_DumpPoints('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 0 0,0 1 0)),((0 0 1,1 0 1,1 1 1,0 1 1,0 0 1)) )' ) AS gdump
  ) AS g;
```

Output

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)

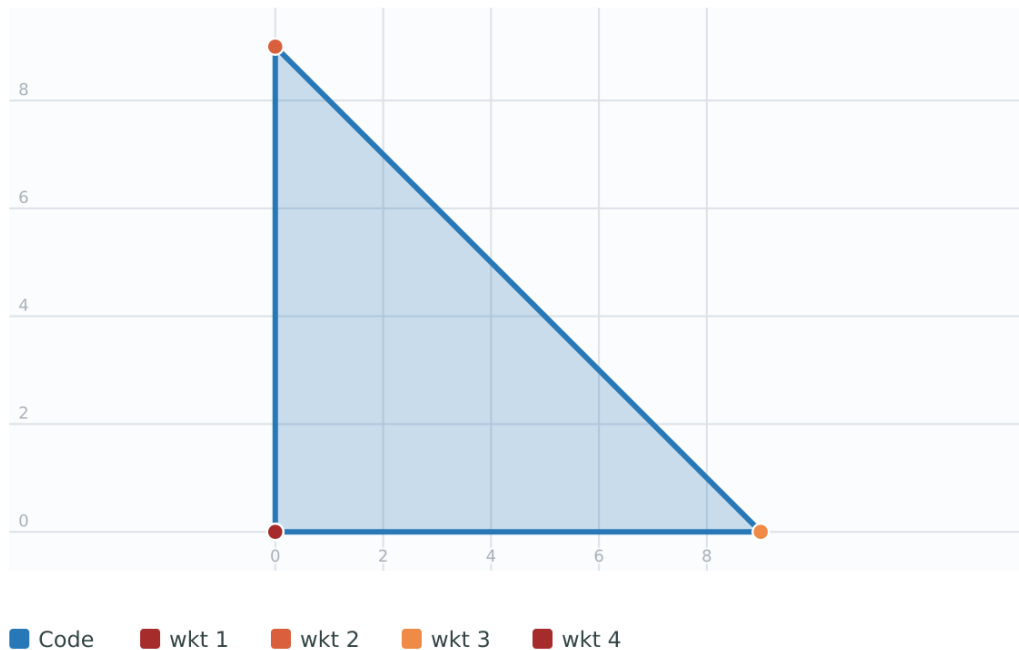

```
{6,1,3} | POINT(1 1)
{6,1,4} | POINT(0 1)
{6,1,5} | POINT(0 0)
(30 rows)
```

This example uses a Triangle geometry.

Code

```
SELECT (g.gdump).path,(g.gdump).geom as wkt
FROM
  (SELECT
    ST_DumpPoints('TRIANGLE ((
      0 0,
      0 9,
      9 0,
      0 0
    ))' ) AS gdump
  ) AS g;
```

Figure



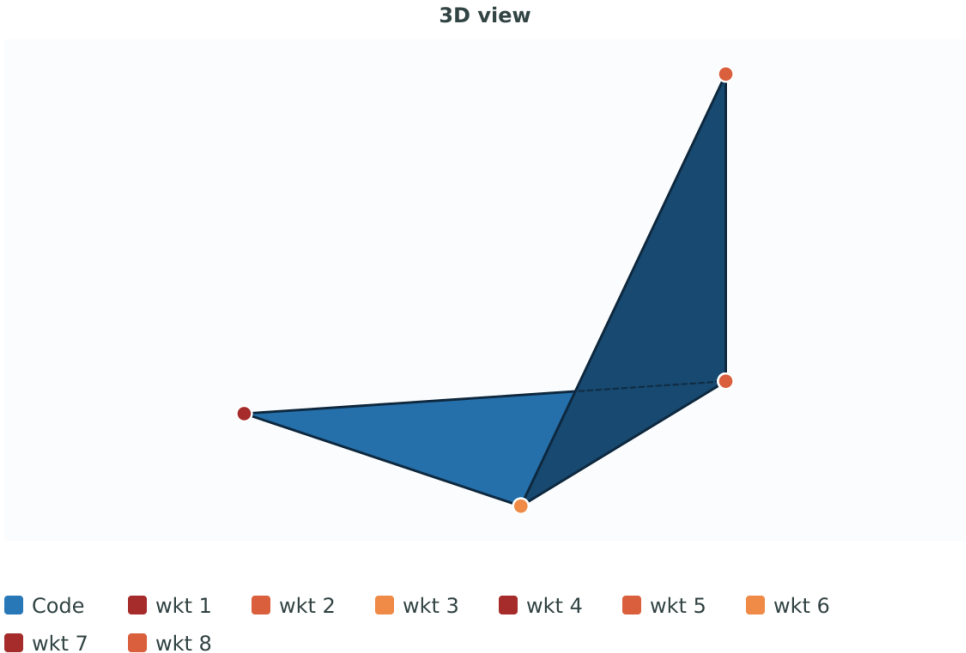
This example uses a TIN geometry.

Code

```
SELECT (g.gdump).path,(g.gdump).geom as wkt
FROM
  (SELECT
    ST_DumpPoints('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;
```

```
    ))
  )' ) AS gdump
) AS g;
```

Figure



相关信息

[geometry_dump](#), [Section 13.6](#), [ST_Dump](#), [ST_DumpRings](#), [ST_Points](#)

7.4.8 ST_DumpSegments

ST_DumpSegments — 将几何图形中的各个段返回一行 geometry_dump 行。

Synopsis

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

描述

提取几何图形的集合返回函数 (SRF)。它返回一行 geometry_dump 行，每行包含一个几何图形 (geom 字段) 和一个整数数 (path 字段)。

- geom 字段中的 LINESTRING 表示定几何体的段，而 CIRCULARSTRING 表示弧段。
- path 字段 (integer[]) 是枚所提供的几何元素中的段起点位置的索引。索引是从 1 开始的。例如，对于 LINESTRING，路径 {i}，其中 i 是 LINESTRING 中的第 n 个段起点。对于 POLYGON，路径 {i,j}，其中 i 是环号 (1 是外环；后面是内环)，j 是环中的段起点位置。

可用性 : 3.2.0

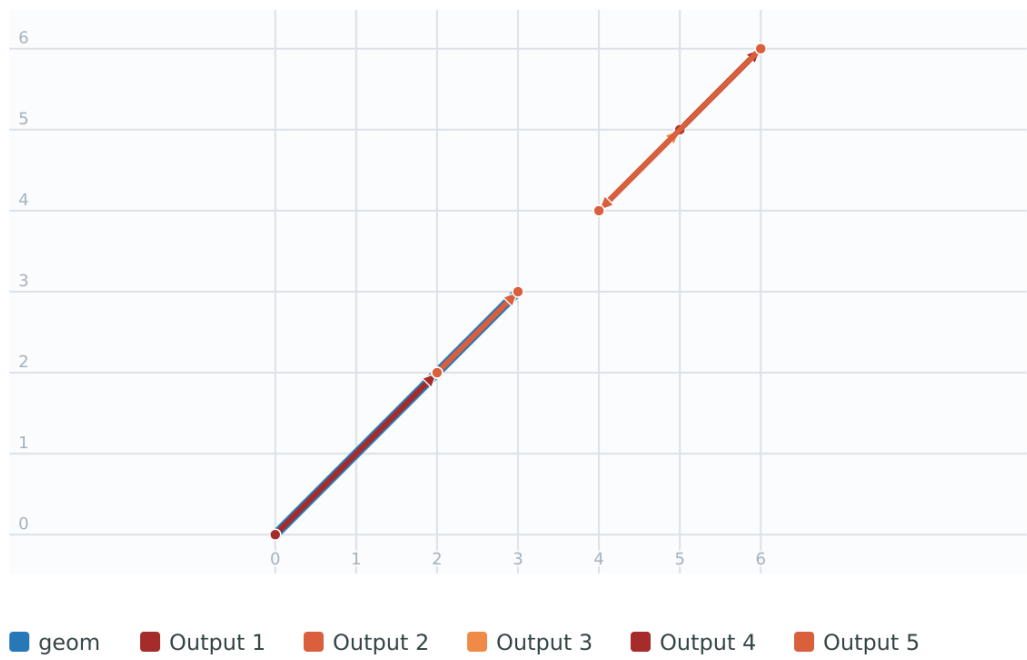
- ✓ 此函数支持三角形和不规则三角网面 (TIN)。
- ✓ 函数支持 3d 并且不会丢失 z-index。

几何示例

Code

```
SELECT path,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;
```

Figure



TIN 和三角形示例

This example uses a Triangle geometry.

Code

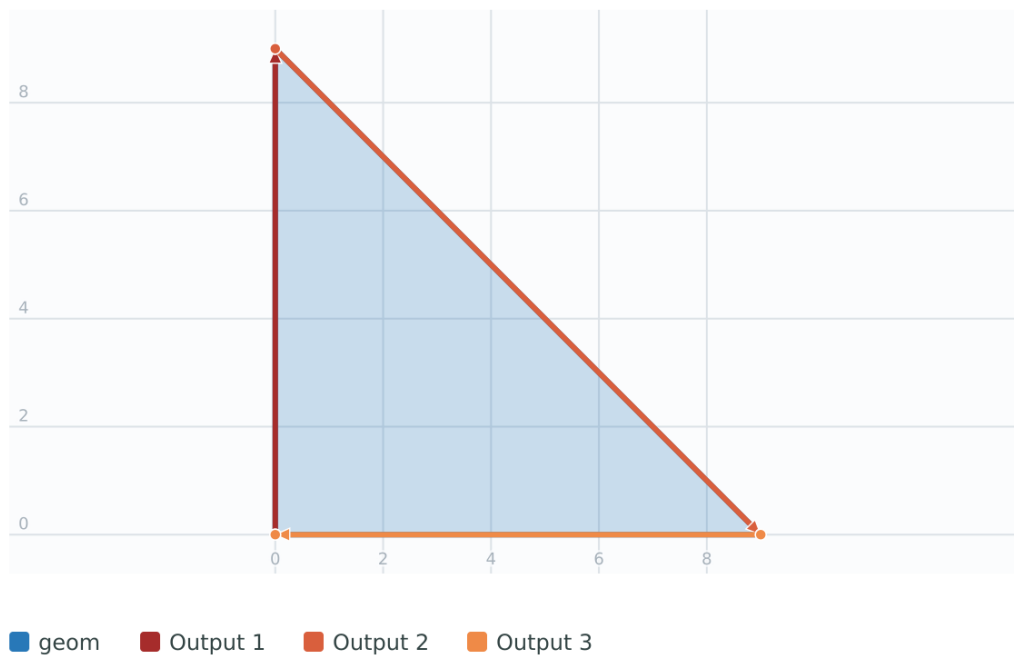
```
SELECT path,geom
FROM (
  SELECT (ST_DumpSegments(g.geom)).*
  FROM (SELECT 'TRIANGLE((
    0 0,
    0 9,
```

```

    9 0,
    0 0
  ))'::geometry AS geom
  ) AS g
) j;

```

Figure



This example uses a TIN geometry.

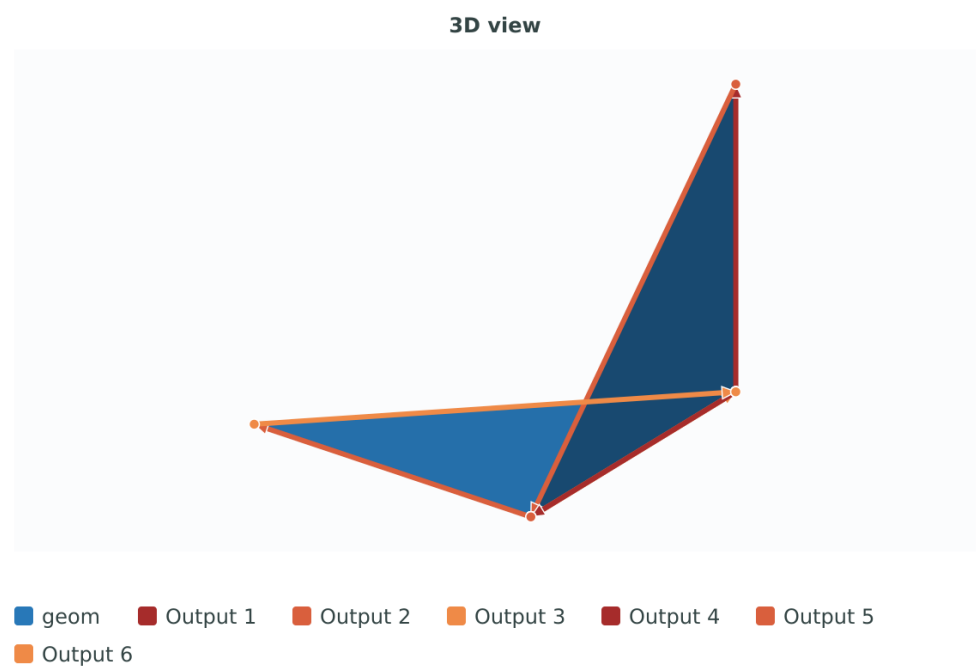
Code

```

SELECT path,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;

```

Figure



相关信息

[geometry_dump](#), [Section 13.6](#), [ST_Dump](#), [ST_DumpRings](#)

7.4.9 ST_DumpRings

`ST_DumpRings` — 返回多边形外环和内环的一行 `geometry_dump` 行。

Synopsis

```
geometry_dump[] ST_DumpRings(geometry a_polygon);
```

描述

提取多边形集合返回函数 (SRF)。它返回一行 `geometry_dump` 行，每行包含一个几何形状 (`geom` 字段) 和一个整数数 (`path` 字段)。

`geom` 字段将每个包含 POLYGON。`path` 字段是一个度 1 的整数数，包含多边形索引。外环 (壳) 的索引 0。内环 (孔) 的索引 1 及更高。



Note

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

可用性：需要 PostGIS 1.1.3 PostgreSQL 7.3 或更高版本。



函数支持 3d 并且不会失 z-index。

示例

将所有 $\square$ 提取 $\square$ polygon。

Code

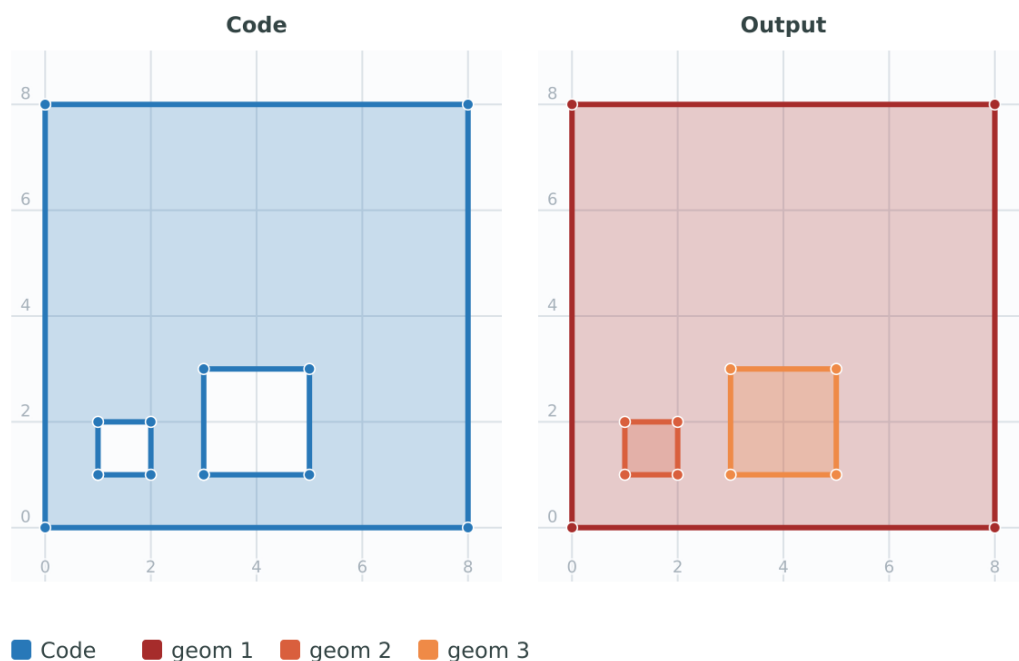
```
SELECT polyTable.field1, polyTable.field1,
       (ST_DumpRings(polyTable.geom)).geom As geom
FROM polyTable
```

从 polygon 中提取外壳 (shell) 和洞 (holes)。

Code

```
SELECT path,geom As geom
FROM ST_DumpRings('POLYGON ((1 9,9 9,9 1,1 1,1 9),(2 2,2 3,3 3,3 2,2 2),(4 2,4 4,6 4,6 2,4 2))');
↔
```

Figure



Finding interior rings which lie close to the exterior ring.

Code

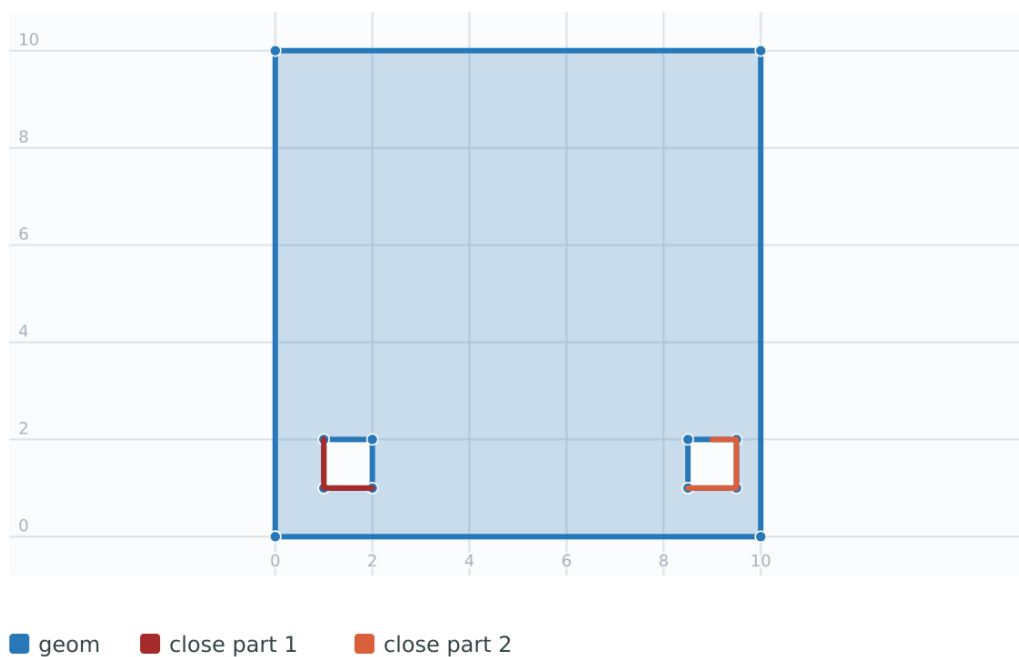
```
WITH poly AS (
  SELECT 'POLYGON ((
    0 0, 10 0, 10 10, 0 10, 0 0
  ), (
    1 1, 1 2, 2 2, 2 1, 1 1
  ), (
    8.5 1, 8.5 2, 9.5 2, 9.5 1, 8.5 1
  ))'::geometry AS geom
),
rings AS (
  SELECT d.path[1] AS ring_no,
         ST_Boundary(d.geom) AS ring_geom
  FROM poly
  CROSS JOIN LATERAL ST_DumpRings(poly.geom) AS d
```

```

),
shell AS (
  SELECT ring_geom
  FROM rings
  WHERE ring_no = 0
),
holes AS (
  SELECT ring_no, ring_geom
  FROM rings
  WHERE ring_no > 0
)
SELECT h.ring_no,
       ST_Normalize(ST_Multi(ST_LineMerge(
         ST_Intersection(ST_Buffer(s.ring_geom, 1.0), h.ring_geom)
       ))) AS close_part
FROM shell AS s
JOIN holes AS h
  ON ST_DWithin(s.ring_geom, h.ring_geom, 1.0);

```

Figure



相关信息

[geometry_dump](#), [Section 13.6](#), [ST_Boundary](#), [ST_Buffer](#), [ST_Dump](#), [ST_DWithin](#), [ST_ExteriorRing](#), [ST_Interi](#), [ST_Intersection](#)

7.4.10 ST_EndPoint

ST_EndPoint — Returns the last point of a LineString, CircularLineString, or NURBSCurve.

Synopsis

geometry **ST_EndPoint**(geometry g);

描述

Returns the last point of a **LINESTRING**, **CIRCULARLINESTRING**, or **NURBSCURVE** geometry as a **POINT**. Returns **NULL** if the input is not a **LINESTRING**, **CIRCULARLINESTRING**, or **NURBSCURVE**.

✓ 方法符合了 SQL/MM 规范。SQL-MM 3: 7.1.4

✓ 函数支持 3d 并且不会丢失 z-index。

✓ 此方法支持几何字符串和曲线。

更改：2.0.0 不再适用于几何体 **MultiLineStrings**。在旧版本的 PostGIS 中，几何体 **MultiLineString** 可以使用此函数并返回点。在 2.0.0 中，它像任何其他 **MultiLineString** 一样返回 **NULL**。旧的行为是一个未定义的功能，但是那些假设将数据存入 **LINESTRING** 的人可能会在 2.0.0 中遇到一些返回 **NULL** 的情况。

示例

字符串 (LineString) 端点

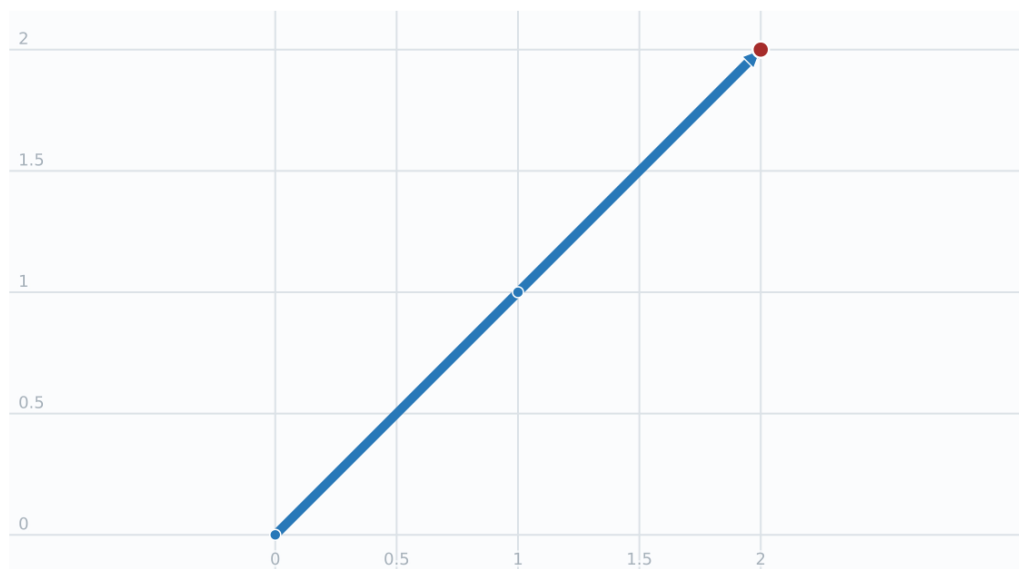
Code

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

Output

```
POINT(3 3)
```

Figure



■ Code ■ Output

非字符串止点 NULL

Code

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

Output

```
t
```

3D 线串 (LineString) 端点

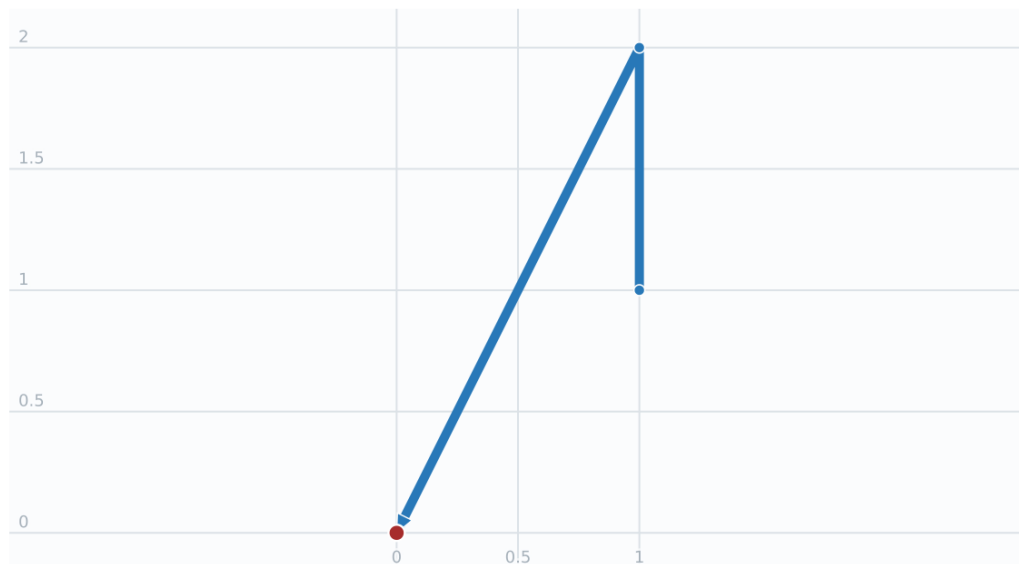
3D endpoint.

Code

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

Output

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

Figure

■ Code ■ Output

弧的端点

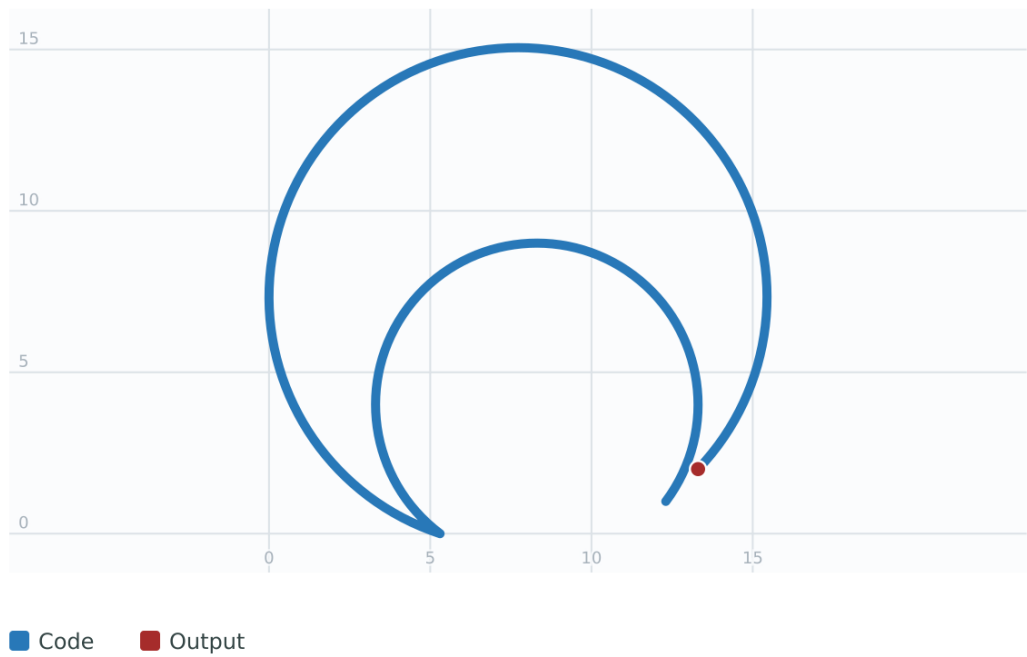
Code

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

Output

```
POINT(6 3)
```

Figure



End point of a NURBSCurve

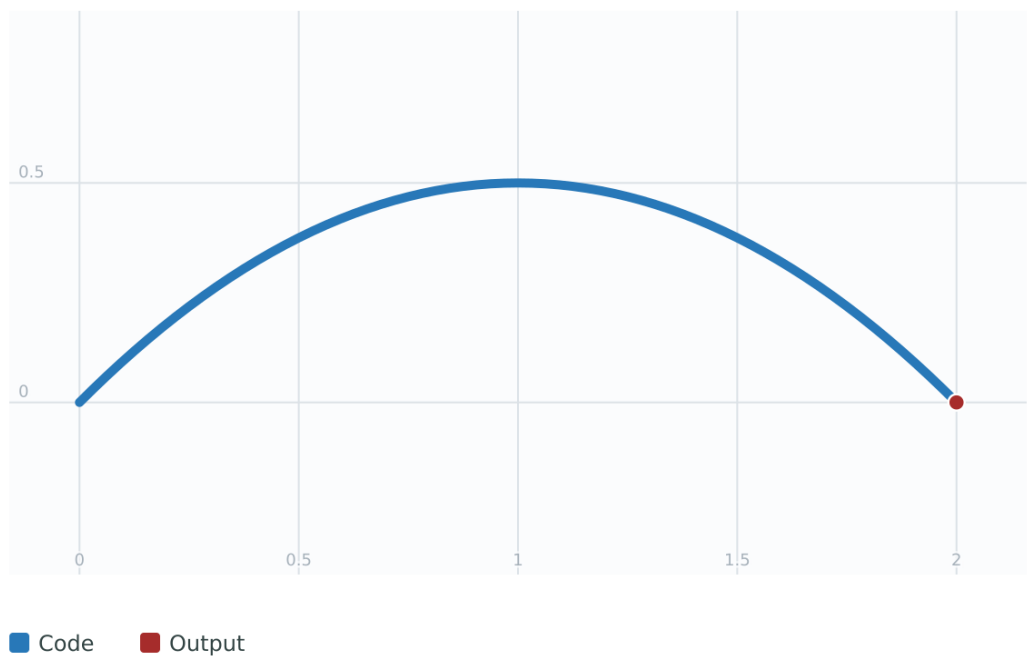
Code

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

Output

```
POINT(2 0)
```

Figure



相关信息

[ST_PointN](#), [ST_StartPoint](#)

7.4.11 ST_Envelope

ST_Envelope — 返回表示几何图形边界框的几何图形。

Synopsis

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

描述

以几何体的形式返回所提供几何体的双精度 (float8) 最小边界框。多边形由边界框的角点定义 ((MINX, MINY)、(MINX, MAXY)、(MAXX, MAXY)、(MAXX, MINY)、(MINX, MINY))。(PostGIS 也会添加 ZMIN/ZMAX 坐标)。

退化情况 (垂直线、点) 将返回比 POLYGON 更低维度的几何图形, 即 POINT 或 LINESTRING。

可用性: 1.5.0 行更改输出双精度而不是 float4



此方法符合了 [SQL 1.1 的 OGC 功能规范](#)。s2.1.1.1



此方法符合了 SQL/MM 规范。SQL-MM 3: 5.1.19

示例

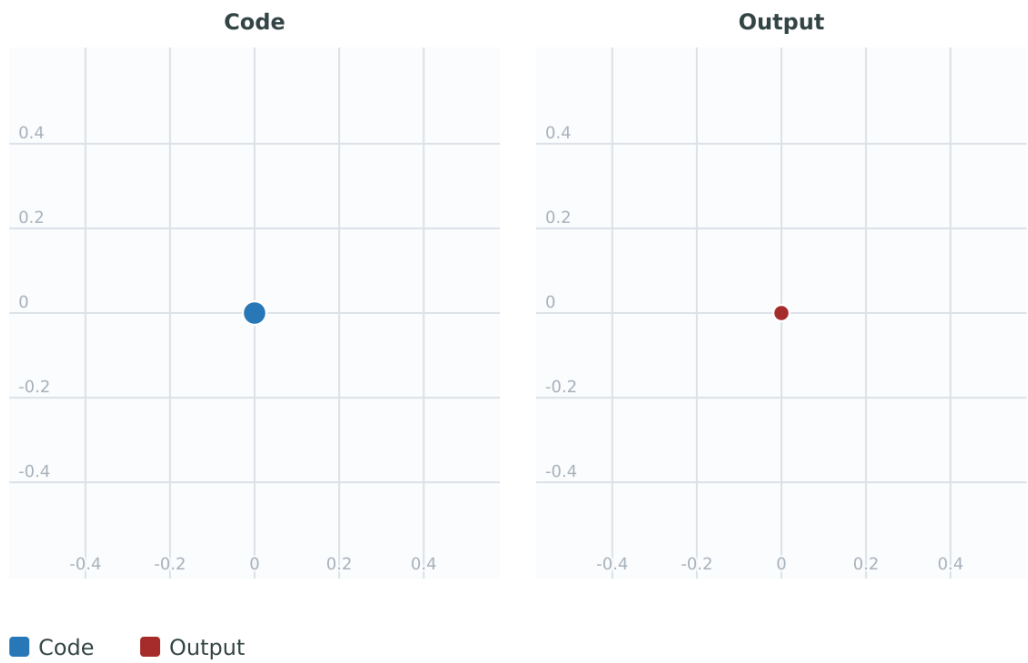
Code

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

Output

```
POINT(1 3)
```

Figure



■ Code

■ Output

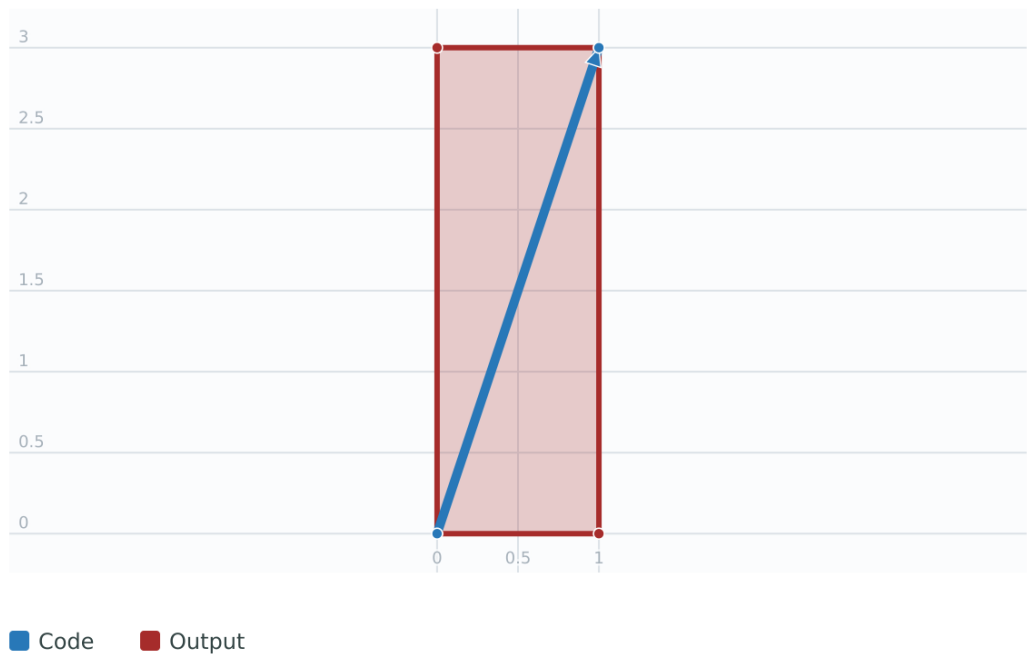
Code

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

Output

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

Figure

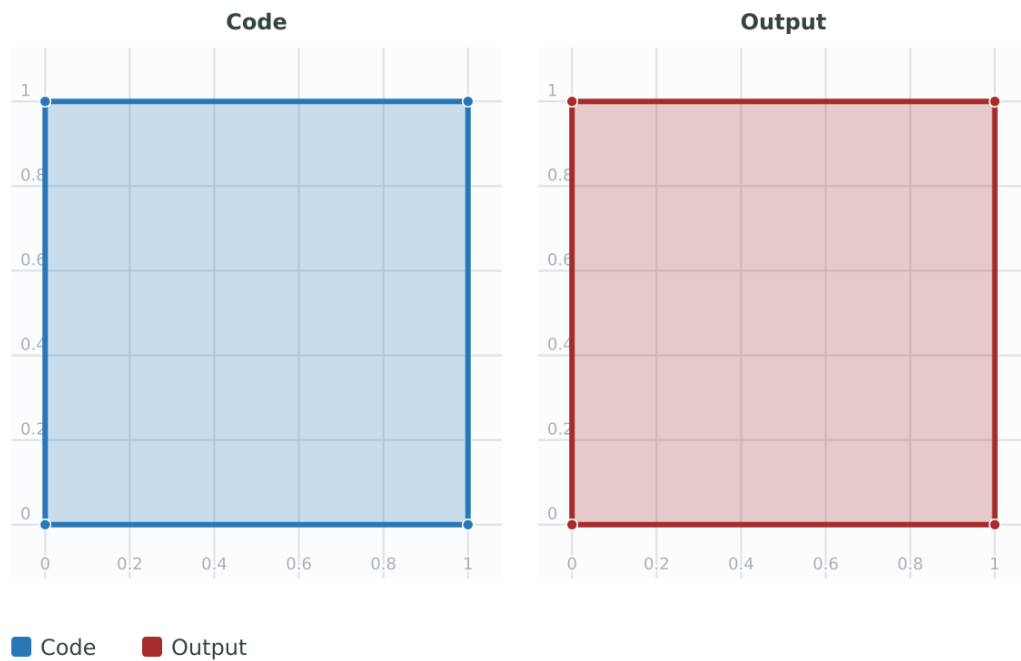


Code

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

Output

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

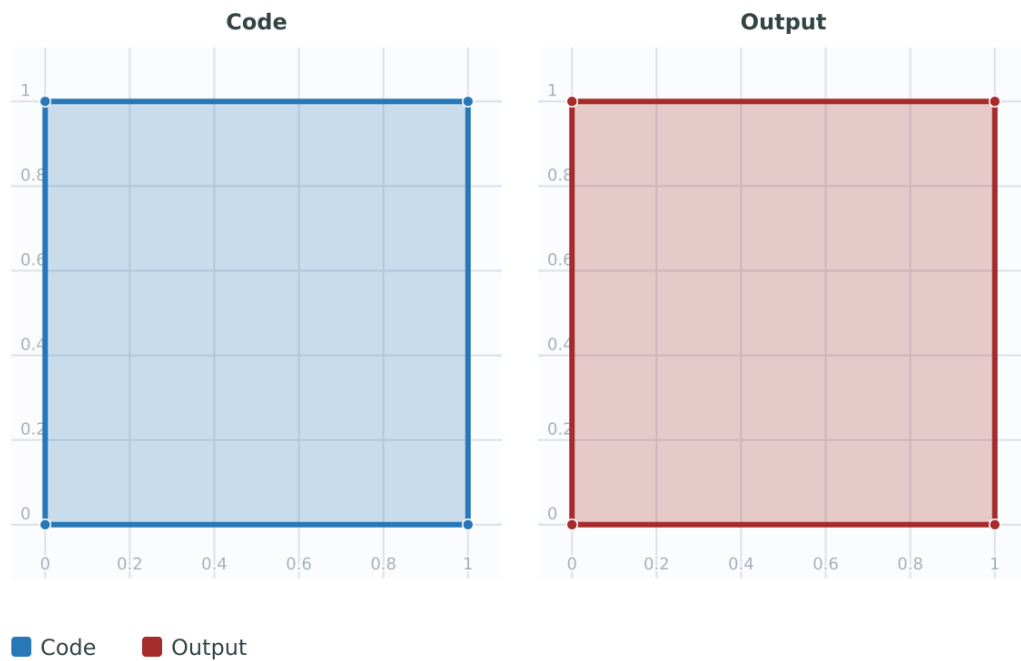
Figure**Code**

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

Output

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

Figure



Code

```
SELECT Box3D(geom), Box2D(geom), ST_Envelope(geom) As envelopewkt
FROM (SELECT 'POLYGON((0 0,0 1000012333334.34545678,1.0000001 1,1.0000001 0,0 0))'::
geometry As geom) As foo;
```

Envelope of a point and LineString.

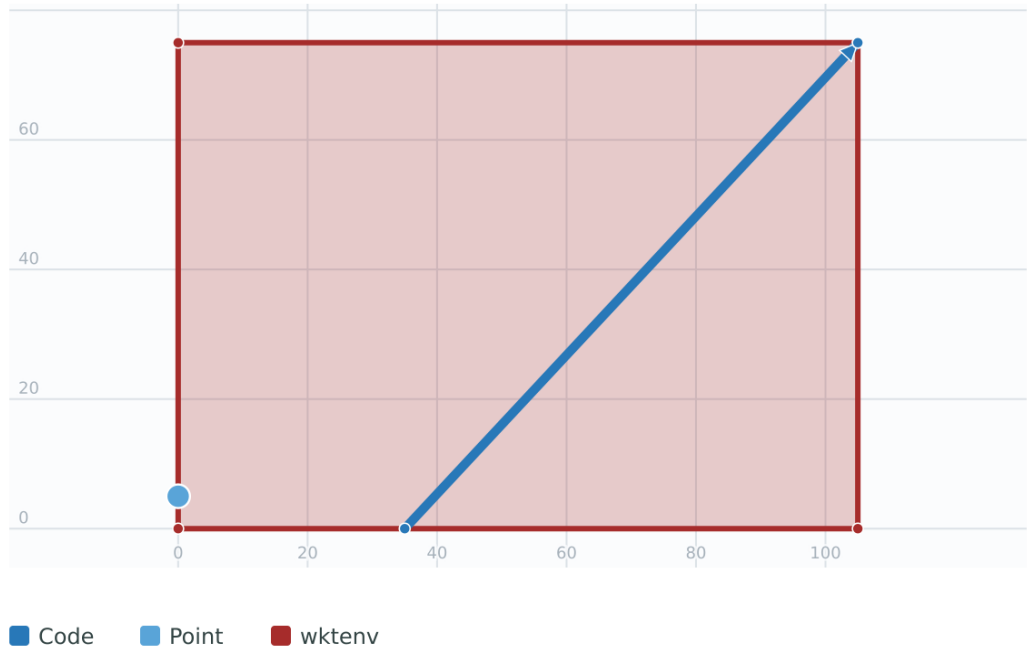
Code

```
SELECT ST_Envelope(ST_Collect(
  'LINESTRING(55 75,125 150)',
  ST_Point(20, 80)
)) As wktenv;
```

Output

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

Figure



相关信息

[Box2D](#), [Box3D](#), [ST_OrientedEnvelope](#)

7.4.12 ST_ExteriorRing

ST_ExteriorRing — 返回表示多边形外部的 LineString。

Synopsis

geometry **ST_ExteriorRing**(geometry a_polygon);

描述

返回表示多边形外部的线串。如果几何图形不是多边形，则返回 NULL。



Note

此函数不支持 MULTIPOLYGON。用于 MULTIPOLYGON，与 [ST_GeometryN](#) 或 [ST_Dump](#) 结合使用

- ✓ 此方法符合了 [SQL 1.1](#) 的 OGC 功能规范。2.1.5.1
- ✓ 该方法符合了 SQL/MM 规范。SQL-MM 3: 8.2.3, 8.3.3
- ✓ 该函数支持 3d 并且不会丢失 z-index。

示例

Extract the exterior ring from a Polygon with a hole.

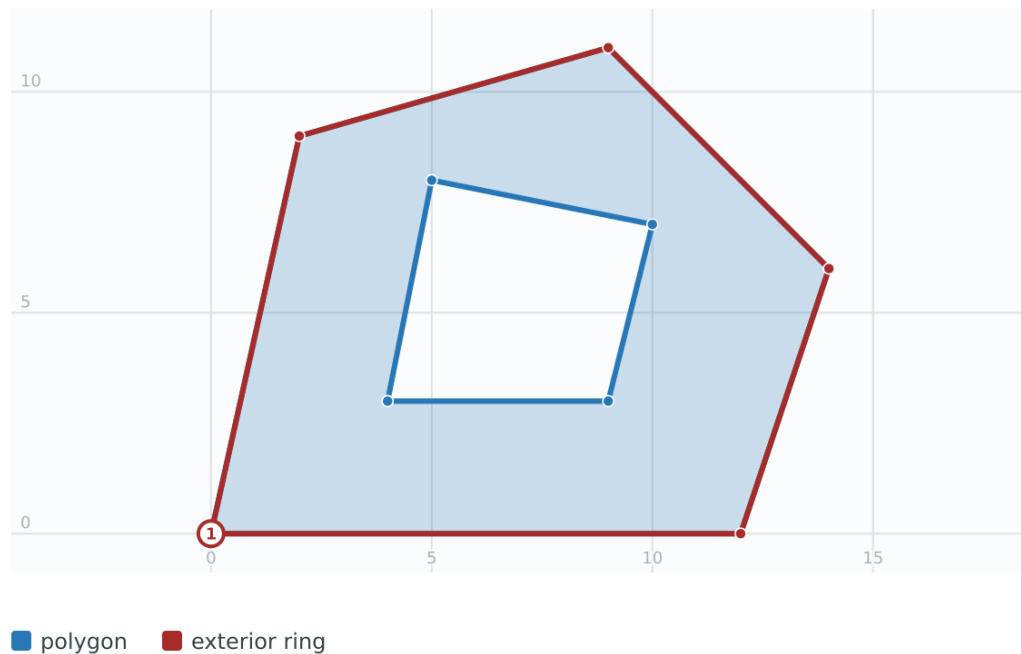
| Code

```
WITH input(geom) AS (VALUES (
  'POLYGON((0 0,2 9,9 11,14 6,12 0,0 0),(4 3,9 3,10 7,5 8,4 3))'::geometry
))
SELECT geom AS input_polygon,
       ST_ExteriorRing(geom) AS exterior_ring
FROM input;
```

| Output

```
POLYGON((0 0,2 9,9 11,14 6,12 0,0 0),(4 3,9 3,10 7,5 8,4 3)) | LINESTRING(0 0,2 9,9 11,14 6,12 0,0 0)
```

| Figure



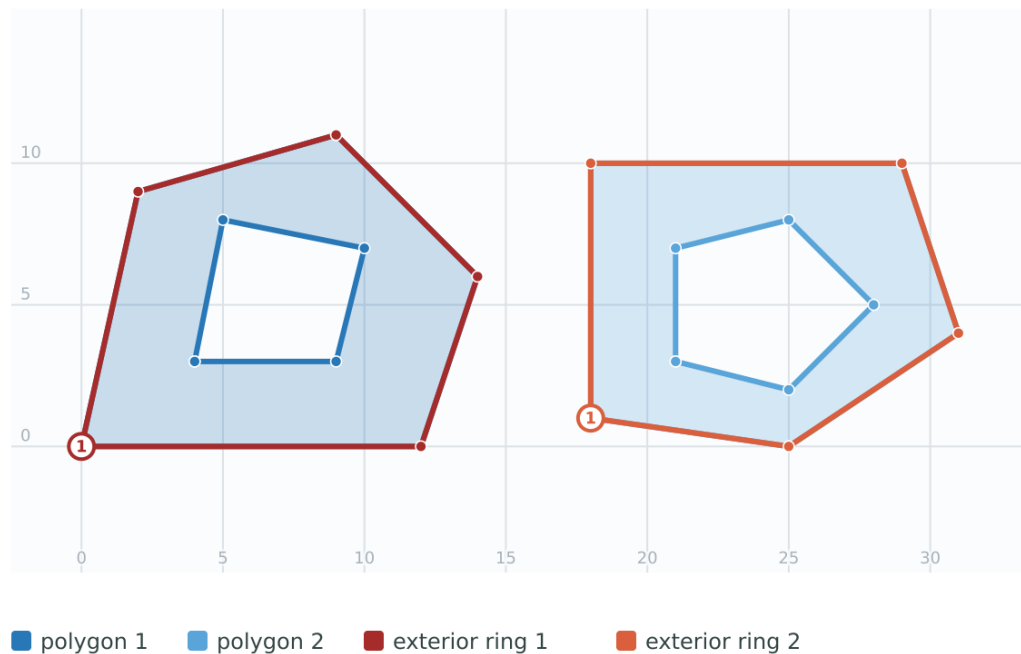
For a MultiPolygon, dump the component Polygons before extracting their exterior rings. Both component Polygons in this example have holes.

| Code

```
WITH input(geom) AS (VALUES (
  'MULTIPOLYGON(
    ((0 0,2 9,9 11,14 6,12 0,0 0),(4 3,9 3,10 7,5 8,4 3)),
    ((18 1,18 10,29 10,31 4,25 0,18 1),(21 3,25 2,28 5,25 8,21 7,21 3))
  )'::geometry
))
SELECT part.geom AS input_polygon,
       ST_ExteriorRing(part.geom) AS exterior_ring
FROM input
CROSS JOIN LATERAL ST_Dump(geom) AS part
ORDER BY part.path;
```


Output

```
POLYGON((0 0,2 9,9 11,14 6,12 0,0 0),(4 3,9 3,10 7,5 8,4 3)) | LINESTRING(0 0,2 9,9 11,14 6,12 0,0 0)
POLYGON((18 1,18 10,29 10,31 4,25 0,18 1),(21 3,25 2,28 5,25 8,21 7,21 3)) | LINESTRING(18 1,18 10,29 10,31 4,25 0,18 1) ↵
```

Figure

Extract exterior rings from a table of Polygons.

Code

```
SELECT gid, ST_ExteriorRing(geom) AS ering
FROM sometable;
```

For a table of MultiPolygons, dump the component Polygons and collect their exterior rings into a MultiLineString.

Code

```
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;
```

This example returns the 3D exterior ring of a Polygon.

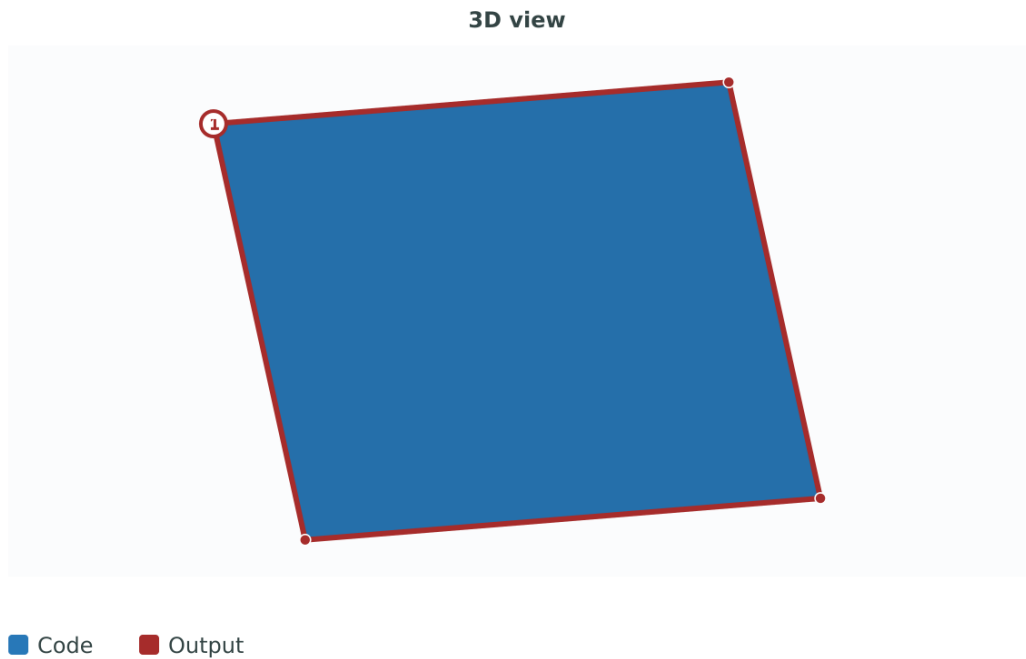
Code

```
SELECT ST_ExteriorRing(
  'POLYGON Z ((0 0 5,4 0 5,4 3 5,0 3 5,0 0 5))'
);
```

| **Output**

```
LINESTRING(0 0 5,4 0 5,4 3 5,0 3 5,0 0 5)
```

| **Figure**



相关信息

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

7.4.13 ST_GeometryN

ST_GeometryN — 返回几何集合的一个元素。

Synopsis

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

描述

Returns the 1-based Nth element of a GEOMETRYCOLLECTION, MULTIPOINT, MULTILINESTRING, MULTICURVE, or MULTIPOLYGON. For a unitary geometry, including a TIN or PolyhedralSurface, an index of 1 returns the input geometry and any other index returns NULL. Use [ST_PatchN](#) and [ST_NumPatches](#) to access and count the faces of a TIN or PolyhedralSurface.



Note
自版本 0.8.0 以来，OGC 规范的索引从 1 开始。以前的版本将其基于 0。

**Note**

要提取几何图形的所有元素，**ST_Dump** 效率更高，并且适用于基本几何图形。

增强功能：引入了 2.0.0 多面体曲面、三角形和三角网的支持。

更改：2.0.0 版本。之前的版本对于奇异几何图形会返回 NULL。现在在已更改在 ST_GeometryN(..,1) 情况下返回几何图形。

Changed: 3.6.0 TIN and PolyhedralSurface are treated as unitary geometries and are not decomposed into patches; use ST_PatchN to access faces.

- ✓ 此方法符合了 SQL 1.1 的 OGC 功能规范。
- ✓ 此方法符合了 SQL/MM 规范。SQL-MM 3: 9.1.5
- ✓ 此函数支持 3d 并且不会丢失 z-index。
- ✓ 此方法支持图形字符串和曲线。
- ✓ 此函数支持多面体曲面。
- ✓ 此函数支持三角形和不规则三角网面 (TIN)。

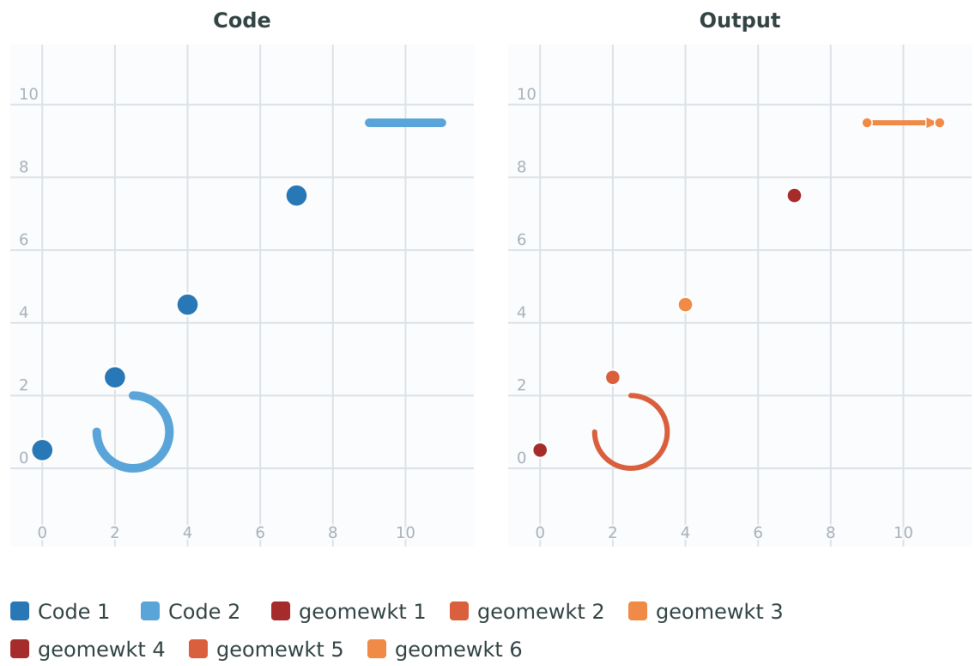
示例

Extracting a subset of points from a 3D multipoint.

Code

```
SELECT n, ST_GeometryN(geom, n) As geomewkt
FROM (
VALUES
  (1, 'MULTIPOINT((1 2 7), (3 4 7), (5 6 7), (8 9 10))'::geometry ),
  (2, 'MULTICURVE(CIRCULARSTRING(2.5 2.5,4.5 2.5, 3.5 3.5), (10 11, 12 11))'::geometry )
) As foo(id, geom)
CROSS JOIN generate_series(1,100) n
WHERE n <= ST_NumGeometries(geom)
ORDER BY id, n;
```

Figure



Extracting all geometries (useful when you want to assign an id).

Code

```
SELECT gid, n, ST_GeometryN(geom, n)
FROM sometable CROSS JOIN generate_series(1,100) n
WHERE n <= ST_NumGeometries(geom);
```

多面体曲面、**TIN** 和三角形的示例

A PolyhedralSurface or TIN is treated as a single geometry, not as a collection of faces. ST_GeometryN does not decompose the surface into patches, so an index greater than 1 returns NULL. Use **ST_PatchN** and **ST_NumPatches** to access individual faces.

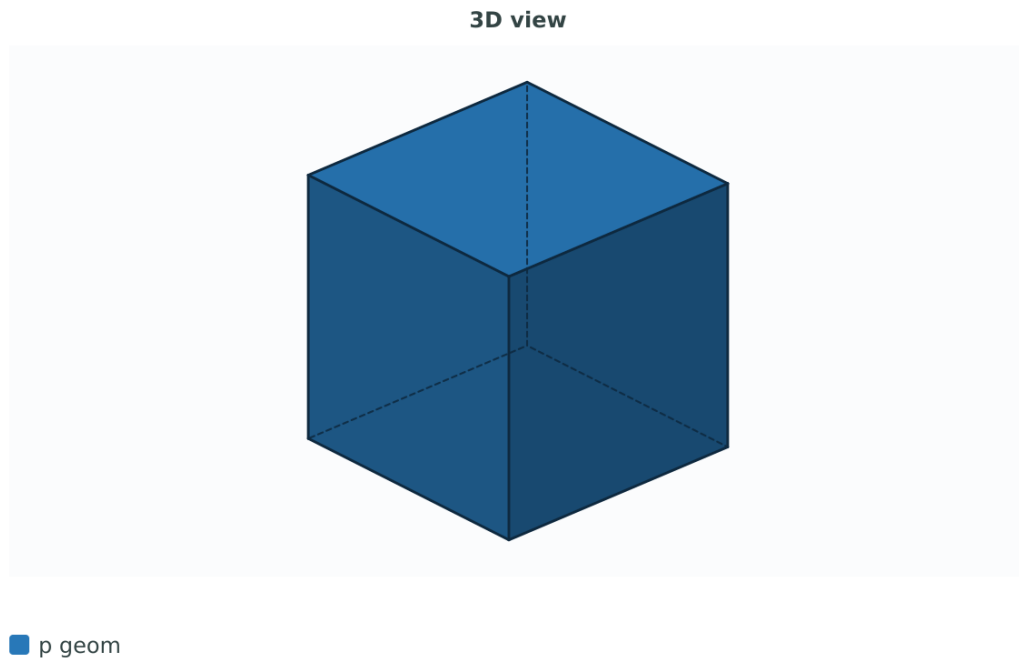
This example requests the third element of a PolyhedralSurface.

Code

```
SELECT ST_GeometryN(p_geom, 3) As geom_ewkt
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 p_geom ) AS a;
```

Output

NULL

Figure

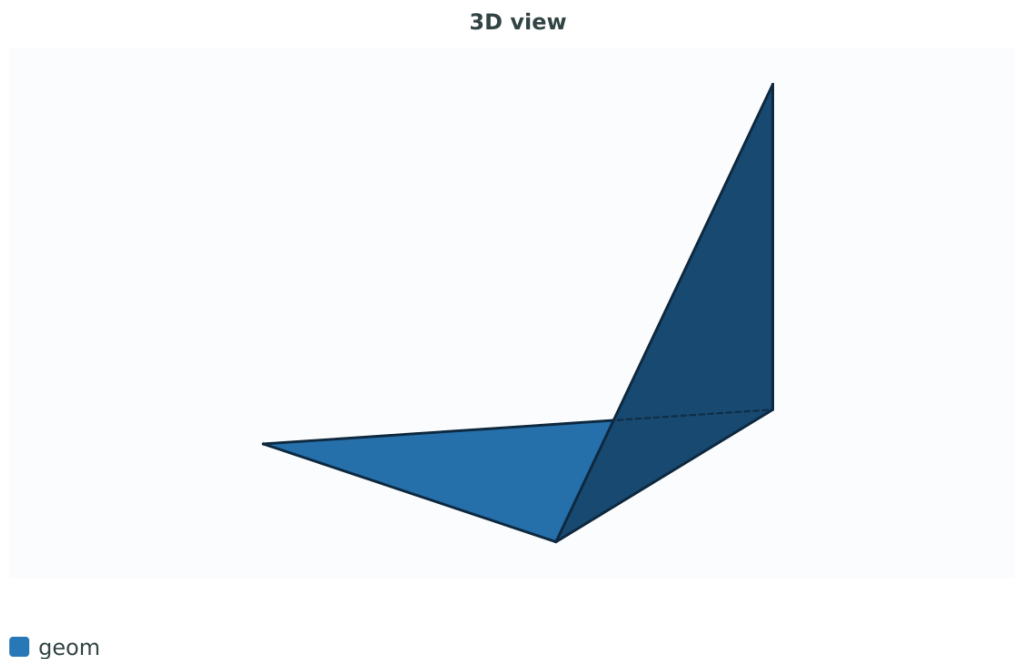
This example requests the second element of a TIN.

Code

```
SELECT ST_GeometryN(geom, 2) as wkt
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;
```

Output

NULL

Figure

相关信息

[ST_Dump](#), [ST_NumGeometries](#), [ST_PatchN](#), [ST_NumPatches](#)

7.4.14 ST_GeometryType

ST_GeometryType — 以文本形式返回几何图形的 SQL-MM 型。

Synopsis

```
text ST_GeometryType(geometry g1);
```

描述

以字符串形式返回几何型，例如“ST_LineString”、“ST_Polygon”、“ST_MultiPolygon”等。与 GeometryType 不同，此函数前面有“ST”，并不指示它是否具有 M 。

增：2.0.0 引入了多面体曲面的支持。

✓ 方法了 SQL/MM 范。SQL-MM 3: 5.1.4

✓ 函数支持 3d 并且不会失 z-index。

✓ 函数支持多面体曲面。

示例

Code

```
SELECT ST_GeometryType('LINESTRING(77.29 29.07,77.42 29.26,77.27 29.31,77.29 29.07)'):: geometry);
```

Output

ST_LineString

Code

```
SELECT ST_GeometryType('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 0,0 1 0)),((0 0 1,1 0 1,1 1 1,0 1 1,0 0 1)) )'::geometry);
```

Output

ST_PolyhedralSurface

Code

```
SELECT ST_GeometryType(geom) AS result
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;
```

Output

ST_Tin

相关信息

GeometryType

7.4.15 ST_HasArc

ST_HasArc — 几何形是否包含弧

Synopsis

boolean **ST_HasArc**(geometry geomA);

描述

如果几何或几何集合包含圆弧字符串，返回 `true`

可用性：1.2.3？

- ✓ 函数支持 3d 并且不会丢失 z-index。
- ✓ 此方法支持圆弧字符串和圆弧。

示例

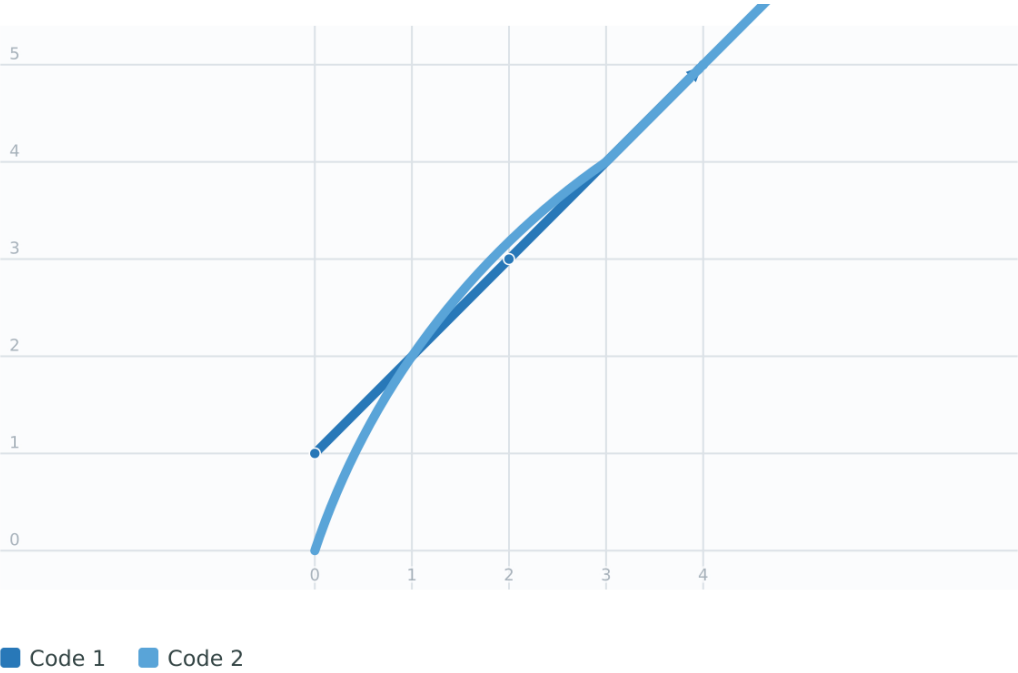
Code

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

Output

t

Figure



相关信息

[ST_CurveToLine](#), [ST_LineToCurve](#)

7.4.16 ST_InteriorRingN

`ST_InteriorRingN` — 返回多边形的第 N 个内环（孔）。

Synopsis

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

描述

以 LINESTRING 形式返回 POLYGON 几何体的第 N 个内环（孔）。索引从 1 开始。如果几何形状不是多边形或索引超出范围，则返回 NULL。



Note

此函数不支持 MULTIPOLYGON。用于 MULTIPOLYGON，与 **ST_GeometryN** 或 **ST_Dump** 结合使用



此方法符合了 SQL 1.1 的 OGC 功能规范。



该方法符合了 SQL/MM 规范。SQL-MM 3: 8.2.6, 8.3.5

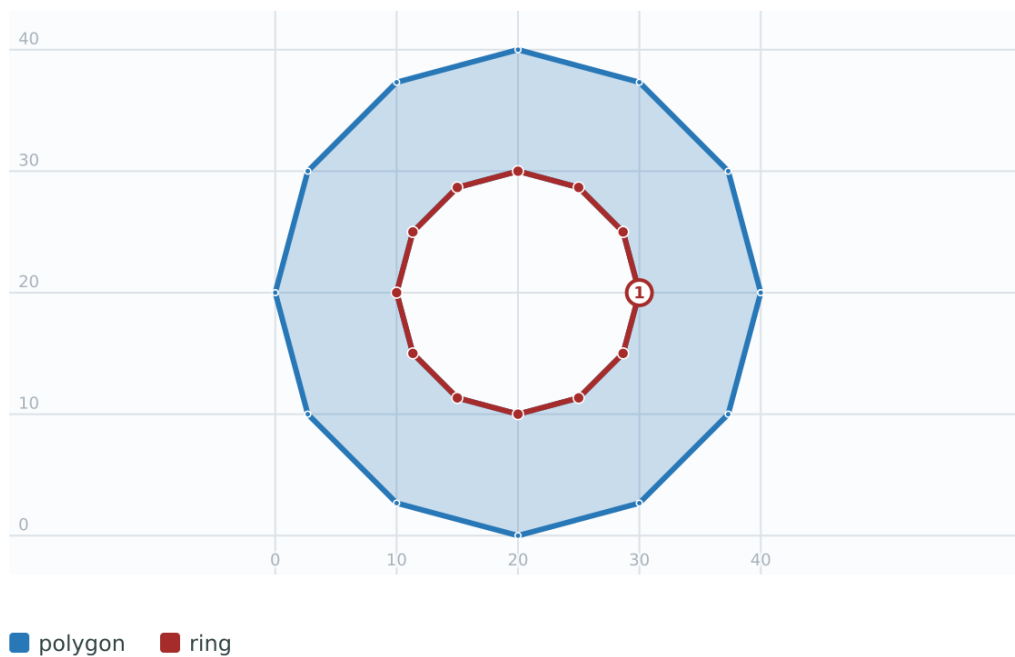


该函数支持 3d 并且不会丢失 z-index。

示例

Code

```
WITH data AS (  
  SELECT ST_BuildArea(ST_Collect(  
    ST_Buffer(ST_Point(1, 2), 20, 3),  
    ST_Buffer(ST_Point(1, 2), 10, 3)  
  )) AS geom  
)  
SELECT geom AS input_polygon,  
       ST_InteriorRingN(geom, 1) AS ring  
FROM data;
```

Figure

相关信息

[ST_ExteriorRing](#), [ST_BuildArea](#), [ST_Collect](#), [ST_Dump](#), [ST_NumInteriorRing](#), [ST_NumInteriorRings](#)

7.4.17 ST_NumCurves

ST_NumCurves — 返回复合曲线中的曲线数量。

Synopsis

```
integer ST_NumCurves(geometry a_compoundcurve);
```

描述

返回复合曲线中的曲线数量。如果输入空的复合曲线，返回 0；如果不是复合曲线，返回 NULL。

✓ 方法符合了 SQL/MM 规范。SQL-MM 3: 8.2.6, 8.3.5

✓ 函数支持 3d 并且不会丢失 z-index。

示例

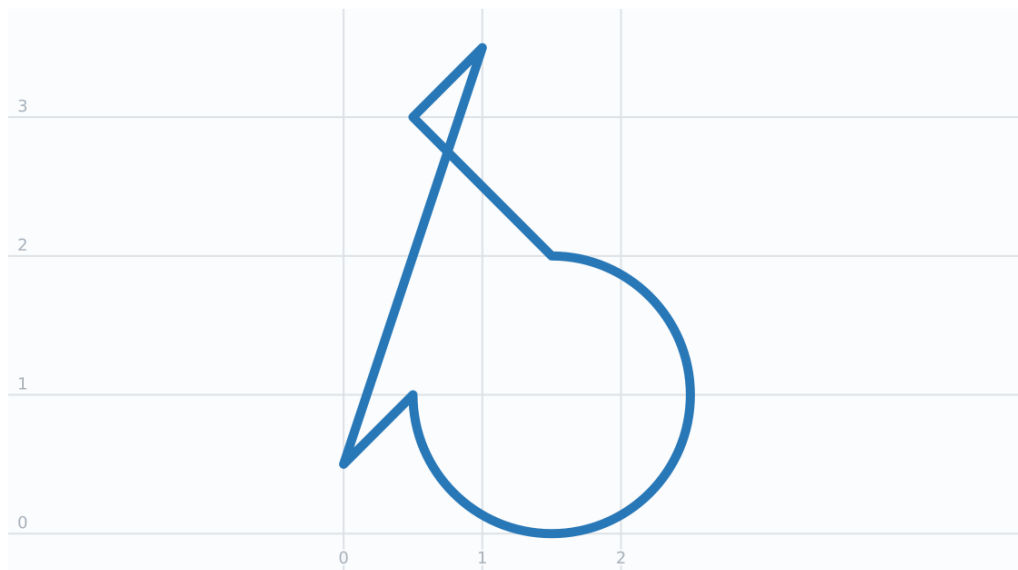
This CompoundCurve contains three component curves.

Code

```
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)  
)' );
```

Output

```
3
```

Figure

■ Code

An empty CompoundCurve has no component curves.

Code

```
SELECT ST_NumCurves( 'COMPOUNDCURVE EMPTY' );
```

Output

```
0
```

相关信息

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

7.4.18 ST_CurveN

`ST_CurveN` — 返回复合曲线（CompoundCurve）中的第 N 个曲线几何体。

Synopsis

geometry **ST_CurveN**(geometry a_compoundcurve, integer index);

描述

返回复合曲线 (CompoundCurve) 中的第 N 个曲线几何体。索引从 1 开始。如果输入的几何体不是复合曲线，或者索引超出范围，返回 NULL。

✓ 方法符合了 SQL/MM 规范。SQL-MM 3: 8.2.6, 8.3.5

✓ 函数支持 3d 并且不会丢失 z-index。

示例

This example extracts the second component curve, a CircularString, from a CompoundCurve.

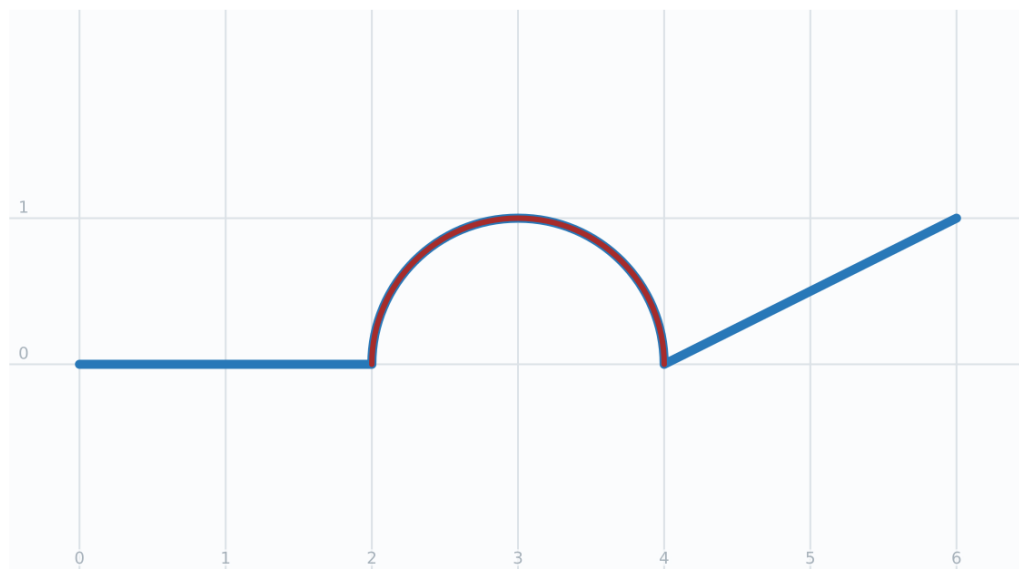
Code

```
SELECT ST_CurveN(  
  'COMPOUNDCURVE(  
    (0 0,2 0),  
    CIRCULARSTRING(2 0,3 1,4 0),  
    (4 0,6 1)  
  )',  
  2) AS curve;
```

Output

```
CIRCULARSTRING(2 0,3 1,4 0)
```

Figure



■ Code ■ curve

相关信息

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

7.4.19 ST_IsClosed

ST_IsClosed — 检查 LineStrings 的起点和终点是否重合。对于多面体表面检查是否闭合（实心）。

Synopsis

```
boolean ST_IsClosed(geometry g);
```

描述

如果 **LINestring** 的起点和终点重合，返回 **TRUE**。对于多面体曲面，检查曲面是面状的（开放的）还是体状的（闭合的）。

✓ 此方法符合了 **SQL 1.1** 的 **OGC 简单功能规范**。

✓ 此方法符合了 **SQL/MM** 规范。SQL-MM 3: 7.1.5, 9.3.3



Note

SQL-MM 定义了 ST_IsClosed(NULL) 的结果为 0，而 PostGIS 返回 NULL。

✓ 该函数支持 3d 并且不会丢失 z-index。

✓ 此方法支持几何字符串和曲线。

增强：2.0.0 引入了对多面体曲面的支持。

✓ 该函数支持多面体曲面。

字符串和点的示例

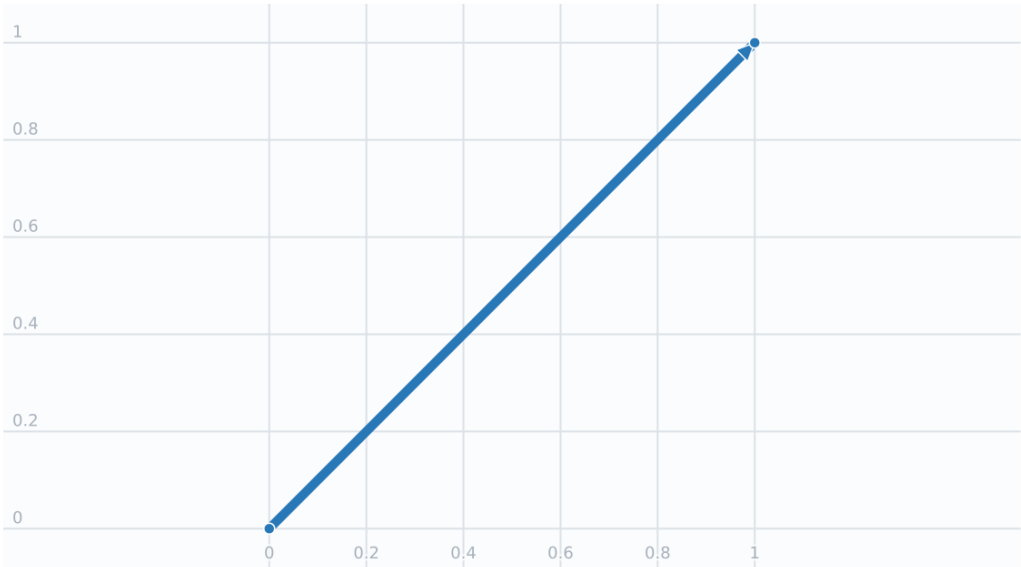
Code

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

Output

```
f
```

Figure



Code

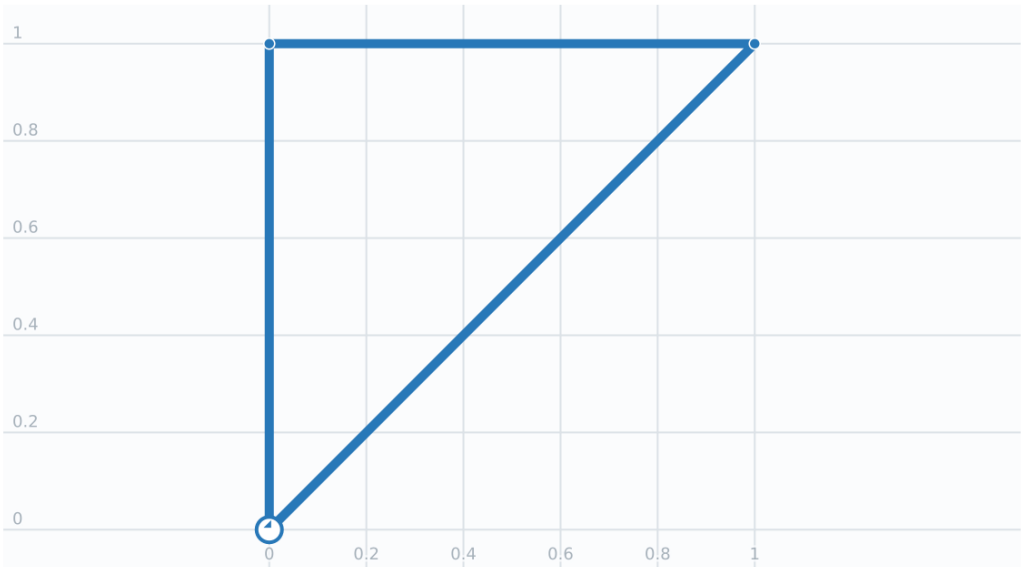
Code

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

Output

t

Figure



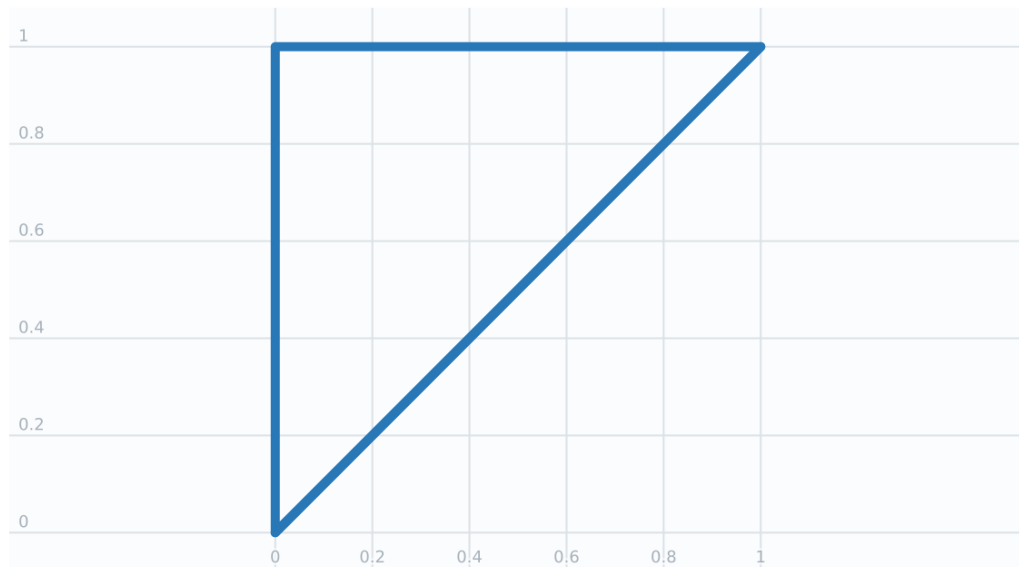
Code

Code

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

Output

f

Figure

■ Code

Code

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

Output

t

Code

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

Output

t

多面体曲面示例

This example uses a cube.

Code

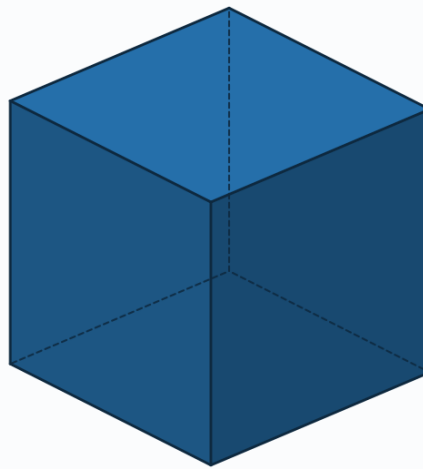
```
SELECT ST_IsClosed('POLYHEDRALSURFACE( ((0 0 0,0 0 1,0 1 1,0 1 0,0 0 0)),
((0 0 0,0 1 0,1 1 0,1 0 0,0 0 0)),((0 0 0,1 0 0,1 0 1,0 0 1,0 0 0)),
((1 1 0,1 1 1,1 0 1,1 0 0,1 1 0)),
((0 1 0,0 1 1,1 1 1,1 1 0,0 1 0)),((0 0 1,1 0 1,1 1 1,0 1 1,0 0 1)) )');
```

Output

t

Figure

3D view


[Code](#)

This example uses the same cube with one side missing.

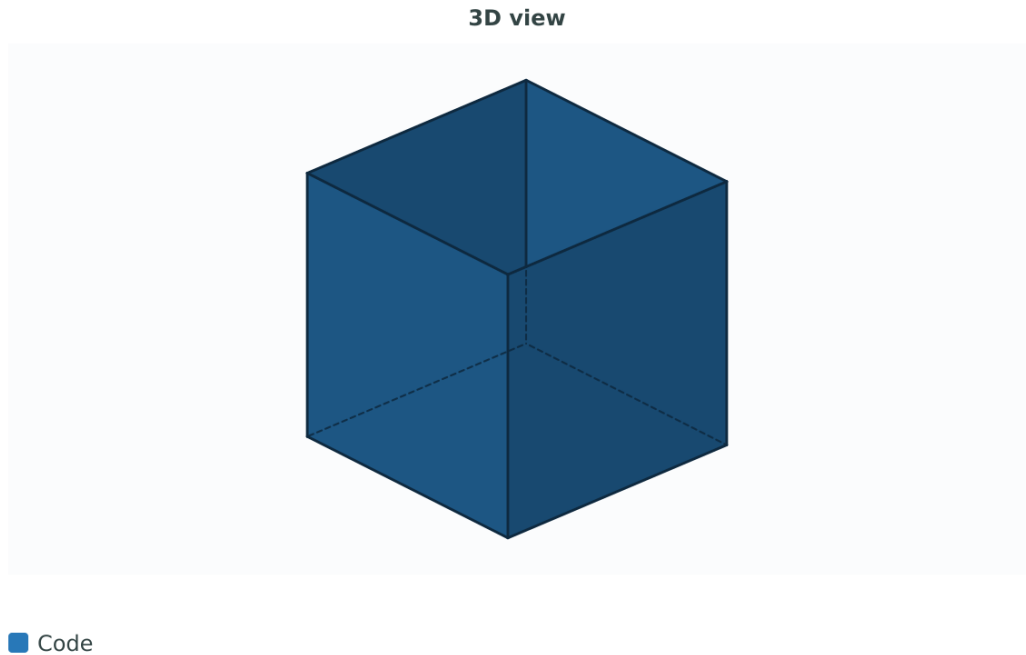
Code

```
SELECT ST_IsClosed('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)) )');
```

Output

f

Figure



相关信息

[ST_IsRing](#)

7.4.20 ST_IsCollection

`ST_IsCollection` — 判断几何类型是否是几何集合。

Synopsis

```
boolean ST_IsCollection(geometry g);
```

描述

如果参数的几何类型是几何集合类型，则返回 `TRUE`。集合类型有以下几种：

- 几何集合
- 多点、多多边形、多线串、多曲面、多曲面
- 复合曲面



Note

该函数分析几何形状的类型。这意味着它将在空集合或包含 0 个元素的集合上返回 `TRUE`。



该函数支持 3d 并且不会丢失 z-index。



此方法支持形状字符串和曲面。

示例

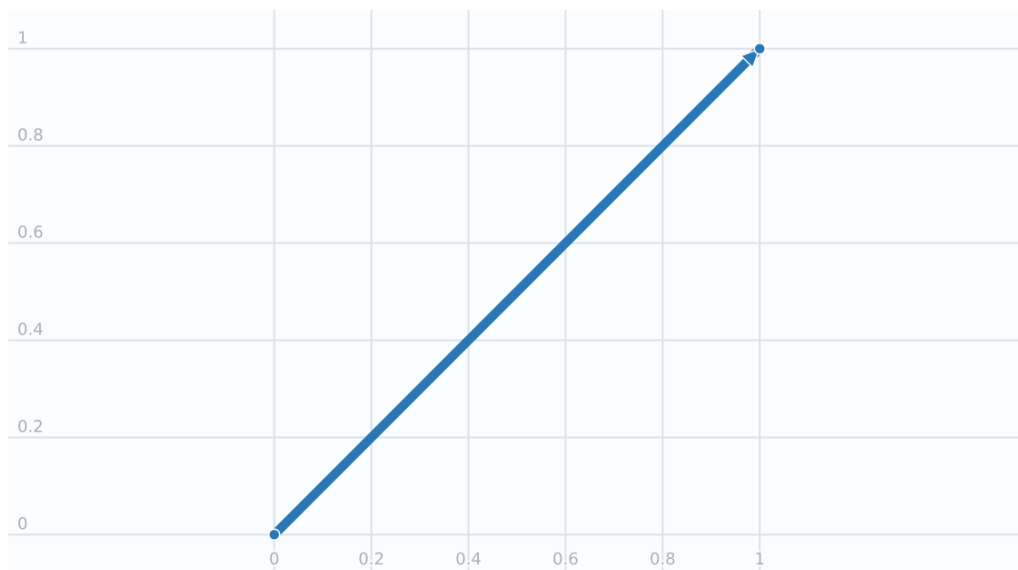
Code

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

Output

f

Figure



Code

Code

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

Output

t

Code

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

Output

t

Code

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

| Output

```
t
```

| Code

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

| Output

```
t
```

相关信息

[ST_NumGeometries](#)

7.4.21 ST_IsEmpty

ST_IsEmpty — 几何图形是否空。

Synopsis

boolean **ST_IsEmpty**(geometry geomA);

描述

如果几何图形是空几何，则返回 true; 如果 false，则此几何图形是空几何集合、多边形、点等。

**Note**

在 SQL-MM 中，ST_IsEmpty (NULL) 返回 0，而在 PostGIS 中返回 NULL。

✓ 此方法符合了 [SQL 1.1 的 OGC 功能规范](#)。s2.1.1.1

✓ 此方法符合了 SQL/MM 规范。SQL-MM 3: 5.1.7

✓ 此方法支持图形字符串和曲线。

已更改：2.0.0 之前的 PostGIS 版本允许 ST_GeomFromText (“GEOMETRYCOLLECTION(EMPTY)”)。在 PostGIS 2.0.0 中，这是不正确的，因为它更符合 SQL/MM 规范

示例

| Code

```
SELECT ST_IsEmpty('GEOMETRYCOLLECTION EMPTY'::geometry);
```

| Output

t

| Code

```
SELECT ST_IsEmpty('POLYGON EMPTY'::geometry);
```

| Output

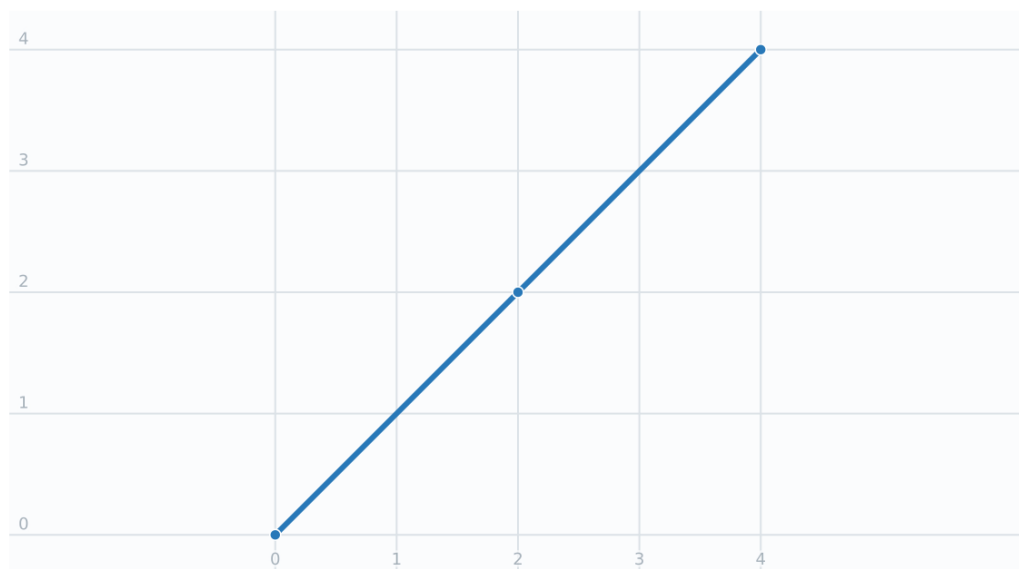
t

| Code

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

| Output

f

| Figure

■ Code

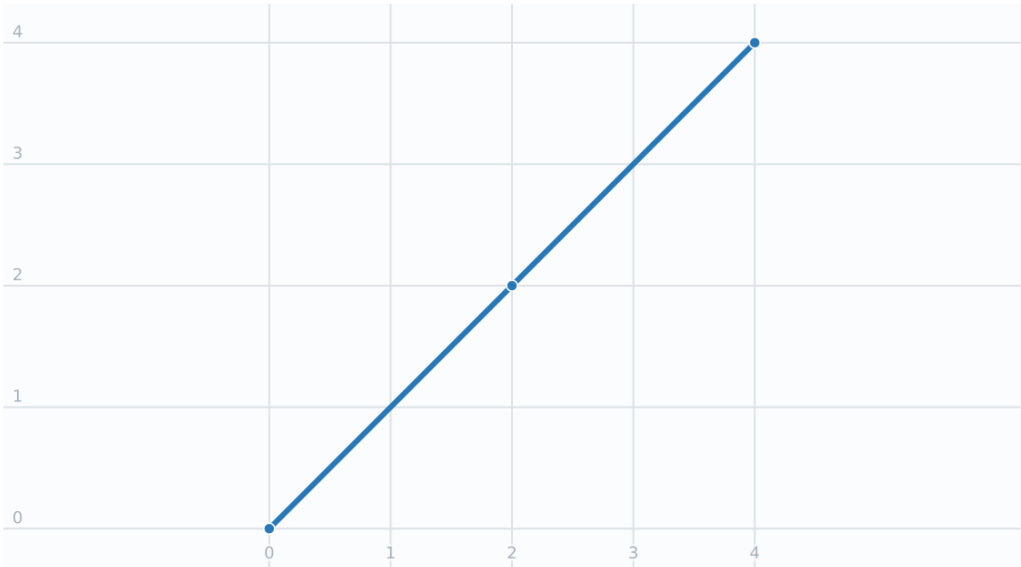
| Code

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

| Output

t
(1 row)

Figure



Code

Code

```
SELECT ST_IsEmpty('CIRCULARSTRING EMPTY'::geometry);
```

Output

```
t
```

7.4.22 ST_IsPolygonCCW

ST_IsPolygonCCW — 判断多边形是否具有逆时针方向的外环和顺时针方向的内环。

Synopsis

boolean **ST_IsPolygonCCW**(geometry geom);

描述

如果输入几何体的所有多边形元素其外环使用逆时针方向，所有内环使用顺时针方向，返回 true。
如果几何体没有多边形元素，返回 true。



Note
闭合串不被认为是多边形元素。如果输入是个闭合串，无论它是顺时针还是逆时针，都会得到 TRUE。

**Note**

如果多边形元素的内部未反时针（即，有一个或多个内部以与外部相同的方向旋转），则 `ST_IsPolygonCW` 和 `ST_IsPolygonCCW` 都返回 `FALSE`。

可用性：2.4.0



该函数支持 3d 并且不会丢失 z-index。



该功能支持 M 坐标。

相关信息

[ST_ForcePolygonCW](#), [ST_ForcePolygonCCW](#), [ST_IsPolygonCW](#)

7.4.23 ST_IsPolygonCW

`ST_IsPolygonCW` — 检查多边形是否具有顺时针外部和逆时针内部。

Synopsis

boolean **ST_IsPolygonCW**(geometry geom);

描述

如果输入几何体的所有多边形元素其外部使用顺时针方向，所有内部使用逆时针方向，则返回 `true`。

如果几何体没有多边形元素，则返回 `true`。

**Note**

闭合串不被认为是多边形元素。如果每个闭合串，无论它是顺时针还是逆时针，都会得到 `TRUE`。

**Note**

如果多边形元素的内部未反时针（即，有一个或多个内部以与外部相同的方向旋转），则 `ST_IsPolygonCW` 和 `ST_IsPolygonCCW` 都返回 `FALSE`。

可用性：2.4.0



该函数支持 3d 并且不会丢失 z-index。



该功能支持 M 坐标。

相关信息

[ST_ForcePolygonCW](#), [ST_ForcePolygonCCW](#), [ST_IsPolygonCW](#)

7.4.24 ST_IsRing

ST_IsRing — 判断字符串是否是闭合的。

Synopsis

boolean **ST_IsRing**(geometry g);

描述

Returns TRUE if this LINESTRING is both **ST_IsClosed** (ST_StartPoint(g) ~= ST_EndPoint(g)) and **ST_IsSimple** (does not self intersect).

- ✓ 此方法符合了 SQL 1.1 的 OGC 功能规范。 2.1.5.1
- ✓ 此方法符合了 SQL/MM 规范。 SQL-MM 3: 7.1.6



Note
SQL-MM 定义了 ST_IsRing(NULL) 的结果为 0，而 PostGIS 返回 NULL。

示例

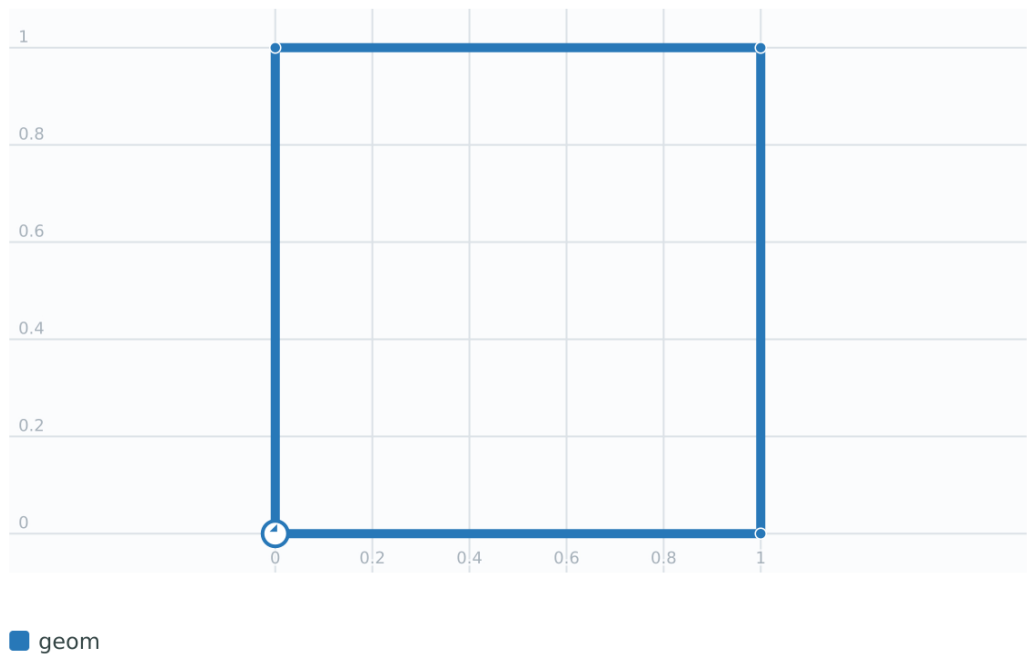
Code

```
SELECT ST_IsRing(geom), ST_IsClosed(geom), ST_IsSimple(geom)
FROM (SELECT 'LINESTRING(0 0,0 1,1 1,1 0,0 0) '::geometry AS geom) AS foo;
```

Output

st_isring	st_isclosed	st_issimple
t	t	t
(1 row)		

Figure



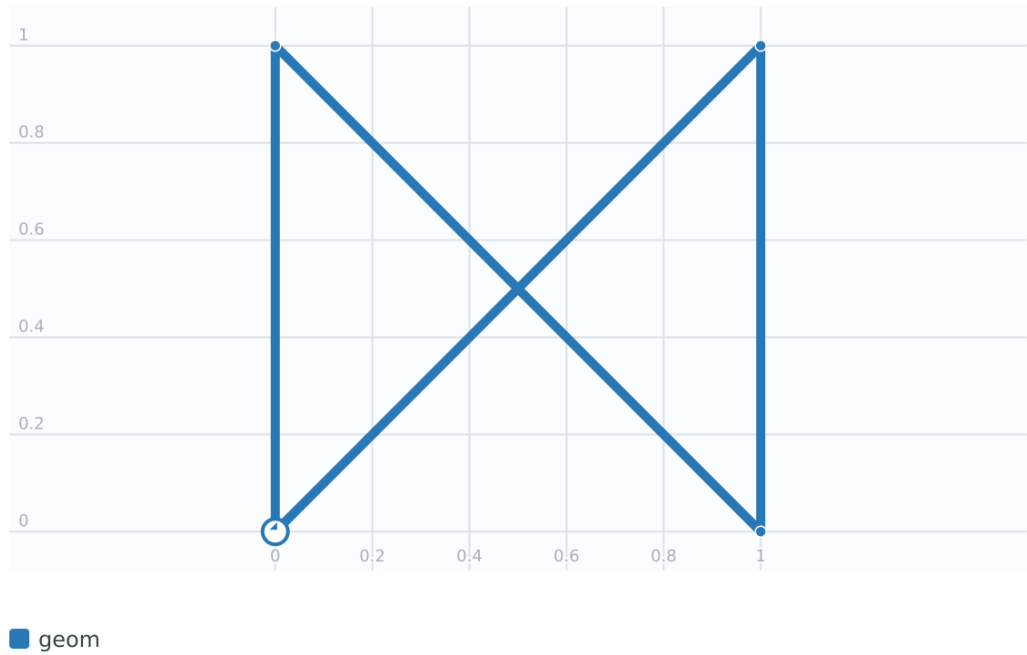
Code

```
SELECT ST_IsRing(geom), ST_IsClosed(geom), ST_IsSimple(geom)
FROM (SELECT 'LINESTRING(0 0,0 1,1 0,1 1,0 0)'::geometry AS geom) AS foo;
```

Output

st_isring	st_isclosed	st_issimple
f	t	f
(1 row)		

Figure



相关信息

[ST_IsClosed](#), [ST_IsSimple](#), [ST_StartPoint](#), [ST_EndPoint](#)

7.4.25 ST_IsSimple

`ST_IsSimple` — 检查几何体的自完整性或自接触点。

Synopsis

boolean **ST_IsSimple**(geometry geomA);

描述

如果此 Geometry 没有异常几何点（例如自相交或自相切），则返回 true。有关 OGC 几何类型性和有效性定义的更多信息，请参考“[确保几何类型的 OpenGIS 兼容性](#)”



Note

SQL-MM 定义 `ST_IsSimple(NULL)` 的结果为 0，而 PostGIS 返回 NULL。

- ✓ 此方法符合了 [SQL 1.1 的 OGC 功能规范](#)。s2.1.1.1
- ✓ 该方法符合了 SQL/MM 规范。SQL-MM 3: 5.1.8
- ✓ 该函数支持 3d 并且不会丢失 z-index。

示例

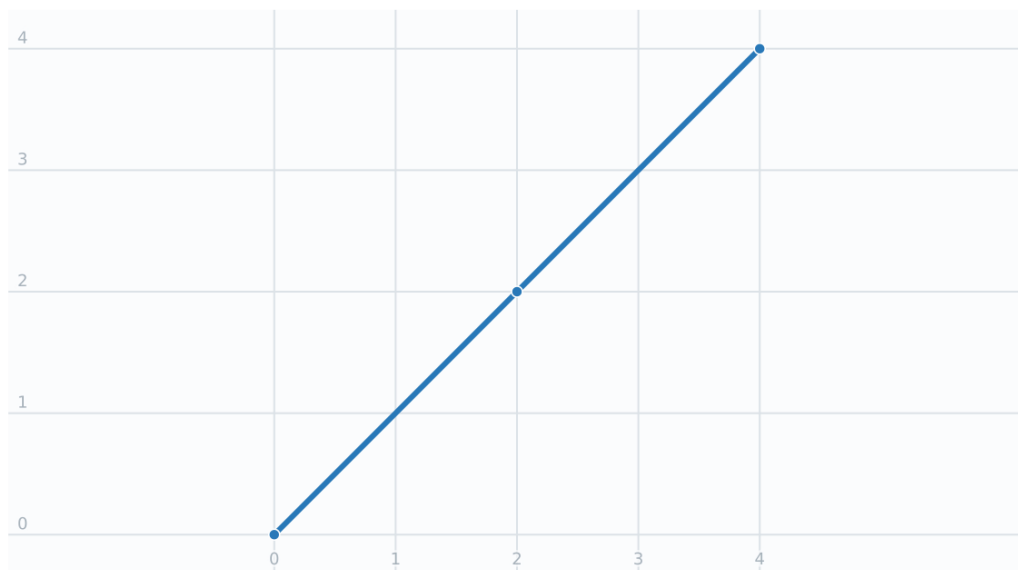
Code

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

Output

f

Figure



■ Code

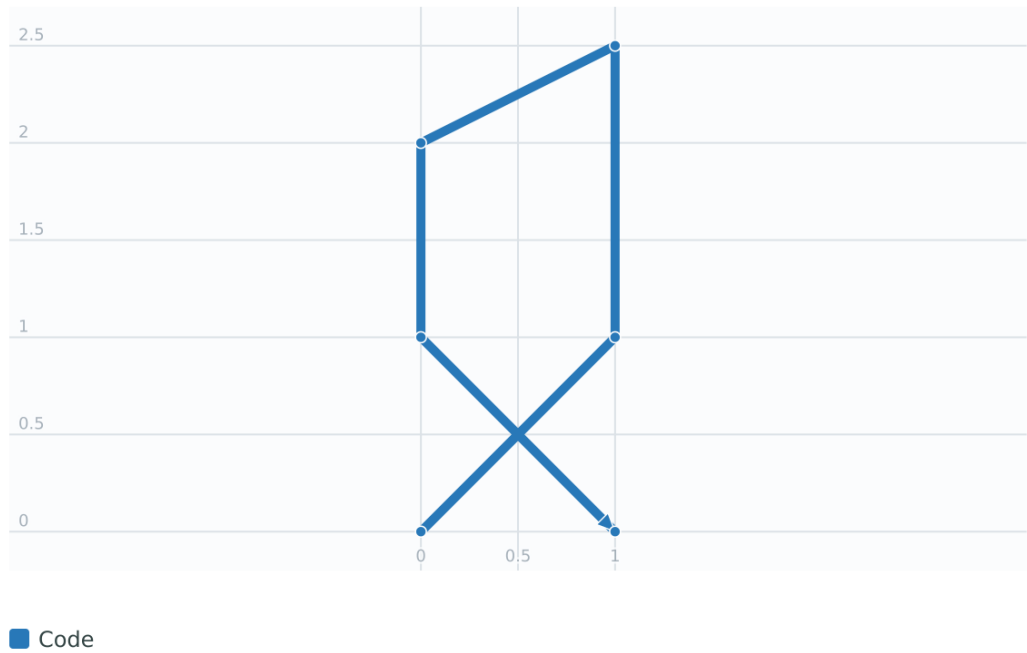
Code

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

Output

f

Figure



相关信息

[ST_IsValid](#)

7.4.26 ST_M

ST_M — 返回点的 M 值。

Synopsis

```
float ST_M(geometry a_point);
```

描述

返回点的 M 坐标值，如果无效，返回 NULL。输入必须是点。



Note

此函数不是 OGC 规范的一部分，但在此列出以完成点坐标提取器功能列表。

- ✓ 此方法符合了 SQL 1.1 的 OGC 功能规范。
- ✓ 此方法符合了 SQL/MM 规范。
- ✓ 此函数支持 3d 并且不会丢失 z-index。

示例

Code

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

Output

```
4
```

相关信息

[ST_GeomFromEWKT](#), [ST_X](#), [ST_Y](#), [ST_Z](#)

7.4.27 ST_MemSize

`ST_MemSize` — 返回几何图形占用的内存空间量。

Synopsis

integer **ST_MemSize**(geometry geomA);

描述

返回几何图形占用的内存空间量（以字节位）。

补充了 PostgreSQL 内置 [数据对象函数](#) `pg_column_size`, `pg_size_pretty`, `pg_relation_size`, `pg_total_relation_size`。

Note



输出表的字节大小的 `pg_relation_size` 可能返回小于 `ST_MemSize` 的字节大小。这是因为 `pg_relation_size` 不会添加 toasted 表贡献，并且大型几何图形存储在 TOAST 表中。
`pg_total_relation_size` - 包括表、索引表和索引。
`pg_column_size` 返回考虑到几何图形在列中占用多少空间，因此可能低于 `ST_MemSize`

✓ 函数支持 3d 并且不会丢失 z-index。

✓ 此方法支持图形字符串和曲线。

✓ 函数支持多面体曲面。

✓ 此函数支持三角形和不规则三角网面 (TIN)。

更改：2.2.0 名称更改为 `ST_MemSize` 以遵循命名约定。

示例

Compare the memory used by all geometries in a Massachusetts town dataset with the subset for Boston.

Code

```
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) * 100.0
         / sum(ST_MemSize(geom)) AS numeric(10, 2)
       ) AS perbos
FROM towns;
```

Output

```
totgeomsum | bossum | perbos
-----+-----+-----
1522 kB    | 30 kB  | 1.99
(1 row)
```

Return the memory size of a CircularString in bytes.

Code

```
SELECT ST_MemSize('CIRCULARSTRING(220268 150415,220227 150505,220227 150406)::geometry');
```

Output

```
st_memsize
-----
80
(1 row)
```

Calculate the percentage of a table's total size used by geometry data.

Code

```
SELECT pg_total_relation_size('public.neighborhoods') AS fulltable_size,
       sum(ST_MemSize(geom)) AS geomsizes,
       sum(ST_MemSize(geom)) * 100.0
       / pg_total_relation_size('public.neighborhoods') AS pergeom
FROM neighborhoods;
```

Output

```
fulltable_size | geomsizes | pergeom
-----+-----+-----
262144 | 96238 | 36.71188354492187500000
(1 row)
```

7.4.28 ST_NDims

ST_NDims — 返回几何体的坐度。

Synopsis

integer **ST_NDims**(geometry g1);

描述

返回几何体的坐度。PostGIS 支持 2 - (x,y)、3 - (x,y,z) 或 2D 量 - x,y,m 和 4 - 3D 量空 x,y,z,m

 函数支持 3d 并且不会失 z-index。

示例

Code

```
SELECT ST_NDims('POINT(1 1)') As d2point,
ST_NDims('POINT(1 1 2)') As d3point,
ST_NDims('POINTM(1 1 0.5)') As d2pointm;
```

Output

d2point	d3point	d2pointm
2	3	3

相关信息

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

7.4.29 ST_NPoints

ST_NPoints — 返回几何形中的点数（点）。

Synopsis

integer **ST_NPoints**(geometry g1);

描述

返回几何形中的点数。适用于所有几何形状。

增：2.0.0 引入了多面体曲面的支持。



Note
在 1.3.4 之前，此函数在与包含曲的几何形一起使用崩。此已在 1.3.4 及更高版本中得到正

-  函数支持 3d 并且不会失 z-index。
-  此方法支持形字符串和曲。
-  函数支持多面体曲面。

示例

Count the points in a 2D LineString.

Code

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

Output

```
st_npoints
-----
          4
(1 row)
```

Count the points in a 3D LineString.

Code

```
SELECT ST_NPoints('LINESTRING(77.29 29.07 1,77.42 29.26 0,77.27 29.31 -1,77.29 29.07 3)');
```

Output

```
st_npoints
-----
          4
(1 row)
```

相关信息

[ST_NumPoints](#)

7.4.30 ST_NRings

ST_NRings — 返回多边形几何中的环数。

Synopsis

```
integer ST_NRings(geometry geomA);
```

描述

如果几何图形是多边形或多多边形，它返回环数。与 NumInteriorRings 不同，它也算外环。



该函数支持 3d 并且不会丢失 z-index。



此方法支持图形字符串和曲线。

示例

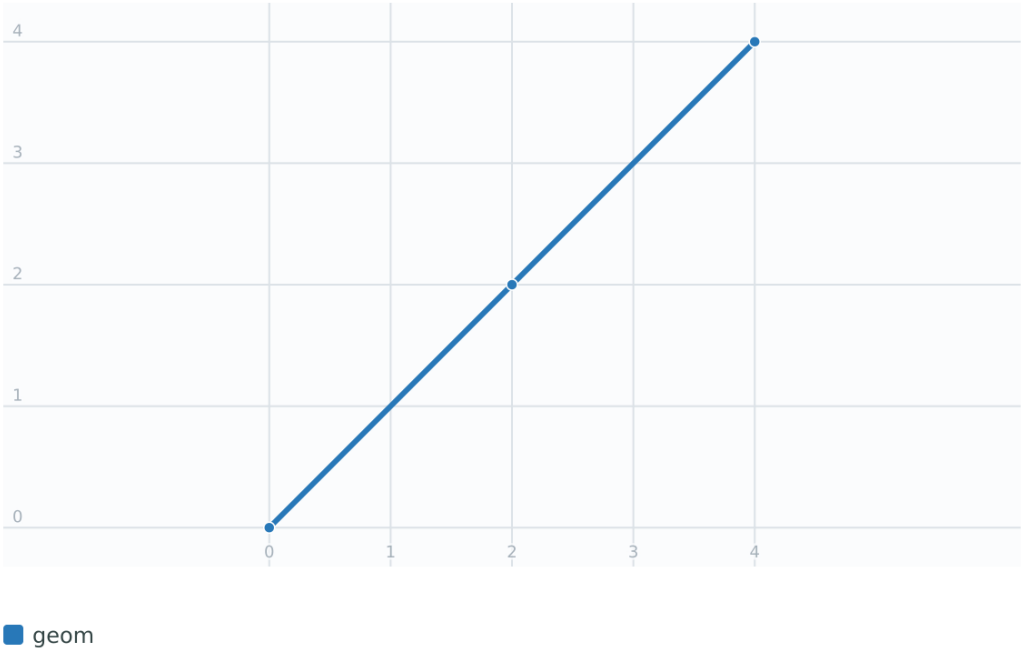
Code

```
SELECT ST_NRings(geom) As Nrings, ST_NumInteriorRings(geom) As ninterrings
FROM (SELECT 'POLYGON((1 2,3 4,5 6,1 2))'::geometry As geom) As foo;
```

Output

nrings	ninterrings
1	0
(1 row)	

Figure



相关信息

ST_NumInteriorRings

7.4.31 ST_NumGeometries

ST_NumGeometries — 返回几何集合中的元素数量。

Synopsis

integer **ST_NumGeometries**(geometry geom);

描述

该函数返回几何集合（GEOMETRYCOLLECTION 或 MULTI*）中的元素数量。对于非空的原子几何体，它返回 1。对于空的几何体，它返回 0。

新增功能：引入了 2.0.0 对多面体曲面、三角形和三角网的支持。

更改：2.0.0 在之前的版本中，如果几何类型不是 collection/MULTI 类型，它会返回 NULL。2.0.0 在对空几何类型返回 1，例如 POLYGON、LINESTRING、POINT。

Changed: 3.6.0 TIN and PolyhedralSurface are treated as unitary geometries and return 1; use ST_NumPatches to count faces.

- ✓ 方法支持了 SQL/MM 规范。SQL-MM 3: 9.1.4
- ✓ 函数支持 3d 并且不会丢失 z-index。
- ✓ 函数支持多面体曲面。
- ✓ 此函数支持三角形和不规则三角网面 (TIN)。

示例

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

Code

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

Output

```
st_numgeometries
-----
1
(1 row)
```

A MultiPoint counts as one geometry within a GeometryCollection.

Code

```
SELECT ST_NumGeometries('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)))');
```

Output

```
st_numgeometries
-----
3
(1 row)
```

相关信息

[ST_GeometryN](#), [ST_Multi](#)

7.4.32 ST_NumInteriorRings

ST_NumInteriorRings — 返回多边形的内环（孔）数。

Synopsis

integer **ST_NumInteriorRings**(geometry a_polygon);

描述

返回多边形几何体的内环数。如果几何体不是多边形，返回 NULL。



该方法符合了 SQL/MM 规范。SQL-MM 3: 8.2.5

更改：2.0.0 - 在之前的版本中，它允许 MULTIPOLYGON，返回第一个 POLYGON 的内环数量。

示例

Count the interior rings of a Polygon.

Code

```
SELECT gid, field1, field2, ST_NumInteriorRings(geom) AS numholes
FROM sometable;
```

For a MultiPolygon, dump its component Polygons and sum their interior ring counts.

Code

```
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;
```

相关信息

[ST_NumInteriorRing](#), [ST_InteriorRingN](#)

7.4.33 ST_NumInteriorRing

`ST_NumInteriorRing` — Returns the number of interior rings (holes) of a Polygon. Alias for `ST_NumInteriorRings`.

Synopsis

integer **ST_NumInteriorRing**(geometry a_polygon);

描述

This function is an alias for [ST_NumInteriorRings](#).

可用性：1.2.2

相关信息

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

7.4.34 ST_NumPatches

ST_NumPatches — Returns the number of faces in a PolyhedralSurface or TIN.

Synopsis

```
integer ST_NumPatches(geometry g1);
```

描述

Returns the number of faces in a PolyhedralSurface or TIN. Returns NULL for other geometries. Unlike [ST_NumGeometries](#), this function counts individual faces.

可用性: 2.0.0

- ✓ 函数支持 3d 并且不会丢失 z-index。
- ✓ 此方法符合了 [SQL 1.1 的 OGC 功能规范](#)。
- ✓ 方法符合了 SQL/MM 规范。SQL-MM ISO/IEC 13249-3: 8.5
- ✓ 函数支持多面体曲面。

示例

Code

```
SELECT ST_NumPatches('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)) )');
```

Output

```
st_numpatches
-----
              6
(1 row)
```

相关信息

[ST_GeomFromEWKT](#), [ST_NumGeometries](#), [ST_PatchN](#)

7.4.35 ST_NumPoints

ST_NumPoints — 返回 LineString 或 CircularString 中的点数。

Synopsis

```
integer ST_NumPoints(geometry g1);
```

描述

返回 `ST_LineString` 或 `ST_CircularString` 中的点数。1.4 之前的版本适用于范围定义的字符串。从 1.4 开始，它是 `ST_NPoints` 的别名，它不返回字符串的点数，考虑使用 `ST_NPoints`，它是多用途的并且适用于多几何类型。

- ✔ 此方法实现了 SQL 1.1 的 OGC 功能规范。
- ✔ 该方法实现了 SQL/MM 规范。SQL-MM 3: 7.2.4

示例

Code

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

Output

```
st_numpoints
-----
4
(1 row)
```

相关信息

[ST_NPoints](#)

7.4.36 ST_PatchN

`ST_PatchN` — Returns the Nth face of a PolyhedralSurface or TIN.

Synopsis

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

描述

Returns the 1-based Nth face of a PolyhedralSurface or TIN. Returns NULL for other geometries or an out-of-range index. Use this function to access faces, because [ST_GeometryN](#) treats PolyhedralSurface and TIN as unitary geometries.



Note
索引从 1 开始。



Note
如果要提取几何图形的所有元素，[ST_Dump](#) 效率更高。

可用性: 2.0.0

- ✓ 方法符合了 SQL/MM 规范。SQL-MM ISO/IEC 13249-3: 8.5
- ✓ 函数支持 3d 并且不会丢失 z-index。
- ✓ 函数支持多面体曲面。
- ✓ 此函数支持三角形和不规则三角网面 (TIN)。

示例

Extract the 2nd face of the polyhedral surface.

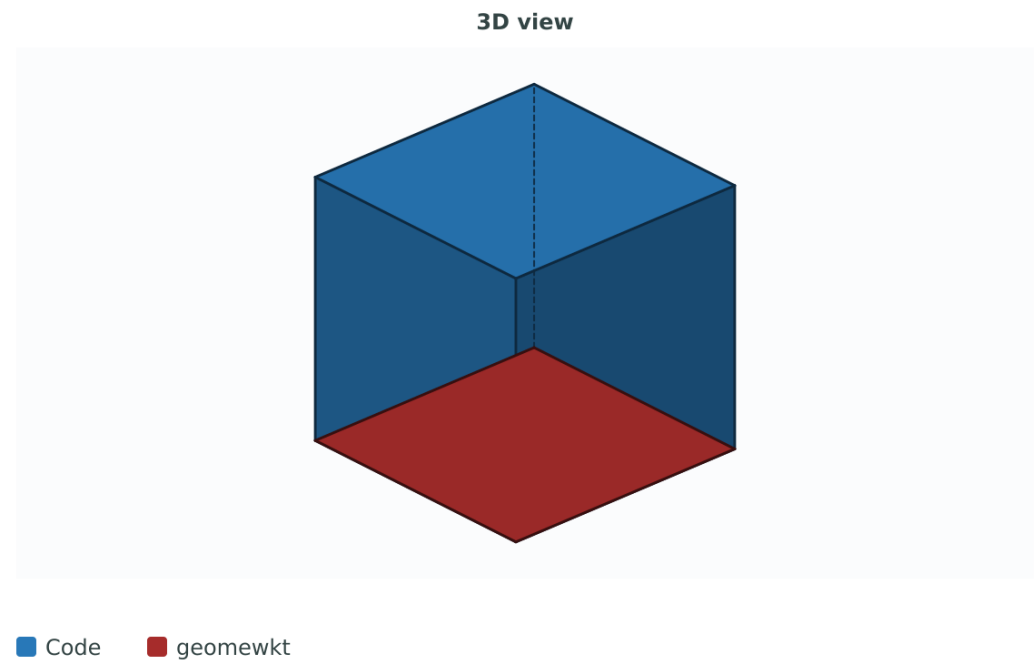
Code

```
SELECT ST_PatchN(geom, 2) As geomewkt
FROM (
VALUES ( '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
foo(geom);
```

Output

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

Figure



相关信息

[ST_AsEWKT](#), [ST_GeomFromEWKT](#), [ST_Dump](#), [ST_GeometryN](#), [ST_NumGeometries](#), [ST_NumPatches](#)

7.4.37 ST_PointN

ST_PointN — 返回几何图形中第一个线串或图形线串中的第 N 个点。

Synopsis

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

描述

返回几何图形中第 n 个线串或图形线串中的第 N 个点。索引从 LineString 末尾开始向后计数，因此 -1 是最后一个点。如果几何图形中没有线串，则返回 NULL。



Note

Index is 1-based as for OGC specs since version 0.8.0. Backward indexing (negative index) is not in OGC. Previous versions implemented this as 0-based instead.



Note

如果要提取 MultiLineString 中每个 LineString 的第 N 个点，可与 ST_Dump 合使用



此方法符合了 SQL 1.1 的 OGC 功能规范。



该方法符合了 SQL/MM 规范。SQL-MM 3: 7.2.5, 7.3.5



该函数支持 3d 并且不会丢失 z-index。



此方法支持图形字符串和曲线。

更改：2.0.0 不再适用于多个几何 multilinestrings。在旧版本的 PostGIS 中——该行 multilinestring 可以与此函数很好地配合并返回起点。在 2.0.0 中，它像任何其他 multilinestring 一样只返回 NULL。

更改：2.3.0：索引可用（-1 是最后一点）

示例

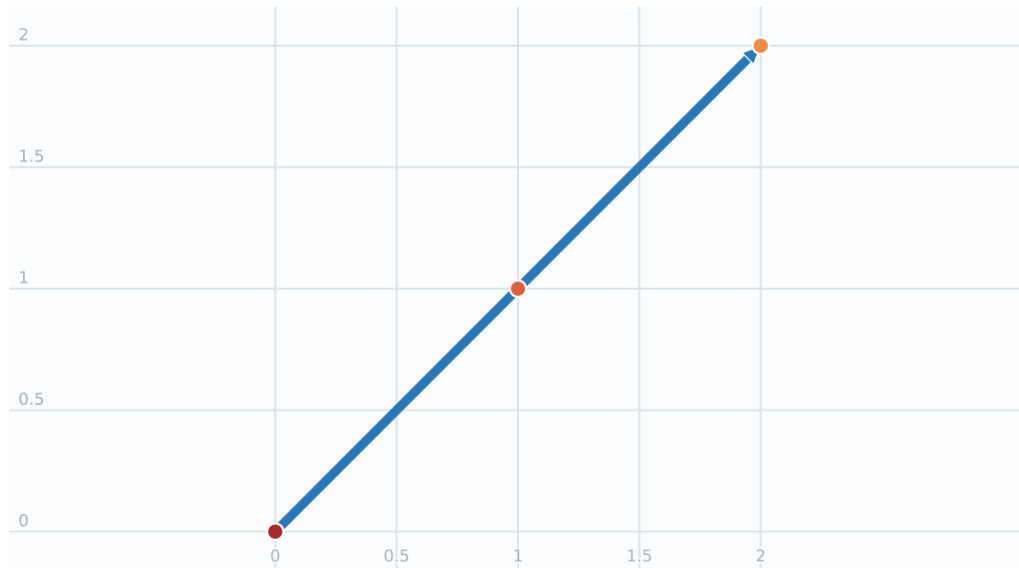
This example extracts all POINT geometries from a LineString.

Code

```
SELECT ST_PointN(
    column1,
    generate_series(1, ST_NPoints(column1))
)
FROM ( VALUES ( 'LINESTRING(0 0,1 1,2 2)::geometry' ) AS foo;
```

Output

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

Figure

■ Code ■ Output 1 ■ Output 2 ■ Output 3

This example uses a CircularString.

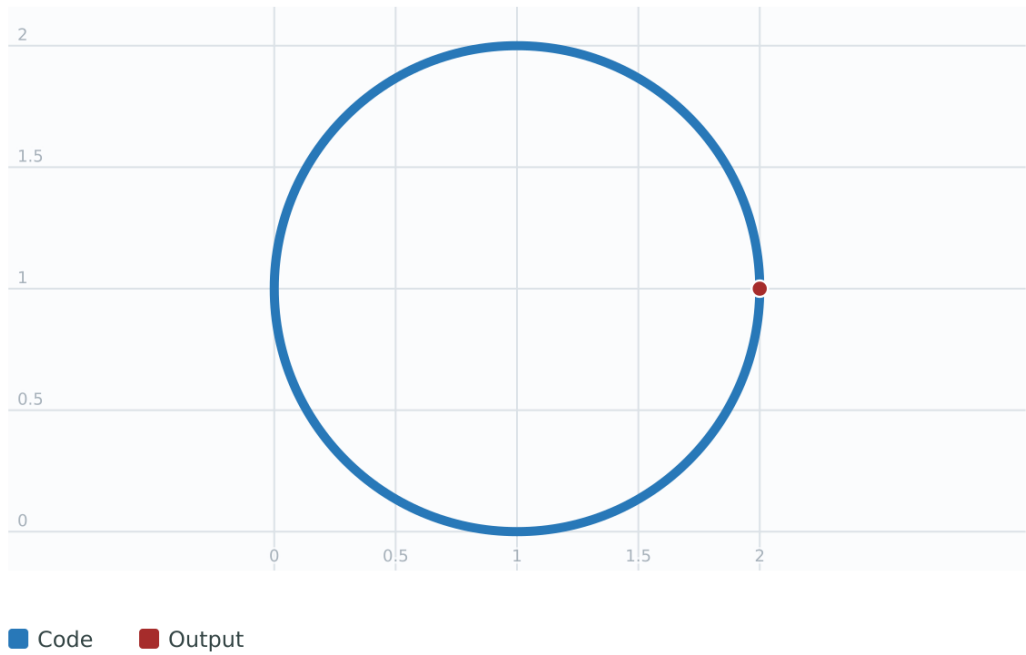
Code

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

Output

```
POINT(3 2)
```

Figure



The index is 1-based. A negative index counts backward from the end of the LineString.

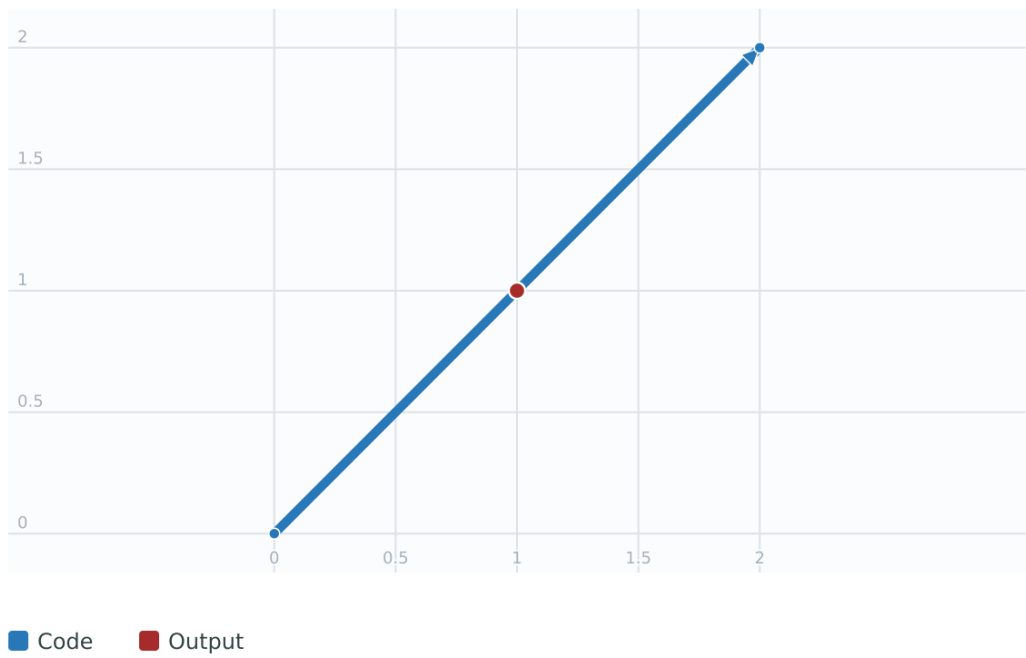
Code

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

Output

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

Figure



相关信息

[ST_NPoints](#)

7.4.38 ST_Points

ST_Points — 返回包含几何坐标的 MultiPoint。

Synopsis

```
geometry ST_Points( geometry geom );
```

描述

返回包含几何体所有坐标的 MultiPoint。保留重复点，包括几何形状的起点和点。（如果需要，可以通过[ST_RemoveRepeatedPoints](#)来去除重复点）。

要取有关父几何体中每个坐标的位置的信息，使用 [ST_DumpPoints](#)。

如果存在，保留 M 和 Z 。



此方法支持形字符串和曲线。



函数支持 3d 并且不会失 z-index。

可用性 : 2.3.0

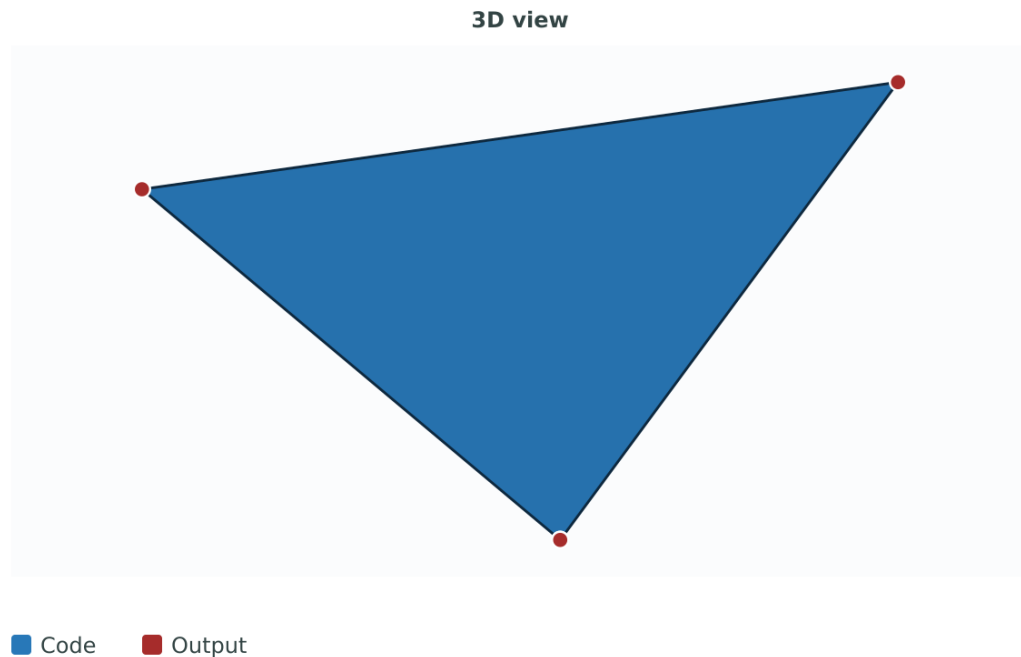
示例

Code

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

Output

```
MULTIPOINT Z ((30 10 4),(10 30 5),(40 40 6),(30 10 4))
```

Figure

相关信息

[ST_RemoveRepeatedPoints](#), [ST_DumpPoints](#)

7.4.39 ST_StartPoint

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

Synopsis

geometry **ST_StartPoint**(geometry geomA);

描述

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



该方法符合了 SQL/MM 规范。SQL-MM 3: 7.1.3



该函数支持 3d 并且不会丢失 z-index。



此方法支持几何字符串和曲线。

新增：3.2.0 返回所有几何图形的点。如果输入不是 LineString，则先前的行将返回 NULL。

更改：2.0.0 不再适用于几何体 MultiLineStrings。在旧版本的 PostGIS 中，该行 MultiLineString 可以与此函数很好地配合并返回起点。在 2.0.0 中，它像任何其他 MultiLineString 一样只返回 NULL。旧的行是一个未定义的功能，但是那些假设将数据存入 LINESTRING 的人可能会在 2.0.0 中遇到一些返回 NULL 的情况。

示例

LineString 起点

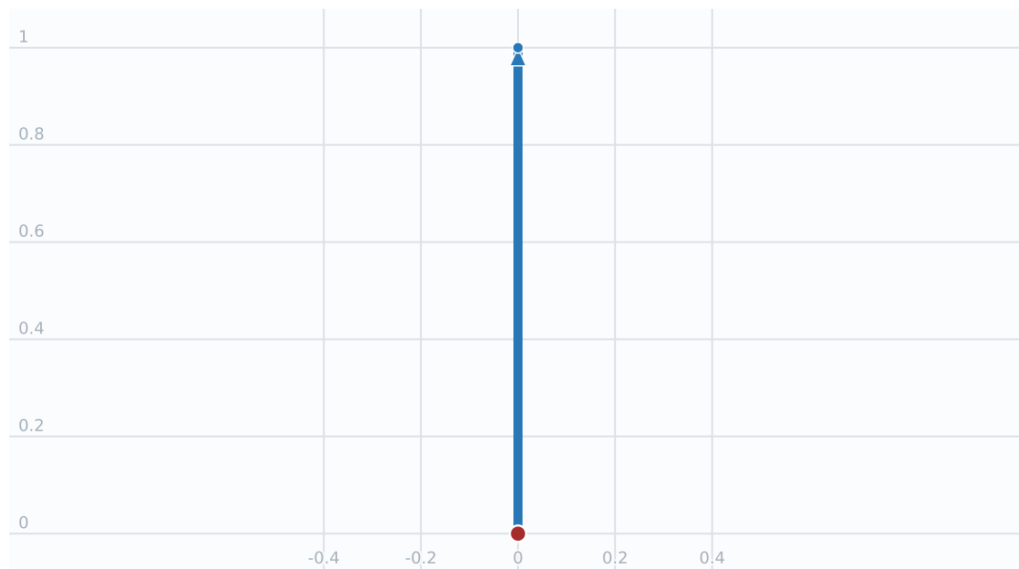
Code

```
SELECT ST_StartPoint('LINESTRING(0 1,0 2)::geometry');
```

Output

```
POINT(0 1)
```

Figure



■ Code ■ Output

Start point of a Point is the Point itself

Code

```
SELECT ST_StartPoint('POINT(0 1)::geometry');
```

Output

```
POINT(0 1)
```

Figure



3D LineString 起点

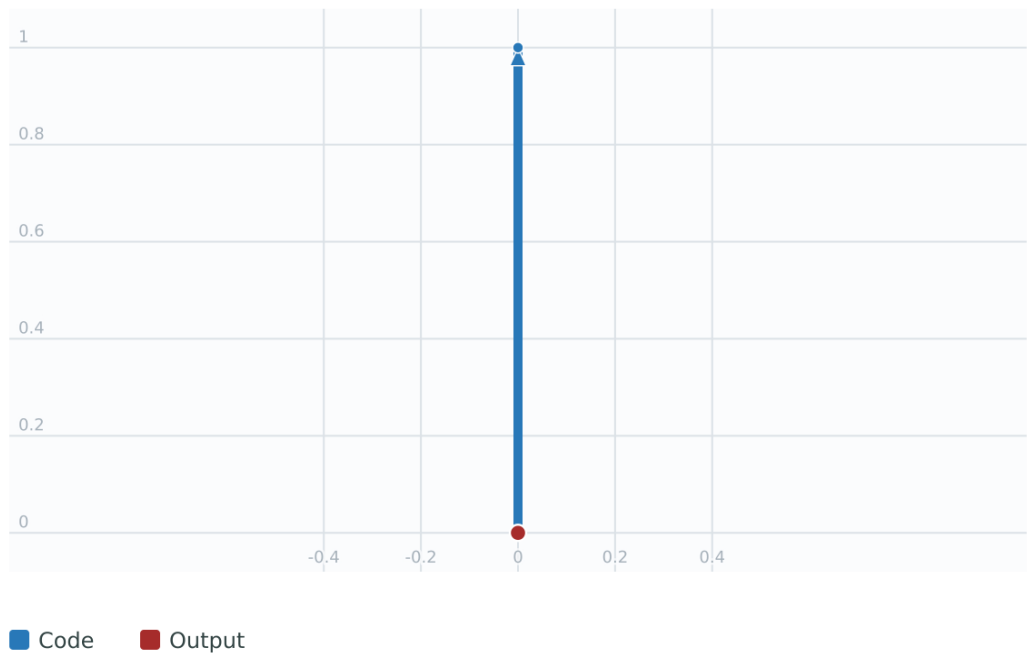
Code

```
SELECT ST_StartPoint('LINESTRING(0 1 1,0 2 2)::geometry');
```

Output

```
POINT(0 1 1)
```

Figure



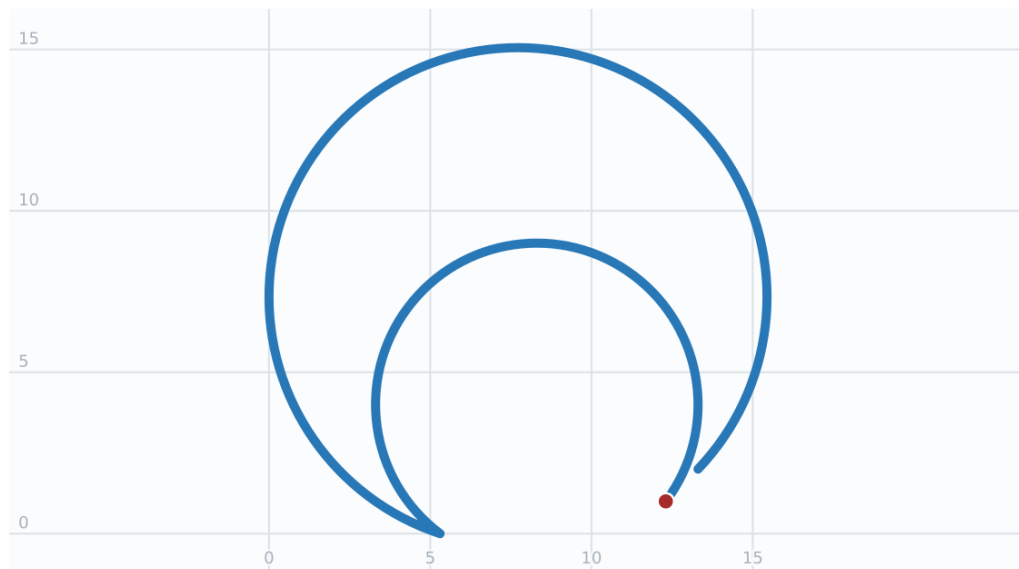
CircularString 起点

Code

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

Output

```
POINT(5 2)
```

Figure

■ Code ■ Output

Start point of a NURBS Curve

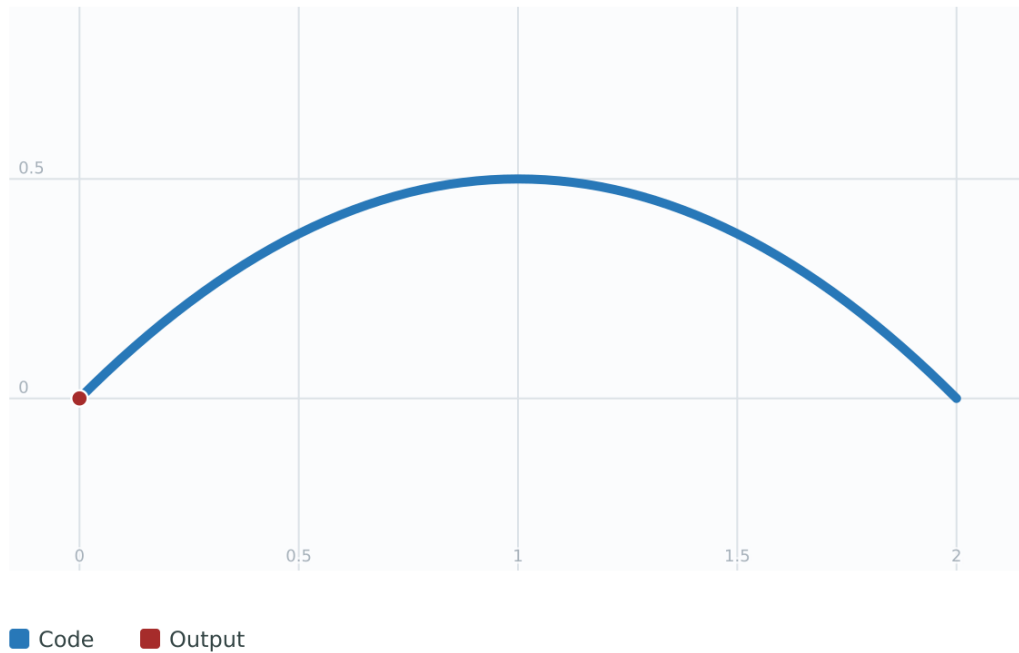
Code

```
SELECT ST_StartPoint('NURBS CURVE(2, (0 0, 1 1, 2 0))::geometry');
```

Output

```
POINT(0 0)
```

Figure



Build edge node references from the start and end points of a LineString table. The example uses exact endpoint equality, so linework with nearly coincident endpoints should be snapped or otherwise cleaned first.

Code

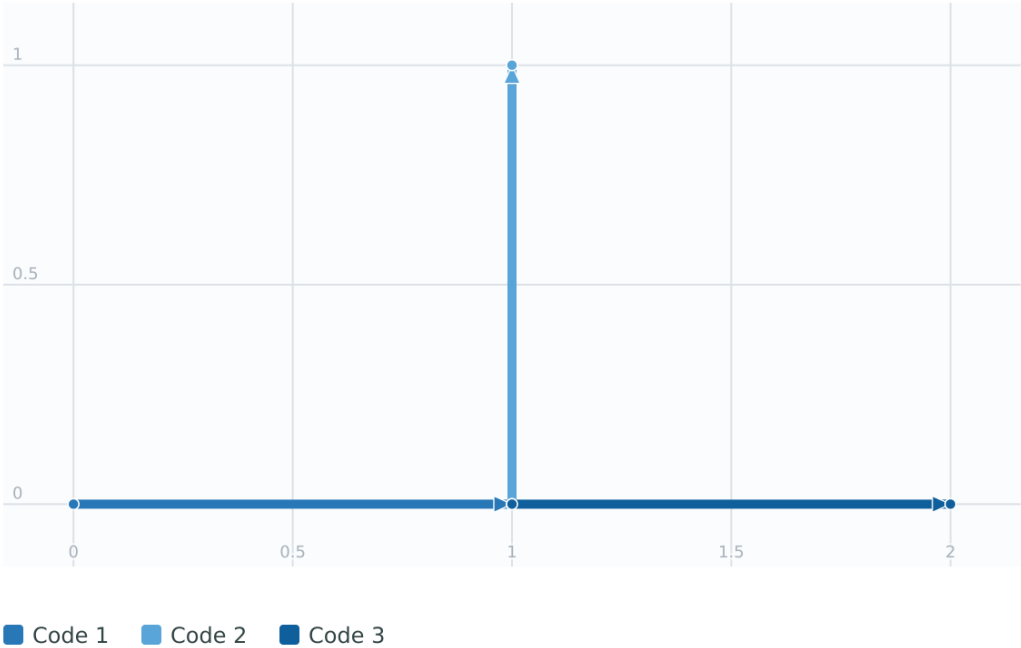
```
WITH roads(road_id, geom) AS (
  VALUES
    (1, 'LINESTRING(0 0, 1 0)::geometry'),
    (2, 'LINESTRING(1 0, 1 1)::geometry'),
    (3, 'LINESTRING(1 0, 2 0)::geometry')
),
endpoints AS (
  SELECT road_id, 'from' AS end_name, ST_StartPoint(geom) AS geom,
         ST_AsEWKB(ST_StartPoint(geom)) AS endpoint_key
  FROM roads
  UNION ALL
  SELECT road_id, 'to' AS end_name, ST_EndPoint(geom) AS geom,
         ST_AsEWKB(ST_EndPoint(geom)) AS endpoint_key
  FROM roads
),
nodes AS (
  SELECT dense_rank() OVER (ORDER BY ST_X(geom), ST_Y(geom), endpoint_key) AS node_id,
         endpoint_key
  FROM (
    SELECT DISTINCT ON (endpoint_key) endpoint_key, geom
    FROM endpoints
    ORDER BY endpoint_key
  ) AS distinct_endpoints
)
SELECT e.road_id,
       max(n.node_id) FILTER (WHERE e.end_name = 'from') AS from_node,
       max(n.node_id) FILTER (WHERE e.end_name = 'to') AS to_node
FROM endpoints AS e
JOIN nodes AS n USING (endpoint_key)
GROUP BY e.road_id
```

```
ORDER BY e.road_id;
```

Output

road_id	from_node	to_node
1	1	2
2	2	3
3	2	4

Figure



相关信息

[ST_EndPoint](#), [ST_PointN](#)

7.4.40 ST_StartM

ST_StartM — Returns the M coordinate of the first point of a LineString, CircularLineString, or NURBSCurve.

Synopsis

```
float8 ST_StartM(geometry geom);
```

描述

Returns the M coordinate of the first point of a **LINestring**, **CIRCULARLINestring**, or **NURBSCURVE** geometry. Returns NULL if the input geometry does not have an M dimension or is not a **LINestring**, **CIRCULARLINestring**, or **NURBSCURVE**.

 此方法支持☐形字符串和曲☐。

示例

Start M value of a LineString

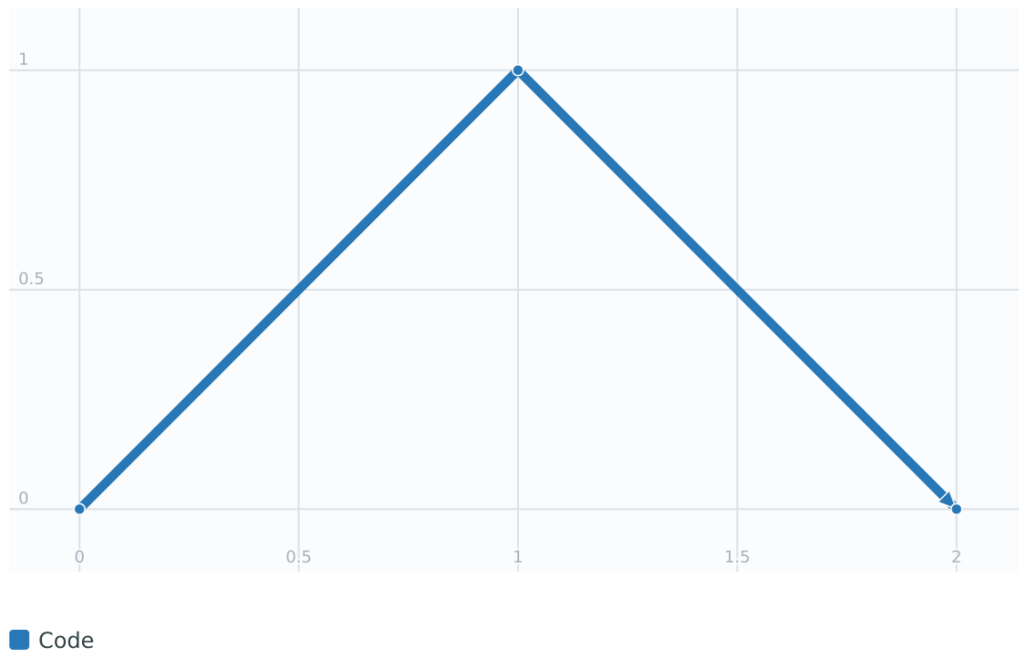
| Code

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

| Output

10

| Figure



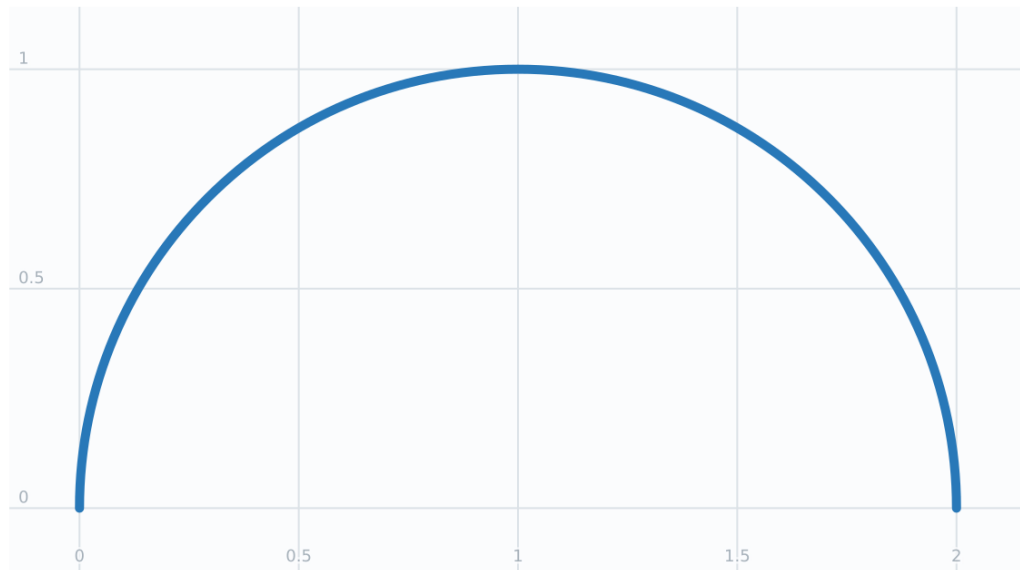
Start M value of a CircularString

| Code

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

| Output

5

Figure

■ Code

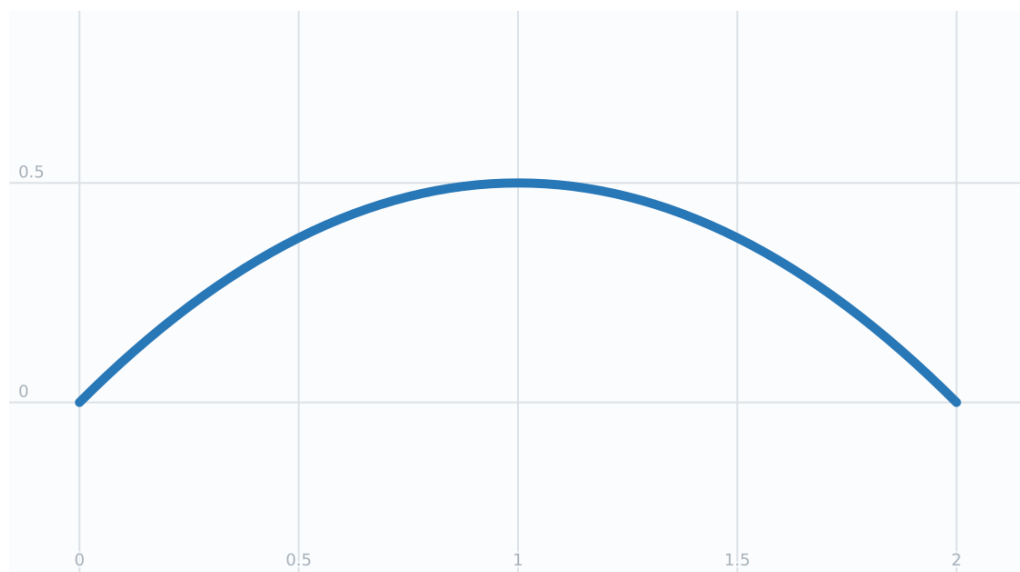
Start M value of a NURBSCurve

Code

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

Output

```
100
```

Figure

■ Code

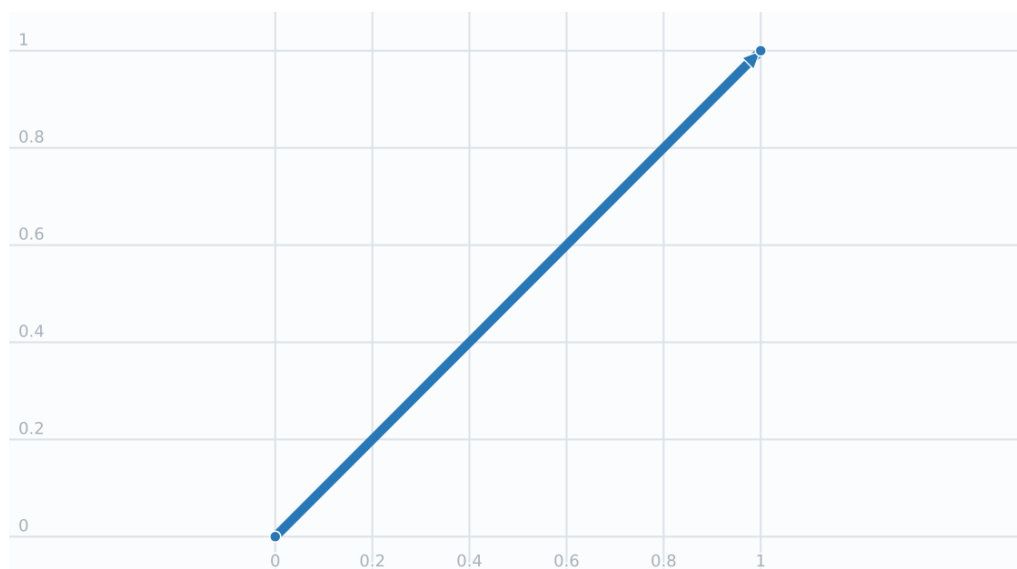
Geometry without M dimension returns NULL

Code

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

Output

```
t
```

Figure

■ Code

相关信息

[ST_EndM](#), [ST_SetStartM](#), [ST_StartPoint](#)

7.4.41 ST_EndM

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

Synopsis

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

描述

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



此方法支持2D形字符串和曲线。

示例

End M value of a LineString

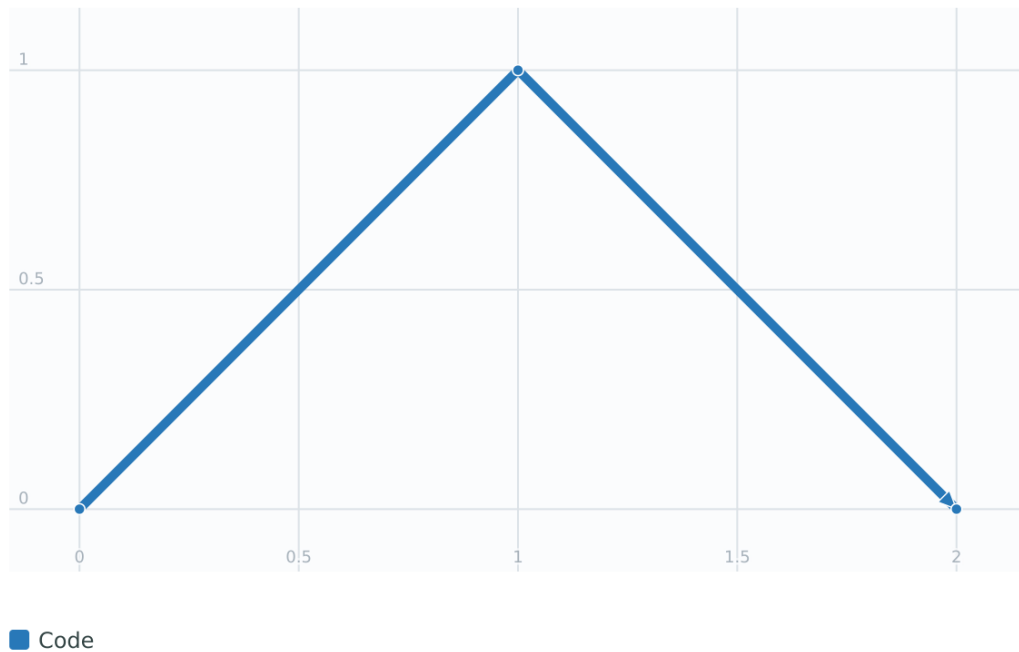
| Code

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

| Output

30

| Figure



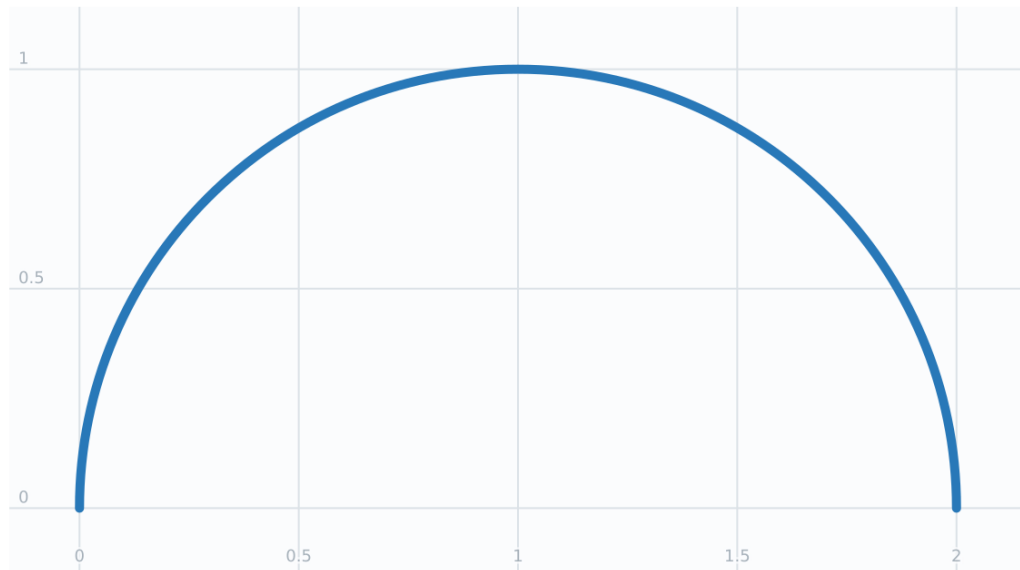
End M value of a CircularString

| Code

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

| Output

15

Figure

■ Code

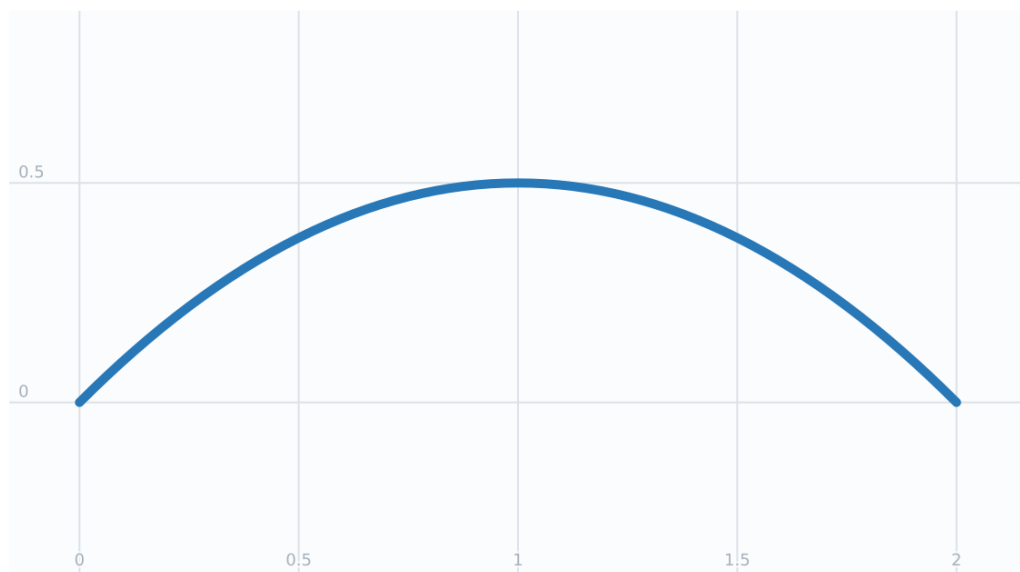
End M value of a NURBSCurve

Code

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

Output

```
300
```

Figure

■ Code

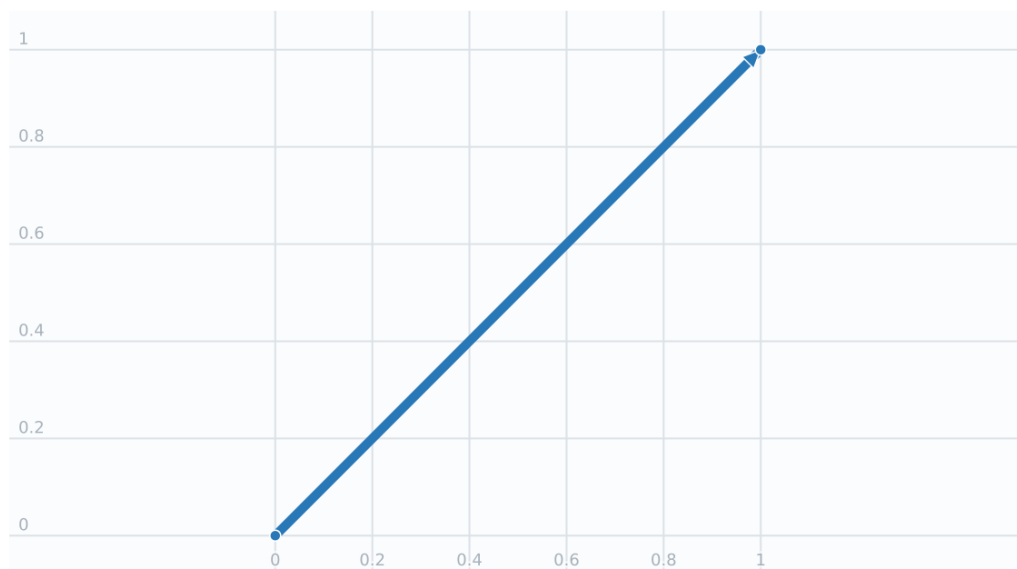
Geometry without M dimension returns NULL

Code

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

Output

```
t
```

Figure

Code

相关信息

[ST_StartM](#), [ST_SetEndM](#), [ST_EndPoint](#)

7.4.42 ST_SetStartM

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

Synopsis

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

描述

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



此方法支持☐形字符串和曲☐。

示例

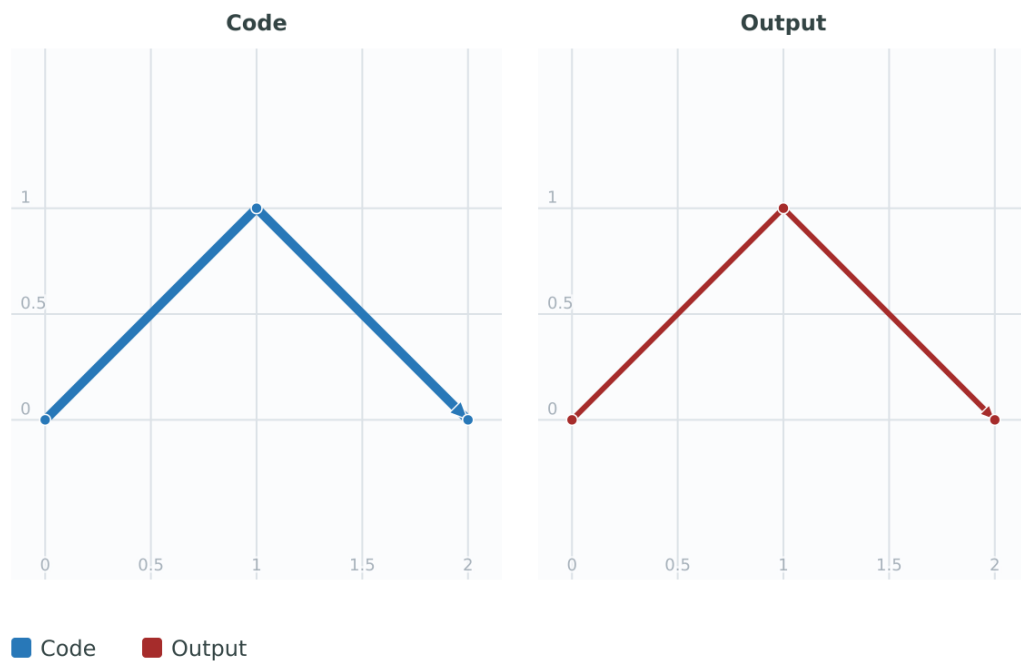
Set start M value of a LineString

Code

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

Output

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

Figure

Add M dimension and set start M value

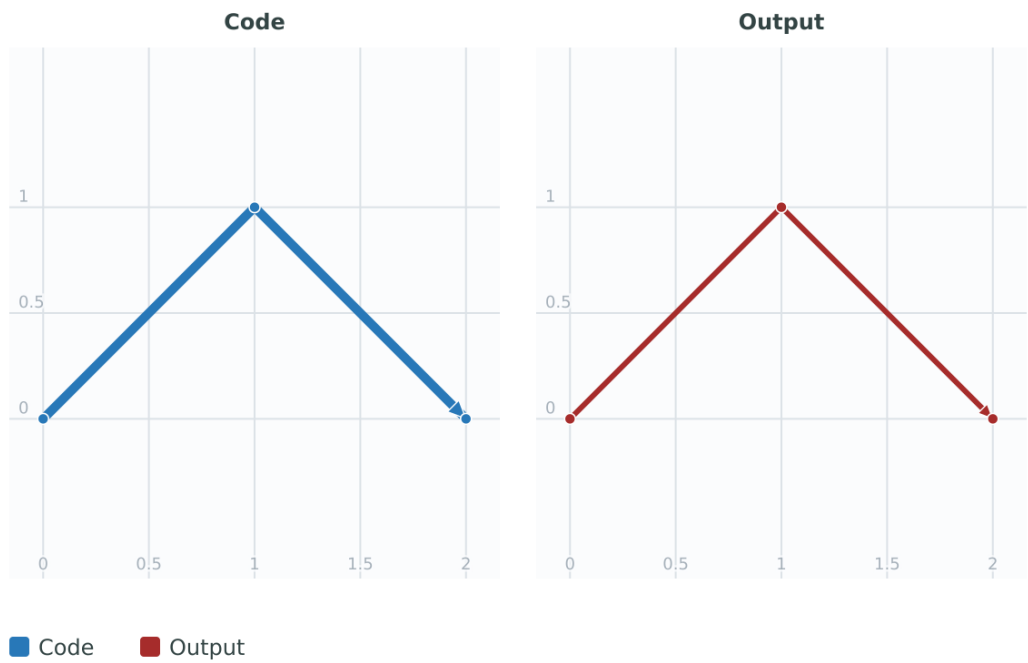
Code

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

Output

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

Figure



Set start M value of a CircularString

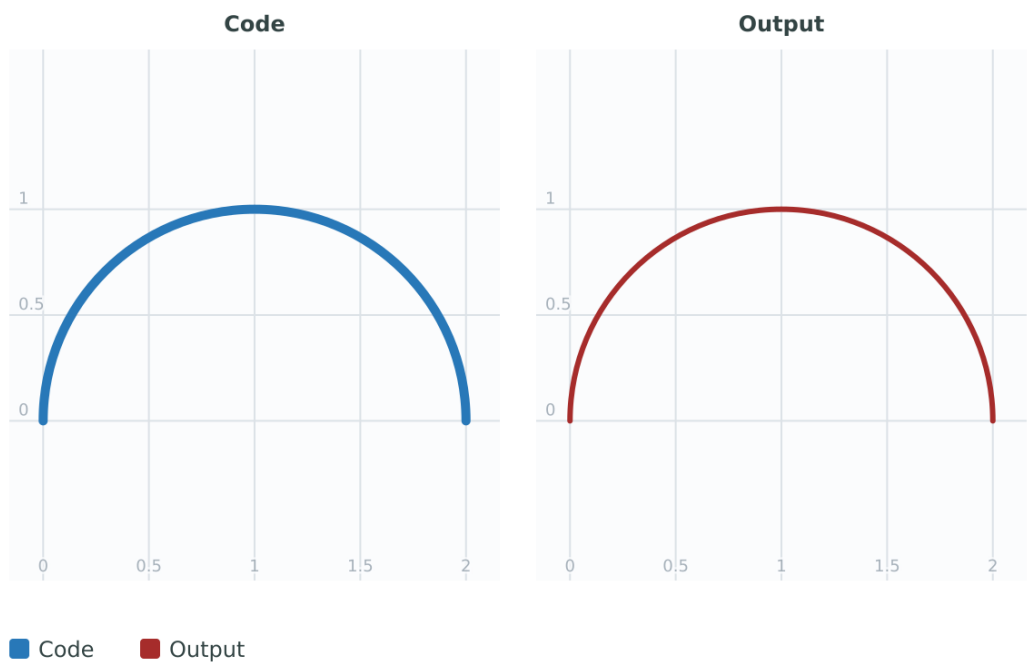
Code

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

Output

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

Figure



Set start M value of a NURBSCurve

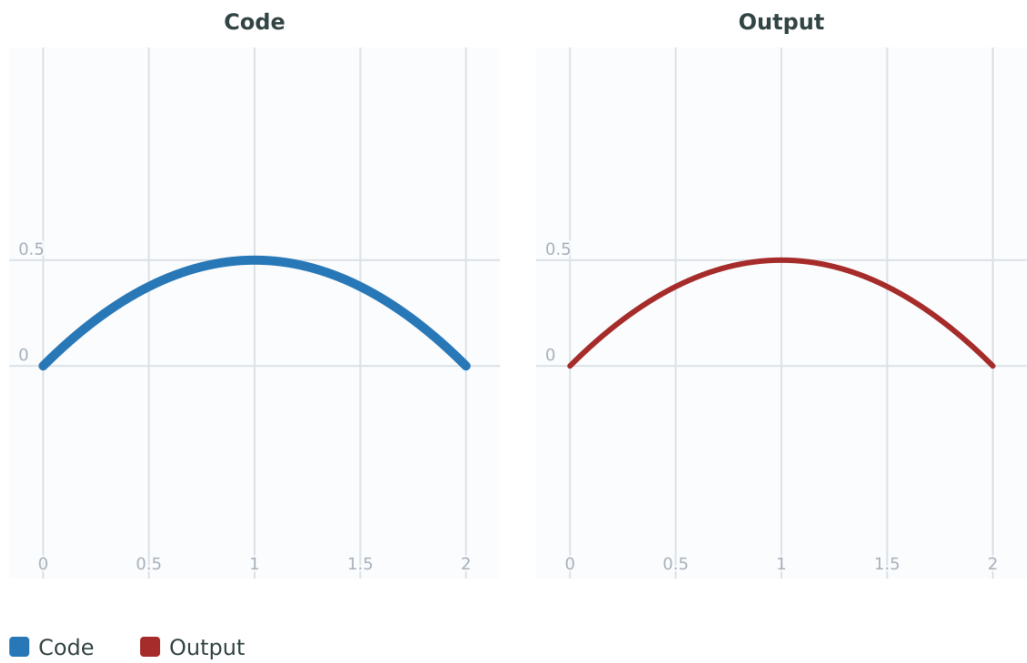
Code

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

Output

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

Figure



相关信息

ST_SetEndM, ST_StartM, ST_StartPoint

7.4.43 ST_SetEndM

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

Synopsis

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

描述

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

 此方法支持☐形字符串和曲☐。

示例

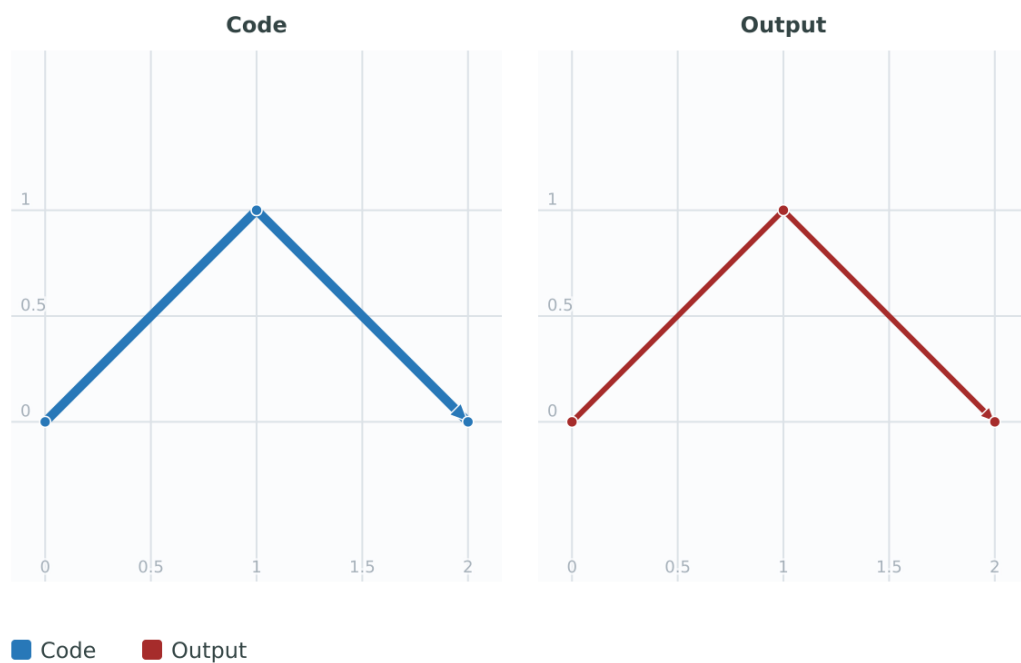
Set end M value of a LineString

Code

```
SELECT ST_SetEndM('LINESTRING M(0 0 10,1 1 20,2 0 30)::geometry, 999);
```

Output

```
LINESTRING M (0 0 10,1 1 20,2 0 999)
```

Figure

Add M dimension and set end M value

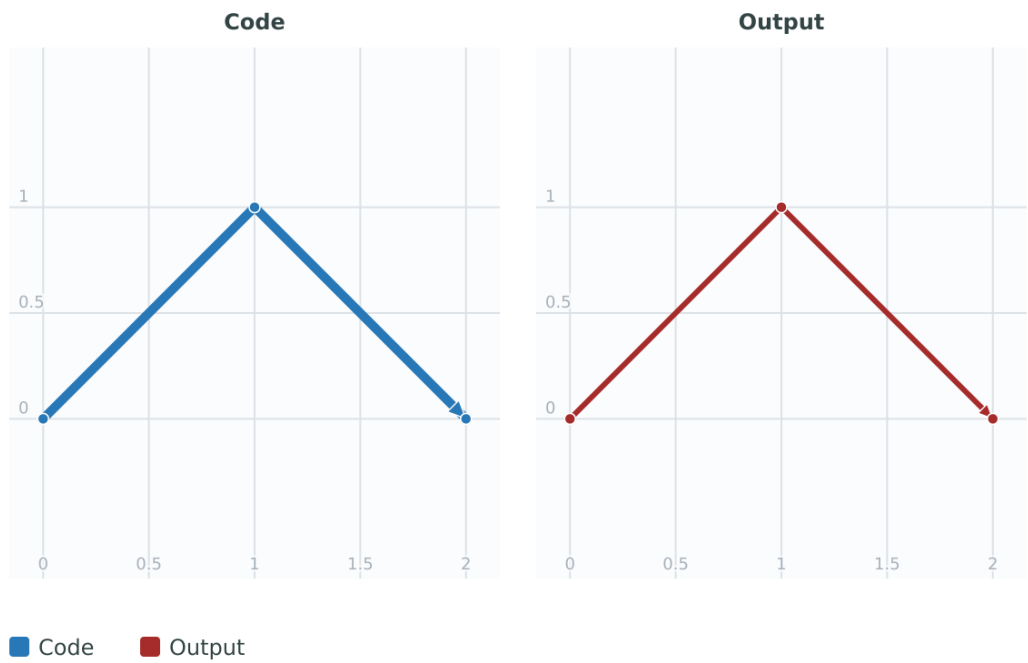
Code

```
SELECT ST_SetEndM('LINESTRING(0 0,1 1,2 0)::geometry, 75);
```

Output

```
LINESTRING M (0 0 0,1 1 0,2 0 75)
```

Figure



Set end M value of a CircularString

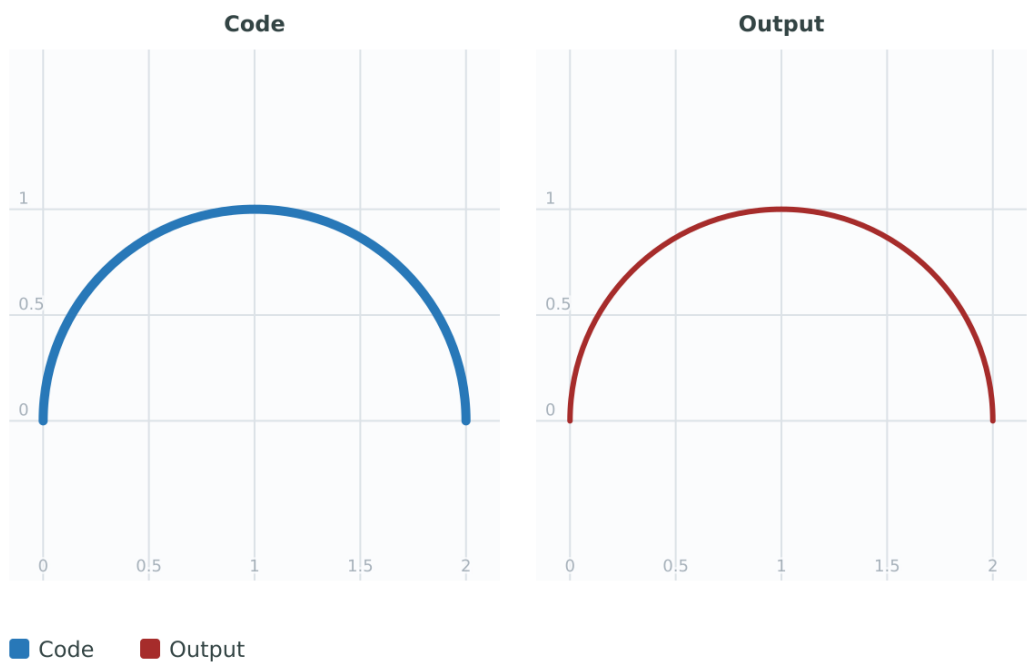
Code

```
SELECT ST_SetEndM('CIRCULARSTRING M(0 0 5,1 1 10,2 0 15)::geometry, 88);
```

Output

```
CIRCULARSTRING M (0 0 5,1 1 10,2 0 88)
```

Figure



Set end M value of a NURBSCurve

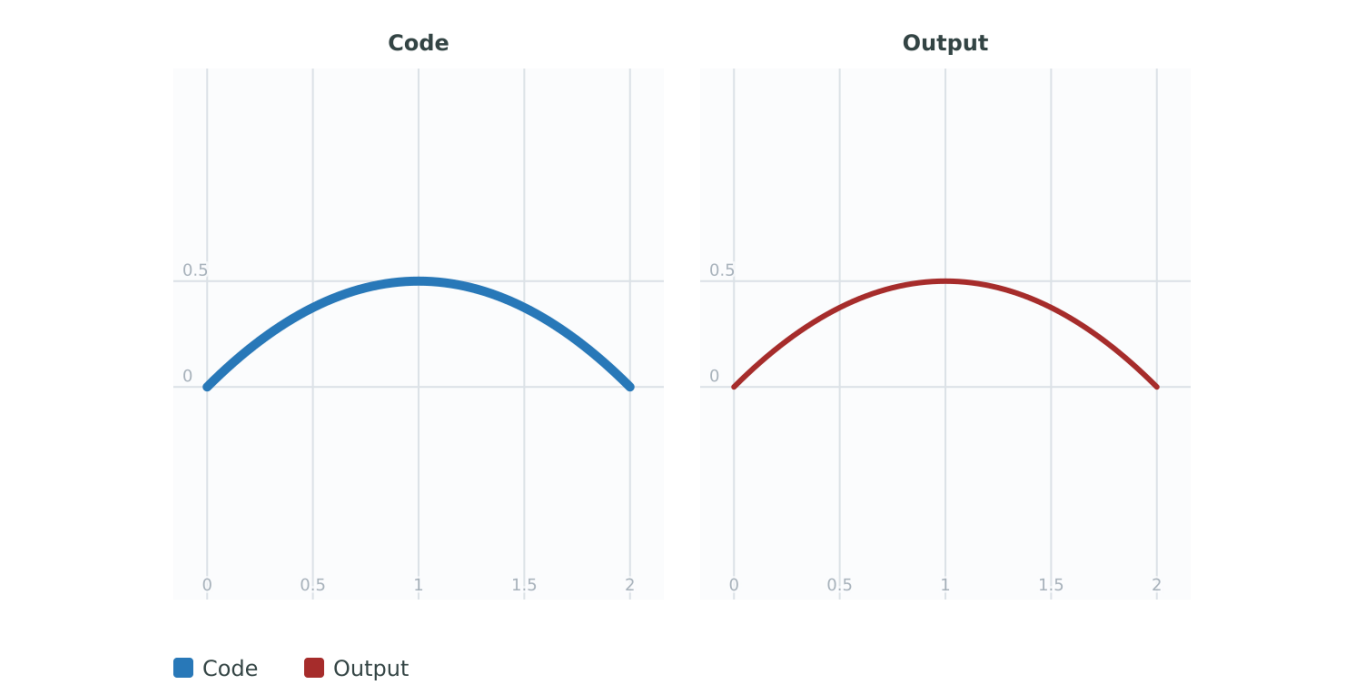
Code

```
SELECT ST_SetEndM('NURBSCURVE M(2, (0 0 100, 1 1 200, 2 0 300))'::geometry, 999);
```

Output

```
NURBSCURVE M (DEGREE 2,CONTROLPOINTS M (NURBSPOINT(WEIGHTEDPOINT M (0 0 100),WEIGHT 1), NURBSPOINT(WEIGHTEDPOINT M (1 1 200),WEIGHT 1), NURBSPOINT(WEIGHTEDPOINT M (2 0 999),WEIGHT 1)),KNOTS (KNOT(0,3), KNOT(1,3)))
```

Figure



相关信息

ST_SetStartM, ST_EndM, ST_EndPoint

7.4.44 ST_Summary

ST_Summary — 返回几何内容的文本摘要。

Synopsis

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

描述

返回几何内容的文本摘要。
几何 $\Box$ 型后方括号中 $\Box$ 示的 $\Box$ 志具有以下含 $\Box$ ：

- M：具有 M $\Box$

- Z : 具有 Z 值
- B : 有一个边界框
- G : 大地坐标系（地理）
- S : 具有空参考系

- ✔ 此方法支持字符串和曲线。
- ✔ 函数支持多面体曲面。
- ✔ 此函数支持三角形和不规则三角网面 (TIN)。

可用性 : 1.2.2

增加 : 在 2.0.0 中添加了地理支持

增加 : 添加了 S 标志以指示其是否具有 2.1.0 空参考系

增加 : 添加了 2.2.0 TIN 和曲线的支持

示例

Code

```
SELECT ST_Summary('LINESTRING(0 0,1 1)::geometry') as geom,
ST_Summary('POLYGON((0 0,1 1,1 2,1 1,0 0))::geography') geog;
```

Output

geom	geog
-----+-----	
LineString[B] with 2 points	Polygon[BGS] with 1 rings
	ring 0 has 5 points
	:
(1 row)	

Code

```
SELECT ST_Summary('LINESTRING(0 0 1,1 1 1)::geography') As geog_line,
ST_Summary('SRID=4326;POLYGON((0 0 1,1 1 2,1 2 3,1 1 1,0 0 1))::geometry') As geom_poly;
```

Output

geog_line	geom_poly
-----+-----	
LineString[ZBGS] with 2 points	Polygon[ZBS] with 1 rings
	: ring 0 has 5 points
	:
(1 row)	

相关信息

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

7.4.45 ST_X

ST_X — 返回点的 X 坐标。

Synopsis

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

描述

返回点的 X 坐标，如果无效，则返回 NULL。输入必须是点。



Note

要获取几何图形的最小和最大 X 值，请使用 `ST_XMin` 和 `ST_XMax` 函数。



该方法符合了 SQL/MM 规范。SQL-MM 3: 6.1.3



该函数支持 3d 并且不会丢失 z-index。

示例

Code

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

Output

```
1
```

Code

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

Output

```
1.5
```

相关信息

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

7.4.46 ST_Y

ST_Y — 返回点的 Y 坐标。

Synopsis

```
float ST_Y(geometry a_point);
```

描述

返回点的 Y 坐标，如果无效，返回 NULL。输入必须是点。



Note

要获取几何图形的最小和最大 Y 值，使用 `ST_YMin` 和 `ST_YMax` 函数。

- ✓ 此方法符合 SQL 1.1 的 OGC 功能规范。
- ✓ 此方法符合 SQL/MM 规范。SQL-MM 3: 6.1.4
- ✓ 此函数支持 3d 并且不会丢失 z-index。

示例

Code

```
SELECT ST_Y('POINT(1 2 3 4)');
```

Output

```
2
```

Code

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

Output

```
1.5
```

相关信息

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

7.4.47 ST_Z

`ST_Z` — 返回点的 Z 坐标。

Synopsis

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

描述

返回点的 Z 坐标，如果无效，返回 NULL。输入必须是点。



Note

要获取几何图形的最小和最大 Z 值，使用 `ST_ZMin` 和 `ST_ZMax` 函数。



该方法符合 SQL/MM 规范。



该函数支持 3d 并且不会丢失 z-index。

示例

Code

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

Output

```
3
```

相关信息

`ST_GeomFromEWKT`, `ST_M`, `ST_X`, `ST_Y`, `ST_ZMax`, `ST_ZMin`

7.4.48 ST_ZMFlag

`ST_ZMFlag` — 返回指示几何体的 ZM 坐标度的代码。

Synopsis

smallint `ST_ZMFlag`(geometry geomA);

描述

返回指示几何体的 ZM 坐标度的代码。

代码 0=XY、1=XYM、2=XYZ、3=XYZM。



该函数支持 3d 并且不会丢失 z-index。



此方法支持图形字符串和曲线。

示例

Code

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

Output

```
0
```

Code

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

Output

```
1
```

Code

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

Output

```
2
```

Code

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

Output

```
3
```

相关信息

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

7.4.49 ST_HasZ

ST_HasZ — 几何体是否具有 Z 维度。

Synopsis

boolean **ST_HasZ**(geometry geom);

描述

返回几何体是否具有 Z 维度并返回布尔值。如果几何体有 Z 维度，返回 **true**；否则，返回 **false**。具有 Z 维度的几何对象通常表示三维 (3D) 几何形状，而没有 Z 维度的几何对象表示二维 (2D) 几何形状。此函数对于确定几何形状是否具有高程或高度信息非常有用。

可用性：3.5.0

- ✓ 函数支持 3d 并且不会丢失 z-index。
- ✓ 功能支持 M 坐标。

示例

Code

```
SELECT ST_HasZ('POINT(1 2 3)');
```

Output

t

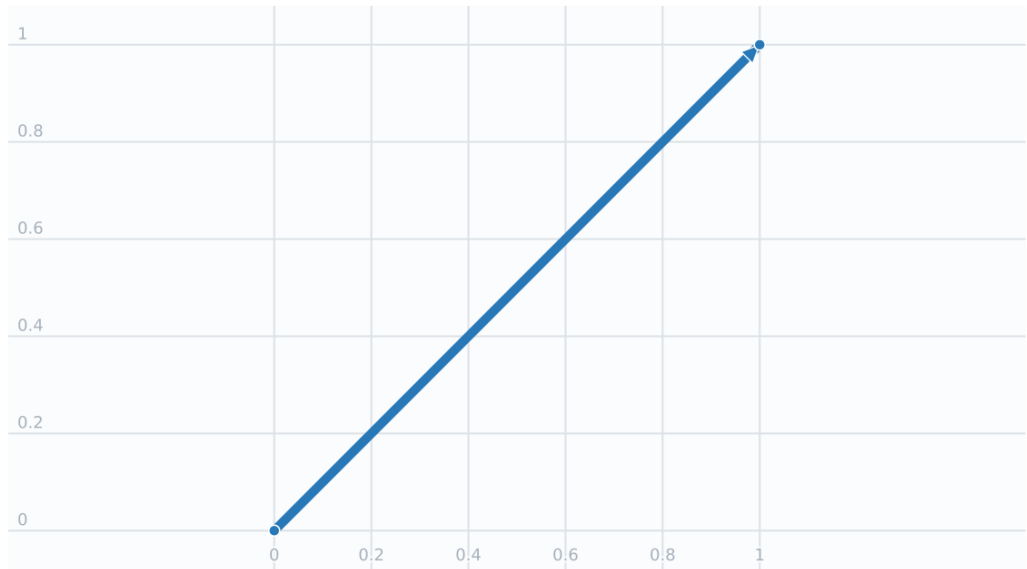
Code

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

Output

f

Figure



Code

相关信息

[ST_ZMFlag](#)
[ST_HasM](#)

7.4.50 ST_HasM

ST_HasM — 几何体是否具有 M（量）度。

Synopsis

boolean **ST_HasM**(geometry geom);

描述

返回几何对象是否具有 M（度量）维度并返回布尔值。如果几何体具有 M 维度，返回 true；否则，返回 false。具有 M 维度的几何对象通常表示与空间特征相关的测量或附加数据。

此函数用于确定几何对象是否包含测量信息非常有用。

可用性：3.5.0

✓ 函数支持 3d 并且不会丢失 z-index。

✓ 功能支持 M 坐标。

示例

Code

```
SELECT ST_HasM('POINTM(1 2 3)');
```

Output

```
t
```

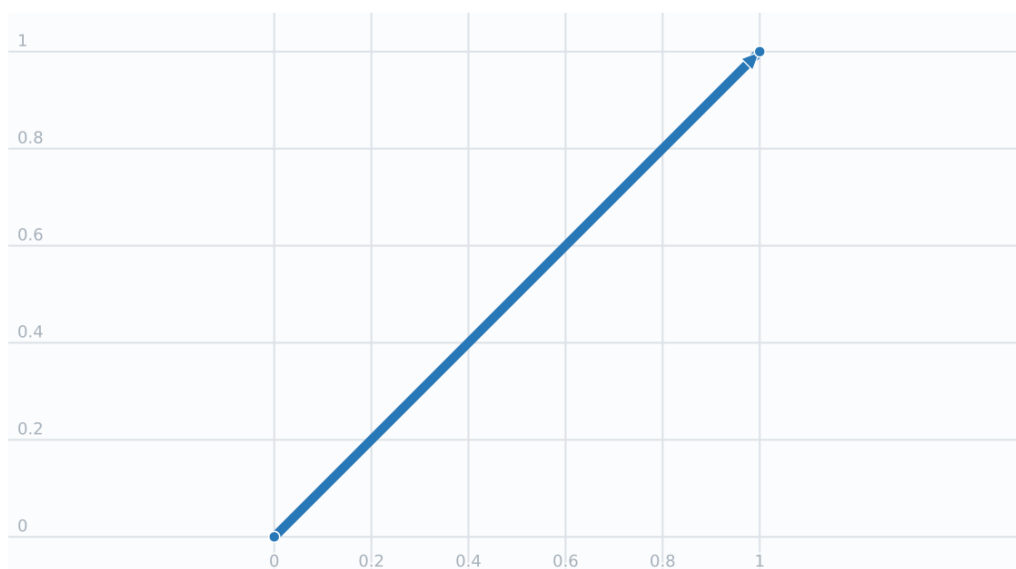
Code

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

Output

```
f
```

Figure



Code

相关信息

[ST_ZMFlag](#)

[ST_HasZ](#)

7.4.51 ST_ControlPoints

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

Synopsis

geometry **ST_ControlPoints**(geometry nurbscurve);

描述

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

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

Availability: 3.7.0

示例

This example extracts control points from a NURBS curve.

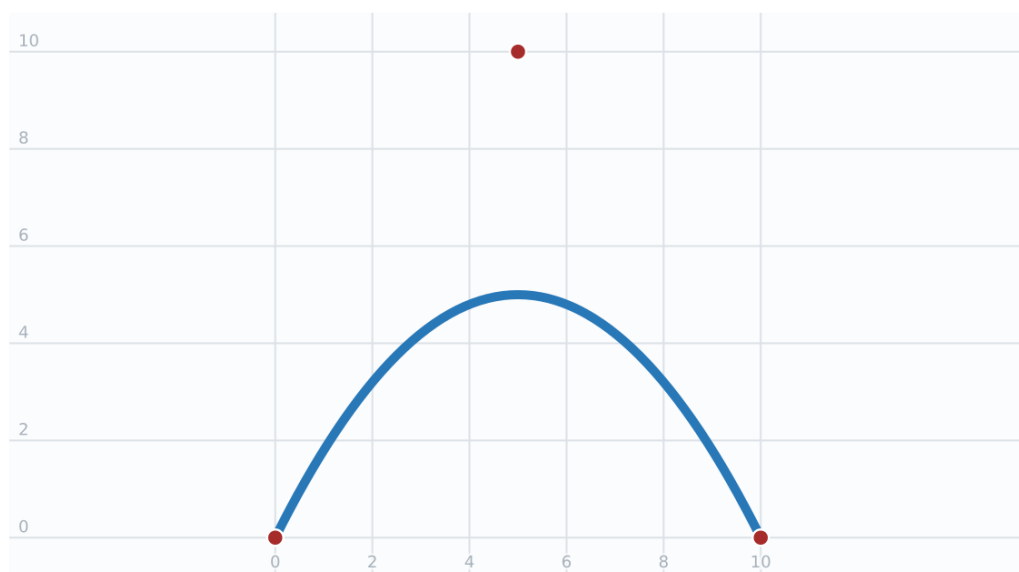
Code

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

Output

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

Figure



■ Code ■ Output

This example counts the control points.

Code

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

Output

```
3
```

相关信息

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

7.4.52 ST_Degree

ST_Degree — Returns the polynomial degree of a NURBS curve.

Synopsis

integer **ST_Degree**(geometry nurbscurve);

描述

Returns the polynomial degree of a NURBS curve as an integer. The degree determines the curve's smoothness and flexibility. Higher degrees produce smoother curves but require more control points.

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

Availability: 3.7.0

示例

This example returns the degree of a NURBS curve.

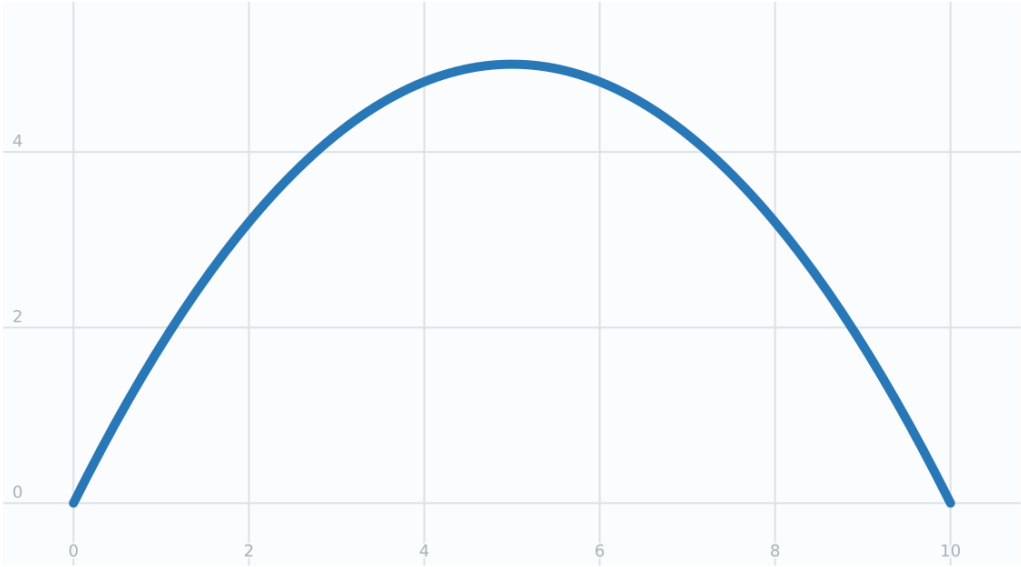
Code

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

Output

```
2
```

Figure



Code

This example checks whether a NURBS curve is cubic.

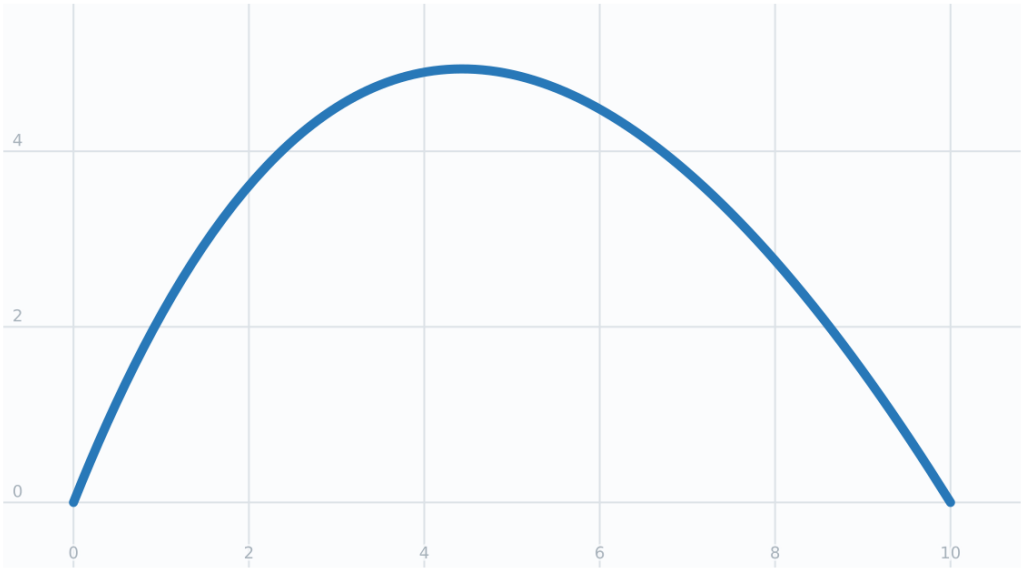
Code

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

Output

t

Figure



Code

相关信息

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

7.4.53 ST_Weights

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

Synopsis

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

描述

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

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

Availability: 3.7.0

示例

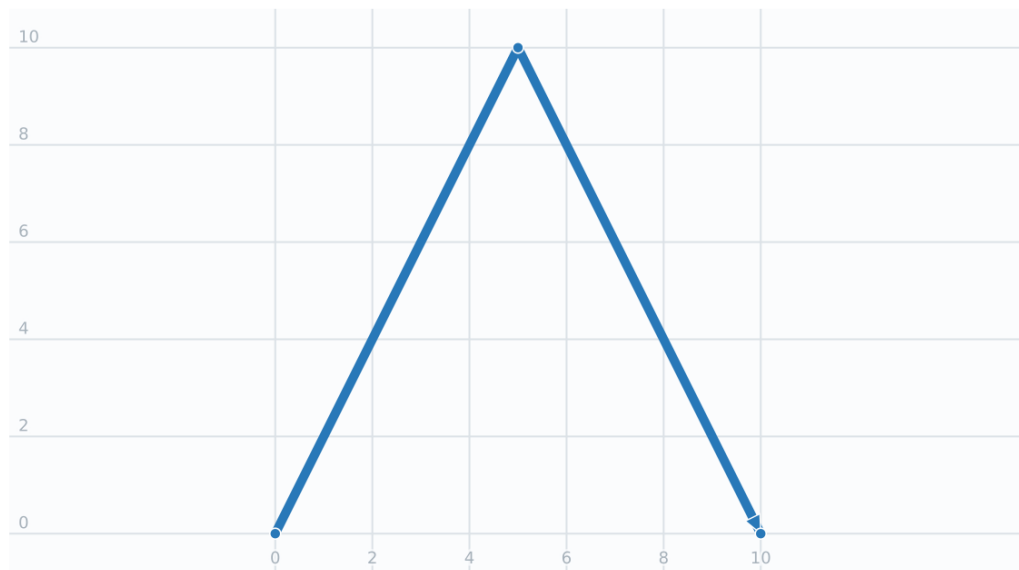
This example returns weights from a rational NURBS curve.

Code

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

Output

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

| Figure

■ Code

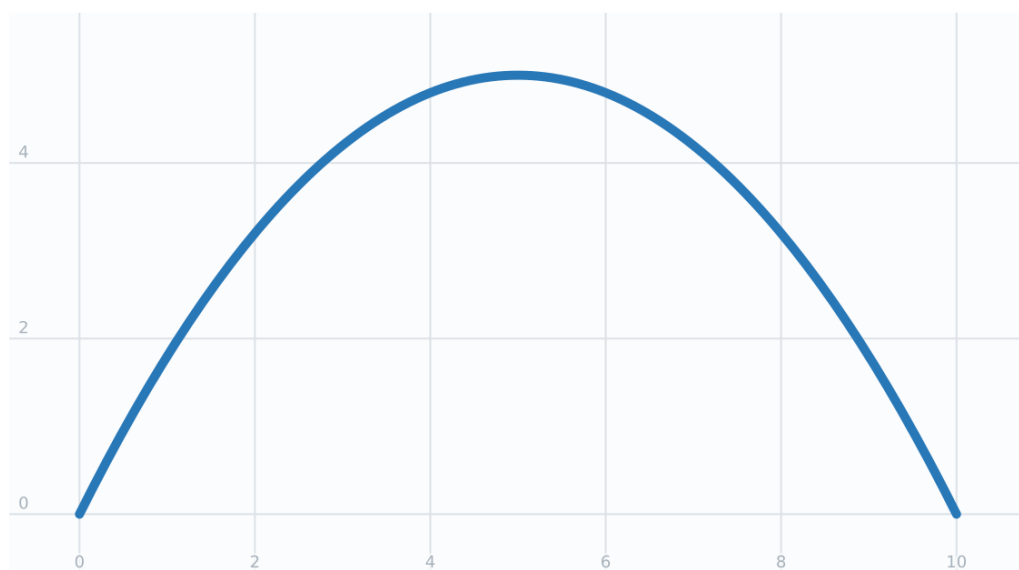
This example checks the weights of a non-rational curve.

| Code

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

| Output

```
null
```

| Figure

■ Code

相关信息

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

7.4.54 ST_Knots

ST_Knots — Returns the knot vector of a NURBS curve.

Synopsis

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

描述

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

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

Availability: 3.7.0

示例

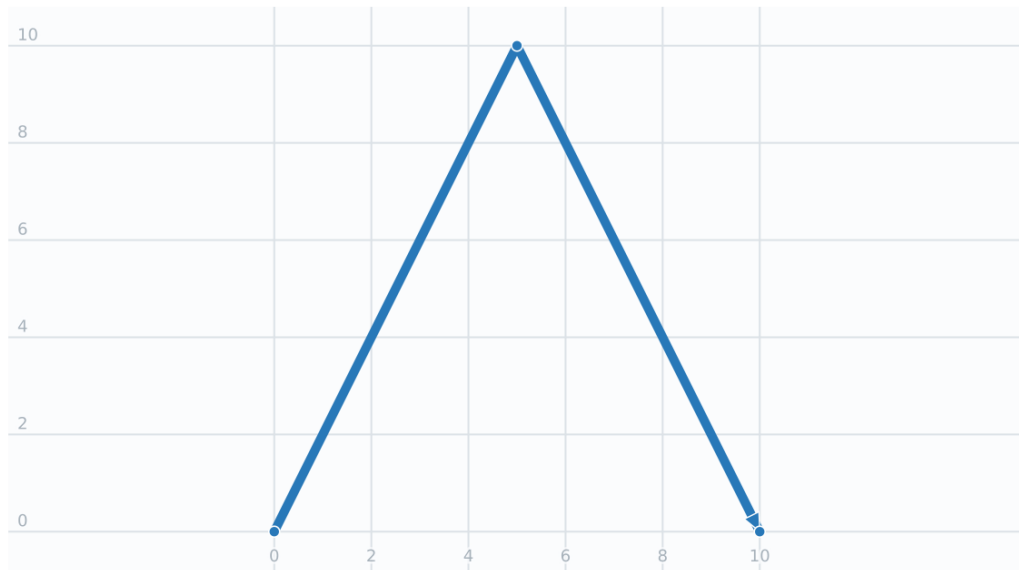
This example returns knots from a NURBS curve with a custom knot vector.

Code

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

Output

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


| Figure

■ Code

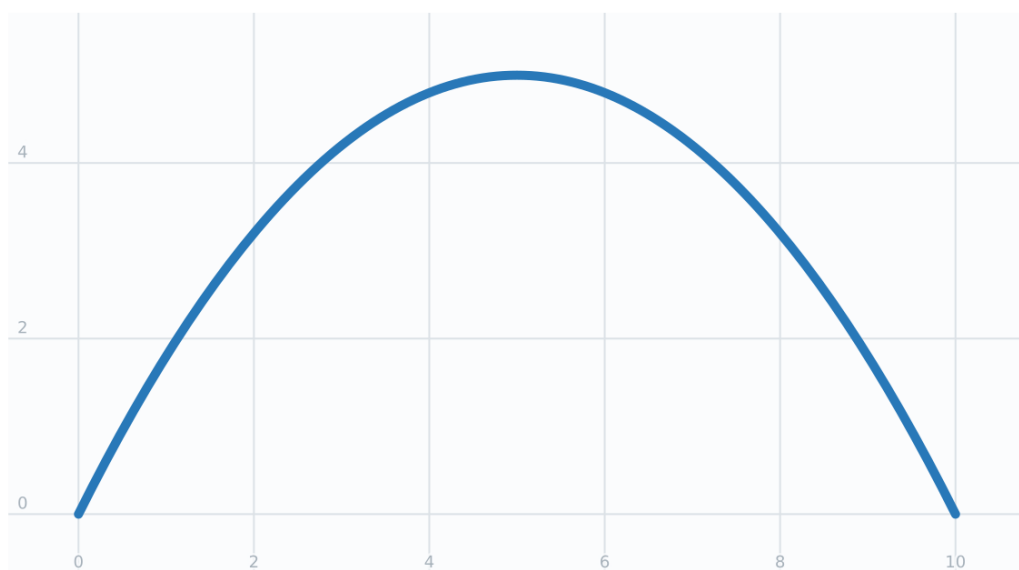
This example checks the knots of a curve using the default knot vector.

| Code

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

| Output

```
null
```

| Figure

■ Code

相关信息

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

7.4.55 ST_NumControlPoints

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

Synopsis

```
integer ST_NumControlPoints(geometry nurbscurve);
```

描述

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

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

Availability: 3.7.0

示例

This example returns the number of control points.

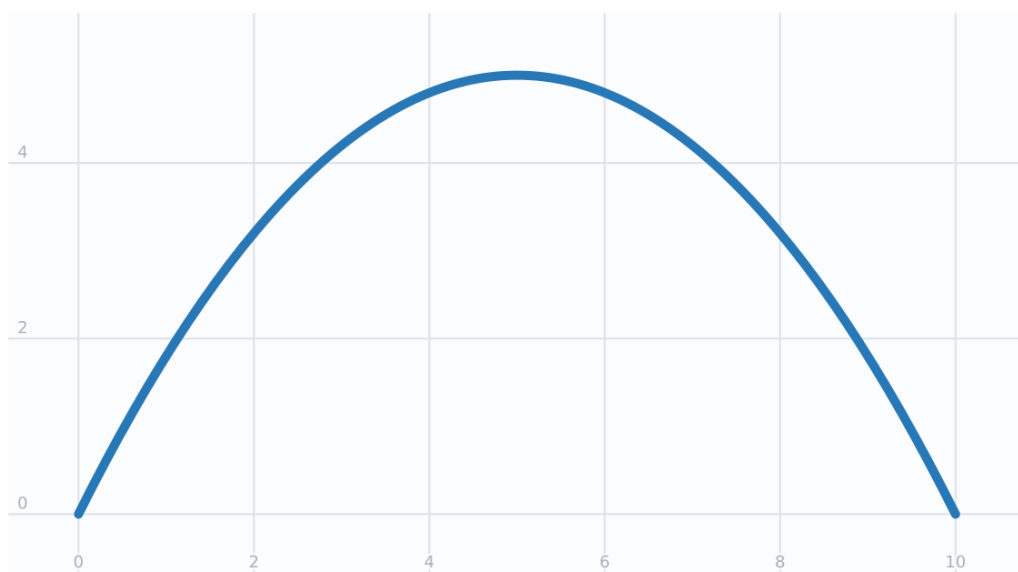
Code

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

Output

```
3
```

Figure



■ Code

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

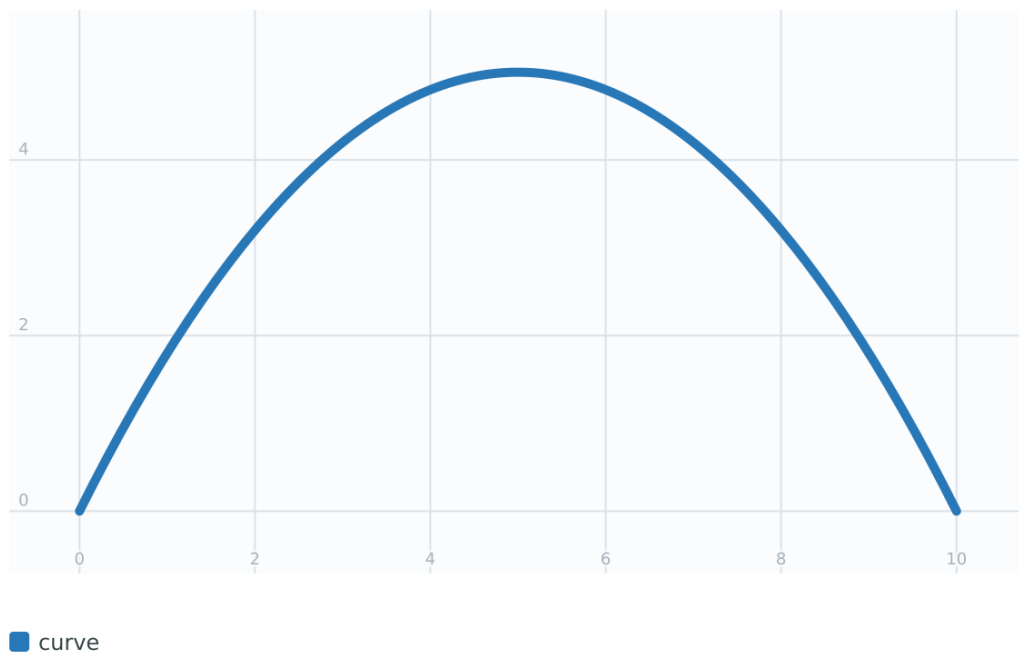
Code

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

Output

t

Figure



相关信息

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

7.4.56 ST_NurbsCurveIsRational

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

Synopsis

boolean **ST_NurbsCurveIsRational**(geometry nurbscurve);

描述

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

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

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

Availability: 3.7.0

示例

This example checks whether a curve is rational.

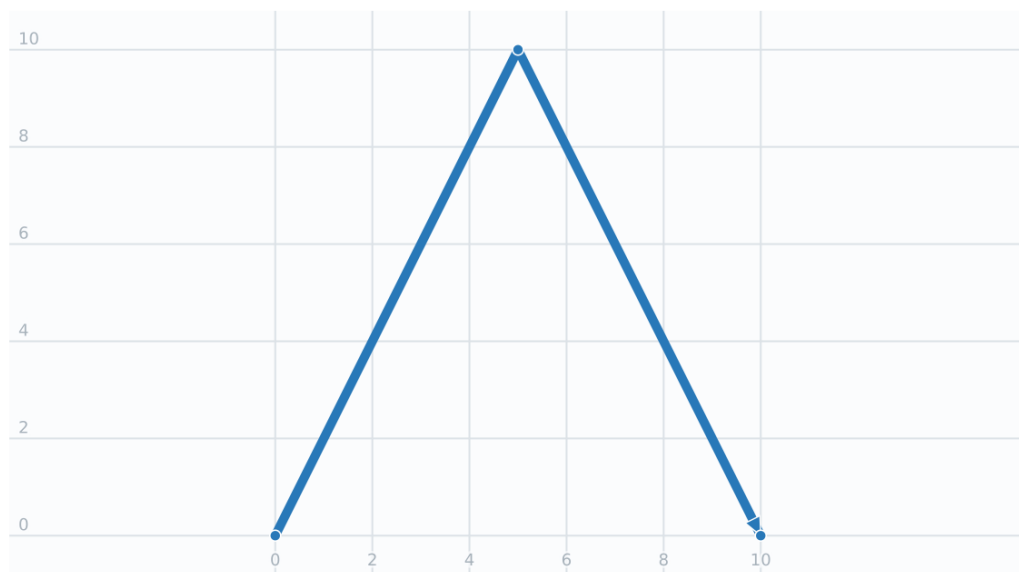
Code

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

Output

t

Figure



■ Code

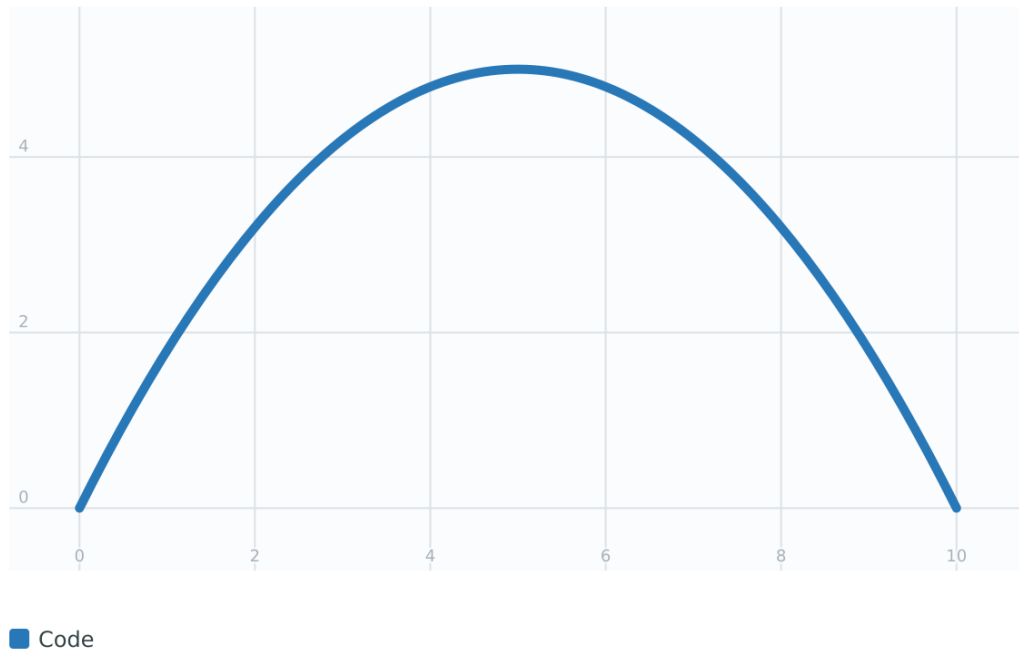
This example checks a non-rational curve.

Code

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

Output

f

Figure

相关信息

[ST_MakeNurbsCurve](#), [ST_Weights](#), [ST_IsNurbsCurve](#)

7.4.57 ST_IsNurbsCurve

ST_IsNurbsCurve — Checks if a geometry is a NURBS curve.

Synopsis

```
boolean ST_IsNurbsCurve(geometry geom);
```

描述

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

Returns NULL if the input is NULL.

Availability: 3.7.0

示例

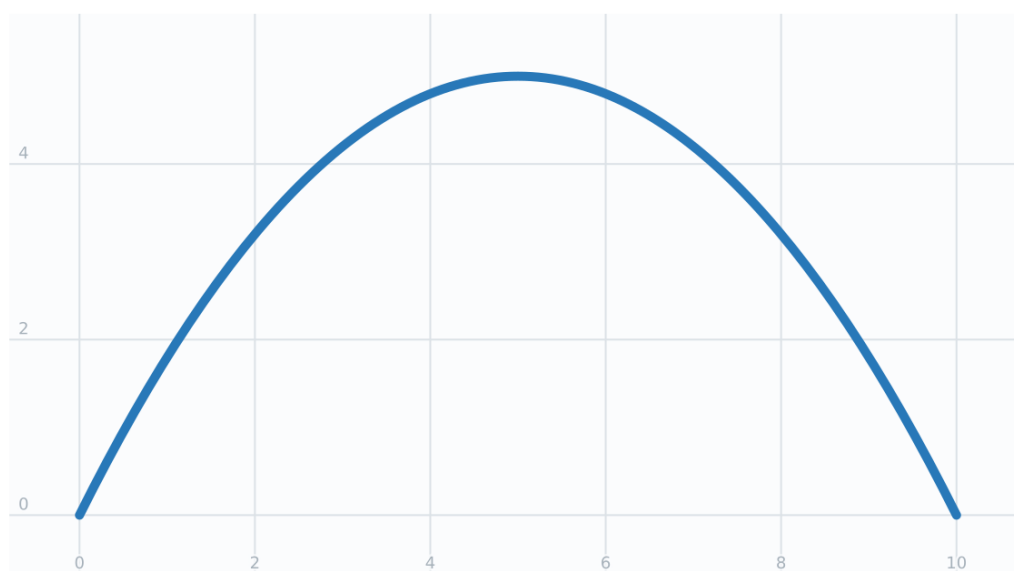
This example checks whether a geometry is a NURBS curve.

| Code

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

| Output

```
t
```

| Figure

■ Code

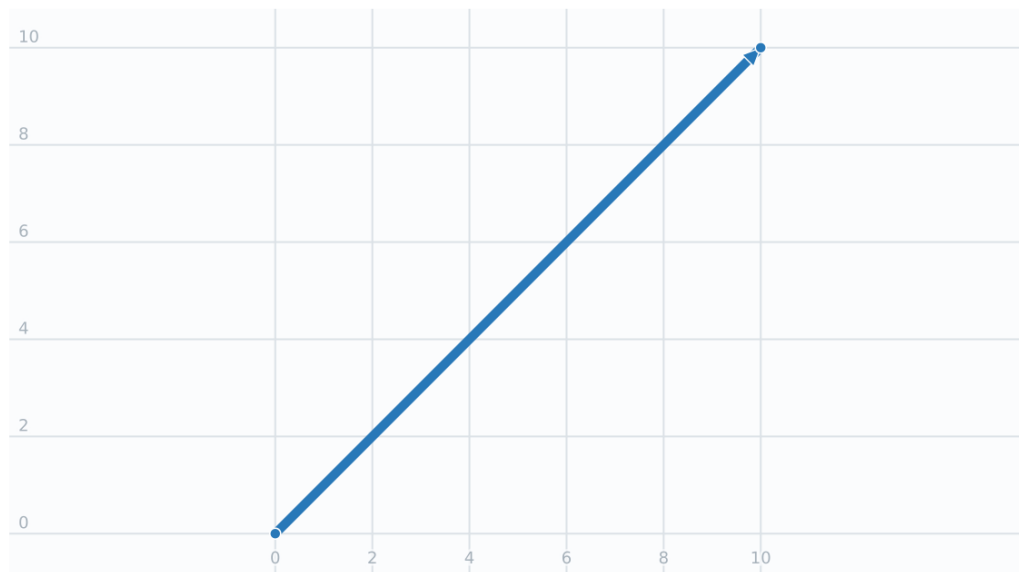
This example tests a regular LineString.

| Code

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

| Output

```
f
```

Figure

■ Code

相关信息

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

7.4.58 ST_Evaluate

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

Synopsis

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

描述

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

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

Availability: 3.7.0

示例

This example returns the midpoint of a NURBS curve.

Code

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

This example samples multiple points along a curve.

Code

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

相关信息

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

7.4.59 ST_NurbsToLineString

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

Synopsis

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

描述

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

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

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

Availability: 3.7.0

示例

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

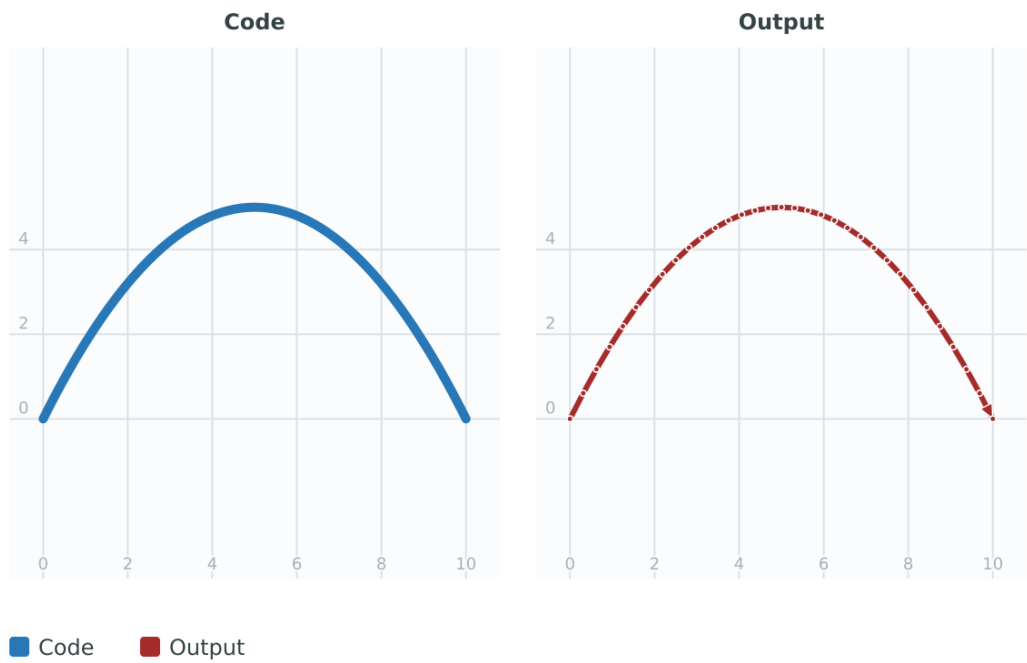
Code

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

Output

```
LINESTRING(0 0,0.3125 0.60546875,0.625 1.171875,0.9375 1.69921875,1.25 2.1875,1.5625 2.63671875,1.875 3.046875,2.1875 3.41796875,2.5 3.75,2.8125 4.04296875,3.125 4.296875,3.4375 4.51171875,3.75 4.6875,4.0625 4.82421875,4.375 4.921875,4.6875 4.98046875,5 5,5.3125 4.98046875,5.625 4.921875,5.9375 4.82421875,6.25 4.6875,6.5625 4.51171875,6.875 4.296875,7.1875 4.04296875,7.5 3.75,7.8125 3.41796875,8.125 3.046875,8.4375 2.63671875,8.75 2.1875,9.0625 1.69921875,9.375 1.171875,9.6875 0.60546875,10 0)
```


Figure

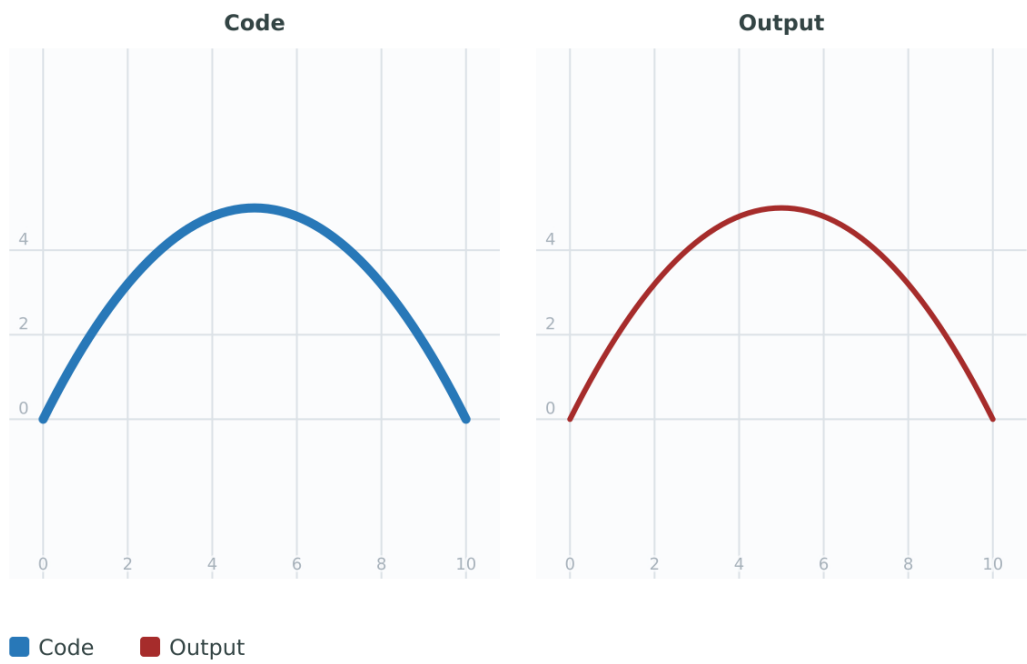


This example creates a high-quality approximation with 64 segments.

Code

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

Figure



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

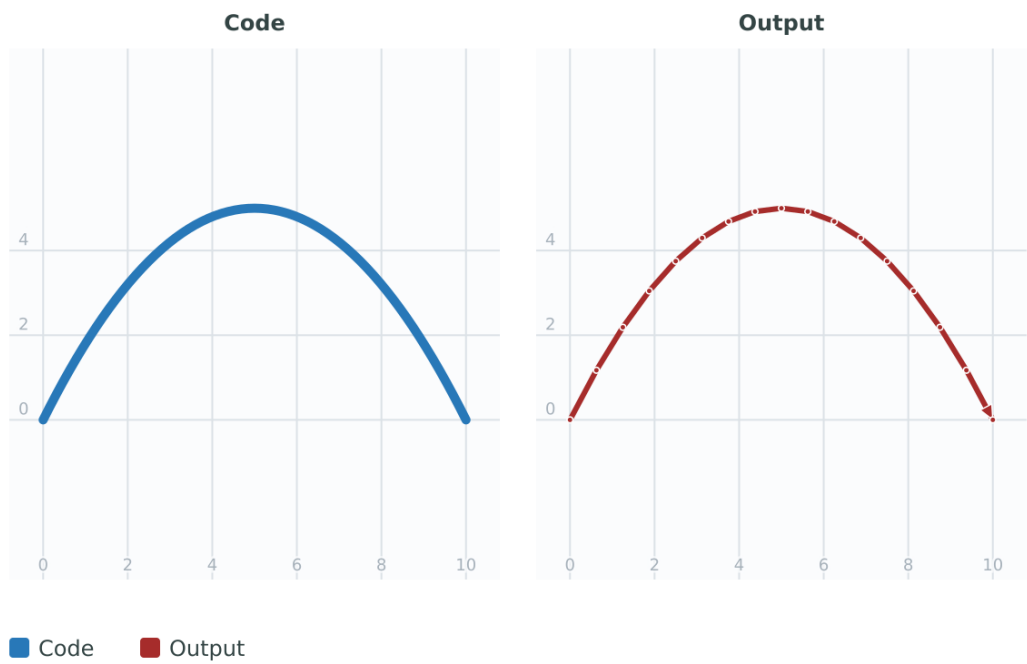
Code

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

Output

```
LINESTRING(0 0,0.625 1.171875,1.25 2.1875,1.875 3.046875,2.5 3.75,3.125 4.296875,3.75 4.6875,4.375 4.921875,5 5,5.625 4.921875,6.25 4.6875,6.875 4.296875,7.5 3.75,8.125 3.046875,8.75 2.1875,9.375 1.171875,10 0)
```

Figure



相关信息

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

7.5 几何构造器

7.5.1 ST_AddPoint

ST_AddPoint — 将点添加到串 (LineString)。

Synopsis

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

描述

在索引 *position* 之前向 `LineString` 添加一个点（使用从 0 开始的索引）。如果 *position* 参数被省略或 `-1`，点将附加到 `LineString` 的末尾。

可用性 : 1.1.0

 函数支持 3d 并且不会丢失 z-index。

示例

在 3D 线的末尾添加一个点

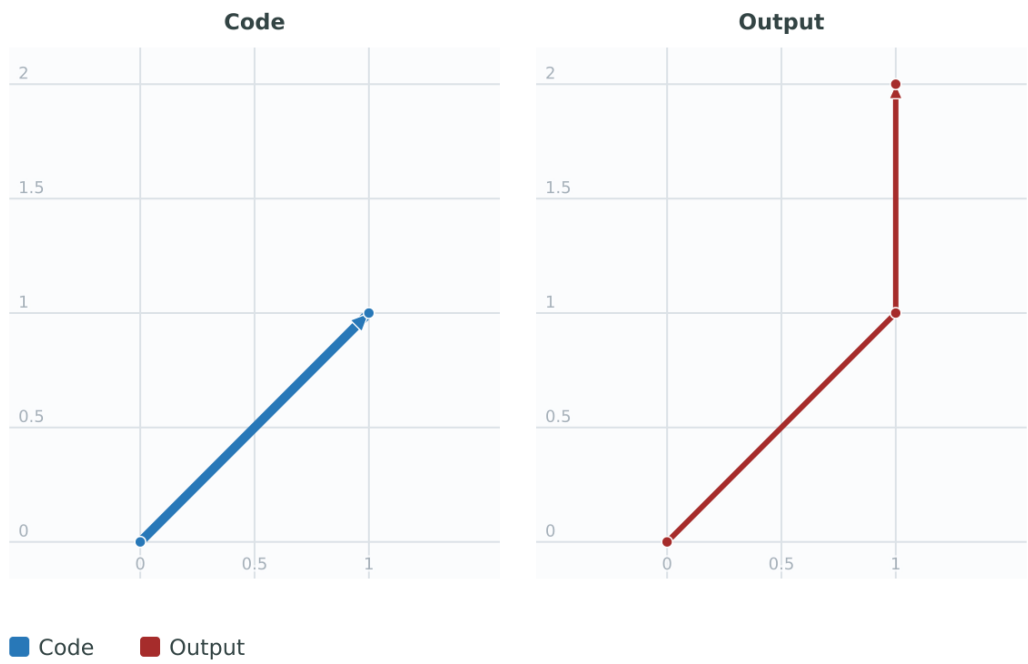
Code

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

Output

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

Figure



对于表中那些未闭合的线，函数将每条线的起点添加到线的末尾来保证表中的所有线都是闭合的。

Code

```
UPDATE sometable
SET geom = ST_AddPoint(geom, ST_StartPoint(geom))
FROM sometable
WHERE ST_IsClosed(geom) = false;
```

相关信息

[ST_RemovePoint](#), [ST_SetPoint](#)

7.5.2 ST_CollectionExtract

ST_CollectionExtract — 给定一个几何集合，返回包含指定型元素的多几何形。

Synopsis

```
geometry ST_CollectionExtract(geometry collection);
```

```
geometry ST_CollectionExtract(geometry collection, integer type);
```

描述

给定一个几何集合，返回同种型多几何。

如果未指定型，返回包含最高度几何的多几何。因此，多边形先于线，线先于点。

如果指定了型，返回包含指定型的多几何。如果没有指定型的元素，返回 EMPTY 几何形。支持点、线和面。型号如下：

- 1 == POINT
- 2 == LINESTRING
- 3 == POLYGON

于基本几何入，如果入型与求的型匹配，几何形将保持不返回。否则，果是指定型空几何形。如果需要，可以使用 **ST_Multi** 将它转换为多几何形。



Warning

不 MultiPolygon 果的有效性。如果多边形件相或重，果将无效。（例如，将此函数用于 **ST_Split** 果可能会生情况。）可以使用 **ST_IsValid** 检查情况并使用 **ST_MakeValid** 修复。

可用性：1.1.0



Note

在 1.5.3 之前，无论型如何，函数都会返回同种型的基本几何入。在 1.5.3 中，不匹配的个几何形返回 NULL 果。在 2.0.0 中，不匹配的个几何形返回求型的 EMPTY 果。

示例

提取最大度型：

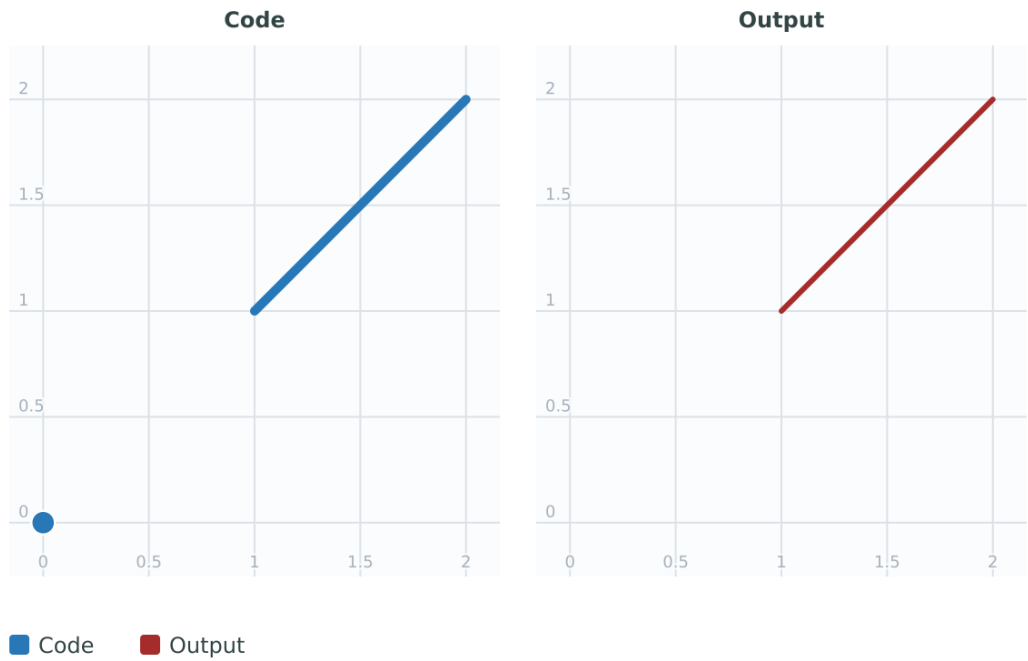
Code

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

Output

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

Figure



提取点 (type 1 == POINT) :

Code

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

Output

```
MULTIPOINT((0 0))
```

Figure



提取☐ (type 2 == LINESTRING) :

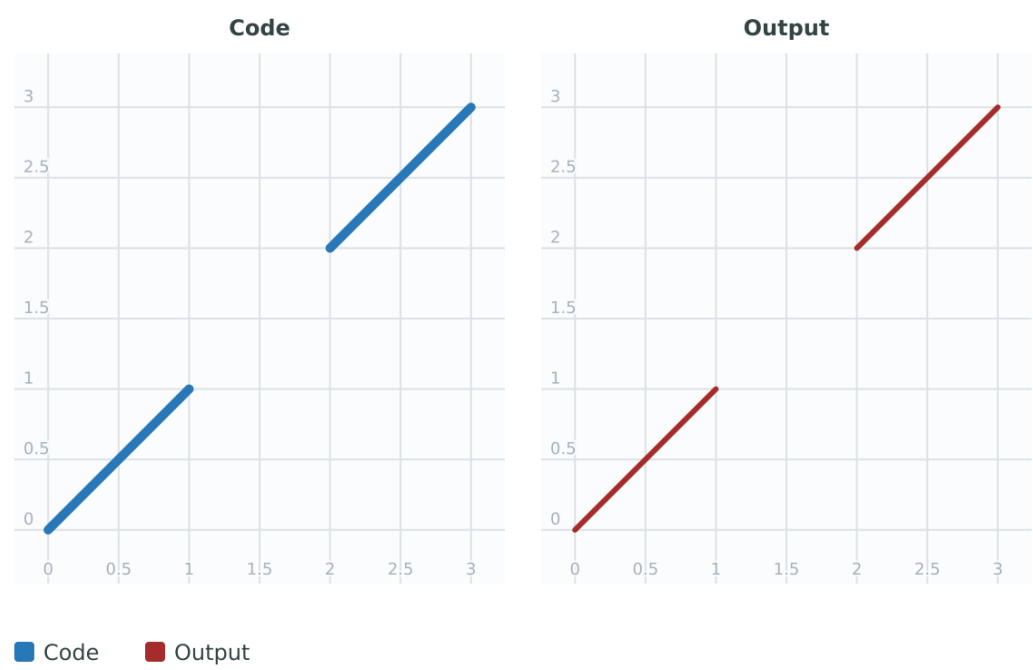
Code

```
SELECT ST_CollectionExtract('GEOMETRYCOLLECTION(GEOMETRYCOLLECTION(LINESTRING(0 0,1 1)), ↔  
    LINESTRING(2 2,3 3))',  
    2);
```

Output

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

Figure



相关信息

[ST_CollectionHomogenize](#), [ST_Multi](#), [ST_IsValid](#), [ST_MakeValid](#)

7.5.3 ST_CollectionHomogenize

ST_CollectionHomogenize — 返回几何集合的最☐☐表示。

Synopsis

geometry **ST_CollectionHomogenize**(geometry collection);

描述

☐定几何集合，它返回“最☐☐”的表示形式。

- 同☐集合将以适当的多几何体形式返回。

- 混合类型集合将每个平面几何集合。
- 包含个基本元素的集合将作元素返回。
- 基本几何形状返回不。如果需要，可以使用ST_Multi将它多几何形。



Warning
函数不保证果有效。特别是，包含相或重多形的集合将建无效的多多形。这种情况可以使用ST_IsValid并使用ST_MakeValid修复。

可用性: 2.0.0

示例

个元素集合基本集合

Code

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

Output

POINT(0 0)

Figure



将嵌套的元素集合基本几何：

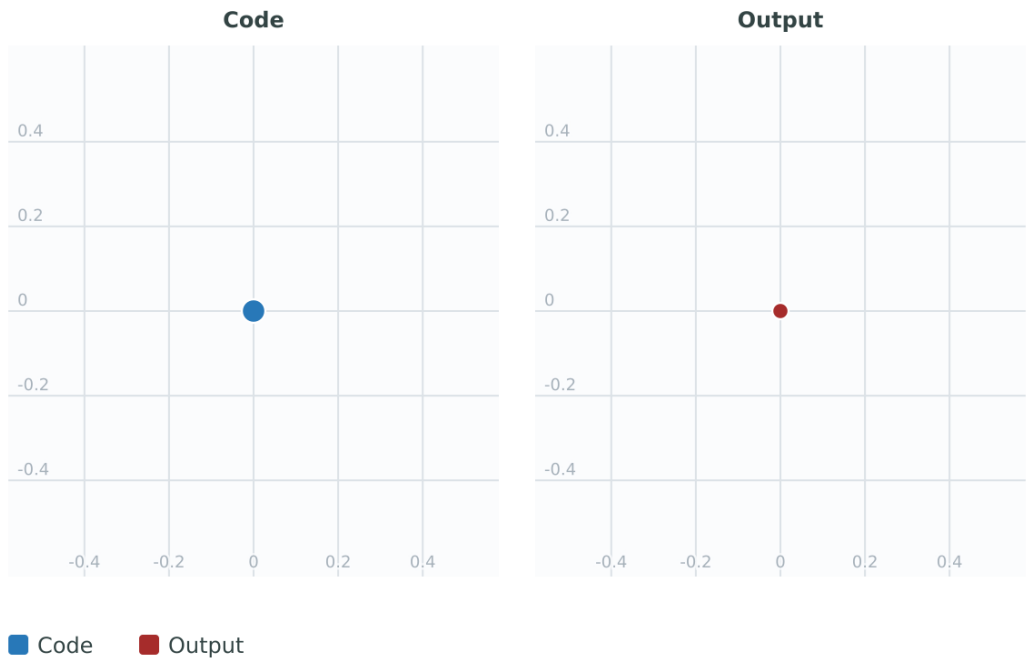
Code

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

Output

```
POINT(0 0)
```

Figure



集合多几何形：

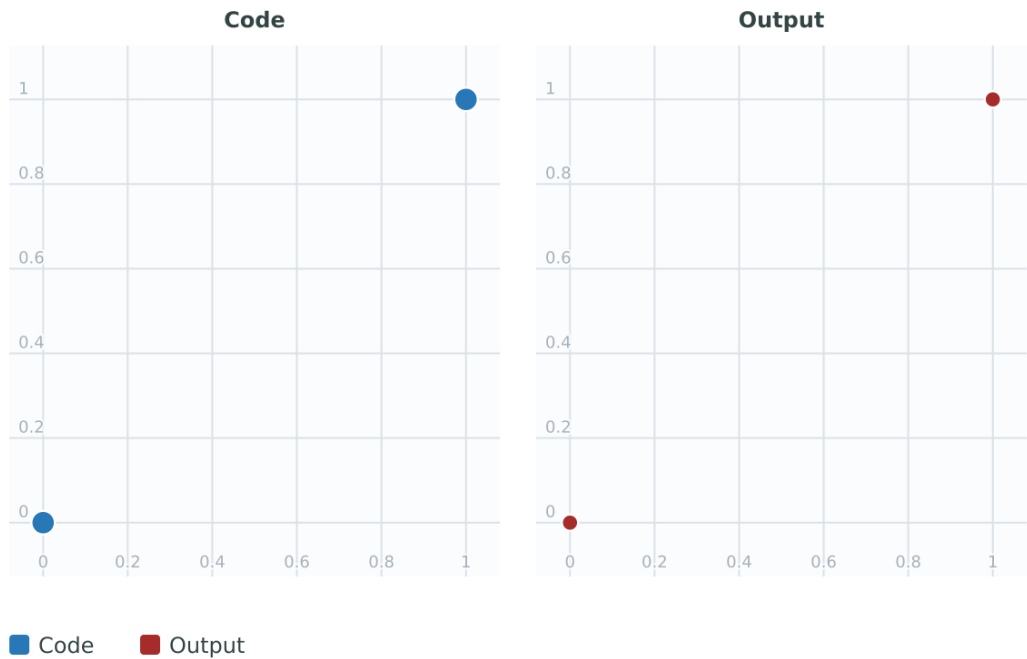
Code

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

Output

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


Figure



将嵌套混合☐型集合☐☐☐平面几何集合：

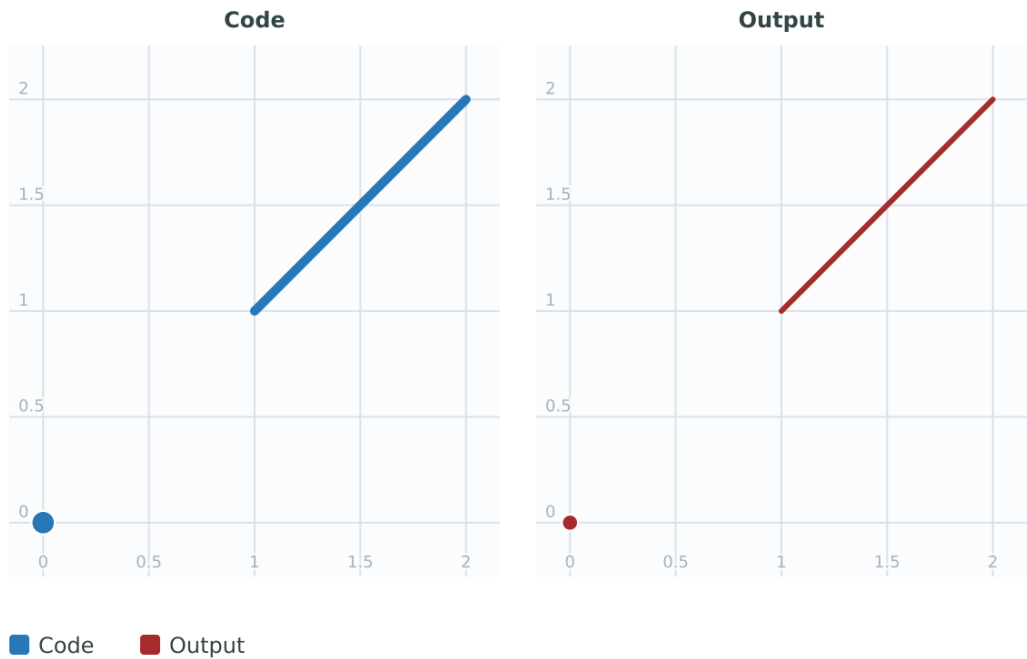
Code

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

Output

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

Figure



将多边形集合（无效的）多多边形：

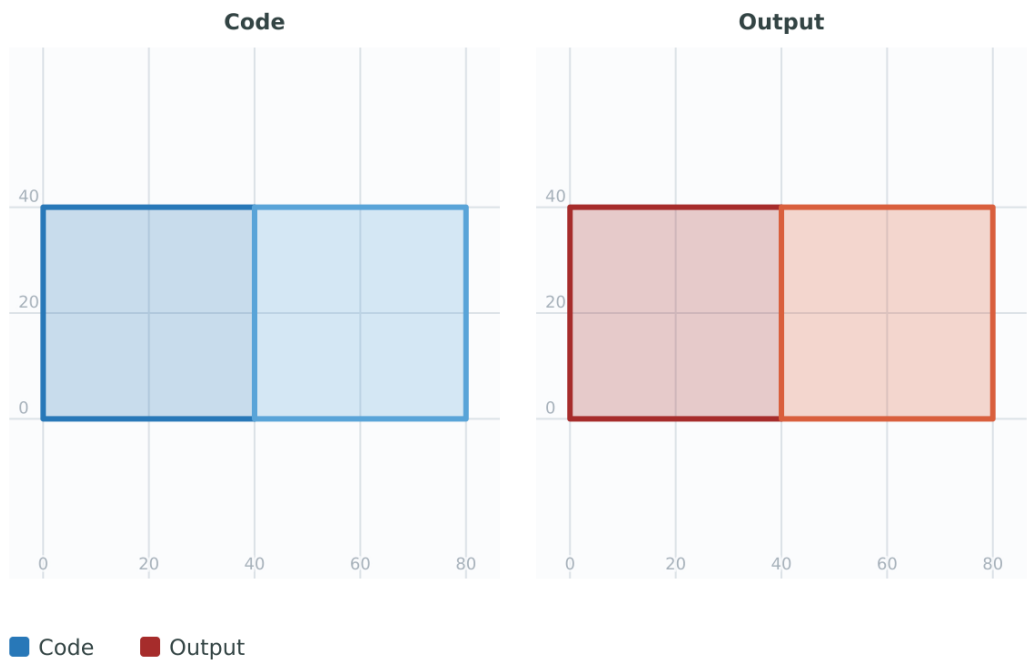
Code

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

Output

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

Figure



相关信息

[ST_CollectionExtract](#), [ST_Multi](#), [ST_IsValid](#), [ST_MakeValid](#)

7.5.4 ST_CurveToLine

ST_CurveToLine — 将包含曲线的几何形转换为线性几何形。

Synopsis

geometry **ST_CurveToLine**(geometry curveGeom, float tolerance, integer tolerance_type, integer flags);

描述

将 CIRCULAR STRING 转换为 LINESTRING，或将 CURVEPOLYGON 转换为 POLYGON，或将 MULTISURFACE 转换为 MULTIPOLYGON。对于不支持 CIRCULARSTRING 几何类型的数据库很有用

将☐定几何☐形☐☐☐性几何☐形。使用☐定的“公差”和☐☐（每个象限（一个☐的四个部分）32 个☐段，默☐情况下没有☐☐）将每个弯曲几何☐形或☐段☐☐☐☐性近似☐。

“tolerance_type” 参数决定 “tolerance” 参数的解☐。它可以采用以下☐：

- 0（默☐）：公差是四分之一☐的最大☐数。
- 1：公差是从曲☐到直☐的最大差☐。☐位是☐入几何的☐位。
- 2：公差是由半径形成的角度的最大☐（以弧度☐☐位）。

‘flags’ 参数是一个位字段。默☐☐ 0。支持的位有：

- 1：☐称（与方向无关）☐出。
- 2：保留角度，避免在☐生☐称☐出☐☐少角度（☐段☐度），☐称☐志关☐☐无效。

可用性：1.3.0

增☐：2.4.0 支持最大距离差公差和最大角度公差，支持☐称☐出。

增☐：3.0.0 ☐☐了每弧的最小☐性分数。防止拓扑崩☐。

- ☑ 此方法☐☐了 [SQL 1.1 的 OGC ☐☐功能☐范](#)。
- ☑ ☐方法☐☐了 SQL/MM ☐范。SQL-MM 3: 7.1.7
- ☑ ☐函数支持 3d 并且不会☐失 z-index。
- ☑ 此方法支持☐形字符串和曲☐。

示例

Code

```
SELECT ST_CurveToLine('CIRCULARSTRING(220268 150415,220227 150505,220227 150406)');
```

Output

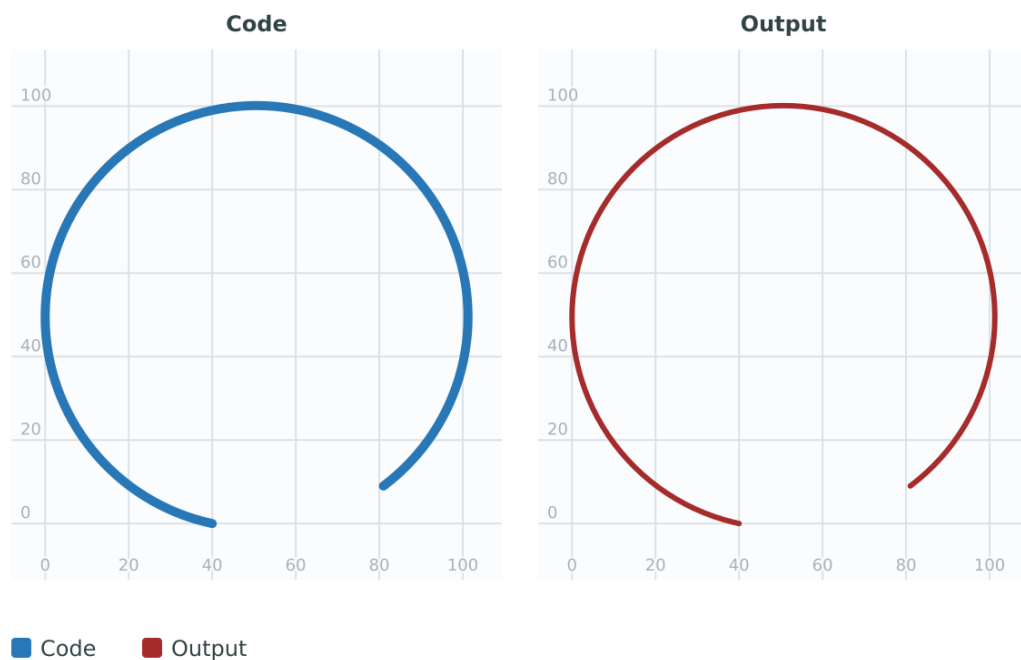
```
LINESTRING(220268 150415,220269.95064912 150416.539364228,220271.823415575 150418.17258804,220273.613787707 ↵
150419.895736857,
220275.317452352 150421.704659462,220276.930305234 150423.594998003,220278.448460847 150425.562198489,
220279.868261823 150427.60152176,220281.186287736 150429.708054909,220282.399363347 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)

```

Figure



This example uses 3D input.

Code

```
SELECT ST_CurveToLine('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.

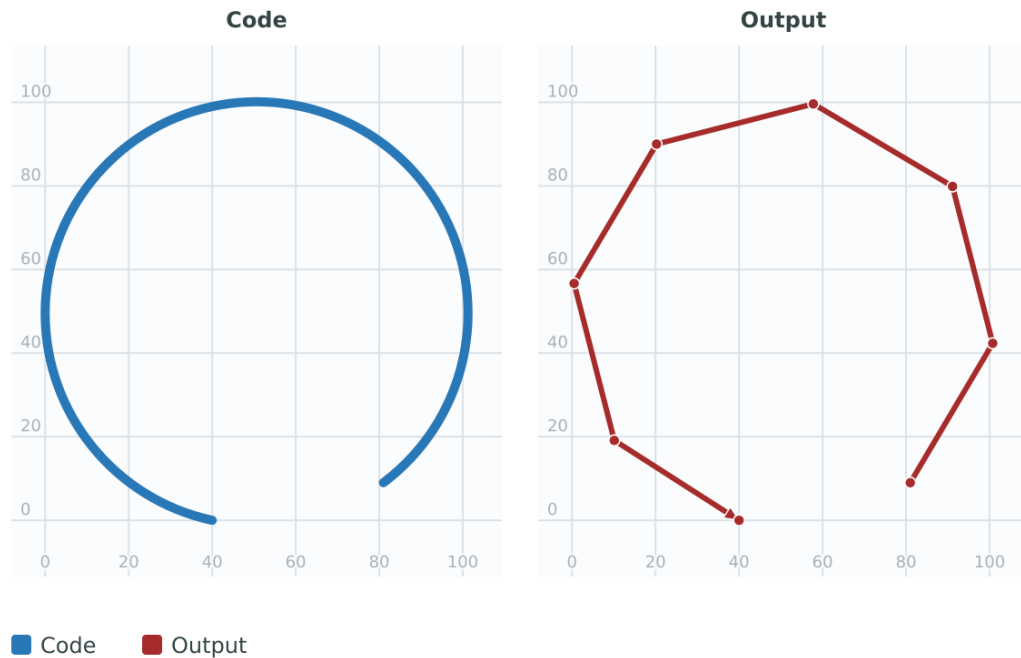
Code

```
SELECT ST_CurveToLine('CIRCULARSTRING(220268 150415,220227 150505,220227 150406)', 2);
```

Output

```
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)
```

Figure



Ensure the approximated line is no more than 20 units away from the original curve, and make the result direction-neutral.

Code

```
SELECT ST_CurveToLine(
  'CIRCULARSTRING(0 0,100 -100,200 0)::geometry,
  20,
  1,
  1
);
```

Output

```
LINESTRING(0 0,49.999999999999996 -86.60254037844383,149.99999999999994 -86.6025403784439,200 0)
```

Figure



相关信息

[ST_LineToCurve](#)

7.5.5 ST_Scroll

ST_Scroll — 更改☐☐串的起点。

Synopsis

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

描述

将☐☐串的起点/☐点更改☐☐定的 *point*。

可用性：3.2.0

☑☐函数支持 3d 并且不会☐失 z-index。

☑☐功能支持 M 坐☐。

示例

Make a closed line start at its 3rd vertex.

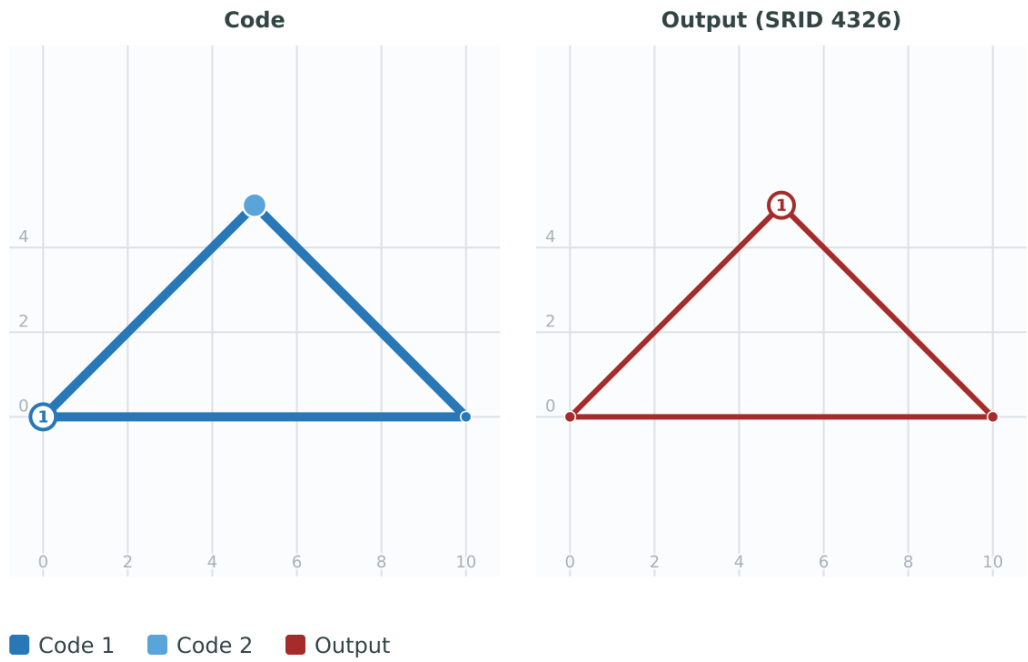
Code

```
SELECT 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)' ↔ );
```

Output

```
SRID=4326;LINESTRING(5 5 4 2,0 0 0 1,10 0 2 0,5 5 4 2)
```

Figure



相关信息

[ST_Normalize](#)

7.5.6 ST_FlipCoordinates

ST_FlipCoordinates — 返回 X 和 Y 的几何版本。

Synopsis

```
geometry ST_FlipCoordinates(geometry geom);
```

描述

返回几何的 X 和 Y 的版本。用于修改包含以度/度 (Y,X) 表示的坐标的几何形很有用。

可用性: 2.0.0

 此方法支持形字符串和曲线。

- ✔ 函数支持 3d 并且不会丢失 z-index。
- ✔ 功能支持 M 坐标。
- ✔ 函数支持多面体曲面。
- ✔ 此函数支持三角形和不规则三角网面 (TIN)。

示例

Code

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

Output

POINT(2 1)

Figure



相关信息

[ST_SwapOrdinates](#)

7.5.7 ST_Force2D

ST_Force2D — 强制几何图形进入“二维模式”。

Synopsis

geometry **ST_Force2D**(geometry geomA);

描述

将几何图形放入“二维模式”，以便所有输出表示都具有 X 和 Y 坐标。这对于 OGC 兼容输出非常有用（因为 OGC 指定二维几何形状）。

增加：2.0.0 引入了多面体曲面的支持。

更改：2.1.0 在 2.0.x 期间，它被称为 ST_Force_2D。

- ✓ 此方法支持图形字符串和曲线。
- ✓ 该函数支持多面体曲面。
- ✓ 该函数支持 3d 并且不会丢失 z-index。

示例

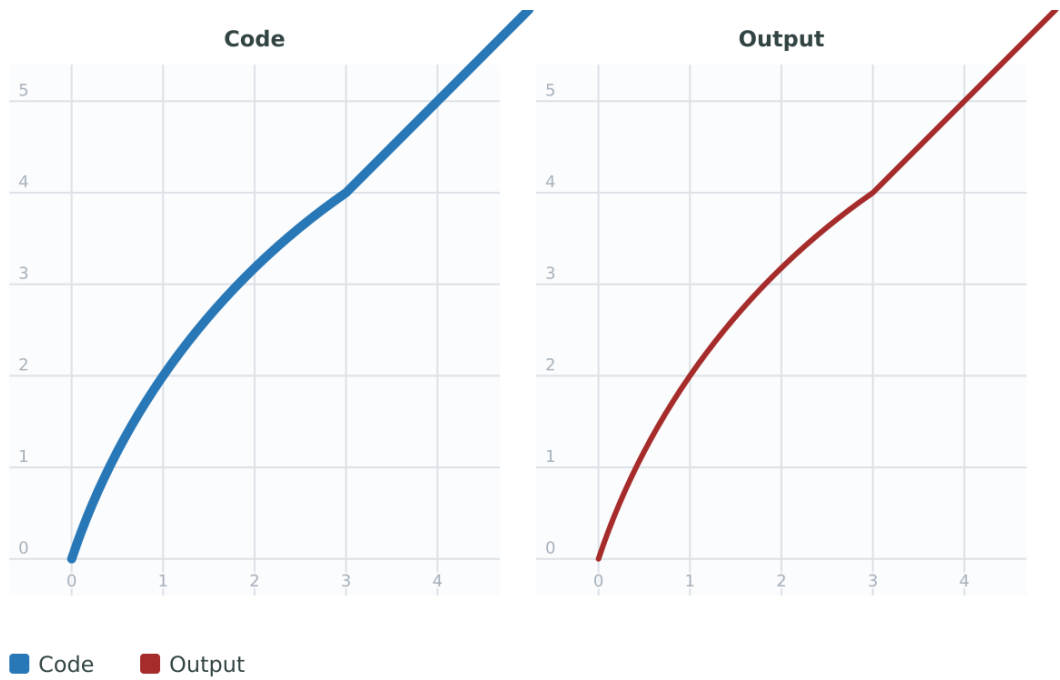
Code

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

Output

```
CIRCULARSTRING(1 1,2 3,4 5,6 7,5 6)
```

Figure



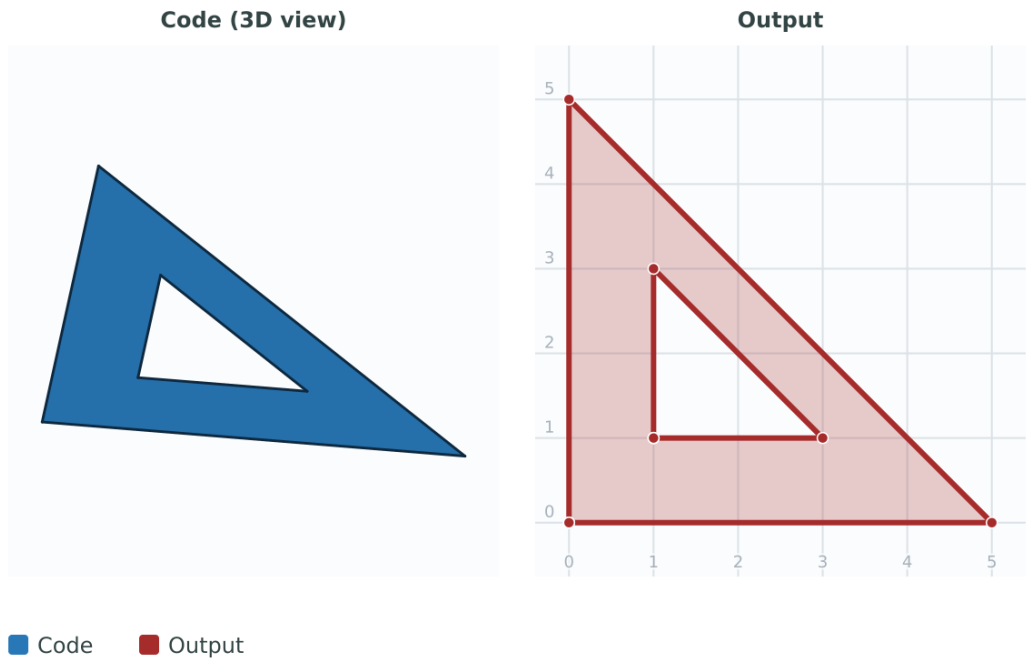
Code

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

Output

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

Figure



相关信息

ST_Force3D

7.5.8 ST_Force3D

ST_Force3D — 强制几何图形进入 XYZ 模式。它是 ST_Force3DZ 的同名。

Synopsis

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

描述

强制几何图形进入 XYZ 模式。它是 ST_Force3DZ 的同名。如果几何体没有 Z 分量，它会附加 z 或 Z 坐标。

新增：2.0.0 引入了多面体曲面的支持。

更改：2.1.0 在 2.0.x 期间，它被称为 ST_Force_3D。

更改：3.1.0. 您现在可以指定一个非零 Z 值。

- ✓ 该函数支持多面体曲面。
- ✓ 此方法支持图形字符串和曲线。
- ✓ 该函数支持 3d 并且不会丢失 z-index。

示例

Nothing happens to an already 3D geometry.

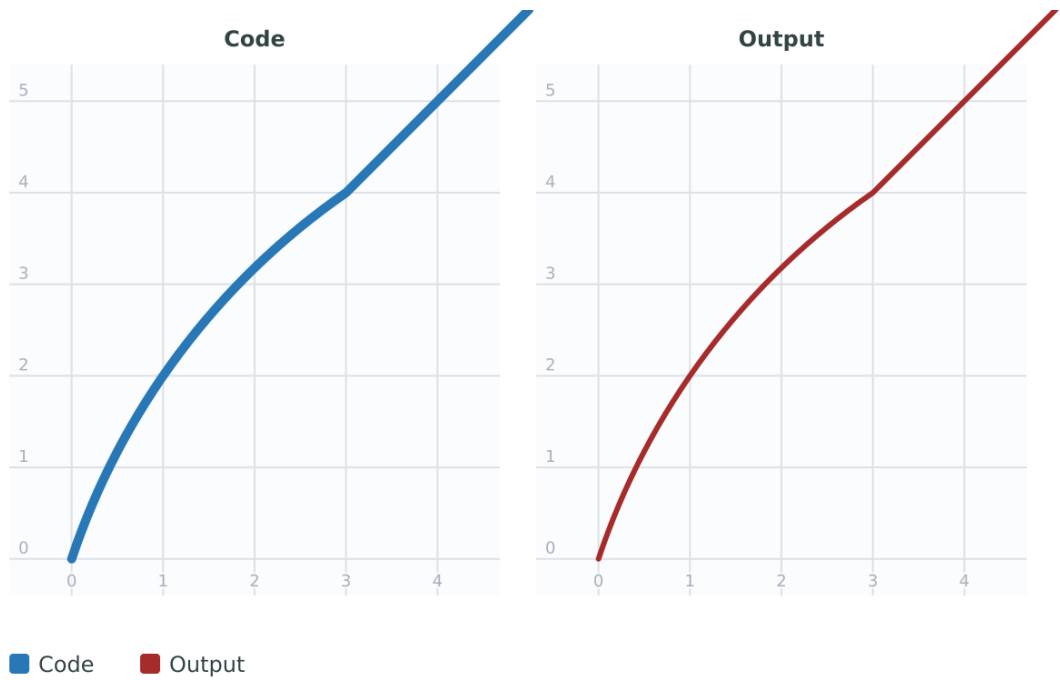
Code

```
SELECT ST_Force3D('CIRCULARSTRING(1 1 2,2 3 2,4 5 2,6 7 2,5 6 2)');
```

Output

```
CIRCULARSTRING(1 1 2,2 3 2,4 5 2,6 7 2,5 6 2)
```

Figure



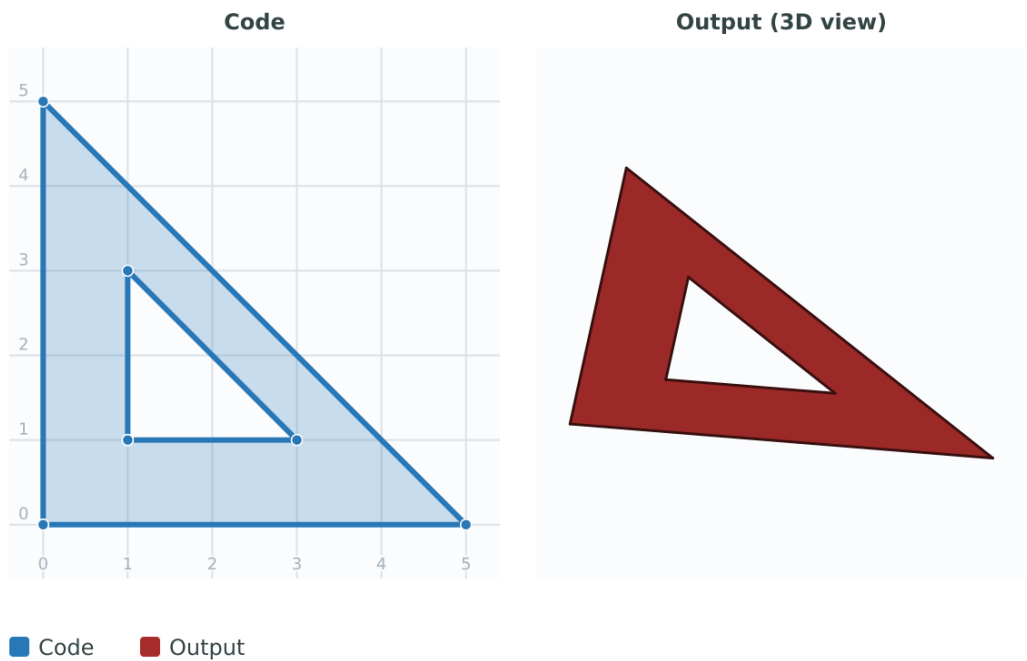
Code

```
SELECT ST_Force3D('POLYGON((0 0,0 5,5 0,0 0),(1 1,3 1,1 3,1 1))');
```

Output

```
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))
```

Figure



相关信息

[ST_AsEWKT](#), [ST_Force2D](#), [ST_Force3DM](#), [ST_Force3DZ](#)

7.5.9 ST_Force3DZ

ST_Force3DZ — 强制几何图形进入 XYZ 模式。

Synopsis

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

描述

强制几何图形进入 XYZ 模式。如果几何体没有 Z 分量，会附加 Z 的 Z 坐标。

增加：2.0.0 引入了多面体曲面的支持。

更改：2.1.0。在 2.0.x 期，它被称为 ST_Force_3DZ。

更改：3.1.0。您现在可以指定一个非零 Z 值。

- ✓ 函数支持多面体曲面。
- ✓ 函数支持 3d 并且不会丢失 z-index。
- ✓ 此方法支持图形字符串和曲线。

示例

Nothing happens to an already 3D geometry.

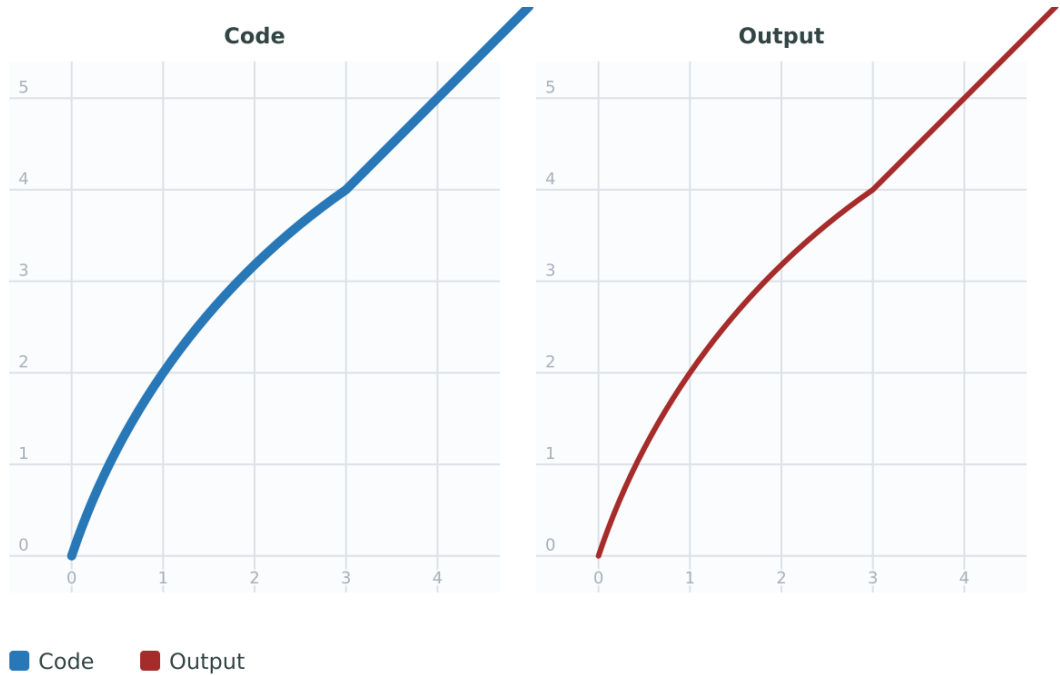
Code

```
SELECT ST_Force3DZ('CIRCULARSTRING(1 1 2,2 3 2,4 5 2,6 7 2,5 6 2)');
```

Output

```
CIRCULARSTRING(1 1 2,2 3 2,4 5 2,6 7 2,5 6 2)
```

Figure



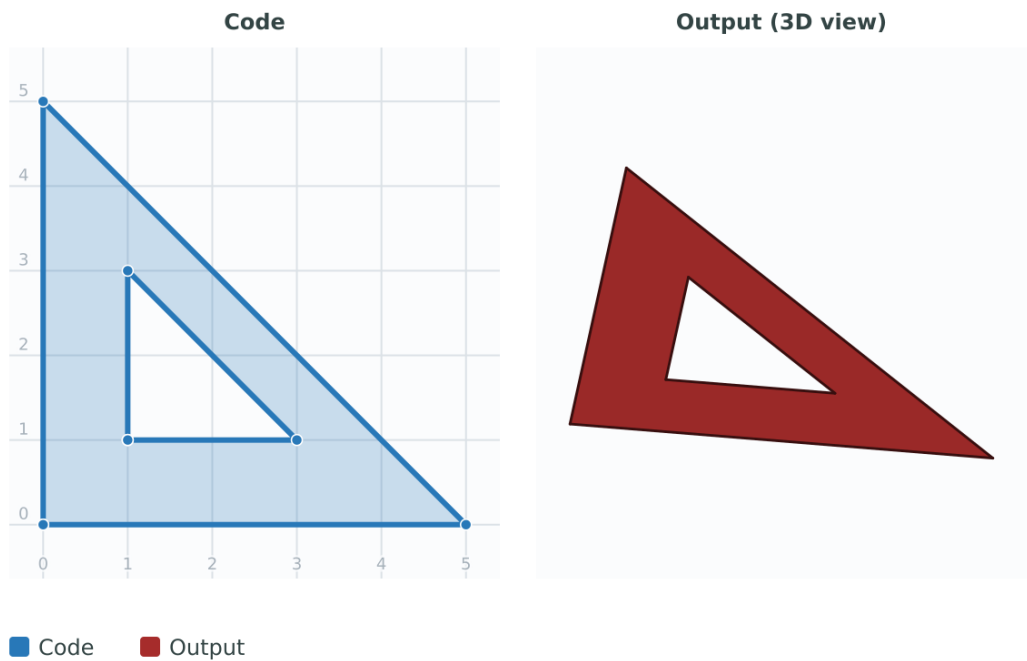
Code

```
SELECT ST_Force3DZ('POLYGON((0 0,0 5,5 0,0 0),(1 1,3 1,1 3,1 1))');
```

Output

```
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))
```

Figure



相关信息

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

7.5.10 ST_Force3DM

ST_Force3DM — 强制几何体形入 XYM 模式。

Synopsis

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

描述

强制几何体形入 XYM 模式。如果几何体没有 M 分量，附加 M 坐标。如果它有 Z 分量，Z 被移除
更改：2.1.0。在 2.0.x 期，它被称为 ST_Force_3DM。
更改：3.1.0。现在可以指定一个非零 M 值。

 此方法支持几何字符串和曲线。

示例

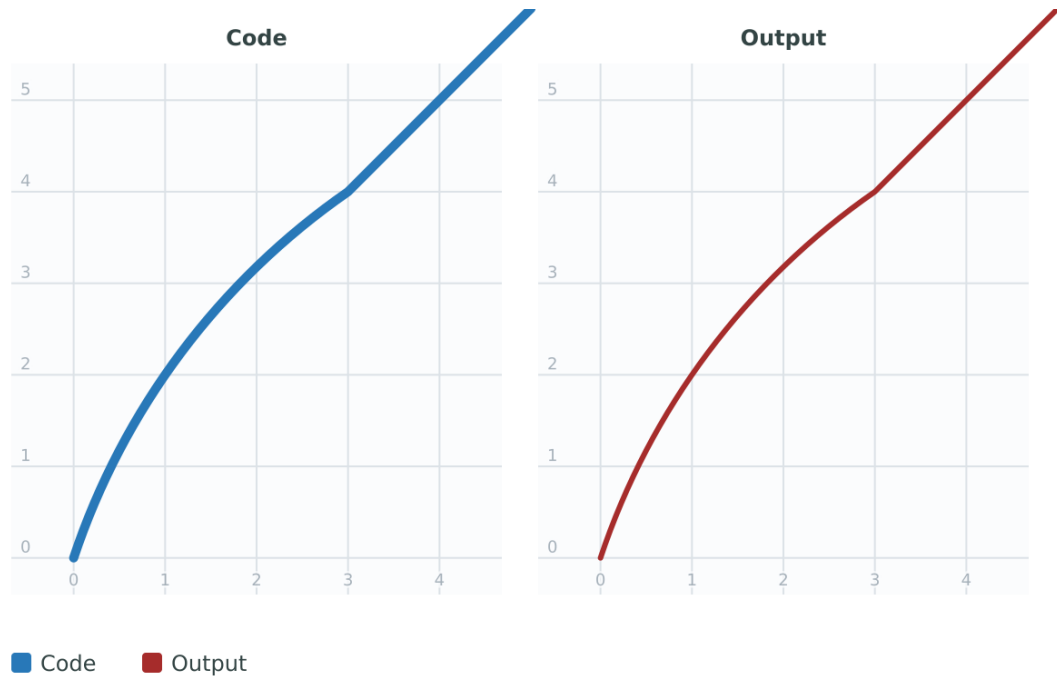
Nothing happens to an already 3D geometry.

Code

```
SELECT ST_Force3DM('CIRCULARSTRING(1 1 2,2 3 2,4 5 2,6 7 2,5 6 2)');
```

Output

```
CIRCULARSTRINGM(1 1 0,2 3 0,4 5 0,6 7 0,5 6 0)
```

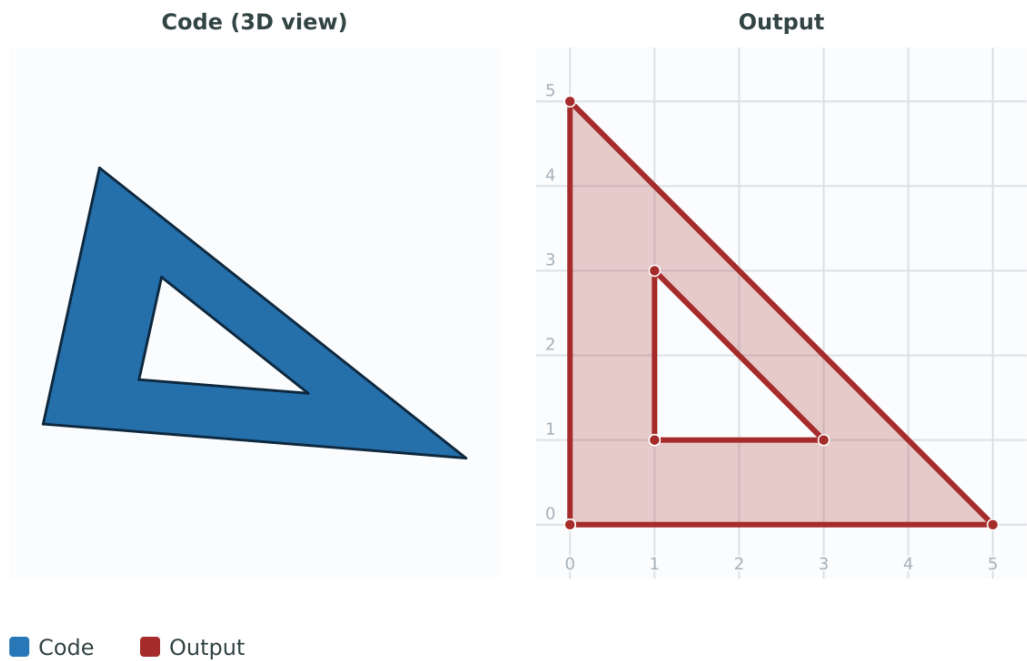
Figure**Code**

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

Output

```
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))
```

Figure



相关信息

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

7.5.11 ST_Force4D

ST_Force4D — 强制几何体进入 XYZM 模式。

Synopsis

geometry **ST_Force4D**(geometry geomA, float Zvalue = 0.0, float Mvalue = 0.0);

描述

强制几何体进入 XYZM 模式。如果没有 Z 或 M，添加 Z 和 M。

更改：2.1.0。在 2.0.x 期，它被称为 ST_Force_4D。

更改：3.1.0。现在可以指定非零 Z 和 M。

- ✓ 函数支持 3d 并且不会丢失 z-index。
- ✓ 此方法支持几何字符串和曲线。

示例

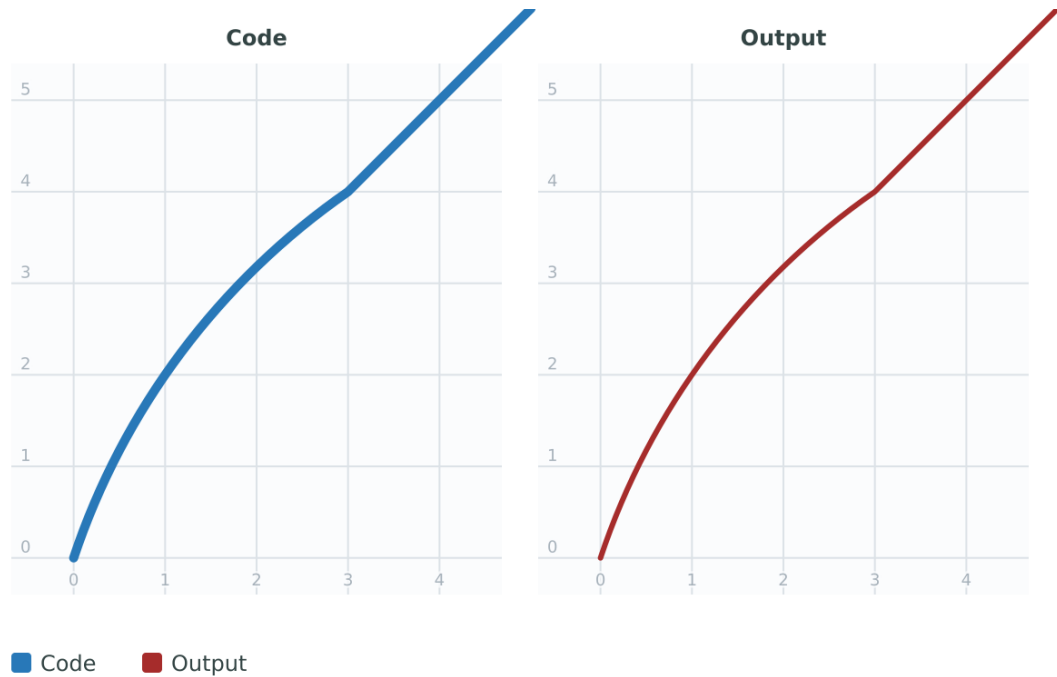
Nothing happens to an already 3D geometry.

Code

```
SELECT ST_Force4D('CIRCULARSTRING(1 1 2,2 3 2,4 5 2,6 7 2,5 6 2)');
```

Output

```
CIRCULARSTRING(1 1 2 0,2 3 2 0,4 5 2 0,6 7 2 0,5 6 2 0)
```

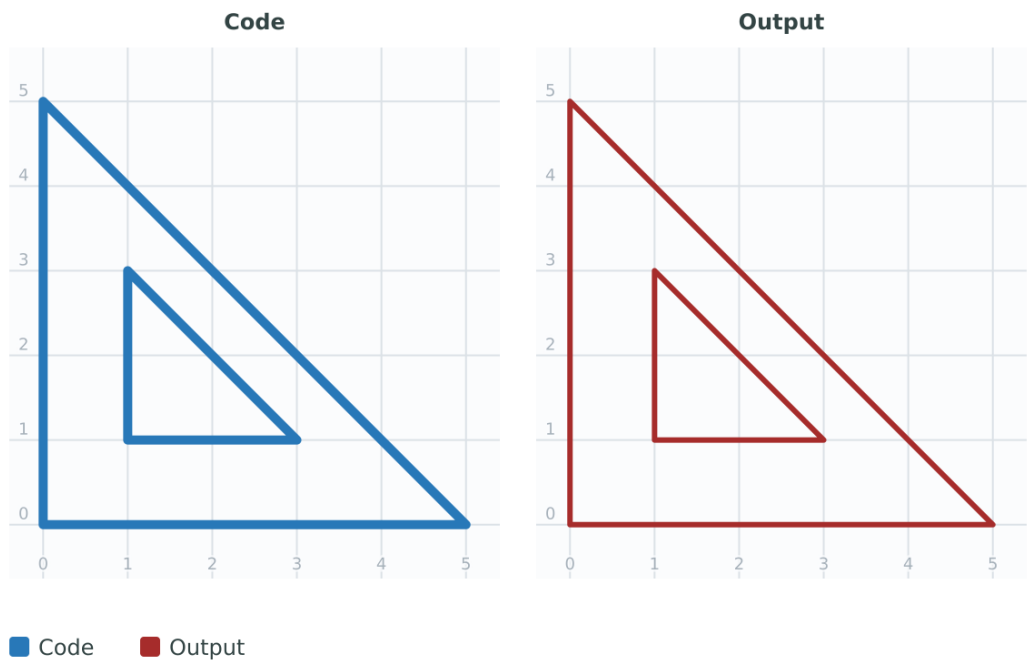
Figure**Code**

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

Output

```
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))
```

Figure



相关信息

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

7.5.12 ST_ForceCollection

`ST_ForceCollection` — 将几何图形强制为几何集合 (GEOMETRYCOLLECTION)。

Synopsis

geometry **ST_ForceCollection**(geometry geomA);

描述

将几何图形强制为几何集合。强制为 WKB 表示很有用。

增修：2.0.0 引入了对多面体曲面的支持。

可用性：1.2.2，在 1.3.4 之前，当与包含曲线的几何一起使用时，它会崩溃。此问题已在 1.3.4 及更高版本中得到修正

更改：2.1.0。在 2.0.x 期间，它被称为 `ST_Force_Collection`。

- ✓ 该函数支持多面体曲面。
- ✓ 该函数支持 3d 并且不会丢失 z-index。
- ✓ 此方法支持几何字符串和曲线。

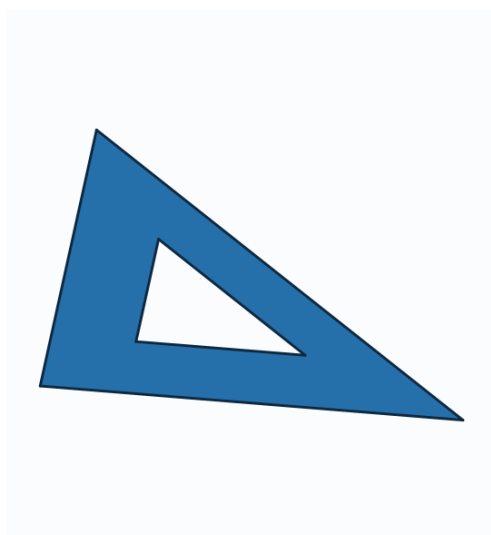
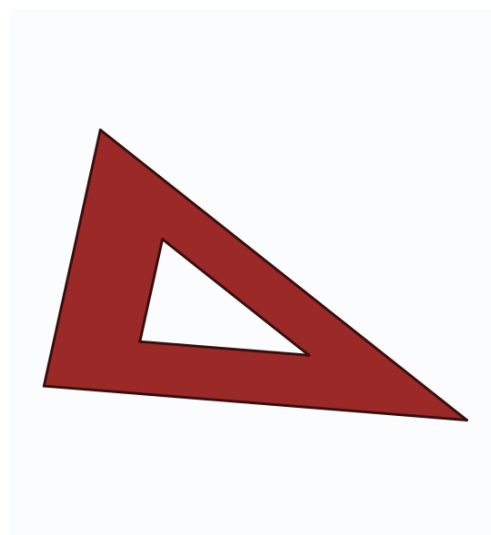
示例

Code

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

Output

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

Figure**Code (3D view)****Output (3D view)**

■ Code ■ Output

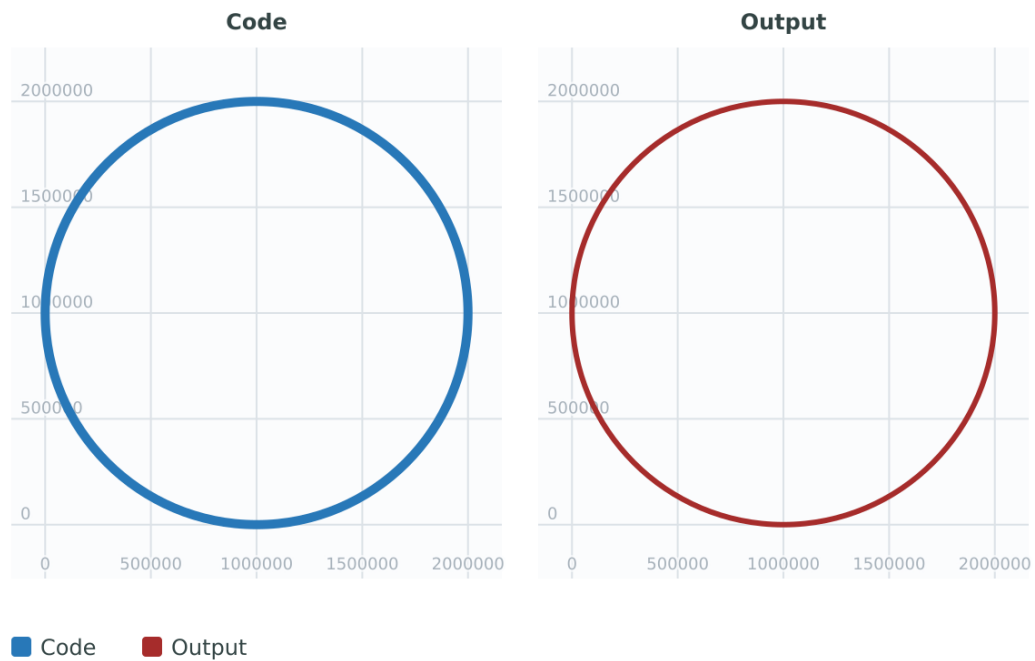
Code

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

Output

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

Figure



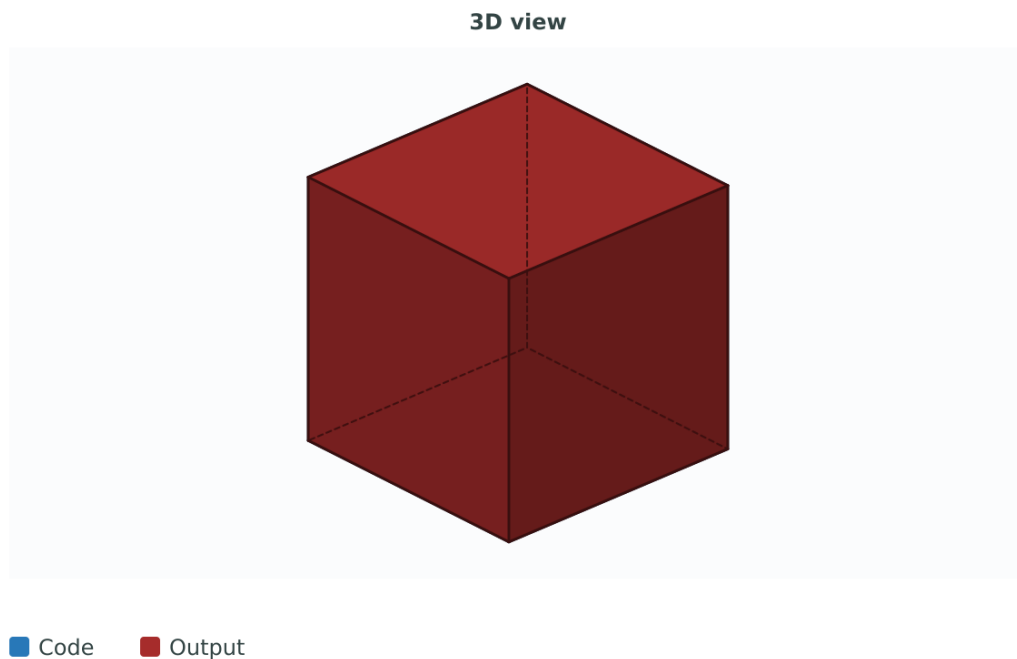
This example uses a PolyhedralSurface.

Code

```
SELECT ST_ForceCollection('POLYHEDRALSURFACE(((0 0 0,0 0 1,0 1 1,0 1 0,0 0 0)),
((0 0 0,0 1 0,1 1 0,1 0 0,0 0 0)),
((0 0 0,1 0 0,1 0 1,0 0 1,0 0 0)),
((1 1 0,1 1 1,1 0 1,1 0 0,1 1 0)),
((0 1 0,0 1 1,1 1 1,1 1 0,0 1 0)),
((0 0 1,1 0 1,1 1 1,0 1 1,0 0 1)))');
```

Output

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

Figure

相关信息

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

7.5.13 ST_ForceCurve

`ST_ForceCurve` — 如果适用，将一个几何图形上升到其曲面型。

Synopsis

geometry **ST_ForceCurve**(geometry g);

描述

将几何图形转换为其曲面表示形式（如果适用）：直线复合曲面，多面体多曲面，多边形曲面多边形，多重多边形多重曲面。如果几何输入已经是曲面表示，则返回与输入相同的几何。

可用性：2.2.0

✓ 该函数支持 3d 并且不会丢失 z-index。

✓ 此方法支持几何字符串和曲面。

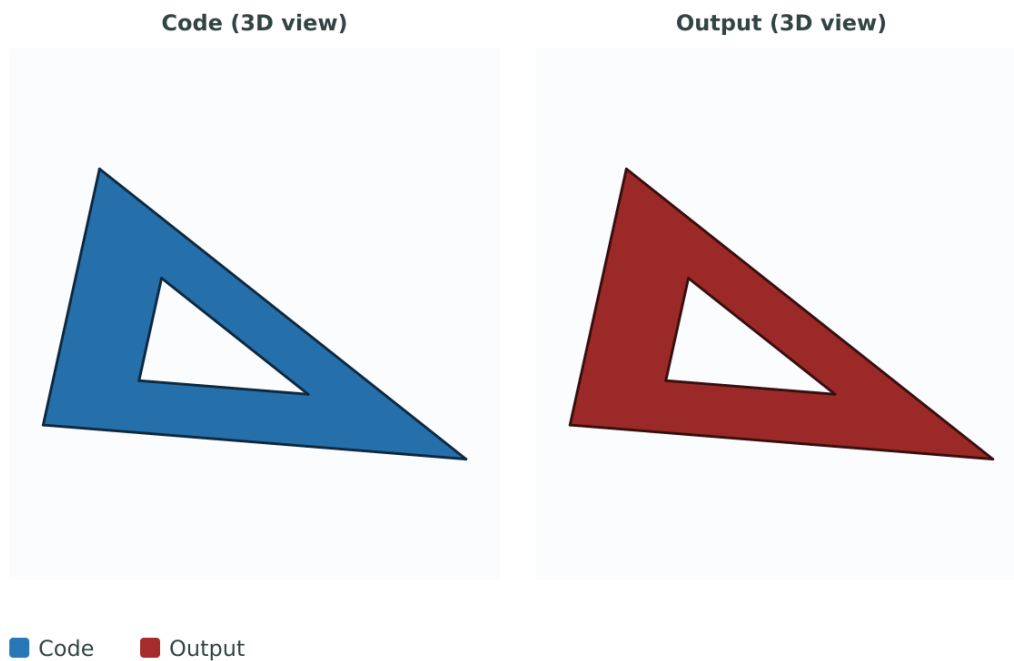
示例

Code

```
SELECT 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
);
```

Output

```
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))
```

Figure

相关信息

[ST_LineToCurve](#)

7.5.14 ST_ForcePolygonCCW

ST_ForcePolygonCCW — 将所有外环逆时针定向，将所有内环顺时针定向。

Synopsis

geometry **ST_ForcePolygonCCW**(geometry geom);

描述

限制（多）多边形其外环使用逆时针方向，其内环使用顺时针方向。非多边形几何保持不变。

可用性：2.4.0

✓ 函数支持 3d 并且不会丢失 z-index。

✓ 功能支持 M 坐标。

示例

Force a polygon and its hole to use counter-clockwise exterior rings and clockwise interior rings.

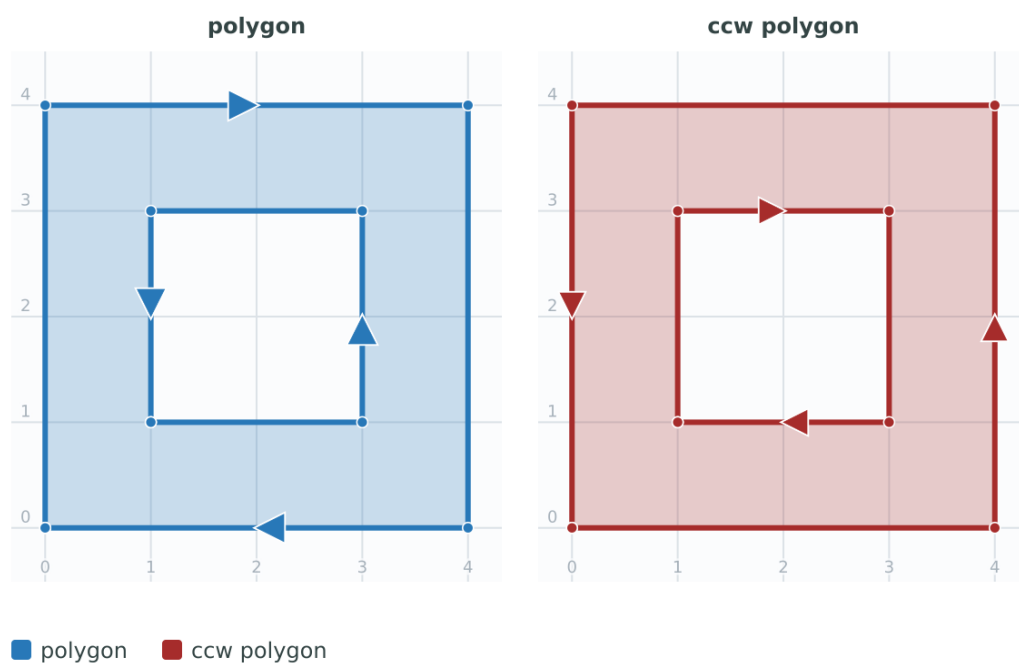
Code

```
WITH data AS (
  SELECT 'POLYGON((0 0,0 4,4 4,4 0,0 0),
    (1 1,3 1,3 3,1 3,1 1))'::geometry AS geom
)
SELECT geom AS input_polygon,
       ST_ForcePolygonCCW(geom) AS ccw_polygon
FROM data;
```

Output

```
POLYGON((0 0,0 4,4 4,4 0,0 0),(1 1,3 1,3 3,1 3,1 1)) | POLYGON((0 0,4 0,4 4,0 4,0 0),(1 1,1 3,3 3,3 1,1 1))
```

Figure



相关信息

[ST_ForcePolygonCW](#), [ST_IsPolygonCCW](#), [ST_IsPolygonCW](#)

7.5.15 ST_ForcePolygonCW

ST_ForcePolygonCW — 强制所有外环为逆时针方向，强制所有内环为顺时针方向。

Synopsis

geometry **ST_ForcePolygonCW**(geometry geom);

描述

强制（多）多边形使其外环使用顺时针方向，其内环使用逆时针方向。非多边形几何体原封不动地返回。

可用性：2.4.0

- ✓ 函数支持 3d 并且不会丢失 z-index。
- ✓ 功能支持 M 坐标。

示例

Force a polygon and its hole to use clockwise exterior rings and counter-clockwise interior rings.

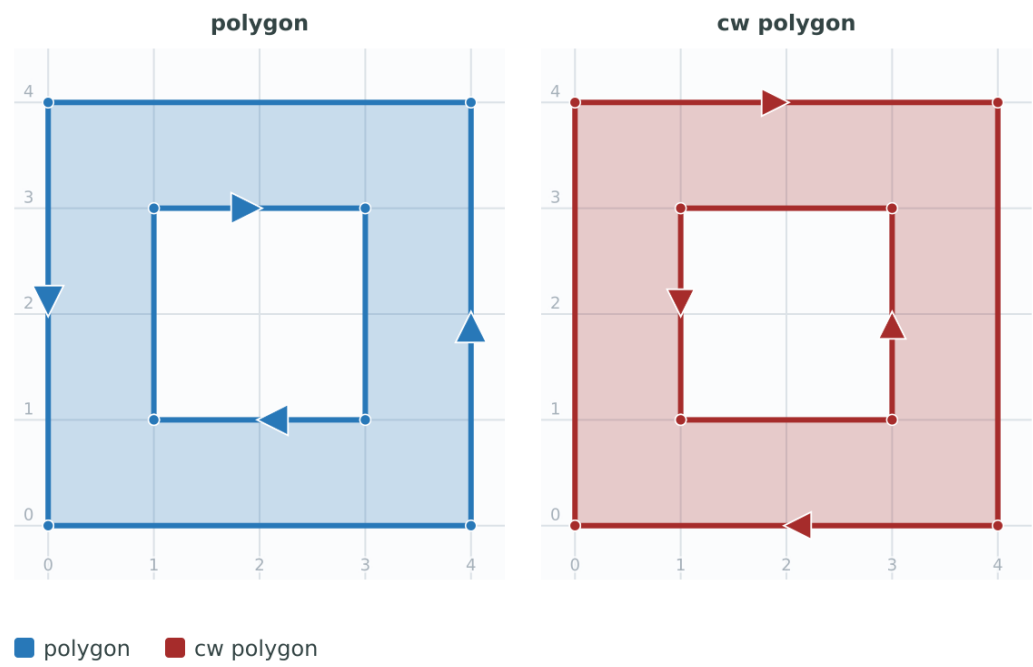
Code

```
WITH data AS (  
  SELECT 'POLYGON((0 0,4 0,4 4,0 4,0 0),  
                  (1 1,1 3,3 3,1 1))'::geometry AS geom  
)  
SELECT geom AS input_polygon,  
       ST_ForcePolygonCW(geom) AS cw_polygon  
FROM data;
```

Output

POLYGON((0 0,4 0,4 4,0 4,0 0),(1 1,1 3,3 3,1 1)) | POLYGON((0 0,0 4,4 4,4 0,0 0),(1 1,3 1,3 3,1 3,1 1))

Figure



相关信息

[ST_ForcePolygonCCW](#), [ST_IsPolygonCCW](#), [ST_IsPolygonCW](#)

7.5.16 ST_ForceSFS

ST_ForceSFS — Force geometries to use SFS 1.1 or 1.2 geometry types.

Synopsis

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

描述

Forces a geometry to use the geometry types defined by the requested version of the OGC Simple Features Specification. The default is SFS 1.1. Curved geometries are converted to linear representations, TRIANGLE is converted to POLYGON, and TIN and POLYHEDRALSURFACE are converted to GEOMETRYCOLLECTION values containing POLYGON elements.

If *version* begins with 1.2, curved geometries are still converted to linear representations, but TRIANGLE, TIN, and POLYHEDRALSURFACE are preserved. Other version values use the default SFS 1.1 behavior. Geometry collections are processed recursively, and the SRID and Z/M coordinates are preserved.

可用性 : 2.1.0

- ☑函数支持多面体曲面。
- ☑此函数支持三角形和不☑☑三角网面 (TIN)。
- ☑此方法支持☑形字符串和曲☑。
- ☑☑函数支持 3d 并且不会☑失 z-index。

示例

Compare the geometry types produced for SFS 1.1 (the default) and SFS 1.2. Read each row from the input type in the direction of the result columns.

Code

```
WITH inputs(geom) AS (VALUES
    ('CIRCULARSTRING(0 0, 1 1, 2 0)::geometry),
    ('TRIANGLE((0 0, 0 2, 2 0, 0 0))::geometry),
    ('TIN(((0 0, 0 2, 2 0, 0 0)))::geometry),
    ('POLYHEDRALSURFACE(((0 0, 0 2, 2 0, 0 0)))::geometry)
)
SELECT ST_GeometryType(geom) || ' b''->b'' AS input,
       ST_GeometryType(ST_ForceSFS(geom)) AS "SFS 1.1 (default)",
       ST_GeometryType(ST_ForceSFS(geom, '1.2')) AS "SFS 1.2"
FROM inputs;
```

Output

input	SFS 1.1 (default)	SFS 1.2
ST_CircularString b''->b''	ST_LineString	ST_LineString
ST_Triangle b''->b''	ST_Polygon	ST_Triangle
ST_Tin b''->b''	ST_GeometryCollection	ST_Tin
ST_PolyhedralSurface b''->b''	ST_GeometryCollection	ST_PolyhedralSurface
(4 rows)		

相关信息

[ST_CurveToLine](#), [ST_ForceCurve](#)

7.5.17 ST_ForceRHR

ST_ForceRHR — 强制多边形点的方向遵循右手定则。

Synopsis

geometry **ST_ForceRHR**(geometry g);

描述

强制多边形中点的方向遵循右手定则，其中多边形所包围的区域位于界的右侧。特别地，外沿沿逆时针方向定向，而内沿沿顺时针方向定向。该函数是 [ST_ForcePolygonCW](#) 的同义词。



Note

右手定则的上述定义与其他上下文中使用的定义相冲突。为避免混淆，建议使用 ST_ForcePolygonCW。

增修：2.0.0 引入了多面体曲面的支持。

- ✓ 该函数支持 3d 并且不会丢失 z-index。
- ✓ 该函数支持多面体曲面。

示例

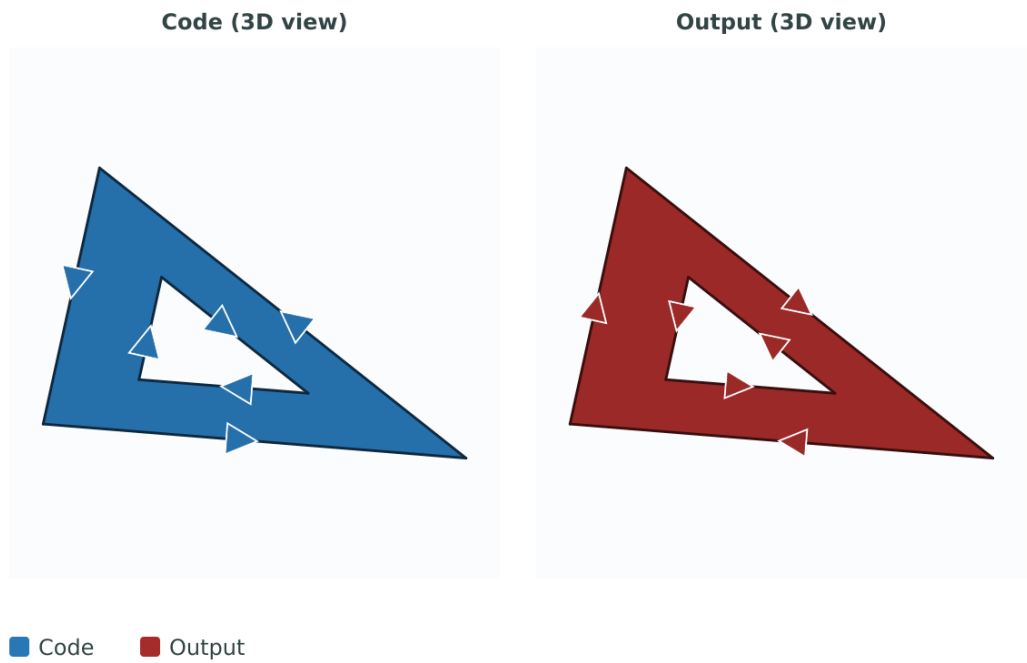
Code

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

Output

```
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))
```

Figure



相关信息

[ST_ForcePolygonCCW](#), [ST_ForcePolygonCW](#), [ST_IsPolygonCCW](#), [ST_IsPolygonCW](#), [ST_BuildArea](#), [ST_Polygon](#), [ST_Reverse](#)

7.5.18 ST_LineExtend

ST_LineExtend — 返回一条线，向前和向后延伸指定的距离。

Synopsis

geometry **ST_LineExtend**(geometry line, float distance_forward, float distance_backward=0.0);

描述

返回一条线，通过在指定的距离添加新的起始点（和结束点），向前和向后延伸。距离为零不会添加点。只允许非零距离。所添加点的方向由线的第一个（和最后一个）不同的点确定。重复的点将被忽略。

可用性：3.4.0

例如：将一条线向前延伸 5 个单位，向后延伸 6 个单位

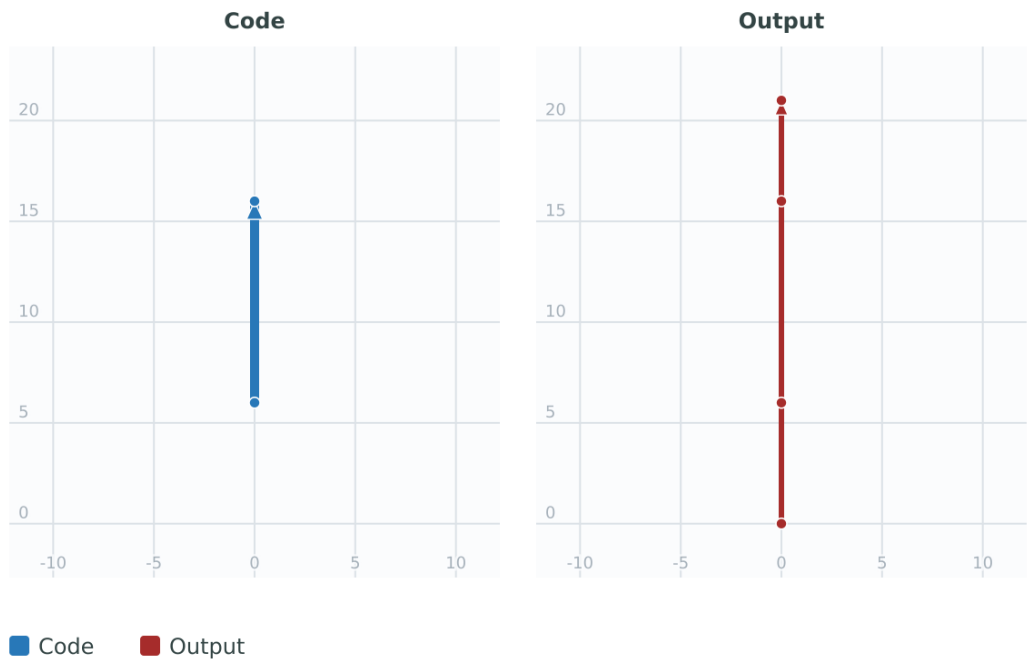
Code

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

Output

```
LINESTRING(0 -6,0 0,0 10,0 15)
```

Figure



相关信息

[ST_LineSubstring](#), [ST_LocateAlong](#), [ST_Project](#)

7.5.19 ST_LineToCurve

ST_LineToCurve — 将线性几何图形转换为曲线几何图形。

Synopsis

geometry **ST_LineToCurve**(geometry geomANoncircular);

描述

将普通多边形/多边形和曲线多边形。注意，描述等效曲线所需的点要少得多。

**Note**

如果输入串/多边形不足以清楚地表示曲线，则函数返回与输入几何图形相同的。

可用性：1.3.0

- ✓ 函数支持 3d 并且不会丢失 z-index。
- ✓ 此方法支持图形字符串和曲线。

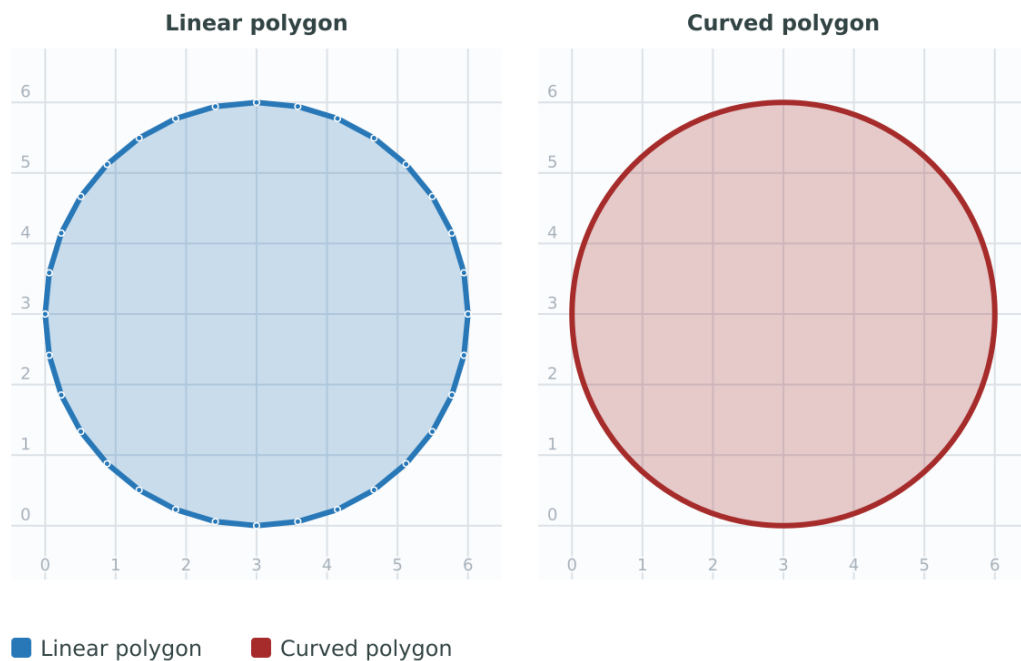
示例

This example converts a linear Polygon approximating a circle to a CurvePolygon.

Code

```
WITH data AS (
  SELECT ST_Buffer('POINT(1 3)::geometry', 3) AS geom
)
SELECT geom AS "input_Linear polygon",
       ST_LineToCurve(geom) AS "Curved polygon"
FROM data;
```

Figure

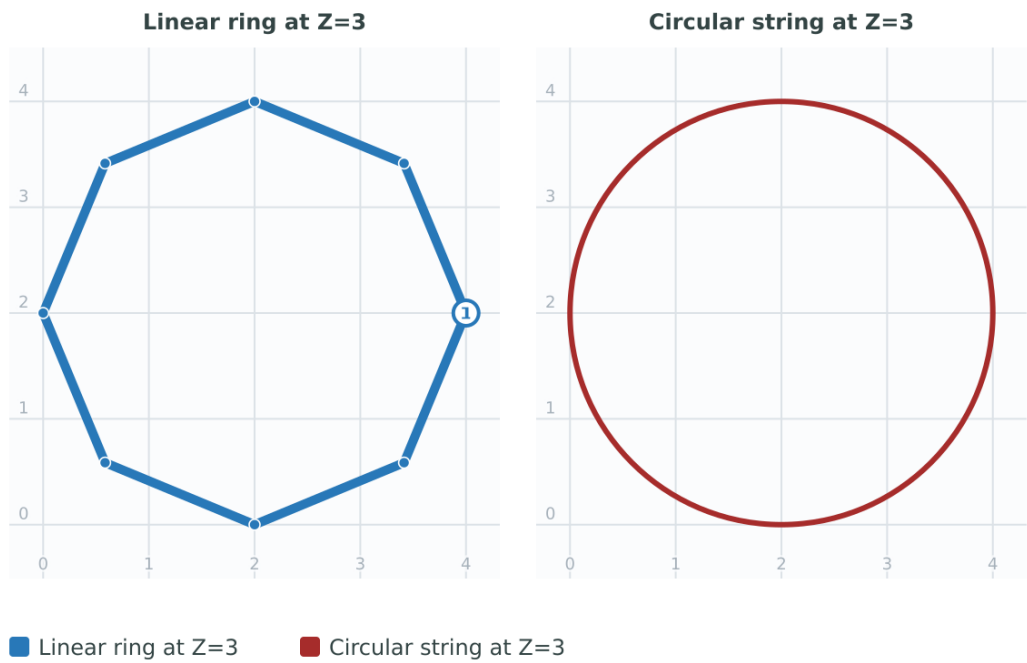


This example converts a 3D linear ring to a CircularString while preserving Z.

Code

```
WITH data AS (
  SELECT ST_Translate(
    ST_Force3D(ST_Boundary(ST_Buffer(ST_Point(1, 3), 2, 2))),
    0, 0, 3
  ) AS geom
)
SELECT geom AS "input_Linear ring at Z=3",
       ST_LineToCurve(geom) AS "Circular string at Z=3"
FROM data;
```

Figure



相关信息

[ST_CurveToLine](#)

7.5.20 ST_Multi

ST_Multi — 将几何☐形返回☐ MULTI* 几何☐形。

Synopsis

geometry **ST_Multi**(geometry geom);

描述

以 MULTI* 几何集合的形式返回几何☐形。如果几何☐形已☐是集合，☐原☐返回。

示例

A Polygon is converted to a single-element MultiPolygon. The coordinates are unchanged.

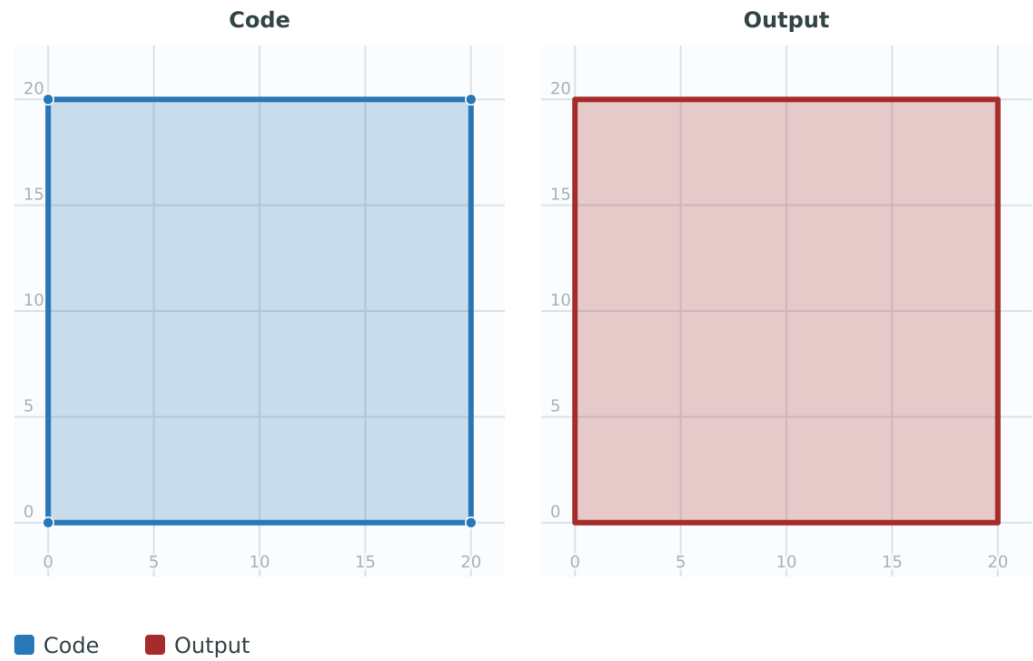
Code

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

Output

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

Figure



相关信息

[ST_AsText](#)

7.5.21 ST_Normalize

ST_Normalize — 返回规范化形式的几何图形。

Synopsis

geometry **ST_Normalize**(geometry geom);

描述

以规范化/标准化格式返回几何图形。多边形中的点顺序、多边形中的顺序或复合几何中元素的顺序可能会更改。在大多数情况下，它用于比较目的（将预期结果与预期结果进行比较）。

可用性：2.3.0

示例

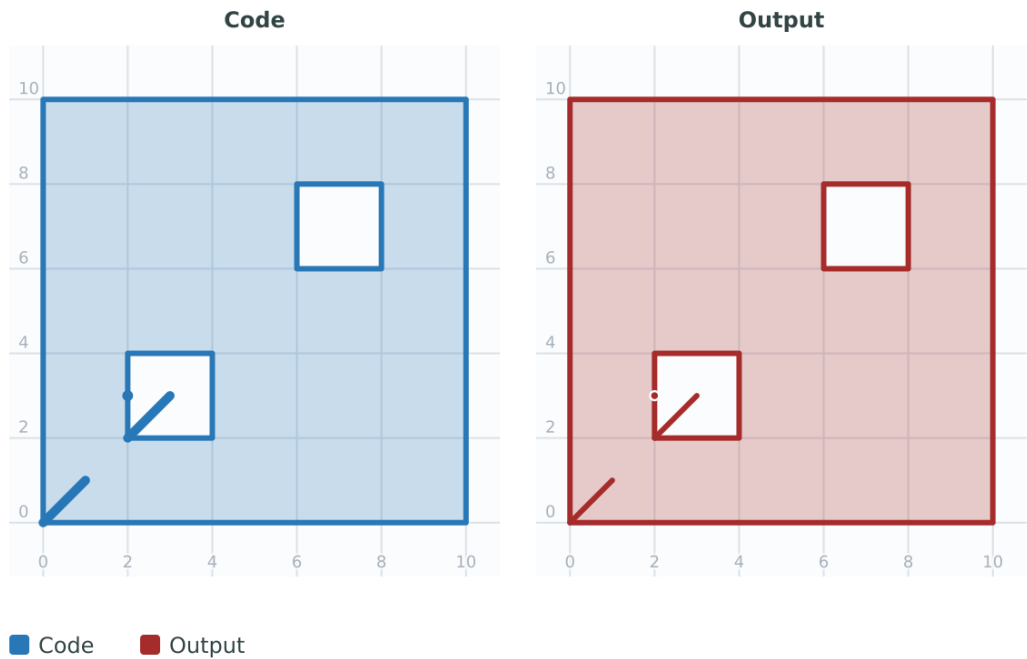
Code

```
SELECT ST_Normalize('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)
  )
)');
```

Output

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

Figure



相关信息

[ST_Equals](#),

7.5.22 ST_Project

ST_Project — 返回从起点按距离和方位角（方位角）投影的点。

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

返回从具有固定距离的起点计算并沿地方位角计算的点。称直接地。

点使用从第一个点到第二个点的路径来式定方位角，并像先前一使用距离。

距离以米位。支持。

方向角（也称航向或方位角）以弧度出。它是从正北方向量的。

- 正北是零度方位角（0 度）
- 正的方位角是 $\pi/2$ （90 度）
- 正南方位角是 π （180 度）
- 正西方位角是 $3\pi/2$ （270 度）

支持方位角和大于 2π （360 度）的。

可用性: 2.0.0

增：2.4.0 允许距离和非准化方位角。

增：3.4.0 允许几何参数和无方位角的点格式。

示例：延伸点位于 **100,000** 米、方位角 **45** 度

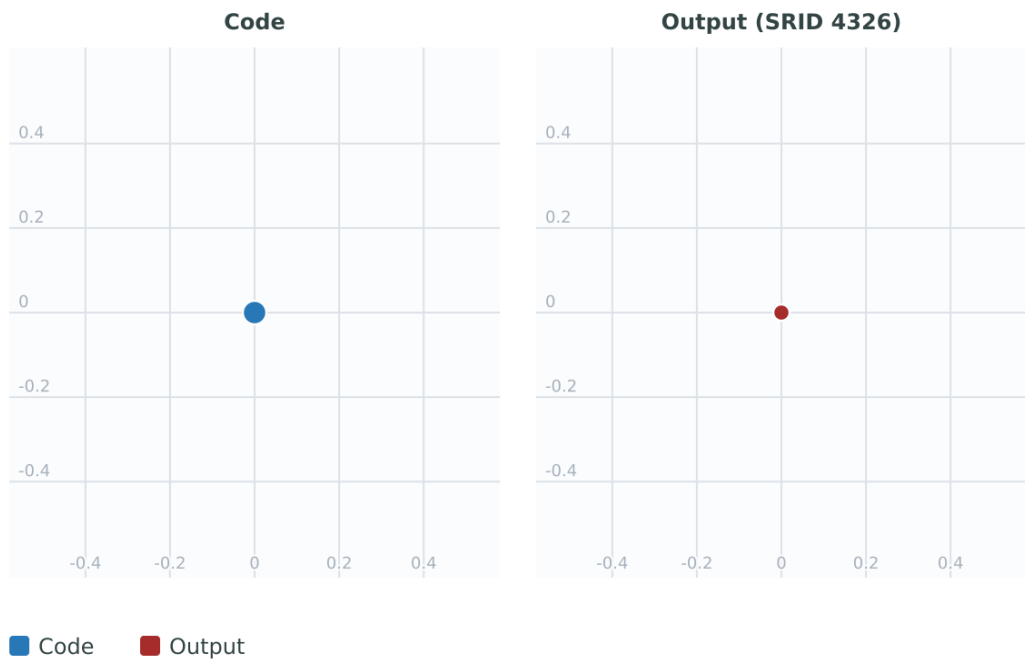
Code

```
SELECT ST_Project('POINT(0 0)::geography', 100000, radians(45.0));
```

Output

```
POINT(0.635231029125537 0.639472334729198)
```

Figure



相关信息

[ST_Azimuth](#), [ST_Distance](#), [PostgreSQL function radians\(\)](#)

7.5.23 ST_QuantizeCoordinates

ST_QuantizeCoordinates — 将坐标的最低有效位位置置零

Synopsis

```
geometry ST_QuantizeCoordinates(geometry g, int prec_x, int prec_y = NULL, int prec_z = NULL,
int prec_m = NULL);
```

描述

ST_QuantizeCoordinates 确定表示小数点后具有指定位数的坐标所需的位数 (N)，然后将除 N 个最高有效位之外的所有位位置置零。生成的坐标仍将舍入原始值，但会提高可精确性。如果几何列使用可精确存储型，可能会致磁盘使用量显著减少。该功能允许在每个维度中指定不同的小数点后位数；未指定的尺寸假定具有 x 维度的精度。函数被解释为指小数点左边的数字（即 `prec_x=-2` 将保留最接近 100 的坐标）。

ST_QuantizeCoordinates 生成的坐标独立于包含这些坐标的几何形状以及这些坐标在几何形状中的相对位置。因此，几何形状之间的拓扑关系不会受到使用此函数的影响。当使用低于几何的固有精度的位数调用函数时，函数可能会生成无效的几何。

可用性：2.5.0

技术背景

PostGIS 将所有坐标存储为双精度浮点整数，可以可靠地表示 15 位有效数字。然而，PostGIS 可用于管理本地上有效数字少于 15 位的数据。一个例子是 TIGER 数据，它以地理坐标的形式提供，小数点后精度为 6 位（因此需要 9 位有效的经度数字和 8 位有效的纬度数字。）

当有 15 位有效数字可用时，具有 9 位有效数字的数字有多种可能的表示形式。双精度浮点数使用 52 个二进制位来表示坐标的有效数（尾数）。只需要 30 位即可表示 9 位有效数字的尾数，剩下 22 位无效位；我们可以将它的位置为我喜欢的任何值，并且最终仍然得到一个四舍五入到我们输入的数字。例如，值 100.123456 可以由最接近 100.1234560000000、100.1234560000001 和 100.123456432199 的浮点数表示。所有这些都同样有效，因此 `ST_AsText(geom, 6)` 将任何这些输入返回相同的结果。由于我们可以将这些位位置任何值，因此 **ST_QuantizeCoordinates** 将 22 个无效位位置置零。对于坐标序列，它会构建零值的模式，PostgreSQL 可以更有效地存储模式。



Note

只有几何形状的磁盘大小可能会受到 **ST_QuantizeCoordinates** 的影响。[ST_MemSize](#) 用于报告几何形状的内存使用情况，无论几何形状使用的磁盘空间如何，都将返回相同的值。

示例

Code

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

Output

POINT(100.12344360351562 0)

Figure



Code

```
WITH test AS (SELECT 'POINT (123.456789123456 123.456789123456)::geometry AS geom)
SELECT
  digits,
  encode(ST_QuantizeCoordinates(geom, digits), 'hex'),
  ST_QuantizeCoordinates(geom, digits)
FROM test, generate_series(15, -15, -1) AS digits;
```

Output

digits	encode	st_astext
15	010100000005f9a72083cdd5e405f9a72083cdd5e40	POINT(123.456789123456 123.456789123456)
14	010100000005f9a72083cdd5e405f9a72083cdd5e40	POINT(123.456789123456 123.456789123456)
13	010100000005e9a72083cdd5e405e9a72083cdd5e40	POINT(123.45678912345599 123.45678912345599)
12	01010000000409a72083cdd5e40409a72083cdd5e40	POINT(123.45678912345556 123.45678912345556)
11	0101000000009a72083cdd5e40009a72083cdd5e40	POINT(123.45678912345466 123.45678912345466)
10	0101000000009872083cdd5e40009872083cdd5e40	POINT(123.45678912344738 123.45678912344738)
9	0101000000008072083cdd5e40008072083cdd5e40	POINT(123.45678912336007 123.45678912336007)
8	010100000000070083cdd5e4000070083cdd5e40	POINT(123.45678912103176 123.45678912103176)
7	010100000000060083cdd5e4000060083cdd5e40	POINT(123.4567891061306 123.4567891061306)
6	010100000000000083cdd5e400000000083cdd5e40	POINT(123.45678901672363 123.45678901672363)
5	010100000000000003cdd5e40000000003cdd5e40	POINT(123.456787109375 123.456787109375)
4	010100000000000003cdd5e40000000003cdd5e40	POINT(123.456787109375 123.456787109375)
3	0101000000000000038dd5e400000000038dd5e40	POINT(123.45654296875 123.45654296875)
2	010100000000000000dd5e40000000000dd5e40	POINT(123.453125 123.453125)
1	010100000000000000dc5e40000000000dc5e40	POINT(123.4375 123.4375)
0	010100000000000000c05e400000000000c05e40	POINT(123 123)
-1	01010000000000000005e4000000000005e40	POINT(120 120)
-2	0101000000000000000584000000000005840	POINT(96 96)
-3	0101000000000000000584000000000005840	POINT(96 96)
-4	0101000000000000000584000000000005840	POINT(96 96)
-5	0101000000000000000584000000000005840	POINT(96 96)

```
-6 | 01010000000000000000000058400000000000005840 | POINT(96 96)
-7 | 01010000000000000000000058400000000000005840 | POINT(96 96)
-8 | 01010000000000000000000058400000000000005840 | POINT(96 96)
-9 | 01010000000000000000000058400000000000005840 | POINT(96 96)
-10 | 01010000000000000000000058400000000000005840 | POINT(96 96)
-11 | 01010000000000000000000058400000000000005840 | POINT(96 96)
-12 | 01010000000000000000000058400000000000005840 | POINT(96 96)
-13 | 01010000000000000000000058400000000000005840 | POINT(96 96)
-14 | 01010000000000000000000058400000000000005840 | POINT(96 96)
-15 | 01010000000000000000000058400000000000005840 | POINT(96 96)
(31 rows)
```

相关信息

[ST_SnapToGrid](#)

7.5.24 ST_RemovePoint

ST_RemovePoint — 从☐串中☐除一个点。

Synopsis

geometry **ST_RemovePoint**(geometry linestring, integer offset);

描述

从☐串中☐除点。索引从零开始。用于将☐合☐☐☐☐开放☐串。

增☐：3.2.0

可用性：1.1.0

 ☐函数支持 3d 并且不会☐失 z-index。

示例

通☐☐除☐合☐（☐）的端点来保☐没有☐合☐。假☐ geom 的☐型☐ LINESTRING

Code

```
UPDATE sometable
SET geom = ST_RemovePoint(geom, ST_NPoints(geom) - 1)
FROM sometable
WHERE ST_IsClosed(geom);
```

相关信息

[ST_AddPoint](#), [ST_NPoints](#), [ST_NumPoints](#)

7.5.25 ST_RemoveRepeatedPoints

ST_RemoveRepeatedPoints — 返回☐除了重复点的几何☐形。

Synopsis

geometry **ST_RemoveRepeatedPoints**(geometry geom, float8 tolerance = 0.0);

描述

返回给定几何图形的版本，其中除了重复的点。该函数处理（多）线串、（多）多边形和多点，但可以使用任何类型的几何图形进行使用。GeometryCollections 的元素是独立处理的。LineStrings 的端点被保留。

如果提供了非零的 *tolerance* 参数，则在容差距离内的点将被视为重复点。距离在二维平面（XY 平面）上计算。

增加：3.2.0

可用性：2.2.0

- ✓ 该函数支持多面体曲面。
- ✓ 该函数支持 3d 并且不会丢失 z-index。

示例

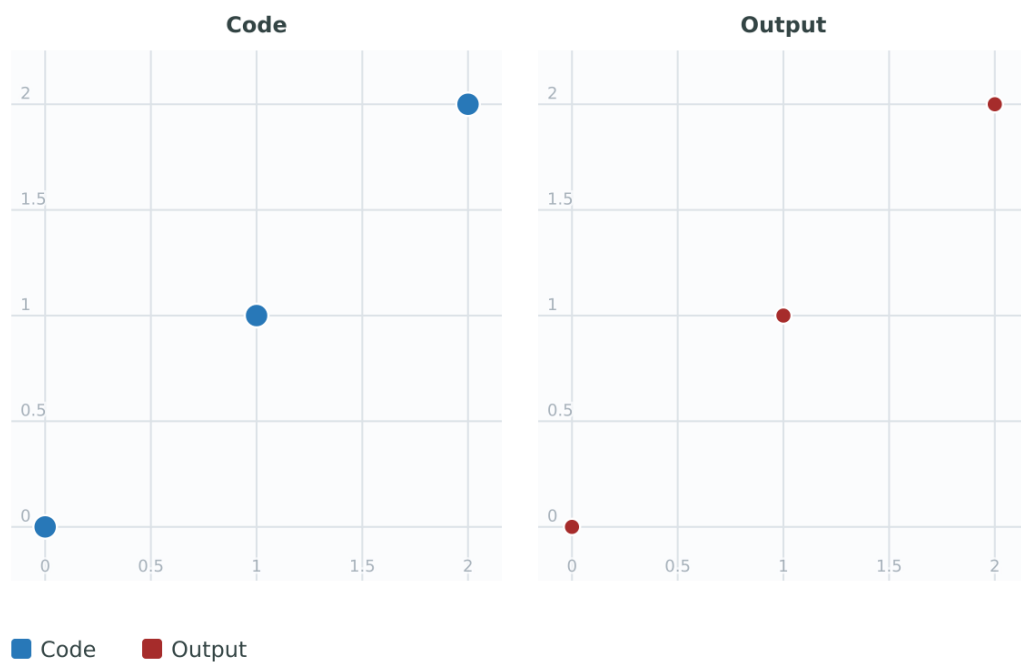
Code

```
SELECT ST_RemoveRepeatedPoints('MULTIPOINT ((1 1),(2 2),(3 3),(2 2))');
```

Output

```
MULTIPOINT((1 1),(2 2),(3 3))
```

Figure



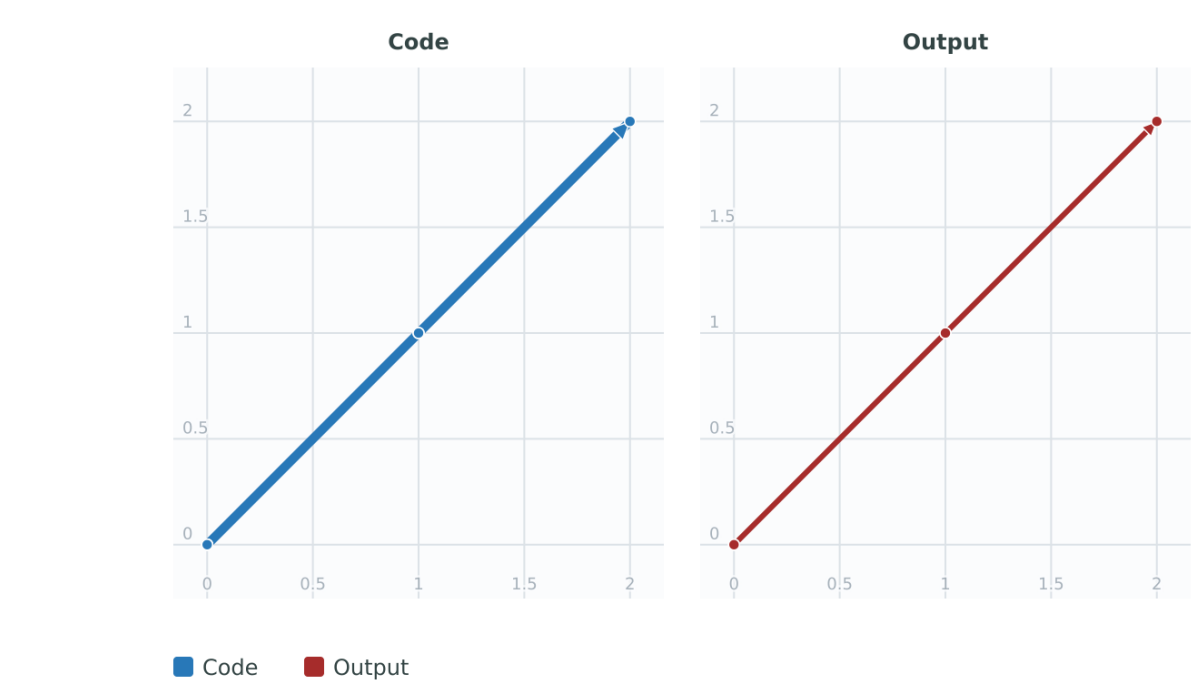
Code

```
SELECT ST_RemoveRepeatedPoints('LINESTRING (0 0,0 0,1 1,0 0,1 1,2 2)');
```

Output

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

Figure



示例：集合元素是互斥的。

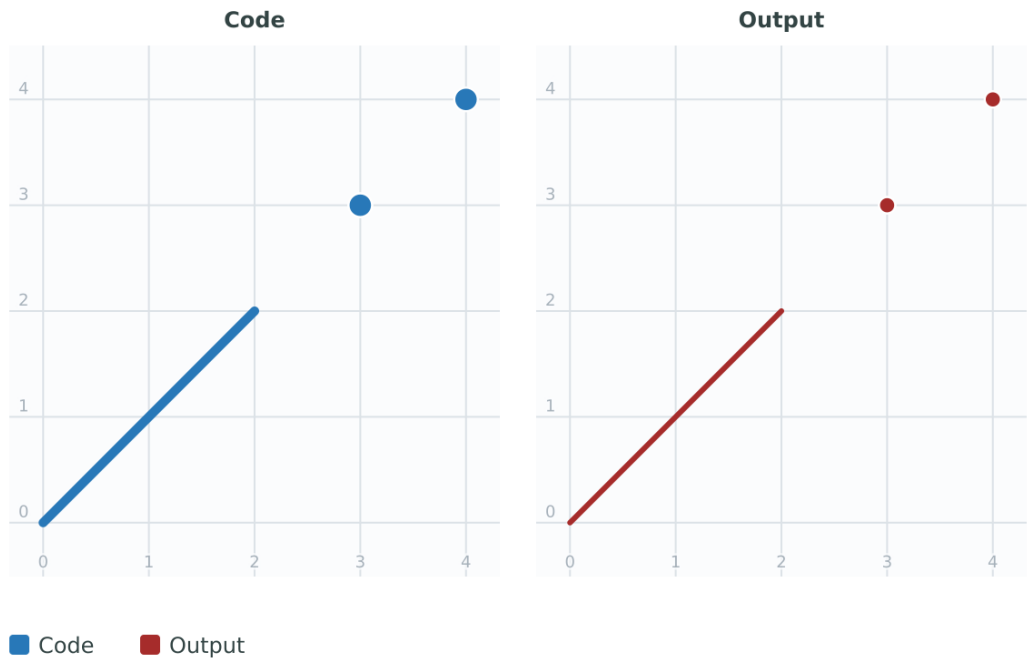
Code

```
SELECT ST_RemoveRepeatedPoints('GEOMETRYCOLLECTION (LINESTRING (1 1,2 2,2 2,3 3),POINT (4 4),POINT (4 4),POINT (5 5))');
```

Output

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

Figure



示例: 除在容差距离内的重复点。

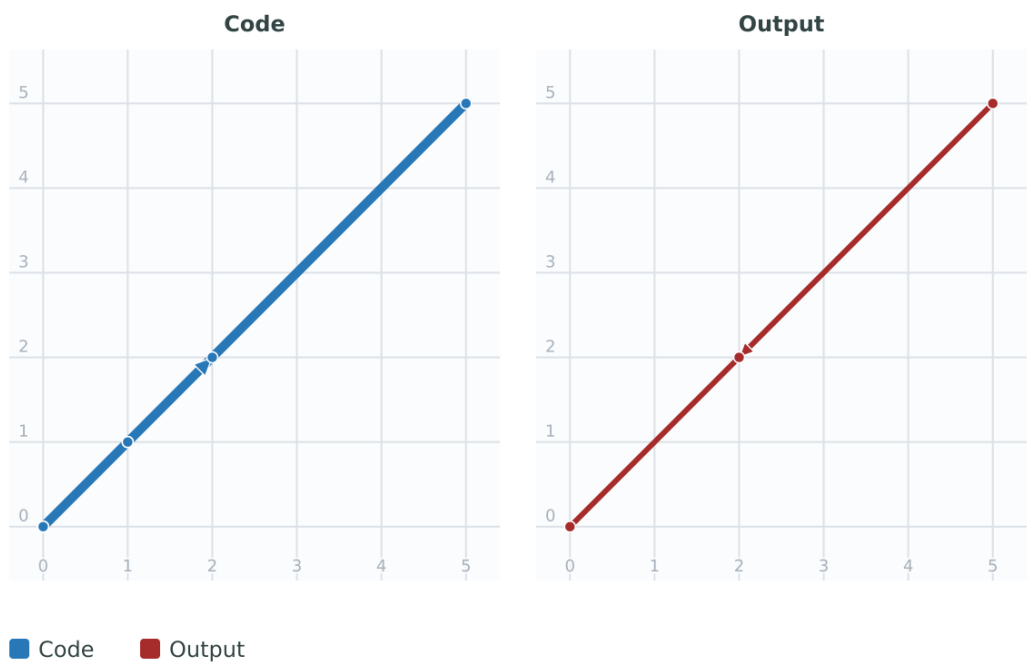
Code

```
SELECT ST_RemoveRepeatedPoints('LINESTRING (0 0,0 0,1 1,5 5,1 1,2 2)', 2);
```

Output

```
LINESTRING(0 0,5 5,2 2)
```

Figure



相关信息

[ST_Simplify](#)

7.5.26 ST_RemoveIrrelevantPointsForView

`ST_RemoveIrrelevantPointsForView` — 删除在指定矩形表示范区内与视制无关的几何图形点。

Synopsis

```
geometry ST_RemoveIrrelevantPointsForView(geometry geom, box2d bounds, boolean cartesian_hint = false);
```

描述

返回一个 **geometry**，移除了在指定矩形表示范区内视制无关的点。

使用此函数可以在特定范区内渲染的几何图形进行快速修剪。

只有 (MULTI) POLYGON 和 (MULTI) LINESTRING 类型的几何图形会被修剪。其他几何图形不会被更改。

该函数与 `ST_ClipByBox2D()` 的区别如下

- 此函数通过点行排序而不计算新的交点，从而避免了舍入误差，并通常提高了效率，
- 返回一个具有相同或接近点数的几何图形，
- 在指定的视制范区内生成相同的视制结果，
- 可能会致自交现象，从而使结果几何图形无效（参下方示例）。

如果将 `cartesian_hint` 置为 `TRUE`，算法会使用笛卡尔的化，包括使用笛卡儿数学，一步减少结果中的点数。需要注意的是，如果结果坐被投影到其他（非笛卡尔）坐标系中，使用此可参数可能会导致视制中的不自然阴影或噪点。



Warning

对于多边形来，当前此函数无法保证结果的有效性。可以通过 `ST_IsValid` 来检查这种情况，并使用 `ST_MakeValid` 来修复。

可用性：3.5.0

示例

The input polygon has 55 points. The result retains 15 points, while rendering the same shape inside the view envelope.

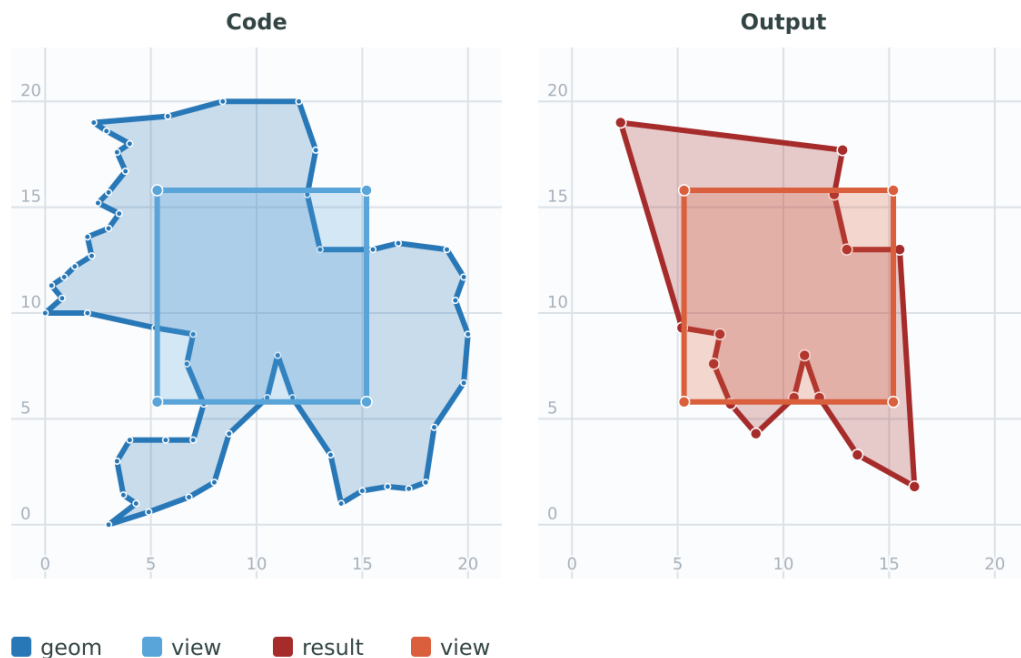
Code


```

WITH data(geom, view) AS (VALUES (
  'POLYGON((3.2 14.2,6.8 14.2,7.6 11.9,7.2 9.8,7.8 7.2,10.3 7.2,11.5 7.5,13.8 7.2,14.6 5.9,14.2 4.8,14.8 3.2,14.6 0.9,13.2 -1.2,12.8 -3.8,12 -4.1,11 -4.9,8 -4.2,8.8 -4.8,8.3 -2.5,6.5 0.2,5.8 2.2,5.3 0.2,3.5 -1.5,2.8 -3.8,1.6 -4.5,-0.3 -5.2,-2.2 -5.8,-0.9 -4.8,-1.5 -4.4,-1.8 -2.8,-1.2 -1.8,0.5 -1.8,1.8 -1.8,2.3 -0.1,1.5 1.8,1.8 3.2,0 3.5,-3.2 4.2,-5.2 4.2,-4.4 4.9,-4.9 5.5,-4.3 5.9,-3.8 6.4,-3 6.9,-3.2 7.8,-2.2 8.2,-1.7 8.9,-2.7 9.4,-2.2 9.9,-1.4 10.9,-1.8 11.8,-1.2 12.2,-2.3 12.8,-2.9 13.2,0.6 13.5,3.2 14.2))':: geometry,
  ST_MakeEnvelope(0.1, 0, 10, 10)
))
SELECT geom AS input_geom,
       view AS input_view,
       ST_RemoveIrrelevantPointsForView(geom, view, true) AS result,
       view AS view
FROM data;

```

Figure



Only linework affecting the view is retained; existing vertices are reused.

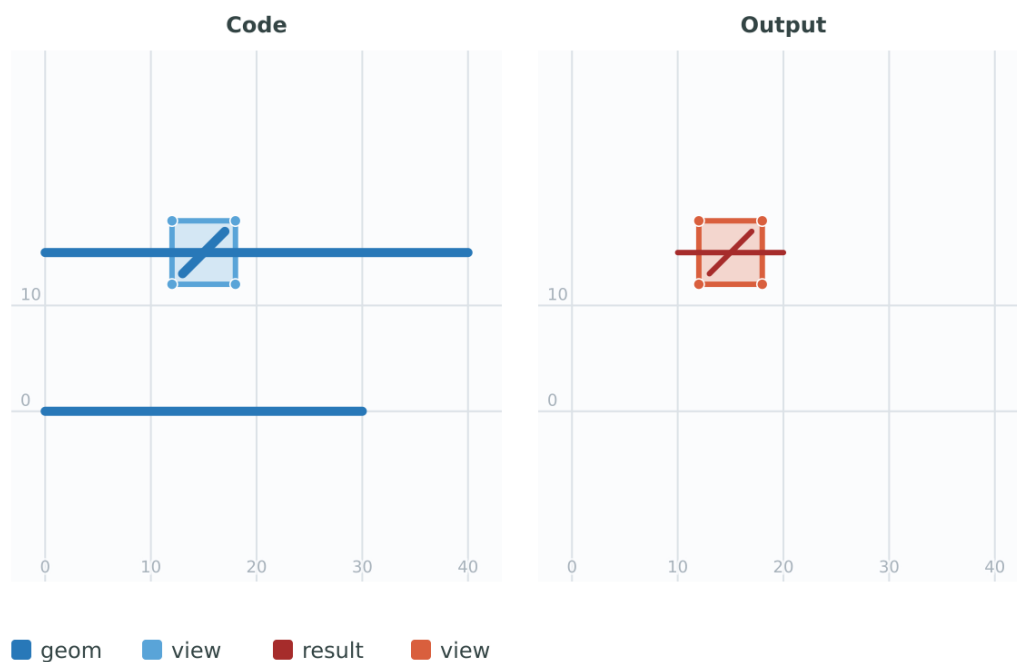
Code

```

WITH data(geom, view) AS (VALUES (
  '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))'::geometry,
  ST_MakeEnvelope(12, 12, 18, 18)
))
SELECT geom AS input_geom,
       view AS input_view,
       ST_RemoveIrrelevantPointsForView(geom, view, true) AS result,
       view AS view
FROM data;

```

Figure



A line with no segment relevant to the view becomes empty.

Code

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

Output

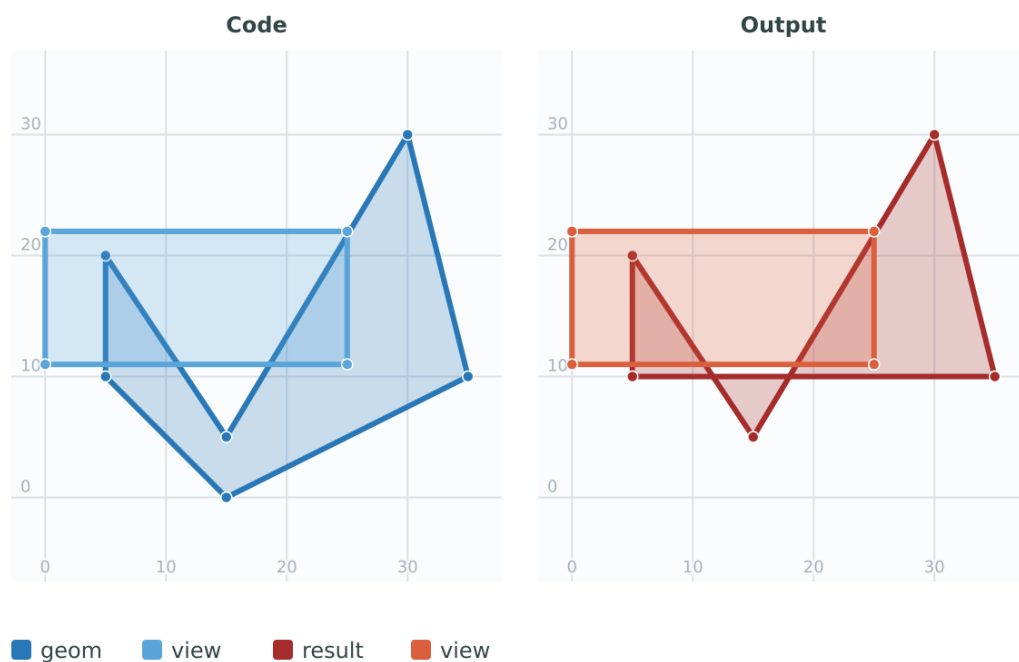
```
LINESTRING EMPTY
```

Removing vertices without computing intersections can introduce self-intersections in the result.

Code

```
WITH data(geom, view) AS (VALUES (
  'POLYGON((0 10,0 0,10 -10,30 0,25 20,10 -5,0 10))'::geometry,
  ST_MakeEnvelope(-5, 1, 20, 12)
))
SELECT geom AS input_geom,
       view AS input_view,
       ST_RemoveIrrelevantPointsForView(geom, view, true) AS result,
       view AS view
FROM data;
```

Figure

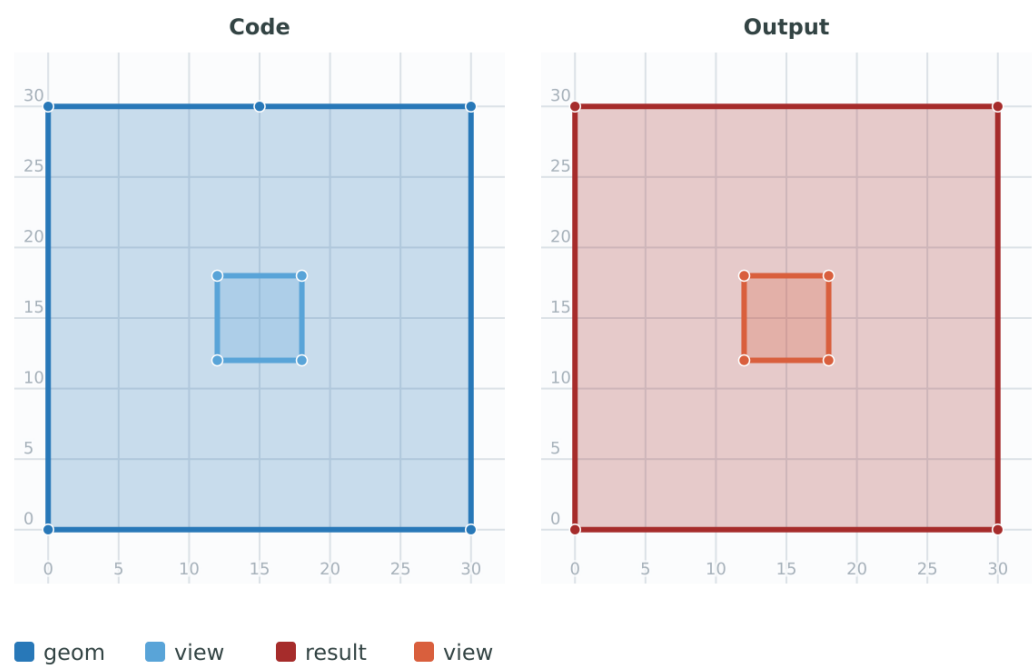


With the default *cartesian_hint*, the function removes only points known to be irrelevant to the view.

Code

```
WITH data(geom, view) AS (VALUES (
  'POLYGON((0 30,15 30,30 30 0,0 0,0 30))'::geometry,
  ST_MakeEnvelope(12, 12, 18, 18)
))
SELECT geom AS input_geom,
       view AS input_view,
       ST_RemoveIrrelevantPointsForView(geom, view) AS result,
       view AS view
FROM data;
```

Figure



相关信息

[ST_ClipByBox2D](#), [ST_Intersection](#)

7.5.27 ST_RemoveSmallParts

`ST_RemoveSmallParts` — 删除几何图形中的所有小部分（多边形或线串）。

Synopsis

geometry **ST_RemoveSmallParts**(geometry geom, double precision minSizeX, double precision minSizeY);

描述

返回不包含小部分（外环、内环或线串）的 **geometry**。

此函数可用于地图化中的清理步骤，例如删除小环或孔洞。

只有 (MULTI)POLYGON 和 (MULTI)LINestring 类型的几何体会被评估。其他几何体不会生成化。

如果 *minSizeX* 大于 0，会分割出宽度小于 *minSizeX* 的部分。

如果 *minSizeY* 大于 0，会分割出高度小于 *minSizeY* 的部分。

minSizeX 和 *minSizeY* 都是以几何体坐系的位行度量的。

对于多边形型，评估是每个环独立行的，可能会得出以下结果之一：

- 原始几何形，

- 所有内环点少的多边形 (POLYGON),
- 内环数量少的多边形 (POLYGON) (可能会少点数),
- POLYGON EMPTY, 或者
- 一个多边形数量少的 MULTIPOLYGON (可能少了内部环的数量或点数), 或者
- MULTIPOLYGON EMPTY。

由于字符串型，估计是每个字符串行的，因此可能会得出以下结果：

- 原始几何图形,
- 一个点数少的 LINESTRING,
- LINESTRING EMPTY,
- 一个字符串数量少的 MULTILINESTRING (可能少了点数), 或者
- MULTILINESTRING EMPTY。

可用性 : 3.5.0

示例

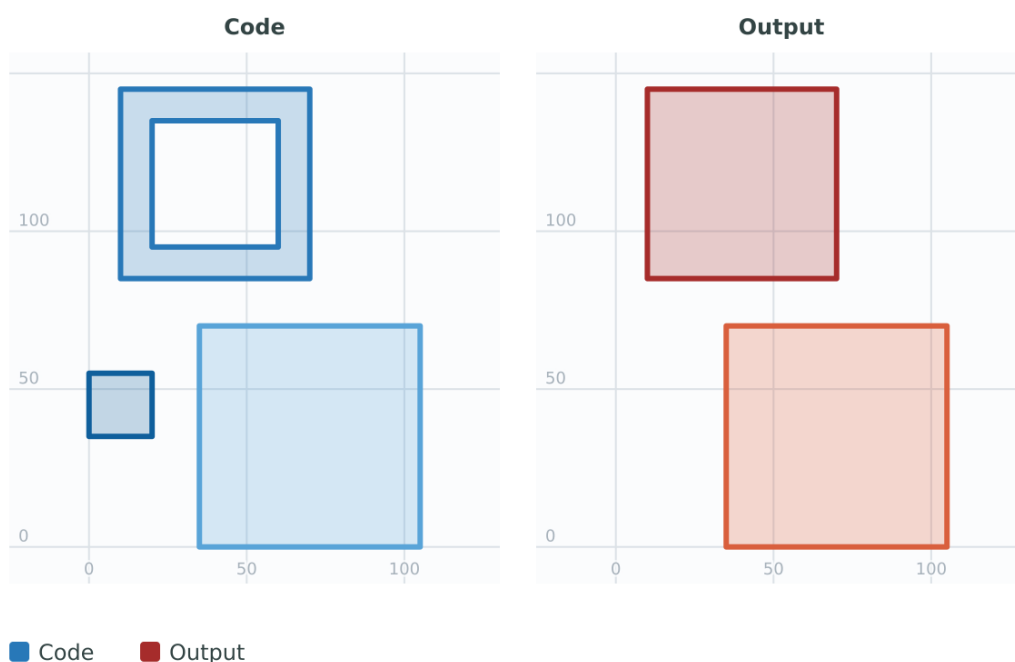
Code

```
SELECT ST_RemoveSmallParts('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);
```

Output

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

Figure



Code

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

Output

```
LINESTRING EMPTY
```

7.5.28 ST_Reverse

ST_Reverse — 返回点序相反的几何体。

Synopsis

geometry **ST_Reverse**(geometry g1);

描述

可用于任何几何体并反转点的序。

增：2.4.0 引入了曲线支持。



函数支持 3d 并且不会丢失 z-index。

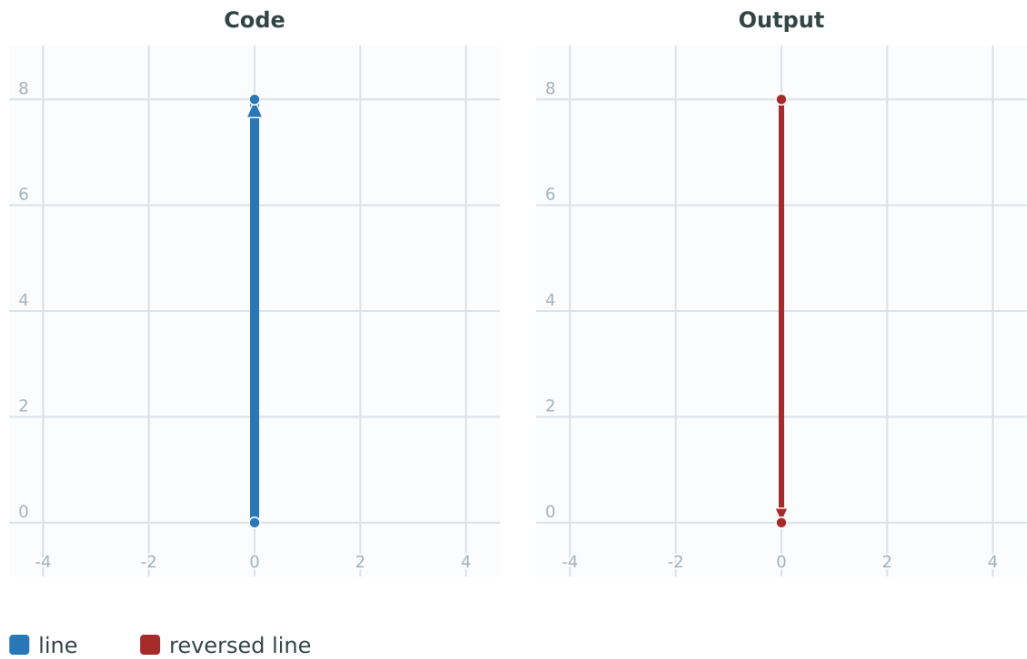


函数支持多面体曲面。

示例**Code**

```
WITH data(geom) AS (VALUES (  
    ST_MakeLine(ST_Point(1, 2), ST_Point(1, 10))  
))  
SELECT geom AS input_line,  
    ST_Reverse(geom) AS reversed_line  
FROM data;
```

Figure



7.5.29 ST_Segmentize

ST_Segmentize — 返回修改后的几何形状/地理，其段不于固定距离。

Synopsis

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

描述

返回修改后的几何形状/地理，其段度不超 max_segment_length。段度以二算。段段是被分成等的子段。

- 于几何形状，最大度以空参考系的位表示。
- 于地理来，最大度以米位。距离是在球体上算的。添加的点是沿着段端点定的球面大弧建的。



Note
只会短段。它不会延比最大度短的段。



Warning
于包含段的入，指定相短的 max_segment_length 可能会致添加大量点。如果意外地将参数指定多个段而不是最大度，可能会无意中生种情况。

可用性：1.2.2

增：3.0.0 分段几何可在可生成等长的子分段

增：2.3.0 地理分段可在可生成等长的子分段

增：2.1.0 引入了地理的支持。

更改：2.1.0 由于引入了地理支持，使用 `ST_Segmentize('LINESTRING(1 2, 3 4)', 0.5)` 会导致不明确的函数。输入需要正确输入几何或地理。使用 `ST_GeomFromText`、`ST_GeogFromText` 或指定所需类型（例如 `ST_Segmentize('LINESTRING(1 2, 3 4)::geometry, 0.5) )`）

示例

分割一条线。线段均分，短段不平均。

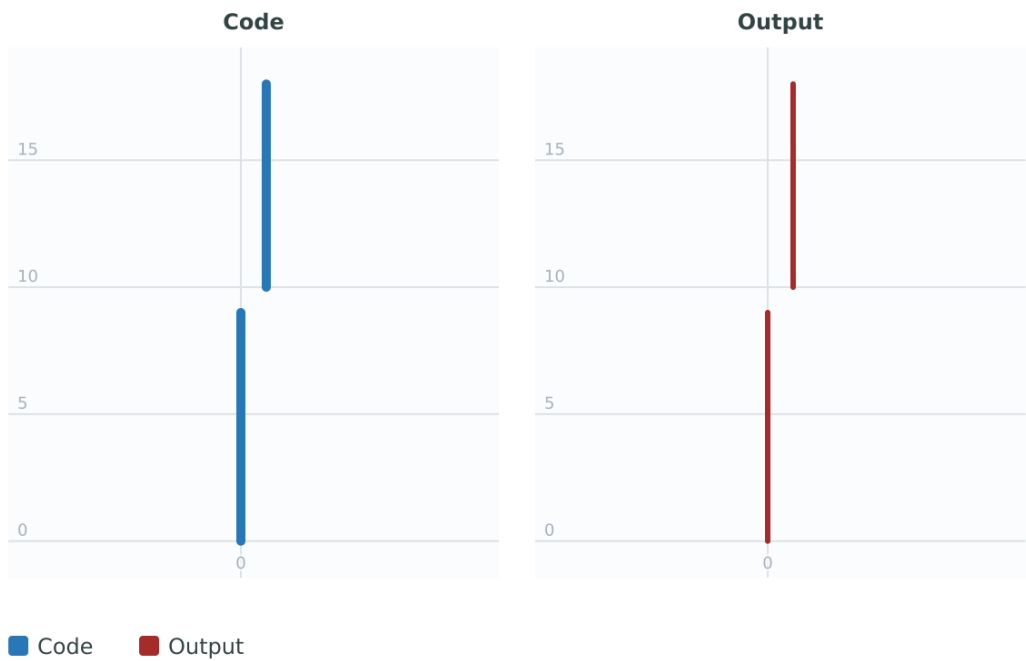
Code

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

Output

```
MULTILINESTRING((0 0,0 1,0 5,0 9),(1 10,1 14,1 18))
```

Figure



多边形分割：

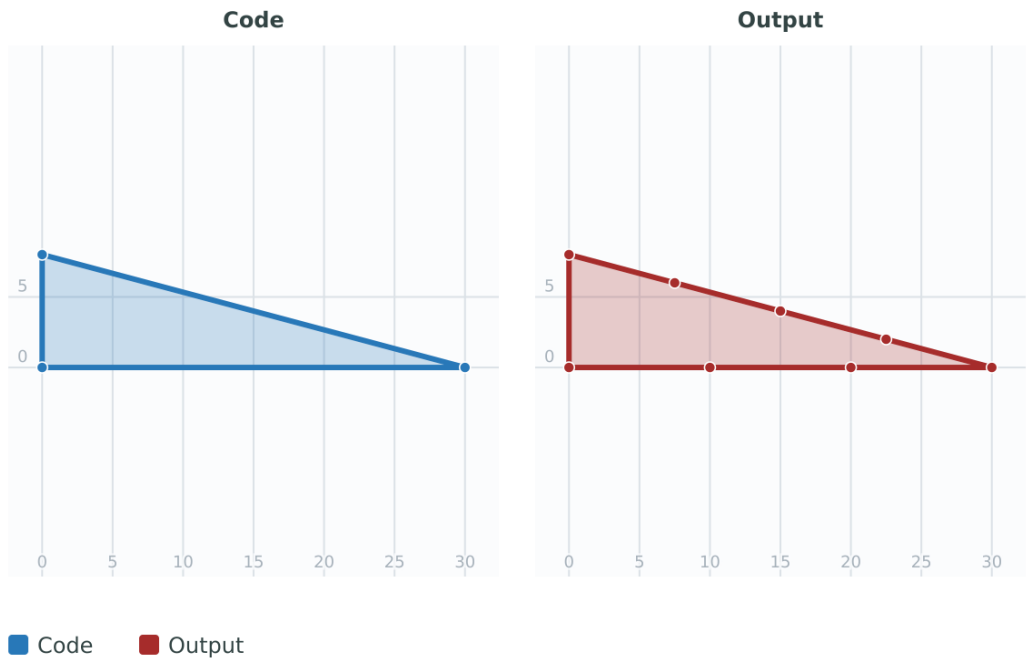
Code

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


Output

```
POLYGON((0 0,0 8,7.5 6,15 4,22.5 2,30 0,20 0,10 0,0 0))
```

Figure



Code Output

地理行分段，最大分段度 2000 公里。沿着接端点的大弧添加点。

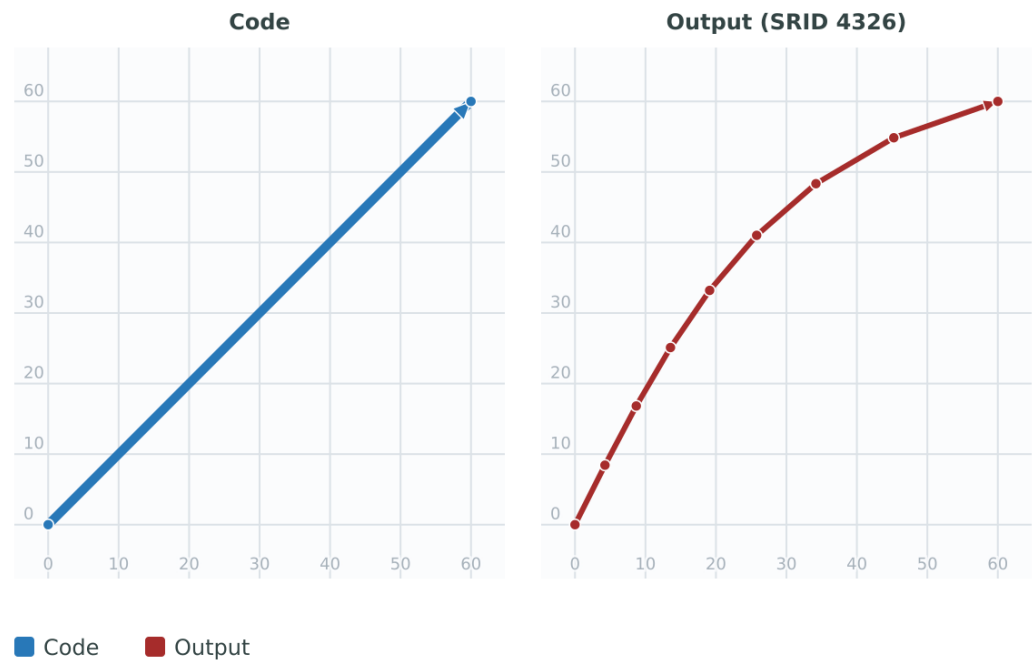
Code

```
SELECT ST_Segmentize('LINESTRING (0 0,60 60)::geography', 2000000);
```

Output

```
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)
```

Figure



相关信息

[ST_LineSubstring](#)

7.5.30 ST_SetPoint

ST_SetPoint — 用指定点替换串的点。

Synopsis

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

描述

将串的 N 点替换指定点。索引从 0 开始。索引向后数，因此 -1 是最后一个点。当一个点移动到保持关系，在触发器中特别有用。

可用性：1.1.0

更新 2.3.0：索引

 函数支持 3d 并且不会丢失 z-index。

示例

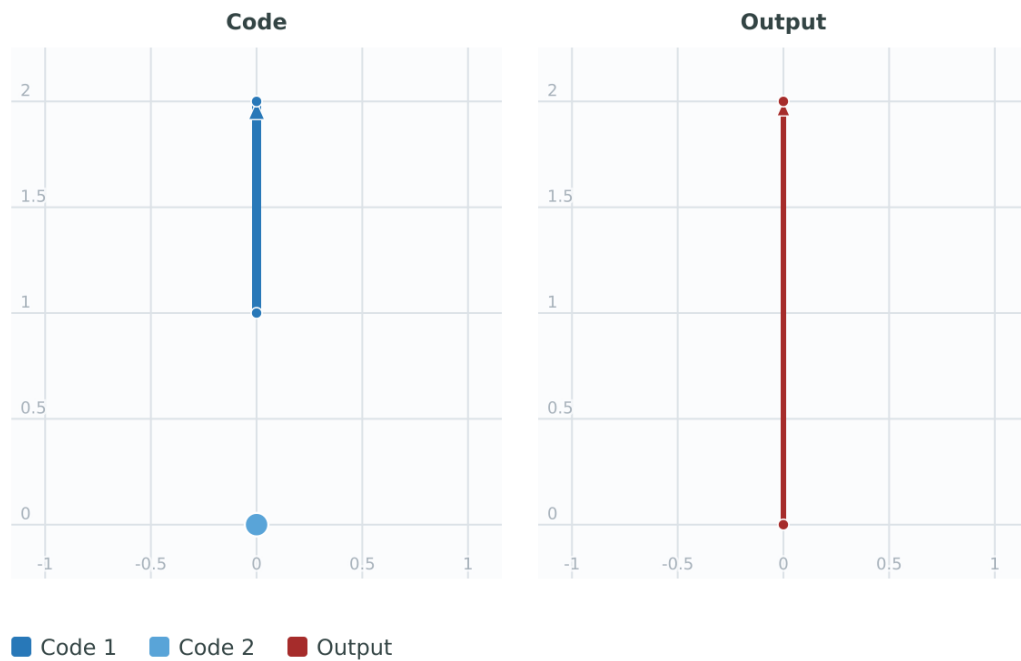
Change first point in line string from -1 3 to -1 1.

Code

```
SELECT ST_SetPoint('LINESTRING(-1 2,-1 3)', 0, 'POINT(-1 1)');
```

Output

```
LINESTRING(-1 1,-1 3)
```

Figure

Change last point in a line string (let's play with 3D linestring this time)

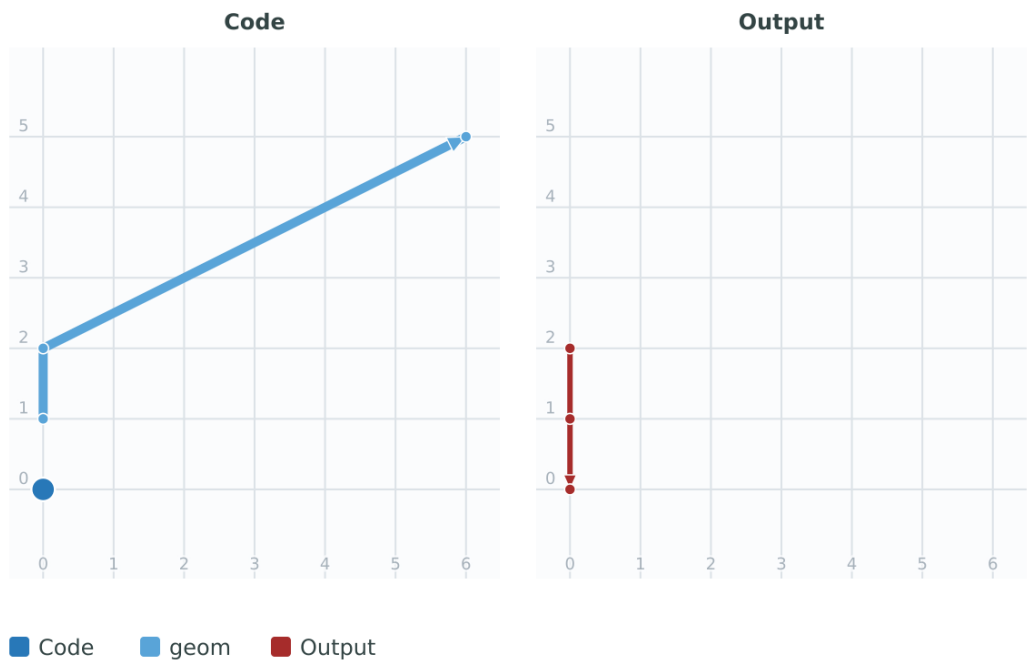
Code

```
SELECT ST_SetPoint(foo.geom, ST_NumPoints(foo.geom) - 1, 'POINT(-1 1 3)')  
FROM (SELECT 'LINESTRING(-1 2 3,-1 3 4,5 6 7)::geometry As geom) As foo;
```

Output

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

Figure



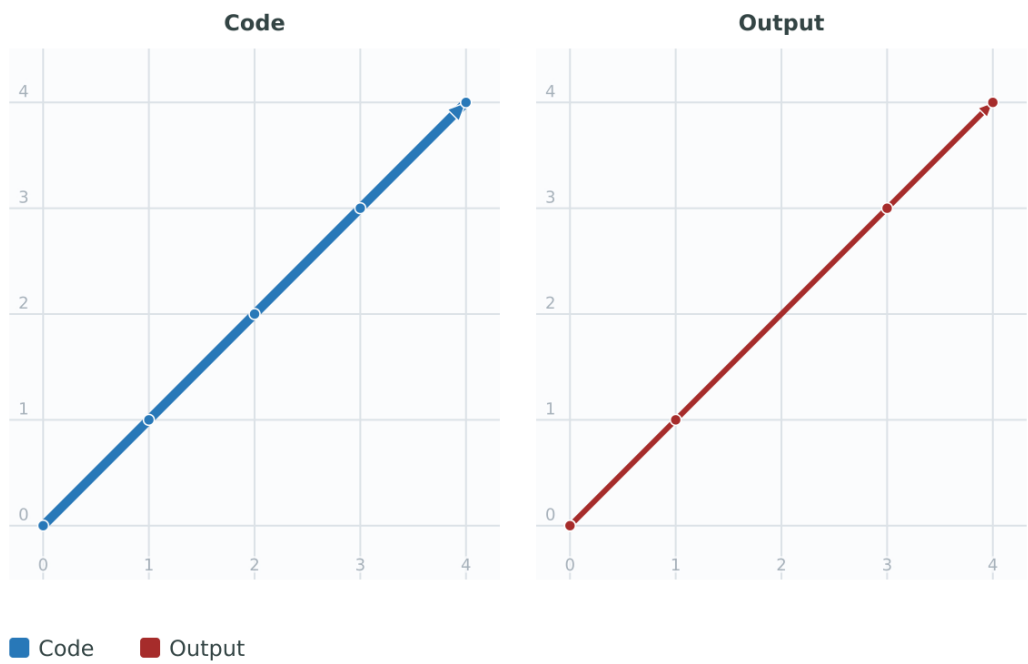
Code

```
SELECT ST_SetPoint(g, -3, p)
FROM ST_GeomFromText('LINESTRING(0 0,1 1,2 2,3 4 4)') AS g
CROSS JOIN ST_PointN(g, 1) as p;
```

Output

```
LINESTRING(0 0,1 1,0 0,3 4 4)
```

Figure



相关信息

[ST_AddPoint](#), [ST_NPoints](#), [ST_NumPoints](#), [ST_PointN](#), [ST_RemovePoint](#)

7.5.31 ST_ShiftLongitude

ST_ShiftLongitude — 在 -180-180 和 0-360 之间移动几何图形的经度坐标。

Synopsis

geometry **ST_ShiftLongitude**(geometry geom);

描述

该函数取几何图形的所有点/点，并将 -180 到 0 度范围内的经度移动到 180 到 360 度，并将 180 到 360 度范围内的经度从 -180 度移动到 0 度。该函数是周期的，因此 -180 到 180 度范围内的数据用 0 到 360 度范围来表示，0 到 360 度范围内的数据用 -180 到 180 度范围来表示。



Note

该函数具有经度/经度坐标的数据有用；例如 SRID 4326 (WGS 84 地理坐标系)。



Warning

1.3.4 之前的版本导致该功能无法用于 MULTIPOINT。1.3.4 也可与 MULTIPOINT 配合使用。

✓ 该函数支持 3d 并且不会丢失 z-index。

增强功能：2.0.0 支持多面体曲面和 TIN。

注意：该函数在 2.2.0 中由“ST_Shift_Longitude”重命名。

✓ 该函数支持多面体曲面。

✓ 此函数支持三角形和不规则三角网面 (TIN)。

示例

This example performs a forward longitude shift on a single point.

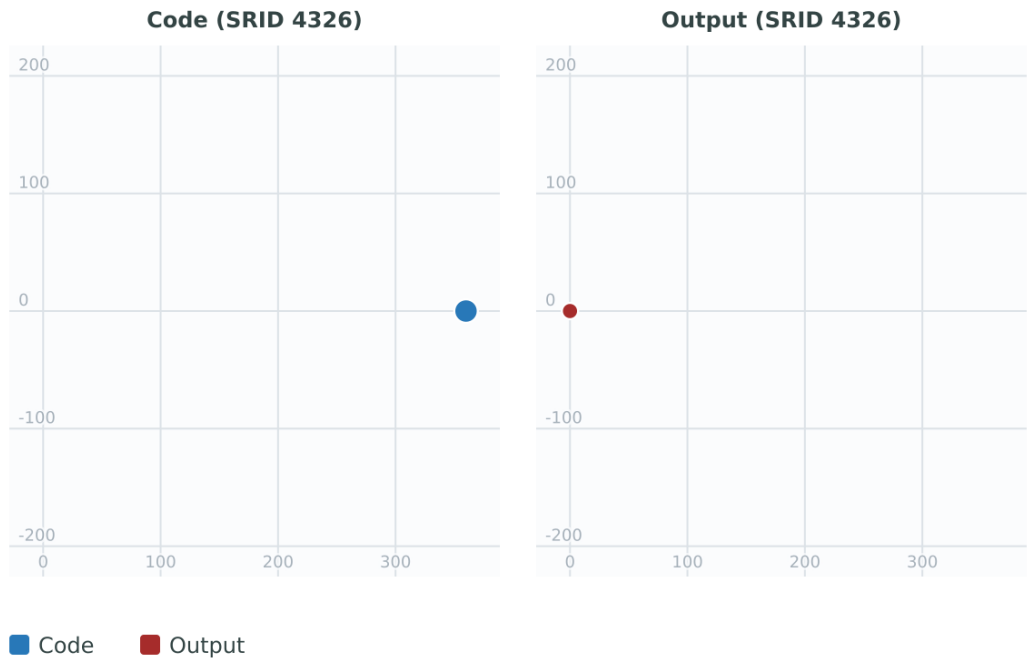
Code

```
SELECT ST_ShiftLongitude('SRID=4326;POINT(270 0)::geometry)
```

Output

```
POINT(-90 0)
```

Figure



This example performs a reverse longitude shift on a single point.

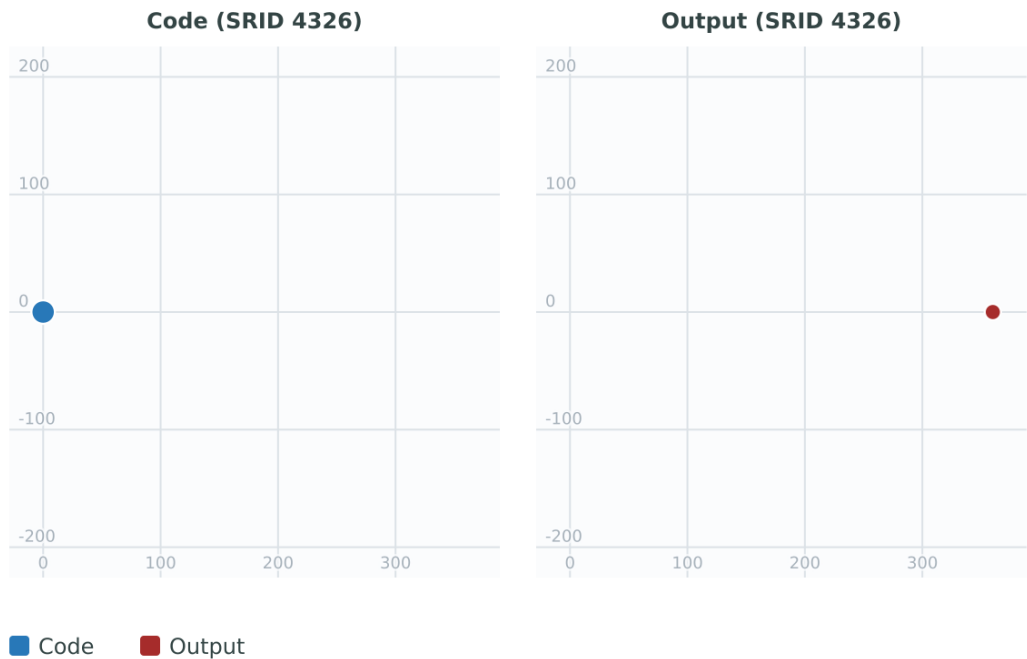
Code

```
SELECT ST_ShiftLongitude('SRID=4326;POINT(-90 0)::geometry)
```

Output

```
POINT(270 0)
```

Figure



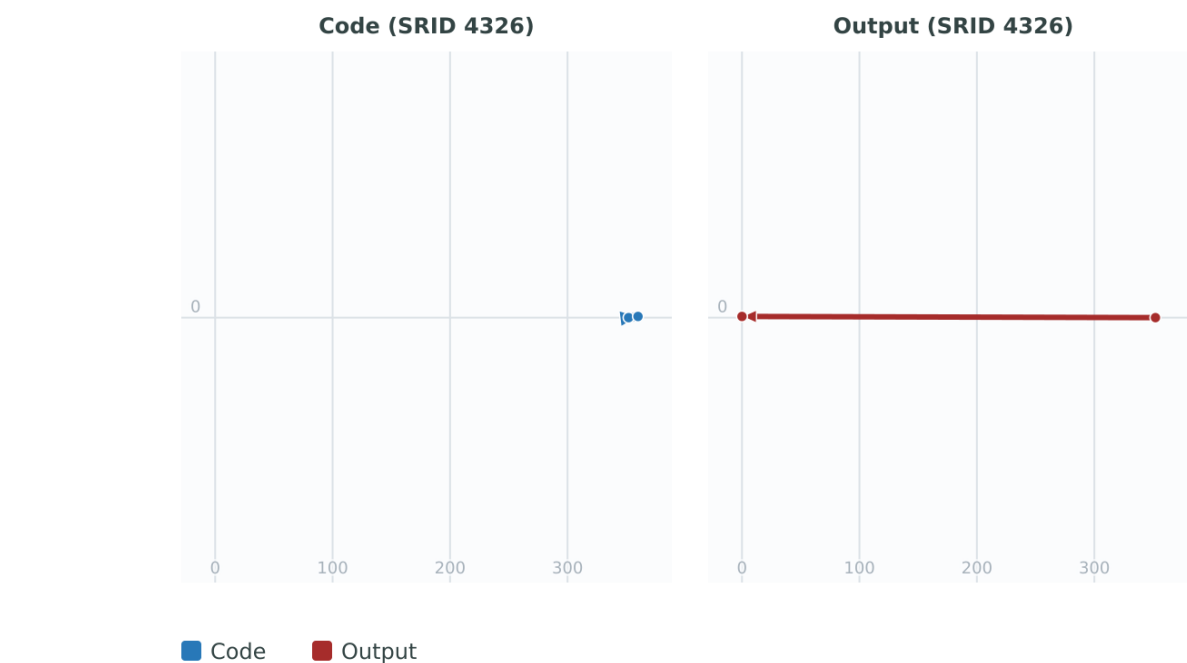
This example shows that LineStrings are shifted only where coordinates require it.

Code

```
SELECT ST_ShiftLongitude('SRID=4326;LINESTRING(174 12,182 13)::geometry)
```

Output

```
LINESTRING(174 12,-178 13)
```

Figure

相关信息

[ST_WrapX](#)

7.5.32 ST_WrapX

`ST_WrapX` — 将几何体在 X 周围。

Synopsis

```
geometry ST_WrapX(geometry geom, float8 wrap, float8 move);
```

描述

此函数分割输入的几何形状，然后按照“move”参数指定的方向移落在指定“wrap”右（对于“移”）或左（对于正“移”）的每个结果件，最后将碎片重新组合在一起。

**Note**

对于“重新居中”程度输入以使感兴趣的特征不会从一生成到一非常有用。

可用性 : 2.3.0 需要 GEOS

 函数支持 3d 并且不会丢失 z-index。

示例

Move all components whose bounding boxes fall completely to the left of $x = 0$ by +360.

Code

```
SELECT ST_WrapX(geom, 0, 360);
```

Move all components whose bounding boxes fall completely to the left of $x = -30$ by +360.

Code

```
SELECT ST_WrapX(geom, -30, 360);
```

相关信息

[ST_ShiftLongitude](#)

7.5.33 ST_SnapToGrid

ST_SnapToGrid — 将输入几何体的所有点捕捉到网格。

Synopsis

```
geometry ST_SnapToGrid(geometry geomA, float originX, float originY, float sizeX, float sizeY);
geometry ST_SnapToGrid(geometry geomA, float sizeX, float sizeY);
geometry ST_SnapToGrid(geometry geomA, float size);
geometry ST_SnapToGrid(geometry geomA, geometry pointOrigin, float sizeX, float sizeY, float sizeZ,
float sizeM);
```

描述

概要 1,2,3 : 将输入几何体的所有点捕捉到由其原点和像元大小定义的网格。除落在同一元上的点，如果出点不足以定义定型的几何形，则返回 NULL。集合中折线的几何形将从其中剥离。于降低精度很有用。

概要 4 : 引入 1.1.0 - 将输入几何体的所有点捕捉到由其原点（第二个参数必须是点）和像元大小定义的网格。将 0 指定您不想捕捉到网格的任何尺寸的大小。



Note

返回的几何形可能会失去其简单性（参看 [ST_IsSimple](#)）。



Note

在版本 1.1.0 之前，此函数始终返回 2d 几何形。从 1.1.0 开始，返回的几何形将具有与输入几何形相同的维度，但更高的维度保持不。使用有第二个几何参数的版本来定义所有网格尺寸。

可用性 : 1.0.0RC1

可用性 : 1.1.0 - 添加了 Z 和 M 的支持



函数支持 3d 并且不会丢失 z-index。

示例

Snap a geometry column to a precision grid of 10^{-3} .

Code

```
UPDATE mytable
SET geom = ST_SnapToGrid(geom, 0.001);
```

Display how the same grid size rounds a LineString.

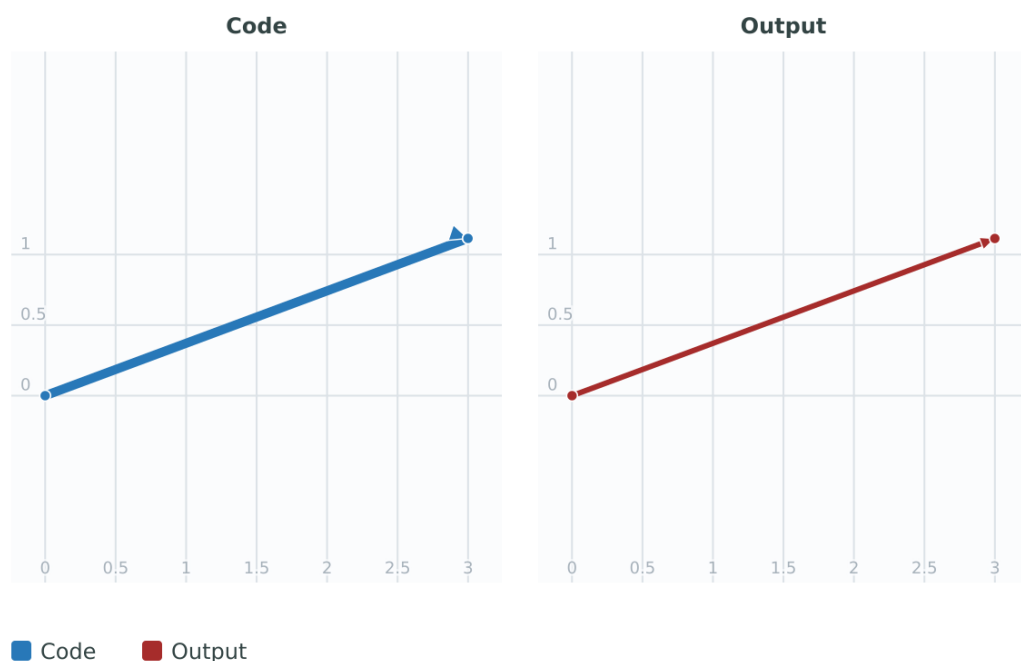
Code

```
SELECT ST_SnapToGrid('LINESTRING(1.1115678 2.123,4.111111 3.2374897,4.11112 3.23748667)'::
  geometry,
  0.001);
```

Output

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

Figure



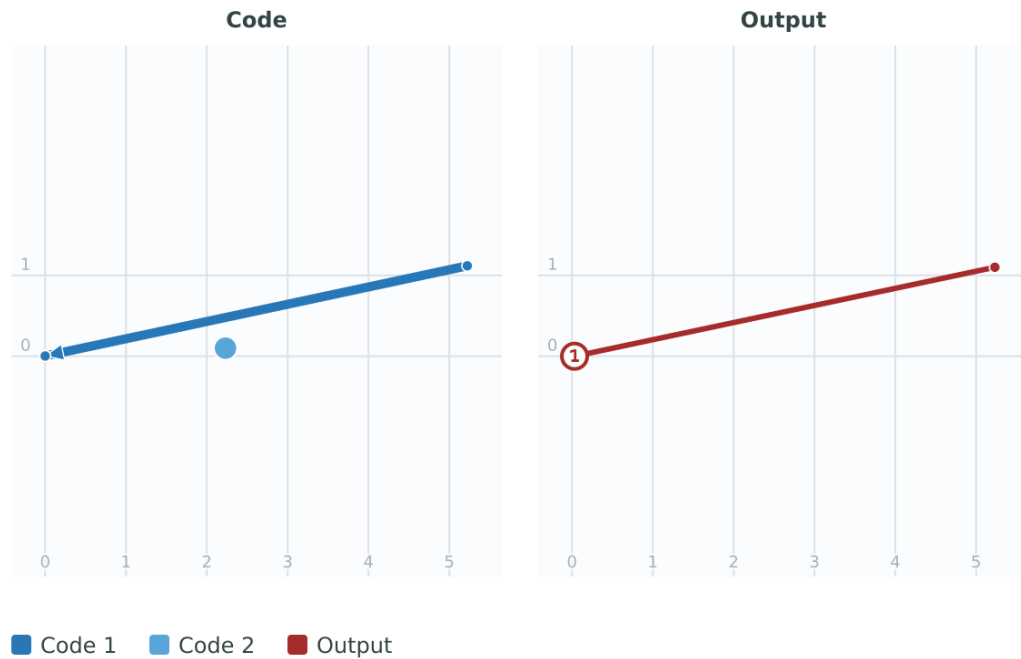
This example snaps a 4D geometry.

Code

```
SELECT ST_SnapToGrid('LINESTRING(-1.1115678 2.123 2.3456 1.11111,
  4.111111 3.2374897 3.1234 1.1111,-1.1111112 2.123 2.3456 1.111112)'::geometry,
  'POINT(1.12 2.22 3.2 4.4444)',
  0.1, 0.1, 0.1, 0.01);
```

Output

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

Figure

With a 4D geometry - the `ST_SnapToGrid(geom,size)` only touches x and y coords but keeps m and z the same.

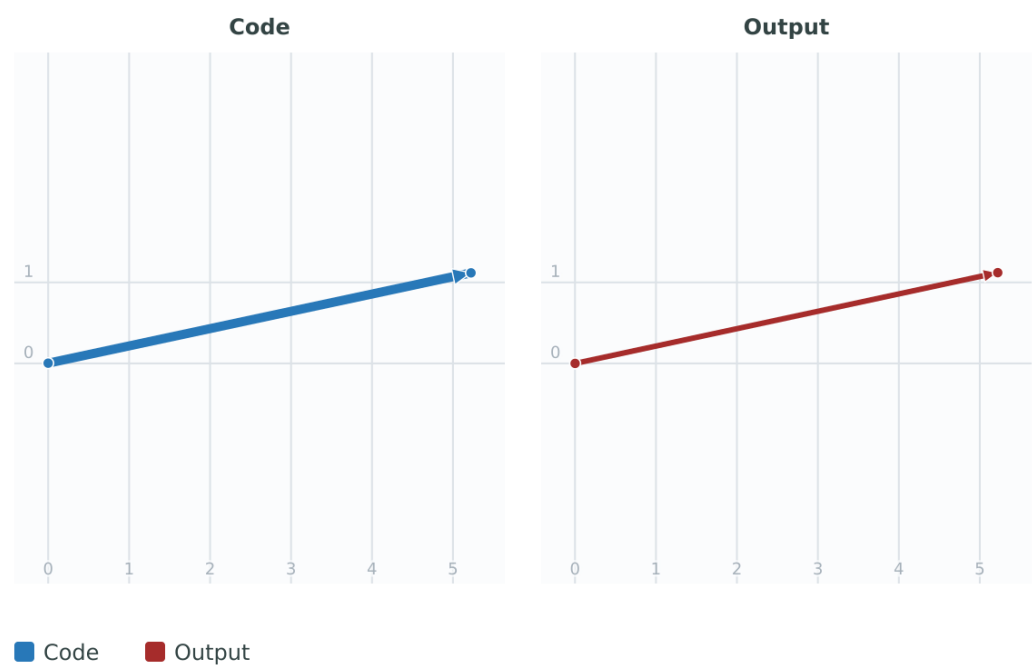
Code

```
SELECT ST_SnapToGrid('LINESTRING(-1.1115678 2.123 3 2.3456,
4.111111 3.2374897 3.1234 1.1111)>::geometry,
0.01);
```

Output

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

Figure



相关信息

[ST_Snap](#), [ST_AsEWKT](#), [ST_AsText](#), [ST_GeomFromText](#), [ST_GeomFromEWKT](#), [ST_Simplify](#)

7.5.34 ST_Snap

ST_Snap — 将输入几何体的线段和点捕捉到参考几何体的点。

Synopsis

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

描述

将一个几何体的点和线捕捉到一个几何体的点。捕捉距离公差用于控制行捕捉的位置。如果几何体是点被捕捉的输入几何体。如果没有生成捕捉，输入几何体将原样返回。

将一个几何体捕捉到一个几何体可以通过消除几何重合的（会在点和相交算期引起）来提高加操作的可靠性。

多的捕捉可能会致建无效的拓扑，因此使用式方法确定捕捉点的数量和位置，以确定何可以安全捕捉。然而，可能会致一些潜在的捕捉被忽略。



Note

返回的几何形可能已失去其性（参[ST_IsSimple](#)）或可能无效（参[ST_IsValid](#)）。

个函数是由 GEOS 模行的。

可用性: 2.0.0

示例

A MultiPolygon snapped to a LineString at 1.01 times their distance.

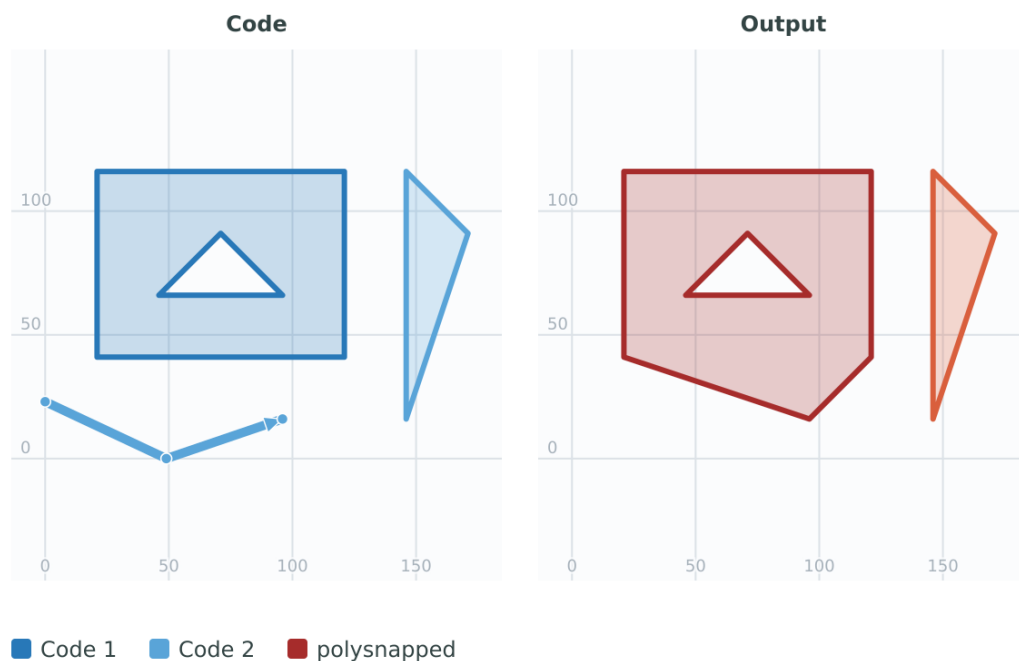
| Code

```
SELECT 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;
```

| Output

```
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)))
```

| Figure



A MultiPolygon snapped to a LineString at 1.25 times their distance.

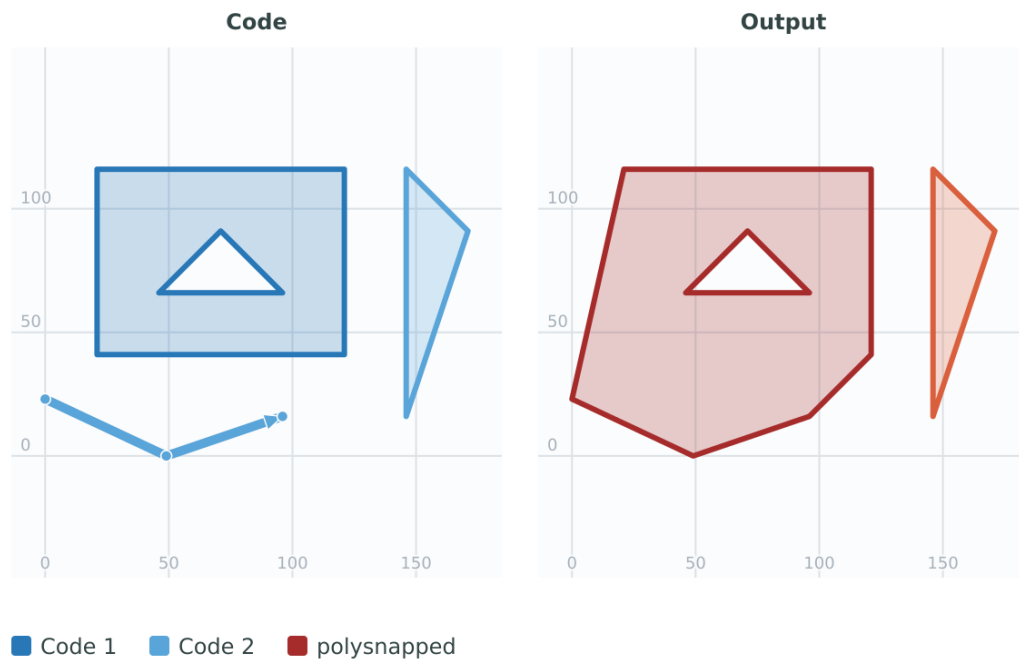
| Code

```
SELECT 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;
```

Output

```
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)))
```

Figure



The LineString snapped to the original MultiPolygon at 1.01 times their distance.

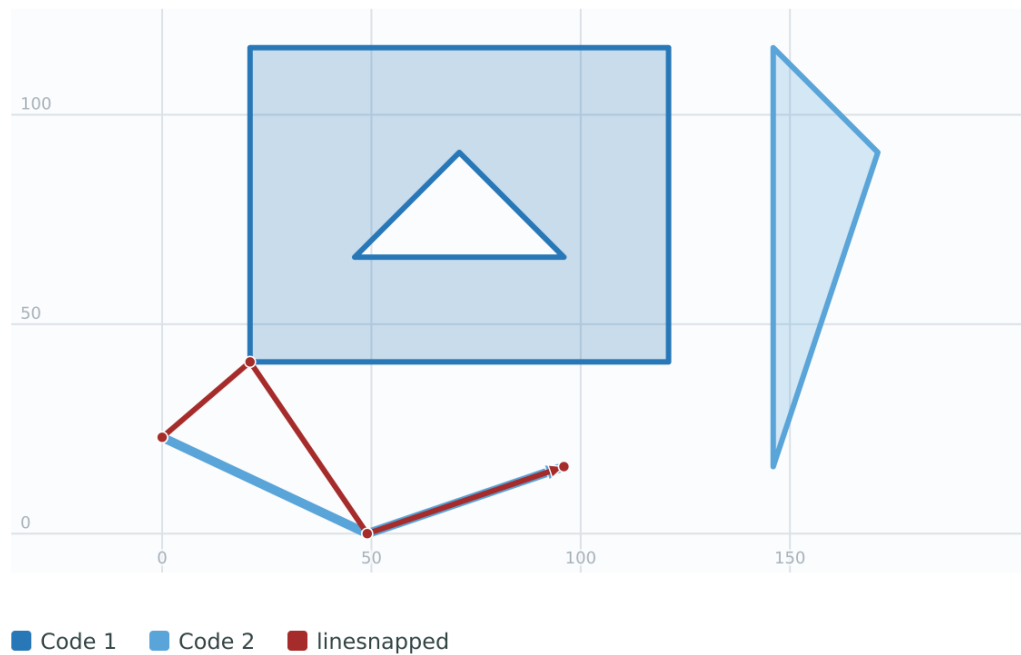
Code

```
SELECT 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;
```

Output

```
LINESTRING(5 107,26 125,54 84,101 100)
```

Figure



The LineString snapped to the original MultiPolygon at 1.25 times their distance.

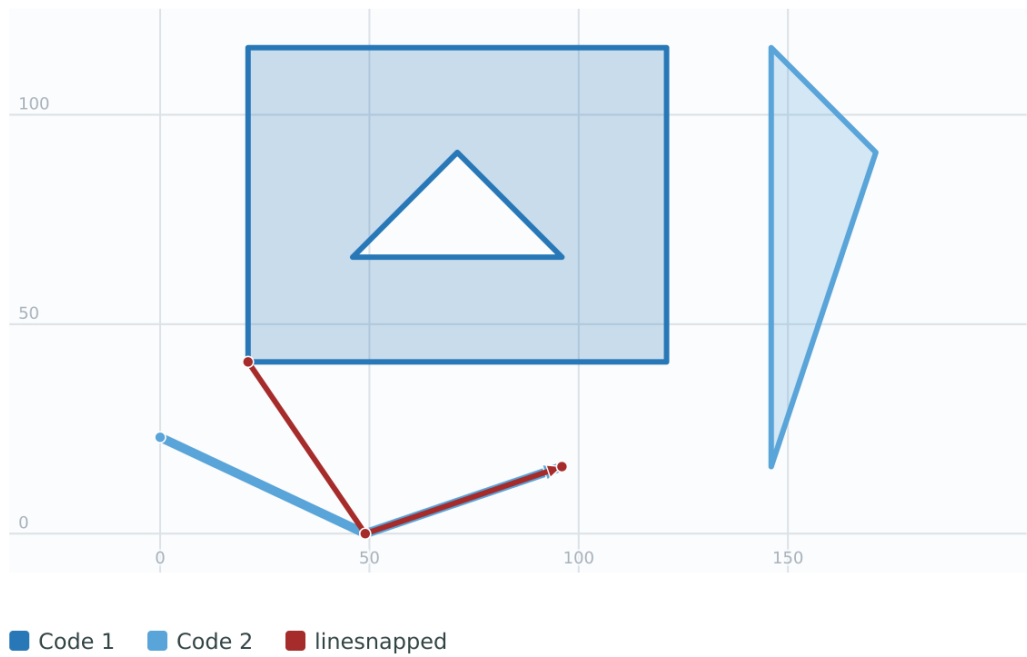
Code

```
SELECT 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;
```

Output

```
LINESTRING(26 125,54 84,101 100)
```

Figure



相关信息

[ST_SnapToGrid](#)

7.5.35 ST_SwapOrdinates

ST_SwapOrdinates — 返回更改后的几何图形，其中交换了指定的坐标。

Synopsis

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

描述

返回一个几何图形，其中指定坐标的在指定几何图形中交换。

ords 参数是一个个字符的字符串，指定要交换的坐标。有效名称：x、y、z 和 m。

可用性：2.2.0

- ✓ 此方法支持图形字符串和曲线。
- ✓ 函数支持 3d 并且不会丢失 z-index。
- ✓ 功能支持 M 坐标。
- ✓ 函数支持多面体曲面。
- ✓ 此函数支持三角形和不规则三角网面 (TIN)。

示例

This example scales the M value by 2.

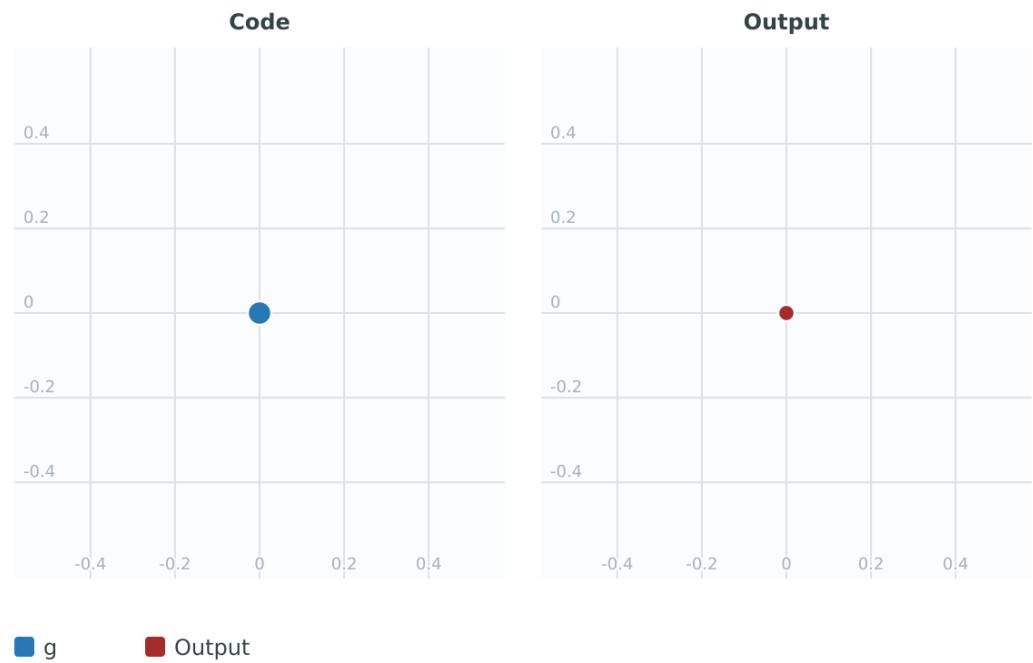
Code

```
SELECT ST_SwapOrdinates(ST_Scale(ST_SwapOrdinates(g, 'xm'),
                                2, 1),
                        'xm') FROM ( SELECT 'POINT ZM (0 0 0 2)::geometry g ) foo;
```

Output

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

Figure



相关信息

[ST_FlipCoordinates](#)

7.6 几何有效性☒☒

7.6.1 ST_IsValid

ST_IsValid — ☒☒几何☒形在 2D 中是否有效。

Synopsis

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


描述

根据 OGC 规范，ST_Geometry 在 2D 中是否格式良好且有效。对于 3 维和 4 维几何形状，有效性仍然在 2 维中执行。对于无效的几何形状，会发出 PostgreSQL NOTICE，提供其无效原因的详细信息。

对于有 flags 参数的版本，支持的规范在 [ST_IsValidDetail](#) 中。此版本不会打印解释无效性的通知。

有关几何有效性定义的更多信息，请参考第 [Section 4.4](#)



Note
SQL-MM 将 ST_IsValid(NULL) 的结果定为 0，而 PostGIS 返回 NULL。

这个函数是由 GEOS 模块执行的。

接受规范的版本从 2.0.0 开始可用。

- ✓ 此方法符合了 [SQL 1.1](#) 的 OGC 功能规范。
- ✓ 该方法符合了 SQL/MM 规范。SQL-MM 3: 5.1.9



Note
OGC-SFS 和 SQL-MM 规范均不包含 ST_IsValid 的标志参数。标志是 PostGIS 扩展。

示例

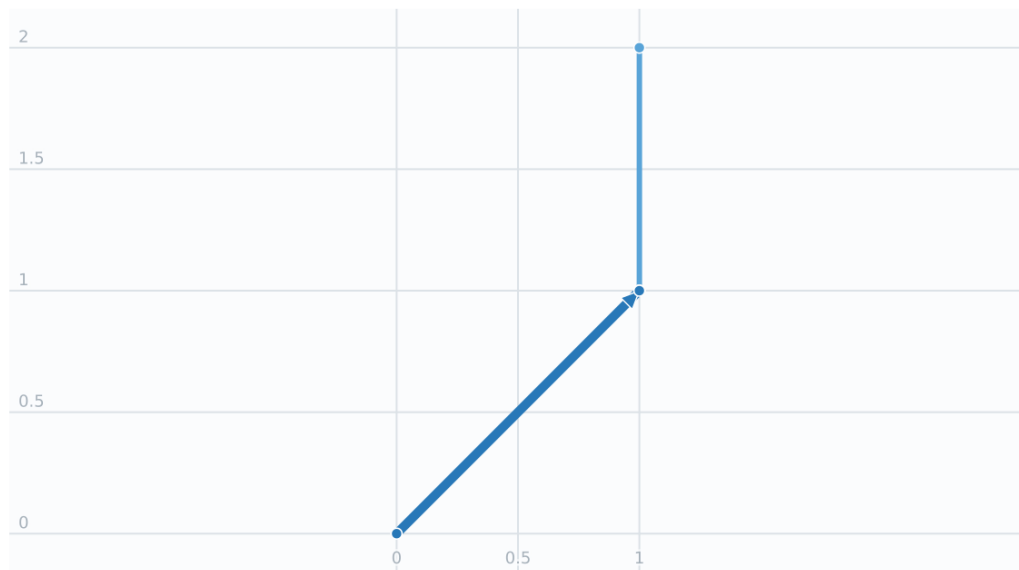
Code

```
SELECT ST_IsValid('LINESTRING(0 0,1 1)') As good_line,
ST_IsValid('POLYGON((0 0,1 1,1 2,1 1,0 0))') As bad_poly;
```

Output

```
NOTICE: Self-intersection at or near point 0 0
good_line | bad_poly
-----+-----
t         | f
```

Figure



■ Code 1 ■ Code 2

Find tables with invalid geometries in registered geometry columns.

Code

```
CREATE TEMP TABLE invalid_geometry_columns (
    table_name text,
    column_name text,
    invalid_count bigint
);

DO $$
DECLARE
    rec record;
    invalid_count bigint;
BEGIN
    FOR rec IN
        SELECT f_table_schema, f_table_name, f_geometry_column
        FROM geometry_columns
        ORDER BY f_table_schema, f_table_name, f_geometry_column
    LOOP
        EXECUTE format(
            'SELECT count(*) FROM %I.%I WHERE NOT ST_IsValid(%I, 0)',
            rec.f_table_schema,
            rec.f_table_name,
            rec.f_geometry_column
        ) INTO invalid_count;

        IF invalid_count
        > 0 THEN
            INSERT INTO invalid_geometry_columns
            VALUES (
                format('%I.%I', rec.f_table_schema, rec.f_table_name),
                rec.f_geometry_column,
                invalid_count
            );
        END IF;
    END LOOP;
```

```
END;
$$;

SELECT * FROM invalid_geometry_columns;
```

相关信息

[ST_IsSimple](#), [ST_IsValidReason](#), [ST_IsValidDetail](#),

7.6.2 ST_IsValidDetail

`ST_IsValidDetail` — 返回 `valid_detail` 行，说明几何图形是否有效或无效，说明原因和位置。

Synopsis

`valid_detail` **ST_IsValidDetail**(geometry geom, integer flags);

描述

返回一个 `valid_detail` 行，其中包含一个 `boolean`（有效），说明几何图形是否有效；一个 `varchar`（原因），说明其无效原因；以及一个 `geometry`（位置），指出其无效位置。

有助于改 [ST_IsValid](#) 和 [ST_IsValidReason](#) 的符合，以生成无效几何的警告。

可的 `flags` 参数是一个位字段。它可以具有以下：

- 0：使用常用的 OGC SFS 有效性。
- 1：考虑某些型的自接触（倒壳和倒孔）是有效的。也称“ESRI 标志”，因它是某些工具使用的有效性模型。注意，在 OGC 模型下是无效的。

个函数是由 GEOS 模行的。

可用性: 2.0.0

示例

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

Code

```
SELECT gid, reason(ST_IsValidDetail(geom)), 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;
```

Output

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

This example reports a valid LineString.

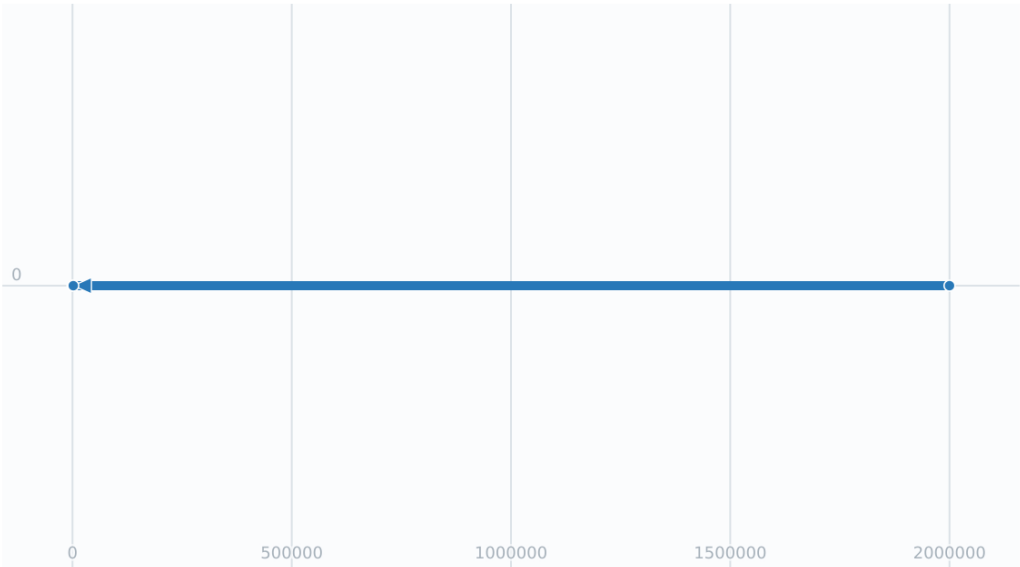
Code

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

Output

valid	reason	location
t		

Figure



Code

相关信息

ST_IsValid, ST_IsValidReason

7.6.3 ST_IsValidReason

`ST_IsValidReason` — 返回说明几何图形是否有效或无效原因的文本。

Synopsis

```
text ST_IsValidReason(geometry geomA);  
text ST_IsValidReason(geometry geomA, integer flags);
```

描述

返回文本，说明几何图形是否有效，如果无效，说明原因。

与 `ST_IsValid` 合使用可生成无效几何形状和原因的警告。

允许的 `flags` 在 `ST_IsValidDetail` 中。

这个函数是由 GEOS 模块行的。

可用性：1.4

可用性：2.0 版本正在接受。

示例

This example uses an invalid bow-tie polygon.

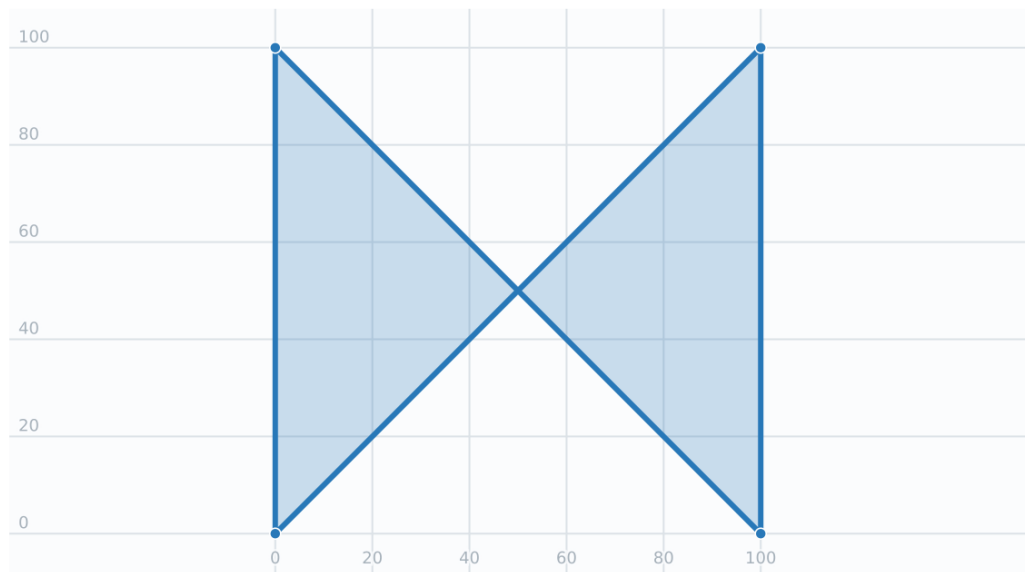
Code

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

Output

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

Figure



Code

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

Code

```
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;
```

Output

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

Simple example.

Code

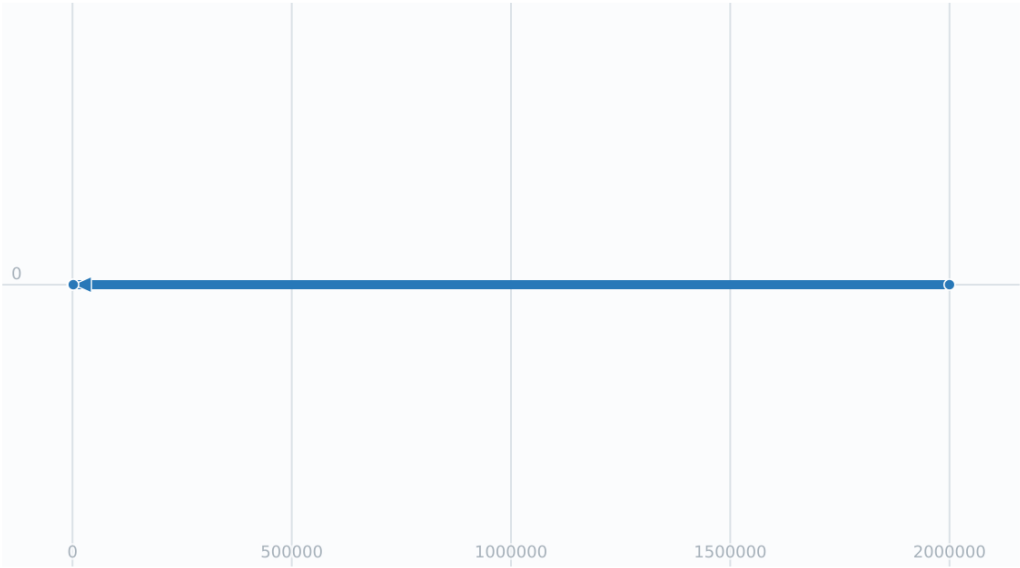
```
SELECT ST_IsValidReason('LINESTRING(220227 150406,220227 150407,22020 150410)');
```

Output

st_isvalidreason

Valid Geometry

Figure



Code

相关信息

[ST_IsValid](#), [ST_Summary](#)

7.6.4 ST_MakeValid

ST_MakeValid — 在不丢失点的情况下使无效几何体有效。

Synopsis

```
geometry ST_MakeValid(geometry input);  
geometry ST_MakeValid(geometry input, text params);
```

描述

该函数构建指定无效几何体的有效表示，而不丢失任何输入点。有效几何体原封不动地返回。
相关的输入是点、多点、线串、多线串、多边形、多多边形、几何集合及其混合。

在完全或部分尺寸折回的情况下，回出几何是相同或回低回度的几何集合，或者是回低回度几何的集合。

在自相交的情况下，回个多回形可能会回成多重几何回形。

`params` 参数可用于提供回字符串来回用于回建有效几何回形的方法。回字符串的格式回“`method=linework|structure` `keepcollapsed=true|false`”。如果未提供“`params`”参数，回将使用“`linework`”算法作回默回。

“`method`”回有回个回。

- “`linework`”是默回算法，其回理流程是：先提取所有回，再回回些回回行回点化（`nodong`），然后基于回些回生成有效的几何回出。回了保回不回失任何回点，回出回果可能会形成复回的几何集合。
- “`structure`”是一种算法，它会区分内回和外回：先回外回做并集回建新几何，再用所有内回做差。回果通常更直回，折回的回会在回出中被回弃。

“`keepcollapsed`”回回回回 `structure` 算法有效，取回回 `true` 或 `false`。

- “`keepcollapsed`”默回回 `false`，表示当几何回件被折回回低回度回（例如只有一个点的 `LineString`），回回件会被回弃。
- “`keepcollapsed`”回置回 `true` 表示即使几何回件被折回回低回度，也会被保留。

回个函数是由 GEOS 模回回的。

可用性: 2.0.0

增回: 2.0.1 速度提升

增回: 2.1.0 添加了回几何集合和多点的支持。

增回: 3.1.0 除了具有 NaN 回的坐回。

增回: 在 3.2.0 中，添加了可回的算法参数“`linework`”和“`structure`”。需要 `GEOS >= 3.10.0` 或更高版本。



回函数支持 3d 并且不会回失 `z-index`。

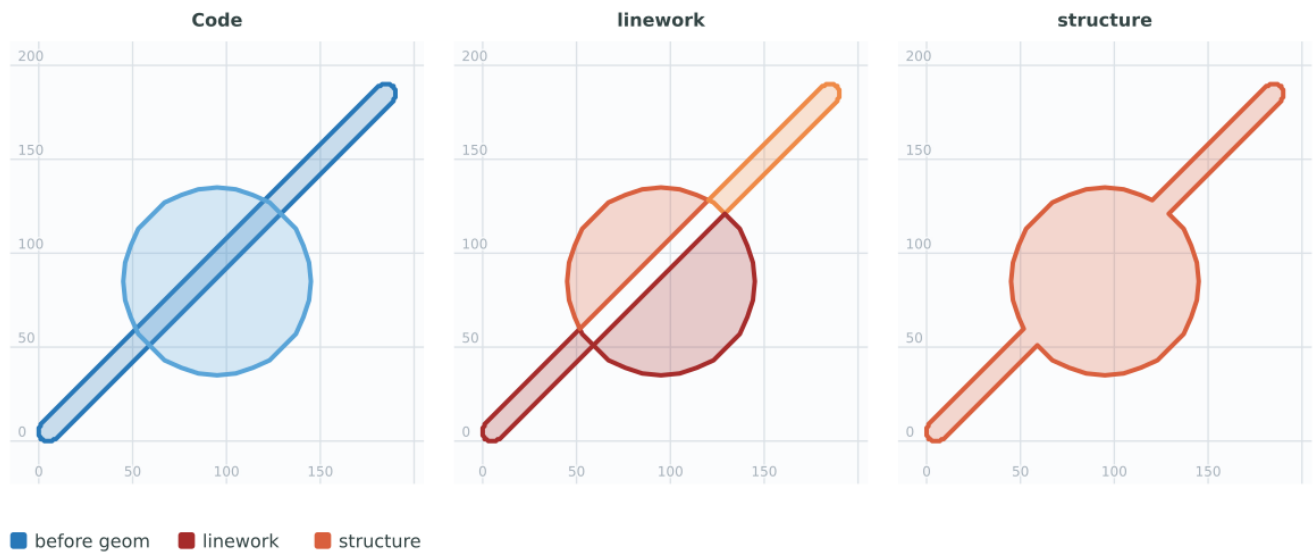
示例

Make valid a MultiPolygon of two overlapping polygons using the `linework` and `structure` methods.

Code

```
SELECT ST_MakeValid(before_geom) AS linework,
       ST_MakeValid(before_geom, 'method=structure') AS structure
FROM (SELECT 'MULTIPOLYGON(((186 194,187 194,188 195,189 195,190 195,
191 195,192 195,193 194,194 194 193,195 192,195 191,
195 190,195 189,195 188,194 187,194 186,14 6,13 6,12 5,11 5,
10 5,9 5,8 5,7 6,6 6,6 7,5 8,5 9,5 10,5 11,5 12,6 13,6 14,186 194))),
((150 90,149 80,146 71,142 62,135 55,128 48,119 44,110 41,100 40,
90 41,81 44,72 48,65 55,58 62,54 71,51 80,50 90,51 100,
54 109,58 118,65 125,72 132,81 136,90 139,100 140,110 139,
119 136,128 132,135 125,142 118,146 109,149 100,150 90)))'::geometry AS before_geom) AS f;
```


Figure

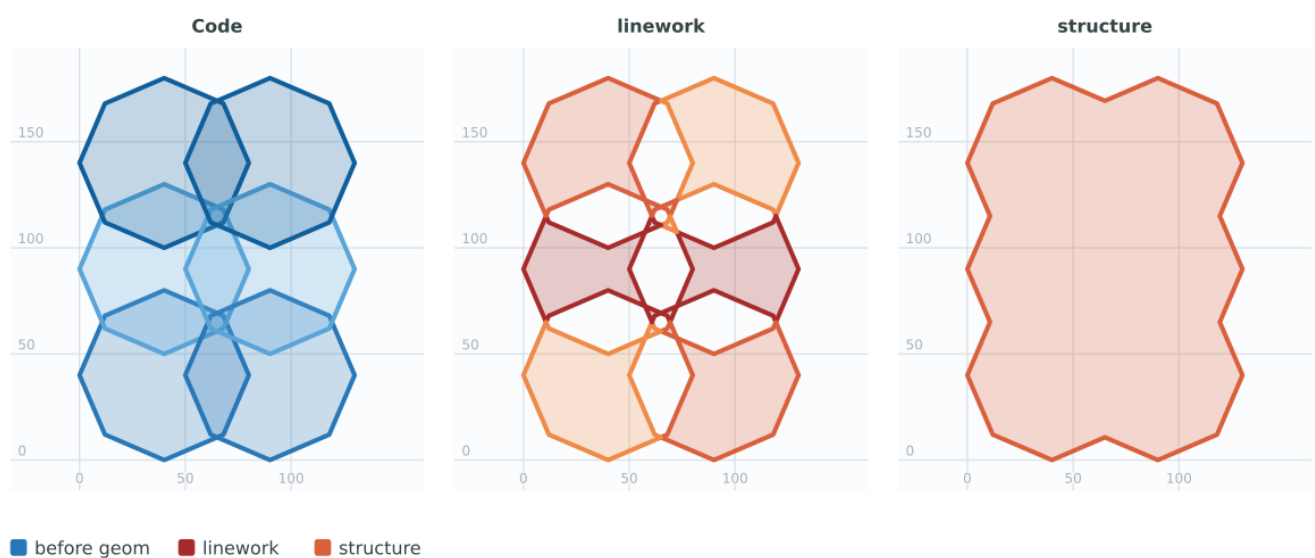


Make valid a MultiPolygon of six overlapping polygons using the linework and structure methods.

Code

```
SELECT ST_MakeValid(before_geom) AS linework,
       ST_MakeValid(before_geom, 'method=structure') AS structure
FROM (SELECT 'MULTIPOLYGON(((91 50,79 22,51 10,23 22,11 50,23 78,51 90,79 78,91 50))),
          ((91 100,79 72,51 60,23 72,11 100,23 128,51 140,79 128,91 100)),
          ((91 150,79 122,51 110,23 122,11 150,23 178,51 190,79 178,91 150)),
          ((141 50,129 22,101 10,73 22,61 50,73 78,101 90,129 78,141 50)),
          ((141 100,129 72,101 60,73 72,61 100,73 128,101 140,129 128,141 100)),
          ((141 150,129 122,101 110,73 122,61 150,73 178,101 190,129 178,141 150)))'::geometry AS before_geom) AS c;
```

Figure



Collapsed Geometry Examples

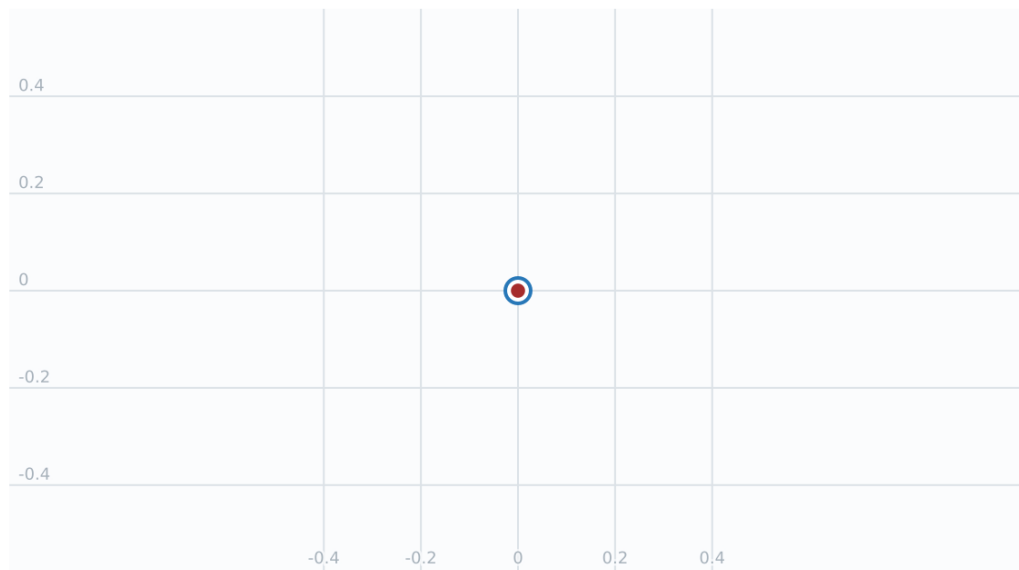
Code

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

Output

```
POINT(0 0)
```

Figure



■ Code ■ Output

Code

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

Output

```
LINESTRING EMPTY
```

相关信息

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

7.7 空参考系功能

7.7.1 ST_InverseTransformPipeline

`ST_InverseTransformPipeline` — 返回一个新的几何体，其坐标使用定义的坐标管道的逆变换到不同的空参考系。

Synopsis

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

描述

返回一个新的几何体，其坐标使用定义的坐标管道的逆变换到不同的空参考系。

有关写管道的信息，参 [ST_TransformPipeline](#)。

可用性：3.4.0

输入几何体的 SRID 将被忽略，输出几何体的 SRID 将置零，除非通可的 `to_srid` 参数提供。使用 [ST_TransformPipeline](#)，管道将向前行。使用 “`ST_InverseTransformPipeline()`”，管道以相反方向行。

使用管道的函数是 [ST_Transform](#) 的版本。在大多数情况下，“`ST_Transform`” 将正确的操作在坐标系之进行，并且是首。

示例

使用 EPSG:16031 将 WGS 84 度更改 UTM 31N

This example transforms in the inverse direction.

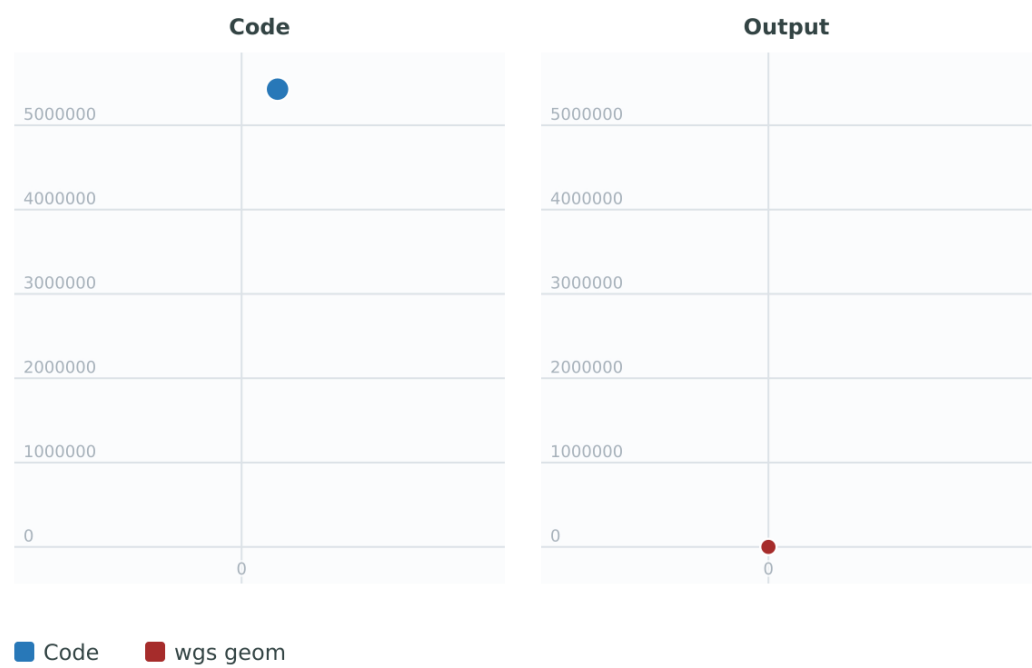
Code

```
SELECT ST_InverseTransformPipeline(
    'POINT(426857.9877165967 5427937.523342293)::geometry',
    'urn:ogc:def:coordinateOperation:EPSG::16031'
) AS wgs_geom;
```

Output

```
POINT(2 48.99999999999999)
```

| Figure



GDA2020 示例。
This example uses ST_Transform with automatic selection of a conversion pipeline.

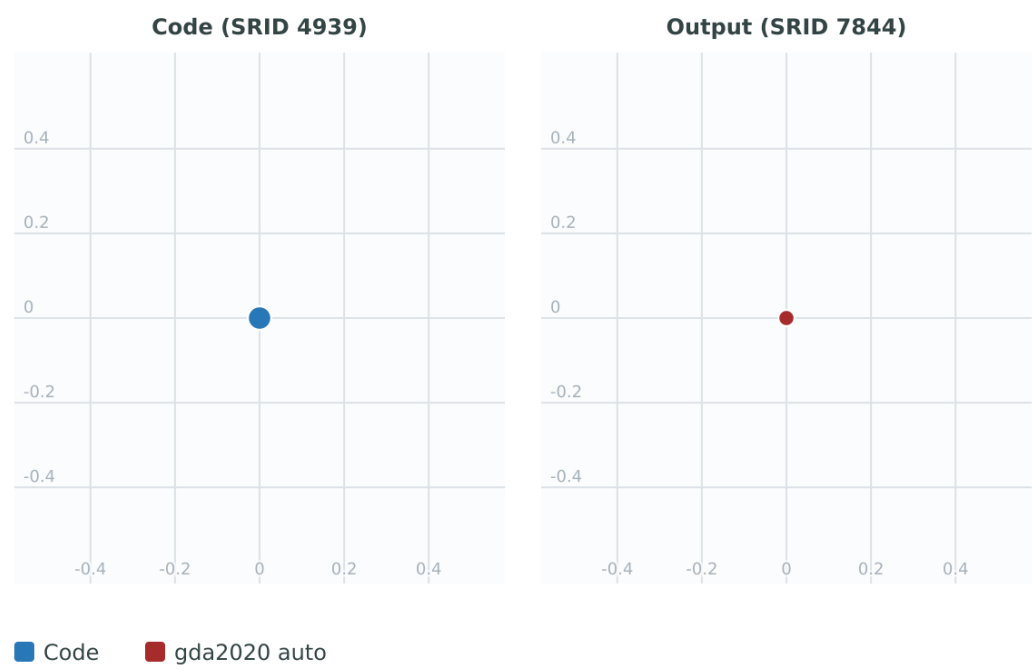
| Code

```
SELECT ST_Transform('SRID=4939;POINT(143.0 -37.0) '::geometry, 7844) AS gda2020_auto;
```

| Output

```
POINT(143.00000635638918 -36.999986706128176)
```

| Figure



相关信息

[ST_Transform](#), [ST_TransformPipeline](#)

7.7.2 ST_SetSRID

ST_SetSRID — 在几何体上设置 SRID。

Synopsis

geometry **ST_SetSRID**(geometry geom, integer srid);

描述

将几何体的 SRID 设置为特定的整数。用于生成边界框。



Note

此函数不会修改几何，它只是设置几何所需的空参考系的元数据。如果要将几何重新投影，请使用 [ST_Transform](#)。



此方法符合了 SQL 1.1 的 OGC 功能规范。



此方法支持点字符串和曲线。

示例

Mark a point as WGS 84 longitude/latitude.

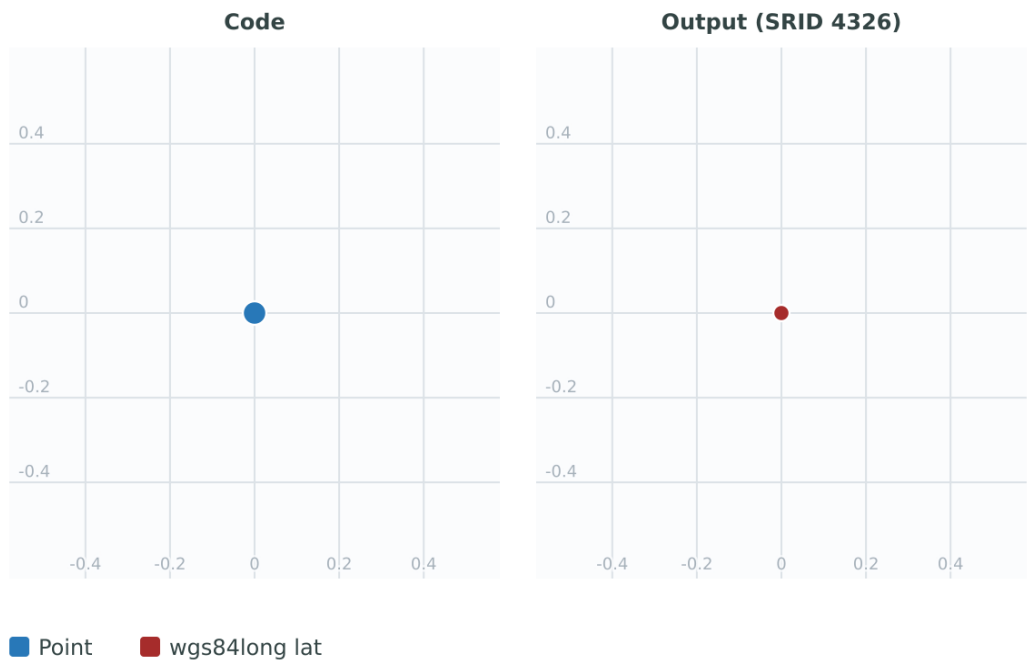
Code

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

Output

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

Figure



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

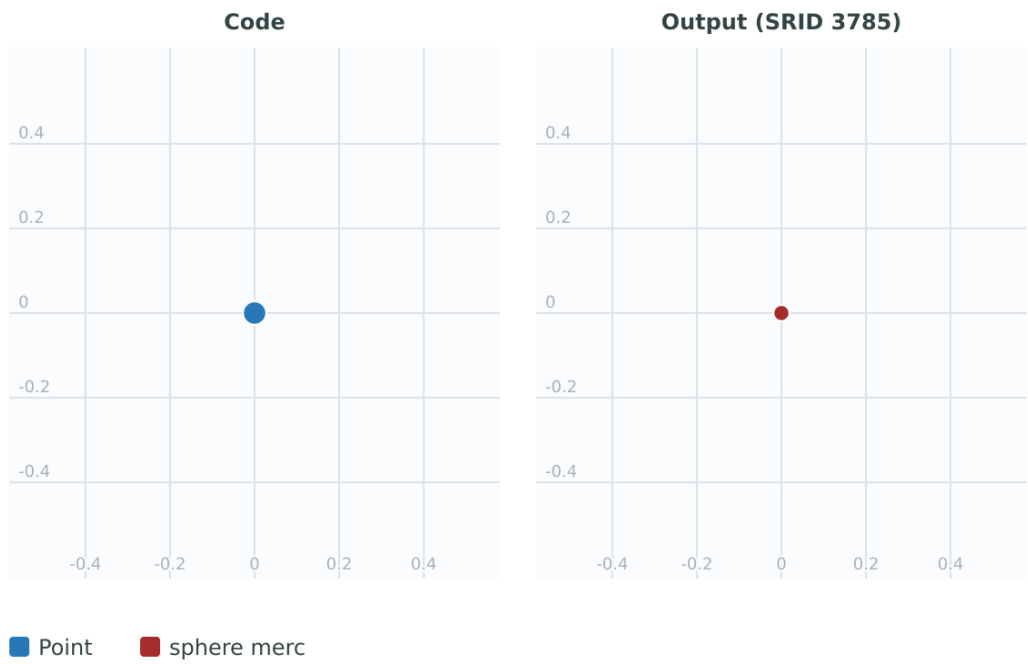
Code

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

Output

```
SRID=3785;POINT(-13732990.8753491 6178458.96425423)
```

Figure



相关信息

Section 4.5, [ST_SRID](#), [ST_Transform](#), [UpdateGeometrySRID](#)

7.7.3 ST_SRID

ST_SRID — 返回几何图形的空参考系符。

Synopsis

```
integer ST_SRID(geometry g1);
```

描述

返回 ST_Geometry spatial_ref_sys 表中定义的空参考系号。参见 Section 4.5



Note

spatial_ref_sys 表包含 PostGIS 已知的所有参考系行目，并用于从一个空参考系到一个空参考系。如果划几何，必须确保具有正确的空参考系号。

- ✓ 此方法符合了 SQL 1.1 的 OGC 功能规范。 s2.1.1.1
- ✓ 此方法符合了 SQL/MM 规范。 SQL-MM 3: 5.1.5
- ✓ 此方法支持图形字符串和曲线。

示例

Code

```
SELECT ST_SRID(ST_GeomFromText('POINT(-71.1043 42.315)', 4326));
```

Output

```
st_srid
-----
    4326
(1 row)
```

相关信息

Section 4.5, [ST_SetSRID](#), [ST_Transform](#), [ST_SRID](#), [ST_SRID](#)

7.7.4 ST_Transform

ST_Transform — 返回坐标不同空参考系的新几何形。

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);
```

描述

返回一个新的几何图形，其坐标位于不同的空间参考系。目标空间参考 `to_srid` 可以通过有效的 SRID 整数参数来指定（即，它必须存在于 `spatial_ref_sys` 表中）。或者，定义 PROJ.4 字符串的空间参考可用于 `to_proj` 和/或 `from_proj`，但某些方法并未国际化。如果目标空间参考系使用 PROJ.4 字符串而不是 SRID 表示，输出几何的 SRID 将置零。除了 `from_proj` 的函数之外，输入几何图形必须具有已定义的 SRID。

`ST_Transform` 常与 `ST_SetSRID` 混淆。`ST_Transform` 上将几何图形的坐标从一个空间参考系更改为一个空间参考系，而 `ST_SetSRID()` 只是更改几何图形的 SRID 值。

`ST_Transform` 会在指定源和目标空间参考系的情况下自动选择合适的管道。要使用特定的方法，请使用 `ST_TransformPipeline`。



Note

PostGIS 必须在 PROJ 支持下运行。使用 `PostGIS_Full_Version` 检查它是否在 PROJ 支持下。



Note

如果使用多个索引，在常用索引上有一个功能索引以充分利用索引的使用是很有用的。



Note

在 1.3.4 之前，此函数在与包含曲线的几何图形一起使用时会崩溃。此问题已在 1.3.4 及更高版本中得到修正。

增强：2.0.0 引入了多面体曲面的支持。

增强：2.3.0 引入了直接 PROJ.4 字符串的支持。



该方法符合了 SQL/MM 规范。SQL-MM 3: 5.1.6



此方法支持图形字符串和曲线。



该函数支持多面体曲面。

示例

将马里兰州平面美国英尺几何形状更改为 WGS 84 坐标度

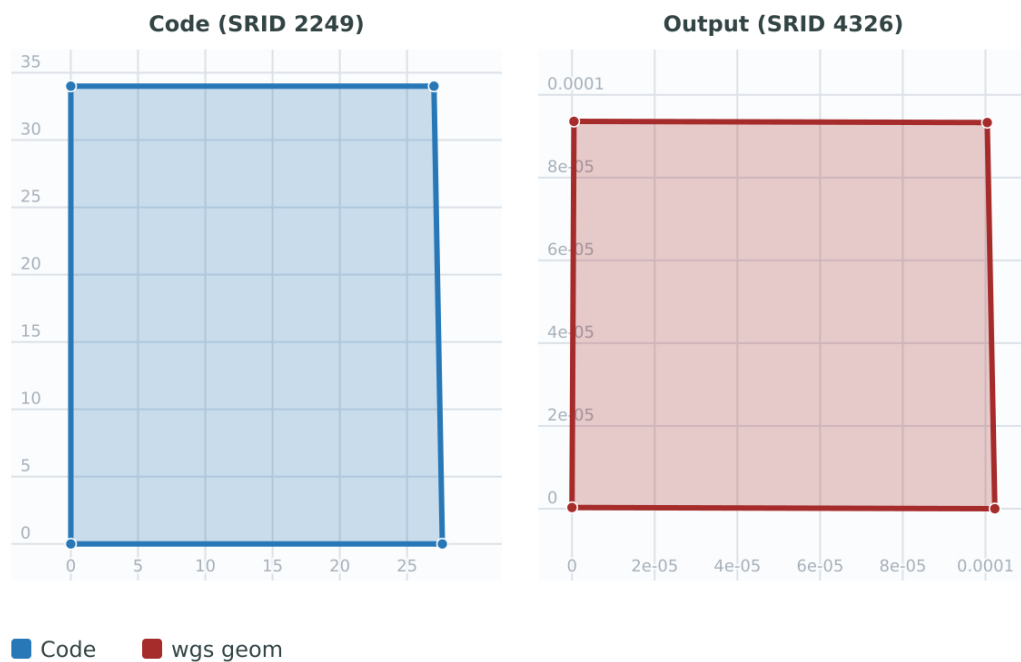
Code

```
SELECT ST_Transform(
  'SRID=2249;POLYGON((-71.1776848522251 42.3902896512903,-71.17768437663261 42.390382947801015,-71.17758443054647 ↵
    743265 2967450,743265.625 2967416,743238 2967416))'::geometry,
  4326) AS wgs_geom;
```

Output

```
SRID=4326;POLYGON((-71.1776848522251 42.3902896512903,-71.17768437663261 42.390382947801015,-71.17758443054647 ↵
  42.390382667791854,-71.17758259272304 42.390289364798825,-71.1776848522251 42.3902896512903))
```

Figure



3D Circular String example.

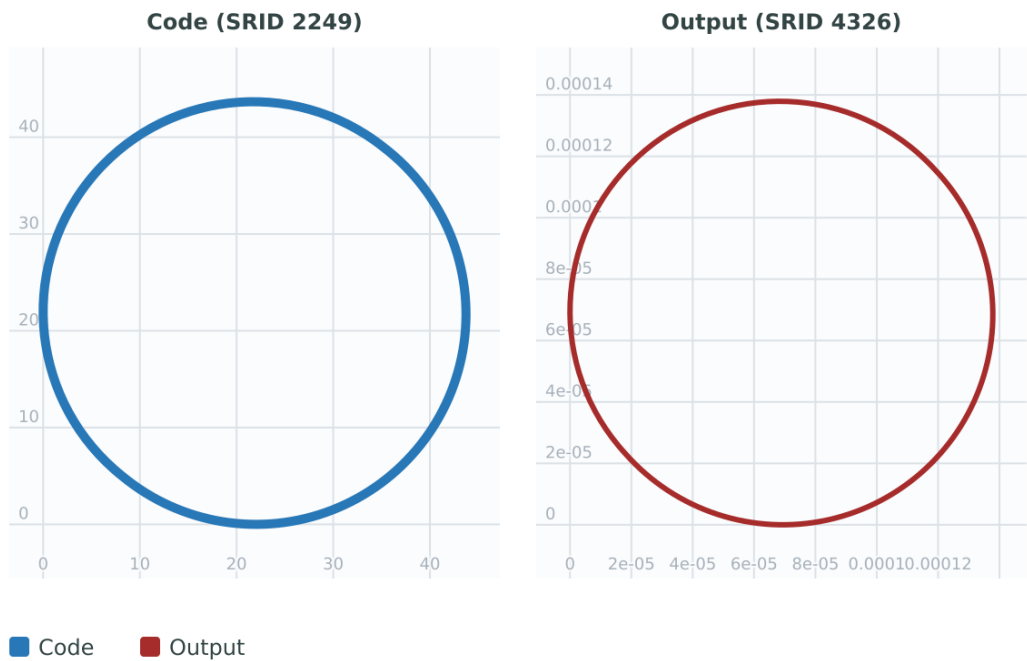
Code

```
SELECT ST_Transform('SRID=2249;CIRCULARSTRING(743238 2967416 1,743238 2967450 2,743265 ↵
  2967450 3,743265.625 2967416 3,743238 2967416 4)'::geometry,
  4326);
```

Output

```
SRID=4326;CIRCULARSTRING(-71.1776848522251 42.3902896512903 1,-71.17768437663261 42.390382947801015 2,
-71.17758443054647 42.390382667791854 3,
-71.17758259272304 42.390289364798825 3,-71.1776848522251 42.3902896512903 4)
```

Figure



创建部分功能索引的示例。由于不确定所有几何图形都将被填充的表，最好使用部分索引来省略空几何图形，既可以节省空间，又可以使索引更小、更高效。

Code

```
CREATE INDEX idx_geom_26986_parcel
ON parcels
USING gist
(ST_Transform(geom, 26986))
WHERE geom IS NOT NULL;
```

使用 PROJ.4 文本自定义空参考行号的示例。

Find intersection of two polygons near the North pole, using a custom Gnomonic projection. See <http://boundlethe-peel/>.

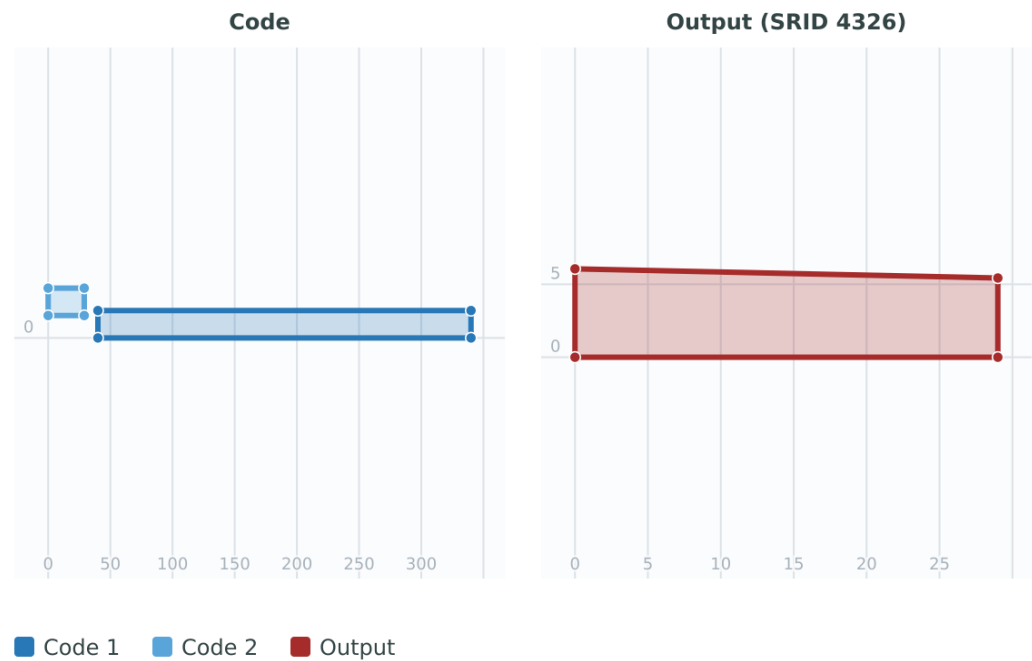
Code

```
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_Transform(ST_Intersection(ST_Transform(p1, gnom), ST_Transform(p2, gnom)),
  gnom, 4326)
FROM data;
```

Output

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

Figure



配置行

有，涉及网格移位的坐可能会失，例如，如果 PROJ.4 尚未使用网格移位文件建，或者坐不在定网格移位的范内。默情况下，如果网格偏移文件不存在，PostGIS 将抛出，但可以通 PROJ.4 文本的不同 to_proj 或更改 Spatial_ref_sys 表中的 proj4text，在每个 SRID 的基上配置此行。

例如，proj4text 参数 datum=NAD87 是以下 nadgrids 参数的写形式：

Code

```
+nadgrids=@conus, @alaska, @ntv2_0.gsb, @ntv1_can.dat
```

@ 前意味着如果文件不存在，不会告，但如果到列表末尾而没有合适的文件（即找到并重），会出。

相反，如果您想确保至少存在准文件，但如果描所有文件而没有命中，用空，您可以使用：

Code

```
+nadgrids=@conus, @alaska, @ntv2_0.gsb, @ntv1_can.dat, null
```

空网格平移文件是覆盖整个世界并且不用平移的有效网格平移文件。因此，于一个完整的示例，如果您想更改 PostGIS，以便到不在正确范内的 SRID 4267 不会引，您可以使用以下命令：

Code

```
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;
```

相关信息

Section 4.5, ST_SetSRID, ST_SRID, UpdateGeometrySRID, ST_TransformPipeline

7.7.5 ST_TransformPipeline

ST_TransformPipeline — 返回一个新的几何图形，其坐标使用定义的坐标管道与不同的空参考系。

Synopsis

```
geometry ST_TransformPipeline(geometry g1, text pipeline, integer to_srid);
```

描述

返回一个新的几何图形，其坐标使用定义的坐标管道与不同的空参考系。

管道使用以下任意字符串格式定义：

- `urn:ogc:def:coordinateOperation:AUTHORITY::CODE`。注意，CODE 的 EPSG:CODE 字符串不能唯一标识坐标操作：相同的 EPSG 代码可用于 CRS 定义。
- PROJ 管道字符串的形式：`proj=pipeline ....` 将不会用自规范化，并且如有必要，用户将需要添加外的管道步骤，或删除 `axiswap` 步骤。
- 形式的串操作：`urn:ogc:def:coordinateOperation,coordinateOperation:EPSG::3895,coordinateOp`

可用性：3.4.0

输入几何体的 SRID 将被忽略，输出几何体的 SRID 将置零，除非通过可选项的 `to_srid` 参数提供。当使用“ST_TransformPipeline()”时，管道将向前行。使用 **ST_InverseTransformPipeline** 管道以相反方向行。

使用管道的版本是 **ST_Transform** 的版本。在大多数情况下，“ST_Transform”将正确的操作在坐标系之间行，并且是首选。

示例

使用 EPSG:16031 将 WGS 84 度更改为 UTM 31N

This example transforms in the forward direction.

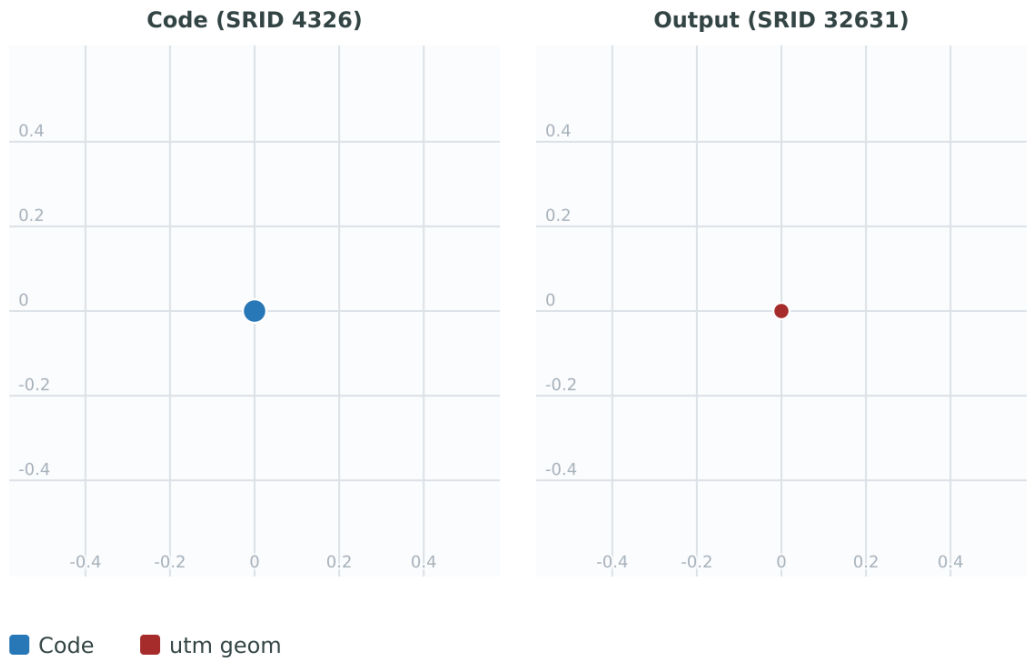
Code

```
SELECT
  ST_SetSRID(ST_TransformPipeline(
    'SRID=4326;POINT(2 49)::geometry',
    'urn:ogc:def:coordinateOperation:EPSG::16031',
    32631) AS utm_geom;
```

Output

```
SRID=32631;POINT(426857.9877165967 5427937.523342293)
```

Figure



This example transforms in the inverse direction.

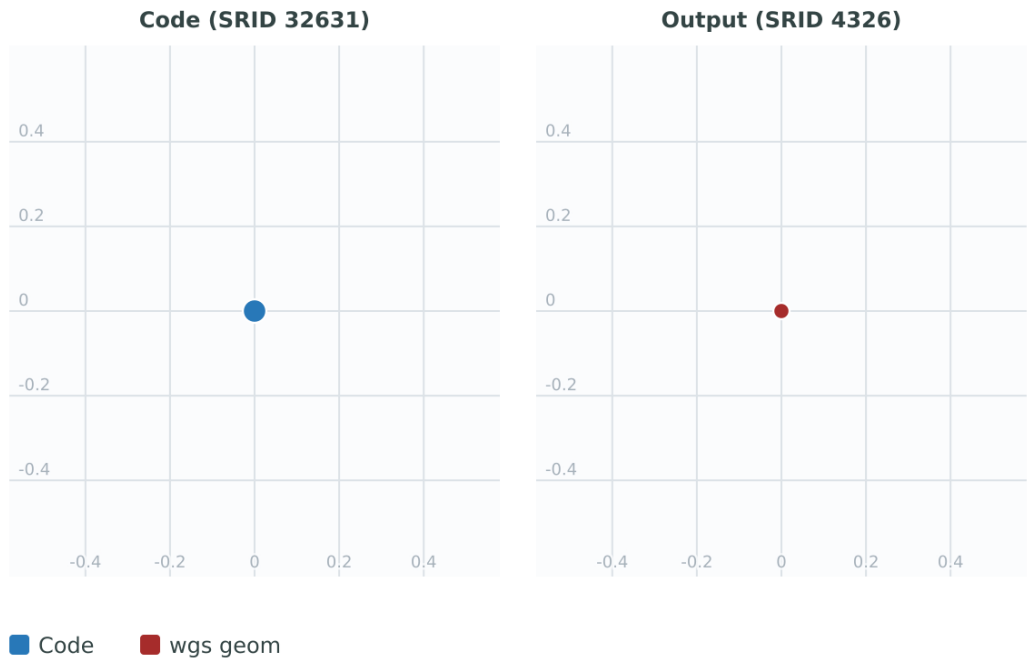
Code

```
SELECT
  ST_SetSRID(ST_InverseTransformPipeline(
    'SRID=32631;POINT(426857.9877165967 5427937.523342293)::geometry,
    'urn:ogc:def:coordinateOperation:EPSG::16031'
  ), 4326) AS wgs_geom;
```

Output

```
SRID=4326;POINT(2 48.999999999999999)
```

Figure



GDA2020 示例。
This example uses ST_Transform with automatic selection of a conversion pipeline.

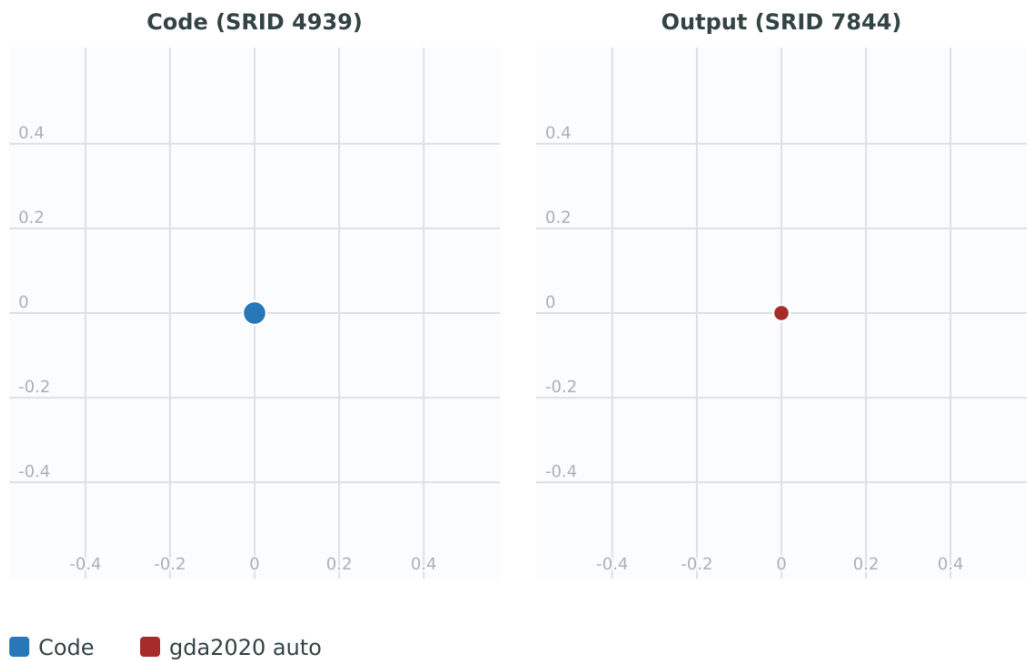
Code

```
SELECT ST_Transform('SRID=4939;POINT(143.0 -37.0) '::geometry, 7844) AS gda2020_auto;
```

Output

```
POINT(143.00000635638918 -36.999986706128176)
```

Figure



Using a defined conversion (EPSG:8447)

Code

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

Output

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

This example uses a PROJ pipeline definition matching EPSG:8447, as returned from projinfo. Any axisswap steps must be removed.

Code

```
SELECT
  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;
```

Output

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

相关信息

[ST_Transform](#), [ST_InverseTransformPipeline](#)

7.7.6 postgis_srs_codes

postgis_srs_codes — 返回与指定相关的 SRS 代码列表。

Synopsis

```
setof text postgis_srs_codes(text auth_name);
```

描述

返回指定 auth_name 的所有 auth_srid 的集合。

可用性：3.4.0

Proj version 6+

示例

获取与 EPSG 权威相关的前 10 个代码的列表。

Code

```
SELECT * FROM postgis_srs_codes('EPSG') ORDER BY 1 LIMIT 10;
```

Output

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

相关信息

[postgis_srs](#), [postgis_srs_all](#), [postgis_srs_search](#)

7.7.7 postgis_srs

postgis_srs — 返回所请求的界限和 srid 的元数据。

Synopsis

setof record **postgis_srs**(text auth_name, text auth_srid);

描述

返回给定 auth_name 所请求的 auth_srid 的元数据。结果将包含 auth_name、auth_srid、sname、srtext、proj4text 以及使用区域的角点 point_sw 和 point_ne。

可用性：3.4.0

Proj version 6+

示例

获取 EPSG 的元数据：3005。

Code

```
SELECT auth_name, auth_srid, sname,point_sw AS point_sw,point_ne AS point_ne
FROM postgis_srs('EPSG', '3005');
```

Output

```
-[ RECORD 1 ]-----
auth_name | EPSG
auth_srid | 3005
sname     | NAD83 / BC Albers
point_sw  | POINT(-139.04 48.25)
point_ne  | POINT(-114.08 60.01)
```


相关信息

[postgis_srs_codes](#), [postgis_srs_all](#), [postgis_srs_search](#)

7.7.8 postgis_srs_all

postgis_srs_all — 返回底层的 Proj 数据库中每个空参考系的元数据。

Synopsis

setof record **postgis_srs_all**(void);

描述

返回底层的 Proj 数据库中所有元数据的集合。这些将包含 auth_name、auth_srid、sname、srtext、proj4text 以及使用区域的角点 point_sw 和 point_ne。

可用性：3.4.0

Proj version 6+

示例

从 Proj 数据库中取前 10 条元数据。

Code

```
SELECT auth_name, auth_srid, sname FROM postgis_srs_all() ORDER BY auth_name, auth_srid
LIMIT 10;
```

Output

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

相关信息

[postgis_srs_codes](#), [postgis_srs](#), [postgis_srs_search](#)

7.7.9 postgis_srs_search

postgis_srs_search — 返回具有完全包含边界参数的使用区域的投影坐标系的元数据。

Synopsis

setof record **postgis_srs_search**(geometry bounds, text auth_name=EPSG);

描述

返回投影坐标系的一元数据行，这些坐标系的使用区域完全包含边界参数。每行将包含 auth_name、auth_srid、srsname、srtext、proj4text 以及使用区域的角点 point_sw 和 point_ne。
搜索行找投影坐标系，旨在用行探索适用于其数据范围的可能坐标系。

可用性：3.4.0

Proj version 6+

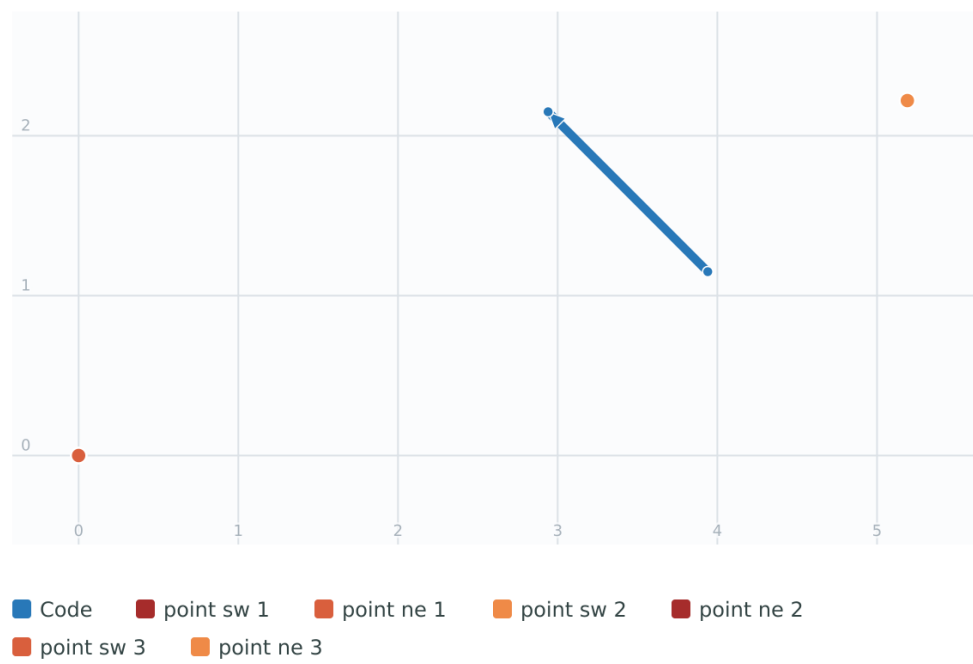
示例

搜索路易斯安那州的投影坐标系。

Code

```
SELECT auth_name, auth_srid, srsname,point_sw AS point_sw,point_ne AS point_ne
FROM postgis_srs_search('SRID=4326;LINESTRING(-90 30,-91 31)')
WHERE auth_srid IN ('2801', '3452', '3457')
ORDER BY auth_name, auth_srid::integer
```

Figure



描述表格以行得最大范围并找到可能适合的投影坐标系。

Code

```
WITH ext AS (
  SELECT ST_Extent(geom) AS geom, Max(ST_SRID(geom)) AS srid
  FROM foo
)
```

```
SELECT auth_name, auth_srid, sname,
       point_sw AS point_sw,
       point_ne AS point_ne
FROM ext
CROSS JOIN postgis_srs_search(ST_SetSRID(ext.geom, ext.srid))
LIMIT 3;
```

相关信息

[postgis_srs_codes](#), [postgis_srs_all](#), [postgis_srs](#)

7.8 几何输入

7.8.1 已知文本 (WKT)

7.8.1.1 ST_BdPolyFromText

`ST_BdPolyFromText` — 给定任意组合串集合作为 `MultiLineString` 熟知的文本表示形式 (WKT)，构造一个多边形。

Synopsis

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

描述

从组合串的任意集合构造多边形，有些串具有多串的已知文本表示形式。



Note

如果 WKT 不是 MULTILINESTRING，会引发错误。如果给出的是 MULTIPOLYGON，抛出错误；在这种情况下使用 `ST_BdMPolyFromText`，或者参见 `ST_BuildArea()` 了解 postgis 特定的方法。



此方法符合了 SQL 1.1 的 OGC 功能规范。 s3.2.6.2

该函数是由 GEOS 模块实现的。

可用性：1.1.0

相关信息

[ST_BuildArea](#), [ST_BdMPolyFromText](#)

7.8.1.2 ST_BdMPolyFromText

`ST_BdMPolyFromText` — 从任何封闭串集合构造多面，并具有多串的已知文本表示形式。

Synopsis

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

描述

给定合串、多边形、多字符串的任意集合作已知文本表示，构造一个多边形。



Note

如果 WKT 不是 MULTILINESTRING，会引发错误。即使结果集上由一个 POLYGON 组成，也制造出 MULTIPOLYGON；如果您确定操作将生成一个 POLYGON，请使用 **ST_BdPolyFromText**，或者参考 **ST_BuildArea()** 了解 postgis 特定的方法。



此方法符合了 SQL 1.1 的 OGC 功能规范。 s3.2.6.2

该函数是由 GEOS 模块执行的。

可用性：1.1.0

相关信息

ST_BuildArea, **ST_BdPolyFromText**

7.8.1.3 ST_GeogFromText

ST_GeogFromText — 从已知的文本表示或扩展 (WKT) 返回指定的地理。

Synopsis

geography **ST_GeogFromText**(text EWKT);

描述

从已知的文本或扩展的已知的表示中返回地理对象。如果未指定，假定 SRID 4326。是 ST_GeographyFromText 的别名。点始以经度形式表示。

示例

Add a WGS 84 geography point column to a table.

Code

```
ALTER TABLE sometable ADD COLUMN geog geography(POINT, 4326);
```

Populate the geography column from longitude and latitude columns.

Code

```
UPDATE sometable
SET geog = ST_GeogFromText('SRID=4326;POINT(' || lon || ' ' || lat || ')');
```

Specify a geography point using EPSG:4267, NAD27.

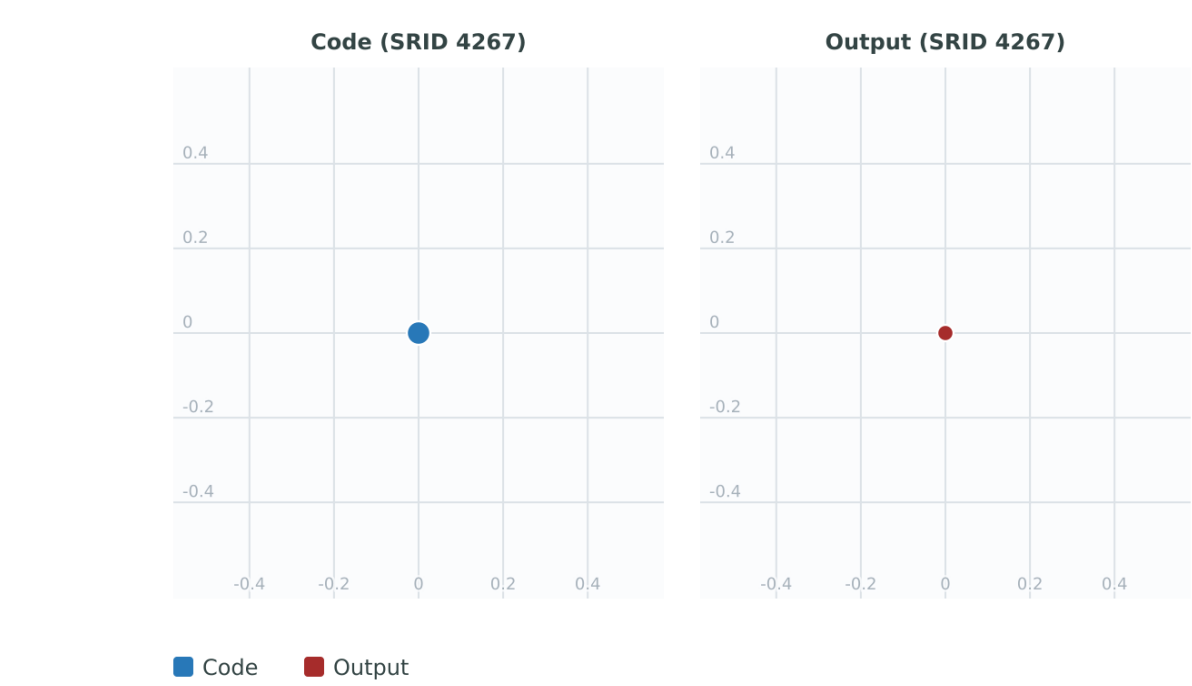
Code

```
SELECT ST_GeogFromText('SRID=4267;POINT(-77.0092 38.889588)');
```

Output

```
SRID=4267;POINT(-77.0092 38.889588)
```

Figure



相关信息

[ST_AsText](#), [ST_GeographyFromText](#)

7.8.1.4 ST_GeographyFromText

ST_GeographyFromText — 从已知的文本表示或展 (WKT) 返回指定的地理。

Synopsis

geography **ST_GeographyFromText**(text EWKT);

描述

从已知的文本表示形式返回地理象。如果未指定，假定 SRID 4326。

相关信息

[ST_GeogFromText](#), [ST_AsText](#)

7.8.1.5 ST_GeomCollFromText

ST_GeomCollFromText — 使用指定的 SRID 从集合 WKT 中构建集合几何体。如果未给出 SRID，默认为 0。

Synopsis

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

描述

使用指定的 SRID 从已知的文本 (WKT) 表示形式构建几何集合。如果未给出 SRID，默认为 0。

OGC 规范 3.2.6.2 - 可指定 SRID 用于规范

如果 WKT 不是几何集合，返回 NULL



Note

如果您确定所有 WKT 几何形状都是集合，请不要使用此函数。它比 ST_GeomFromText 慢，因为它添加了额外的步骤。



此方法符合了 SQL 1.1 的 OGC 功能规范。 s3.2.6.2



此方法符合了 SQL/MM 规范。

示例

Code

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

相关信息

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

7.8.1.6 ST_GeomFromEWKT

ST_GeomFromEWKT — 从扩展已知的文本表示 (EWKT) 返回指定的 ST_Geometry 值。

Synopsis

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

描述

根据 OGC 展示已知的文本 (EWKT) 表示构造 PostGIS ST_Geometry 对象。



Note
EWKT 格式不是 OGC 标准，而是包含空参考系 (SRID) 符合的 PostGIS 特定格式

增强功能：2.0.0 支持多面体曲面和 TIN。

- ✓ 函数支持 3d 并且不会丢失 z-index。
- ✓ 此方法支持形状字符串和曲线。
- ✓ 函数支持多面体曲面。
- ✓ 此函数支持三角形和不规则三角网面 (TIN)。

示例

Create a LineString with SRID 4269.

Code

```
SELECT ST_GeometryType(geom) AS geometry_type, ST_SRID(geom) AS srid
FROM ST_GeomFromEWKT('SRID=4269;LINESTRING(-71.160281 42.258729,-71.160837 42.259113,-71.161144 42.25932)') AS geom;
```

Output

geometry_type	srid
ST_LineString	4269
(1 row)	

Create a MultiLineString with SRID 4269.

Code

```
SELECT ST_GeometryType(geom) AS geometry_type, ST_SRID(geom) AS srid
FROM ST_GeomFromEWKT('SRID=4269;MULTILINESTRING((-71.160281 42.258729,-71.160837 42.259113,-71.161144 42.25932))') AS geom;
```

Output

geometry_type	srid
ST_MultiLineString	4269
(1 row)	

Create a Point with SRID 4269.

Code

```
SELECT ST_GeometryType(geom) AS geometry_type, ST_SRID(geom) AS srid
FROM ST_GeomFromEWKT('SRID=4269;POINT(-71.064544 42.28787)') AS geom;
```

Output

```
geometry_type | srid
-----+-----
ST_Point      | 4269
(1 row)
```

Create a Polygon with SRID 4269.

Code

```
SELECT ST_GeometryType(geom) AS geometry_type, ST_SRID(geom) AS srid
FROM 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)))') AS geom;
```

Output

```
geometry_type | srid
-----+-----
ST_Polygon    | 4269
(1 row)
```

Create a MultiPolygon with SRID 4269.

Code

```
SELECT ST_GeometryType(geom) AS geometry_type, ST_SRID(geom) AS srid
FROM 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)))') AS geom;
```

Output

```
geometry_type | srid
-----+-----
ST_MultiPolygon | 4269
(1 row)
```

Create a 3D CircularString without an SRID.

Code

```
SELECT ST_GeometryType(geom) AS geometry_type, ST_SRID(geom) AS srid
FROM ST_GeomFromEWKT('CIRCULARSTRING(220268 150415 1,220227 150505 2,220227 150406 3)') AS geom;
```

Output

geometry_type	srid
ST_CircularString	0
(1 row)	

Create a PolyhedralSurface without an SRID.

Code

```
SELECT ST_GeometryType(geom) AS geometry_type, ST_SRID(geom) AS srid
FROM 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 geom;
```

Output

geometry_type	srid
ST_PolyhedralSurface	0
(1 row)	

相关信息

[ST_AsEWKT](#), [ST_GeomFromText](#)

7.8.1.7 ST_GeomFromMARC21

ST_GeomFromMARC21 — 将 MARC21/XML 地理数据并返回 PostGIS 几何对象。

Synopsis

geometry **ST_GeomFromMARC21** (text marcxml);

描述

此函数从 MARC21/XML 构建 PostGIS 几何对象,其中可以包含 POINT 或 POLYGON。如果同一 MARC21/XML 对象中有多个地理数据条目,将返回 MULTIPOINT 或 MULTIPOLYGON。如果对象包含混合几何类型,将返回 GEOMETRYCOLLECTION。如果 MARC21/XML 对象不包含任何地理数据 (数据字段: 034), 返回 NULL。

支持的 LOC MARC21/XML 版本:

- [MARC21/XML 1.1](#)

可用性: 3.3.0, 需要 libxml2 2.6+

**Note**

在 MARC21/XML 的地理数据中，目前无法描述坐标的空参考系，因此函数将始终返回 SRID 0 的几何形。

**Note**

返回的 POLYGON 几何形将始终是逆时针方向。

示例

包含 hddd.dddddd 的 POINT 的 MARC21/XML 地理数据

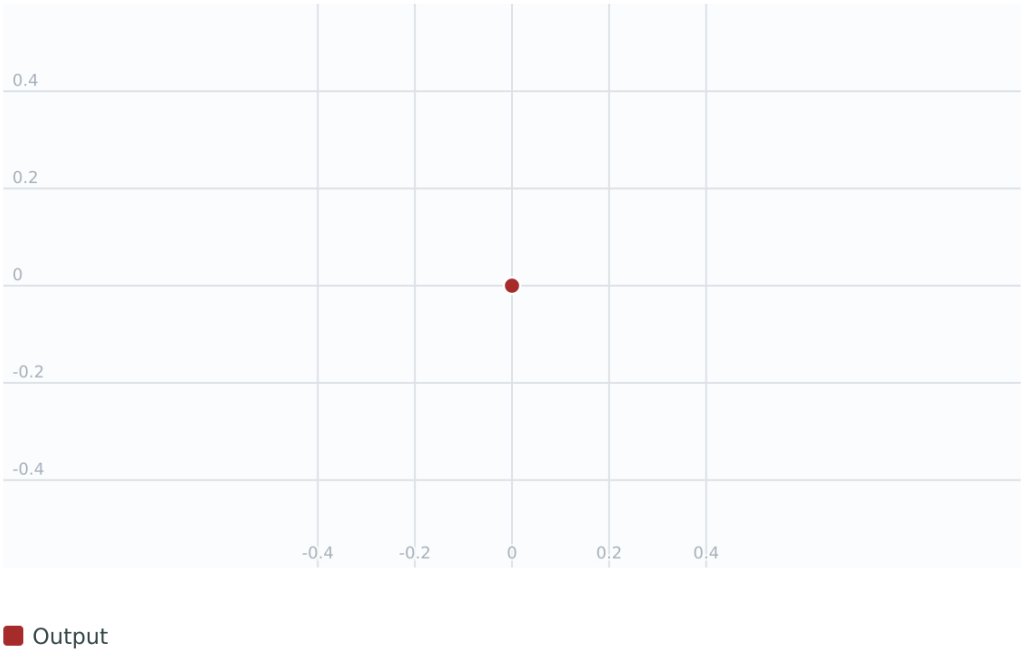
Code

```
SELECT 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
>'
);
```

Output

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

Figure



包含 hddmmss 的 个 POLYGON 的 MARC21/XML 地理数据

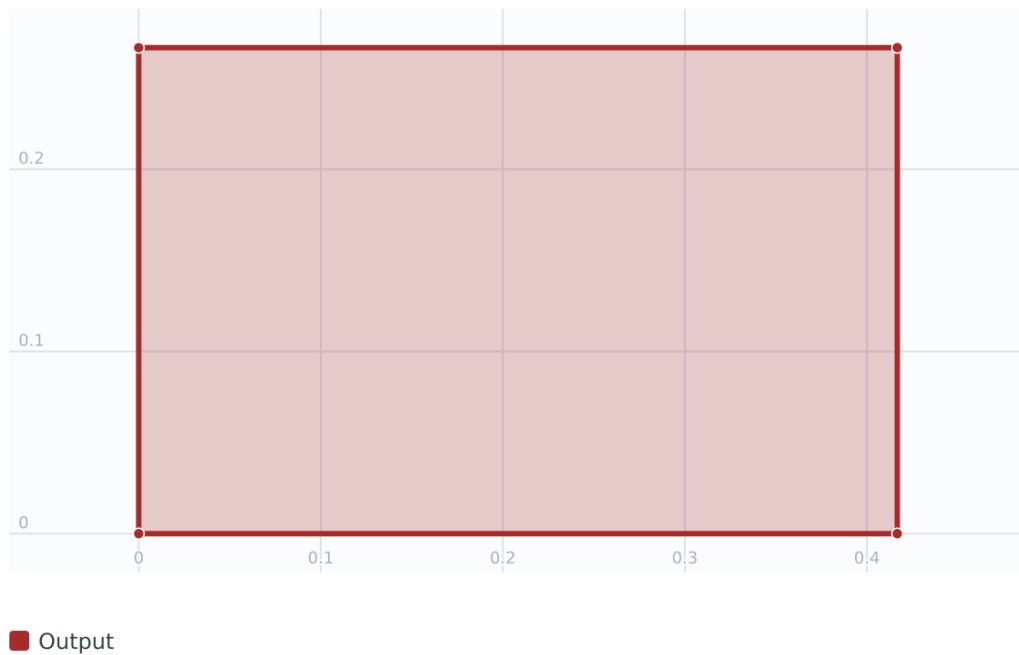
Code

```
SELECT 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
>'
);
```

Output

POLYGON((13.1 52.65,13.51666666666667 52.65,13.51666666666667 52.38333333333333,13.1 52.38333333333333,13.1 52.65))

Figure



☒☒包含 POLYGON 和 POINT 的 MARC21/XML 地理数据：

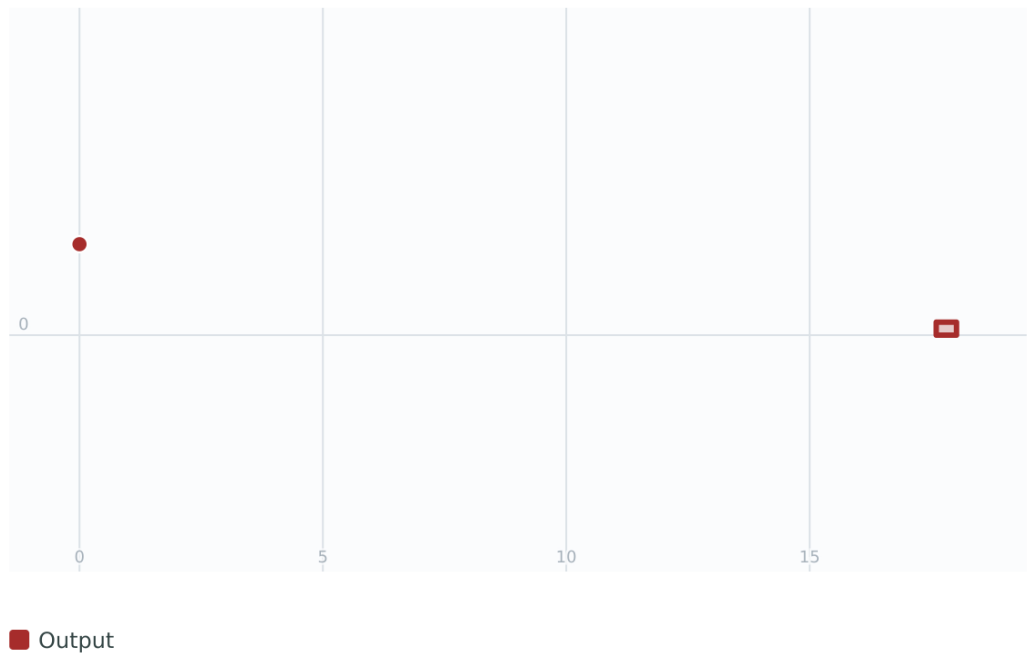
Code

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

Output

```
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))
```

Figure



相关信息

[ST_AsMARC21](#)

7.8.1.8 ST_GeometryFromText

ST_GeometryFromText — Returns a geometry from Well-Known Text (WKT). Alias for ST_GeomFromText.

Synopsis

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

描述

This function is an alias for [ST_GeomFromText](#) and provides the same signatures.

可用性：1.2.2

- ✓ 此方法符合了 SQL 1.1 的 OGC 功能规范。
- ✓ 此方法符合了 SQL/MM 规范。SQL-MM 3: 5.1.40

相关信息

[ST_GeomFromText](#)

7.8.1.9 ST_GeomFromText

ST_GeomFromText — 从已知的文本表示 (WKT) 返回指定的 ST_Geometry 对象。

Synopsis

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

描述

从 OGC 已知的文本表示 (WKT) 构造 PostGIS ST_Geometry 对象。



Note

ST_GeomFromText 函数有两种变体。第一个不采用 SRID，并返回没有定义空参考系 (SRID=0) 的几何对象。第二个将 SRID 作为第二个参数，并返回一个几何对象，该几何对象包含此 SRID 作为其元数据的一部分。



此方法符合了 **SQL 1.1 的 OGC 功能规范**。s3.2.6.2 - 可参数 SRID 用于规范一致性。



该方法符合了 SQL/MM 规范。SQL-MM 3: 5.1.40



此方法支持几何字符串和曲线。



Note

它不符合 OGC，但 **ST_MakePoint** 比 ST_GeomFromText 和 ST_PointFromText 更快。如果一个点是，如果使用数字作为坐标，则更容易。一种变体是 **ST_Point**，它在速度方面与 **ST_MakePoint** 相似，并且符合 OGC 规范，但它支持 2D 点。

更改：2.0.0 在 PostGIS 的早期版本中，允许 ST_GeomFromText('GEOMETRYCOLLECTION(EMPTY)')。为了更好地符合 SQL/MM 规范，这在 PostGIS 2.0.0 中是不合法的。现在写成 ST_GeomFromText('GEOMETRYCOLLECTION(EMPTY)')

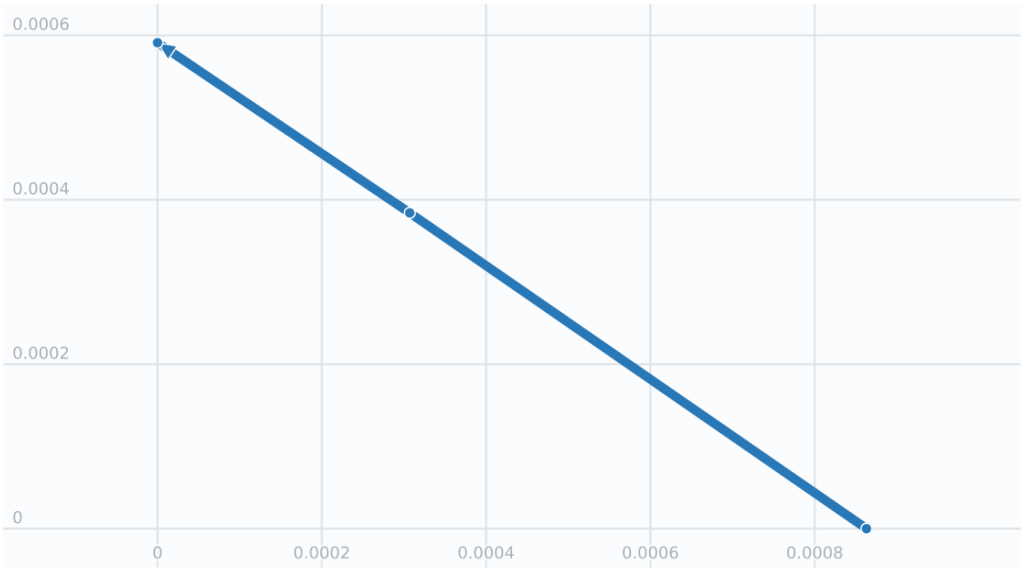
示例

Create a LineString with SRID 0.

Code

```
SELECT ST_SRID(ST_GeomFromText('LINESTRING(-71.160281 42.258729,-71.160837 42.259113,-71.161144 42.25932)'));
```

Figure



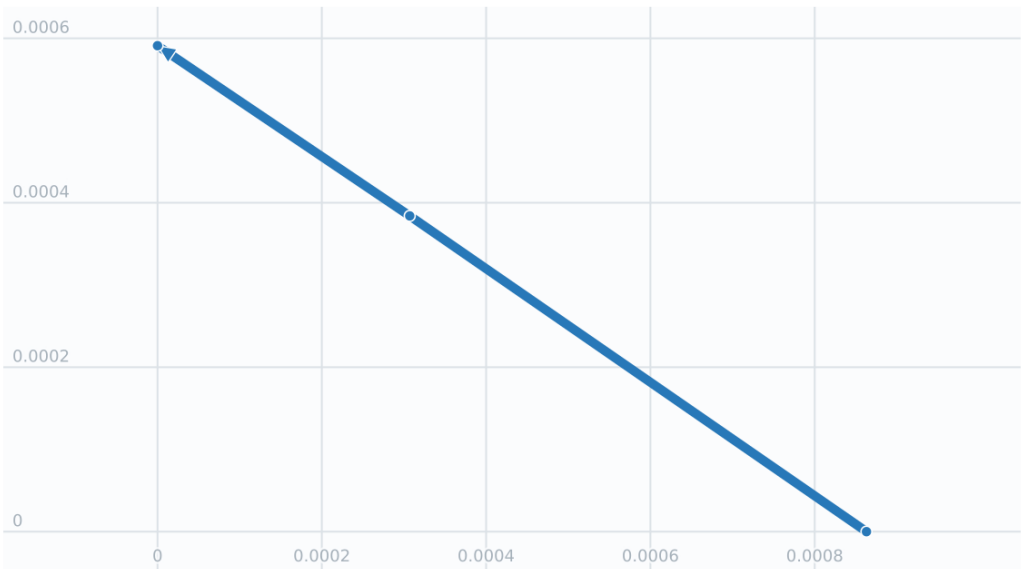
Code

Create the same LineString with SRID 4269.

Code

```
SELECT ST_SRID(ST_GeomFromText( 'LINESTRING(-71.160281 42.258729,-71.160837 ↵
42.259113,-71.161144 42.25932)', 4269));
```

Figure

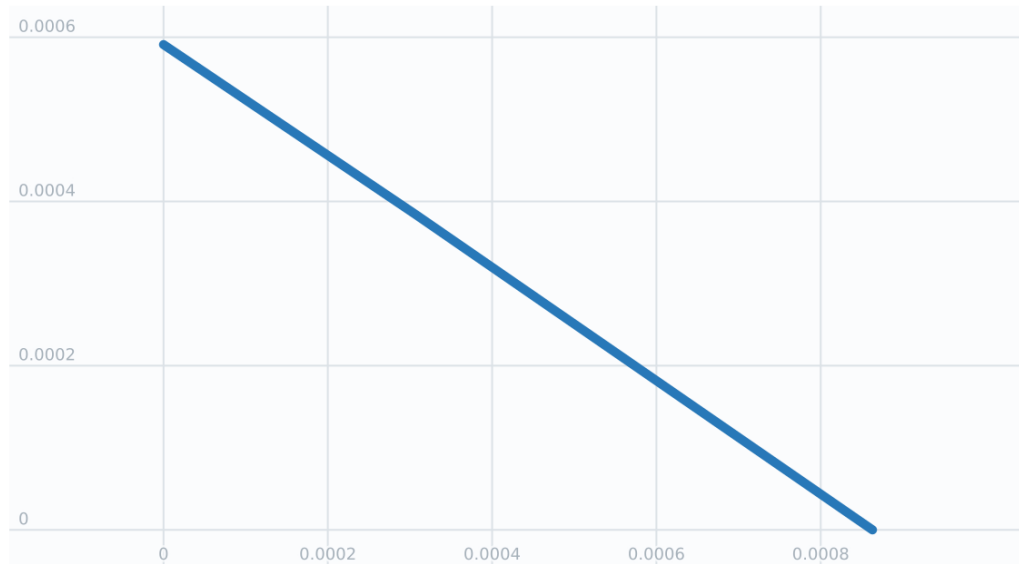


Code

Create a MultiLineString.

Code

```
SELECT ST_GeometryType(ST_GeomFromText('MULTILINESTRING((-71.160281 42.258729,-71.160837 ↵
42.259113,-71.161144 42.25932))'));
```

Figure

■ Code

Create a Point.

Code

```
SELECT ST_GeometryType(ST_GeomFromText('POINT(-71.064544 42.28787)'));
```

Output

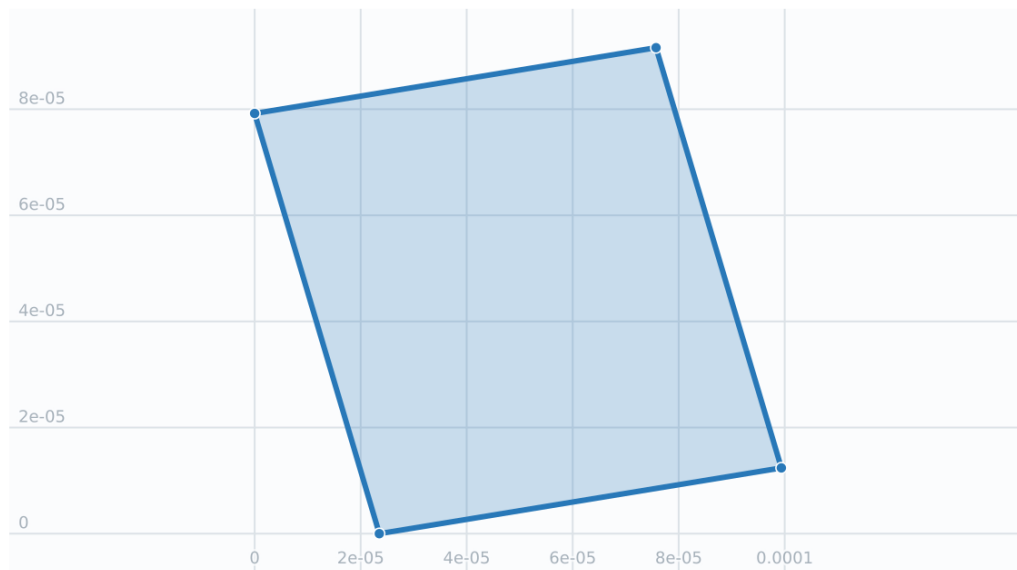
ST_Point

Create a Polygon.

Code

```
SELECT ST_GeometryType(ST_GeomFromText('POLYGON((-71.1776585052917 ↵
42.3902909739571,-71.1776820268866 42.3903701743239,
-71.1776063012595 42.3903825660754,-71.1775826583081 42.3903033653531,-71.1776585052917 ↵
42.3902909739571))'));
```


Figure

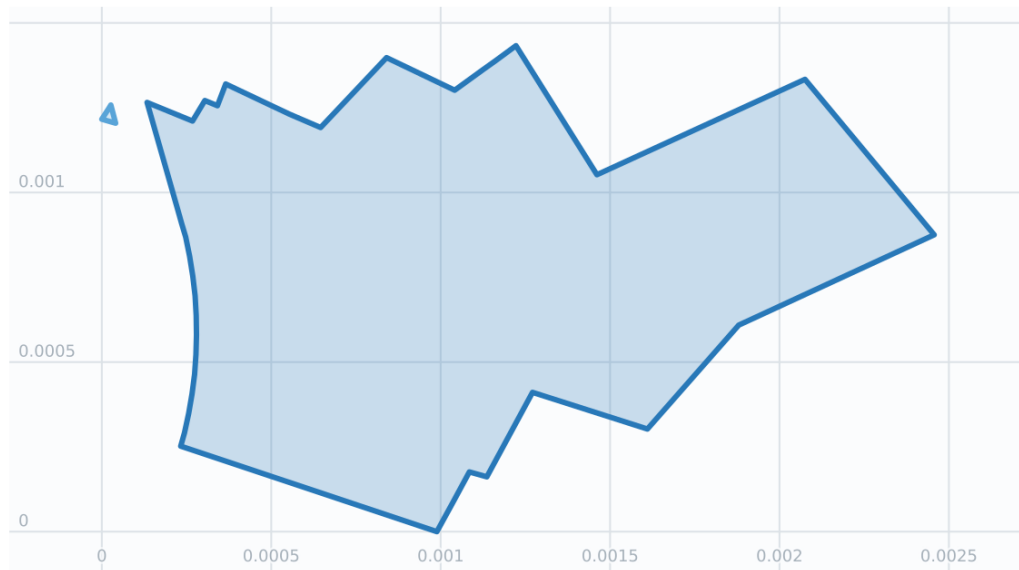


■ Code

Create a MultiPolygon with SRID 4326.

Code

```
SELECT ST_GeometryType(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));
```

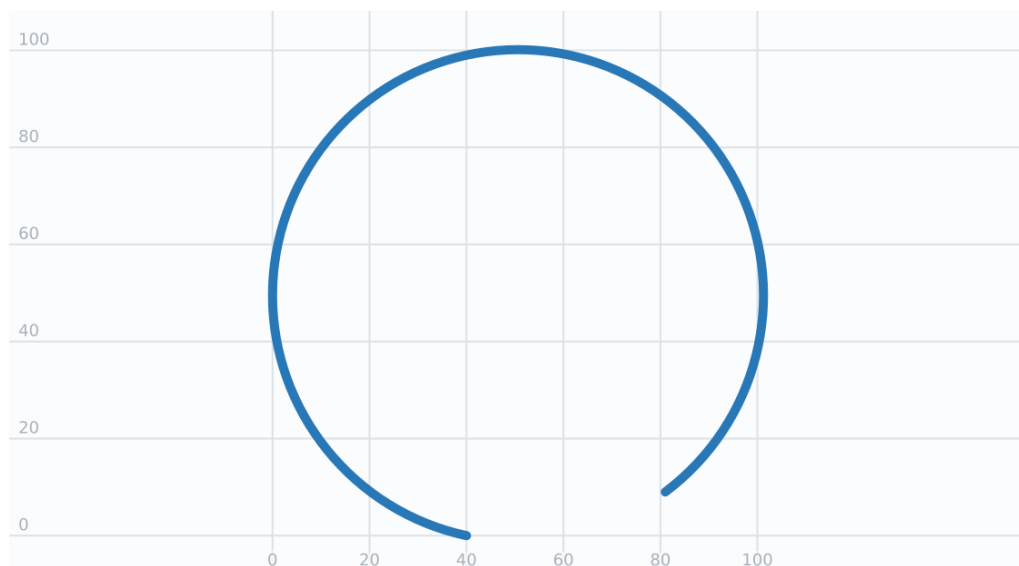
Figure

■ Code

Create a CircularString.

Code

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

Figure

■ Code

相关信息

[ST_GeomFromEWKT](#), [ST_GeomFromWKB](#), [ST_SRID](#)

7.8.1.10 ST_LineFromText

`ST_LineFromText` — 使用指定的 SRID 根据 WKT 表示构建几何图形。如果未给出 SRID，默认为 0。

Synopsis

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

描述

使用指定的 SRID 从 WKT 构建几何图形。如果未给出 SRID，默认为 0。如果输入的 WKT 不是 LINESTRING，返回 null。



Note
OGC 规范 3.2.6.2 - 可指定 SRID 用于规范。



Note
如果您知道所有几何图形都是字符串，使用 `ST_GeomFromText` 会更有效。此函数调用 `ST_GeomFromText` 并评估是否返回字符串。

- ✔ 此方法符合了 [SQL 1.1 的 OGC 规范功能规范](#)。 s3.2.6.2
- ✔ 此方法符合了 SQL/MM 规范。 SQL-MM 3: 7.2.8

示例

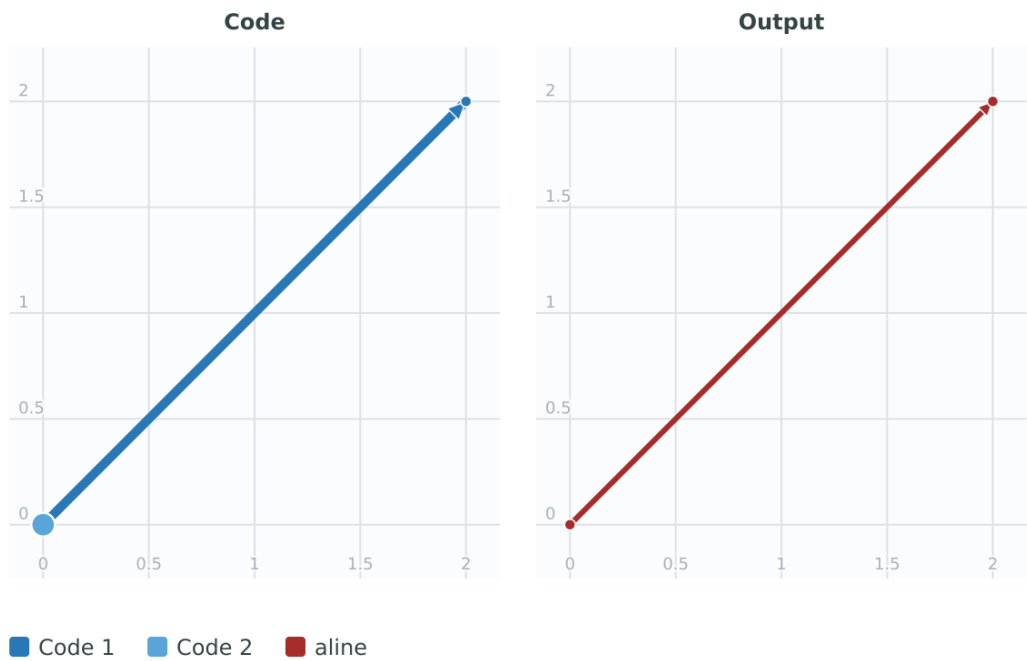
Code

```
SELECT ST_LineFromText('LINESTRING(1 2,3 4)') AS aline,
ST_LineFromText('POINT(1 2)') IS NULL AS null_return;
```

Output

aline	null_return
LINESTRING(1 2,3 4)	t

Figure



相关信息

[ST_GeomFromText](#)

7.8.1.11 ST_MLineFromText

ST_MLineFromText — 从 WKT 表示形式返回指定的 ST_MultiLineString 。

Synopsis

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

描述

使用指定的 SRID 从已知的文本 (WKT) 构建几何图形。如果未给出 SRID，默认为 0。

OGC 规范 3.2.6.2 - 可指定 SRID 用于规范

如果 WKT 不是多行字符串，返回 NULL



Note
如果您确定所有 WKT 几何图形都是点，不要使用此函数。它比 ST_GeomFromText 慢，因为它添加了额外的步骤。

- ✓ 此方法符合了 SQL 1.1 的 OGC 功能规范。 s3.2.6.2
- ✓ 方法符合了 SQL/MM 规范。 SQL-MM 3: 9.4.4

示例

Code

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

相关信息

[ST_GeomFromText](#)

7.8.1.12 ST_MPointFromText

ST_MPointFromText — 使用指定的 SRID 从 WKT 构建几何图形。如果未给出 SRID，默认为 0。

Synopsis

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

描述

从已知文本（WKT）表示和指定 SRID 生成几何图形。如果未给出 SRID，为 0（未知）。

OGC 规范 3.2.6.2 - 可指定 SRID 用于规范

如果 WKT 不是多点，返回 NULL



Note

如果您确定所有 WKT 几何图形都是点，不要使用此函数。它比 ST_GeomFromText 慢，因为它添加了额外的步骤。



此方法符合了 SQL 1.1 的 OGC 规范功能规范。 3.2.6.2



该方法符合了 SQL/MM 规范。 SQL-MM 3: 9.2.4

示例

Code

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

相关信息

[ST_GeomFromText](#)

7.8.1.13 ST_MPolyFromText

ST_MPolyFromText — 使用指定的 SRID 从 WKT 制作多多边形几何体。如果未给出 SRID，默认为 0。

Synopsis

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

描述

从已知文本（WKT）表示形式和指定 SRID 生成多面。如果未给出 SRID，则为 0（未知）。

OGC 规范 3.2.6.2 - 可指定 SRID 用于规范

如果 WKT 不是多面形，将引发错误



Note

如果您确定所有 WKT 几何形都是多面形，则不要使用此函数。它比 ST_GeomFromText 慢，因为它添加了额外的步骤。



此方法符合了 SQL 1.1 的 OGC 规范功能规范。 s3.2.6.2



此方法符合了 SQL/MM 规范。 SQL-MM 3: 9.6.4

示例

Create a 3D MultiPolygon with an interior ring and SRID 0.

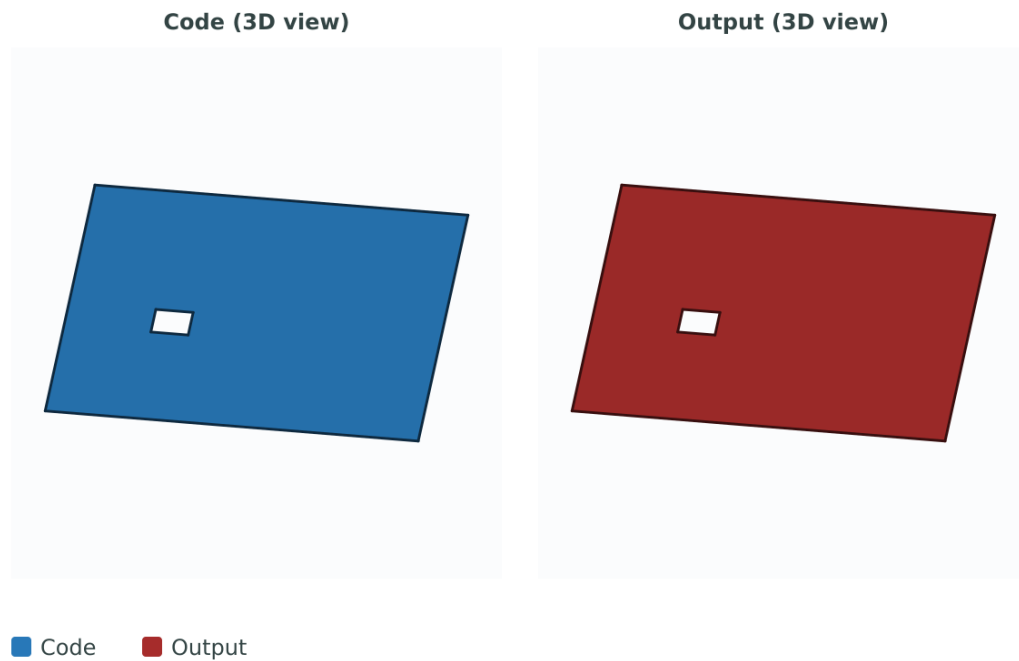
Code

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

Output

```
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)))
```

| Figure

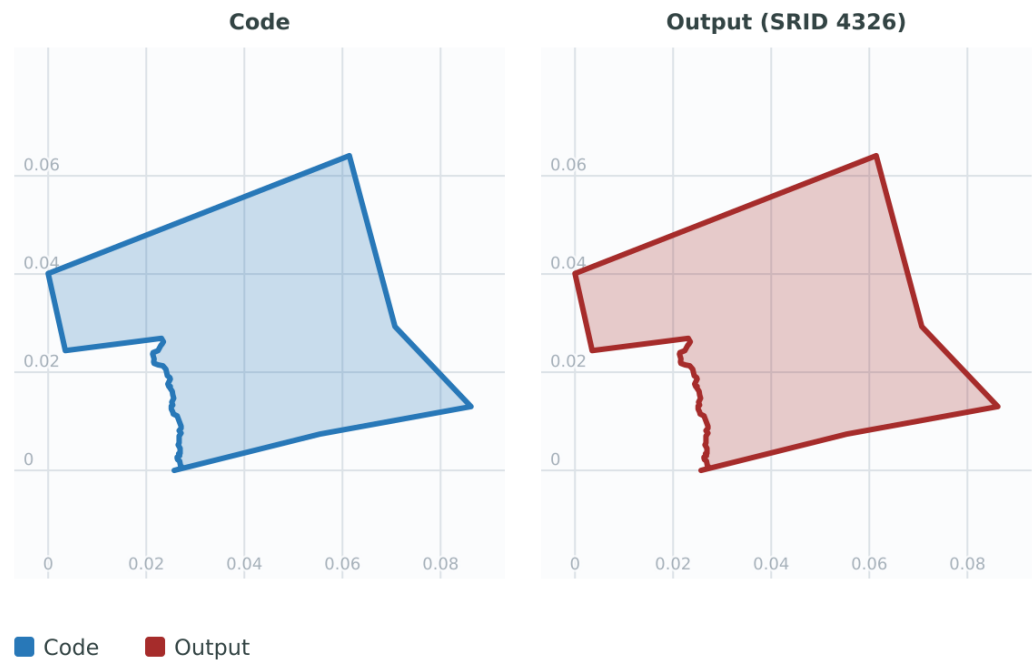


Create a MultiPolygon with SRID 4326.

| Code

```
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
);
```

Figure



相关信息

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

7.8.1.14 ST_PointFromText

`ST_PointFromText` — 使用指定的 SRID 从 WKT 构建点几何图形。如果未输出 SRID，则默认为未知。

Synopsis

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

描述

从 OGC 已知的文本表示构造 PostGIS `ST_Geometry` 点对象。如果未输出 SRID，则默认为未知（当前为 0）。如果几何图形不是 WKT 点表示，则返回 `null`。如果 WKT 完全无效，则抛出错误。



Note
`ST_PointFromText` 函数有 2 个形式，第一个不采用 SRID，并返回没有定义空参考系的几何图形。第二个将空参考 id 作为第二个参数，并返回一个 `ST_Geometry`，其中包含此 srid 作为其元数据的一部分。srid 必须在 `spatial_ref_sys` 表中定义。



Note
如果您确定所有 WKT 几何图形都是点，则不要使用此函数。它比 `ST_GeomFromText` 慢，因为它添加了额外的步骤。如果您从纬度/经度坐标构建点并且更关心性能和准确性而不是 OGC 合规性，请使用 `ST_MakePoint` 或 OGC 合规名称 `ST_Point`。

- ✓ 此方法符合了 SQL 1.1 的 OGC 功能规范。s3.2.6.2 - 可参数 SRID 用于规范一致性。
- ✓ 此方法符合了 SQL/MM 规范。SQL-MM 3: 6.1.8

示例

Create a point with the default SRID 0.

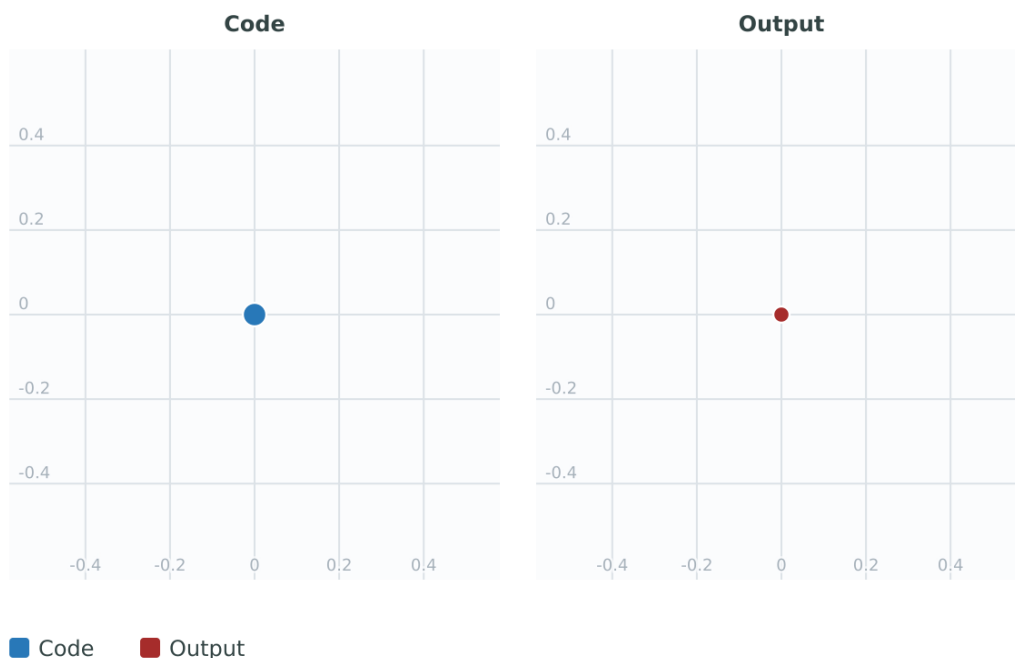
Code

```
SELECT ST_PointFromText('POINT(-71.064544 42.28787)');
```

Output

```
POINT(-71.064544 42.28787)
```

Figure



Create a point with SRID 4326.

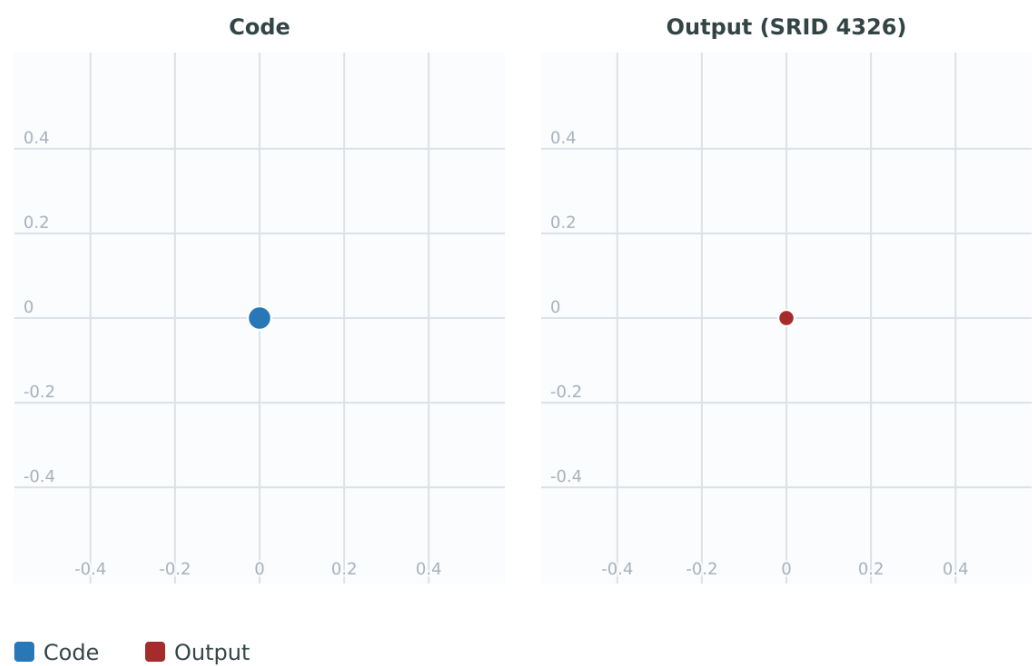
Code

```
SELECT ST_PointFromText('POINT(-71.064544 42.28787)', 4326);
```

Output

```
SRID=4326;POINT(-71.064544 42.28787)
```

Figure



相关信息

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

7.8.1.15 ST_PolygonFromText

ST_PolygonFromText — 使用指定的 SRID 从 WKT 构建几何图形。如果未给出 SRID，默认为 0。

Synopsis

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

描述

使用指定的 SRID 从 WKT 构建几何图形。如果未给出 SRID，默认为 0。如果 WKT 不是多边形，返回 null。
OGC 规范 3.2.6.2 - 可指定 SRID 用于规范



Note

如果您确定所有 WKT 几何图形都是多边形，请不要使用此函数。它比 ST_GeomFromText 慢，因为它添加了额外的步骤。

- 此方法符合了 SQL 1.1 的 OGC 功能规范。s3.2.6.2
- 该方法符合了 SQL/MM 规范。SQL-MM 3: 8.3.6

示例

The result is returned as a native geometry. The documentation test runner compares that value directly with the readable EWKT shown below and uses the same geometry for the generated figure.

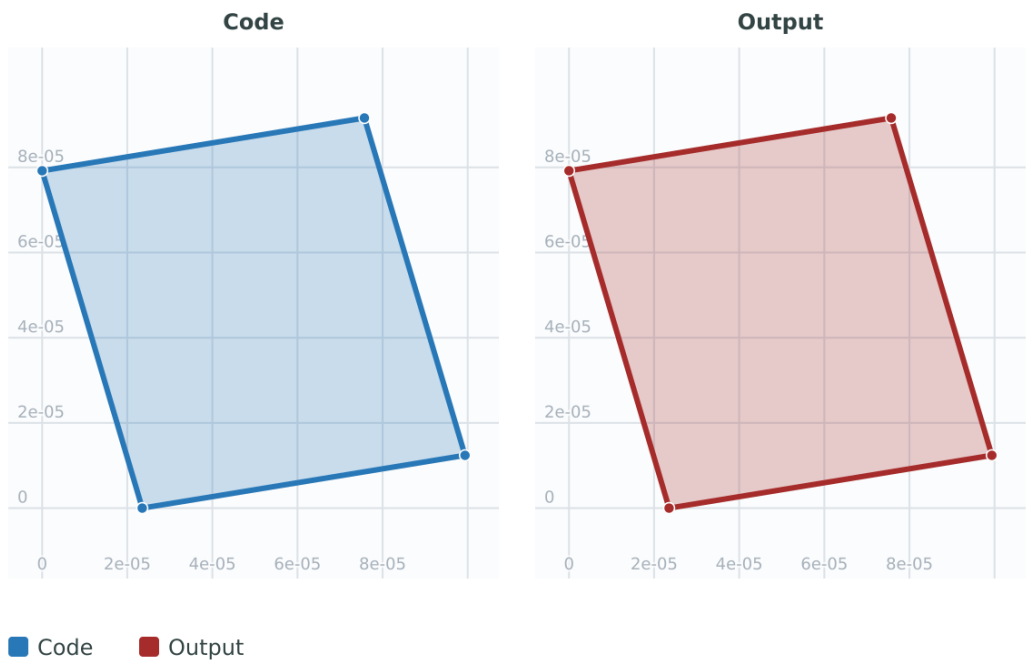
Code

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

Output

```
POLYGON((-71.1776585052917 42.3902909739571,-71.1776820268866 42.3903701743239,-71.1776063012595 42.3903825660754,-71.1775826583081 42.3903033653531,-71.1776585052917 42.3902909739571))
```

Figure



Code

```
SELECT ST_PolygonFromText('POINT(1 2)') IS NULL AS point_is_notpoly;
```

Output

```
t
```

相关信息

[ST_GeomFromText](#)

7.8.1.16 ST_WKTToSQL

ST_WKTToSQL — Returns a geometry from Well-Known Text (WKT). SQL/MM alias for ST_GeomFromText.

Synopsis

geometry **ST_WKTToSQL**(text WKT);

描述

This function is the SQL/MM alias for **ST_GeomFromText** with no SRID argument.



该方法符合了 SQL/MM 规范。SQL-MM 3: 5.1.34

相关信息

ST_GeomFromText

7.8.2 已知的二进制文件 (WKB)

7.8.2.1 ST_GeogFromWKB

ST_GeogFromWKB — 从已知的二进制几何表示 (WKB) 或扩展的已知的二进制 (EWKB) 构建地理示例。

Synopsis

geography **ST_GeogFromWKB**(bytea wkb);

描述

ST_GeogFromWKB 函数采用已知的几何二进制表示 (WKB) 或 PostGIS 扩展 WKB，并构建适当地理类型的示例。该函数起到了 SQL 中几何工厂的作用。

如果未指定 SRID，默认为 4326 (WGS 84 经纬度)。



此方法支持点字符串和曲线。

示例

Although bytea rep contains single \, these need to be escaped when inserting into a table.

Code

```
SELECT 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@');
```

Output

```
LINESTRING(-113.98 39.198, -113.981 39.195)
```

Figure



相关信息

[ST_GeogFromText](#), [ST_AsBinary](#)

7.8.2.2 ST_GeomFromEWKB

ST_GeomFromEWKB — 从扩展已知二进制表示 (EWKB) 返回指定的 ST_Geometry。

Synopsis

geometry **ST_GeomFromEWKB**(bytea EWKB);

描述

Constructs a PostGIS geometry from an Extended Well-Known Binary (EWKB) representation.



Note
EWKB 格式不是 OGC 标准，而是包含空参考系 (SRID) 符号的 PostGIS 特定格式

增强功能：2.0.0 支持多面体曲面和 TIN。

- ✓ 函数支持 3d 并且不会丢失 z-index。
- ✓ 此方法支持形状字符串和曲线。
- ✓ 函数支持多面体曲面。
- ✓ 此函数支持三角形和不规则三角网面 (TIN)。

示例

This example constructs `LINESTRING(-71.160281 42.258729, -71.160837 42.259113, -71.161144 42.259113)` in NAD83 longitude/latitude (SRID 4269) from its EWKB representation.

PostgreSQL escape string syntax (`E'...'`) always interprets backslash escapes. Each backslash therefore has to be doubled so that the bytea input parser receives the octal escapes.

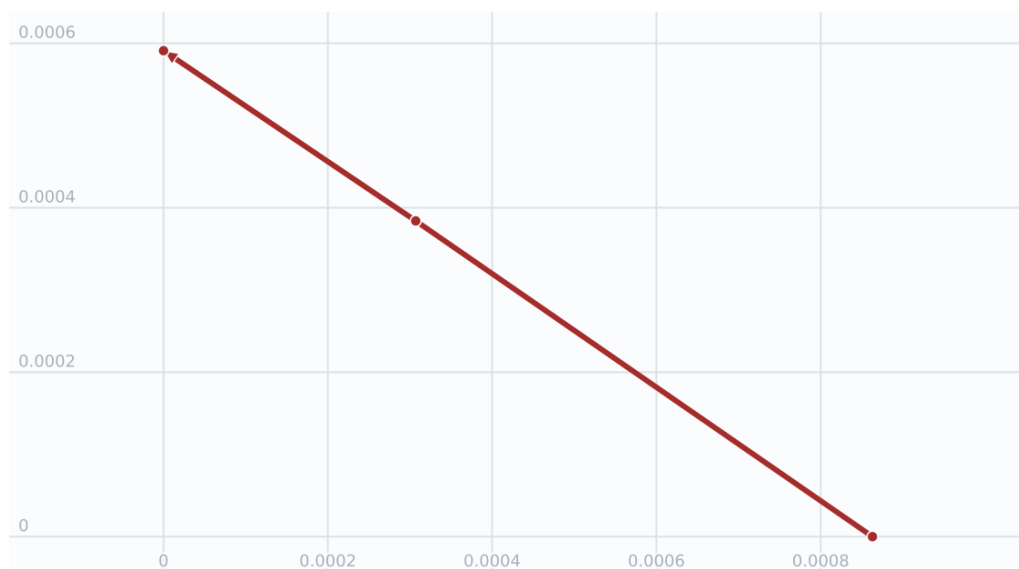
Code

```
SELECT ST_GeomFromEWKB((
  E'\\001\\002\\000\\000\\040\\255\\020\\000\\000\\003\\000\\000\\000\\344\\112\\075' ||
  E'\\013\\102\\312\\121\\300\\156\\303\\050\\010\\036\\041\\105\\100\\047\\277\\105' ||
  E'\\047\\113\\312\\121\\300\\366\\173\\142\\235\\052\\041\\105\\100\\225\\174\\354' ||
  E'\\056\\120\\312\\121\\300\\160\\231\\323\\145\\061\\041\\105\\100'
)::bytea);
```

Output

```
0102000020 ←
AD10000003000000E44A3D0B42CA51C06EC328081E21454027BF45274BCA51C0F67B629D2A214540957CEC2E50CA51C07099D36531214540
```

Figure



■ Output

With `standard_conforming_strings` enabled (the default), ordinary string literals preserve backslashes. The same EWKB can therefore be written with a single backslash before each octal byte value.

Code

```
SELECT ST_GeomFromEWKB((
  '\001\002\000\000\040\255\020\000\000\003\000\000\000\344\112\075' ||
  '\013\102\312\121\300\156\303\050\010\036\041\105\100\047\277\105' ||
  '\047\113\312\121\300\366\173\142\235\052\041\105\100\225\174\354' ||
  '\056\120\312\121\300\160\231\323\145\061\041\105\100'
)::bytea);
```

Output

```
0102000020 ↵
AD10000003000000E44A3D0B42CA51C06EC328081E21454027BF45274BCA51C0F67B629D2A214540957CEC2E50CA51C0
```

相关信息

[ST_AsBinary](#), [ST_AsEWKB](#), [ST_GeomFromWKB](#)

7.8.2.3 ST_GeomFromWKB

ST_GeomFromWKB — 从已知的二进制几何表示 (WKB) 和可选的 SRID 构建几何实例。

Synopsis

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

描述

ST_GeomFromWKB 函数采用已知的几何形状二进制表示形式和可选参考系 ID (SRID)，并构建相应几何形状类型的实例。该函数起到了 SQL 中几何工厂的作用。它是 **ST_WKBToSQL** 的替代名称。

如果未指定 SRID，则默认为 0（未知）。

- ✓ 此方法符合了 [SQL 1.1 的 OGC 功能规范](#)。s3.2.7.2 - 可选参数 SRID 用于符合规范
- ✓ 该方法符合了 SQL/MM 规范。SQL-MM 3: 5.1.41
- ✓ 此方法支持形状字符串和曲线。

示例

Although bytea rep contains single \, these need to be escaped when inserting into a table. Unless `standard_conforming_strings` is set to on.

Code

```
SELECT ST_GeomFromWKB(E'\\001\\002\\000\\000\\000\\002\\000\\000\\000\\037\\205\\353Q ↵
\\270~\\\\\\\\300\\323Mb\\020X\\231C@\\020X9\\264\\310~\\\\\\\\300)\\\\\\\\217\\302\\365\\230 ↵
C@', 4326);
```

Output

```
SRID=4326;LINESTRING(-113.98 39.198,-113.981 39.195)
```

Figure



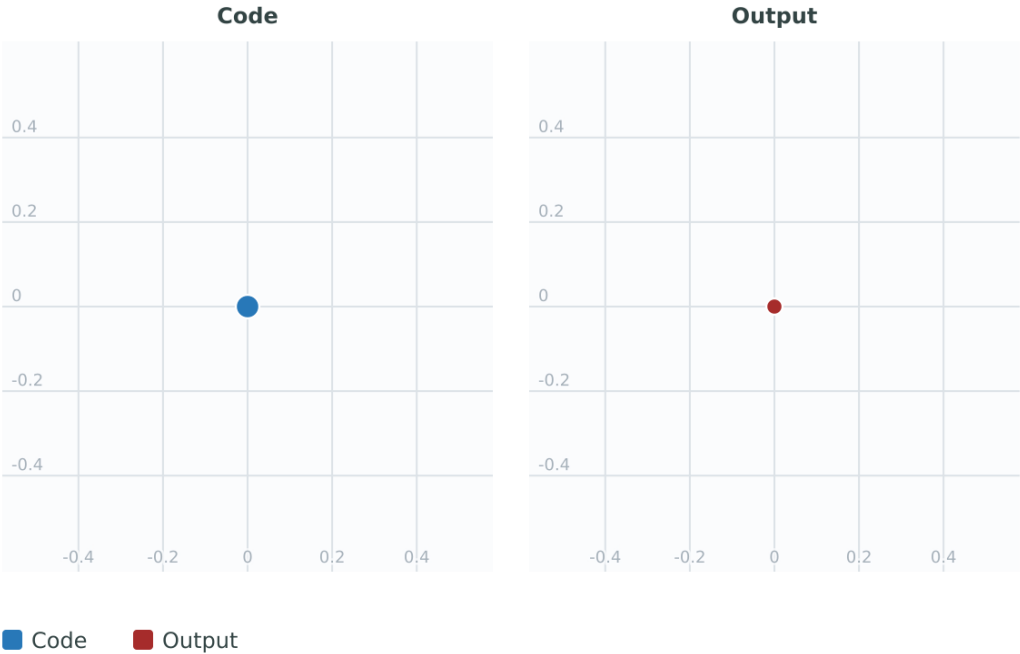
Code

```
SELECT ST_GeomFromWKB(ST_AsEWKB( 'POINT(2 5)'::geometry)
);
```

Output

POINT(2 5)

Figure



The PostgreSQL **decode** function converts a hexadecimal WKB string to the bytea value accepted by ST_GeomFromWKB.

Code

```
SELECT ST_AsEWKT(ST_GeomFromWKB(
    decode('0101000000e5d022dbf93e2e40dbf97e6abc743540', 'hex'),
    4326
)) AS geom;
```

Output

```
SRID=4326;POINT(15.123 21.456)
```

相关信息

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

7.8.2.4 ST_LineFromWKB

ST_LineFromWKB — Creates a LINESTRING from a WKB representation, optionally using a given SRID.

Synopsis

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

描述

The ST_LineFromWKB function takes a Well-Known Binary (WKB) representation of a geometry and an optional Spatial Reference System ID (SRID), and creates a LINESTRING geometry. It is a type-specific geometry factory in SQL.

If an SRID is not specified, it defaults to 0. NULL is returned if the input bytea does not represent a LINESTRING geometry.



Note

OGC 规范 3.2.6.2 - 可选项 SRID 用于选项。



Note

If you know all your geometries are LINESTRINGs, it is more efficient to use [ST_GeomFromWKB](#) directly. This function calls [ST_GeomFromWKB](#) and adds validation that the result is a LINESTRING.



此方法符合了 [SQL 1.1](#) 的 OGC 功能规范。 s3.2.6.2



该方法符合了 SQL/MM 规范。 SQL-MM 3: 7.2.9

示例

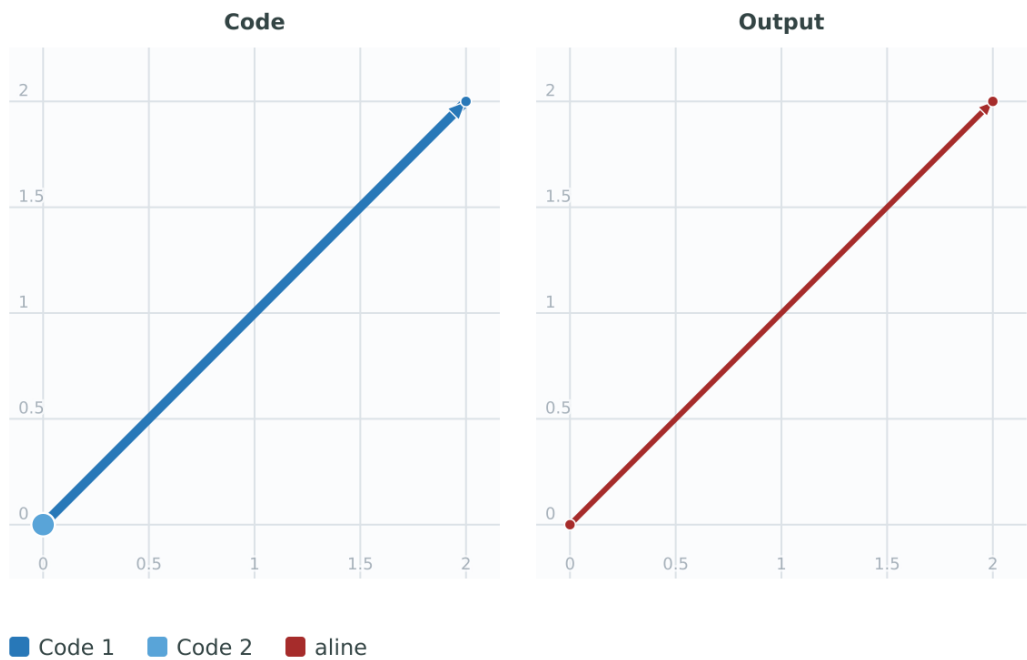
Code

```
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;
```

Output

aline	null_return
LINESTRING(1 2,3 4)	t

Figure



相关信息

ST_GeomFromWKB, ST_LinestringFromWKB

7.8.2.5 ST_LinestringFromWKB

ST_LinestringFromWKB — Creates a LINESTRING from a WKB representation, optionally using a given SRID.

Synopsis

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

描述

The `ST_LinestringFromWKB` function takes a Well-Known Binary (WKB) representation of a geometry and an optional Spatial Reference System ID (SRID), and creates a `LINESTRING` geometry. It is an alias for `ST_LineFromWKB`.

If an SRID is not specified, it defaults to 0. `NULL` is returned if the input bytea does not represent a `LINESTRING` geometry.



Note
OGC 规范 3.2.6.2 - 可 SRID 用于规范。



Note
If you know all your geometries are `LINESTRING`s, it is more efficient to use `ST_GeomFromWKB` directly. This function calls `ST_GeomFromWKB` and adds validation that the result is a `LINESTRING`.

- ✔ 此方法符合了 SQL 1.1 的 OGC 规范功能规范。 s3.2.6.2
- ✔ 方法符合了 SQL/MM 规范。 SQL-MM 3: 7.2.9

示例

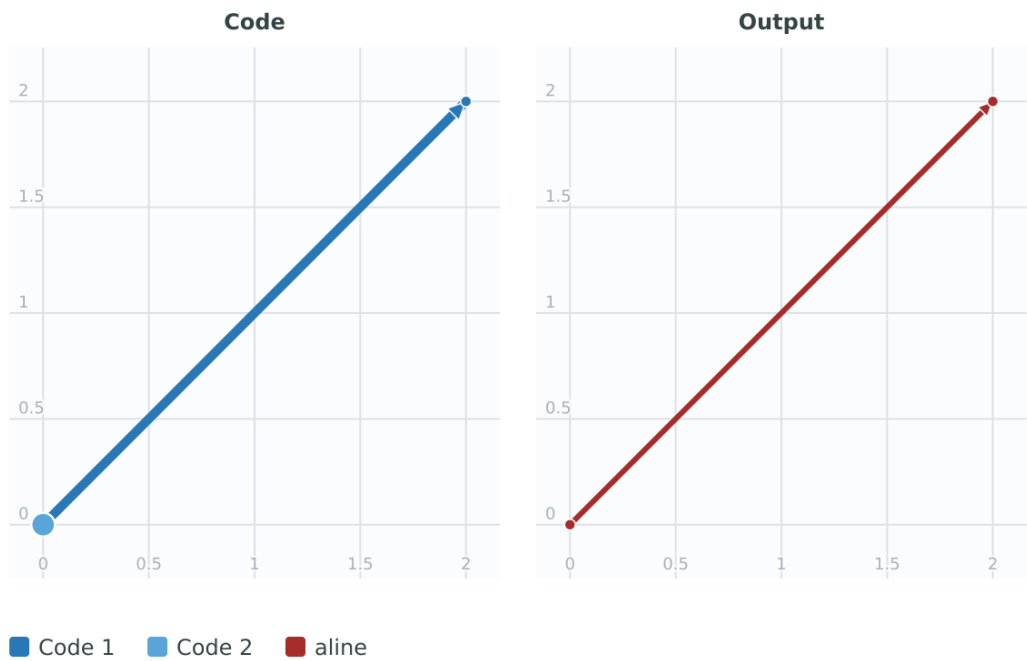
Code

```
SELECT ST_LinestringFromWKB(
  ST_AsBinary(ST_GeomFromText('LINESTRING(1 2,3 4)'))
) AS aline,
ST_LinestringFromWKB(
  ST_AsBinary(ST_GeomFromText('POINT(1 2)'))
) IS NULL AS null_return;
```

Output

aline	null_return
LINESTRING(1 2,3 4)	t

Figure



相关信息

[ST_GeomFromWKB](#), [ST_LineFromWKB](#)

7.8.2.6 ST_PointFromWKB

ST_PointFromWKB — 使用指定的 SRID 从 WKB 构建几何图形

Synopsis

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

描述

ST_PointFromWKB 函数采用已知的几何二进制表示形式和空间参考系 ID (SRID)，并构建适当几何类型的实例-在本例中为 POINT 几何。该函数起到了 SQL 中几何工厂的作用。

如果未指定 SRID，则默认为 0。如果输入字不表示 POINT 几何图形，则返回 NULL。

- ✓ 此方法符合了 SQL 1.1 的 OGC 功能规范。 s3.2.7.2
- ✓ 该方法符合了 SQL/MM 规范。 SQL-MM 3: 6.1.9
- ✓ 该函数支持 3d 并且不会丢失 z-index。
- ✓ 此方法支持图形字符串和曲线。

示例

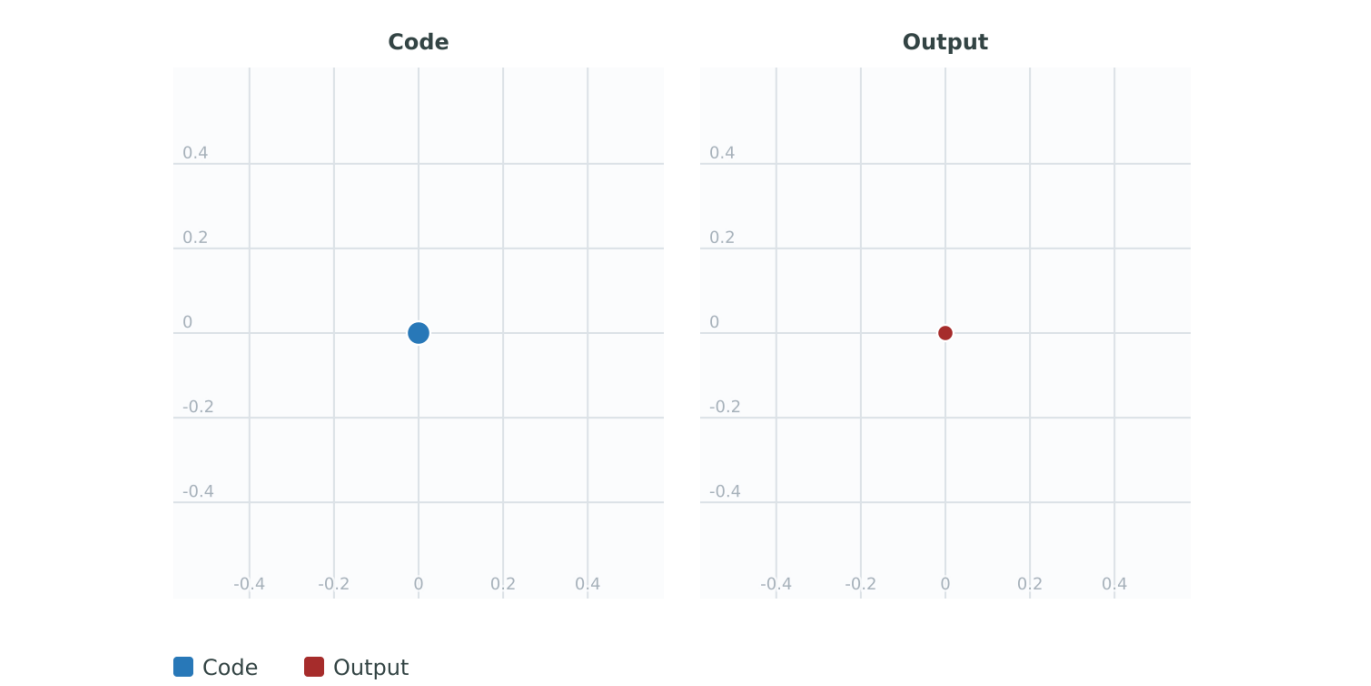
Code

```
SELECT ST_PointFromWKB(ST_AsEWKB('POINT(2 5)::geometry));
```

Output

POINT(2 5)

Figure



Code

```
SELECT ST_PointFromWKB(ST_AsEWKB('LINESTRING(2 5,2 6)::geometry));
```

Output

null

相关信息

[ST_GeomFromWKB](#), [ST_LineFromWKB](#)

7.8.2.7 ST_WKBToSQL

ST_WKBToSQL — Returns a geometry from Well-Known Binary (WKB). SQL/MM alias for ST_GeomFromWKB without an SRID argument.

Synopsis

geometry **ST_WKBToSQL**(bytea WKB);

描述

This function is the SQL/MM alias for **ST_GeomFromWKB** with no SRID argument.



☑方法☑☑了 SQL/MM ☑范。SQL-MM 3: 5.1.36

相关信息

ST_GeomFromWKB

7.8.3 其它格式

7.8.3.1 ST_Box2dFromGeoHash

ST_Box2dFromGeoHash — 从 GeoHash 字符串返回 BOX2D。

Synopsis

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

描述

从 GeoHash 字符串返回 BOX2D。

如果未指定 `precision`, ST_Box2dFromGeoHash 将返回基于☑入 GeoHash 字符串的完整精度的 BOX2D。

如果指定 `precision`, ST_Box2dFromGeoHash 将使用 GeoHash 中的☑多字符来☑建 BOX2D。☑低的精度☑会☑致☑大的 BOX2D, ☑大的☑会提高精度。

可用性 : 2.1.0

示例

Code

```
SELECT ST_Box2dFromGeoHash('9qqj7nmxcggy4d0dbxqz0');
```

Output

```
BOX(-115.172816 36.114646,-115.172816 36.114646)
```

Code

```
SELECT ST_Box2dFromGeoHash('9qqj7nmxcggy4d0dbxqz0', 0);
```

Output

```
BOX(-180 -90,180 90)
```

Code

```
SELECT ST_Box2dFromGeoHash('9qqj7nmxcggy4d0dbxqz0', 10);
```

| Output

```
BOX(-115.17282128334 36.1146408319473, -115.172810554504 36.1146461963654)
```

相关信息

[ST_GeoHash](#), [ST_GeomFromGeoHash](#), [ST_PointFromGeoHash](#)

7.8.3.2 ST_GeomFromGeoHash

`ST_GeomFromGeoHash` — 从 GeoHash 字符串返回几何图形。

Synopsis

geometry **ST_GeomFromGeoHash**(text geohash, integer precision=full_precision_of_geohash);

描述

从 GeoHash 字符串返回几何图形。几何图形将是表示 GeoHash 边界的多边形。

如果未指定 `precision`, `ST_GeomFromGeoHash` 返回基于输入 GeoHash 字符串的完整精度的多边形。

如果指定 `precision`, `ST_GeomFromGeoHash` 将使用 GeoHash 中的那么多字符来构建多边形。

可用性 : 2.1.0

示例

The cells are returned together so their relative sizes remain visible in a shared coordinate frame.

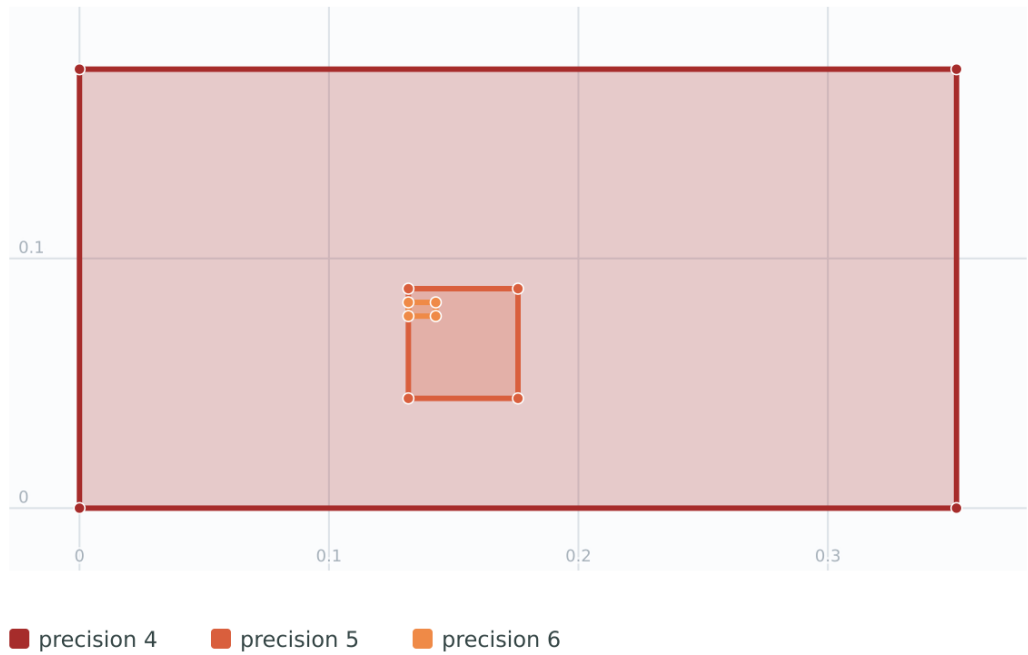
| Code

```
SELECT ST_GeomFromGeoHash('9qqj7nmxcggy4d0dbxqz0', 4) AS "precision 4", ST_GeomFromGeoHash(↵
('9qqj7nmxcggy4d0dbxqz0', 5) AS "precision 5", ST_GeomFromGeoHash('9 ↵
qqj7nmxcggy4d0dbxqz0', 6) AS "precision 6";
```

| Output

```
-[ RECORD 1 ]-----
precision 4 | POLYGON((-115.3125 36.03515625,-115.3125 36.2109375,-114.9609375 36.2109375,-114.9609375 ↵
36.03515625,-115.3125 36.03515625))
precision 5 | POLYGON((-115.1806640625 36.0791015625,-115.1806640625 36.123046875,-115.13671875 ↵
36.123046875,-115.13671875 36.0791015625,-115.1806640625 36.0791015625))
precision 6 | POLYGON((-115.1806640625 36.112060546875,-115.1806640625 36.1175537109375,-115.169677734375 ↵
36.1175537109375,-115.169677734375 36.112060546875,-115.1806640625 36.112060546875))
```

Figure



相关信息

[ST_GeoHash](#), [ST_Box2dFromGeoHash](#), [ST_PointFromGeoHash](#)

7.8.3.3 ST_GeomFromGML

ST_GeomFromGML — 将几何图形的 GML 表示形式作输入并输出 PostGIS 几何对象

Synopsis

```
geometry ST_GeomFromGML(text geomgml);
geometry ST_GeomFromGML(text geomgml, integer srid);
```

描述

根据 OGC GML 制表生成 PostGIS ST_Geometry 对象。
ST_GeomFromGML 适用于 GML 的几何部分。将其用于整个 GML 文档会引发错误。
支持的 OGC GML 版本包括：

- GML 3.2.1 命名空间
- GML 3.1.1 功能配置文件 SF-2（向后兼容 GML 3.1.0 和 3.0.0）
- GML 2.1.2

有关 OGC GML 规范，参看<http://www.opengeospatial.org/standards/gml>：

可用性：需要 1.5 libxml2 1.6+

增强功能：2.0.0 支持多面体曲面和 TIN。

增强：2.0.0 引入了多面体曲面支持和 TIN 支持。

Enhanced: 3.7.0 support for GML ArcString, curved Ring, and CurvePolygon geometries was introduced.

✓ 函数支持 3d 并且不会丢失 z-index。

✓ 函数支持多面体曲面。

✓ 此函数支持三角形和不规则三角网面 (TIN)。

GML 允许混合维度（例如，同一 MultiGeometry 内的 2D 和 3D）。由于 PostGIS 几何形状不能这样做，如果一旦缺失的 Z 维度，ST_GeomFromGML 会将整个几何形状强制为 2D。

GML 支持同一 MultiGeometry 内的混合 SRS。由于 PostGIS 几何形状不能这样做，因此在本例中 ST_GeomFromGML 会将所有子几何形状重新投影到 SRS 根点。如果 GML 根点没有可用的 srsName 属性，函数会抛出错误。

ST_GeomFromGML 函数用于 GML 命名空间并不迂腐。您可以避免在常用用法中明确提及它。但如果您想在 GML 中使用 XLink 功能，需要它。

示例

A single geometry with srsName.

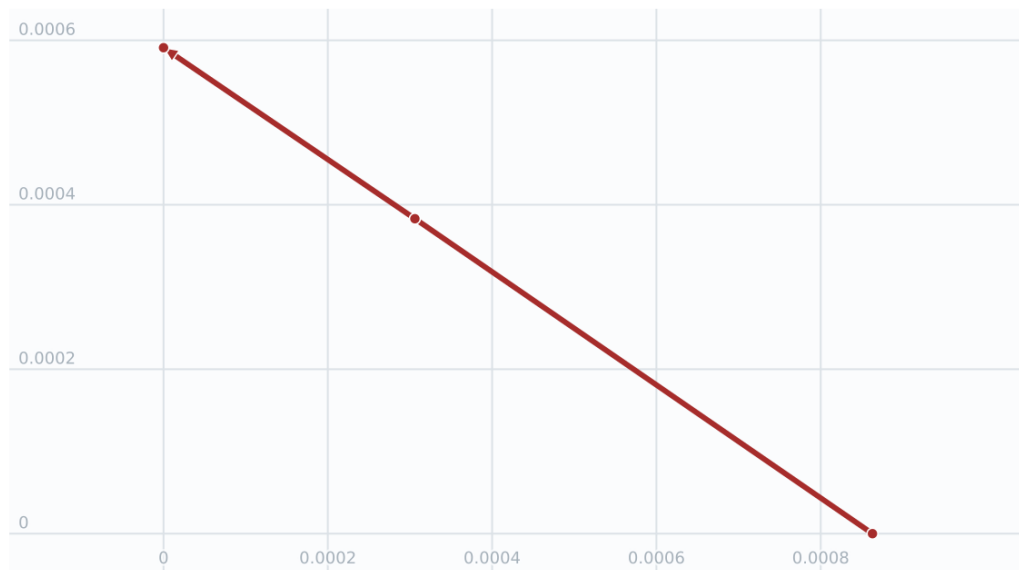
Code

```
SELECT ST_AsEWKT(ST_GeomFromGML($$
  <gml:LineString xmlns:gml="http://www.opengis.net/gml"
    srsName="EPSG:4269">
    <gml:coordinates>
      -71.16028,42.258729 -71.160837,42.259112 -71.161143,42.25932
    </gml:coordinates>
  </gml:LineString>
$$));
```

Output

```
SRID=4269;LINESTRING(-71.16028 42.258729,-71.160837 42.259112,-71.161143 42.25932)
```

Figure



■ Output

XLink usage.

Code

```
SELECT ST_AsEWKT(ST_GeomFromGML($$
  <gml:LineString xmlns:gml="http://www.opengis.net/gml"
    xmlns:xlink="http://www.w3.org/1999/xlink"
    srsName="EPSG:4269">
    <gml:pointProperty>
      <gml:Point gml:id="p1"
        ><gml:pos
        >-71.16028 42.258729</gml:pos
        </gml:Point>
      </gml:pointProperty>
      <gml:pos
        >-71.160837 42.259112</gml:pos>
      <gml:pos
        >-71.161143 42.258729</gml:pos>
      <gml:pointProperty>
        <gml:Point xlink:type="simple" xlink:href="#p1"/>
      </gml:pointProperty>
    </gml:LineString>
  $$));
```

Output

```
SRID=4269;LINESTRING(-71.16028 42.258729,-71.160837 42.259112,-71.161143 42.258729,-71.16028 42.258729)
```

Figure



Polyhedral Surface.

Code

```
SELECT ST_AsEWKT(ST_GeomFromGML('
<gml:PolyhedralSurface xmlns:gml="http://www.opengis.net/gml">
<gml:polygonPatches>
  <gml:PolygonPatch>
    <gml:exterior>
      <gml:LinearRing
><gml:posList srsDimension="3"
>0 0 0 0 0 1 0 1 1 0 1 0 0 0 0 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 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 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 0</gml:posList
></gml:LinearRing>
```

```

    </gml:exterior>
  </gml:PolygonPatch>
  <gml:PolygonPatch>
    <gml:exterior>
      <gml:LinearRing
><gml:posList srsDimension="3"
>0 1 0 0 1 1 1 1 1 1 0 0 1 0</gml:posList
</gml:LinearRing>
    </gml:exterior>
  </gml:PolygonPatch>
  <gml:PolygonPatch>
    <gml:exterior>
      <gml:LinearRing
><gml:posList srsDimension="3"
>0 0 1 1 0 1 1 1 1 0 1 1 0 0 1</gml:posList
</gml:LinearRing>
    </gml:exterior>
  </gml:PolygonPatch>
</gml:polygonPatches>
</gml:PolyhedralSurface
>'))';

```

Output

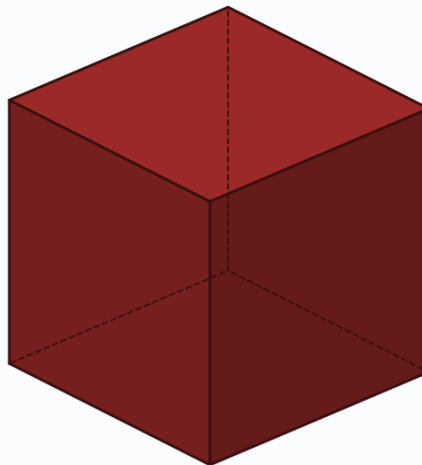
```

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 1,1 0 1 1,0 0 1)))

```

Figure

3D view



■ Output

相关信息

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

7.8.3.4 ST_GeomFromGeoJSON

ST_GeomFromGeoJSON — 将几何图形的 geojson 表示形式作输入并输出 PostGIS 几何对象

Synopsis

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

描述

从 GeoJSON 制表生成 PostGIS 几何对象。

ST_GeomFromGeoJSON 适用于 JSON 的几何部分。如果使用整个 JSON 文档，会抛出错误。

增：3.0.0 如果未指定其他 SRID 解析几何，默认 SRID 为 4326。

增：2.5.0 在可以接受 json 和 jsonb 作输入。

可用性：2.0.0 需要 JSON-C 0.9 或更高版本



Note

如果您没有用 JSON-C，支持您将收到错误通知而不是看到输出。要用 JSON-C，运行 configure --with-jsondir=/path/to/json-c。有关更多信息，参看 Section 2.2.3。



该函数支持 3d 并且不会丢失 z-index。

示例

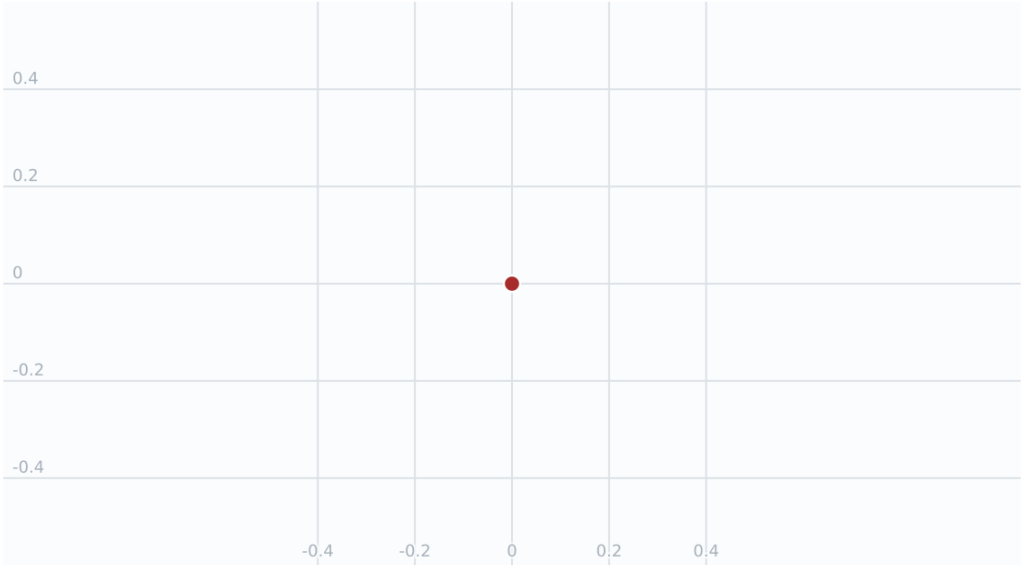
Code

```
SELECT ST_GeomFromGeoJSON('{"type":"Point","coordinates":[-48.23456,20.12345]}') AS wkt;
```

Output

```
POINT(-48.23456 20.12345)
```

Figure



Output

This example creates a 3D LineString.

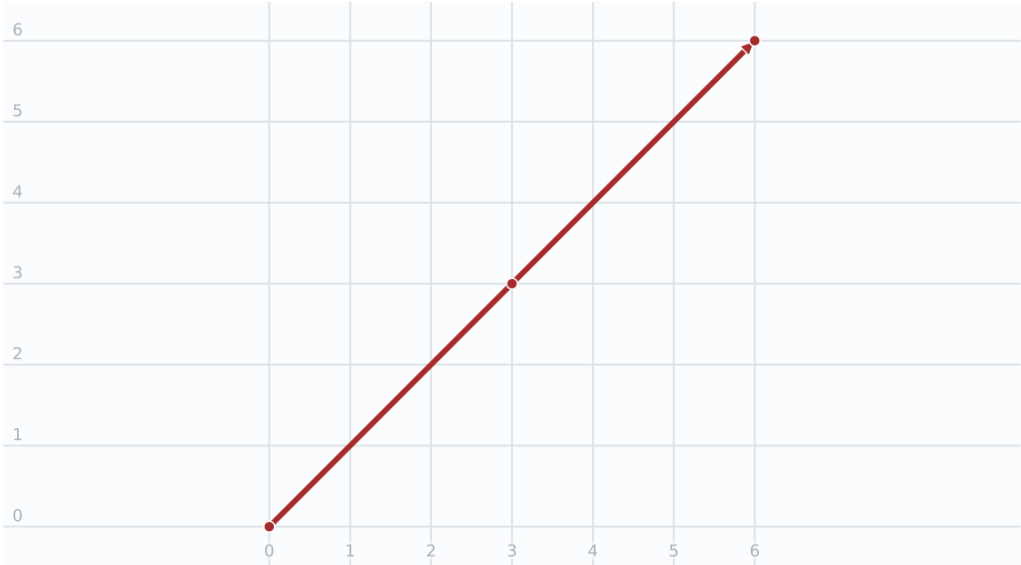
Code

```
SELECT ST_GeomFromGeoJSON('{"type":"LineString","coordinates":[[1,2,3],[4,5,6],[7,8,9]]}') AS wkt;
```

Output

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

Figure



Output

相关信息

[ST_AsText](#), [ST_AsGeoJSON](#), [Section 2.2.3](#)

7.8.3.5 ST_GeomFromKML

ST_GeomFromKML — 将几何图形的 KML 表示形式作输入并输出 PostGIS 几何对象

Synopsis

geometry **ST_GeomFromKML**(text geomkml);

描述

Constructs a PostGIS geometry from an OGC KML representation.

ST_GeomFromKML 适用于 KML 几何片段。如果您在整个 KML 文档上使用它，它会引发错误。

OGC KML 的相应版本包括：

- KML 2.2.0 命名空间

有关 OGC KML 规范，请参考<http://www.opengeospatial.org/standards/kml>：

可用性：1.5，需要 libxml2 2.6+



该函数支持 3d 并且不会丢失 z-index。



Note

ST_GeomFromKML does not support SQL/MM curve geometries.

示例

A single LineString geometry.

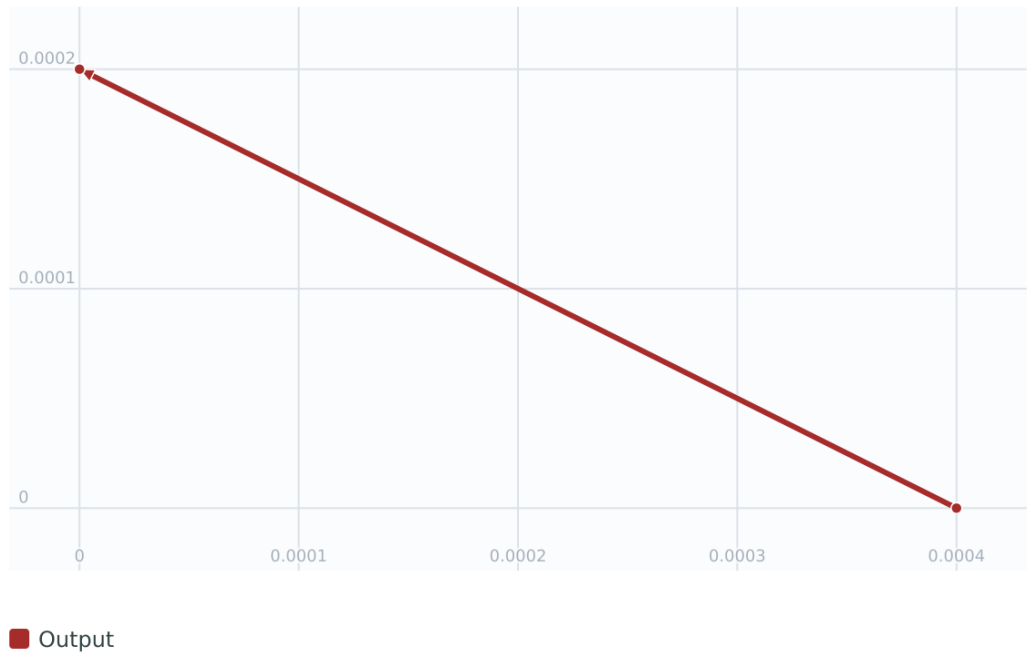
Code

```
SELECT ST_AsEWKT(ST_GeomFromKML($$
  <LineString>
    <coordinates>
      -71.1663, 42.2614
      -71.1667, 42.2616</coordinates>
    </LineString>
  $$));
```

Output

```
SRID=4326;LINESTRING(-71.1663 42.2614,-71.1667 42.2616)
```

Figure



相关信息

Section 2.2.3, [ST_AsKML](#)

7.8.3.6 ST_GeomFromTWKB

ST_GeomFromTWKB — 从 TWKB (“微小的已知二进制制”) 几何表示构建几何对象。

Synopsis

geometry **ST_GeomFromTWKB**(bytea twkb);

描述

ST_GeomFromTWKB 采用 TWKB (“微小的已知二进制制”) 几何表示形式，并构建相应几何类型的对象。

示例

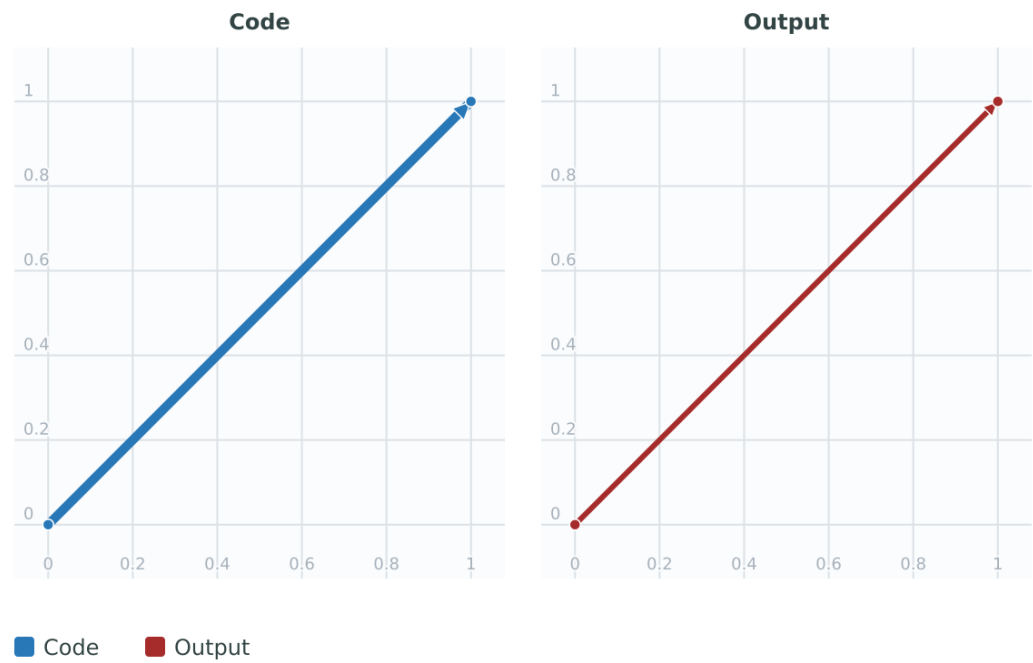
Code

```
SELECT ST_GeomFromTWKB(ST_AsTWKB('LINESTRING(126 34,127 35)::geometry));
```

Output

```
LINESTRING(126 34, 127 35)
```


Figure



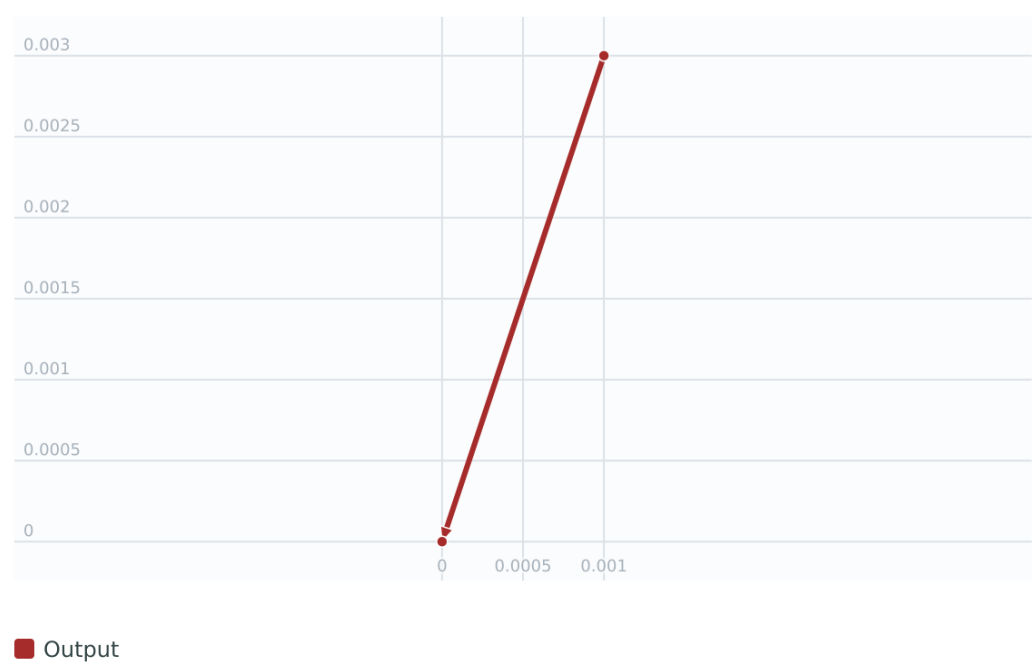
Code

```
SELECT ST_GeomFromTWKB(E'\\x620002f7f40dbce4040105');
```

Output

```
LINESTRING(-113.98 39.198,-113.981 39.195)
```

Figure



相关信息

[ST_AsTWKB](#)

7.8.3.7 ST_GMLToSQL

ST_GMLToSQL — Returns a geometry from GML. Alias for ST_GeomFromGML.

Synopsis

```
geometry ST_GMLToSQL(text geomgml);  
geometry ST_GMLToSQL(text geomgml, integer srid);
```

描述

This function is an alias for [ST_GeomFromGML](#) and provides the same signatures.



☐方法☐了 SQL/MM ☐范。SQL-MM 3: 5.1.50.

可用性：需要 1.5 libxml2 1.6+

增☐功能：2.0.0 支持多面体曲面和 TIN。

增☐：2.0.0 引入了多面体曲面支持和 TIN 支持。

Enhanced: 3.7.0 support for GML ArcString, curved Ring, and CurvePolygon geometries was introduced.

相关信息

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

7.8.3.8 ST_LineFromEncodedPolyline

ST_LineFromEncodedPolyline — 从☐☐折☐☐建 LineString。

Synopsis

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

描述

从☐☐多段☐字符串☐建 LineString。

可☐ precision 指定☐☐折☐中保留多少个小数位。☐☐和解☐☐的☐☐☐相同，否☐坐☐将不正确。

参照 <http://developers.google.com/maps/documentation/utilities/polylinealgorithm>

可用性：2.2.0

示例

This example creates a LineString from a polyline.

Code

```
SELECT ST_LineFromEncodedPolyline('_p~iF~ps|U_ulLnnqC_mqNvxq`@');
```

Output

```
SRID=4326;LINESTRING(-120.2 38.5,-120.95 40.7,-126.453 43.252)
```

Figure

■ Output

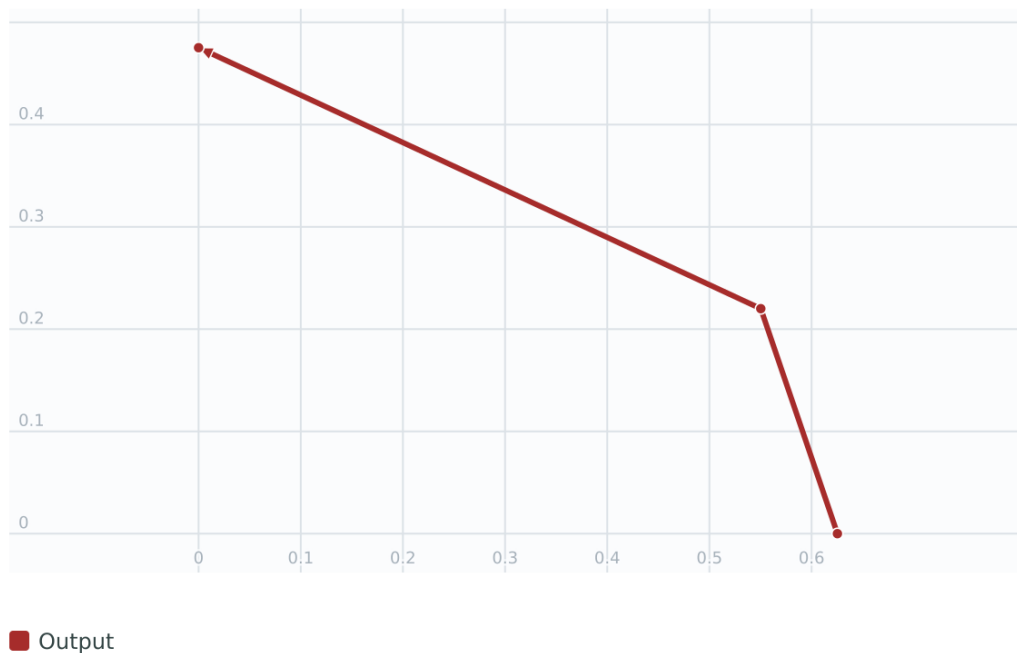
This example selects a different precision than the one used for polyline encoding.

Code

```
SELECT ST_LineFromEncodedPolyline('_p~iF~ps|U_ulLnnqC_mqNvxq`@', 6);
```

Output

```
SRID=4326;LINESTRING(-12.02 3.85,-12.095 4.07,-12.6453 4.3252)
```

Figure

相关信息

[ST_AsEncodedPolyline](#)**7.8.3.9 ST_PointFromGeoHash**

ST_PointFromGeoHash — 从 GeoHash 字符串返回一个点。

Synopsis

```
point ST_PointFromGeoHash(text geohash, integer precision=full_precision_of_geohash);
```

描述

从 GeoHash 字符串返回一个点。☒点代表 GeoHash 的中心点。

如果未指定 `precision`, ST_PointFromGeoHash 返回基于☒入 GeoHash 字符串的完整精度的点。

如果指定 `precision`, ST_PointFromGeoHash 将使用 GeoHash 中的多个字符来☒建点。

可用性 : 2.1.0

示例

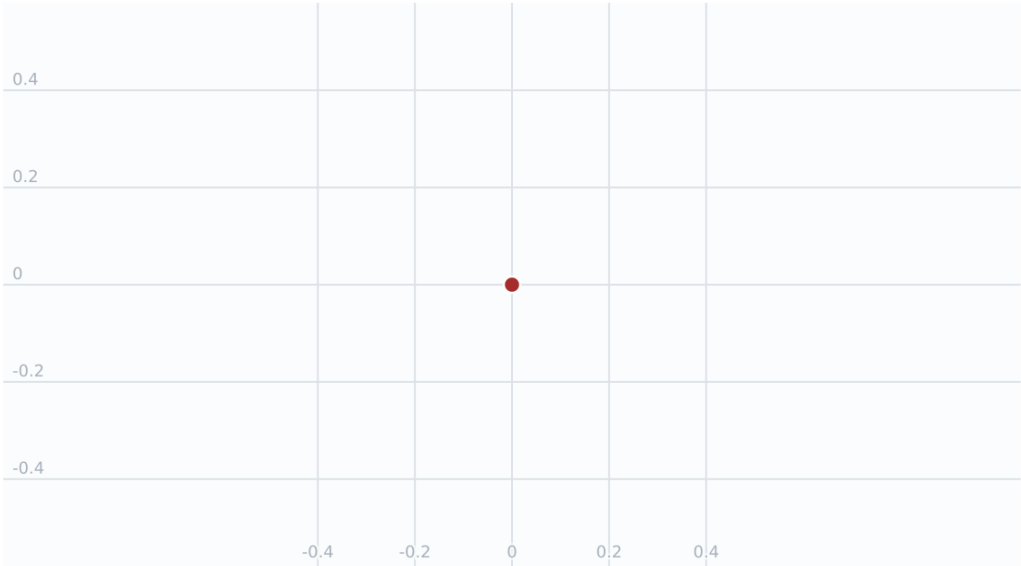
Code

```
SELECT ST_PointFromGeoHash('9qqj7nmxcggy4d0dbxqz0');
```

Output

POINT(-115.172816 36.114646)

Figure



Output

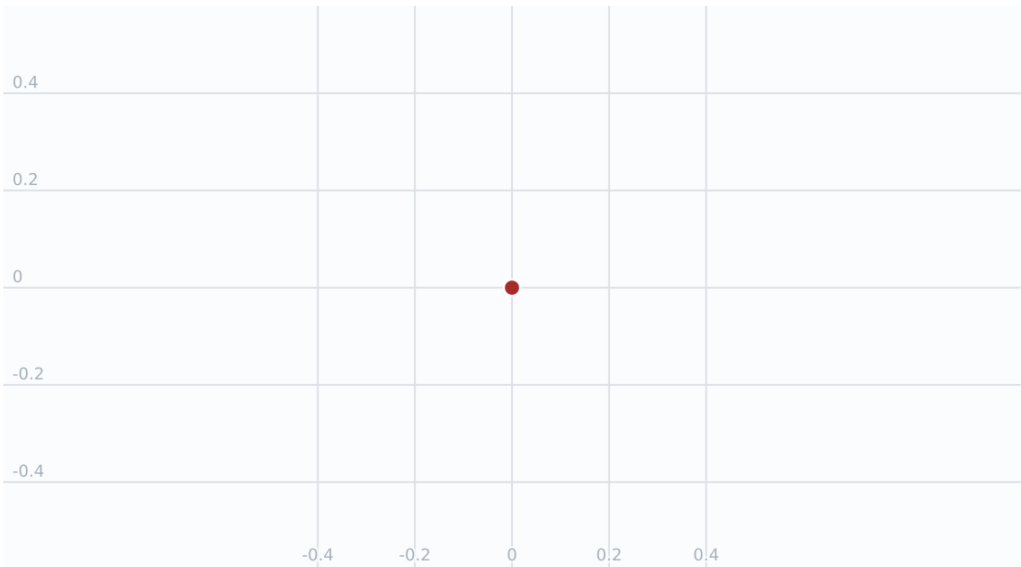
Code

```
SELECT ST_PointFromGeoHash('9qqj7nmxncgyy4d0dbxqz0', 4);
```

Output

POINT(-115.13671875 36.123046875)

Figure



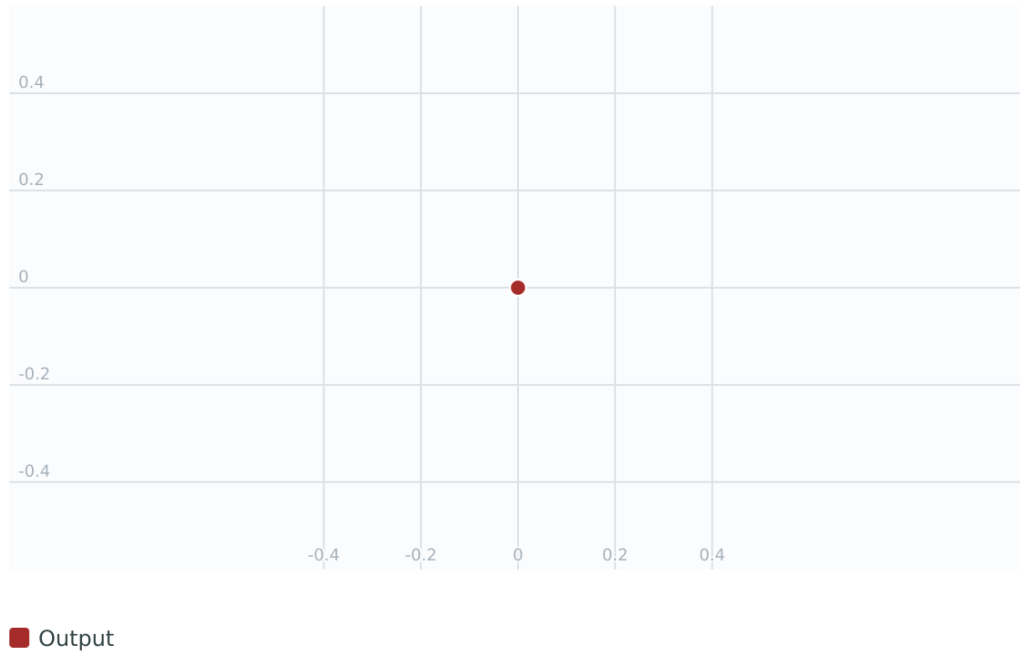
Output

Code

```
SELECT ST_PointFromGeoHash('9qqj7nmxncgyy4d0dbxqz0', 10);
```

Output

```
POINT(-115.172815918922 36.1146435141563)
```

Figure**相关信息**

[ST_GeoHash](#), [ST_Box2dFromGeoHash](#), [ST_GeomFromGeoHash](#)

7.8.3.10 ST_FromFlatGeobufToTable

`ST_FromFlatGeobufToTable` — 根据 FlatGeobuf 数据的 `schema` 建一个表。

Synopsis

```
void ST_FromFlatGeobufToTable(text schemaname, text tablename, bytea FlatGeobuf input data);
```

描述

根据 FlatGeobuf 数据的 `schema` 建一个表。 (<http://flatgeobuf.org>)。

`schema` 架构名称。

`table` 表名。

`data` 输入 FlatGeobuf 数据。

可用性 : 3.2.0

7.8.3.11 ST_FromFlatGeobuf

ST_FromFlatGeobuf — 从 FlatGeobuf 数据。

Synopsis

setof anyelement **ST_FromFlatGeobuf**(anyelement Table reference, bytea FlatGeobuf input data);

描述

从 FlatGeobuf 数据 (<http://flatgeobuf.org>)。注意：PostgreSQL bytea 型不能超过 1GB。

tabletype 表型的引用。

data 入 FlatGeobuf 数据。

可用性：3.2.0

7.9 几何输出

7.9.1 已知文本 (WKT)

7.9.1.1 ST_AsEWKT

ST_AsEWKT — 使用 SRID 元数据返回几何形的已知文本 (WKT) 表示形式。

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);
```

描述

返回以 SRID 前缀的几何形的已知的文本表示形式 (WKT)。可选项的 *maxdecimaldigits* 参数可用于减少输出中使用的浮点后的最大十进制位数 (默认为 15)。

要行 EWKT 表示形式到 PostGIS 几何形的逆过程，使用 **ST_GeomFromEWKT**。



Warning

使用 *maxdecimaldigits* 参数可能会导致几何形无效。为了避免这种情况，首先使用 **ST_ReducePrecision** 和合适的网格大小。



Note

WKT 范不包括 SRID。要取 OGC WKT 格式，使用 **ST_AsText**。



Warning

WKT 格式不保持精度，因此⊠了防止浮⊠截断，⊠使用ST_AsBinary 或 ST_AsEWKB 格式⊠行⊠⊠。

增☐：3.1.0 支持可☐精度参数。

增☒：2.0.0 引入了☒地理、多面体曲面、三角形和 TIN 的支持。



☑函数支持 3d 并且不会☑失 z-index。



此方法支持☐形字符串和曲☐。



☒函数支持多面体曲面。



此函数支持三角形和不~~三角~~三角网面 (TIN)。

示例

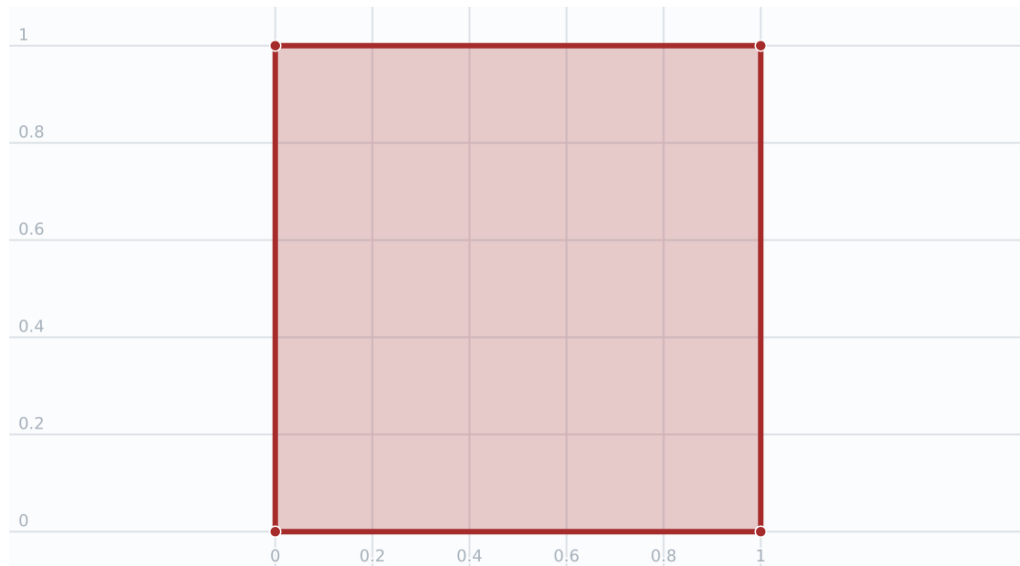
Code

[illegible]

Output

```
SRID=4326;POLYGON((0 0,0 1,1 1,1 0,0 0))
```

Figure



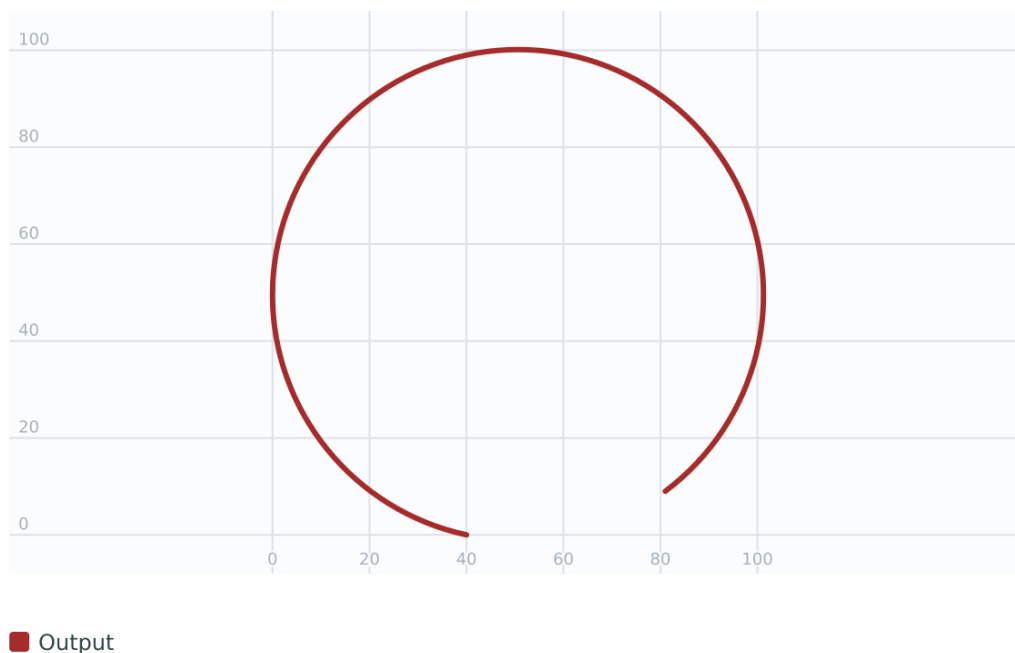
■ Output

Code

```
SELECT ST_AsEWKT('0108000080030000000000000060 ↵
E30A4100000000785C0241000000000000F03F0000000018E20A4100000000485F0241000000000000400000000018
');
```

Output

```
CIRCULARSTRING(220268 150415 1,220227 150505 2,220227 150406 3)
```

Figure**相关信息**

[ST_AsBinary](#), [ST_AsEWKB](#), [ST_AsText](#), [ST_GeomFromEWKT](#)

7.9.1.2 ST_AsText

ST_AsText — 返回不 ☐ SRID 元数据的几何/地理的已知文本 (WKT) 表示形式。

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);
```

描述

返回几何/地理的 OGC **已知文本** (WKT) 表示形式。可 ☐ 的 *maxdecimaldigits* 参数可用于限制 ☐ 出坐 ☐ 中小数点后的位数（默 ☐ 15）。

要 ☐ 行 WKT 表示形式到 PostGIS 几何 ☐ 形的逆 ☐ ☐，☐ 使用 [ST_GeomFromText](#)。



Note

☒ 准 OGC WKT 表示不包括 SRID。要将 SRID 作☒☒出表示的一部分，☒使用非☒准 PostGIS 函数 **ST_AsEWKT**



Warning

WKT 中数字的文本表示可能无法保持完整的浮点精度。☒了确保数据存☒或☒☒的完全准确性，最好使用已知的二☒制 (WKB) 格式 (☒☒☒ **ST_AsBinary** 和 *maxdecimaldigits*)。



Warning

使用 `maxdecimaldigits` 参数可能会导致几何图形无效。为了避免这种情况，首先使用 **ST_ReducePrecision** 和合适的网格大小。

可用性：1.5 - 引入了地理支持。

增☐：2.5 - 引入了可☐的精度参数。



此方法扩展了 SQL 1.1 的 OGC 功能规范。 s2.1.1.1



方法了 SQL/MM 范。SQL-MM 3: 5.1.25



此方法支持☐形字符串和曲☐。

示例

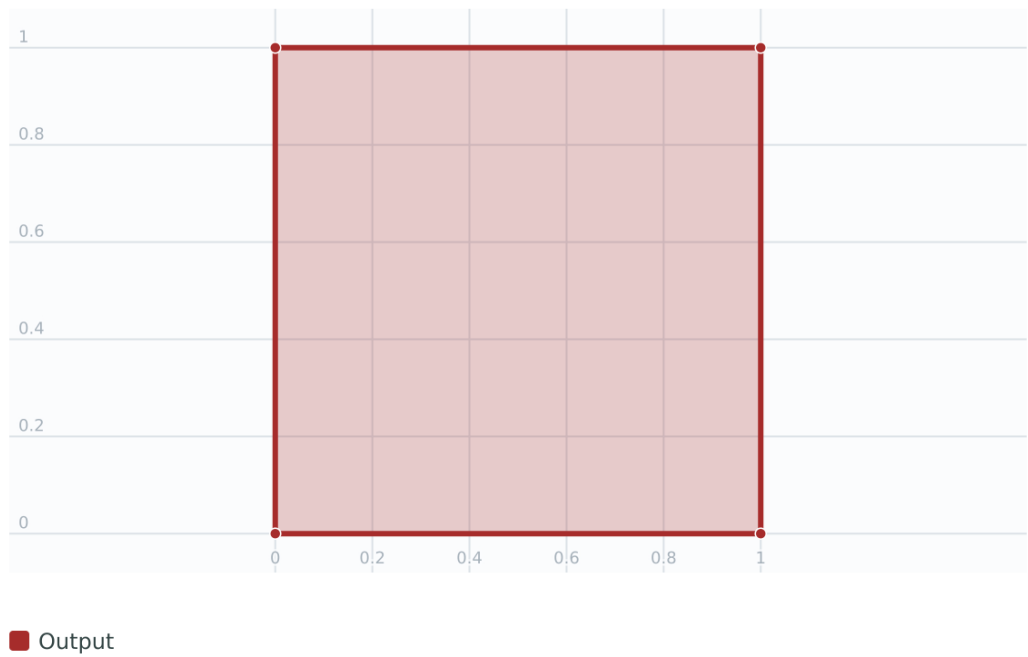
Code

[illegible]

Output

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

Figure



默认情况下输出全精度。

Code

```
SELECT ST_AsText('POINT(111.111111 1.111111)::geometry');
```

Output

```
POINT(111.111111 1.111111)
```

maxdecimaldigits 参数用于限制输出精度。

Code

```
SELECT ST_AsText('POINT(111.111111 1.111111)::geometry, 2);
```

Output

```
POINT(111.11 1.11)
```

相关信息

[ST_AsBinary](#), [ST_AsEWKB](#), [ST_AsEWKT](#), [ST_GeomFromText](#)

7.9.2 已知的二进制制文件 (WKB)

7.9.2.1 ST_AsBinary

ST_AsBinary — 返回不带 SRID 元数据的几何/地理的 OGC/ISO 已知的二进制制 (WKB) 表示形式。

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);
```

描述

返回几何对象的 OGC/ISO **Well-Known Binary** (WKB) 表示形式。第一个函数形式默认为使用服务器主机字节序行输出。第二个函数形式接受一个文本参数，用于指定字节序方式：'NDR' 表示小端字节，'XDR' 表示大端字节。如输入未知参数，将默认生成小端字节输出。

WKB 格式由于从数据中提取几何数据并保持完整的数值精度非常有用。它避免了 WKT 等文本格式可能产生的精确舍入。

要行 WKB 到 PostGIS 几何形的逆过程，请使用 **ST_GeomFromWKB**。



Note

OGC/ISO WKB 格式不包括 SRID。要提取包含 SRID 的 EWKB 格式，请使用 **ST_AsEWKB**



Note

PostgreSQL 默认以十六进制输出 bytea。如果你的 GUI 工具仍然需要旧的输出方式，可以在数据行中执行：SET bytea_output='escape'。

增强功能：引入了 2.0.0 多面体曲面、三角形和三角网的支持。

增强：2.0.0 支持更高坐度。

增强：2.0.0 支持地理中的字节序。

可用性：1.5.0 支持地理位置。

Changed: 2.0.0 Inputs to this function can not be unknown -- must be geometry. Constructs such as ST_AsBinary('POINT(1 2)') are no longer valid and you will get an n ST_AsBinary(unknown) is not unique error. Code like that needs to be changed to ST_AsBinary('POINT(1 2)::geometry');. If that is not possible, then install legacy.sql.

- ✓ 此方法实现了 **SQL 1.1 的 OGC 功能规范**。s2.1.1.1
- ✓ 此方法实现了 **SQL/MM 规范**。SQL-MM 3: 5.1.37
- ✓ 此方法支持几何字符串和曲线。
- ✓ 此函数支持多面体曲面。
- ✓ 此函数支持三角形和不规则三角网面 (TIN)。
- ✓ 此函数支持 3d 并且不会丢失 z-index。

示例



Note

要☑取 OGC/ISO WKB 格式，☑使用 **ST_AsBinary**。☑注意，OGC/ISO WKB 格式不包括 SRID。

增☑功能：引入了 2.0.0 ☑多面体曲面、三角形和三角网的支持。



☑函数支持 3d 并且不会☑失 z-index。



此方法支持☐形字符串和曲☐。



☒函数支持多面体曲面。



此函数支持三角形和不▢▢三角网面 (TIN)。

示例

Code

```
SELECT ST_AsEWKB(ST_GeomFromText('POLYGON((0 0,0 1,1 1,1 0,0 0))', 4326));
```

Output

[illegible]

Code

```
SELECT ST_AsEWKB(ST_GeomFromText('POLYGON((0 0,0 1,1 1,1 0,0 0))', 4326), 'XDR');
```

Output

[illegible]

相关信息

ST AsBinary, ST_GeomFromEWKB, ST_SRID

7.9.2.3 ST_AsHEXEWKB

ST_AsHEXEWKB — 使用小端 (NDR) 或大端 (XDR) 返回 HEXEWKB 格式 (作文本) 的几何形。

Synopsis

```
text ST_AsHEXEWKB(geometry g1, text NDRorXDR);
text ST_AsHEXEWKB(geometry g1);
```

描述

使用小端 (NDR) 或大端 (XDR) 返回 HEXEWKB 格式 (作文本) 的几何形。如果未指定, 使用 NDR。

可用性 : 1.2.2

✓ 函数支持 3d 并且不会失 z-index。

✓ 此方法支持形字符串和曲线。

示例

Code

```
SELECT ST_AsHEXEWKB(ST_GeomFromText('POLYGON((0 0,0 1,1 1,1 0,0 0))', 4326));
```

Output

```
0103000020E6100000010000000500
000000000000000000000000000000
000000000000000000000000000000F03F
000000000000000000000000000000F03F
000000000000000000000000000000F03F
0000000000000000000000000000000000
```

Which gives the same answer as:

Code

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

7.9.3 其它格式

7.9.3.1 ST_AsEncodedPolyline

ST_AsEncodedPolyline — 从 LineString 几何体返回折。

Synopsis

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

描述

Returns the geometry in the **Encoded Polyline Algorithm Format**. The format stores latitude and longitude deltas as integers after multiplying coordinates by 10 raised to the requested precision.

The informal names *polyline5* and *polyline6* refer to the same encoding with precision 5 and 6. The precision must match when encoding and decoding; otherwise the coordinates are scaled incorrectly.

The input geometry must use SRID 4326, with longitude as X and latitude as Y. Use **ST_Transform** to transform geometries in other coordinate systems before encoding.

The **OSRM API** returns precision 5 for geometries=polyline and precision 6 for geometries=polyline6. Decode the resulting route.geometry with **ST_LineFromEncodedPolyline** using the matching precision.

可用性 : 2.2.0

示例

Encode the same WGS 84 LineString using precision 5 and 6.

Code

```
WITH route AS (
  SELECT ST_GeomFromEWKT(
    'SRID=4326;LINESTRING(-120.2 38.5,-120.95 40.7,-126.453 43.252)'
  ) AS geom
)
SELECT ST_AsEncodedPolyline(geom, 5) AS polyline5,
       ST_AsEncodedPolyline(geom, 6) AS polyline6
FROM route;
```

Output

polyline5	polyline6
_p~iF~ps U_uLlnqC_mqNvxq`@	_izlhA~rlgdF_{geC~ywl@_kwzCn`{nI

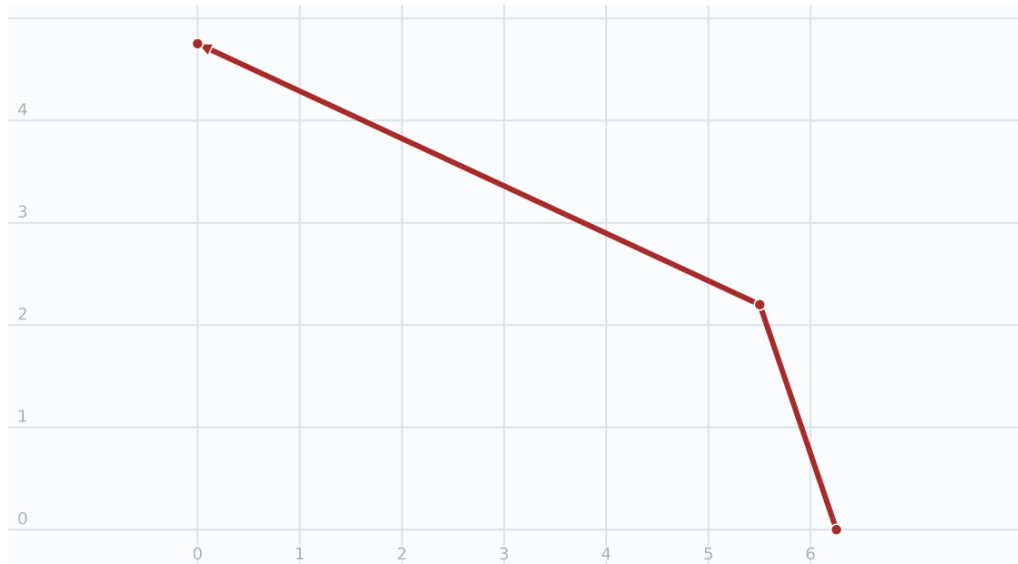
Decode an OSRM route.geometry returned with geometries=polyline6.

Code

```
SELECT ST_AsEWKT(ST_LineFromEncodedPolyline(
  '_izlhA~rlgdF_{geC~ywl@_kwzCn`{nI',
  6
)) AS route;
```

Output

```
SRID=4326;LINESTRING(-120.2 38.5,-120.95 40.7,-126.453 43.252)
```

Figure

■ Output

The Google Maps JavaScript Geometry library decodes precision-5 strings. This browser example assumes that map is an initialized Google map.

I Code

```

<script src="https://maps.googleapis.com/maps/api/js?key=YOUR_API_KEY&libraries=geometry"
></script>
<script>
  const encodedPath = "_p~iF~ps|U_uLLnnqC_mqNvxq`@";
  const flightPath = new google.maps.Polyline({
    path: google.maps.geometry.encoding.decodePath(encodedPath),
    map: map,
    strokeColor: "#0000CC",
    strokeOpacity: 1.0,
    strokeWeight: 4
  });
</script>

```

相关信息

ST_LineFromEncodedPolyline, ST_Transform

7.9.3.2 ST_AsFlatGeobuf

ST_AsFlatGeobuf — 返回一行的 FlatGeobuf 表示形式。

Synopsis

```

bytea ST_AsFlatGeobuf(anyelement set row);
bytea ST_AsFlatGeobuf(anyelement row, bool index);
bytea ST_AsFlatGeobuf(anyelement row, bool index, text geom_name);

```

描述

返回与 FeatureCollection 的一行的 FlatGeobuf 表示形式 (<http://flatgeobuf.org>)。注意：PostgreSQL bytea 不能超 1GB。

row 至少包含一个几何列的行数据。

index 控制空索引生成。默认为 FALSE。

geom_name 行数据中几何列的列名。如果 NULL，它是找到的第一个几何列。

可用性：3.2.0

7.9.3.3 ST_AsGeobuf

ST_AsGeobuf — 返回一行的 Geobuf 表示。

Synopsis

```

bytea ST_AsGeobuf(anyelement set row);
bytea ST_AsGeobuf(anyelement row, text geom_name);

```

描述

返回与 `FeatureCollection` 兼容的一行的 `Geobuf` 表示 (<https://github.com/mapbox/geobuf>)。分析每个输入几何形状以确定最佳存储的最大精度。注意，当前形式的 `Geobuf` 无法行流式输出，因此完整的输出将在内存中包装。

`row` 至少包含一个几何列的行数据。

`geom_name` 行数据中几何列的列名。如果 `NULL`，它是找到的第一个几何列。

可用性：2.4.0

示例

Code

```
SELECT encode(ST_AsGeobuf(q, 'geom'), 'base64')
FROM (SELECT 'POLYGON((0 0,0 1,1 1,1 0,0 0))'::geometry AS geom) AS q;
```

Output

```
GAAiEAo0CgwIBBoIAAAAAgIAAAE=
```

7.9.3.4 ST_AsGeoJSON

`ST_AsGeoJSON` — 以 `GeoJSON` 格式返回一个几何体或要素。

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);
```

描述

返回一个几何体作为 `GeoJSON` 中的 “geometry” 对象，或者返回一行作为 `GeoJSON` 中的 “feature” 对象。

生成的 `GeoJSON` 几何体和要素遵循 [GeoJSON 规范 RFC 7946](#)，但当解析的几何体使用 WGS84 经度和纬度之外的 CRS（如 [EPSG:4326](#)，[urn:ogc:def:crs:OGC::CRS84](#)）进行引用时，`GeoJSON` 几何对象将默认附加一个短的 CRS SRID 标识符。支持 2D 和 3D 几何。`GeoJSON` 支持 SFS 1.1 几何类型（例如，不支持曲线）。

`geom_column` 参数用于区分多个几何列。如果省略该参数，将确定输出中的第一个几何列。相反，该参数将省略列类型查找。

`maxdecimaldigits` 参数可用于减少输出中使用的最大小数位数（默认为 9）。如果您使用 `EPSG:4326` 并且输出几何形状用于显示，将 `maxdecimaldigits=6` 用于多边形来避免可能是一个无效的圆。



Warning

使用 `maxdecimaldigits` 参数可能会导致输出几何形状无效。为了避免这种情况，首先使用 `ST_ReducePrecision` 和合适的网格大小。

`options` 参数可用于在 `GeoJSON` 输出中添加 BBOX 或 CRS：

- 0 : 表示没有 `id`
- 1 : GeoJSON BBOX
- 2 : GeoJSON 短 CRS (例如 EPSG : 4326)
- 4 : GeoJSON 长 CRS (例如 urn:ogc:def:crs:EPSG::4326)
- 8 : GeoJSON 短 CRS, 如果不是 EPSG : 4326 (默认)

`id_column` 参数用于设置返回的 GeoJSON 要素的“id”属性。根据 GeoJSON RFC, 属性在要素具有常用属性符 (例如主键) 时使用此参数。当未指定时, 生成的要素将不会有“id”属性, 除了几何信息之外的任何列, 包括任何可能的属性, 都将最位于要素的“properties”属性内。

当输入中包含重复的列名时, 生成的“properties”对象也会输出重复的属性。在这种情况下, `ST_AsGeoJSON` 会发出警告, 因为 PostgreSQL 的 jsonb 对于重复属性只会保留最后一个。

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 可以在 geojson.io 和 geojsonlint.com 上在线查看和验证。它受到网络地图框架的广泛支持：

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

可用性：1.3.4

可用性：1.5.0 支持地理位置。

更改：2.0.0 支持默认参数和命名参数。

更改：3.0.0 支持属性作输入

更改：3.0.0 输出 SRID (如果不是 EPSG : 4326)。

更改：3.5.0 允许指定包含要素 ID 的列

Changed: 3.7.0 -- 增加了关于重复属性的警告



函数支持 3d 并且不会丢失 z-index。

示例

生成特征集合：

Code

```
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);
```

Output

```
{ "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" } } ] }
```

生成一个特征：

Code

```
SELECT ST_AsGeoJSON(t.*, id_column =
> 'id')
FROM (VALUES (1, 'one', 'POINT(1 1)::geometry)) AS t(id, name, geom);
```

Output

```
{ "type": "Feature", "geometry": { "type": "Point", "coordinates": [1,1] }, "id": 1, "properties": { "name": "one" } }
```

不要忘记将数据从 WGS84 度、度以符合 GeoJSON 范：

Code

```
SELECT ST_AsGeoJSON(ST_Transform(geom, 4326)) FROM fe_edges LIMIT 1;
```

Output

```
{ "type": "MultiLineString", "coordinates": [ [ [ -89.734634999999997, 31.492072000000000 ], [ -89.734955999999997, 31.492237999999997 ] ] ] }
```

支持 3D 几何形状：

Code

```
SELECT ST_AsGeoJSON('LINESTRING(1 2 3,4 5 6)');
```

Output

```
{ "type": "LineString", "coordinates": [ [ 1, 2, 3 ], [ 4, 5, 6 ] ] }
```

Options 参数用于在 GeoJSON 出中添加 BBOX 或 CRS：

Code

```
SELECT ST_AsGeoJSON(ST_SetSRID('POINT(1 1)::geometry', 4326), 9, 4|1);
```

Output

```
{ "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 ] }
```

相关信息

[ST_GeomFromGeoJSON](#), [ST_ForcePolygonCCW](#), [ST_Transform](#)

7.9.3.5 ST_AsGML

ST_AsGML — 将几何形作 GML 版本 2 或 3 元素返回。

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);
```

描述

将几何图形作为地理语言 (GML) 元素返回。版本参数（如果指定）可以是 2 或 3。如果未指定版本参数，默认为 2。`maxdecimaldigits` 参数可用于减少输出中使用的最大小数位数（默认为 15）。



Warning

使用 `maxdecimaldigits` 参数可能会导致输出几何图形无效。为了避免这种情况，首先使用 `ST_ReducePrecision` 和合适的网格大小。

GML 2 指 2.1.2 版本，GML 3 指 3.1.1 版本

“`options`” 参数是一个位字段。它可用于在 GML 输出中定义 CRS 输出类型，并将数据声明为度/度：

- 0 : GML 短 CRS（例如 EPSG:4326），默认。
- 1 : GML 长 CRS（例如 urn:ogc:def:crs:EPSG::4326）
- 2 : 用于 GML 3，从输出中删除 `srsDimension` 属性。
- 4 : 用于 GML 3，使用 `<LineString>` 而不是 `<Curve>` 输出。
- 16 : 声明数据为度/度（例如 `srid=4326`）。默认情况下假设数据是平面的。此选项适用于与顺序相关的 GML 3.1.1 输出。因此，如果您设置它，它将交换坐标，因此顺序是 `lat lon` 而不是数据 `lon lat`。
- 32 : 输出几何体的框（最小外接矩形）。

“命名空间前缀” 参数可用于指定自定义命名空间前缀或无前缀（如果为空）。如果使用 `null` 或省略 `'gml'` 前缀

可用性：1.3.2

可用性：1.5.0 支持地理位置。

增强：2.0.0 引入了前缀支持。引入了 GML3 的选项 4，以允许使用 `LineString` 而不是线条的 `Curve` 输出。引入了多面体曲面和 TINS 的 GML3 支持。引入选项 32 来输出框。

更改：2.0.0 使用默认命名参数

增强：GML 3 引入了 2.1.0 id 支持。



Note

ST_AsGML 版本 3 + 支持多面体曲面和 TINS。

- ✓ 方法遵循了 SQL/MM 规范。SQL-MM IEC 13249-3: 17.2
- ✓ 函数支持 3d 并且不会丢失 z-index。
- ✓ 函数支持多面体曲面。
- ✓ 此函数支持三角形和不规则三角网面 (TIN)。

示例

Version 2.

| Code

```
SELECT ST_AsGML(ST_GeomFromText('POLYGON((0 0,0 1,1 1,1 0,0 0))', 4326));
```

| Output

```
<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
>
```

Version 3.

Flip coordinates and output extended EPSG (16 | 1)

| Code

```
SELECT ST_AsGML(3, ST_GeomFromText('POINT(5.234234233242 6.34534534534)', 4326), 5, 17);
```

| Output

```
<gml:Point srsName="urn:ogc:def:crs:EPSG::4326"
><gml:pos srsDimension="2"
>6.34535 5.23423</gml:pos
></gml:Point
>
```

Output the envelope (32).

| Code

```
SELECT ST_AsGML(3, ST_GeomFromText('LINESTRING(1 2, 3 4, 10 20)',4326), 5, 32);
```

| Output

```
<gml:Envelope srsName="EPSG:4326" srsDimension="2"
><gml:lowerCorner
>1 2</gml:lowerCorner
><gml:upperCorner
>10 20</gml:upperCorner
></gml:Envelope
>
```

Output the envelope (32), reverse (lat lon instead of lon lat) (16), long srs (1) = 32 | 16 | 1 = 49.

| Code

```
SELECT ST_AsGML(3, ST_GeomFromText('LINESTRING(1 2, 3 4, 10 20)',4326), 5, 49);
```

| Output

```
<gml:Envelope srsName="urn:ogc:def:crs:EPSG::4326" srsDimension="2"
><gml:lowerCorner
>2 1</gml:lowerCorner
><gml:upperCorner
>20 10</gml:upperCorner
></gml:Envelope
>
```

This example uses a PolyhedralSurface.

I Code

```
SELECT ST_AsGML(
3,
'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);
```

I Output

```
<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 1 0 0 1 0 0</gml:posList>
      </gml:LinearRing>
    </gml:exterior>
  </gml:PolygonPatch>
  <gml:PolygonPatch>
    <gml:exterior>
      <gml:LinearRing>
        <gml:posList srsDimension="3"
>1 1 0 1 1 1 1 0 1 1 0 0 1 1 0</gml:posList>
      </gml:LinearRing>
    </gml:exterior>
  </gml:PolygonPatch>
  <gml:PolygonPatch>
    <gml:exterior>
      <gml:LinearRing>
        <gml:posList srsDimension="3"
>0 1 0 0 1 1 1 1 1 1 1 0 0 1 0</gml:posList>
      </gml:LinearRing>
    </gml:exterior>
  </gml:PolygonPatch>
  <gml:PolygonPatch>
    <gml:exterior>
      <gml:LinearRing>
        <gml:posList srsDimension="3"
>0 0 1 1 0 1 1 1 1 0 1 1 0 0 1</gml:posList>
      </gml:LinearRing>
    </gml:exterior>
  </gml:PolygonPatch>
</gml:polygonPatches>
</gml:PolyhedralSurface>
>
```

相关信息

ST_GeomFromGML

7.9.3.6 ST_AsKML

ST_AsKML — 将几何图形作 KML 元素返回。

Synopsis

```
text ST_AsKML(geometry geom, integer maxdecimaldigits=15, text nprefix=NULL);  
text ST_AsKML(geography geog, integer maxdecimaldigits=15, text nprefix=NULL);
```

描述

将几何图形作为 Keyhole 标记语言 (KML) 元素返回。默认最大小数位数 15，默认命名空间无前缀。



Warning

使用 `maxdecimaldigits` 参数可能会导致几何图形无效。为了避免这种情况，首先使用 `ST_ReducePrecision` 和合适的网格大小。

可用性：1.2.2 - 包含版本参数的更高版本出现在 1.3.2 中

增加：2.0.0 - 添加前缀命名空间，使用默认和命名参数

更改：3.0.0 - 除了“版本化”几何体名



Note

需要在 Proj 支持下使用 PostGIS。使用 `PostGIS_Full_Version` 确认您已安装支持。



Note

AsKML 输出不适用于没有 SRID 的几何图形



该函数支持 3d 并且不会丢失 z-index。

示例

Code

```
SELECT ST_AsKML(ST_GeomFromText('POLYGON((0 0,0 1,1 1,1 0,0 0))', 4326));
```

Output

```
<Polygon  
><outerBoundaryIs  
><LinearRing  
><coordinates  
>0,0 0,1 1,1 1,0 0,0</coordinates  
></LinearRing  
></outerBoundaryIs  
></Polygon  
>
```

This example outputs a 3D LineString.

Code

```
SELECT ST_AsKML('SRID=4326;LINESTRING(1 2 3,4 5 6)');
```

Output

```
<LineString
><coordinates
>1,2,3 4,5,6</coordinates
></LineString
>
```

相关信息

[ST_AsSVG](#), [ST_AsGML](#)

7.9.3.7 ST_AsLatLonText

ST_AsLatLonText — 返回点的度、分、秒表示形式。

Synopsis

text **ST_AsLatLonText**(geometry pt, text format=');

描述

返回点的度、分和秒表示形式。



Note

假设点位于度/度投影中。X（度）和 Y（度）坐标在输出中标准化“正常”范围（度 -180 到 180，度 -90 到 90）。

text 参数是一个格式字符串，包含结果文本的格式，类似于日期格式字符串。有效标志“D”表示度、“M”表示分、“S”表示秒、“C”表示基本方向 (NSEW)。DMS 令牌可以重复以指示所需的度和精度（“SSS.SSSS”表示“1.0023”）。

“M”、“S”和“C”是可选项的。如果省略“C”，南或西的度数将显示“-”符号。如果省略“S”，分将显示十进制，其精度位数与您指定的位数相同。如果省略“M”，度数将显示十进制，其精度与您指定的位数相同。

如果格式字符串被省略（或零度），将使用默认格式。

可用性：2.0

示例

默认格式。

Code

```
SELECT (ST_AsLatLonText('POINT (-3.2342342 -2.32498)'));
```

| Output

```
2°19'29.928"S 3°14'3.243"W
```

指定格式（与默认相同）。

| Code

```
SELECT (ST_AsLatLonText('POINT (-3.2342342 -2.32498)', 'D°M'S.SSS"C'));
```

| Output

```
2°19'29.928"S 3°14'3.243"W
```

除 D、M、S 和 C 以外的字符通。

| Code

```
SELECT (ST_AsLatLonText('POINT (-3.2342342 -2.32498)', 'D degrees, M minutes, S seconds to the C'));
```

| Output

```
2 degrees, 19 minutes, 30 seconds to the S 3 degrees, 14 minutes, 3 seconds to the W
```

有符号的度数而不是基本方向。

| Code

```
SELECT (ST_AsLatLonText('POINT (-3.2342342 -2.32498)', 'D°M'S.SSS'));
```

| Output

```
-2°19'29.928" -3°14'3.243"
```

十进制度数。

| Code

```
SELECT (ST_AsLatLonText('POINT (-3.2342342 -2.32498)', 'D.DDDD degrees C'));
```

| Output

```
2.3250 degrees S 3.2342 degrees W
```

To convert parsed degree, minute, and second components to decimal degrees, apply the cardinal-direction sign explicitly.

| Code

```
WITH dms(label, degrees, minutes, seconds, hemisphere) AS (
  VALUES
    ('latitude', 43::numeric, 0::numeric, 50.60::numeric, 'S'),
    ('longitude', 147::numeric, 12::numeric, 18.20::numeric, 'E')
)
SELECT label,
       round(
         (CASE WHEN hemisphere IN ('S', 'W') THEN -1 ELSE 1 END)
         * (degrees + minutes / 60 + seconds / 3600),
         9
       ) AS decimal_degrees
FROM dms;
```

Output

label	decimal_degrees
latitude	-43.014055556
longitude	147.205055556

大的被一化。

Code

```
SELECT (ST_AsLatLonText('POINT (-302.2342342 -792.32498)'));
```

Output

72°19'29.928"S 57°45'56.757"E

7.9.3.8 ST_AsMARC21

ST_AsMARC21 — 将几何形返回有地理数据字段 (034) 的 MARC21/XML 。

Synopsis

text **ST_AsMARC21** (geometry geom , text format='hdddmss');

描述

此函数返回一个 MARC21/XML ，其中包含表示定几何形的界框的制数学数据。格式参数允许以 MARC21/XML 准支持的所有格式子字段 \$d、\$e、\$f 和 \$g 中的坐行。有效格式：

- 度和度，度、分、秒（默）：hdddmss
- 度和度的十制度：hddd.ddddd
- 没有度和度的十制度：ddd.ddddd
- 有度和度的小数分：hddmm.mmmm
- 不度和度的小数分：ddmm.mmmm
- 有度和度的小数秒：hddmmss.sss

小数点符号也可以是逗号，例如 hddddmm, mmmm。
小数格式的精度可能受到小数点后字符数的限制，例如 hddmm.mm 表示小数分，精度位小数。
此函数忽略 Z 和 M 度。

支持的 LOC MARC21/XML 版本：

- MARC21/XML 1.1

可用性：3.3.0



Note
此函数不支持非度/度几何形状，因它不受 MARC21/XML 准（制数学数据）支持。



Note
MARC21/XML 准没有提供任何方法来注制数学数据的空参考系，意味着些信息在 MARC21/XML 后将会失。

示例

将 POINT 转换为 MARC21/XML, 格式为 dd mm ss (默认)

Code

```
SELECT ST_AsMARC21('SRID=4326;POINT(-4.504289 54.253312)::geometry');
```

Output

```
----- 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>
```

将 POLYGON 转换为十进制格式的马克21/XML

Code

```
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>
```

将 GEOMETRYCOLLECTION 以十进制分格式表示的 MARC21/XML。MARC21/XML 输出中的几何顺序与其在集合中的顺序相同。

Code

```
SELECT ST_AsMARC21('SRID=4326;GEOMETRYCOLLECTION(POLYGON((13.1 52.65,13.516666666666667 52.65,13.516666666666667 52.38333333333333,13.1 52.38333333333333,13.1 52.65)),POINT(-4.5 54.25))::geometry, 'hdddmm.mmmm');
```

Output

```

                                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>

```

相关信息

[ST_GeomFromMARC21](#)

7.9.3.9 ST_AsMVTGeom

ST_AsMVTGeom — 将几何形状 MVT 瓦片的坐空。

Synopsis

geometry **ST_AsMVTGeom**(geometry geom, box2d bounds, integer extent=4096, integer buffer=256, boolean clip_geom=true);

描述

将几何形状 MVT ([Mapbox 矢量切片](#)) 瓦片的坐空，如果需要，将其剪切到边界。几何形状必须位于目标地的坐系中（如果需要，使用 [ST_Transform](#)）。通常是 [Web 墨卡托](#) (SRID:3857)。

函数保持几何有效性，并在需要时进行修正。可能会导致几何体塌陷到低的度。

The function performs geometry simplification appropriate to the integer tile coordinate space. After transforming to tile coordinates it snaps coordinates to the integer grid, removes repeated points, and removes points that lie on straight lines. It does not perform general-purpose, scale-dependent simplification such as [ST_Simplify](#). To reduce detail before tile conversion, simplify the geometry in map coordinates before calling ST_AsMVTGeom.

必须提供目标地坐空中瓦片的矩形边界，以便可以几何形状，并在需要时进行裁剪。可以使用 [ST_TileEnvelope](#) 生成边界。

[ST_AsMVT](#) 函数用于将几何形状 ST_AsMVT 所需的瓦片坐空。

geom 是目标地坐系中要几何形状。

bounds 是瓦片在地坐空中的矩形边界，没有缓冲区。

`extent` 是由 **MVT 规范** 定义的切片坐空中切片的大小。默认为 4096。

`buffer` 是切片坐空中用于裁剪几何形的缓冲区大小。默认为 256。

`Clip_geom` 是一个布尔值，用于控制几何形是否按原样剪切或否。默认为 `true`。

可用性：2.4.0



Note
从 3.0 开始，可以在配置中使用 `Wagyu` 来修剪和修剪 MVT 多边形。它比 GEOS 默认速度更快，修剪的结果更正确，但它可能会丢失小多边形。

示例

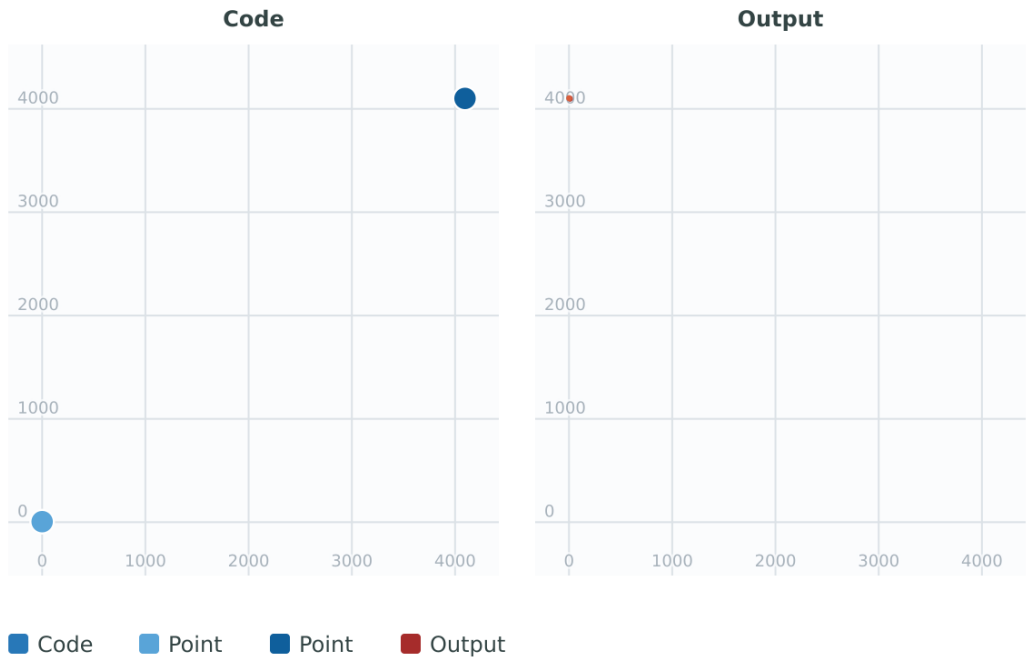
Code

```
SELECT ST_AsMVTGeom('POLYGON ((0 0,10 0,10 5,0 -5,0 0))',
  ST_MakeBox2D(ST_Point(0, 0), ST_Point(4096, 4096)),
  4096, 0, false);
```

Output

```
MULTIPOLYGON(((5 4096,10 4091,10 4096,5 4096)),((5 4096,0 4101,0 4096,5 4096)))
```

Figure



使用计算出的瓦片边界行和裁剪几何的 Web Mercator 瓦片典型示例。本示例假设 `data.geom` 列的 SRID 为 4326。

Code

```
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)

```

相关信息

[ST_AsMVT](#), [ST_TileEnvelope](#), [PostGIS_Wagyu_Version](#)

7.9.3.10 ST_AsMVT

ST_AsMVT — 返回一行的 MVT 表示形式的聚合函数。

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);

```

描述

一个聚合函数，返回与 `ST_AsMVT` 的一行的二进制 **Mapbox 矢量瓦片** 表示。这些行必须包含一个几何列，几何列将被要素几何。几何形必须位于屏幕空中，并且根据 **MVT 规范** 有效。**ST_AsMVTGeom** 可用于将几何形映射到屏幕空。其他行列被特征属性。

Mapbox 矢量切片 格式可以存储具有不同属性集的要素。要使用此功能，在包含一深度的 **Json** 对象的行数据中提供一个 **JSONB** 列。**JSONB** 中的和将被功能属性。

可以通过使用 `||` 连接此函数的多个用来构建具有多片的切片。或使用 **STRING_AGG**。



Important

不要将 **GEOMETRYCOLLECTION** 作行中的元素行用。但是，您可以使用 **ST_AsMVTGeom** 准要包含的几何集合。

row 至少包含一个几何列的行数据。

name 是名称。默认为字符串 “default”。

extent 是规范定的屏幕空片范围。如果 **NULL**，默认为 4096。

geom_name 是行数据中几何列的名称。默认为第一个几何列。注意，PostgreSQL 默情况下会自 **将未加引号的符号折小写**，这意味着除非几何列被加引号，例如 “MyMVTGeom”，此参数必须以小写形式提供。

feature_id_name 是行数据中功能 ID 列的名称。如果 **NULL** 或数，不置功能 ID。匹配名称和有效型 (**smallint**、**integer**、**bigint**) 的第一列将用作功能 ID，任何后列将添加属性。不支持 **JSON** 属性。

增：3.0 - 添加了要素 ID 的支持。

增：2.5.0 - 添加了并行支持。

可用性：2.4.0

示例

Code

```
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;
```

相关信息

[ST_AsMVTGeom](#), [ST_TileEnvelope](#)

7.9.3.11 ST_AsSVG

ST_AsSVG — 返回几何体的 SVG 路径数据。

Synopsis

text **ST_AsSVG**(geometry geom, integer rel=0, integer maxdecimaldigits=15);
text **ST_AsSVG**(geography geog, integer rel=0, integer maxdecimaldigits=15);

描述

将几何体形作为量矢量形 (SVG) 路径数据返回。使用 1 作为第二个参数可以根据相移路径数据，默认为 (或 0) 使用相移。第三个参数可用于限制输出中使用的最大十进制位数 (默认为 15)。当 'rel' arg 为 0 时，点几何体将渲染为 cx/cy，当 'rel' arg 为 1 时，将渲染为 x/y。多点几何体由逗号 (",") 分隔，GeometryCollection 几何体由分号 (";") 分隔。

要使用 PostGIS SVG 格式，请查看 [pg_svg](#)，它提供了用于处理 ST_AsSVG 输出的 plpgsql 函数。

新增：3.4.0 支持所有曲线类型

更改：2.0.0 - 添加了默认参数和命名参数的支持

可用性：1.2.2. 可用性：1.4.0 PostGIS 1.4.0 中更改在路径中包含 L 命令以符合 <http://www.w3.org/TR/SVG/paths.html#PathDataBNF>



此方法支持形字符串和曲线。

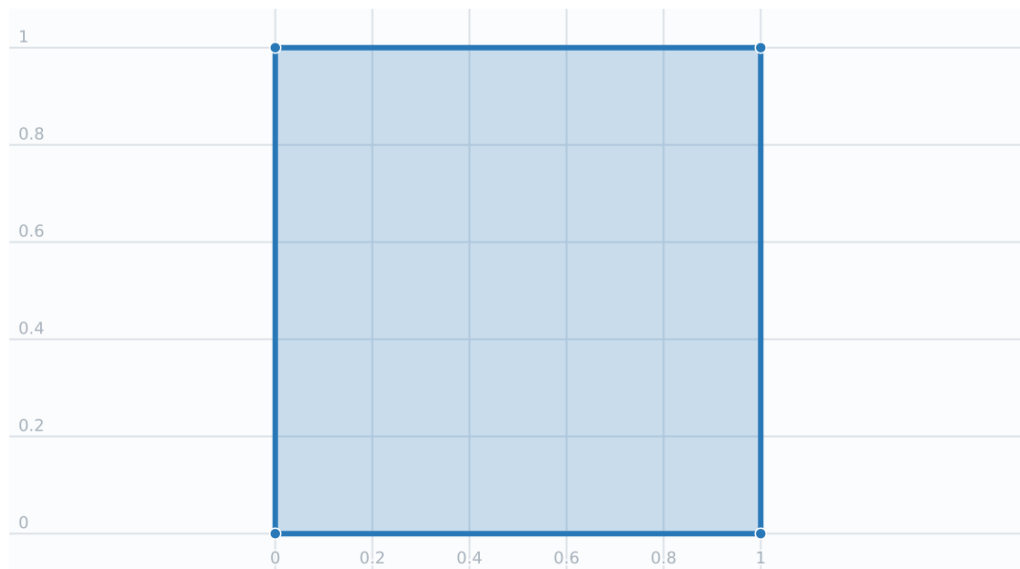
示例

Code

```
SELECT ST_AsSVG('POLYGON((0 0,0 1,1 1,1 0,0 0))'::geometry);
```

Output

```
M 0 0 L 0 -1 1 -1 1 0 Z
```

Figure

 Code

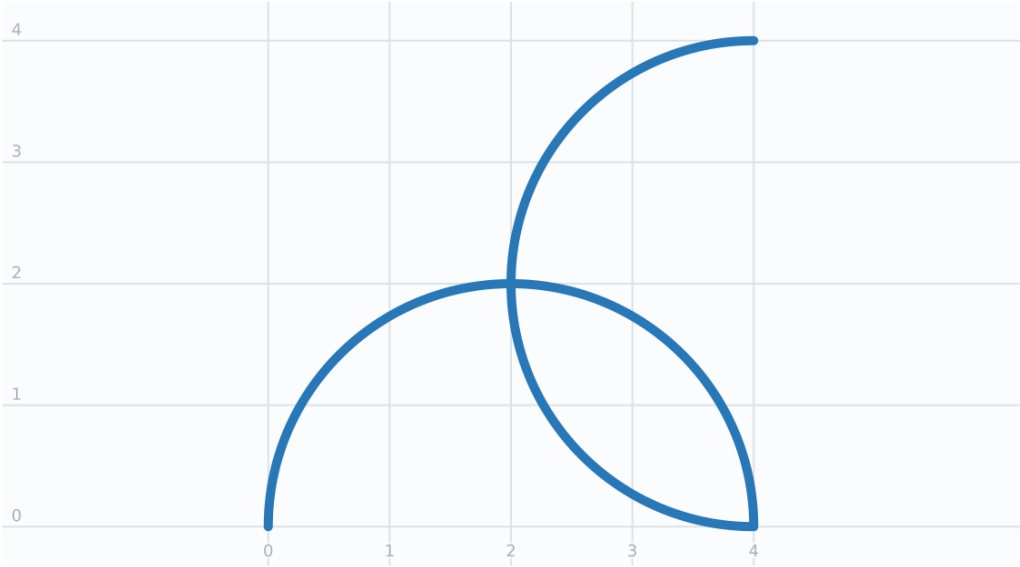
**Code**

```
SELECT ST_AsSVG('CIRCULARSTRING(-2 0,0 2,2 0,0 2,2 4)'::geometry);
```

Output

```
M -2 0 A 2 2 0 0 1 2 0 A 2 2 0 0 1 2 -4
```

Figure



Code

多曲

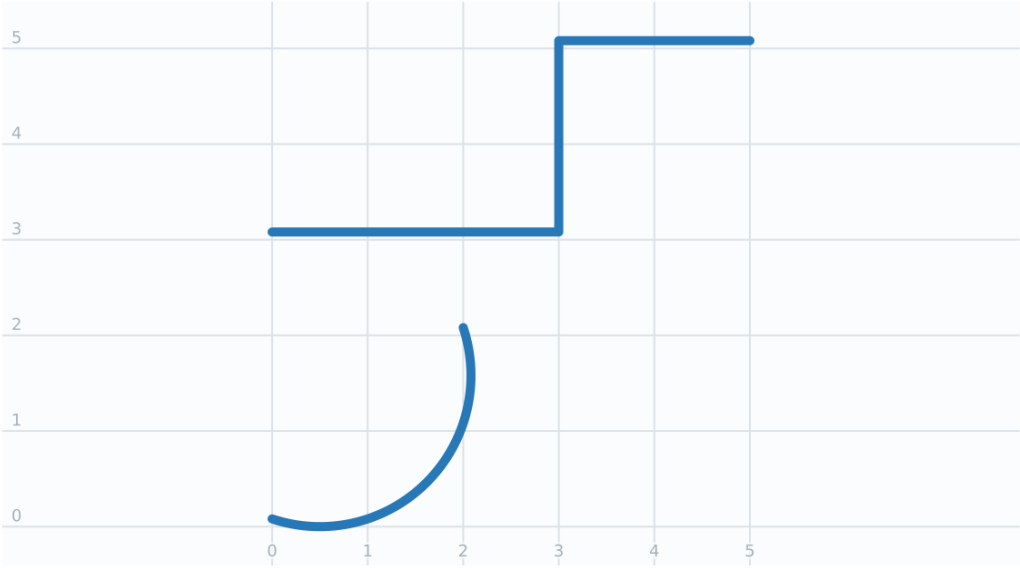
Code

```
SELECT ST_AsSVG('MULTICURVE((5 5,3 5,3 3,0 3),  
CIRCULARSTRING(0 0,2 1,2 2))'::geometry, 0, 0);
```

Output

```
M 5 -5 L 3 -5 3 -3 0 -3 M 0 0 A 2 2 0 0 0 2 -2
```

Figure



Code

多曲面

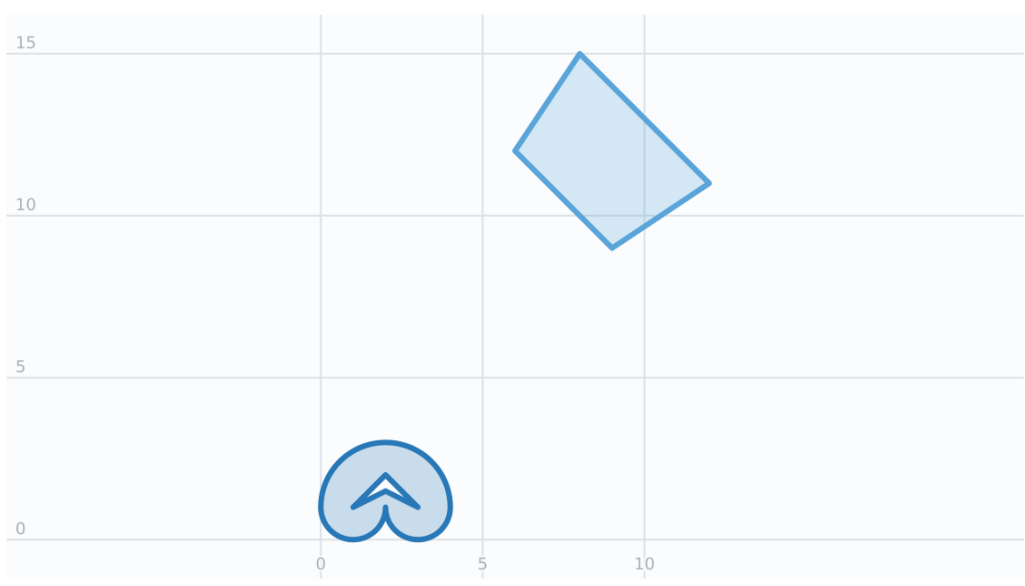
Code

```
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);
```

Output

```
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
```

Figure



Code

7.9.3.12 ST_AsTWKB

ST_AsTWKB — 返回几何形式为 TWKB，又名“微小的已知的二进制制”

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

返回 TWKB（微小已知的二进制制）格式的几何形状。TWKB 是一种 **变长的二进制制格式**，注重最小化输出的大小。小数位参数控制输出中存储的精度。默认情况下，在输出前四舍五入到最接近的位。如果要输出更高的精度，增加数字。例如，1 表示将保留小数点右边的第一位数字。

大小和边界框参数控制输出中是否包含有关几何体的维度和几何体边界的可变信息。默认情况下，它不是。除非您的客户端组件需要使用它，否则不要打开它，因为它只会占用空间（节省空间是 TWKB 的目的）。

该函数的数字输入形式用于将几何体和唯一标识符的集合保留为 TWKB 集合。对于希望解集合然后有关内部几何体的更多信息的客户端非常有用。您可以使用 `array_agg` 函数构建数字。其他参数的操作与函数的数字形式相同。



Note

格式范例可在 <https://github.com/TWKB/Specification> 在获取，构建 JavaScript 客户端的代码可在 <https://github.com/TWKB/twkb.js> 找到。

增加：2.4.0 内存和速度改进。

可用性：2.2.0

示例

Code

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

Output

```
\x0200020202020808
```

要构建包含标识符的聚合 TWKB 几何体，首先使用 “array_agg()” 聚合所需的几何体和标识，然后用适当的 TWKB 函数。

Code

```
SELECT ST_AsTWKB(array_agg(geom), array_agg(gid)) FROM mytable;
```

Output

```
\x040402020400000202
```

相关信息

[ST_GeomFromTWKB](#), [ST_AsBinary](#), [ST_AsEWKB](#), [ST_AsEWKT](#), [ST_GeomFromText](#)

7.9.3.13 ST_AsX3D

ST_AsX3D — 返回 X3D xml 点元素格式的几何体：ISO-IEC-19776-1.2-X3DEncodings-XML

Synopsis

text **ST_AsX3D**(geometry g1, integer maxdecimaldigits=15, integer options=0);

描述

以 X3D xml 格式的几何元素形式返回几何图形 <http://www.web3d.org/standards/number/19776-1>。如果未指定 `maxdecimaldigits` (精度), 默认为 15。

Note

有多种几何类型可用于将 PostGIS 几何类型转换为 X3D, 因为 X3D 几何类型不会直接映射到 PostGIS 几何类型, 而一些新的 X3D 类型可能是我们避免的更好的映射, 因为大多数渲染工具当前不支持它们。这些是我们已确定的映射。如果您允许他人表示他们首选映射的想法或方式有任何想法, 请随时发布通知。

以下是我们目前如何将 PostGIS 2D/3D 类型映射到 X3D 类型

`options` 参数是一个位字段, 用于控制是否使用 X3D 的 `GeoCoordinates` 点来表示坐标, 以及是否进行 x/y 翻转。默认情况下, `ST_AsX3D` 按数据方式输出 (long,lat 或 X,Y), 但如果需要, 也可以采用 X3D 默认的 lat/lon、y/x 输出形式。

- 0 : 数据顺序中的 X/Y (例如, 经/纬度 = X,Y 是标准数据顺序)、默认和非空坐标 (只是常规的旧坐标)。
- 1 : 翻转 X 和 Y。如果与地理坐标开关合使用, 输出将默认为 “latitude_first”, 并且坐标也会翻转。
- 2 : GeoSpatial `GeoCoordinates` 中的输出坐标。如果几何类型不在 WGS 84 经度 (srid: 4326) 中, 此输出将索引。这是目前唯一支持的地理坐标类型。请参考指定空参考系的 X3D 规范。默认输出将 `GeoCoordinate` `geoSystem="GD" "WE" "longitude_first"`。如果您更喜欢 `GeoCoordinate` `geoSystem="GD" "WE" "latitude_first"` 的 X3D 默认, 使用 $(2+1) = 3$

PostGIS 类型	2D X3D 类型	3D X3D 类型
LINESTRING	LineSet (coordinates are promoted to 3D)	LineSet
MULTILINESTRING	IndexedLineSet (coordinates are promoted to 3D)	IndexedLineSet
MULTIPOINT	Polypoint2D	PointSet
POINT	输出以空格分隔的坐标	输出以空格分隔的坐标
(MULTI) POLYGON, POLYHEDRALSURFACE	IndexedFaceSet (coordinates are promoted to 3D; inner rings currently output as another faceset)	IndexedFaceSet (内当前输出为一个面集)
TIN	TriangleSet2D (尚未实现)	IndexedTriangleSet (索引三角形集)
CIRCULARSTRING, COMPOUNDCURVE, CURVEPOLYGON, MULTICURVE, MULTISURFACE	linearized, then emitted with the linear geometry mapping	linearized, then emitted with the linear geometry mapping
NURBSCURVE	NurbsCurve with controlPoint, knot, weight, and order fields	NurbsCurve with controlPoint, knot, weight, and order fields

**Note**

2D 几何支持尚未完成。内当前限制独特的多边形。我们正在研究这些。


```

    </IndexedFaceSet>
  </Shape>
</Transform>
</Scene>
</X3D>

```

PostGIS buildings.

将此文件的输出复制并粘贴到x3d 场景查看器，然后显示“显示”

Code

```

SELECT string_agg('<Shape
>' || ST_AsX3D(CG_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);

```

The following preview shows the extruded geometry generated from a simplified version of the same source lettering. It is a geometry preview, not X3D markup; use the preceding ST_AsX3D query output in an X3D viewer to inspect the native X3D scene.

Code

```

SELECT
  CG_Extrude(
    ST_SimplifyPreserveTopology(ST_Letters('PostGIS'), 2),
    0, 0, 10
  ) AS buildings;

```

Figure

3D view



Output

An Octagon elevated 3 Units and decimal precision of 6.

Code

```
SELECT ST_AsX3D(ST_Translate(ST_Force3D(ST_Buffer(ST_Point(10,10),5, 'quad_segs=2')), 0,0,
3),
6) As x3dfrag;
```

Output

```
x3dfrag
-----
<IndexedFaceSet  convex='false' coordIndex='0 1 2 3 4 5 6 7'
><Coordinate point='15 10 3 13.535534 6.464466 3 10 5 3 6.464466 6.464466 3 5 10 3 6.464466 13.535534 3 10 15 3  ↵
13.535534 13.535534 3 ' /></IndexedFaceSet>
```

TIN.

Code

```
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;
```

Output

```
x3dfrag
-----
<IndexedTriangleSet  index='0 1 2 3 4 5'
><Coordinate point='0 0 0 0 1 0 1 0 0 0 0 1 0 1 1 0' /></IndexedTriangleSet>
```

Closed multilinestring (the boundary of a polygon with holes).

Code

```
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;
```

Output

```
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 — 返回几何图形的 GeoHash 表示形式。

Synopsis

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

描述

几何形状的 **GeoHash** 表示。GeoHash 将地理点可基于前排序和搜索的文本形式。短的 GeoHash 是点的不太精确的表示。它可以被是一个包含点的盒子。

具有非零范围的非点几何也可以映射到 GeoHash 代码。代码的精度取决于几何的地理范围。

如果未指定 `maxchars`，返回的 GeoHash 代码适用于包含几何的最小元格。`Points` 返回精度 20 个字符的 GeoHash（大足以容纳的完整双精度）。其他几何型可能会返回精度低的 GeoHash，具体取决于几何形状的范围。大的几何的精度低，小的几何的精度高。由 GeoHash 代码确定的框始终包含特征。

如果指定了 `maxchars` 参数，返回的 GeoHash 代码最多包含指定数量的字符。它映射到几何体的（可能）低精度表示。对于非点几何体，计算的起始点是几何体边框的中心点。

可用性：1.4.0



Note

ST_GeoHash 要求输入几何采用地理（度/度）坐标。



此方法支持形字符串和曲线。

示例

Code

```
SELECT ST_GeoHash(ST_Point(-126, 48));
```

Output

```
c0w3hfls70w3hfls70w3
```

Code

```
SELECT ST_GeoHash(ST_Point(-126, 48), 5);
```

Output

```
c0w3h
```

This example shows that the GeoHash for a line containing a point is a prefix of the point code.

Code

```
SELECT ST_GeoHash('LINESTRING(-126 48,-126.1 48.1)::geometry');
```

Output

```
c0w3
```

相关信息

[ST_GeomFromGeoHash](#), [ST_PointFromGeoHash](#), [ST_Box2dFromGeoHash](#)

7.10 ☒算符

7.10.1 ☒界框☒算符

7.10.1.1 &&

&& — 如果 A 的 2D ☒界框与 B 的 2D ☒界框相交，☒返回 TRUE。

Synopsis

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

描述

如果 A 的二☒☒界框与 B 的二☒☒界框交互，**&&** 返回 TRUE。



Note

☒操作符将利用几何上可能可用的任何索引。

增☒：2.0.0 引入了☒多面体曲面的支持。

可用性：1.5.0 引入了☒地理的支持。

✔ 此方法支持☒形字符串和曲☒。

✔ ☒函数支持多面体曲面。

示例

Code

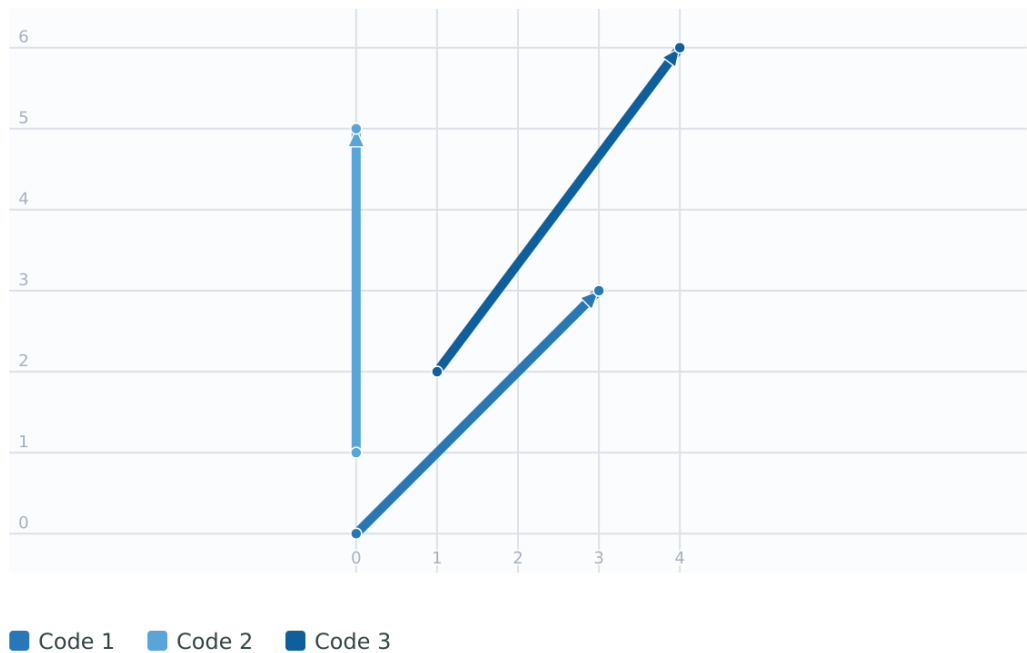
```
SELECT tbl1.column1, tbl2.column1, tbl1.column2 && tbl2.column2 AS overlaps
FROM ( VALUES
      (1, 'LINESTRING(0 0, 3 3)::geometry),
      (2, 'LINESTRING(0 1, 0 5)::geometry)) AS tbl1,
( VALUES
      (3, 'LINESTRING(1 2, 4 6)::geometry)) AS tbl2;
```

Output

column1	column1	overlaps
1	3	t
2	3	f

(2 rows)

Figure



相关信息

`ST_Intersects`, `ST_Extent`, `|&>`, `&>`, `&<|`, `&<`, `~`, `@`

7.10.1.2 `&&(geometry,box2df)`

`&&(geometry,box2df)` — 如果几何体的 (存储的) 2D 边界框与 2D 浮点精度边界框 (BOX2DF) 相交, 返回 TRUE。

Synopsis

boolean `&&( geometry A , box2df B );`

描述

如果几何体 A 的存储 2D 边界框与 2D 边界框 B 相交 (使用浮点精度), `&&` 算符返回 TRUE。这意味着如果 B 是 (双精度) `box2d`, 它将在内部转换为浮点精度 2D 边界框 (BOX2DF)



Note
该操作数旨在由 BRIN 索引内部使用, 而不是由用户使用。

可用性 : 2.3.0 - 引入了对 BRIN 的支持。

- ✓ 此方法支持几何字符串和曲线。
- ✓ 该函数支持多面体曲面。

示例

Code

```
SELECT ST_Point(1,1) && ST_MakeBox2D(ST_Point(0,0), ST_Point(2,2)) AS overlaps;
```

Output

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

相关信息

`&&(box2df,geometry)`, `&&(box2df,box2df)`, `~(geometry,box2df)`, `~(box2df,geometry)`, `~(box2df,box2df)`, `@(geometry,box2df)`, `@(box2df,geometry)`, `@(box2df,box2df)`

7.10.1.3 &&(box2df,geometry)

`&&(box2df,geometry)` — 如果 2D 浮点精度 `BOX2DF` 与几何体的（`BOX2DF`）2D `BOX2DF` 相交，返回 TRUE。

Synopsis

boolean `&&( box2df A , geometry B );`

描述

`&&` 算符在使用浮点精度 `BOX2DF`，如果 2D `BOX2DF` A 与几何体 B 的 `BOX2DF` 2D `BOX2DF` 相交，返回 TRUE。`&&` 意味着如果 A 是一个（双精度）`box2d`，它将在内部被 `BOX2DF` 一个浮点精度的 2D `BOX2DF`。



Note

`&&` 操作数旨在由 BRIN 索引内部使用，而不是由用户使用。

可用性：2.3.0 - 引入了 `BRIN` 的支持。



此方法支持 `LINESTRING` 和 `CURVE`。



`&&` 函数支持多面体曲面。

示例

Code

```
SELECT ST_MakeBox2D(ST_Point(0,0), ST_Point(2,2)) && ST_Point(1,1) AS overlaps;
```

Output

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

相关信息

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

7.10.1.4 &&(box2df,box2df)

[&&\(box2df,box2df\)](#) — 如果两个 2D 浮点精度边界框 (BOX2DF) 彼此相交，则返回 TRUE。

Synopsis

boolean **&&**(box2df A , box2df B);

描述

[&&](#) 运算符在使用浮点精度时，如果两个 2D 边界框 A 和 B 相交，则返回 TRUE。这意味着如果 A（或 B）是一个（双精度）box2d，它将在内部被转换为一个浮点精度的 2D 边界框 (BOX2DF)



Note
该运算符旨在供 BRIN 索引内部使用，而不是供用户使用。

可用性：2.3.0 - 引入了对 BRIN 的支持。

- ✓ 此方法支持几何字符串和曲线。
- ✓ 该函数支持多面体曲面。

示例

Code

```
SELECT ST_MakeBox2D(ST_Point(0,0), ST_Point(2,2)) && ST_MakeBox2D(ST_Point(1,1), ST_Point(3,3)) AS overlaps;
```

Output

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

相关信息

`&&(geometry,box2df), &&(box2df,geometry), ~(geometry,box2df), ~(box2df,geometry), ~(box2df,box2df), @ (geometry,box2df), @ (box2df,geometry), @ (box2df,box2df)`

7.10.1.5 &&&

`&&&` — 如果 A 的 n 维界框与 B 的 n 维界框相交，返回 TRUE。

Synopsis

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

描述

`&&&` 运算符返回 TRUE，如果 A 的 n 维界框与 B 的 n 维界框相交。



Note
该操作符将利用几何上可能可用的任何索引。

可用性: 2.0.0

- ✓ 此方法支持几何字符串和曲线。
- ✓ 该函数支持多面体曲面。
- ✓ 此函数支持三角形和不规则三角网面 (TIN)。
- ✓ 该函数支持 3d 并且不会丢失 z-index。

示例

3D LineStrings.

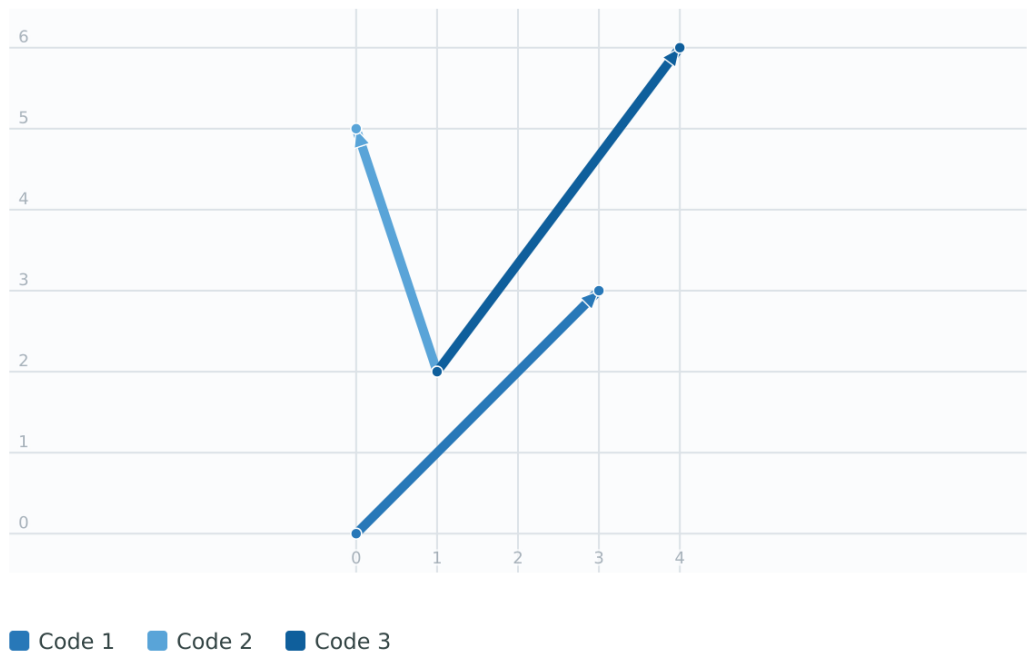
Code

```
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;
```

Output

column1	column1	overlaps_3d	overlaps_2d
1	3	t	t
2	3	f	t

Figure



3M LineStrings.

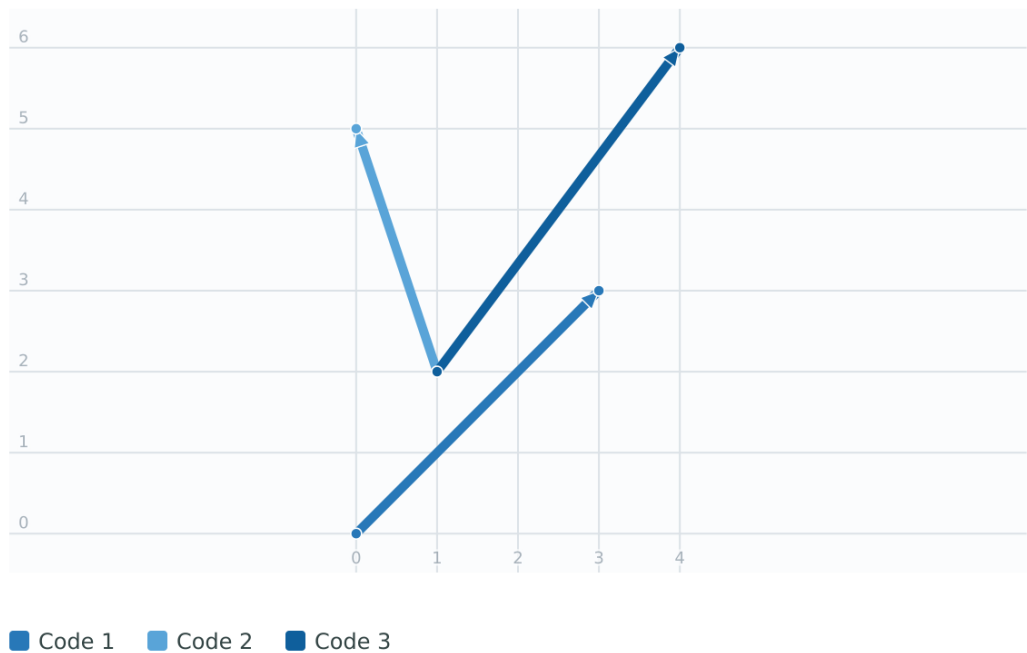
Code

```
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;
```

Output

column1	column1	overlaps_3zm	overlaps_2d
1	3	t	t
2	3	f	t

Figure



相关信息

&&

7.10.1.6 &&&(geometry,gidx)

&&&(geometry,gidx) — 如果几何体的（[存储的](#)）[n 维](#) [边界框](#)与 [n 维浮点精度边界框 \(GIDX\)](#) 相交，[返回 TRUE](#)。

Synopsis

boolean &&&(geometry A , gidx B);

描述

[&&&](#) [算符](#)在使用[浮点精度](#)，如果几何体 A 的[存储 n-D 边界框](#)与 [n-D 边界框 B](#) 相交，[返回 TRUE](#)。[意味着如果 B 是一个（双精度）box3d，它将在内部被转换为一个浮点精度的 3D 边界框 \(GIDX\)](#)



Note
[算符](#)旨在供 BRIN 索引内部使用，而不是供[用](#)使用。

可用性：2.3.0 - 引入了[BRIN](#) 的支持。

- [✓](#) 此方法支持[形字符串](#)和[曲](#)。
- [✓](#) [函数](#)支持多面体曲面。
- [✓](#) 此函数支持三角形和[不](#)三角网面 (TIN)。
- [✓](#) [函数](#)支持 3d 并且不会[失 z-index](#)。

示例

Code

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

Output

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

相关信息

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

7.10.1.7 &&&(gidx,geometry)

&&&(gidx,geometry) — 如果 n 浮点精度 **框** (GIDX) 与几何体的 (存的) n **框** 相交, 返回 **TRUE**。

Synopsis

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

描述

&&& 算符在使用浮点精度, 如果 n -D **框** A 与几何体 B 的存 n -D **框** 相交, 返回 **TRUE**。意味着如果 A 是一个 (双精度) **box3d**, 它将在内部被 **框** 一个浮点精度的 3D **框** (GIDX)



Note

框算符旨在供 BRIN 索引内部使用, 而不是供用 **框** 使用。

可用性 : 2.3.0 - 引入了 **框** BRIN 的支持。

- 此方法支持 **框**形字符串和曲 **框**。
- 框**函数支持多面体曲面。
- 此函数支持三角形和不 **框**三角网面 (TIN)。
- 框**函数支持 3d 并且不会 **框**失 z-index。

示例

Code

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

Output

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

相关信息

[&&&\(geometry,gidx\), &&&\(gidx,gidx\)](#)

7.10.1.8 &&&(gidx,gidx)

[&&&\(gidx,gidx\)](#) — 如果 n 个浮点精度 [GIDX](#) 彼此相交，返回 `TRUE`。

Synopsis

boolean [&&&](#)(gidx A , gidx B);

描述

[&&&](#) 算符在使用浮点精度，如果 n -D [GIDX](#) A 和 B 相交，返回 `TRUE`。这意味着如果 A（或 B）是一个（双精度）`box3d`，它将在内部被转换为一个浮点精度的 3D [GIDX](#)。

**Note**

[&&&](#) 算符旨在供 BRIN 索引内部使用，而不是供用户使用。

可用性：2.3.0 - 引入了 [&&&](#) BRIN 的支持。

- ✓ 此方法支持 [&&&](#) 形字符串和曲线。
- ✓ [&&&](#) 函数支持多面体曲面。
- ✓ 此函数支持三角形和不 [&&&](#) 三角网面 (TIN)。
- ✓ [&&&](#) 函数支持 3d 并且不会 [&&&](#) 失 z-index。

示例

Code

```
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;
```

Output

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

相关信息

[ST_Overlaps\(geometry,gidx\)](#), [ST_Overlaps\(gidx,geometry\)](#)

7.10.1.9 <

< — 如果 A 的边界框与 B 的边界框重叠或位于其左侧，则返回 TRUE。

Synopsis

boolean <(geometry A , geometry B);

描述

< 运算符返回 TRUE 如果几何体 A 的边界框与几何体 B 的边界框重叠或在几何体 B 的边界框的左侧，或者更准确地，不位于几何体 B 的边界框的右侧。



Note
该操作符将利用几何上可能可用的任何索引。

示例

Code

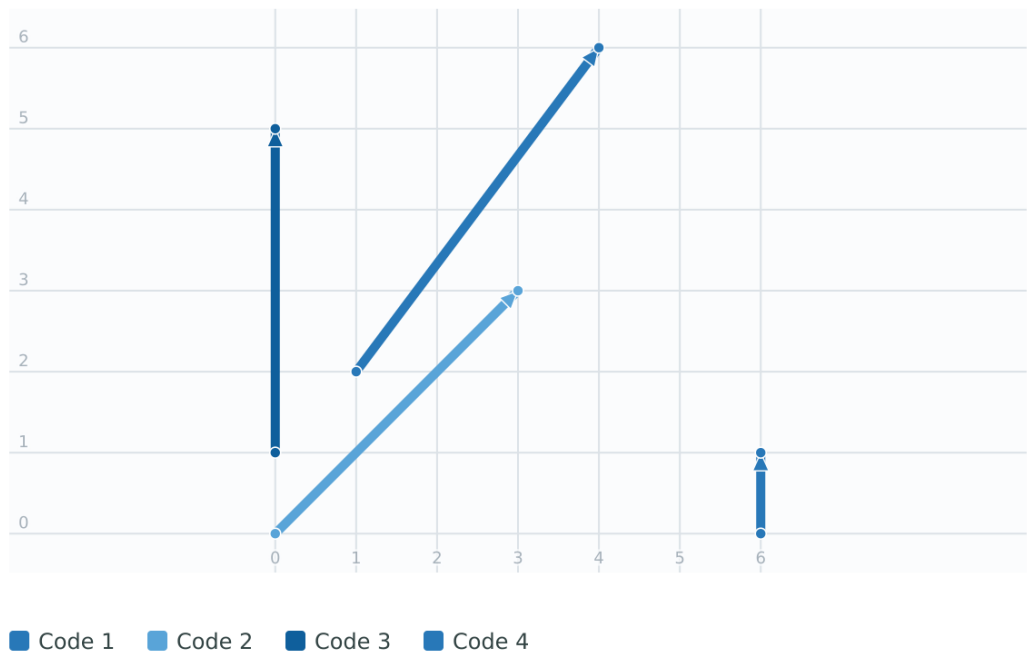
```
SELECT tbl1.column1, tbl2.column1, tbl1.column2 < tbl2.column2 AS overleft
FROM
  ( VALUES
    (1, 'LINESTRING(1 2, 4 6)::geometry)) AS tbl1,
  ( VALUES
    (2, 'LINESTRING(0 0, 3 3)::geometry),
    (3, 'LINESTRING(0 1, 0 5)::geometry),
    (4, 'LINESTRING(6 0, 6 1)::geometry)) AS tbl2;
```

Output

column1	column1	overleft
1	2	f
1	3	f
1	4	t

(3 rows)

Figure



相关信息

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

7.10.1.10 `&<|`

`&<|` — 如果 A 的边界框与 B 的边界框重叠或低于 B 的边界框，返回 TRUE。

Synopsis

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

描述

`&<|` 运算符返回 TRUE，如果几何体 A 的边界框与几何体 B 的边界框重叠或低于几何体 B 的边界框，或者更准确地说，不位于几何体 B 的边界框的上方。

- ✓ 此方法支持几何字符串和曲线。
- ✓ 函数支持多面体曲面。



Note
操作符将利用几何上可能可用的任何索引。

示例

Code

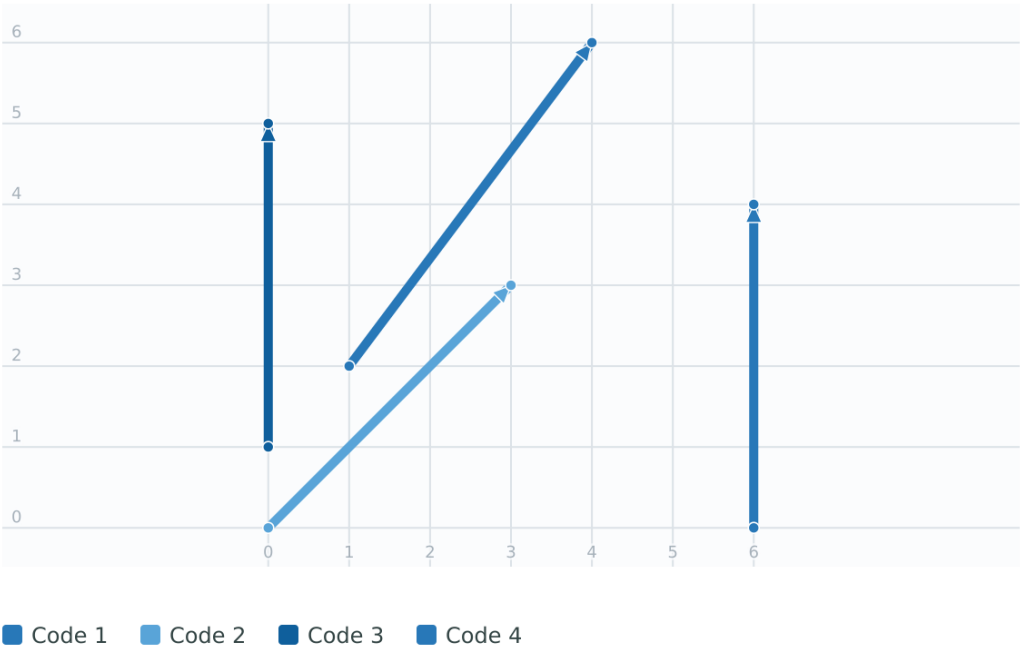
```
SELECT tbl1.column1, tbl2.column1, tbl1.column2 &<| tbl2.column2 AS overbelow
FROM
  ( VALUES
    (1, 'LINESTRING(6 0, 6 4)::geometry') AS tbl1,
    ( VALUES
      (2, 'LINESTRING(0 0, 3 3)::geometry'),
      (3, 'LINESTRING(0 1, 0 5)::geometry'),
      (4, 'LINESTRING(1 2, 4 6)::geometry') AS tbl2;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

Output

column1	column1	overbelow
1	2	f
1	3	t
1	4	t

(3 rows)

Figure



相关信息

[&&](#), [|&>](#), [&>](#), [&<](#)

7.10.1.11 &>

[&>](#) — 如果 A 的 [☐](#)界框与 B 的 [☐](#)界框重[☐](#)或位于 B 右[☐](#)， [☐](#)返回 TRUE。

Synopsis

boolean [&>](#)(geometry A , geometry B);

描述

`&>` 算符 返回 `TRUE`，如果几何体 A 的边界框与几何体 B 的边界框重叠或位于其右侧，更准确地，它与几何体 B 的边界框不重叠于其左侧。



Note
操作符将利用几何上可能可用的任何索引。

示例

Code

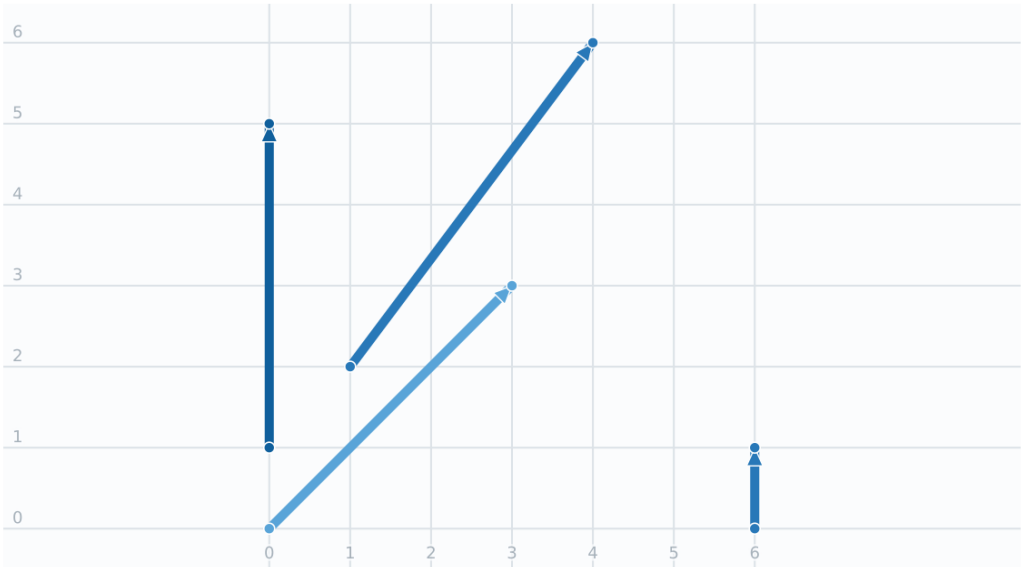
```
SELECT tbl1.column1, tbl2.column1, tbl1.column2 &
> tbl2.column2 AS overright
FROM
  ( VALUES
    (1, 'LINESTRING(1 2,4 6)::geometry)) AS tbl1,
  ( VALUES
    (2, 'LINESTRING(0 0,3 3)::geometry),
    (3, 'LINESTRING(0 1,0 5)::geometry),
    (4, 'LINESTRING(6 0,6 1)::geometry)) AS tbl2;
```

Output

column1	column1	overright
1	2	t
1	3	t
1	4	f

(3 rows)

Figure



Code 1 Code 2 Code 3 Code 4

相关信息

[&&](#), [|&>](#), [&<|](#), [&<](#)

7.10.1.12 <<

<< — 如果 A 的☐界框☐格位于 B 的左☐，☐返回 TRUE。

Synopsis

boolean <<(geometry A , geometry B);

描述

如果 A 的☐界框☐格位于 B 的☐界框的左☐，☐ << ☐算符返回 TRUE。



Note
☐操作符将利用几何上可能可用的任何索引。

示例

Code

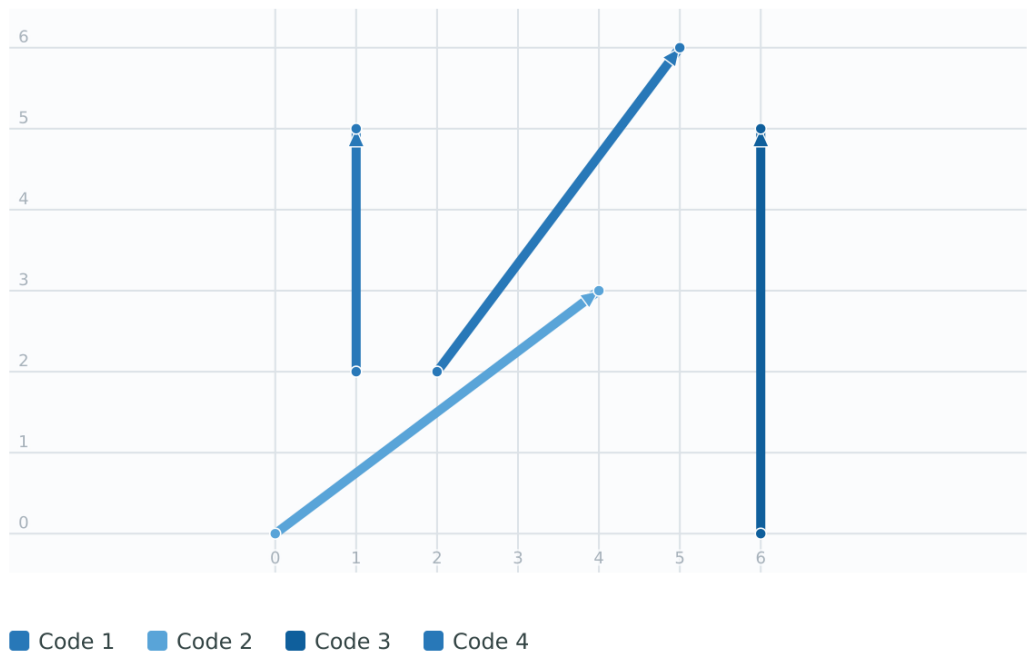
```
SELECT tbl1.column1, tbl2.column1, tbl1.column2 << tbl2.column2 AS left
FROM
  ( VALUES
    (1, 'LINESTRING (1 2, 1 5)'::geometry)) AS tbl1,
  ( VALUES
    (2, 'LINESTRING (0 0, 4 3)'::geometry),
    (3, 'LINESTRING (6 0, 6 5)'::geometry),
    (4, 'LINESTRING (2 2, 5 6)'::geometry)) AS tbl2;
```

Output

column1	column1	left
1	2	f
1	3	t
1	4	t

(3 rows)

Figure



相关信息

>>, |>>, <<|

7.10.1.13 <<|

<<| — 如果 A 的☐界框☐格低于 B 的☐界框，☐返回 TRUE。

Synopsis

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

描述

如果 A 的☐界框☐格低于 B 的☐界框，☐☐算符 <<| 返回 TRUE。



Note
☐操作符将利用几何上可能可用的任何索引。

示例

Code

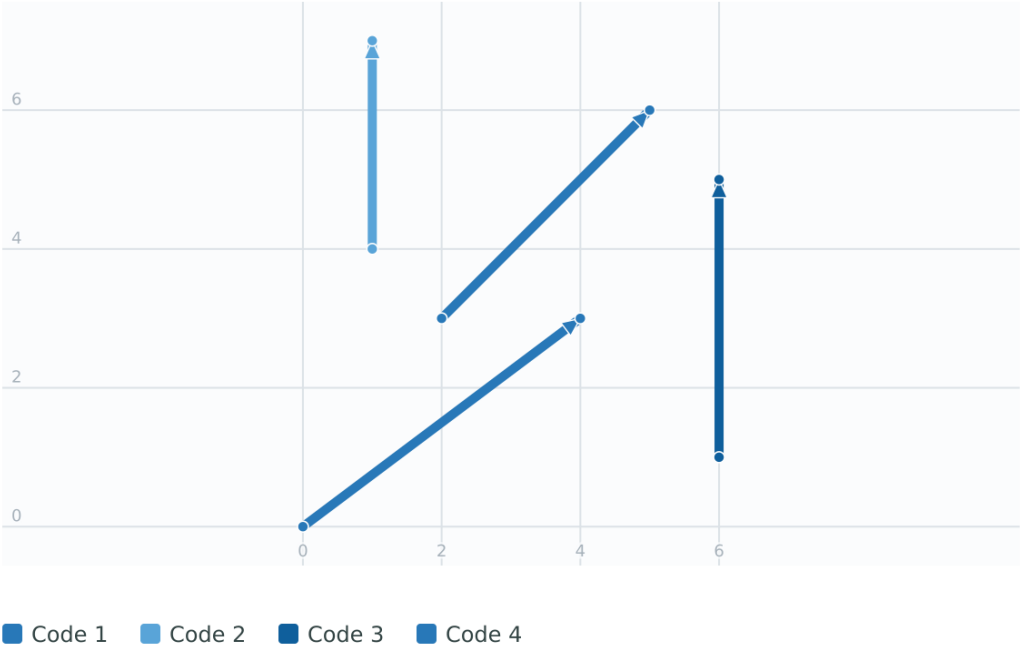
```
SELECT tbl1.column1, tbl2.column1, tbl1.column2 <<| tbl2.column2 AS below
FROM
  ( VALUES
    (1, 'LINESTRING (0 0, 4 3)::geometry')) AS tbl1,
  ( VALUES
    (2, 'LINESTRING (1 4, 1 7)::geometry'),
    (3, 'LINESTRING (6 1, 6 5)::geometry'),
    (4, 'LINESTRING (2 3, 5 6)::geometry')) AS tbl2;
```

Output

column1	column1	below
1	2	t
1	3	f
1	4	f

(3 rows)

Figure



相关信息

<<, >>, |>>

7.10.1.14 =

= — 如果几何/地理 A 的坐☐和坐☐☐序与几何/地理 B 的坐☐和坐☐☐序相同，☐返回 TRUE。

Synopsis

boolean =(geometry A , geometry B);
boolean =(geography A , geography B);

描述

如果几何/地理 A 的坐点序和坐点序与几何/地理 B 的坐点和坐点序相同，= 运算符返回 TRUE。PostgreSQL 使用几何定义的 =、< 和 > 运算符来对行内部排序和比较几何形状（即在 GROUP BY 或 ORDER BY 子句中）。



Note 只有在所有方面都完全相同、具有相同坐点、相同点序的几何/地理才被该运算符相等。对于“空点相等”，它忽略坐点序等内容，并且可以以不同表示覆盖相同空点区域的要素，使用 **ST_OrderingEquals** 或 **ST_Equals**



Caution 此运算符不几何形状使用任何有效索引。要使用索引行的键的相等性，使用 = 和 &&。

更改：2.4.0，在之前的版本中，是边界框相等而不是几何相等。如果需要边界框相等，使用 **~=** 代替。

- ✓ 此方法支持点字符串和曲线。
- ✓ 函数支持多面体曲面。

示例

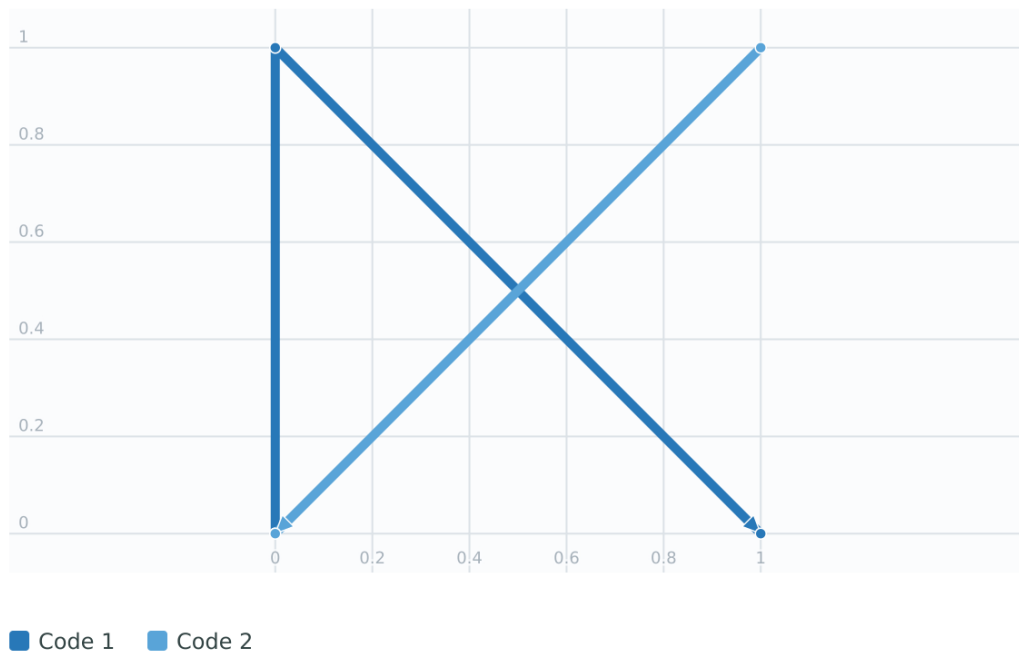
Code

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

Output

```
?column?
-----
f
(1 row)
```

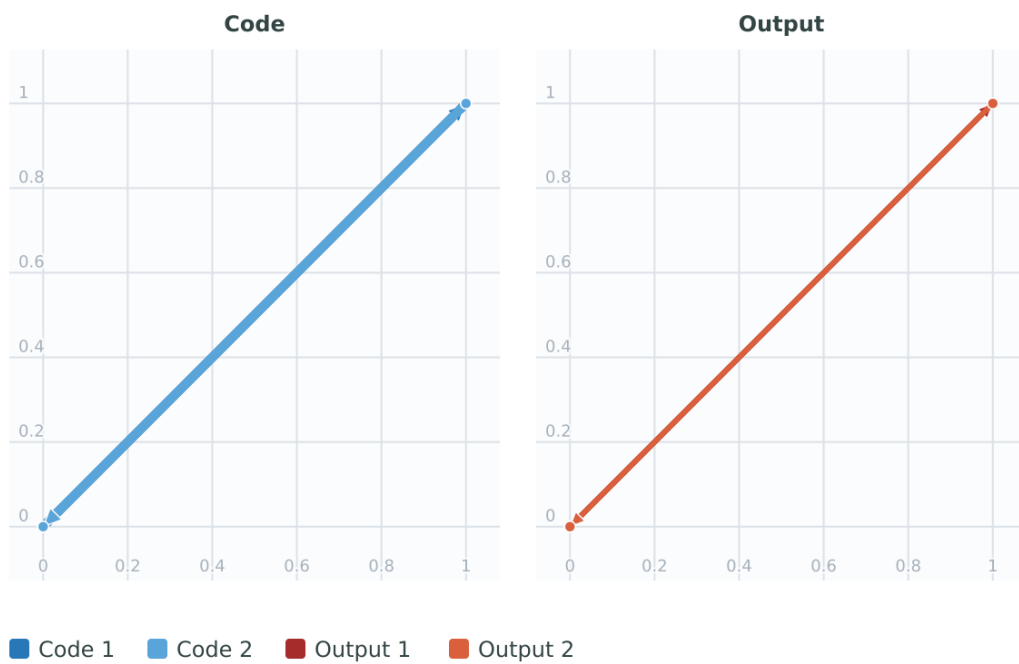
Figure



Code

```
SELECT column1
FROM ( VALUES
      ('LINESTRING(0 0, 1 1)::geometry),
      ('LINESTRING(1 1, 0 0)::geometry)) AS foo;
```

Figure

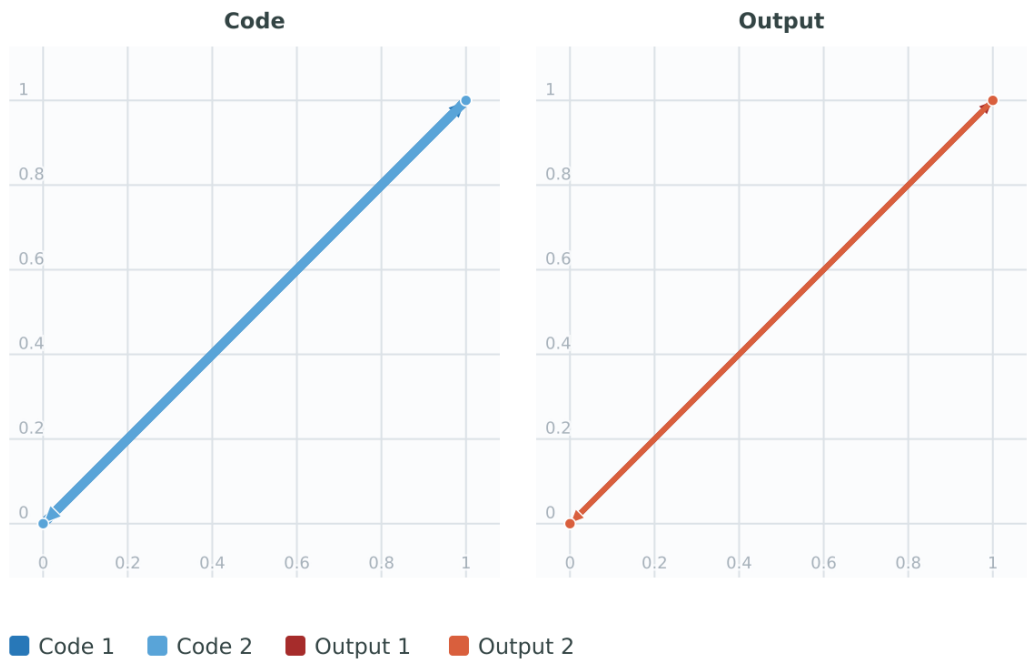


Note: the GROUP BY uses the "=" to compare for geometry equivalency.

Code

```
SELECT column1
FROM ( VALUES
      ('LINESTRING(0 0, 1 1)::geometry),
      ('LINESTRING(1 1, 0 0)::geometry)) AS foo
GROUP BY column1;
```

Figure



In versions prior to 2.0, this used to return true.

Code

```
SELECT 'POINT(1707296.37 4820536.77)::geometry' =
      'POINT(1707296.27 4820536.87)::geometry' AS pt_intersect;
```

Output

```
pt_intersect
-----
f
(1 row)
```

相关信息

[ST_Equals](#), [ST_OrderingEquals](#), [~=](#)

7.10.1.15 >>

>> — 如果 A 的 [边界框](#) 格位于 B 的右 [边界框](#)，[返回 TRUE](#)。

Synopsis

boolean >>(geometry A , geometry B);

描述

如果几何体 A 的☐界框☐格位于几何体 B 的☐界框右☐，☐ >> ☐算符返回 TRUE。



Note
☐操作符将利用几何上可能可用的任何索引。

示例

Code

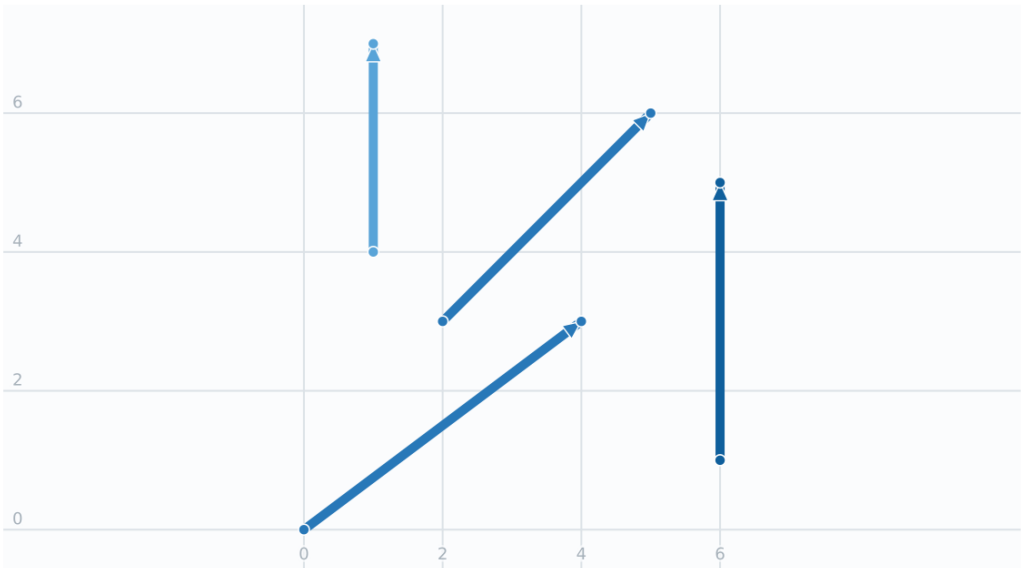
```
SELECT tbl1.column1, tbl2.column1, tbl1.column2
>
> tbl2.column2 AS right
FROM
( VALUES
  (1, 'LINESTRING (2 3,5 6)')::geometry)) AS tbl1,
( VALUES
  (2, 'LINESTRING (1 4,1 7)')::geometry),
  (3, 'LINESTRING (6 1,6 5)')::geometry),
  (4, 'LINESTRING (0 0,4 3)')::geometry)) AS tbl2;
```

Output

column1	column1	right
1	2	t
1	3	f
1	4	f

(3 rows)

Figure



Code 1 Code 2 Code 3 Code 4

相关信息

<<, |>>, <<|

7.10.1.16 @

@ — 如果 A 的☒界框包含在 B 的☒界框中，☒返回 TRUE。

Synopsis

boolean @(geometry A , geometry B);

描述

如果几何体 A 的☒界框完全包含在几何体 B 的☒界框内，☒ @ ☒算符返回 TRUE。



Note

☒操作符将利用几何上可能可用的任何索引。

示例

Code

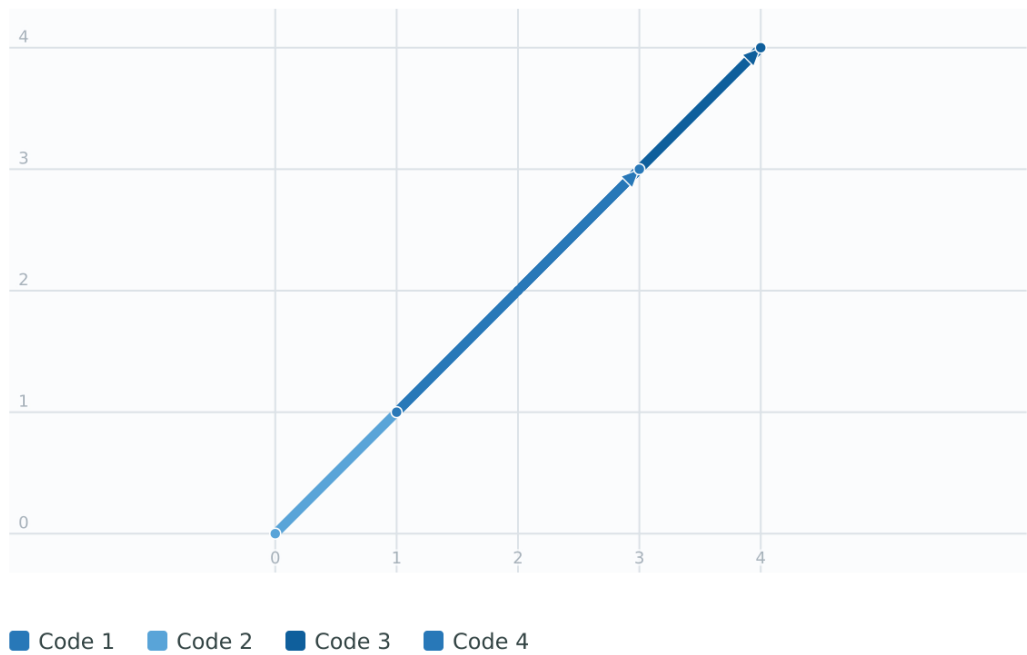
```
SELECT tbl1.column1, tbl2.column1, tbl1.column2 @ tbl2.column2 AS contained
FROM
  ( VALUES
    (1, 'LINESTRING (1 1, 3 3)::geometry')) AS tbl1,
  ( VALUES
    (2, 'LINESTRING (0 0, 4 4)::geometry'),
    (3, 'LINESTRING (2 2, 4 4)::geometry'),
    (4, 'LINESTRING (1 1, 3 3)::geometry')) AS tbl2;
```

Output

column1	column1	contained
1	2	t
1	3	f
1	4	t

(3 rows)

Figure



相关信息

~, &&

7.10.1.17 @(geometry,box2df)

@(geometry,box2df) — 如果几何体的 2D 边界框包含在 2D 浮点精度边界框 (BOX2DF) 中，返回 TRUE。

Synopsis

boolean @(geometry A , box2df B);

描述

如果 A 几何形状的 2D 边界框包含 2D 边界框 B（使用浮点精度），@ 运算符返回 TRUE。这意味着如果 B 是（双精度）box2d，它将在内部转换为浮点精度 2D 边界框（BOX2DF）



Note
操作数旨在由 BRIN 索引内部使用，而不是由用户使用。

可用性：2.3.0 - 引入了对 BRIN 的支持。

- ✓ 此方法支持几何字符串和曲线。
- ✓ 函数支持多面体曲面。

示例

Code

```
SELECT ST_Buffer('POINT(2 2)::geometry, 1) @ ST_MakeBox2D(ST_Point(0, 0), ST_Point(5, 5)) ↔
AS is_contained;
```

Output

```
t
```

相关信息

`&&(geometry,box2df), &&(box2df,geometry), &&(box2df,box2df), ~(geometry,box2df), ~(box2df,geometry), ~(box2df,box2df), @(box2df,geometry), @(box2df,box2df)`

7.10.1.18 @ (box2df, geometry)

`@(box2df, geometry)` — 如果 2D 浮点精度 `BOX2DF` 包含在几何体的 2D `BOX2DF` 中，返回 `TRUE`。

Synopsis

```
boolean @( box2df A , geometry B );
```

描述

如果 2D `BOX2DF` A 使用浮点精度包含在 B 几何体的 2D `BOX2DF` 中，`@` 算符返回 `TRUE`。这意味着如果 B 是（双精度）`box2d`，它将在内部 `BOX2DF` 浮点精度 2D `BOX2DF`。

**Note**

`@` 操作数旨在由 BRIN 索引内部使用，而不是由用户使用。

可用性：2.3.0 - 引入了 `BRIN` 的支持。



此方法支持 `LINESTRING` 和 `CURVE`。



`@` 函数支持多面体曲面。

示例

Code

```
SELECT ST_MakeBox2D(ST_Point(2, 2), ST_Point(3, 3)) @ ST_Buffer('POINT(1 1)::geometry, 10) ↔
AS is_contained;
```

Output

```
t
```


相关信息

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

7.10.1.19 @ (box2df,box2df)

@(box2df,box2df) — 如果 2D 浮点精度 [BOX2DF](#) 包含在 [BOX2DF](#) 内，返回 TRUE。

Synopsis

```
boolean @( box2df A , box2df B );
```

描述

如果 2D [BOX2DF](#) A 包含在 2D [BOX2DF](#) B 中，[@](#) 算符使用浮点精度返回 TRUE。[@](#) 意味着如果 A（或 B）是（双精度）[box2d](#)，它将在内部 [BOX2DF](#) 浮点精度 2D [BOX2DF](#)。



Note

[@](#) 操作数旨在由 BRIN 索引内部使用，而不是由用 [@](#) 使用。

可用性：2.3.0 - 引入了 [@](#) BRIN 的支持。



此方法支持 [BOX2DF](#) 字符串和 [BOX2DF](#)。



[@](#) 函数支持多面体曲面。

示例

Code

```
SELECT ST_MakeBox2D(ST_Point(2, 2), ST_Point(3, 3)) @ ST_MakeBox2D(ST_Point(0, 0), ST_Point(5, 5)) AS is_contained;
```

Output

```
t
```

相关信息

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

7.10.1.20 |&>

|&> — 如果 A 的 [BOX2DF](#) 与 B 的 [BOX2DF](#) 重叠或位于其上方，返回 TRUE。

Synopsis

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

描述

`|&>` 算符 返回 `TRUE`，如果几何体 A 的边界框与几何体 B 的边界框重叠或位于其上方，更准确地说，它与几何体 B 的边界框不重叠于其下方。



Note

`|&>` 操作符将利用几何上可能可用的任何索引。

示例

Code

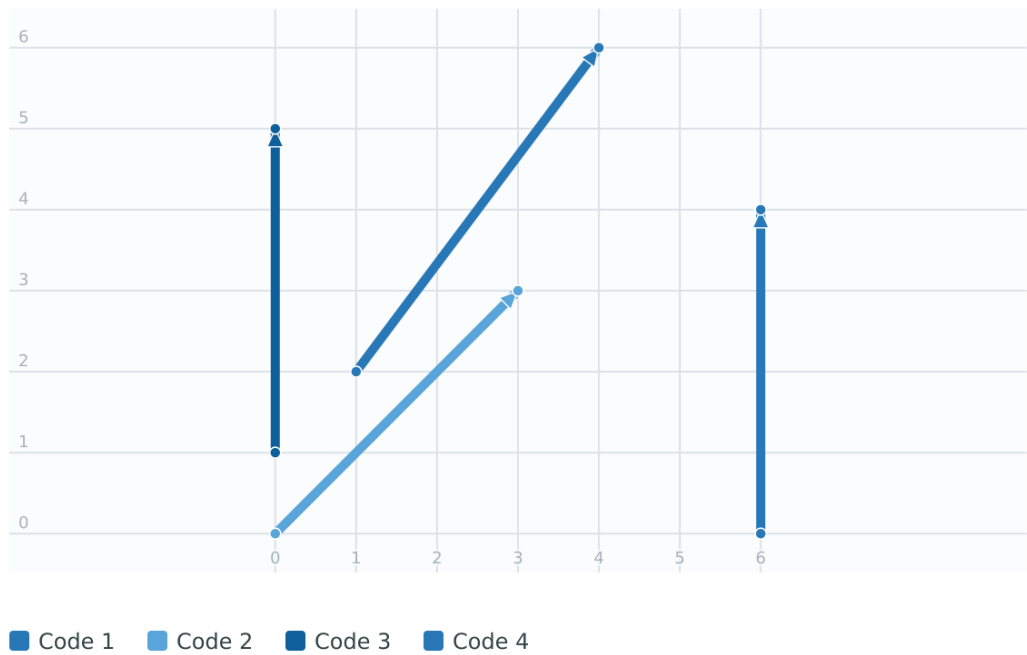
```
SELECT tbl1.column1, tbl2.column1, tbl1.column2 |&
> tbl2.column2 AS overabove
FROM
  ( VALUES
    (1, 'LINESTRING(6 0,6 4)::geometry)) AS tbl1,
  ( VALUES
    (2, 'LINESTRING(0 0,3 3)::geometry),
    (3, 'LINESTRING(0 1,0 5)::geometry),
    (4, 'LINESTRING(1 2,4 6)::geometry)) AS tbl2;
```

Output

column1	column1	overabove
1	2	t
1	3	f
1	4	f

(3 rows)

Figure



相关信息

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

7.10.1.21 |>>

|>> — 如果 A 的☐界框☐格位于 B 的☐界框上方，☐返回 TRUE。

Synopsis

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

描述

如果几何体 A 的☐界框☐格位于几何体 B 的☐界框上方，☐ |>> ☐算符返回 TRUE。



Note
☐操作符将利用几何上可能可用的任何索引。

示例

Code

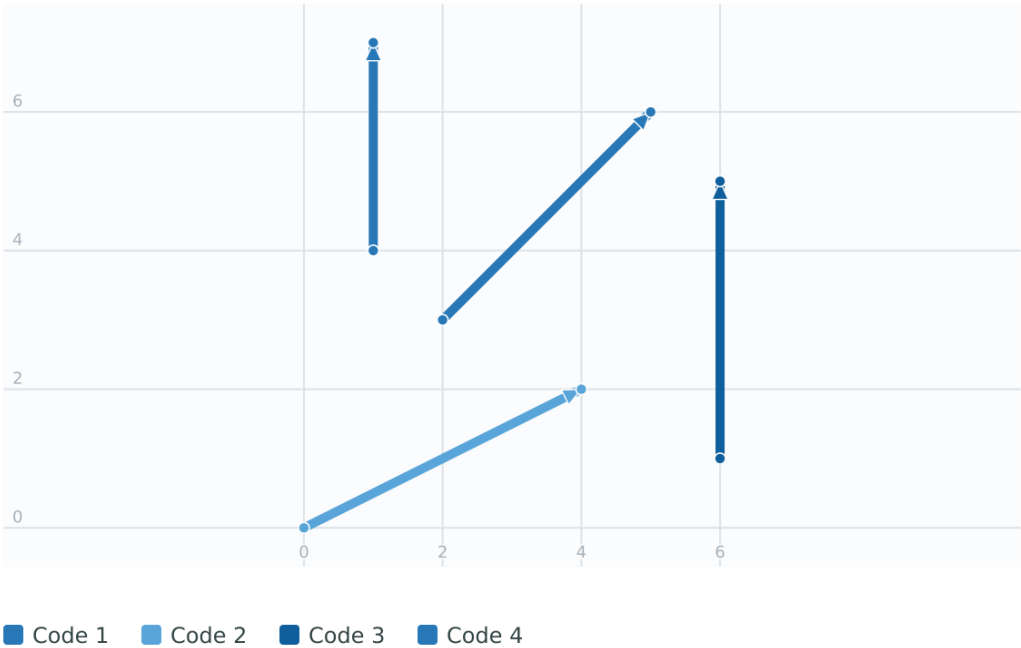
```
SELECT tbl1.column1, tbl2.column1, tbl1.column2 |>> tbl2.column2 AS above
FROM
  ( VALUES
    (1, 'LINESTRING (1 4,1 7)')::geometry)) AS tbl1,
  ( VALUES
    (2, 'LINESTRING (0 0,4 2)')::geometry),
    (3, 'LINESTRING (6 1,6 5)')::geometry),
    (4, 'LINESTRING (2 3,5 6)')::geometry)) AS tbl2;
```

Output

column1	column1	above
1	2	t
1	3	f
1	4	f

(3 rows)

Figure



相关信息

<<, >>, <<|

7.10.1.22 ~

~ — 如果 A 的☐界框包含 B 的☐界框，☐返回 TRUE。

Synopsis

boolean ~(geometry A , geometry B);

描述

如果几何体 A 的☒界框完全包含几何体 B 的☒界框，☒ ~ ☒算符返回 TRUE。



Note
☒操作符将利用几何上可能可用的任何索引。

示例

Code

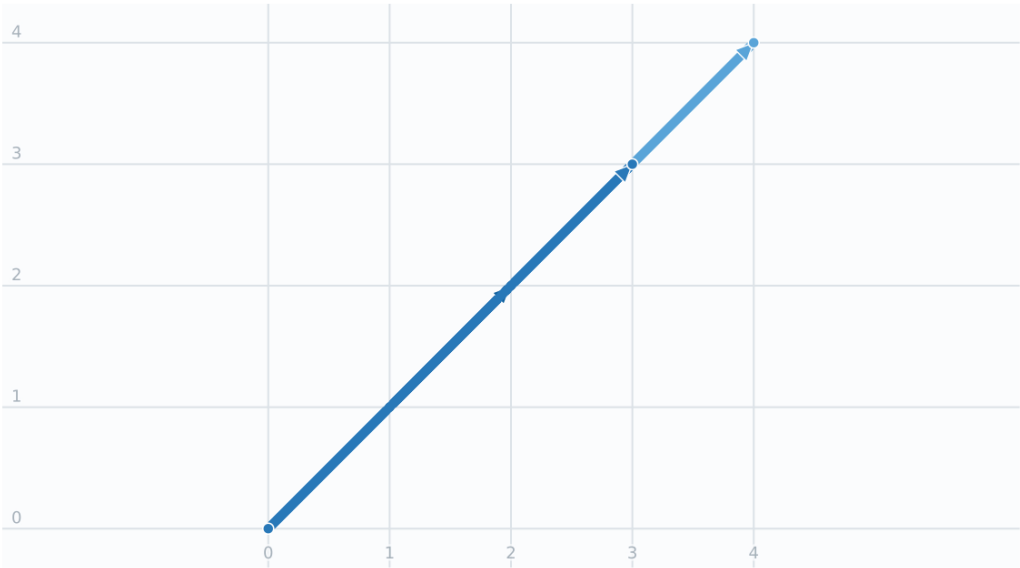
```
SELECT tbl1.column1, tbl2.column1, tbl1.column2 ~ tbl2.column2 AS contains
FROM
  ( VALUES
    (1, 'LINESTRING (0 0,3 3)::geometry)) AS tbl1,
  ( VALUES
    (2, 'LINESTRING (0 0,4 4)::geometry),
    (3, 'LINESTRING (1 1,2 2)::geometry),
    (4, 'LINESTRING (0 0,3 3)::geometry)) AS tbl2;
```

Output

column1	column1	contains
1	2	f
1	3	t
1	4	t

(3 rows)

Figure



Code 1 Code 2 Code 3 Code 4

相关信息

[@](#), [&&](#)

7.10.1.23 ~(**geometry**,**box2df**)

~(**geometry**,**box2df**) — 如果几何体的 2D 粘合框包含 2D 浮点精度框 (GIDX)，返回 TRUE。

Synopsis

```
boolean ~( geometry A , box2df B );
```

描述

如果几何形 A 的 2D 框包含 2D 框 B (使用浮点精度)，~ 算符返回 TRUE。意味着如果 B 是 (双精度) box2d，它将在内部框浮点精度 2D 框 (BOX2DF)



Note

操作数旨在由 BRIN 索引内部使用，而不是由用使用。

可用性：2.3.0 - 引入了 BRIN 的支持。



此方法支持形字符串和曲线。



函数支持多面体曲面。

示例

Code

```
SELECT ST_Buffer('POINT(1 1)::geometry', 10) ~ ST_MakeBox2D(ST_Point(0, 0), ST_Point(2, 2)) ←
AS contains;
```

Output

```
t
```

相关信息

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

7.10.1.24 ~(**box2df**,**geometry**)

~(**box2df**,**geometry**) — 如果 2D 浮点精度框 (BOX2DF) 包含几何体的 2D 框，返回 TRUE。

Synopsis

```
boolean ~( box2df A , geometry B );
```

描述

如果 2D $\square$ 界框 A 包含 B 几何体的 $\square$ 界框, $\square \sim \square$ 算符使用浮点精度返回 **TRUE**。 $\square$ 意味着如果 A 是 (双精度) box2d, 它将在内部 $\square \square \square$ 浮点精度 2D $\square$ 界框 (BOX2DF)



Note

$\square$ 操作数旨在由 BRIN 索引内部使用, 而不是由用 $\square$ 使用。

可用性 : 2.3.0 - 引入了 $\square$ BRIN 的支持。



此方法支持 $\square$ 形字符串和曲 $\square$ 。



$\square$ 函数支持多面体曲面。

示例

Code

```
SELECT ST_MakeBox2D(ST_Point(0, 0), ST_Point(5, 5)) ~ ST_Buffer('POINT(2 2)::geometry, 1) ←
      AS contains;
```

Output

```
t
```

相关信息

$\&\&(\text{geometry}, \text{box2df}), \&\&(\text{box2df}, \text{geometry}), \&\&(\text{box2df}, \text{box2df}), \sim(\text{geometry}, \text{box2df}), \sim(\text{box2df}, \text{box2df}),$
 $\text{@}(\text{geometry}, \text{box2df}), \text{@}(\text{box2df}, \text{geometry}), \text{@}(\text{box2df}, \text{box2df})$

7.10.1.25 $\sim(\text{box2df}, \text{box2df})$

$\sim(\text{box2df}, \text{box2df})$ — 如果 2D 浮点精度 $\square$ 界框 (BOX2DF) 包含 $\square$ 一个 2D 浮点精度 $\square$ 界框 (BOX2DF), $\square$ 返回 **TRUE**。

Synopsis

boolean $\sim(\text{box2df A}, \text{box2df B});$

描述

如果 2D 浮点精度 $\square$ 界框 A 包含 2D 浮点精度 $\square$ 界框 B, $\square \sim \square$ 算符返回 **TRUE**。 $\square$ 意味着如果 A 是 (双精度) box2d, 它将在内部 $\square \square \square$ 浮点精度 2D $\square$ 界框 (BOX2DF)



Note

$\square$ 操作数旨在由 BRIN 索引内部使用, 而不是由用 $\square$ 使用。

可用性 : 2.3.0 - 引入了 $\square$ BRIN 的支持。



此方法支持 $\square$ 形字符串和曲 $\square$ 。



$\square$ 函数支持多面体曲面。

示例

Code

```
SELECT ST_MakeBox2D(ST_Point(0, 0), ST_Point(5, 5)) ~ ST_MakeBox2D(ST_Point(2, 2), ST_Point(3, 3)) AS contains;
```

Output

```
t
```

相关信息

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

7.10.1.26 ~=

~= — 如果 A 的边界框与 B 的边界框相同，返回 TRUE。

Synopsis

boolean ~= (geometry A , geometry B);

描述

如果几何/地理 A 的边界框与几何/地理 B 的边界框相同， ~= 算符返回 TRUE。



Note

操作符将利用几何上可能可用的任何索引。

可用性：1.5.0 改进了行



函数支持多面体曲面。



Warning

算符已将 PostGIS 1.5 中的行从几何相等性更改为边界框相等性。事情变得复杂的是，它取决于您是否进行了硬升级或升级，数据中的行是什么。要了解您的数据有些行，您可以运行下面的查询。要检查是否真正相等，使用 [ST_OrderingEquals](#) 或 [ST_Equals](#)。

示例

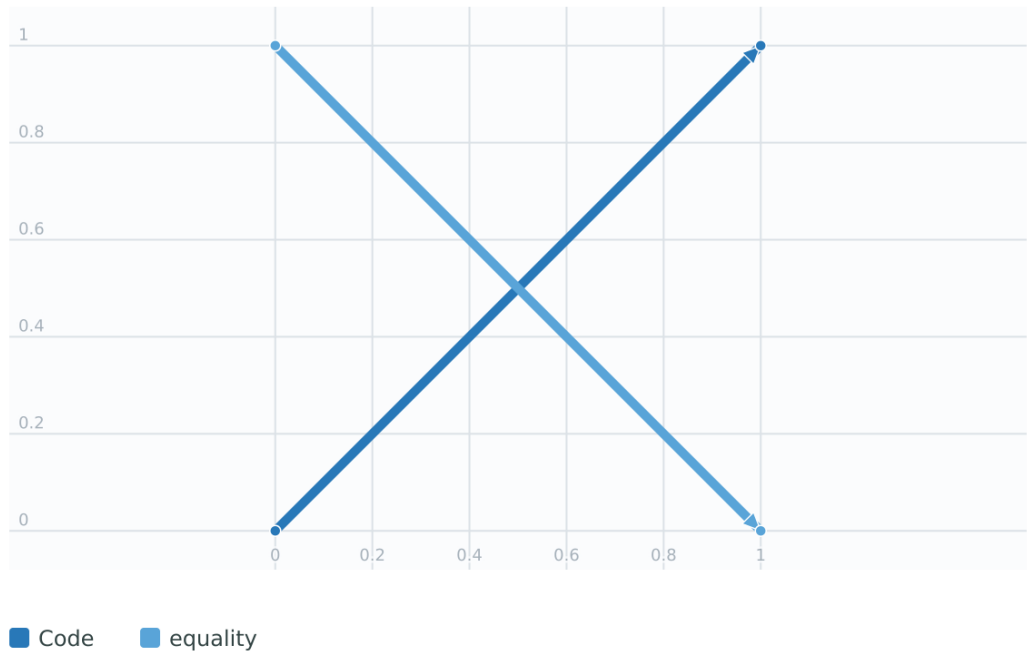
Code

```
SELECT 'LINESTRING(0 0,1 1)::geometry' ~= 'LINESTRING(0 1,1 0)::geometry' AS equality;
```


Output

t

Figure



相关信息

ST_Equals, ST_OrderingEquals, =

7.10.2 距离运算符

7.10.2.1 <->

<-> — 返回 A 和 B 之间的 2D 距离。

Synopsis

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

描述

<-> 运算符返回两个几何对象之间的二距离。用于 ORDER BY 子句，可以借助索引进行最近排序：geometry 行真正的 KNN 距离搜索，geography 返回球面距离。



Note
该操作符将利用几何上可能可用的 2D GiST 索引。它与其他使用空索引的运算符不同，只有当该运算符位于 ORDER BY 子句中才会使用空索引。



Note

☐ 当其中一个几何☐形是常量（不在子☐☐/cte 中）☐，索引才会☐☐。例如'SRID=3005;POINT(1011102 450541)>::geometry 而不是 a.geom

有关☐☐示例，☐参☐[PostGIS 研☐会：最近☐搜索](#)。

Enhanced: 2.2.0 -- 支持 geometry 和 geography 的真☐ KNN 行☐。注意 geography 的 KNN 基于球体而非☐球体。

Changed: 2.2.0 -- 在提供真☐ KNN 之后，旧的混合☐法☐通方式可能会更慢。参☐下方示例。

可用性：2.0.0 -- 弱 KNN 基于几何☐象☐心距离来找最近☐，而非真☐距离。☐点来☐☐果是准确的，☐其他☐型☐是近似的。

示例

Code

```
SELECT ST_Distance(geom, 'SRID=3005;POINT(1011102 450541)>::geometry) as d, edabbr, vaabbr
FROM va2005
ORDER BY d limit 10;
```

Output

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)

KNN 的原始答案是：

Code

```
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;
```

Output

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)

如果您运行“EXPLAIN ANALYZE”，您将看到第二个的性能有所提高。

当无法直接出精确距离，可以使用混合来取真正的最近。做法是先用 CTE 利用索引助的 KNN 得到候集，再做一次精确排序，流程如下：

Code

```
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;
```

Output

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)

相关信息

ST_DWithin, ST_Distance, <#>

7.10.2.2 |=|

|=| — 返回 A 和 B 迹在最接近点的距离。

Synopsis

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

描述

|=| 算符返回个迹的 3D 距离（参 ST_IsValidTrajectory）。它的作用等同于 ST_DistanceCPA，但作算符可以合 N 索引用于最近搜索。



Note

操作数将利用几何上可能可用的 ND GiST 索引。它与其他使用空索引的算符不同，只有当算符位于 ORDER BY 子句中才会使用空索引。



Note

☐ 当其中一个几何☐形是常量（不在子☐☐/cte 中）☐，索引才会☐☐。例如‘SRID=3005;LINESTRING(0 0,0 0 1)’::geometry 而不是 a.geom

可用性：2.2.0。

示例

This example orders stored trajectories by their distance from a literal query trajectory.

Code

```
\set qt 'ST_AddMeasure(ST_MakeLine(ST_MakePointM(-350, 300, 0), ST_MakePointM(-410, 490, 0) ↵
), 10, 20)'\n
SELECT track_id, dist FROM (\n
  SELECT track_id,\n
    ST_DistanceCPA(tr, :qt) dist\n
  FROM trajectories\n
  ORDER BY tr || :qt\n
  LIMIT 5\n
) foo;
```

Output

track_id	dist
395	0.576496831518066
380	5.06797130410151
390	7.72262293958322
385	9.8004461358071
405	10.9534397988433
(5 rows)	

相关信息

[ST_DistanceCPA](#), [ST_ClosestPointOfApproach](#), [ST_IsValidTrajectory](#)

7.10.2.3 <#>

<#> — 返回 A 和 B ☐界框之☐的 2D 距离。

Synopsis

double precision <#>(geometry A , geometry B);

描述

<#> ☐算符返回☐个浮点包☐盒之☐的距离。适用于☐行最近☐的 **approximate** 距离排序。



Note
☐操作符将利用几何上可能可用的任何索引。它与其他使用空☐索引的☐算符不同，只有当☐☐算符位于 ORDER BY 子句中☐才会使用空☐索引。



Note
☐当其中一个几何☐形是常量☐，索引才会生效，例如 ORDER BY (ST_GeomFromText('POINT(1 2)') <#> geom) 而不是 g1.geom <#> 。

可用性：2.0.0。

示例

Code

```
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;
```

Output

tlid	mtfcc	b_dist	act_dist
85732027	S1400	0	0
85732029	S1400	0	0
85732031	S1400	0	0
85734335	S1400	0	0
85736037	S1400	0	0
624683742	S1400	0	128.528874268666
85719343	S1400	260.839270432962	260.839270432962
85741826	S1400	164.759294123275	260.839270432962
85732032	S1400	277.75	311.830282365264
85735592	S1400	222.25	311.830282365264

(10 rows)

相关信息

ST_DWithin, ST_Distance, <->

7.10.2.4 <<->>

<<->> — 返回 A 和 B 几何☐形或☐界框之☐的 n ☐距离

Synopsis

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

描述

<<->> 运算符返回两个几何体之间的 n （欧几里德）距离。它于行最近近似距离排序很有用。



Note

操作数将利用几何上可能可用的 n -D GiST 索引。它与其他使用空索引的运算符不同，只有当运算符位于 ORDER BY 子句中才会使用空索引。



Note

当其中一个几何体是常量（不在子句/cte 中），索引才会。例如 'SRID=3005;POINT(1011102 450541)::geometry' 而不是 a.geom

可用性：2.2.0。

相关信息

<->

7.11 空关系

7.11.1 拓扑关系

7.11.1.1 ST_3DIntersects

ST_3DIntersects — 两个几何体在 3D 空间中是否相交 - 适用于点、串、多边形、多面体曲面（区域）

Synopsis

```
boolean ST_3DIntersects( geometry geomA , geometry geomB );
```

描述

重叠、接触、内在都意味着空相交。如果上述任何一个返回 true，几何体也在空上相交。不相交意味着空相交假。



Note

此功能自包括利用几何上可用的任何空索引的边界框。



Note
由于浮点数精度问题，几何处理后，几何体不会像是您期望的那样相交。例如，到几何体上的线串的最近点可能不位于线串上。于这种情况，如果您希望将一厘米的距离强制相交，可以使用 **ST_3DDWithin** 函数。

更改：3.0.0 除了 SFCGAL 后端，GEOS 后端支持 TIN。
可用性: 2.0.0

- ✓ 函数支持 3d 并且不会丢失 z-index。
- ✓ 函数支持多面体曲面。
- ✓ 此函数支持三角形和不规则三角网面 (TIN)。
- ✓ 方法符合了 SQL/MM 规范。SQL-MM IEC 13249-3: 5.1

几何示例

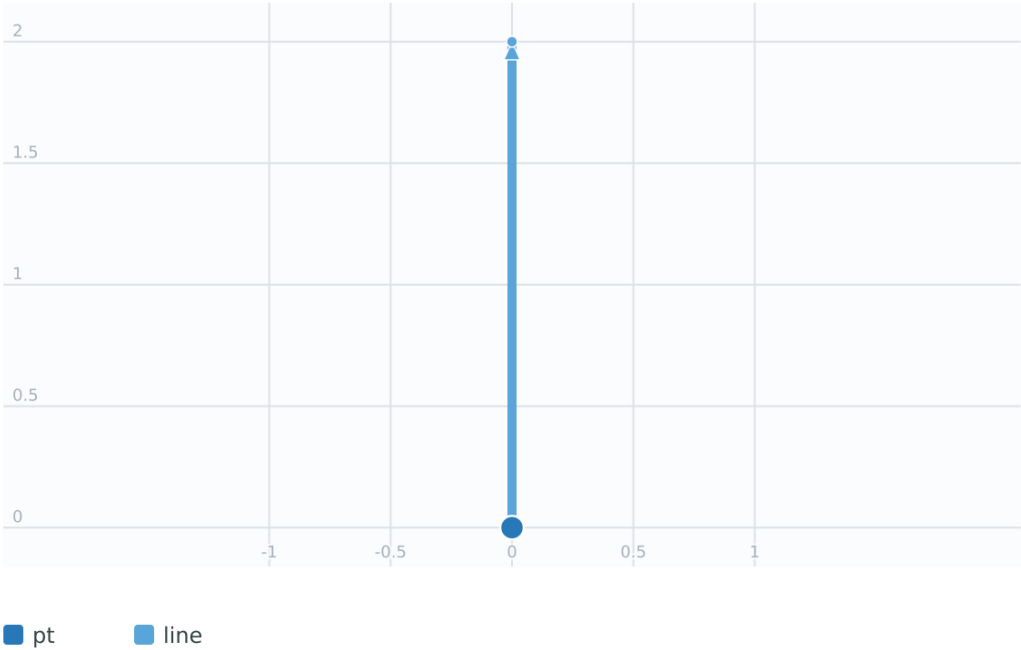
Code

```
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;
```

Output

st_3dintersects	st_intersects
f	t
(1 row)	

Figure



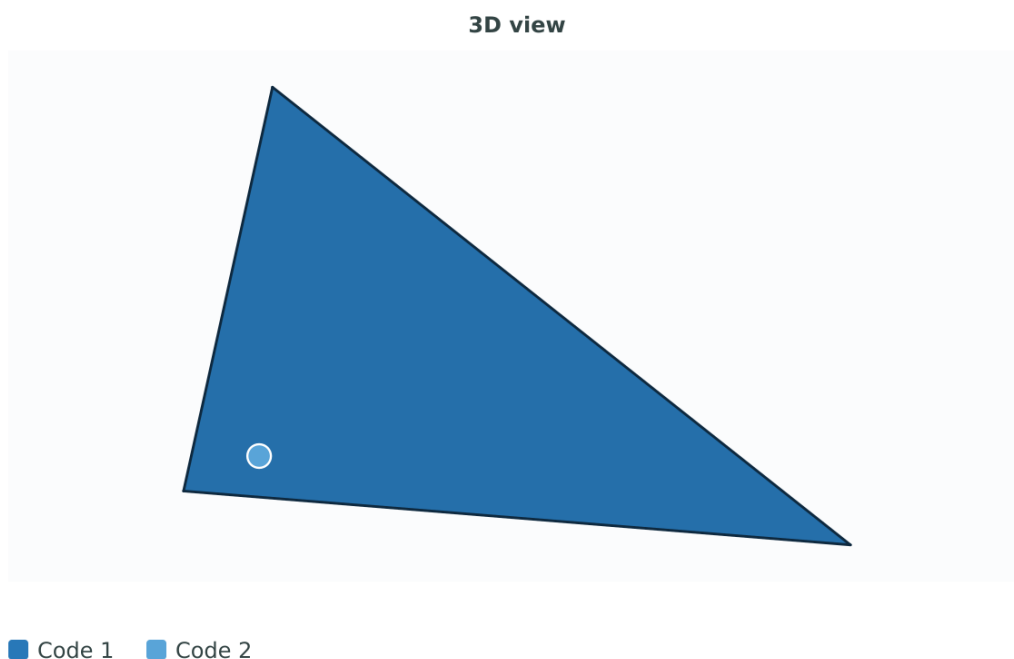
TIN 示例

Code

```
SELECT ST_3DIntersects('TIN(((0 0 0,1 0 0,0 1 0,0 0 0)))'::geometry, 'POINT(.1 .1 0)':: ←
      geometry);
```

Output

```
t
```

Figure

相关信息

[ST_3DDWithin](#), [ST_Intersects](#)

7.11.1.2 ST_Contains

ST_Contains — 是否 B 的每个点是否都位于 A 中，并且它的内部是否有一个共同点

Synopsis

boolean **ST_Contains**(geometry geomA, geometry geomB);

描述

如果几何图形 A 包含几何图形 B，则返回 TRUE。当且仅当 B 的所有点都位于 A 内部（即在 A 的内部或边界中），A 包含 B（或者等效地，B 中没有点位于 A 的外部），并且 A 和 B 的内部至少有一个共同点。

用数学公式来写： $ST_Contains(A, B) \Leftrightarrow (A \supset B = B) \wedge (Int(A) \cap Int(B) \neq \emptyset)$

包含关系是自反的：每个几何图形都包含其自身。（相反，在 [ST_ContainsProperly](#) 中，几何图形未正确包含自身。）该关系是反称的：如果 $ST_Contains(A, B) = true$ 且 $ST_Contains(B, A) = true$ ，则这两个几何图形必在拓扑上相等 ($ST_Equals(A, B) = true$)。

ST_Contains 与 [ST_Within](#) 相反。因此， $ST_Contains(A, B) = ST_Within(B, A)$ 。

**Note**

因☐内部必☐有一个公共点，所以定☐的一个微妙之☐是多☐形和☐不包含完全位于其☐界内的☐和点。有关更多☐☐信息，☐参☐ [OGC 涵盖、包含、内部的微妙之☐](#)。[ST_Covers](#) ☐☐提供了更具包容性的关系。

**Note**

此功能自☐包括利用几何上可用的任何空☐索引的☐界框比☐。要避免使用索引，☐使用函数 `_ST_Contains`。

它是通☐ GEOS 模☐☐☐的

增☐：2.3.0 PIP 短路（快速判断☐限于多☐形和点）已得到增☐，以支持具有更少点的多点。以前的版本☐支持面和点☐合。

增☐：3.0.0 ☐用了☐ GEOMETRYCOLLECTION 的支持

**Important**

☐勿将此函数用于无效的几何☐形。你会得到意想不到的☐果。

注意：☐是返回布☐☐而不是整数的“允☐”版本。



此方法☐☐了 [SQL 1.1 的 OGC ☐☐功能☐范](#)。s2.1.1.2 // s2.1.13.3 - 与内部相同（几何 B、几何 A）



☐方法☐☐了 SQL/MM ☐范。SQL-MM 3: 5.1.31

示例

`ST_Contains` 在以下情况下返回 TRUE：

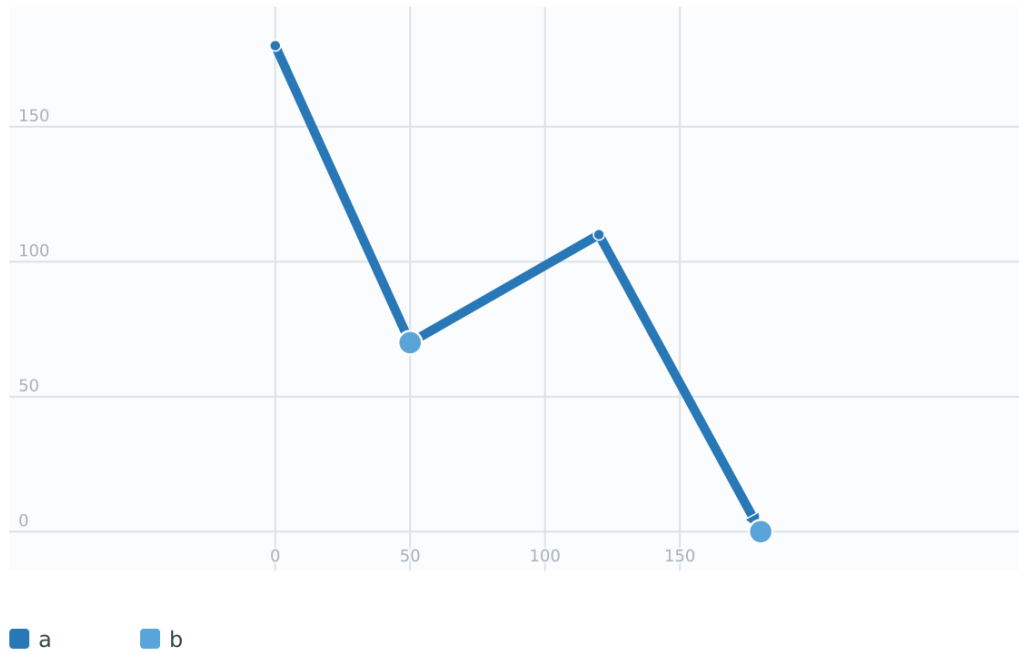
A LineString contains a MultiPoint whose points lie in its interior or boundary.

Code

```
SELECT ST_Contains(a, b)
FROM (SELECT
  'LINESTRING (10 190,60 80,130 120,190 10)::geometry AS a,
  'MULTIPOINT (60 80,190 10)::geometry AS b
) AS example;
```

Output

```
t
```

Figure

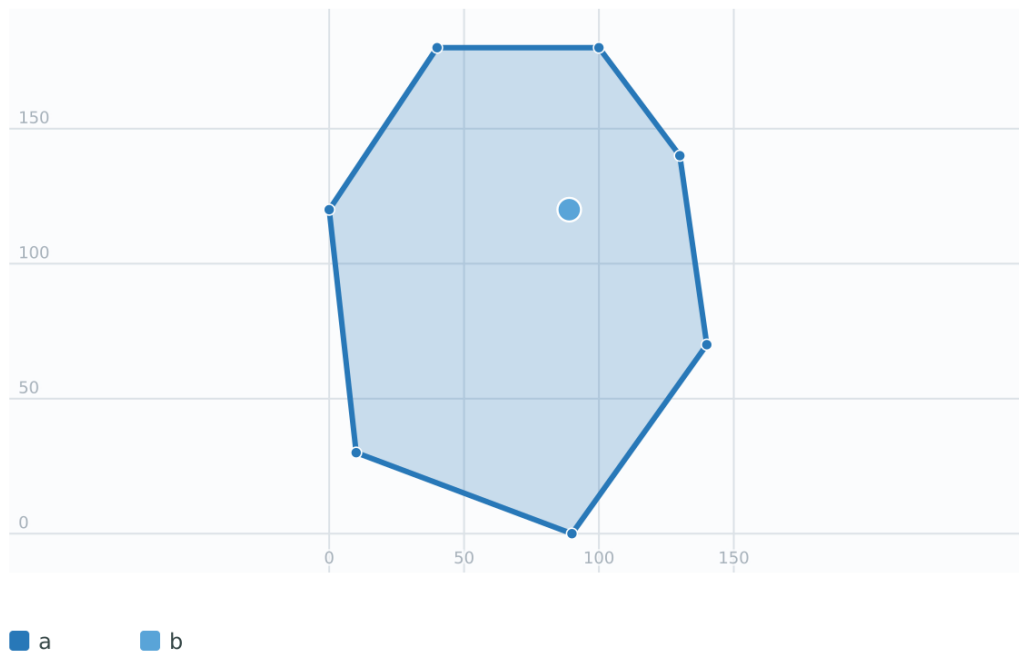
A Polygon contains a Point in its interior.

Code

```
SELECT ST_Contains(a, b)
FROM (SELECT
  'POLYGON ((10 130,50 190,110 190,140 150,150 80,100 10,20 40,10 130))'::geometry AS a,
  'POINT (99 130)'::geometry AS b
) AS example;
```

Output

t

Figure

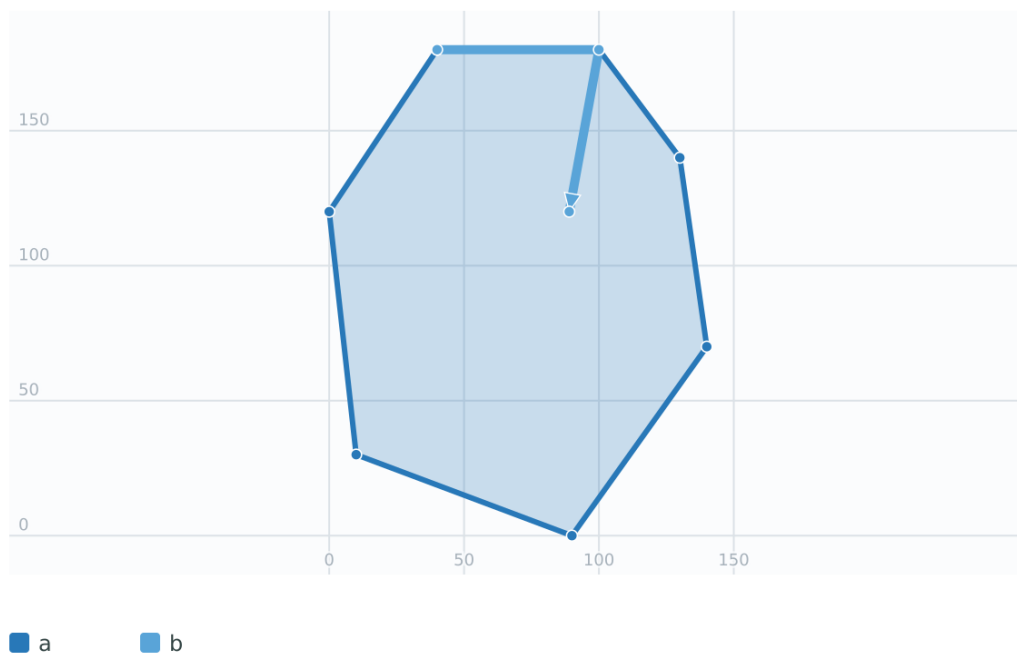
A Polygon contains a LineString with interior points inside the Polygon.

Code

```
SELECT ST_Contains(a, b)
FROM (SELECT
  'POLYGON ((10 130,50 190,110 190,140 150,150 80,100 10,20 40,10 130))'::geometry AS a,
  'LINESTRING (50 190,110 190,99 130)'::geometry AS b
) AS example;
```

Output

t

Figure

A Polygon contains another Polygon in its interior.

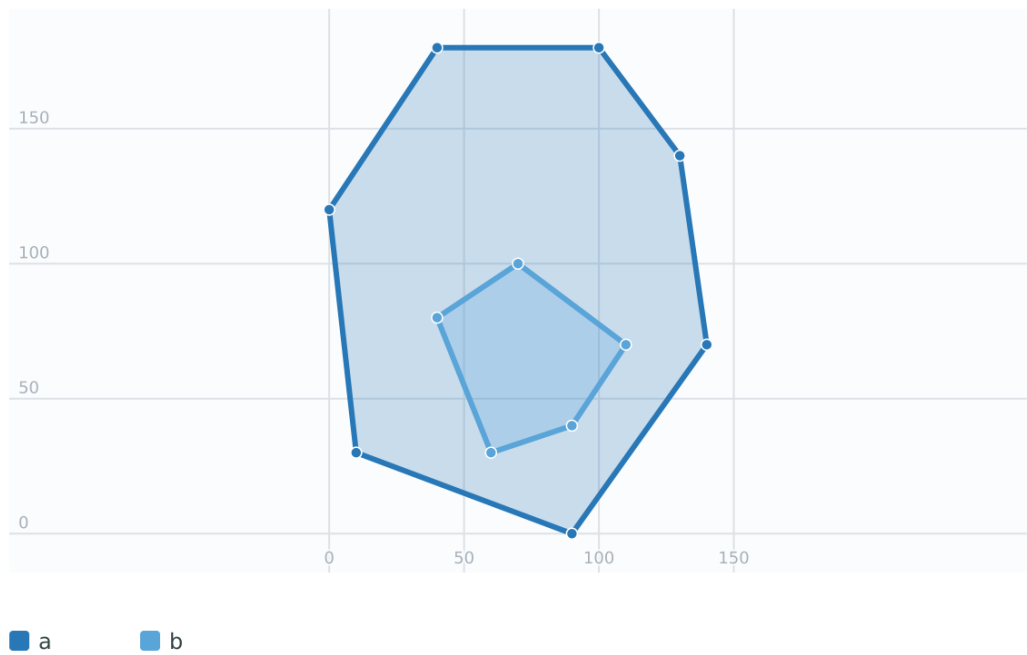
Code

```
SELECT ST_Contains(a, b)
FROM (SELECT
  'POLYGON ((10 130,50 190,110 190,140 150,150 80,100 10,20 40,10 130))'::geometry AS a,
  'POLYGON ((70 40,100 50,120 80,80 80 110,50 90,70 40))'::geometry AS b
) AS example;
```

Output

t

Figure



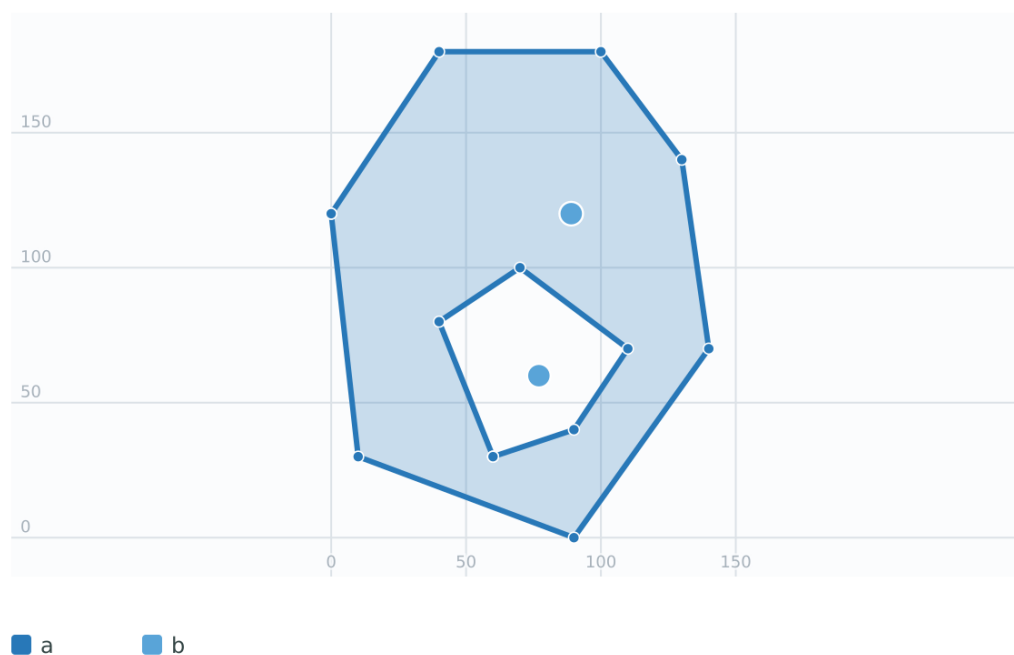
ST_Contains 在以下情况下返回 FALSE :
A Polygon does not contain a MultiPoint when one point lies in a hole.

Code

```
SELECT ST_Contains(a, b)
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 a,
  'MULTIPOINT (99 130,87 70)'::geometry AS b
) AS example;
```

Output

f

| Figure

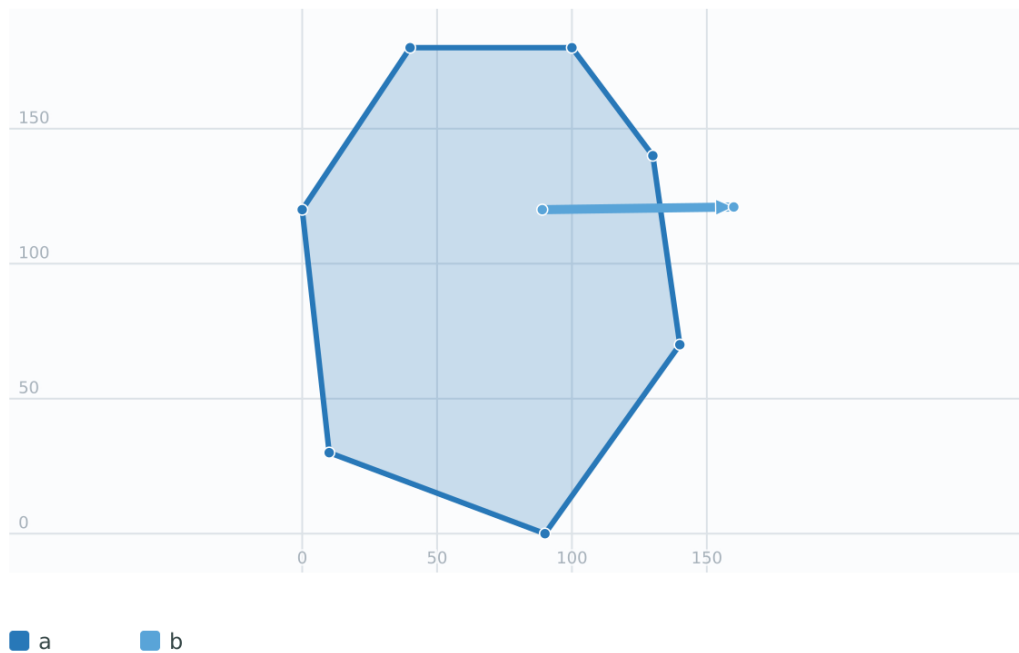
A Polygon does not contain a LineString extending outside it.

| Code

```
SELECT ST_Contains(a, b)
FROM (SELECT
  'POLYGON ((10 130,50 190,110 190,140 150,150 80,100 10,20 40,10 130))'::geometry AS a,
  'LINESTRING (99 130,170 131)'::geometry AS b
) AS example;
```

| Output

f

Figure

由于内部相交条件，`ST_Contains` 在以下情况下返回 `FALSE`（而 `ST_Covers` 返回 `TRUE`）：

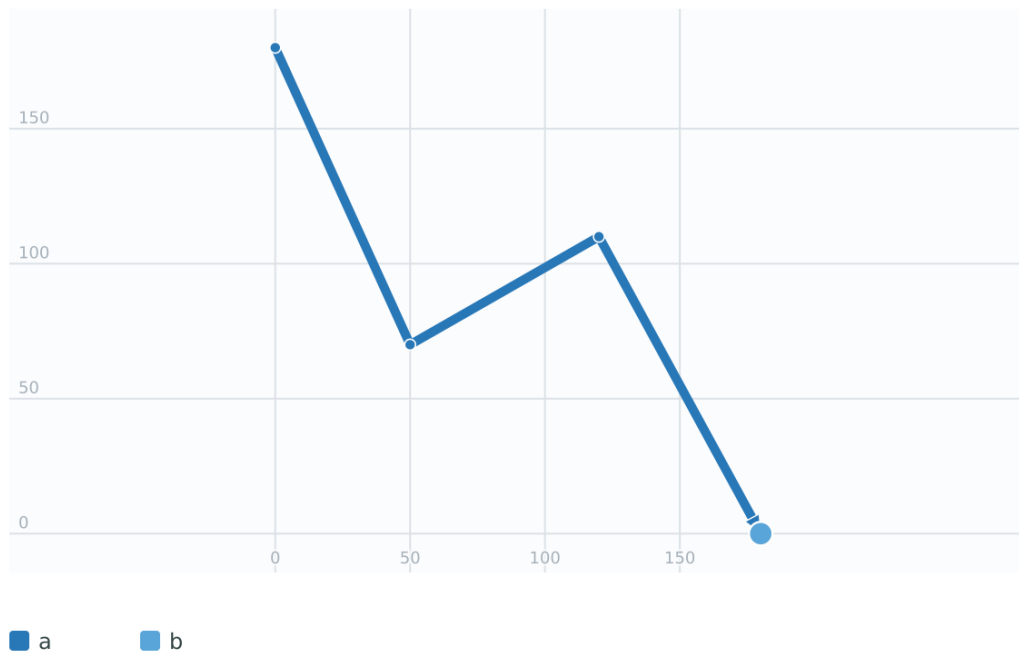
A LineString does not contain a Point lying only on its boundary.

Code

```
SELECT ST_Contains(a, b), ST_Covers(a, b)
FROM (SELECT
  'LINESTRING (10 190,60 80,130 120,190 10)::geometry AS a,
  'POINT (190 10)::geometry AS b
) AS example;
```

Output

f | t

Figure

A Polygon does not contain a LineString lying entirely on its boundary.

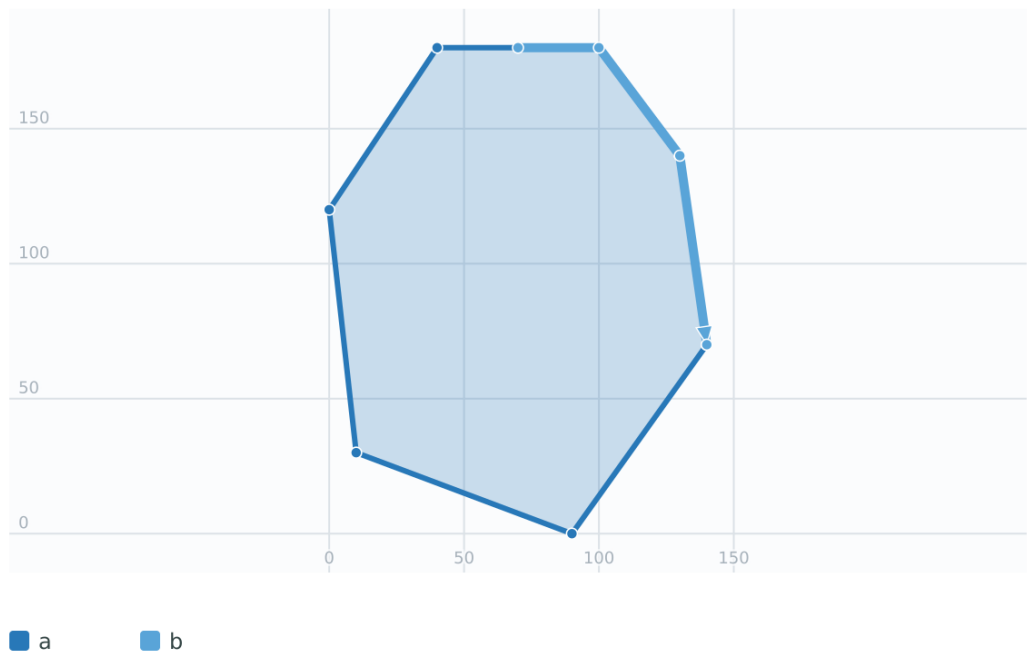
Code

```
SELECT ST_Contains(a, b), ST_Covers(a, b)
FROM (SELECT
  'POLYGON ((10 130,50 190,110 190,140 150,150 80,100 10,20 40,10 130))'::geometry AS a,
  'LINESTRING (80 190,110 190,140 150,150 80)'::geometry AS b
) AS example;
```

Output

f | t

Figure



This example tests a circle within another circle.

Code

```
WITH circles AS (  
  SELECT ST_Buffer(center, 10) AS smallc,  
         ST_Buffer(center, 20) AS bigc  
  FROM (SELECT 'POINT(1 2)::geometry AS center) AS p  
)  
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 circles;
```

Output

```
-[ RECORD 1 ]-----  
smallcontainsbig | f  
bigcontainssmall | t  
bigcontainsunion | t  
bigisunion       | t  
bigcoversexterior | t  
bigcontainsexterior | f
```

Figure



This example demonstrates the difference between contains and contains-properly predicates.

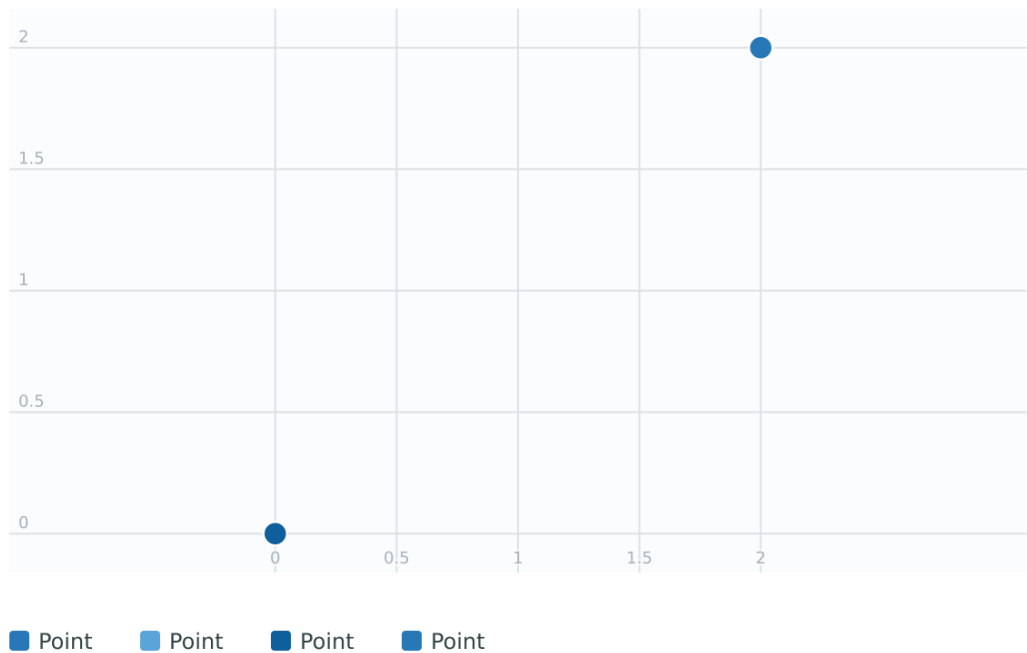
Code

```
SELECT
  ST_GeometryType(geom_a) AS geomtype,
  ST_Contains(geom_a, geom_a) AS acontainsa,
  ST_ContainsProperly(geom_a, geom_a) AS acontainspropa,
  ST_Contains(geom_a, ST_Boundary(geom_a)) AS acontainsba,
  ST_ContainsProperly(geom_a, ST_Boundary(geom_a)) 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(geom_a);
```

Output

geomtype	acontainsa	acontainspropa	acontainsba	acontainspropba
ST_Polygon	t	f	f	f
ST_LineString	t	f	f	f
ST_Point	t	t	f	f

Figure



相关信息

[ST_Boundary](#), [ST_ContainsProperly](#), [ST_Covers](#), [ST_CoveredBy](#), [ST_Equals](#), [ST_Within](#)

7.11.1.3 ST_ContainsProperly

`ST_ContainsProperly` — 检查 B 的每个点是否都位于 A 的内部

Synopsis

boolean **ST_ContainsProperly**(geometry geomA, geometry geomB);

描述

如果 B 的每个点都位于 A 的内部（或者等效地，B 的任何点都不位于 A 的边界或外部），则返回 `true`。
用数学公式来写： $ST_ContainsProperly(A, B) \Leftrightarrow Int(A) \supset B = B$
如果一个几何图形的 DE-9IM 交集矩阵匹配 `[T**FF*FF*]`，则 A 正确包含 B
A 没有正确地包含自身，但确实包含自身。
计算一个几何图形与大型多边形几何图形的交集时，使用此函数。交叉点相当慢，因此您可以使用 `ContainsProperly` 提取完全位于区域内的感兴趣的几何图形，非常高效。在某些场景中，直接地看出交集确实是原始目标几何体。



Note
此功能自 3.1 起包括利用几何上可用的任何空间索引的边界框比。要避免使用索引，请使用函数 `_ST_ContainsProperly`。



Note

相交于 **ST_Contains** 和 **ST_Intersects** 的点是可以更有效地计算，无需计算各个点的拓扑。

这个函数是由 GEOS 模块执行的。

可用性：1.4.0

增强：3.0.0 用了 GEOMETRYCOLLECTION 的支持



Important

勿将此函数用于无效的几何图形。你会得到意想不到的结果。

示例

This example tests a circle within another circle.

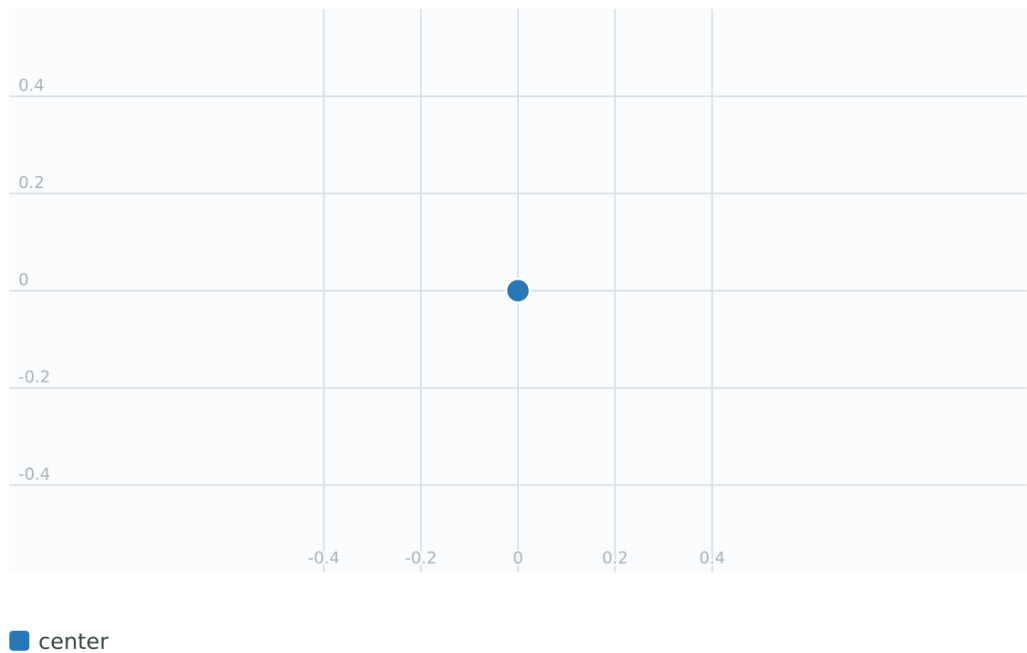
Code

```
WITH circles AS (  
  SELECT ST_Buffer(center, 10) AS smallc,  
         ST_Buffer(center, 20) AS bigc  
  FROM (SELECT 'POINT(1 2)::geometry AS center) AS p  
)  
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 circles;
```

Output

```
-[ RECORD 1 ]-----  
smallcontainspropbig | f  
bigcontainspropsmall | t  
bigcontainspropunion | f  
bigisunion           | t  
bigcoversexterior    | t  
bigcontainsexterior  | f
```

| **Figure**



This example demonstrates the difference between contains and contains-properly predicates.

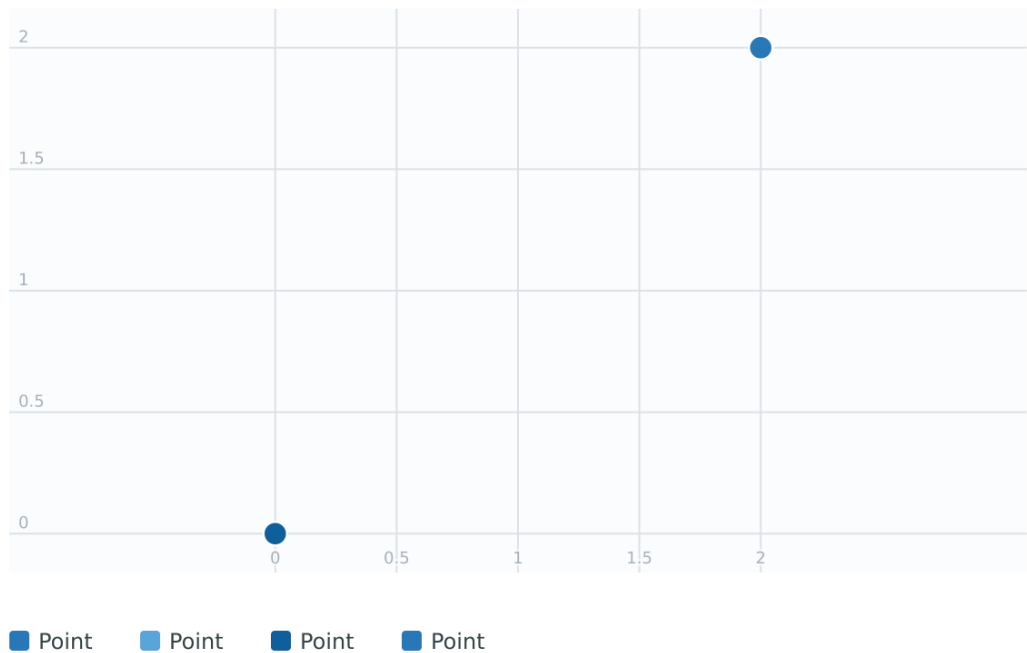
| **Code**

```
WITH input(geom) AS (VALUES
  (ST_Buffer(ST_Point(1, 1), 5, 1)),
  (ST_MakeLine(ST_Point(1, 1), ST_Point(-1, -1))),
  (ST_Point(1, 1))
)
SELECT ST_GeometryType(geom) AS geomtype,
       ST_Contains(geom, geom) AS acontainsa,
       ST_ContainsProperly(geom, geom) AS acontainspropa,
       ST_Contains(geom, ST_Boundary(geom)) AS acontainsba,
       ST_ContainsProperly(geom, ST_Boundary(geom)) AS acontainspropba
FROM input;
```

| **Output**

geomtype	acontainsa	acontainspropa	acontainsba	acontainspropba
ST_Polygon	t	f	f	f
ST_LineString	t	f	f	f
ST_Point	t	t	f	f

Figure



相关信息

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

7.11.1.4 ST_CoveredBy

ST_CoveredBy — 是否 A 的每个点是否都位于 B 中

Synopsis

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

描述

如果几何/地理 A 中的每个点都位于几何/地理 B 的内部（即与其内部或边界相交），则返回 true。否则，如果 A 是否没有位于 B 之外的点。

用数学公式来写： $ST_CoveredBy(A, B) \Leftrightarrow A \sqsubset B = A$

ST_CoveredBy 与 ST_Covers 相反。因此， $ST_CoveredBy(A, B) = ST_Covers(B, A)$ 。

一般来说，建议使用此函数而不是 ST_Within，因为它具有更严格的定义，不存在“边界不在其几何形状内”的奇怪情况。

 **Note**

此功能自 2.2.0 起包括利用几何上可用的任何空间索引的边界框比。要避免使用索引，请使用函数 `_ST_CoveredBy`。

增修：3.0.0 修用了 GEOMETRYCOLLECTION 的支持



Important

勿将此函数用于无效的几何形。你会得到意想不到的果。

它是通 GEOS 模的

可用性：1.2.2

注意：是返回布而不是整数的“允”版本。

不是 OGC 准，但 Oracle 也有。

示例

This example tests a circle covered by another circle.

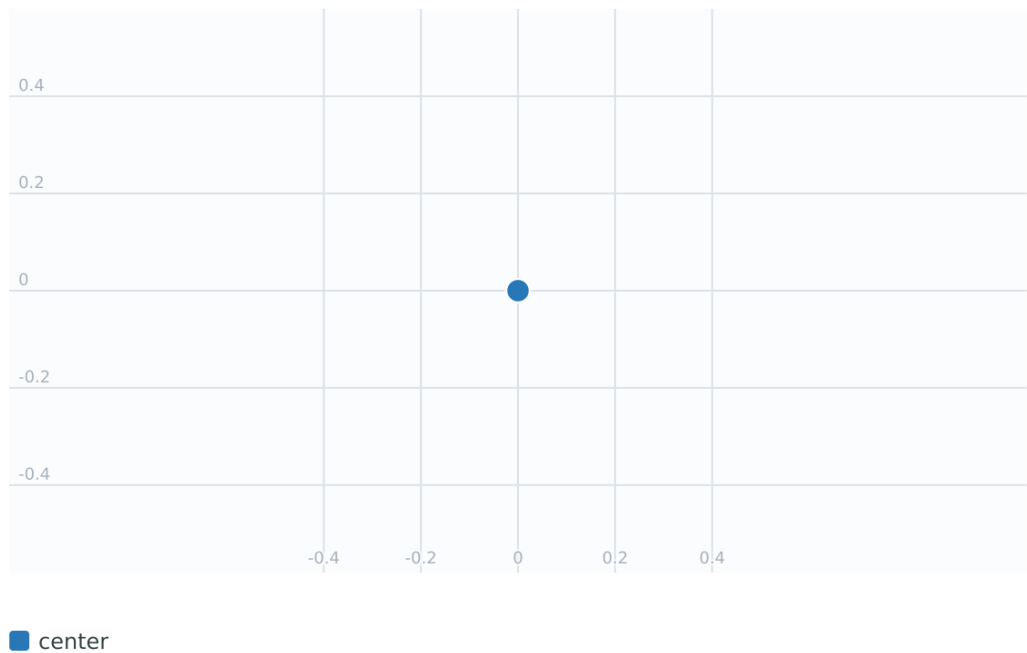
Code

```
WITH circles AS (  
  SELECT ST_Buffer(center, 10) AS smallc,  
         ST_Buffer(center, 20) AS bigc  
  FROM (SELECT 'POINT(1 2)::geometry AS center) AS p  
)  
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 exteriorwithinbig  
FROM circles;
```

Output

smallinsmall	smallcoveredbybig	exteriorcoveredbybig	exteriorwithinbig
t	t	t	f
(1 row)			

Figure



相关信息

[ST_Contains](#), [ST_Covers](#), [ST_ExteriorRing](#), [ST_Within](#)

7.11.1.5 ST_Covers

ST_Covers — 是否 B 的每个点是否都位于 A 中

Synopsis

```
boolean ST_Covers(geometry geomA, geometry geomB);
boolean ST_Covers(geography geogpolyA, geography geogpointB);
```

描述

如果几何/地理 B 中的每个点都位于几何/地理 A 内部（即与其内部或边界相交），则返回 `true`。同样，如果 B 中没有点位于 A 外部（外部）。

用数学符号表示： $ST_Covers(A, B) \Leftrightarrow A \sqsupset B = B$

ST_Covers 与 [ST_CoveredBy](#) 相反。因此， $ST_Covers(A, B) = ST_CoveredBy(B, A)$ 。

一般来说，使用 `ST_CoveredBy` 而不是 [ST_Contains](#)，因为它有一个更严格的定义，不存在“几何形状不包含其边界”的奇怪情况。



Note
此功能自 3.1 起包括利用几何上可用的任何空间索引的边界框比较。为了避免使用索引，使用函数 `_ST_Covers`。

增修：3.0.0 使用了 GEOMETRYCOLLECTION 的支持



Important

勿将此函数用于无效的几何图形。你会得到意想不到的结果。

它是通过 GEOS 模型实现的

增修：2.4.0 地理类型添加了多边形中的多边形和多边形中的圆的支持

增修：由于 2.3.0 几何图形，PIP 短路（限于多边形和点的快速判断）已得到增修，以支持由更少点组成的多点。以前的版本不支持面和点组合。

可用性：1.5 - 引入了地理支持。

可用性：1.2.2

注意：它是返回布尔值而不是整数的“允许”版本。

不是 OGC 标准，但 Oracle 也有。

示例

几何示例

This example tests a circle covering another circle.

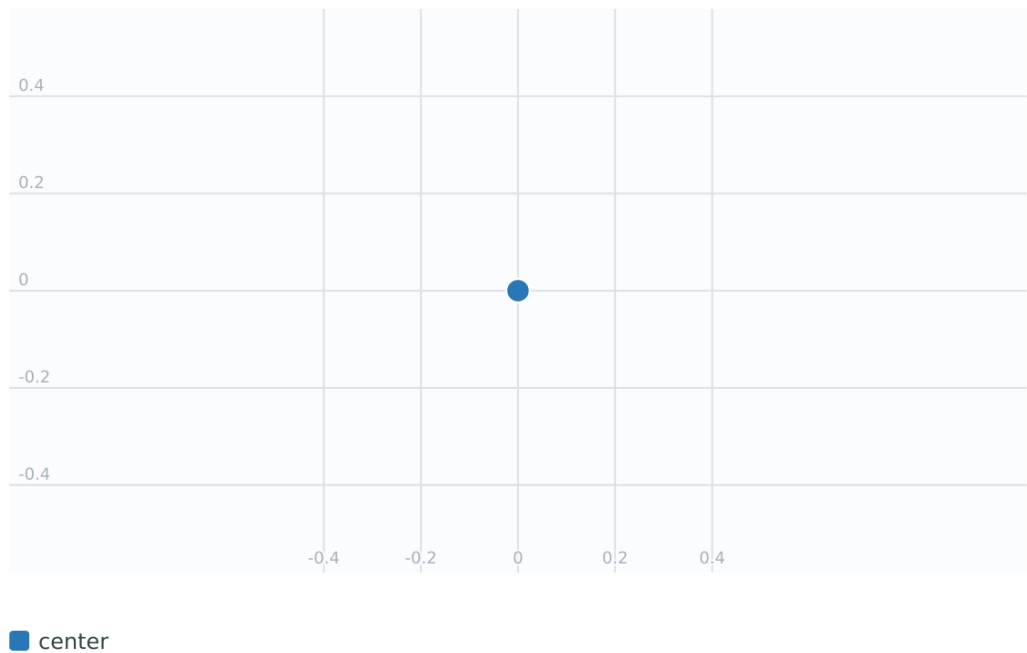
Code

```
WITH circles AS (  
  SELECT ST_Buffer(center, 10) AS smallc,  
         ST_Buffer(center, 20) AS bigc  
  FROM (SELECT 'POINT(1 2)::geometry' AS center) AS p  
)  
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 circles;
```

Output

smallinsmall	smallcoversbig	bigcoversexterior	bigcontainsexterior
t	f	t	f
(1 row)			

| **Figure**



Geography example

This example compares a point with a 300-meter buffer against a point and against a 10-meter buffer around that point.

| **Code**

```
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('SRID=4326;POINT(-99.327 31.4821) '::geography, 300) As geog_poly,
       'SRID=4326;POINT(-99.33 31.483) '::geography As geog_pt ) As foo;
```

| **Output**

poly_covers_pt	buff_10m_covers_cent
f	t

相关信息

[ST_Contains](#), [ST_CoveredBy](#), [ST_Within](#)

7.11.1.6 ST_Crosses

ST_Crosses — 两个几何图形是否有一些（但不是全部）共同的内点

Synopsis

boolean **ST_Crosses**(geometry g1, geometry g2);

描述

比较两个几何对象，如果它们的交集“空交叉”，则返回 **true**；也就是说，几何形状有一些但不是所有的内部点是共同的。几何形状内部的交集必须非空，并且维度必须小于每个输入几何形状的最大维度，并且两个几何形状的交集不得等于任一几何形状。否则，返回 **false**。交叉关系是称且非自反的。

用数学公式表示： $ST_Crosses(A, B) \Leftrightarrow (dim(Int(A) \cap Int(B)) < \max(dim(Int(A)), dim(Int(B))) \wedge (A \cap B \neq A) \wedge (A \cap B \neq \bar{B}))$

如果 DE-9IM 交集矩阵匹配，几何形状交叉：

- **T*T******* 适用于点/线、点/面和线/面情况
- **T***T**** 用于线/点、区域/点和区域/线情况
- **0******* 用于线/线情况
- 线于点/点和区域/区域情况，结果是 **false**



Note

OpenGIS 规范要素规范线/点、点/面、线/线和线/面情况定义此函数。JTS / GEOS 扩展了定义，也适用于线/点、区域/点和区域/线情况。这使得关系称。



Note

此功能自包括利用几何上可用的任何空索引的边界框比较。

增强：3.0.0 使用了 GEOMETRYCOLLECTION 的支持



此方法符合 SQL 1.1 的 OGC 功能规范。s2.1.13.3



此方法符合 SQL/MM 规范。SQL-MM 3: 5.1.29

示例

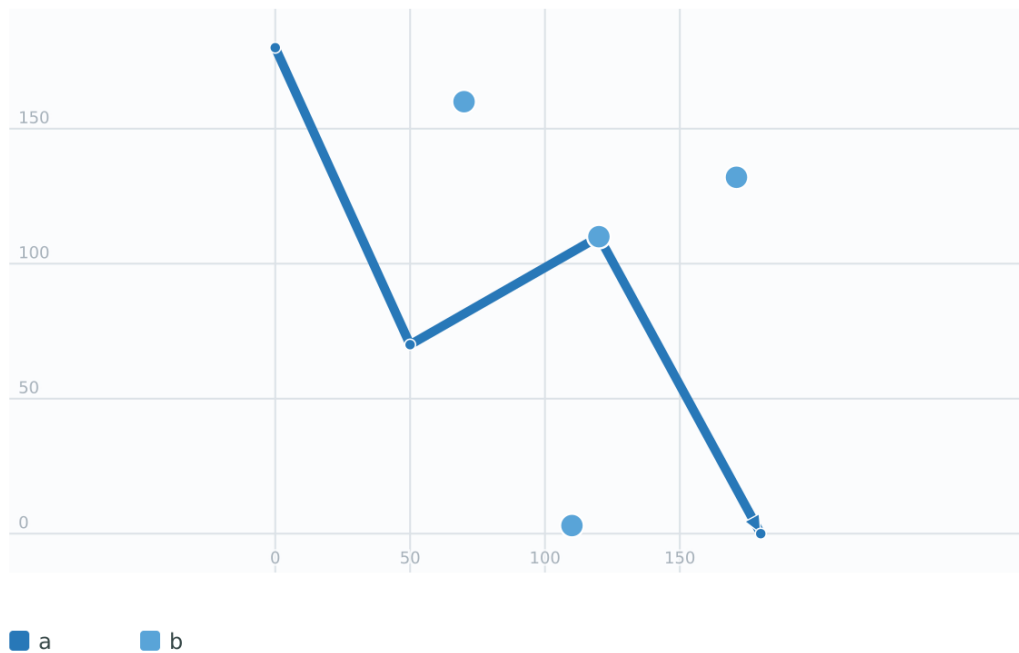
以下情况均返回 **true**。

A LineString crosses a MultiPoint when some, but not all, points lie on the line.

Code

```
SELECT ST_Crosses(a, b)
FROM (SELECT
  'LINESTRING (10 190,60 80,130 120,190 10)::geometry AS a,
  'MULTIPOINT (80 170,120 13,130 120,181 142)::geometry AS b
) AS example;
```

Figure

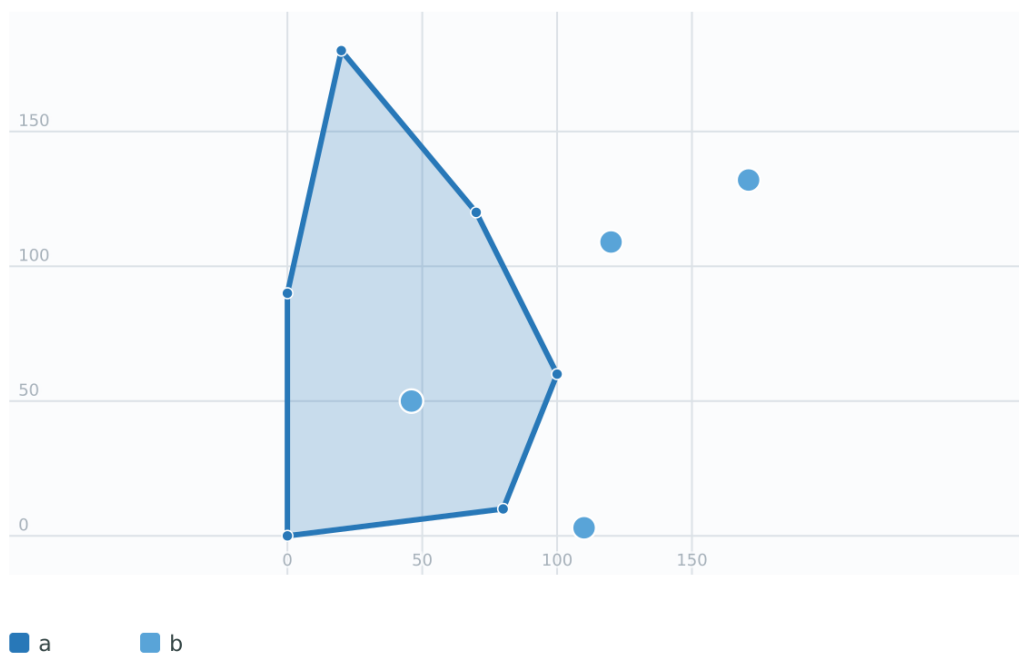


A Polygon crosses a MultiPoint when some, but not all, points lie in its interior.

Code

```
SELECT ST_Crosses(a, b)
FROM (SELECT
  'POLYGON ((30 190,10 100,10 10,90 20,110 70,80 130,30 190))'::geometry AS a,
  'MULTIPOINT (56 60,120 13,130 119,181 142)'::geometry AS b
) AS example;
```

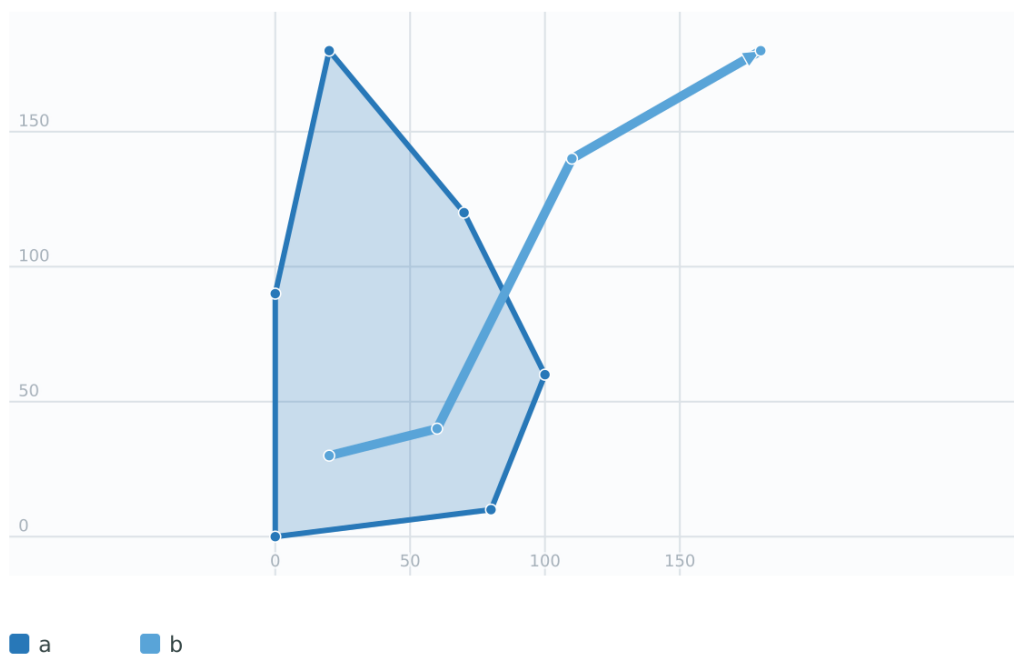
Figure



A LineString crosses a Polygon when part of the line lies inside and part outside.

Code

```
SELECT ST_Crosses(a, b)
FROM (SELECT
  'POLYGON ((30 190,10 100,10 10,90 20,110 70,80 130,30 190))'::geometry AS a,
  'LINESTRING (30 40,70 50,120 150,190 190)'::geometry AS b
) AS example;
```

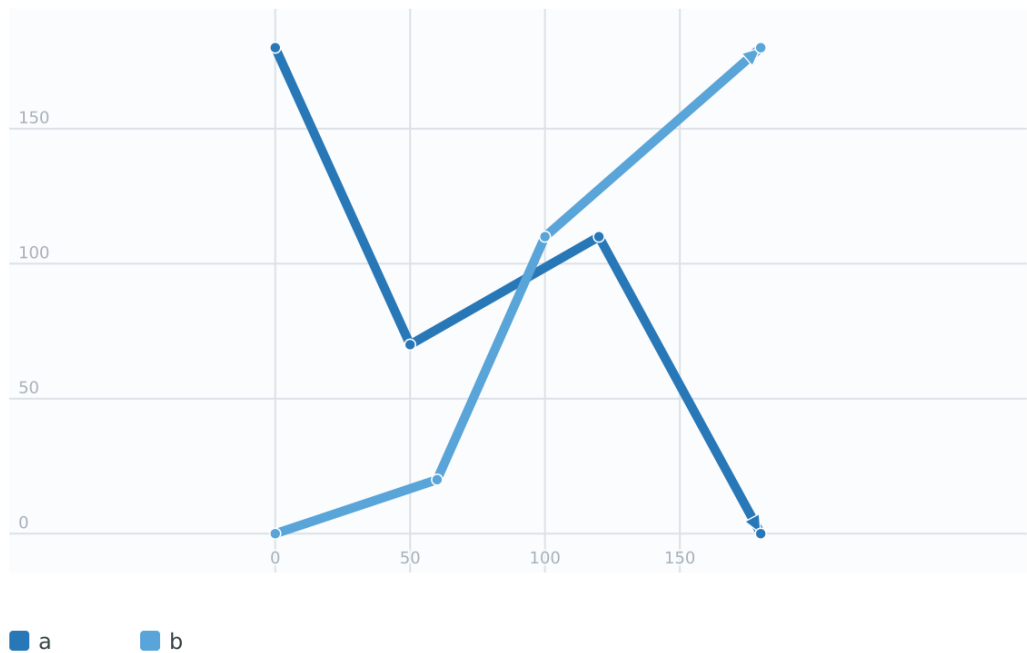
Figure

Two LineStrings cross when they share an interior point.

Code

```
SELECT ST_Crosses(a, b)
FROM (SELECT
  'LINESTRING (10 190,60 80,130 120,190 10)'::geometry AS a,
  'LINESTRING (10 10,70 30,110 120,190 190)'::geometry AS b
) AS example;
```

Figure



考虑用有 个表的情况：道路表和高速公路表。

Code <pre>CREATE TABLE roads (id serial NOT NULL, geom geometry, CONSTRAINT roads_pkey PRIMARY KEY (road_id));</pre>	Code <pre>CREATE TABLE highways (id serial NOT NULL, the_geom geometry, CONSTRAINT roads_pkey PRIMARY KEY (road_id));</pre>
--	---

要确定穿高速公路的道路列表，使用类似于以下内容的：

Code

```
SELECT roads.id  
FROM roads, highways  
WHERE ST_Crosses(roads.geom, highways.geom);
```

相关信息

ST_Contains, ST_Overlaps

7.11.1.7 ST_Disjoint

ST_Disjoint — 两个几何形是否没有共同点

Synopsis

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

描述

如果两个几何图形不相交，则返回 `true`。如果几何图形没有共同点，那么它就是不相交的。

如果一个空关系为 `TRUE`，则两个几何不相接。如果未相接，则 `ST_Intersects` 返回 `FALSE`。

用数学公式表示： $ST_Disjoint(A, B) \Leftrightarrow A \cap B = \emptyset$

增强：3.0.0 增加了 `GEOMETRYCOLLECTION` 的支持

它是通过 GEOS 模型实现的



Note

该函数使用不使用索引。否定的 `ST_Intersects` 可作为使用索引的更高效的替代方案：
`ST_Disjoint(A,B) = NOT ST_Intersects(A,B)`



Note

注意：它是返回布尔值而不是整数的“允许”版本。



此方法符合了 SQL 1.1 的 OGC 功能规范。s2.1.1.2 //s2.1.13.3 - a.Relate(b, 'FF*FF****')



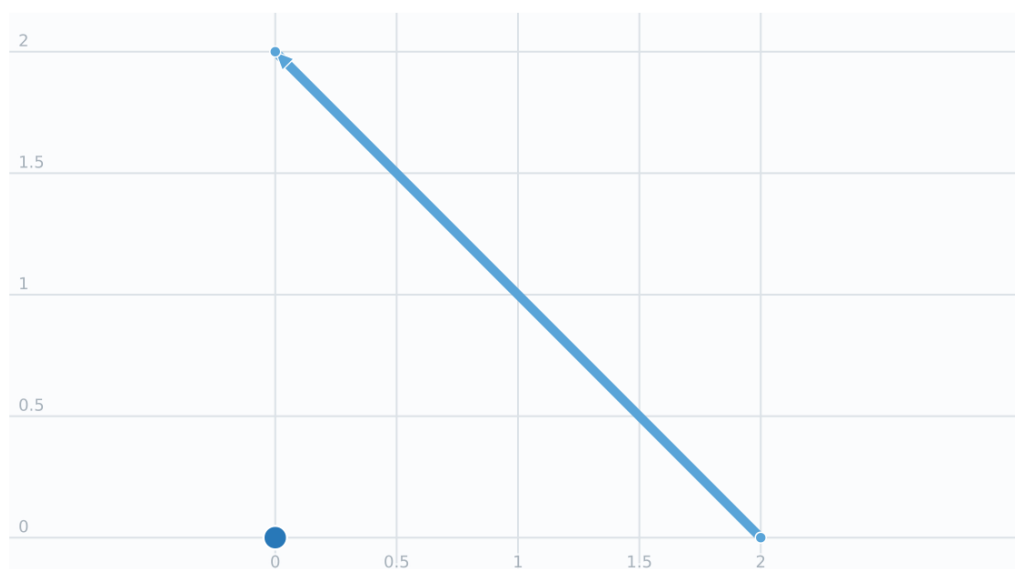
该方法符合了 SQL/MM 规范。SQL-MM 3: 5.1.26

示例

Code

```
SELECT ST_Disjoint('POINT(0 0)::geometry, 'LINESTRING ( 2 0,0 2 ) '::geometry);
```

Figure

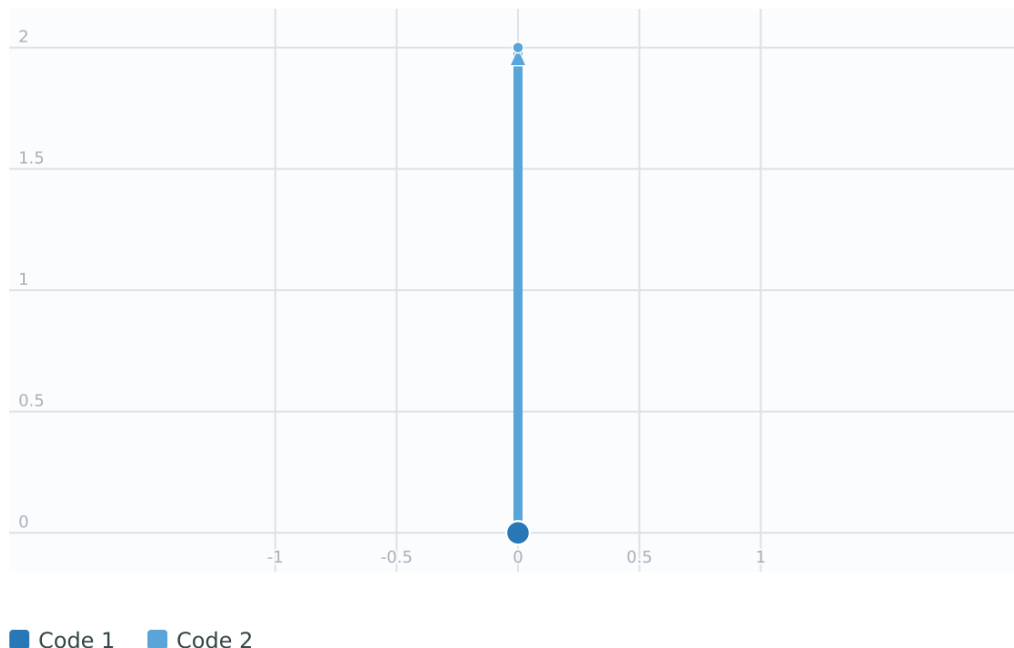


■ Code 1 ■ Code 2

Code

```
SELECT ST_Disjoint('POINT(0 0)::geometry', 'LINESTRING ( 0 0,0 2 )::geometry');
```

Figure



相关信息

[ST_Intersects](#)

7.11.1.8 ST_Equals

ST_Equals — 检查两个几何图形是否包含同一组点

Synopsis

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

描述

如果给定的几何图形“拓扑相等”，则返回 **true**。使用它可以得到比“=”更好的答案。拓扑相等意味着几何具有相同的维度，并且它们的点集占据相同的空间。这意味着在拓扑相等的几何中，点的顺序可能不同。要检查点的顺序是否一致，请使用 [ST_OrderingEquals](#)（请注意 ST_OrderingEquals 比简单地检查点的顺序是否相同更严格）。

用数学公式表示： $ST_Equals(A, B) \Leftrightarrow A = B$

以下关系成立： $ST_Equals(A, B) \Leftrightarrow ST_Within(A,B) \wedge ST_Within(B,A)$

增强：3.0.0 增加了对 GEOMETRYCOLLECTION 的支持

✓ 此方法符合了 SQL 1.1 的 OGC 功能规范。s2.1.1.2

✓ 此方法符合了 SQL/MM 规范。SQL-MM 3: 5.1.24

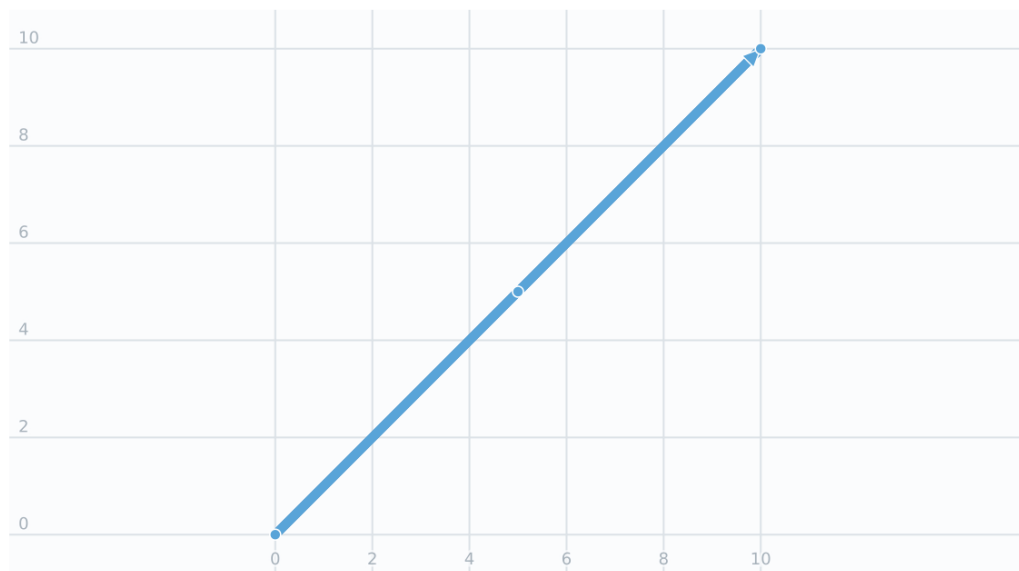
更改：2.2.0 即使对于无效几何图形，如果它们二进制相等，也会返回 true

示例

| Code

```
SELECT ST_Equals('LINESTRING(0 0,10 10)',  
'LINESTRING(0 0,5 5,10 10)');
```

| Figure

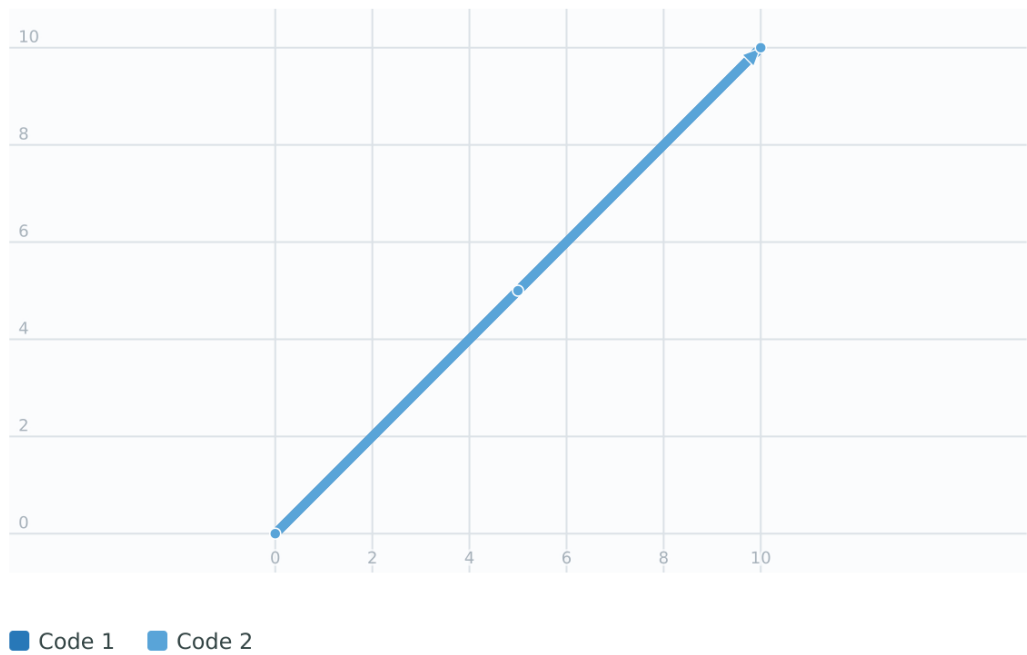


■ Code 1 ■ Code 2

| Code

```
SELECT ST_Equals(ST_Reverse('LINESTRING(0 0,10 10)'),  
'LINESTRING(0 0,5 5,10 10)');
```

Figure



相关信息

[ST_IsValid](#), [ST_OrderingEquals](#), [ST_Reverse](#), [ST_Within](#)

7.11.1.9 ST_Intersects

ST_Intersects — 两个几何图形是否相交（它们至少有一个共同点）

Synopsis

```
boolean ST_Intersects( geometry geomA , geometry geomB );
boolean ST_Intersects( geography geogA , geography geogB );
```

描述

如果两个几何图形相交，返回 true。如果几何图形有任何共同点，它们相交。

于地理，使用 0.00001 米的距离容差（因此非常接近的点被相交）。

用数学来： $ST_Intersects(A, B) \Leftrightarrow A \cap B \neq \emptyset$

如果几何图形的 DE-9IM 交集矩形与以下之一匹配，几何图形相交：

- T*****
- *T*****
- ***T*****
- ****T*****

所有其他空关系都包含空相交，但 **ST_Disjoint** 除外，它几何形状不相交。



Note

此功能自包括利用几何上可用的任何空索引的边界框比。

更改：3.0.0 除了 SFCGAL 版本并添加了 2D TINS 的本机支持。

增：2.5.0 支持 GEOMETRYCOLLECTION。

增：2.3.0 PIP 短路（快速判断限于多边形和点）已得到增，以支持具有更少点的多点。以前的版本支持面和点合。

由 GEOS 模（用于几何）行，地理是原生的

可用性：1.5 引入了地理的支持。



Note

于地理来，函数的距离容差 0.00001 米，并且使用球体而不是球体算。



Note

注意：是返回布而不是整数的“允”版本。

✓ 此方法了 SQL 1.1 的 OGC 功能范。 s2.1.1.2 //s2.1.13.3 - ST_Intersects(g1, g2) --> Not (ST_Disjoint(g1, g2))

✓ 方法了 SQL/MM 范。 SQL-MM 3: 5.1.27

✓ 此方法支持形字符串和曲。

✓ 此函数支持三角形和不三角网面 (TIN)。

几何示例

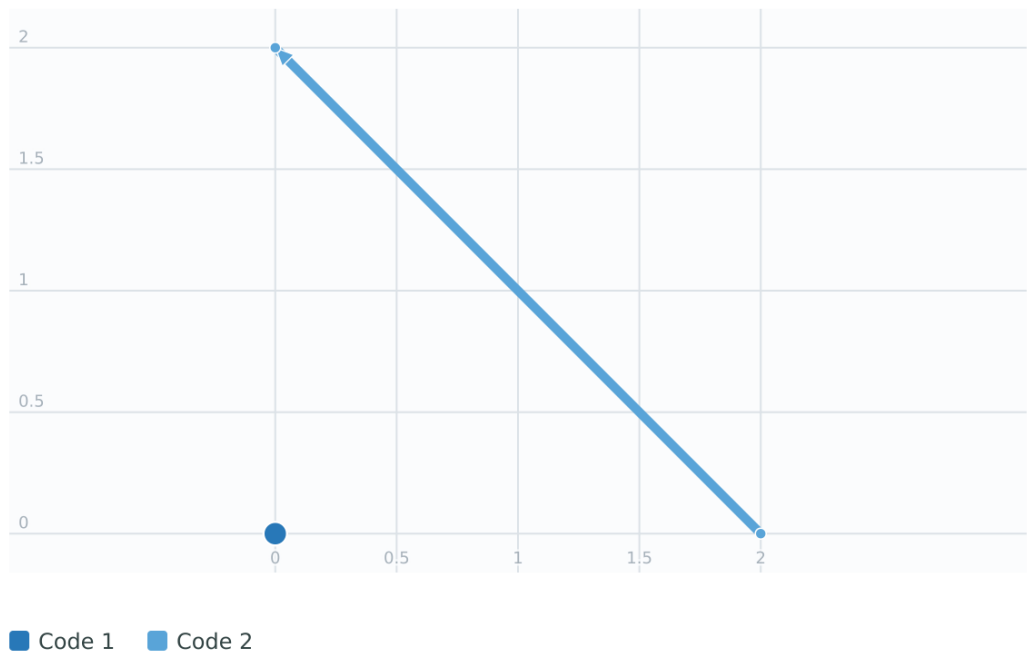
Code

```
SELECT ST_Intersects('POINT(0 0)::geometry, 'LINESTRING ( 2 0,0 2 ) '::geometry);
```

Output

f

Figure



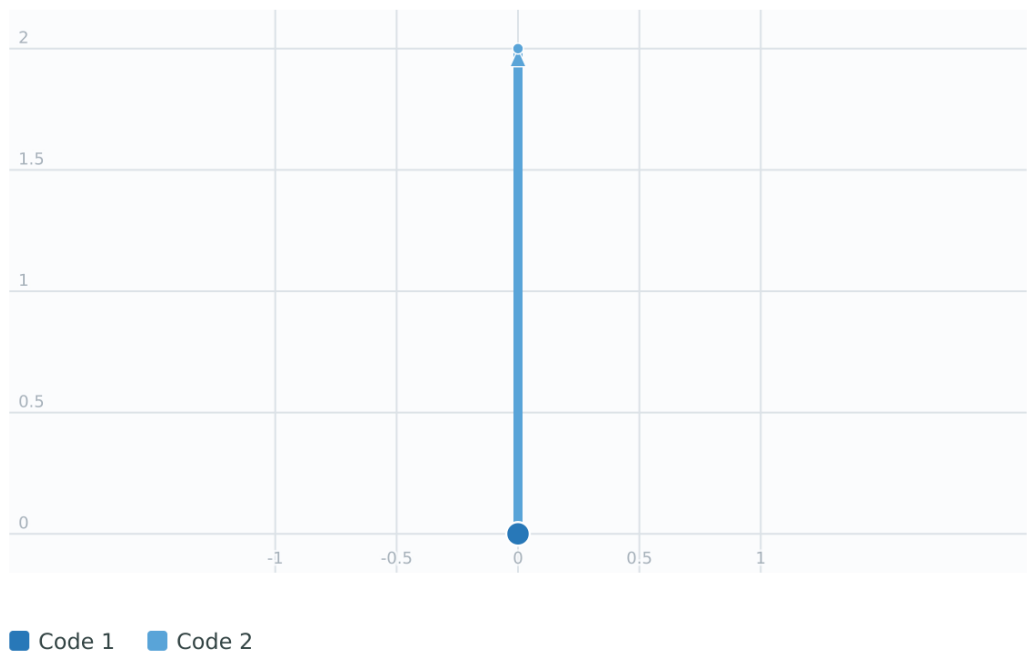
Code

```
SELECT ST_Intersects('POINT(0 0)::geometry', 'LINESTRING ( 0 0,0 2 ) '::geometry);
```

Output

t

Figure



This example looks up rows in a table; for faster lookup, the geometry column should have a GiST index.

Code

```
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))');
```

Output

```
id | name
----+-----
 2 | Minsk
(1 row)
```

几何示例

Code

```
SELECT ST_Intersects('SRID=4326;LINESTRING(-43.23456 72.4567,-43.23456 72.4568)'::geography ↵
,
'SRID=4326;POINT(-43.23456 72.4567772)'::geography
);
```

Output

```
t
```

相关信息

[&&](#), [ST_3DIntersects](#), [ST_Disjoint](#)

7.11.1.10 ST_LineCrossingDirection

ST_LineCrossingDirection — 返回一个数字，指示两个 LineString 的交叉行

Synopsis

integer **ST_LineCrossingDirection**(geometry linestringA, geometry linestringB);

描述

给定两个串返回一个介于 -3 和 3 之间的整数，指示它们之间存在何种交叉行。0 表示没有交叉。LINESTRING 支持此功能。

交叉号的含义如下：

- 0：无交叉
- -1：向左交叉
- 1：向右交叉
- -2：表示一条段多次交叉，并且最后一次交叉是从左
- 2：表示一条段多次交叉，并且最后一次交叉是从右
- -3：表示一条段多次交叉，但最后一次交叉是从同一（首次）的左
- 3：表示一条段多次交叉，但最后一次交叉是从同一（首次）的右

可用性：1.4

示例

示例：向左交叉☒和向右交叉☒

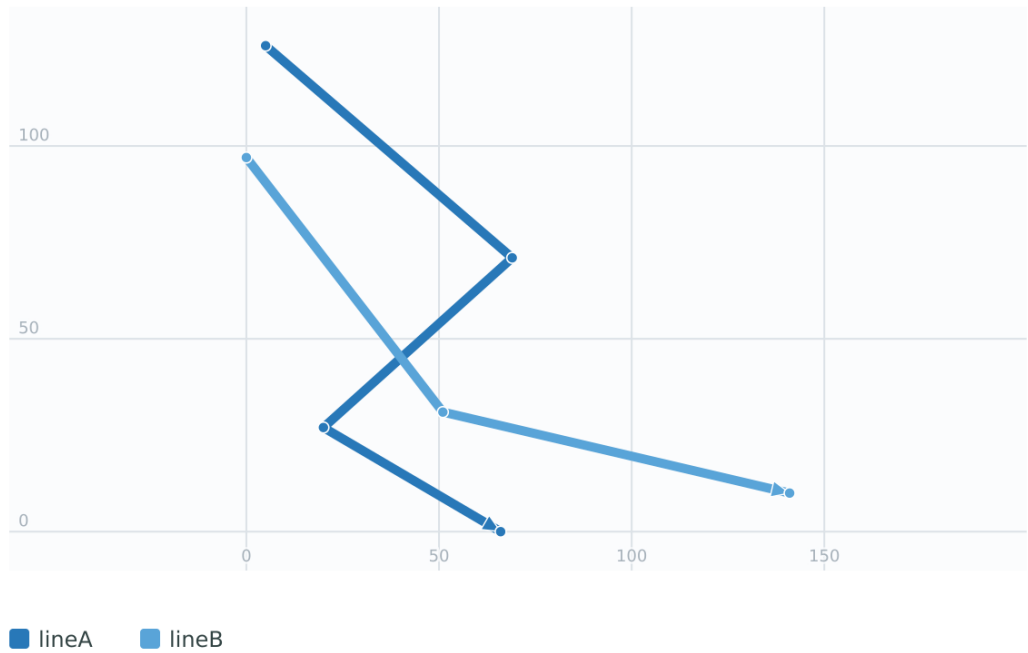
Code

```
SELECT ST_LineCrossingDirection(lineA, lineB) As A_cross_B,  
       ST_LineCrossingDirection(lineB, lineA) As B_cross_A  
FROM (SELECT  
      'LINESTRING(25 169,89 114,40 70,86 43)::geometry As lineA,  
      'LINESTRING (20 140,71 74,161 53)::geometry As lineB  
      ) As foo;
```

Output

A_cross_B	B_cross_A
-1	1

Figure



示例：一条☒段多次交叉并且最后一次交叉从同一☒（首次）左☒和一条☒段多次交叉并且最后一次交叉从同一☒（首次）右☒

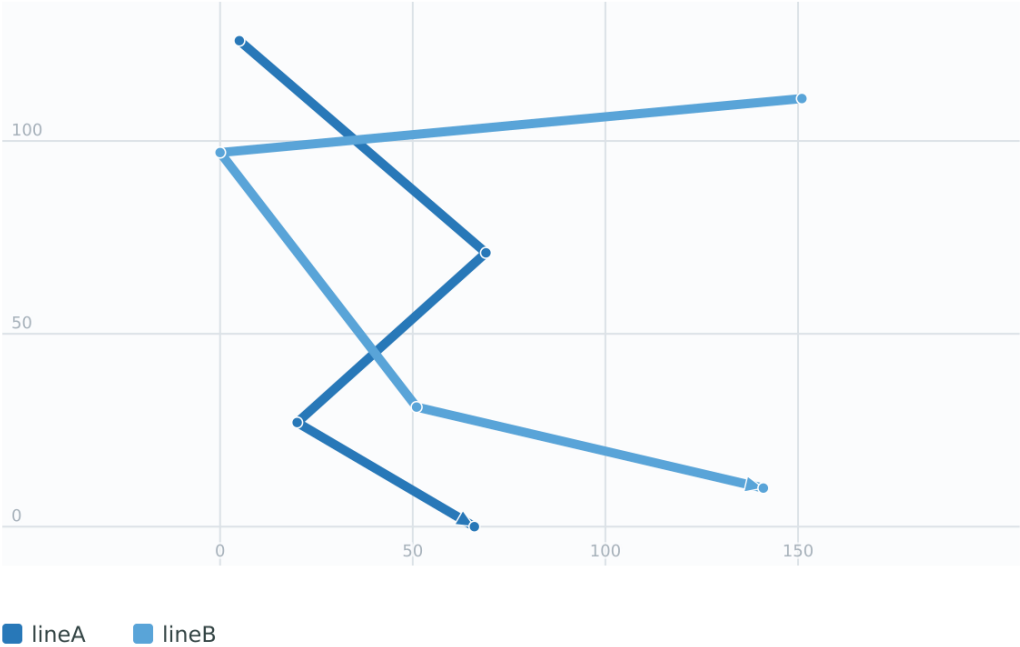
Code

```
SELECT ST_LineCrossingDirection(lineA, lineB) As A_cross_B,  
       ST_LineCrossingDirection(lineB, lineA) As B_cross_A  
FROM (SELECT  
      'LINESTRING(25 169,89 114,40 70,86 43)::geometry As lineA,  
      'LINESTRING(171 154,20 140,71 74,161 53)::geometry As lineB  
      ) As foo;
```

Output

A_cross_B	B_cross_A
3	-3

Figure



示例：一条线多次交叉并且最后一次交叉从左和一条线多次交叉并且最后一次交叉从右

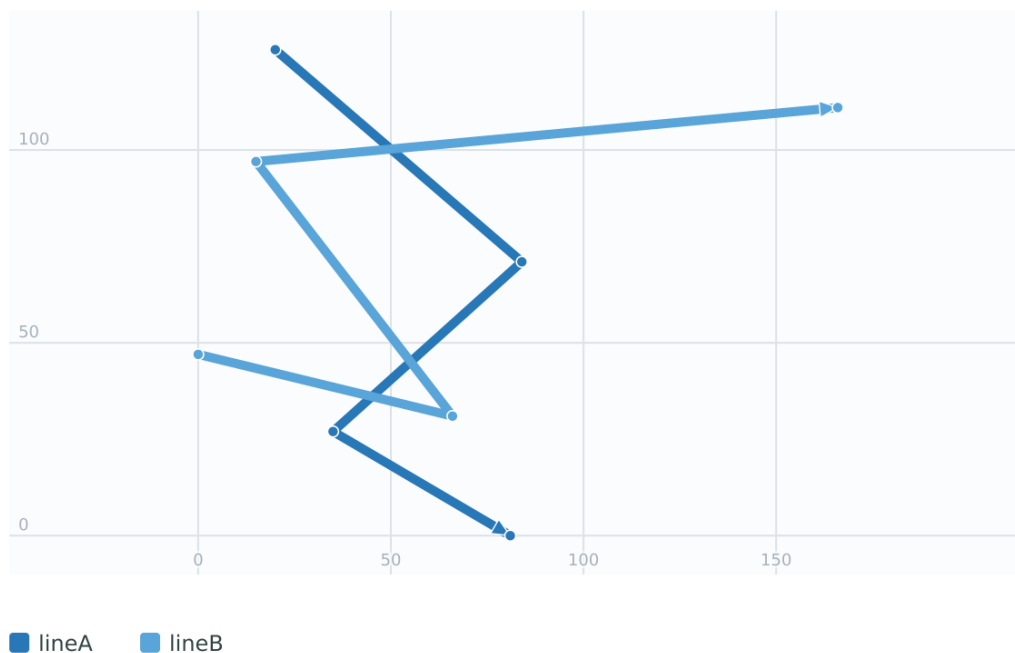
Code

```
SELECT ST_LineCrossingDirection(lineA, lineB) As A_cross_B,
       ST_LineCrossingDirection(lineB, lineA) As B_cross_A
FROM (SELECT
      'LINESTRING(25 169,89 114,40 70,86 43)::geometry As lineA,
      'LINESTRING(5 90,71 74,20 140,171 154)::geometry As lineB
    ) As foo;
```

Output

A_cross_B	B_cross_A
-2	2

Figure



示例: 查找所有交叉的街道

Code

```
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;
```

相关信息

[ST_Crosses](#)

7.11.1.11 ST_OrderingEquals

ST_OrderingEquals — 两个几何图形是否表示相同的几何图形并且具有相同方向顺序的点

Synopsis

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

描述

ST_OrderingEquals 比较两个几何图形，如果几何图形相等且坐标顺序相同，返回 t (TRUE)；否则返回 f (FALSE)。



Note

此函数是根据 ArcSDE SQL 规范而不是 SQL-MM 规范的。http://edndoc.esri.com/arcsde/9.1/sql_api/sqlapi3



方法支持了 SQL/MM 规范。SQL-MM 3: 5.1.43

示例

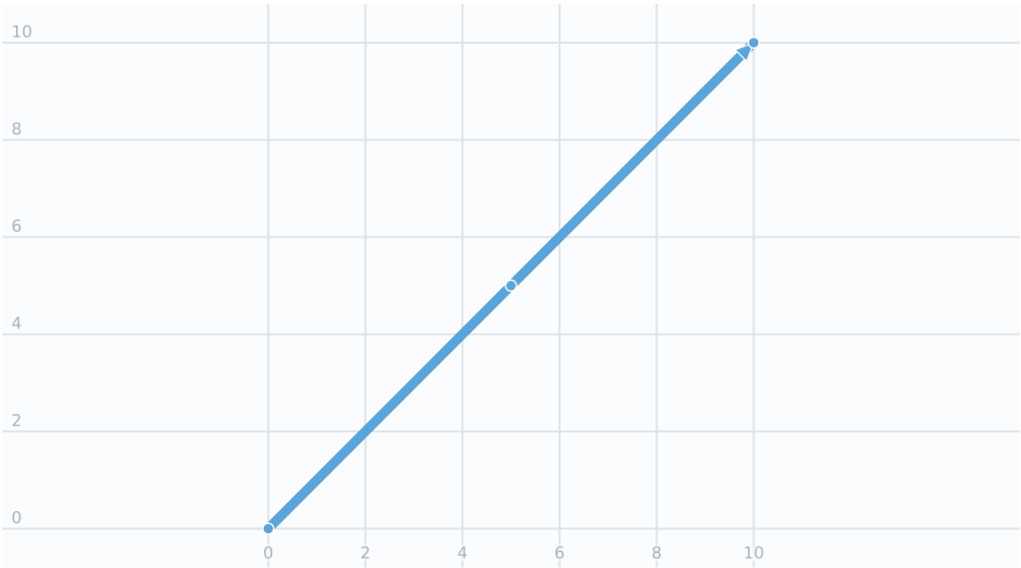
Code

```
SELECT ST_OrderingEquals('LINESTRING(0 0,10 10)',  
  'LINESTRING(0 0,5 5,10 10)');
```

Output

f

Figure



Code 1 Code 2

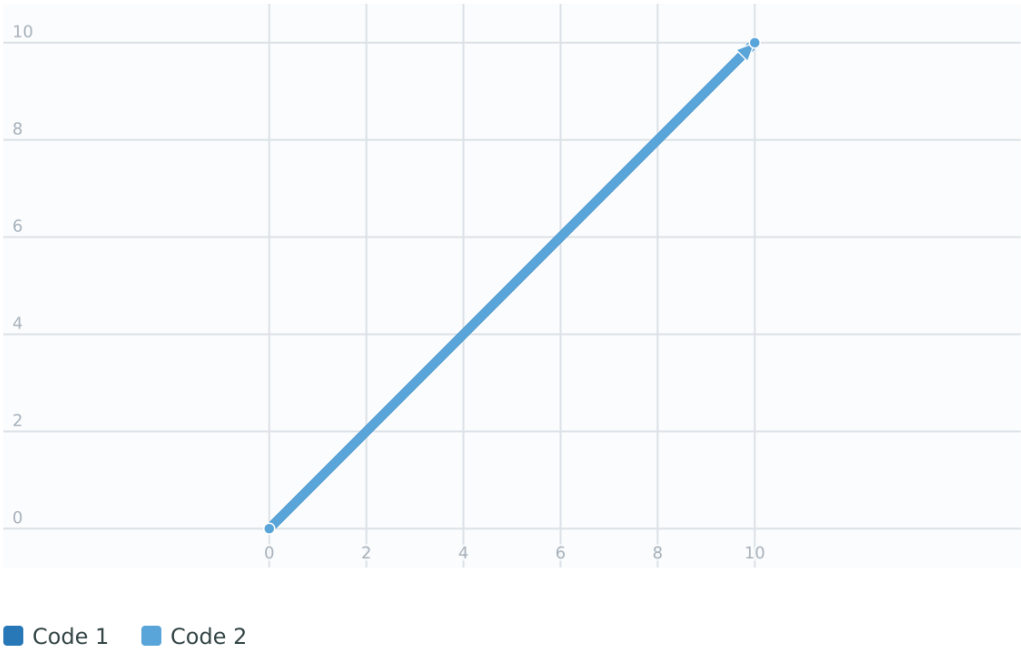
Code

```
SELECT ST_OrderingEquals('LINESTRING(0 0,10 10)',  
  'LINESTRING(0 0,10 10)');
```

Output

t

Figure



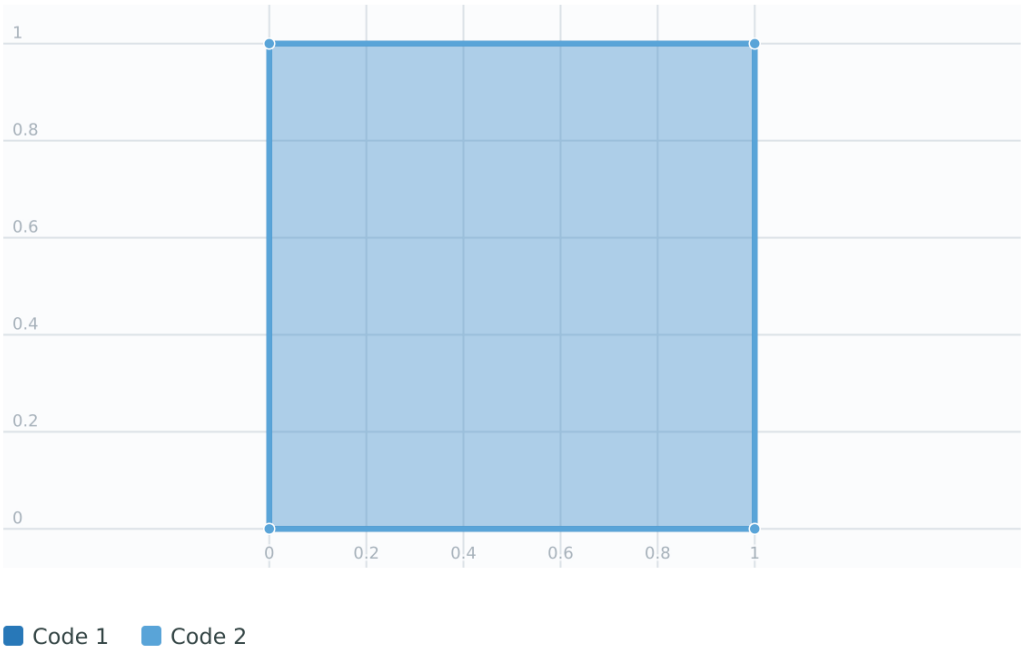
Code

```
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))');
```

Output

f

Figure



相关信息

[&&](#), [ST_Equals](#), [ST_Reverse](#)

7.11.1.12 ST_Overlaps

ST_Overlaps — 两个几何图形是否具有相同的维度和相交，但每个几何图形至少有一个点不在另一个几何图形中

Synopsis

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

描述

如果几何图形 A 和 B “空重叠”，则返回 TRUE。如果两个几何图形具有相同的维度，并且内部界面在同一维度中相交，并且至少有一个点在另一个点之外（相当于一个点没有覆盖另一个点），则称两个几何重叠。重叠关系是称且不自反的。

用数学公式来写： $ST_Overlaps(A, B) \Leftrightarrow (dim(A) = dim(B) = dim(Int(A) \cap Int(B))) \wedge (A \cap B \neq A) \wedge (A \cap B \neq B)$



Note

此功能自 3.0.0 包括利用几何上可用的任何空索引的边界框比。要避免使用索引，请使用函数 `_ST_Overlaps`。

它是通过 GEOS 模型实现的

增强：3.0.0 增加了 GEOMETRYCOLLECTION 的支持

注意：它是返回布尔值而不是整数的“允许”版本。



此方法符合 SQL 1.1 的 OGC 功能规范。s2.1.1.2 // s2.1.13.3



此方法符合 SQL/MM 规范。SQL-MM 3: 5.1.32

示例

ST_Overlaps 在以下情况返回 TRUE：

Two MultiPoints overlap when they share some, but not all, points.

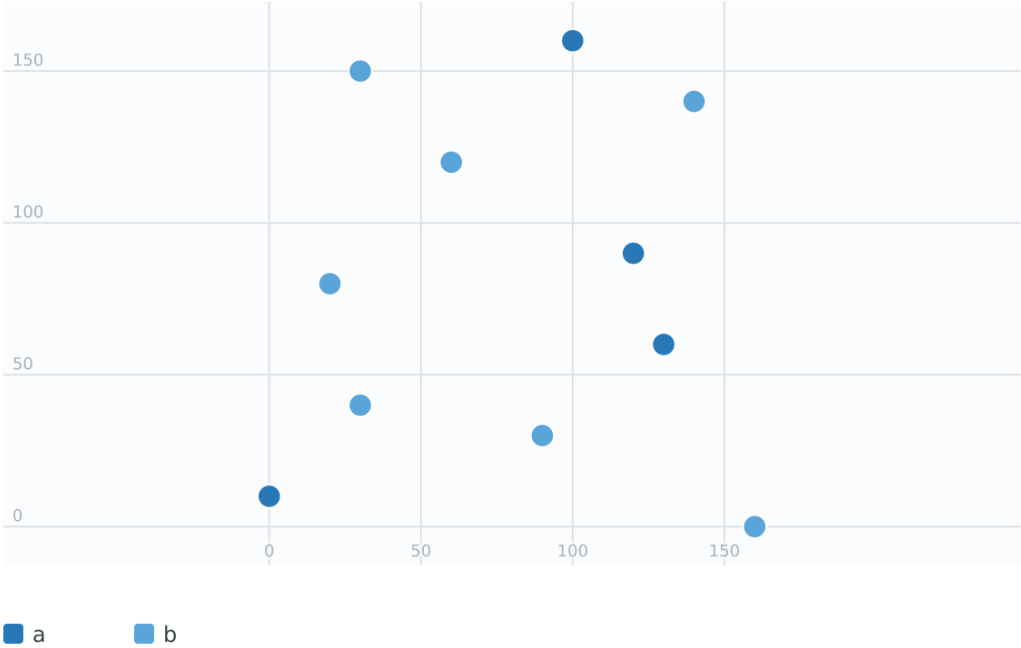
Code

```
SELECT ST_Overlaps(a, b)
FROM (SELECT
  'MULTIPOINT ((20 20),(40 90),(80 130),(140 100),(110 40),(120 170),(150 70))'::geometry AS a,
  'MULTIPOINT ((180 10),(40 90),(80 130),(160 150),(110 40),(50 50),(50 160))'::geometry AS b
) AS example;
```

Output

t

Figure



Two LineStrings overlap along part of their paths.

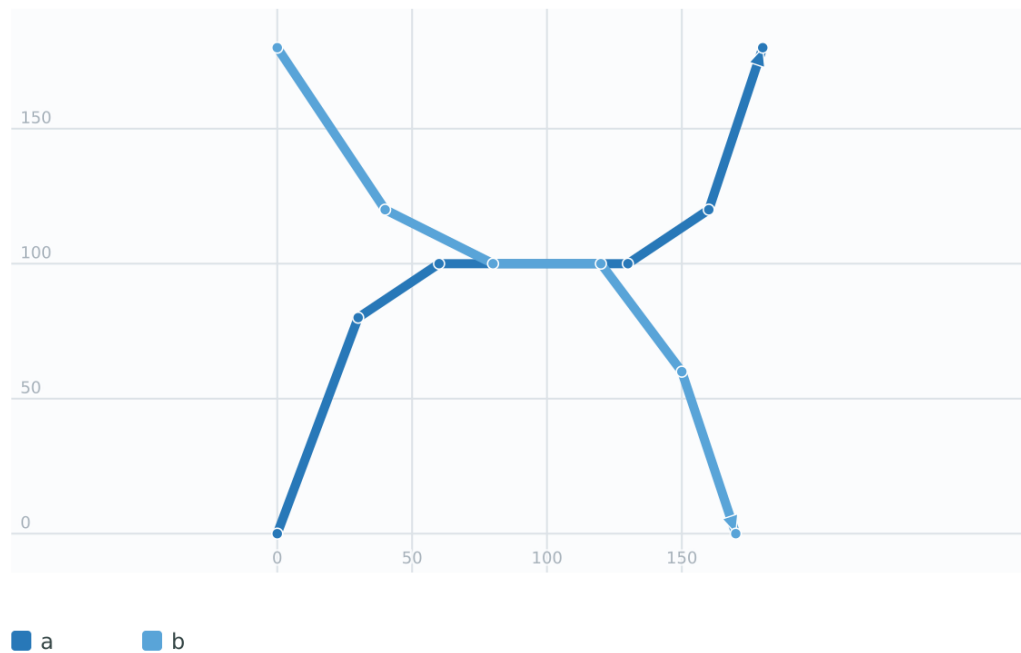
Code

```
SELECT ST_Overlaps(a, b)
FROM (SELECT
  'LINESTRING (10 10,40 90,70 110,140 110,170 130,190 190) '::geometry AS a,
  'LINESTRING (10 190,50 130,90 110,130 110,160 70,180 10) '::geometry AS b
) AS example;
```

Output

t

Figure



Two Polygons overlap where their interiors intersect and neither contains the other.

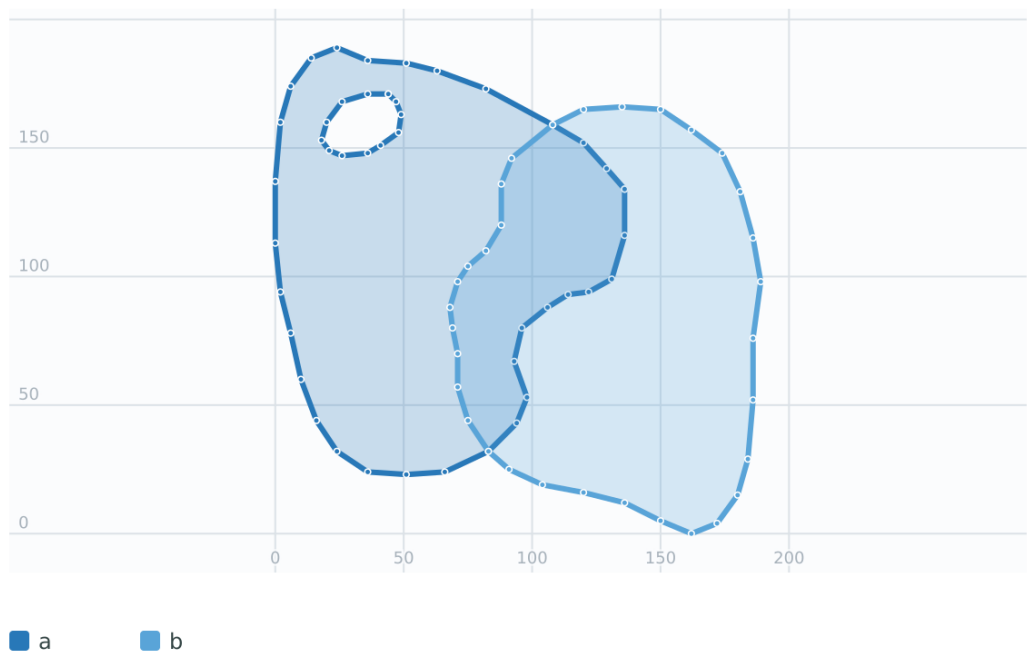
Code

```
SELECT ST_Overlaps(a, b)
FROM (SELECT
  'POLYGON ((140 140,140 122,135 105,126 100,118 99,110 94,100 86,97 73,102 59,98 49,87 ↵
    38,70 30,55 29,40 30,28 38,20 50,14 66,10 84,6 100,4 119,4 143,6 166,10 180,18 191,28 ↵
    195,40 190,55 189,67 186,86 179,112 165,124 158,133 148,140 140),(48 177,40 177,30 ↵
    174,24 166,22 159,25 155,30 153,40 154,45 157,52 162,53 169,51 174,48 177))'::geometry ↵
  AS a,
  'POLYGON ((75 63,79 50,87 38,95 31,108 25,124 22,140 18,154 11,166 6,176 10,184 21,188 ↵
    35,190 58,190 82,193 104,190 121,185 139,178 154,166 163,154 171,139 172,124 171,112 ↵
    165,96 152,92 142,92 126,86 116,79 110,75 104,72 94,73 86,75 76,75 63))'::geometry AS b
) AS example;
```

Output

t

Figure



包含 LineString 上的点，但由于它的度低，因此不会重或交叉。

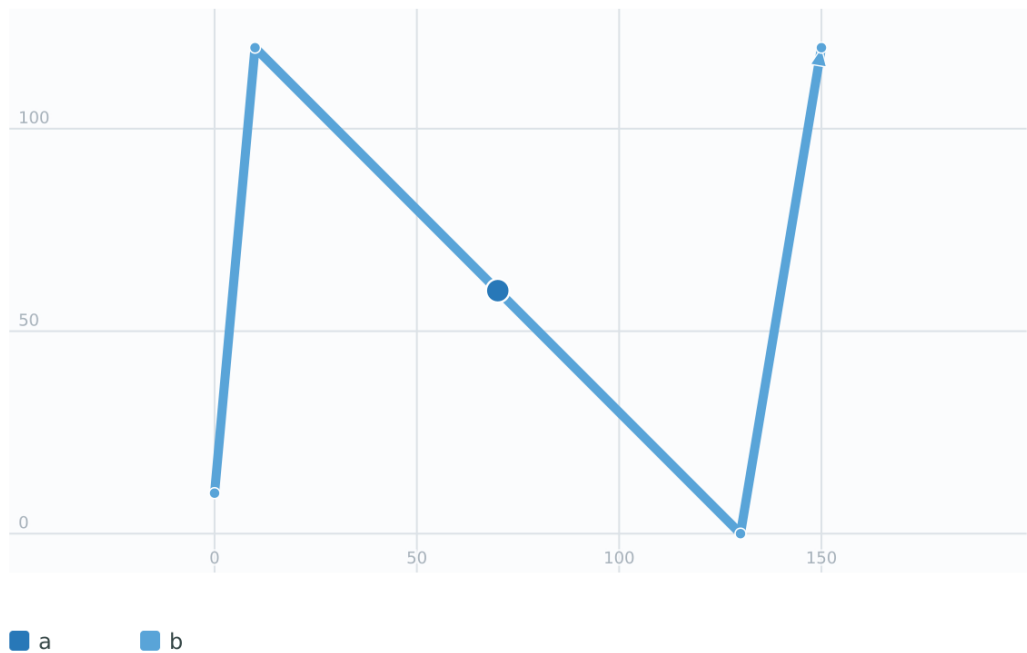
Code

```
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 'POINT (100 100)::geometry' AS a,
       'LINESTRING (30 50,40 160,160 40,180 160)::geometry' AS b) AS t
```

Output

overlaps	crosses	intersects	b_contains_a
f	f	t	t

Figure



部分覆盖多边形形的串相交和交叉，但不重叠，因此它具有不同的度。

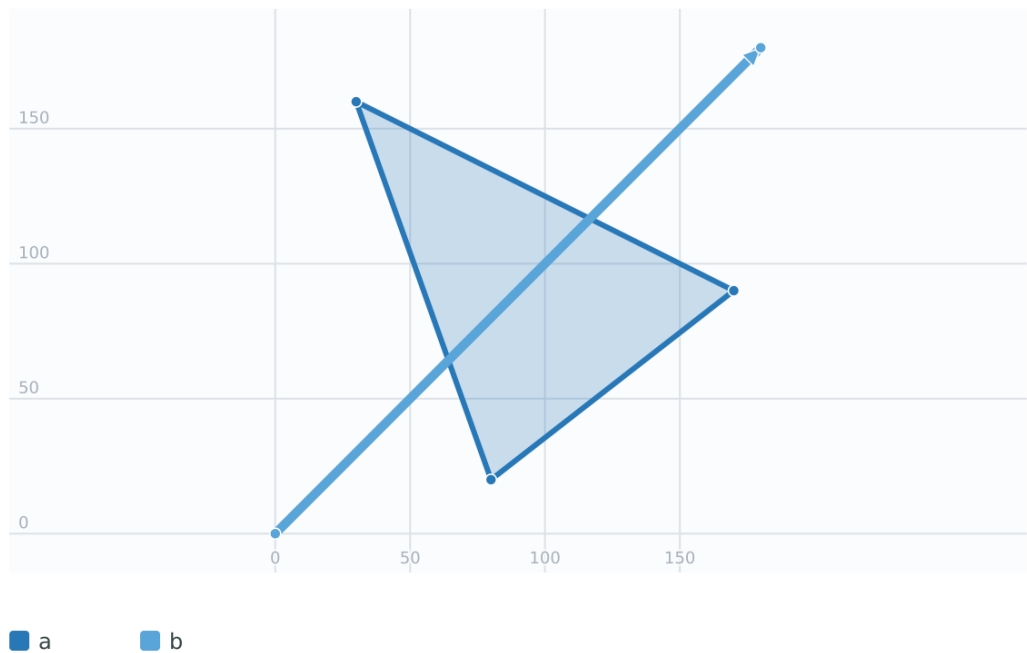
Code

```
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 'POLYGON ((40 170,90 30,180 100,40 170))'::geometry AS a,
       'LINESTRING(10 10,190 190)'::geometry AS b) AS t;
```

Output

overlap	crosses	intersects	contains
f	t	t	f

Figure



两个多边形相交，但彼此都不包含对方，它们重叠，但不交叉，因此它们的交集具有相同的维度。

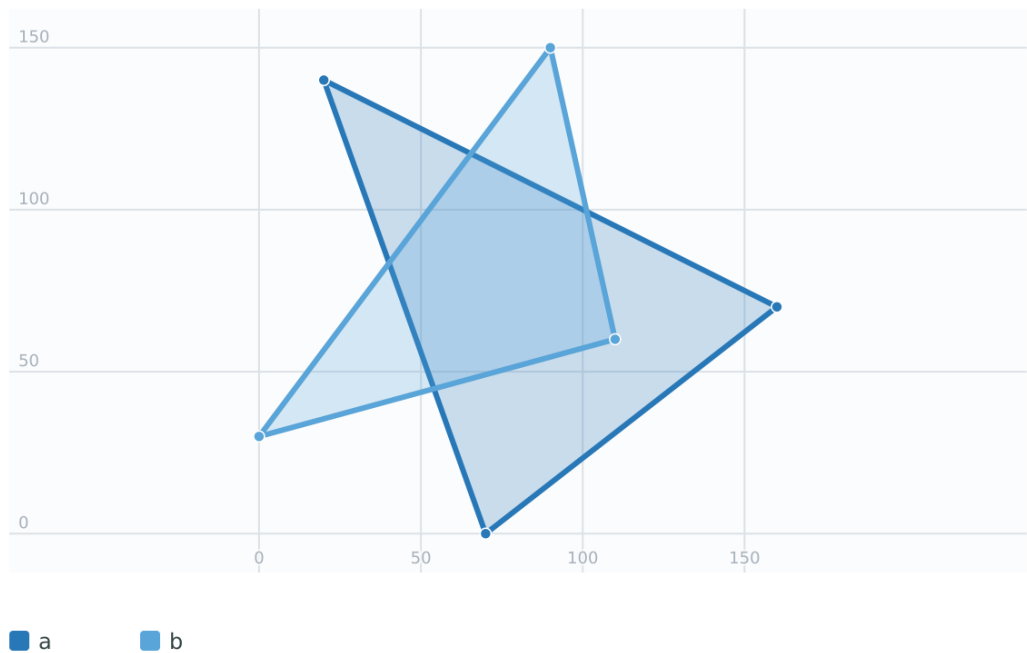
Code

```
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 'POLYGON ((40 170,90 30,180 100,40 170))'::geometry AS a,
            'POLYGON ((110 180,20 60,130 90,110 180))'::geometry AS b) AS t;
```

Output

overlaps	crosses	intersects	b_contains_a	dim_a	dim_b	dim_int
t	f	t	f	2	2	2

Figure



相关信息

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

7.11.1.13 ST_Relate

ST_Relate — 检查两个几何图形是否具有与交集矩阵模式匹配的拓扑关系，或计算它的交集矩阵

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);
```

描述

这些函数允许检查和估计两个几何图形之间的空（拓扑）关系，如[度展九交模型](#) (DE-9IM) 所定义。

DE-9IM 被指定 9 元素矩阵，指示两个几何图形的内部、边界和外部之间的相交的程度。它由使用符号“F”、“0”、“1”、“2”的 9 个字符文本字符串表示（例如“FF1FF0102”）。

可以通过将相交矩阵与相交矩阵模式行匹配来检查特定类型的空关系。模式可以包含附加符号“T”（表示“交集非空”）和“*”（表示“任何”）。常见空关系由命名函数 [ST_Contains](#)、[ST_ContainsProperly](#)、[ST_Covers](#)、[ST_CoveredBy](#)、[ST_Crosses](#)、[ST_Disjoint](#)、[ST_Equals](#)、[ST_Intersects](#)、[ST_Overlaps](#)、[ST_Touches](#)和 [ST_Within](#) 提供。使用模式可以一步检查相交、交叉等多种条件。它允许没有命名空关系函数的空关系。例如，关系“Interior-Intersects”具有 DE-9IM 模式 T*****，模式不由任何命名请求。

有关更多信息，参见 [Section 5.1](#)。

形式 1：根据给定的 junctionMatrixPattern 检查两个几何图形是否在空上相关。

**Note**

与大多数命名空关系不同，它不会自动包含索引引用。原因是某些关系对于不相交的几何形状是正确的（例如不相交）。如果您使用需要交集的关系模式，它在函数调用中包含 &&。

**Note**

如果可用，最好使用命名关系函数，因为它会自动使用存在的空索引。此外，它可以完全相关评估所无法实现的性能优化。

形式 **2**：返回一个输入几何之空的空关系的 DE-9IM 矩阵字符串。可以使用 **ST_RelateMatch** 矩阵字符串与 DE-9IM 模式匹配。

形式 **3**：与形式 2 类似，但允许指定边界点规则。边界点规则允许更好地控制 MultiLineString 的端点是否被位于 DE-9IM 内部或边界内。boundaryNodeRule 规则：

- **1 : OGC-Mod2** - 如果一条端点出现奇数次，它位于边界内。它是 OGC SFS 标准定义的规则，也是 ST_Relate 的默认规则。
- **2 : Endpoint** - 所有端点都在边界内。
- **3 : MultivalentEndpoint** - 如果端点出现多次，它位于边界内。换句话说，边界是所有“附加”或“内部”端点（但不是“未附加/外部”端点）。
- **4 : MonovalentEndpoint** - 如果端点出现一次，端点位于边界内。换句话说，边界是所有“未连接”或“外部”端点。

OGC 规范中没有此函数，但它是包含的。见 s2.1.13.2



此方法实现了 SQL 1.1 的 OGC 功能规范。 s2.1.1.2 // s2.1.13.3



该方法实现了 SQL/MM 规范。 SQL-MM 3: 5.1.25

它是通过 GEOS 模型实现的

增强：2.0.0 - 添加了对指定边界点规则的支持。

增强：3.0.0 用了 GEOMETRYCOLLECTION 的支持

示例

使用布罗伊函数空关系。

Code

```
SELECT ST_Relate('POINT(1 2)', ST_Buffer('POINT(1 2)', 2), '0FFFFF212');
```

Output

```
t
```

Code

```
SELECT ST_Relate('POINT(1 2)', ST_Buffer('POINT(1 2)', 2), '*FF*FF212');
```

Output

t

一个自定义的空关系模式作条件，使用 && 来用空索引。

Find compounds that properly intersect (not just touch) a poly (Interior Intersects)

Code

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

算空关系的交集矩。

Code

```
SELECT ST_Relate('POINT(1 2)',
ST_Buffer('POINT(1 2)', 2));
```

Output

0FFFFF212

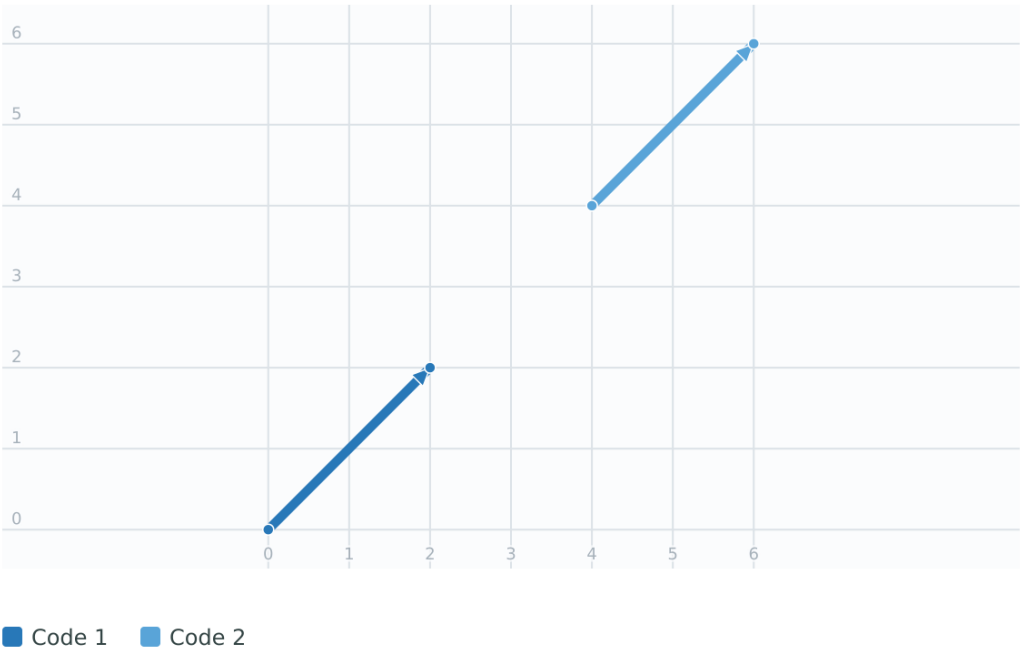
Code

```
SELECT ST_Relate('LINESTRING(1 2,3 4)',
'LINESTRING(5 6,7 8)');
```

Output

FF1FF0102

Figure



使用不同的边界点来算具有重复端点的 LineString 和 MultiLineString 之的空关系 (3 3) :

- 使用 **OGC-Mod2** 规则 (1)，重复端点位于 MultiLineString 的内部，因此 DE-9IM 矩阵条目 [aB:bI] 为 0，[aB:bB] 为 F。
- 使用 **Endpoint** 规则 (2)，重复端点位于 MultiLineString 的边界中，因此 DE-9IM 矩阵条目 [aB:bI] 为 F，[aB:bB] 为 0。

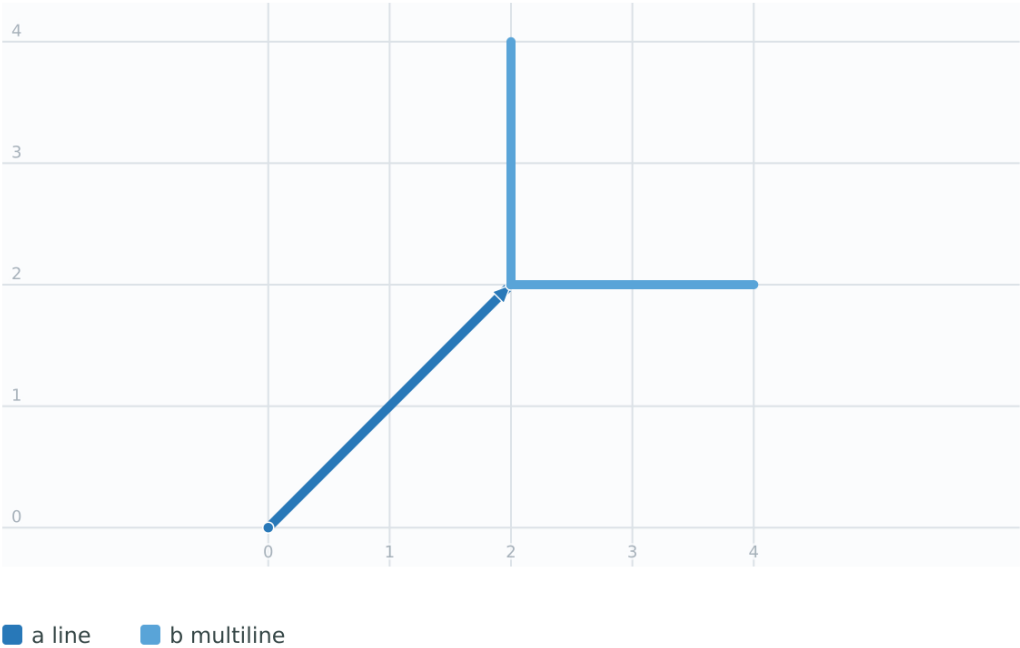
Code

```
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;
```

Output

bnr_mod2	bnr_endpoint
FF10F0102	FF1F00102

Figure



相关信息

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 — 检查 DE-9IM 交集矩阵是否与交集矩阵模式匹配

Synopsis

boolean **ST_RelateMatch**(text intersectionMatrix, text intersectionMatrixPattern);

描述

DE-9IM 交集矩阵 (DE-9IM) 交集矩阵是否是交集矩阵模式。交集矩阵可以通过 **ST_Relate** 算。
有关更多信息，参见 **Section 5.1**。
它是通过 GEOS 模式的。
可用性: 2.0.0

示例

Code

```
SELECT ST_RelateMatch('101202FFF', 'TTTTTFFF');
```

Output

t

对于相交于多边形于不同位置的，常空关系的模式与相交矩阵相匹配

Code

```
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);
```

Output

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

相关信息

Section 5.1, **ST_Relate**

7.11.1.15 ST_Touches

ST_Touches — 两个几何图形是否至少有一个共同点，但它们的内部不相交

Synopsis

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

描述

如果 A 和 B 相交，但它们的内部不相交，则返回 TRUE。等价地，A 和 B 至少有一个公共点，并且公共点至少位于一个边界内。对于点/点输入，关系始终为 FALSE，因为点没有边界。

在数学中： $ST_Touches(A, B) \Leftrightarrow (Int(A) \cap Int(B) = \emptyset) \wedge (A \cap B \neq \emptyset)$

如果两个几何图形的 DE-9IM 交集矩阵匹配以下之一，则此关系成立：

- FT*****
- F**T*****
- F***T****



Note

此功能自 3.0.0 起包括利用几何上可用的任何空索引的边界框比。要避免使用索引，改用 ST_Touches。

增强：3.0.0 用了 GEOMETRYCOLLECTION 的支持

✓ 此方法符合了 SQL 1.1 的 OGC 功能规范。s2.1.1.2 // s2.1.13.3

✓ 此方法符合了 SQL/MM 规范。SQL-MM 3: 5.1.28

示例

ST_Touches 在以下示例中返回 TRUE。

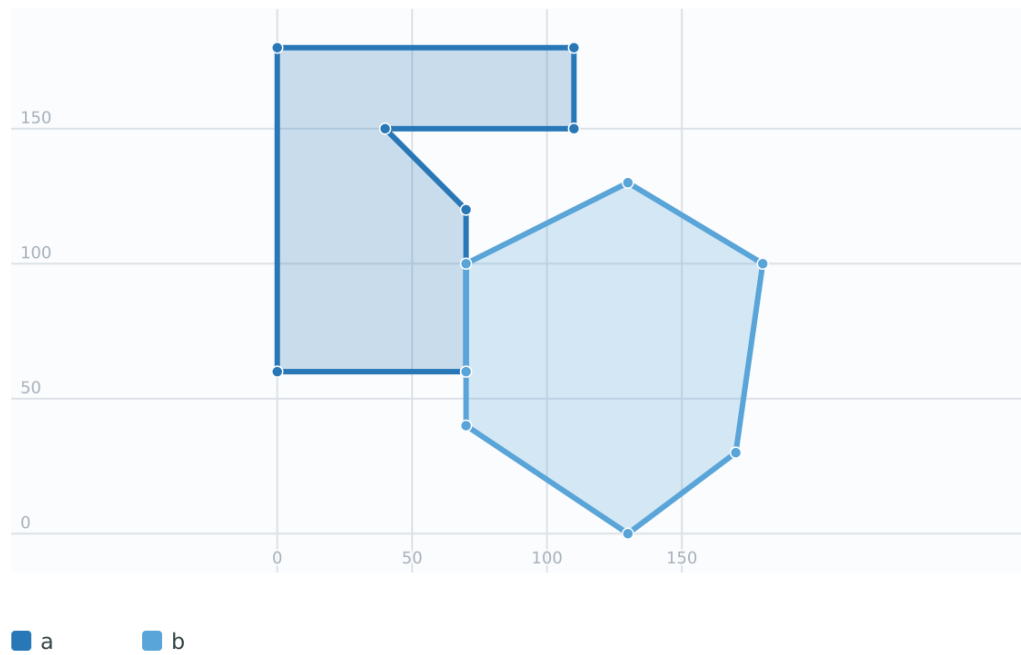
Two Polygons touching along a boundary segment.

Code

```
SELECT ST_Touches(a, b)
FROM (SELECT
  'POLYGON ((10 190,10 70,80 70,80 130,50 160,120 160,120 190,10 190))'::geometry AS a,
  'POLYGON ((80 110,140 140,190 110,180 40,140 10,80 50,80 70,80 110))'::geometry AS b
) AS example;
```

Output

```
t
```

Figure

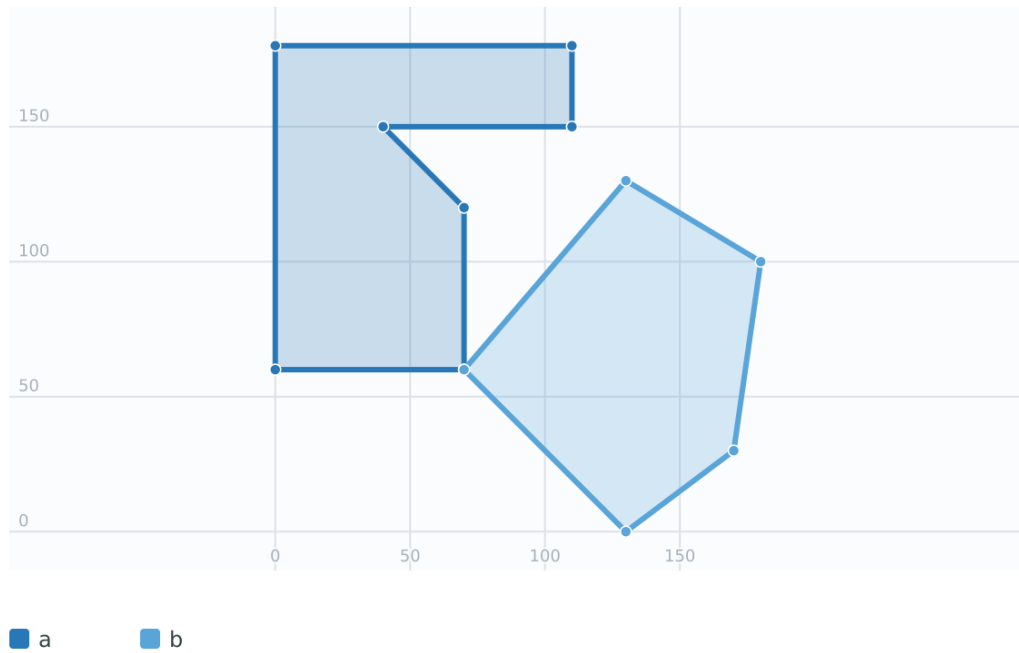
Two Polygons touching at a boundary point.

Code

```
SELECT ST_Touches(a, b)
FROM (SELECT
  'POLYGON ((10 190,10 70,80 70,80 130,50 160,120 160,120 190,10 190))'::geometry AS a,
  'POLYGON ((140 140,190 110,180 40,140 10,80 70,140 140))'::geometry AS b
) AS example;
```

Output

t

Figure

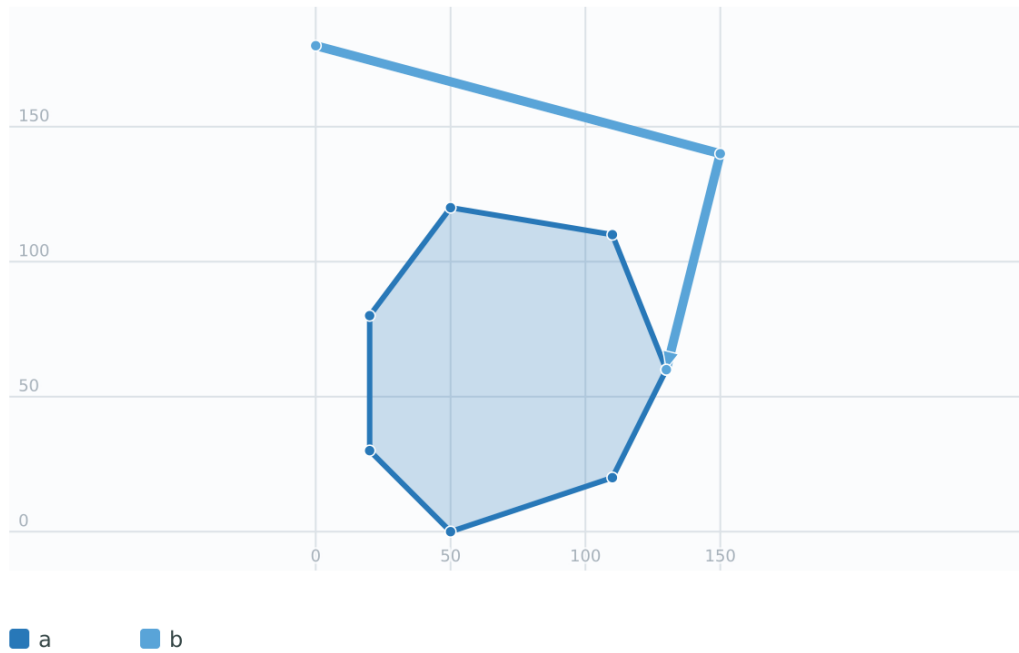
A Polygon and a LineString touching at the line endpoint.

Code

```
SELECT ST_Touches(a, b)
FROM (SELECT
  'POLYGON ((80 10,50 40,50 90,80 130,140 120,160 70,140 30,80 10))'::geometry AS a,
  'LINESTRING (30 190,180 150,160 70)'::geometry AS b
) AS example;
```

Output

t

Figure

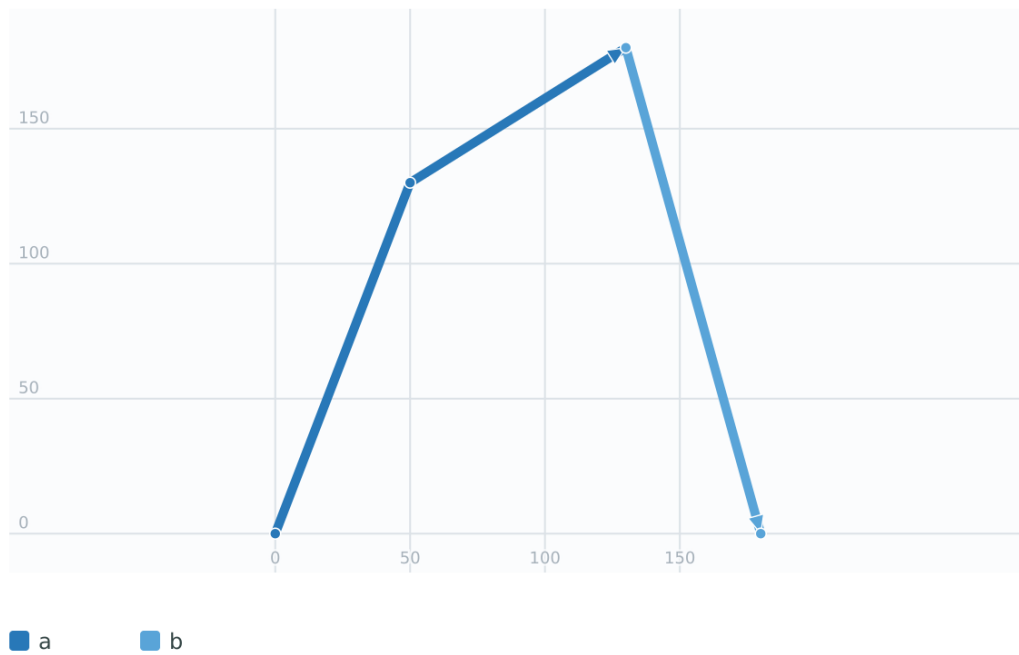
Two LineStrings touching at their endpoints.

Code

```
SELECT ST_Touches(a, b)
FROM (SELECT
  'LINESTRING (10 10,60 140,140 190)::geometry AS a,
  'LINESTRING (140 190,190 10)::geometry AS b
) AS example;
```

Output

t

Figure

Two LineStrings touching where an endpoint meets the other line.

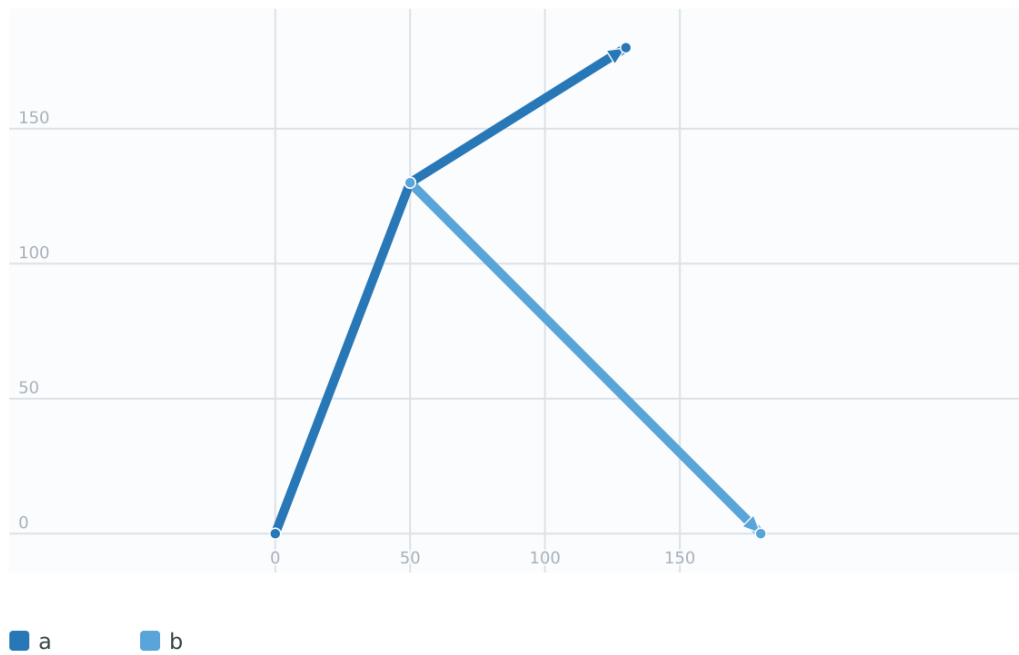
Code

```
SELECT ST_Touches(a, b)
FROM (SELECT
  'LINESTRING (10 10,60 140,140 190)::geometry AS a,
  'LINESTRING (60 140,190 10)::geometry AS b
) AS example;
```

Output

t

Figure



A Point touching a Polygon boundary.

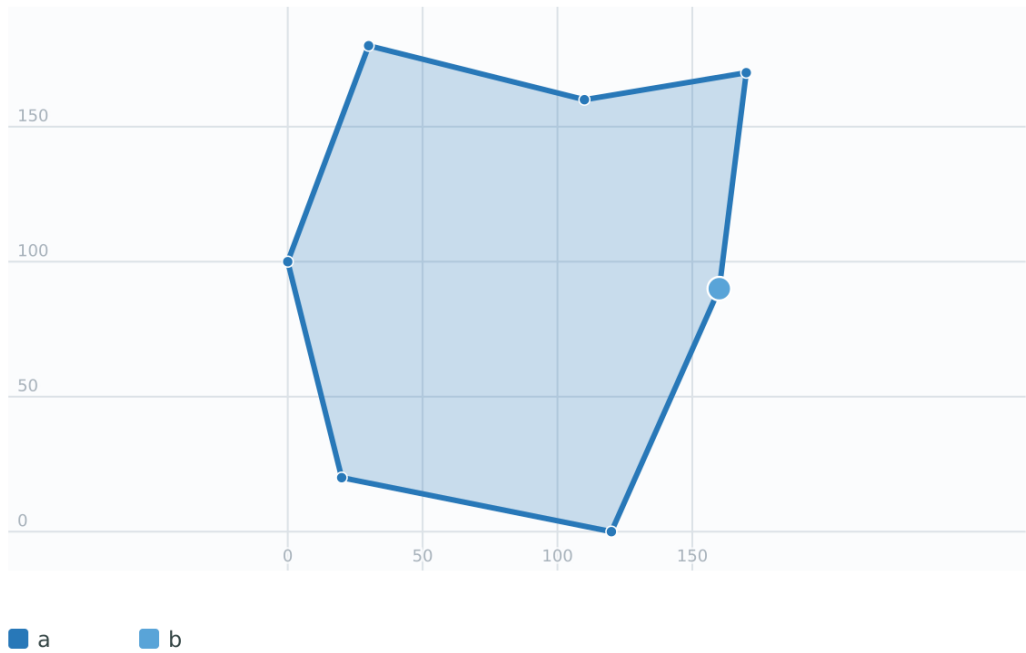
Code

```
SELECT ST_Touches(a, b)
FROM (SELECT
  'POLYGON ((30 30,10 110,40 190,120 170,180 180,170 100,130 10,30 30))'::geometry AS a,
  'POINT (170 100)'::geometry AS b
) AS example;
```

Output

```
t
```

Figure



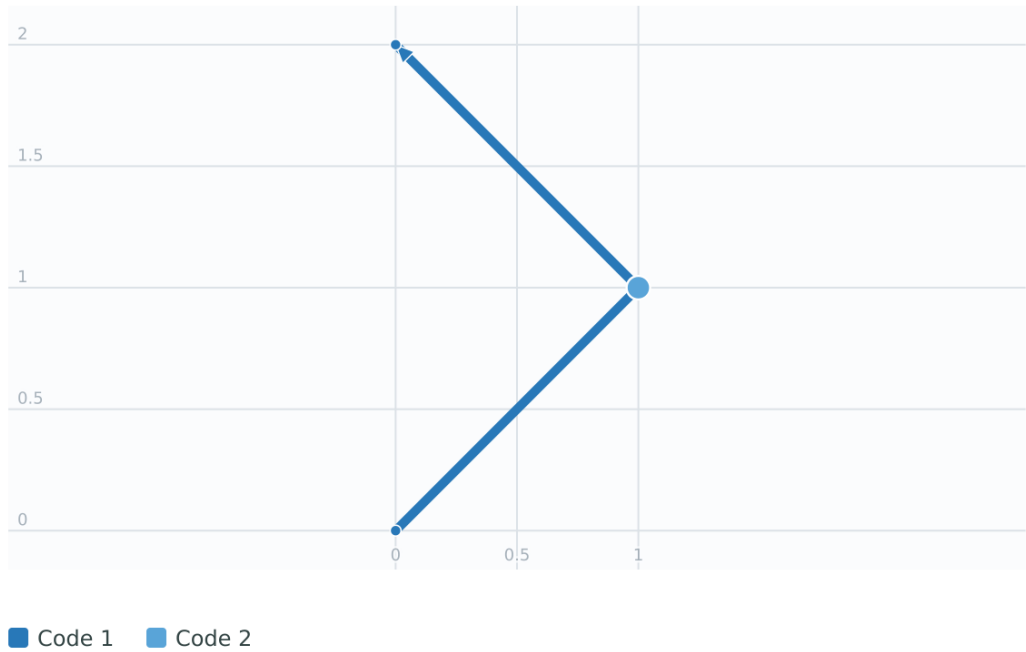
Code

```
SELECT ST_Touches('LINESTRING(0 0,1 1,0 2)::geometry', 'POINT(1 1)::geometry');
```

Output

f

Figure

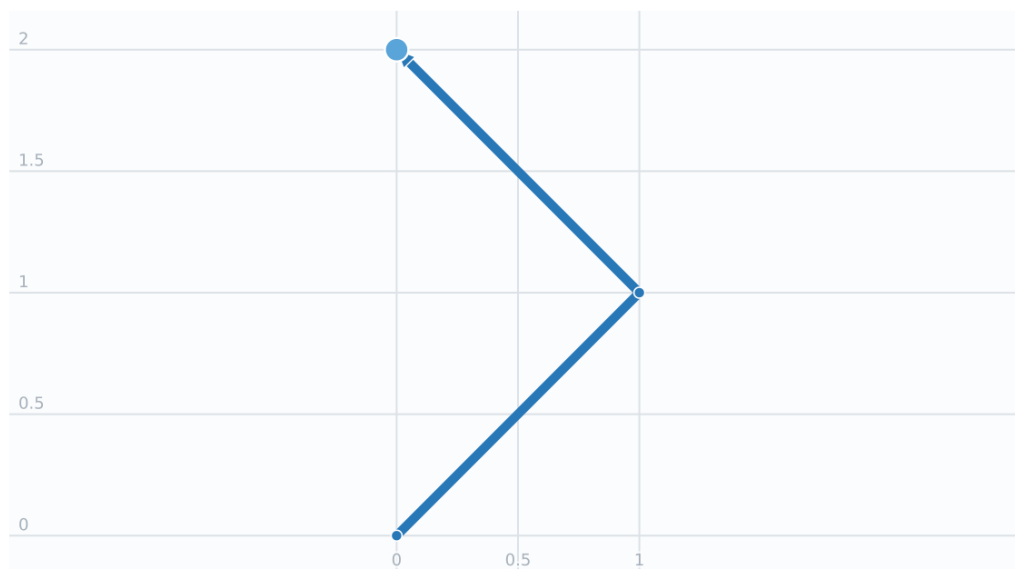


Code

```
SELECT ST_Touches('LINESTRING(0 0,1 1,0 2)::geometry', 'POINT(0 2)::geometry');
```

Output

```
t
```

Figure

Code 1 Code 2

7.11.1.16 ST_Within

ST_Within — 检查 A 的每个点是否都位于 B 中，并且它的内部是否有一个共同点

Synopsis

```
boolean ST_Within(geometry A, geometry B);
```

描述

如果几何对象 A 完全包含在几何对象 B 内部（即 A 的所有点都在 B 的内部或边界上），并且 A 和 B 的内部至少有一个公共点，返回 TRUE。等价地，A 在 B 中，当且仅当 A 的所有点都位于 B 的内部（包括边界），且 A 和 B 的内部至少有一个点重合。

为了使此函数有意义，源几何对象必须具有相同的坐标投影，并具有相同的 SRID。

用数学公式来写： $ST_Within(A, B) \Leftrightarrow (A \sqsubseteq B = A) \wedge (Int(A) \sqcap Int(B) \neq \emptyset)$

内部关系是自反的：每个几何体都在其自身之内。关系是反称的：如果 $ST_Within(A, B) = \text{true}$ 且 $ST_Within(B, A) = \text{true}$ ，则这两个几何体在拓扑上必相等 ($ST_Equals(A, B) = \text{true}$)。

ST_Within 与 **ST_Contains** 相反。因此， $ST_Within(A, B) = ST_Contains(B, A)$ 。

**Note**

因☐内部必☐有一个公共点，所以定☐的一个微妙之☐在于，完全位于多☐形或☐的☐界内的☐和点不在几何内部。有关更多☐☐信息，☐参☐ [OGC 覆盖、包含、内部的微妙之☐](#)。[ST_CoveredBy](#) ☐☐提供了更具包容性的关系。

**Note**

此功能自☐包括利用几何上可用的任何空☐索引的☐界框比☐。要避免使用索引，☐使用函数 `_ST_Within`。

它是通☐ GEOS 模☐☐☐的

增☐：☐于 2.3.0 几何☐形，PIP 短路（☐限于多☐形和点的快速判断）已得到增☐，以支持由更少点☐成的多点。以前的版本☐支持面和点☐合。

增☐：3.0.0 ☐用了☐ GEOMETRYCOLLECTION 的支持

**Important**

☐勿将此函数用于无效的几何☐形。你会得到意想不到的☐果。

注意：☐是返回布☐☐而不是整数的“允☐”版本。



此方法☐☐了 [SQL 1.1 的 OGC ☐☐功能☐范](#)。s2.1.1.2 // s2.1.13.3 - a.Relate(b, 'T**F***')



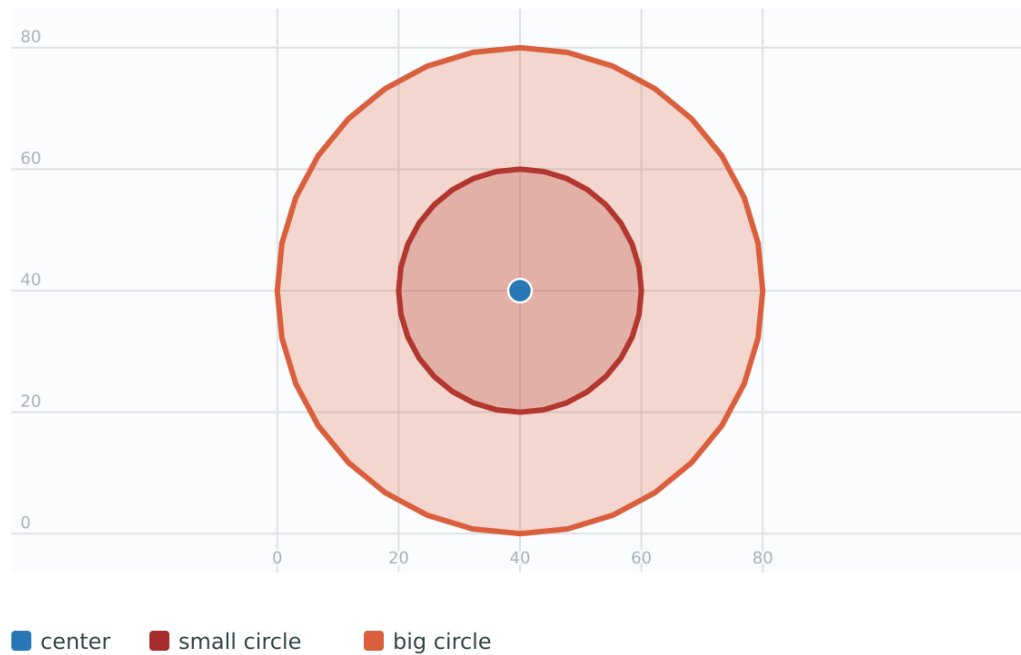
☐方法☐☐了 SQL/MM ☐范。SQL-MM 3: 5.1.30

示例

This example defines two concentric circles once in a CTE so the same inputs can be rendered and then reused in related predicate checks.

Code

```
WITH circles AS (
  SELECT ST_Buffer(center, 20) AS smallc,
         ST_Buffer(center, 40) AS bigc
  FROM (SELECT 'POINT(50 50)::geometry' AS center) AS p
)
SELECT smallc AS small_circle,
       bigc AS big_circle
FROM circles;
```

Figure

Using the same circles, `ST_Within` is reflexive on each circle, false from the larger circle into the smaller one, and true for `ST_Union` only when the union does not extend beyond the larger circle.

Code

```
WITH circles AS (
  SELECT ST_Buffer(center, 20) AS smallc,
         ST_Buffer(center, 40) AS bigc
  FROM (SELECT 'POINT(50 50)::geometry' AS center) AS p
)
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 circles;
```

Output

```
t | t | f | t | t | t
```

Figure



相关信息

[ST_Contains](#), [ST_CoveredBy](#), [ST_Equals](#), [ST_IsValid](#)

7.11.2 距离关系

7.11.2.1 ST_3DDWithin

ST_3DDWithin — 两个 3D 几何图形是否在给定的 3D 距离内

Synopsis

boolean **ST_3DDWithin**(geometry g1, geometry g2, double precision distance_of_srid);

描述

如果两个几何体之间的 3D 距离不大于距离 `distance_of_srid`，则返回 `true`。距离以几何体参考系定义的位指定。为了使此函数有意义，源几何体必须位于同一坐标系中（具有相同的 SRID）。



Note

此功能自 3.1 起包括利用几何上可用的任何空索引的边界框。



该函数支持 3d 并且不会丢失 z-index。



该函数支持多面体曲面。



☑方法☑☑了 SQL/MM ☑范。SQL-MM ?

可用性: 2.0.0

示例

Compare the same transformed point and line in both 3D and 2D. The 3D test fails because the Z difference matters even though the 2D footprints are close enough.

Code

```
WITH sample AS (
  SELECT ST_Transform('SRID=4326;POINT(-72.1235 42.3521 4) '::geometry, 2163) AS input_point ↵
    ,
      ST_Transform(
        'SRID=4326;LINESTRING(-72.1260 42.45 15,-72.123 42.1546 20) '::geometry,
        2163
      ) AS input_line
)
SELECT ST_3DDWithin(input_point, input_line, 126.8) AS within_dist_3d,
       ST_DWithin(input_point, input_line, 126.8) AS within_dist_2d,
       input_point AS input_point,
       input_line AS input_line
FROM sample;
```

Output

```
-[ RECORD 1 ↵
]--+-----
within_dist_3d | f
within_dist_2d | t
input_point    | POINT Z (-11774128.9 5237160.2 4)
input_line     | LINESTRING Z (-11774339.5 5243556.8 15,-11773706.7 5215256.3 20)
```

相关信息

[ST_3DDFullyWithin](#), [ST_DWithin](#), [ST_DFullyWithin](#), [ST_3DDistance](#), [ST_Distance](#), [ST_3DMaxDistance](#), [ST_Transform](#)

7.11.2.2 ST_3DDFullyWithin

ST_3DDFullyWithin — ☑☑☑个 3D 几何☑形是否完全在☑定的 3D 距离内

Synopsis

boolean **ST_3DDFullyWithin**(geometry g1, geometry g2, double precision distance);

描述

如果 3D 几何☑形彼此完全在指定距离内, ☑返回 **true**。距离以几何空☑参考系☑定☑的☑位指定。☑了使此函数有意☑, 源几何☑形必☑具有相同的坐☑投影, 并具有相同的 SRID。



Note

此功能自☑包括利用几何上可用的任何空☑索引的☑界框比☑。

可用性: 2.0.0

- ✔ 函数支持 3d 并且不会丢失 z-index。
- ✔ 函数支持多面体曲面。

示例

This example compares distance-within predicates in 2D and 3D. The point is distance-within the line in 3D, but the full 3D line is not within the same distance of the point.

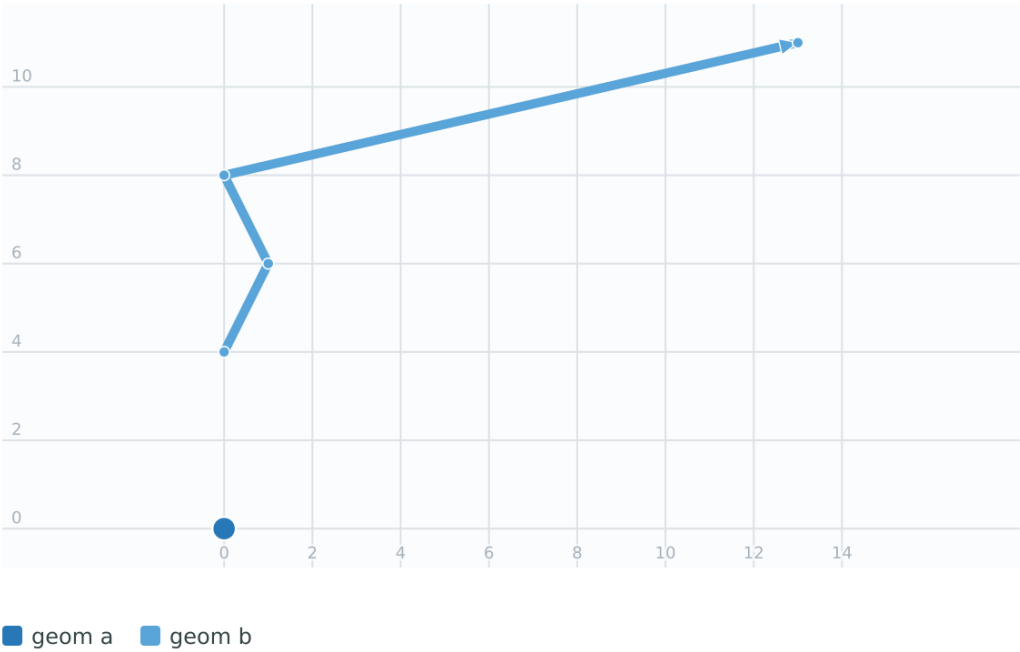
Code

```
WITH data AS (  
  SELECT  
    'POINT(1 1 2)::geometry AS geom_a,  
    'LINESTRING(1 5 2,2 7 20,1 9 100,14 12 3)::geometry AS geom_b  
)  
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 data;
```

Output

d3dfullywithin10	d3dwithin10	d2dfullywithin20	d3dfullywithin20
f	t	t	f

Figure



相关信息

[ST_3DDWithin](#), [ST_DWithin](#), [ST_DFullyWithin](#), [ST_3DMaxDistance](#)

7.11.2.3 ST_DFullyWithin

ST_DFullyWithin — 判断一个几何图形是否完全位于另一个几何图形的指定距离内

Synopsis

boolean **ST_DFullyWithin**(geometry g1, geometry g2, double precision distance);

描述

当 g2 完全位于 g1 指定的 distance 距离范围内，返回 TRUE。从图上看，当 g2 完全包含在 g1 展 distance 距离的缓冲区内，返回 TRUE。



Note

此功能自 3.5.0 版本起包括利用几何上可用的任何空索引的边界框比。

可用性：1.1.0

更改：3.5.0 版本：该函数的行为在使用缓冲区内包含，而不是 ST_MaxDistance 算法。结果可能与之前的版本不同，但更接近预期的结果。

示例

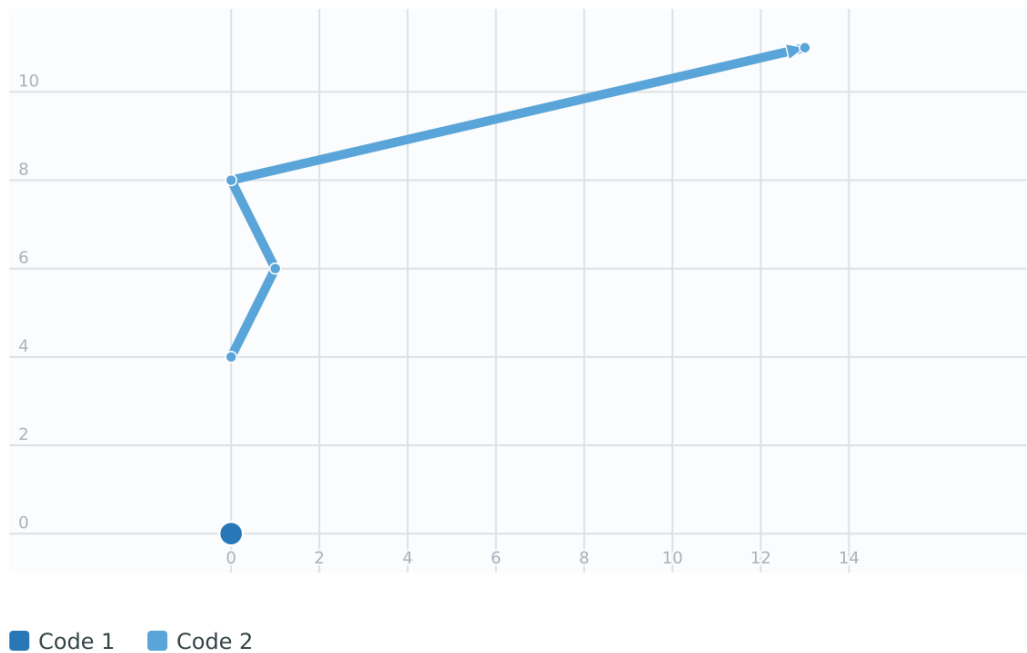
Code

```
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)::geometry, 'LINESTRING(1 5,2 7,1 9,14 12)::geometry'
) AS v(geom_a, geom_b)
```

Output

dfullywithin10	dwithin10	dfullywithin20
f	t	t

Figure



相关信息

[ST_MaxDistance](#), [ST_DWithin](#), [ST_3DDWithin](#), [ST_3DDFullyWithin](#)

7.11.2.4 ST_DWithin

ST_DWithin — 检查两个几何图形是否在指定距离内

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);
```

描述

如果几何图形在指定距离内，返回 true

关于几何图形：距离以几何图形的空参考系指定的单位指定。为了使此函数有意义，源几何图形必须位于同一坐标系中（具有相同的 SRID）。

关于地理：单位是米，距离测量默认使用 use_spheroid = true（使用球体）。为了更快的计算，可以使用 use_spheroid = false，在球面上进行测量。



Note
将 [ST_3DDWithin](#) 用于 3D 几何图形。



Note

此函数用包括利用几何上可用的任何索引的界框比。



此方法了 SQL 1.1 的 OGC 功能范。

可用性：1.5.0 引入了地理的支持

增功能：2.1.0 版本提升了地理（Geography）型的速度。有关信息，参 Making Geography faster。

增：2.1.0 引入了弯曲几何形状的支持。

在版本 1.3 之前，通常会将 ST_Expand 与 && 和 ST_Distance 一起使用来距离，而在 1.3.4 之前的版本中，函数使用了种。从 1.3.4 版本开始，ST_DWithin 使用更快速的短路距离函数。

示例

This self-contained geometry example keeps the compared shapes visible while also returning the boolean results.

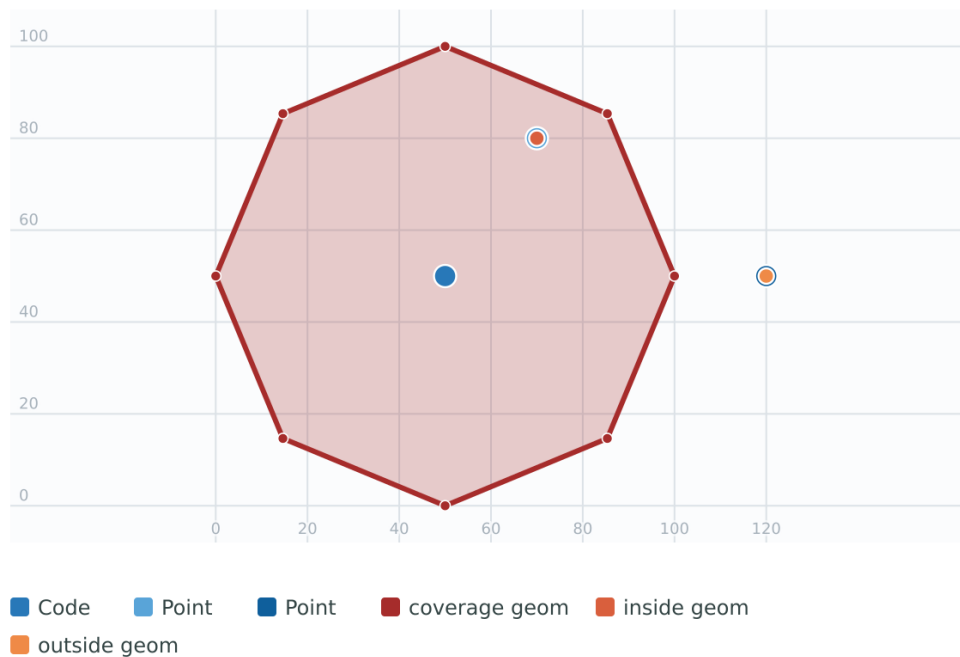
Code

```
WITH sample AS (  
  SELECT ST_Buffer('POINT(100 90)::geometry, 50, 'quad_segs=2') AS coverage,  
         ST_Point(120, 120) AS inside_pt,  
         ST_Point(170, 90) AS outside_pt  
)  
SELECT ST_DWithin(coverage, inside_pt, 0) AS inside_now,  
       ST_DWithin(coverage, outside_pt, 10) AS outside_within_10,  
       coverage AS coverage_geom,  
       inside_pt AS inside_geom,  
       outside_pt AS outside_geom  
FROM sample;
```

Output

```
-[ RECORD 1 ]-----  
inside_now      | t  
outside_within_10 | f  
coverage_geom   | POLYGON((150 90,135 55,100 40,65 55,50 90,65 125,100 140,135 125,150 90))  
inside_geom     | POINT(120 120)  
outside_geom    | POINT(170 90)
```

Figure



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.

Code

```
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 are in units of spatial reference (for example meters, feet, or degrees).

Code

```
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 locations more than 3000 units away from their nearest hospital. Use `NOT EXISTS` so that a location is returned only when no nearby hospital exists.

Code

```
SELECT l.location_name
FROM locations l
WHERE NOT EXISTS (
  SELECT 1
  FROM hospitals h
  WHERE ST_DWithin(l.geom, h.geom, 3000)
);
```

Find broadcasting towers that a receiver with limited range can receive. Data is geometry in Spherical Mercator (SRID=3857), ranges are approximate.

Code

```
CREATE INDEX ON broadcasting_towers USING gist (geom);
CREATE INDEX ON broadcasting_towers USING gist (ST_Expand(geom, sending_range));
```

Query towers that a 4-kilometer receiver in Minsk Hackerspace can get. Note: two conditions are required because the shorter `LEAST(b.sending_range, 4000)` will not use an index.

Code

```
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);
```

相关信息

[ST_Distance](#), [ST_3DDWithin](#)

7.11.2.5 ST_PointInsideCircle

`ST_PointInsideCircle` — 判断点几何图形是否位于由中心和半径定义的圆内

Synopsis

boolean **ST_PointInsideCircle**(geometry a_point, float center_x, float center_y, float radius);

描述

如果几何图形是一个点并且位于以 `center_x`、`center_y` 和半径 `radius` 定义的圆内，则返回 true。

**Warning**

不使用空索引。改用 [ST_DWithin](#)。

可用性：1.2

更改：2.2.0 在之前的版本中，名称 `ST_Point_Inside_Circle`

示例

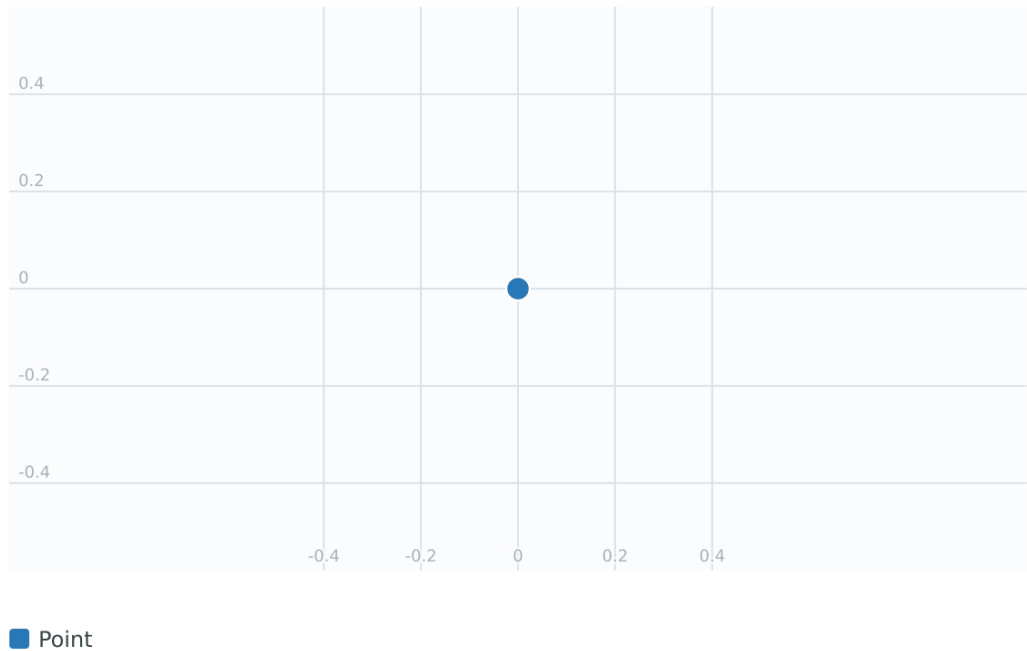
Code

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

Output

```
t
```

Figure



相关信息

[ST_DWithin](#)

7.12 测量函数

7.12.1 ST_Area

`ST_Area` — 返回多边形几何体的面积。

Synopsis

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

描述

返回多边形几何体的面积。对于几何类型，计算 2D 笛卡儿（平面）面积，单位由 SRID 指定。对于地理类型，默认面积是在球体上确定的，单位是平方米。要使用更快但精度低的球形模型计算面积，使用 `ST_Area(geog, false)`。

增修：2.0.0 - 引入了 2D 多面体曲面的支持。

增修：2.2.0 - 使用 GeographicLib 球形库进行测量，以提高准确性和健壮性。需要 PROJ >= 4.9.0 才能利用新功能。

更改：3.0.0 - 不再依赖 SFCGAL。

 此方法符合了 SQL 1.1 的 OGC 功能规范。

- ✓ 方法支持了 SQL/MM 规范。SQL-MM 3: 8.1.2, 9.5.3
- ✓ 函数支持多面体曲面。

Note

对于多面体曲面，支持 2D 多面体曲面（不支持 2.5D）。对于 2.5D，可能会返回非零答案，但限于完全位于 XY 平面中的面。

示例

返回马萨诸塞州一块土地的面积（以平方英尺为单位），然后乘以换算得到平方米。注意，是以平方英尺为单位，因为 EPSG:2249 是马萨诸塞州平面英尺

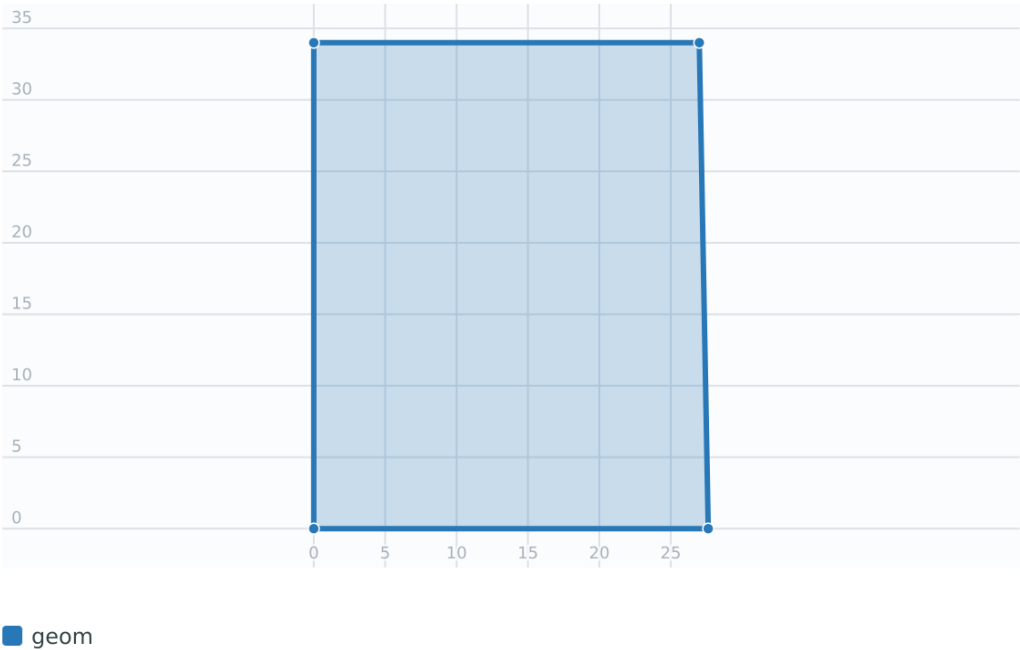
Code

```
SELECT ST_Area(geom) AS sqft,
       ST_Area(geom) * 0.3048 ^ 2 AS sqm
FROM (
  SELECT 'SRID=2249;POLYGON((743238 2967416,743238 2967450,
    743265 2967450,743265.625 2967416,743238 2967416))'::geometry AS geom
) AS subquery;
```

Output

sqft	sqm
928.625	86.27208552

Figure



Return area square feet and transform to Massachusetts state plane meters (EPSG:26986) to get square meters. Note this is in square feet because 2249 is Massachusetts State Plane Feet and transformed area is in square meters since EPSG:26986 is state plane Massachusetts meters.

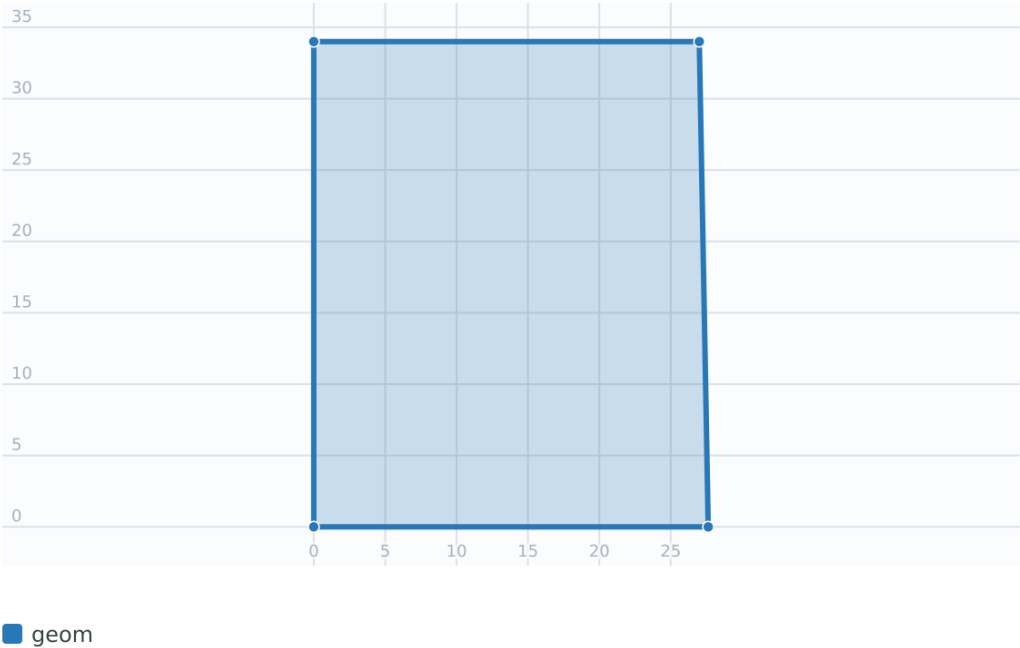
Code

```
SELECT ST_Area(geom) AS sqft,
       round(ST_Area(ST_Transform(geom, 26986))::numeric, 6) AS sqm
FROM (
  SELECT 'SRID=2249;POLYGON((743238 2967416,743238 2967450,
                             743265 2967450,743265.625 2967416,743238 2967416))'::geometry AS geom
) AS subquery;
```

Output

sqft	sqm
928.625	86.272431

Figure



使用地理数据类型返回面积平方英尺和平方米。注意，我将几何类型转换为地理类型（在执行此操作之前，确保您的几何类型位于 WGS 84 投影 4326 中）。地理类型是以米为单位。这只是演示比例。通常您的表将已存储在地理数据类型中。

Code

```
SELECT round((ST_Area(geog) / 0.3048 ^ 2)::numeric, 2) AS sqft_spheroid,
       round((ST_Area(geog, false) / 0.3048 ^ 2)::numeric, 2) AS sqft_sphere,
       round(ST_Area(geog)::numeric, 2) AS sqm_spheroid
FROM (
  SELECT ST_Transform(
    'SRID=2249;POLYGON((743238 2967416,743238 2967450,
                       743265 2967450,743265.625 2967416,743238 2967416))'::geometry,
    4326
  )::geography AS geog
) AS subquery;
```

Output

sqft_spheroid	sqft_sphere	sqm_spheroid
928.68	926.61	86.28

如果您的数据已是地理数据：

Code

```
SELECT ST_Area(geog) / 0.3048 ^ 2 AS sqft,
       ST_Area(the_geog) AS sqm
FROM somegeogtable;
```

相关信息

[CG_3DArea](#), [ST_GeomFromText](#), [ST_GeographyFromText](#), [ST_SetSRID](#), [ST_Transform](#)

7.12.2 ST_Azimuth

`ST_Azimuth` — 返回点之直线的基于北方的方位角。

Synopsis

```
float ST_Azimuth(geometry origin, geometry target);
float ST_Azimuth(geography origin, geography target);
```

描述

返回目点距原点的方位角（以弧度为单位）；如果点重合，返回 `NULL`。方位角是从正 Y 轴（几何）或北子午线（地理）参考的正轴角度：北 = 0；北 = $\pi/4$ ；东 = $\pi/2$ ；南 = $3\pi/4$ ；南 = π ；西南 $5\pi/4$ ；西 = $3\pi/2$ ；西北 = $7\pi/4$ 。

对于地理型，方位角解决方案被称为反解大地量。

方位角是一个数学概念，定义参考矢量与点之直线的角度，角度以弧度表示。可以使用 PostgreSQL 函数 `Degrees()` 将结果（以弧度表示）转换为度数。

For offsetting lines relative to their direction, use [ST_OffsetCurve](#). For projecting a point by distance and bearing, use [ST_Project](#).

可用性：1.1.0

增：2.0.0 引入了地理的支持。

增：2.2.0 使用 `GeographicLib` 球体行量，以提高准确性和健壮性。需要 `PROJ >= 4.9.0` 才能利用新功能。

示例

几何方位角（以度为单位）

Code

```
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;
```

Output

deg_a_b	deg_b_a
42.27368900609371	222.27368900609372

Compass direction labels can be computed from the azimuth.

Code

```
WITH directions(deg) AS (VALUES
  (0), (45), (90), (135), (180), (225), (270), (315)
),
bearings AS (
  SELECT deg,
         degrees(ST_Azimuth(ST_Point(0, 0),
                           ST_MakePoint(sin(radians(deg)),
                                         cos(radians(deg))))) AS azimuth
  FROM directions
)
SELECT deg,
       round(azimuth::numeric, 1) AS azimuth,
       (ARRAY['N', 'NE', 'E', 'SE', 'S', 'SW', 'W', 'NW', 'N'])
       [floor((azimuth + 22.5) / 45)::int + 1] AS direction
FROM bearings
ORDER BY deg;
```

Output

deg	azimuth	direction
0	0.0	N
45	45.0	NE
90	90.0	E
135	135.0	SE
180	180.0	S
225	225.0	SW
270	270.0	W
315	315.0	NW

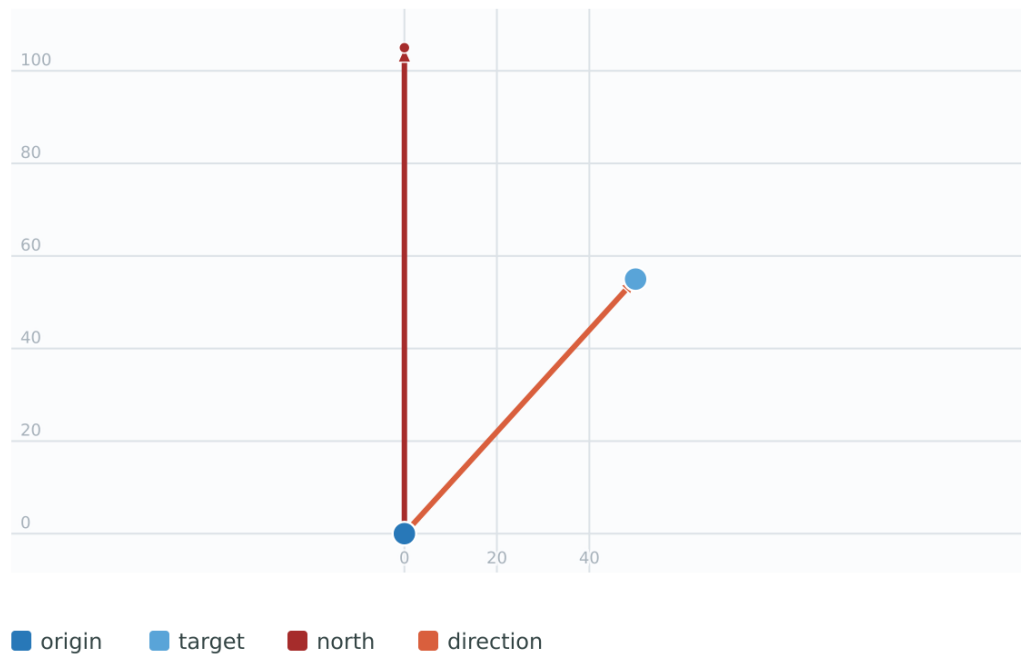
Azimuth from Point(25, 45) to Point(75, 100), shown against the positive Y axis (north).

Code

```
WITH points AS (
  SELECT 'POINT(25 45)::geometry' AS origin,
         'POINT(75 100)::geometry' AS target
)
SELECT round(degrees(ST_Azimuth(origin, target))::numeric, 6) AS degrees,
       ST_MakeLine(origin, ST_Translate(origin, 0, 105)) AS north,
       ST_MakeLine(origin, target) AS direction
FROM points;
```

Output

42.273689	LINESTRING(25 45,25 150)	LINESTRING(25 45,75 100)
-----------	--------------------------	--------------------------

Figure

Reverse azimuth from Point(75, 100) to Point(25, 45).

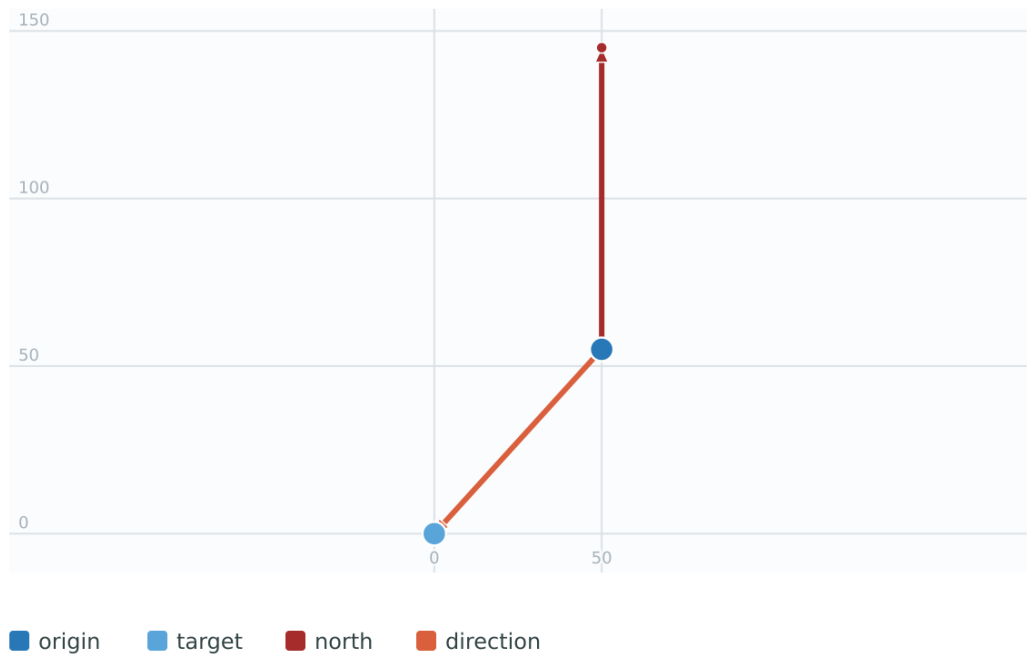
Code

```
WITH points AS (  
  SELECT 'POINT(75 100)::geometry AS origin,  
         'POINT(25 45)::geometry AS target  
)  
SELECT round(degrees(ST_Azimuth(origin, target))::numeric, 6) AS degrees,  
       ST_MakeLine(origin, ST_Translate(origin, 0, 90)) AS north,  
       ST_MakeLine(origin, target) AS direction  
FROM points;
```

Output

```
222.273689 | LINESTRING(75 100,75 190) | LINESTRING(75 100,25 45)
```

Figure



相关信息

[ST_Angle](#), [ST_Point](#), [ST_Translate](#), [ST_Project](#), [PostgreSQL Math Functions](#)

7.12.3 ST_Angle

ST_Angle — 返回由 3 或 4 个点或 2 条定义的向量之间的角度。

Synopsis

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

描述

计算两个向量之间的角度。

形式 **1**：计算点 P1-P2-P3 所成的角度。如果提供了第四个点则计算角度点 P1-P2 和 P3-P4

形式 **2**：计算两个向量 S1-E1 和 S2-E2 之间的角度，由输入的起点和点定义

结果是 0 到 2π 弧度之间的正角。可以使用 PostgreSQL 函数 `Degrees()` 将弧度结果转换为度数。

注意，`ST_Angle(P1,P2,P3) = ST_Angle(P2,P1,P2,P3)`。

可用性：2.5.0

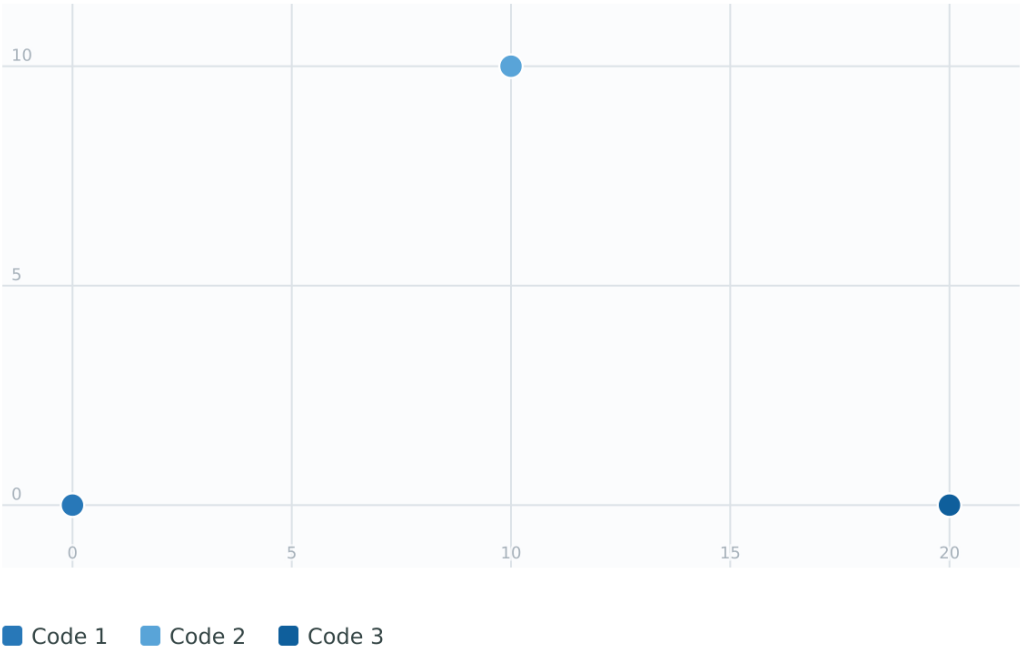
示例

三点之间的角度

Code

```
SELECT degrees(ST_Angle('POINT(0 0)', 'POINT(10 10)', 'POINT(20 0)'));
```

Figure

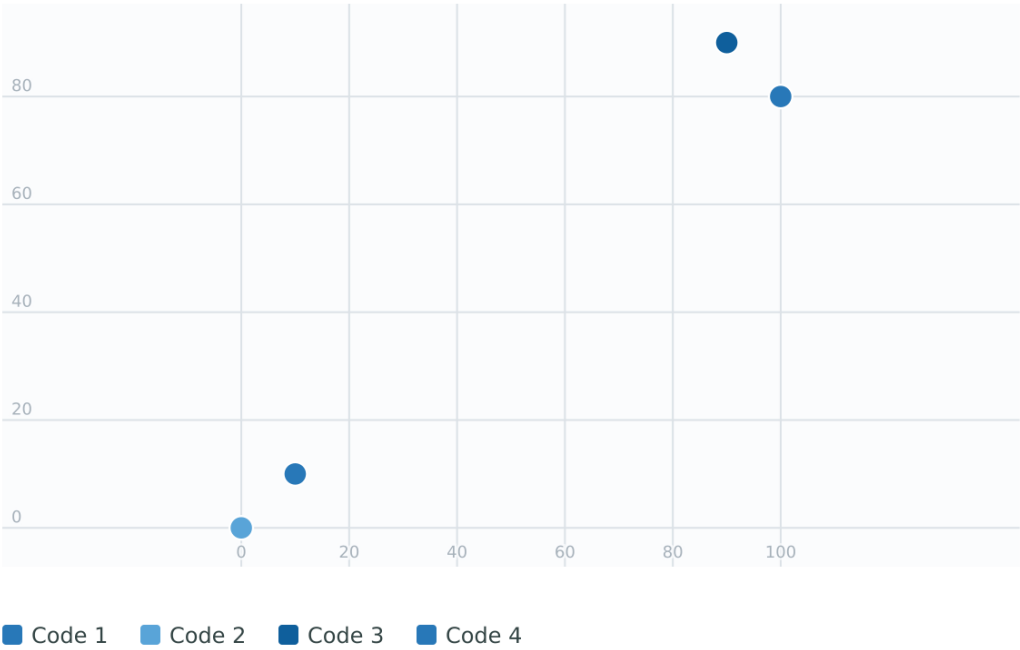


由四个点定义的向量之间的角度

Code

```
SELECT degrees(ST_Angle('POINT (10 10)', 'POINT (0 0)', 'POINT(90 90)', 'POINT (100 80)'));
```

Figure

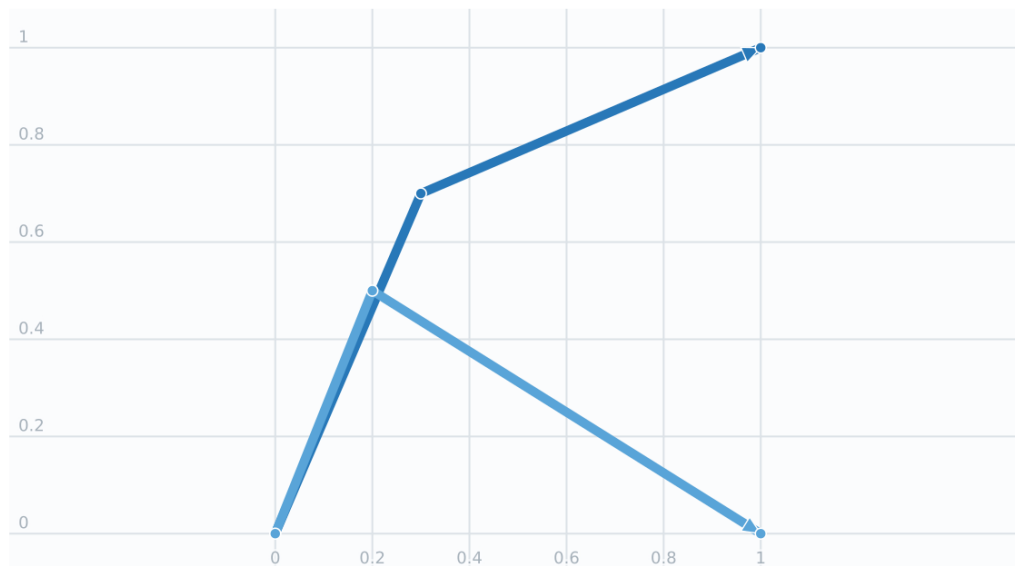


由定义的起点和点定义的向量之间的角度

Code

```
SELECT degrees(ST_Angle('LINESTRING(0 0,0.3 0.7,1 1)', 'LINESTRING(0 0,0.2 0.5,1 0)'));
```

Figure



■ Code 1 ■ Code 2

相关信息

[ST_Azimuth](#)

7.12.4 ST_ClosestPoint

ST_ClosestPoint — 返回 *g1* 上最接近 *g2* 的 2D 点。它是从一个几何体到另一个几何体的最短直线的第一个点。

Synopsis

```
geometry ST_ClosestPoint(geometry geom1, geometry geom2);
geography ST_ClosestPoint(geography geom1, geography geom2, boolean use_spheroid = true);
```

描述

返回 *geom1* 上最接近 *geom2* 的二点。它是几何形状之最短线的第一个点（由 [ST_ShortestLine](#) 算）。



Note

如果您有 3D 几何形状，您可能更喜欢使用 [ST_3DClosestPoint](#)。

增：3.4.0 - 支持地理。

可用性：1.1.0

示例

Point 和 LineString 的最近点是点本身。LineString 和 Point 的最近点是 线上的点。

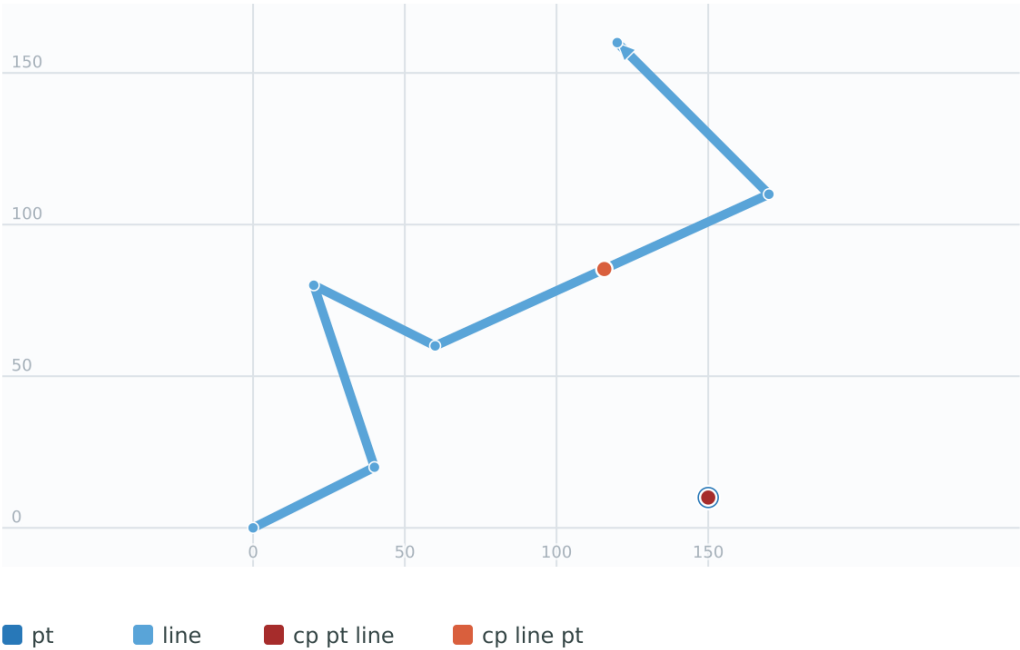
Code

```
SELECT ST_ClosestPoint(pt, line) AS cp_pt_line, 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;
```

Output

cp_pt_line	cp_line_pt
POINT(160 40)	POINT(125.75342465753425 115.34246575342466)

Figure

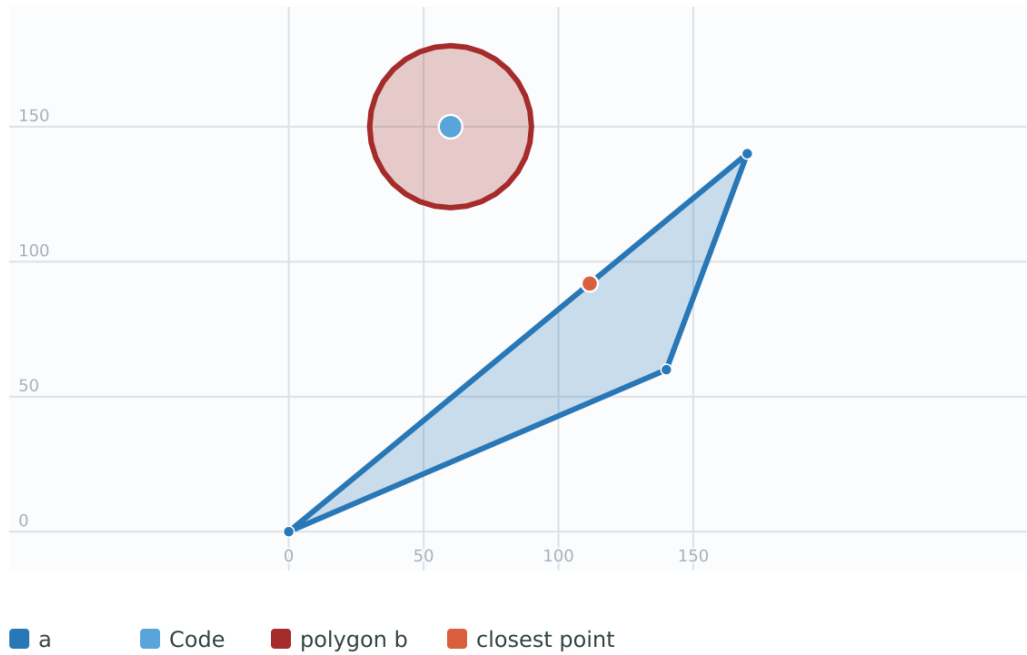


Find the closest point on polygon A to polygon B.

Code

```
WITH source AS (
  SELECT 'POLYGON ((190 150,20 10,160 70,190 150))::geometry AS a,
    ST_Buffer('POINT(80 160)::geometry, 30) AS b
)
SELECT b AS polygon_b,
  ST_ClosestPoint(a, b) AS closest_point
FROM source;
```

Figure



相关信息

[ST_3DClosestPoint](#), [ST_Distance](#), [ST_LongestLine](#), [ST_ShortestLine](#), [ST_MaxDistance](#)

7.12.5 ST_3DClosestPoint

ST_3DClosestPoint — 返回 g1 上最接近 g2 的 3D 点。它是 3D 最短线的第一个点。

Synopsis

geometry **ST_3DClosestPoint**(geometry g1, geometry g2);

描述

返回 g1 上最接近 g2 的 3 点。它是 3D 最短线的第一个点。3D 最短线的 3D 度是 3D 距离。

- ✓ 函数支持 3d 并且不会失 z-index。
- ✓ 函数支持多面体曲面。

可用性: 2.0.0

更改: 2.2.0 - 如果输入 2 个 2D 几何形，返回 2D 点（而不是假设缺失 Z 为 0 的旧行）。在 2D 和 3D 情况下，对于缺失的 Z，Z 不再被假定 0。

示例

Compare the 3D closest point to the 2D closest point for a LineString and a Point.

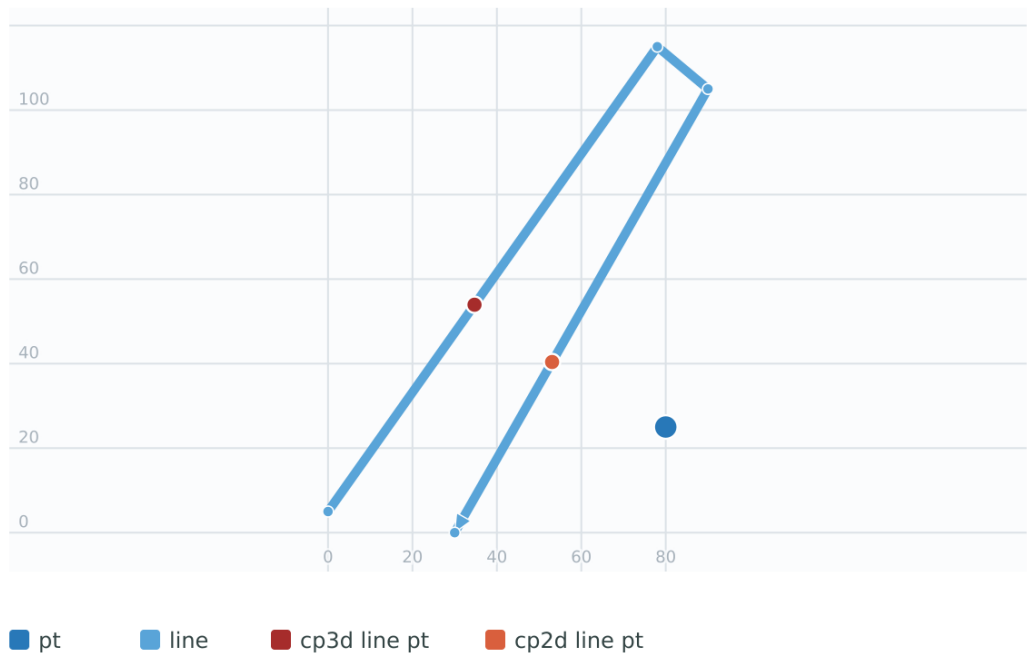
Code

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

Output

-[RECORD 1]-----	
cp3d_line_pt	POINT(54.69937988676193 128.93502291722837 11.547586950660556)
cp2d_line_pt	POINT(73.07692307692307 115.38461538461539)

Figure



Compare the 3D closest point to the 2D closest point for a LineString and a MultiPoint.

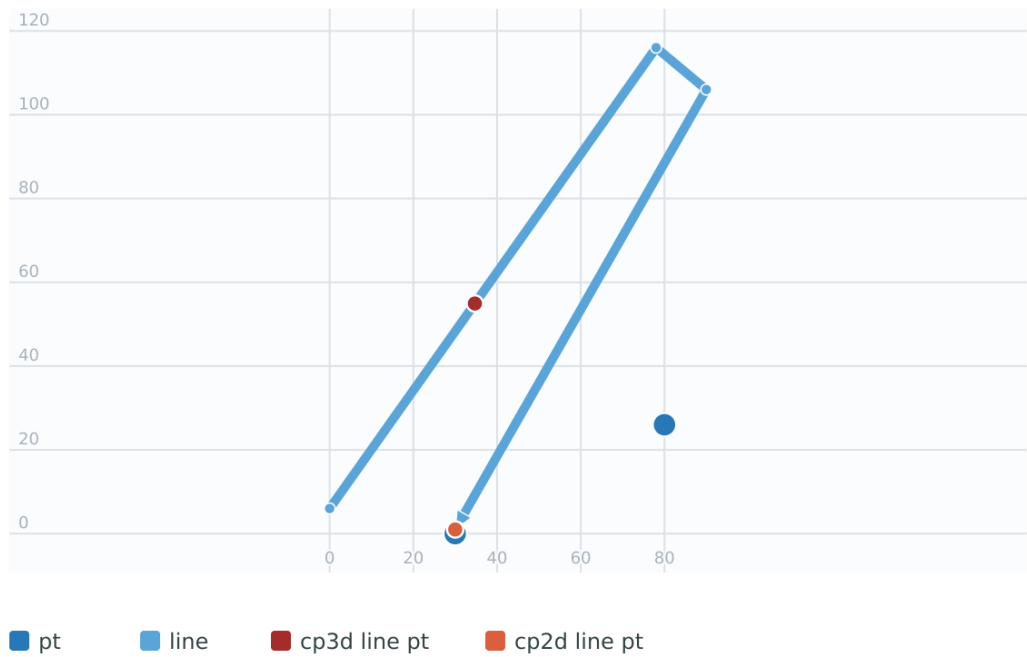
Code

```
WITH input AS (  
  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  
)  
SELECT ST_3DClosestPoint(line, pt) AS cp3d_line_pt,  
       ST_ClosestPoint(line, pt) AS cp2d_line_pt  
FROM input;
```

Output

-[RECORD 1]-----	
cp3d_line_pt	POINT(54.69937988676193 128.93502291722837 11.547586950660556)
cp2d_line_pt	POINT(50 75)

Figure



Compare the 3D closest point to the 2D closest point for a Polygon and a MultiLineString.

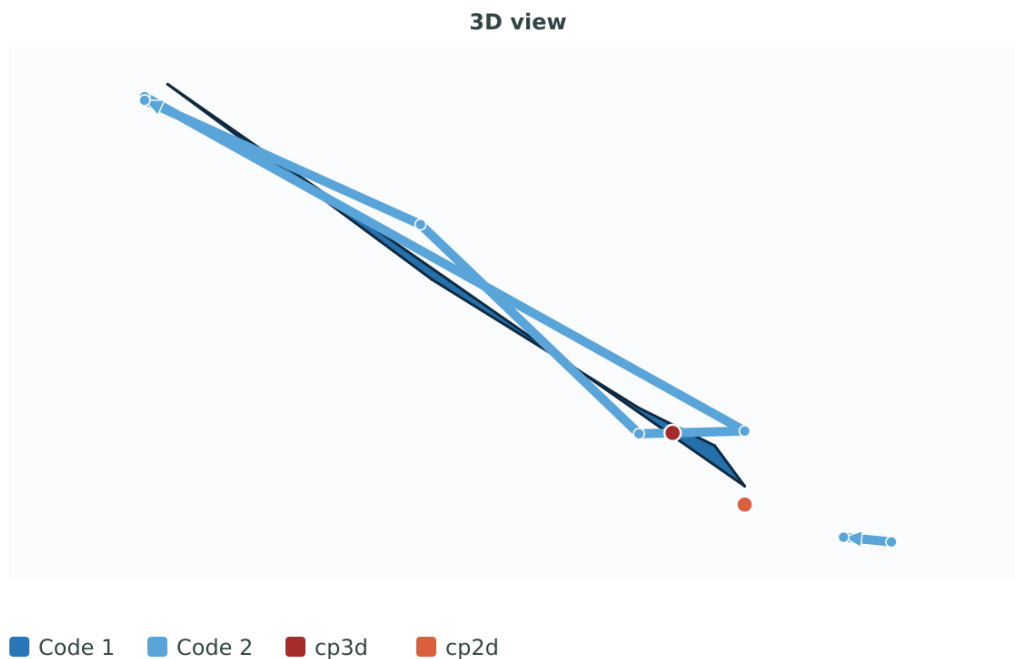
Code

```
WITH input AS (  
  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  
)  
SELECT ST_3DClosestPoint(poly, mline) AS cp3d,  
       ST_ClosestPoint(poly, mline) AS cp2d  
FROM input;
```

Output

-[RECORD 1]-----	
cp3d	POINT(40.45454545454545 53.63636363636364 5)
cp2d	POINT(20 40)

Figure



相关信息

[ST_AsEWKT](#), [ST_ClosestPoint](#), [ST_3DDistance](#), [ST_3DShortestLine](#)

7.12.6 ST_Distance

`ST_Distance` — 返回两个几何或地理对象之间的距离。

Synopsis

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

描述

对于 **geometry**，返回两个几何对象的最小 2D 笛卡儿（平面）距离，以投影单位（空参考单位）表示。

对于 **geography**，默认返回两个地理位置对象的最小大地距离（以米为单位），在由 SRID 确定的球体上进行计算。如果 `use_spheroid` 为 `false`，使用更快的球面计算。

✓ 此方法符合了 [SQL 1.1](#) 的 OGC 功能规范。

✓ 该方法符合了 SQL/MM 规范。SQL-MM 3: 5.1.23

✓ 此方法支持几何字符串和曲线。

可用性：1.5.0 地理支持在 1.5 中引入。提高了平面的速度，以更好地处理大型或多个点几何形状。

增强功能：2.1.0 版本提升了地理（Geography）类型的速度。有关更多信息，参见 [Making Geography faster](#)。

增⊞：2.1.0 - 引入了⊞弯曲几何形状的支持。

增⊞：2.2.0 - 使用 GeographicLib ⊞球体⊞行⊞量，以提高准确性和⊞健性。需要 PROJ >= 4.9.0 才能利用新功能。

更改：3.0.0 - 不再依⊞ SFCGAL。

几何示例

几何示例 - 平面度数 4326 是 WGS 84 ⊞⊞度，⊞位是度。

Code

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

几何示例 - 以米⊞⊞位 (SRID : 3857, 与流行网⊞地⊞上的像素成比例)。尽管⊞⊞关⊞，但可以正确比⊞附近的⊞，⊞使其成⊞ KNN 或 KMeans 等算法的不⊞⊞⊞。

Code

```
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) );
```

几何示例 - ⊞位⊞米 (SRID : 3857, 如上所述，但通⊞ cos(lat) ⊞行校正以考⊞失真)

Code

```
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);
```

几何示例 - ⊞位⊞米 (SRID : 26986 ⊞⊞⊞塞州平面米) (⊞⊞⊞塞州最准确)

Code

```
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) );
```

几何示例 - ⊞位⊞米 (SRID : 2163 美国国家地⊞集等面⊞) (最不准确)

Code

```
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) );
```

几何示例

与几何示例相同，但注意⊞位⊞米 - 使用球体⊞行稍快但不太准确的⊞算。

Code

```
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 ;
```

Output

spheroid_dist	sphere_dist
123.802076746848	123.475736916397

相关信息

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

7.12.7 ST_3DDistance

ST_3DDistance — 返回两个几何图形之间的 3D 笛卡儿最小距离（基于空参考）（以投影单位表示）。

Synopsis

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

描述

返回两个几何图形之间的 3 维最小笛卡儿距离（以投影单位（空参考单位）表示）。

- ✓ 函数支持 3d 并且不会丢失 z-index。
- ✓ 函数支持多面体曲面。
- ✓ 方法遵循了 SQL/MM 规范。SQL-MM ISO/IEC 13249-3

可用性: 2.0.0

更改: 2.2.0 - 在 2D 和 3D 的情况下，对于缺失的 Z，Z 不再被假定 0。

更改: 3.0.0 - SFCGAL 版本已移除

示例

Geometry example - units in meters (SRID: 2163 US National Atlas Equal area) (3D point and line compared 2D point and line) Note that currently no vertical datum support so Z is not transformed and assumed to be same units as final.

Code

```
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;
```

Output

dist_3d	dist_2d
127.295059324629	126.66425605671

Multilinestring and polygon both 3D and 2D distance. Same example as 3D closest point example.

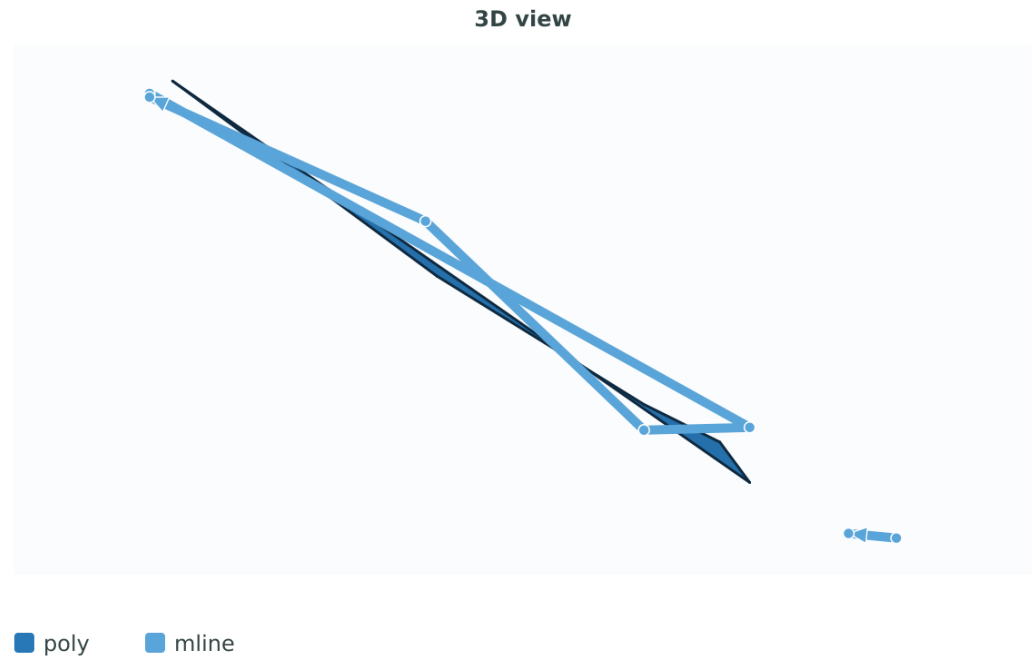
Code

```
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;
```

Output

dist3d	dist2d
0	0

Figure



相关信息

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

7.12.8 ST_DistanceSphere

ST_DistanceSphere — 使用球形地球模型返回两个点/度几何形状之间的最小距离（以米为单位）。

Synopsis

```
float ST_DistanceSphere(geometry geom1, geometry geom2, float8 radius=6371008);
```

描述

返回两个点之间的最小距离（以米为单位）。使用球形地球和从 SRID 定义的球体半径。比 [ST_DistanceSpheroid](#) 更快，但精度较差。PostGIS 1.5 之前的版本不支持点。

可用性：1.5 - 引入了除点之外的其他几何类型的支持。之前的版本适用于点。

更改：2.2.0 在之前的版本中，它曾被称 `ST_Distance_Sphere`

示例

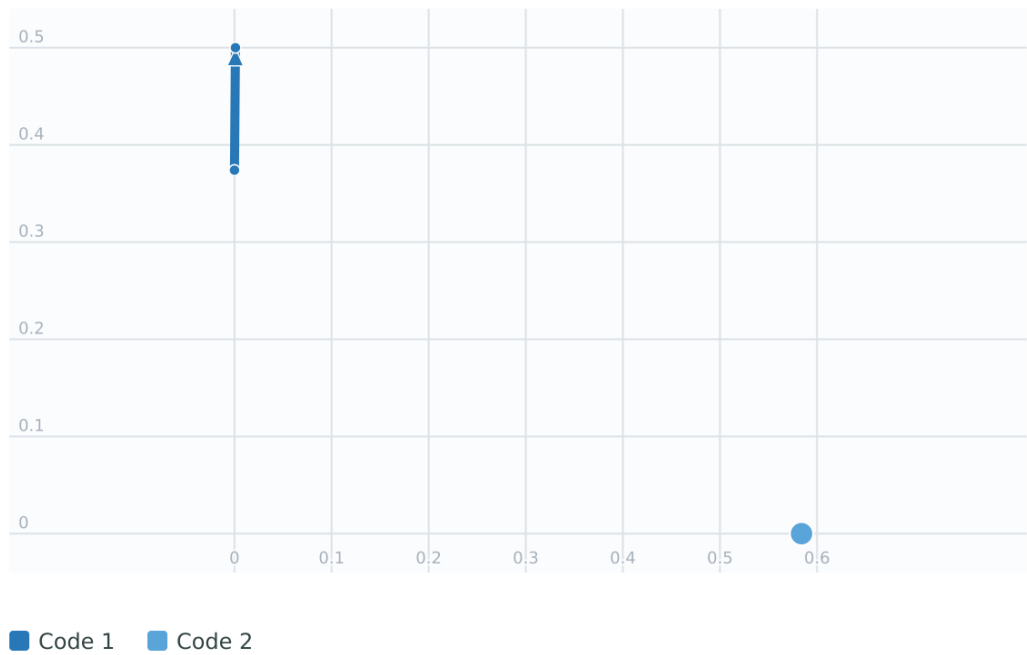
Code

```
WITH sample AS (
  SELECT
    ST_GeomFromText('LINESTRING(-118.584 38.374,-118.583 38.5)', 4326) AS geom,
    ST_GeomFromText('POINT(-118 38)', 4326) AS pt
)
SELECT
  round(ST_DistanceSphere(ST_Centroid(geom), pt)::numeric, 2) AS dist_meters,
  round(ST_Distance(
    ST_Transform(ST_Centroid(geom), 32611),
    ST_Transform(pt, 32611)
  )::numeric, 2) AS dist_utm11_meters,
  round(ST_Distance(ST_Centroid(geom), pt)::numeric, 5) AS dist_degrees,
  round(ST_Distance(
    ST_Transform(geom, 32611),
    ST_Transform(pt, 32611)
  )::numeric, 2) AS min_dist_line_point_meters
FROM sample;
```

Output

dist_meters	dist_utm11_meters	dist_degrees	min_dist_line_point_meters
70424.71	70438.00	0.72900	65871.18

Figure



相关信息

[ST_Distance](#), [ST_DistanceSpheroid](#)

7.12.9 ST_DistanceSpheroid

`ST_DistanceSpheroid` — 使用球体模型返回两个度/度几何形状之间的最小距离。

Synopsis

`float ST_DistanceSpheroid(geometry geom1lonlatA, geometry geom1lonlatB, spheroid measurement_spheroid)`

描述

返回特定球体的两个度/度几何形状之间的最小距离（以米为单位）。参考 [ST_LengthSpheroid](#) 出的球体的解。



Note
此函数不看几何体的 SRID。它假设几何坐标基于提供的球体。

可用性：1.5 - 引入了除点之外的其他几何类型的支持。之前的版本适用于点。

更改：2.2.0 在之前的版本中，称 `ST_Distance_Spheroid`

示例

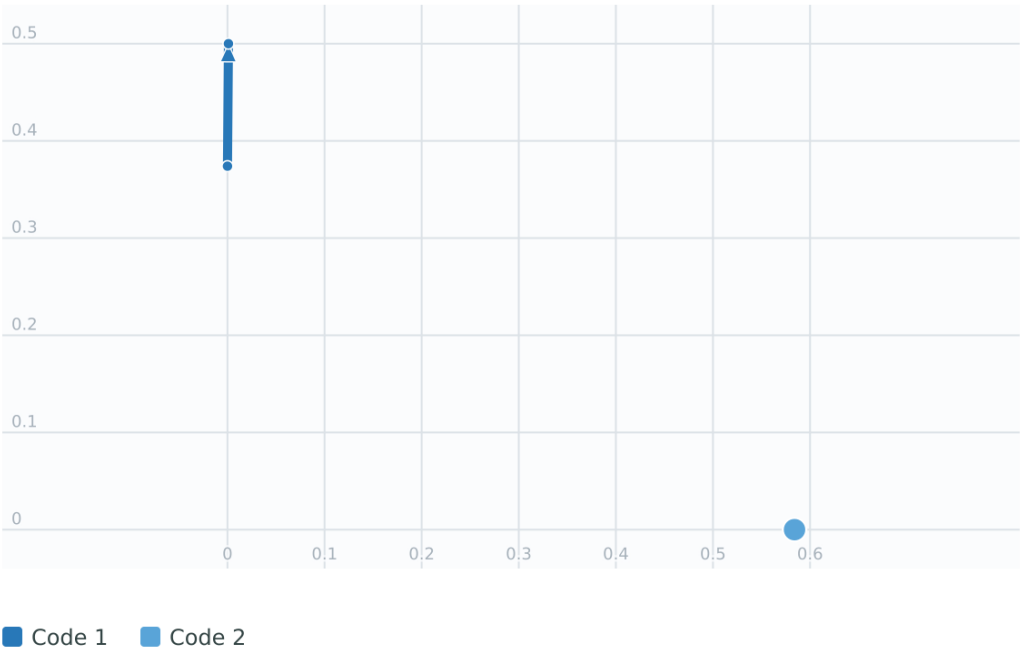
Code

```
WITH input AS (  
  SELECT ST_Centroid(  
    ST_GeomFromText(  
      'LINESTRING(-118.584 38.374,-118.583 38.5)', 4326  
    )  
  ) AS source,  
  ST_GeomFromText('POINT(-118 38)', 4326) AS target  
)  
SELECT round(  
  ST_DistanceSpheroid(  
    source,  
    target,  
    'SPHEROID["WGS 84",6378137,298.257223563]'  
  )::numeric,  
  2  
) AS dist_meters_spheroid,  
round(ST_DistanceSphere(source, target)::numeric, 2)  
  AS dist_meters_sphere,  
round(  
  ST_Distance(  
    ST_Transform(source, 32611),  
    ST_Transform(target, 32611)  
  )::numeric,  
  2  
) AS dist_utm11_meters  
FROM input;
```

Output

dist_meters_spheroid	dist_meters_sphere	dist_utm11_meters
70454.92	70424.71	70438.00

Figure



相关信息

[ST_Distance](#), [ST_DistanceSphere](#)

7.12.10 ST_FrechetDistance

ST_FrechetDistance — 返回两个几何图形之间的 Fréchet 距离。

Synopsis

```
float ST_FrechetDistance(geometry g1, geometry g2, float densifyFrac = -1);
```

描述

基于离散 **Fréchet 距离**，该算法限于两个几何图形的离散点的 Fréchet 距离的算法。Fréchet 距离是曲线之间相似性的度量，考虑了曲线上点的位置和顺序。因此它通常比 Hausdorff 距离更好。

当指定可选的 densifyFrac 时，该函数在计算离散 Fréchet 距离之前进行分段致密化。densifyFrac 参数用于致密每个段的分数。每个段将被分成多个等长的子段，其长度的分数最接近指定的分数。

该位采用几何空参考系的位。



Note

当前的版本支持点作为离散位置。它可以扩展到允许使用任意密度的点。



Note

我指定的 densifyFrac 越小，我得到的 Fréchet 距离就越准确。但是，计算量和内存使用量随着子段数量的平方而增加。

该函数是由 GEOS 模块实现的。

可用性：2.4.0 - 需要 GEOS >= 3.7.0

示例

Both lines are traversed from left to right. The first line is straight, while the second bows upward through its middle vertex. The point ordering matters, so keeping both inputs in the same direction makes the comparison clearer.

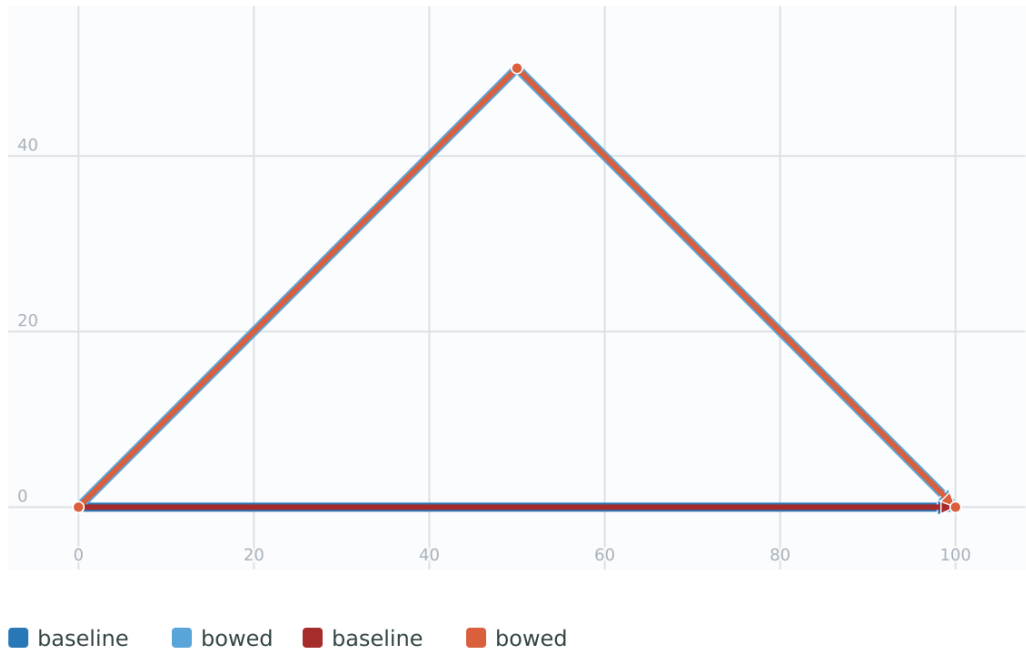
Code

```
WITH lines AS (
  SELECT 'LINESTRING (0 0,100 0)::geometry' AS baseline,
         'LINESTRING (0 0,50 50,100 0)::geometry' AS bowed
)
SELECT baseline AS baseline,
       bowed AS bowed,
       ST_FrechetDistance(baseline, bowed) AS discrete_distance,
       ST_FrechetDistance(baseline, bowed, 0.5) AS densified_distance
FROM lines;
```

Output

```
-[ RECORD 1 ]-----+-----
baseline      | LINESTRING(0 0,100 0)
bowed         | LINESTRING(0 0,50 50,100 0)
discrete_distance | 70.7106781186548
densified_distance | 50
```

Figure



相关信息

[ST_HausdorffDistance](#)

7.12.11 ST_HausdorffDistance

ST_HausdorffDistance — 返回两个几何形状之间的 Hausdorff 距离。

Synopsis

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

描述

返回两个几何形状之间的 **Hausdorff 距离**。Hausdorff 距离是两个几何形状相似或不相似程度的度量。

该函数计算“离散 Hausdorff 距离”。它是在几何上的离散点计算的 Hausdorff 距离。可以指定 *densifyFrac* 参数，以便在计算离散 Hausdorff 距离之前通过致密分段来提供更准确的答案。每个段被分成多个等度的子段，其段度的分数最接近指定的分数。

它采用几何空参考系的位置。



Note

该算法不等同于标准 Hausdorff 距离。然而，它计算出的近似值对于大部分有用案例都是正确的。一种重要的情况是两串彼此大致平行且长度大致相等。它是路匹配的有用指标。

可用性：1.1.0

示例

Hausdorff distance and minimum distance between two lines. The Hausdorff witness segment from the original illustration is retained and its length is checked against the computed distance.

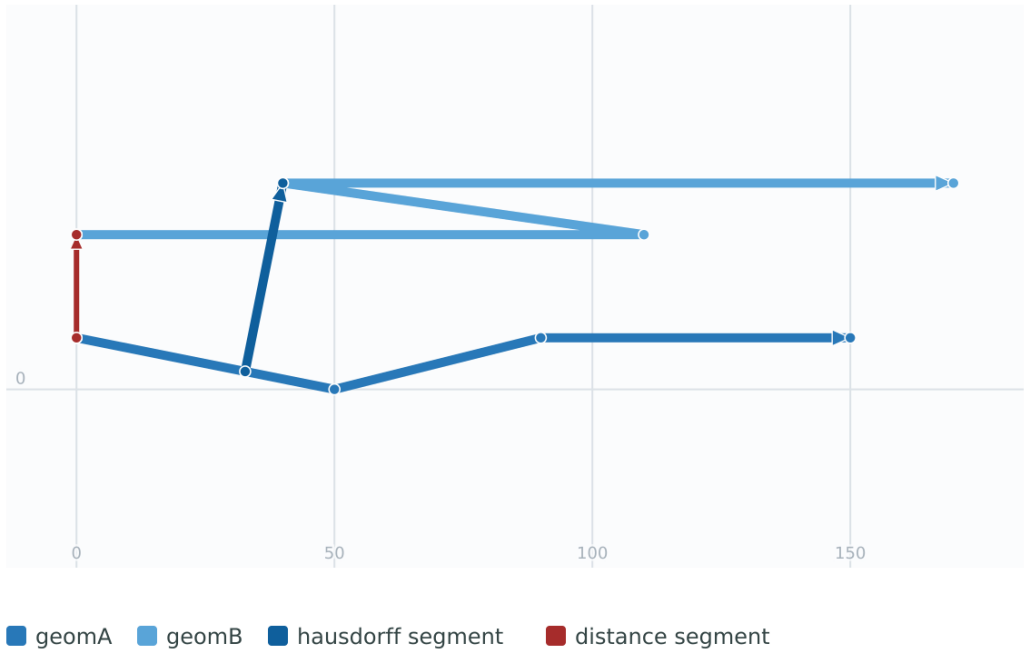
Code

```
WITH lines AS (  
  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,  
         'LINESTRING (52.69230769230769 63.46153846153846,60 100)>:::geometry AS hausdorff_segment  
)  
SELECT round(ST_HausdorffDistance(geomA, geomB)::numeric, 6) AS hausdorff_distance,  
       round(ST_Distance(geomA, geomB)::numeric, 6) AS distance,  
       round(ST_Length(hausdorff_segment)::numeric, 6) AS witness_length,  
       ST_ShortestLine(geomA, geomB) AS distance_segment  
FROM lines;
```

Output

37.262066 | 20.000000 | 37.262066 | LINESTRING(20 70,20 90)

Figure

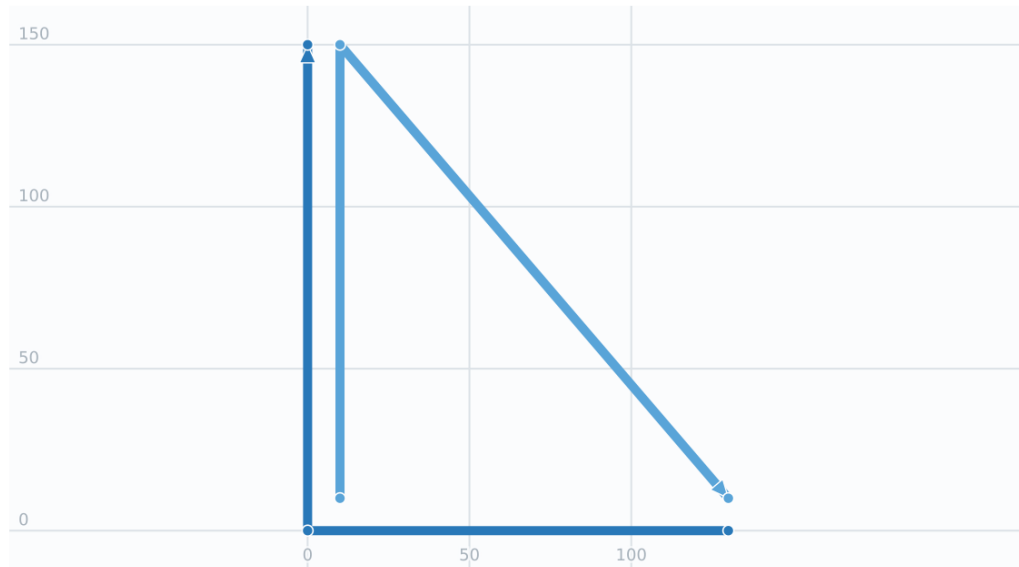


示例：致密化的 Hausdorff 距离。

Code

```
SELECT ST_HausdorffDistance(
  'LINESTRING (130 0,0 0,0 150)::geometry',
  'LINESTRING (10 10,10 150,130 10)::geometry',
  0.5);
```

Figure



■ Code 1 ■ Code 2

示例：对于每个建筑物，找到最能代表它的地块。首先，我要求地块与建筑物几何形状相交。**DISTINCT ON** 保证我每块建筑只列出一行。**ORDER BY .. ST_HausdorffDistance** 与建筑物最相似的地块。

Code

```
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);
```

相关信息

[ST_FrechetDistance](#)

7.12.12 ST_Length

ST_Length — 返回线性几何体的二角度。

Synopsis

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

描述

对于几何类型：如果几何是 `LineString`、`MultiLineString`、`ST_Curve`、`ST_MultiCurve`，返回几何的 2D 笛卡儿长度。对于面几何形状，返回 0；改用 `ST_Perimeter`。长度单位由几何的空参考系确定。

对于 `geography` 类型：计算使用反大地测量方法，长度单位是米。所用球体由 `SRID` 指定；如果没有 `SRID`，默认使用 `WGS84`。若 `use_spheroid = false`，计算基于球体模型而非地球体。

目前对于几何体，是 `ST_Length2D` 的别名，但它可能会更改以支持更高的长度。

更改：2.0.0 重大更改 - 在之前的版本中，将此用于地理类型的多/多边形将为您提供多边形/多多边形的周长。在 2.0.0 中，它已更改以返回 0 以符合几何行。如果您想要多边形的周长，请使用 `ST_Perimeter`



Note

对于地理，计算默认使用球体模型。要使用更快但不太准确的球面计算，使用 `ST_Length(gg,false);`

✓ 此方法符合了 **SQL 1.1 的 OGC 功能规范**。s2.1.5.1

✓ 此方法符合了 **SQL/MM 规范**。SQL-MM 3: 7.1.2, 9.3.4

可用性：1.5.0 地理支持在 1.5 中引入。

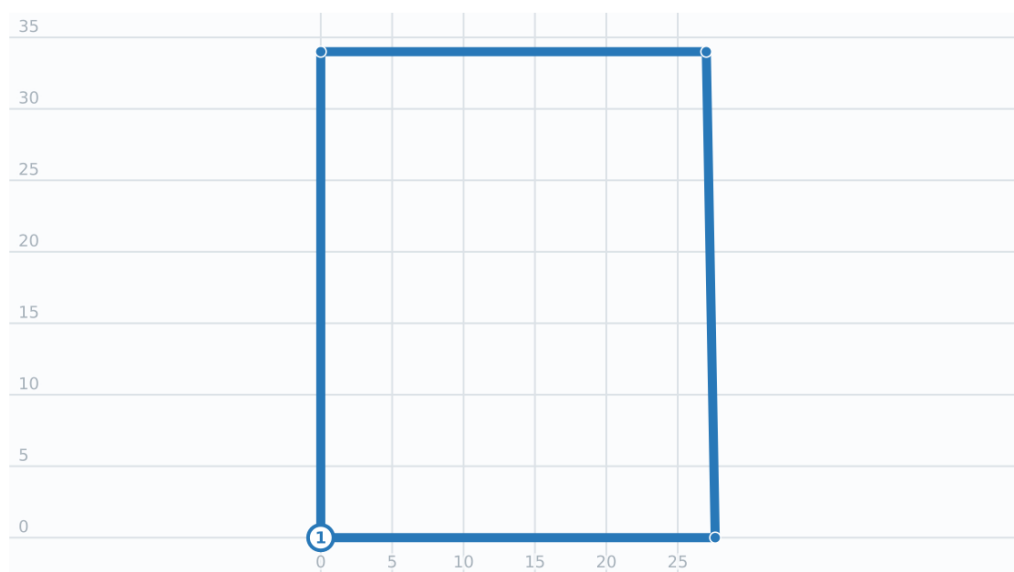
几何示例

返回字符串的长度（以英尺为单位）。注意，单位是英尺，因为 `EPSG:2249` 是马萨诸塞州平面英尺

Code

```
SELECT ST_Length(ST_GeomFromText(
'LINESTRING(743238 2967416,743238 2967450,743265 2967450,
743265.625 2967416,743238 2967416)',
2249));
```

Figure



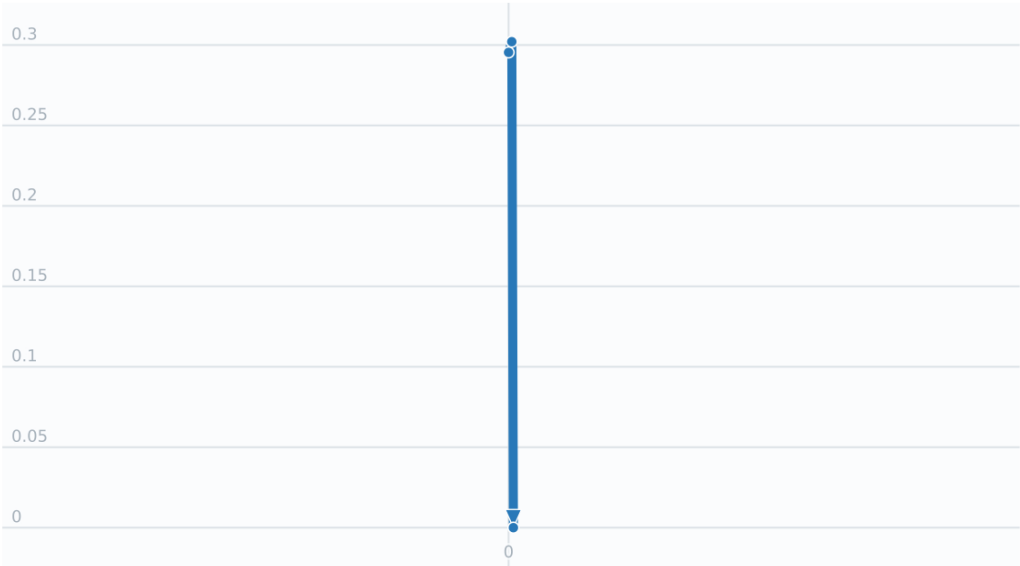
■ Code

This example transforms a WGS 84 LineString to Massachusetts State Plane meters.

Code

```
SELECT ST_Length(ST_Transform(
    'SRID=4326;LINESTRING(-72.1260 42.45,-72.1240 42.45666,-72.123 42.1546)')::geometry,
    26986
);
```

Figure



Code

几何示例

WGS 84 地理↔返回↔度

This example uses the default spheroid calculation.

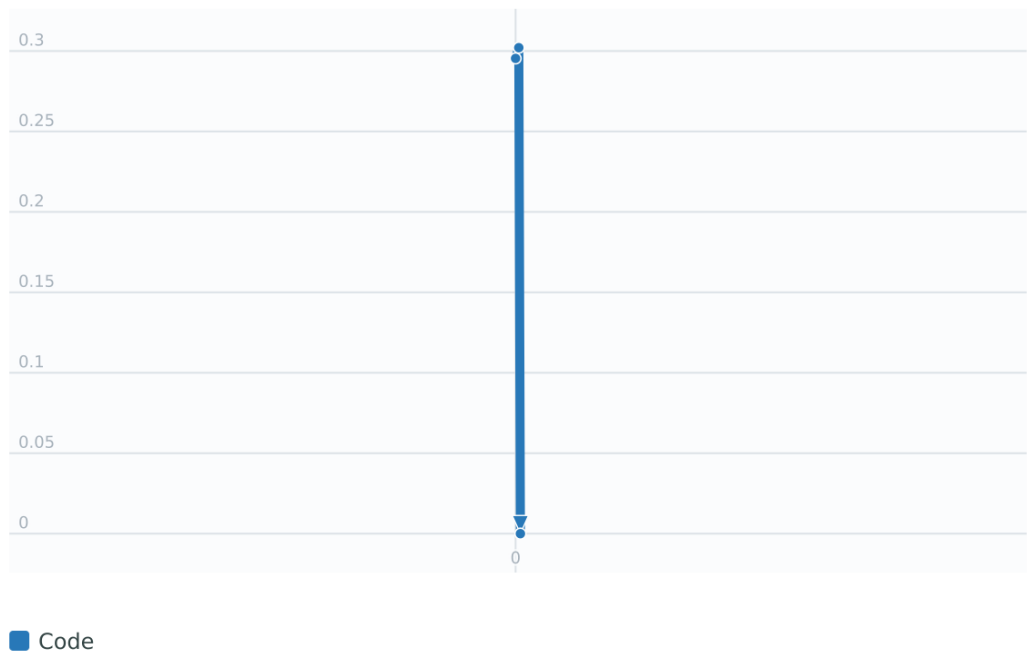
Code

```
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;
```

Output

length_spheroid	length_sphere
34310.5703627284	34346.20609607416

Figure



相关信息

[ST_GeographyFromText](#), [ST_GeomFromEWKT](#), [ST_LengthSpheroid](#), [ST_Perimeter](#), [ST_Transform](#)

7.12.13 ST_Length2D

ST_Length2D — 返回线性几何体的二维长度。ST_Length 的同名

Synopsis

```
float ST_Length2D(geometry a_2dlinestring);
```

描述

如果几何图形是线串或多线串，返回几何图形的 2D 长度。是 ST_Length 的同名

相关信息

[ST_Length](#), [ST_3DLength](#)

7.12.14 ST_3DLength

ST_3DLength — 返回线性几何体的 3D 长度。

Synopsis

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

描述

如果几何体是 `LineString` 或 `MultiLineString`，它返回几何体的 3 维或 2 维长度。对于 2 维，它将返回 2 维长度（与 `ST_Length` 和 `ST_Length2D` 相同）

✓ 函数支持 3d 并且不会丢失 z-index。

✓ 方法符合了 SQL/MM 规范。SQL-MM IEC 13249-3: 7.1, 10.3

更改：2.0.0 在之前的版本中，它曾被称作 `ST_Length3D`

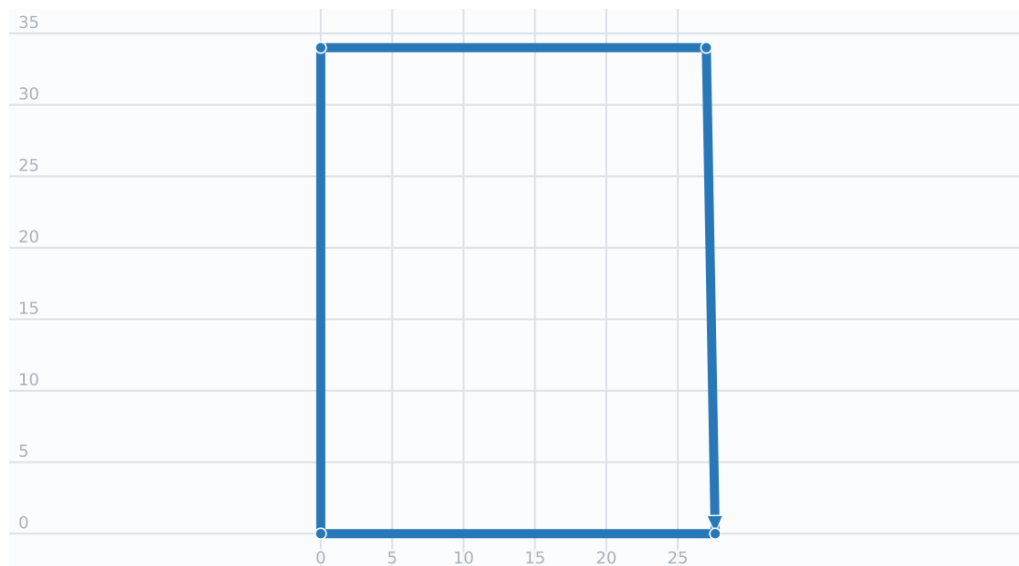
示例

返回 3D 长度的度（以英尺为单位）。注意，它是英尺，因为 EPSG:2249 是塞州平面英尺

Code

```
SELECT ST_3DLength(ST_GeomFromText(
'LINESTRING(743238 2967416 1,743238 2967450 1,743265 2967450 3,
743265.625 2967416 3,743238 2967416 3)',
2249));
```

Figure



■ Code

相关信息

[ST_Length](#), [ST_Length2D](#)

7.12.15 ST_LengthSpheroid

ST_LengthSpheroid — 返回球体上度/度几何体的 2D 或 3D 度/周。

Synopsis

```
float ST_LengthSpheroid(geometry a_geometry, spheroid a_spheroid);
```

描述

Calculates the length or perimeter of a geometry on a spheroid. This is useful if the coordinates of the geometry are in longitude/latitude and a length is desired without reprojection. The spheroid is specified by a text value as follows:

```
SPHEROID[<NAME>,<SEMI-MAJOR AXIS>,<INVERSE FLATTENING>]
```

例如：

```
SPHEROID["GRS_1980",6378137,298.257222101]
```

可用性：1.2.2

更改：2.2.0 在之前的版本中，称 ST_Length_Spheroid 并具有名 ST_3DLength_Spheroid



函数支持 3d 并且不会失 z-index。

示例

Measure a table geometry directly on a spheroid when the coordinates are stored in longitude and latitude.

Code

```
SELECT ST_LengthSpheroid(
    geometry_column,
    'SPHEROID["GRS_1980",6378137,298.257222101]'
)
FROM geometry_table;
```

This example measures a 2D MultiLineString and each component separately on the same spheroid.

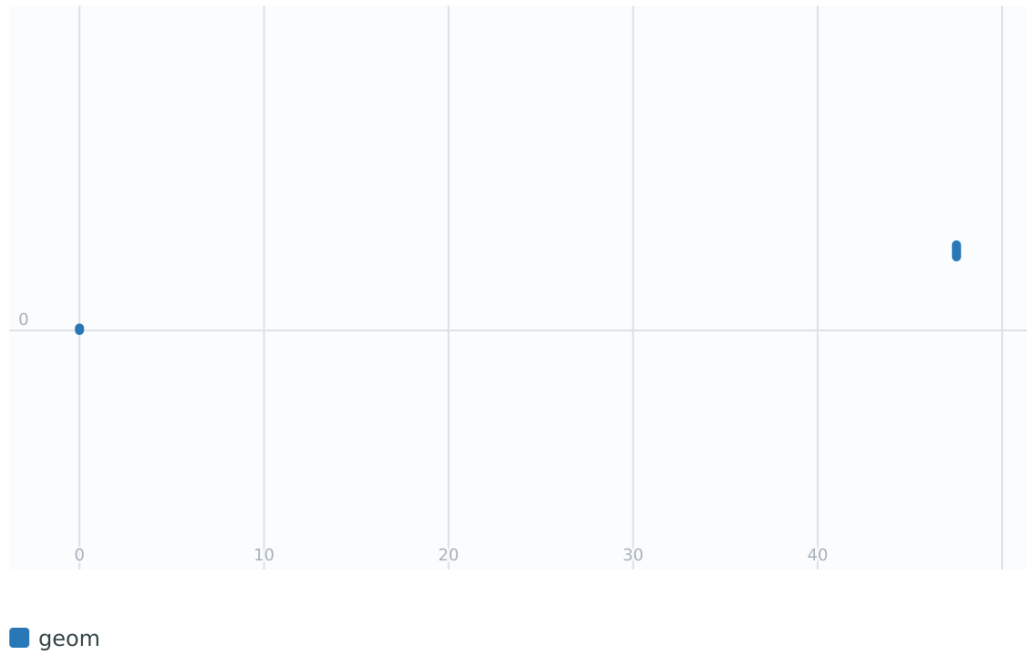
Code

```
WITH sample AS (
    SELECT 'MULTILINESTRING((-118.584 38.374,-118.583 38.5),
        (-71.05957 42.3589,-71.061 43))'::geometry AS geom,
        CAST('SPHEROID["GRS_1980",6378137,298.257222101]' AS spheroid) AS sph_m
)
SELECT ST_LengthSpheroid(geom, sph_m) AS tot_len,
    ST_LengthSpheroid(ST_GeometryN(geom, 1), sph_m) AS len_line1,
    ST_LengthSpheroid(ST_GeometryN(geom, 2), sph_m) AS len_line2
FROM sample;
```

Output

tot_len	len_line1	len_line2
85204.52077117894	13986.87252824332	71217.64824293563

Figure



This example uses 3D input; Z contributes to the measured result.

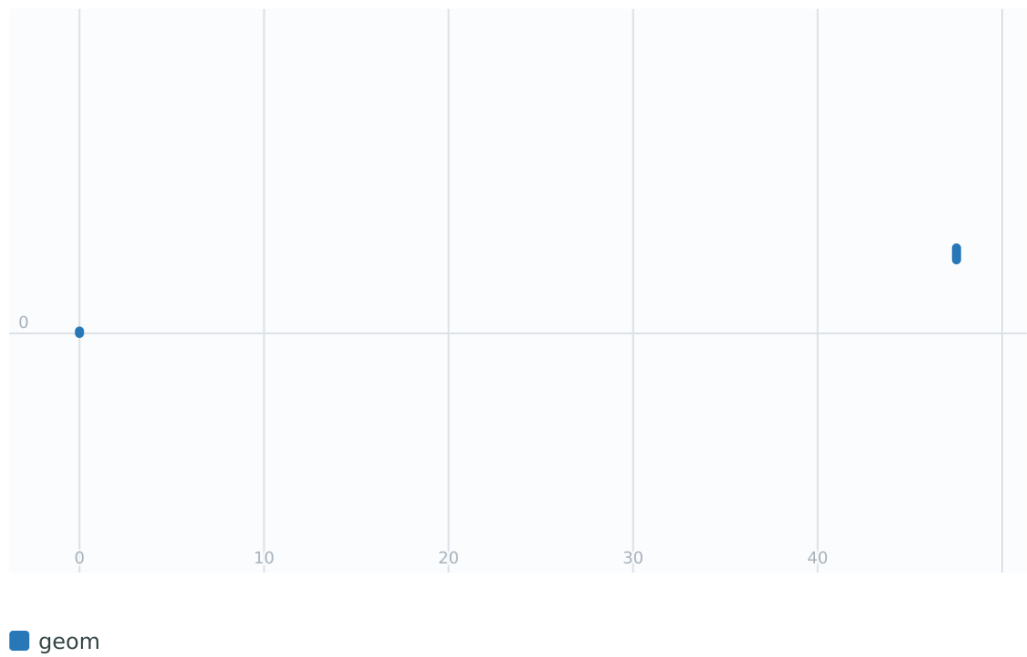
Code

```
WITH sample AS (  
  SELECT 'MULTILINESTRING((-118.584 38.374 20,-118.583 38.5 30),  
    (-71.05957 42.3589 75,-71.061 43 90))'::geometry AS geom,  
    CAST('SPHEROID["GRS_1980",6378137,298.257222101]' AS spheroid) AS sph_m  
)  
SELECT ST_LengthSpheroid(geom, sph_m) AS tot_len,  
  ST_LengthSpheroid(ST_GeometryN(geom, 1), sph_m) AS len_line1,  
  ST_LengthSpheroid(ST_GeometryN(geom, 2), sph_m) AS len_line2  
FROM sample;
```

Output

tot_len	len_line1	len_line2
85204.52592562366	13986.876103023424	71217.64982260024

Figure



相关信息

[ST_GeometryN](#), [ST_Length](#)

7.12.16 ST_LongestLine

ST_LongestLine — 返回两个几何形状之间的二点最远点。

Synopsis

```
geometry ST_LongestLine(geometry g);
geometry ST_LongestLine(geometry g1, geometry g2);
```

描述

返回两个几何点之间的二点最远点。返回的行从 g1 开始，到 g2 结束。
最远的点是出现在两个点之间。如果找到多个点，函数将返回第一个最远的点。直线的长度等于 [ST_MaxDistance](#) 返回的距离。
If only one geometry is provided, or g1 and g2 are the same geometry, returns the line between the two vertices farthest apart in the geometry. The endpoints of the line lie on the circle computed by [ST_MinimumBoundingCircle](#).

可用性 : 1.1.0
Enhanced: 3.7.0 - support for a single geometry input.

示例

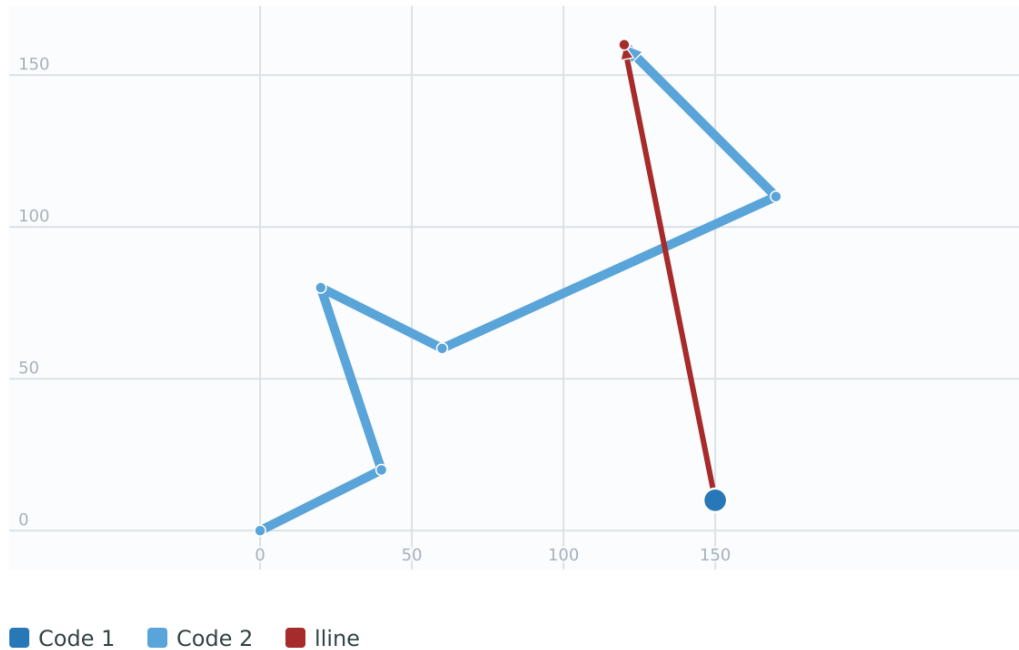
Longest line between a Point and a LineString.

Code

```
SELECT ST_LongestLine(
  'POINT (160 40)',
  'LINESTRING (10 30,50 50,30 110,70 90,180 140,130 190)' ) AS lline;
```

Output

```
LINESTRING(160 40,130 190)
```

Figure

Longest line between two polygons.

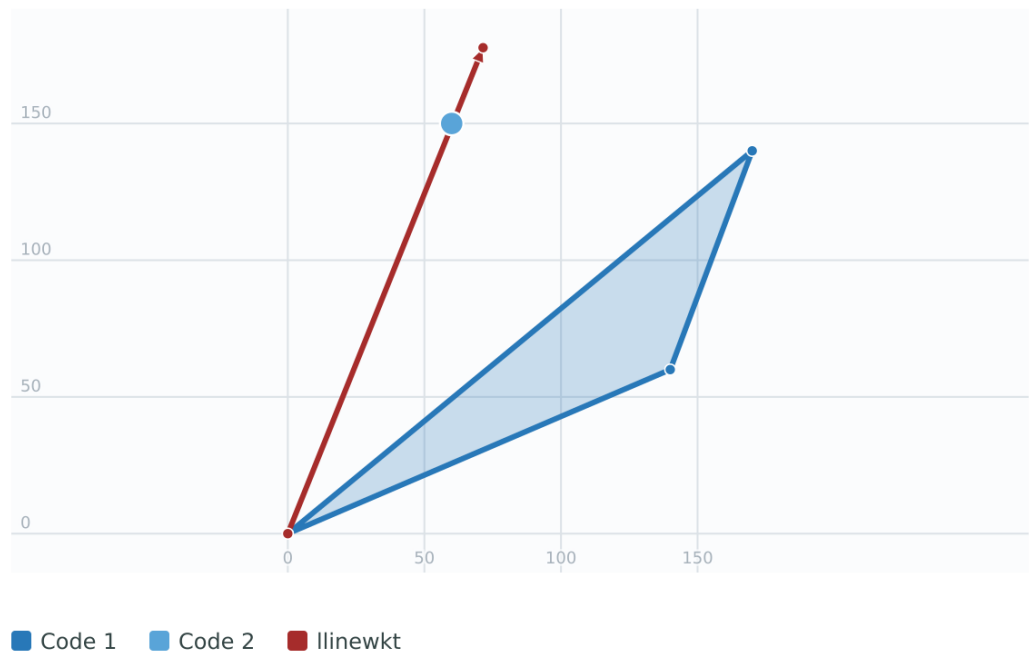
Code

```
SELECT ST_LongestLine(
  'POLYGON ((190 150,20 10,160 70,190 150))',
  ST_Buffer('POINT(80 160)', 30)
) AS llinewkt;
```

Output

```
LINESTRING(20 10,91.4805029709527 187.7163859753386)
```

Figure



跨个几何体的最。的度等于最大距离。的端点位于最小界上。

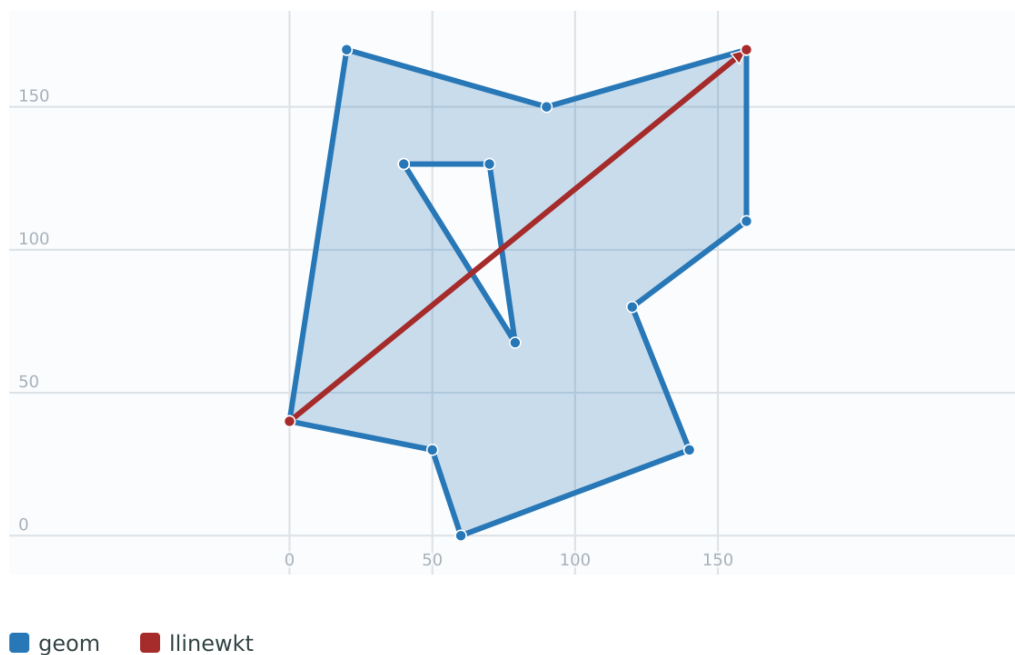
Code

```
SELECT ST_LongestLine(geom) AS llinewkt,
       ST_MaxDistance(geom) AS max_dist,
       ST_Length(ST_LongestLine(geom)) AS lenll
FROM (SELECT 'POLYGON ((40 180,110 160,180 180 120,140 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;
```

Output

llinewkt	max_dist	lenll
LINESTRING(20 50,180 180)	206.15528128088303	206.15528128088303

Figure



相关信息

[ST_MaxDistance](#), [ST_ShortestLine](#), [ST_3DLongestLine](#), [ST_MinimumBoundingCircle](#)

7.12.17 ST_3DLongestLine

ST_3DLongestLine — 返回两个几何体之间的 3D 最长直线

Synopsis

```
geometry ST_3DLongestLine(geometry g1, geometry g2);
```

描述

返回两个几何体之间的 3D 最长直线。如果有多个，该函数将返回第一个最长的。返回的行以 g1 开始，以 g2 结束。该 3D 长度等于 [ST_3DMaxDistance](#) 返回的距离。

可用性: 2.0.0

更改: 2.2.0 - 如果输入 2 个 2D 几何体，返回 2D 点（而不是假设缺失 Z 为 0 的旧行）。在 2D 和 3D 情况下，对于缺失的 Z，Z 不再被假定 0。

✓ 该函数支持 3d 并且不会丢失 z-index。

✓ 该函数支持多面体曲面。

示例

Longest line between a LineString and a Point, comparing the 3D and 2D results.

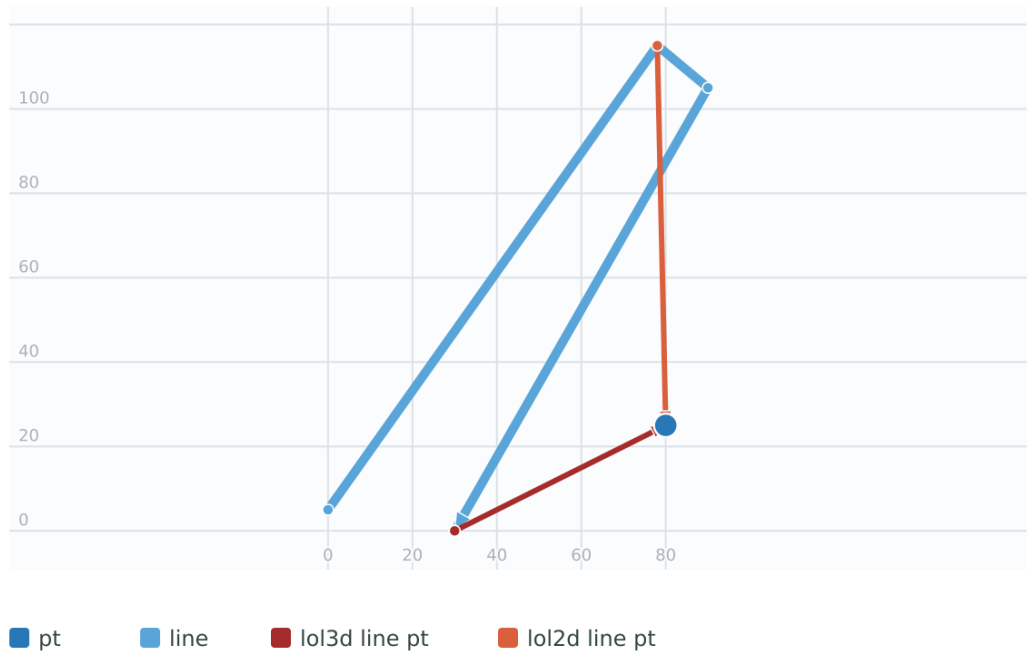
Code

```
SELECT ST_3DLongestLine(line, pt) AS lol3d_line_pt, ST_LongestLine(line, pt) AS lol2d_line_pt
FROM (
  SELECT 'POINT(100 100 30)::geometry' AS pt,
         'LINESTRING (20 80 20,98 190 1,110 180 3,50 75 1000)::geometry' AS line
) AS foo;
```

Output

lol3d_line_pt	lol2d_line_pt
LINESTRING(50 75 1000,100 100 30)	LINESTRING(98 190,100 100)

Figure



Longest line between a LineString and a MultiPoint, comparing the 3D and 2D results.

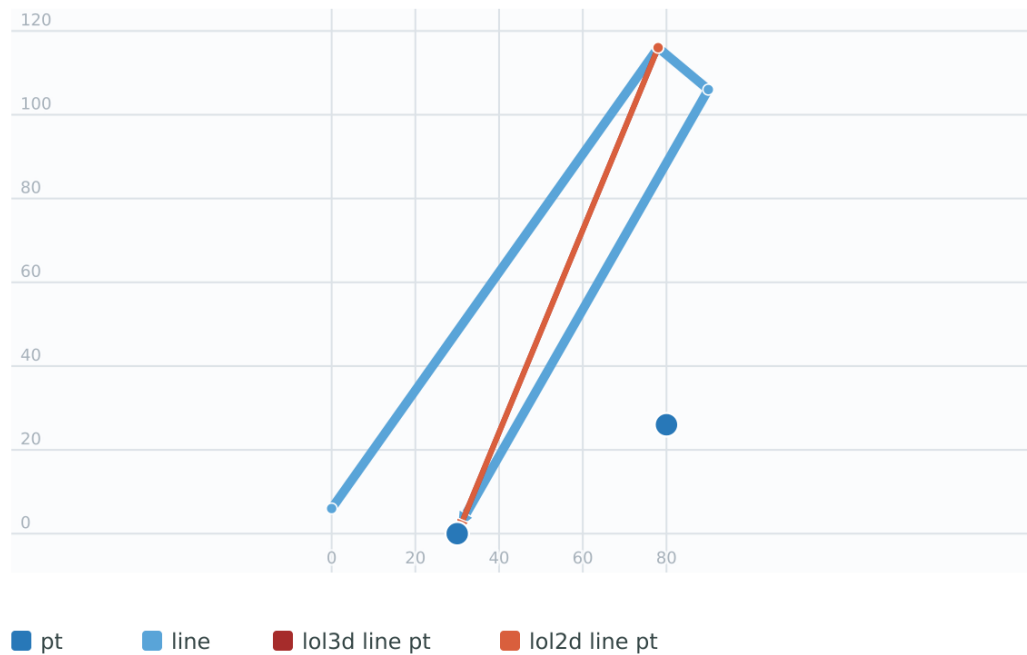
Code

```
SELECT ST_3DLongestLine(line, pt) AS lol3d_line_pt, 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;
```

Output

lol3d_line_pt	lol2d_line_pt
LINESTRING(98 190 1,50 74 1000)	LINESTRING(98 190,50 74)

Figure



Longest line between a Polygon and a MultiLineString, comparing the 3D and 2D results.

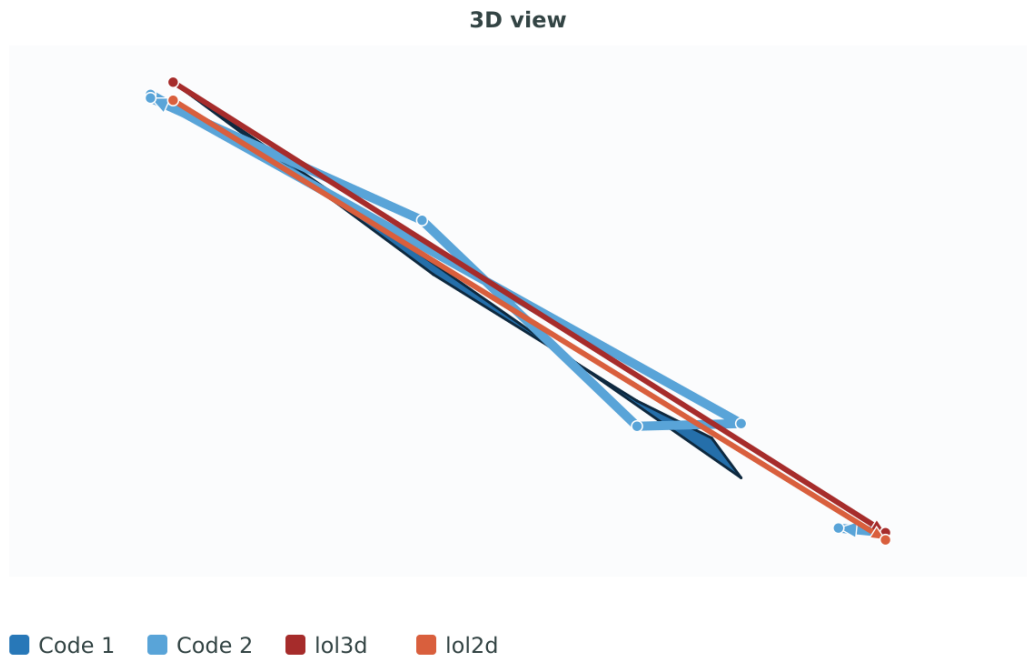
Code

```
SELECT ST_3DLongestLine(poly, mline) AS lol3d, 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;
```

Output

lol3d	lol2d
LINESTRING(175 150 5,1 10 2)	LINESTRING(175 150,1 10)

Figure



相关信息

[ST_3DClosestPoint](#), [ST_3DDistance](#), [ST_LongestLine](#), [ST_3DShortestLine](#), [ST_3DMaxDistance](#)

7.12.18 ST_MaxDistance

`ST_MaxDistance` — 返回两个几何图形之间的二最大距离（以投影单位表示）。

Synopsis

```
float ST_MaxDistance(geometry g);  
float ST_MaxDistance(geometry g1, geometry g2);
```

描述

返回两个几何图形之间的二最大距离（以投影单位表示）。最大距离是出现在两个点之间。是 [ST_LongestLine](#) 返回的线的长度。

If only one geometry is provided, or g1 and g2 are the same geometry, returns the distance between the two vertices farthest apart in that geometry.

可用性：1.1.0

Enhanced: 3.7.0 - support for a single geometry input.

示例

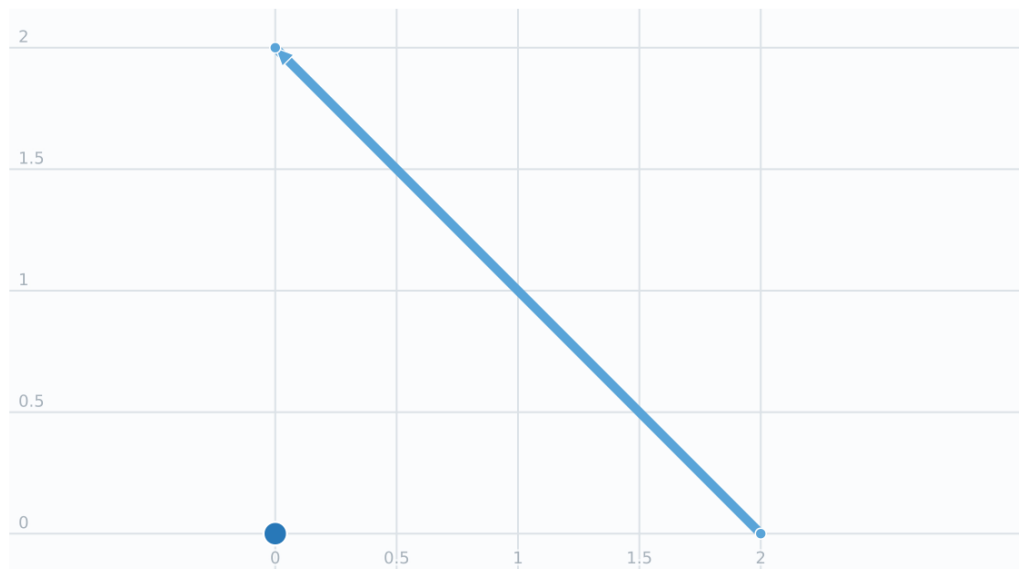
点与线之间的最大距离。

Code

```
SELECT ST_MaxDistance('POINT(0 0)::geometry, 'LINESTRING ( 2 0,0 2 ) '::geometry);
```

Output

2

Figure

■ Code 1 ■ Code 2

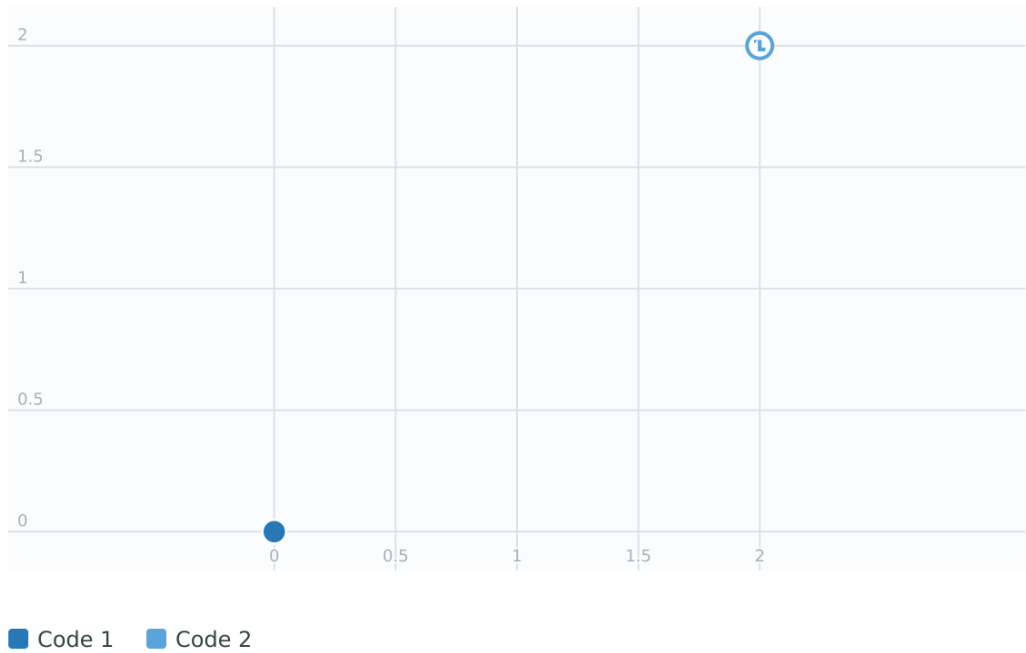
Code

```
SELECT ST_MaxDistance('POINT(0 0)::geometry, 'LINESTRING ( 2 2,2 2 ) '::geometry);
```

Output

2.82842712474619

Figure



两个几何体点之间的最大距离。

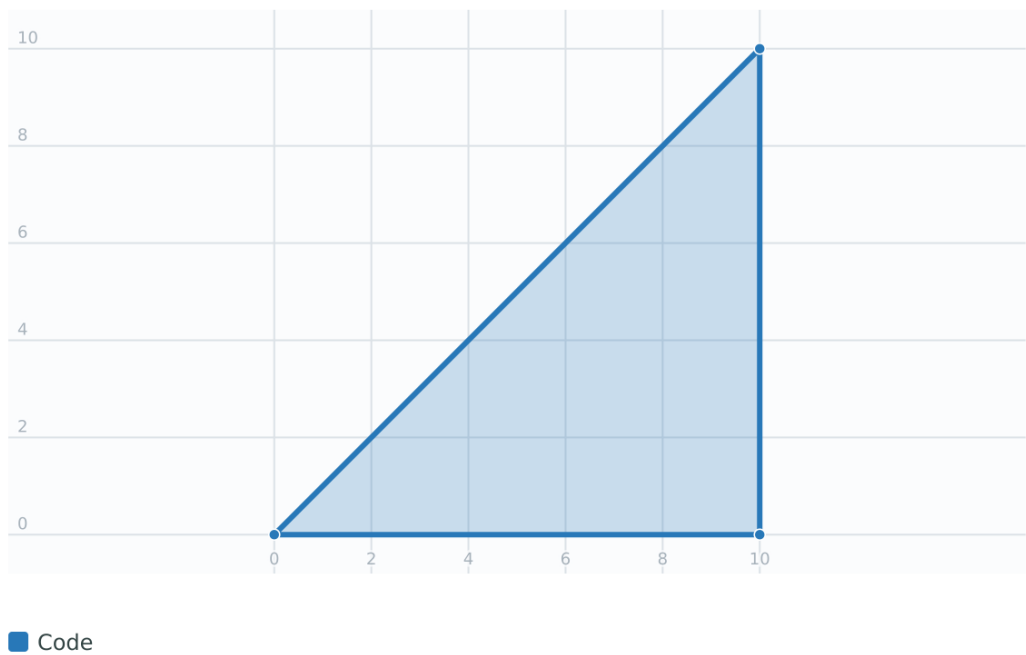
Code

```
SELECT ST_MaxDistance('POLYGON ((10 10,10 0,0 0,10 10))'::geometry);
```

Output

14.142135623730951

Figure



相关信息

[ST_Distance](#), [ST_LongestLine](#), [ST_DFullyWithin](#)

7.12.19 ST_3DMaxDistance

`ST_3DMaxDistance` — 返回两个几何图形之间的 3D 笛卡儿最大距离（基于空参考）（以投影单位表示）。

Synopsis

`float ST_3DMaxDistance(geometry g1, geometry g2);`

描述

返回两个几何图形之间的 3 最大笛卡儿距离（以投影单位（空参考单位）表示）。

✔ 函数支持 3d 并且不会丢失 z-index。

✔ 函数支持多面体曲面。

可用性: 2.0.0

更改: 2.2.0 - 在 2D 和 3D 的情况下，对于缺失的 Z，Z 不再被假定 0。

示例

Geometry example - units in meters (SRID: 2163 US National Atlas Equal area) (3D point and line compared 2D point and line) Note that currently no vertical datum support so Z is not transformed and assumed to be same units as final.

Code

```
SELECT ST_3DMaxDistance(
    ST_Transform('SRID=4326;POINT(-72.1235 42.3521 10000)::geometry, 2163),
    ST_Transform('SRID=4326;LINESTRING(-72.1260 42.45 15,-72.123 42.1546 20)::' geometry, 2163)
) As dist_3d,
ST_MaxDistance(
    ST_Transform('SRID=4326;POINT(-72.1235 42.3521 10000)::geometry, 2163),
    ST_Transform('SRID=4326;LINESTRING(-72.1260 42.45 15,-72.123 42.1546 20)::' geometry, 2163)
) As dist_2d;
```

Output

dist_3d	dist_2d
24383.7467488441	22247.8472107251

相关信息

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

7.12.20 ST_MinimumClearance

`ST_MinimumClearance` — 返回几何体的最小间隙，是几何体健壮性的度量。

Synopsis

```
float ST_MinimumClearance(geometry g);
```

描述

几何图形有可能满足 `ST_IsValid` (多边形) 或 `ST_IsSimple` (线) 的有效性标准，但如果其点之一是无效的，该几何图形将变得无效。移动了一小段距离。产生这种情况的原因可能是输入文本格式 (例如 WKT、KML、GML、GeoJSON) 或不使用双精度浮点坐标的二进制格式 (例如 MapInfo TAB) 期间的精度损失。

最小间隙是几何形状坐标精度化的健壮性的定量度量。它是在不构建无效几何体的情况下可以移动几何体点的最大距离。最小间隙越大，表明健壮性越好。

如果几何形状的最小间隙 e ，则：

- 几何中没有不同的点比距离 e 更近。
- 没有点比 e 更接近它不是端点的线段。

如果几何体不存在最小间隙 (例如，两个点或点相同的多点)，则返回 `Infinity`。

为了避免精度损失导致的有效性错误，`ST_ReducePrecision` 可以降低坐标精度，同时确保多边形几何保持有效。

可用性：2.3.0

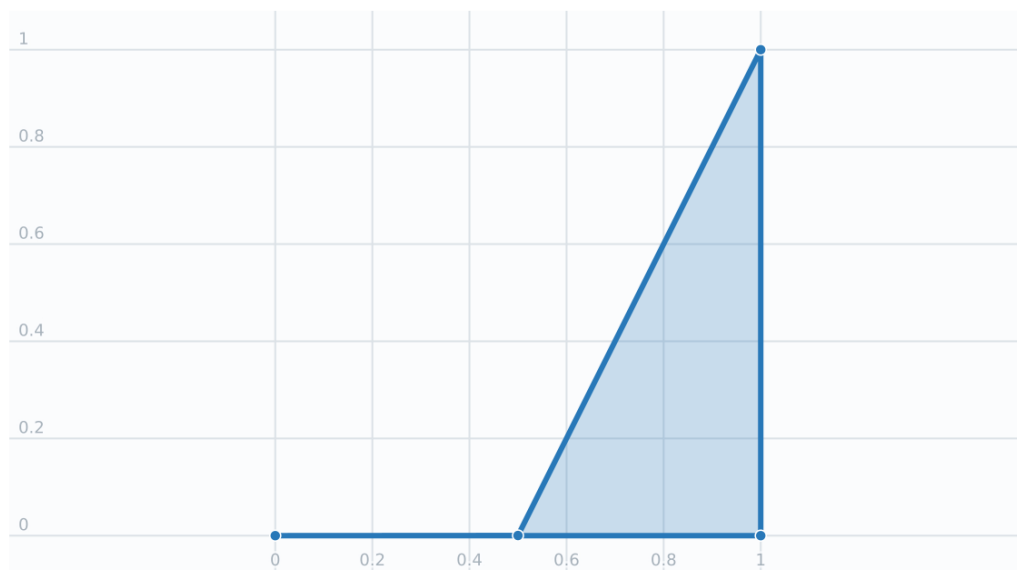
示例

Code

```
SELECT ST_MinimumClearance('POLYGON ((0 0,1 0,1 1,0.5 3.2e-4,0 0))');
```

Output

```
0.00032
```


Figure

■ Code

相关信息

[ST_MinimumClearanceLine](#), [ST_IsSimple](#), [ST_IsValid](#), [ST_ReducePrecision](#)

7.12.21 ST_MinimumClearanceLine

`ST_MinimumClearanceLine` — 返回跨越几何体最小间隙的点串。

Synopsis

Geometry **ST_MinimumClearanceLine**(geometry g);

描述

返回跨越几何体最小间隙的点串。如果几何形没有最小间隙，返回 `LINestring EMPTY`。

该函数是由 GEOS 模块实现的。

可用性：2.3.0-需要 GEOS >= 3.6.0

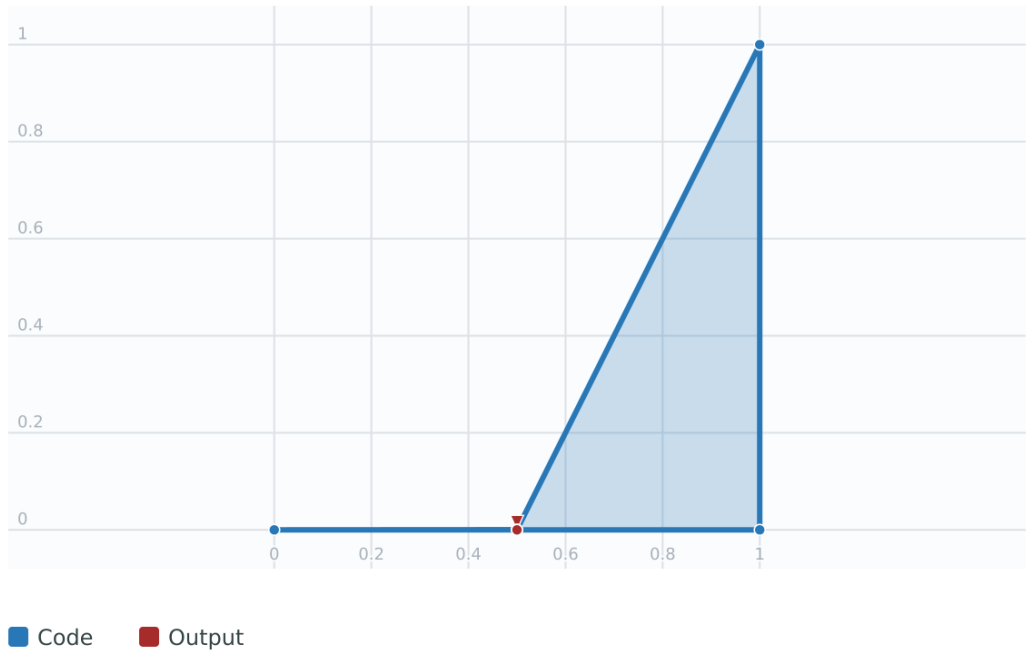
示例**Code**

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

Output

```
LINESTRING(0.5 0.00032,0.5 0)
```

Figure



相关信息

[ST_MinimumClearance](#)

7.12.22 ST_Perimeter

ST_Perimeter — 返回多边形几何或地理的周界长度。

Synopsis

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

描述

如果几何/地理是 ST_Surface、ST_MultiSurface (Polygon、MultiPolygon)，则返回其 2D 周长。对于非区域几何形状返回 0。对于线性几何形状，则使用 [ST_Length](#)。对于几何类型，周长测量的单位由几何的空参考系指定。

对于 geography 类型，则算基于反大地测量方法进行，周长单位米。所用地球由 SRID 指定；如果未提供 SRID，则默认使用 WGS84。如果 use_spheroid = false，则计算会基于球体近似而非地球体。

目前，它是 ST_Perimeter2D 的别名，但可能会更改以支持更高的维度。

 此方法符合了 [SQL 1.1](#) 的 OGC 功能规范。 s2.1.5.1



方法实现了 SQL/MM 规范。SQL-MM 3: 8.1.3, 9.5.4
可用性 2.0.0 : 引入了地理的支持

示例

Geometry.

返回多边形和多重多边形的周长（以英尺为单位）。注意，单位为英尺，因为 EPSG:2249 是塞州平面英尺

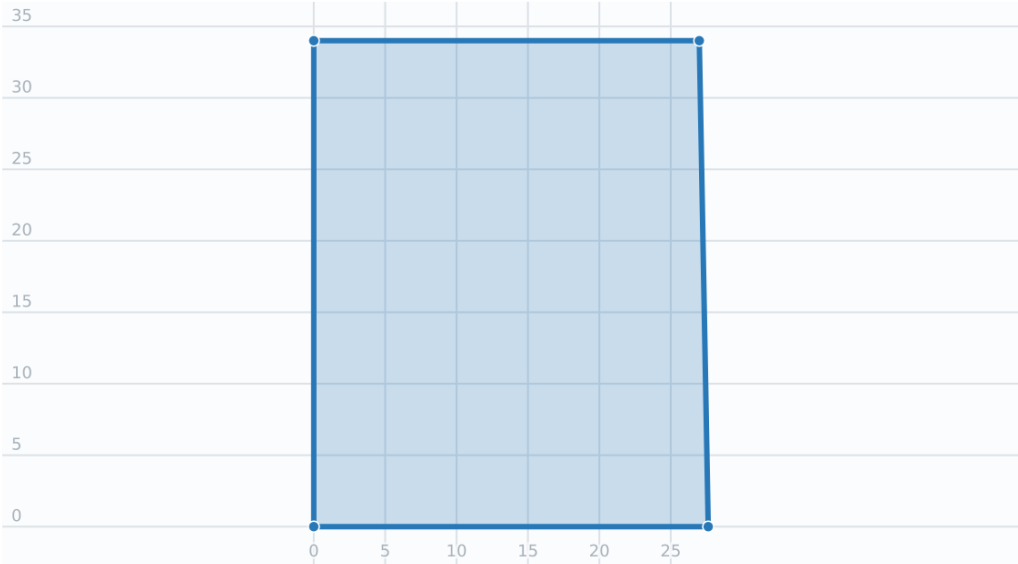
Code

```
SELECT ST_Perimeter(ST_GeomFromText(
'POLYGON((743238 2967416,743238 2967450,743265 2967450,
743265.625 2967416,743238 2967416))',
2249));
```

Output

122.63074400009504

Figure



 Code

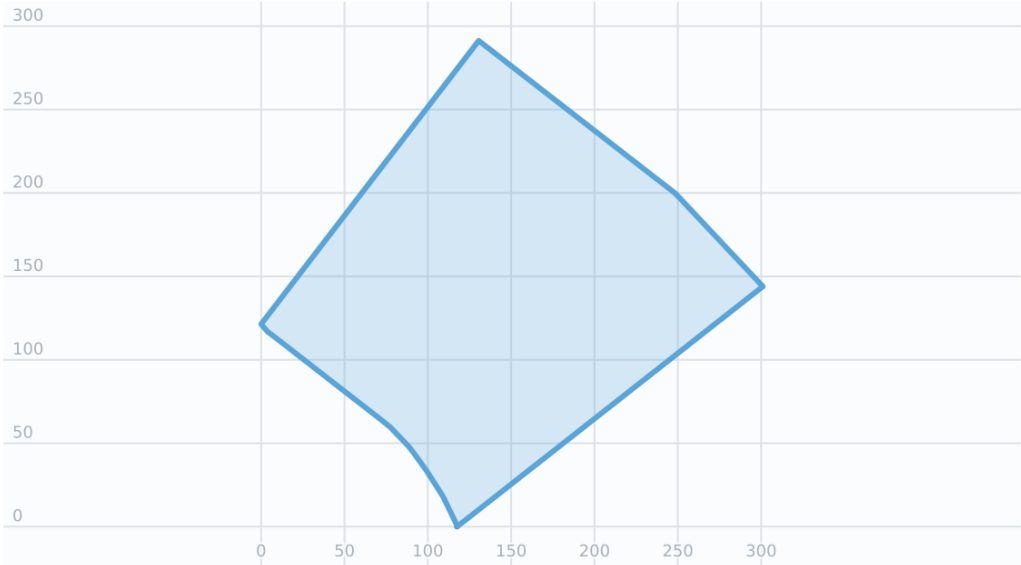
Code

```
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));
```

Output

845.227713366825

Figure



Code

返回多边形和多重多边形的周长（以米和英尺为单位）。注意，是地理（WGS 84 度）

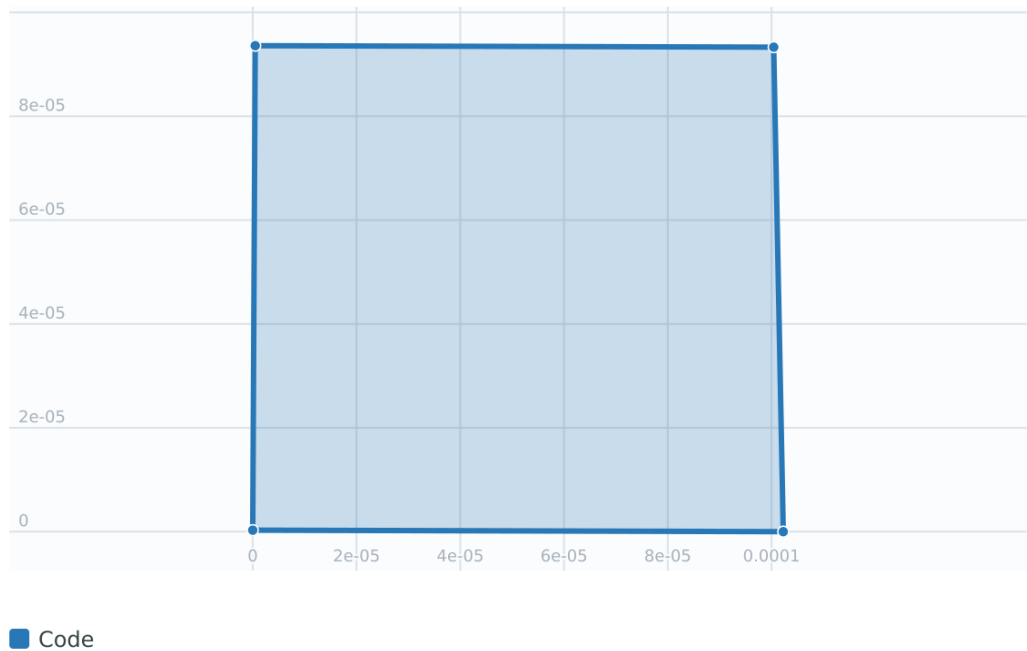
Code

```
SELECT round(ST_Perimeter(geog)::numeric, 8) As per_meters,
       round((ST_Perimeter(geog)/0.3048)::numeric, 8) 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;
```

Output

per_meters	per_ft
37.37904625	122.63466619

Figure



This example uses a MultiPolygon.

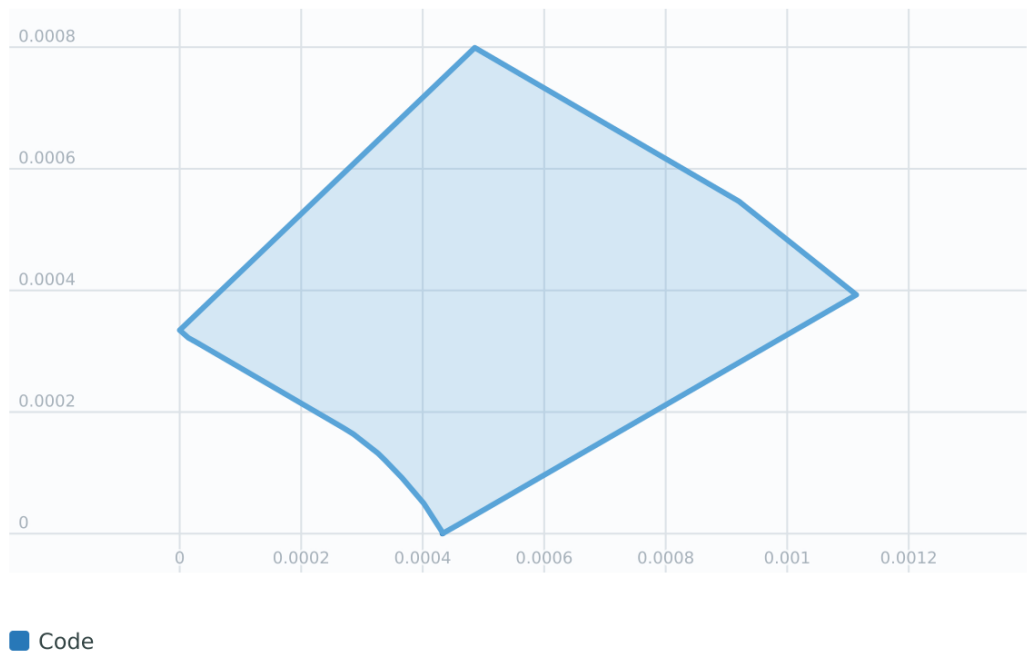
Code

```
SELECT round(ST_Perimeter(geog)::numeric, 8) As per_meters,
       round(ST_Perimeter(geog, false)::numeric, 8) As per_sphere_meters,
       round((ST_Perimeter(geog)/0.3048)::numeric, 8) 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;
```

Output

per_meters	per_sphere_meters	per_ft
257.63428368	257.41231145	845.25683623

Figure



相关信息

[ST_GeogFromText](#), [ST_GeomFromText](#), [ST_Length](#)

7.12.23 ST_Perimeter2D

ST_Perimeter2D — 返回多边形几何体的 2D 周长。ST_Perimeter 的同名。

Synopsis

float **ST_Perimeter2D**(geometry geomA);

描述

返回多边形几何体的二维周长。

Note

目前它是 ST_Perimeter 的同名。在未来版本中，ST_Perimeter 可能会返回几何图形的最高精度周长。这个问题在考虑中

相关信息

[ST_Perimeter](#)

7.12.24 ST_3DPerimeter

ST_3DPerimeter — 返回多边形几何体的 3D 周长。

Synopsis

float **ST_3DPerimeter**(geometry geomA);

描述

如果几何体是多边形或多面体，返回几何体的 3 维周长。如果几何体是二维的，返回二维周长。

- ✓ 函数支持 3d 并且不会丢失 z-index。
- ✓ 方法符合了 SQL/MM 规范。SQL-MM ISO/IEC 13249-3: 8.1, 10.5

更改：2.0.0 在之前的版本中，曾被称作 ST_Perimeter3D

示例

计算塞州机场英尺空中稍高的多边形的周长

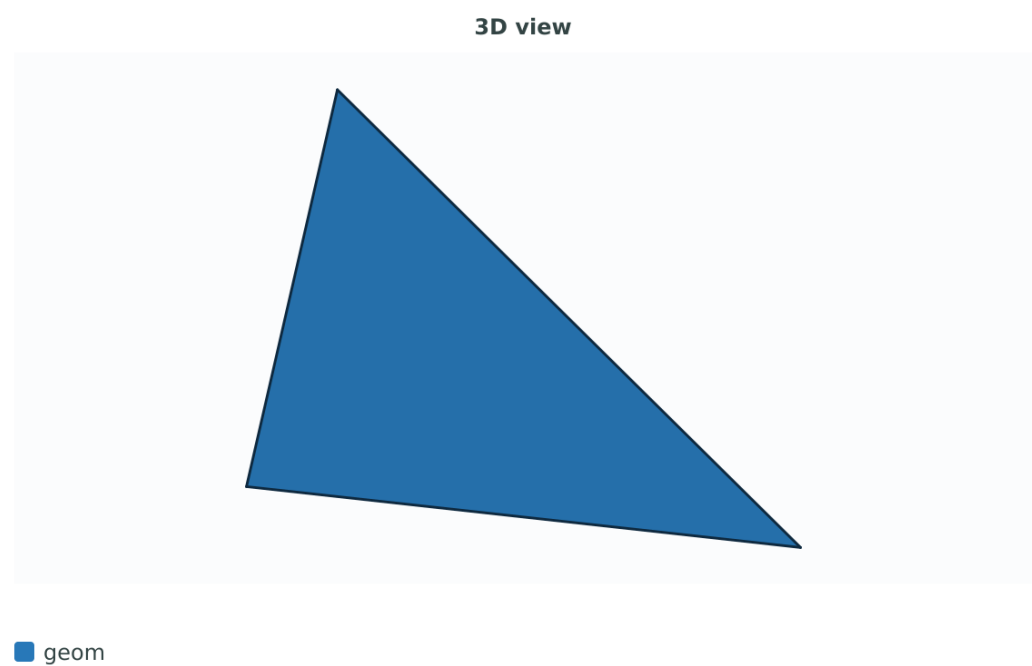
Code

```
SELECT ST_3DPerimeter(geom), ST_Perimeter2D(geom), ST_Perimeter(geom) FROM
  (SELECT 'SRID=2249;POLYGON((743238 2967416 2,743238 2967450 1,
743265.625 2967416 1,743238 2967416 2))'::geometry As geom) As foo;
```

Output

ST_3DPerimeter	st_perimeter2d	st_perimeter
105.46579359767418	105.43299727218763	105.43299727218763

Figure



相关信息

[ST_GeomFromEWKT](#), [ST_Perimeter](#), [ST_Perimeter2D](#)

7.12.25 ST_ShortestLine

ST_ShortestLine — 返回两个几何图形之间的 2D 最短线

Synopsis

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

描述

返回两个几何图形之间的二维最短线。返回的线从 geom1 开始，到 geom2 结束。如果 geom1 和 geom2 相交，结果是一条起点和点位于交点的线。线的长度与 g1 和 g2 的 [ST_Distance](#) 返回的长度相同。

增修：3.4.0 - 支持地理。

可用性：1.1.0

示例

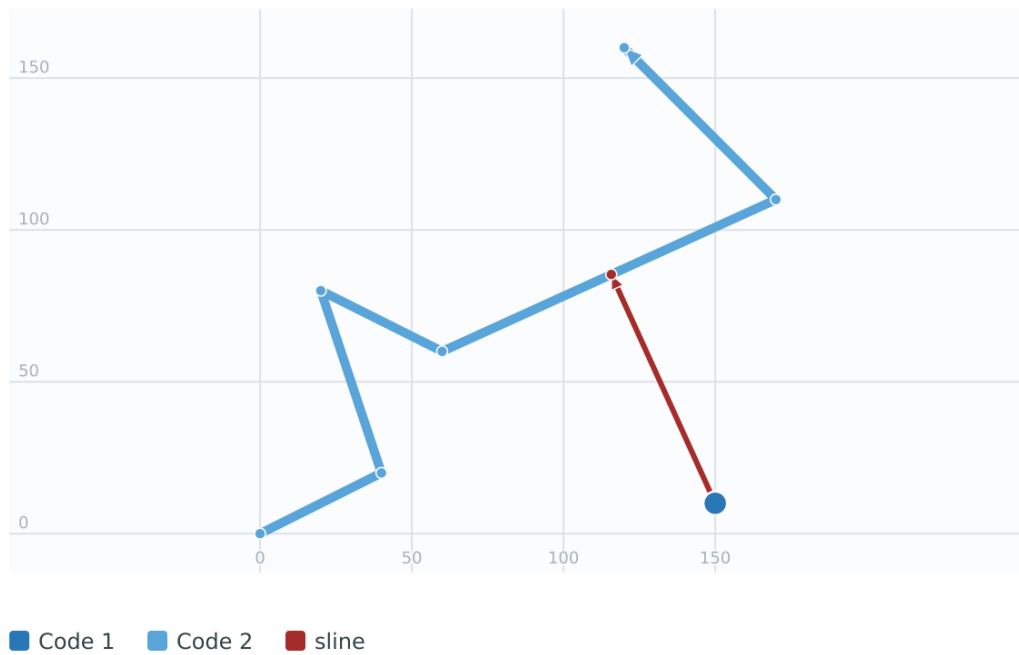
Shortest line between a Point and a LineString.

Code

```
SELECT ST_ShortestLine(  
    'POINT (160 40)',  
    'LINESTRING (10 30,50 50,30 110,70 90,180 140,130 190)') As sline;
```

Output

```
LINESTRING(160 40,125.75342465753425 115.34246575342466)
```


Figure

Shortest line between polygons.

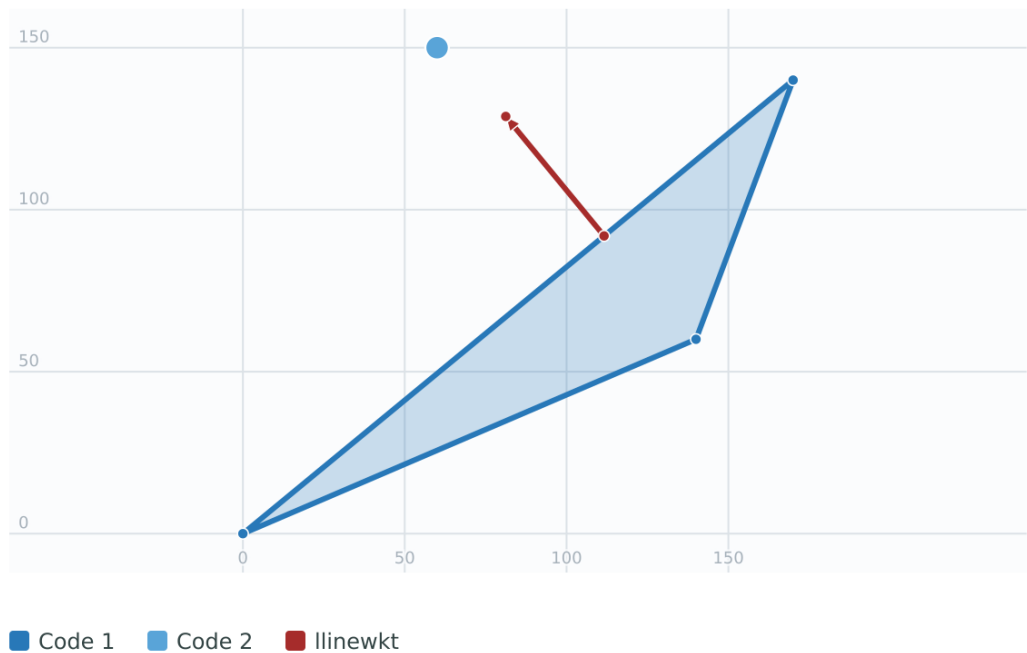
Code

```
SELECT ST_ShortestLine(  
  'POLYGON ((190 150,20 10,160 70,190 150))',  
  ST_Buffer('POINT(80 160)', 30)  
  ) AS llinewkt;
```

Output

```
LINESTRING(131.59149149528952 101.89887534906195,101.21320343559643 138.78679656440357)
```

Figure



相关信息

[ST_ClosestPoint](#), [ST_Distance](#), [ST_LongestLine](#), [ST_MaxDistance](#)

7.12.26 ST_3DShortestLine

ST_3DShortestLine — 返回两个几何图形之间的 3D 最短线

Synopsis

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

描述

返回两个几何图形之间的 3D 最短线。如果函数找到多个最短线，该函数将返回第一个最短线。如果 g1 和 g2 相交于一个点，该函数将返回一条起点和点均位于交点的线。如果 g1 和 g2 与多个点相交，该函数将返回一条起点和点位于同一点的线，但它可以是任何相交点。返回的线始终以 g1 开始并以 g2 结束。此函数返回的线的 3D 角度始终与 ST_3DDistance 返回的 g1 和 g2 相同。

可用性: 2.0.0

更改: 2.2.0 - 如果输入 2 个 2D 几何图形，返回 2D 点（而不是假设缺失 Z 为 0 的旧行）。在 2D 和 3D 情况下，对于缺失的 Z，Z 不再被假定为 0。

- ✓ 该函数支持 3d 并且不会丢失 z-index。
- ✓ 该函数支持多面体曲面。

示例

Compare the 3D shortest line to the 2D shortest line for a LineString and a Point.

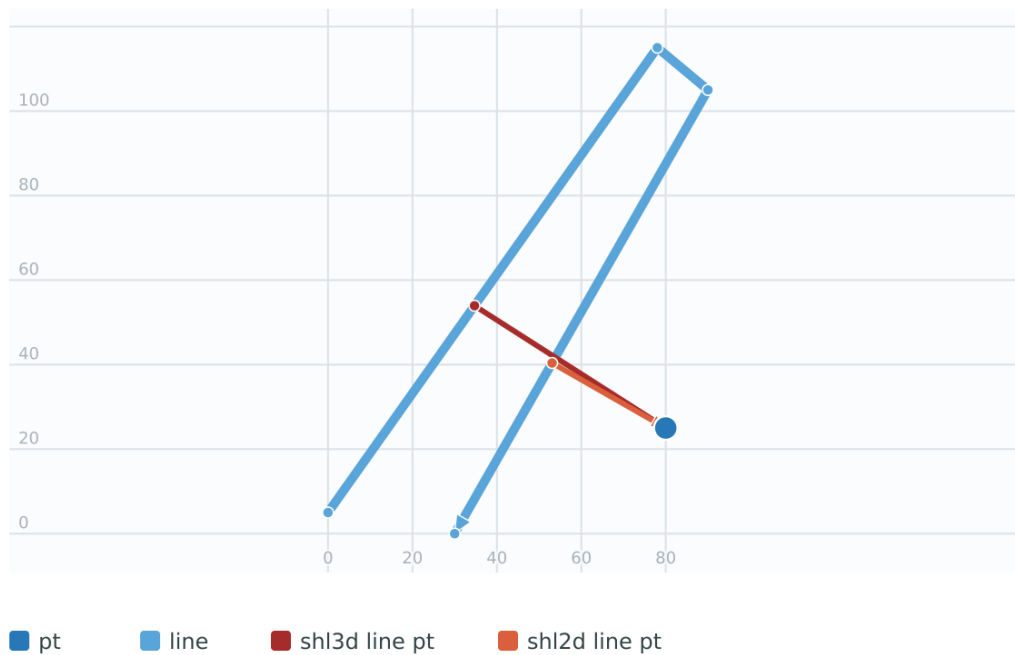
Code

```
WITH input AS (
  SELECT 'POINT(100 100 30)::geometry AS pt,
         'LINESTRING (20 80 20,98 190 1,110 180 3,50 75 1000)::geometry AS line
)
SELECT ST_3DShortestLine(line, pt) AS shl3d_line_pt,
       ST_ShortestLine(line, pt) AS shl2d_line_pt
FROM input;
```

Output

```
-[ RECORD 1 ]-----
shl3d_line_pt | LINESTRING(54.69937988676193 128.93502291722837 11.547586950660556,100 100 30)
shl2d_line_pt | LINESTRING(73.07692307692307 115.38461538461539,100 100)
```

Figure



Compare the 3D shortest line to the 2D shortest line for a LineString and a MultiPoint.

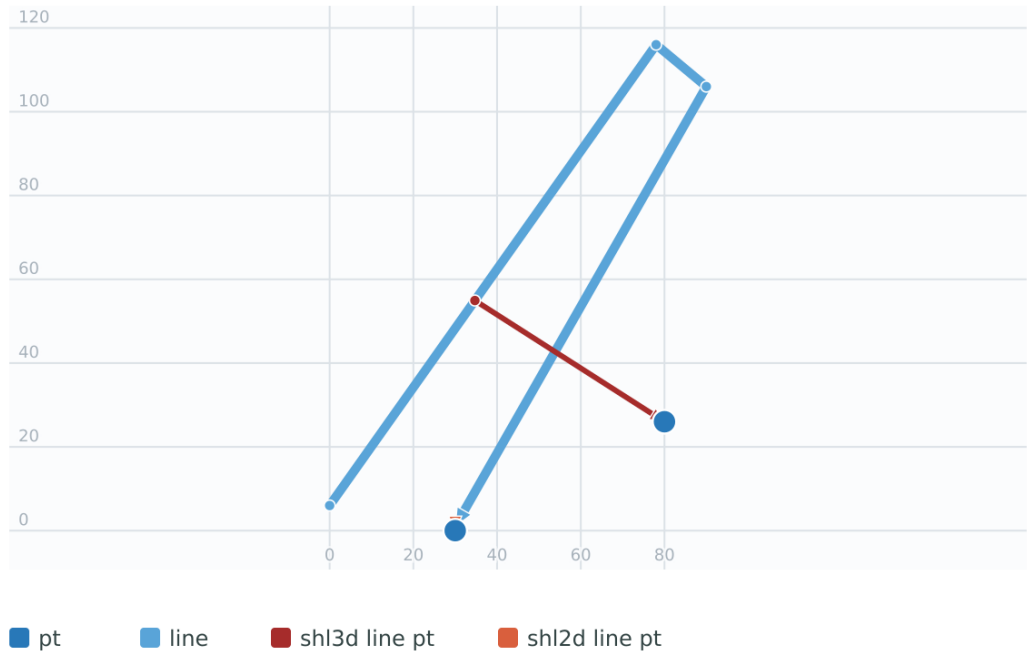
Code

```
WITH input AS (
  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
)
SELECT ST_3DShortestLine(line, pt) AS shl3d_line_pt,
       ST_ShortestLine(line, pt) AS shl2d_line_pt
FROM input;
```

Output

-[RECORD 1]-----	
shl3d_line_pt	LINESTRING(54.69937988676193 128.93502291722837 11.547586950660556,100 100 30)
shl2d_line_pt	LINESTRING(50 75,50 74)

Figure



Compare the 3D shortest line to the 2D shortest line for a Polygon and a MultiLineString.

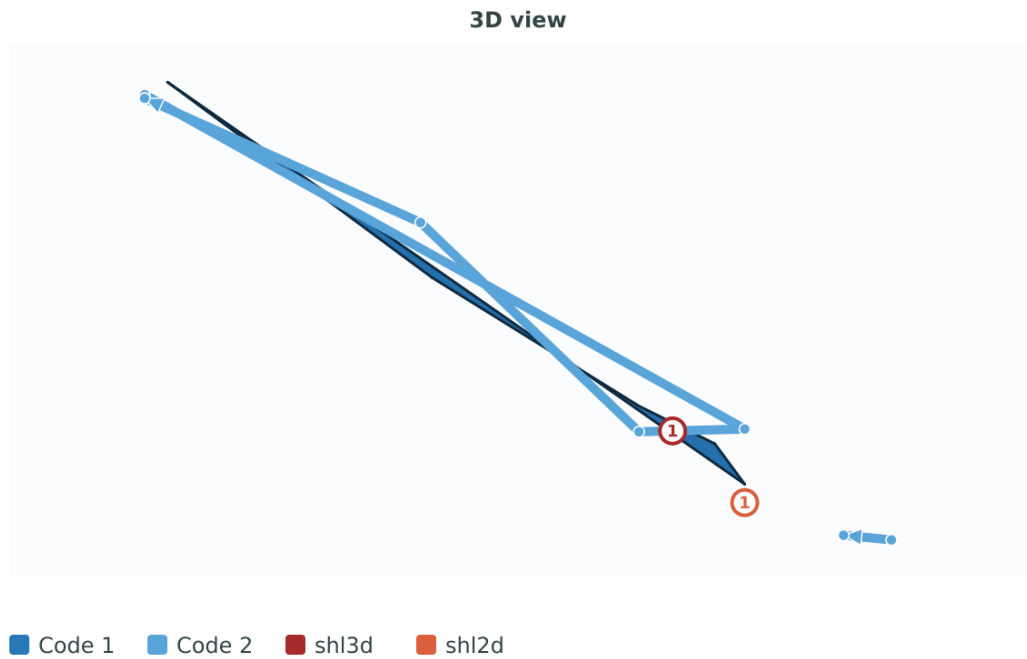
Code

```
WITH input AS (  
  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  
)  
SELECT ST_3DShortestLine(poly, mline) AS shl3d,  
       ST_ShortestLine(poly, mline) AS shl2d  
FROM input;
```

Output

-[RECORD 1]-----	
shl3d	LINESTRING(40.45454545454545 53.63636363636364 5,40.45454545454545 53.63636363636364 5)
shl2d	LINESTRING(20 40,20 40)

Figure



相关信息

[ST_3DClosestPoint](#), [ST_3DDistance](#), [ST_LongestLine](#), [ST_ShortestLine](#), [ST_3DMaxDistance](#)

7.13 加函数

7.13.1 ST_ClipByBox2D

ST_ClipByBox2D — 计算几何图形落在矩形内的部分。

Synopsis

geometry **ST_ClipByBox2D**(geometry geom, box2d box);

描述

以快速且宽松但可能无效的方式通过 2D 框剪切几何体。拓扑上无效的输入几何图形不会导致抛出异常。不保证输出几何图形有效（特别是，可能会引入多图形的自相交）。

这个函数是由 GEOS 模块执行的。

可用性：2.2.0

示例

This example relies on the implicit cast from geometry to box2d for the second parameter.

Code

```
SELECT ST_ClipByBox2D(geom, ST_MakeEnvelope(0, 0, 10, 10)) FROM mytab;
```

相关信息

[ST_Intersection](#), [ST_MakeBox2D](#), [ST_MakeEnvelope](#)

7.13.2 ST_Difference

`ST_Difference` — 计算表示几何 A 中不与几何 B 相交的部分的几何。

Synopsis

geometry **ST_Difference**(geometry geomA, geometry geomB, float8 gridSize = -1);

描述

返回一个几何图形，表示几何图形 A 中不与几何图形 B 相交的部分。相当于 `A - ST_Intersection(A,B)`。如果 A 完全包含在 B 中，返回适当类型的空原子几何。

**Note**

是唯一一个与输入顺序有关的加函数。`ST_Difference(A, B)` 始终返回 A 的一部分。

If the optional `gridSize` parameter is given (GEOS-3.9.0 or higher required), all result vertices are guaranteed to fall on a snap-rounded grid of the specified size. Note that operations performed on a grid may contain small artifacts produced during grid alignment, see [ST_ReducePrecision](#).

它是通过 GEOS 模块实现的

增加：3.1.0 接受 `gridSize` 参数。

需要 GEOS >= 3.9.0 才能使用 `gridSize` 参数。

- ✓ 此方法符合了 [SQL 1.1 的 OGC 功能规范](#)。s2.1.1.3
- ✓ 方法符合了 SQL/MM 规范。SQL-MM 3: 5.1.20
- ✓ 函数支持 3d 并且不会丢失 z-index。但是，如果使用 XY 计算。如果 Z 被复制、平均或插值。

示例

2D 字符串的差异。

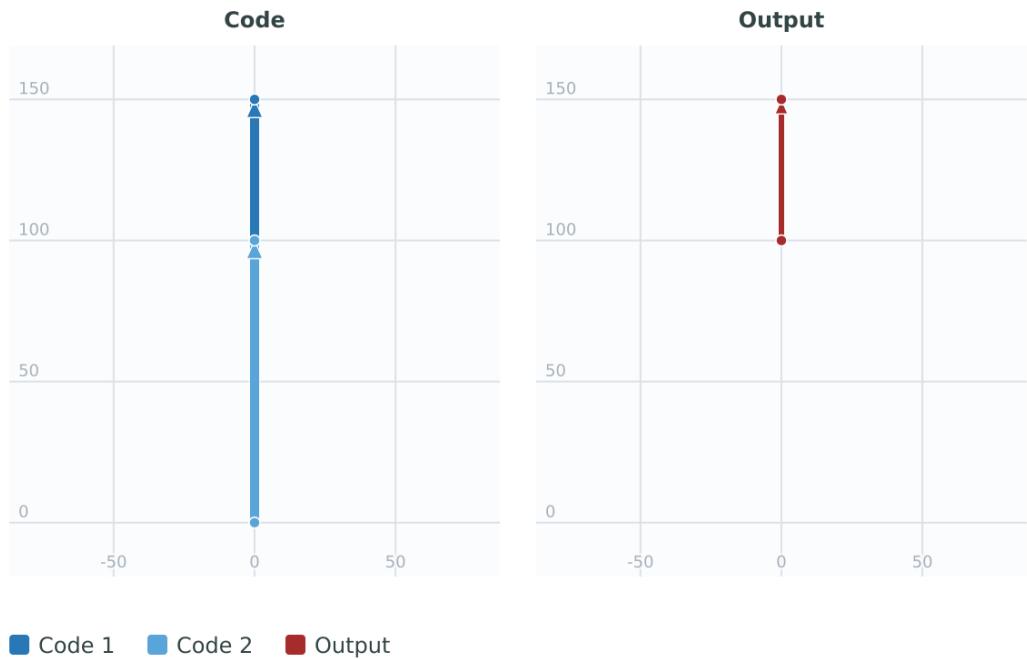
Code

```
SELECT ST_Difference('LINESTRING(50 100,50 200)::geometry',
    'LINESTRING(50 50,50 150)::geometry'
);
```

Output

```
LINESTRING(50 150,50 200)
```

Figure



3D 点的差异。

Code

```
SELECT 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
);
```

Output

```
MULTIPOINT(-118.6 38.329 6,-118.58 38.38 5)
```

Figure



相关信息

[ST_SymDifference](#), [ST_Intersection](#), [ST_Union](#), [ST_ReducePrecision](#)

7.13.3 ST_Intersection

`ST_Intersection` — 计算表示几何 A 和 B 的共享部分的几何。

Synopsis

```
geometry ST_Intersection( geometry geomA , geometry geomB , float8 gridSize = -1 );
geography ST_Intersection( geography geogA , geography geogB );
```

描述

返回表示两个几何图形的点集交集的几何图形。换句话说，几何图形 A 和几何图形 B 在两个几何图形之间共享的部分。

如果几何图形没有共同点（即不相交），则返回适当类型的空原子几何图形。

If the optional `gridSize` parameter is given (GEOS-3.9.0 or higher required), all result vertices are guaranteed to fall on a snap-rounded grid of the specified size. Note that operations performed on a grid may contain small artifacts produced during grid alignment, see [ST_ReducePrecision](#).

`ST_Intersection` 与 [ST_Intersects](#) 结合使用于剪切几何图形非常有用，例如在边界框、缓冲区或区域中，您只需要感兴趣的地区内的几何图形部分。

Note

Spatial predicates such as **ST_Intersects** and overlay functions such as **ST_Intersection** are computed by different algorithms. For very close or nearly coincident coordinates, floating-point precision and geometric robustness may make a predicate return true while the computed intersection is empty, or place an intersection point slightly away from the input line. For repeatable overlay results, snap inputs to a suitable grid first, for example with **ST_ReducePrecision**, or use the `gridSize` parameter when all input vertices already lie on the grid.

Note

对于地理数据，它是几何数据的一个量封装。它首先确定适合 2 个地理象界框的最佳 SRID（如果地理象位于 UTM 的半个区域内，但不同的 UTM 将其中之一）（向于 UTM 或伯特方位角等 (LAEA) 北/南极点，并在最坏的情况下回到墨卡托），然后在最适合的平面空参考中相交并重新回 WGS84 地理。

**Warning**

函数将除 M 坐标（如果存在）。

**Warning**

If working with 3D geometries, you may want to use SFCGAL based **CG_3DIntersection** which does a proper 3D intersection for 3D geometries. Although this function works with Z-coordinate, it does an averaging of Z-Coordinate.

它是通 GEOS 模的

增：3.1.0 接受 `gridSize` 参数

需要 GEOS >= 3.9.0 才能使用 `gridSize` 参数

更改：3.0.0 不依赖于 SFCGAL。

可用性：1.5 引入了地理数据型的支持。



此方法符合 SQL 1.1 的 OGC 功能范。 s2.1.1.3



方法符合 SQL/MM 范。 SQL-MM 3: 5.1.18



函数支持 3d 并且不会失 z-index。但是，果使用 XY 算。果 Z 被复制、平均或插。

示例

Code

```
SELECT ST_Intersection('POINT(0 0)::geometry', 'LINESTRING ( 2 0,0 2 )::geometry);
```



```

    ) 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));

```

Note this is not a true 3D intersection. It uses 2.5D geometries (geometries with Z ordinates, but where calculations are performed in 2D). The same input is compared with **CG_3DIntersection**, which performs a true 3D intersection.

Code

```

WITH data AS (
  SELECT 'LINESTRING Z (2 2 6,1.5 1.5 7,1 1 8,0.5 0.5 8,0 0 10)>:::geometry AS ↵
    input_linestring,
    'POLYGON Z ((0 0 8,0 1 8,1 1 8,1 0 8,0 0 8))>:::geometry AS input_polygon
)
SELECT input_linestring AS input_linestring,
       input_polygon AS input_polygon,
       ST_Intersection(input_linestring, input_polygon) AS intersection_2d,
       CG_3DIntersection(input_linestring, input_polygon) AS intersection_3d
FROM data;

```

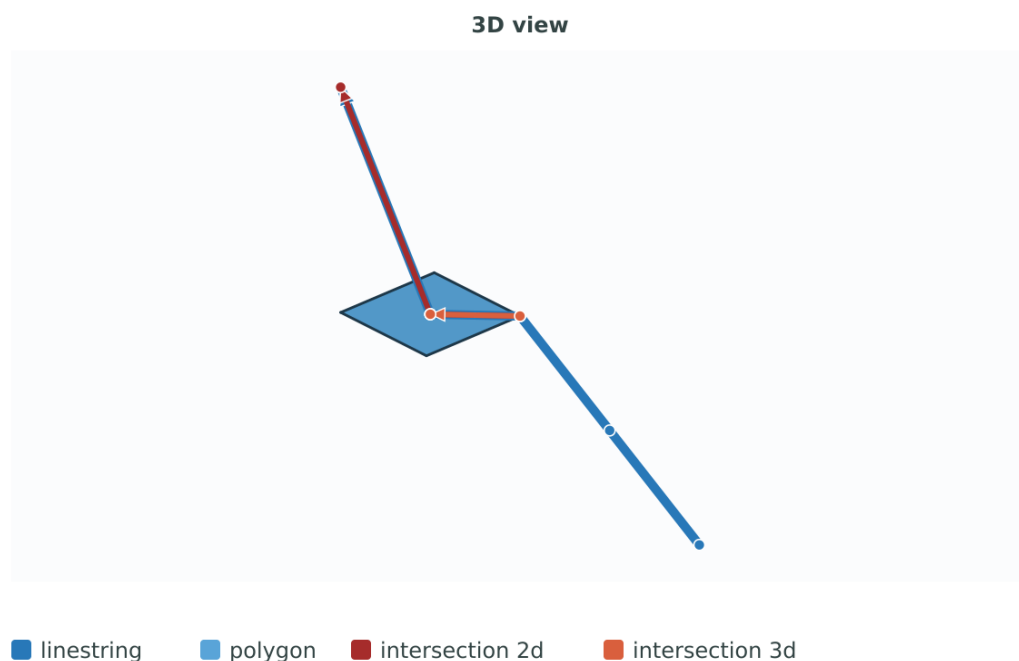
Output

```

LINESTRING Z (2 2 6,1.5 1.5 7,1 1 8,0.5 0.5 8,0 0 10) | POLYGON Z ((0 0 8,0 1 8,1 1 8,1 0 8,0 0 8)) | LINESTRING Z (1 ↵
1 8,0.5 0.5 8,0 0 10) | LINESTRING Z (1 1 8,0.5 0.5 8)

```

Figure



相关信息

CG_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 — 聚合函数，以内存高效但速度较慢的方式聚合几何图形

Synopsis

geometry **ST_MemUnion**(geometry set geomfield);

描述

聚合函数，用于聚合输入几何图形，将它合并以生成没有重量的结果几何图形。输出可以是单个几何体、多几何体或几何体集合。



Note

生成与 **ST_Union** 相同的结果，但使用更少的内存和更多的推理。此聚合函数对几何图形的顺序加法进行操作，与 ST_Union 聚合不同，ST_Union 聚合首先累积一个数，然后使用快速算法合并内容。



该函数支持 3d 并且不会丢失 z-index。但是，结果使用 XY 计算。结果 Z 被复制、平均或插值。

示例

Code

```
SELECT id,
       ST_MemUnion(geom) as singlegeom
FROM sometable f
GROUP BY id;
```

相关信息

ST_Union

7.13.5 ST_Node

ST_Node — 将点是集合。

Synopsis

geometry **ST_Node**(geometry geom);

描述

返回一个 (Multi)LineString，表示线串集合的完全节点版本。节点保留所有输入点，并引入尽可能少的新点。生成的线条被溶解（重复的线条被删除）。

它是构建适合用作 **ST_Polygonize** 输入的全点线的好方法。

ST_UnaryUnion 也可用于点和溶解线条。它提供了一个指定 `gridSize` 的选项，它可以提供更精确、更稳健的输出。请参见 **ST_Union** 了解聚合版本。

 该函数支持 3d 并且不会丢失 z-index。

该函数是由 GEOS 模块执行的。

可用性: 2.0.0

更改: 2.4.0 该函数在内部使用 GEOSNode 而不是 GEOSUnaryUnion。与 PostGIS < 2.4 相比，它可能会导致生成的线串具有不同的顺序和方向。

示例

Noding a 3D LineString which self-intersects.

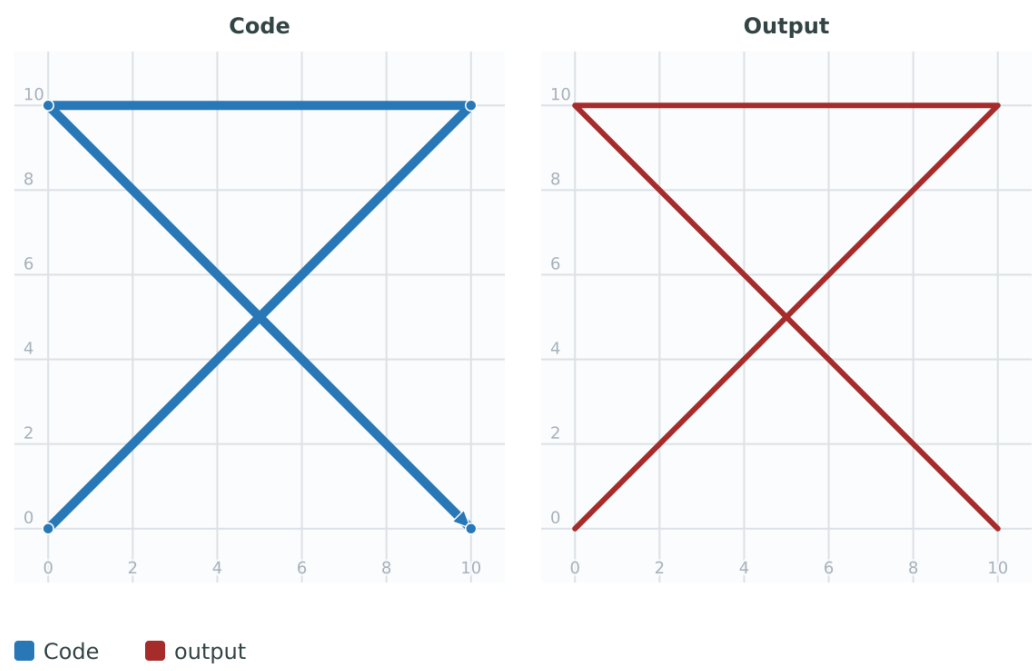
Code

```
SELECT 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))
```

Figure



将节点添加到共享一条线的每个线串。生成的线被溶解。

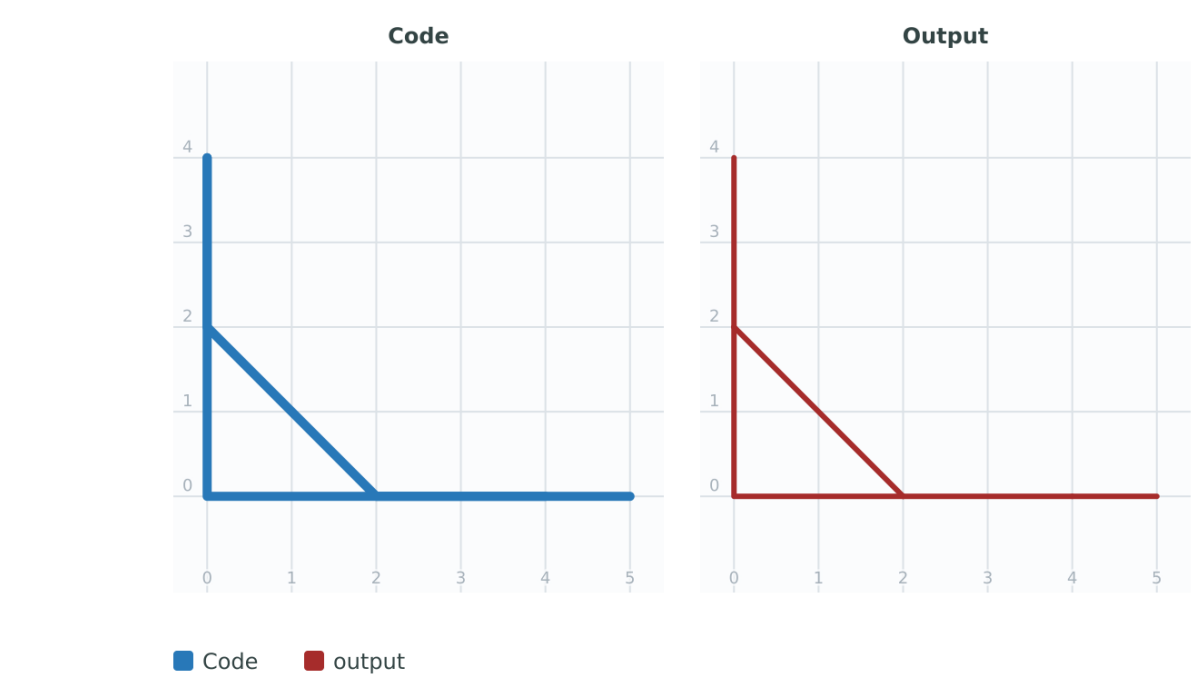
Code

```
SELECT 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))
```

Figure



相关信息

[ST_UnaryUnion](#), [ST_Union](#)

7.13.6 ST_Split

ST_Split — 返回通~~过~~将一个几何体分割~~成~~一个几何体而~~建~~的几何体集合。

Synopsis

geometry **ST_Split**(geometry input, geometry blade);

描述

~~该~~函数支持按（多）点、（多）~~线~~串或（多）多~~面~~形~~面~~界分割~~线~~串，或按~~线~~串分割（多）多~~面~~形。当（多）多~~面~~形用作~~面~~卡~~片~~，其~~面~~性分量（~~面~~界）用于分割~~线~~入。~~该~~果几何~~面~~形始~~终~~是一个集合。

~~该~~函数在某种意~~义~~上与 [ST_Union](#) 相反。将 ST_Union ~~用~~于返回的集合理~~解~~上~~该~~会~~生~~原始几何~~面~~形（尽管由于数字舍入，情况可能并非完全如此）。



Note
如果由于数值精度输入和刀片不相交，输入可能不会按期分割。为了避免这种情况，可能需要首先使用具有小公差的 **ST_Snap** 将输入捕捉到刀片。

可用性：2.0.0 需要 GEOS
增强：2.2.0 引入了通多、多点或（多）多边形边界分割的支持。
增强：2.5.0 引入了通多分割多边形的支持。

示例

Split a circular polygon by a diagonal line.

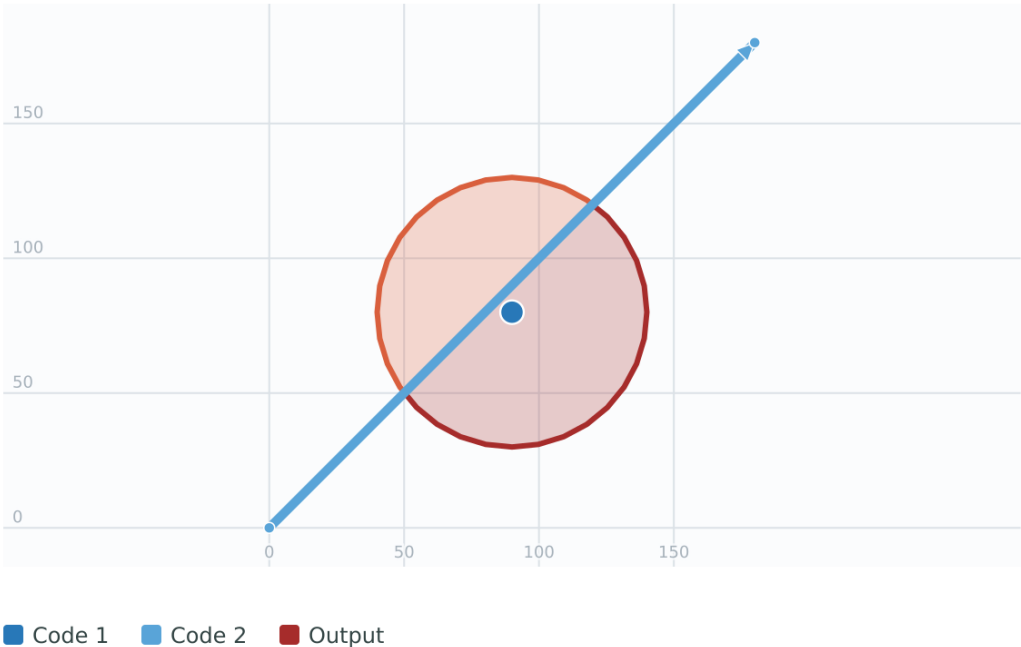
Code

```
SELECT
  ST_Split(
    ST_Buffer('POINT(100 90)::geometry, 50),
    'LINESTRING(10 10,190 190)::geometry'
  );
```

Output

```
GEOMETRYCOLLECTION(POLYGON((150 90,149 80.2,146.2 70.9,141.6 62.2,135.4 54.6,127.8 48.4,119.1 43.8,109.8 41,100 40,90.2 41,80.9 43.8,72.2 48.4,64.6 54.6,60.1 60.1,129.9 129.9,135.4 125.4,141.6 117.8,146.2 109.1,149 99.8,150 90)), POLYGON((60.1 60.1,58.4 62.2,53.8 70.9,51 80.2,50 90,51 99.8,53.8 109.1,58.4 117.8,64.6 125.4,72.2 131.6,80.9 136.2,90.2 139,100 140,109.8 139,119.1 136.2,127.8 131.6,129.9 129.9,60.1 60.1)))
```

Figure



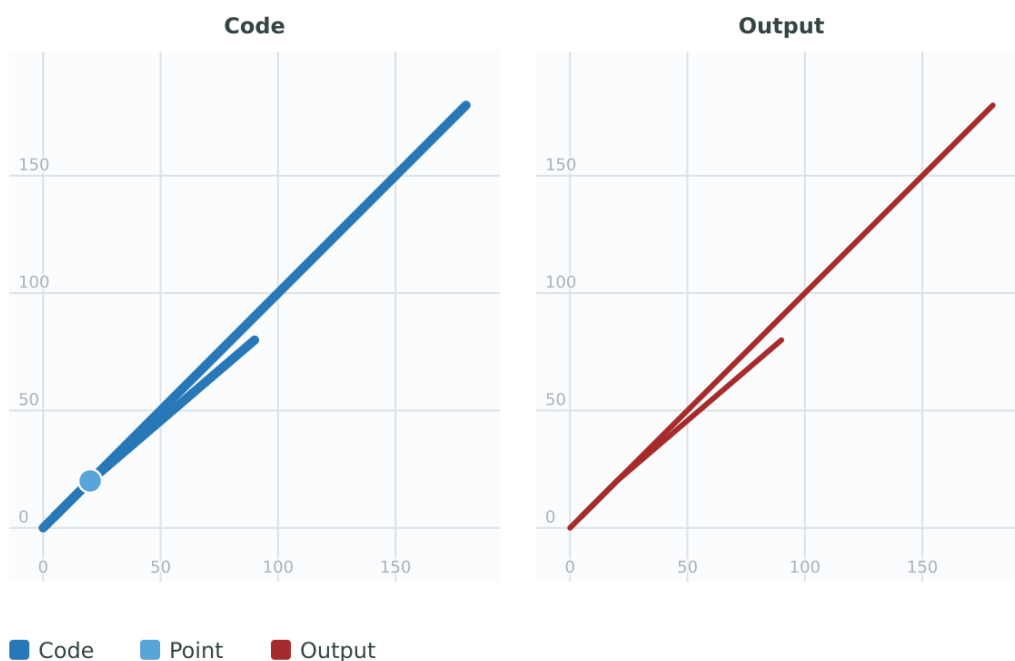
将一个多线串（MultiLineString）通一个点分割，其中点正好位于个线串元素上。

Code

```
SELECT ST_Split('MULTILINESTRING((10 10,190 190),(15 15,30 30,100 90))',
  ST_Point(30, 30));
```

Output

```
GEOMETRYCOLLECTION(
  LINESTRING(10 10,30 30),
  LINESTRING(30 30,190 190),
  LINESTRING(15 15,30 30),
  LINESTRING(30 30,100 90)
)
```

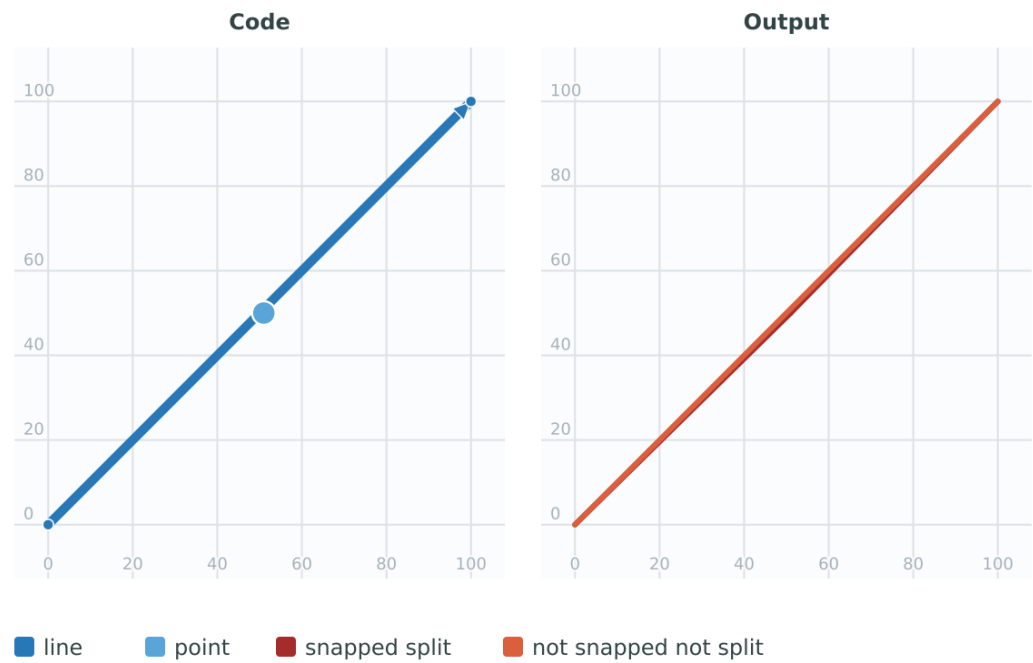
Figure

将 LineString 用点分割，其中点并不正好位于线上。示使用 **ST_Snap** 将捕捉到点以允将其分割。

Code

```
WITH data AS (SELECT
  'LINESTRING(0 0,100 100)::geometry AS line,
  'POINT(51 50):: geometry AS point
)
SELECT ST_Split(ST_Snap(line, point, 1), point) AS snapped_split,
  ST_Split(line, point) AS not_snapped_not_split
FROM data;
```


Figure



相关信息

[ST_Snap](#), [ST_Union](#)

7.13.7 ST_Subdivide

ST_Subdivide — 计算几何体的直线细分。

Synopsis

setof geometry **ST_Subdivide**(geometry geom, integer max_vertices=256, float8 gridSize = -1);

描述

返回一个几何图形，这些几何图形是使用直线将 `geom` 分成多个部分的结果，每个部分包含不超过 `max_vertices`。
`max_vertices` 必须不小于 5，因为表示一个封闭盒子至少需要 5 个点。

If the optional `gridSize` parameter is given (GEOS-3.9.0 or higher required), all result vertices are guaranteed to fall on a snap-rounded grid of the specified size. Note that operations performed on a grid may contain small artifacts produced during grid alignment, see [ST_ReducePrecision](#).

对于索引分数数据集，多边形内的点和其他空操作通常更快。由于零件的边界框通常覆盖比原始几何图形边界框更小的区域，因此索引生成的“命中”情况更少。“命中”情况更快，因为索引重新索引行的空操作处理的点更少。



Note

将面分后的 geometry 转成 geography 面，得到的 geography 可能和原始的不一致。面分操作是在平面 (geometry) 空间里加点的，如果某些新点落在面上，会改变地理空间的表示——因为在 geography 中，面被解成面地。为了尽量减少这种偏差，建议先对 geography 使用 **ST_Segmentize** 做面地加密 (添加地理空间下的点)，再转成 geometry 后面分。



Note

面是一个返回集合的 **集合返回函数** (SRF) 包含多个几何面的面。它可以在 SELECT 列表或 FROM 子句中使用，以生成一个面集，其中每个面几何面都有一个面。

面函数是由 GEOS 模型行的。

可用性 : 2.2.0

增面 : 2.5.0 重用多面形分割上的面有点，面点数从 8 面少到 5。

增面 : 3.1.0 接受 gridSize 参数。

需要 GEOS >= 3.9.0 才能使用 gridSize 参数

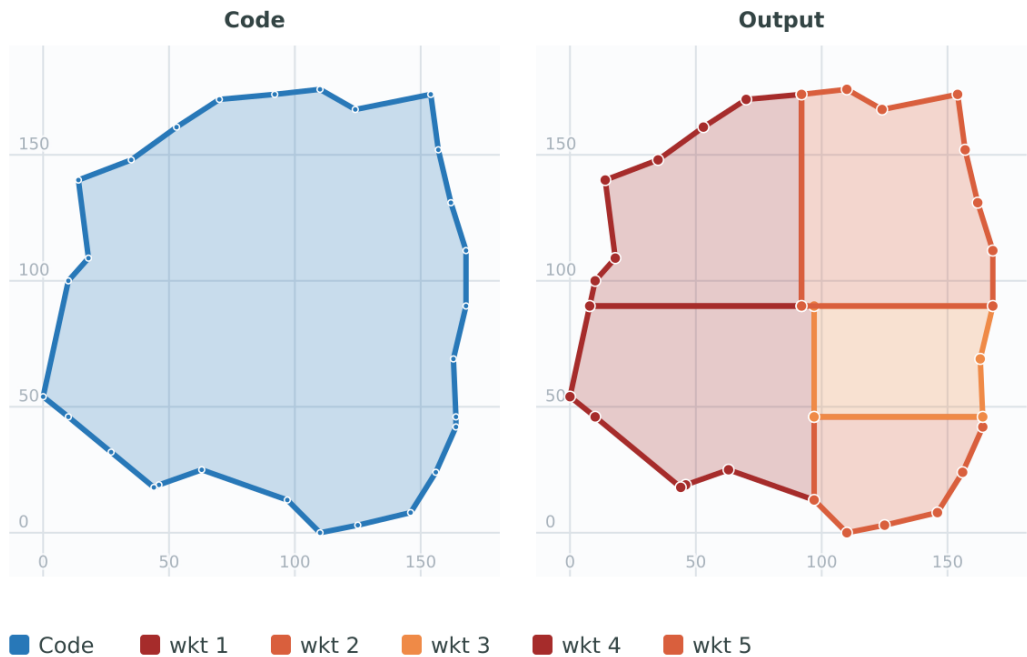
示例

示例：将多面形面分面不超过 10 个面的部分，并对每个部分分配一个唯一的 id。

Code

```
SELECT row_number() OVER() As rn,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);
```

Figure



示例：使用 `ST_Segmentize(geometry, distance)` 对 `geometry` 行加密，然后用 `ST_Subdivide` 将加密后的行拆分成每段包含 8 个点的子行。通常先加密，可在分降低 `geometry` 表示化来的影。

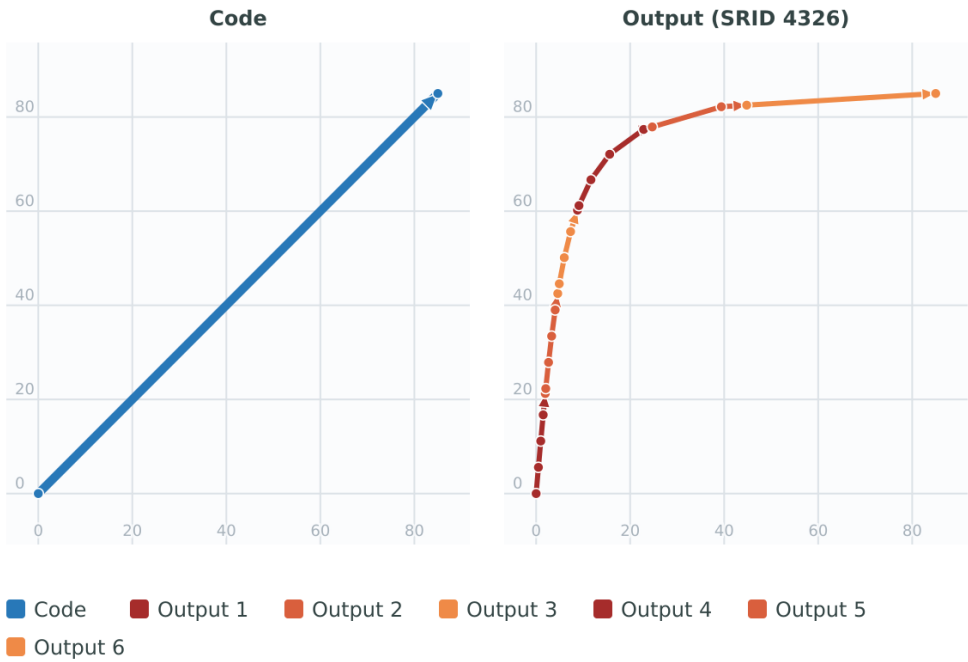
Code

```
SELECT ST_Subdivide(ST_Segmentize('LINESTRING(0 0,85 85)::geometry',
                                  1200000)::geometry, 8);
```

Output

```
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)
```

Figure



示例：在表中的适当位置分割复杂几何图形。具有原始几何的行将从源表中删除，拆分后生成的几何将成其位置的新行。

Code

```
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;
```

示例：创建具有分区几何形状的新表。保留原始几何形状的，新表可以与原始表连接。由于 `ST_Subdivide` 是一个返回集合的函数，并且行具有多个，因此此方法会自生成一个表，其中每个部分的每个结果都有一行。

Code

```
CREATE TABLE subdivided_geoms AS
  SELECT pkey, ST_Subdivide(geom) AS geom
  FROM original_geoms;
```

相关信息

[ST_ClipByBox2D](#), [ST_Segmentize](#), [ST_Split](#), [ST_NPoints](#), [ST_ReducePrecision](#)

7.13.8 ST_SymDifference

`ST_SymDifference` — 计算表示几何形状 A 和 B 不相交部分的几何形状。

Synopsis

geometry **ST_SymDifference**(geometry geomA, geometry geomB, float8 gridSize = -1);

描述

返回表示几何形状 A 和 B 不相交部分的几何形状。相当于 `ST_Union(A,B) - ST_Intersection(A,B)`。之所以称对称差，是因为 `ST_SymDifference(A,B) = ST_SymDifference(B,A)`。

If the optional `gridSize` parameter is given (GEOS-3.9.0 or higher required), all result vertices are guaranteed to fall on a snap-rounded grid of the specified size. Note that operations performed on a grid may contain small artifacts produced during grid alignment, see [ST_ReducePrecision](#).

它是通过 GEOS 模块实现的

增加：3.1.0 接受 `gridSize` 参数。

需要 GEOS >= 3.9.0 才能使用 `gridSize` 参数



此方法符合了 SQL 1.1 的 OGC 功能规范。s2.1.1.3



方法符合了 SQL/MM 规范。SQL-MM 3: 5.1.21



函数支持 3d 并且不会丢失 z-index。但是，结果使用 XY 计算。结果 Z 被复制、平均或插值。

示例

This example shows a 2D-safe symmetric difference of two LineStrings.

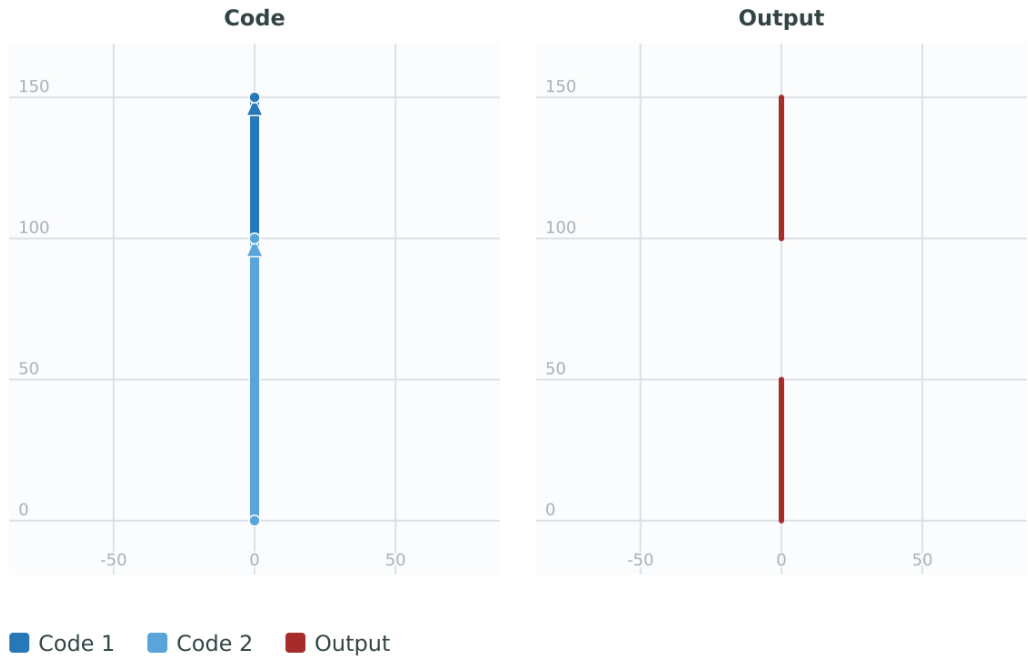
Code

```
SELECT ST_SymDifference('LINESTRING(50 100,50 200)',
  'LINESTRING(50 50,50 150)'
);
```

Output

```
MULTILINESTRING((50 150,50 200),(50 50,50 100))
```

Figure



In 3D, this example does not produce the expected result.

Code

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

Output

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

Figure



相关信息

[ST_Difference](#), [ST_Intersection](#), [ST_Union](#), [ST_ReducePrecision](#)

7.13.9 ST_UnaryUnion

`ST_UnaryUnion` — 计算多个几何体的并集。

Synopsis

```
geometry ST_UnaryUnion(geometry geom, float8 gridSize = -1);
```

描述

ST_Union 的输入形式。输入可以是多个几何图形、`MultiGeometry` 或 `GeometryCollection`。并集用于输入的各个元素。

此函数可用于修复由于重叠面而无效的多重多面体。但是，每个输入面都必须有效。无效的输入面（例如多面体）可能会致。因此，最好使用 **ST_MakeValid**。

此函数的一个用途是交叉或重叠的面串集合行点化和分解，以使它们。（**ST_Node** 也行此操作，但它不提供 `gridSize`。）

可以将 `ST_UnaryUnion` 与 **ST_Collect** 合起来，以微一次合的几何图形数量。允许在内存使用和算之之行衡，从而在 `ST_Union` 和 **ST_MemUnion** 之取得平衡。

If the optional `gridSize` parameter is given (GEOS-3.9.0 or higher required), all result vertices are guaranteed to fall on a snap-rounded grid of the specified size. Note that operations performed on a grid may contain small artifacts produced during grid alignment, see **ST_ReducePrecision**.



函数支持 3d 并且不会丢失 z-index。但是，如果使用 XY 算。如果 Z 被复制、平均或插值。

增量：3.1.0 接受 gridSize 参数。

需要 GEOS >= 3.9.0 才能使用 gridSize 参数

可用性: 2.0.0

相关信息

[ST_Union](#), [ST_MemUnion](#), [ST_MakeValid](#), [ST_Collect](#), [ST_Node](#), [ST_ReducePrecision](#)

7.13.10 ST_Union

ST_Union — 计算表示输入几何图形的点集并集的几何图形。

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);
```

描述

组合输入几何图形，合并几何图形以生成没有重复的结果几何图形。输出可以是原子几何、多几何或几何集合。有多种格式：

双输入格式：返回一个几何图形，该几何图形是输入几何图形的并集。如果任一输入 NULL，则返回 NULL。

数组格式：返回一个几何图形，该几何图形是几何图形数组的并集。

聚合函数格式：返回一个几何图形，该几何图形是几何图形集合的并集。ST_Union() 函数是 PostgreSQL 中的“聚合”函数。这意味着它对数据行进行操作，与 SUM() 和 AVG() 函数的操作方式相同，并且与大多数聚合一样，它也会忽略 NULL 几何图形。

参见 [ST_UnaryUnion](#) 了解非聚合、输入格式。

ST_Union 的数组和聚合形式使用<http://blog.cleverelephant.ca/2009/01/must-faster-unions-in-postgis-14.html> 中引入的快速连接算法

If the optional gridSize parameter is given (GEOS-3.9.0 or higher required), all result vertices are guaranteed to fall on a snap-rounded grid of the specified size. Note that operations performed on a grid may contain small artifacts produced during grid alignment, see [ST_ReducePrecision](#).



Note

[ST_Collect](#) 如果不要求结果不重复，有时可以使用 ST_Collect 代替 ST_Union。ST_Collect 通常比 ST_Union 更快，因为它不收集几何图形并执行任何清理。

这些函数是由 GEOS 模块执行的。

ST_Union 构建 MultiLineString 并且不会将 LineString 合成一个 LineString。使用 [ST_LineMerge](#) 组合串。

注意：此函数以前称 `GeomUnion()`，它是从“Union”重命名的，因 `UNION` 是 SQL 保留字。

增：3.1.0 接受 `gridSize` 参数。

需要 GEOS $\geq 3.9.0$ 才能使用 `gridSize` 参数

更改：3.0.0 不依赖于 SFCGAL。

可用性：1.4.0 - `ST_Union` 得到增。PostgreSQL 中引入了 `ST_Union(geomarray)` 以及更快的集合聚合。

✔ 此方法实现了 SQL 1.1 的 OGC 功能规范。 s2.1.1.3



Note

OGC SPEC 中未明确定义聚合版本。

✔ 方法实现了 SQL/MM 规范。SQL-MM 3 : 5.1.19 涉及多面形的 z 索引（高程）。

✔ 函数支持 3d 并且不会丢失 z-index。但是，如果使用了 XY 计算。如果 Z 被复制、平均或插值。

示例

聚合示例

Code

```
SELECT id,
       ST_Union(geom) as singlegeom
FROM sometable f
GROUP BY id;
```

非聚合示例

Code

```
SELECT ST_Union('POINT(1 2)::geometry', 'POINT(-2 3)::geometry');
```

Output

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


Figure



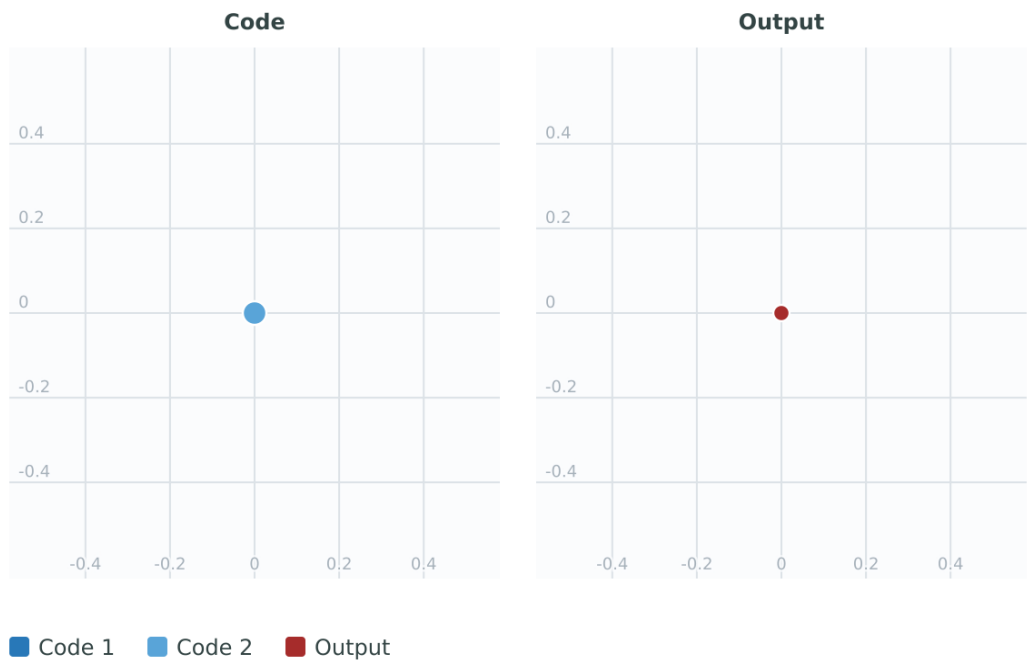
Code

```
SELECT ST_Union('POINT(1 2)::geometry, 'POINT(1 2)::geometry');
```

Output

```
POINT(1 2)
```

Figure



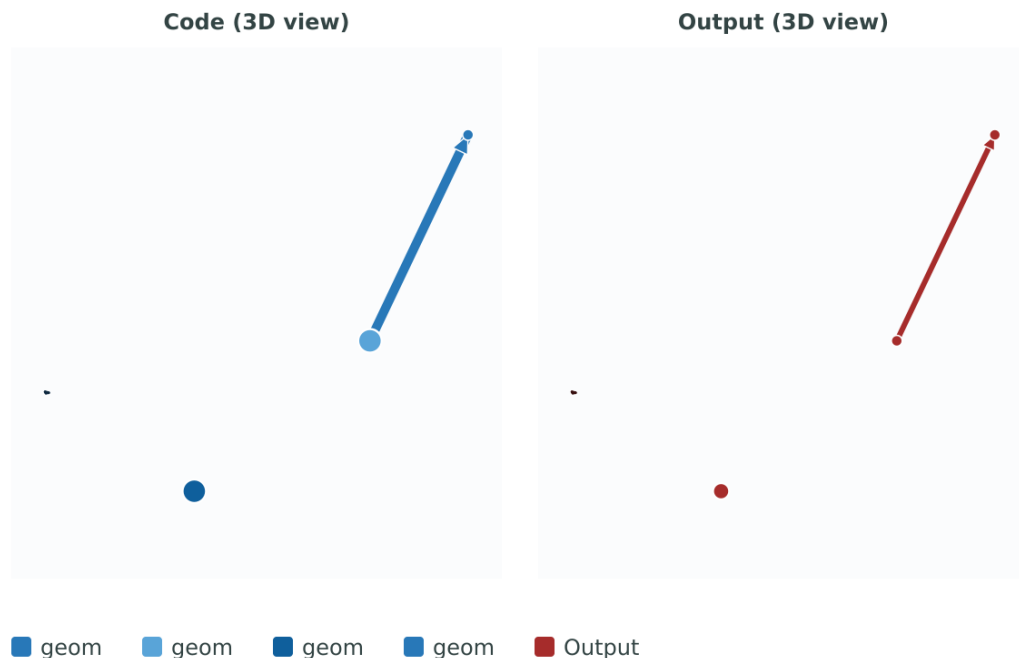
3D example - sort of supports 3D (and with mixed geometry types!)

Code

```
SELECT ST_Union(geom)
FROM (
  SELECT 'POLYGON Z((-7 4.2 5,-7.1 4.2 5,-7.1 4.3 5,-7 4.2 5))'::geometry geom
  UNION ALL
  SELECT 'POINT Z(5 5 5)'::geometry geom
  UNION ALL
  SELECT 'POINT Z(-2 3 1)'::geometry geom
  UNION ALL
  SELECT 'LINESTRING Z(5 5 5,10 10 10)'::geometry geom
) AS foo;
```

Output

```
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)))
```

Figure

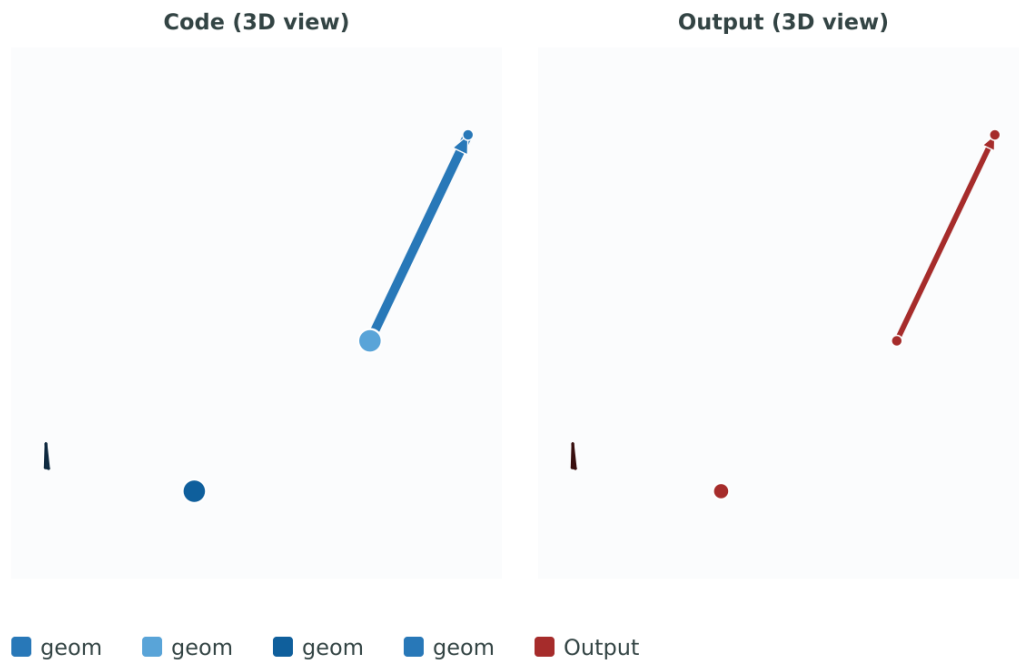
This 3D example does not mix dimensions.

Code

```
SELECT 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;
```

Output

```
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)))
```

Figure

This example unions an array built from a query.

Code

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

This example unions an array of linestrings.

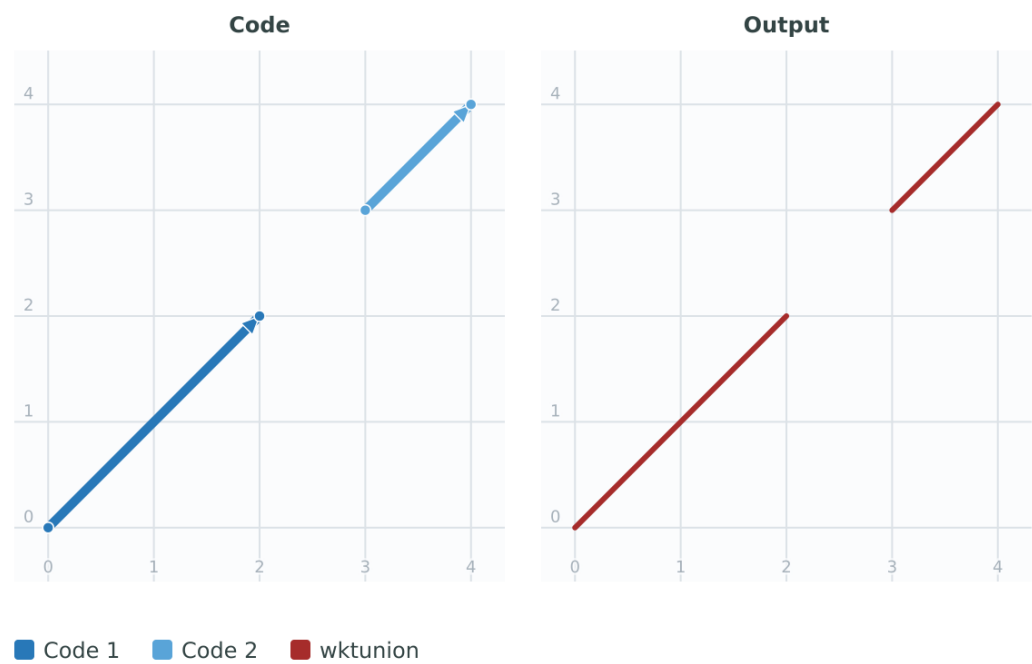
Code

```
SELECT ST_Union(ARRAY[
  'LINESTRING(1 2,3 4)::geometry',
  'LINESTRING(4 5,5 6)::geometry'
]) As wktunion;
```

Output

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

Figure



相关信息

[ST_CoverageUnion](#), [ST_UnaryUnion](#), [ST_MemUnion](#), [ST_Collect](#), [ST_Intersection](#), [ST_Difference](#), [ST_SymDifference](#), [ST_LineMerge](#)

7.14 几何处理

7.14.1 ST_Buffer

`ST_Buffer` — 计算覆盖距几何体指定距离内所有点的几何体。

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

计算表示与几何/地理距离小于或等于指定距离的所有点的 POLYGON 或 MULTIPOLYGON。距离会小于几何形而不是展示它。距离可能会完全小于多边形，在这种情况下返回 POLYGON EMPTY。对于点和线，距离始终返回空结果。

对于几何体，距离以几何体的空参考系的位置指定。对于地理，距离以米的位置指定。

可选项的第三个参数控制缓冲区的精度和样式。缓冲区中弧的精度指定用于近似四分之一圆的段数（默认为 8）。可以通过提供空白分隔的 = 列表来指定缓冲区样式，如下所示：

- 'quad_segs=#' : 用于近似四分之一圆的段数 (默认为 8)。
- 'endcap=round|flat|square' : endcap 样式 (默认为 “round”)。“butt” 被解释为 “flat” 的同义词。
- 'join=round|mitre|bevel' : 连接样式 (默认为 “round”)。“mitre” 被解释为 “mitre” 的同义词。
- 'mitre_limit=#.#' : 斜接比率限制 (影响斜接连接样式)。“miter_limit” 被接受为 “mitre_limit” 的同义词。
- 'side=both|left|right' : 默认为 both。left 或 right 会对 geometry 做缓冲, 缓冲方向以线的走向为参照。此选项适用于 LINESTRING, 对 POINT 或 POLYGON 无效。当指定 left 或 right 时, 线段末端默认使用方形端帽。

**Note**

缓冲用于地理数据, 是几何数据的一个度量封装。它确定最适合地理对象边界框的平面空间参考系 (例如 UTM、兰伯特方位角等 (LAEA) 北/南极, 最后是墨卡托)。缓冲在平面空间中计算, 然后转换回 WGS84。如果输入对象大于 UTM 区域或跨越日期变更线, 可能不会生成所需的行。

**Note**

缓冲操作可处理无效输入, 并始终生成有效的多边形几何对象。以距离 0 缓冲缓冲常被用作修复无效多边形的一种方法。但于此修复操作, **ST_MakeValid** 更适合, 因为它能处理多面体 MultiPolygon 类型。

**Note**

缓冲有用于行近距离搜索。于此用例, 使用 **ST_DWithin** 效率更高。

**Note**

此函数忽略 Z 维度。即使在 3D 几何体上使用, 它也始终输出 2D 结果。

增强: 2.5.0 - **ST_Buffer** 的几何感知版本已得到增强, 允许您指定要缓冲的一边。**side=both|left|right**。

可用性: 1.5 - **ST_Buffer** 已得到增强, 以适应各种端接和连接。例如, 您可能希望将道路串成街道面, 并希望将端点连接成平面或正方形而不是点。添加了地理的薄包装器。

该函数是由 GEOS 模块实现的。



此方法符合了 **SQL 1.1** 的 **OGC 功能规范**。s2.1.1.3



该方法符合了 **SQL/MM** 规范。SQL-MM IEC 13249-3: 5.1.30

示例

The textual results are rounded to whole coordinate units for readability. Each result remains available as text, while the manual initially favors the shared input/output figure.

quad_segs=8 (默认)

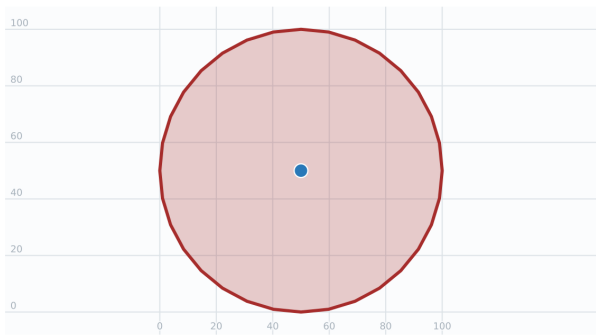
Code

```
SELECT ST_Buffer(  
  50, 'quad_segs=8');
```

Output

```
POLYGON((150 90,149 80,146 71,142 62,135 55,128 48,119  
44,110 41,100 40,90 41,81 44,72 48,65 55,58 62,54 71,51  
80,50 90,51 100,54 109,58 118,65 125,72 132,81 136,90  
139,100 140,110 139,119 136,128 132,135 125,142 118,146  
109,149 100,150 90))
```

Figure



quad_segs=2

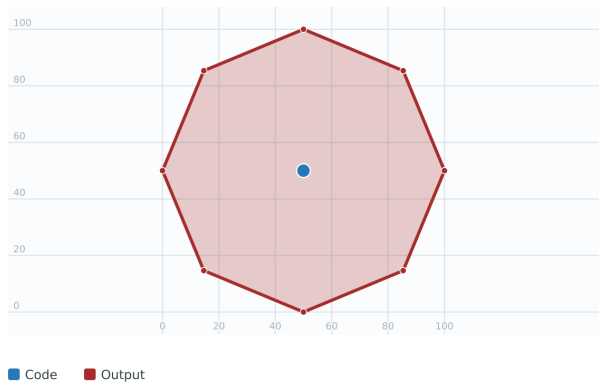
Code

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

Output

```
POLYGON((150 90,135 55,100 40,65 55,50 90,65 125,100  
140,135 125,150 90))
```

Figure



endcap=round join=round (默认)

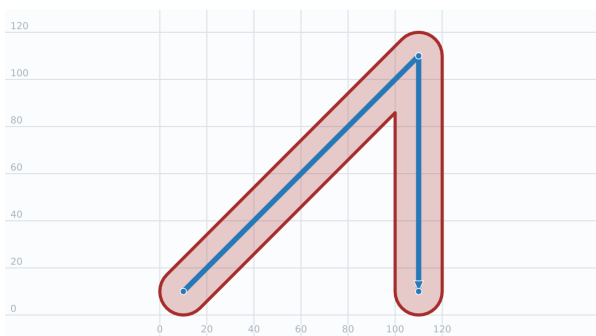
Code

```
SELECT ST_Buffer(  
  10, 'endcap=round join=round');
```

Output

```
POLYGON((143 157,144 158,146 159,148 160,150 160,152  
160,154 159,156 158,157 157,158 156,159 154,160 152,160  
150,160 50,160 48,159 46,158 44,157 43,156 42,154 41,152  
40,150 40,148 40,146 41,144 42,143 43,142 44,141 46,140  
48,140 50,140 126,57 43,56 42,54 41,52 40,50 40,48 40,46  
41,44 42,43 43,42 44,41 46,40 48,40 50,40 52,41 54,42  
56,43 57,143 157))
```

Figure



endcap=square

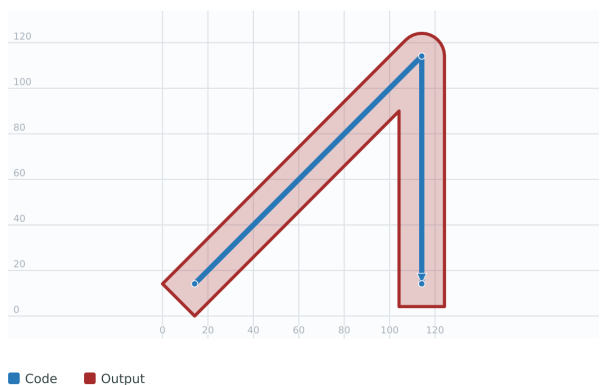
Code

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

Output

```
POLYGON((143 157,144 158,146 159,148 160,150 160,152  
160,154 159,156 158,157 157,158 156,159 154,160 152,160  
150,160 50,160 40,140 40,140 126,57 43,50 36,36 50,143  
157))
```

Figure



endcap=flat

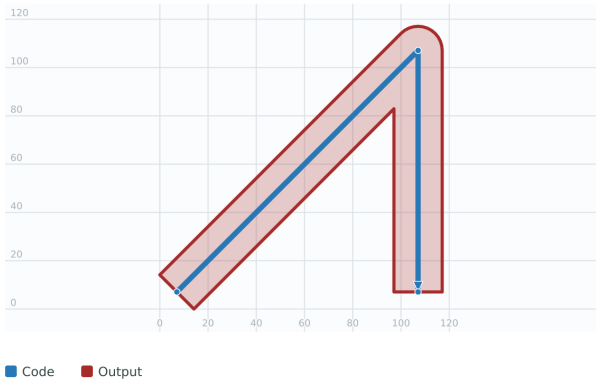
Code

```
SELECT ST_Buffer(  
    10, 'endcap=flat join=round');
```

Output

```
POLYGON((143 157,144 158,146 159,148 160,150 160,152  
160,154 159,156 158,157 157,158 156,159 154,160 152,160  
150,160 50,140 50,140 126,57 43,43 57,143 157))
```

Figure



join=bevel

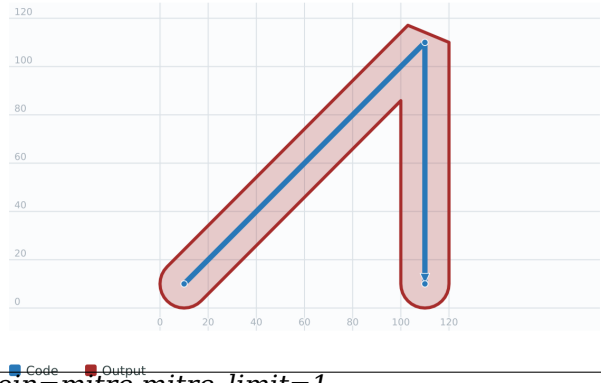
Code

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

Output

```
POLYGON((143 157,160 150,160 50,160 48,159 46,158 44,157  
43,156 42,154 41,152 40,150 40,148 40,146 41,144 42,143  
43,142 44,141 46,140 48,140 50,140 126,57 43,56 42,54  
41,52 40,50 40,48 40,46 41,44 42,43 43,42 44,41 46,40  
48,40 50,40 52,41 54,42 56,43 57,143 157))
```

Figure



join=mitre mitre_limit=5.0 (默认最大斜接比率)

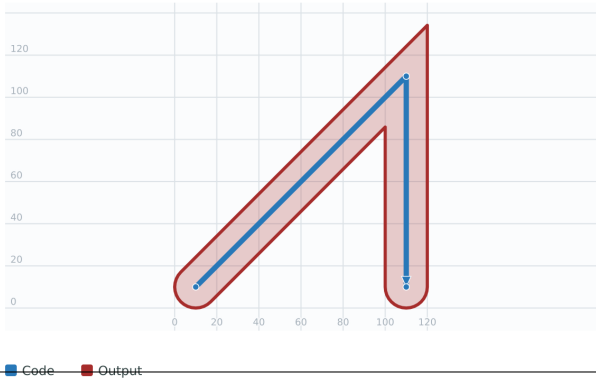
Code

```
SELECT ST_Buffer(  
    10, 'join=mitre mitre_limit=5.0');
```

Output

```
POLYGON((160 174,160 50,160 48,159 46,158 44,157 43,156  
42,154 41,152 40,150 40,148 40,146 41,144 42,143 43,142  
44,141 46,140 48,140 50,140 126,57 43,56 42,54 41,52 40,  
50 40,48 40,46 41,44 42,43 43,42 44,41 46,40 48,40 50,40  
52,41 54,42 56,43 57,160 174))
```

Figure



join=mitre mitre_limit=1

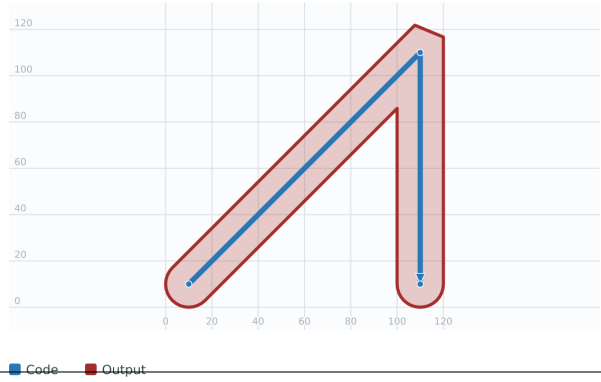
Code

```
SELECT ST_Buffer(  
    ST_GeomFromText('LINESTRING(50 50,150 150,150 50)'),  
    10, 'join=mitre mitre_limit=1.0');
```

Output

```
POLYGON((148 162,160 150,160 50,160 48,159 46,158 44,157  
43,156 42,154 41,152 40,150 40,148 40,146 41,144 42,143  
43,142 44,141 46,140 48,140 50,140 126,57 43,56 42,54  
41,52 40,50 40,48 40,46 41,44 42,43 43,42 44,41 46,40  
48,40 50,40 52,41 54,42 56,43 57,148 162))
```

Figure



side=left

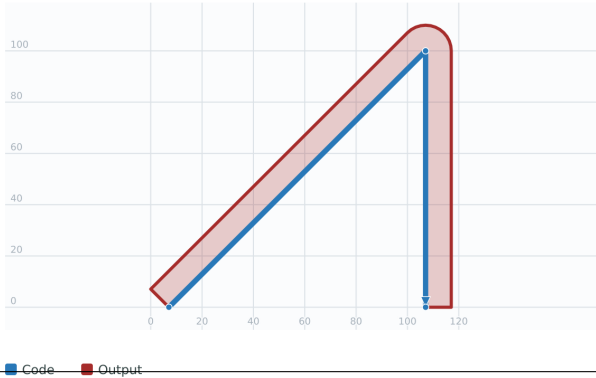
Code

```
SELECT ST_Buffer(  
    10, 'side=left');
```

Output

```
POLYGON((150 50,150 50,50,43 57,143 157,144 158,146  
159,148 160,150 160,152 160,154 159,156 158,157 157,158  
156,159 154,160 152,160 150,160 50,150 50))
```

Figure



side=right

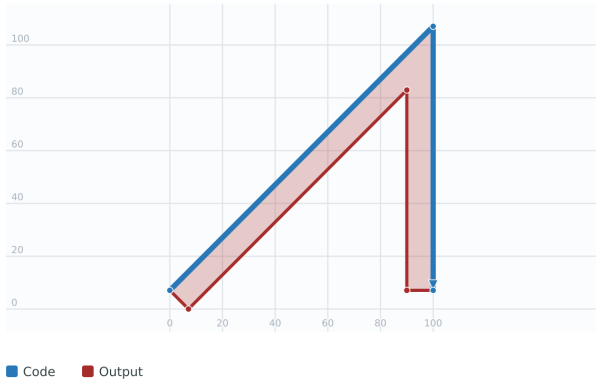
Code

```
ST_GeomFromText('LINESTRING(50 50,150 50)'),  
SELECT ST_Buffer(  
    10, 'side=right');
```

Output

```
POLYGON((50 50,150 50,150,140 50,140 126,57 43,50  
50))
```

Figure



side=left join=mitre

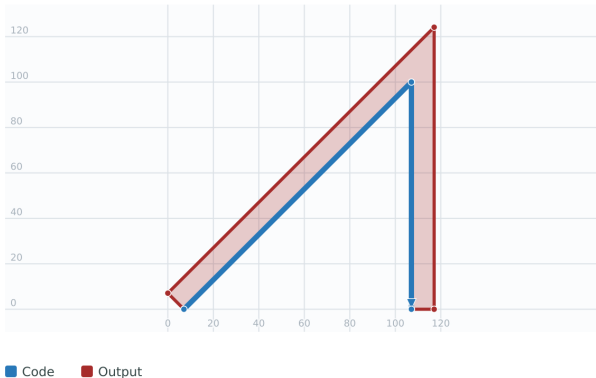
Code

```
SELECT ST_Buffer(  
    10, 'side=left join=mitre');
```

Output

```
POLYGON((150 50,150 50,50,43 57,160 174,160 50,150  
50))
```

Figure



*right-hand winding, polygon boundary
side=left*

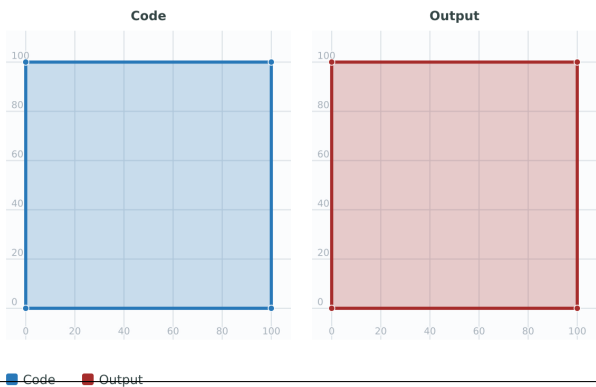
Code

```
SELECT ST_Buffer('LINESTRING(50 50,150 50)',  
    'POLYGON ((50 50,50 150,150 150 50,50 50))'),  
    20, 'side=left');
```

Output

```
POLYGON((50 50,50 150 150,150 50,50 50))
```

Figure



right-hand winding, polygon boundary
side=right

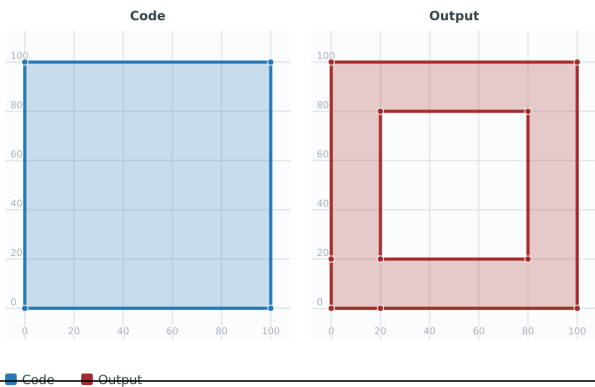
Code

```
SELECT ST_Buffer(  
    'POLYGON ((50 50,50 150,150 150,150 50,50 50))',  
    20, 'side=right');
```

Output

```
POLYGON((50 50,50 70,50 150,150 150,150 50,70 50,50 50),  
(70 70,130 70,130 130,70 130,70 70))
```

Figure



```
ST_ForceRHR(ST_Boundary(ST_GeomFromText (
```

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)

Code

```
SELECT ST_NPoints(ST_Buffer('POINT(100 90)::geometry, 50)) As promisingcircle_pcount,  
ST_NPoints(ST_Buffer('POINT(100 90)::geometry, 50, 2)) As lamecircle_pcount;
```

Output

promisingcircle_pcount	lamecircle_pcount
33	9

A lighter, less smooth circle using only 2 points per quarter circle is an octagon.

The following example creates a 100 meter octagon in NAD83 longitude/latitude by transforming to the Massachusetts state plane meter projection and then buffering. The figure shows the input and output in separate panels because the coordinates are in different SRIDs.

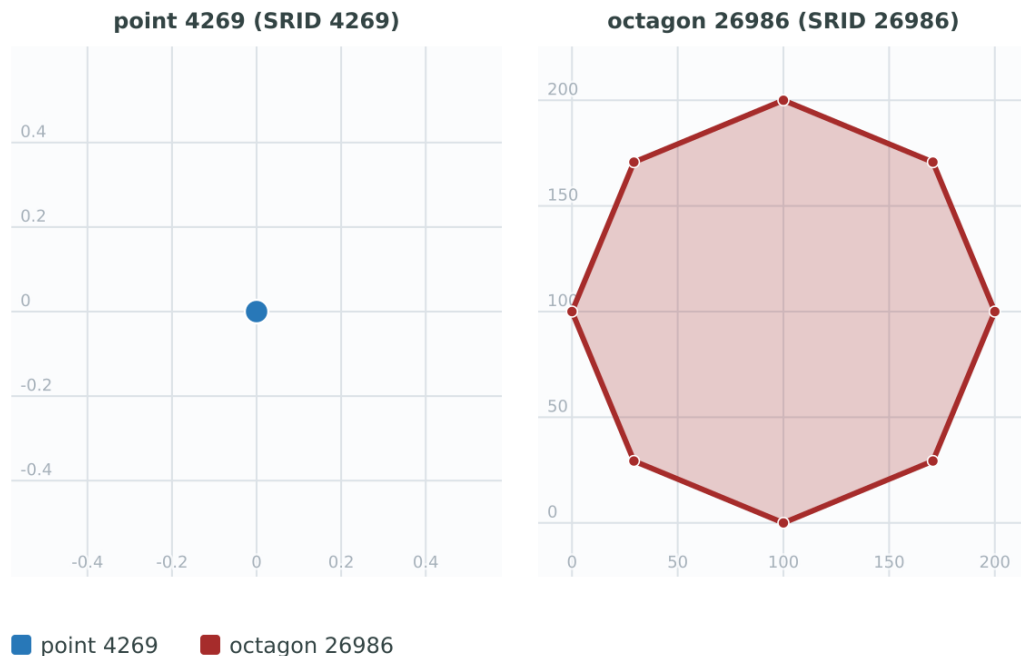
Code

```
WITH input AS (  
    SELECT ST_SetSRID(ST_Point(-71.063526, 42.35785), 4269) AS point_4269  
) , projected AS (  
    SELECT point_4269,  
           ST_Transform(point_4269, 26986) AS point_26986  
    FROM input  
)  
SELECT point_4269 AS input_point_4269,  
       ST_Buffer(point_26986, 100, 2) AS octagon_26986  
FROM projected;
```

Output

```
SRID=4269;POINT(-71.063526 42.35785) | SRID=26986;POLYGON((236057.5905746494 900908.7599186979,236028.30125276805
900838.0492405792,235957.5905746494 900808.7599186979,235886.87989653074 900838.0492405792,235857.5905746494
900908.7599186979,235886.87989653074 900979.4705968165,235957.5905746494 901008.7599186979,236028.30125276805
900979.4705968165,236057.5905746494 900908.7599186979))
```

Figure



Buffering a polygon by a positive distance and then by the same negative distance is a morphological closing operation. It can be useful for smoothing noisy boundaries, such as simplifying a coastline by removing narrow inlets and bays. The result intentionally changes the shape and area of the input, and the distance must be chosen in the units of the input SRS. For data in a geographic SRS, transform to a projected SRS suited to the operation first; for cartographic visualization in Web Mercator, using that display SRS can be appropriate.

Smooth a coastline using a 1 km closing operation, then simplify the result. The example uses Massachusetts State Plane meter units for the operation.

Code

```
WITH projected AS (
  SELECT id, ST_Transform(geom, 26986) AS geom
  FROM coast
),
closed AS (
  SELECT id, ST_Buffer(ST_Buffer(geom, 1000), -1000) AS geom
  FROM projected
)
SELECT id, ST_Transform(ST_SimplifyPreserveTopology(geom, 50), 4326) AS geom
FROM closed;
```

The same technique is described in Paul Ramsey's [Removing Complexities](#). The final simplification step is optional; use [ST_SimplifyPreserveTopology](#) when polygonal validity matters, and check the result against the cartographic or analytical tolerance required by the application.

A small negative buffer can also be used as a heuristic to find narrow polygon spikes. The query below compares how much area and perimeter remain after erosion. A large `erosion_index` indicates that the perimeter dropped proportionally more than the area. This is only a preselection aid; the threshold and buffer distance must be chosen for the data units and the kind of spike being checked.

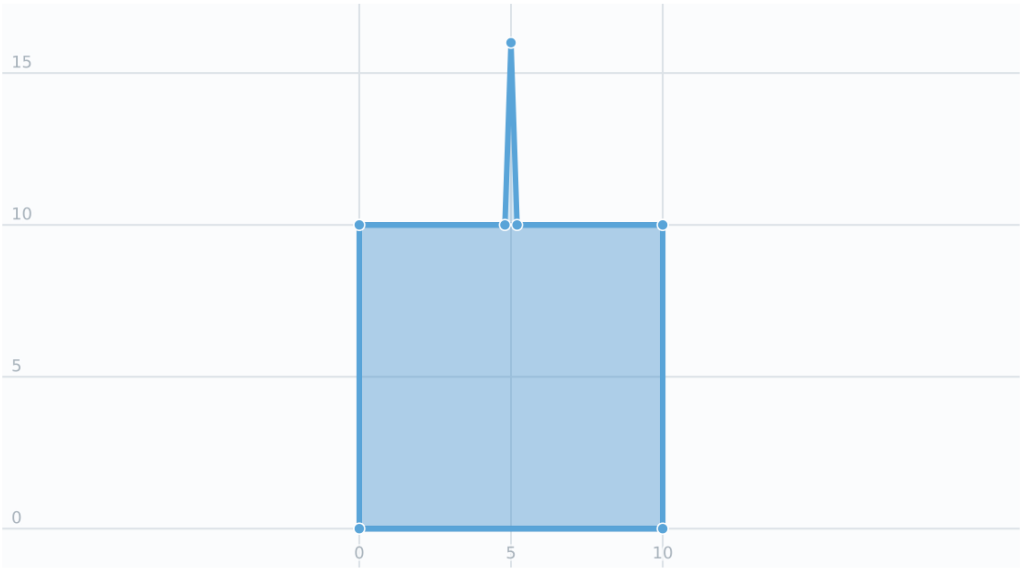
Code

```
WITH sample(id, geom) AS (  
  VALUES  
    ('plain', 'POLYGON((0 0,10 0,10 10,0 10,0 0))'::geometry),  
    ('spike', 'POLYGON((0 0,10 0,10 10,5.2 10,5 16,4.8 10,0 10,0 0))'::geometry)  
),  
eroded AS (  
  SELECT id, geom, ST_Buffer(geom, -0.5) AS eroded_geom  
  FROM sample  
),  
scored AS (  
  SELECT id,  
    round(  
      ST_Area(eroded_geom) / ST_Area(geom) /  
      NULLIF(ST_Perimeter(eroded_geom) / ST_Perimeter(geom), 0)  
    )::numeric, 3) AS erosion_index  
  FROM eroded  
  WHERE NOT ST_IsEmpty(eroded_geom)  
)  
SELECT id, erosion_index, erosion_index > 1.1 AS possible_spike  
FROM scored  
ORDER BY id;
```

Output

id	erosion_index	possible_spike
plain	0.900	f
spike	1.147	t

Figure



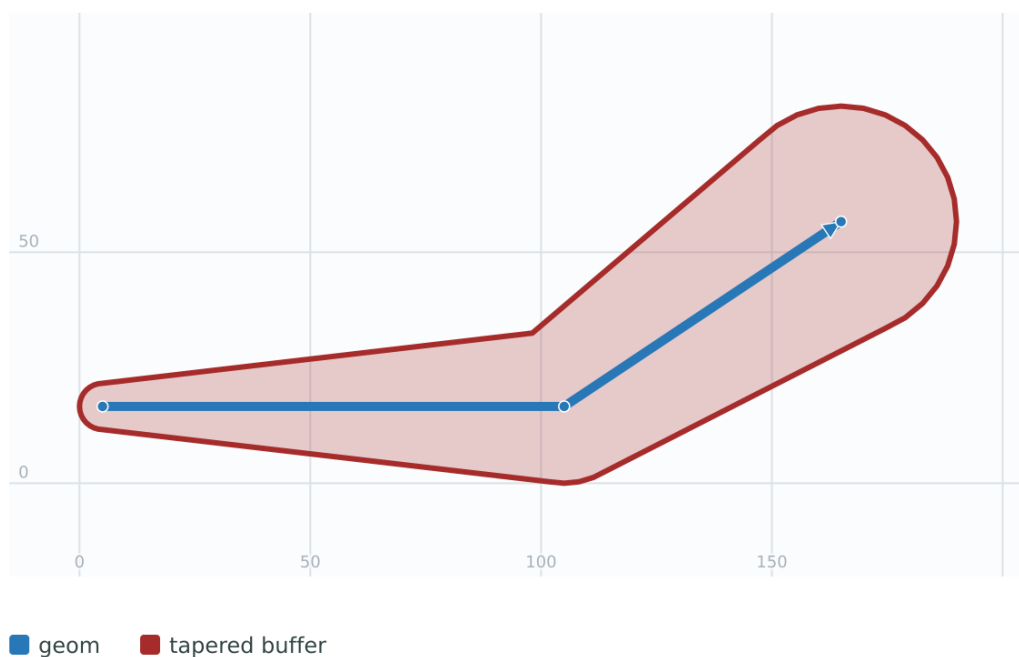
Code 1 Code 2

A variable-distance buffer can be approximated by buffering each vertex of a line by a distance interpolated along the line, building the convex hull for each consecutive pair of vertex buffers, and unioning the segment hulls. As with any geometry buffer, distances are in the units of the input SRS. Taper a line buffer from 5 to 25 units along the line.

Code

```
WITH line AS (
  SELECT 'LINESTRING(0 0, 100 0, 160 40)::geometry AS geom
),
vertices AS (
  SELECT (dump).path[1] AS n,
         (dump).geom AS geom,
         line.geom AS line_geom
  FROM line
  CROSS JOIN LATERAL ST_DumpPoints(line.geom) AS dump
),
radii AS (
  SELECT n,
         geom,
         5 + (25 - 5) * ST_LineLocatePoint(line_geom, geom) AS radius
  FROM vertices
),
segments AS (
  SELECT n,
         geom,
         radius,
         lead(geom) OVER (ORDER BY n) AS next_geom,
         lead(radius) OVER (ORDER BY n) AS next_radius
  FROM radii
)
SELECT ST_Union(
  ST_ConvexHull(
    ST_Collect(
      ST_Buffer(geom, radius),
      ST_Buffer(next_geom, next_radius)
    )
  )
) AS tapered_buffer
FROM segments
WHERE next_geom IS NOT NULL;
```

Figure



相关信息

[ST_Collect](#), [ST_ConvexHull](#), [ST_DWithin](#), [ST_DumpPoints](#), [ST_LineLocatePoint](#), [ST_SetSRID](#), [ST_SimplifyPreserve](#), [ST_Transform](#), [ST_Union](#), [ST_MakeValid](#)

7.14.2 ST_BuildArea

`ST_BuildArea` — 构建由几何体的线条形成的多边形几何体。

Synopsis

geometry **ST_BuildArea**(geometry geom);

描述

构建由输入几何体的线条形成的面几何体。输入可以是 `LineString`、`MultiLineString`、`Polygon`、`MultiPolygon` 或 `GeometryCollection`。如果是多边形或多多边形，具体取决于输入。如果输入线条未形成多边形，返回 `NULL`。

与 [ST_MakePolygon](#) 不同，此函数接受由多条线形成的面，并且可以形成任意数量的多边形。

此功能将内孔填充。要将内孔也填充为多边形，使用 [ST_Polygonize](#)。



Note

输入线条必须正确，点才能使此功能正常工作。[ST_Node](#) 可用于点。如果输入线条交叉，此函数将生成无效的多边形。[ST_MakeValid](#) 可用于确保输出有效。

可用性：1.1.0

示例

Building an area from five input lines.

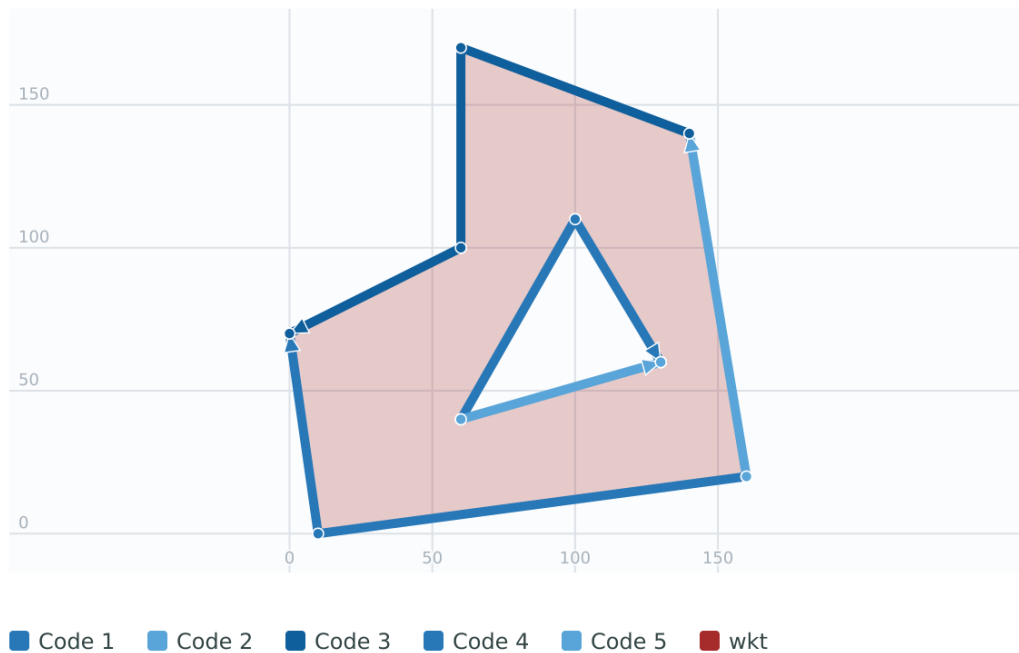
Code

```
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_BuildArea(ST_Collect(geom )) AS wkt
FROM data;
```

Output

```
POLYGON((180 40,30 20,20 90,80 120,80 190,160 160,180 40),(150 80,120 130,80 60,150 80))
```

Figure

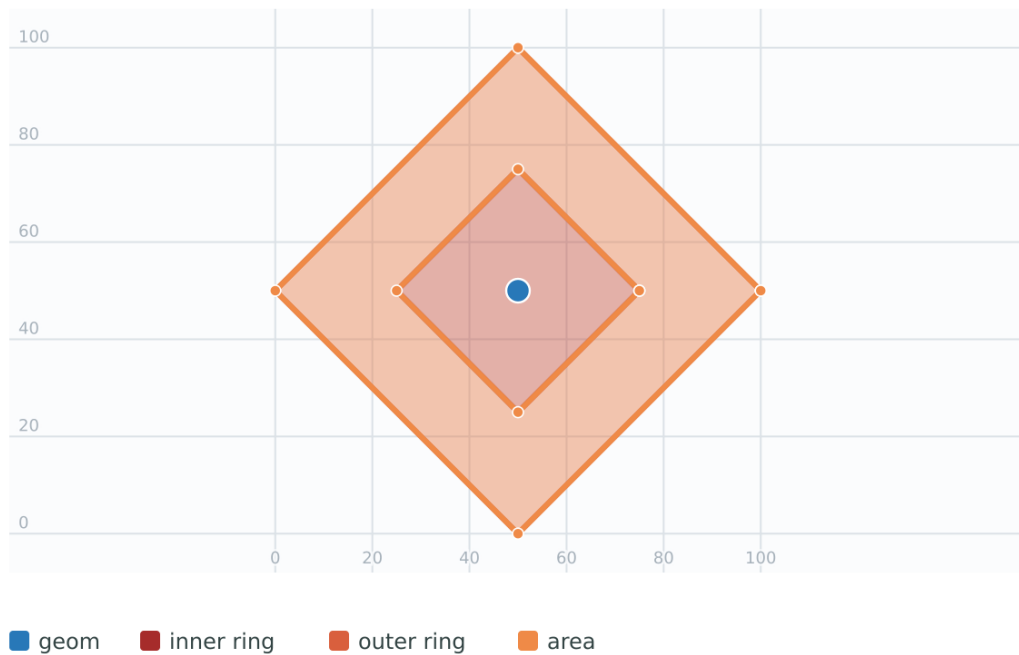


Create a donut from two circular polygons.

Code

```
WITH input AS (
  SELECT 'POINT(100 90)::geometry AS geom
),
rings AS (
  SELECT ST_Buffer(geom, 25, 'quad_segs=1') AS inring,
         ST_Buffer(geom, 50, 'quad_segs=1') AS outring
  FROM input
)
SELECT inring AS inner_ring,
       outring AS outer_ring,
       ST_BuildArea(ST_Collect(inring, outring)) AS area
FROM rings;
```

Figure



Input linework that crosses at non-endpoint intersections must be noded before building areas:

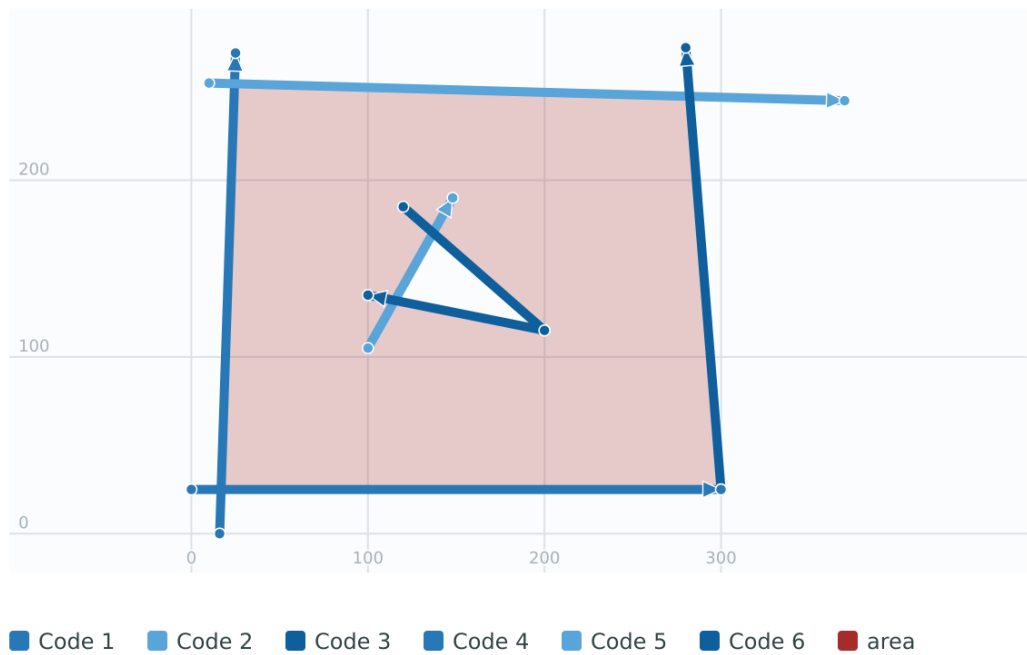
Code

```
WITH lines(geom) AS (VALUES
  ('LINESTRING(36 35,45 307)')::geometry),
  ('LINESTRING(30 290,390 280)')::geometry),
  ('LINESTRING(320 60,300 310)')::geometry),
  ('LINESTRING(20 60,320 60)')::geometry),
  ('LINESTRING(120 140,168 225)')::geometry),
  ('LINESTRING(140 220,220 150,120 170)')::geometry)
),
noded AS (
  SELECT ST_Node(ST_UnaryUnion(ST_Collect(geom))) AS geom FROM lines
)
SELECT ST_BuildArea(geom) AS area
FROM noded;
```

Output

```
POLYGON((320 60,36.8 60,44.4 289.6,302.2 282.4,320 60),(156.9 205.3,135.2 167,220 150,156.9 205.3))
```

Figure



相关信息

[ST_Collect](#), [ST_MakePolygon](#), [ST_MakeValid](#), [ST_Node](#), [ST_Polygonize](#), [ST_BdPolyFromText](#), [ST_BdMPolyFromText](#)
(使用 `ogr2ogr` 接口 `ogr2ogr` 函数 `ogr2ogr` 行包装)

7.14.3 ST_Centroid

ST_Centroid — 返回几何体的几何中心。

Synopsis

```
geometry ST_Centroid(geometry g1);
geography ST_Centroid(geography g1, boolean use_spheroid = true);
```

描述

计算几何体的几何中心的点。对于 [MULTI]POINT，中心是输入坐标的算术平均。对于 [MULTI]LINESTRING，中心是使用每条线段的加权长度计算的。对于 [MULTI]POLYGON，中心是根据面积计算的。如果提供了空几何形状，返回空的 GEOMETRYCOLLECTION。如果提供 NULL，返回 NULL。如果提供了 CIRCULARSTRING 或 COMPOUNDCURVE，它首先使用 CurveToLine 将其转换为线串，然后与 LINESTRING 相同。

对于混合维度输入，结果等于最高维度的几何体形状的中心（因为低维度的几何体形状的中心“重量”贡献为零）。

注意，对于多边形几何形状，中心不一定位于多边形的内部。例如，参见下图 C 形多边形的中心。要确保位于多边形内部的点，使用 [ST_PointOnSurface](#)。

2.3.0 中的新增功能：支持 CIRCULARSTRING 和 COMPOUNDCURVE（使用 CurveToLine）

可用性：2.4.0 引入了地理的支持。

-  此方法符合了 SQL 1.1 的 OGC 功能规范。
-  此方法符合了 SQL/MM 规范。 SQL-MM 3: 8.1.4, 9.5.5

示例

Centroid of a MultiPoint.

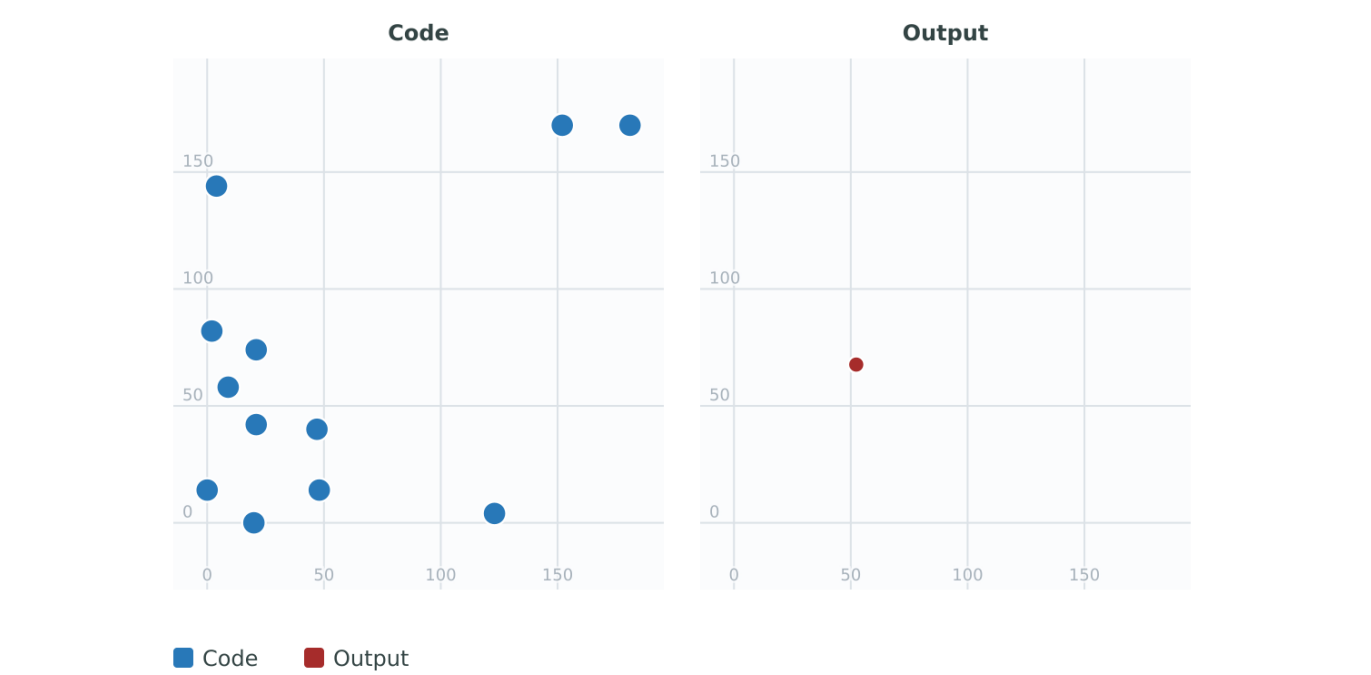
Code

```
SELECT ST_Centroid(  
  'MULTIPOINT (8 24,10 92,12 154,17 68,28 10,29 52,29 84,55 50,56 24,131 14,160 180,189 180)' );
```

Output

```
POINT(60.333 77.667)
```

Figure



Centroid of a LineString.

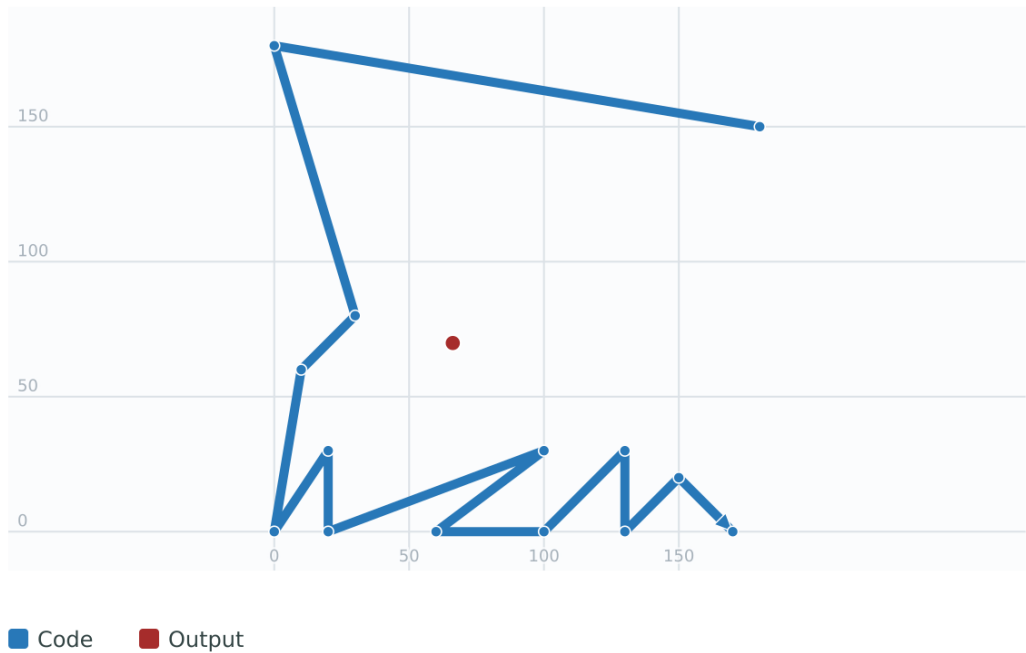
Code

```
SELECT ST_Centroid(  
  'LINESTRING (190 160,10 190,40 90,20 70,10 10,30 40,30 10,110 40,70 10,110 10,140 40,140 10,160 30,180 10)' );
```

Output

```
POINT(76.191 79.876)
```

Figure



Centroid of a Polygon.

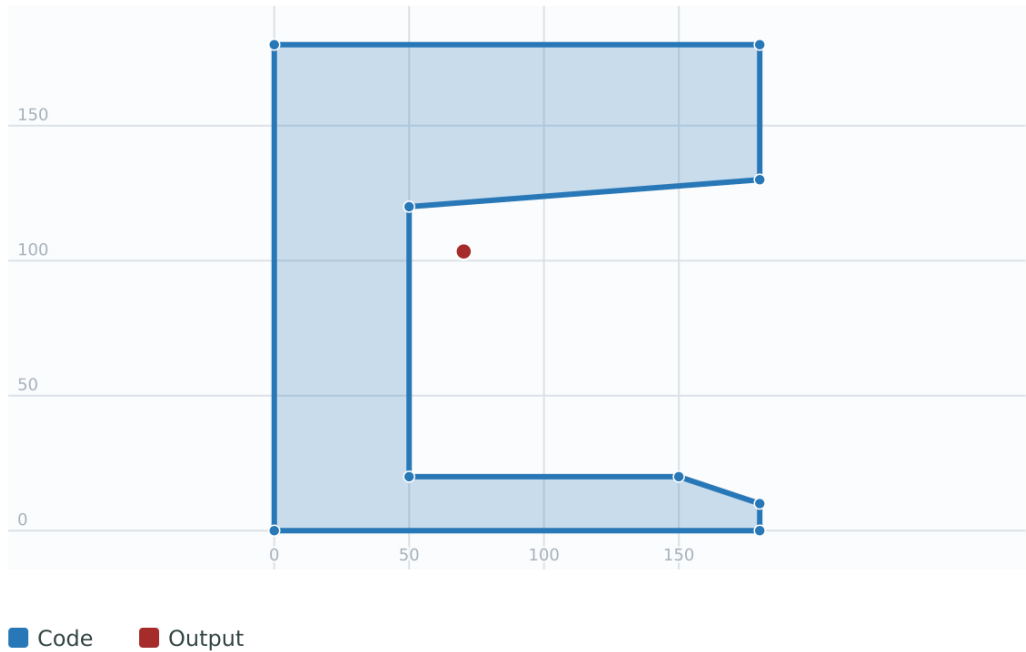
Code

```
SELECT ST_Centroid(  
  'POLYGON ((190 190,10 190,10 10,190 10,190 20,160 30,60 30,60 130,190 140,190 190))' );
```

Output

```
POINT(80.251 113.405)
```

Figure



Centroid of the highest-dimension components of a GeometryCollection.

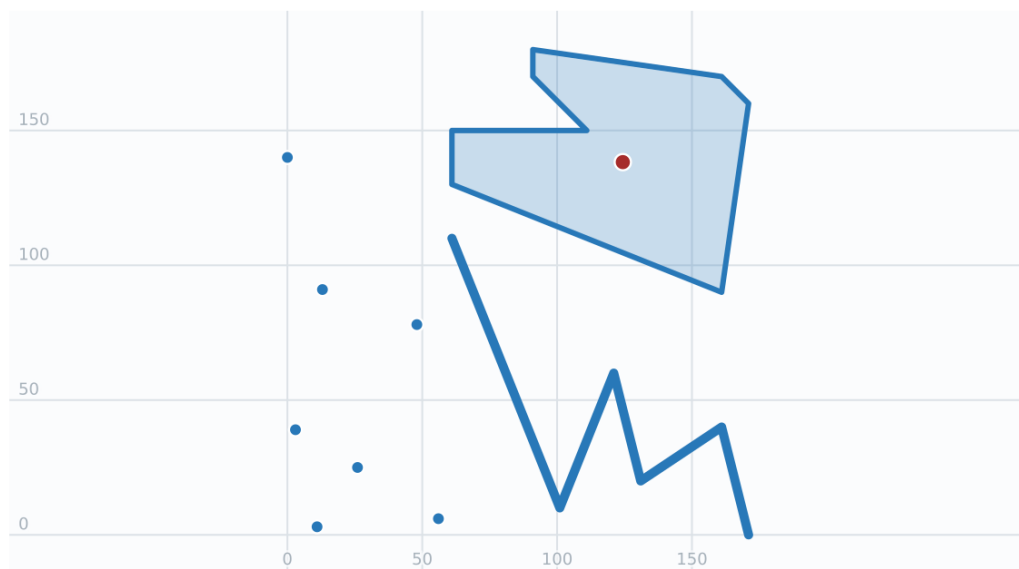
Code

```
SELECT ST_Centroid(
  'GEOMETRYCOLLECTION (POLYGON ((190 170,180 100,80 140,80 160,130 160,110 180,110 190,180 180,190 170)),
    LINestring (80 120,120 20,140 70,150 30,180 50,190 10),
    MULTIPOINT (19 150,22 49,30 13,32 101,45 35,67 88,75 16))')');
```

Output

```
POINT(143.361 148.263)
```

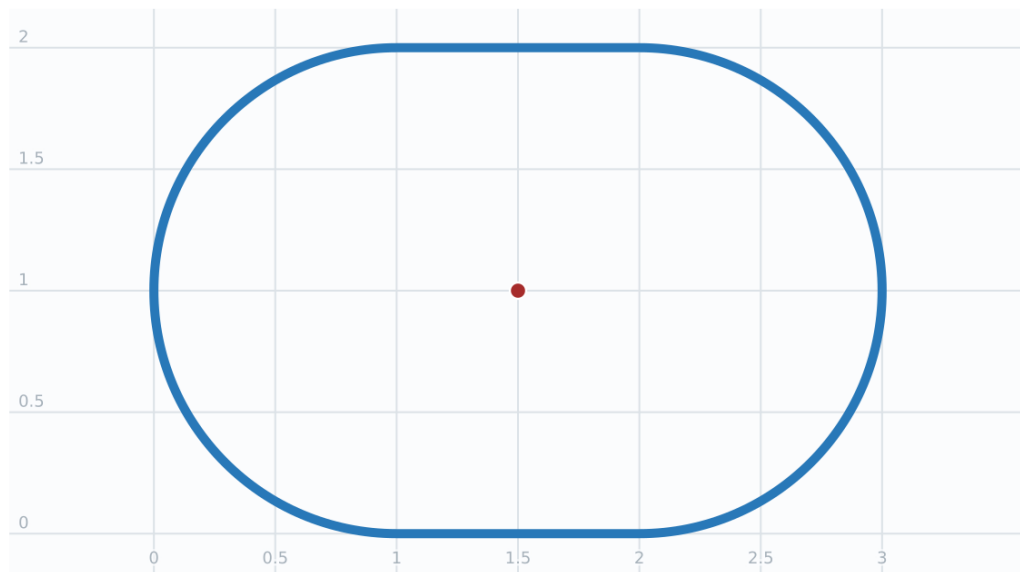
Figure



■ Code ■ Output

Code

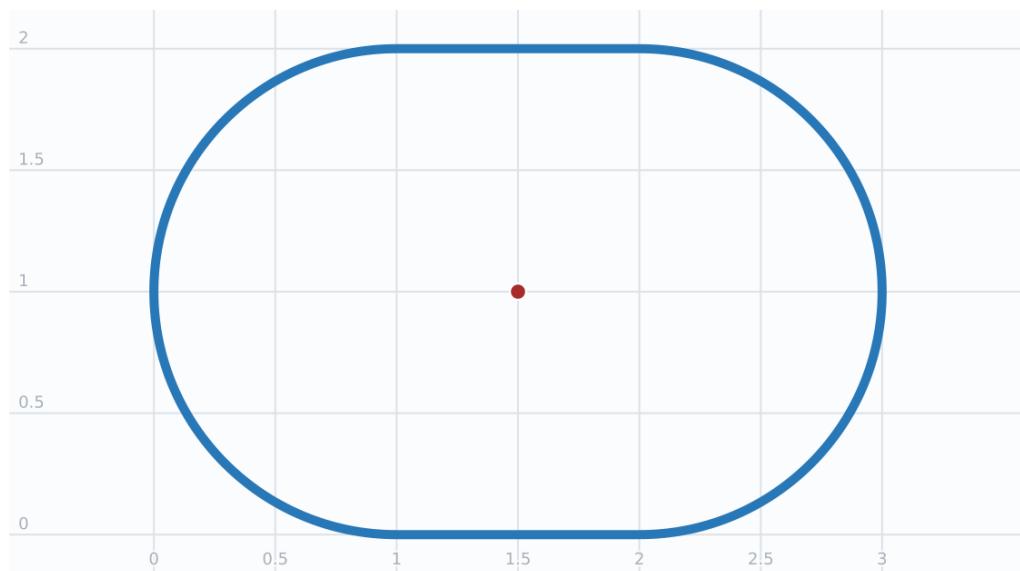
```
SELECT ST_Centroid(g)
FROM ST_GeomFromText('CIRCULARSTRING(0 2,-1 1,0 0,0.5 0,1 0,2 1,1 2,0.5 2,0 2)') AS g ;
```

Figure

■ Code ■ Output

Code

```
SELECT 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;
```

Figure

■ Code ■ Output

Centroids of the parts of a MultiPolygon.

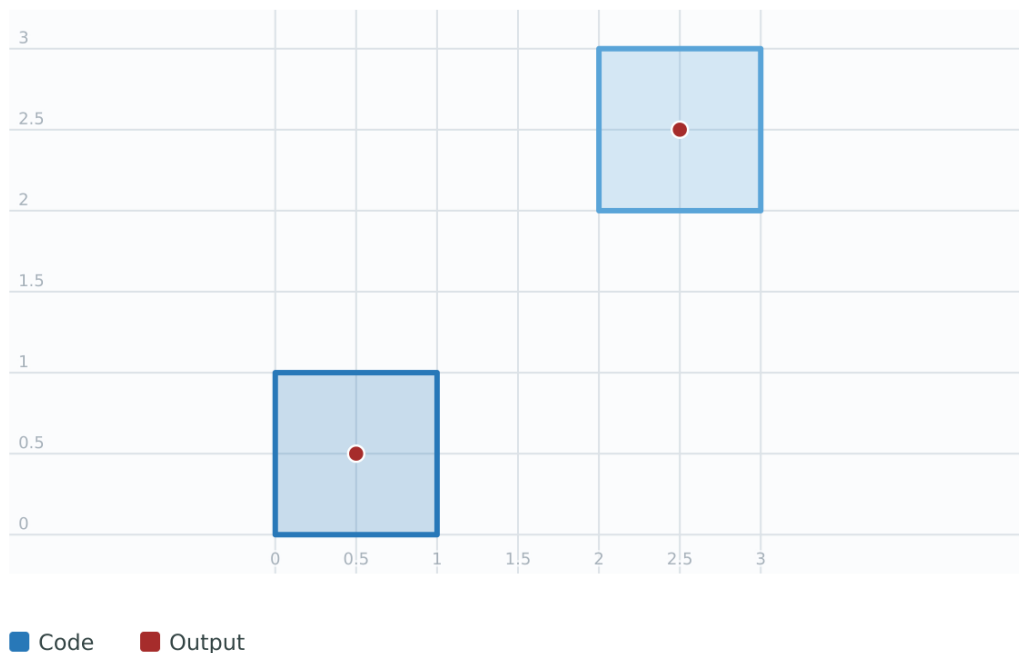
Code

```
SELECT ST_Collect(ST_Centroid(geom) ORDER BY path)
FROM ST_Dump('MULTIPOLYGON (
  ((0 0,0 1,1 1,1 0,0 0)),
  ((2 2,2 3,3 3,3 2,2 2))
) '::geometry);
```

Output

```
MULTIPOINT((0.5 0.5),(2.5 2.5))
```

Figure



相关信息

[ST_Collect](#), [ST_Dump](#), [ST_PointOnSurface](#), [ST_GeometricMedian](#)

7.14.4 ST_ChaikinSmoothing

`ST_ChaikinSmoothing` — 使用 Chaikin 算法返回几何图形的平滑版本

Synopsis

geometry **ST_ChaikinSmoothing**(geometry geom, integer nIterations = 1, boolean preserveEndpoints = false);

描述

使用 **Chaikin 算法** 平滑点或面几何体。平滑程度由 `nIterations` 参数控制。在每次迭代中，每个内部点都被位于点前后段度 1/4 的点替换。3 次迭代提供了合理的平滑度；最大限制 5。

如果 `preserveEndpoints` 为 `true`，多面形的端点不会被平滑。`LineStrings` 的端点始终被保留。

**Note**

每次迭代点数量都会加倍，因此如果几何图形的点可能比输入多得多。要减少点数，如果使用简化函数（参考 [ST_Simplify](#)、[ST_SimplifyPreserveTopology](#) 和 [ST_SimplifyVW](#)）。

如果具有 Z 和 M 度（如果存在）的插值。



函数支持 3d 并且不会丢失 z-index。

可用性：2.5.0

示例

Smoothing a Polygon using 1, 2 and 3 iterations.

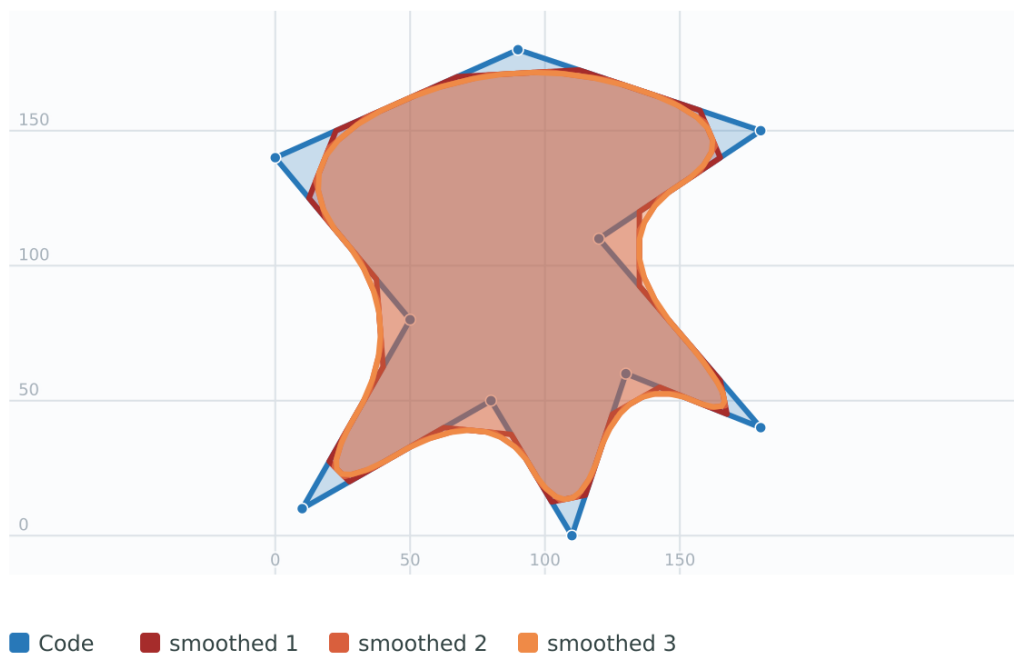
Code

```
SELECT n AS iterations, 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) ←
  )', n
) AS smoothed
FROM generate_series(1, 3) AS n;
```

Output

```
1 | POLYGON((30 37.5,50 72.5,47.5 105,22.5 135,32.5 160,77.5 180,122.5 182.5,167.5 167.5,175 150,145 130,145 102.5,175 ←
  67.5,177.5 55,152.5 65,135 55,125 25,112.5 22.5,97.5 47.5,72.5 50,37.5 30,30 37.5))
2 | POLYGON((35 46.25,45 63.75,49.375 80.625,48.125 96.875,41.25 112.5,28.75 127.5,25 141.25,30 153.75,43.75 165,66.25 ←
  175,88.75 180.625,111.25 181.875,133.75 178.75,156.25 171.25,169.375 163.125,173.125 154.375,167.5 145,152.5 ←
  135,145 123.125,145 109.375,152.5 93.75,167.5 76.25,175.625 64.375,176.875 58.125,171.25 57.5,158.75 62.5,148.125 ←
  62.5,139.375 57.5,132.5 47.5,127.5 32.5,121.875 24.375,115.625 23.125,108.75 28.75,101.25 41.25,91.25 ←
  48.125,78.75 49.375,63.75 45,46.25 35,35.625 31.875,31.875 35.625,35 46.25))
3 | POLYGON((37.5 50.625,42.5 59.375,46.09375 67.96875,48.28125 76.40625,49.0625 84.6875,48.4375 92.8125,46.40625 ←
  100.78125,42.96875 108.59375,38.125 116.25,31.875 123.75,27.8125 130.9375,25.9375 137.8125,26.25 144.375,28.75 ←
  150.625,33.4375 156.5625,40.3125 162.1875,49.375 167.5,60.625 172.5,71.875 176.40625,83.125 179.21875,94.375 ←
  180.9375,105.625 181.5625,116.875 181.09375,128.125 179.53125,139.375 176.875,150.625 173.125,159.53125 ←
  169.21875,166.09375 165.15625,170.3125 160.9375,172.1875 156.5625,171.71875 152.03125,168.90625 147.34375,163.75 ←
  142.5,156.25 137.5,150.625 132.03125,146.875 126.09375,145 119.6875,145 112.8125,146.875 105.46875,150.625 ←
  97.65625,156.25 89.375,163.75 80.625,169.53125 73.28125,173.59375 67.34375,175.9375 62.8125,176.5625 ←
  59.6875,175.46875 57.96875,172.65625 57.65625,168.125 58.75,161.875 61.25,156.09375 62.5,150.78125 62.5,145.9375 ←
  61.25,141.5625 58.75,137.65625 55,134.21875 50,131.25 43.75,128.75 36.25,126.09375 30.46875,123.28125 ←
  26.40625,120.3125 24.0625,117.1875 23.4375,113.90625 24.53125,110.46875 27.34375,106.875 31.875,103.125 ←
  38.125,98.75 42.96875,93.75 46.40625,88.125 48.4375,81.875 49.0625,75 48.28125,67.5 46.09375,59.375 42.5,50.625 ←
  37.5,43.59375 34.21875,38.28125 32.65625,34.6875 32.8125,32.8125 34.6875,32.65625 38.28125,34.21875 43.59375,37.5 ←
  50.625))
```

Figure



Smoothing a LineString using 1, 2 and 3 iterations.

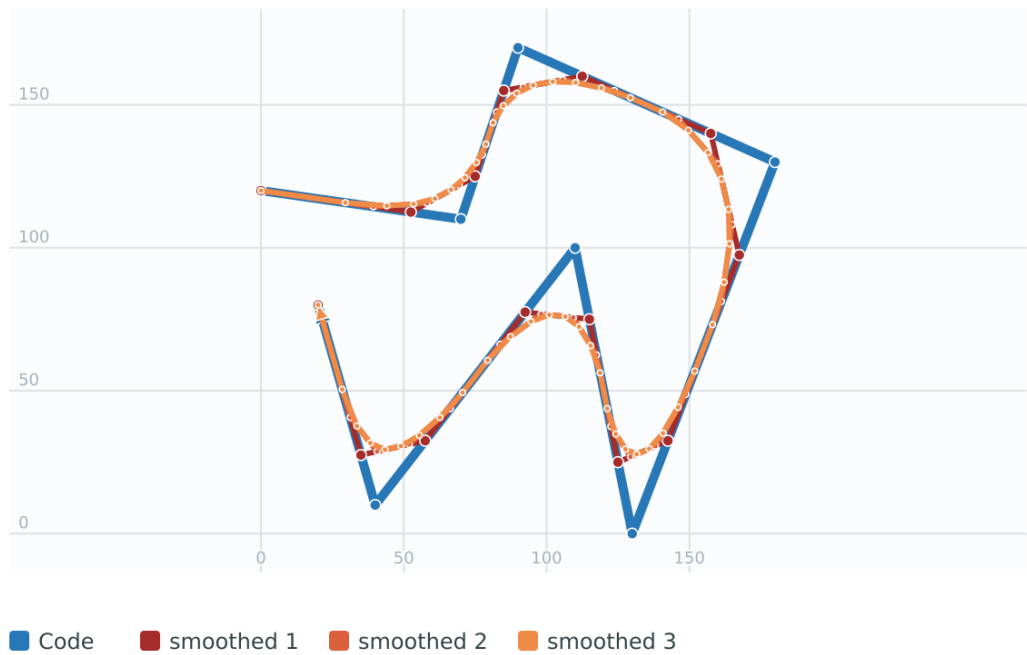
Code

```
SELECT n AS iterations, ST_ChaikinSmoothing(
  'LINESTRING (10 140,80 130,100 190,190 150,140 20,120 120,50 30,30 100)', n
) AS smoothed
FROM generate_series(1, 3) AS n;
```

Output

```
1 | LINESTRING(10 140,62.5 132.5,85 145,95 175,122.5 180,167.5 160,177.5 117.5,152.5 52.5,135 45,125 95,102.5 97.5,67.5 52.5,45 47.5,30 100) ↵
2 | LINESTRING(10 140,49.375 134.375,68.125 135.625,79.375 141.875,87.5 152.5,92.5 167.5,101.875 176.25,115.625 178.75,133.75 175,156.25 165,170 149.375,175 128.125,171.25 101.25,158.75 68.75,148.125 50.625,139.375 46.875,132.5 57.5,127.5 82.5,119.375 95.625,108.125 96.875,93.75 86.25,76.25 63.75,61.875 51.25,50.625 48.75,41.25 60.625,30 100) ↵
3 | LINESTRING(10 140,39.53125 135.78125,54.0625 134.6875,63.4375 135.3125,70.9375 137.1875,76.5625 140.3125,81.40625 144.53125,85.46875 149.84375,88.75 156.25,91.25 163.75,94.84375 169.6875,99.53125 174.0625,105.3125 176.875,112.1875 178.125,120.15625 177.8125,129.21875 175.9375,139.375 172.5,150.625 167.5,159.6875 161.09375,166.5625 153.28125,171.25 144.0625,173.75 133.4375,174.0625 121.40625,172.1875 107.96875,168.125 93.125,161.875 76.875,156.09375 64.21875,150.78125 55.15625,145.9375 49.6875,141.5625 47.8125,137.65625 49.53125,134.21875 54.84375,131.25 63.75,128.75 76.25,125.46875 85.78125,121.40625 92.34375,116.5625 95.9375,110.9375 96.5625,104.53125 94.21875,97.34375 88.90625,89.375 80.625,80.625 69.375,72.65625 60.625,65.46875 54.375,59.0625 50.625,53.4375 49.375,48.28125 51.71875,43.59375 57.65625,38.4375 70.46875,30 100) ↵
```

| **Figure**



Smoothing a triangle shows the weakest visual change, so it is kept last.

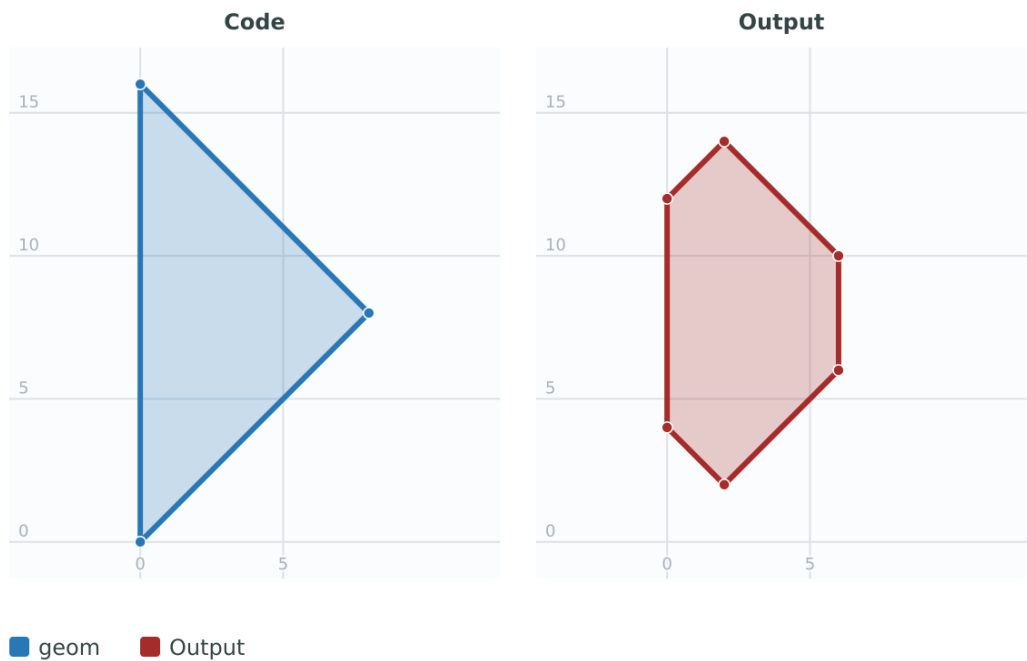
| **Code**

```
SELECT ST_ChaikinSmoothing(geom) smoothed
FROM (SELECT 'POLYGON((0 0,8 8,0 16,0 0))'::geometry geom) AS foo;
```

| **Output**

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

| **Figure**



相关信息

[ST_Simplify](#), [ST_SimplifyPreserveTopology](#), [ST_SimplifyVW](#)

7.14.5 ST_CatmullRomSmoothing

ST_CatmullRomSmoothing — Returns a smoothed version of a geometry, using the Catmull-Rom spline algorithm

Synopsis

geometry **ST_CatmullRomSmoothing**(geometry geom, integer nSegments = 5);

描述

Smooths a linear or polygonal geometry using the [Catmull-Rom spline](#) algorithm. Unlike [ST_ChaikinSmooth](#) this is an *interpolating* spline: the output curve passes through every original vertex. Between each pair of consecutive original vertices, `nSegments - 1` new intermediate points are inserted.

The `nSegments` parameter controls the density of the output. With `nSegments = 5` (the default), each original span is divided into 5 sub-segments (inserting 4 new points per span). The minimum value is 2.

At least 4 input vertices are required to apply smoothing; geometries with fewer vertices are returned unchanged. Points and multipoints are always returned unchanged.

Note



The output vertex count grows as $1 + (N-1) * nSegments$ for open lines, and $1 + N * nSegments$ for closed rings, where N is the number of input vertices. For large geometries, use a simplification function on the result to reduce the number of points (see [ST_Simplify](#), [ST_SimplifyPreserveTopology](#) and [ST_SimplifyVW](#)).

☑️ 果具有 Z 和 M ☑️ 度（如果存在）的插☑️。



☑️ 函数支持 3d 并且不会☑️ 失 z-index。

Availability: 3.7.0

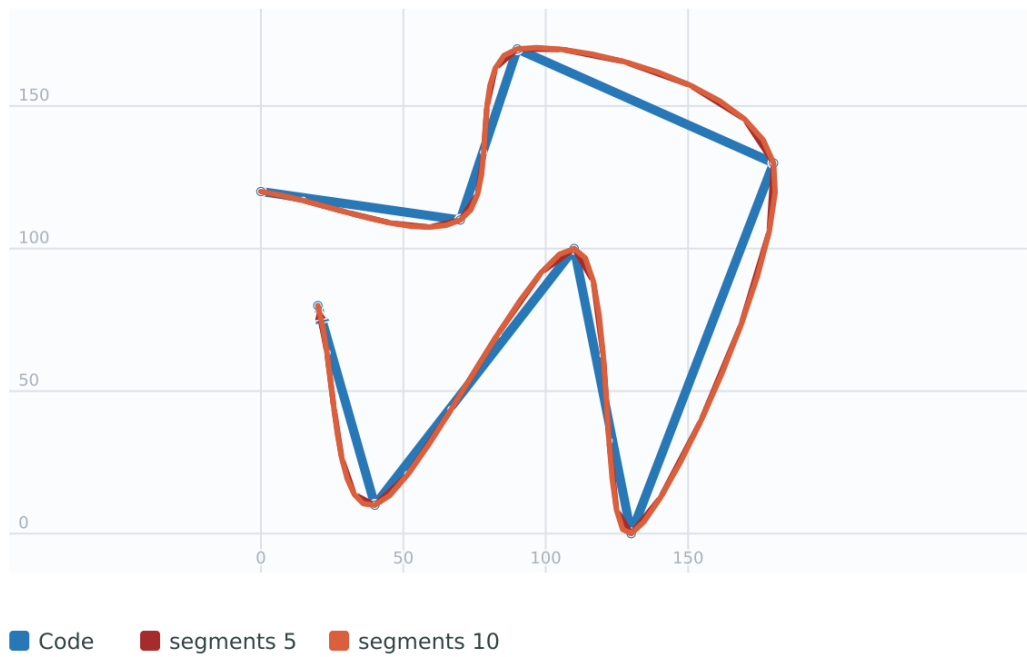
示例

Smoothing a LineString using `nSegments = 5` and 10.

Code

```
WITH input(geom) AS (VALUES
  ('LINESTRING (10 140,80 130,100 190,190 150,140 20,120 120,50 30,30 100)')::geometry)
SELECT ST_CatmullRomSmoothing(geom, 5) AS segments_5,
       ST_CatmullRomSmoothing(geom, 10) AS segments_10
FROM input;
```

Figure



Smoothing a 4-point collinear line with default nSegments=5 shows the minimal change case.

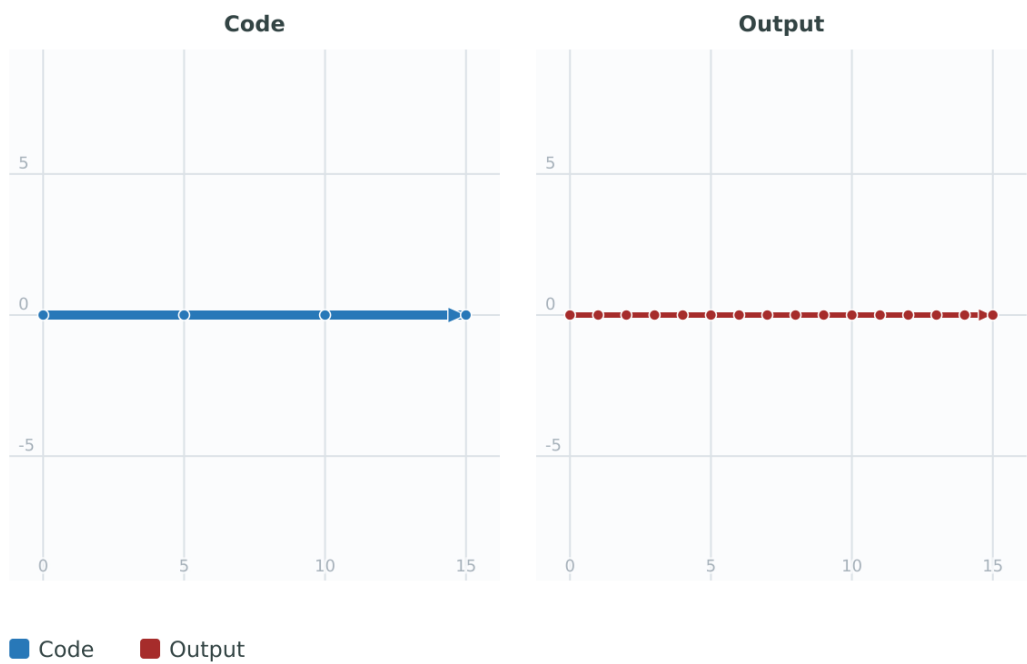
Code

```
SELECT ST_CatmullRomSmoothing('LINESTRING(0 0,5 0,10 0,15 0)');
```

Output

```
LINESTRING(0 0,1 0,2 0,3 0,4 0,5 0,6 0,7 0,8 0,9 0,10 0,11 0,12 0,13 0,14 0,15 0)
```

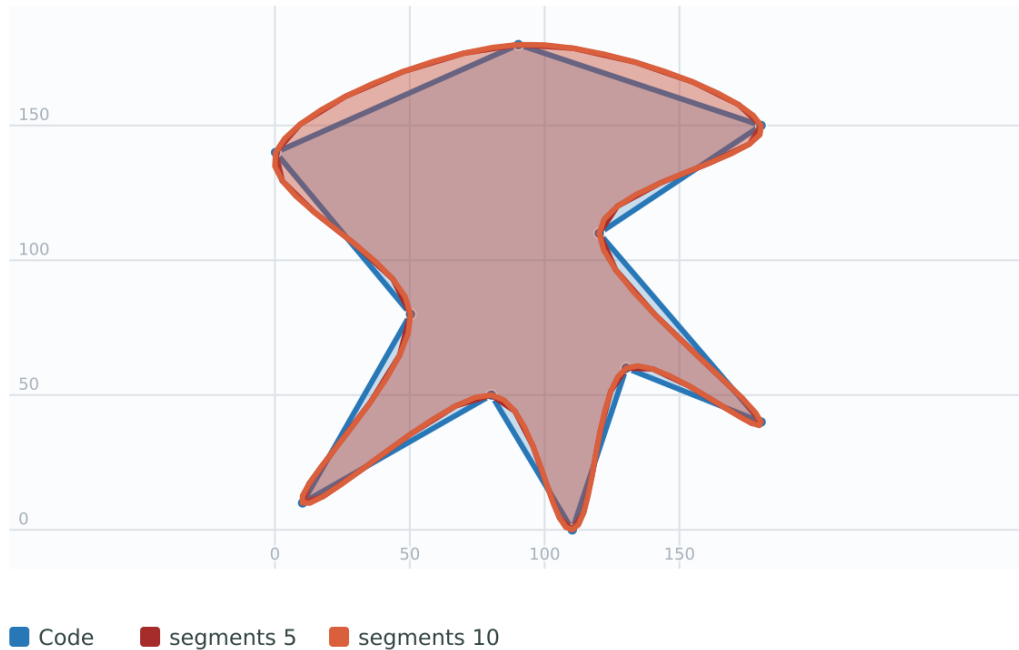
Figure



Smoothing a Polygon using nSegments = 5 and 10. This larger example is kept last.

Code

```
WITH input(geom) AS (VALUES
  ('POLYGON ((20 20,60 90,10 150,100 190,190 160,130 120,190 50,140 70,120 10,90 60,20 20))' ::geometry)
)
SELECT ST_CatmullRomSmoothing(geom, 5) AS segments_5,
       ST_CatmullRomSmoothing(geom, 10) AS segments_10
FROM input;
```

Figure

相关信息

[ST_ChaikinSmoothing](#), [ST_Simplify](#), [ST_SimplifyPreserveTopology](#), [ST_SimplifyVW](#)

7.14.6 ST_ConcaveHull

`ST_ConcaveHull` — 计算包含所有输入几何点的可能凹几何

Synopsis

geometry **ST_ConcaveHull**(geometry param_geom, float param_pctconvex, boolean param_allow_holes = false);

描述

凹壳是（通常）凹几何体，包含输入，其点是输入点的子集。一般情况下，凹壳是多边形。个或多个共点的凹包是点串。一个或多个相同点的凹壳是一个点。除非可选项 `param_allow_holes` 参数指定 `true`，否则多边形将不包含孔。

凹壳可以被看作是“真空包装”的几何形状的集合。它与凸壳不同，凸壳似乎是用橡皮筋将其封住。凹形的角通常面积极小，并且具有由输入点形成的自然边界。

`param_pctconvex` 控制计算的外壳的凹度。1 会生成凸包。1 到 0 之间的值会生成凹度逐渐增加的外壳。0 会生成具有最大凹度的外壳（但仍然是个多边形）。合适的值取决于输入数据的性质，但通常 0.3 到 0.1 之间的值会生成合理的结果。



Note

从技术上讲，`param_pctconvex` 将凹度确定输入点的 Delaunay 三角剖分中最窄和最短边的差的三分之一。小于凹度的值会被三角测量“入侵”。剩下的三角形形成凹形外壳。

对于点和性质输入，外壳将包含输入的所有点。对于多边形输入，外壳将包含输入的所有点以及输入覆盖的所有区域。如果您想要多边形的逐点外壳，您首先使用 `ST_Points` 将其转换为点。

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)`).

可用性: 2.0.0

增强: 3.3.0, 使用 GEOS 3.11+ 使用 GEOS 本机函数

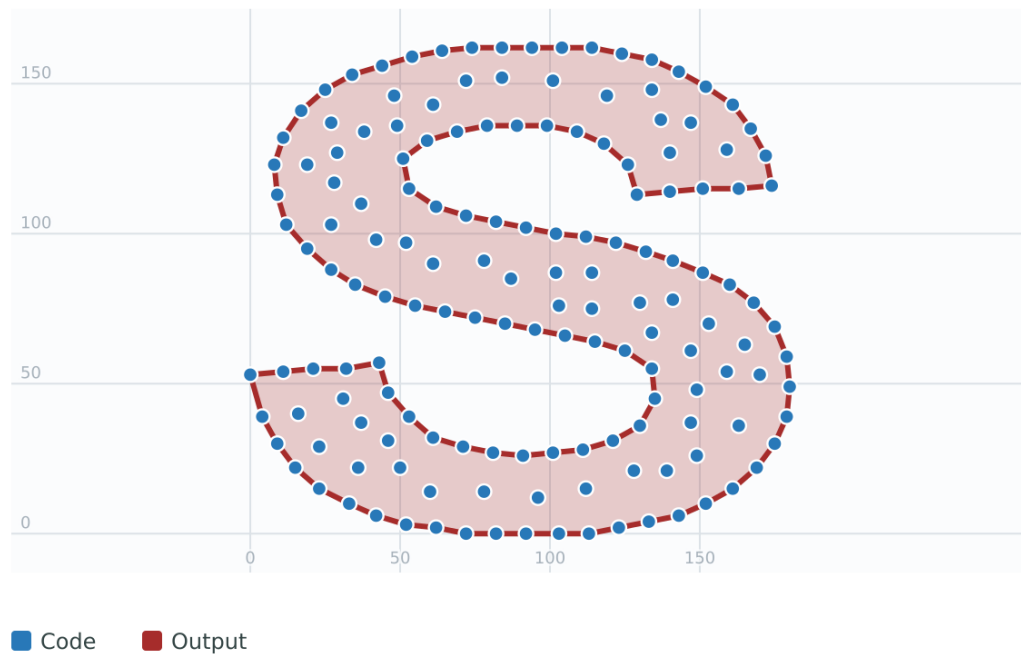
示例

Concave hull of a MultiPoint.

Code

```
SELECT 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 );
```

Figure

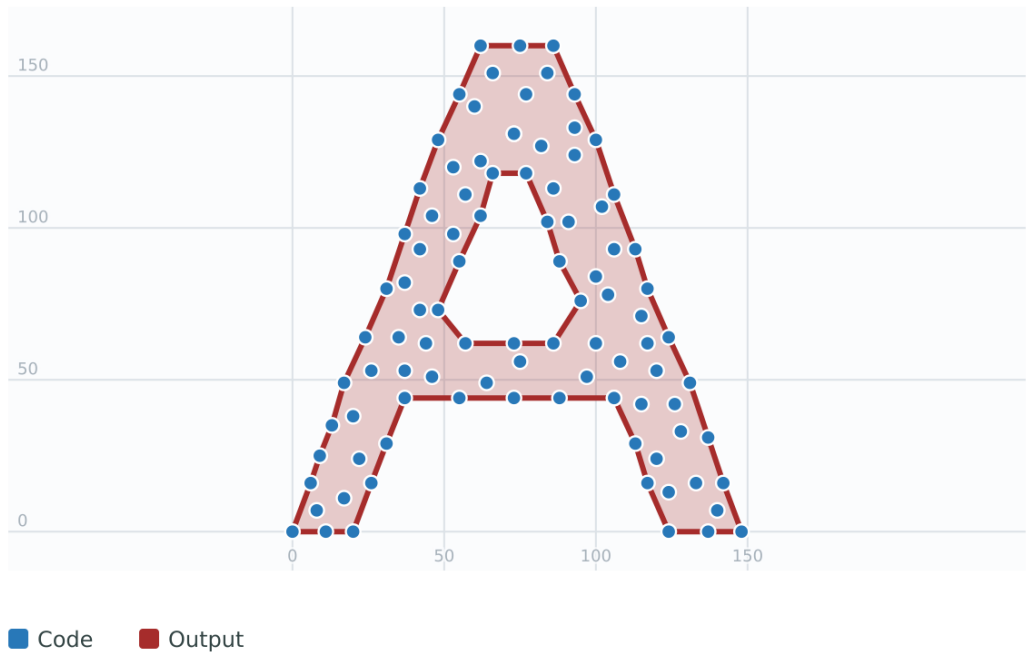


Concave hull of a MultiPoint, allowing holes.

Code

```
SELECT 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 );
```

Figure



将多边形形的凹壳与点集的凹壳进行比较。外壳遵循多边形形的边界，而基于点的外壳则不然。

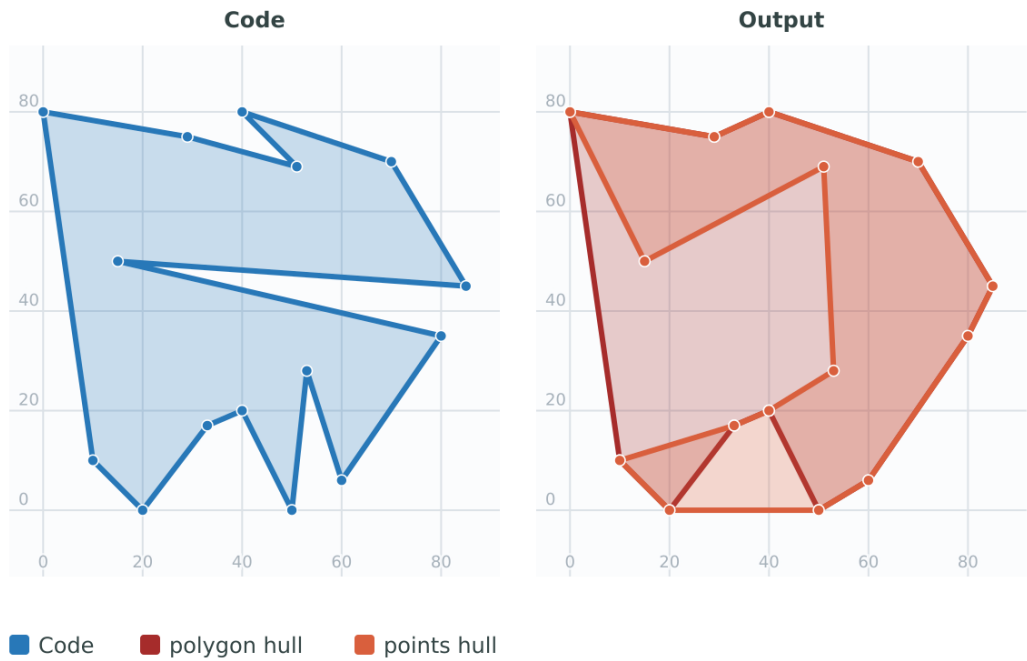
Code

```
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;
```

Output

```
POLYGON((60 10,50 30,43 27,30 10,20 20,10 90,39 85,50 90,80 80,95 55,90 45,70 16,60 10)) | POLYGON((25 60,10 90,39
85,50 90,80 80,95 55,90 45,70 16,60 10,30 10,20 20,43 27,50 30,63 38,61 79,25 60))
```

Figure



与 `ST_Collect` 一起使用来估算几何图形集的凹壳。
This example estimates an infected area from point observations.

Code

```
SELECT disease_type,  
       ST_ConcaveHull(ST_Collect(obs_pnt), 0.3 ) AS geom  
FROM disease_obs  
GROUP BY disease_type;
```

相关信息

[ST_ConvexHull](#), [ST_Collect](#), [CG_AlphaShape](#), [CG_OptimalAlphaShape](#)

7.14.7 ST_ConvexHull

`ST_ConvexHull` — 估算几何体的凸包。

Synopsis

geometry **ST_ConvexHull**(geometry geomA);

描述

估算几何图形的凸包。凸包是包含所有输入几何的最小凸几何体。
人可以将凸包通过将橡皮筋套在一几何图形上而得的几何图形。与凹形外壳不同，凹形外壳似于“收缩包装”几何形状。凸包通常用于根据一点来确定受影的区域。

一般情况下，凸包是多边形。两个或多个共线点的凸包是点串。一个或多个相同点的凸包是一个点。
这不是聚合函数。要计算一个几何图形的凸包，使用 **ST_Collect** 将它聚合到几何集合中(例如 **ST_ConvexHull(ST_Collect(...))**)。它是通过 GEOS 模块实现的。

- ✓ 此方法实现了 **SQL 1.1 的 OGC 功能规范**。 s2.1.1.3
- ✓ 该方法实现了 **SQL/MM 规范**。 SQL-MM IEC 13249-3: 5.1.16
- ✓ 该函数支持 3d 并且不会丢失 z-index。

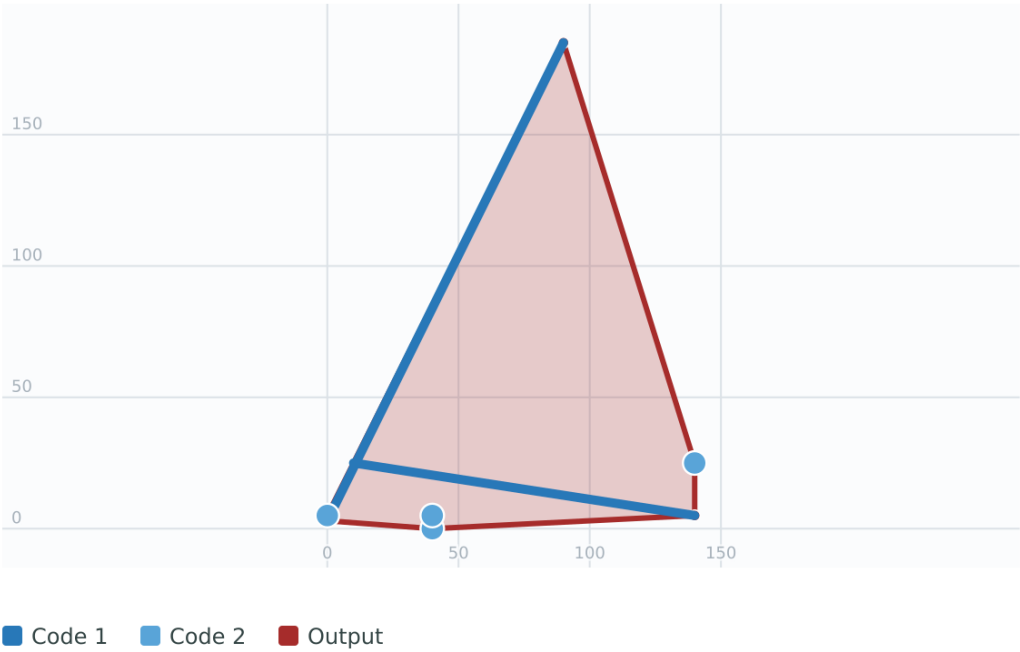
示例

Compute the convex hull of a MultiLineString and a MultiPoint.

Code

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

Figure



与 **ST_Collect** 一起使用来计算几何图形集的凸包。
This example returns an estimated infected area from point observations.

Code

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


相关信息

[ST_Collect](#), [ST_ConcaveHull](#), [ST_MinimumBoundingCircle](#)

7.14.8 ST_DelaunayTriangles

ST_DelaunayTriangles — 返回几何体点的 Delaunay 三角剖分。

Synopsis

geometry **ST_DelaunayTriangles**(geometry g1, float tolerance = 0.0, int4 flags = 0);

描述

计算输入几何体点的 **Delaunay 三角剖分**。可设置的容差可用于将附近的输入点捕捉在一起，在某些情况下提高了健壮性。如果几何体形以输入点的凸包为边界。如果几何体表示由标志代确定：

- 0 - 三角形 POLYGON 的 GEOMETRYCOLLECTION（默认）
- 1 - 三角量的多面体
- 2 - 三角量的 TIN

这个函数是由 GEOS 模块执行的。

可用性：2.1.0



该函数支持 3d 并且不会丢失 z-index。



此函数支持三角形和不规则三角网面 (TIN)。

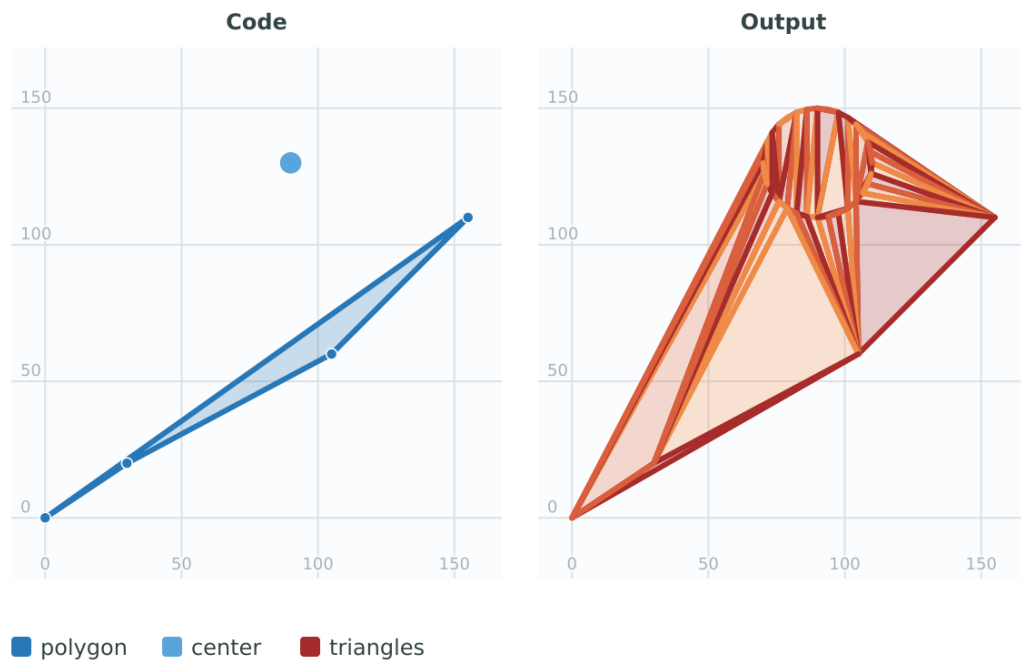
示例

Triangulation of the vertices of a polygon and a buffered point.

Code

```
WITH data AS (  
  SELECT polygon, ST_Buffer(center, 20) AS point_buffer  
  FROM (SELECT  
    'POLYGON((175 150,20 40,50 60,125 100,175 150))'::geometry AS polygon,  
    'POINT(110 170)'::geometry AS center  
  ) AS input  
)  
SELECT ST_DelaunayTriangles(ST_Union(polygon, point_buffer)) AS triangles  
FROM data;
```

Figure

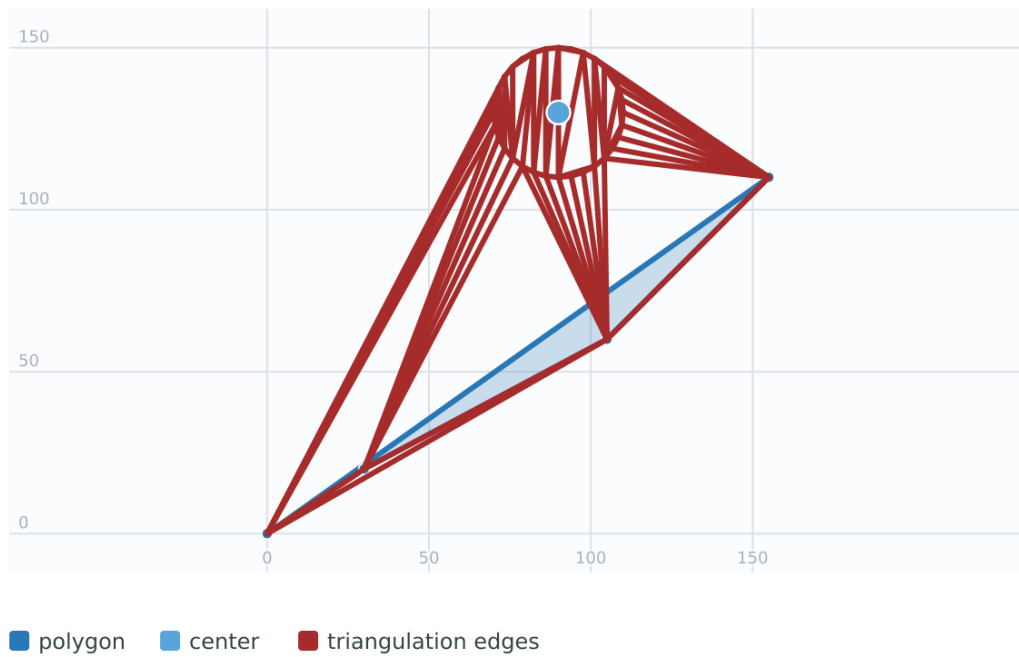


Triangulation edges returned as a MULTILINESTRING.

Code

```
WITH data AS (
  SELECT polygon, ST_Buffer(center, 20) AS point_buffer
  FROM (SELECT
    'POLYGON((175 150,20 40,50 60,125 100,175 150))'::geometry AS polygon,
    'POINT(110 170)'::geometry AS center
  ) AS input
)
SELECT ST_DelaunayTriangles(ST_Union(polygon, point_buffer), 0.001, 1) AS ↵
  triangulation_edges
FROM data;
```

Figure

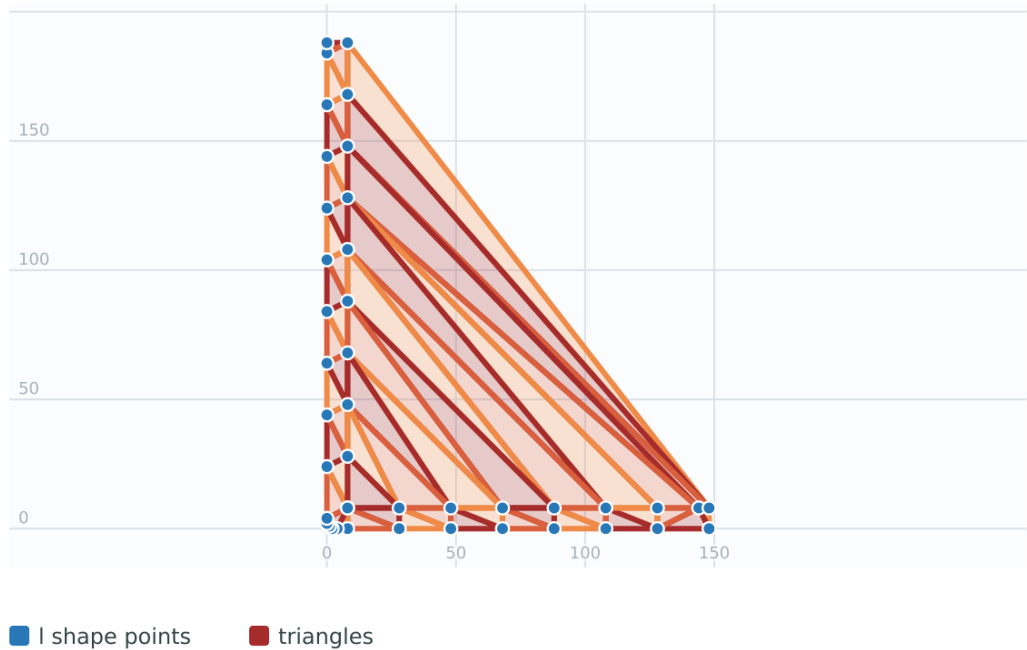


Triangulation of 42 points forming an L shape.

Code

```
WITH data AS (
  SELECT '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)'::geometry AS l_shape_points
)
SELECT ST_DelaunayTriangles(l_shape_points) AS triangles
FROM data;
```

Figure



使用具有 Z 值的点的示例。

Code

```
SELECT ST_DelaunayTriangles(  
  'MULTIPOINT Z(14 14 10,150 14 100,34 6 25,20 10 150)':geometry);
```

相关信息

[ST_VoronoiPolygons](#), [ST_TriangulatePolygon](#), [CG_ConstrainedDelaunayTriangles](#), [ST_VoronoiLines](#), [ST_ConvexHull](#)

7.14.9 ST_FilterByM

ST_FilterByM — 根据 M 值删除点

Synopsis

geometry **ST_FilterByM**(geometry geom, double precision min, double precision max = null, boolean returnM = false);

描述

根据 M 值删除点。返回包含 M 值大于或等于最小值且小于或等于最大值的点的几何图形。如果省略最大值参数，则考虑最小值。如果省略第四个参数，则 m 值将不会出现在生成的几何图形中。如果生成的几何图形其几何类型留下的点太少，将返回空几何图形。在几何集合中，没有足够点的几何图形将被默默地排除。
函数主要与 ST_SetEffectiveArea 合使用。ST_EffectiveArea 用其 m 值置点的有效区域。使用 ST_FilterByM，无需任何计算，只需即可得几何的化版本



Note
与 ST_FilterByM 相比,当没有足够的点满足条件时,ST_SimplifyVW 返回的结果有所不同。ST_SimplifyVW 返回具有足够点的几何图形,而 ST_FilterByM 返回空几何图形



Note
注意,返回的几何图形可能无效



Note
函数返回所有维度,包括 Z 和 M

可用性 : 2.5.0

示例

串被

Code

```
SELECT ST_FilterByM(geom, 30) simplified
FROM (SELECT ST_SetEffectiveArea('LINESTRING(5 2,3 8,6 20,7 25,10 10)')::geometry) geom) As
foo;
```

Output

LINESTRING(5 2,7 25,10 10)

Figure



相关信息

[ST_SetEffectiveArea](#), [ST_SimplifyVW](#)

7.14.10 ST_GeneratePoints

ST_GeneratePoints — 生成一个包含在多边形（Polygon）或多重多边形（MultiPolygon）内的随机点的多点对象。

Synopsis

geometry **ST_GeneratePoints**(geometry g, integer npoints, integer seed = 0);

描述

ST_GeneratePoints 生成一个包含指定数量随机点的多点对象，这些点位于输入区域内。可选的 **seed** 用于重新生成一个确定性的点序列，必须大于零。

可用性：2.3.0

增加：3.0.0，添加种子参数

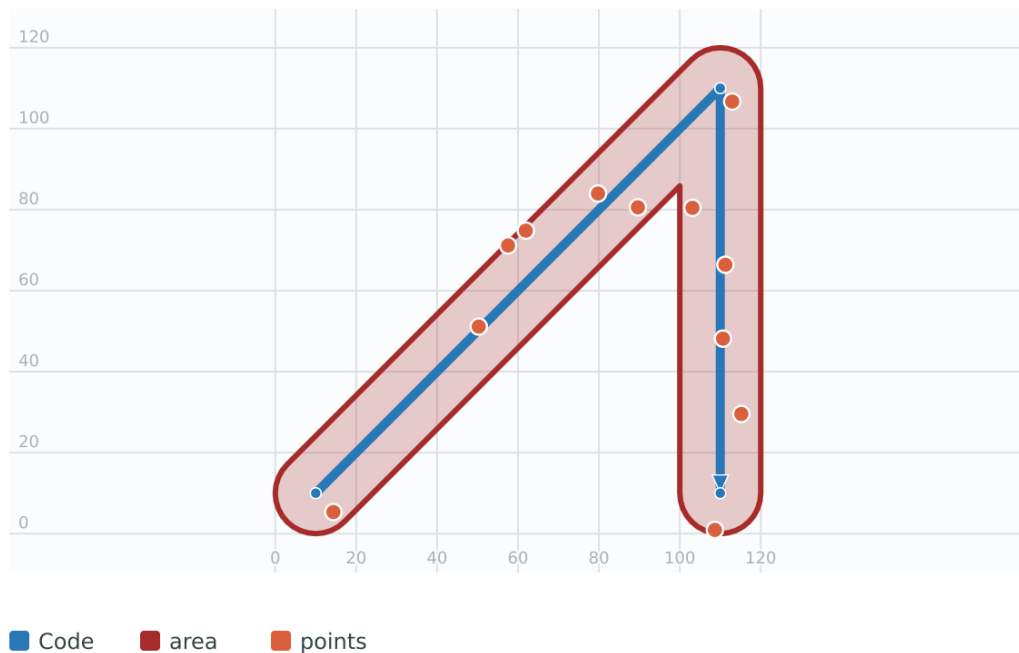
示例

Generate 12 points inside a buffered LineString using the deterministic seed 1996.

Code

```
WITH source AS (  
    SELECT ST_Buffer(ST_GeomFromText('LINESTRING(50 50,150 150,150 50)'),  
        10, 'endcap=round join=round') AS geom  
)  
SELECT geom AS area,  
       ST_GeneratePoints(geom, 12, 1996) AS points  
FROM source;
```

Figure



给定一个多边形表 s，返回每个多边形 12 个独立的点。每次行结果都会不同。

Code

```
SELECT s.id, dp.path[1] AS pt_id, dp.geom
FROM s, ST_DumpPoints(ST_GeneratePoints(s.geom, 12)) AS dp;
```

Generate dot-density points from a polygon table, using the absolute value of an attribute as the number of points and keeping the sign of the attribute for styling.

Code

```
SELECT z.id,
       CASE WHEN z.count_delta < 0 THEN -1 ELSE 1 END AS sign,
       dp.path[1] AS pt_id,
       dp.geom
FROM zones AS z
CROSS JOIN LATERAL ST_DumpPoints(
    ST_GeneratePoints(z.geom, abs(z.count_delta)::integer, z.id + 1)
) AS dp
WHERE z.count_delta <> 0;
```

相关信息

[ST_DumpPoints](#)

7.14.11 ST_GeometricMedian

ST_GeometricMedian — 返回多点的几何中位数。

Synopsis

geometry **ST_GeometricMedian** (geometry geom, float8 tolerance = NULL, int max_iter = 10000, boolean fail_if_not_converged = false);

描述

使用 Weiszfeld 算法计算多点几何的近似几何中点。几何中位数是最小化到输入点的距离之和的点。它提供了一种中心性度量，与重心（重心）相比，它对异常点不太敏感。

该算法进行迭代，直到每次迭代之间的距离变化小于提供的容差参数。如果在 `max_iterations` 次迭代后仍未满足此条件，该函数会生成警告并退出，除非 `fail_if_not_converged` 置为 `false`（默认值）。

如果未提供容差参数，将根据输入几何图形的范围计算公差。

如果存在，输入点 `M` 将被解为其自身的相重重。

可用性：2.3.0

增强：2.5.0 添加了 `M` 作点重重的支持。



该函数支持 3d 并且不会丢失 z-index。



该功能支持 `M` 坐。

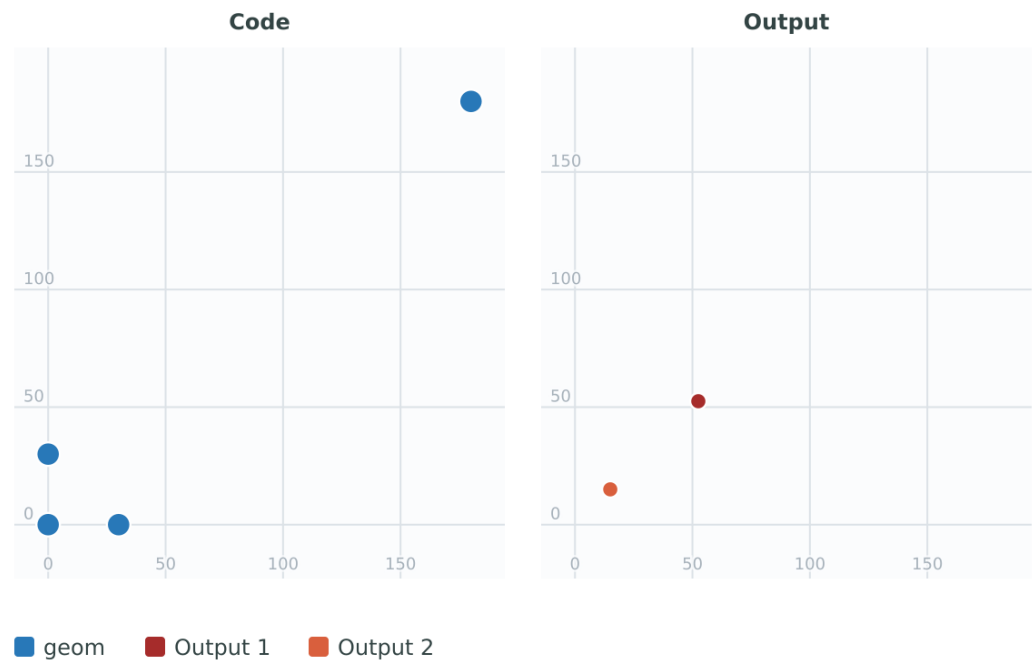
示例

Compare the geometric median and centroid of a MultiPoint.

Code

```
WITH test AS (
SELECT 'MULTIPOINT((10 10),(10 40),(40 10),(190 190))'::geometry geom)
SELECT
  ST_Centroid(geom) centroid,
  ST_GeometricMedian(geom) median
FROM test;
```


Figure



相关信息

ST_Centroid

7.14.12 ST_LineMerge

ST_LineMerge — 返回通将 MultiLineString 合在一起形成的。

Synopsis

```
geometry ST_LineMerge(geometry amultilinestring);
geometry ST_LineMerge(geometry amultilinestring, boolean directed);
```

描述

返回通将 MultiLineString 的元素接在一起而形成的 LineString 或 MultiLineString。 在 2 交叉点的端点接。 不会在三向或更高次的交叉点上接。

如果定向 TRUE， ST_LineMerge 将不会更改 LineStrings 内的点序，因此方向相反的不会被合并



Note
与 MultiLineString/LineString 一起使用。其他几何型返回空的 GeometryCollection

个函数是由 GEOS 模行的。

增：3.3.0 接受定向参数。

需要 GEOS $\geq 3.11.0$ 才能使用定向参数。
可用性 : 1.1.0



Warning
此函数剥离 M 维度。

示例

次数 **2**

次数 2 的交点会被合并移除。

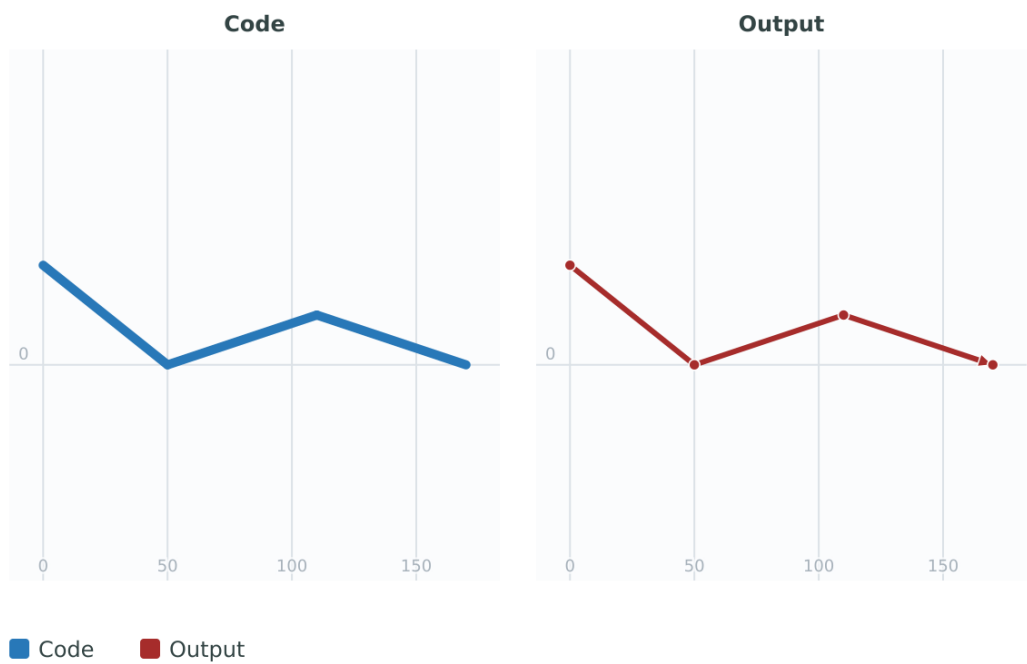
Code

```
SELECT ST_LineMerge('MULTILINESTRING((10 160,60 120),(120 140,60 120),(120 140,180 120))')
);
```

Output

```
LINESTRING(10 160,60 120,120 140,180 120)
```

Figure



次数 **3** 以上

在交点的度数大于 2 时，线段不会合并。

Code

```
SELECT ST_LineMerge('MULTILINESTRING((10 160,60 120),(120 140,60 120),(120 140,180 120) ↵
, (100 180,120 140))'
);
```

Output

```
MULTILINESTRING((10 160,60 120,120 140),(100 180,120 140),(120 140,180 120))
```

Figure

不相交的☒

如果由于不接触☒而无法合并，☒返回原始 MultiLineString。

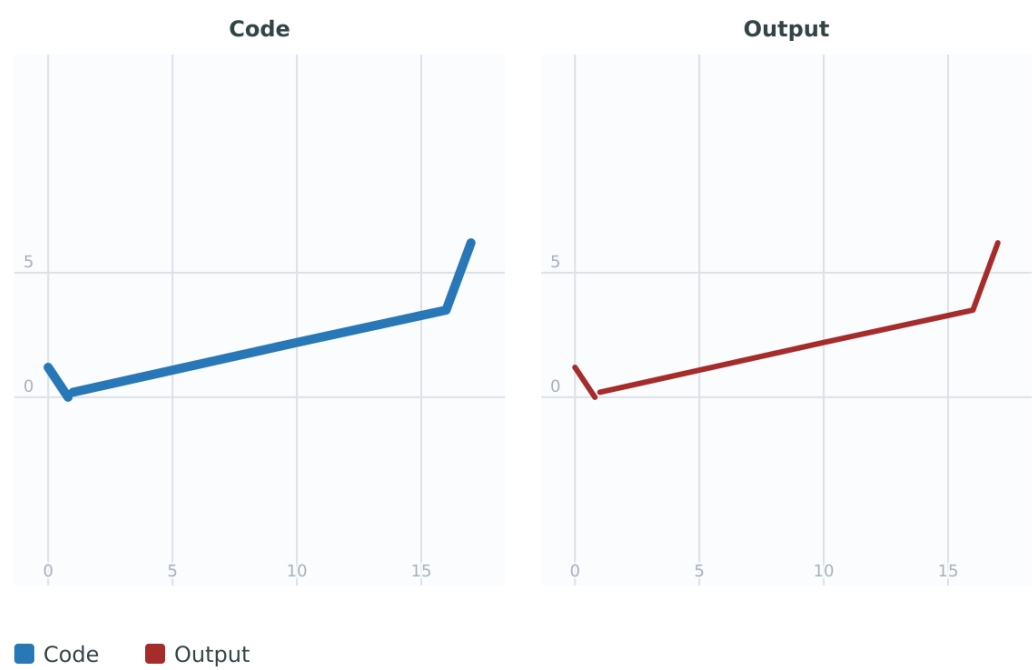
Code

```
SELECT ST_LineMerge('MULTILINESTRING((-29 -27,-30 -29.7,-36 -31,-45 -33),(-45.2 -33.2,-46 ↵
-32))'
);
```

Output

```
MULTILINESTRING((-45.2 -33.2,-46 -32),(-29 -27,-30 -29.7,-36 -31,-45 -33))
```

Figure



有向参数

Lines with opposite directions are not merged if *directed* is TRUE.

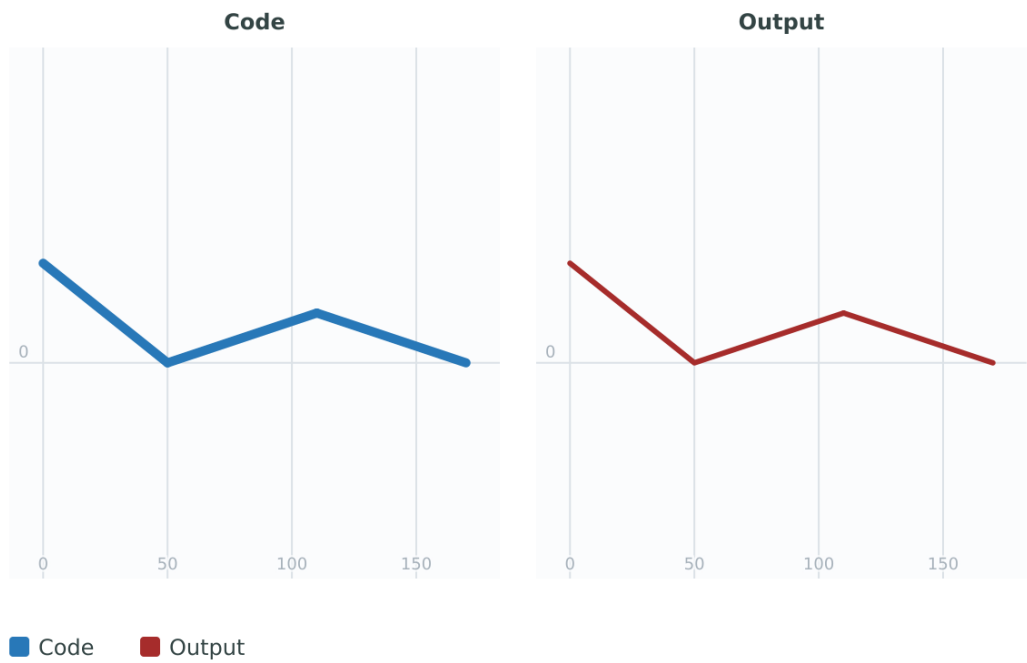
Code

```
SELECT ST_LineMerge( 'MULTILINESTRING((60 30,10 70),(120 50,60 30),(120 50,180 30))',  
TRUE);
```

Output

```
MULTILINESTRING((120 50,60 30,10 70),(120 50,180 30))
```

Figure



Z 度理

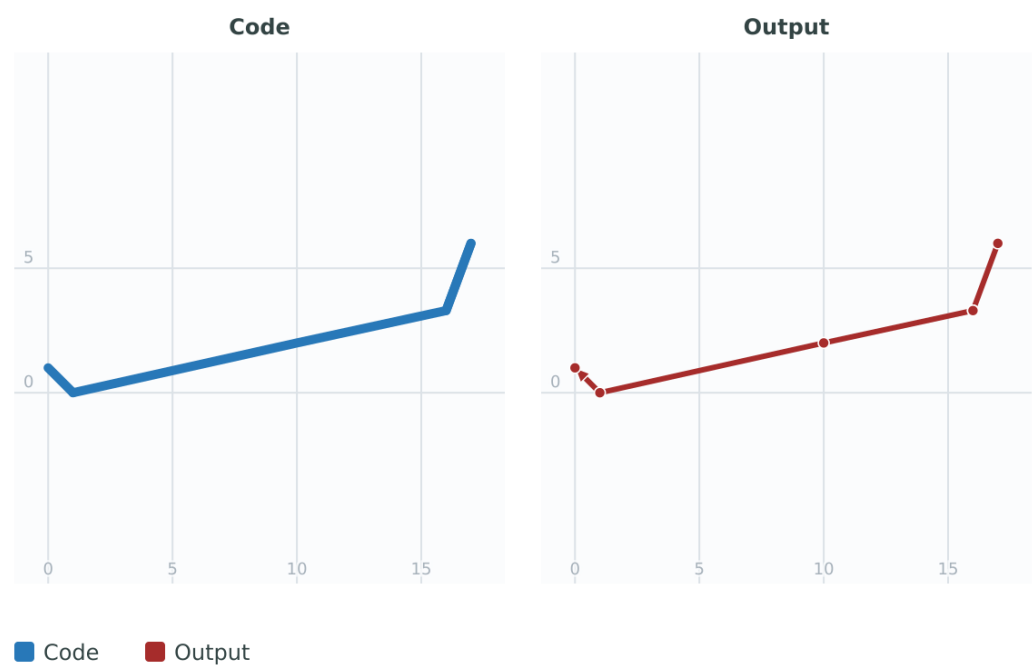
Code

```
SELECT ST_LineMerge('MULTILINESTRING((-29 -27 11,-30 -29.7 10,-36 -31 5,-45 -33 6),(-29 -27 ↵
    12,-30 -29.7 5),(-45 -33 1,-46 -32 11))'
);
```

Output

```
LINESTRING Z (-30 -29.7 5,-29 -27 11,-30 -29.7 10,-36 -31 5,-45 -33 1,-46 -32 11)
```

Figure



相关信息

[ST_Segmentize](#), [ST_LineSubstring](#)

7.14.13 ST_MaximumInscribedCircle

ST_MaximumInscribedCircle — 计算几何体中包含的最大圆。

Synopsis

(geometry, geometry, double precision) **ST_MaximumInscribedCircle**(geometry geom);

描述

查找（多）多边形内包含的最大圆，或者不与任何圆和点重叠的最大圆。返回包含字段的圆：

- center-圆的中心点
- nearest-几何上最接近中心的点
- radius-圆的半径

圆于多边形内，圆内接于边界内，使用内圆作边界。圆于点和点内，圆内切于点的凸包内，使用点和点作圆一步的边界。

可用性：3.1.0。

需要 GEOS >= 3.9.0。

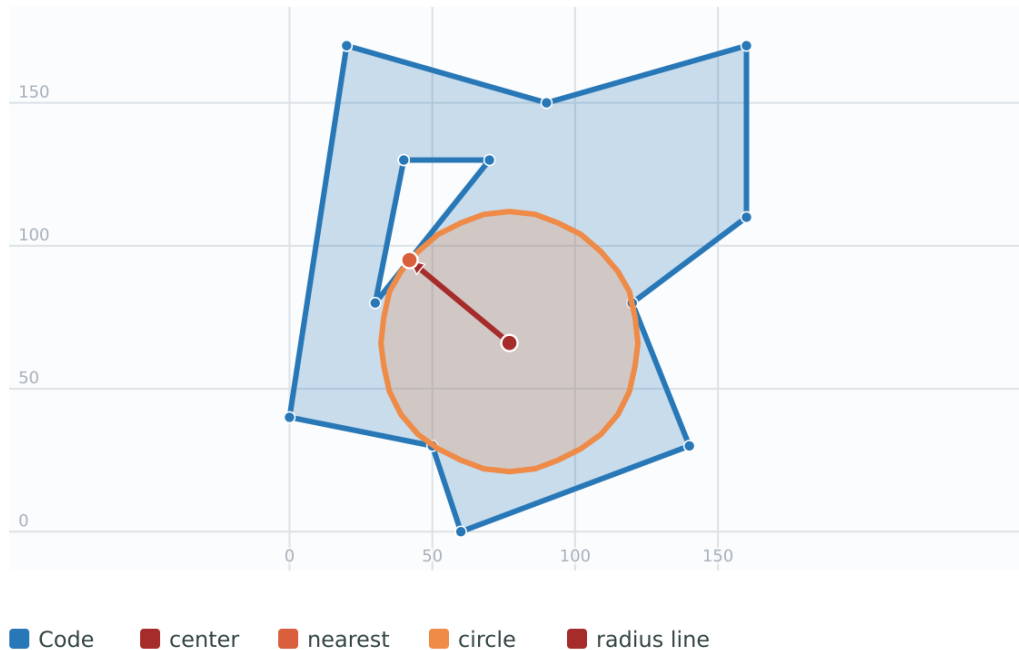
示例

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

Code

```
WITH mic AS (
  SELECT * FROM ST_MaximumInscribedCircle(
    'POLYGON ((40 180,110 160,180 180 120,140 90,160 40,80 10,70 40,20 50,40 180),
      (60 140,50 90,90 140,60 140))'
  )
)
SELECT round(radius::numeric, 3) AS radius,
  ST_SnapToGrid(center, 1) AS center,
  ST_SnapToGrid(nearest, 1) AS nearest,
  ST_SnapToGrid(ST_Buffer(center, radius), 1) AS circle,
  ST_SnapToGrid(ST_MakeLine(center, nearest), 1) AS radius_line
FROM mic;
```

Figure

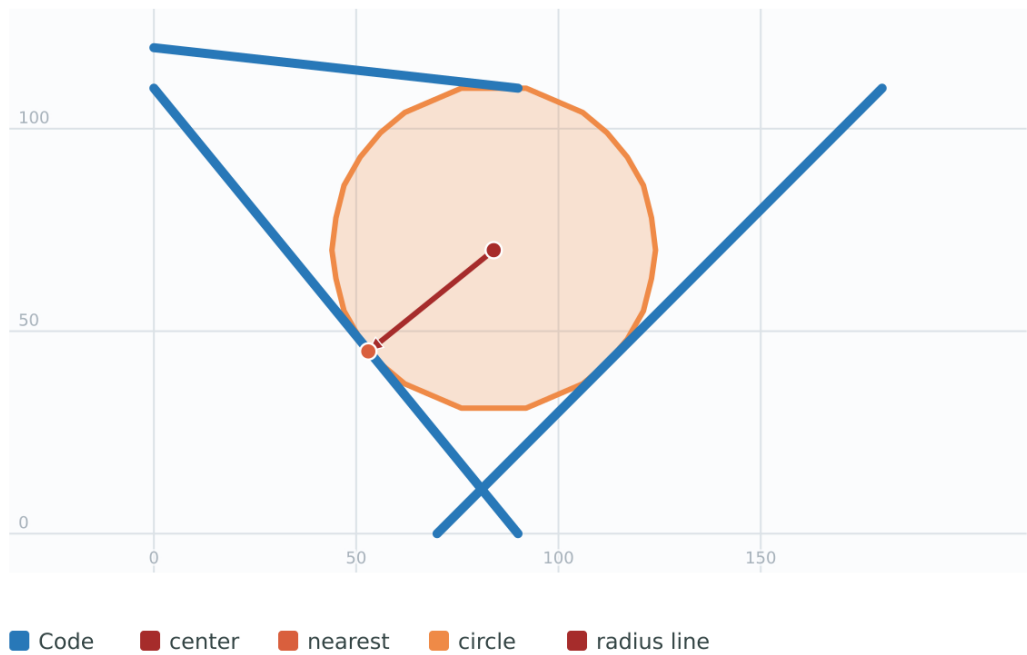


Maximum inscribed circle of a MultiLineString.

Code

```
WITH mic AS (
  SELECT * FROM ST_MaximumInscribedCircle(
    'MULTILINESTRING((100 40,10 150),(10 160,100 150),(190 150,80 40))'
  )
)
SELECT round(radius::numeric, 3) AS radius,
  ST_SnapToGrid(center, 1) AS center,
  ST_SnapToGrid(nearest, 1) AS nearest,
  ST_SnapToGrid(ST_Buffer(center, radius), 1) AS circle,
  ST_SnapToGrid(ST_MakeLine(center, nearest), 1) AS radius_line
FROM mic;
```

Figure



相关信息

[ST_MinimumBoundingRadius](#), [ST_LargestEmptyCircle](#)

7.14.14 ST_LargestEmptyCircle

ST_LargestEmptyCircle — 计算不与几何形状重叠的最大圆。

Synopsis

(geometry, geometry, double precision) **ST_LargestEmptyCircle**(geometry geom, double precision tolerance=0.0, geometry boundary=POINT EMPTY);

描述

找到不与一点和障碍物重叠的最大圆。（多边形几何形状可以作障碍物包含在内，但不使用它的边界。）圆的中心被限制位于多边形边界内，默认情况下，多边形边界是输入几何形状的凸包。圆心是边界内部距离障碍物最近的点。圆本身由中心点和位于确定圆半径的障碍物上的最近点提供。

If a boundary is supplied it must be a valid polygonal geometry with positive area.
使用迭代算法将圆心确定由距离容差指定的精度。如果未指定精度距离，使用合理的默认值。
返回包含字段的表：

- center-圆的中心点
- nearest-几何上最接近中心的点
- radius-圆的半径

要找到多边形内部最大的空圆，请参考 [ST_MaximumInscribedCircle](#)。

可用性：3.4.0。

需要 GEOS >= 3.9.0。

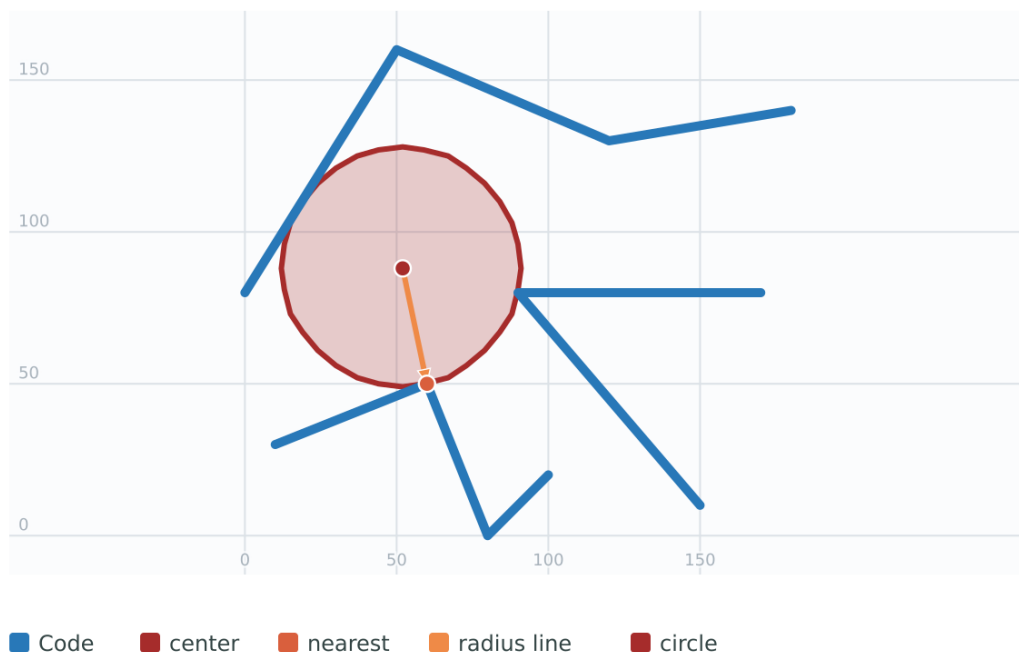
示例

Largest empty circle within a set of lines.

Code

```
WITH obstacles(geom) AS (VALUES ('MULTILINESTRING (
    (10 100,60 180,130 150,190 160),
    (20 50,70 90 20,110 40),
    (160 30,100 100,180 100))'::geometry)),
circle AS (
    SELECT result.*
    FROM obstacles
    CROSS JOIN LATERAL ST_LargestEmptyCircle(obstacles.geom) AS result
)
SELECT round(radius::numeric, 6) AS radius,
       ST_SnapToGrid(center, 1) AS center,
       ST_SnapToGrid(nearest, 1) AS nearest,
       ST_SnapToGrid(ST_MakeLine(center, nearest), 1) AS radius_line,
       ST_SnapToGrid(ST_Buffer(center, radius), 1) AS circle
FROM circle;
```

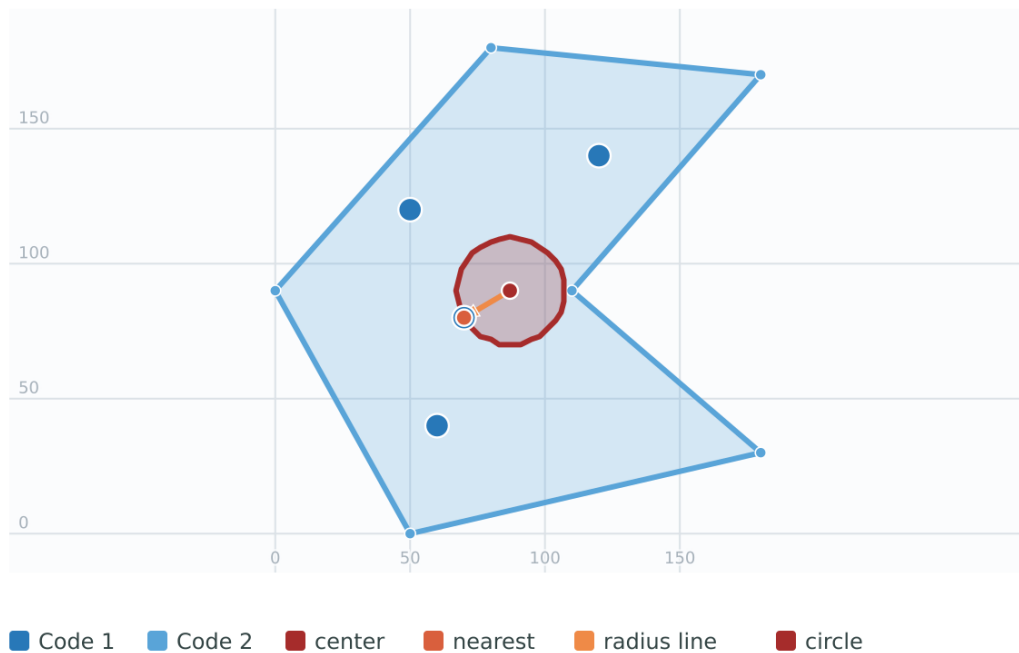
Figure



Largest empty circle within a set of points, constrained to lie in a polygon. The constraint polygon boundary is included both as an obstacle and as the constraint for the circle center.

Code

```
WITH inputs(points, boundary) AS (VALUES (
  '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
)),
circle AS (
  SELECT result.*
  FROM inputs
  CROSS JOIN LATERAL ST_LargestEmptyCircle(
    ST_Collect(points, boundary), 0, boundary
  ) AS result
)
SELECT round(radius::numeric, 6) AS radius,
       ST_SnapToGrid(center, 1) AS center,
       ST_SnapToGrid(nearest, 1) AS nearest,
       ST_SnapToGrid(ST_MakeLine(center, nearest), 1) AS radius_line,
       ST_SnapToGrid(ST_Buffer(center, radius), 1) AS circle
FROM circle;
```

Figure

相关信息

[ST_MinimumBoundingRadius](#)

7.14.15 ST_MinimumBoundingCircle

ST_MinimumBoundingCircle — 返回包含几何图形的最小圆形多边形。

Synopsis

geometry **ST_MinimumBoundingCircle**(geometry geomA, integer num_segs_per_qt_circ=48);

描述

返回包含几何图形的最小圆形多边形。



Note 边界近似多边形，默认情况下每四分之一圆有 48 个圆段。由于多边形是最小外接圆的近似圆，因此圆入几何中的某些点可能不包含在多边形内。可以通过增加分段数量来改圆近似圆。圆于近似圆不合适的圆用，可以使用 **ST_MinimumBoundingRadius**。

与**ST_Collect** 一起使用以圆得一圆几何图形的最小圆边界圆。
要圆算位于最小圆（“最大直径”）上的圆个点，圆使用**ST_LongestLine**。
多圆形的面圆除以最小包含圆的面圆之比称圆 **REOCK** 圆致度得分。
圆个函数是由 GEOS 模圆圆行的。
可用性：1.4.0

示例

Code

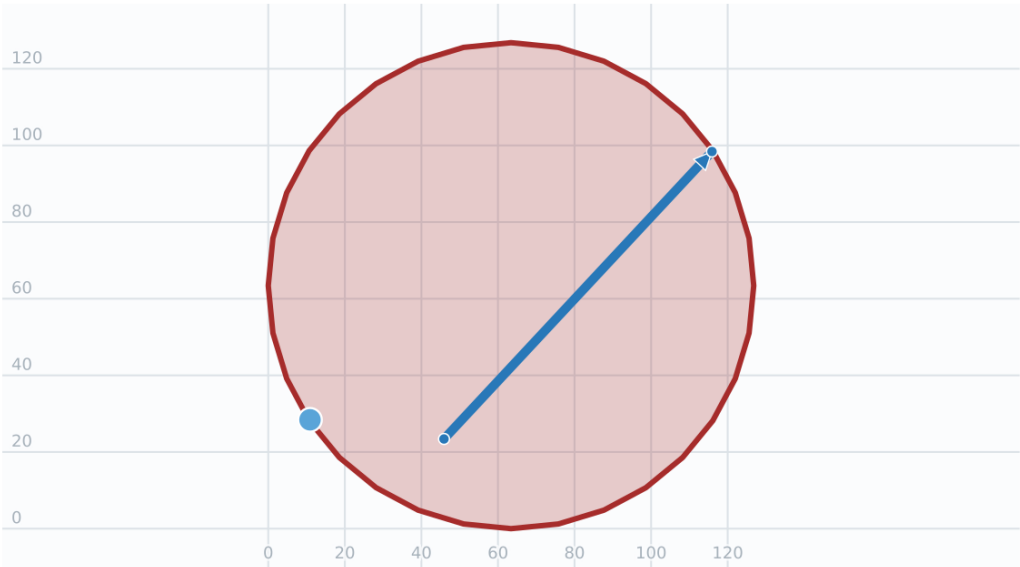
```
SELECT d.disease_type,  
ST_MinimumBoundingCircle(ST_Collect(d.geom)) As geom  
FROM disease_obs As d  
GROUP BY d.disease_type;
```

Minimum bounding circle of a point and linestring, using 8 segments to approximate a quarter circle.

Code

```
SELECT ST_MinimumBoundingCircle(ST_Collect('LINESTRING(55 75,125 150)',  
ST_Point(20, 80)), 8  
) As wktmlbc;
```

Figure



Code Point wktmlbc

相关信息

[ST_Collect](#), [ST_MinimumBoundingRadius](#), [ST_LargestEmptyCircle](#), [ST_LongestLine](#)

7.14.16 ST_MinimumBoundingRadius

`ST_MinimumBoundingRadius` — 返回包含几何图形的最小圆的中心点和半径。

Synopsis

(geometry, double precision) **ST_MinimumBoundingRadius**(geometry geom);

描述

计算包含几何图形的最小圆的中心点和半径。返回包含字段的元组：

- `center`-圆的中心点
- `radius`-圆的半径

与[ST_Collect](#)一起使用以得到一个几何图形的最小圆边界。

要计算位于最小圆（“最大直径”）上的点，使用[ST_LongestLine](#)。

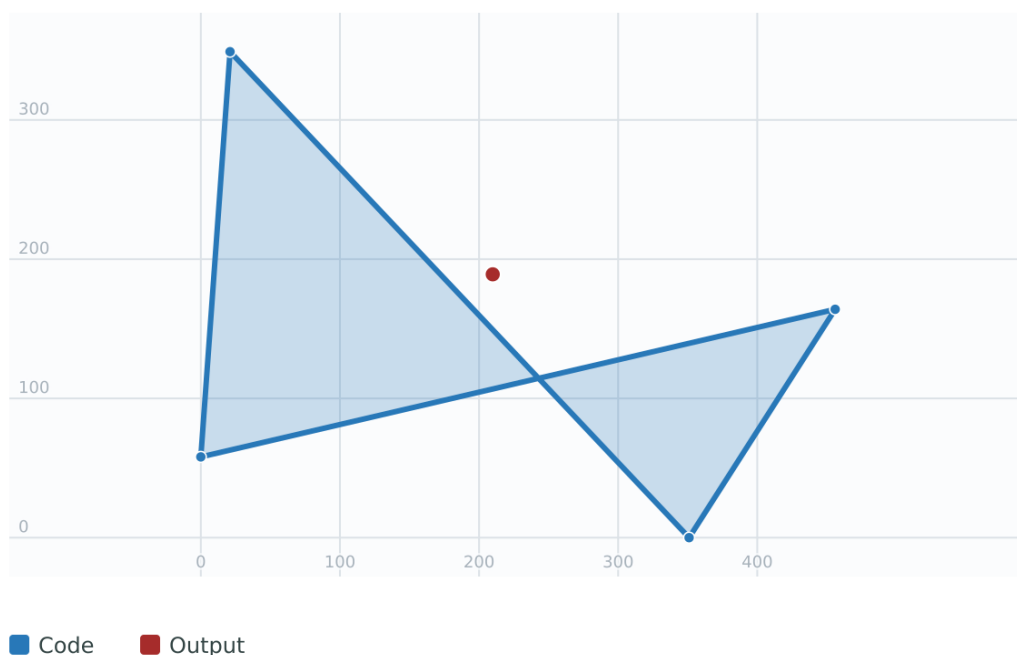
可用性 - 2.3.0

示例

Code

```
SELECT center, radius FROM ST_MinimumBoundingRadius('POLYGON((26426 65078,26531 65242,26075 65136,26096 65427,26426 65078))');
```

Figure



相关信息

[ST_Collect](#), [ST_MinimumBoundingCircle](#), [ST_LongestLine](#)

7.14.17 ST_OrientedEnvelope

ST_OrientedEnvelope — 返回包含几何图形的最小面积矩形。

Synopsis

geometry **ST_OrientedEnvelope**(geometry geom);

描述

返回包围几何体的最小面积旋转矩形。注意，可能存在多个这样的矩形。在退化输入的情况下可能返回 Point 或 LineString。

可用性：2.5.0。

需要 GEOS >= 3.6.0。

示例

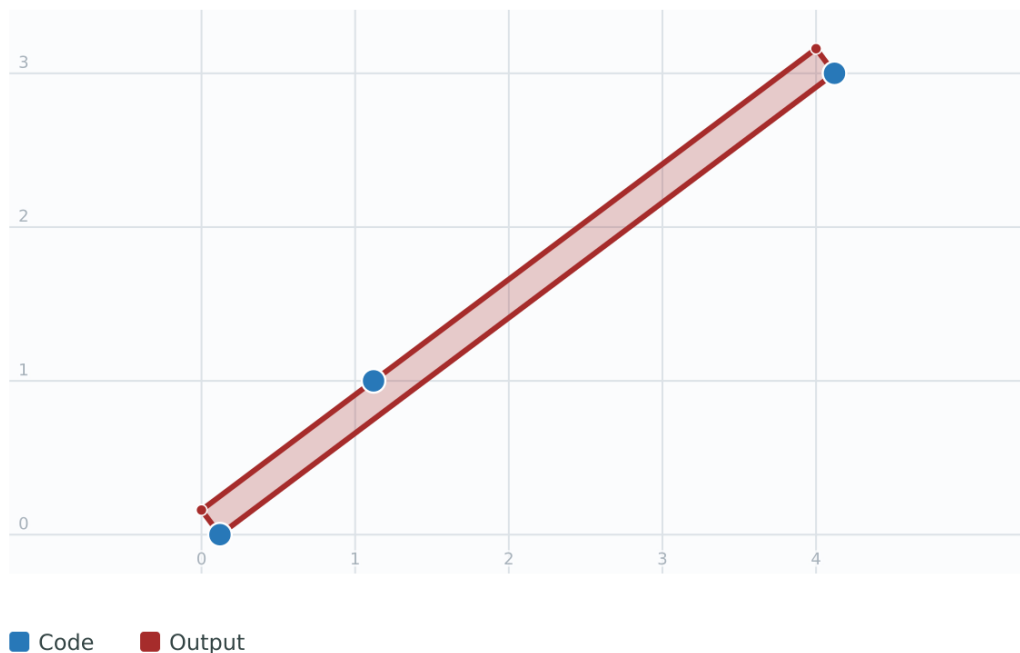
Code

```
SELECT ST_OrientedEnvelope('MULTIPOINT ((0 0),(-1 -1),(3 2))');
```

Output

```
POLYGON((3 2,-1 -1,-1.12 -0.84,2.88 2.16,3 2))
```

Figure



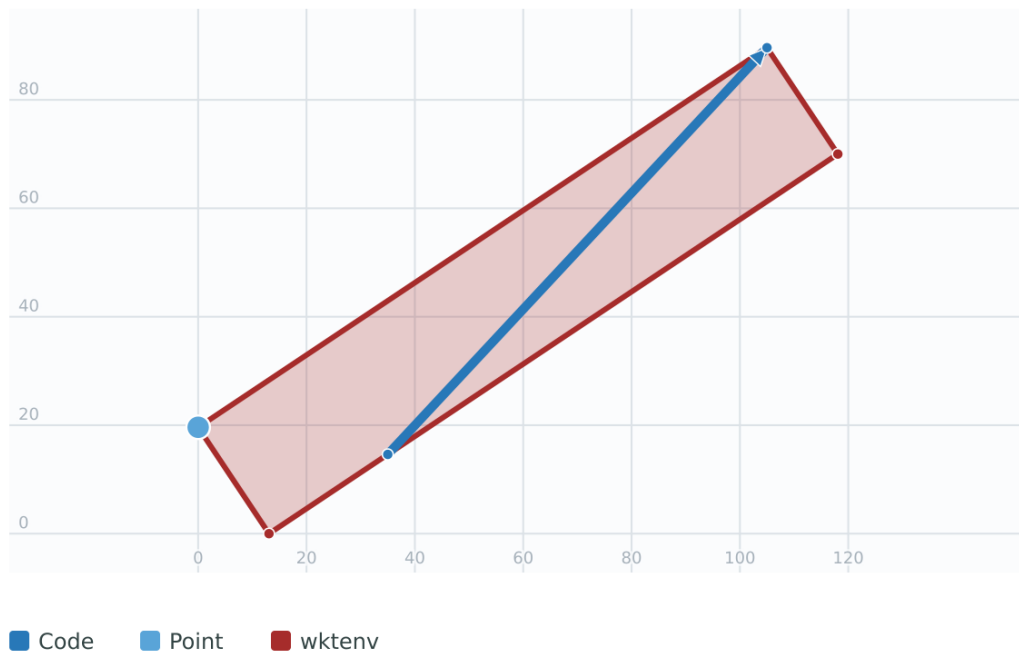
Oriented envelope of a Point and LineString.

Code

```
SELECT ST_OrientedEnvelope(ST_Collect('LINESTRING(55 75,125 150)',
                                     ST_Point(20, 80))
                           ) As wktenv;
```

Output

```
POLYGON((20 80,125 150,138.0769230769241 130.38461538461573,33.07692307692291 60.384615384615195,20 80))
```

Figure

相关信息

[ST_Envelope](#) [ST_MinimumBoundingCircle](#)

7.14.18 ST_OffsetCurve

ST_OffsetCurve — 返回距输入线指定距离和方向的偏移线。

Synopsis

```
geometry ST_OffsetCurve(geometry line, float signed_distance, text style_parameters=“);
```

描述

返回距输入线指定距离和方向的偏移线。返回几何图形的所有点与输入几何图形的距离不超过指定距离。对于算中心线的平行线很有用。

对于正距离，偏移位于输入的左并保持相同的方向。对于负距离，它位于右且方向相反。

距离单位以空参考系的单位来量。

注意，对于某些拼形状的输入几何图形，输出可能是 MULTILINESTRING 或 EMPTY。

可的第三个参数允许指定空白分隔的 = 的列表来整操作，如下所示：

- 'quad_segs=#'：用于近似四分之一圆的段数（默认为 8）。
- 'join=round|mitre|bevel'：连接式（默认为“round”）。“mitre”也被认为是“mitre”的同义词。
- 'mitre_limit=#.#'：斜接比率限制（影响斜接连接式）。“miter_limit”也被接受为“mitre_limit”的同义词。

该函数是由 GEOS 模块执行的。

在 GEOS 3.11 中，行生成了化，因此在正偏移下，偏移曲线与输入具有相同的方向。

可用性：2.0

增：2.5 - 添加了 GEOMETRYCOLLECTION 和 MULTILINESTRING 的支持



Note

此函数忽略 Z 度。即使在 3D 几何体上使用，它也始终输出 2D 结果。

示例

Compute an open buffer around roads.

Code

```
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;
```

Offset 15 units with quad_segs=4 join=round.

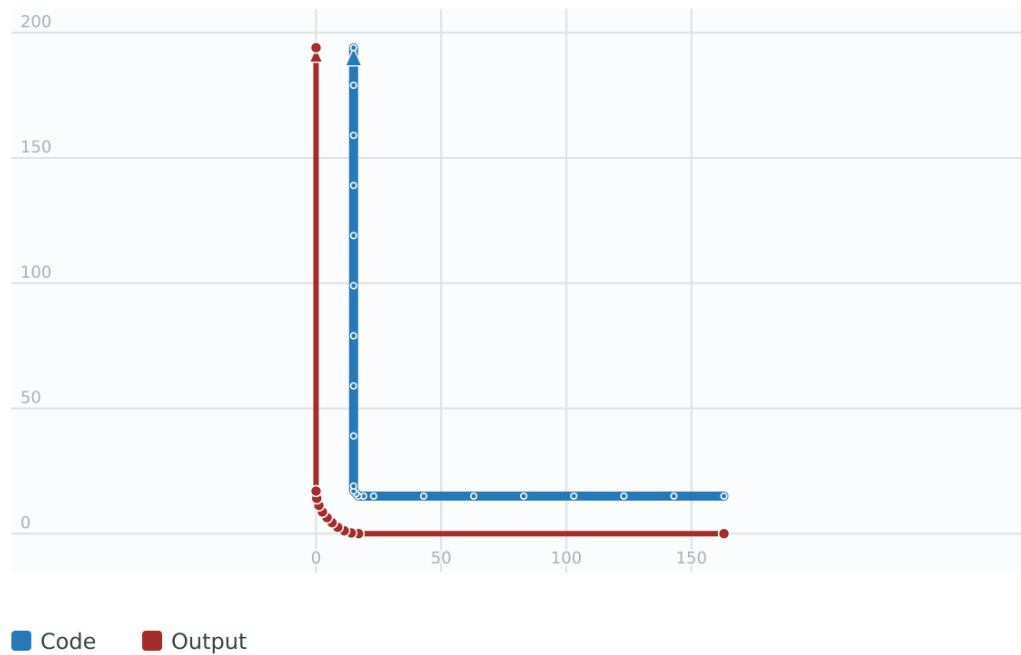
Code

```
SELECT 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 1,15.073645169758077 1.288220793951544,
12.259748514523654 2.141807012330698,9.66644650470597 3.527955815461818,
7.393398282201788 5.393398282201788,5.393398282201788 7.393398282201788,
3.527955815461818 9.66644650470597,2.141807012330696 12.259748514523654,
1.288220793951544 15.073645169758075,1 18,1 195)
```

Figure



Offset -15 units with quad_segs=4 join=round.

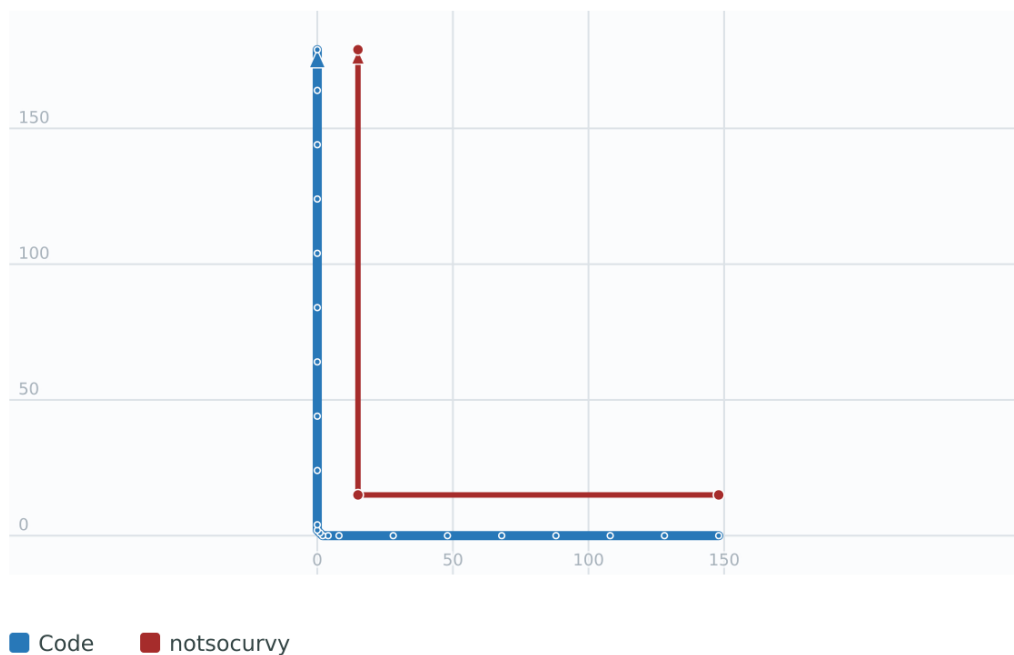
Code

```
SELECT 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;
```

Output

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


Figure



Apply two negative offsets to obtain a smoother curve; the distances combine to -45 units.

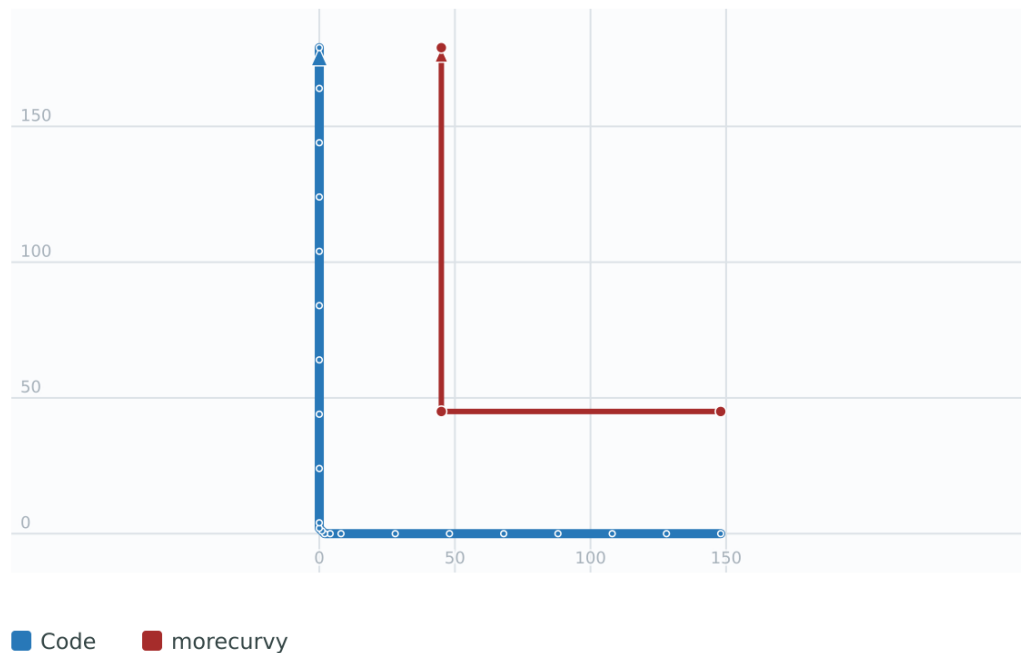
Code

```
SELECT 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;
```

Output

```
LINESTRING(164 61,61 61,61 195)
```

Figure



Collect the positive offset and the double-offset to produce parallel curves.

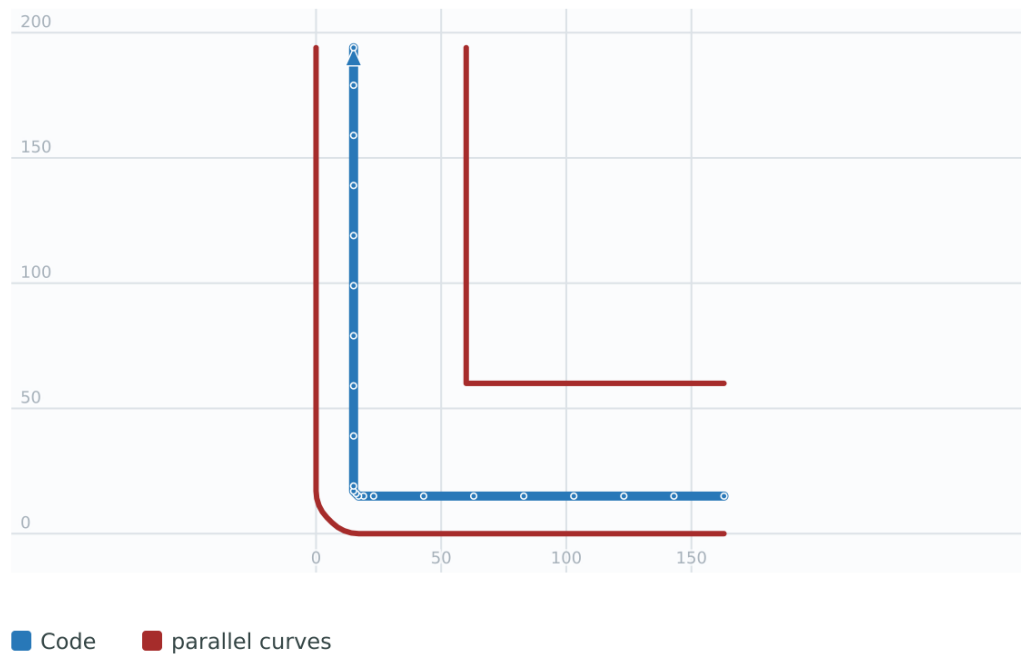
Code

```
SELECT 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;
```

Output

```
MULTILINESTRING((164 1,18 1,15.073645169758077 1.288220793951544,
12.259748514523654 2.141807012330698,9.66644650470597 3.527955815461818,
7.393398282201788 5.393398282201788,5.393398282201788 7.393398282201788,
3.527955815461818 9.66644650470597,2.141807012330696 12.259748514523654,
1.288220793951544 15.073645169758075,1 18,1 195),(164 61,61 61,61 195))
```

Figure



Offset 15 units with a bevel join.

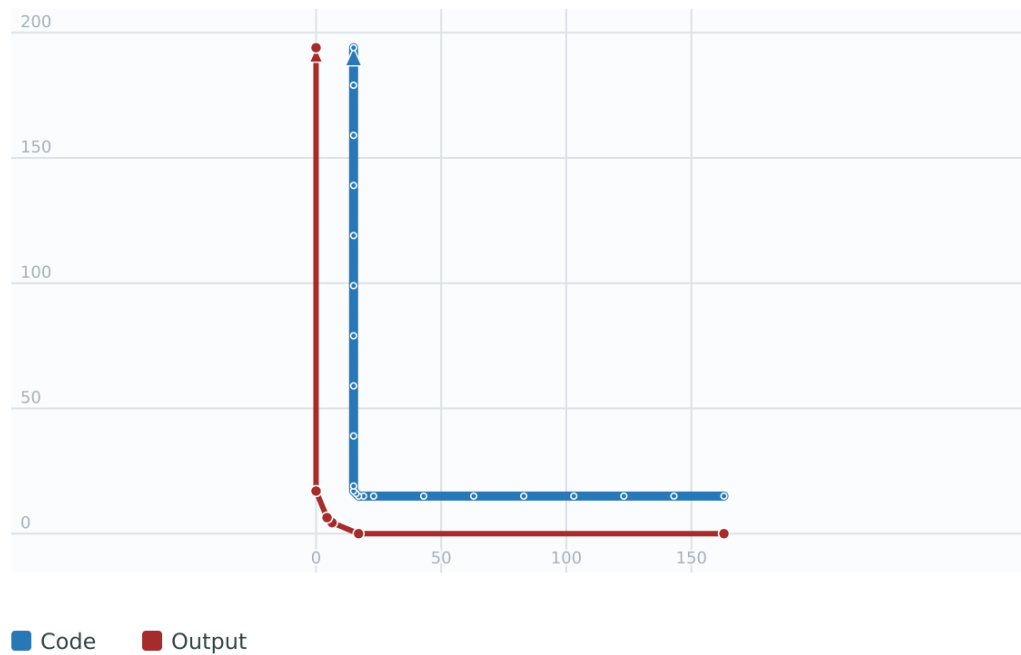
Code

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

Output

```
LINESTRING(164 1,18 1,7.393398282201788 5.393398282201788,
5.393398282201788 7.393398282201788,1 18,1 195)
```

Figure



Collect offsets on both sides using a mitre join and a mitre limit of 2.2.

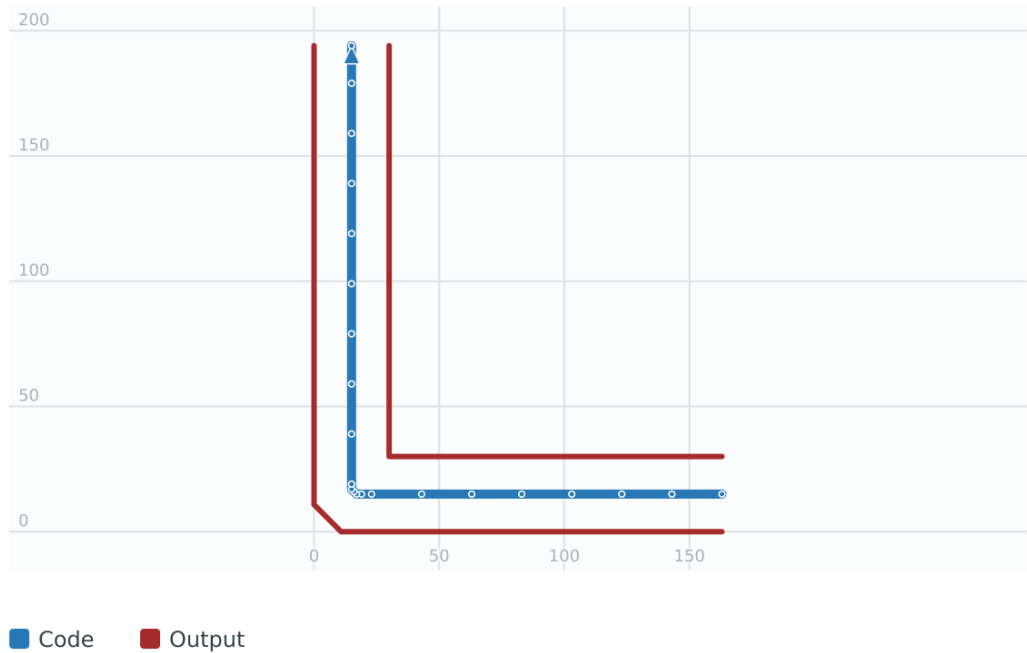
Code

```
SELECT ST_Collect(
  ST_OffsetCurve(geom, 15, 'quad_segs=4 join=mitre mitre_limit=2.2'),
  ST_OffsetCurve(geom, -15, 'quad_segs=4 join=mitre mitre_limit=2.2')
)
FROM ST_GeomFromText('LINESTRING(164 16,144 16,124 16,104 16,84 16,64 16,
44 16,24 16,20 16,18 16,17 17,
16 18,16 20,16 40,16 60,16 80,16 100,
16 120,16 140,16 160,16 180,16 195)') As geom;
```

Output

```
MULTILINESTRING((164 1,11.786796564403577 1,1 11.786796564403577,1 195),
(164 31,31 31,31 195))
```

Figure



相关信息

[ST_Buffer](#)

7.14.19 ST_PointOnSurface

ST_PointOnSurface — 计算并返回位于多边形或几何体上的点。

Synopsis

geometry **ST_PointOnSurface**(geometry g1);

描述

返回并位于曲面内部的 POINT (POLYGON、MULTIPOLYGON 和 CURVEPOLYGON)。在 PostGIS 中，此函数也适用于线和点几何形状。

- ✓ 此方法符合了 [SQL 1.1 的 OGC 功能规范](#)。 s3.2.14.2 // s3.2.18.2
- ✓ 该方法符合了 SQL/MM 规范。SQL-MM 3 : 8.1.5,9.5.6。规范定义了表面几何形状定义 ST_PointOnSurface。PostGIS 扩展了功能以支持所有常见的几何类型。其他数据库（Oracle、DB2、ArcSDE）似乎不支持曲面的此功能。SQL Server 2008 支持所有常见的几何类型。
- ✓ 该函数支持 3d 并且不会丢失 z-index。

示例

Point on surface of a MultiPoint.

Code

```
SELECT ST_PointOnSurface(  
  'MULTIPOINT (8 24,10 92,12 154,17 68,28 10,29 52,29 84,55 50,56 24,131 14,160 180,189  
    180)'); ↵
```

Output

```
POINT(55 50)
```

Figure



Point on surface of a LineString.

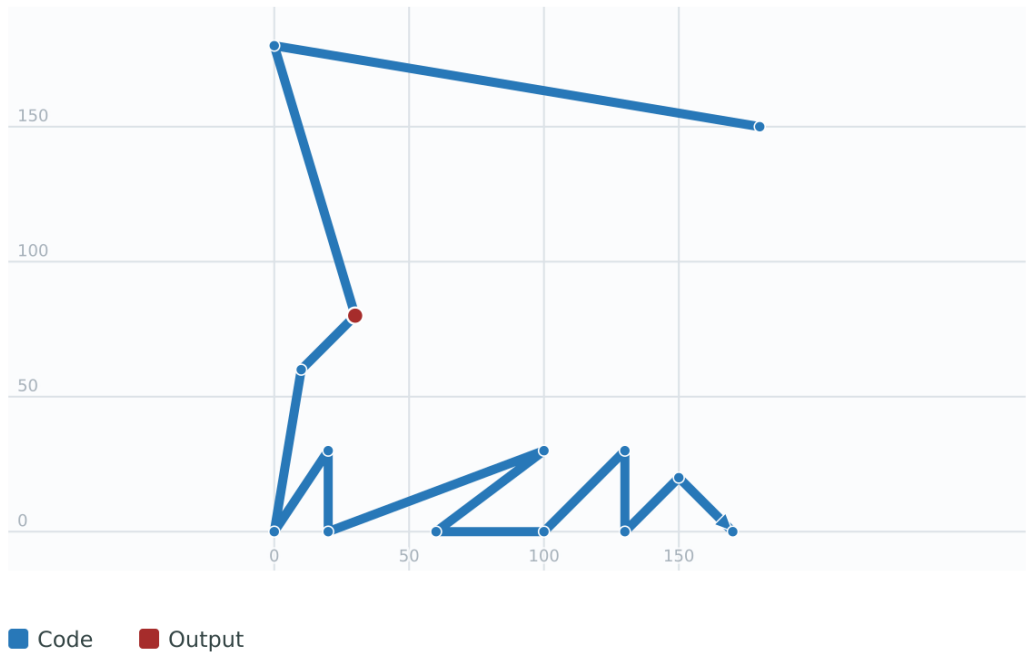
Code

```
SELECT ST_PointOnSurface(  
  'LINESTRING (190 160,10 190,40 90,20 70,10 10,30 40,30 10,110 40,70 10,110 10,140 40,140  
    10,160 30,180 10)'); ↵
```

Output

```
POINT(40 90)
```

Figure



Point on surface of a Polygon.

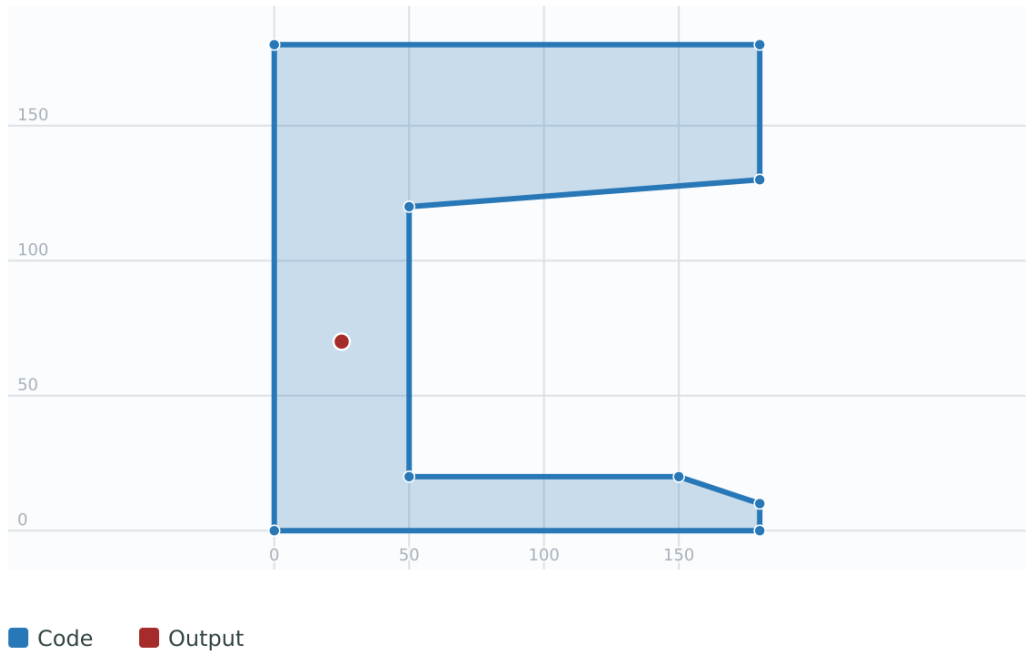
Code

```
SELECT ST_PointOnSurface(
  'POLYGON ((190 190,10 190,10 10,190 10,190 20,160 30,60 30,60 130,190 140,190 190))');
```

Output

```
POINT(35 80)
```

Figure



Point on surface of the highest-dimension components of a GeometryCollection.

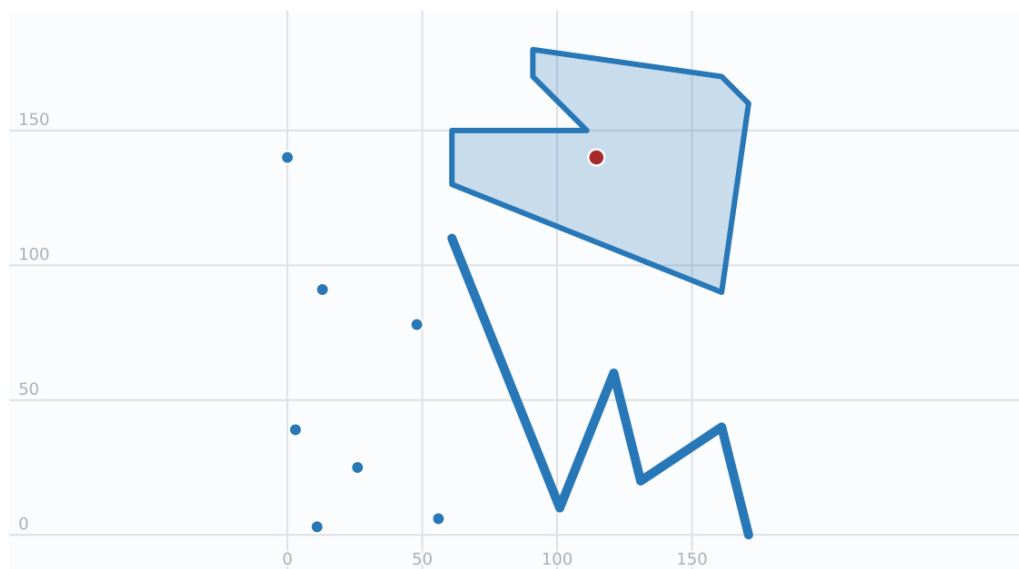
Code

```
SELECT ST_PointOnSurface(
  'GEOMETRYCOLLECTION (POLYGON ((190 170,180 100,80 140,80 160,130 160,110 180,110 190,180 190 170)),
    LINESTRING (80 120,120 20,140 70,150 30,180 50,190 10),
    MULTIPOINT (19 150,22 49,30 13,32 101,45 35,67 88,75 16))');
```

Output

```
POINT(133.571 150)
```

Figure

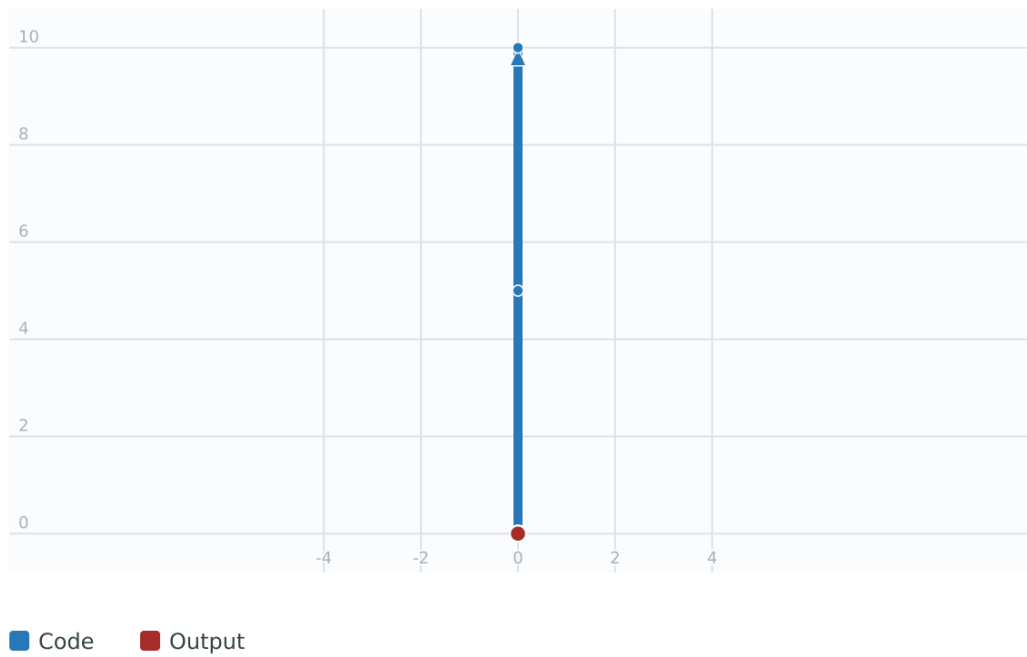


■ Code ■ Output

Code

```
SELECT ST_PointOnSurface('LINESTRING(0 5 1,0 0 1,0 10 2)');
```


Figure

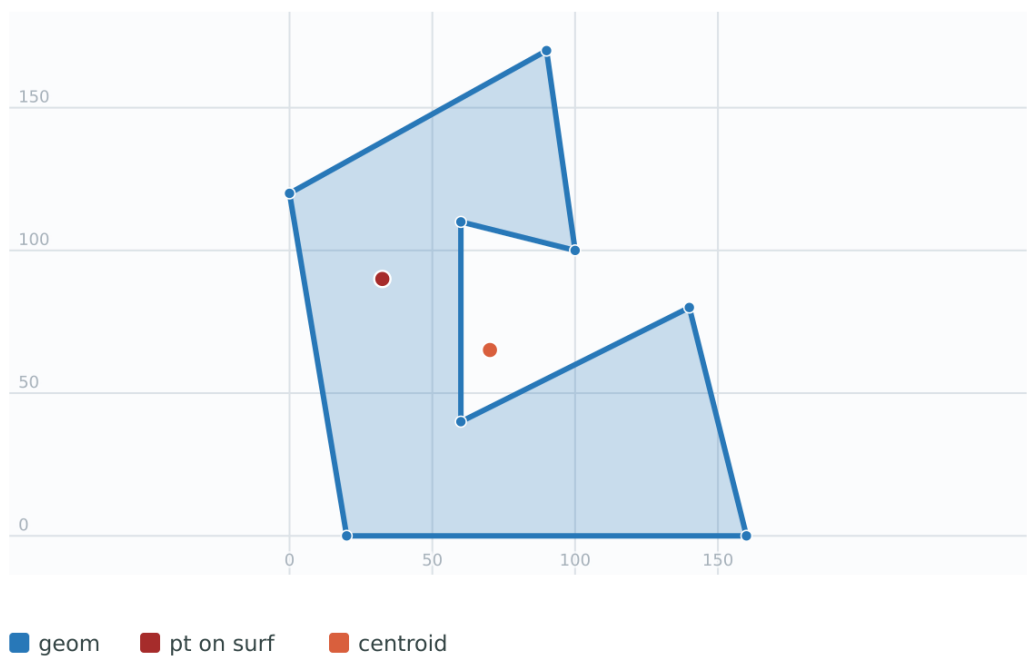


示例：ST_PointOnSurface 的结果保证位于多边形内，而 ST_Centroid 算的点可能位于多边形外。

Code

```
SELECT ST_PointOnSurface(geom) AS pt_on_surf, 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;
```

Figure



相关信息

[ST_Centroid](#), [ST_MaximumInscribedCircle](#)

7.14.20 ST_Polygonize

`ST_Polygonize` — 计算由一组几何线形的线形成的多边形集合。

Synopsis

```
geometry ST_Polygonize(geometry set geomfield);  
geometry ST_Polygonize(geometry[] geom_array);
```

描述

构建一个 `GeometryCollection`，其中包含由一组几何线形的线形成的多边形。如果输入线未形成任何多边形，返回空的 `GeometryCollection`。

此函数构建覆盖所有分隔区域的多边形。如果结果旨在形成有效的多边形几何体，使用 [ST_BuildArea](#) 来防止填充孔。



Note

输入线必须是正确点才能使功能正常工作。为了确保输入已点化，在多边形化之前在输入几何体上使用 [ST_Node](#)。



Note

使用外部工具可能很管理几何集合。使用[ST_Dump](#) 将多边形结果分解为独立的多边形。

该函数是由 GEOS 模块执行的。

可用性：1.0.0RC1

示例

Polygonize a set of input lines.

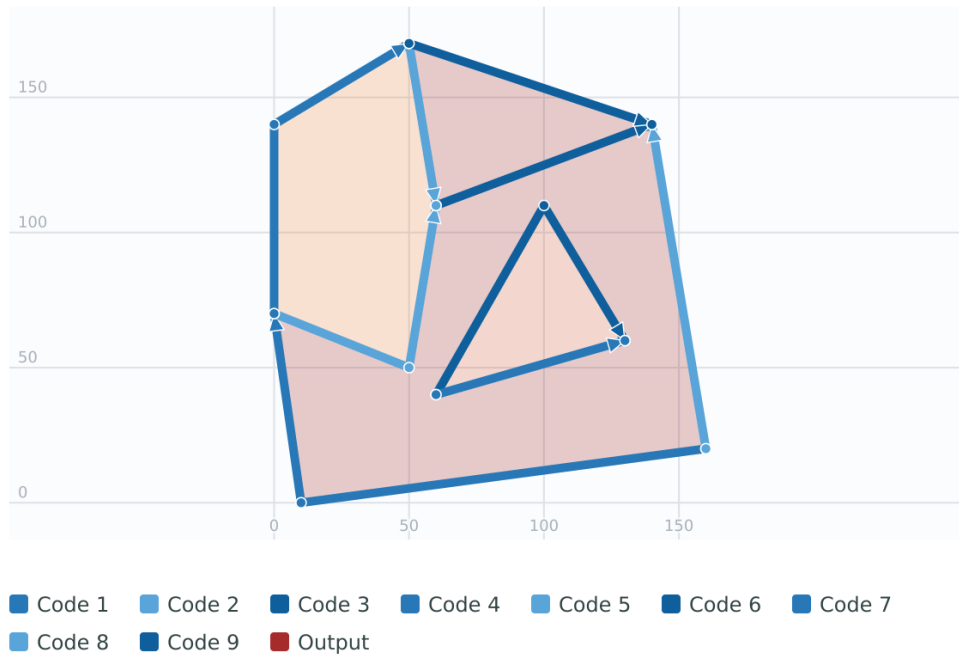
Code

```
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_Polygonize(geom)  
FROM data;
```

Output

```
GEOMETRYCOLLECTION(POLYGON((180 40,30 20,20 90,70 70,80 130,160 160,180 40)), POLYGON ←
((80 60,120 130,150 80,80 60)), POLYGON((80 130,70 70,20 90,20 160,70 190,80 130)), POLYGON((160 160,80 130,70 ←
190,160 160)))
```

Figure

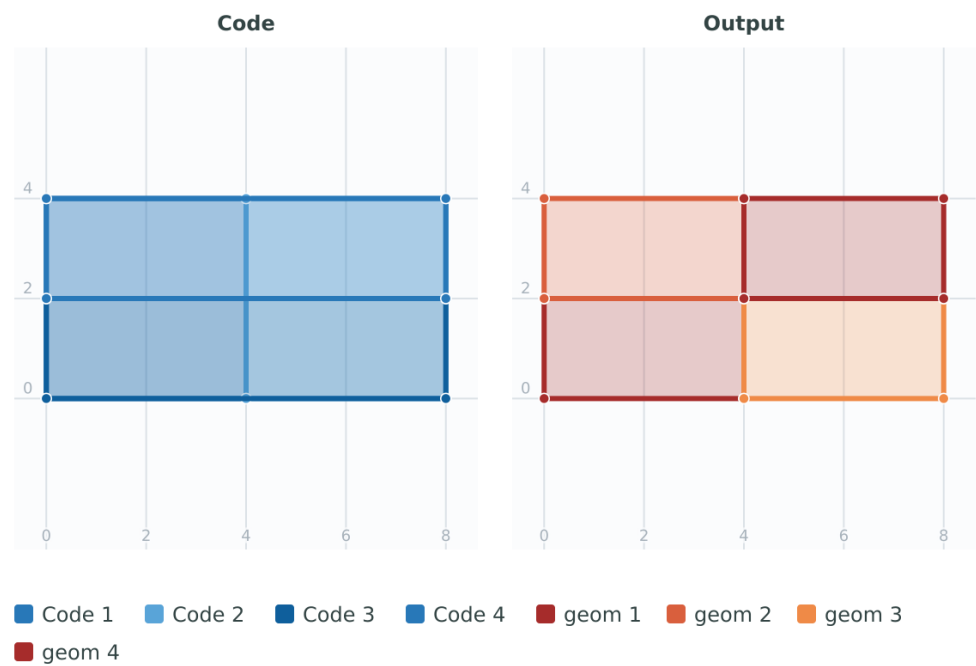


Polygonizing two polygon coverages and transferring attributes to the resultants:

Code

```
WITH poly_a(id, geom) AS (VALUES
  ('a1', 'POLYGON ((0 0, 4 0, 4 4, 0 4, 0 0))'::geometry),
  ('a2', 'POLYGON ((4 0, 8 0, 8 4, 4 4, 4 0))'::geometry)
),
poly_b(id, geom) AS (VALUES
  ('b1', 'POLYGON ((0 0, 8 0, 8 2, 0 2, 0 0))'::geometry),
  ('b2', 'POLYGON ((0 2, 8 2, 8 4, 0 4, 0 2))'::geometry)
),
lines AS (
  SELECT ST_Boundary(geom) AS geom FROM poly_a
  UNION ALL
  SELECT ST_Boundary(geom) FROM poly_b
),
noded AS (
  SELECT ST_UnaryUnion(ST_Collect(geom)) AS geom FROM lines
),
cells AS (
  SELECT (ST_Dump(ST_Polygonize(geom))).geom AS geom FROM noded
)
SELECT a.id AS id_a,
       b.id AS id_b,
       c.geom AS geom
FROM cells AS c
LEFT JOIN poly_a AS a ON ST_Within(ST_PointOnSurface(c.geom), a.geom)
LEFT JOIN poly_b AS b ON ST_Within(ST_PointOnSurface(c.geom), b.geom)
WHERE a.id IS NOT NULL OR b.id IS NOT NULL
ORDER BY id_a, id_b;
```

Figure



多边形串表：

Code

```
SELECT ST_Polygonize(geom_4269) As geomtextrep
FROM (SELECT geom_4269 FROM ma.suffolk_edges) As foo;
```

Output

```
geomtextrep
-----
SRID=4269;GEOMETRYCOLLECTION(POLYGON((-71.040878 42.285678,-71.040943 42.2856,-71.04096 42.285752,-71.040878 42.285678)),
POLYGON((-71.17166 42.353675,-71.172026 42.354044,-71.17239 42.354358,-71.171794 42.354971,-71.170511 42.354855,-71.17112 42.354238,-71.17166 42.353675)))
(1 row)
```

Use ST_Dump to dump out the polygonize geoms into individual polygons.

Code

```
SELECT (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;
```

Output

```
geomtextrep
-----
SRID=4269;POLYGON((-71.040878 42.285678,-71.040943 42.2856,-71.04096 42.285752,-71.040878 42.285678))
SRID=4269;POLYGON((-71.17166 42.353675,-71.172026 42.354044,-71.17239 42.354358,-71.171794 42.354971,-71.170511 42.354855,-71.17112 42.354238,-71.17166 42.353675))
(2 rows)
```

相关信息

[ST_BuildArea](#), [ST_Boundary](#), [ST_Collect](#), [ST_Dump](#), [ST_Node](#), [ST_PointOnSurface](#), [ST_UnaryUnion](#), [ST_Within](#)

7.14.21 ST_ReducePrecision

`ST_ReducePrecision` — 返回有效的几何图形，其点舍入到网格公差。

Synopsis

geometry **ST_ReducePrecision**(geometry g, float8 gridsize);

描述

返回有效的几何图形，其中所有点均舍入到提供的网格公差，并删除低于公差的要素。

与 [ST_SnapToGrid](#) 不同，返回的几何图形将是有效的，没有自相交或折回件。

精度降低可用于：

- 将坐标精度与数据精度相匹配
- 减少表示几何图形所需的坐标数量
- 确保有效的几何图形输出使用低精度的格式（例如，当输出小数位数有限时，WKT、GeoJSON 或 KML 等文本格式）。
- 将有效几何图形输出到使用低或有限精度的系统（例如 SDE、Oracle 公差）

可用性：3.1.0。

需要 GEOS >= 3.9.0。

示例

Code

```
SELECT ST_ReducePrecision('POINT(1.412 19.323)', 0.1);
```

Output

```
POINT(1.4 19.3)
```

Figure



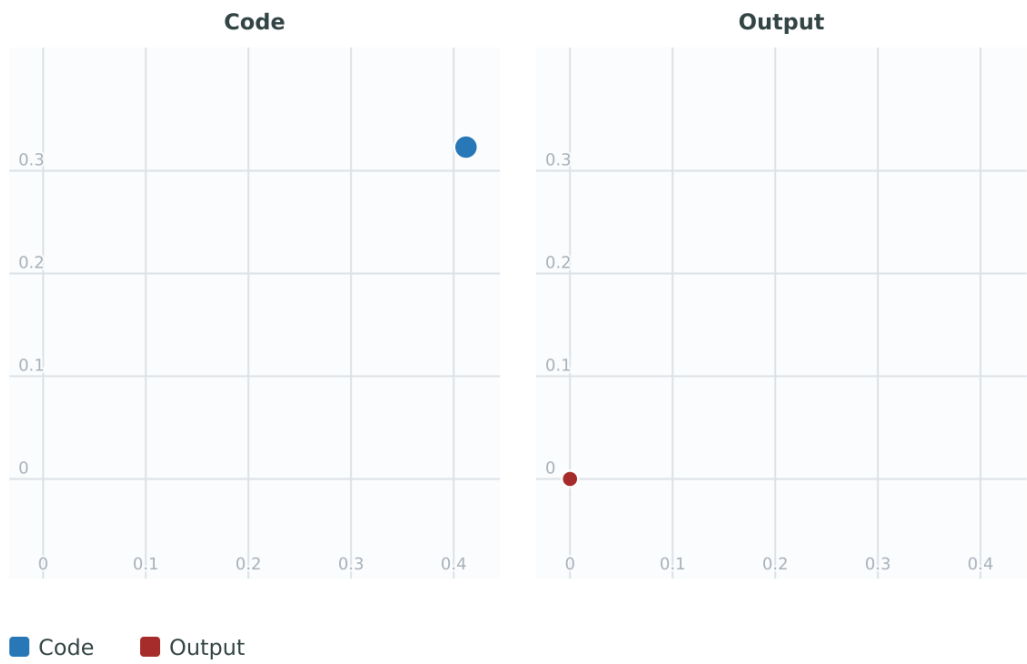
Code

```
SELECT ST_ReducePrecision('POINT(1.412 19.323)', 1.0);
```

Output

```
POINT(1 19)
```

Figure



Code

```
SELECT ST_ReducePrecision('POINT(1.412 19.323)', 10);
```

Output

```
POINT(0 20)
```

Figure



降低精度可以减少点数量

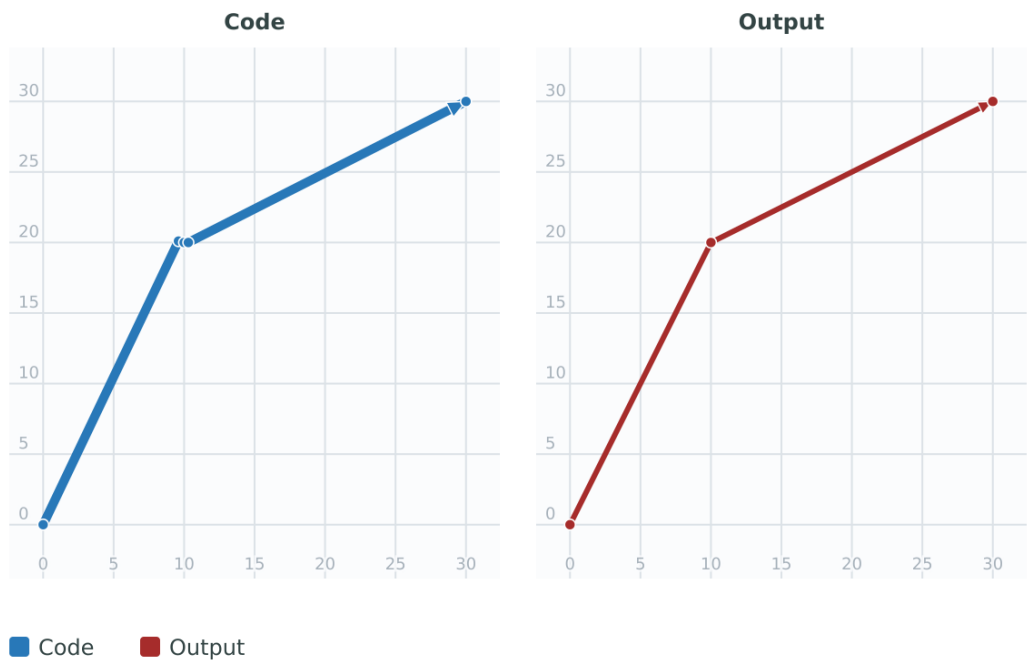
Code

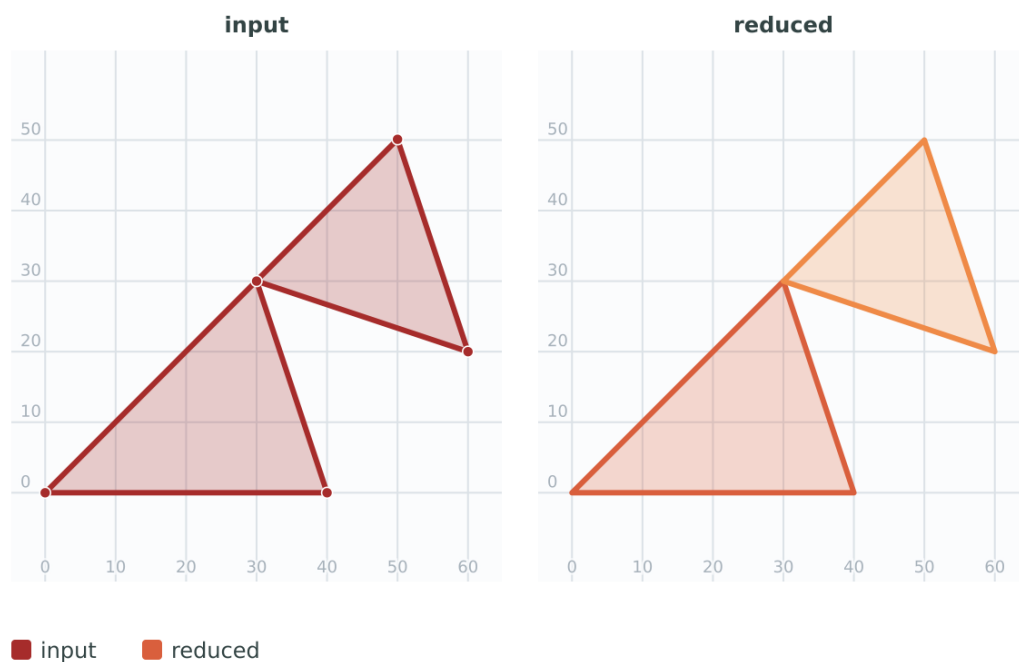
```
SELECT ST_ReducePrecision('LINESTRING (10 10,19.6 30.1,20 30,20.3 30,40 40)', 1);
```

Output

```
LINESTRING (10 10, 20 30, 40 40)
```

Figure



Figure

相关信息

[ST_SnapToGrid](#), [ST_Simplify](#), [ST_SimplifyVW](#)

7.14.22 ST_SharedPaths

`ST_SharedPaths` — 返回一个集合，其中包含一个输入串/多串共享的路径。

Synopsis

```
geometry ST_SharedPaths(geometry lineal1, geometry lineal2);
```

描述

返回一个集合，其中包含一个输入几何体共享的路径。那些朝相同方向去的在集合的第一个元素中，那些朝相反方向去的在第二个元素中。路径本身是在第一个几何体的方向上输出的。

这个函数是由 GEOS 模块实现的。

可用性: 2.0.0

示例：找共享路径

Adjacent shared path segments may be returned as separate or merged LineStrings depending on the GEOS version.

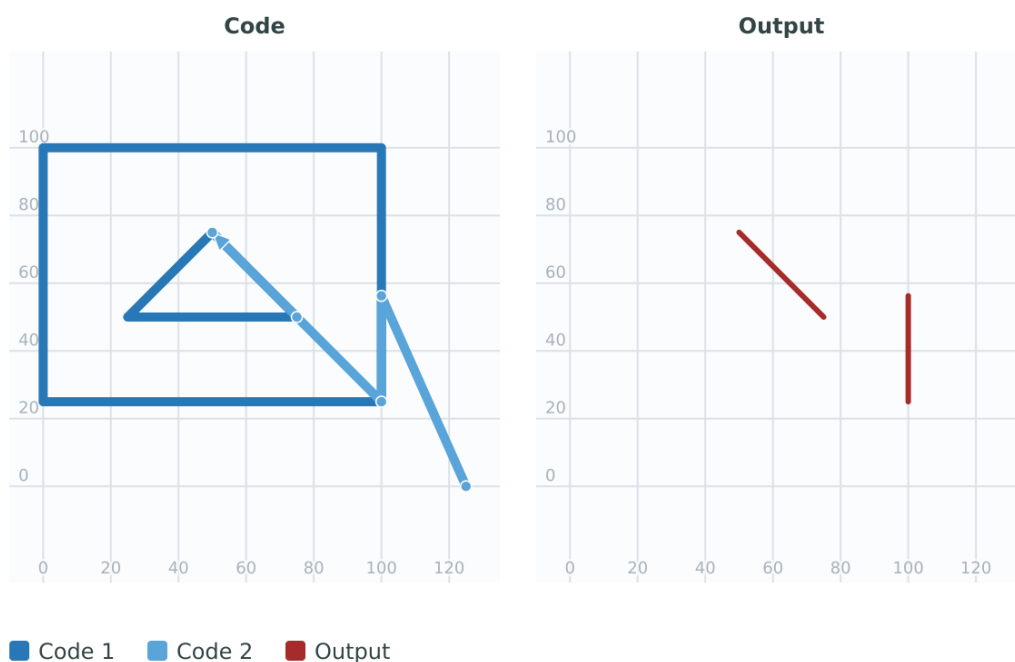
Find the shared paths of a MultiLineString and a LineString.

Code

```
SELECT 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(151 100,126 156.25,126 125,101 150,76 175)'))
```

Output

```
GEOMETRYCOLLECTION(MULTILINESTRING((126 156.25,126 125),
(101 150,76 175)),MULTILINESTRING EMPTY)
```

Figure

Flip the LineString orientation to return the same paths in the opposite-direction member.

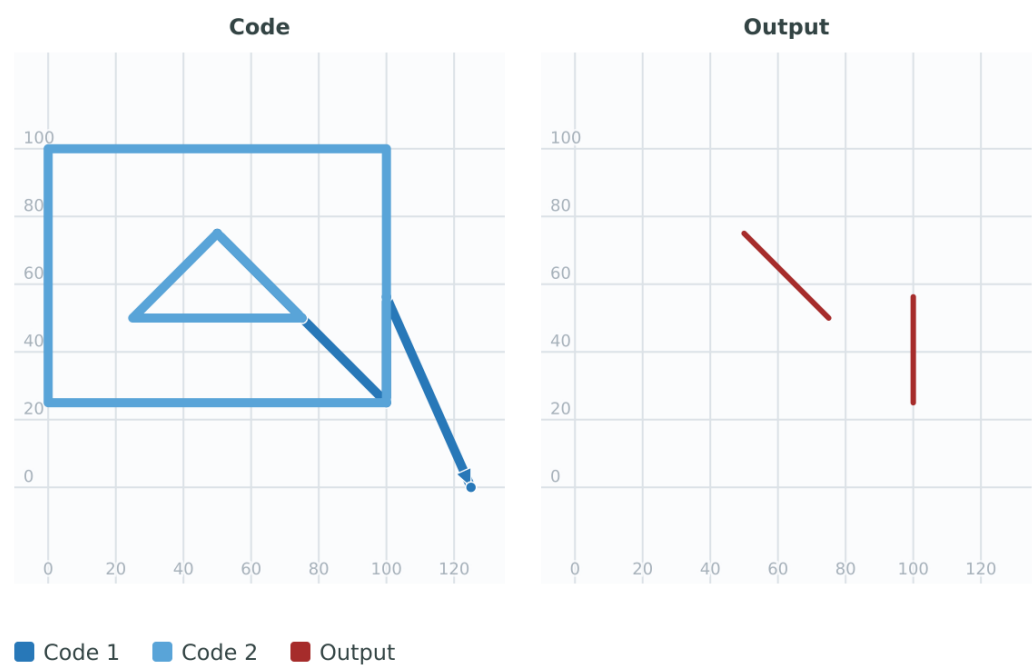
Code

```
SELECT ST_SharedPaths(ST_GeomFromText('LINESTRING(76 175,101 150,126 125,126 156.25,151 100)'),
ST_GeomFromText('MULTILINESTRING((26 125,26 200,126 200,126 125,26 125),
(51 150,101 150,76 175,51 150))'))
```

Output

```
GEOMETRYCOLLECTION(MULTILINESTRING EMPTY,
MULTILINESTRING((76 175,101 150),(126 125,126 156.25)))
```

Figure



相关信息

[ST_Dump](#), [ST_GeometryN](#), [ST_NumGeometries](#)

7.14.23 ST_Simplify

`ST_Simplify` — 返回几何图形的简化表示，使用 Douglas-Peucker 算法。

Synopsis

```
geometry ST_Simplify(geometry geom, float tolerance);
geometry ST_Simplify(geometry geom, float tolerance, boolean preserveCollapsed);
```

描述

使用 [Douglas-Peucker 算法](#) 计算几何图形的简化表示。简化的 `tolerance` 是一个距离值，以输入 SRS 的方位为准。简化会移除在简化线条的公差距离内的点。即使输入是有效的，结果可能不是有效的。

该函数可用于任何类型的几何图形（包括 `GeometryCollections`），但只有线和多边形元素会被简化。线性几何的端点将被保留。

`preserveCollapsed` 标志保留那些在给定公差下本会被移除的小几何图形。例如，如果将一个 1 米宽的线条使用 10 米的公差进行简化，当 `preserveCollapsed` 为 `true` 时，该线条将不会消失。该标志对于渲染目的很有用，以防止地图上非常小的特征消失。



Note

返回的几何图形可能会失去其拓扑性（参见 [ST_IsSimple](#)），拓扑可能不会被保留，并且多边形结果可能是无效的（参见 [ST_IsValid](#)）。使用 [ST_SimplifyPreserveTopology](#) 来保留拓扑并确保有效性。



Note

☐函数不保留多☐形之☐共享的☐界。如果需要保留共享☐界，☐使用ST_CoverageSimplify。

可用性 : 1.2.2

示例

A circle simplified with a medium tolerance becomes an octagon, and with a large tolerance disappears.

Code

```
SELECT ST_NPoints(geom) AS np_before,
       ST_NPoints(ST_Simplify(geom, 0.1)) AS np01_notbadcircle,
       ST_NPoints(ST_Simplify(geom, 0.5)) AS np05_notquitecircle,
       ST_NPoints(ST_Simplify(geom, 1)) AS np1_octagon,
       ST_NPoints(ST_Simplify(geom, 10)) AS np10_gone,
       (ST_Simplify(geom, 100) is null) AS np100_geometrygoesaway
FROM (SELECT ST_Buffer('POINT(1 3)', 10, 12) As geom) AS t;
```

Output

```
-[ RECORD 1 ]-----
np_before           | 49
np01_notbadcircle   | 33
np05_notquitecircle | 17
np1_octagon         | 9
np10_gone           | null
np100_geometrygoesaway | t
```

☐化一☐☐。☐化后，☐可能会相交。

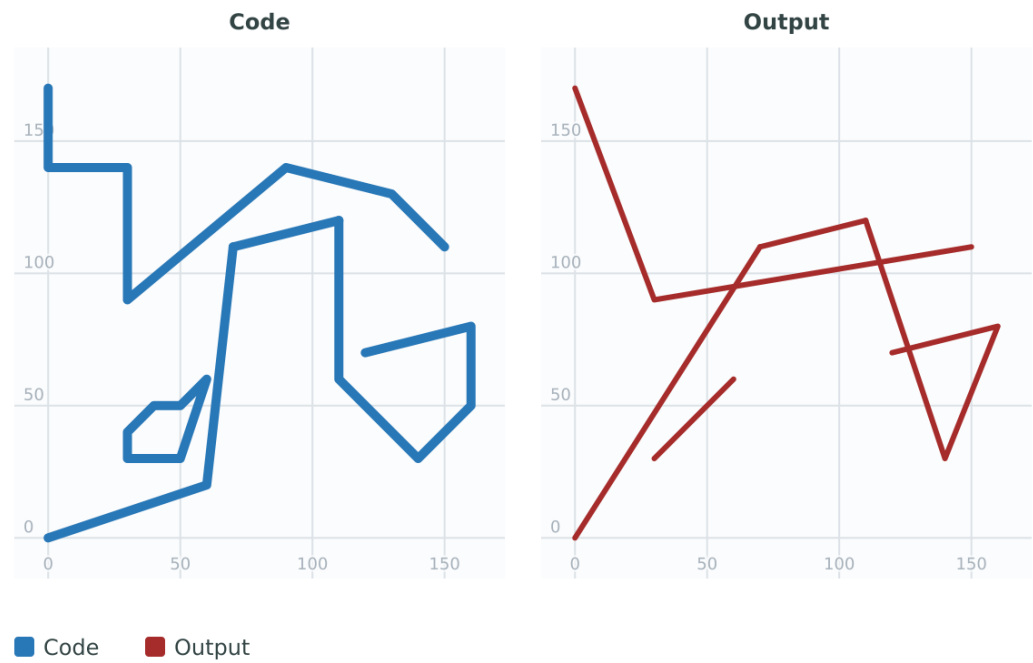
Code

```
SELECT ST_Simplify('MULTILINESTRING ((20 180,20 150,50 150,50 100,110 150,150 140,170 120) ↵
, (20 10,80 30,90 120), (90 120,130 130), (130 130,130 70,160 40,180 60,180 90,140 80), (50 ↵
40,70 40,80 70,70 60,60 60,50 50,50 40))'::geometry,
40);
```

Output

```
MULTILINESTRING((20 180,50 100,170 120), (20 10,90 120), (90 120,130 130), (130 130,160 40,180 90,140 80), (50 40,80 70,50 ↵
40))
```

Figure



简化一个 MultiPolygon。多边形形的结果可能是无效的。

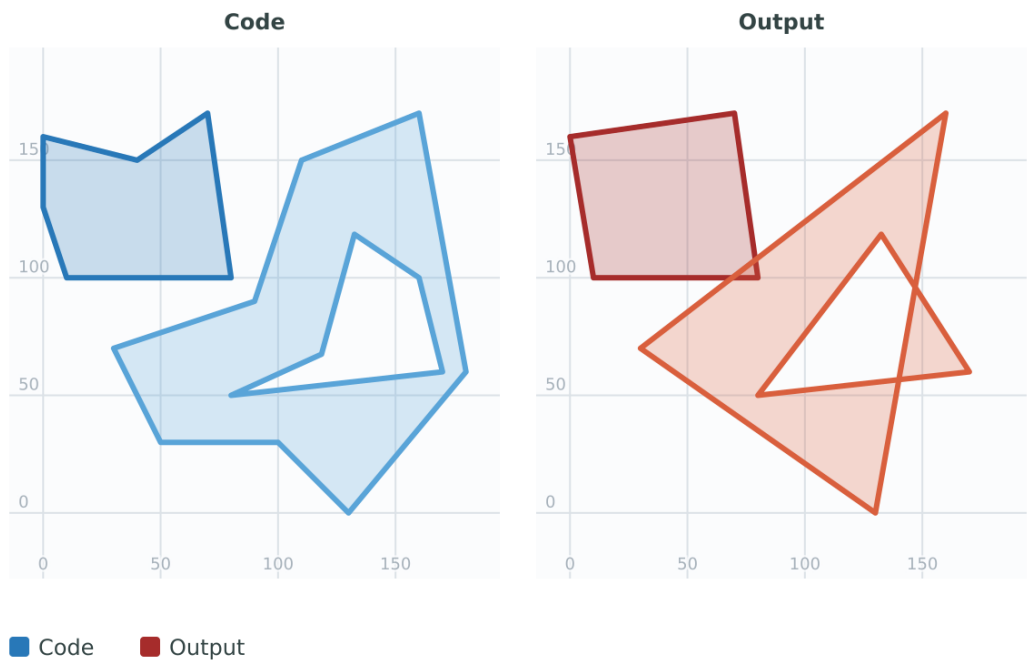
Code

```
SELECT ST_Simplify('MULTIPOLYGON (((90 110,80 180,50 160,10 170,10 140,20 110,90 110)),((40 80,100 100,120 160,170 180,190 70,140 10,110 40,60 40,40 80)),(180 70,170 110,142.5 128.5,128.5 77.5,90 60,180 70)))'::geometry, 40);
```

Output

```
MULTIPOLYGON(((90 110,80 180,10 170,20 110,90 110)),((40 80,170 180,140 10,40 80)),(180 70,142.5 128.5,90 60,180 70)))
```

Figure



相关信息

[ST_IsSimple](#), [ST_SimplifyPreserveTopology](#), [ST_SimplifyVW](#), [ST_CoverageSimplify](#), [Topology](#) [ST_Simplify](#)

7.14.24 ST_SimplifyPreserveTopology

ST_SimplifyPreserveTopology — 使用 Douglas-Peucker 算法返回几何图形的简化且有效表示。

Synopsis

geometry **ST_SimplifyPreserveTopology**(geometry geom, float tolerance);

描述

使用一种具体的 **Douglas-Peucker 算法** 计算几何图形的简化表示，该算法限制简化以确保结果具有与输入相同的拓扑。简化的 **tolerance** 是一个距离值，以输入 SRS 的位为准。只要保留拓扑，简化会移除在简化线条的公差距离内的点。如果输入是有效和简单的，结果也将是有效和简单的。

该函数可用于任何类型的几何图形（包括 GeometryCollections），但只有面和多面体元素会被简化。对于多面体输入，结果将具有相同数量的面（外壳和内面），并且这些面不会相交。面的端点可能会被简化。对于线性输入，结果将具有相同数量的面，如果在原始几何中它们不相交，结果也不会相交。线性几何的端点将被保留。



Note

该函数不保留多面体之面共享的边界。如果需要保留共享边界，请使用 [ST_CoverageSimplify](#)。

该函数是由 GEOS 模块执行的。

可用性：1.3.3

示例

For the same example as [ST_Simplify](#), ST_SimplifyPreserveTopology prevents the geometry from disappearing.

Code

```
SELECT ST_NPoints geom AS np_before,
       ST_NPoints(ST_SimplifyPreserveTopology(geom, 0.1)) AS np01_notbadcircle,
       ST_NPoints(ST_SimplifyPreserveTopology(geom, 0.5)) AS np05_notquitecircle,
       ST_NPoints(ST_SimplifyPreserveTopology(geom, 1)) AS np1_octagon,
       ST_NPoints(ST_SimplifyPreserveTopology(geom, 10)) AS np10_minimal,
       ST_NPoints(ST_SimplifyPreserveTopology(geom, 100)) AS np100_minimal
FROM (SELECT ST_Buffer('POINT(1 3)', 10, 12) AS geom) AS t;
```

Output

np_before	np01_notbadcircle	np05_notquitecircle	np1_octagon	np10_minimal	
49	33	17	9	4	
4					

简化几何，保留非相交几何的拓扑。

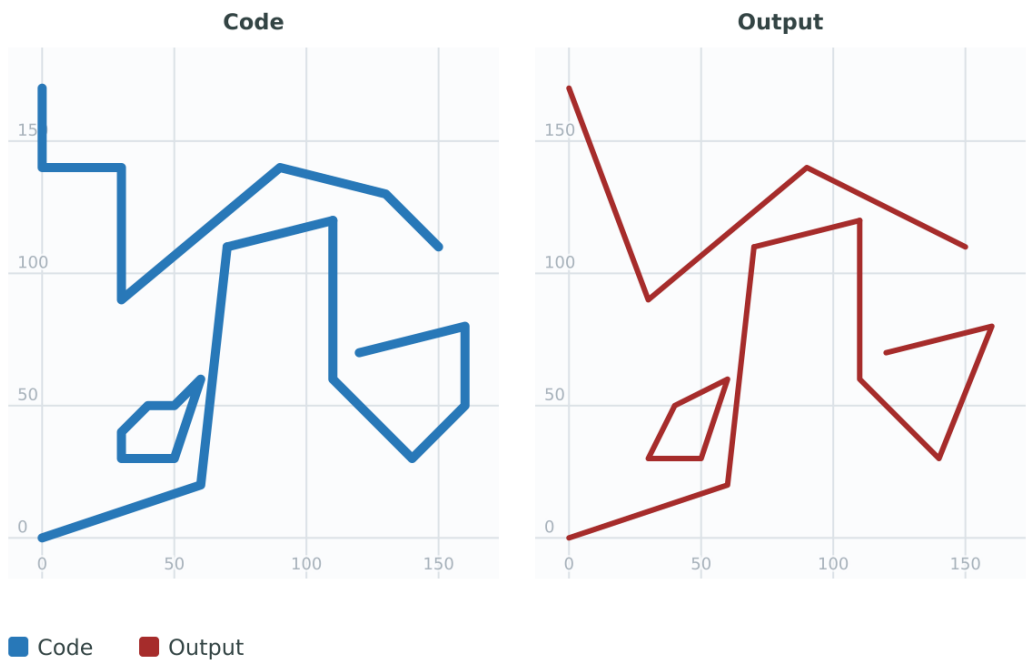
Code

```
SELECT ST_SimplifyPreserveTopology('MULTILINESTRING ((20 180,20 150,50 150,50 100,110 150,150 140,170 120),(20 10,80 30,90 120),(90 120,130 130),(130 130,130 70,160 40,180 60,180 90,140 80),(50 40,70 40,80 70,70 60,60 60,60 50,50 50 40))'::geometry, 40);
```

Output

```
MULTILINESTRING((20 180,50 100,110 150,170 120),(20 10,80 30,90 120),(90 120,130 130),(130 130,130 70,160 40,180 90,140 80),(50 40,70 40,80 70,60 60,50 40))
```

Figure



简化一个 MultiPolygon，保留外壳和内环的拓扑。

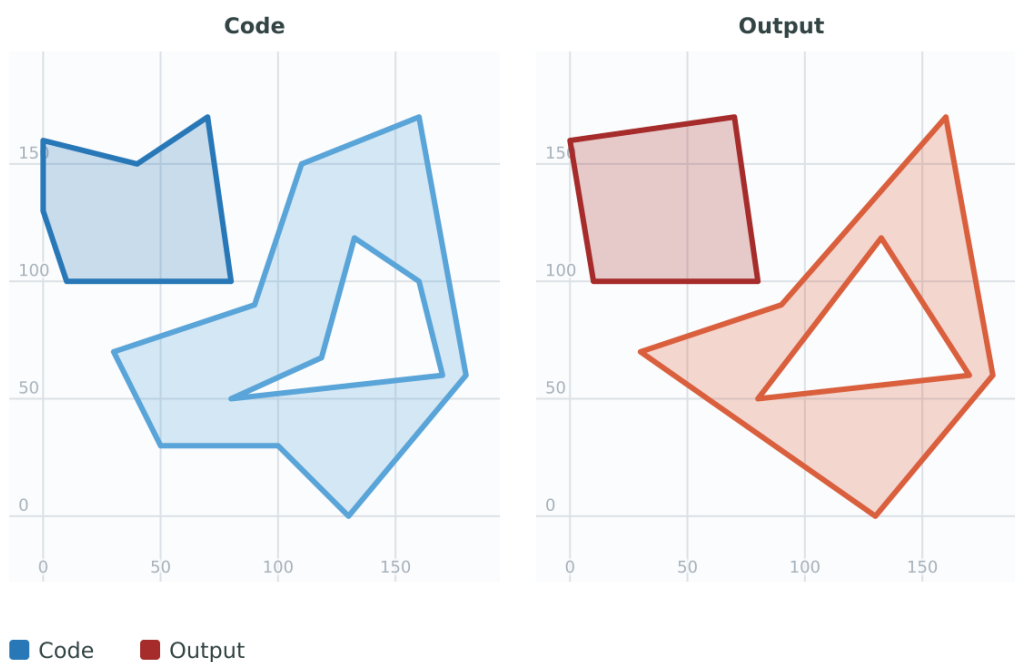
Code

```
SELECT ST_SimplifyPreserveTopology('MULTIPOLYGON (((90 110,80 180,50 160,10 170,10 140,20 110,90 110))),((40 80,100 100,120 160,170 180,190 70,140 10,110 40,60 40,40 80)),(180 70,170 110,142.5 128.5,128.5 77.5,90 60,180 70)))'::geometry, 40);
```

Output

```
MULTIPOLYGON(((90 110,80 180,10 170,20 110,90 110))),((40 80,100 100,170 180,190 70,140 10,40 80)),(180 70,142.5 128.5,90 60,180 70)))
```

Figure



相关信息

[ST_Simplify](#), [ST_SimplifyVW](#), [ST_CoverageSimplify](#)

7.14.25 ST_SimplifyPolygonHull

ST_SimplifyPolygonHull — 计算多边形几何的简化的保留拓扑的外部或内部外壳。

Synopsis

```
geometry ST_SimplifyPolygonHull(geometry param_geom, float vertex_fraction, boolean is_outer = true);
```


描述

计算多边形几何的简化的保留拓扑的外部或内部外壳。外壳完全覆盖输入几何体。内部外壳完全被输入几何体覆盖。结果是由输入点的子集形成的多边形几何体。处理多重多边形和孔并生成与输入相同的结果。

点数的减少可以通过 `vertex_fraction` 参数进行控制。参数的取值范围是 0 到 1。小的值会使简化过程更深入，点数减少，凹性降低。对于外部和内部的凹包，当 `vertex_fraction` 为 1.0 时，会生成与原始几何体相同的形状；而当为 0.0 时，对于外部凹包会生成凸包（一多边形），而对于内部凹包会生成三角形。

简化过程通过逐步删除包含最少面数的凹角来进行，直到达到点数目。它可以防止交叉，因此结果始终是有效的多边形几何体。

为了得到包含相关线段的几何体的更好结果，可能需要“分段”输入，如下所示。

该函数是由 GEOS 模块行的。

可用性：3.3.0。

需要 GEOS >= 3.11.0。

示例

Outer hull of a Polygon.

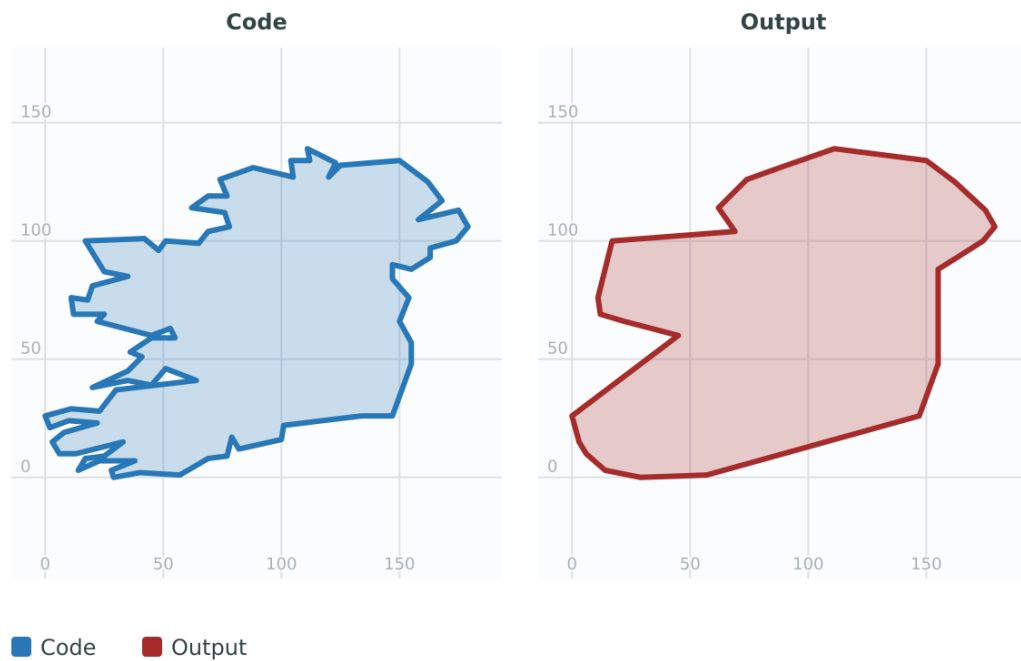
Code

```
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)) ↵
  '::geometry,
  0.3
);
```

Output

```
POLYGON((161 165,173 156,186 144,190 137,185 131,174 124,166 119,166 79,158 57,68 32,40 31,25 34,17 41,14 46,11 57,56 ↵
  91,33 97,23 100,22 107,28 131,80 135,73 145,85 157,99 162,122 170,161 165))
```

Figure



Inner hull of a Polygon.

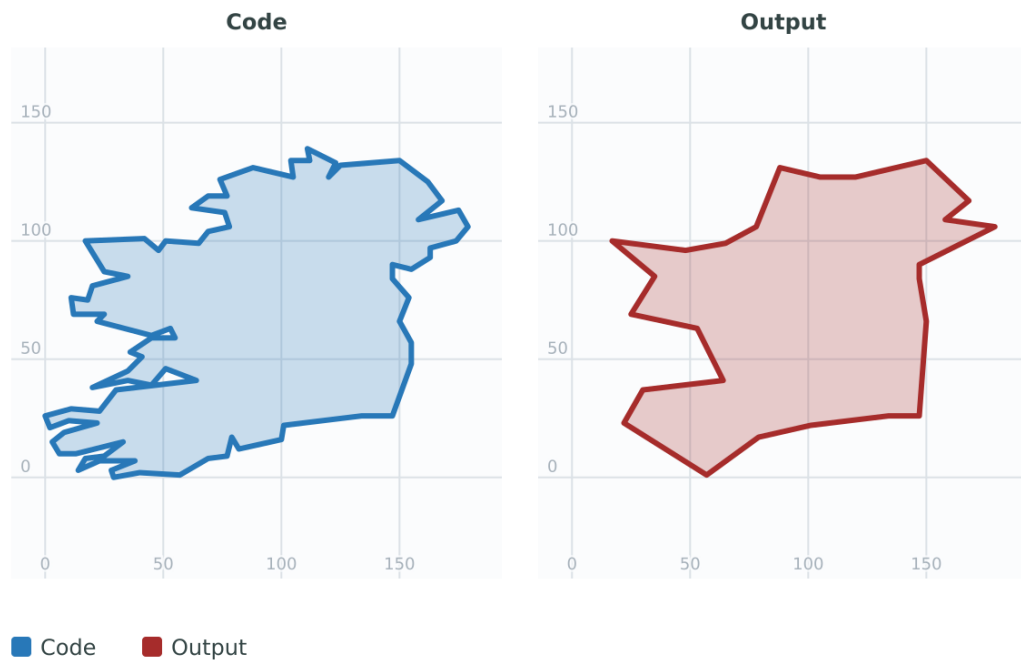
Code

```
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)) ↵
  '::geometry,
  0.3, false
);
```

Output

```
POLYGON((131 158,116 158,99 162,89 137,76 130,59 127,28 131,46 116,36 100,64 94,75 72,41 68,33 54,68 32,90 48,112 ↵
  53,145 57,158 57,161 97,158 115,158 121,190 137,169 140,179 148,161 165,131 158))
```

Figure

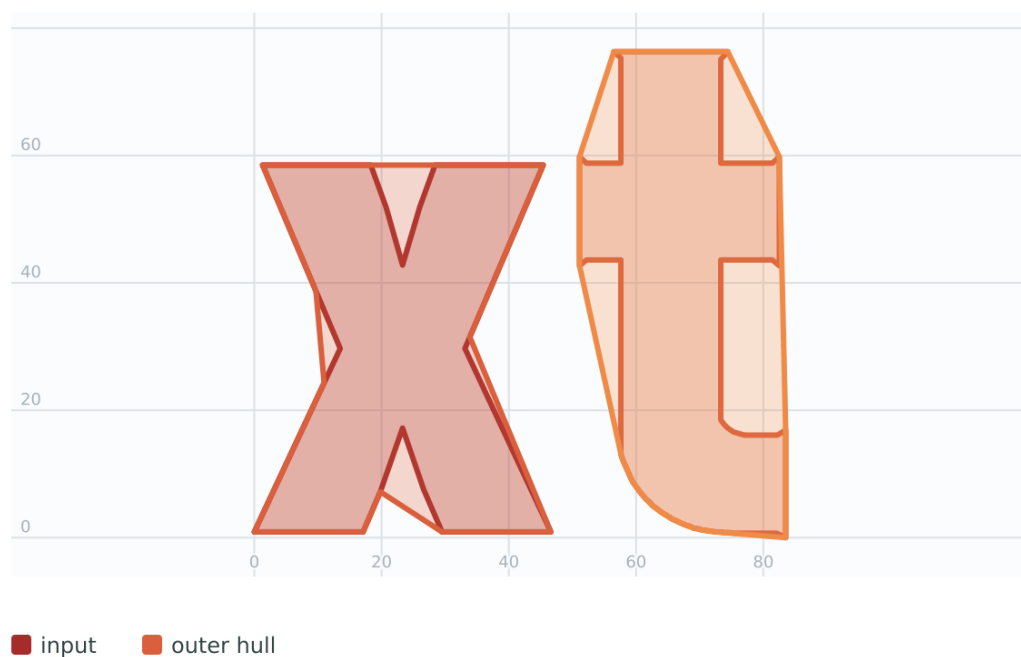


Outer hull simplification of a MultiPolygon, with segmentization.

Code

```
WITH letters AS (
  SELECT ST_Letters('xt') AS geom
)
SELECT geom AS input,
  ST_SimplifyPolygonHull(ST_Segmentize(geom, 2.0), 0.1) AS outer_hull
FROM letters;
```

Figure



相关信息

[ST_ConvexHull](#), [ST_SimplifyVW](#), [ST_ConcaveHull](#), [ST_Segmentize](#)

7.14.26 ST_SimplifyVW

ST_SimplifyVW — 使用 Visvalingam-Whyatt 算法返回几何图形的简化表示

Synopsis

geometry **ST_SimplifyVW**(geometry geom, float tolerance);

描述

Returns a simplified representation of a geometry using the [Visvalingam-Whyatt algorithm](#). The simplification tolerance is an area value, in squared 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.

该函数可用于任何图形的几何图形（包括 GeometryCollections），但只有点和多边形元素会被简化。线性几何的端点将被保留。



Note

返回的几何图形可能会失去其简单性（参看[ST_IsSimple](#)），拓扑可能不会被保留，并且多边形的结果可能是无效的（参看[ST_IsValid](#)）。使用[ST_SimplifyPreserveTopology](#)来保留拓扑并确保有效性。[ST_CoverageSimplify](#)也会保留拓扑和有效性。

This is a general-purpose simplification function. It may remove narrow polygon spikes when their effective area is below the tolerance, but it is not a dedicated spike-removal or polygon-repair operation. Use [ST_IsValid](#) to check the result and [ST_MakeValid](#) to repair invalid polygonal output if needed.

该函数不保留多边形之间的共享边界。如果需要保留共享边界，请使用[ST_CoverageSimplify](#)。

该函数处理 3D，第三维将影响结果。

可用性：2.2.0

示例

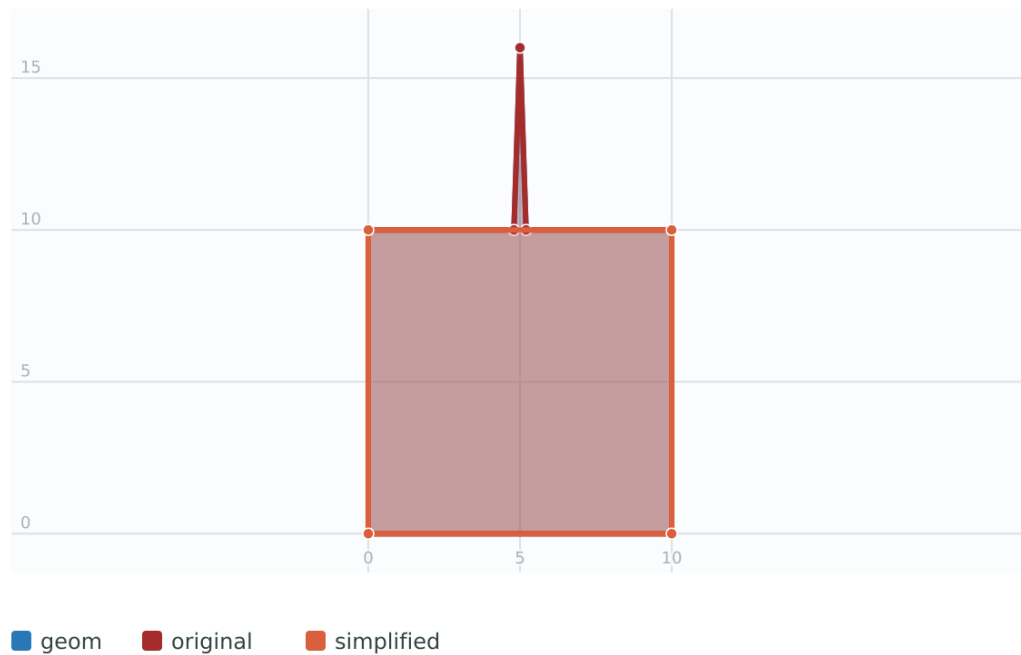
This polygon has a narrow spike whose effective area is small enough to be removed by a tolerance of 20 square units.

Code

```
SELECT geom AS original, ST_SimplifyVW(geom, 20) AS simplified
FROM (SELECT 'POLYGON((0 0,10 0,10 10,5.2 10,5 16,4.8 10,0 10,0 0))'::geometry AS geom) AS t;
```

Output

```
POLYGON((0 0,10 0,10 10,5.2 10,5 16,4.8 10,0 10,0 0)) | POLYGON((0 0,10 0,10 10,0 10,0 0))
```

| Figure

A LineString is simplified with a minimum-area tolerance of 30. For a linear simplification using **ST_Simplify**, a comparable tolerance would be a length in input-SRS units, so the value here is roughly that linear tolerance squared.

| Code

```
SELECT ST_SimplifyVW(geom, 30) simplified
FROM (SELECT 'LINESTRING(5 2,3 8,6 20,7 25,10 10)::geometry AS geom) AS t;
```

| Output

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

Figure



一条行化。

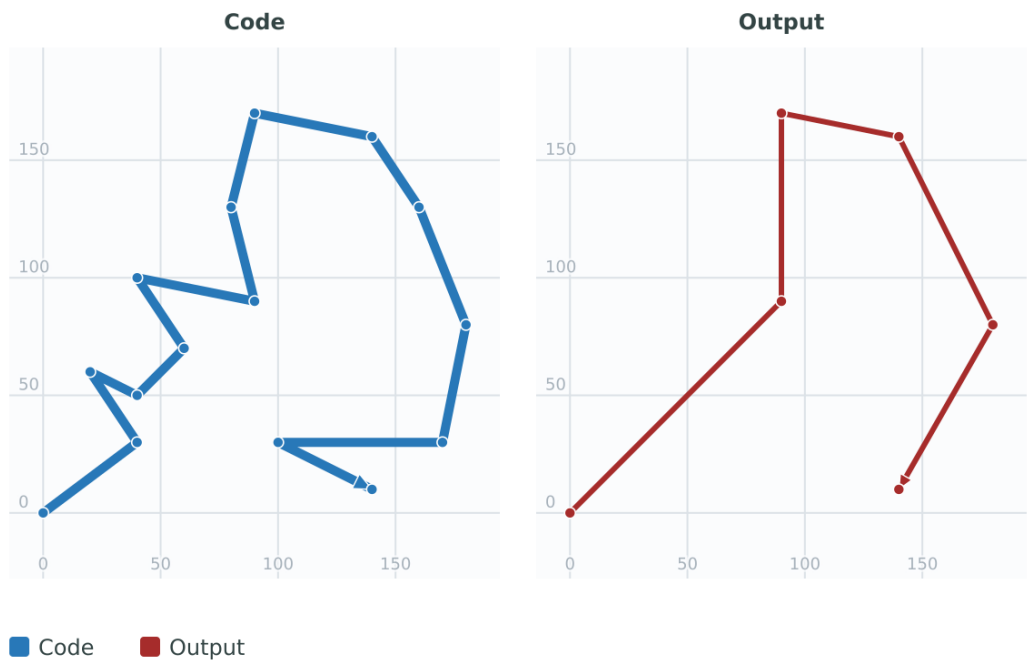
Code

```
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)'::geometry,  
1600);
```

Output

```
LINESTRING(10 10,100 100,180 150 170,190 90,150 20)
```

Figure



将多边形行简化。

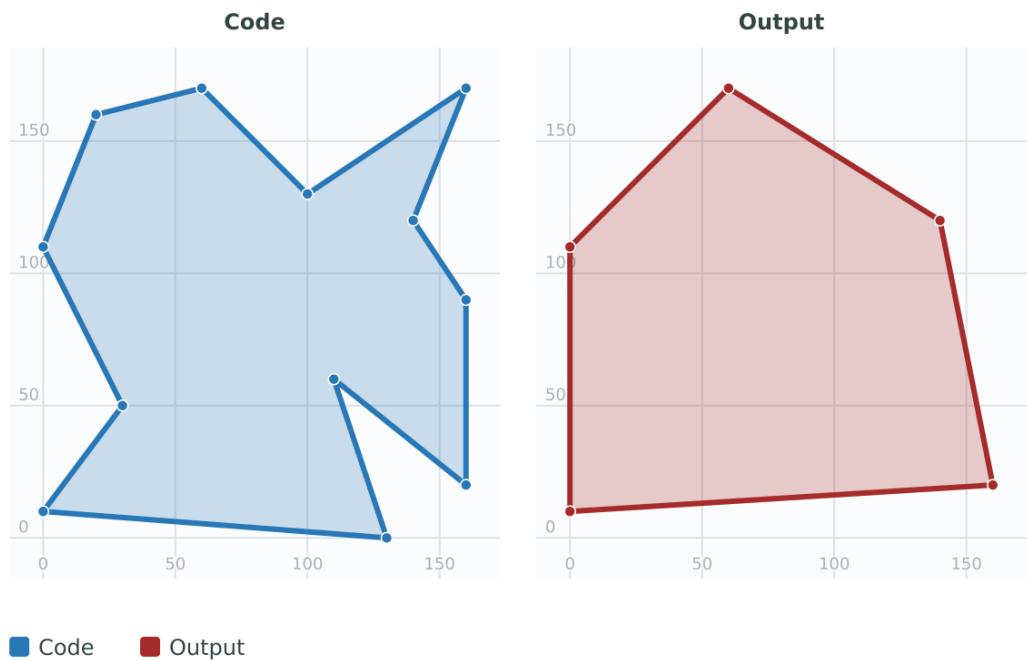
Code

```
SELECT ST_SimplifyVW('POLYGON ((20 20,50 60,20 120,40 170,80 180,120 140,180 180,160 130,180 100,180 30,130 70,150 10,20 20))'::geometry, 1600);
```

Output

```
POLYGON((20 20,20 120,80 180,160 130,180 30,20 20))
```

Figure



相关信息

[ST_SetEffectiveArea](#), [ST_Simplify](#), [ST_SimplifyPreserveTopology](#), [ST_CoverageSimplify](#), [Topology ST_Simpl](#)

7.14.27 ST_SetEffectiveArea

`ST_SetEffectiveArea` — 使用 Visvalingam-Whyatt 算法设置每个点的有效区域。

Synopsis

geometry **ST_SetEffectiveArea**(geometry geom, float threshold = 0, integer set_area = 1);

描述

使用 Visvalingam-Whyatt 算法设置每个点的有效区域。有效区域存储每个点的 M 值。如果使用可选的 “threshold” 参数，将返回简化的几何形状，包含有效区域大于或等于阈值的点。

当指定 `M`，此函数可用于服务器端简化。一种情况是使用零。在这种情况下，将返回完整的几何图形，其中有效区域作为 `M`，客户端可以使用它来快速简化。

图形上只会（多）和（多）多边形行某些操作，但您可以使用任何类型的几何体安全地使用它。由于简化是在逐个象的基上行，因此您可以将 `GeometryCollection` 提供此函数。



Note
注意，返回的几何图形可能会失去其简单性（参看 `ST_IsSimple`）



Note
注意，拓扑可能不会保留，并可能导致无效的几何图形。使用（参看 `ST_SimplifyPreserveTopology`）保留拓扑。



Note
输出几何图形将丢失 `M` 中的所有先前信息



Note
此功能处理 3D，三度会影响有效区域

可用性：2.2.0

示例

计算 `LineString` 的有效区域。因为我使用零，所以返回输入几何体中的所有点。

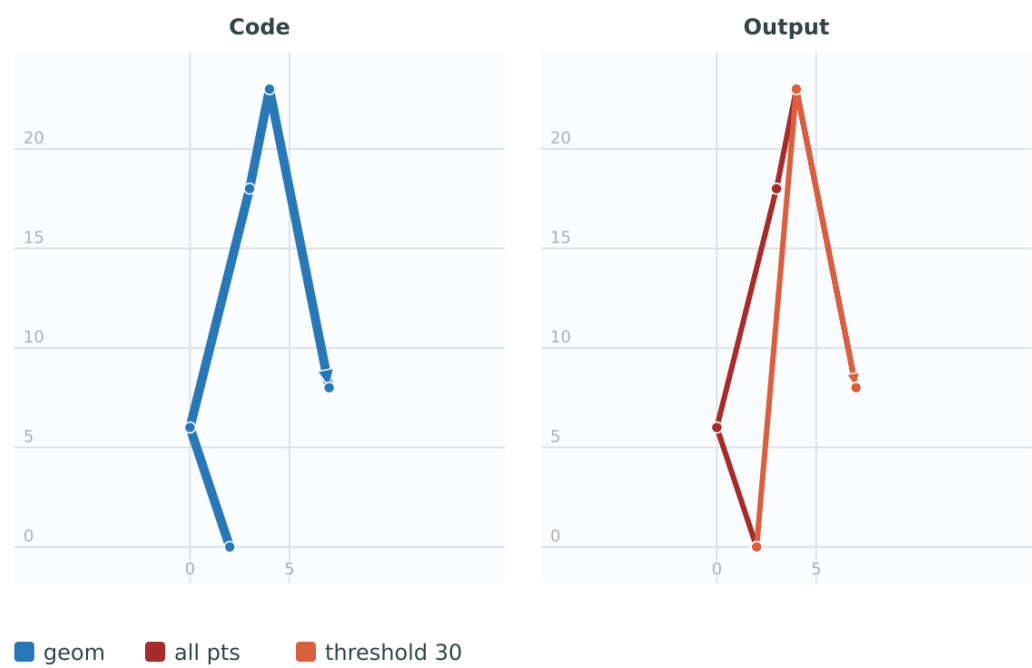
Code

```
SELECT ST_SetEffectiveArea(geom) AS all_pts,
       ST_SetEffectiveArea(geom, 30) AS threshold_30
FROM (SELECT 'LINESTRING(5 2,3 8,6 20,7 25,10 10)::geometry geom) As foo;
```

Output

```
-[ RECORD 1 ]-----
all_pts      | LINESTRING M (5 2 3.4e+38,3 8 29,6 20 1.5,7 25 49.5,10 10 3.4e+38)
threshold_30 | LINESTRING M (5 2 3.4e+38,7 25 49.5,10 10 3.4e+38)
```


Figure



相关信息

ST_SimplifyVW

7.14.28 ST_TriangulatePolygon

ST_TriangulatePolygon — 计算多面形的约束 Delaunay 三角剖分

Synopsis

geometry ST_TriangulatePolygon(geometry geom);

描述

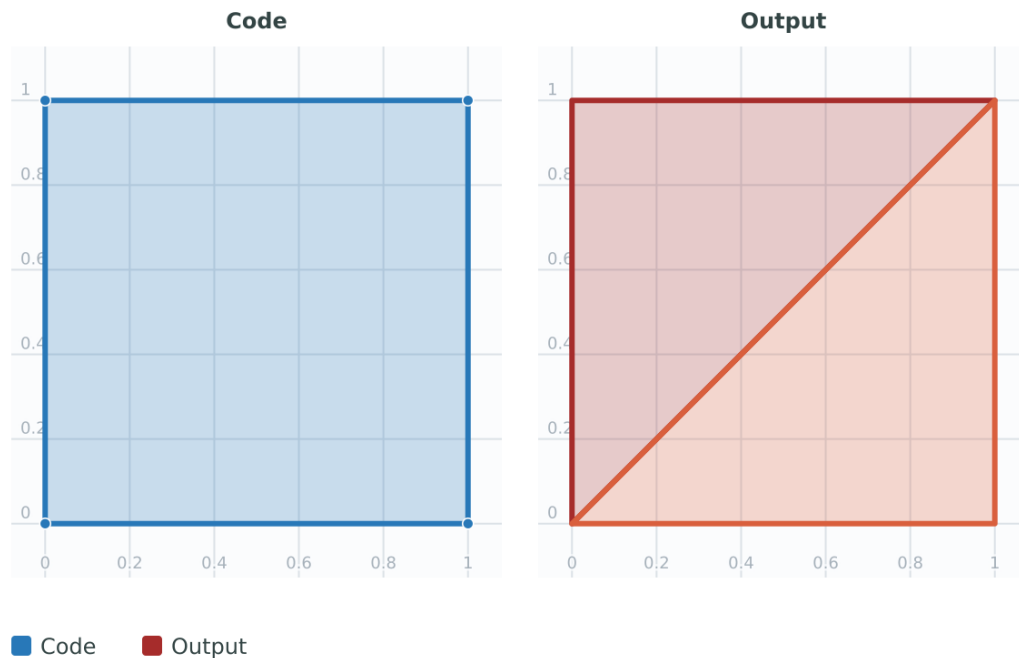
计算多面形的约束 Delaunay 三角剖分。支持孔和多面形。
多面形的“约束 Delaunay 三角剖分”是由多面形的点形成的一三角形，并精确地覆盖它，并且在所有可能的三角剖分上具有最大内角。它提供了多面形的“最佳质量”三角质量。
可用性：3.3.0。
需要 GEOS >= 3.11.0。

示例

正方形的三角剖分。

Code

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

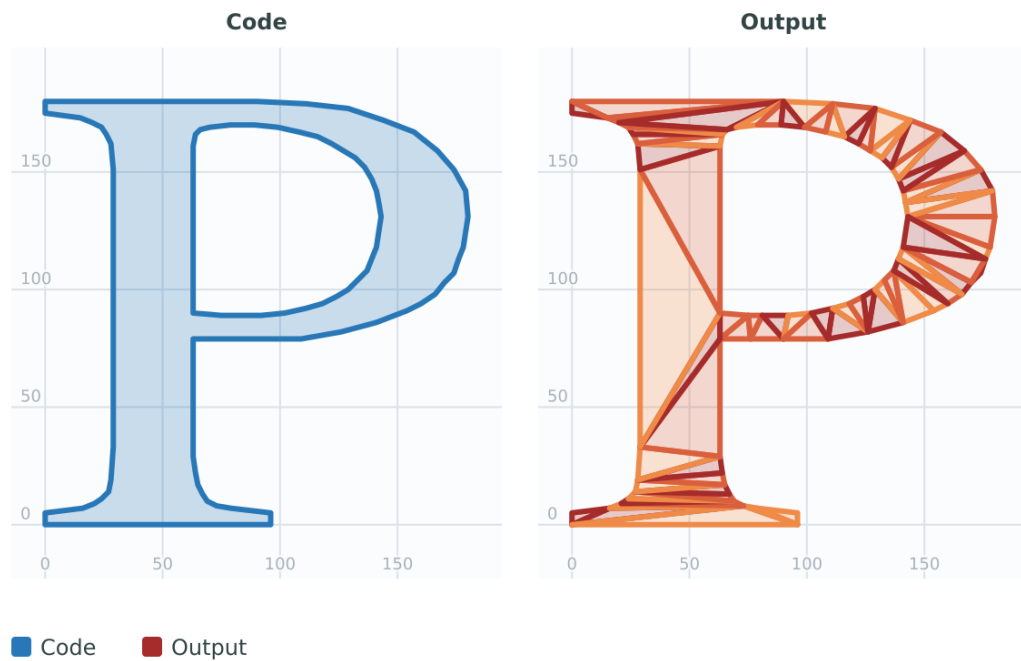
Figure

字母 P 的三角剖分。

Code

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

Figure

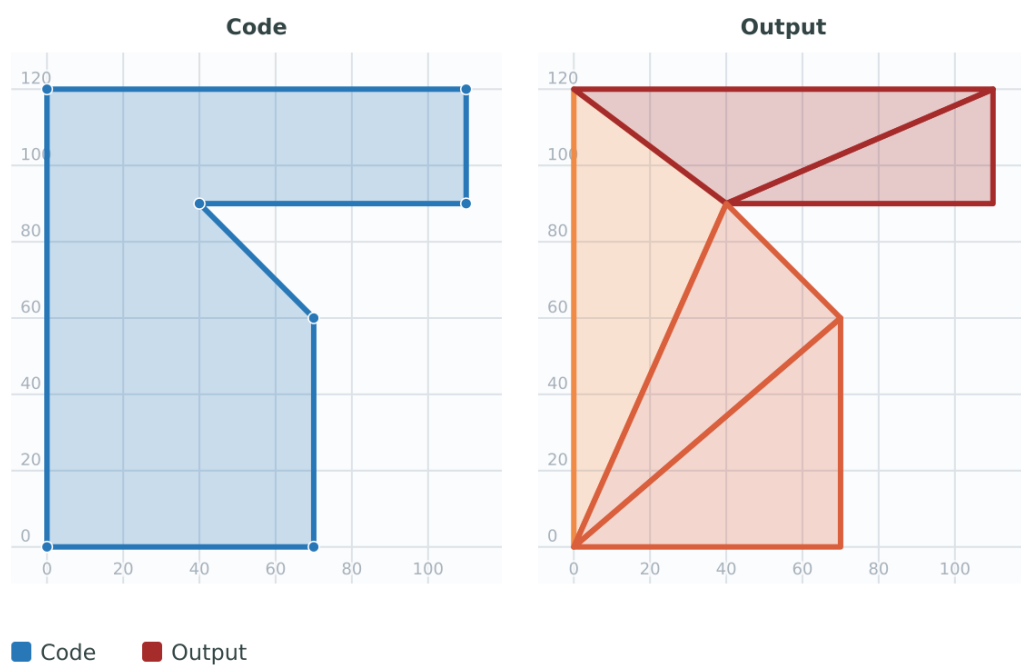


Triangulation of the polygon used in the [CG_Tessellate](#) example.

Code

```
SELECT ST_TriangulatePolygon('POLYGON (( 10 190,10 70,80 70,80 130,50 160,120 160,120 190,10 190 ))'::geometry
);
```

Figure



相关信息

[CG_ConstrainedDelaunayTriangles](#), [ST_DelaunayTriangles](#), [CG_Tessellate](#)

7.14.29 ST_VoronoiLines

ST_VoronoiLines — 返回几何体点集的 Voronoi 图边界。

Synopsis

```
geometry ST_VoronoiLines( geometry geom , float8 tolerance = 0.0 , geometry extend_to = NULL );
```

描述

根据提供的几何体点集计算二值 Voronoi 图，并将图中元格之图的边界作为 MultiLineString 返回。如果输入几何体为 null，则返回 null。如果输入几何体包含一个点，则返回一个空几何体集合。如果 extend_to 最小外接矩形的面为零，则返回空几何集合。

可选项参数：

- **tolerance**：点被图相等的距离。通提供非零容差距离可以提高算法的健壮性。（默认 = 0.0）
- **extend_to**：如果存在，表将图展以覆盖所提供几何体的最小外接矩形，除非小于默认最小外接矩形（默认 = NULL，默认最小外接矩形是输入图展 50% 的图界框）。

该函数是由 GEOS 模块行的。

可用性：2.3.0

示例

Voronoi diagram lines with a tolerance of 30 units.

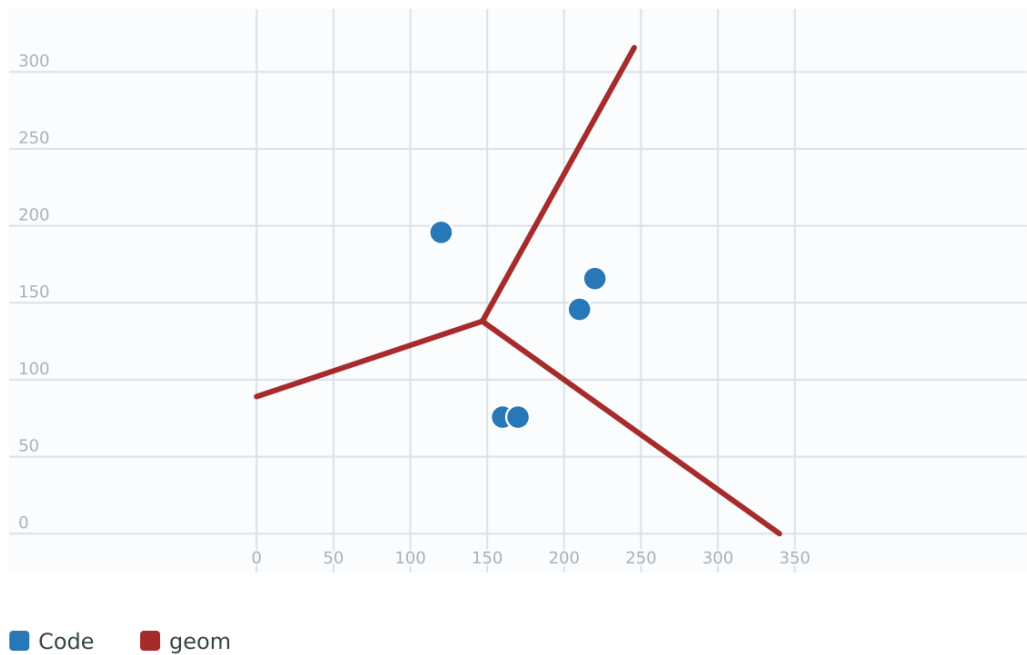
Code

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

Output

```
MULTILINESTRING((135.555555555556 270,36.8181818181818 92.2727272727273),(36.8181818181818 92.2727272727273,-110 43.3333333333333),(230 -45.7142857142858,36.8181818181818 92.2727272727273))
```

Figure



相关信息

[ST_DelaunayTriangles](#), [ST_VoronoiPolygons](#)

7.14.30 ST_VoronoiPolygons

ST_VoronoiPolygons — 返回几何体点的 Voronoi 的元格。

Synopsis

geometry **ST_VoronoiPolygons**(geometry geom , float8 tolerance = 0.0 , geometry extend_to = NULL);

描述

根据提供的几何形的点计算二 Voronoi 。如果是 POLYGON 的 GEOMETRYCOLLECTION，其覆盖的包大于入点的范。如果入几何形 null，返回 null。如果入几何形包含一个点，返回一个空几何形集合。如果 extend_to 最小外接矩形的面零，返回空几何集合。

可参数：

- tolerance：点被相等的距离。通提供非零容差距离可以提高算法的健性。（默 = 0.0）
- extend_to：如果存在，表将展以覆盖所提供几何形的最小外接矩形，除非小于默最小外接矩形（默 = NULL，默最小外接矩形是入展 50% 的界框）。

个函数是由 GEOS 模行的。

可用性：2.3.0

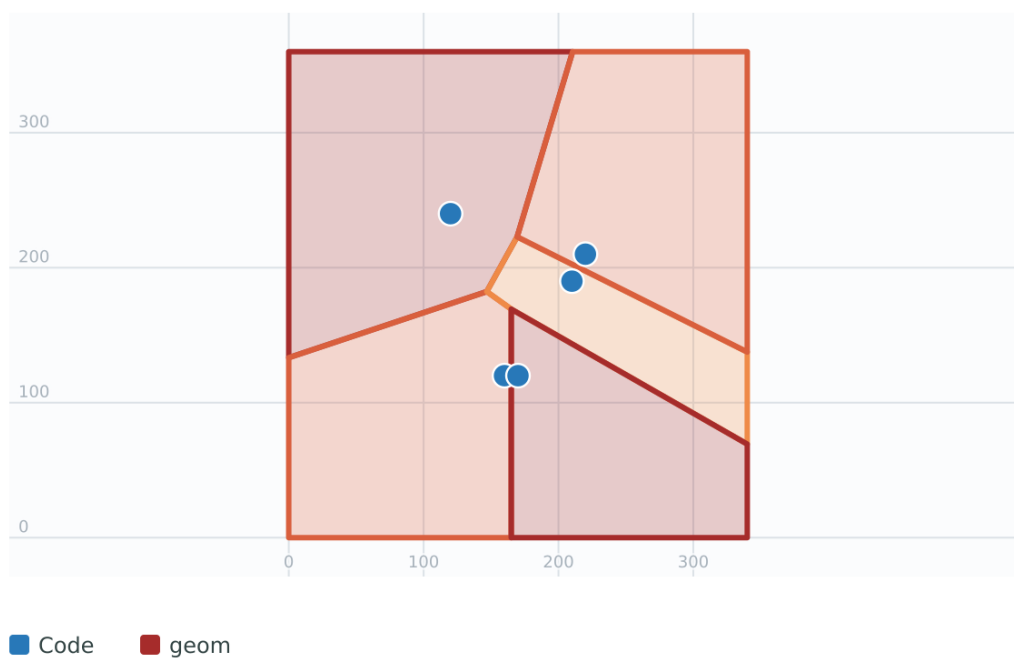
示例

Points overlaid on top of the Voronoi diagram.

| Code

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

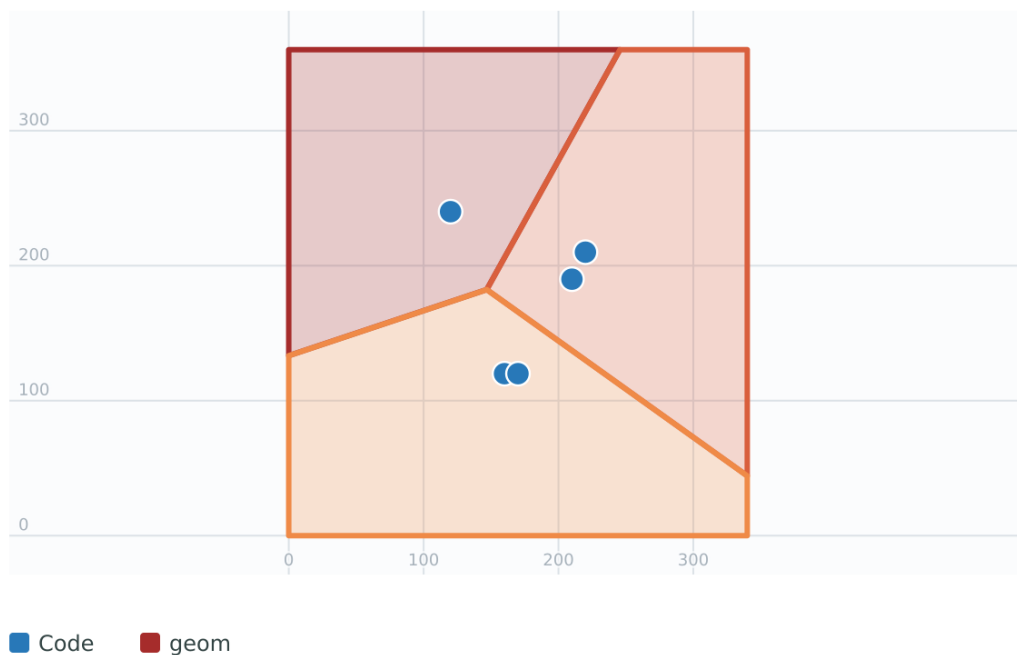
| Figure



Voronoi diagram with a tolerance of 30 units.

| Code

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

Figure

相关信息

[ST_DelaunayTriangles](#), [ST_VoronoiLines](#)

7.15 覆盖范围

7.15.1 ST_CoverageInvalidEdges

`ST_CoverageInvalidEdges` — 用于查找多边形无法形成有效覆盖范围的位置的窗口函数。

Synopsis

```
geometry ST_CoverageInvalidEdges(geometry winset geom, float8 tolerance = 0);
```

描述

一个窗口函数，用于检查窗口分区中的多边形是否形成有效的多边形覆盖范围。它返回指示器，指示每个多边形中无效（如果有）的位置。

如果满足以下条件，则一个有效多边形是有效的覆盖范围：

- **Non-overlapping**-多边形不重叠（它们的内部不相交）
- **Edge-Matched**-沿共享边的点相同

作窗口函数，每个输入多边形返回一个值。对于违反一个或多个有效性条件的多边形，返回值是包含有错误的MULTILINESTRING。覆盖范围有效的面返回 NULL。非多边形或空几何图形也会生成 NULL 值。

只要周围的多边形匹配，某些条件就允许有效的覆盖范围包含孔（多边形之间的间隙）。然而，非常窄的间隙通常是不希望的。如果使用非零距离指定公差参数，形成窄间隙的边也将被返回无效。

正在覆盖范围有效性的多边形也必须是有效的几何图形。可以使用 `ST_IsValid` 行。

可用性：3.4.0

需要 GEOS >= 3.12.0

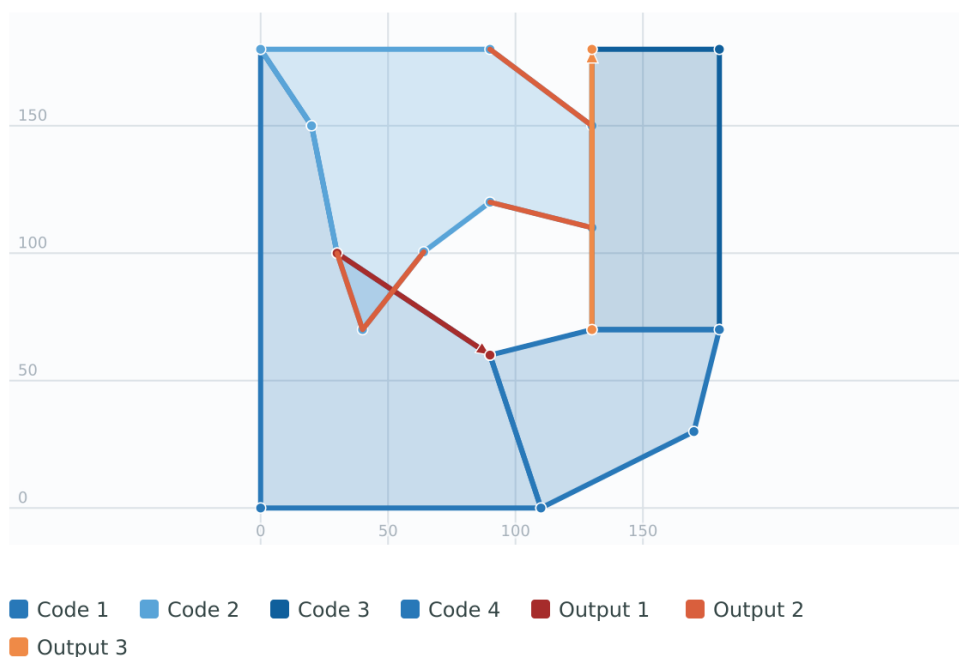
示例

Invalid edges caused by overlap and non-matching vertices.

Code

```
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_CoverageInvalidEdges(geom) OVER ()
FROM coverage;
```

Figure



This example tests the entire table for coverage validity.

Code

```
SELECT true = ALL (
  SELECT ST_CoverageInvalidEdges(geom) OVER () IS NULL
  FROM coverage
);
```


相关信息

[ST_IsValid](#), [ST_CoverageUnion](#), [ST_CoverageClean](#), [ST_CoverageSimplify](#)

7.15.2 ST_CoverageClean

`ST_CoverageClean` — 将非规范化输入算出一个规范的多边形覆盖结果（边界匹配、无重叠、无间隙）。

Synopsis

```
geometry ST_CoverageClean(geometry winset geom, float8 gapMaximumWidth = 0, float8 snappingDistance = -1, text overlapMergeStrategy = 'MERGE_LONGEST_BORDER');
```

描述

一个窗口函数，用于整理有效的面状几何的边界，使其形成干净的覆盖（clean coverage）。清理过程包括：

- 将点和行靠拢（snapping），消除微小偏差，确保共有点一致、拓扑正确
- 将重叠区域合并到上父 polygon 中
- 把狭窄的间隙并入相邻的 polygon 中

`gapMaximumWidth` 控制某些多边形之间的空隙会被合并；程度 $\leq$ 的空隙会被并入相邻的 polygon。

`snappingDistance` 控制点和行的靠拢（snapping）。默认为 -1 会根据输入范围自动计算靠拢距离；置为 0.0 表示关闭靠拢。

`overlapMergeStrategy` 用来指定重叠区域如何合并到父 polygon 中：

- `MERGE_LONGEST_BORDER` - 合并到与其共有最长边的 polygon
- `MERGE_MAX_AREA` - 合并到面积最大的 polygon
- `MERGE_MIN_AREA` - 合并到面积最小的 polygon
- `MERGE_MIN_INDEX` - 合并到索引最小的 polygon（按输入序定）

结果是一个干净的面覆盖（polygonal coverage），能通过 [ST_CoverageInvalidEdges](#) 的校验，并可作后续覆盖处理函数的输入。

Note



便于确定合适的最大间隙程度，可以先将 `gapMaximumWidth = > 0` 来先行清理，然后使用 [ST_CoverageUnion](#) 清理后的覆盖行合并。合并结果中的“洞”也就是原始数据中的间隙。接着，可以将这些洞提取为 polygon，并对每个 polygon 行 [ST_MaximumInscribedCircle](#) 来测量它的最大内切圆；间隙度 = 计算得到的半径 $\times 2$ 。

可用性: 3.6.0 - 需要 GEOS $\geq$ 3.14.0

示例

Clean a coverage containing overlaps and narrow gaps, and compare the cleaned polygons with the original invalid edges.

Code

```
WITH coverage(id, geom) AS (VALUES
  (1, 'POLYGON ((10 190,30 160,27 134.5,40 110,122 47,120 10,10 10,10 190))'::geometry),
  (2, 'POLYGON ((150 190,10 190,30 160,50 140,40 110,50 80,130 70,135 111,140 130,140 160,150 190))'::geometry),
  (3, 'POLYGON ((140 190,190 190,190 80,140 80,140 190))'::geometry),
  (4, 'POLYGON ((190 10,120 10,97 77,160 90,170 70,190 80,190 10))'::geometry)
), processed AS (
  SELECT id,
    ST_CoverageClean(geom, 1) OVER () AS cleaned,
    ST_CoverageInvalidEdges(geom) OVER () AS invalid_edges
  FROM coverage
)
SELECT ST_Collect(cleaned ORDER BY id) AS cleaned,
  ST_Collect(invalid_edges ORDER BY id) AS invalid_edges
FROM processed;
```

Figure



相关信息

[ST_CoverageInvalidEdges](#), [ST_CoverageUnion](#), [ST_CoverageSimplify](#), [ST_MaximumInscribedCircle](#)

7.15.3 ST_CoverageSimplify

ST_CoverageSimplify — 简化多边形覆盖范围的窗口函数。

Synopsis

geometry **ST_CoverageSimplify**(geometry winset geom, float8 tolerance, boolean simplifyBoundary = true);

描述

一种窗函数，可简化多边形覆盖范围内的多边形。简化保留了覆盖拓扑。这意味着简化的输出多边形沿着共享边界是一致的，并且仍然形成有效的覆盖范围。

简化使用了 [Visvalingam-Whyatt](#) 算法的格式。公差参数以距离单位，大致等于待简化的三角形面积的平方根。

要简化 Coverage 的“内部”边界（由多个多边形共享的边界），请将 *SimplifyBoundary* 参数置为 false。

Use this function to simplify adjacent polygon layers when shared boundaries must remain edge-matched. For a single geometry where only self-intersection and ring validity matter, see [ST_SimplifyPreserve](#)



Note

如果输入不是有效的覆盖范围，输出中可能会出现意外的阴影（例如边界交叉点或看似共享的分离边界）。使用 [ST_CoverageInvalidEdges](#) 确定覆盖范围是否有效。

可用性 : 3.4.0

需要 GEOS >= 3.12.0

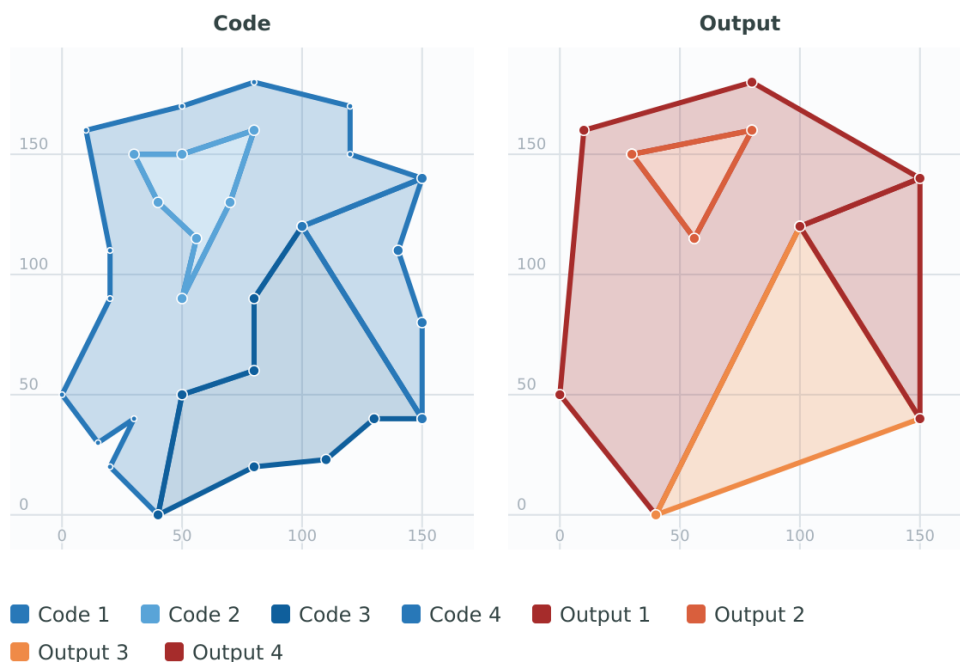
示例

Simplify a polygonal coverage while preserving shared boundaries.

Code

```
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_CoverageSimplify(geom, 30) OVER ()
FROM coverage;
```

Figure



相关信息

[ST_CoverageInvalidEdges](#), [ST_CoverageUnion](#), [ST_CoverageClean](#)

7.15.4 ST_CoverageEdges

`ST_CoverageEdges` — Computes the unique edges of a polygonal coverage.

Synopsis

```
geometry ST_CoverageEdges(geometry geom, integer edgetype = 0);
geometry ST_CoverageEdges(geometry[] geoms, integer edgetype = 0);
```

描述

Returns a `MultiLineString` representing the unique edges of a polygonal coverage. A polygonal coverage is a set of non-overlapping polygons where adjacent polygons have matching vertices along shared edges.

The *edgetype* parameter can be used to select which edges are returned:

- **0 (ALL)** - all unique edges (default)
- **1 (EXTERIOR)** - only exterior edges (non-shared)
- **2 (INTERIOR)** - only interior edges (shared)

Availability: 3.7.0

Requires GEOS >= 3.15.0

示例

This example returns all unique edges of a coverage.

Code

```
WITH coverage(geom) AS (
  VALUES (
    'GEOMETRYCOLLECTION(
      POLYGON((0 0,10 0,10 10,0 10,0 0)),
      POLYGON((10 0,20 0,20 10,10 10,10 0))
    )'::geometry
  )
)
SELECT geom AS input_coverage,
       ST_CoverageEdges(geom) AS all_edges
FROM coverage;
```

Output

```
-[ RECORD 1 ]-----
input_coverage | GEOMETRYCOLLECTION(POLYGON((0 0,10 0,10 10,0 10,0 0)),POLYGON((10 0,20 0,20 10,10 10,10 0)))
all_edges      | MULTILINESTRING((10 0,10 10),(10 10,0 10,0 0,10 0),(10 0,20 0,20 10,10 10))
```

This example returns only the interior shared edges of a coverage.

Code

```
WITH coverage(geom) AS (
  VALUES (
    'GEOMETRYCOLLECTION(
      POLYGON((0 0,10 0,10 10,0 10,0 0)),
      POLYGON((10 0,20 0,20 10,10 10,10 0))
    )'::geometry
  )
)
SELECT geom AS input_coverage,
       ST_CoverageEdges(geom, 2) AS shared_edges
FROM coverage;
```

Output

```
-[ RECORD 1 ]-----
input_coverage | GEOMETRYCOLLECTION(POLYGON((0 0,10 0,10 10,0 10,0 0)),POLYGON((10 0,20 0,20 10,10 10,10 0)))
shared_edges   | MULTILINESTRING((10 0,10 10))
```

相关信息

[ST_CoverageInvalidEdges](#), [ST_CoverageSimplify](#), [ST_CoverageUnion](#)

7.15.5 ST_CoverageUnion

`ST_CoverageUnion` — 通过去除共享来算形成覆盖范围的一多形的并集。

Synopsis

geometry **ST_CoverageUnion**(geometry set geom);

描述

将一个多边形合起来形成多边形覆盖范围的聚合函数。它是覆盖与覆盖范围相同的区域的多边形几何体。该函数生成与 **ST_Union** 相同的结果，但使用覆盖范围来更快地计算并集。



Note

如果输入不是有效的覆盖范围，输出中可能会出现意外的阴影（例如未合并或重叠的多边形）。使用 **ST_CoverageInvalidEdges** 确定覆盖范围是否有效。

可用性：3.4.0-需要 GEOS >= 3.8.0

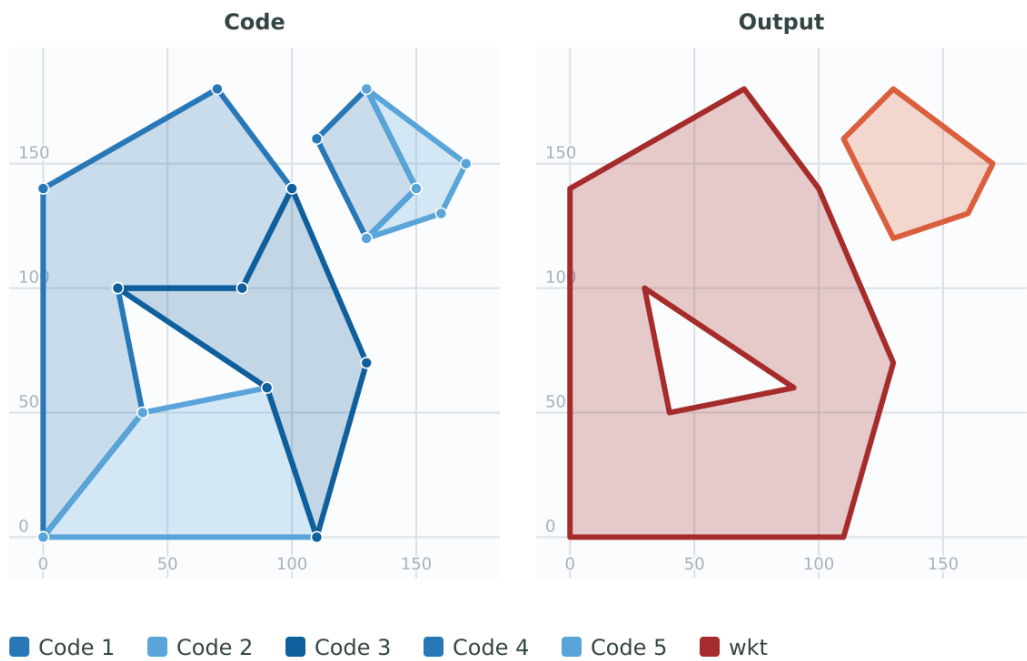
示例

Union a polygonal coverage.

Code

```
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_CoverageUnion(geom) AS wkt
FROM coverage;
```

Figure



相关信息

ST_CoverageInvalidEdges, **ST_CoverageSimplify**, **ST_CoverageClean**, **ST_Union**

7.16 仿射变换

7.16.1 ST_Affine

ST_Affine — 对几何体应用 3D 仿射变换。

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);
```

描述

对几何体应用 3D 仿射变换，一步完成平移、旋转、缩放等操作。

版本 1：应用

Code

```
ST_Affine(geom, a, b, c, d, e, f, g, h, i, xoff, yoff, zoff)
```

表示变换矩阵

Code

```
/ a b c xoff \
| d e f yoff |
| g h i zoff |
\ 0 0 0      1 /
```

并且点变换如下：

Code

```
x' = a*x + b*y + c*z + xoff
y' = d*x + e*y + f*z + yoff
z' = g*x + h*y + i*z + zoff
```

下面所有的平移/缩放函数都是通过仿射变换来表示的。

版本 2：对几何体应用 2d 仿射变换。应用

Code

```
ST_Affine(geom, a, b, d, e, xoff, yoff)
```

表示变换矩阵

Code

```
/ 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 /
\ 0 0 0      1 /
```

并且点变换如下：

Code

```
x' = a*x + b*y + xoff
y' = d*x + e*y + yoff
z' = z
```

此方法是上述 3D 方法的子情况。

增ⓧ功能：引入了 2.0.0 ⓧ多面体曲面、三角形和三角网的支持。

可用性：1.1.2。1.2.2 中名称从 Affine 更改ⓧ ST_Affine



Note
在 1.3.4 之前，此函数在与包含曲ⓧ的几何ⓧ形一起使用ⓧ崩ⓧ。此ⓧⓧ已在 1.3.4 及更高版本中得到ⓧ正

- ✓ ⓧ函数支持多面体曲面。
- ✓ 此函数支持三角形和不ⓧⓧ三角网面 (TIN)。
- ✓ ⓧ函数支持 3d 并且不会ⓧ失 z-index。
- ✓ 此方法支持ⓧ形字符串和曲ⓧ。

示例

Rotate a 3D line 180 degrees about the z axis. Note this is long-hand for doing ST_Rotate();

Code

```
SELECT ST_Affine(geom, cos(pi()), -sin(pi()), 0, sin(pi()), cos(pi()), 0, 0, 0, 1, 0, 0, 0) ←
  As using_affine,
  ST_Rotate(geom, pi()) As using_rotate
FROM (SELECT 'LINESTRING(1 2 3,1 4 3)::geometry As geom) As foo;
```

Figure



Rotate a 3D line 180 degrees in both the x and z axis.

Code

```
SELECT ST_Affine(geom, cos(pi()), -sin(pi()), 0, sin(pi()), cos(pi()), -sin(pi()), 0, sin(↵
    pi()), cos(pi()), 0, 0, 0)
FROM (SELECT 'LINESTRING(1 2 3,1 4 3)::geometry As geom) As foo;
```

Figure



相关信息

[ST_Rotate](#), [ST_Scale](#), [ST_Translate](#), [ST_TransScale](#)

7.16.2 ST_Rotate

ST_Rotate — 绕原点旋转几何体。

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);
```

描述

绕原点逆时针旋转几何体 `rotRadians`。旋转原点可以指定为 POINT 几何形状，也可以指定 `x` 和 `y` 坐标。如果未指定原点，几何形状将绕 POINT(0 0) 旋转。

增强功能：引入了 2.0.0 对多面体曲面、三角形和三角网的支持。

增强：2.0.0 添加了用于指定旋转原点的附加参数。

可用性 : 1.1.2。1.2.2 中名称从 Rotate 更改为 ST_Rotate

- ✓ 函数支持 3d 并且不会丢失 z-index。
- ✓ 此方法支持几何字符串和曲线。
- ✓ 函数支持多面体曲面。
- ✓ 此函数支持三角形和不规则三角网面 (TIN)。

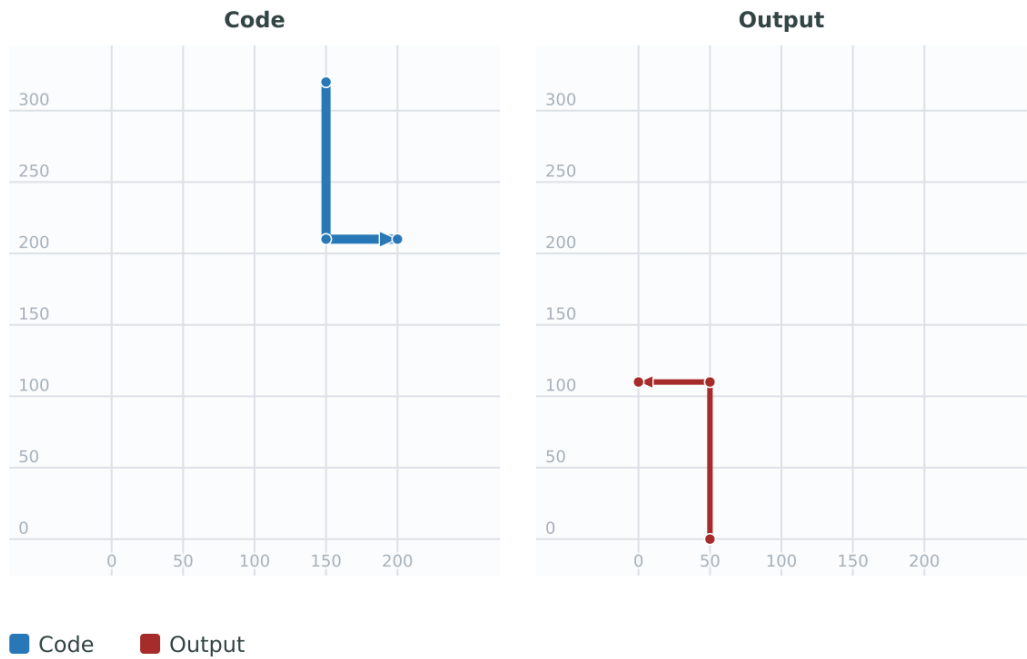
示例

This example rotates a geometry by 180 degrees.

Code

```
SELECT ST_Rotate('LINESTRING (50 160,50 50,100 50)::geometry, pi());
```

Figure

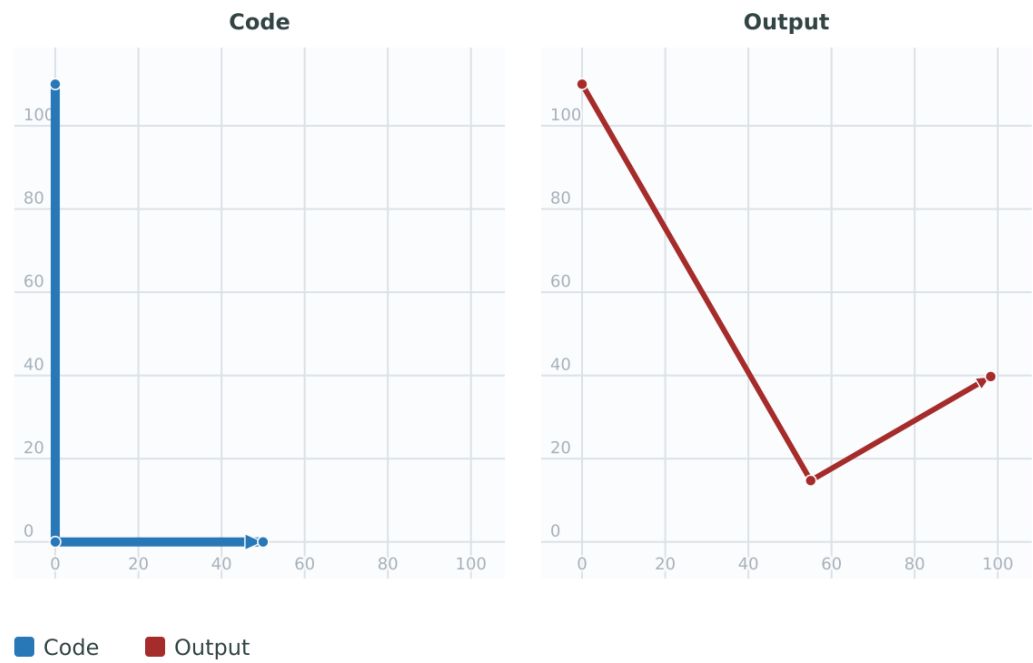


This example rotates a geometry 30 degrees counter-clockwise around x=50, y=160.

Code

```
SELECT ST_Rotate('LINESTRING (50 160,50 50,100 50)::geometry, pi()/6, 50, 160);
```

Figure



This example rotates a geometry 60 degrees clockwise around its centroid.

Code

```
SELECT ST_Rotate(geom, -pi()/3, ST_Centroid(geom))
FROM (SELECT 'LINESTRING (50 160,50 50,100 50)>:::geometry AS geom) AS foo;
```

Figure



相关信息

[ST_Affine](#), [ST_RotateX](#), [ST_RotateY](#), [ST_RotateZ](#)

7.16.3 ST_RotateX

ST_RotateX — 绕 X 轴旋转几何体。

Synopsis

geometry **ST_RotateX**(geometry geomA, float rotRadians);

描述

绕 X 轴旋转几何体 geomA - rotRadians。



Note

ST_RotateX(geomA, rotRadians) 是 ST_Affine(geomA, 1, 0, 0, 0, cos(rotRadians), -sin(rotRadians), 0, sin(rotRadians), cos(rotRadians), 0, 0, 0) 的简写。

新增功能：引入了 2.0.0 对多面体曲面、三角形和三角网的支持。

可用性：1.1.2。1.2.2 中名称从 RotateX 更改为 ST_RotateX



该函数支持多面体曲面。



该函数支持 3d 并且不会丢失 z-index。



此函数支持三角形和不规则三角网面 (TIN)。

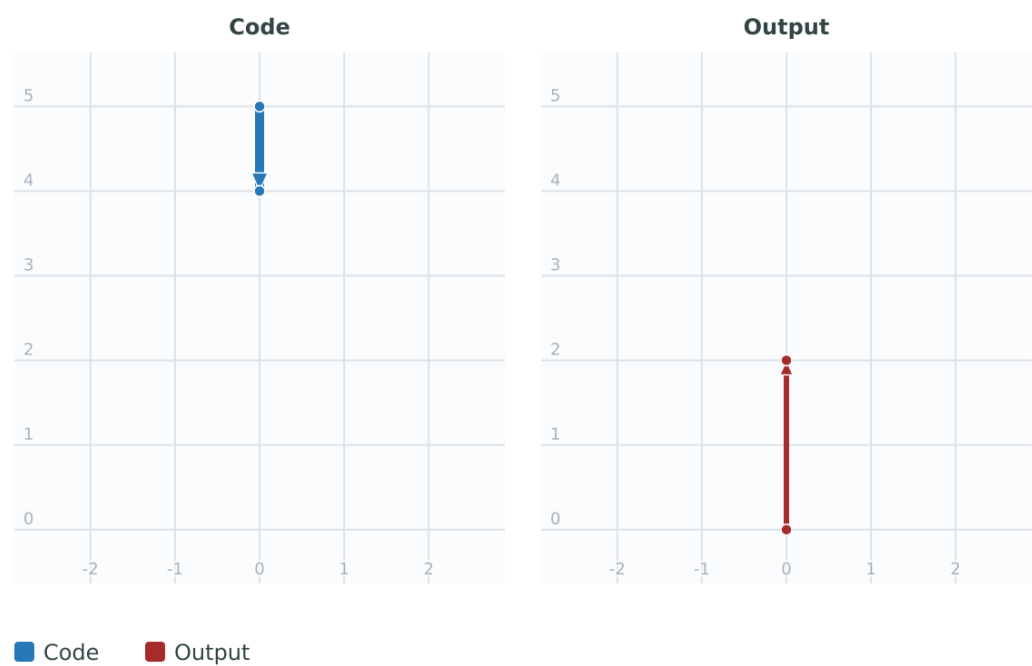
示例

This example rotates a line 90 degrees around the X axis.

Code

```
SELECT ST_RotateX('LINESTRING(1 2 3,1 1 1)', pi()/2);
```

Figure



相关信息

[ST_Affine](#), [ST_RotateY](#), [ST_RotateZ](#)

7.16.4 ST_RotateY

ST_RotateY — 绕 Y 轴旋几何体。

Synopsis

geometry **ST_RotateY**(geometry geomA, float rotRadians);

描述

绕 y 轴旋几何体 geomA - rotRadians。



Note ST_RotateY(geomA, rotRadians) 是 ST_Affine(geomA, cos(rotRadians), 0, sin(rotRadians), 0, 1, 0, -sin(rotRadians), 0, cos(rotRadians), 0, 0, 0) 的简写。

可用性：1.1.2。1.2.2 中名称从 RotateY 更改为 ST_RotateY
新增功能：引入了 2.0.0 对多面体曲面、三角形和三角网的支持。

- ✓ 该函数支持多面体曲面。
- ✓ 该函数支持 3d 并且不会丢失 z-index。
- ✓ 此函数支持三角形和不规则三角网面 (TIN)。

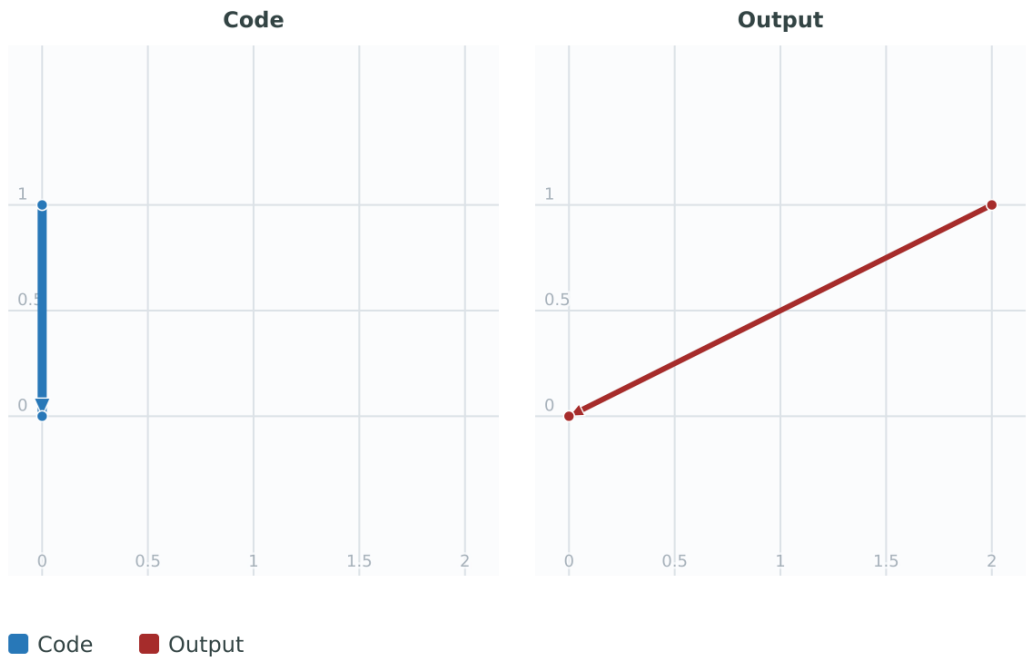
示例

This example rotates a line 90 degrees around the Y axis.

Code

```
SELECT ST_RotateY('LINESTRING(1 2 3,1 1 1)', pi()/2);
```

Figure



相关信息

[ST_Affine](#), [ST_RotateX](#), [ST_RotateZ](#)

7.16.5 ST_RotateZ

ST_RotateZ — 绕 Z 轴旋转几何体。

Synopsis

geometry **ST_RotateZ**(geometry geomA, float rotRadians);

描述

绕 Z 轴旋转几何体 geomA - rotRadians。



Note
这是 ST_Rotate 的同义词



Note
ST_RotateZ(geomA, rotRadians) 是 SELECT ST_Affine(geomA, cos(rotRadians), -sin(rotRadians), 0, sin(rotRadians), cos(rotRadians), 0, 0, 0, 1, 0, 0, 0) 的简写。

新增功能：引入了 2.0.0 对多面体曲面、三角形和三角网的支持。
可用性：1.1.2。1.2.2 中名称从 RotateZ 更改为 ST_RotateZ



Note
在 1.3.4 之前，此函数在与包含曲线的几何图形一起使用时会崩溃。此问题已在 1.3.4 及更高版本中得到修正

- ✓ 该函数支持 3d 并且不会丢失 z-index。
- ✓ 此方法支持几何字符串和曲线。
- ✓ 该函数支持多面体曲面。
- ✓ 此函数支持三角形和不规则三角网面 (TIN)。

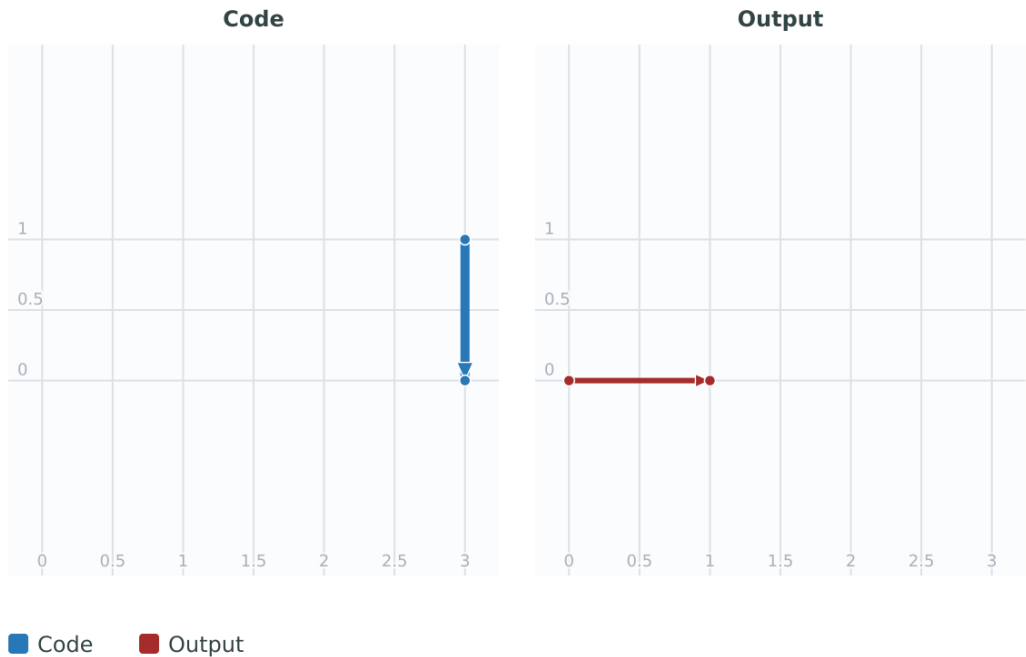
示例

This example rotates a line 90 degrees around the Z axis.

Code

```
SELECT ST_RotateZ('LINESTRING(1 2 3,1 1 1)', pi()/2);
```

Figure

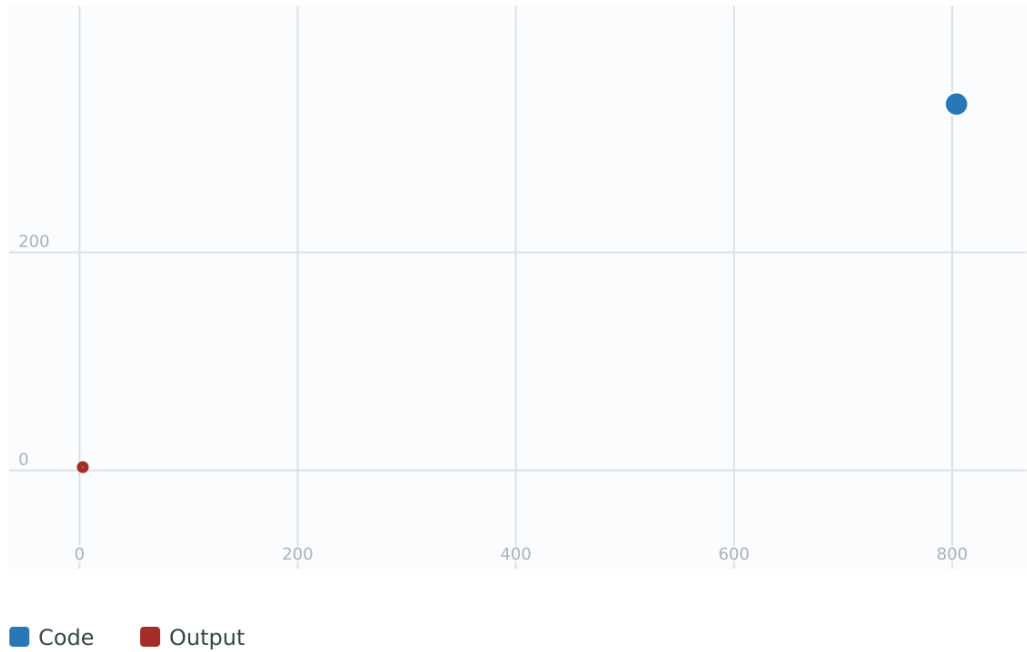


This example rotates a curved circle around the Z axis.

Code

```
SELECT ST_RotateZ(geom, pi()/2)
FROM (SELECT ST_LineToCurve(ST_Buffer('POINT(234 567) '::geometry, 3)) As geom) As foo;
```

Figure



相关信息

[ST_Affine](#), [ST_RotateX](#), [ST_RotateY](#)

7.16.6 ST_Scale

ST_Scale — 按因子缩放几何形。

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);
```

描述

通常将坐与相的因子参数相乘，将几何体放到新的尺寸。
采用几何形作 **factor** 参数的版本允 2d、3dm、3dz 或 4d 点来所有支持的度置放因子。**factor** 点中缺少度相当于没有放相的度。
三几何体体允入放的“假原点”。允“就地放”，例如使用几何体的心作假原点。如果没有假原点，放是相于原点行的，因此所有坐都只是乘以比例因子。

Note

在 1.3.4 之前，此函数在与包含曲面的几何图形一起使用时崩溃。此问题已在 1.3.4 及更高版本中得到修正

可用性：1.1.0。

增强功能：引入了 2.0.0 对多面体曲面、三角形和三角网的支持。

增强：2.2.0 引入了缩放所有维度（**factor** 参数）的支持。

增强：2.5.0 引入了相对于本地原点（**origin** 参数）进行缩放的支持。

- ✓ 该函数支持多面体曲面。
- ✓ 该函数支持 3d 并且不会丢失 z-index。
- ✓ 此方法支持 WKT 字符串和曲面。
- ✓ 此函数支持三角形和不规则三角网面 (TIN)。
- ✓ 该功能支持 M 坐标。

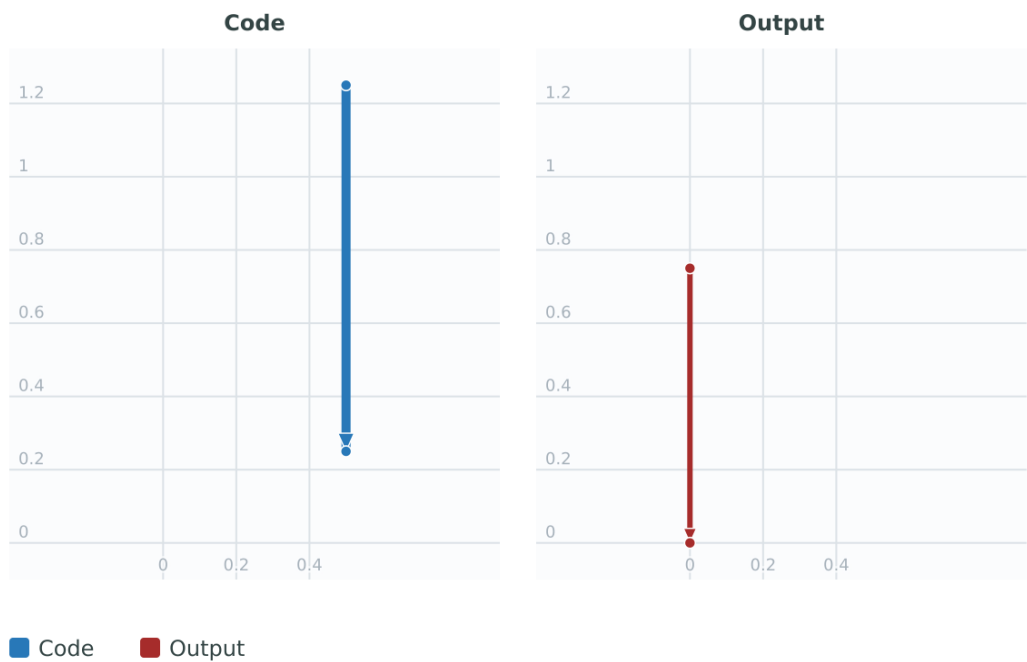
示例

This example scales X, Y, and Z values.

Code

```
SELECT ST_Scale('LINESTRING(1 2 3,1 1 1)', 0.5, 0.75, 0.8);
```

Figure

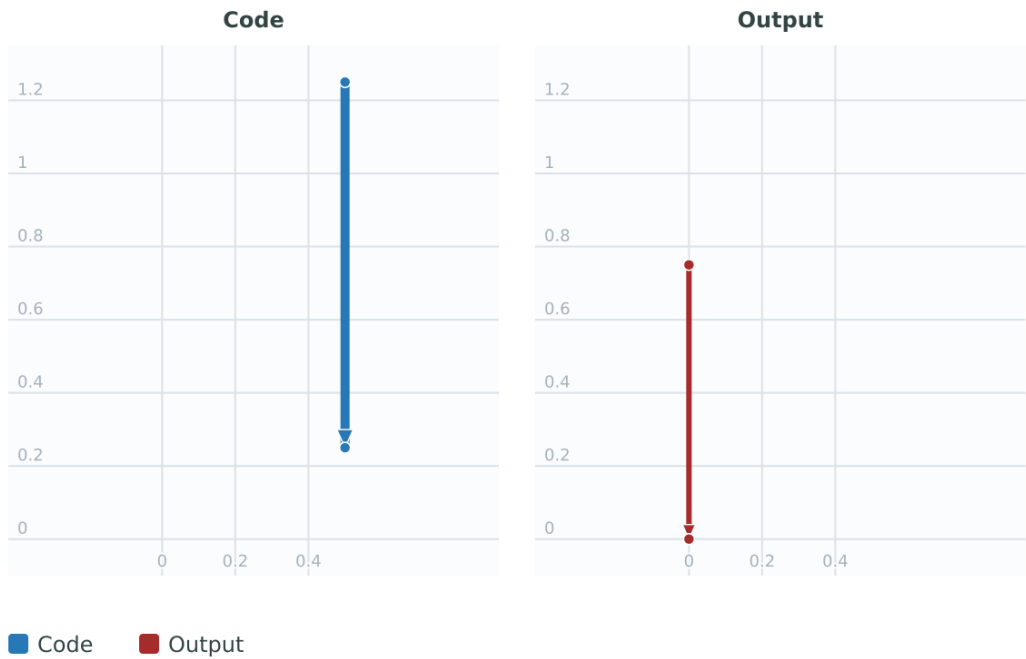


This example scales X and Y values.

Code

```
SELECT ST_Scale('LINESTRING(1 2 3,1 1 1)', 0.5, 0.75);
```

Figure

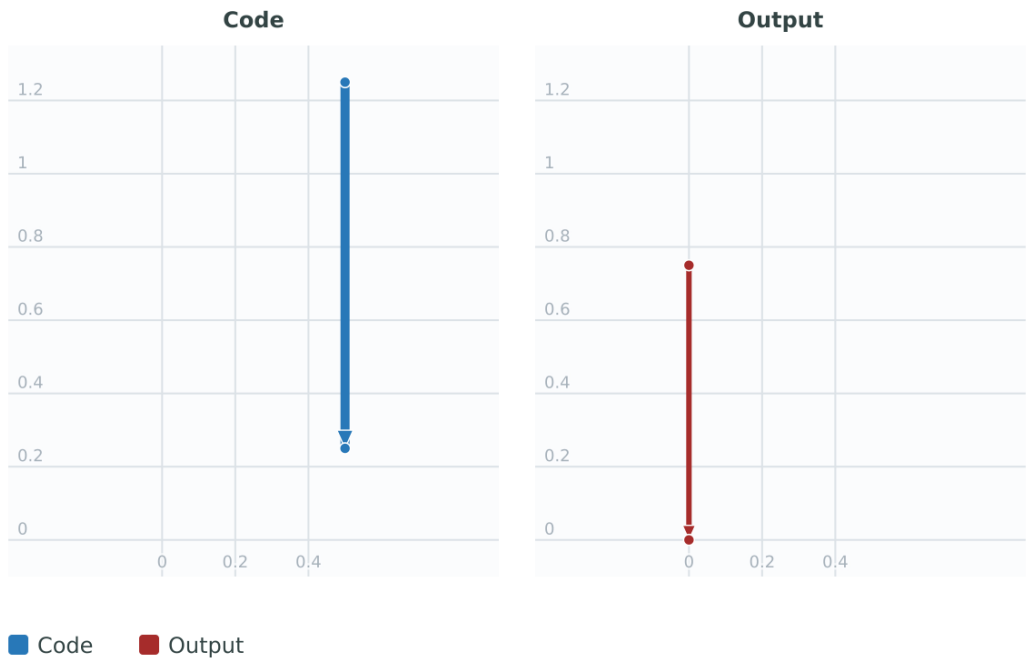


This example scales X, Y, Z, and M values.

Code

```
SELECT ST_Scale('LINESTRING(1 2 3 4,1 1 1 1)',  
  ST_MakePoint(0.5, 0.75, 2, -1));
```

Figure

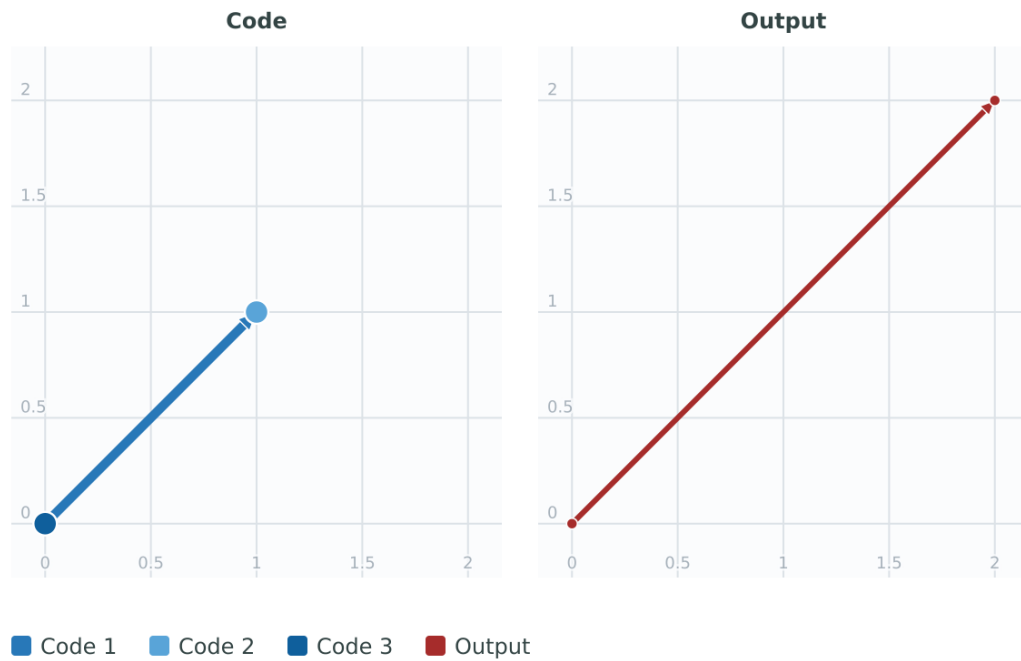


This example scales X and Y values using a false origin.

Code

```
SELECT ST_Scale('LINESTRING(1 1,2 2)', 'POINT(2 2)', 'POINT(1 1)::geometry');
```

Figure



This example constructs an ellipse centered at (10, 20), with X radius 8, Y radius 4, and rotated 30 degrees counter-clockwise. Scaling and rotation use the center as their origin, so the ellipse keeps both its position and SRID.

Code

```
WITH params AS (
  SELECT
    ST_SetSRID(ST_Point(10, 20), 3857) AS center,
    8.0 AS x_radius,
    4.0 AS y_radius,
    radians(30) AS angle
)
SELECT ST_Rotate(
  ST_Scale(
    ST_Buffer(center, 1, 'quad_segs=32'),
    ST_MakePoint(x_radius, y_radius),
    center
  ),
  angle,
  center
) AS ellipse
FROM params;
```

相关信息

[ST_Affine](#), [ST_Buffer](#), [ST_MakePoint](#), [ST_Rotate](#), [ST_TransScale](#)

7.16.7 ST_Translate

ST_Translate — 按指定的偏移量平移几何图形。

Synopsis

```
geometry ST_Translate(geometry g1, float deltax, float deltax);  
geometry ST_Translate(geometry g1, float deltax, float deltax, float deltax);
```

描述

Returns a new geometry whose coordinates are translated delta x,delta y,delta z units. Units are based on the units defined in spatial reference (SRID) for this geometry. M coordinates are preserved.



Note

在 1.3.4 之前，此函数在与包含曲线的几何图形一起使用时会崩溃。此问题已在 1.3.4 及更高版本中得到修正

可用性：1.2.2



函数支持 3d 并且不会丢失 z-index。



此方法支持图形字符串和曲线。



功能支持 M 坐标。

示例

将点移动 1 度经度

Code

```
SELECT ST_Translate(ST_GeomFromText('POINT(-71.01 42.37)', 4326), 1, 0) AS wgs_transgeomtxt ↵  
;
```

Output

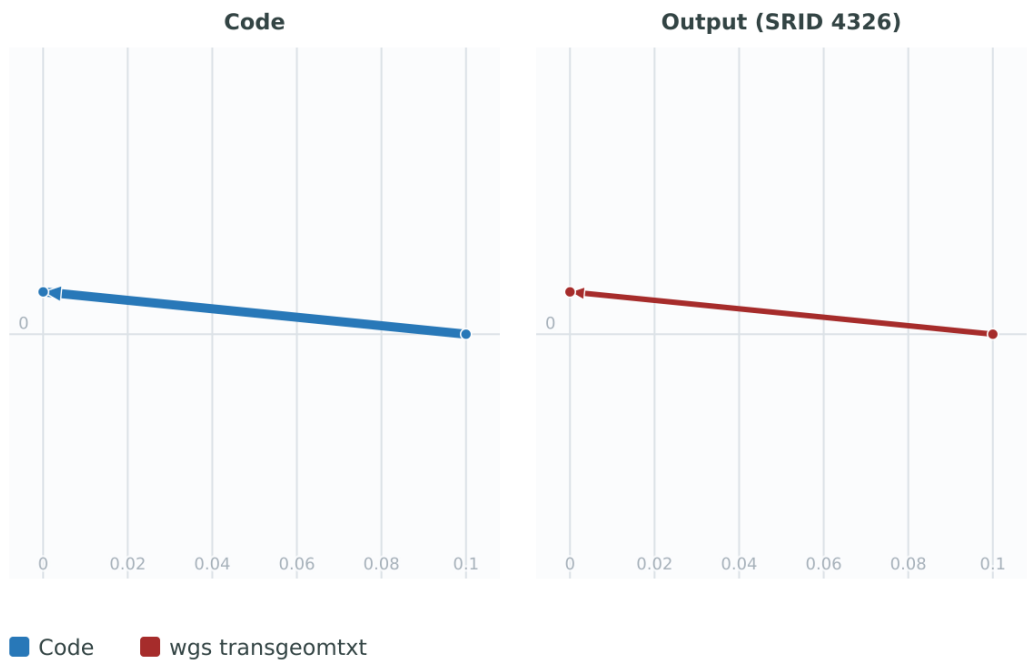
```
POINT(-70.01 42.37)
```

将字符串移动 1 度经度和 1/2 度纬度

Code

```
SELECT ST_Translate(ST_GeomFromText('LINESTRING(-71.01 42.37,-71.11 42.38)', 4326), 1, 0.5) ↵  
AS wgs_transgeomtxt;
```

Figure



移动 3d 点

Code

```
SELECT ST_Translate(CAST('POINT(0 0 0)' AS geometry), 5, 12, 3);
```

Output

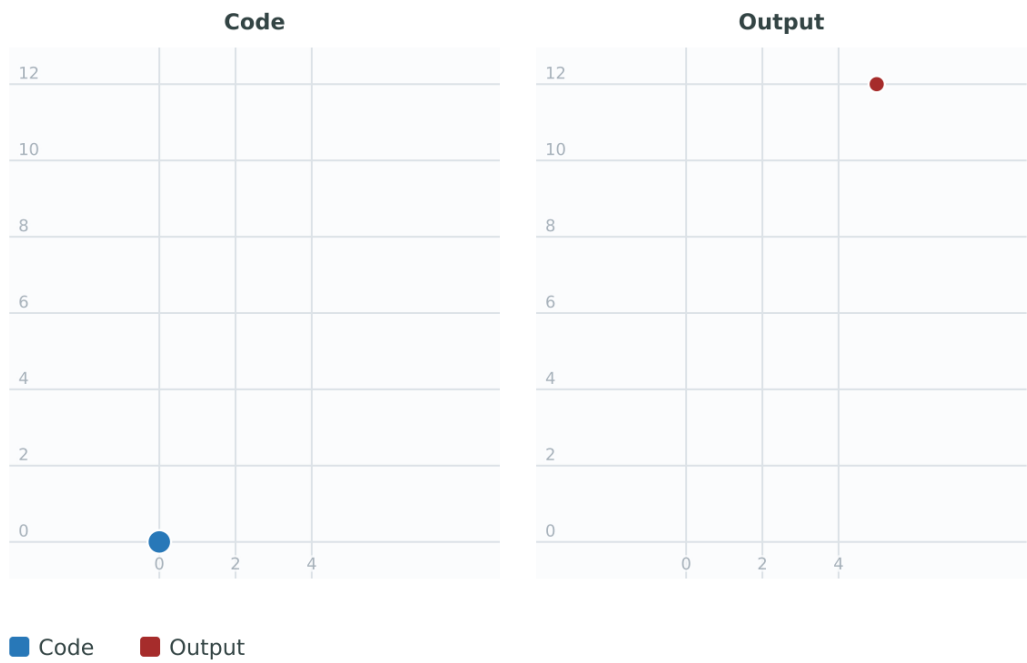
```
POINT(5 12 3)
```

Move points with a measure coordinate

Code

```
SELECT ST_Translate('POINTM(0 0 11)', 5, 12);
```

| **Figure**



| **Code**

```
SELECT ST_Translate('POINT(0 0 7 11)', 5, 12, 3);
```

| **Output**

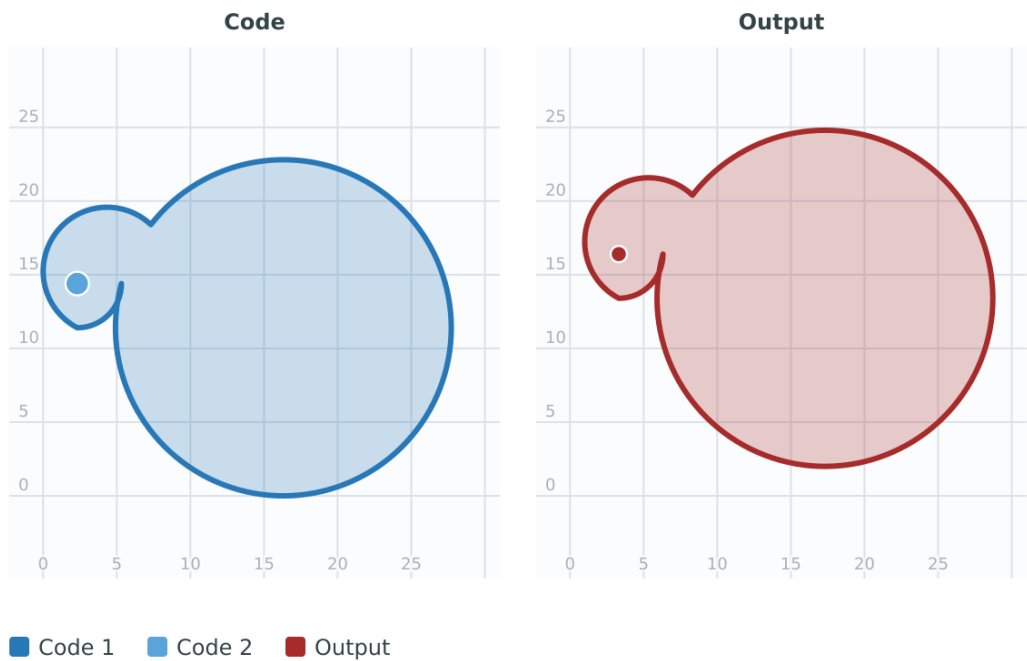
```
POINT(5 12 10 11)
```

移曲和点

| **Code**

```
SELECT 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);
```

Figure



相关信息

[ST_Affine](#), [ST_AsText](#), [ST_GeomFromText](#)

7.16.8 ST_TransScale

ST_TransScale — 按给定的偏移量和系数平移和缩放几何图形。

Synopsis

geometry **ST_TransScale**(geometry geomA, float deltaX, float deltaY, float XFactor, float YFactor);

描述

使用 deltaX 和 deltaY 参数缩放几何体，然后使用 XFactor、YFactor 参数缩放它，在 2D 中工作。

Note

ST_TransScale(geomA, deltaX, deltaY, XFactor, YFactor) 是 ST_Affine(geomA, XFactor, 0, 0, 0, YFactor, 0, 0, 0, 1, deltaX*XFactor, deltaY*YFactor, 0) 的简写。

Note

在 1.3.4 之前，此函数在与包含曲线的几何图形一起使用时会崩溃。此问题已在 1.3.4 及更高版本中得到修正

可用性 : 1.1.0。

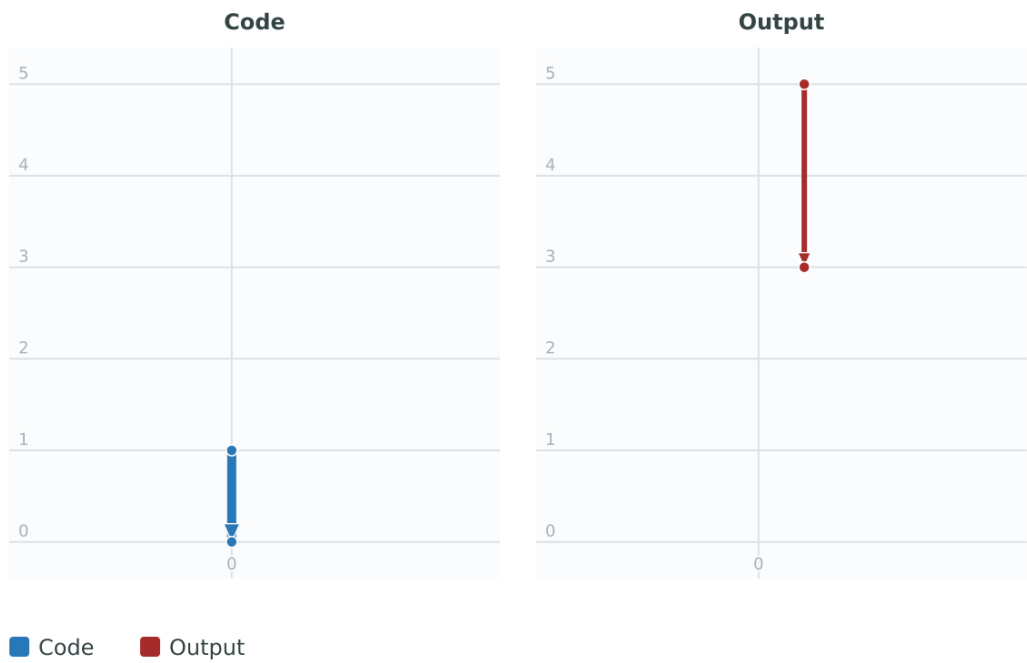
- ✔ 函数支持 3d 并且不会丢失 z-index。
- ✔ 此方法支持圆形字符串和曲线。

示例

Code

```
SELECT ST_TransScale('LINESTRING(1 2 3,1 1 1)', 0.5, 1, 1, 2);
```

Figure

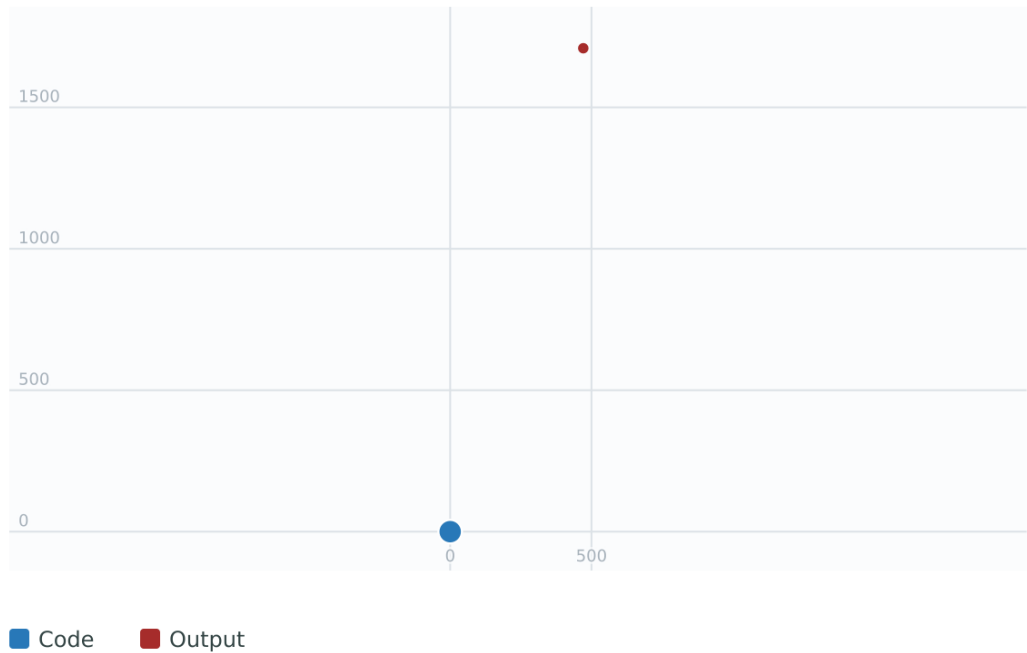


This example buffers a point to approximate a circle, converts it to a curve, translates it by 1,2, and scales it by 3,4.

Code

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


Figure



相关信息

[ST_Affine](#), [ST_Translate](#)

7.17 聚类函数

7.17.1 ST_ClusterDBSCAN

`ST_ClusterDBSCAN` — 使用 DBSCAN 算法返回每个输入几何图形的簇 id 的窗口函数。

Synopsis

```
integer ST_ClusterDBSCAN(geometry winset geom, float8 eps, integer minpoints);
```

描述

一个窗口函数，使用 2D [基于密度的噪声剔除空聚类 \(DBSCAN\)](#) 算法。与 [ST_ClusterKMeans](#) 不同，它不需要指定簇的数量，而是使用所需的[距离 \(eps\)](#) 和密度 ([minpoints](#)) 参数来确定每个簇。

如果输入几何体满足以下任一条件，将其添加到簇中：

- 在 [eps 距离](#) 内的“核心”几何图形，至少有 [minpoints](#) 个输入几何图形（包括它自己）；或
- “边界”几何图形，位于核心几何图形的 [eps 距离](#) 内。

注意，边界几何图形可能位于多个集群中核心几何图形的 [eps](#) 距离内。任一分配都是正确的，因此边界几何图形将被任意分配给可用集群之一。在这种情况下，可以使用少于 [minpoints](#) 几何图形生成正确的簇。为了确保边界几何形状的确定性分配（以便重复调用 `ST_ClusterDBSCAN` 将产生相同的结果），在窗口定义中使用 `ORDER BY` 子句。不明确的簇分配可能与其他 DBSCAN 实现不同。



Note

不足加入任何簇的条件的几何形将被分配簇号 NULL。

可用性 : 2.3.0



此方法支持形字符串和曲。

示例

Cluster polygons within 50 units of each other, requiring at least two polygons per cluster. The isolated polygon is reported as noise.

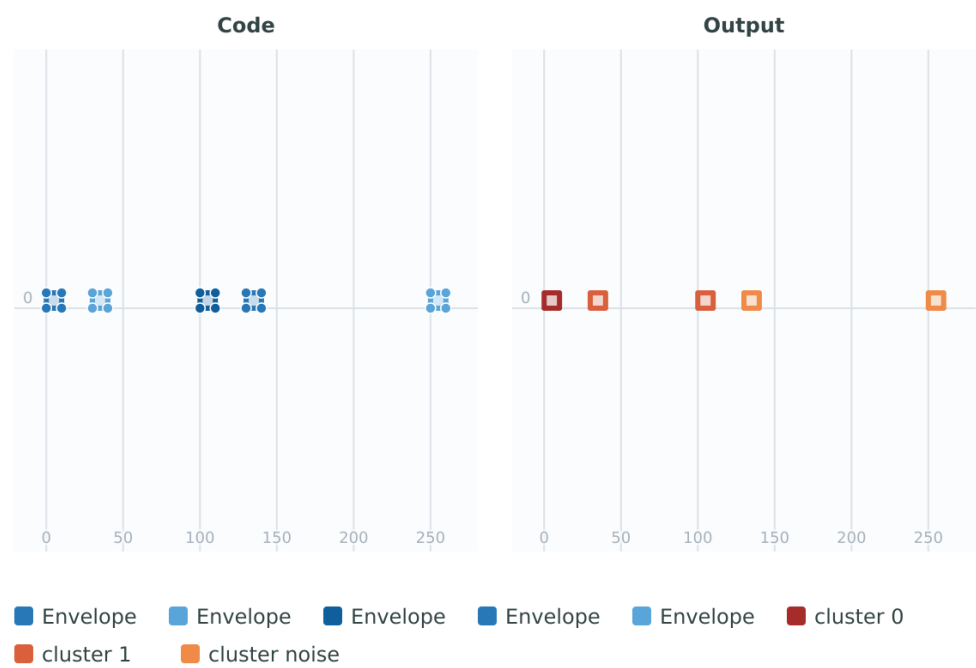
Code

```
WITH input(name, geom) AS (  
  VALUES  
    ('A1', ST_MakeEnvelope(0, 0, 10, 10)),  
    ('A2', ST_MakeEnvelope(30, 0, 40, 10)),  
    ('B1', ST_MakeEnvelope(100, 0, 110, 10)),  
    ('B2', ST_MakeEnvelope(130, 0, 140, 10)),  
    ('noise', ST_MakeEnvelope(250, 0, 260, 10))  
, clustered AS (  
  SELECT name,  
         geom,  
         ST_ClusterDBSCAN(geom, eps => 50, minpoints => 2)  
         OVER (ORDER BY name) AS cid  
  FROM input  
)  
SELECT COALESCE(cid::text, 'noise') AS cluster,  
       string_agg(name, ',' ORDER BY name) AS members,  
       ST_Collect(geom ORDER BY name) AS cluster_geom  
FROM clustered  
GROUP BY cid  
ORDER BY cid NULLS LAST;
```

Output

```
0 | A1,A2 | MULTIPOLYGON(((0 0,0 10,10 10,10 0,0 0)),((30 0,30 10,40 10,40 0,30 0)))  
1 | B1,B2 | MULTIPOLYGON(((100 0,100 10,110 10,110 0,100 0)),((130 0,130 10,140 10,140 0,130 0)))  
noise | noise | MULTIPOLYGON(((250 0,250 10,260 10,260 0,250 0)))
```

Figure



示将具有相同簇号的地合成几何集合的示例。

Code

```
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;
```

相关信息

[ST_DWithin](#), [ST_ClusterKMeans](#), [ST_ClusterIntersecting](#), [ST_ClusterIntersectingWin](#), [ST_ClusterWithin](#), [ST_ClusterWithinWin](#)

7.17.2 ST_ClusterIntersecting

ST_ClusterIntersecting — 将几何形聚成交集的聚合函数。

Synopsis

geometry[] **ST_ClusterIntersecting**(geometry set g);

描述

一个聚合函数，返回 `GeometryCollections` 数组，将几何形划分成不相交的交集簇。簇中的每个几何形与簇中的至少一个其他几何形相交，并且不与其他簇中的任何几何形相交。

可用性：2.2.0

示例

Code

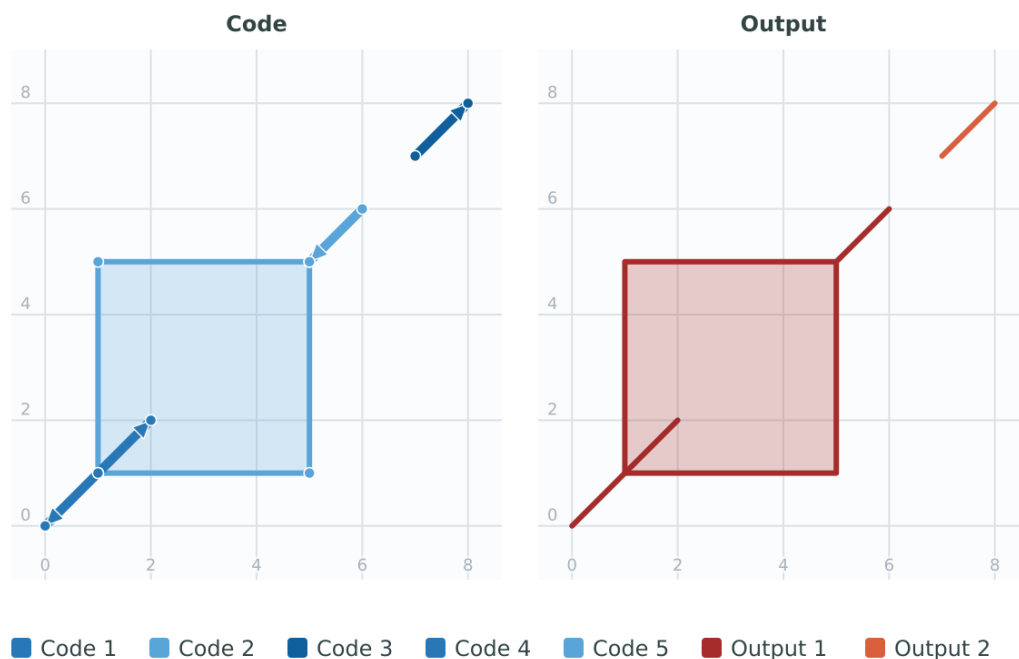
```
WITH testdata AS
  (SELECT unnest(
    ARRAY[ 'LINESTRING (0 0,1 1)::geometry',
           'LINESTRING (5 5,4 4)::geometry',
           'LINESTRING (6 6,7 7)::geometry',
           'LINESTRING (0 0,-1 -1)::geometry',
           'POLYGON ((0 0,4 0,4 4,0 4,0 0))'::geometry] AS geom)

SELECT unnest(ST_ClusterIntersecting(geom)) FROM testdata;
```

Output

```
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))
```

Figure



相关信息

[ST_ClusterIntersectingWin](#), [ST_ClusterWithin](#), [ST_ClusterWithinWin](#)

7.17.3 ST_ClusterIntersectingWin

`ST_ClusterIntersectingWin` — 窗口函数，返回每个输入几何图形的簇 ID，将输入几何图形聚到相接的集合中。

Synopsis

integer **ST_ClusterIntersectingWin**(geometry winset geom);

描述

一种窗口函数，用于构建相交的接几何形簇。可以在不离开集群的情况下遍历集群中的所有几何形。返回是几何参数参与的簇号，或者于空入空。

可用性：3.4.0

示例

Return an explicit cluster id for each input geometry. Using the conventional cid alias keeps the geometry column ready for color-by-cluster rendering.

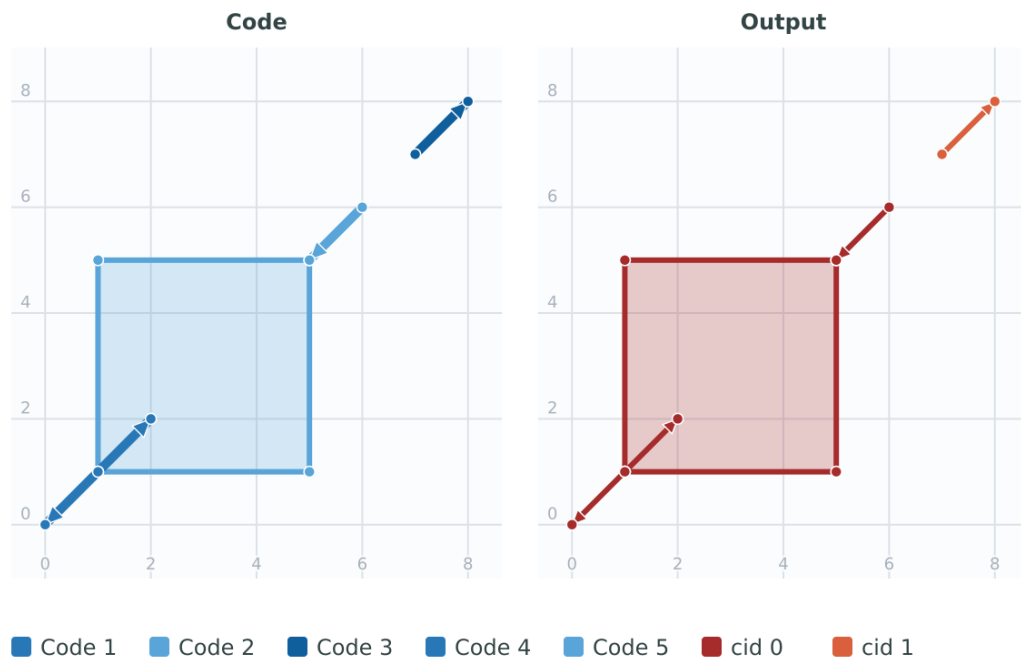
Code

```
WITH testdata(id, geom) AS (  
  VALUES (1, 'LINESTRING (0 0,1 1)::geometry'),  
          (2, 'LINESTRING (5 5,4 4)::geometry'),  
          (3, 'LINESTRING (6 6,7 7)::geometry'),  
          (4, 'LINESTRING (0 0,-1 -1)::geometry'),  
          (5, 'POLYGON ((0 0,4 0,4 4,0 4,0 0))::geometry')  
) , clustered AS (  
  SELECT id,  
         geom,  
         ST_ClusterIntersectingWin(geom) OVER () AS cid  
  FROM testdata  
)  
SELECT id,  
       geom AS geom_wkt,  
       cid  
FROM clustered  
ORDER BY id;
```

Output

id	geom_wkt	cid
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

Figure



相关信息

[ST_ClusterIntersecting](#), [ST_ClusterWithin](#), [ST_ClusterWithinWin](#)

7.17.4 ST_ClusterRelateWin

ST_ClusterRelateWin — Window function that returns a cluster id for each input geometry, clustering input geometries into connected sets using the relate pattern to determine whether the geometries are connected.

Synopsis

```
integer ST_ClusterRelateWin(geometry winset geom, text relate_matrix);
```

描述

A window function that builds connected clusters of geometries that intersect. Geometries are added to a cluster if they share a pairwise DE9IM relationship with another member of the cluster. With this function it is possible to build a cluster of all objects that touch at boundaries, but exclude those that merely overlap.

Availability: 3.7.0

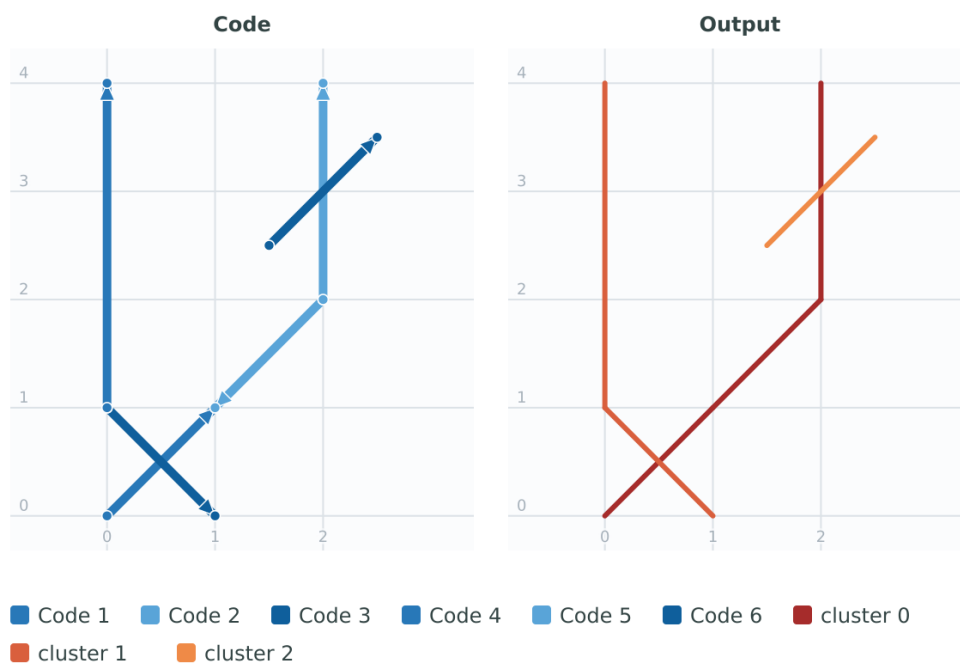
示例

This collection of line strings would form a single cluster using [ST_ClusterIntersectingWin](#), but using [ST_ClusterRelateWin](#) can be clustered into three groups that connect only via their end points.

I Code

```
WITH data(id, geom) AS (
  VALUES (1, 'LINESTRING(0 0,1 1)::geometry'),
          (2, 'LINESTRING(2 2,1 1)::geometry'),
          (3, 'LINESTRING(0 1,1 0)::geometry'),
          (4, 'LINESTRING(0 1,0 4)::geometry'),
          (5, 'LINESTRING(2 2,2 4)::geometry'),
          (6, 'LINESTRING(1.5 2.5,2.5 3.5)::geometry')
), clustered AS (
  SELECT id, geom,
         ST_ClusterRelateWin(geom, '****0****') OVER () AS cluster
  FROM data
)
SELECT cluster,
       string_agg(id::text, ',' ORDER BY id) AS members,
       ST_Collect(geom ORDER BY id) AS cluster_geom
FROM clustered
GROUP BY cluster
ORDER BY cluster;
```

Figure



相关信息

[ST_Relate](#), [ST_ClusterIntersectingWin](#), [ST_ClusterWithinWin](#)

7.17.5 ST_MinimumSpanningTree

ST_MinimumSpanningTree — Window function that returns a tree id for each input geometry, clustering input geometries into connected trees using the Minimum Spanning Tree algorithm.

Synopsis

integer **ST_MinimumSpanningTree**(geometry winset geom);

描述

A window function that builds connected graphs of line strings based on the Minimum Spanning Tree (MST) of the input geometries. The return value is the cluster number that the geometry argument participates in, or zero if it is not part of the minimum tree.

The **Minimum Spanning Tree** connects all geometries in the window partition such that the total length of the connecting lines is minimized. If the graph is not fully connected (e.g. infinite distance between some geometries), it produces a Minimum Spanning Forest, and each connected component is assigned a unique tree ID.

Availability: 3.7.0

Requires GEOS >= 3.15.0

示例

These six edges describe a square with both diagonals. The minimum spanning tree keeps three edges and excludes the rest.

Code

```
WITH edges(id, geom) AS (
  VALUES
    (1, 'LINESTRING(0 0,1 0)::geometry'),
    (2, 'LINESTRING(0 0,0 1)::geometry'),
    (3, 'LINESTRING(1 1,0 1)::geometry'),
    (4, 'LINESTRING(1 1,1 0)::geometry'),
    (5, 'LINESTRING(0 0,1 1)::geometry'),
    (6, 'LINESTRING(1 0,0 1)::geometry')
), marked AS (
  SELECT id,
         geom,
         ST_MinimumSpanningTree(geom) OVER (ORDER BY id) AS tree_id
  FROM edges
)
SELECT string_agg(id::text, ',' ORDER BY id)
       FILTER (WHERE tree_id > 0) AS tree_edge_ids,
       ST_Collect(geom ORDER BY id)
       FILTER (WHERE tree_id > 0) AS tree_edges,
       string_agg(id::text, ',' ORDER BY id)
       FILTER (WHERE tree_id = 0) AS excluded_edge_ids,
       ST_Collect(geom ORDER BY id)
       FILTER (WHERE tree_id = 0) AS excluded_edges
FROM marked;
```

Output

```
-[ RECORD 1 ]-----
tree_edge_ids | 1,2,3
tree_edges    | MULTILINESTRING((0 0,1 0),(0 0,0 1),(1 1,0 1))
excluded_edge_ids | 4,5,6
excluded_edges | MULTILINESTRING((1 1,1 0),(0 0,1 1),(1 0,0 1))
```


相关信息

[ST_ClusterDBSCAN](#), [ST_ClusterKMeans](#), [ST_ClusterIntersectingWin](#)

7.17.6 ST_ClusterKMeans

`ST_ClusterKMeans` — 使用 K 均值算法返回每个输入几何形状的簇 id 的窗口函数。

Synopsis

```
integer ST_ClusterKMeans( geometry winset geom , integer k , float8 max_radius );
```

描述

返回每个输入几何形状的 **K-means** 簇号。用于聚类的距离是 2D 几何形状的中心之心的距离，以及 3D 几何形状的边界框中心之心的距离。对于 POINT 输入，M 坐标将被忽略输入的重量，并且必须大于 0。

`max_radius` 如果设置，将导致 `ST_ClusterKMeans` 生成比 `k` 更多的簇，确保输出中没有簇的半径大于 `max_radius`。这在可行性分析中很有用。

增加：3.2.0 支持 `max_radius`

增加：3.1.0 支持 3D 几何和重量

可用性：2.3.0

示例

Define the parcel geometries once in a CTE and reuse them across the executable examples.

Parcels color-coded by cluster number (cid).

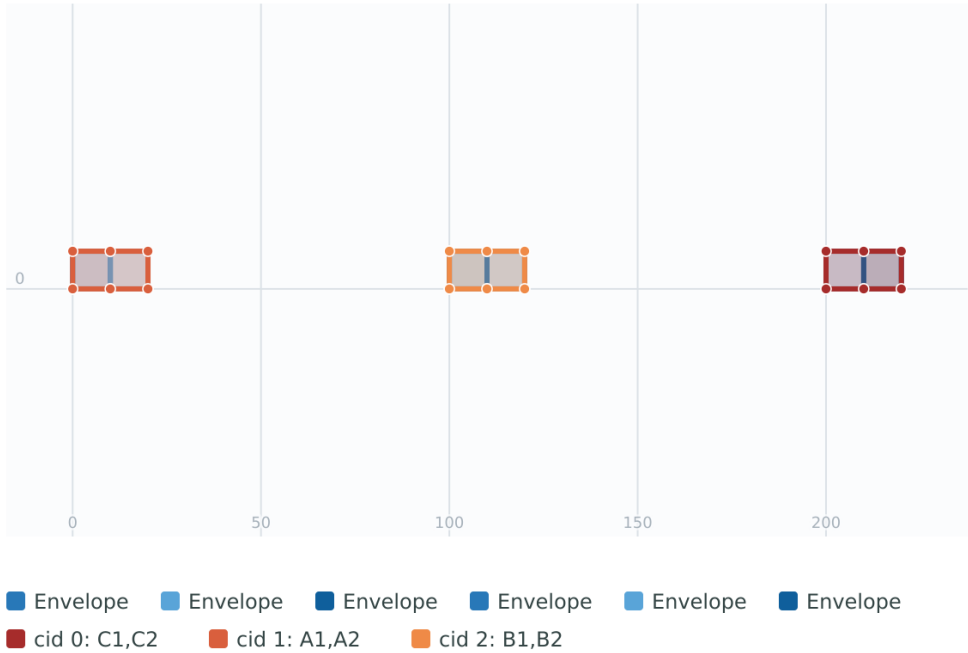
Code

```
WITH parcels(parcel_id, geom) AS (
  VALUES
    ('A1', ST_MakeEnvelope(0, 0, 10, 10)),
    ('A2', ST_MakeEnvelope(10, 0, 20, 10)),
    ('B1', ST_MakeEnvelope(100, 0, 110, 10)),
    ('B2', ST_MakeEnvelope(110, 0, 120, 10)),
    ('C1', ST_MakeEnvelope(200, 0, 210, 10)),
    ('C2', ST_MakeEnvelope(210, 0, 220, 10))
), clustered AS (
  SELECT parcel_id,
         geom,
         ST_ClusterKMeans(geom, 3) OVER (ORDER BY parcel_id) AS cid
  FROM parcels
)
SELECT cid,
       string_agg(parcel_id, ',' ORDER BY parcel_id) AS parcels,
       ST_Union(geom) AS cluster_geom
FROM clustered
GROUP BY cid
ORDER BY cid;
```

Output

```
0 | C1,C2 | POLYGON((200 10,210 10,220 10,220 0,210 0,200 0,200 10))
1 | A1,A2 | POLYGON((0 10,10 10 10,20 10,20 0,10 0,0 0,0 10))
2 | B1,B2 | POLYGON((100 10,110 10,120 10,120 0,110 0,100 0,100 10))
```

Figure



按☐型划分地☐集群：

Code

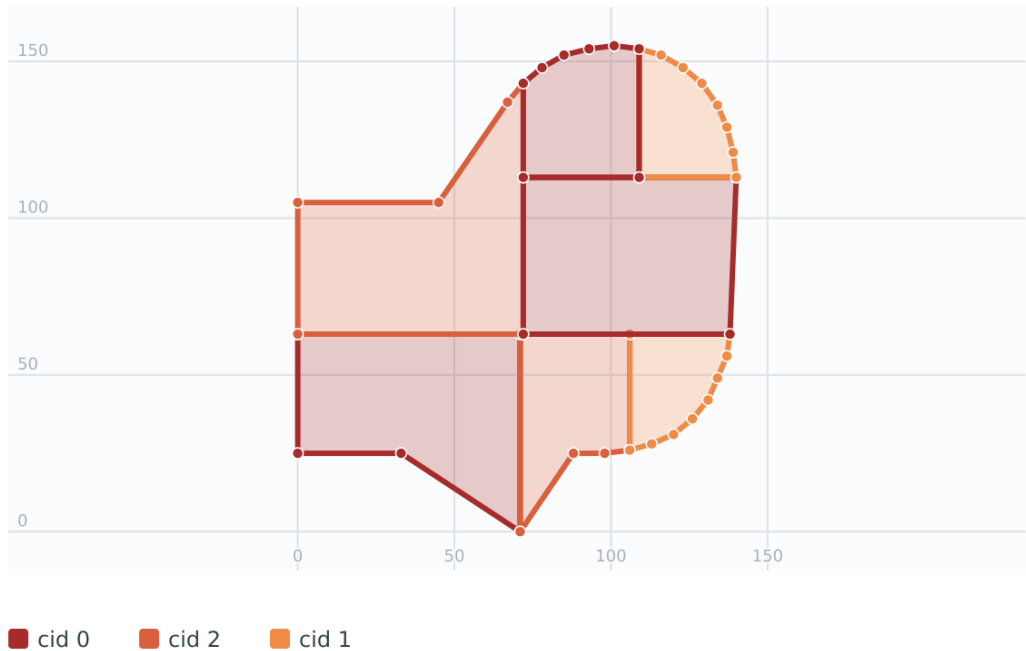
```
WITH parcel_geoms AS (
  SELECT geom
  FROM ST_Subdivide(
    ST_Buffer('SRID=3857;LINESTRING(40 100,98 100,100 150,60 90)::geometry',
      40, 'endcap=square'),
    12) AS geom
), parcels AS (
  SELECT lpad(row_number() OVER (ORDER BY ST_YMin(geom), ST_XMin(geom))::text, 3, '0') AS
    parcel_id,
    geom,
    ('{residential,commercial}'::text[])[1 + mod(row_number() OVER (ORDER BY ST_YMin(
      geom), ST_XMin(geom)), 2)] AS type
  FROM parcel_geoms
)
SELECT ST_ClusterKMeans(geom, 3) over (PARTITION BY type) AS cid,
  parcel_id,
  type,
  ST_SnapToGrid(geom, 1) AS parcel_geom
FROM parcels
ORDER BY type, parcel_id;
```

Output

cid	parcel_id	type	parcel_geom
0	001	commercial	POLYGON((33 60,0 60,0 98,71 98,71 35,33 60))
2	003	commercial	POLYGON((106 98,138 98,137 91,134 84,131 77,126 71,120 66,113 63,106 61,106 98))
2	005	commercial	POLYGON((140 148,138 98,72 98,72 148,140 148))

1	007	commercial	POLYGON((109 189,116 187,123 183,129 178,134 171,137 164,139 156,140 148,109 148,109 ↵
0	002	residential	POLYGON((71 98,106 98,106 61,98 60,88 60,71 35,71 98))
2	004	residential	POLYGON((0 98,0 140,45 140,67 172,72 178,72 98,0 98))
1	006	residential	POLYGON((72 178,78 183,85 187,93 189,101 190,109 189,109 148,72 148,72 178))

Figure



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:

Code

```
CREATE TABLE kontur_population_3000km_clusters AS
SELECT
  geom,
  ST_ClusterKMeans(
    ST_Force4D(
      ST_Transform(ST_Force3D(geom), 4978),
      mvalue => population
    ),
    20,
    max_radius => 3000000
  ) OVER () AS cid
FROM kontur_population;
```

The clustering produces 46 regions. Clusters are centered at well-populated regions such as New York and Moscow. Greenland forms one cluster, several island clusters span the antimeridian, and cluster edges follow the Earth’s curvature.



Kontur population clustered with 3000 km maximum radius.

Cluster weighted city populations on the Earth in geocentric coordinates. The maximum radius is specified in meters, while the returned geometries stay in WGS 84 for display.

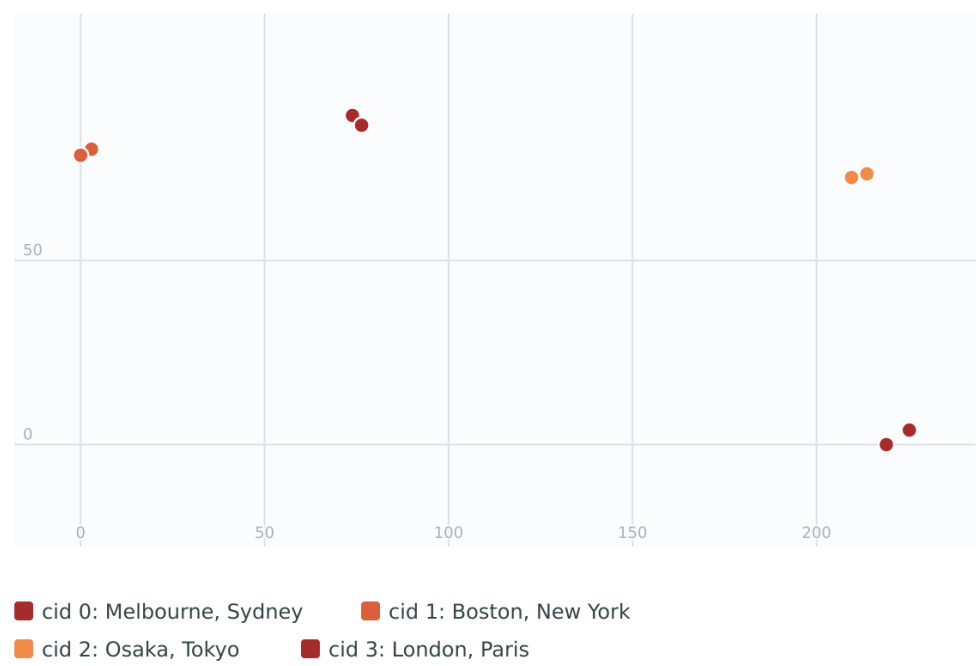
Code

```
WITH population(place, population, geom) AS (
  VALUES
    ('Boston', 5.0, ST_SetSRID(ST_Point(-71.06, 42.36), 4326)),
    ('New York', 20.0, ST_SetSRID(ST_Point(-74.01, 40.71), 4326)),
    ('London', 15.0, ST_SetSRID(ST_Point(-0.13, 51.51), 4326)),
    ('Paris', 11.0, ST_SetSRID(ST_Point(2.35, 48.86), 4326)),
    ('Osaka', 19.0, ST_SetSRID(ST_Point(135.50, 34.69), 4326)),
    ('Tokyo', 37.0, ST_SetSRID(ST_Point(139.69, 35.68), 4326)),
    ('Melbourne', 5.0, ST_SetSRID(ST_Point(144.96, -37.81), 4326)),
    ('Sydney', 5.0, ST_SetSRID(ST_Point(151.21, -33.87), 4326))
), clustered AS (
  SELECT place,
    geom,
    ST_ClusterKMeans(
      ST_Force4D(
        ST_Transform(ST_Force3D(geom), 4978),
        mvalue => population
      ),
      4,
      max_radius => 3000000
    ) OVER (ORDER BY place) AS cid
  FROM population
)
SELECT cid,
  string_agg(place, ', ' ORDER BY place) AS places,
  ST_Collect(geom ORDER BY place) AS cluster_geom
FROM clustered
GROUP BY cid
ORDER BY cid;
```

Output

```
0 | Melbourne, Sydney | MULTIPOINT((144.96 -37.81),(151.21 -33.87))
1 | Boston, New York | MULTIPOINT((-71.06 42.36),(-74.01 40.71))
2 | Osaka, Tokyo | MULTIPOINT((135.5 34.69),(139.69 35.68))
3 | London, Paris | MULTIPOINT((-0.13 51.51),(2.35 48.86))
```

Figure



相关信息

[ST_ClusterDBSCAN](#), [ST_ClusterIntersectingWin](#), [ST_ClusterWithinWin](#), [ST_ClusterIntersecting](#), [ST_Cluster](#), [ST_Subdivide](#), [ST_Force3D](#), [ST_Force4D](#),

7.17.7 ST_ClusterWithin

`ST_ClusterWithin` — 按间隔距离几何形状行聚合的聚合函数。

Synopsis

`geometry[] ST_ClusterWithin(geometry set g, float8 distance);`

描述

返回 `GeometryCollections` 数组的聚合函数，其中每个集合都是包含一些输入几何形状的簇。聚将输入几何形状划分多个集合，其中每个几何形状都在同一簇中至少一个其他几何形状的指定距离内。距离是以 SRID 位的笛卡距离。

`ST_ClusterWithin` 相当于行 [ST_ClusterDBSCAN](#) 使用 `minpoints => 0`。

可用性：2.2.0

 此方法支持形状字符串和曲线。

示例

Code

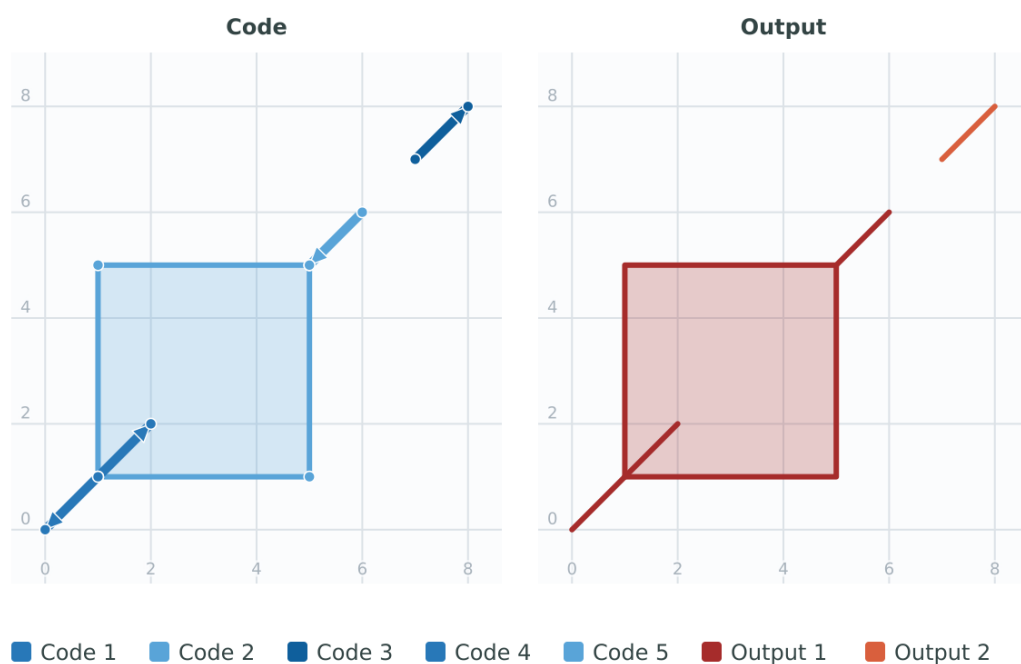
WITH testdata AS

```
(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))'] AS geom)
```

```
SELECT unnest(ST_ClusterWithin(geom, 1.4)) FROM testdata;
```

Output

```
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))
```

Figure**相关信息**

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

7.17.8 ST_ClusterWithinWin

`ST_ClusterWithinWin` — 窗口函数，返回每个输入几何形的簇 ID，使用分离距离进行聚。

Synopsis

integer **ST_ClusterWithinWin**(geometry winset geom, float8 distance);

描述

返回每个几何图形的簇号的窗口函数。聚簇将几何图形划分多个集合，其中每个几何图形都在同一簇中至少一个其他几何图形的指定距离内。距离是以 SRID 为单位的笛卡儿距离。

ST_ClusterWithinWin 等效于行 ST_ClusterDBSCAN 使用 minpoints => 0。

可用性：3.4.0

 此方法支持图形字符串和曲线。

示例

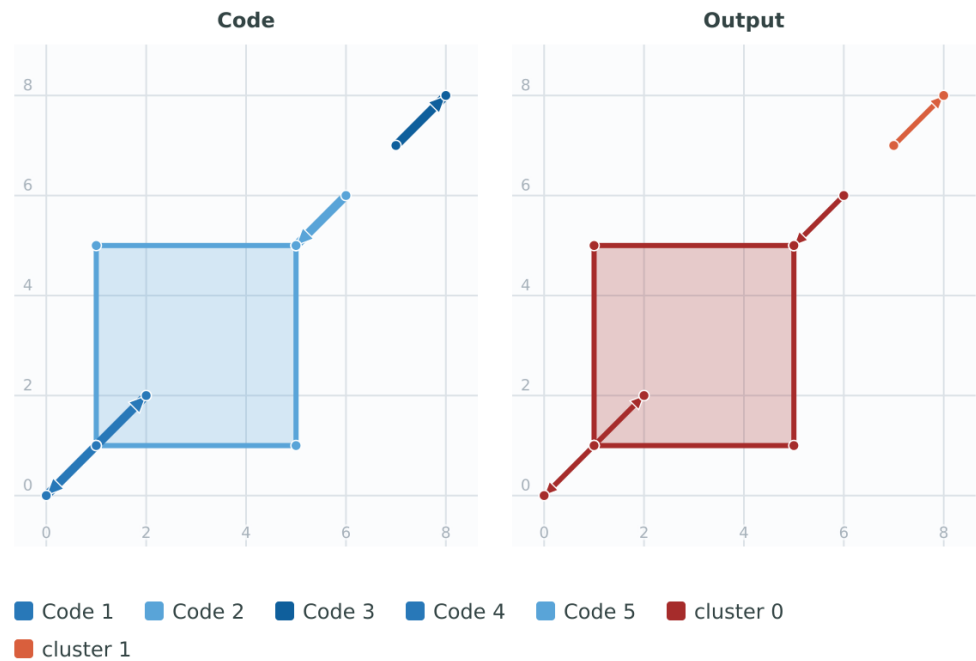
Code

```
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,  
       geom,  
       ST_ClusterWithinWin(geom, 1.4) OVER () AS cluster  
FROM testdata;
```

Output

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

Figure



相关信息

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

7.18 边界框函数

7.18.1 Box2D

Box2D — 返回表示几何图形的 2D 范围的 BOX2D。

Synopsis

box2d **Box2D**(geometry geom);

描述

返回表示几何图形的 2D 范围的 **box2d**。

增强功能：引入了 2.0.0 的多面体曲面、三角形和三角网的支持。



此方法支持图形字符串和曲线。



函数支持多面体曲面。



此函数支持三角形和不规则三角网面 (TIN)。

示例

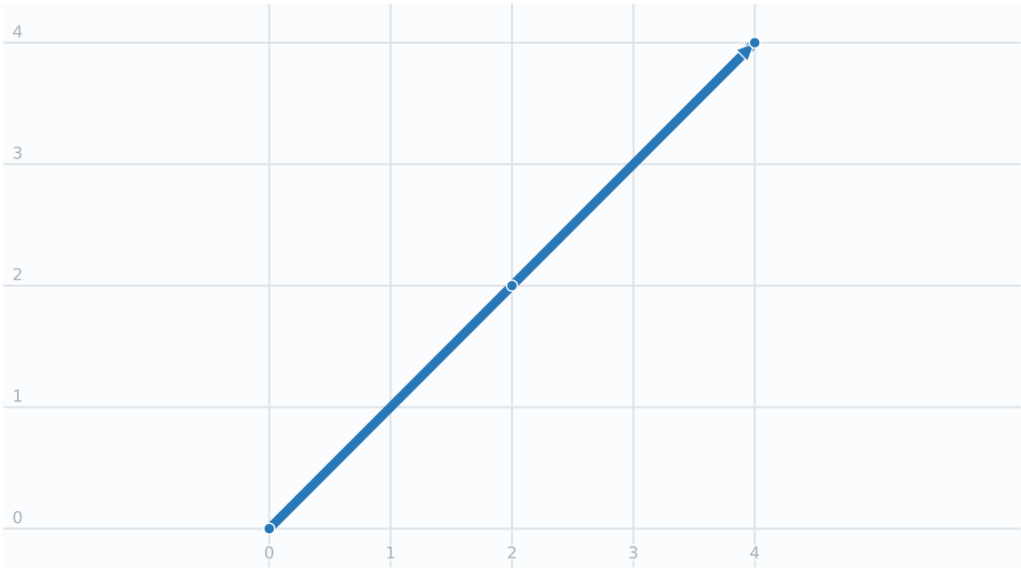
Code

```
SELECT Box2D('LINESTRING(1 2,3 4,5 6)::geometry);
```

Output

```
BOX(1 2,5 6)
```


Figure



Code

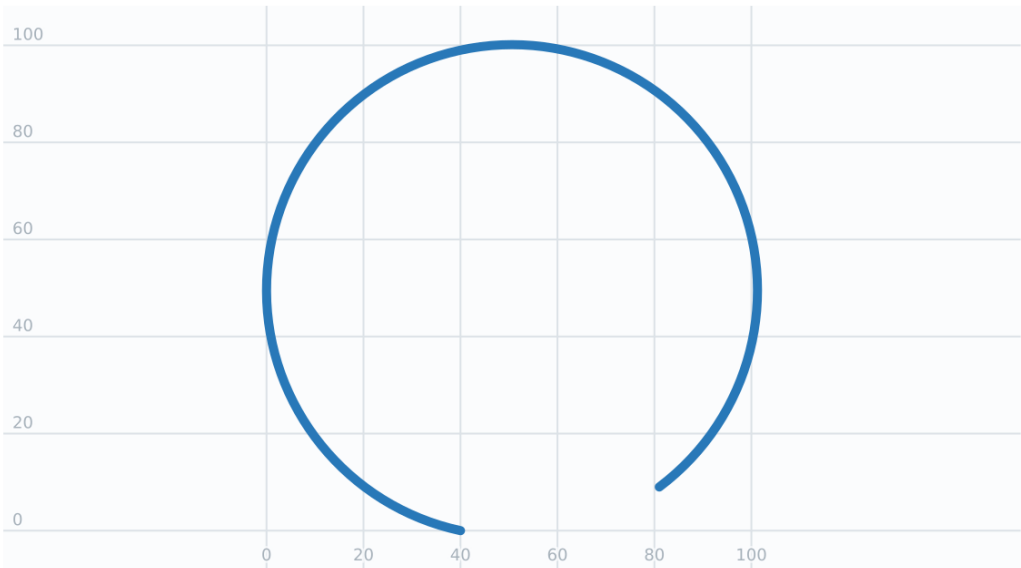
Code

```
SELECT Box2D( 'CIRCULARSTRING(220268 150415,220227 150505,220227 150406) '::geometry);
```

Output

BOX(220186.99512189245 150406,220288.24878054656 150506.12682932706)

Figure



Code

相关信息

[Box3D](#), [ST_GeomFromText](#)

7.18.2 Box3D

Box3D — 返回表示几何体 3D 范围的 BOX3D。

Synopsis

box3d **Box3D**(geometry geom);

描述

返回表示几何体 3D 范围的 **box3d**。

增强功能：引入了 2.0.0 的多面体曲面、三角形和三角网的支持。

- ✓ 此方法支持 WKT 字符串和曲线。
- ✓ 该函数支持多面体曲面。
- ✓ 此函数支持三角形和不规则三角网面 (TIN)。
- ✓ 该函数支持 3d 并且不会丢失 z-index。

示例

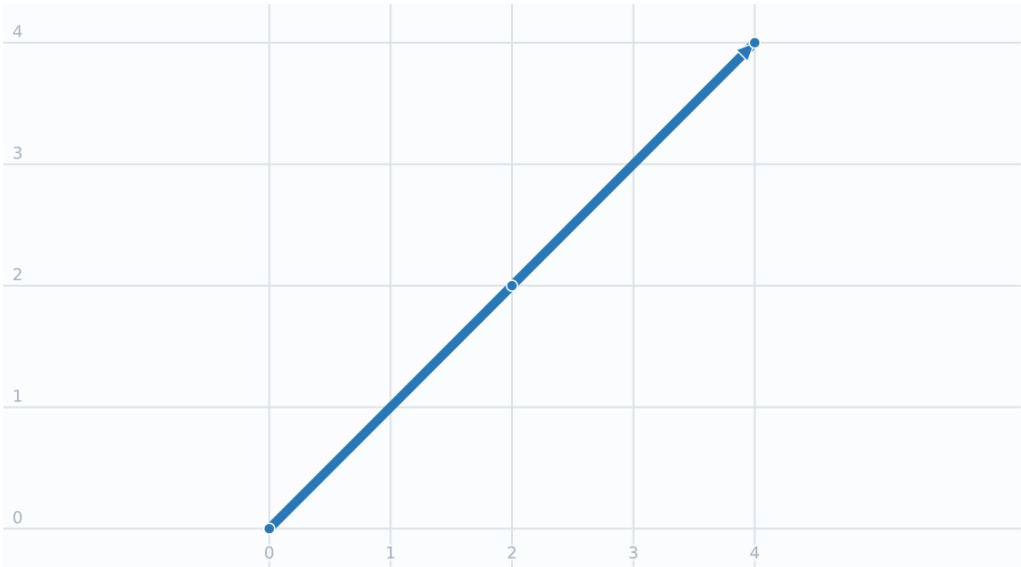
Code

```
SELECT Box3D('LINESTRING(1 2 3,3 4 5,5 6 5)::geometry);
```

Output

```
BOX3D(1 2 3,5 6 5)
```

Figure



Code

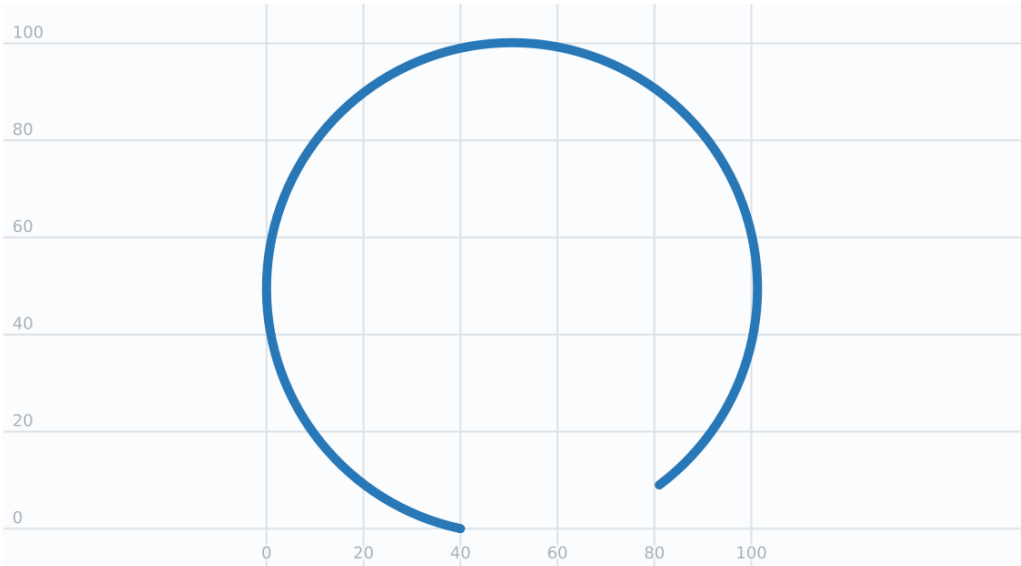
Code

```
SELECT Box3D( 'CIRCULARSTRING(220268 150415 1,220227 150505 1,220227 150406 1)::geometry);
```

Output

```
BOX3D(220186.99512189245 150406 1,220288.24878054656 150506.12682932706 1)
```

Figure



Code

相关信息

[Box2D](#), [ST_GeomFromEWKT](#)

7.18.3 ST_EstimatedExtent

ST_EstimatedExtent — 返回空表的估计范围。

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);
```

描述

以 **box2d** 形式返回空表的估计范围。如果未指定，使用当前架构。估计范围取自几何列的数据。通常比使用 **ST_Extent** 或 **ST_3DExtent** 计算表的精确范围要快得多。

默认行是使用从子表（具有 INHERITS 的表）收集的统计信息（如果可用）。如果 **parent_only** 置为 TRUE，使用指定表的统计信息并忽略子表。

对于 PostgreSQL >= 8.0.0，统计信息由 VACUUM ANALYZE 收集，范围是范围范围的 95%。对于 PostgreSQL < 8.0.0 统计信息是通过行 **update_geometry_stats()** 收集的，范围是准确的。



Note

如果没有统计信息（空表或未用 ANALYZE），此函数将返回 NULL。在 1.5.4 版本之前，会抛出异常。



Note

对于表名或命名空间（schema）的名称，如果包含特殊字符或引号，可能需要进行特殊处理。有些用户会使用“反引号”，“通过加上双引号来生成字符串，然后去掉最前面和最后面的双引号。而对于几何列（geometry column），直接返回原始值，而不进行任何额外的处理。”

可用性：1.0.0

更改：2.1.0。在 2.0.x 之前，名称为 ST_Estimated_Extent。



此方法支持几何字符串和曲线。

示例

Code

```
SELECT ST_EstimatedExtent('ny', 'edges', 'geom');
```

Output

```
st_estimatedextent
-----
BOX(-8877653 4912316,-8010225.5 5589284)
(1 row)
```

Code

```
SELECT ST_EstimatedExtent('feature_poly', 'geom');
```

Output

```
st_estimatedextent
-----
BOX(-124.659652709961 24.6830825805664,-67.7798080444336 49.0012092590332)
(1 row)
```

相关信息

[ST_Extent](#), [ST_3DExtent](#)

7.18.4 ST_Expand

ST_Expand — 返回从一个边界框或几何图形扩展的边界框。

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);
```

描述

返回从输入的边界框扩展的边界框，通过指定框在每个轴上扩展的距离，或通过指定每个轴的扩展距离。使用双精度。可用于距离修剪，或向修剪添加边界框修剪器以利用空索引。

除了接受和返回几何图形的 ST_Expand 版本之外，它提供了接受和返回 **box2d** 和 **box3d** 数据类型形式。

距离采用输入的空参考系的位置。

ST_Expand 与 **ST_Buffer** 类似，不同之处在于缓冲在所有方向上扩展几何图形，而 ST_Expand 沿每个轴扩展边界框。



Note

在版本 1.3 之前，ST_Expand 与 **ST_Distance** 组合使用来执行可索引距离修剪。例如，`geom && ST_Expand('POINT(10 20)', 10) AND ST_Distance(geom, 'POINT(10 20)')<10`。它已被更简单、更高效的 **ST_DWithin** 函数取代。

可用性：1.5.0 行更改出双精度而不是 float4 坐标。

增强功能：引入了 2.0.0 多面体曲面、三角形和三角网的支持。

增强：2.3.0 添加了不同度的盒子行不同数量展的支持。



函数支持多面体曲面。



此函数支持三角形和不三角网面 (TIN)。

示例



Note

以下示例使用美国国家地形平面投影 (SRID=2163)，它是一个以米为单位的投影

10-meter expanded box around bbox of a linestring.

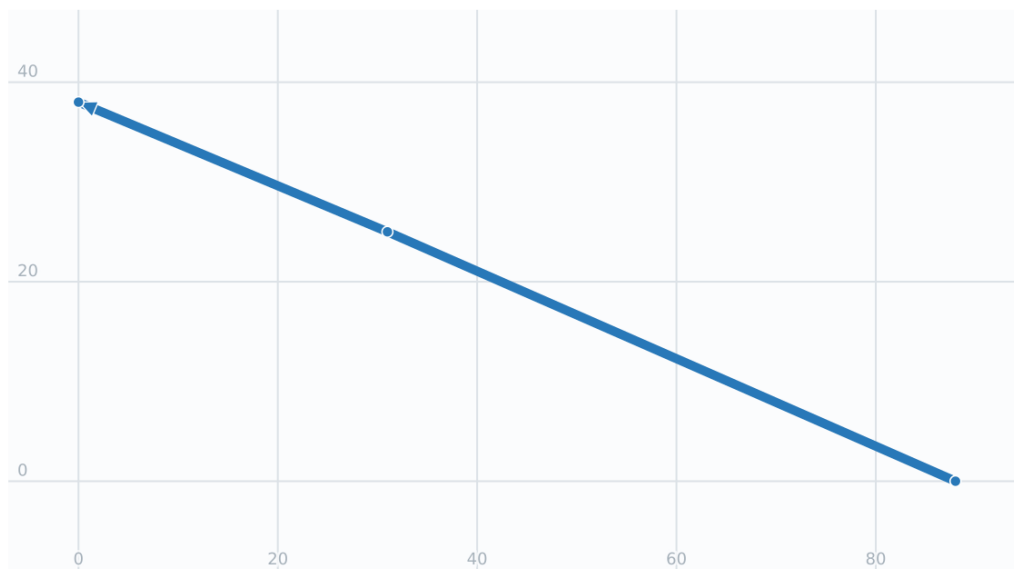
Code

```
SELECT CAST(ST_Expand(ST_GeomFromText('LINESTRING(2312980 110676,2312923 110701,2312892 110714)', 2163), 10) AS box2d);
```

Output

```
BOX(2312882 110666,2312990 110724)
```

Figure



Code

This example expands a 3D box by 10 meters.

Code

```
SELECT ST_Expand(CAST('BOX3D(778783 2951741 1,794875 2970042.61545891 10)' AS box3d), 10)
```

Output

```
BOX3D(778773 2951731 -9,794885 2970052.61545891 20)
```

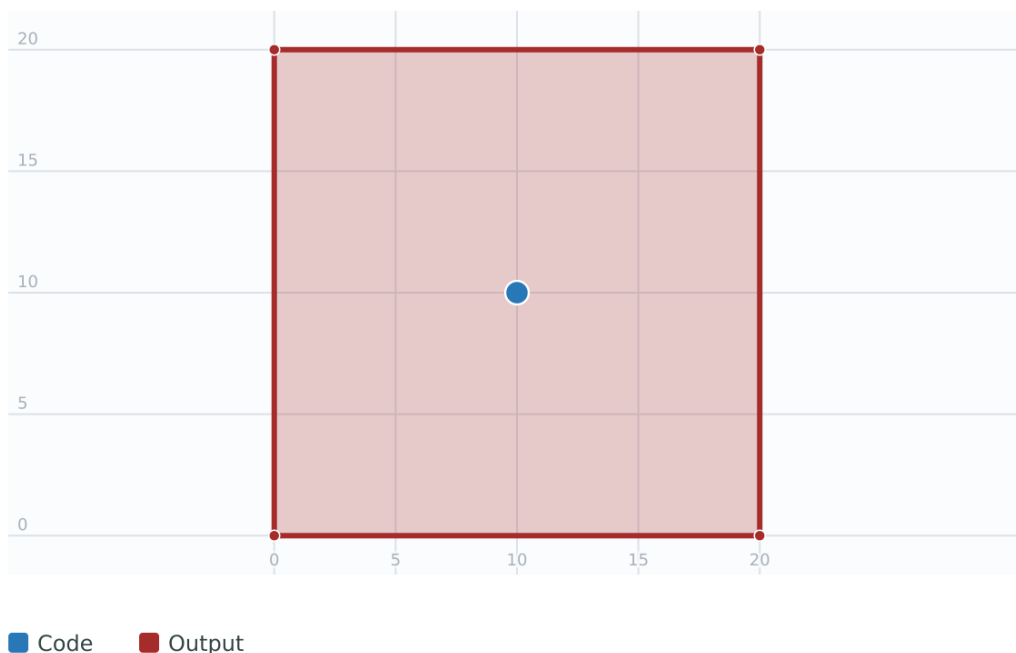
This example renders the text representation of a 10-meter expanded box around a point geometry.

Code

```
SELECT ST_Expand('SRID=2163;POINT(2312980 110676) '::geometry, 10);
```

Output

```
SRID=2163;POLYGON((2312970 110666,2312970 110686,2312990 110686,2312990 110666,2312970 110666))
```

Figure**相关信息**

[ST_Buffer](#), [ST_DWithin](#), [ST_SRID](#)

7.18.5 ST_Extent

ST_Extent — 返回几何图形的聚合函数。

Synopsis

```
box2d ST_Extent(geometry set geomfield);
```

描述

一个聚合函数，返回一个包围几何图形的 **box2d** 边界框。
边界框位于输入几何图形的空参考系中。
ST_Extent 在概念上与 Oracle Spatial/Locator 的 SDO_AGGR_MBR 类似。

Note!

Note
ST_Extent 返回具有 X 和 Y 坐标的框，即使具有 3D 几何图形也是如此。要返回 XYZ 坐标，使用 **ST_3DExtent**。

Note!

Note
返回的 box3d 不包含 SRID。使用 **ST_SetSRID** 将其转换为具有 SRID 元数据的几何图形。SRID 与输入几何形状相同。

增强功能：引入了 2.0.0 多面体曲面、三角形和三角网的支持。

- ✓ 函数支持多面体曲面。
- ✓ 此函数支持三角形和不规则三角网面 (TIN)。

示例

Note!

Note
以下示例使用以英尺为单位的加利福尼亚州平面 (SRID=2249)

Code

```
SELECT ST_Extent(geom) AS bextent FROM sometable;
```

Output

```
BOX(739651.875 2908247.25,794875.8125 2970042.75)
```

Return extent of each geometry category.

Code

```
SELECT ST_Extent(geom) as bextent
FROM sometable
GROUP BY category ORDER BY category;
```

Output

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

This example converts the extent back into a geometry and renders its extended text representation.

Code

```
SELECT ST_SetSRID(ST_Extent(geom), 2249) AS bextent FROM sometable;
```

Output

```
SRID=2249;POLYGON((739651.875 2908247.25,739651.875 2970042.75,794875.8125 2970042.75,794875.8125 2908247.25,739651.875 2908247.25))
```

相关信息

[ST_EstimatedExtent](#), [ST_3DExtent](#), [ST_SetSRID](#)

7.18.6 ST_3DExtent

ST_3DExtent — 返回几何图形的 3D 边界框的聚合函数。

Synopsis

box3d **ST_3DExtent**(geometry set geomfield);

描述

一个聚合函数，返回包围几何图形的 **box3d**（包括 Z 坐标）边界框。

边界框坐标位于输入几何图形的空参考系中。



Note

返回的 **box3d** 不包含 SRID。使用 **ST_SetSRID** 将其转换为具有 SRID 元数据的几何图形。SRID 与输入几何形状相同。

新增功能：引入了 2.0.0 对多面体曲面、三角形和三角网的支持。

更改：2.0.0 在之前的版本中，它曾被称作 **ST_Extent3D**

✓ 该函数支持 3d 并且不会丢失 z-index。

✓ 此方法支持图形字符串和曲面。

✓ 该函数支持多面体曲面。

✓ 此函数支持三角形和不规则三角网面 (TIN)。

示例

Code

```
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;
```

Output

```
BOX3D(1 1 0,3 2 2)
```

This example returns the extent of several elevated CircularStrings.

Code

```
SELECT ST_3DExtent(foo.geom) As b3extent
FROM (SELECT ST_Translate(ST_Force3DZ(ST_LineToCurve(ST_Buffer(ST_Point(x, y), 1))), 0, 0, ↵
      z) As geom
      FROM generate_series(1, 3) As x
      CROSS JOIN generate_series(1, 2) As y
      CROSS JOIN generate_series(0, 2) As Z) As foo;
```

Output

```
BOX3D(0 0 0,4 3 2)
```

相关信息

[ST_Extent](#), [ST_Force3DZ](#), [ST_SetSRID](#)

7.18.7 ST_MakeBox2D

`ST_MakeBox2D` — 建由个 2D 点几何形定形的 BOX2D。

Synopsis

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

描述

建由个 Point 几何形定形的 **box2d**。于行范很有用。

示例

Return all features that fall within 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).

Code

```
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
);
```

相关信息

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

7.18.8 ST_3DMakeBox

ST_3DMakeBox — 由两个 3D 点几何体形定义的 BOX3D。

Synopsis

box3d **ST_3DMakeBox**(geometry point3DLowLeftBottom, geometry point3DUpRightTop);

描述

由两个 3D 点几何体形定义的 **box3d**。



该函数支持 3D 且不会降低 z-index。

更改：2.0.0 在之前的版本中，它曾被称 ST_MakeBox3D

示例

Code

```
SELECT ST_3DMakeBox(ST_MakePoint(-989502.1875, 528439.5625, 10),
  ST_MakePoint(-987121.375 , 529933.1875, 10))
```

Output

```
BOX3D(-989502.1875 528439.5625 10,-987121.375 529933.1875 10)
```

相关信息

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

7.18.9 ST_XMax

ST_XMax — 返回 2D 或 3D 边界框或几何体的 X 最大值。

Synopsis

```
float ST_XMax(box3d aGeomorBox2DorBox3D);
```

描述

返回 2D 或 3D 边界框或几何体的 X 最大值。



Note

虽然此函数对 box3d 定义，但由于自反性，它也适用于 box2d 和几何体。但是，它不会接受几何体或 box2d 文本表示，因为它不会自反。



函数支持 3d 并且不会丢失 z-index。



此方法支持几何字符串和曲线。

示例

Code

```
SELECT ST_XMax('BOX3D(1 2 3,4 5 6)');
```

Output

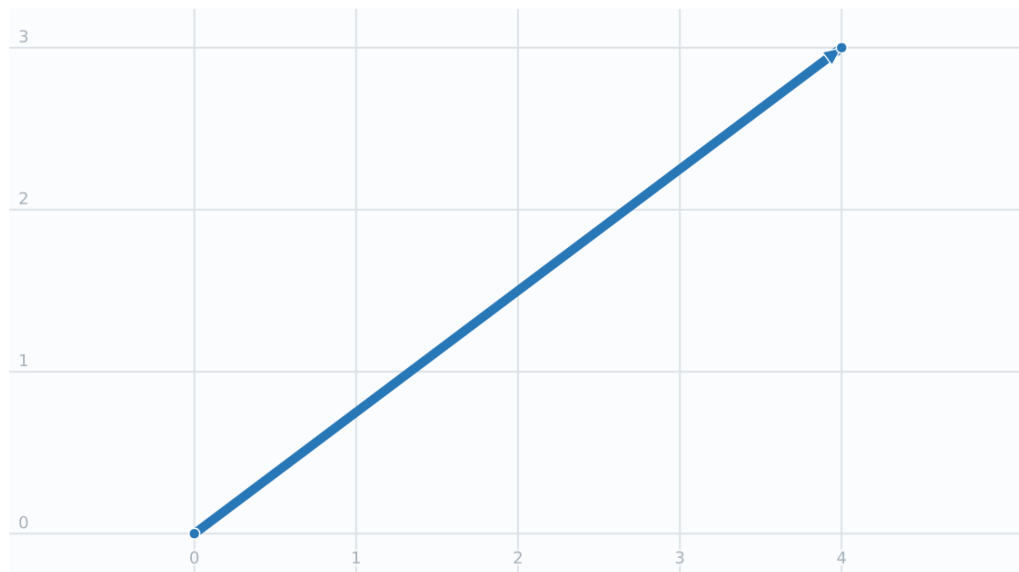
```
4
```

Code

```
SELECT ST_XMax('LINESTRING(1 3 4,5 6 7)::geometry');
```

Output

```
5
```

| Figure**| Code**

```
SELECT ST_XMax(CAST('BOX(-3 2,3 4)' AS box2d));
```

| Output

```
3
```

Observe THIS DOES NOT WORK because it will try to auto-cast the string representation to a BOX3D.

| Code

```
SELECT ST_XMax('LINESTRING(1 3,5 6)');
```

| Output

```
ERROR: BOX3D parser - doesn't start with BOX3D(
```

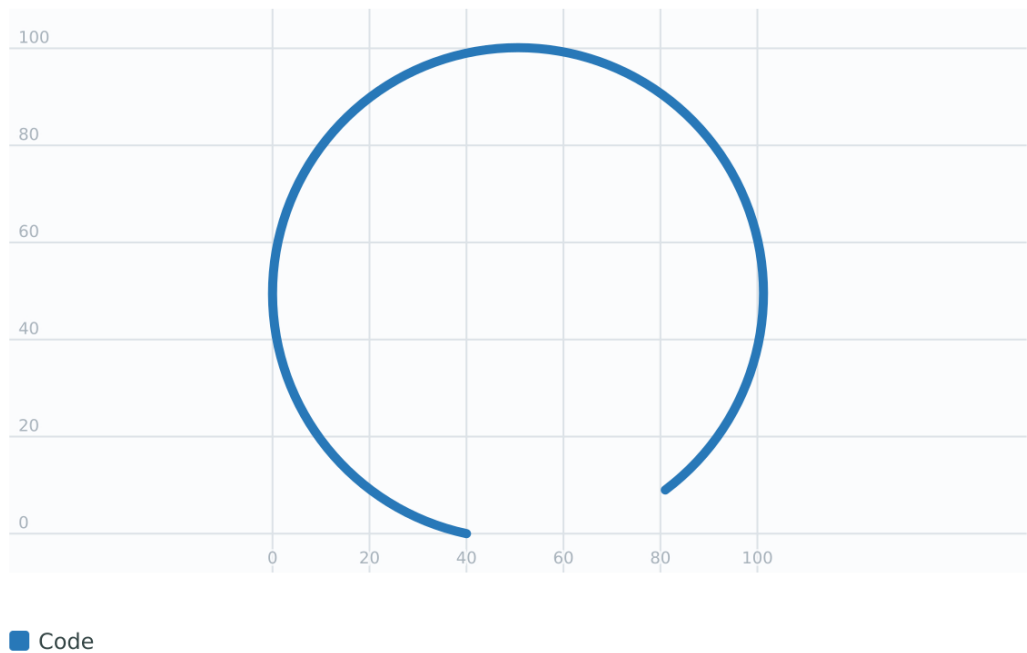
| Code

```
SELECT ST_XMax('CIRCULARSTRING(220268 150415 1,220227 150505 2,220227 150406 3)::geometry) ↵  
;
```

| Output

```
220288.248780547
```

Figure



相关信息

[ST_XMin](#), [ST_YMin](#), [ST_YMax](#), [ST_ZMin](#), [ST_ZMax](#), [ST_MMin](#), [ST_MMax](#)

7.18.10 ST_XMin

ST_XMin — 返回 2D 或 3D 边界框或几何体的 X 最小值。

Synopsis

```
float ST_XMin(box3d aGeomorBox2DorBox3D);
```

描述

返回 2D 或 3D 边界框或几何体的 X 最小值。

Note

虽然此函数对 `box3d` 定义，但由于自 `PostGIS 2.0`，它也适用于 `box2d` 和几何体。但是，它不会接受几何体或 `box2d` 文本表示，因为它不会自 `PostGIS 2.0`。

- ✓ 函数支持 3d 并且不会丢失 z-index。
- ✓ 此方法支持几何字符串和曲线。

示例

Code

```
SELECT ST_XMin('BOX3D(1 2 3,4 5 6)');
```

Output

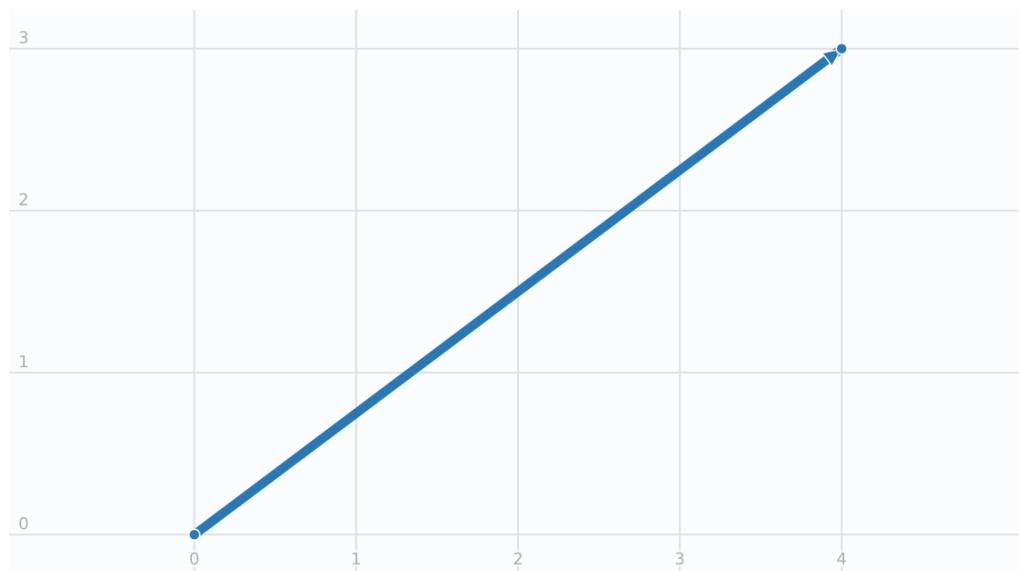
```
1
```

Code

```
SELECT ST_XMin('LINESTRING(1 3 4,5 6 7)::geometry');
```

Output

```
1
```

Figure

■ Code

Code

```
SELECT ST_XMin(CAST('BOX(-3 2,3 4)' AS box2d));
```

Output

```
-3
```

Observe THIS DOES NOT WORK because it will try to auto-cast the string representation to a BOX3D.

Code

```
SELECT ST_XMin('LINESTRING(1 3,5 6)');
```

Output

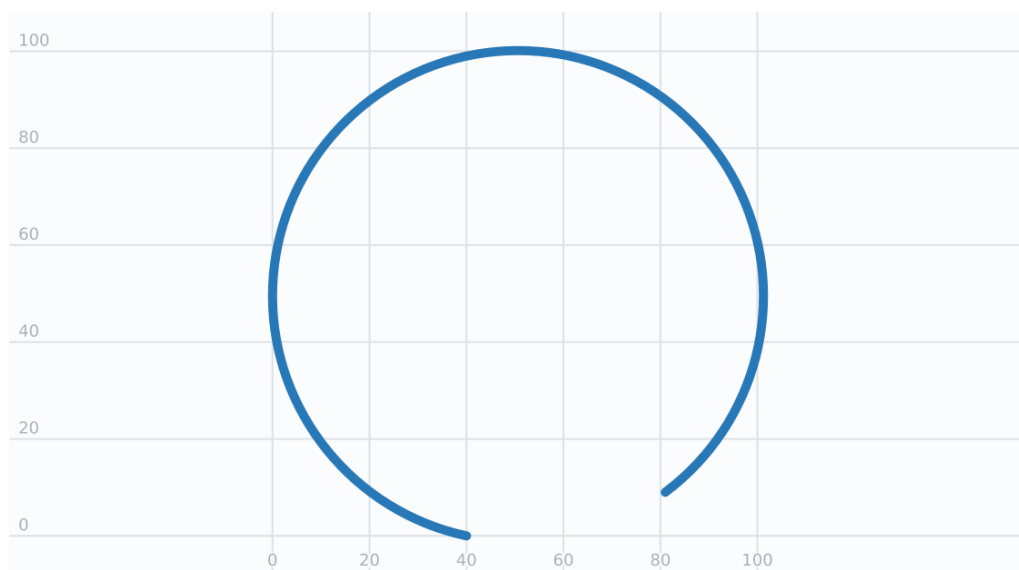
ERROR: BOX3D parser - doesn't start with BOX3D(

Code

```
SELECT ST_XMin('CIRCULARSTRING(220268 150415 1,220227 150505 2,220227 150406 3)')::geometry) ↵  
;
```

Output

220186.995121892

Figure

■ Code

相关信息

[ST_XMax](#), [ST_YMin](#), [ST_YMax](#), [ST_ZMin](#), [ST_ZMax](#), [ST_MMin](#), [ST_MMax](#)

7.18.11 ST_YMax

ST_YMax — 返回 2D 或 3D 边界框或几何体的 Y 最大值。

Synopsis

```
float ST_YMax(box3d aGeomorBox2DorBox3D);
```

描述

返回 2D 或 3D 边界框或几何体的 Y 最大值。



Note

虽然此函数对 `box3d` 定义，但由于自反性，它也适用于 `box2d` 和几何体。但是，它不会接受几何体或 `box2d` 文本表示，因为它不会自反。



函数支持 3d 并且不会丢失 z-index。



此方法支持线形字符串和曲线。

示例

Code

```
SELECT ST_YMax('BOX3D(1 2 3,4 5 6)');
```

Output

5

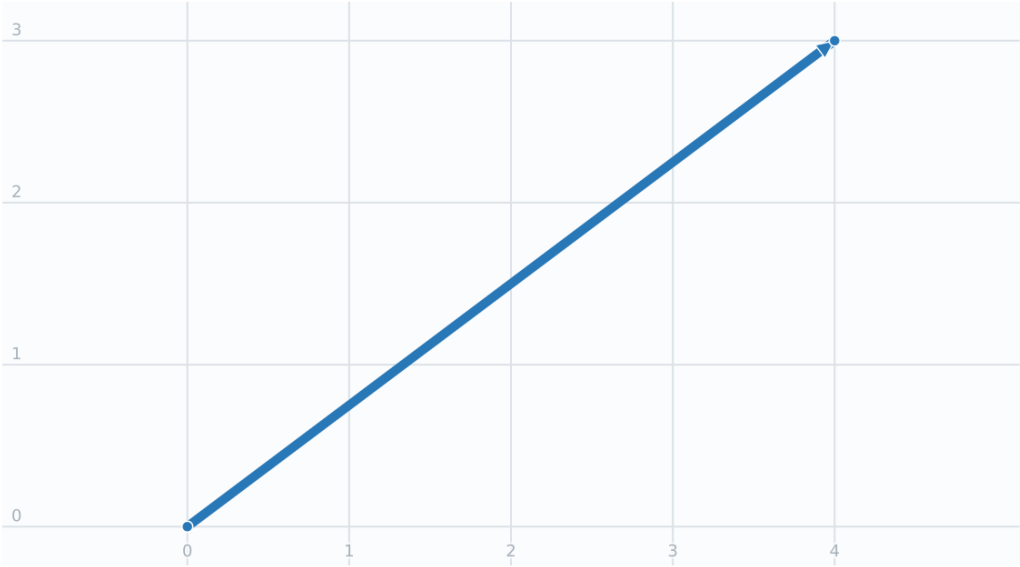
Code

```
SELECT ST_YMax('LINESTRING(1 3 4,5 6 7)::geometry');
```

Output

6

Figure



Code

Code

```
SELECT ST_YMax(CAST('BOX(-3 2,3 4)' AS box2d));
```

Output

```
4
```

Observe THIS DOES NOT WORK because it will try to auto-cast the string representation to a BOX3D.

Code

```
SELECT ST_YMax('LINESTRING(1 3,5 6)');
```

Output

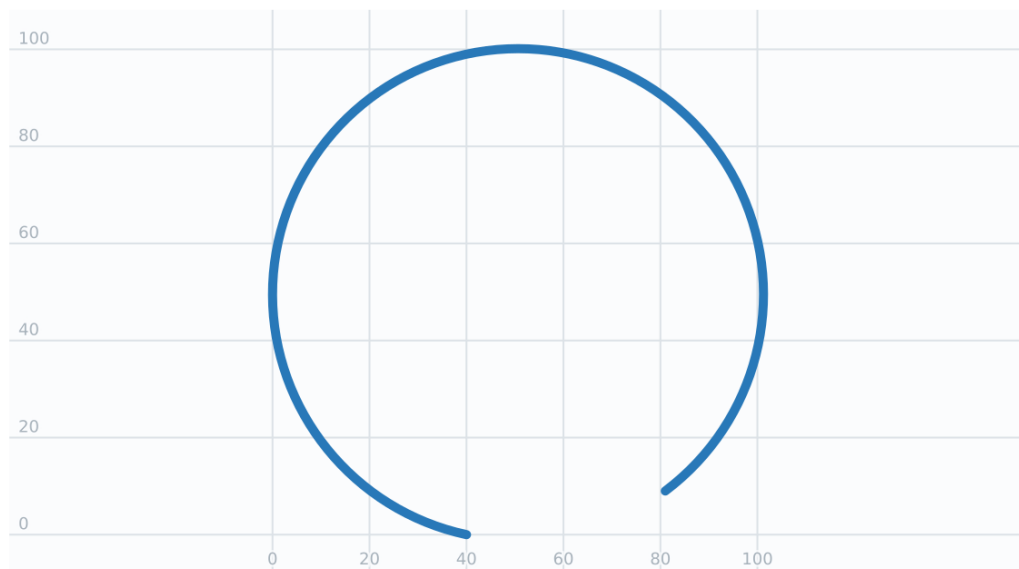
```
ERROR: BOX3D parser - doesn't start with BOX3D(
```

Code

```
SELECT ST_YMax('CIRCULARSTRING(220268 150415 1,220227 150505 2,220227 150406 3)::geometry') ↵  
;
```

Output

```
150506.126829327
```

Figure

■ Code

相关信息

[ST_XMin](#), [ST_XMax](#), [ST_YMin](#), [ST_ZMin](#), [ST_ZMax](#), [ST_MMin](#), [ST_MMax](#)

7.18.12 ST_YMin

ST_YMin — 返回 2D 或 3D 边界框或几何体的 Y 最小值。

Synopsis

float **ST_YMin**(box3d aGeomorBox2DorBox3D);

描述

返回 2D 或 3D 边界框或几何体的 Y 最小值。



Note

虽然此函数对 box3d 定义，但由于自反性，它也适用于 box2d 和几何体。但是，它不会接受几何体或 box2d 文本表示，因为它不会自反。



函数支持 3d 并且不会丢失 z-index。



此方法支持几何字符串和曲线。

示例

Code

```
SELECT ST_YMin('BOX3D(1 2 3,4 5 6)');
```

Output

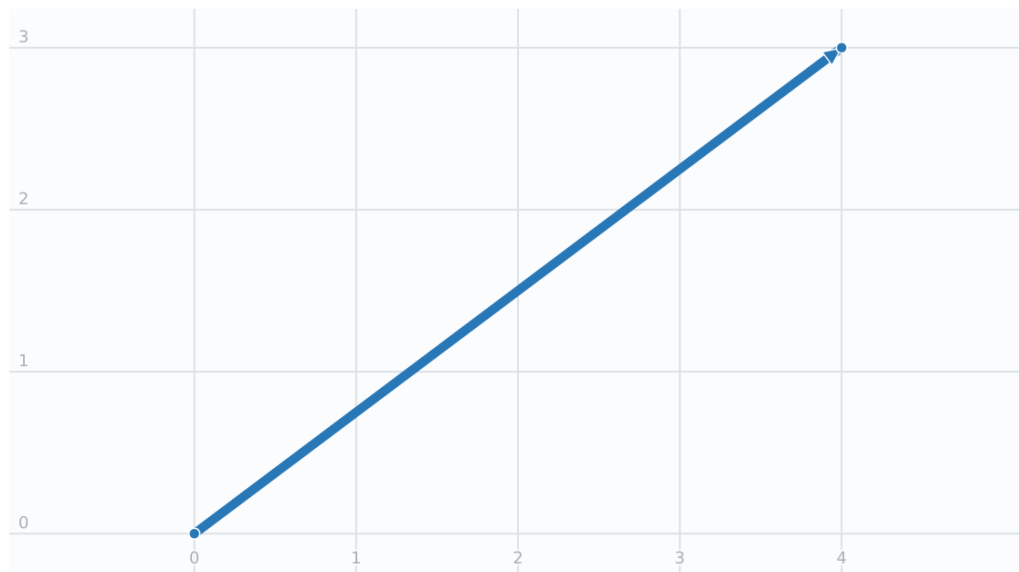
```
2
```

Code

```
SELECT ST_YMin('LINESTRING(1 3 4,5 6 7)::geometry');
```

Output

```
3
```

| Figure

■ Code

| Code

```
SELECT ST_YMin(CAST('BOX(-3 2,3 4)' AS box2d));
```

| Output

2

Observe THIS DOES NOT WORK because it will try to auto-cast the string representation to a BOX3D.

| Code

```
SELECT ST_YMin('LINESTRING(1 3,5 6)');
```

| Output

ERROR: BOX3D parser - doesn't start with BOX3D(

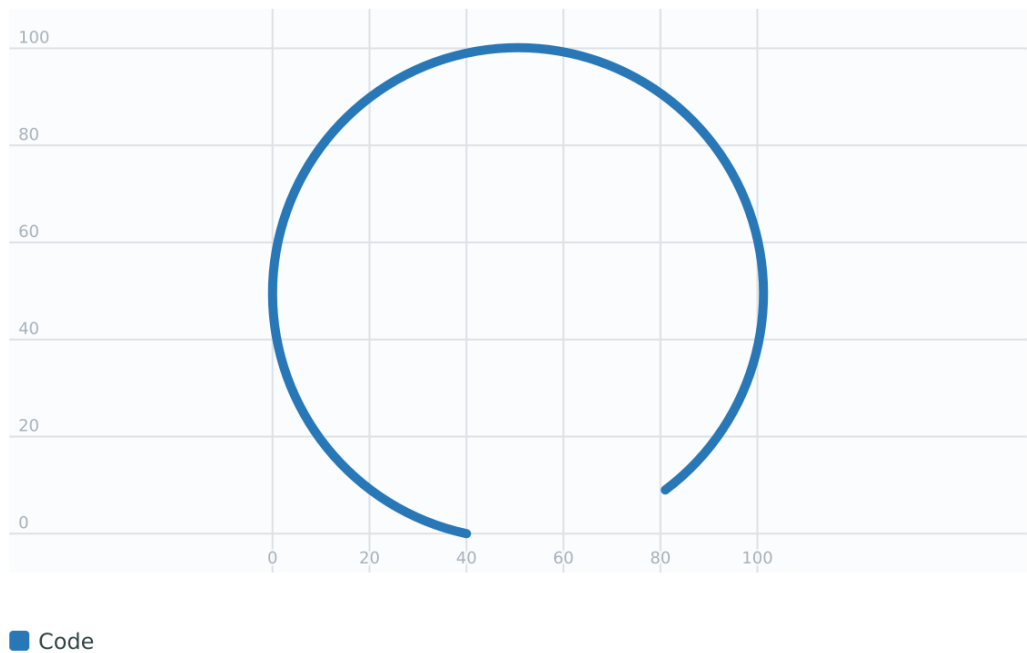
| Code

```
SELECT ST_YMin('CIRCULARSTRING(220268 150415 1,220227 150505 2,220227 150406 3)::geometry) ↵  
;
```

| Output

150406

Figure



相关信息

[ST_GeomFromEWKT](#), [ST_XMin](#), [ST_XMax](#), [ST_YMax](#), [ST_ZMin](#), [ST_ZMax](#), [ST_MMin](#), [ST_MMax](#)

7.18.13 ST_ZMax

ST_ZMax — 返回 2D 或 3D 边界框或几何体的 Z 最大值。

Synopsis

```
float ST_ZMax(box3d aGeomorBox2DorBox3D);
```

描述

返回 2D 或 3D 边界框或几何体的 Z 最大值。

Note

虽然此函数仅对 box3d 定义，但由于自反性，它也适用于 box2d 和几何体。但是，它不会接受几何体或 box2d 文本表示，因为它不会自反。

- ✓ 函数支持 3d 并且不会丢失 z-index。
- ✓ 此方法支持几何字符串和曲线。

示例

Code

```
SELECT ST_ZMax('BOX3D(1 2 3,4 5 6)');
```

Output

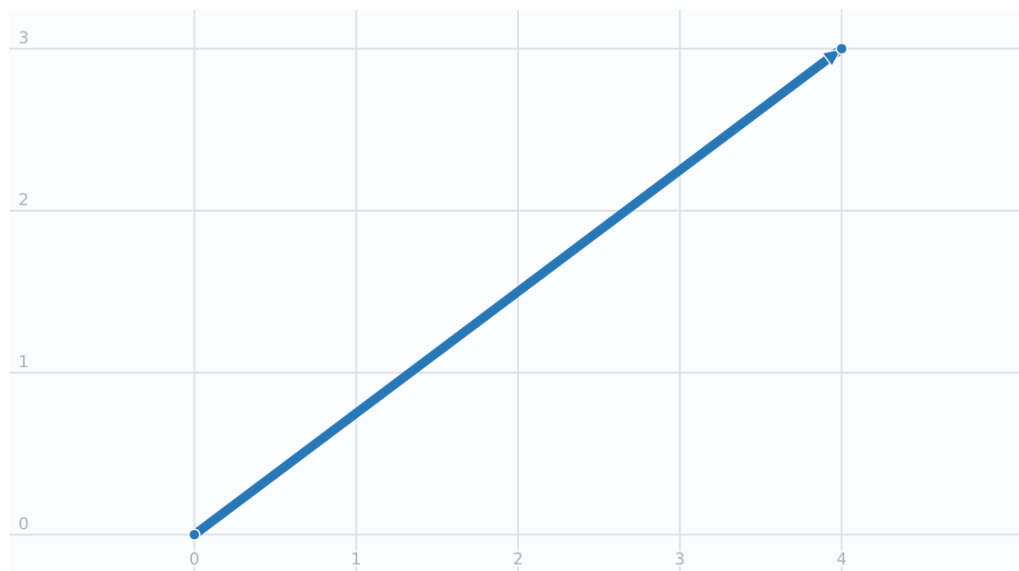
6

Code

```
SELECT ST_ZMax('LINESTRING(1 3 4,5 6 7)::geometry');
```

Output

7

Figure

■ Code

Code

```
SELECT ST_ZMax('BOX3D(-3 2 1,3 4 1)');
```

Output

1

Observe THIS DOES NOT WORK because it will try to auto-cast the string representation to a BOX3D.

Code

```
SELECT ST_ZMax('LINESTRING(1 3 4,5 6 7)');
```

| Output

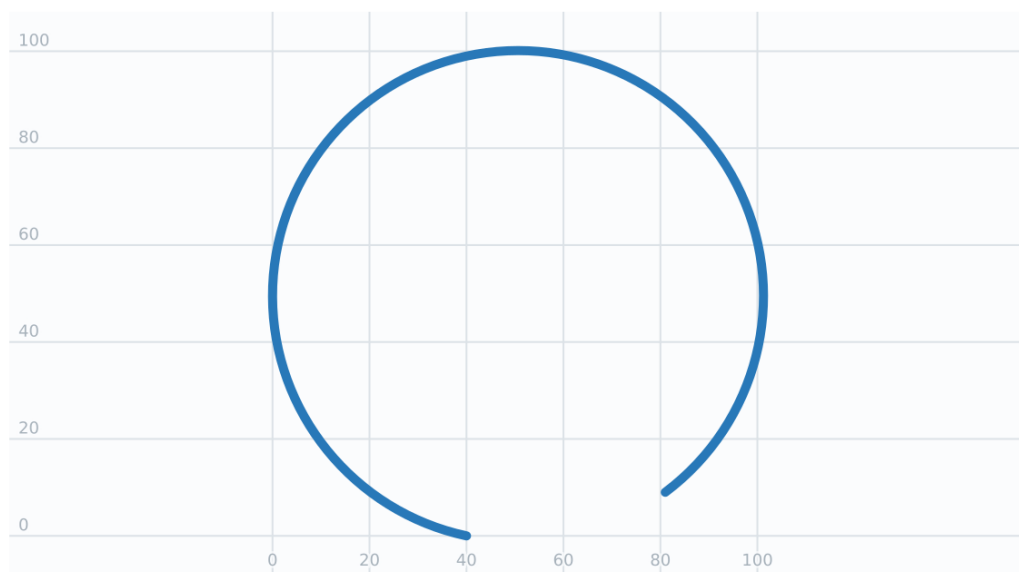
ERROR: BOX3D parser - doesn't start with BOX3D(

| Code

```
SELECT ST_ZMax('CIRCULARSTRING(220268 150415 1,220227 150505 2,220227 150406 3)')::geometry) ↵  
;
```

| Output

3

| Figure

■ Code

相关信息

[ST_GeomFromEWKT](#), [ST_XMin](#), [ST_XMax](#), [ST_YMin](#), [ST_YMax](#), [ST_ZMin](#), [ST_MMin](#), [ST_MMax](#)

7.18.14 ST_ZMin

ST_ZMin — 返回 2D 或 3D 边界框或几何体的 Z 最小值。

Synopsis

```
float ST_ZMin(box3d aGeomorBox2DorBox3D);
```

描述

返回 2D 或 3D 边界框或几何体的 Z 最小值。

**Note**

虽然此函数对 `box3d` 定义，但由于自反性，它也适用于 `box2d` 和几何体。但是，它不会接受几何体或 `box2d` 文本表示，因为它不会自反。



该函数支持 3d 并且不会丢失 z-index。



此方法支持线形字符串和曲线。

示例**Code**

```
SELECT ST_ZMin('BOX3D(1 2 3,4 5 6)');
```

Output

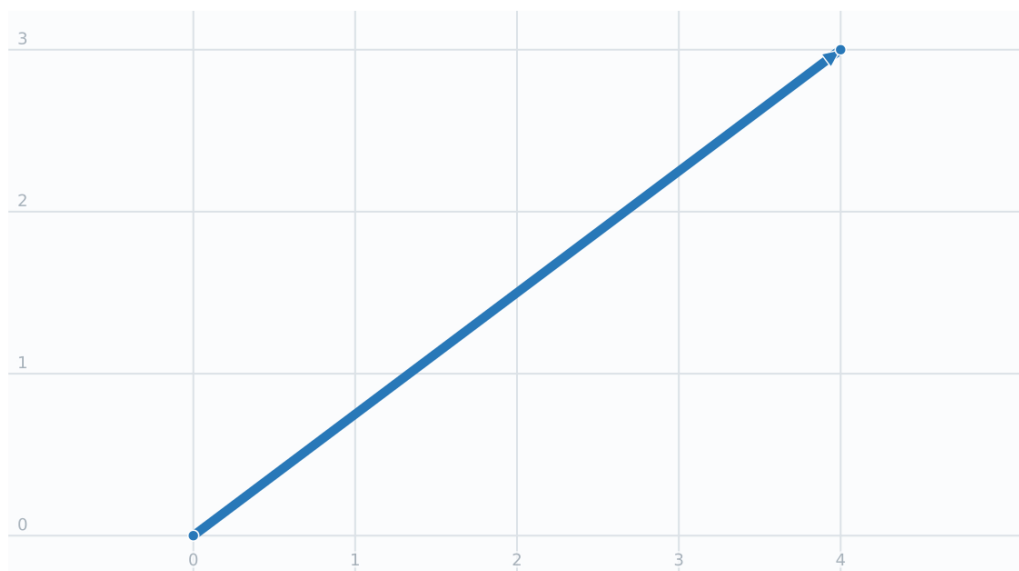
```
3
```

Code

```
SELECT ST_ZMin('LINESTRING(1 3 4,5 6 7)::geometry');
```

Output

```
4
```

Figure

■ Code

Code

```
SELECT ST_ZMin('BOX3D(-3 2 1,3 4 1)');
```

Output

```
1
```

Observe THIS DOES NOT WORK because it will try to auto-cast the string representation to a BOX3D.

Code

```
SELECT ST_ZMin('LINESTRING(1 3 4,5 6 7)');
```

Output

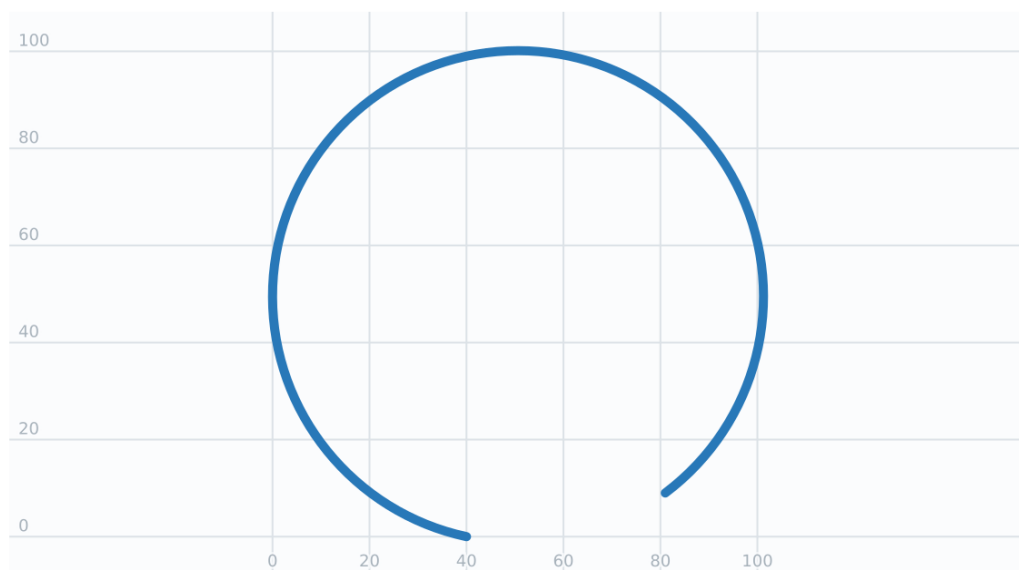
```
ERROR: BOX3D parser - doesn't start with BOX3D(
```

Code

```
SELECT ST_ZMin('CIRCULARSTRING(220268 150415 1,220227 150505 2,220227 150406 3)::geometry) ↵  
;
```

Output

```
1
```

Figure

■ Code

相关信息

[ST_GeomFromEWKT](#), [ST_GeomFromText](#), [ST_XMin](#), [ST_XMax](#), [ST_YMin](#), [ST_YMax](#), [ST_ZMax](#), [ST_MMin](#), [ST_MMax](#)

7.18.15 ST_XSize

ST_XSize — Returns the X size of a 2D or 3D bounding box or a geometry.

Synopsis

```
float ST_XSize(box3d aGeomorBox2DorBox3D);
```

描述

Returns the X size of a 2D or 3D bounding box or a geometry, computed as ST_XMax(aGeomorBox2DorBox3D) - ST_XMin(aGeomorBox2DorBox3D).



Note

虽然此函数对 box3d 定义，但由于自反性，它也适用于 box2d 和几何。但是，它不会接受几何或 box2d 文本表示，因为它不会自反。



函数支持 3d 并且不会丢失 z-index。



此方法支持 WKB 字符串和曲线。

Availability: 3.7.0

示例

Code

```
SELECT ST_XSize('BOX3D(1 2 3,4 5 6)');
```

Output

```
3
```

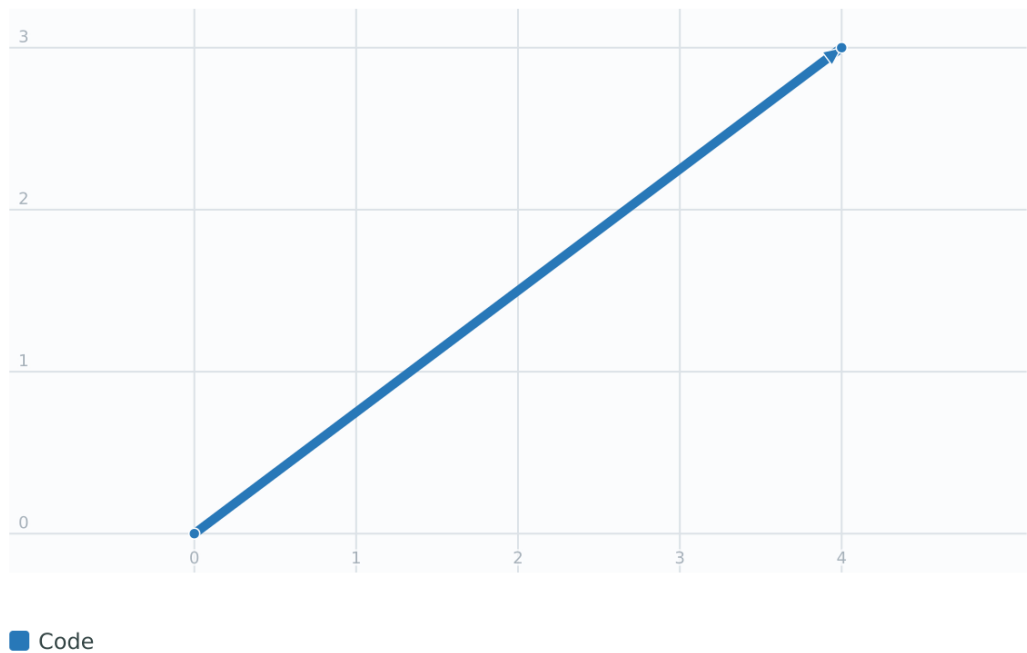
Code

```
SELECT ST_XSize('LINESTRING(1 3,5 6)::geometry');
```

Output

```
4
```

Figure



相关信息

ST_XMin, ST_XMax, ST_YSize, ST_ZSize, ST_MSize

7.18.16 ST_YSize

ST_YSize — Returns the Y size of a 2D or 3D bounding box or a geometry.

Synopsis

```
float ST_YSize(box3d aGeomorBox2DorBox3D);
```

描述

Returns the Y size of a 2D or 3D bounding box or a geometry, computed as ST_YMax(aGeomorBox2DorBox3D) - ST_YMin(aGeomorBox2DorBox3D).



Note 然此函数仅对 box3d 定义，但由于自反性，它也适用于 box2d 和几何体。但是，它不会接受几何体形或 box2d 文本表示，因为它不会自反。

- ✓ 函数支持 3d 并且不会丢失 z-index。
- ✓ 此方法支持几何字符串和曲线。

Availability: 3.7.0

示例

Code

```
SELECT ST_YSize('BOX3D(1 2 3,4 5 6)');
```

Output

```
3
```

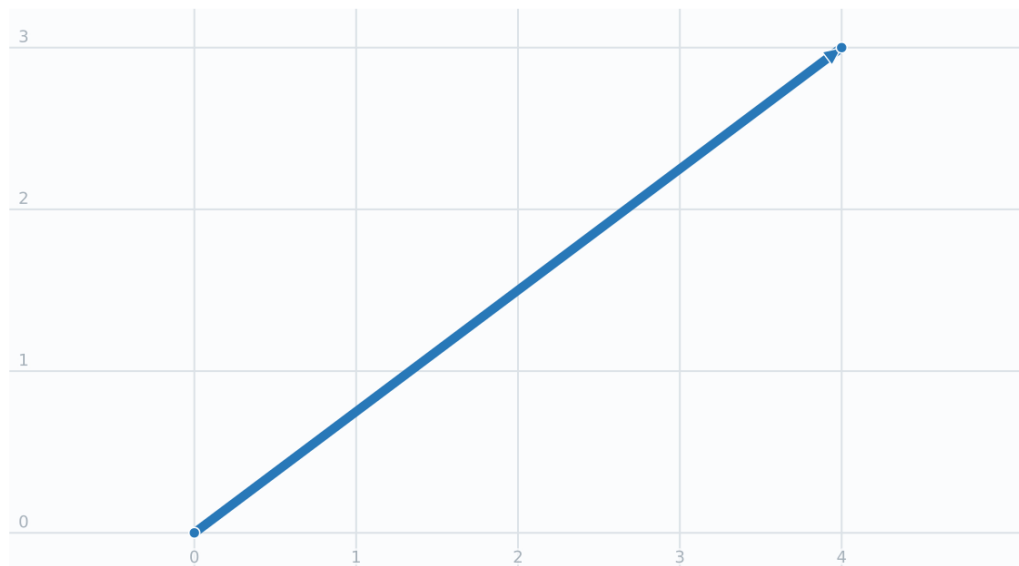
Code

```
SELECT ST_YSize('LINESTRING(1 3,5 6)::geometry');
```

Output

```
3
```

Figure



■ Code

相关信息

[ST_YMin](#), [ST_YMax](#), [ST_XSize](#), [ST_ZSize](#), [ST_MSize](#)

7.18.17 ST_ZSize

ST_ZSize — Returns the Z size of a 2D or 3D bounding box or a geometry.

Synopsis

```
float ST_ZSize(box3d aGeomorBox2DorBox3D);
```

描述

Returns the Z size of a 2D or 3D bounding box or a geometry, computed as `ST_ZMax(aGeomorBox2DorBox3D) - ST_ZMin(aGeomorBox2DorBox3D)`.



Note

虽然此函数对 box3d 定义，但由于自反性，它也适用于 box2d 和几何体。但是，它不会接受几何体或 box2d 文本表示，因为它不会自反。

- ✓ 函数支持 3d 并且不会丢失 z-index。
- ✓ 此方法支持几何字符串和曲线。

Availability: 3.7.0

示例

Code

```
SELECT ST_ZSize('BOX3D(1 2 3,4 5 6)');
```

Output

3

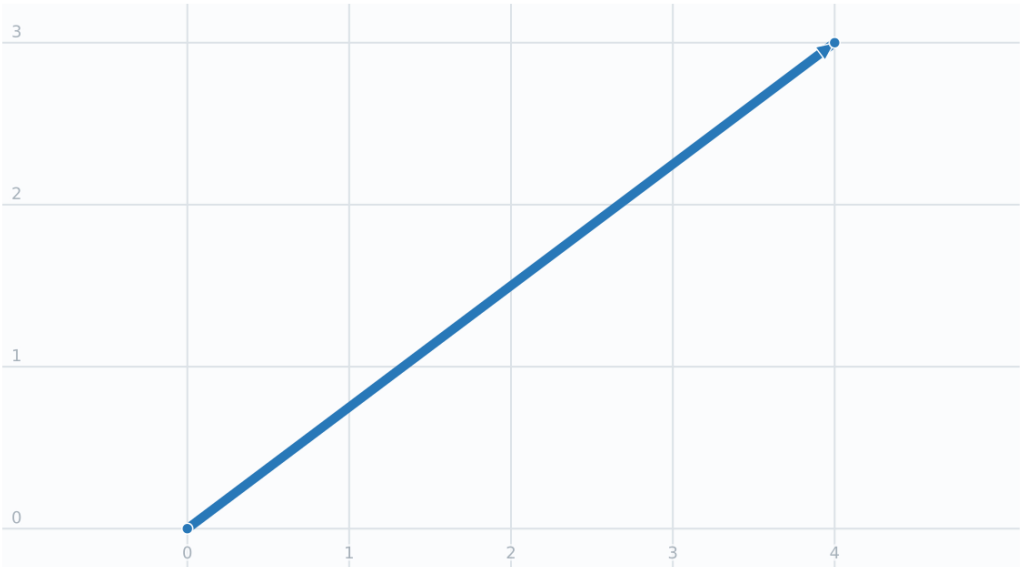
Code

```
SELECT ST_ZSize('LINESTRING(1 3 4,5 6 7)::geometry');
```

Output

3

Figure



Code

相关信息

[ST_ZMin](#), [ST_ZMax](#), [ST_XSize](#), [ST_YSize](#), [ST_MSize](#)

7.18.18 ST_MMin

ST_MMin — 返回几何对象的 M 最小值。

Synopsis

float **ST_MMin**(geometry geom);

描述

Returns the M minima of a geometry, or null if the geometry lacks M values.



☑函数支持 3d 并且不会☑失 z-index。



☑此方法支持☑形字符串和曲线。

示例

Code

```
SELECT ST_MMin('POINT M (1 2 3)');
```

Output

```
3
```

相关信息

[ST_XMin](#), [ST_XMax](#), [ST_YMin](#), [ST_YMax](#), [ST_ZMin](#), [ST_ZMax](#), [ST_MMax](#), [ST_MSize](#)

7.18.19 ST_MMax

ST_MMax — Returns the M maxima of a geometry.

Synopsis

float **ST_MMax**(geometry geom);

描述

Returns the M maxima of a geometry, or null if the geometry lacks M values.



☑函数支持 3d 并且不会☑失 z-index。



☑此方法支持☑形字符串和曲线。

示例

Code

```
SELECT ST_MMax('POINT M (1 2 3)');
```

Output

```
3
```

相关信息

[ST_XMin](#), [ST_XMax](#), [ST_YMin](#), [ST_YMax](#), [ST_ZMin](#), [ST_ZMax](#), [ST_MMin](#), [ST_MSize](#)

7.18.20 ST_MSize

ST_MSize — Returns the M size of a geometry.

Synopsis

float **ST_MSize**(geometry geom);

描述

Returns the M size of a geometry, computed as `ST_MMax(geom) - ST_MMin(geom)`, or null if the geometry lacks M values.



☑函数支持 3d 并且不会☑失 z-index。



☑此方法支持☑形字符串和曲☑。

Availability: 3.7.0

示例

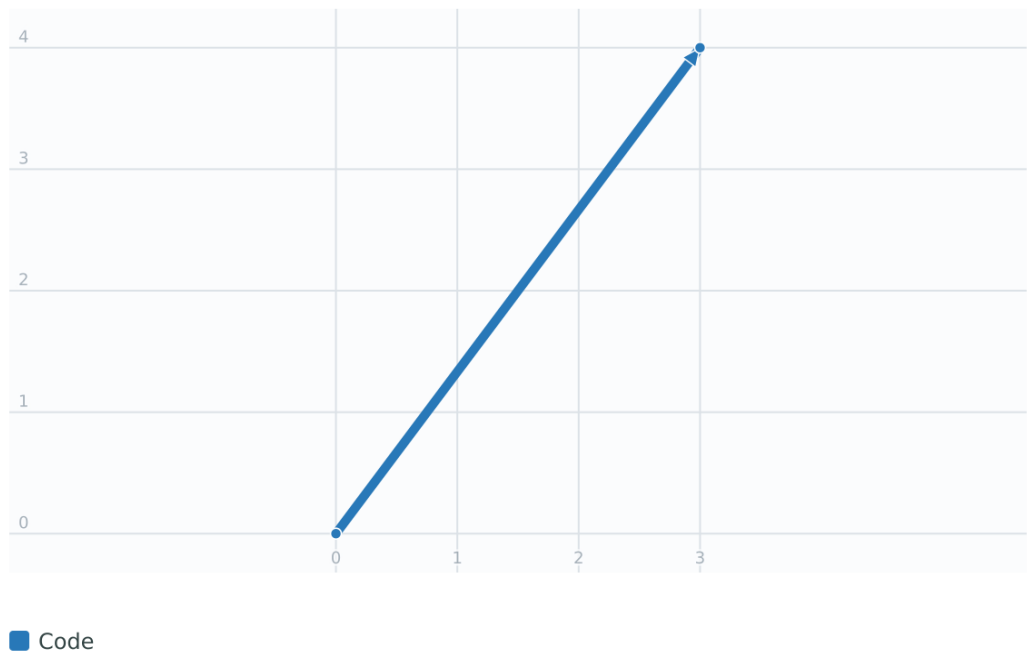
Code

```
SELECT ST_MSize('LINESTRING M (1 2 3,4 6 8)');
```

Output

```
5
```

Figure



相关信息

[ST_MMin](#), [ST_MMax](#), [ST_XSize](#), [ST_YSize](#), [ST_ZSize](#)

7.19 线性参考

7.19.1 ST_LineInterpolatePoint

`ST_LineInterpolatePoint` — 返回沿线在百分比指示位置的插点。

Synopsis

```
geometry ST_LineInterpolatePoint(geometry a_linestring, float8 a_fraction);
geography ST_LineInterpolatePoint(geography a_linestring, float8 a_fraction, boolean use_spheroid = true);
```

描述

返回沿线在百分比指示位置的插点。第一个参数是线串。第二个参数是介于 0 和 1 之间的浮点数，它表示点的位置与线串长度的比率。如果存在 Z 和 M 值，则进行插值计算。

参考 [ST_LineLocatePoint](#) 以计算最接近点的线位置。



Note

此函数计算 2D 中的点，然后对 Z 和 M 的值进行插值，而 [ST_3DLineInterpolatePoint](#) 计算 3D 中的点，然后对 M 值进行插值。

**Note**

从版本 1.1.1 开始，此函数会插入 M 和 Z （如果存在），而之前的版本将它置 0.0。

可用性：0.8.2，1.1.1 中添加支持 Z 和 M

更改：2.1.0。在 2.0.x 之前，称 ST_Line_Interpolate_Point。



函数支持 3d 并且不会失 z-index。

示例

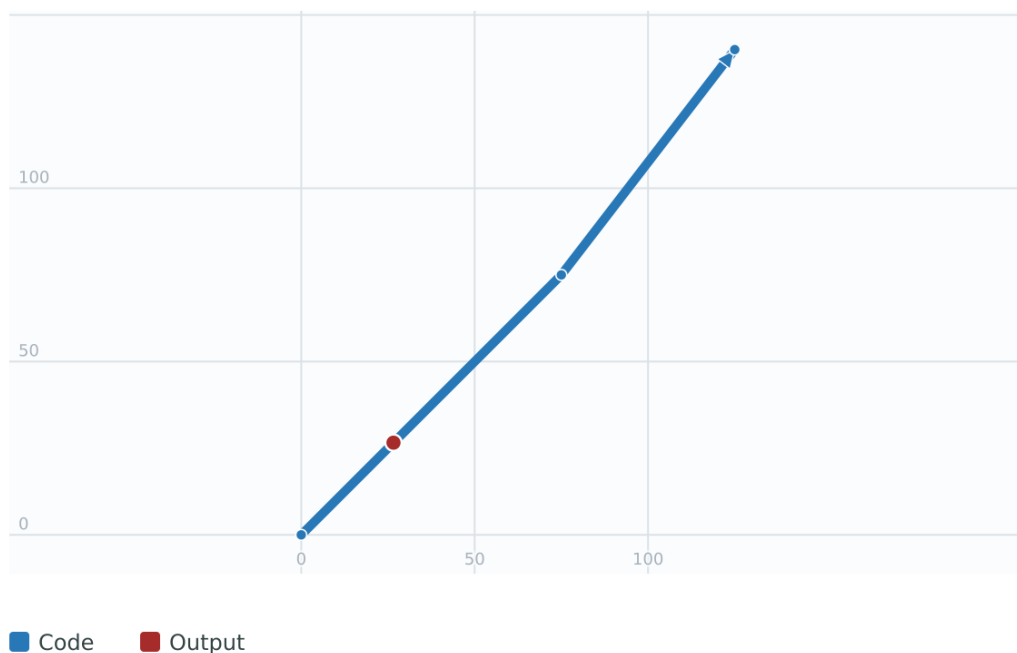
This example returns the point 20 percent along a line.

Code

```
SELECT ST_LineInterpolatePoint(  
'LINESTRING(25 50,100 125,150 190)',  
0.2);
```

Output

```
POINT(51.5974135047432 76.5974135047432)
```

Figure

3D 的中点：

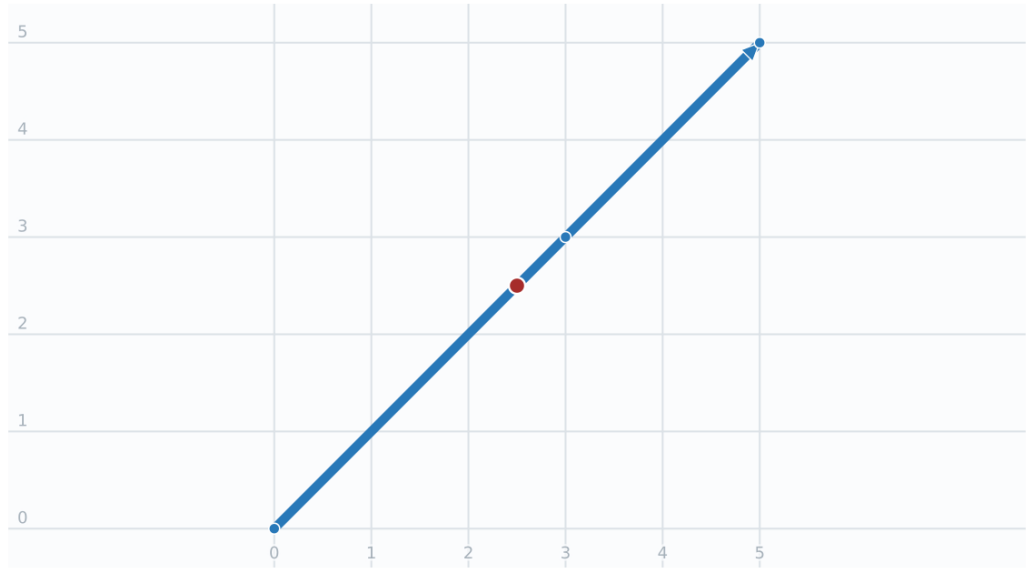
Code

```
SELECT ST_LineInterpolatePoint(  
'LINESTRING(1 2 3,4 5 6,6 7 8)',  
0.5);
```

Output

POINT(3.5 4.5 5.5)

Figure



■ Code ■ Output

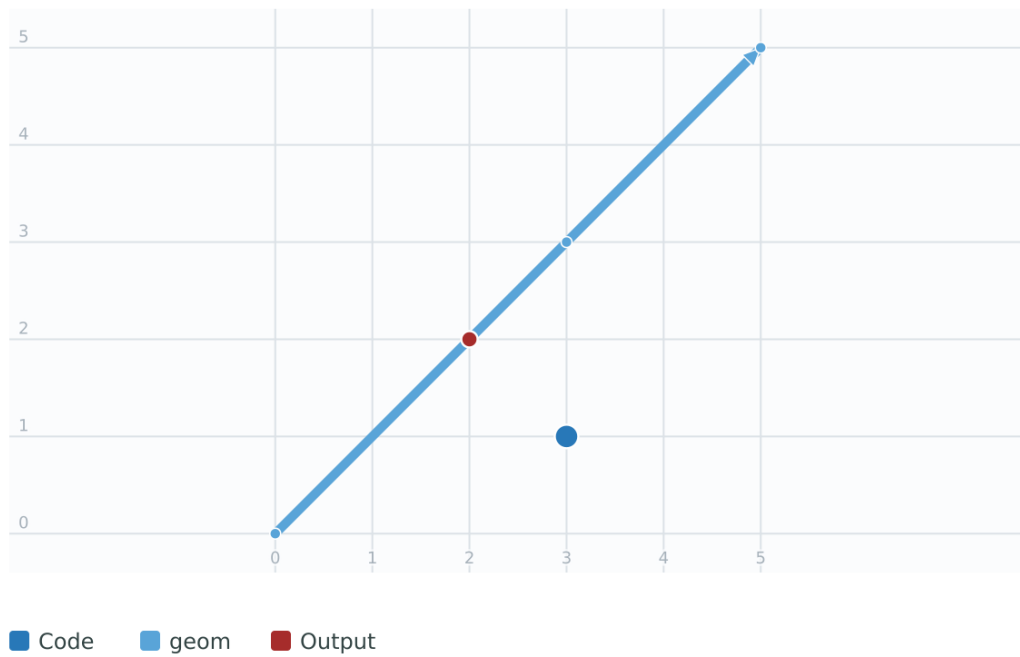
直线上距离某点最近的点：

Code

```
SELECT ST_LineInterpolatePoint(  
    line.geom,  
    ST_LineLocatePoint(line.geom, 'POINT(4 3)'))  
FROM (SELECT 'LINESTRING(1 2,4 5,6 7)::geometry As geom) AS line;
```

Output

POINT(3 4)

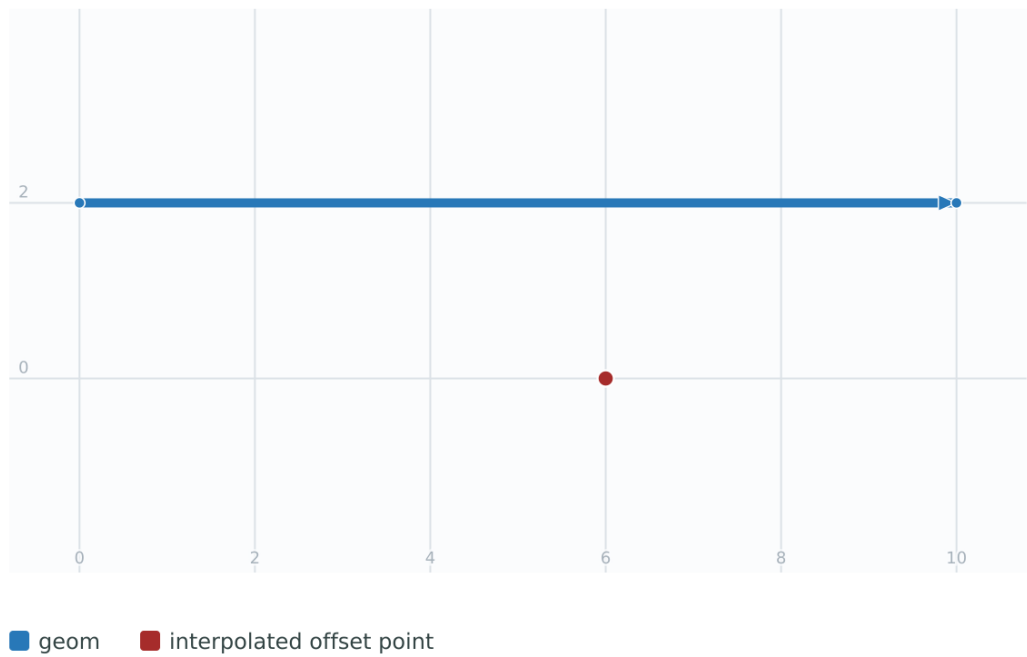
Figure

An interpolated point can be offset to the left or right of a line by using `ST_OffsetCurve` before interpolation. This is useful for locating an address point beside a street centerline.

Code

```
WITH street AS (  
    SELECT 'LINESTRING(0 0, 10 0)::geometry AS geom,  
           100 AS from_num,  
           200 AS to_num  
)  
SELECT  
    ST_LineInterpolatePoint(  
        ST_OffsetCurve(geom, -2),  
        (160 - from_num)::float8 / (to_num - from_num)  
    ) AS interpolated_offset_point  
FROM street;
```

Figure



相关信息

[ST_LineInterpolatePoints](#), [ST_3DLineInterpolatePoint](#), [ST_LineLocatePoint](#), [ST_OffsetCurve](#)

7.19.2 ST_3DLineInterpolatePoint

`ST_3DLineInterpolatePoint` — 返回沿 3D 线的小数指示位置插值的点。

Synopsis

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

描述

返回沿 3D 线的小数指示位置插值的点。第一个参数必须是 `LINESTRING`。第二个参数是 0 到 1 之间的浮点数，表示点位置占线度的一部分。如果存在 M 值，则将其行插值。



Note
`ST_LineInterpolatePoint` 计算 2D 中的点，然后对 Z 和 M 的行插值，而此函数计算 3D 中的点，并且对 M 行插值。

可用性 : 3.0.0



该函数支持 3d 并且不会丢失 z-index。

示例

沿 3D 线返回 20% 的点

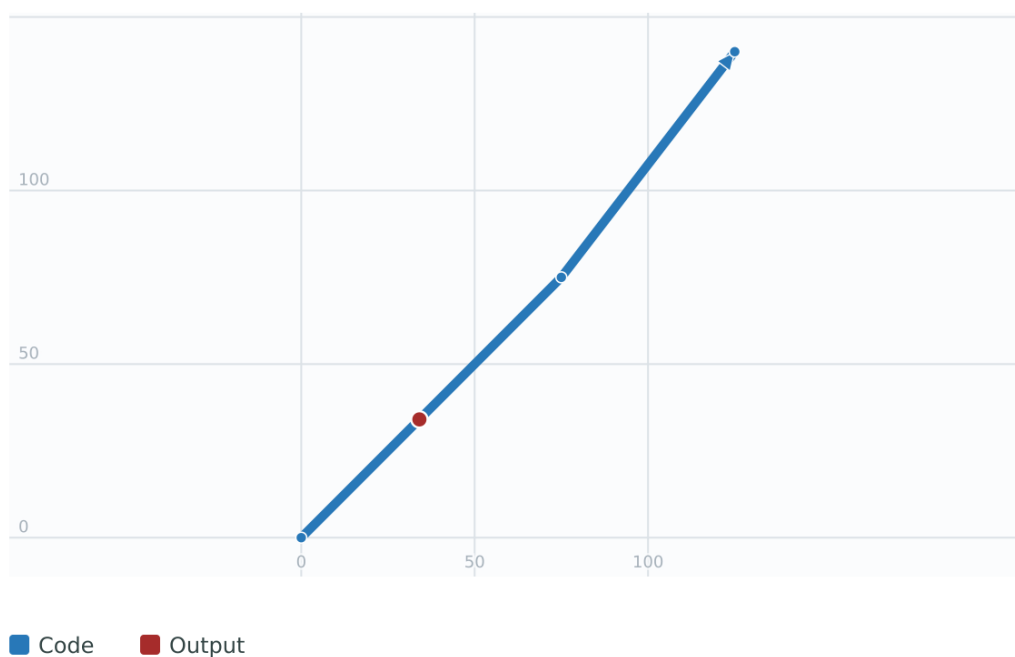
Code

```
SELECT ST_3DLineInterpolatePoint(  
'LINESTRING(25 50 70,100 125 90,150 190 200)',  
0.20);
```

Output

```
POINT Z (59.06758929108215 84.06758929108216 79.0846904776219)
```

Figure



相关信息

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

7.19.3 ST_LineInterpolatePoints

ST_LineInterpolatePoints — 返回沿直线以分数间隔插值的点。

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);
```

描述

返回沿一条以分数间隔插的一个或多个点。第一个参数必须是 `LINESTRING`。第二个参数是一个介于 `float8` 0 和 1 之间的值，表示点之间的距作总长度的一部分。如果第三个参数 `false`，最多创建一个点（相当于 `ST_LineInterpolatePoint`。）

如果结果有零个或一个点，将其作为 `POINT` 返回。如果它有多个或更多点，以 `MULTIPOINT` 形式返回。

可用性：2.5.0

✓ 函数支持 3d 并且不会丢失 z-index。

✓ 功能支持 M 坐标。

示例

This example returns points every 20 percent along a two-dimensional line.

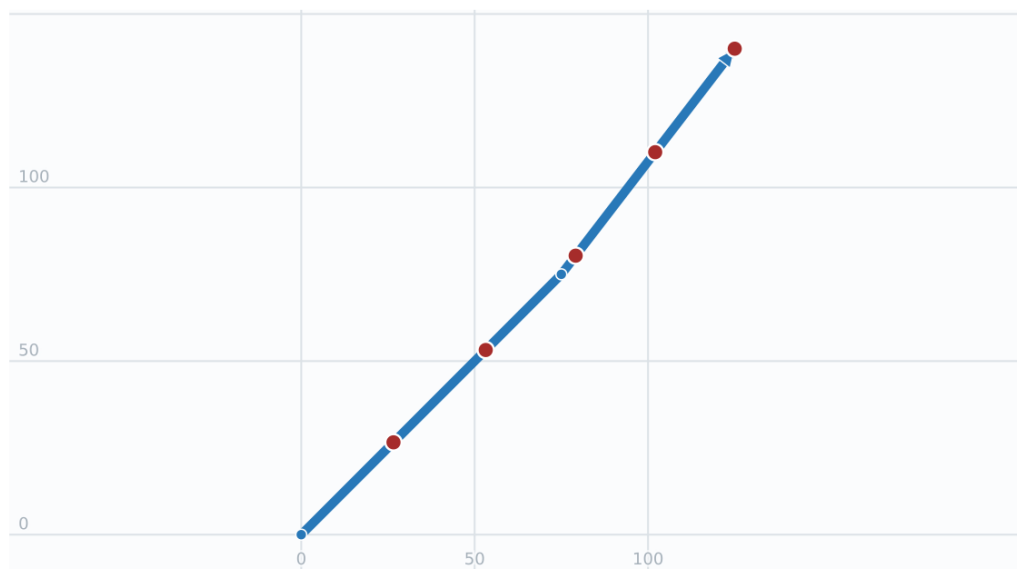
Code

```
SELECT ST_LineInterpolatePoints(  
'LINESTRING(25 50,100 125,150 190)',  
0.20);
```

Output

```
MULTIPOINT((51.5974135047432 76.5974135047432),(78.1948270094864 103.194827009486),(104.132163186446 130.37181214238) ↔  
,(127.066081593223 160.18590607119),(150 190))
```

Figure



■ Code ■ Output

相关信息

[ST_LineInterpolatePoint](#), [ST_LineLocatePoint](#)

7.19.4 ST_LineLocatePoint

ST_LineLocatePoint — 返回线上最接近点的分数位置。

Synopsis

```
float8 ST_LineLocatePoint(geometry a_linestring, geometry a_point);  
float8 ST_LineLocatePoint(geography a_linestring, geography a_point, boolean use_spheroid = true);
```

描述

返回 0 到 1 之间的浮点数，表示 LineString 上距离指定 Point 最近的点的位置，作为 2d 度的一部分。

您可以使用返回的位置来提取点 ([ST_LineInterpolatePoint](#)) 或子字符串 ([ST_LineSubstring](#))。

对于近似地址数量很有用

可用性：1.1.0

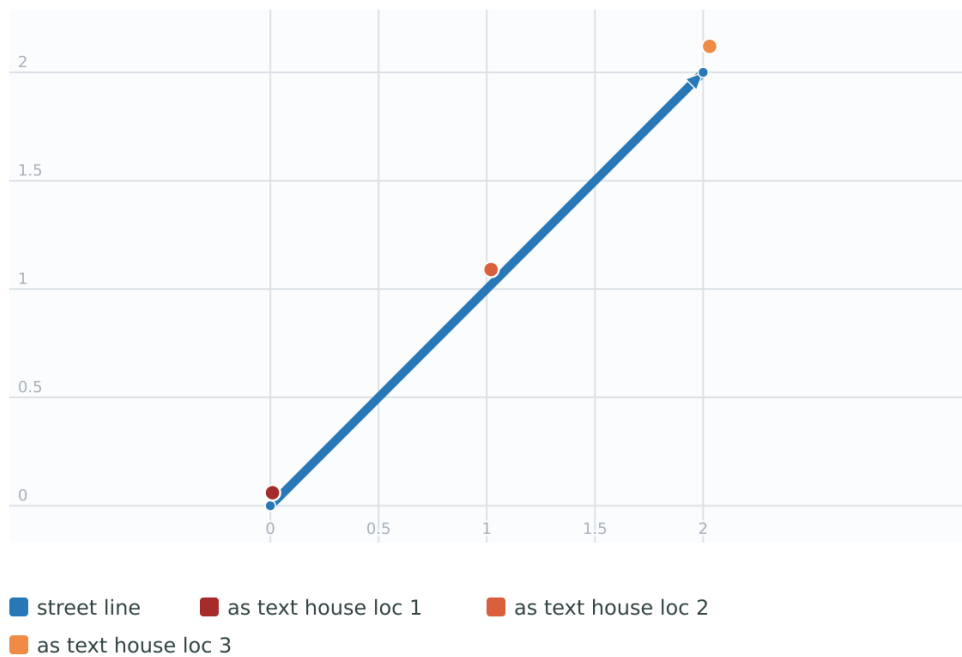
更改：2.1.0。在 2.0.x 之前，名称 ST_Line_Locate_Point。

示例

This example approximates the street number of a house by locating the closest point on a street line. The subquery generates sample house and street data, and ST_DWithin excludes houses that are too far from the street.

Code

```
SELECT house_loc As as_text_house_loc,  
       start_house_num +  
         CAST( (end_house_num - start_house_num)  
             * ST_LineLocatePoint(street_line, house_loc) As integer) As house_num  
FROM  
(SELECT 'LINESTRING(1 2,3 4)::geometry As street_line,  
       ST_Point(x*1.01, y*1.03) As house_loc, 10 As start_house_num,  
       20 As end_house_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);
```

Figure

This example finds the closest point on a line to a point or other geometry.

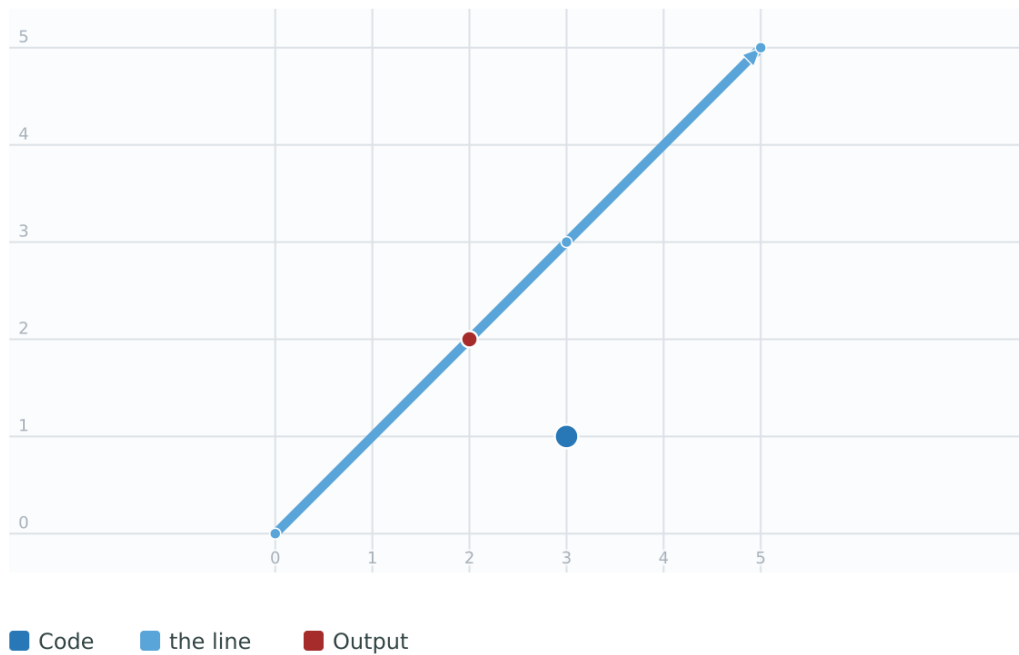
Code

```
SELECT ST_LineInterpolatePoint(  
    foo.the_line,  
    ST_LineLocatePoint(foo.the_line, 'POINT(4 3)::geometry'))  
FROM (SELECT 'LINESTRING(1 2,4 5,6 7)::geometry' As the_line) As foo;
```

Output

```
POINT(3 4)
```


Figure

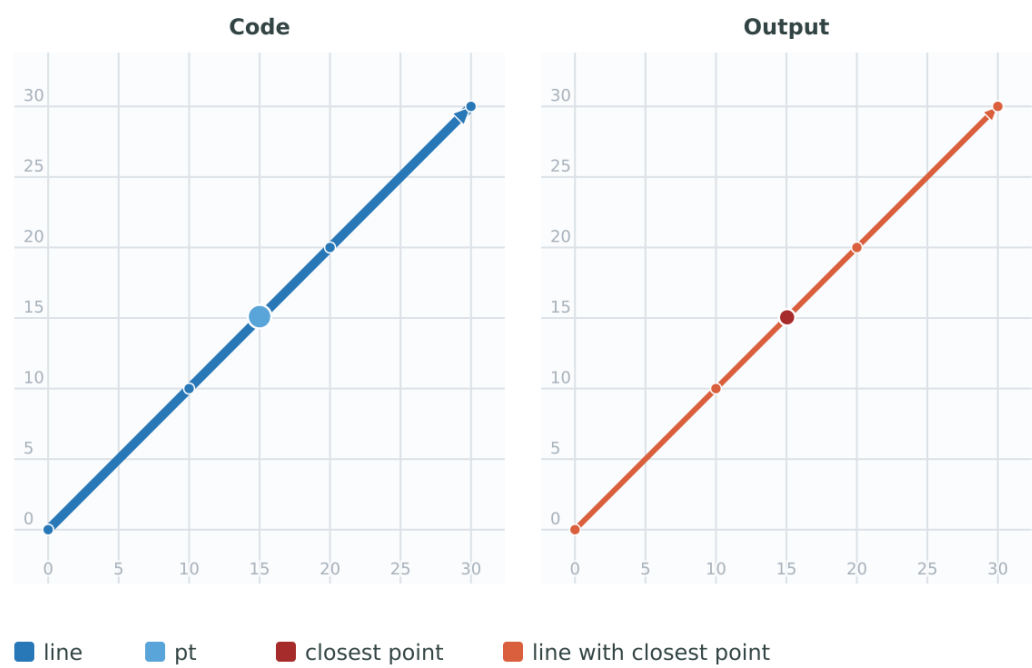


This example locates the closest segment and point on a LineString.

Code

```
WITH data AS (
  SELECT
    'LINESTRING (0 0, 10 10, 20 20, 30 30)::geometry AS line,
    'POINT(15 15.1)::geometry AS pt
),
segments AS (
  SELECT
    s.i,
    d.line,
    d.pt,
    ST_MakeLine(ST_PointN(d.line, s.i), ST_PointN(d.line, s.i + 1)) AS segment
  FROM data AS d
  CROSS JOIN LATERAL generate_series(1, ST_NumPoints(d.line) - 1) AS s(i)
)
SELECT
  i AS segment_number,
  round(ST_Distance(segment, pt)::numeric, 3) AS distance,
  ST_ClosestPoint(segment, pt) AS closest_point,
  ST_AddPoint(line, ST_ClosestPoint(segment, pt), i) AS line_with_closest_point
FROM segments
ORDER BY ST_Distance(segment, pt)
LIMIT 1;
```

Figure



相关信息

[ST_AddPoint](#), [ST_ClosestPoint](#), [ST_DWithin](#), [ST_Length2D](#), [ST_LineInterpolatePoint](#), [ST_LineSubstring](#)

7.19.5 ST_LineSubstring

`ST_LineSubstring` — 返回 n 个小数位置之 m 的直线部分。

Synopsis

```
geometry ST_LineSubstring(geometry a_linestring, float8 startfraction, float8 endfraction);
geography ST_LineSubstring(geography a_linestring, float8 startfraction, float8 endfraction);
```

描述

Computes the line which is the section of the input line starting and ending at the given fractional locations. The first argument must be a LINESTRING. The second and third arguments are values in the range [0, 1] representing the start and end locations. For geometry inputs, the fractions are measured in 2D line length. The Z and M values are interpolated for added endpoints if present.

如果 `startfraction` 和 `endfraction` 具有相同的 n ， n 相当于 [ST_LineInterpolatePoint](#)。



Note
 n 适用于 LINESTRING。要在 n 的 MULTILINESTRING 上使用，首先使用 [ST_LineMerge](#) 将它 n 接起来。

**Note**

从 1.1.1 版开始，此函数对 M 和 Z 进行插值。之前的版本将 Z 和 M 置为未指定的。

增加：3.4.0 - 引入了地理的支持。

更改：2.1.0。在 2.0.x 之前，它被称为 ST_Line_Substring。

可用性：1.1.0，1.1.1 中添加支持 Z 和 M



该函数支持 3d 并且不会丢失 z-index。

示例

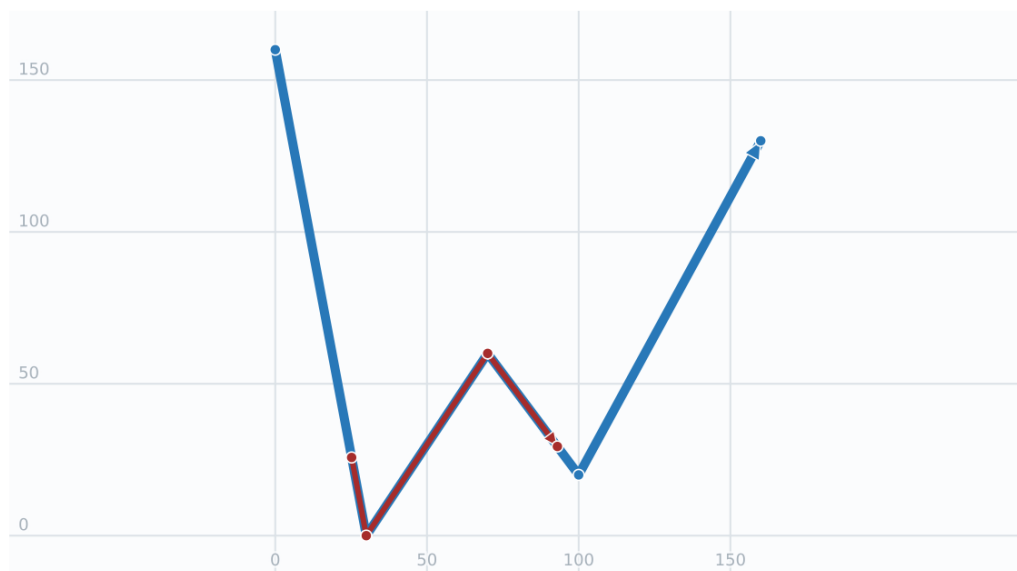
Extract the middle third of a LineString, from 0.333 to 0.666.

Code

```
SELECT ST_LineSubstring(  
'LINESTRING (20 180,50 20,90 80,120 40,180 150)',  
0.333,  
0.666);
```

Output

```
LINESTRING (45.17311810399485 45.74337011202746, 50 20, 90 80, 112.97593050157862 49.36542599789519)
```

Figure

■ Code ■ Output

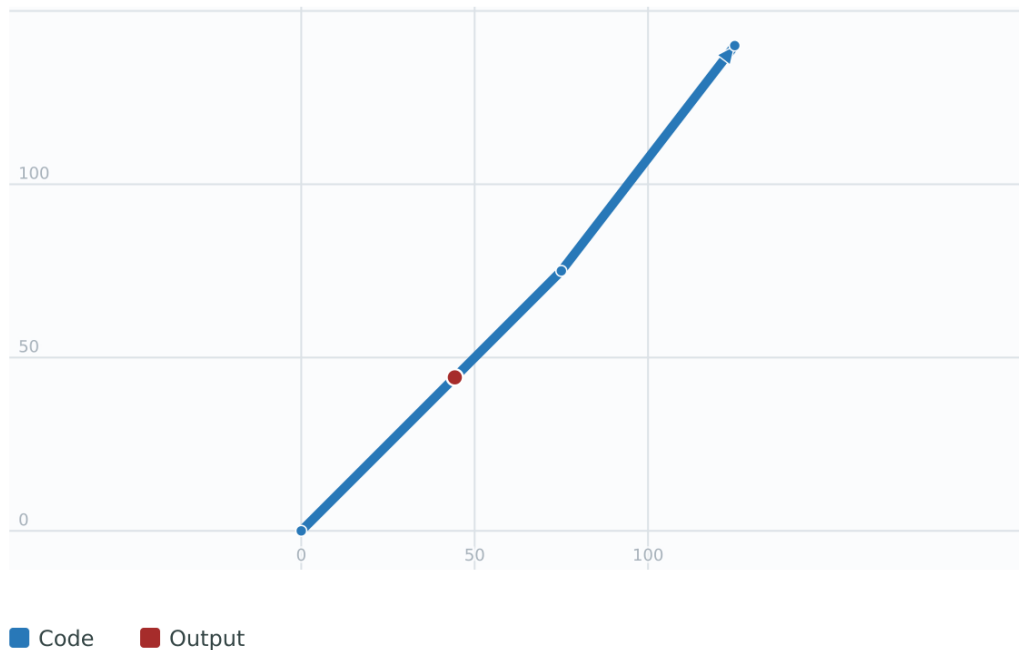
如果起始位置和结束位置相同，结果是一个 POINT。

Code

```
SELECT ST_LineSubstring(
  'LINESTRING(25 50,100 125,150 190)',
  0.333,
  0.333);
```

Output

```
POINT(69.28469348539744 94.28469348539744)
```

Figure

将 LineString 切成长度 100 或更短的部分的图。它使用 `generate_series()` 和 `CROSS JOIN LATERAL` 来生成 FOR 循环的等效图。

The final generated step is filtered out when the line length is an exact multiple of the requested section length.

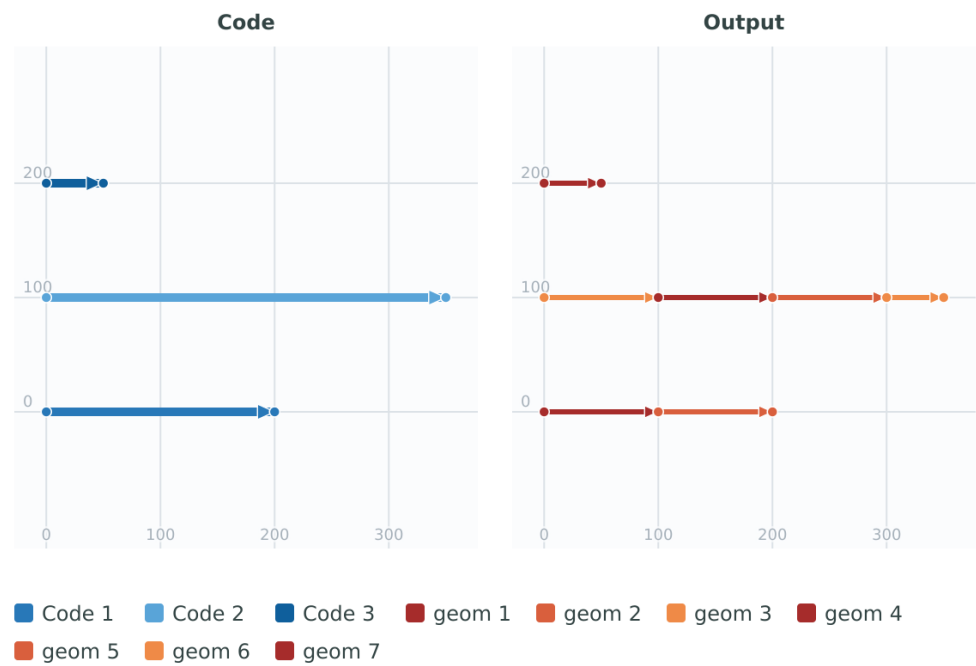
Code

```
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_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)
  WHERE (sublen * i) / len <
    > 1.0
) AS d2;
```

Output

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)

Figure



Split a LineString by points by locating the points along the line, adding the 0 and 1 endpoints, and extracting substrings between adjacent locations.

Code

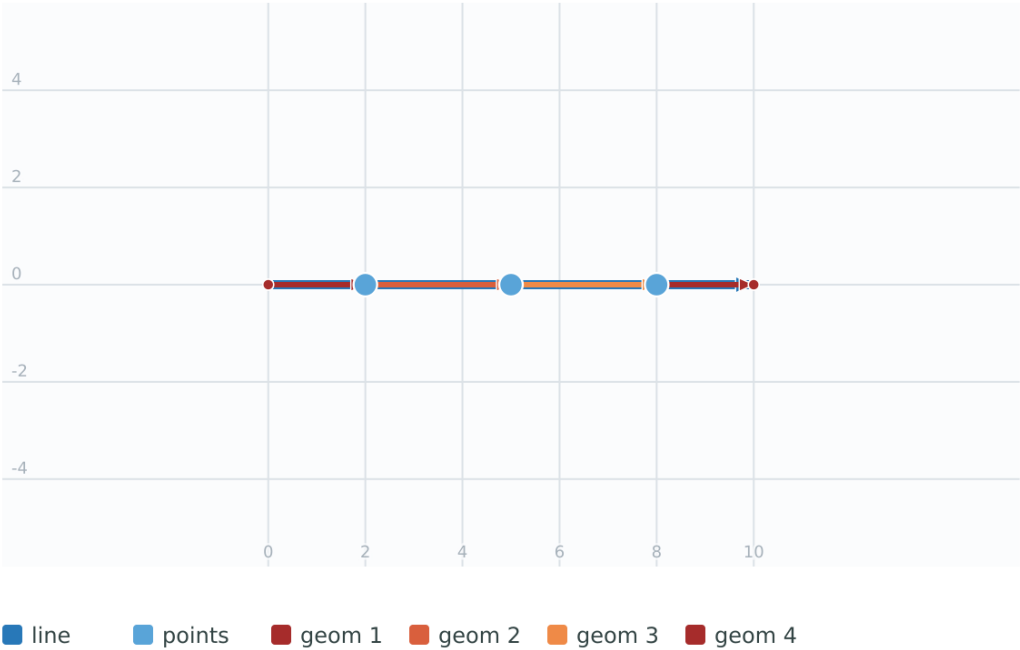
```
WITH data AS (  
  SELECT 'LINESTRING(0 0, 10 0)::geometry' AS line,  
         'MULTIPOINT((2 0),(5 0),(8 0))::geometry' AS points  
)  
fractions AS (  
  SELECT 0.0 AS fraction  
  UNION ALL  
  SELECT 1.0  
  UNION ALL  
  SELECT ST_LineLocatePoint(line, (dp).geom)  
  FROM data  
  CROSS JOIN LATERAL ST_Dump(points) AS dp  
)  
ordered AS (  
  SELECT DISTINCT fraction  
  FROM fractions  
)  
segments AS (  
  SELECT fraction AS start_fraction,  
         lead(fraction) OVER (ORDER BY fraction) AS end_fraction  
  FROM ordered  
)
```

```
SELECT row_number() OVER (ORDER BY start_fraction) AS segment_id,
       ST_LineSubstring(line, start_fraction, end_fraction) AS geom
FROM data
CROSS JOIN segments
WHERE end_fraction IS NOT NULL
      AND start_fraction <
      > end_fraction
ORDER BY start_fraction;
```

Output

segment_id	geom
1	LINESTRING(0 0,2 0)
2	LINESTRING(2 0,5 0)
3	LINESTRING(5 0,8 0)
4	LINESTRING(8 0,10 0)

Figure



地理对象沿球体表面度量，而几何对象沿直线度量

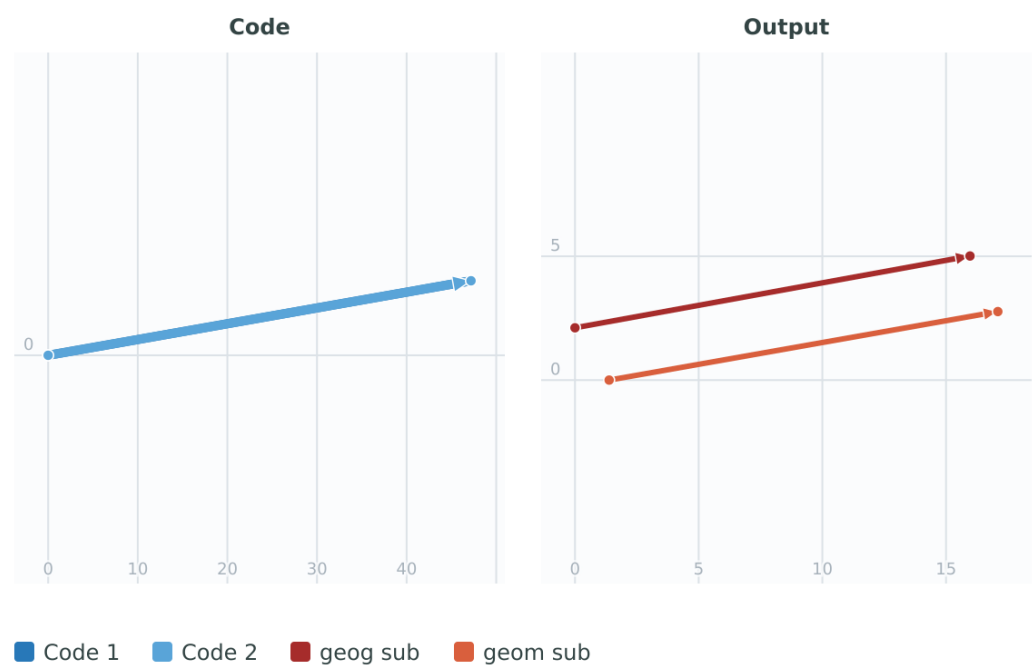
Code

```
SELECT ST_LineSubstring(
  'LINESTRING(-118.2436 34.0522,-71.0570 42.3611)::geography',
  0.333,
  0.666) AS geog_sub,
  ST_LineSubstring(
    'LINESTRING(-118.2436 34.0522,-71.0570 42.3611)::geometry',
    0.333,
    0.666) AS geom_sub;
```

Output

-[RECORD 1]-----	
geog_sub	LINESTRING(-103.911641 38.931128,-87.941787 41.831072)
geom_sub	LINESTRING(-102.530462 36.819064,-86.817324 39.585927)

Figure



For geometry inputs, the fractional locations are based on the 2D length of the line, even when the input has Z values. The resulting Z values are interpolated along the selected 2D location.

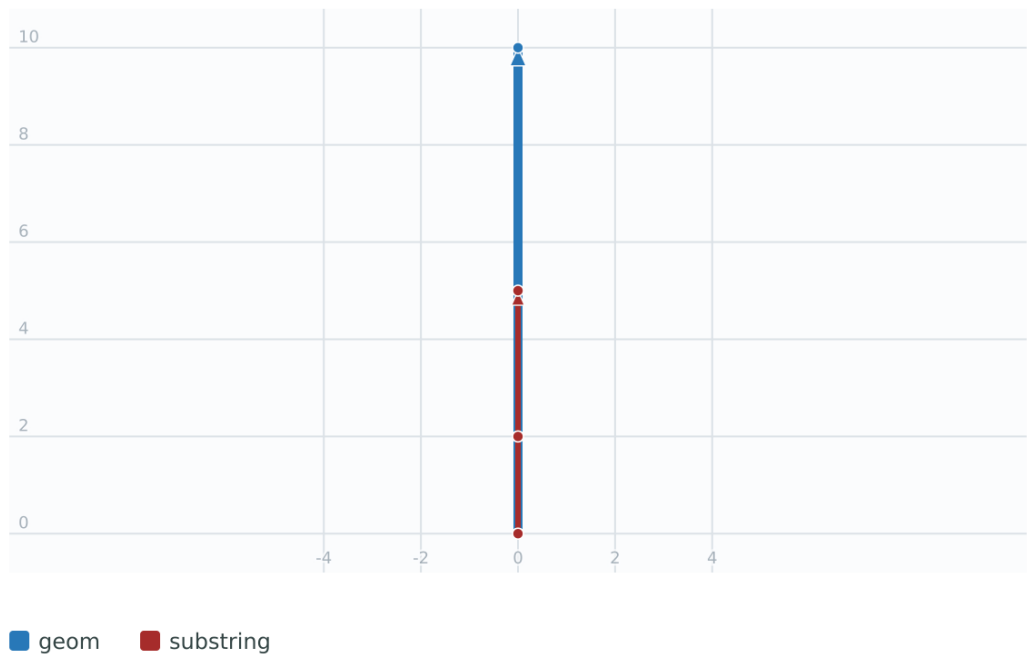
Code

```
WITH data AS (  
  SELECT 'LINESTRING Z (0 0 0,0 2 5,0 10 10)::geometry' AS geom  
)  
SELECT ST_Length(geom) AS length_2d,  
       ST_3DLength(geom) AS length_3d,  
       ST_LineSubstring(geom, 0, 0.5) AS substring  
FROM data;
```

Output

length_2d	length_3d	substring
10	14.819145939191106	LINESTRING Z (0 0 0,0 2 5,0 5 6.875)

Figure



相关信息

[ST_Length](#), [ST_LineExtend](#), [ST_LineInterpolatePoint](#), [ST_LineLocatePoint](#), [ST_LineMerge](#), [ST_Split](#)

7.19.6 ST_LocateAlong

`ST_LocateAlong` — 返回几何上与测量匹配的

Synopsis

geometry **ST_LocateAlong**(geometry geom_with_measure, float8 measure, float8 offset = 0);

描述

返回沿具有固定测量的测量几何形的位置。如果是点或多点。不支持多形输入。
如果提供了 `offset`，结果将向输入的左或右偏移指定的距离。正偏移将向左偏移，偏移将向右偏移。



Note
具有 M 分量的性几何使用此函数

由 *ISO/IEC 13249-3 SQL/MM* 空指定。
可用性：1.1.0（旧名称 `ST_Locate_Along_Measure`）。
更改：2.0.0 在之前的版本中，曾被称 `ST_Locate_Along_Measure`。

- ✓ 功能支持 M 坐标。
- ✓ 方法符合了 SQL/MM 规范。SQL-MM IEC 13249-3: 5.1.13

示例

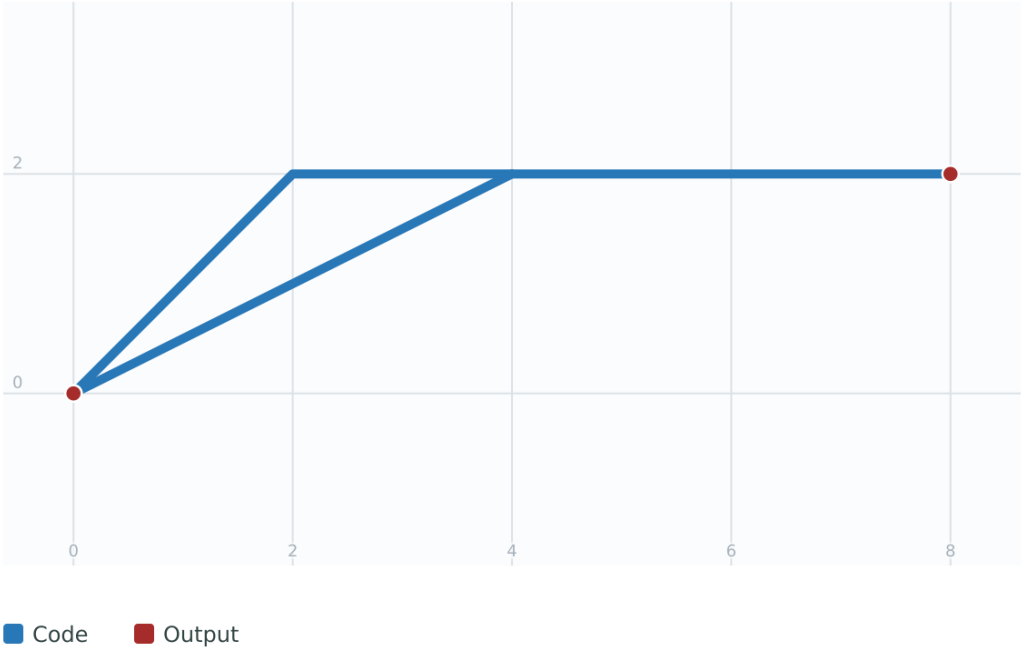
Code

```
SELECT ST_LocateAlong(  
'MULTILINESTRINGM((1 2 3, 3 4 2, 9 4 3),(1 2 3, 5 4 5))'::geometry,  
3);
```

Output

```
MULTIPOINT M ((1 2 3),(9 4 3),(1 2 3))
```

Figure



相关信息

[ST_LocateBetween](#), [ST_LocateBetweenElevations](#), [ST_InterpolatePoint](#)

7.19.7 ST_LocateBetween

ST_LocateBetween — 返回与量范围匹配的几何形部分。

Synopsis

geometry **ST_LocateBetween**(geometry geom, float8 measure_start, float8 measure_end, float8 offset = 0);

描述

返回一个几何图形（集合），其中包含与指定测量范围（包含）匹配的输入测量几何图形部分。

如果提供了偏移量，结果将向输入行的左或右偏移指定的距离。正偏移量将向左偏移，负偏移量将向右偏移。裁剪非凸多边形可能会生成无效的几何图形。

由 *ISO/IEC 13249-3 SQL/MM* 空值指定。

可用性：1.1.0（旧名称 `ST_Locate_Between_Measures`）。

更改：2.0.0 - 在之前的版本中，它曾被称 `ST_Locate_Between_Measures`。

增加：3.0.0 - 添加了多边形、TIN、三角形的支持。



功能支持 M 坐标。



方法符合了 SQL/MM 规范。SQL-MM IEC 13249-3: 5.1

示例

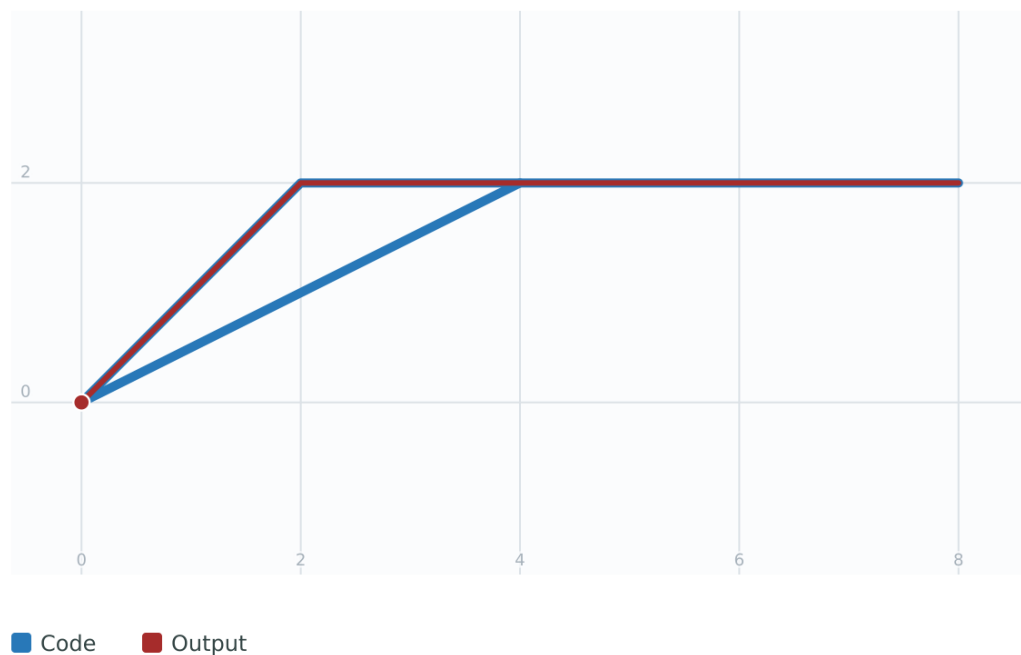
Code

```
SELECT ST_LocateBetween(  
'MULTILINESTRING M ((1 2 3,3 4 2,9 4 3),(1 2 3,5 4 5))'::geometry,  
1.5,  
3);
```

Output

```
GEOMETRYCOLLECTION M (LINESTRING M (1 2 3,3 4 2,9 4 3),POINT M (1 2 3))
```

Figure



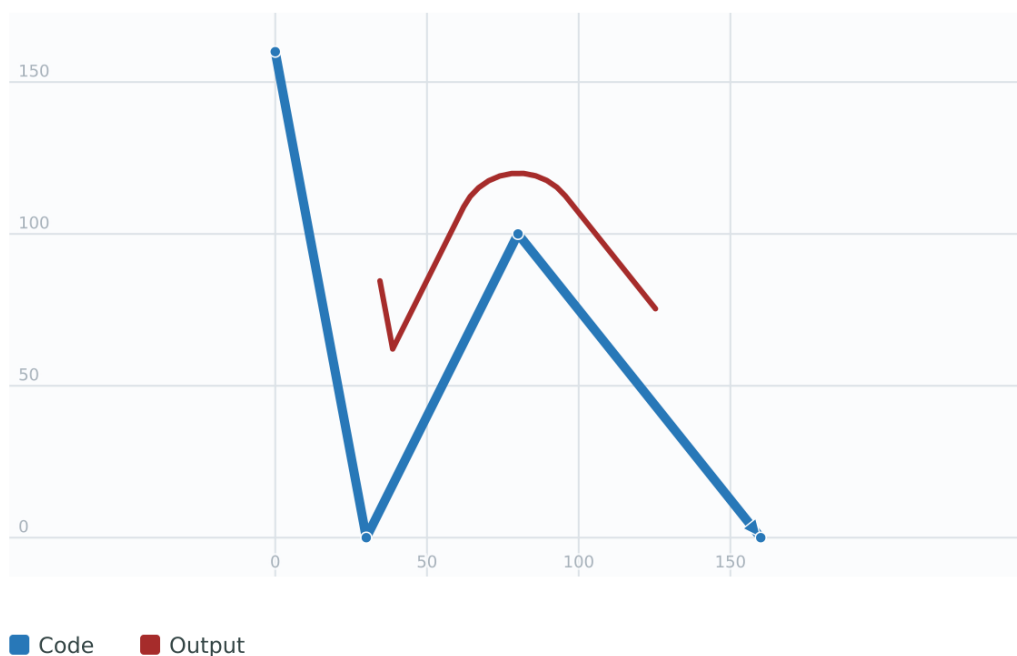
Extract the section between measures 2 and 8, offset to the left. Because the clipped line and the offset result share the same coordinate space, this example is intended for a single overlay panel rather than separate side-by-side figures.

Code

```
SELECT ST_LocateBetween(
ST_AddMeasure(
  'LINESTRING (20 180,50 20,100 120,180 20)',
    0,
    10),
  2,
  8,
  20);
```

Output

```
MULTILINESTRING((54.49835 104.53427,58.700561 82.122481,82.111456 128.944272,84.248937 132.324934,87.01637 ↵
135.21267,90.303073 137.491987,93.977598 139.071724,97.892984 139.888702,101.892639 139.910247,105.816599 ↵
139.135495,109.507928 137.595434,112.818995 135.351657,115.617376 132.493901,145.310173 95.377905))
```

Figure

相关信息

[ST_LocateAlong](#), [ST_LocateBetweenElevations](#)

7.19.8 ST_LocateBetweenElevations

`ST_LocateBetweenElevations` — 返回位于高程 (Z) 范围内的几何形状部分。

Synopsis

geometry **ST_LocateBetweenElevations**(geometry geom, float8 elevation_start, float8 elevation_end);

描述

返回一个几何图形（集合），其中包含位于高程 (Z) 范围内的几何图形部分。

裁剪非凸多边形可能会生成无效的几何图形。

可用性：1.4.0

增强：3.0.0 - 添加了多边形、TIN、三角形的支持。

 函数支持 3d 并且不会丢失 z-index。

示例

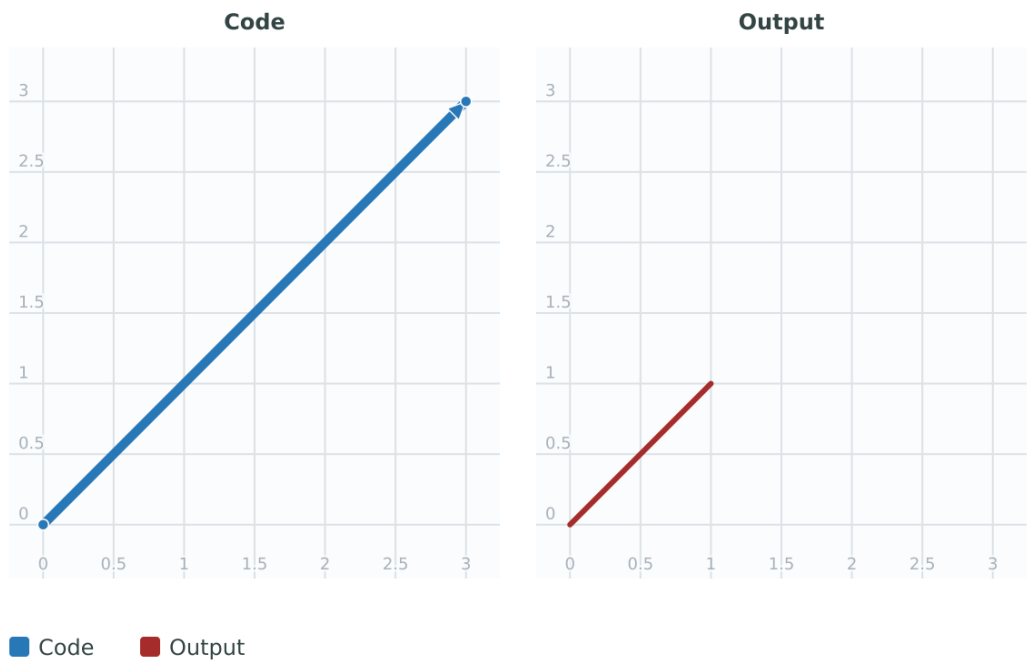
Code

```
SELECT ST_LocateBetweenElevations(  
'LINESTRING(1 2 3,4 5 6)::geometry',  
2,  
4);
```

Output

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

Figure



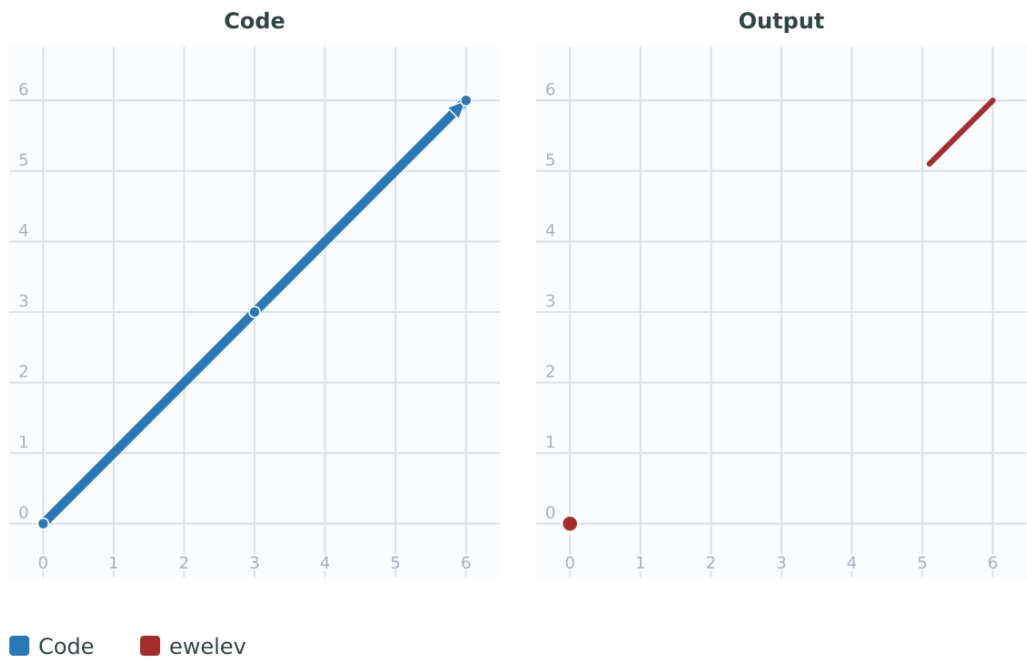
Code

```
SELECT ST_LocateBetweenElevations(  
'LINESTRING(1 2 6,4 5 -1,7 8 9)',  
6,  
9) As ewelev;
```

Output

GEOMETRYCOLLECTION Z (POINT Z (1 2 6),LINESTRING Z (6.1 7.1 6,7 8 9))

Figure



相关信息

ST_Dump, ST_LocateBetween

7.19.9 ST_InterpolatePoint

ST_InterpolatePoint — 返回最接近点的几何图形的插值量。

Synopsis

float8 ST_InterpolatePoint(geometry linear_geom_with_measure, geometry point);

描述

返回最接近指定点的位置的线性几何图形的插值量。



Note
具有 M 分量的线性几何使用此函数

可用性: 2.0.0

函数支持 3d 并且不会丢失 z-index。

示例

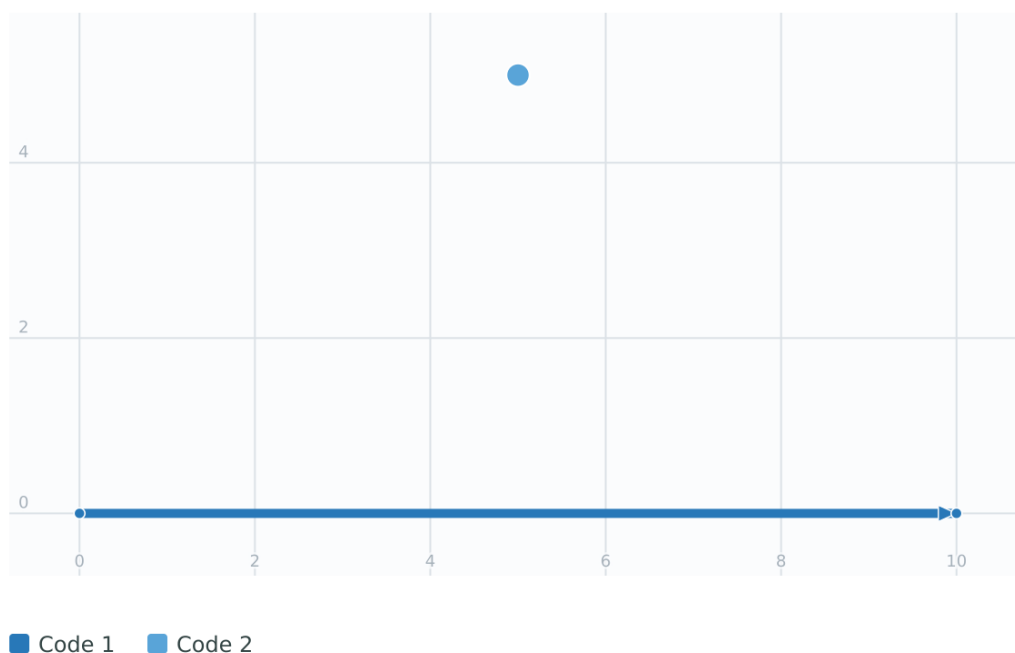
Code

```
SELECT ST_InterpolatePoint(  
'LINESTRING M (0 0 0,10 0 20)',  
'POINT(5 5)');
```

Output

10

Figure



相关信息

[ST_AddMeasure](#), [ST_LocateAlong](#), [ST_LocateBetween](#)

7.19.10 ST_3DInterpolatePoint

ST_3DInterpolatePoint — Returns the interpolated measure of a geometry closest to a point in 3D.

Synopsis

float8 **ST_3DInterpolatePoint**(geometry linear_geom_with_measure, geometry point);

描述

Returns the interpolated measure value of a linear ZM geometry at the location closest to the given point, using 3D (XYZ) distance for the projection. Use this function when the geometry has significant Z variation, such as flight trajectories, where [ST_InterpolatePoint](#) would give incorrect results by ignoring the Z dimension.

**Note**

The line must have both Z and M dimensions. The point should have a Z dimension.

Availability: 3.7.0



☑函数支持 3d 并且不会☒失 z-index。

示例

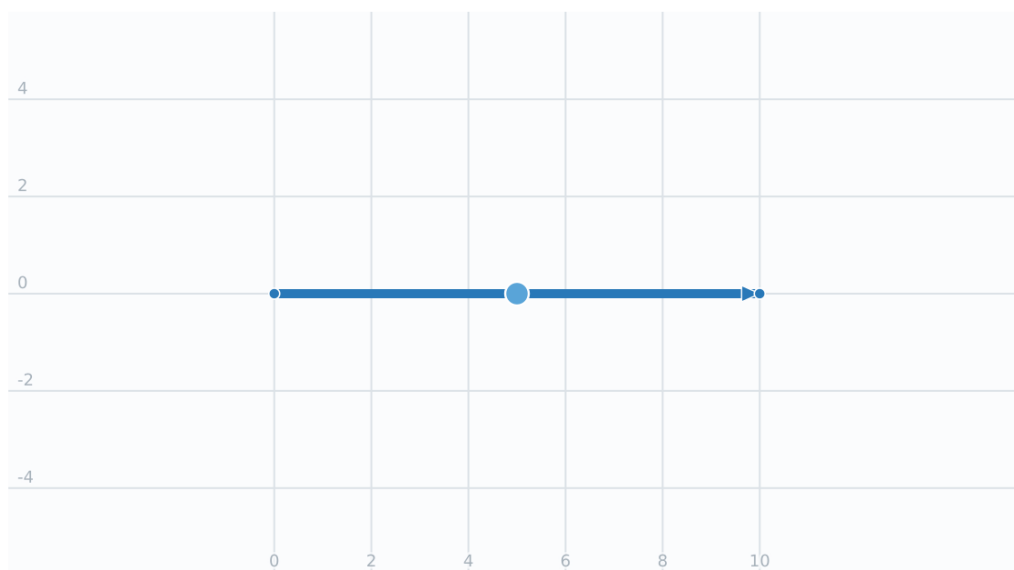
This example interpolates on a line rising diagonally in Z, using a point at the 3D midpoint.

Code

```
SELECT ST_3DInterpolatePoint(  
'LINESTRING ZM (0 0 0 0,10 0 10 20)',  
'POINT Z (5 0 5)');
```

Output

10

Figure

■ Code 1 ■ Code 2

This example uses a vertical line with zero XY extent, where 3D projection works but 2D projection does not.

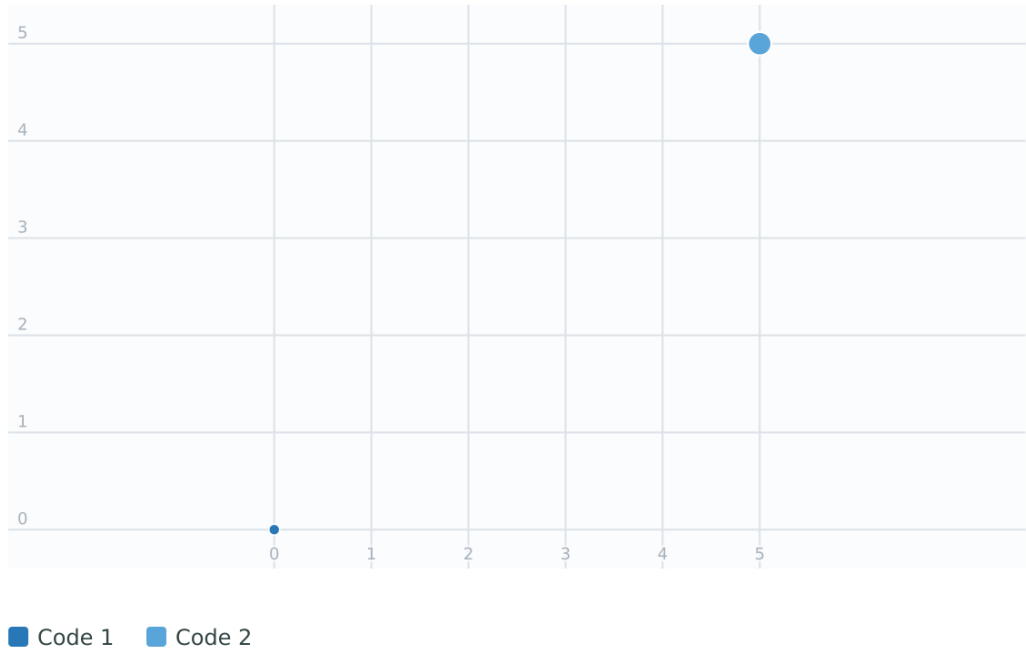
Code

```
SELECT ST_3DInterpolatePoint(  
'LINESTRING ZM (0 0 0 0,0 0 10 100)',  
'POINT Z (5 5 5)');
```

Output

50

Figure



相关信息

[ST_InterpolatePoint](#), [ST_AddMeasure](#), [ST_LocateAlong](#), [ST_LocateBetween](#)

7.19.11 ST_AddMeasure

ST_AddMeasure — 沿线性几何形状插测量。

Synopsis

geometry **ST_AddMeasure**(geometry geom_mline, float8 measure_start, float8 measure_end);

描述

返回派生几何形，其测量在起点和点之间线性插。如果几何形没有测量尺寸，添加一个。如果几何形具有测量尺寸，会用新覆盖它。支持 LINESTRINGS 和 MULTILINESTRINGS。

可用性：1.1.0

 函数支持 3d 并且不会失 z-index。

示例

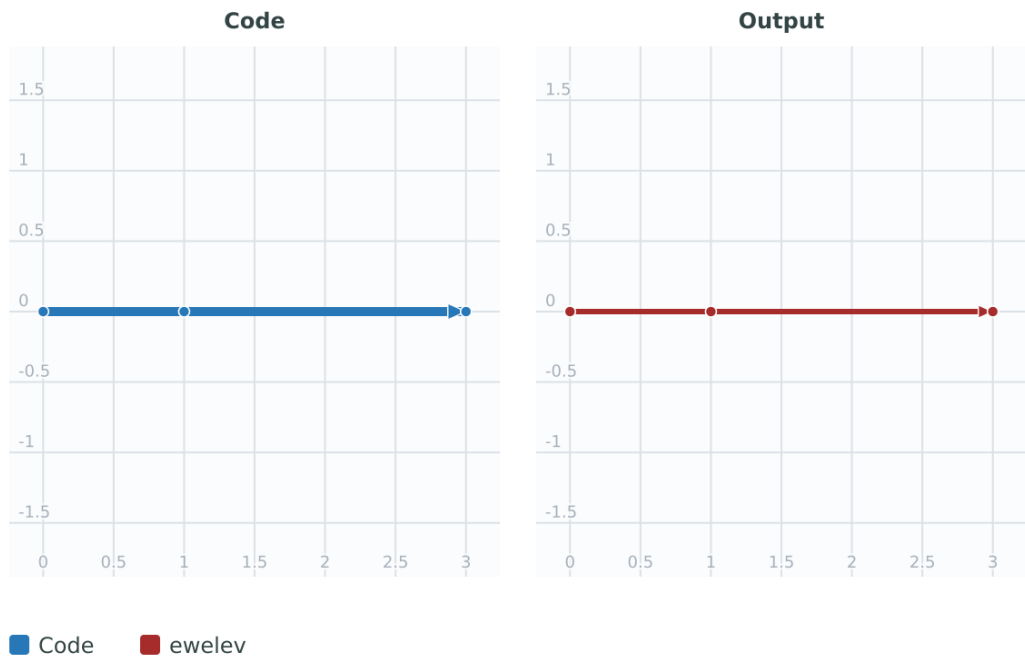
Code

```
SELECT ST_AddMeasure(  
'LINESTRING(1 0,2 0,4 0)',  
1,  
4) As ewelev;
```

Output

```
LINESTRING M (1 0 1,2 0 2,4 0 4)
```

Figure



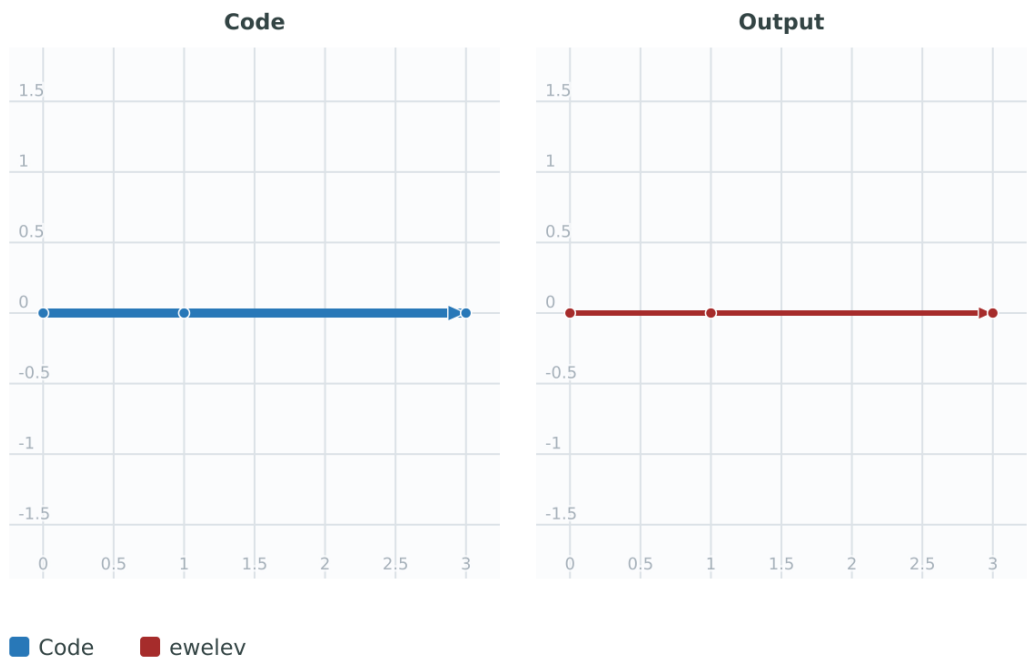
Code

```
SELECT ST_AddMeasure(  
'LINESTRING(1 0 4,2 0 4,4 0 4)',  
10,  
40) As ewelev;
```

Output

```
LINESTRING ZM (1 0 4 10,2 0 4 20,4 0 4 40)
```

Figure



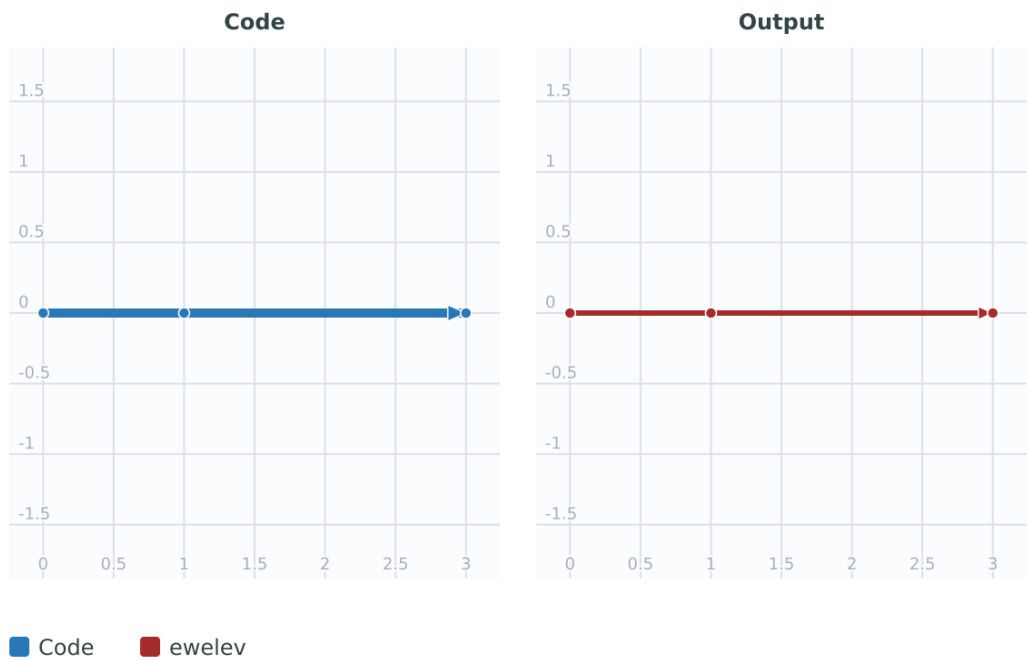
Code

```
SELECT ST_AddMeasure(  
'LINESTRING(1 0 4, 2 0 4, 4 0 4)',  
10,  
40) As ewelev;
```

Output

```
LINESTRING M (1 0 10,2 0 20,4 0 40)
```

Figure



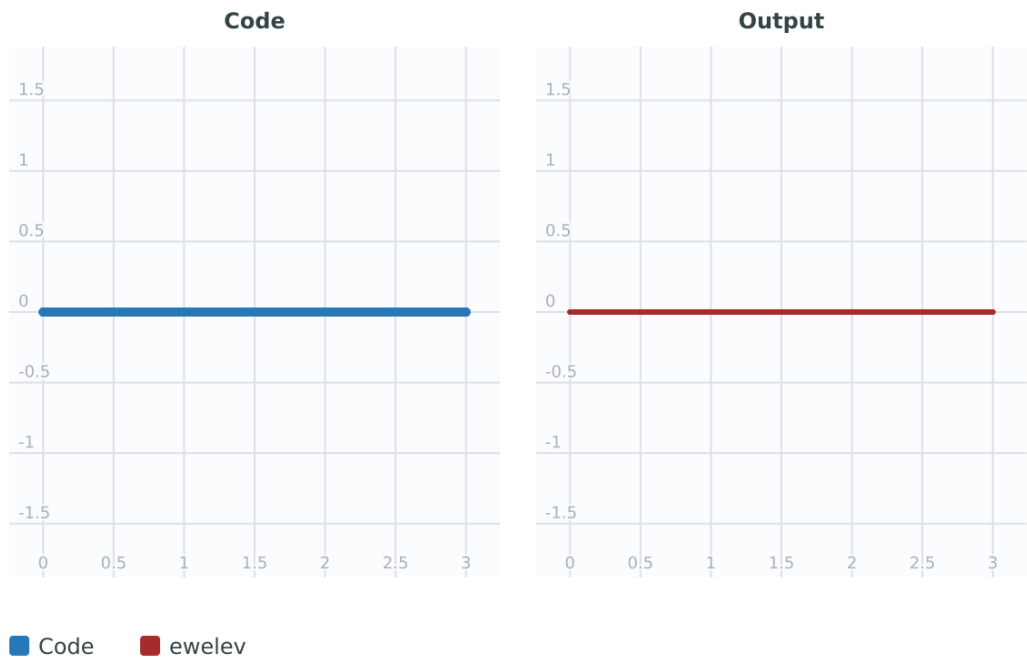
Code

```
SELECT ST_AddMeasure(  
'MULTILINESTRINGM((1 0 4, 2 0 4, 4 0 4),(1 0 4, 2 0 4, 4 0 4))',  
10,  
70) As ewelev;
```

Output

```
MULTILINESTRING M ((1 0 10,2 0 20,4 0 40),(1 0 40,2 0 50,4 0 70))
```

Figure



7.20 轨迹函数

7.20.1 ST_IsValidTrajectory

ST_IsValidTrajectory — 检查几何图形是否有效轨迹。

Synopsis

boolean **ST_IsValidTrajectory**(geometry line);

描述

几何体是否有效的轨迹。有效轨迹表示有度量（M 的）LINESTRING。量必须从每个点到下一个点增加。

有效轨迹作空函数（如 [ST_ClosestPointOfApproach](#)）的输入

可用性：2.2.0

函数支持 3d 并且不会失 z-index。

示例

This example constructs a valid trajectory.

| Code

```
SELECT ST_IsValidTrajectory(ST_MakeLine(ST_MakePointM(0, 0, 1),
ST_MakePointM(0, 1, 2)));
```

| Output

```
t
```

This example constructs an invalid trajectory.

| Code

```
SELECT ST_IsValidTrajectory(ST_MakeLine(ST_MakePointM(0, 0, 1), ST_MakePointM(0, 1, 0)));
```

| Output

```
NOTICE: Measure of vertex 1 (0) not bigger than measure of vertex 0 (1)
f
```

相关信息

[ST_ClosestPointOfApproach](#)

7.20.2 ST_ClosestPointOfApproach

`ST_ClosestPointOfApproach` — 返回两条迹最接近点的度量。

Synopsis

`float8 ST_ClosestPointOfApproach(geometry track1, geometry track2);`

描述

返回沿着给定迹插的点之间距离最短的最小度量。

入参必须是 `ST_IsValidTrajectory` 的有效迹。如果迹在其 M 范围内不重叠，返回 `Null`。

要取算得到的度量上的点，使用 `ST_LocateAlong`。

可用性：2.2.0



函数支持 3d 并且不会失 z-index。

示例

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.

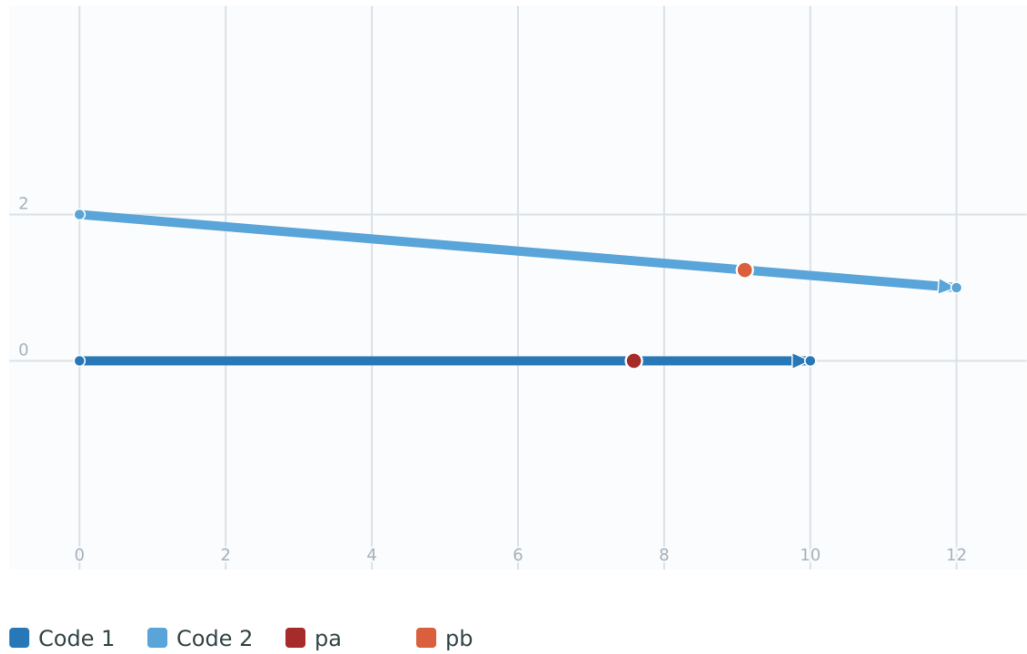
Code

```
WITH inp AS ( SELECT
  ST_AddMeasure('LINESTRING Z (0 0 0,10 0 5)::geometry,
    extract(epoch from timestamp '2015-05-26 10:00' AT TIME ZONE 'UTC'),
    extract(epoch from timestamp '2015-05-26 11:00' AT TIME ZONE 'UTC')
  ) a,
  ST_AddMeasure('LINESTRING Z (0 2 10,12 1 2)::geometry,
    extract(epoch from timestamp '2015-05-26 10:00' AT TIME ZONE 'UTC'),
    extract(epoch from timestamp '2015-05-26 11:00' AT TIME ZONE 'UTC')
  ) b
), cpa AS (
  SELECT ST_ClosestPointOfApproach(a, b) m FROM inp
), points AS (
  SELECT ST_GeometryN(ST_LocateAlong(a, m), 1) pa,
         ST_GeometryN(ST_LocateAlong(b, m), 1) pb
  FROM inp, cpa
)
SELECT to_timestamp(m) AT TIME ZONE 'UTC' t,
       ST_3DDistance(pa, pb) distance,
       pa AS pa, pb AS pb
FROM points, cpa;
```

Output

-[RECORD 1]-----	
t	2015-05-26 10:45:31.034483
distance	1.9652147377620688
pa	POINT ZM (7.59 0 3.79 1432637131.03)
pb	POINT ZM (9.1 1.24 3.93 1432637131.03)

Figure



相关信息

[ST_IsValidTrajectory](#), [ST_DistanceCPA](#), [ST_LocateAlong](#), [ST_AddMeasure](#)

7.20.3 ST_DistanceCPA

ST_DistanceCPA — 返回两条轨迹的最近接近点之间的距离。

Synopsis

```
float8 ST_DistanceCPA(geometry track1, geometry track2);
```

描述

返回两条轨迹在它最接近的相交点之间的距离（在二维空中）。

输入必须是 [ST_IsValidTrajectory](#) 检查的有效轨迹。如果轨迹在其 M 范围内不重叠，则返回 Null。

可用性：2.2.0



函数支持 3d 并且不会丢失 z-index。

示例

Return the minimum distance of two objects moving between 10:00 and 11:00.

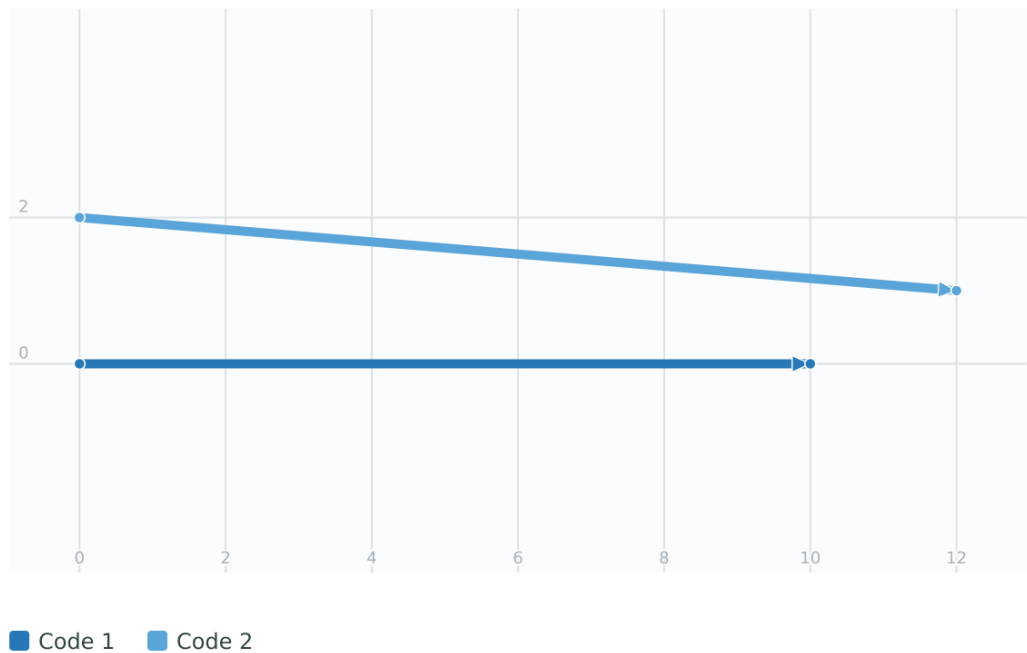
Code

```
WITH inp AS ( SELECT
  ST_AddMeasure('LINESTRING Z (0 0 0,10 0 5)::geometry,
    extract(epoch from timestamp '2015-05-26 10:00' AT TIME ZONE 'UTC'),
    extract(epoch from timestamp '2015-05-26 11:00' AT TIME ZONE 'UTC')
  ) a,
  ST_AddMeasure('LINESTRING Z (0 2 10,12 1 2)::geometry,
    extract(epoch from timestamp '2015-05-26 10:00' AT TIME ZONE 'UTC'),
    extract(epoch from timestamp '2015-05-26 11:00' AT TIME ZONE 'UTC')
  ) b
)
SELECT ST_DistanceCPA(a, b) distance FROM inp;
```

Output

```
1.9652147377620688
```

Figure



相关信息

[ST_IsValidTrajectory](#), [ST_ClosestPointOfApproach](#), [ST_AddMeasure](#), [|](#)

7.20.4 ST_CPAWithin

ST_CPAWithin — 检查两条轨迹的最近接近点是否在指定距离内。

Synopsis

boolean **ST_CPAWithin**(geometry track1, geometry track2, float8 dist);

描述

检查两个移动物体是否曾比指定距离更近。
输入必须是 [ST_IsValidTrajectory](#) 检查的有效轨迹。如果轨迹在其 M 范围内不重叠，则返回 False。
可用性：2.2.0

 该函数支持 3d 并且不会丢失 z-index。

示例

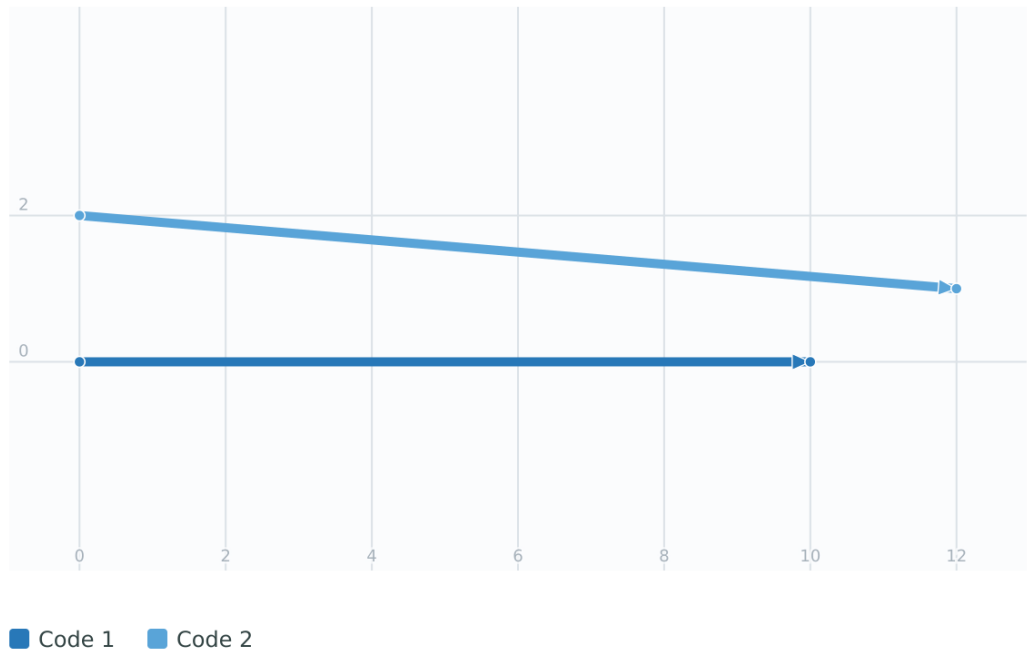
Code

```
WITH inp AS ( SELECT
  ST_AddMeasure('LINESTRING Z (0 0 0,10 0 5)::geometry,
    extract(epoch from timestamp '2015-05-26 10:00' AT TIME ZONE 'UTC'),
    extract(epoch from timestamp '2015-05-26 11:00' AT TIME ZONE 'UTC')
  ) a,
  ST_AddMeasure('LINESTRING Z (0 2 10,12 1 2)::geometry,
    extract(epoch from timestamp '2015-05-26 10:00' AT TIME ZONE 'UTC'),
    extract(epoch from timestamp '2015-05-26 11:00' AT TIME ZONE 'UTC')
  ) b
)
SELECT ST_CPAWithin(a, b, 2), ST_DistanceCPA(a, b) distance FROM inp;
```

Output

st_cpawithin	distance
t	1.9652147377620688

Figure



相关信息

[ST_IsValidTrajectory](#), [ST_ClosestPointOfApproach](#), [ST_DistanceCPA](#), [|](#)

7.21 版本函数

7.21.1 PostGIS_Extensions_Upgrade

PostGIS_Extensions_Upgrade — 将 PostGIS 扩展 (例如 `postgis_raster`, `postgis_topology`, `postgis_sfcgal`) 打包并升级到特定版本或最新版本。

Synopsis

text **PostGIS_Extensions_Upgrade**(text target_version=null);

描述

Packages and upgrades PostGIS extensions to given or latest version. Only extensions you have installed in the database will be packaged and upgraded if needed. Reports full PostGIS version and build configuration infos after. This is short-hand for doing multiple CREATE EXTENSION .. FROM unpackaged and ALTER EXTENSION .. UPDATE for each PostGIS extension. Currently only tries to upgrade extensions postgis, postgis_raster, postgis_sfcgal, and postgis_topology.

可用性：2.5.0

更改：3.4.0 添加 target_version 参数。

更改：3.3.0 支持从任何 PostGIS 版本升级。不适用于所有系统。

更改：3.0.0 重新打包松散扩展并支持 postgis_raster。

示例

Code

```
SELECT PostGIS_Extensions_Upgrade();
```

Output

```
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
```

相关信息

Section 3.4, [PostGIS_GEOS_Version](#), [PostGIS_Lib_Version](#), [PostGIS_LibXML_Version](#), [PostGIS_PROJ_Version](#), [PostGIS_Version](#)

7.21.2 PostGIS_Full_Version

PostGIS_Full_Version — 报告完整的 PostGIS 版本和构建配置信息。

Synopsis

text **PostGIS_Full_Version**();

描述

报告完整的 PostGIS 版本和构建配置信息。通知和脚本之间的同步，构建根据需要执行升级。

新增功能：3.4.0 在包括外的 PROJ 配置 NETWORK_ENABLED、URL_ENDPOINT 和 proj.db 位置的 DATABASE_PATH

示例

Code

```
SELECT PostGIS_Full_Version();
```

Output

```
POSTGIS="3.7.0dev 3.7.0alpha1-477-ga9004e778" [EXTENSION] PGSQL="180" GEOS="3.15.0beta1" SFCGAL="2.3.0" PROJ="9.8.1 ↵  
NETWORK_ENABLED=OFF URL_ENDPOINT=https://cdn.proj.org DATABASE_PATH=/usr/share/proj/proj.db" GDAL="GDAL 3.13.1, ↵  
released 2026/06/01" LIBXML="2.15.3"
```

相关信息

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` — 返回 GEOS 的版本号。

Synopsis

text `PostGIS_GEOS_Version()`;

描述

返回 GEOS 的版本号，如果未用 GEOS 支持，返回 `NULL`。

示例

Code

```
SELECT PostGIS_GEOS_Version();
```

Output

```
3.12.0dev-CAPI-1.18.0
```

相关信息

[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` — 返回构建 PostGIS 所依据的 GEOS 的版本号。

Synopsis

text `PostGIS_GEOS_Compiled_Version()`;

描述

返回 GEOS 的版本号，或构建 PostGIS 的版本号。

可用性：3.4.0

示例

Code

```
SELECT PostGIS_GEOS_Compiled_Version();
```

Output

```
3.12.0
```

相关信息

[PostGIS_GEOS_Version](#), [PostGIS_Full_Version](#)

7.21.5 PostGIS_Liblwgeom_Version

`PostGIS_Liblwgeom_Version` — 返回 liblwgeom 的版本号。与 PostGIS 的版本匹配。

Synopsis

text **PostGIS_Liblwgeom_Version()**;

描述

返回 liblwgeom 的版本号/

示例

Code

```
SELECT PostGIS_Liblwgeom_Version();
```

Output

```
3.4.0dev 3.3.0rc2-993-g61bdf43a7
```

相关信息

[PostGIS_Full_Version](#), [PostGIS_Lib_Version](#), [PostGIS_LibXML_Version](#), [PostGIS_PROJ_Version](#), [PostGIS_Version](#)

7.21.6 PostGIS_LibXML_Version

PostGIS_LibXML_Version — 返回 libxml2 的版本号。

Synopsis

text **PostGIS_LibXML_Version()**;

描述

返回 libxml2 的版本号。

可用性：1.5

示例

| Code

```
SELECT PostGIS_LibXML_Version();
```

| Output

```
2.9.10
```

相关信息

[PostGIS_Full_Version](#), [PostGIS_Lib_Version](#), [PostGIS_PROJ_Version](#), [PostGIS_GEOS_Version](#), [PostGIS_Version](#)

7.21.7 PostGIS_LibJSON_Version

PostGIS_LibJSON_Version — 返回 libjson-c 的版本号。

Synopsis

text **PostGIS_LibJSON_Version()**;

描述

返回 LibJSON-C 的版本号。

示例

| Code

```
SELECT PostGIS_LibJSON_Version();
```

| **Output**

0.17

相关信息

[PostGIS_Full_Version](#), [PostGIS_Lib_Version](#), [PostGIS_PROJ_Version](#), [PostGIS_GEOS_Version](#), [PostGIS_Versio](#)

7.21.8 PostGIS_Lib_Build_Date

`PostGIS_Lib_Build_Date` — 返回 PostGIS 的构建日期。

Synopsis

text **PostGIS_Lib_Build_Date**();

描述

返回 PostGIS 的构建日期。

示例

| **Code**

```
SELECT PostGIS_Lib_Build_Date();
```

| **Output**

2023-06-22 03:56:11

7.21.9 PostGIS_Lib_Version

`PostGIS_Lib_Version` — 返回 PostGIS 的版本号。

Synopsis

text **PostGIS_Lib_Version**();

描述

返回 PostGIS 的版本号。

示例

Code

```
SELECT PostGIS_Lib_Version();
```

Output

```
3.4.0dev
```

相关信息

[PostGIS_Full_Version](#), [PostGIS_GEOS_Version](#), [PostGIS_LibXML_Version](#), [PostGIS_PROJ_Version](#), [PostGIS_Version](#)

7.21.10 PostGIS_PROJ_Version

PostGIS_PROJ_Version — 返回 PROJ4 的版本号。

Synopsis

text **PostGIS_PROJ_Version()**;

描述

返回 PROJ 的版本号和 proj 的一些配置。

增强的功能：3.4.0 在包括 proj.db 位置的 NETWORK_ENABLED、URL_ENDPOINT 和 DATABASE_PATH

示例

Code

```
SELECT PostGIS_PROJ_Version();
```

Output

```
7.2.1 NETWORK_ENABLED=OFF URL_ENDPOINT=https://cdn.proj.org USER_WRITABLE_DIRECTORY=/tmp/ ↵  
proj DATABASE_PATH=/usr/share/proj/proj.db
```

相关信息

[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 — 返回用于构建 PostGIS 的 PROJ 的版本号。

Synopsis

text **PostGIS_PROJ_Compiled_Version()**;

描述

返回的是用于构建 PostGIS 的 PROJ 的版本号。

可用性：3.5.0

示例

| Code

```
SELECT PostGIS_PROJ_Compiled_Version();
```

| Output

```
9.1.1
```

相关信息

[PostGIS_PROJ_Version](#), [PostGIS_Full_Version](#)

7.21.12 PostGIS_Wagyu_Version

PostGIS_Wagyu_Version — 返回内部 Wagyu 的版本号。

Synopsis

text **PostGIS_Wagyu_Version()**;

描述

返回内部 Wagyu 的版本号，如果未用 Wagyu 支持，返回 NULL。

示例

| Code

```
SELECT PostGIS_Wagyu_Version();
```

| Output

```
0.5.0 (Internal)
```

相关信息

[PostGIS_Full_Version](#), [PostGIS_GEOS_Version](#), [PostGIS_PROJ_Version](#), [PostGIS_Lib_Version](#), [PostGIS_LibXML_Version](#), [PostGIS_Version](#)

7.21.13 PostGIS_Scripts_Build_Date

`PostGIS_Scripts_Build_Date` — 返回 PostGIS 脚本的构建日期。

Synopsis

text **PostGIS_Scripts_Build_Date()**;

描述

返回 PostGIS 脚本的构建日期。

可用性 : 1.0.0RC1

示例

Code

```
SELECT PostGIS_Scripts_Build_Date();
```

Output

```
2023-06-22 03:56:11
```

相关信息

[PostGIS_Full_Version](#), [PostGIS_GEOS_Version](#), [PostGIS_Lib_Version](#), [PostGIS_LibXML_Version](#), [PostGIS_Version](#)

7.21.14 PostGIS_Scripts_Installed

`PostGIS_Scripts_Installed` — 返回此数据库中安装的 PostGIS 脚本的版本。

Synopsis

text **PostGIS_Scripts_Installed()**;

描述

返回此数据库中安装的 PostGIS 脚本的版本。



Note

如果此函数的输出与 [PostGIS_Scripts_Released](#) 的输出不匹配，您可能错过了正确升级有数据的机会。有关更多信息，请参阅[升级](#)部分。

可用性 : 0.9.0

示例

| Code

```
SELECT PostGIS_Scripts_Installed();
```

| Output

```
3.4.0dev 3.3.0rc2-993-g61bdf43a7
```

相关信息

[PostGIS_Full_Version](#), [PostGIS_Scripts_Released](#), [PostGIS_Version](#)

7.21.15 PostGIS_Scripts_Released

PostGIS_Scripts_Released — 返回随安装的 PostGIS 一起发布的 postgis.sql 脚本的版本号。

Synopsis

text **PostGIS_Scripts_Released**();

描述

返回随安装的 PostGIS 一起发布的 postgis.sql 脚本的版本号。



Note

从版本 1.1.0 开始，此函数返回与 [PostGIS_Lib_Version](#) 相同的值。保留向后兼容性。

可用性 : 0.9.0

示例

| Code

```
SELECT PostGIS_Scripts_Released();
```

| Output

```
3.4.0dev 3.3.0rc2-993-g61bdf43a7
```

相关信息

[PostGIS_Full_Version](#), [PostGIS_Scripts_Installed](#), [PostGIS_Lib_Version](#)

7.21.16 PostGIS_Version

PostGIS_Version — 返回 PostGIS 版本号和 `USE_STATS`。

Synopsis

text **PostGIS_Version()**;

描述

返回 PostGIS 版本号和 `USE_STATS`。

示例

Code

```
SELECT PostGIS_Version();
```

Output

```
3.4 USE_GEOS=1 USE_PROJ=1 USE_STATS=1
```

相关信息

[PostGIS_Full_Version](#), [PostGIS_GEOS_Version](#), [PostGIS_Lib_Version](#), [PostGIS_LibXML_Version](#), [PostGIS_PROJ_Version](#)

7.22 大一统自定义量 (GUCs)

7.22.1 postgis.gdal_datapath

`postgis.gdal_datapath` — 用于分配 GDAL 的 `GDAL_DATA` 目录的配置项。如果未设置，则使用环境变量 `GDAL_DATA` 的值。

描述

一个 PostgreSQL GUC 变量，用于设置 GDAL 的 `GDAL_DATA` 目录。 `postgis.gdal_datapath` 是 GDAL 数据文件的完整物理路径。

此配置项最适用于 GDAL 数据文件路径未硬编码的 Windows 平台。当 GDAL 的数据文件不在 GDAL 的预期路径中时，也建议设置此配置项。



Note

`GDAL_DATA` 可以在 PostgreSQL 的配置文件 `postgresql.conf` 中设置。也可以通过环境变量来设置。

可用性：2.2.0

**Note**

有关 GDAL_DATA 的其他信息可在 GDAL 的[配置文档](#)中找到。

示例

设置和重置 `postgis.gdal_datapath`

Code

```
SET postgis.gdal_datapath TO '/usr/local/share/gdal.hidden';
SET postgis.gdal_datapath TO default;
```

在 Windows 上设置特定数据路径

Code

```
ALTER DATABASE gisdb
SET postgis.gdal_datapath = 'C:/Program Files/PostgreSQL/9.3/gdal-data';
```

相关信息

[PostGIS_GDAL_Version](#), [ST_Transform](#)

7.22.2 postgis.gdal_enabled_drivers

`postgis.gdal_enabled_drivers` — 用于设置 PostGIS 环境中使用的 GDAL 程序的配置项。影响 GDAL 配置项 `GDAL_SKIP`。

描述

用于设置 PostGIS 环境中使用的 GDAL 程序的配置项。影响 GDAL 配置项 `GDAL_SKIP`。可以在 PostgreSQL 的配置文件：`postgresql.conf` 中设置。也可以通过连接或事件来设置。

`postgis.gdal_enabled_drivers` 的初始值也可以通过将环境变量 `POSTGIS_GDAL_ENABLED_DRIVERS` 和已使用的程序列表一起传递给 PostgreSQL 的程序来设置。

用 GDAL 指定的程序可以通过程序的短名称或代号来指定。程序短名称或代号可以在 [GDAL 格式](#) 中找到。可以通过在每个程序之间添加空格来指定多个程序。

Note

`postgis.gdal_enabled_drivers` 有三个可用的特殊代号。代号区分大小写。

- `DISABLE_ALL` 禁用所有 GDAL 程序。如果存在，`DISABLE_ALL` 会覆盖 `postgis.gdal_enabled_drivers` 中的所有其他值。
- `ENABLE_ALL` 启用所有 GDAL 程序。
- `VSI_CURL` 用 GDAL 的 `/vsicurl/` 虚拟文件系统。

当 `postgis.gdal_enabled_drivers` 设置为 `DISABLE_ALL` 时，使用 `out-db` 格式、`ST_FromGDALRaster()`、`ST_AsGDALRaster()`、`ST_AsTIFF()`、`ST_AsJPEG()` 和 `ST_AsPNG()` 将导致消息。



**Note**

在默认 PostGIS 安装中, `postgis.gdal_enabled_drivers` 设置为 `DISABLE_ALL`。

**Note**

有关 `GDAL_SKIP` 的其他信息可在 GDAL 的[配置文档](#)中找到。

可用性 : 2.2.0

示例

Set and reset `postgis.gdal_enabled_drivers` for the current session.

Code

```
SET postgis.gdal_enabled_drivers = 'ENABLE_ALL';  
SET postgis.gdal_enabled_drivers = default;
```

Set specific drivers for all new connections to a database.

Code

```
ALTER DATABASE mygisdb SET postgis.gdal_enabled_drivers TO 'GTiff PNG JPEG';
```

Set the default for the whole database cluster to enable all drivers. This requires superuser access. Also note that database, session, and user settings override this.

This example writes the setting to `postgres.auto.conf`.

Code

```
ALTER SYSTEM SET postgis.gdal_enabled_drivers TO 'ENABLE_ALL';
```

Reload the PostgreSQL configuration so the changed setting is applied.

Code

```
SELECT pg_reload_conf();
```

相关信息

[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` — 一个布尔配置项，用于启用/禁用数据外部格式波段的存储。

描述

一个布尔配置项，用于启用数据外部格式波段的。可以在 PostgreSQL 的配置文件：postgresql.conf 中设置。也可以通过接口或事件来设置。

postgis.enable_outdb_rasters 的初始值也可以通过将具有非零值的环境变量 POSTGIS_ENABLE_OUTDB_RASTER 传递给 PostgreSQL 的程序来设置。



Note

即使 postgis.enable_outdb_rasters 为 True，GUC postgis.gdal_enabled_drivers 也会确定可使用的格式。



Note

在标准 PostGIS 安装中，postgis.enable_outdb_rasters 设置为 False。

可用性：2.2.0

示例

Set and reset postgis.enable_outdb_rasters for the current session.

Code

```
SET postgis.enable_outdb_rasters TO True;
SET postgis.enable_outdb_rasters = default;
SET postgis.enable_outdb_rasters = True;
SET postgis.enable_outdb_rasters = False;
```

Set the default for all new connections to a database.

Code

```
ALTER DATABASE gisdb SET postgis.enable_outdb_rasters = true;
```

Set the default for the whole database cluster. This requires superuser access. Also note that database, session, and user settings override this.

This example writes the setting to postgres.auto.conf.

Code

```
ALTER SYSTEM SET postgis.enable_outdb_rasters = true;
```

Reload the PostgreSQL configuration so the changed setting is applied.

Code

```
SELECT pg_reload_conf();
```

相关信息

[postgis.gdal_enabled_drivers](#) [postgis.gdal_vsi_options](#)

7.22.4 postgis.gdal_vsi_options

postgis.gdal_vsi_options — 用于设置处理外部数据格式使用的的字符串配置。

描述

用于设置处理外部数据格式使用的的字符串配置。[配置](#)控制如 GDAL 分配本地数据存的空大小、是否取概述以及程 out-db 数据源使用些等。

可用性：3.2.0

示例

当前会设置 postgis.gdal_vsi_options：

Code

```
SET postgis.gdal_vsi_options = 'AWS_ACCESS_KEY_ID=xxxxxxxxxxxxx AWS_SECRET_ACCESS_KEY=
yyyyyyyyyyyyyyyyyyyyyyyyyyyy';
```

使用 LOCAL 关键字当前事设置 postgis.gdal_vsi_options：

Code

```
SET LOCAL postgis.gdal_vsi_options = 'AWS_ACCESS_KEY_ID=xxxxxxxxxxxxx
AWS_SECRET_ACCESS_KEY=yyyyyyyyyyyyyyyyyyyyyyyyyyyy';
```

相关信息

[postgis.enable_outdb_rasters](#) [postgis.gdal_enabled_drivers](#)

7.22.5 postgis.gdal_cpl_debug

postgis.gdal_cpl_debug — 用于开或关 GDAL 信息日志的布型配置。

描述

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 detailed GDAL logging to be sent into the PostgreSQL server log. PostgreSQL clients receive only generic GDAL notice, debug, warning, or error messages.

可用性：3.6.0

相关信息

[postgis.enable_outdb_rasters](#) [postgis.gdal_enabled_drivers](#)

7.23 故障排除函数

7.23.1 PostGIS_AddBBox

PostGIS_AddBBox — 向几何体添加框。

Synopsis

geometry **PostGIS_AddBBox**(geometry geomA);

描述

向几何体添加边界框。这将使基于边界框的查询更快，但会增加几何形状的大小。



Note

边界框会自添加到几何形状中，因此通常不需要这样做，除非生成的边界框以某种方式损坏或者您的旧安装缺少边界框。然后您需要删除旧的并重新获取。



此方法支持几何字符串和曲线。

示例

Code

```
UPDATE sometable
SET geom = PostGIS_AddBBox(geom)
WHERE PostGIS_HasBBox(geom) = false;
```

相关信息

[PostGIS_DropBBox](#), [PostGIS_HasBBox](#)

7.23.2 PostGIS_DropBBox

PostGIS_DropBBox — 从几何体中删除边界框。

Synopsis

geometry **PostGIS_DropBBox**(geometry geomA);

描述

从几何体中删除边界框。这会减少几何尺寸，但会使基于边界框的查询变慢。它用于删除损坏的边界框。损坏边界框的一个迹象是 ST_Intersects 和其他关系函数漏了理应返回 true 的几何形状。



Note

边界框会自添加到几何形状中并提高查询速度，因此通常不需要这样做，除非生成的边界框以某种方式损坏或者您的旧安装缺少边界框。然后您需要删除旧的并重新获取。在 8.3-8.3.6 系列中已观察到这种损坏，其中当几何形状更改时，并不总是重新计算存储的 bbox，并且在不重新添加的情况下升级到新的版本将不会更正已损坏的框。因此，可以使用下面的方法手动更正并重新添加 bbox 或执行重新添加。



此方法支持几何字符串和曲线。

示例

This example drops cached bounding boxes where the stored box is wrong. Applying `Box2D` to `ST_AsBinary` forces a fresh box calculation, while `Box2D` on the table geometry reads the cached box.

Code

```
UPDATE sometable
SET geom = PostGIS_DropBBox(geom)
WHERE Not (Box2D(ST_AsBinary(geom)) = Box2D(geom));
```

Output

```
UPDATE ...
```

After dropping corrupt cached boxes, re-add them where needed.

Code

```
UPDATE sometable
SET geom = PostGIS_AddBBox(geom)
WHERE Not PostGIS_HasBBOX(geom);
```

Output

```
UPDATE ...
```

相关信息

[PostGIS_AddBBox](#), [PostGIS_HasBBox](#), [Box2D](#)

7.23.3 PostGIS_HasBBox

`PostGIS_HasBBox` — 如果几何体的 `bbox` 已存, 返回 `TRUE`, 否则返回 `FALSE`。

Synopsis

boolean **PostGIS_HasBBox**(geometry geomA);

描述

如果几何体的 `bbox` 已存, 返回 `TRUE`, 否则返回 `FALSE`。使用 [PostGIS_AddBBox](#) 和 [PostGIS_DropBBox](#) 来控制存。



此方法支持形字符串和曲线。

示例

I Code

```
SELECT geom
FROM sometable WHERE PostGIS_HasBBox(geom) = false;
```

相关信息

[PostGIS_AddBBox](#), [PostGIS_DropBBox](#)

Chapter 8

SFCGAL 函数参考

SFCGAL 是 CGAL 的 C++ 封装，提供高精度的二维和三维空间函数。为了确保稳健性，几何坐标具有精确的有理数表示。

详细的安装说明可以在 SFCGAL 主页 (<http://www.sfcgal.org>) 上找到。要使用这些功能，请使用 `create extension postgis_sfcgal`。

8.1 SFCGAL 管理函数

8.1.1 `postgis_sfcgal_version`

`postgis_sfcgal_version` — 返回正在使用的 SFCGAL 版本

Synopsis

文本 `postgis_sfcgal_version` 描述 (void);

描述

返回正在使用的 SFCGAL 版本

可用性 : 2.1.0



该方法需要 SFCGAL 后端。

相关信息

[postgis_sfcgal_full_version](#)

8.1.2 `postgis_sfcgal_full_version`

`postgis_sfcgal_full_version` — 返回正在使用的 SFCGAL 的完整版本，包括 CGAL 和 Boost 版本

Synopsis

文本 `postgis_sfcgal_full_version`(void);

描述

返回正在使用的 SFCGAL 的完整版本，包括 CGAL 和 Boost 版本

可用性 : 3.3.0



☒方法需要 SFCGAL 后端。

相关信息

[postgis_sfcgal_version](#)

8.2 SFCGAL ☒☒器和☒置器

8.2.1 CG_ForceLHR

CG_ForceLHR — ☒制 LHR 方向

Synopsis

geometry **CG_ForceLHR**(geometry geom);

描述

Returns an equivalent geometry whose polygonal components follow the left-hand rule: exterior rings are counter-clockwise and interior rings are clockwise. Rings are reversed as needed; coordinates and dimensionality are preserved. The input must be valid.

可用性 : 3.5.0



☒方法需要 SFCGAL 后端。



☒函数支持 3d 并且不会☒失 z-index。



☒函数支持多面体曲面。



此函数支持三角形和不☒☒三角网面 (TIN)。

示例

Reorient a polygon and its hole to the left-hand rule.

Code

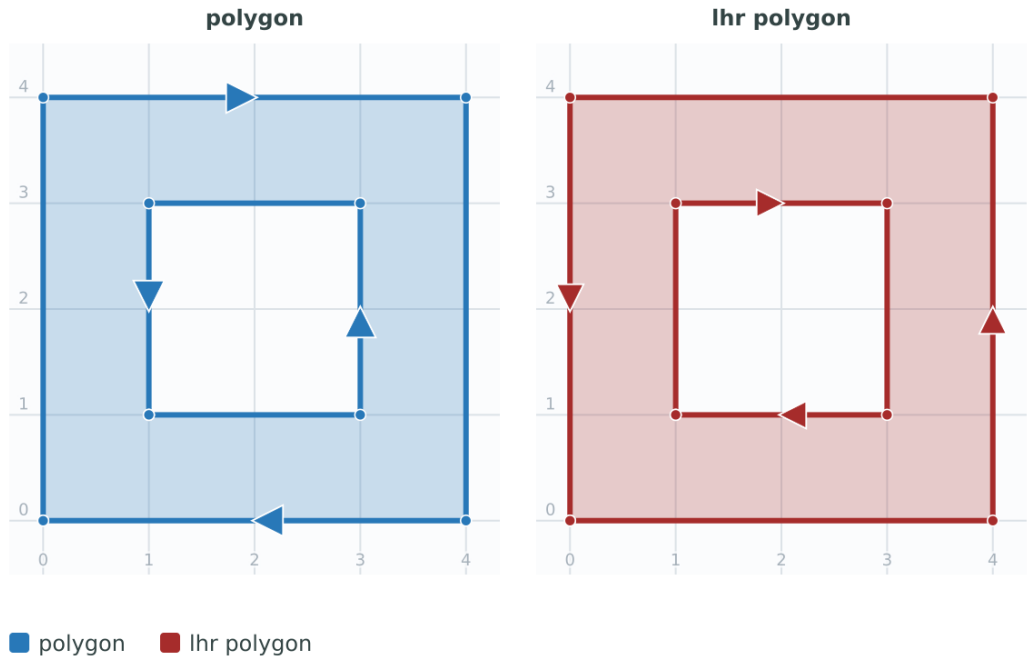
```

WITH data AS (
  SELECT 'POLYGON((0 0,0 4,4 4,4 0,0 0),
                  (1 1,3 1,3 3,1 3,1 1))'::geometry AS geom
)
SELECT geom AS input_polygon,
       CG_ForceLHR(geom) AS lhr_polygon,
       ST_IsPolygonCCW(geom) AS input_is_lhr,
       ST_IsPolygonCCW(CG_ForceLHR(geom)) AS output_is_lhr
FROM data;
```

Output

```
POLYGON((0 0,0 4,4 4,4 0,0 0),(1 1,3 1,3 3,1 3,1 1)) | POLYGON((0 0,4 0,4 4,0 4,0 0),(1 1,1 3,3 3,3 1,1 1)) | f | t
```

Figure



8.2.2 CG_IsPlanar

CG_IsPlanar — 判断表面是否平坦

Synopsis

```
boolean CG_IsPlanar(geometry geom);
```

描述

Tests whether the input geometry lies in a single plane in 3D.

可用性 : 3.5.0

- ✓ 方法需要 SFCGAL 后端。
- ✓ 函数支持 3d 并且不会丢失 z-index。
- ✓ 函数支持多面体曲面。
- ✓ 此函数支持三角形和不规则三角网面 (TIN)。

8.2.3 CG_IsSolid

CG_IsSolid — 判断几何体是否实体。不执行有效性检查。

Synopsis

boolean **CG_IsSolid**(geometry geom1);

描述

Returns true when the input carries the solid flag, such as a geometry produced by **CG_MakeSolid**. A closed PolyhedralSurface remains a surface until it is explicitly converted. This tests the representation, not shell closure or validity; no validity check is performed.

可用性 : 3.5.0



☑方法需要 SFCGAL 后端。



☑函数支持 3d 并且不会☑失 z-index。



☑函数支持多面体曲面。



☑此函数支持三角形和☑☑三角网面 (TIN)。

示例

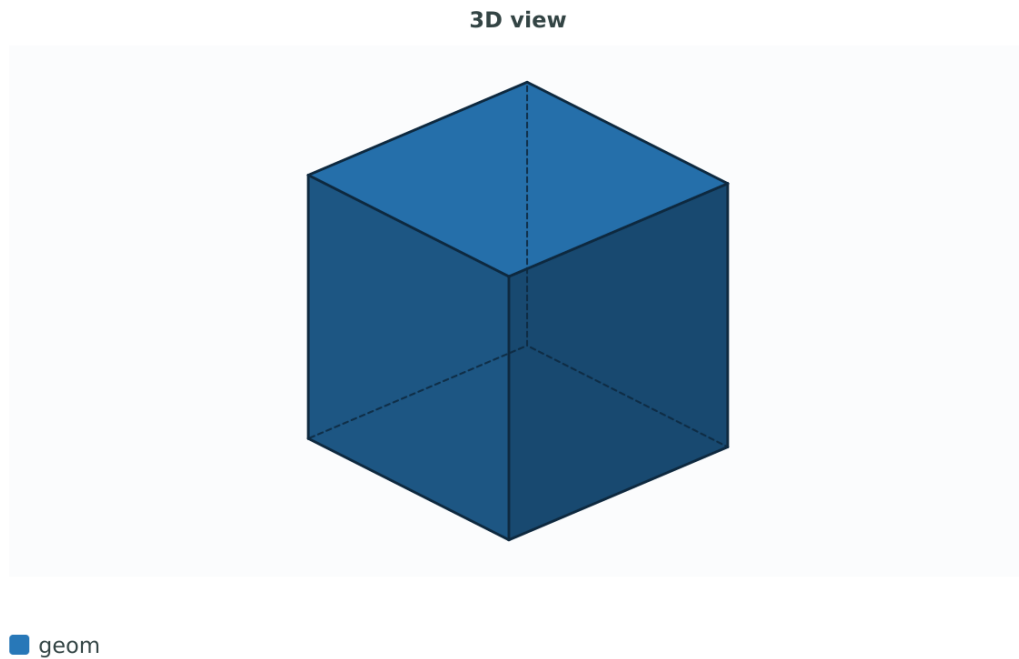
Compare the same closed shell before and after converting it to a solid.

Code

```
WITH data AS (
  SELECT '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))
  )'::geometry AS geom
)
SELECT CG_IsSolid(geom) AS shell_is_solid,
       CG_IsSolid(CG_MakeSolid(geom)) AS solid_is_solid
FROM data;
```

Output

f | t

Figure

8.2.4 CG_MakeSolid

CG_MakeSolid — 将几何体造成体。不行任何。要得有效的体，入几何形必是合多面体曲面或合 TIN。

Synopsis

```
geometry CG_MakeSolid(geometry geom1);
```

描述

Marks a PolyhedralSurface or TIN as a solid. The input is expected to describe a closed shell; the function sets the solid representation and does not validate the shell.

可用性 : 3.5.0

- ✓ 方法需要 SFCGAL 后端。
- ✓ 函数支持 3d 并且不会失 z-index。
- ✓ 函数支持多面体曲面。
- ✓ 此函数支持三角形和不三角网面 (TIN)。

8.2.5 CG_Orientation

CG_Orientation — 确定表面方向

Synopsis

integer **CG_Orientation**(geometry geom);

描述

该函数适用于多边形。如果多边形是逆时针方向，返回 -1；如果多边形是顺时针方向，返回 1。

可用性：3.5.0



该方法需要 SFCGAL 后端。



该函数支持 3d 并且不会丢失 z-index。

8.2.6 CG_Area

CG_Area — 计算几何图形的面积

Synopsis

double precision **CG_Area**(geometry geom);

描述

计算几何图形的面积。

由 SFCGAL 模块执行



Note

注意：此函数返回一个表示面积的双精度浮点数。

可用性：3.5.0



该方法需要 SFCGAL 后端。

几何示例

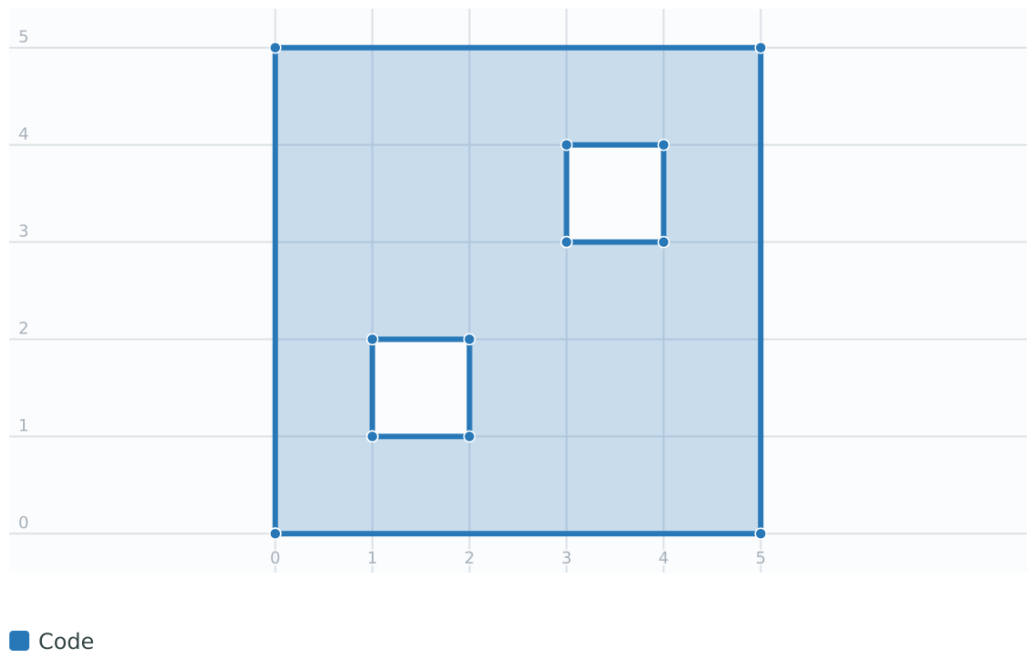
Code

```
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))
```

Output

23

Figure



相关信息

[CG_3DArea](#), [ST_Area](#)

8.2.7 CG_3DArea

CG_3DArea — 计算 3D 表面几何形状的面积。空于固体将返回 0。

Synopsis

```
floatCG_3DArea(geometry geom1);
```

描述

可用性：3.5.0

- ✓ 方法需要 SFCGAL 后端。
- ✓ 方法符合了 SQL/MM 规范。SQL-MM IEC 13249-3: 8.1, 10.5
- ✓ 函数支持 3d 并且不会丢失 z-index。
- ✓ 函数支持多面体曲面。
- ✓ 此函数支持三角形和不规则三角网面 (TIN)。

示例

注意：默认情况下，从 WKT 构建的 PolyhedralSurface 是曲面几何体，而不是体。因此它具有表面。一旦体，就没有表面了。

Code

```
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);
```

Output

cube_surface_area	solid_surface_area
6	0

相关信息

[CG_Area](#), [CG_MakeSolid](#), [CG_IsSolid](#), [CG_Area](#)

8.2.8 CG_Volume

CG_Volume — Computes the volume of a 3D geometry. Closed surfaces can have non-zero volume.

Synopsis

float **CG_Volume**(geometry geom1);

描述

可用性：3.5.0

- ✓ 方法需要 SFCGAL 后端。
- ✓ 函数支持 3d 并且不会丢失 z-index。
- ✓ 函数支持多面体曲面。
- ✓ 此函数支持三角形和不三角网面 (TIN)。
- ✓ 方法遵循了 SQL/MM 规范。SQL-MM IEC 13249-3: 9.1（与 CG_3DVolume 相同）

Closed surface geometries can have non-zero volume. Use [CG_MakeSolid](#) when the geometry value itself must carry the solid flag.

示例

A closed polyhedral surface has volume. [CG_MakeSolid](#) makes the solid representation explicit.

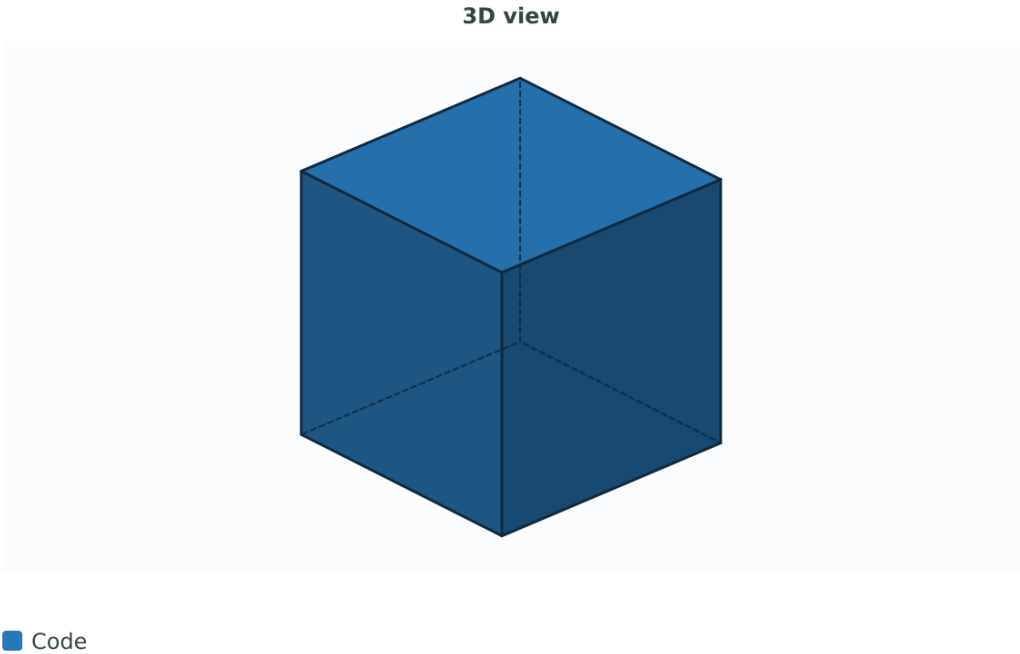
Code

```
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);
```

Output

cube_surface_vol	solid_surface_vol
1	1

Figure



相关信息

[CG_3DArea](#), [CG_MakeSolid](#), [CG_IsSolid](#)

8.3 SFCGAL 几何和关系函数

8.3.1 CG_Intersection

CG_Intersection — 计算两个几何图形的相交（共享）部分

Synopsis

geometry **CG_Intersection**(geometry geomA , geometry geomB);

描述

计算两个几何图形的交集（共享部分）。

由 SFCGAL 模块行



Note

注意：此函数返回表示交集部分的几何图形。

可用性：3.5.0



该方法需要 SFCGAL 后端。

几何示例

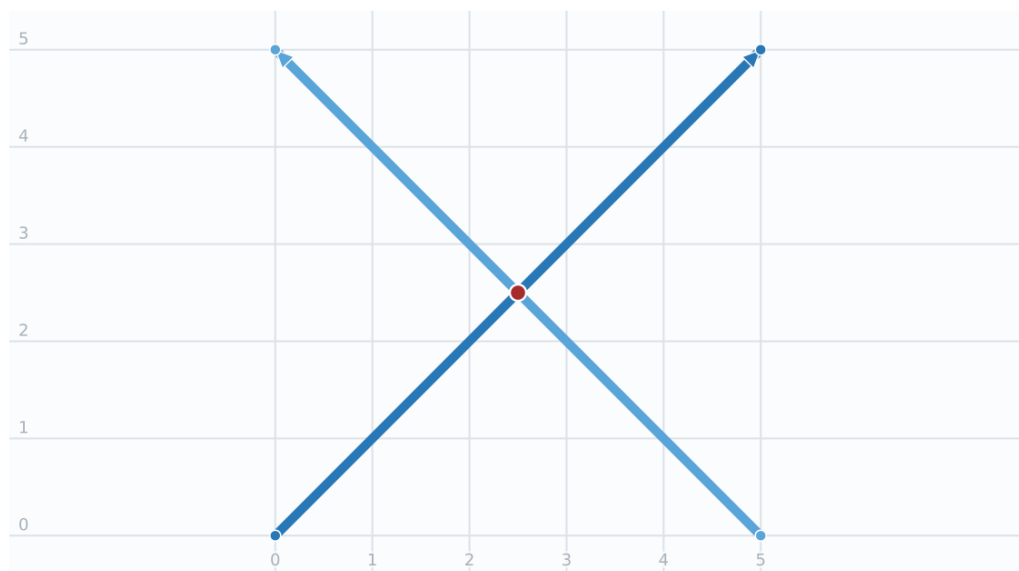
Code

```
SELECT CG_Intersection('LINESTRING(0 0,5 5)', 'LINESTRING(5 0,0 5)');
```

Output

```
POINT(2.5 2.5)
```

Figure



■ Code 1 ■ Code 2 ■ Output

相关信息

[CG_3DIntersection](#), [ST_Intersection](#)

8.3.2 CG_Intersects

CG_Intersects — 两个几何图形是否相交（它们至少有一个共同点）

Synopsis

```
boolean CG_Intersects( geometry geomA , geometry geomB );
```

描述

如果两个几何图形相交，返回 **true**。如果几何图形有任何共同点，它们相交。

由 SFCGAL 模块行



Note

注意：它是返回布尔值而不是整数的“允许”版本。

可用性：3.5.0



此方法需要 SFCGAL 后端。



此函数支持三角形和不规则三角网面 (TIN)。

几何示例

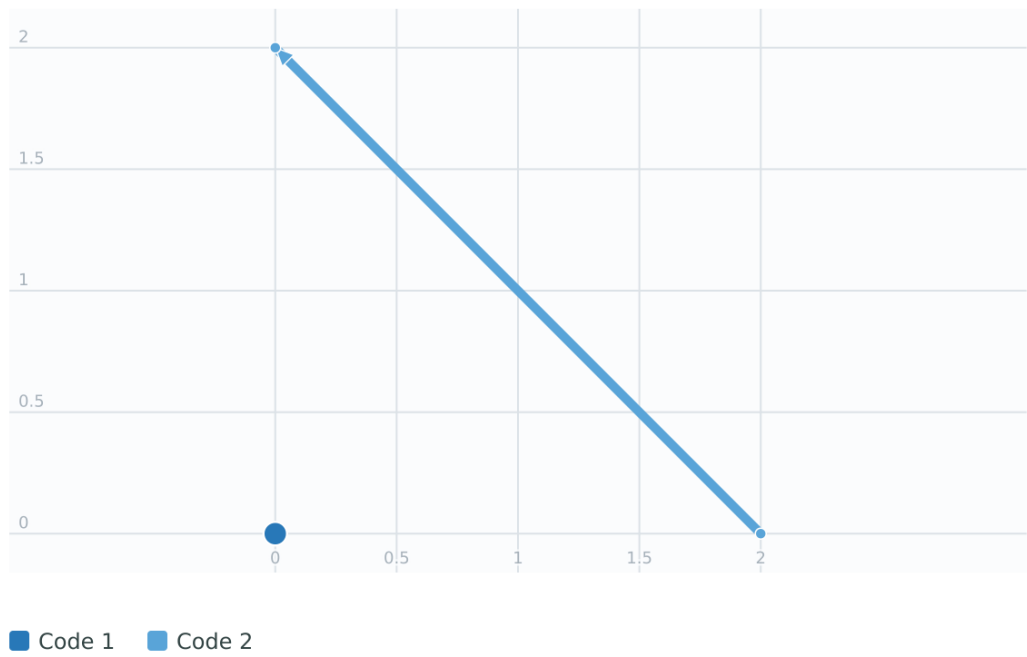
Code

```
SELECT CG_Intersects('POINT(0 0)::geometry', 'LINESTRING ( 2 0,0 2 )::geometry);
```

Output

```
f
```

Figure



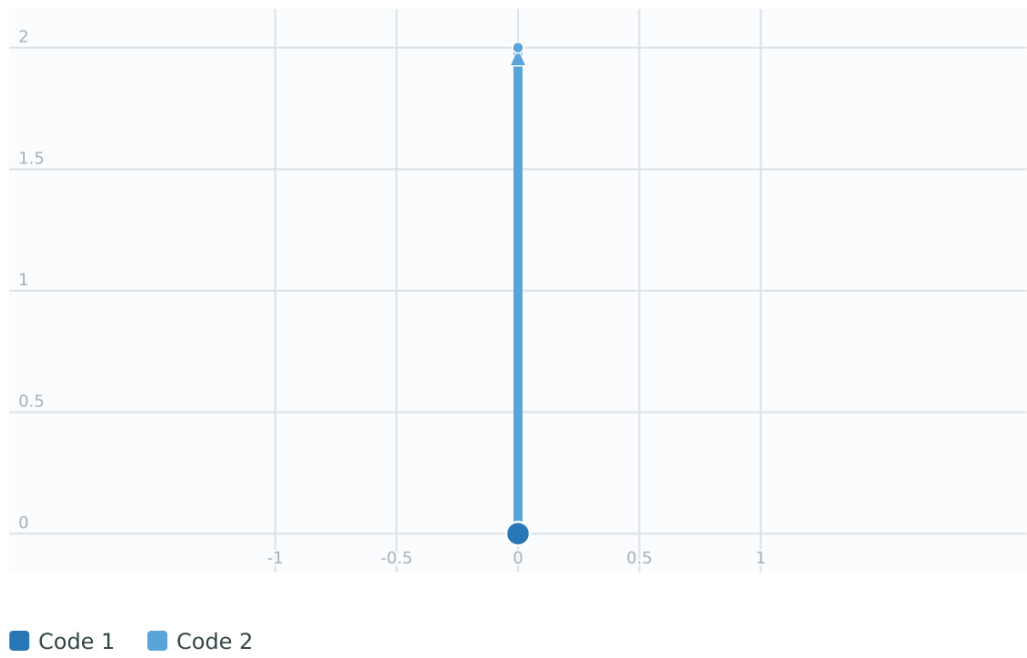
Code

```
SELECT CG_Intersects('POINT(0 0)::geometry', 'LINESTRING ( 0 0,0 2 ) '::geometry);
```

Output

t

Figure



相关信息

[CG_3DIntersects](#), [ST_3DIntersects](#), [ST_Intersects](#), [ST_Disjoint](#)

8.3.3 CG_3DIntersects

CG_3DIntersects — 两个三维几何图形是否相交

Synopsis

boolean **CG_3DIntersects**(geometry geomA , geometry geomB);

描述

两个三维几何图形是否相交。如果在三维空间中存在共同的点，两个三维几何图形相交。
由 SFCGAL 模块行



Note

注意：是返回布尔而不是整数的“允许”版本。

可用性：3.5.0



方法需要 SFCGAL 后端。



此函数支持三角形和不三角网面 (TIN)。

几何示例

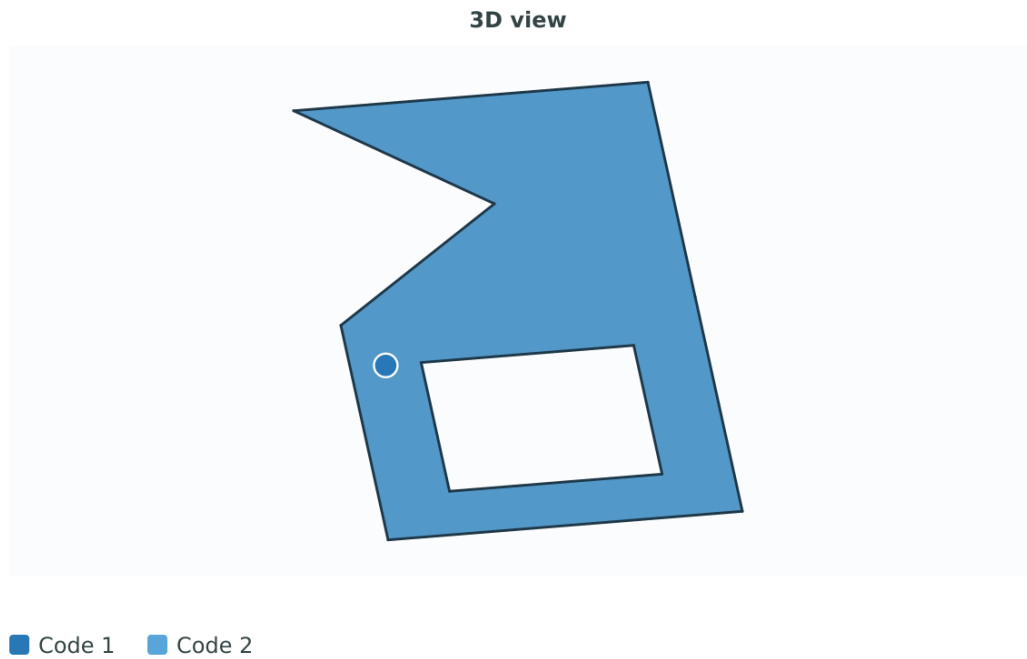
Code

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

Output

t

Figure



相关信息

[CG_Intersects](#), [ST_3DIntersects](#), [ST_Intersects](#), [ST_Disjoint](#)

8.3.4 CG_Difference

CG_Difference — 计算两个几何图形之间的几何差异

Synopsis

```
geometry CG_Difference( geometry geomA , geometry geomB );
```

描述

计算两个几何图形之间的几何差异。返回的几何图形是一个点集，包含在 geomA 中但不在 geomB 中的点。
由 SFCGAL 模块执行



Note

注意：此函数返回几何图形。

可用性：3.5.0



该方法需要 SFCGAL 后端。



此函数支持三角形和不规则三角网面 (TIN)。

几何示例

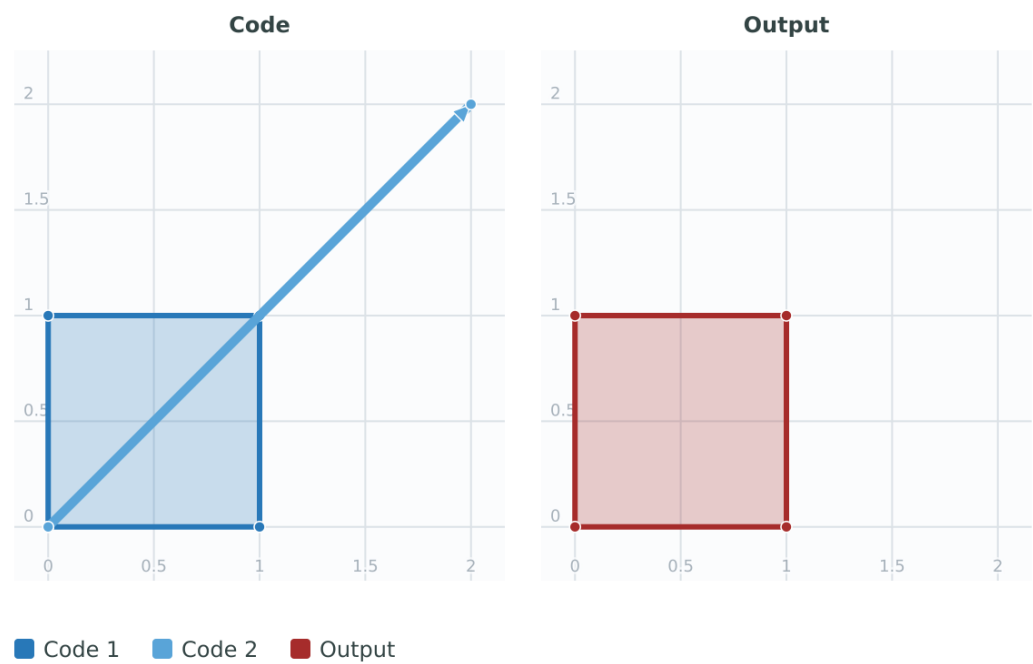
Code

```
SELECT CG_Difference('POLYGON((0 0,0 1,1 1,1 0,0 0))'::geometry, 'LINESTRING(0 0,2 2)':: geometry);
```

Output

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

Figure



相关信息

[CG_3DDifference](#), [ST_Difference](#)

8.3.5 CG_3DDifference

CG_3DDifference — 行 3D 差异

Synopsis

geometry **CG_3DDifference**(geometry geom1, geometry geom2);

描述

返回 geom1 中不属于 geom2 的部分。

可用性 : 3.5.0

- ☑ 方法需要 SFCGAL 后端。
- ☑ 方法☑了 SQL/MM ☑范。SQL-MM IEC 13249-3: 5.1
- ☑ 函数支持 3d 并且不会☑失 z-index。
- ☑ 函数支持多面体曲面。
- ☑ 此函数支持三角形和☑☑三角网面 (TIN)。

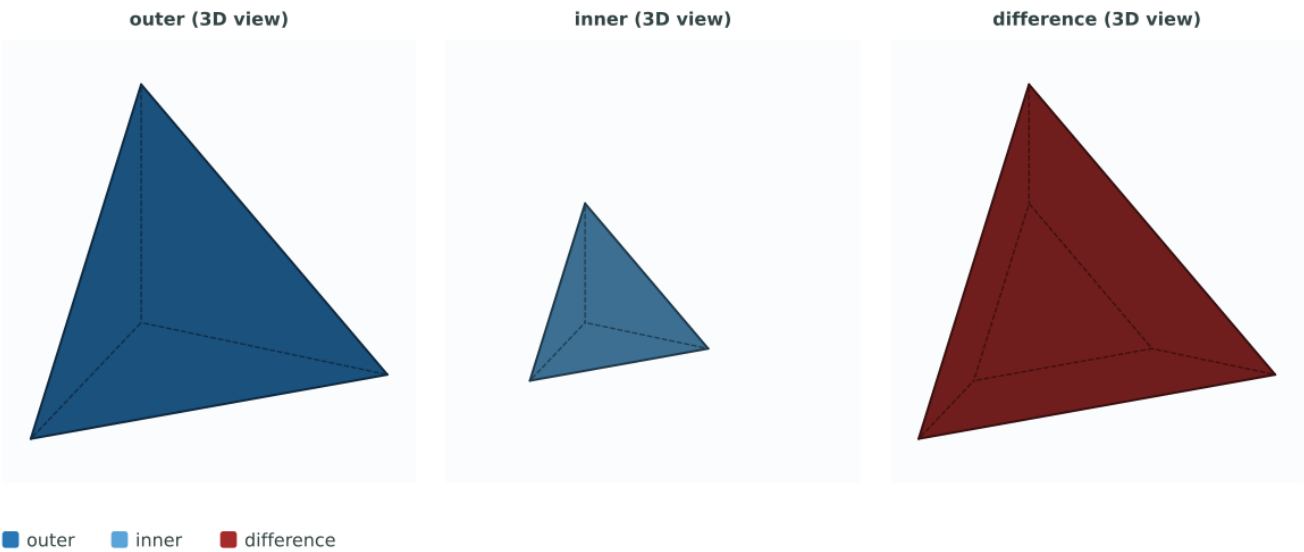
示例

Subtract a smaller tetrahedral solid and render the two inputs and the actual remaining solid separately.

Code

```
WITH data AS (  
  SELECT CG_MakeSolid('POLYHEDRALSURFACE Z (  
    ((0 0 0,0 4 0,4 0 0,0 0 0)),  
    ((0 0 0,4 0 0,0 0 4,0 0 0)),  
    ((0 0 0,0 0 4,0 4 0,0 0 0)),  
    ((4 0 0,0 4 0,0 0 4,4 0 0))  
  )::geometry) AS outer_solid  
, solids AS (  
  SELECT outer_solid,  
    ST_Scale(outer_solid, 0.5, 0.5, 0.5) AS inner_solid  
  FROM data  
)  
SELECT outer_solid AS input_outer,  
  inner_solid AS input_inner,  
  CG_3DDifference(outer_solid, inner_solid) AS difference  
FROM solids;
```

Figure



相关信息

[CG_Extrude](#), [ST_AsX3D](#), [CG_3DIntersection](#) [CG_3DUnion](#)

8.3.6 CG_Distance

CG_Distance — 返回两个几何图形之间的最小距离

Synopsis

double precision **CG_Distance**(geometry geomA , geometry geomB);

描述

返回两个几何图形之间的最小距离。

由 SFCGAL 模块执行



Note

注意：此函数返回一个表示距离的双精度浮点数。

可用性：3.5.0



此方法需要 SFCGAL 后端。



此函数支持三角形和不规则三角网面 (TIN)。

几何示例

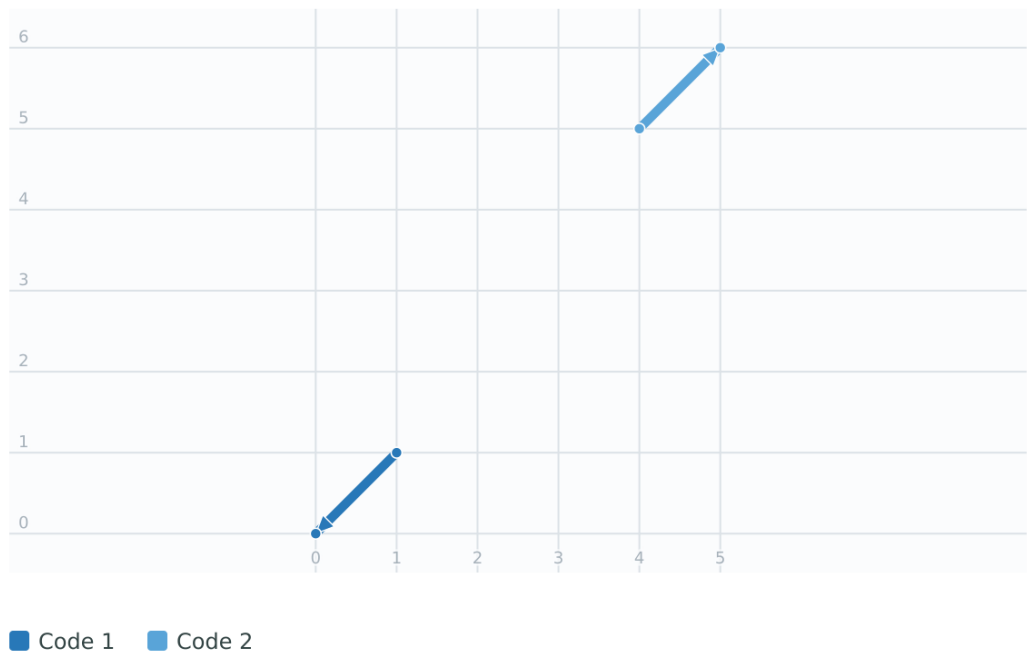
Code

```
SELECT CG_Distance('LINESTRING(0.0 0.0,-1.0 -1.0)', 'LINESTRING(3.0 4.0,4.0 5.0)');
```

Output

5

Figure



相关信息

[CG_3DDistance](#), [CG_Distance](#)

8.3.7 CG_3DDistance

`CG_3DDistance` — 返回两个三维几何图形之间的最小距离

Synopsis

```
double precision CG_3DDistance( geometry geomA , geometry geomB );
```

描述

返回两个几何图形之间的最小三维距离。

由 SFCGAL 模块行



Note

注意：此函数返回一个表示三维距离的双精度浮点数。

可用性：3.5.0

- ✓ 方法需要 SFCGAL 后端。
- ✓ 此函数支持三角形和不规则三角网面 (TIN)。

几何示例

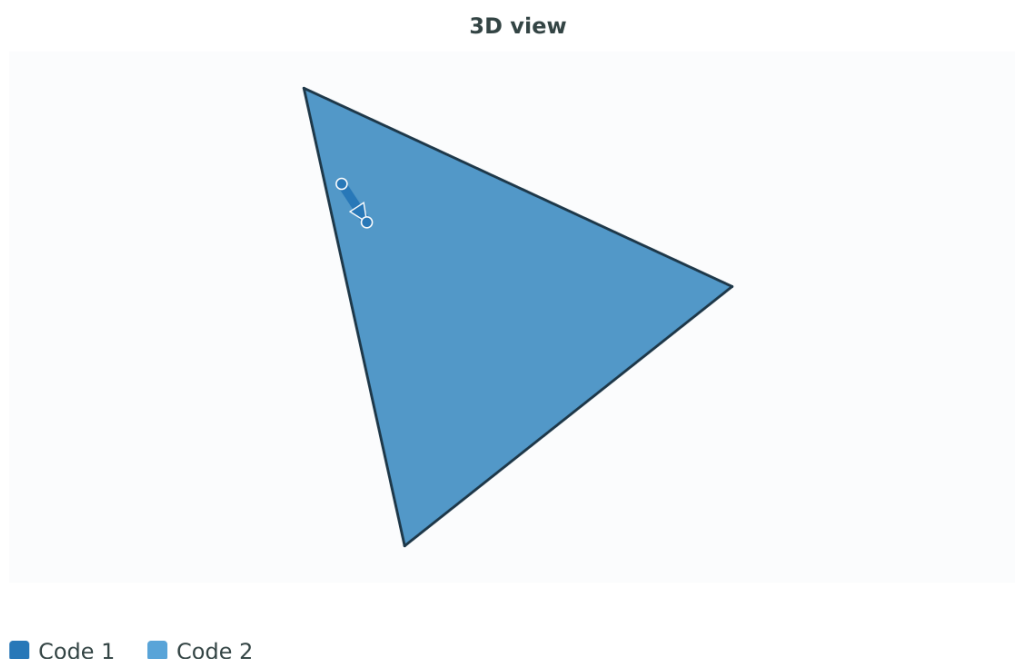
Code

```
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))');
```

Output

1

Figure



相关信息

[CG_Distance](#), [ST_3DDistance](#)

8.3.8 CG_3DConvexHull

CG_3DConvexHull — 计算几何体的 3D 凸包。

Synopsis

```
geometry CG_3DConvexHull(geometry geom1);
```

描述

Computes the smallest convex three-dimensional geometry that contains the input. Unlike [ST_ConvexHull](#), this function uses Z coordinates when constructing the hull.

可用性 : 3.5.0

- ✓ ☑ 方法需要 SFCGAL 后端。
- ✓ ☑ 函数支持 3d 并且不会 ☑ 失 z-index。
- ✓ ☑ 函数支持多面体曲面。
- ✓ ☑ 此函数支持三角形和不 ☑ 三角网面 (TIN)。

示例

Compute the 3D convex hull of a 3D LineString.

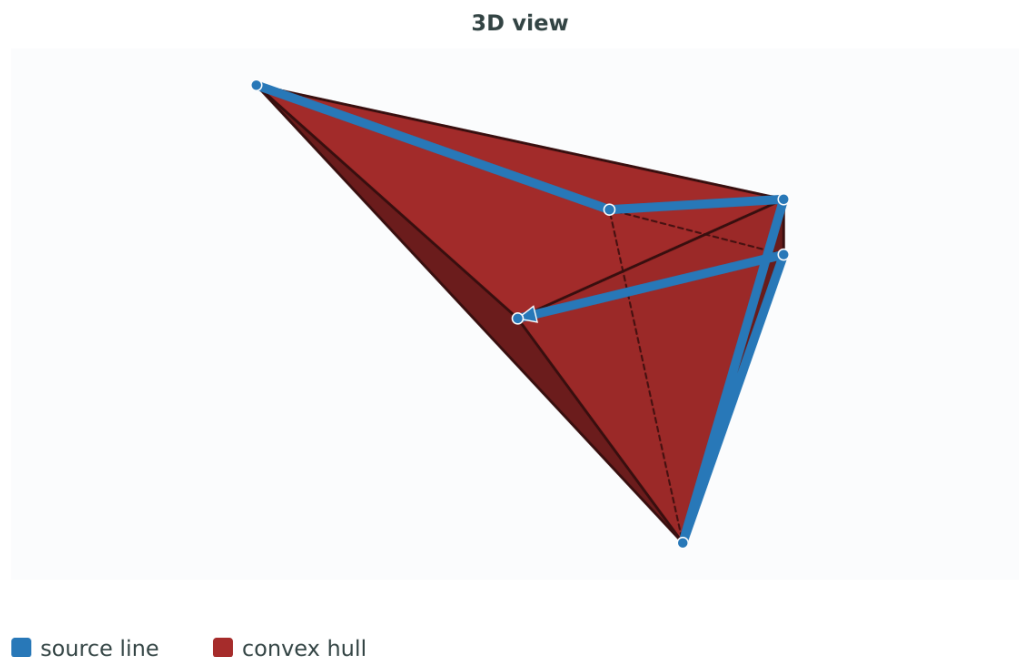
Code

```
WITH data AS (
  SELECT 'LINESTRING Z(0 0 5,1 5 3,5 7 6,9 5 3,5 7 5,6 3 5)::geometry' AS source_line
)
SELECT CG_3DConvexHull(source_line) AS convex_hull
FROM data;
```

Output

```
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)))
```

Figure



Compare the extruded source geometry with its 3D convex hull.

Code

```
WITH footprints(label, geom, height) AS (VALUES
  ('C', 'POLYGON((0 0,0 8,5 8,5 6,2 6,2 2,5 2,5 0,0 0))::geometry', 3),
  ('H', 'POLYGON((7 0,9 0,9 3,12 3,12 0,14 0,14 8,12 8,12 5,9 5,9 8,7 8,7 0))::geometry', 7)
)
```

```

), parts AS (
  SELECT label, CG_Extrude(geom, 0, 0, height) AS geom
  FROM footprints
), shapes AS (
  SELECT ST_Collect(geom ORDER BY label) AS source_geometry
  FROM parts
)
SELECT
  source_geometry AS input_source_geometry,
  CG_3DConvexHull(source_geometry) AS convex_hull
FROM shapes;

```

Output

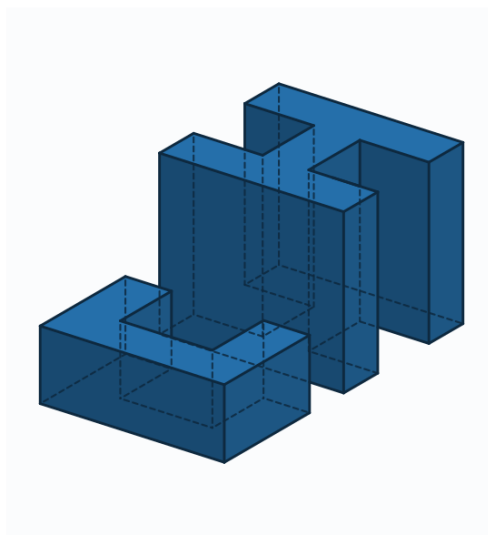
```

-[ RECORD 1 ]-----
input_source_geometry | GEOMETRYCOLLECTION(POLYHEDRALSURFACE(((0 0 0,0 8 0,5 8 0,5 6 0,2 6 0,2 2 0,5 2 0,5 0 0,0 0 0)) ←
,((0 0 3,5 0 3,5 2 3,2 2 3,2 6 3,5 6 3,5 8 3,0 8 3,0 0 3)),((0 0 0,0 0 3,0 8 3,0 8 0,0 0 0)),((0 8 0,0 8 3,5 8 ←
3,5 8 0,0 8 0)),((5 8 0,5 8 3,5 6 3,5 6 0,5 8 0)),((5 6 0,5 6 3,2 6 3,2 6 0,5 6 0)),((2 6 0,2 6 3,2 2 3,2 2 0,2 6 ←
0)),((2 2 0,2 2 3,5 2 3,5 2 0,2 2 0)),((5 2 0,5 2 3,5 0 3,5 0 0,5 2 0)),((5 0 0,5 0 3,0 0 3,0 0 0,5 0 0)), ←
POLYHEDRALSURFACE(((7 0 0,7 8 0,9 8 0,9 5 0,12 5 0,12 8 0,14 8 0,14 0 0,12 0 0,12 3 0,9 3 0,9 0 0,7 0 0)),((7 0 ←
7,9 0 7,9 3 7,12 3 7,12 0 7,14 0 7,14 8 7,12 8 7,12 5 7,9 5 7,9 8 7,7 8 7,7 0 7)),((7 0 0,7 0 7,7 8 7,7 8 0,7 0 ←
0)),((7 8 0,7 8 7,9 8 7,9 8 0,7 8 0)),((9 8 0,9 8 7,9 5 7,9 5 0,9 8 0)),((9 5 0,9 5 7,12 5 7,12 5 0,9 5 0)),((12 ←
5 0,12 5 7,12 8 7,12 8 0,12 5 0)),((12 8 0,12 8 7,14 8 7,14 8 0,12 8 0)),((14 8 0,14 8 7,14 0 7,14 0 0,14 8 0)) ←
,((14 0 0,14 0 7,12 0 7,12 0 0,14 0 0)),((12 0 0,12 0 7,12 3 7,12 3 0,12 0 0)),((12 3 0,12 3 7,9 3 7,9 3 0,12 3 ←
0)),((9 3 0,9 3 7,9 0 7,9 0 0,9 3 0)),((9 0 0,9 0 7,7 0 7,7 0 0,9 0 0)))
convex_hull | POLYHEDRALSURFACE(((14 8 7,14 0 0,14 8 0,14 8 7)),((14 0 7,7 8 7,7 0 7,14 0 7)),((0 8 3,7 8 ←
7,0 8 0,0 8 3)),((0 0 3,7 8 7,0 8 3,0 0 3)),((0 0 3,0 8 3,0 0 0,0 0 3)),((7 0 7,7 8 7,0 0 3,7 0 7)),((14 0 7,14 0 ←
0,14 8 7,14 0 7)),((7 0 7,14 0 0,14 0 7,7 0 7)),((14 8 7,7 8 7,14 0 7,14 8 7)),((0 0 0,0 8 3,0 8 0,0 0 0)),((0 0 ←
0,14 0 0,0 0 3,0 0 0)),((0 0 3,14 0 0,7 0 7,0 0 3)),((0 8 0,14 0 0,0 0 0,0 8 0)),((14 8 0,14 0 0,0 8 0,14 8 0)) ←
,((14 8 0,7 8 7,14 8 7,14 8 0)),((0 8 0,7 8 7,14 8 0,0 8 0)))

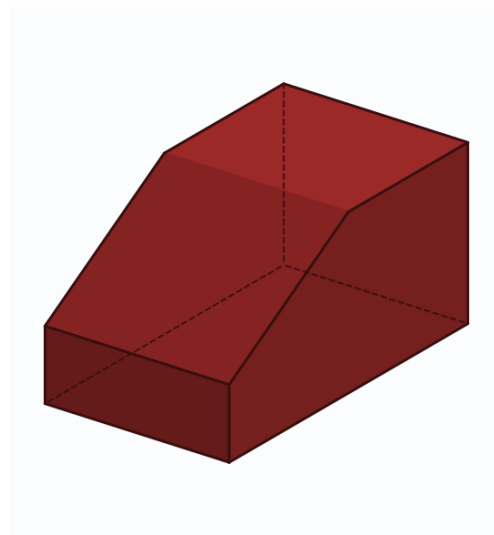
```

Figure

source geometry (3D view)



convex hull (3D view)



■ source geometry ■ convex hull

相关信息

[ST_Letters](#), [ST_AsX3D](#)

8.3.9 CG_3DIntersection

CG_3DIntersection — 行 3D 相交

Synopsis

geometry **CG_3DIntersection**(geometry geom1, geometry geom2);

描述

返回作 geom1 和 geom2 之共享部分的几何形。

可用性 : 3.5.0

- ✓ 方法需要 SFCGAL 后端。
- ✓ 方法符合了 SQL/MM 范。SQL-MM IEC 13249-3: 5.1
- ✓ 函数支持 3d 并且不会失 z-index。
- ✓ 函数支持多面体曲面。
- ✓ 此函数支持三角形和不三角网面 (TIN)。

示例

3D linestrings and polygons can intersect in a 3D segment.

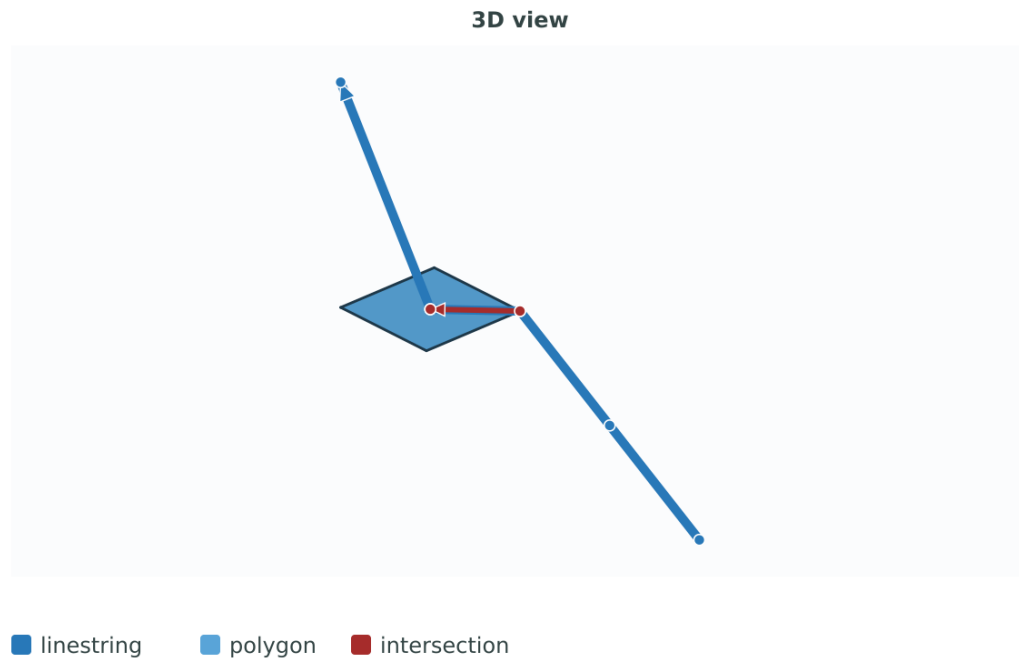
Code

```
WITH data AS (
  SELECT 'LINESTRING Z (2 2 6,1.5 1.5 7,1 1 8,0.5 0.5 8,0 0 10)>:::geometry AS
    input_linestring,
         'POLYGON Z ((0 0 8,0 1 8,1 1 8,1 0 8,0 0 8))>:::geometry AS input_polygon
)
SELECT input_linestring AS input_linestring,
       input_polygon AS input_polygon,
       CG_3DIntersection(input_linestring, input_polygon) AS intersection
FROM data;
```

Output

```
LINESTRING Z (2 2 6,1.5 1.5 7,1 1 8,0.5 0.5 8,0 0 10) | POLYGON Z ((0 0 8,0 1 8,1 1 8,1 0 8,0 0 8)) | LINESTRING Z (1 1 8,0.5 0.5 8)
```

Figure



Intersect a cube shell with a vertical polygon.

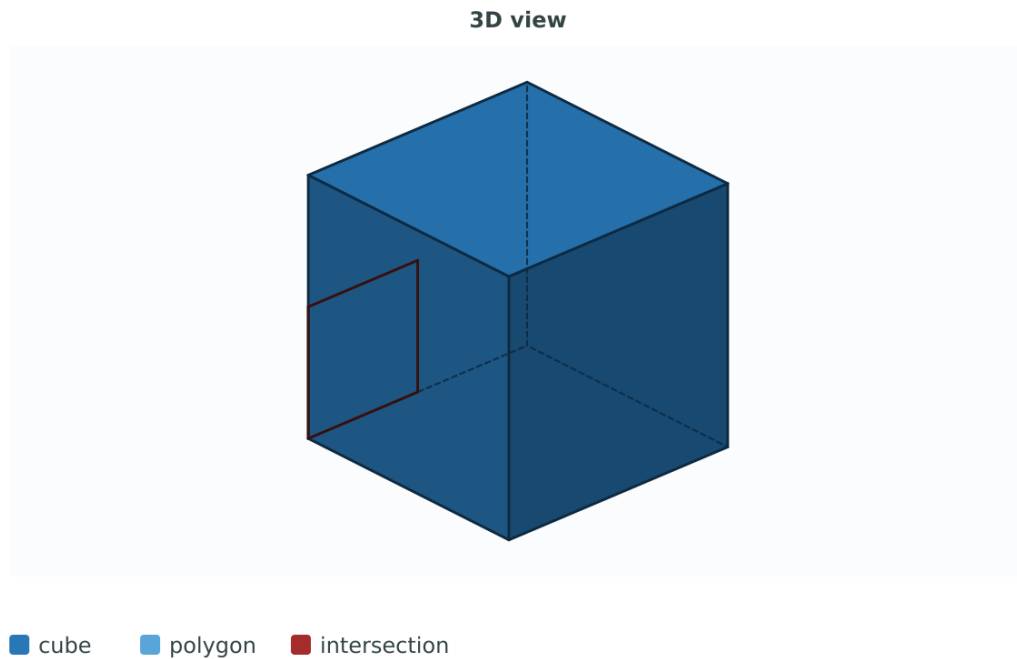
Code

```
WITH data AS (
  SELECT '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))
  )'::geometry AS input_cube,
  'POLYGON Z ((0 0 0,0 0.5,0 0.5 0.5,0 0.5 0,0 0 0))'::geometry AS input_polygon
)
SELECT input_cube AS input_cube,
       input_polygon AS input_polygon,
       CG_3DIntersection(input_cube, input_polygon) AS intersection
FROM data;
```

Output

```
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) ←
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 0.5,0 0.5 0.5,0 0.5 0,0 0 0)) | TIN Z (((0 0 0,0 0.5,0 0.5 0.5,0 0 0)),((0 0.5 0,0 0 0,0 ←
0.5 0.5,0 0.5 0)))
```


Figure



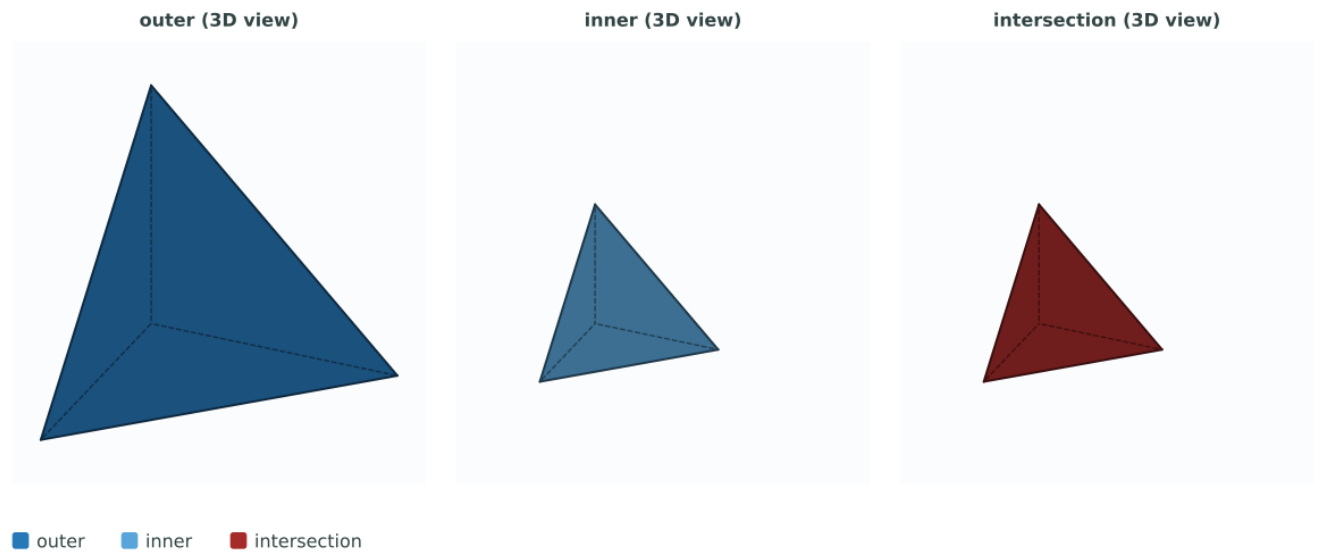
A smaller solid contained by a larger solid remains volumetric after intersection.

Code

```
WITH data AS (
  SELECT CG_MakeSolid('POLYHEDRALSURFACE Z (
    ((0 0 0,0 4 0,4 0 0,0 0 0)),
    ((0 0 0,4 0 0,0 0 4,0 0 0)),
    ((0 0 0,0 0 4,0 4 0,0 0 0)),
    ((4 0 0,0 4 0,0 0 4,4 0 0))
  )'::geometry) AS outer_solid
), solids AS (
  SELECT outer_solid,
         ST_Scale(outer_solid, 0.5, 0.5, 0.5) AS inner_solid
  FROM data
)
SELECT outer_solid AS input_outer,
       inner_solid AS input_inner,
       CG_3DIntersection(outer_solid, inner_solid) AS intersection
FROM solids;
```

Output

```
POLYHEDRALSURFACE Z (((0 0 0,0 4 0,4 0 0,0 0 0)),((0 0 0,4 0 0,0 0 4,0 0 0)),((0 0 0,0 0 4,0 4 0,0 0 0)),((4 0 0,0 4 0,0 0 4,4 0 0))) | POLYHEDRALSURFACE Z (((0 0 0,0 2 0,2 0 0,0 0 0)),((0 0 0,2 0 0,0 0 2,0 0 0)),((0 0 0,0 0 2,0 2 0,0 0 0)),((2 0 0,0 2 0,0 0 2,2 0 0))) | POLYHEDRALSURFACE Z (((0 0 0,0 2 0,2 0 0,0 0 0)),((0 0 0,0 0 2,0 2 0,0 0 0)),((2 0 0,0 0 2,0 0 2,2 0 0)),((2 0 0,0 2 0,0 2 2,2 0 0)))
```

Figure

8.3.10 CG_Union

CG_Union — 计算两个几何图形的合并

Synopsis

geometry **CG_Union**(geometry geomA , geometry geomB);

描述

计算两个几何图形的合并。

由 SFCGAL 模块行

**Note**

注意：此函数返回表示合并结果的几何图形。

可用性：3.5.0



该方法需要 SFCGAL 后端。

几何示例

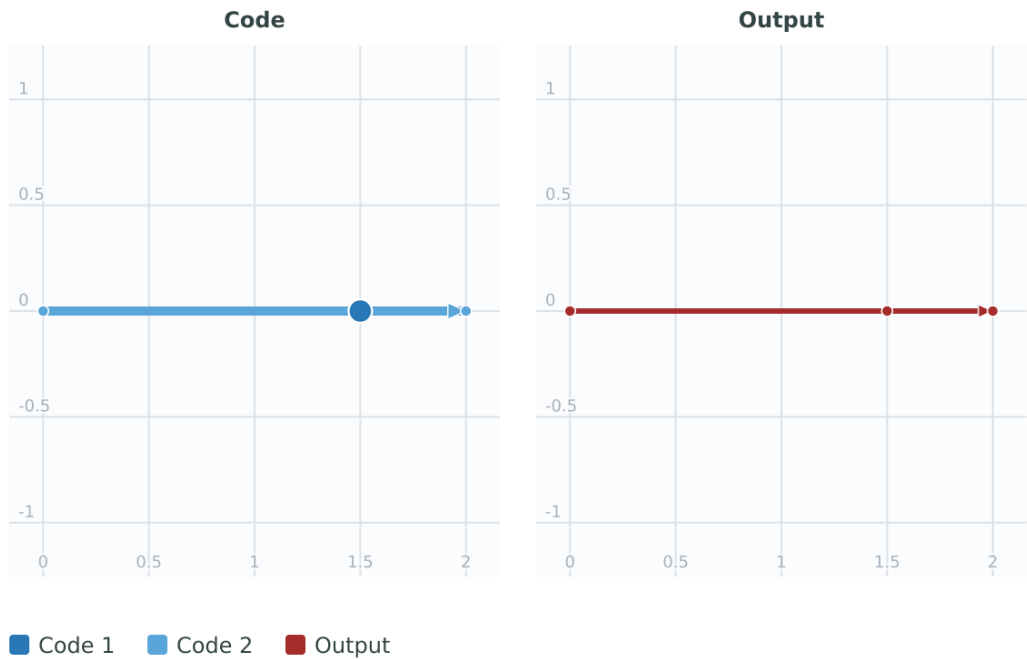
Code

```
SELECT CG_Union('POINT(.5 0)', 'LINESTRING(-1 0,1 0)');
```

Output

```
LINESTRING(-1 0,0.5 0,1 0)
```

Figure



相关信息

CG_3DUnion, ST_Union

8.3.11 CG_3DUnion

CG_3DUnion — 使用 postgis_sfcgal 行三合并。

Synopsis

```
geometry CG_3DUnion(geometry geom1, geometry geom2);
geometry CG_3DUnion(geometry set g1field);
```

描述

可用性 : 3.5.0

- ☑方法需要 SFCGAL 后端。
- ☑方法☑了 SQL/MM ☑范。SQL-MM IEC 13249-3: 5.1
- ☑函数支持 3d 并且不会☑失 z-index。
- ☑函数支持多面体曲面。

 此函数支持三角形和不规则三角网面 (TIN)。

集函数形式: 返回将几何行集合合并为一个三维几何的结果。CG_3DUnion() 函数是一个“集函数”，意味着它与 SUM()、MEAN() 等函数以相同的方式操作多个数据行，并像大多数其他集函数一样忽略 NULL 几何。

示例

Union two overlapping tetrahedral solids.

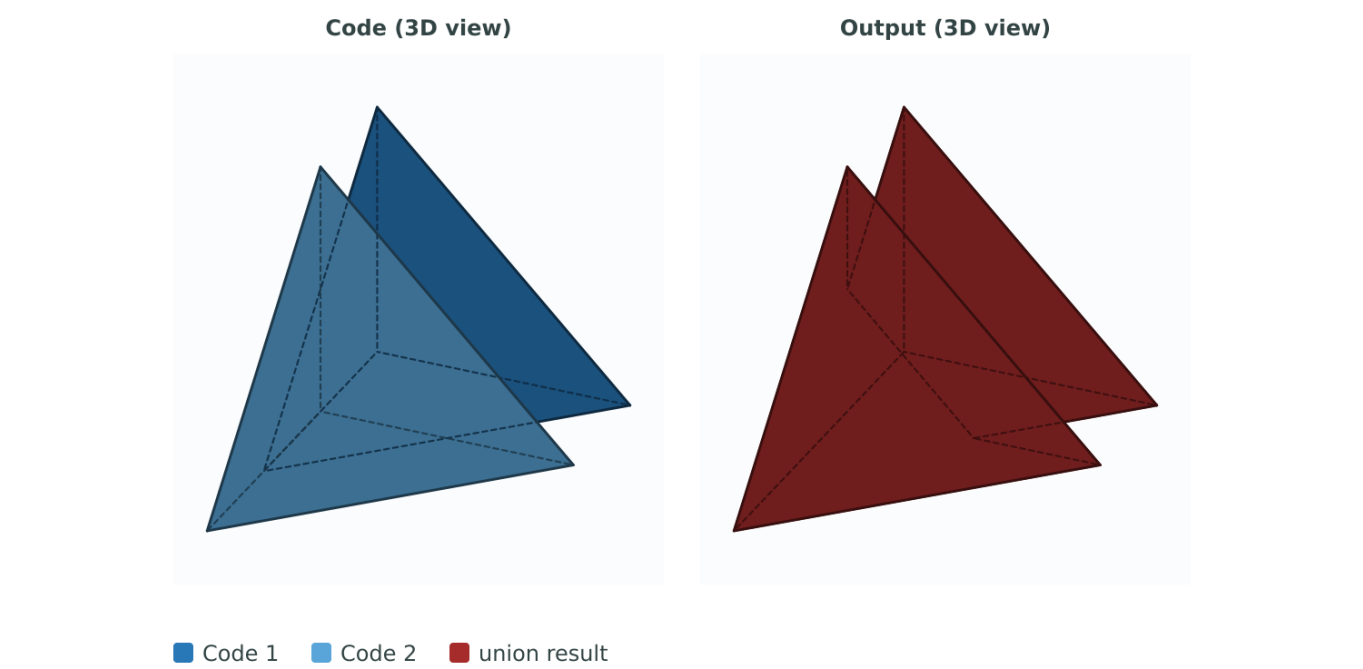
Code

```
WITH solids AS (  
  SELECT CG_MakeSolid('POLYHEDRALSURFACE Z (  
    ((0 0 0,4 0 0,0 4 0,0 0 0)),  
    ((0 0 0,0 0 4,4 0 0,0 0 0)),  
    ((0 0 0,0 4 0,0 0 4,0 0 0)),  
    ((4 0 0,0 0 4,0 4 0,4 0 0)))'::geometry) AS geom1,  
  CG_MakeSolid('POLYHEDRALSURFACE Z (  
    (2 0 0,6 0 0,2 4 0,2 0 0)),  
    (2 0 0,2 0 4,6 0 0,2 0 0)),  
    (2 0 0,2 4 0,2 0 4,2 0 0)),  
    (6 0 0,2 0 4,2 4 0,6 0 0)))'::geometry) AS geom2  
)  
SELECT CG_3DUnion(geom1, geom2) AS union_result  
FROM solids;
```

Output

```
POLYHEDRALSURFACE Z (((2 0 0,0 0 0,2 2 0,2 0 0)),((2 0 2,0 0 0,2 0 0,2 0 2)),((0 0 4,0 4 0,0 0 0,0 0 4)),((2 0 2,0 4  
0,0 0 4,2 0 2)),((0 4 0,2 2 0,0 0 0,0 0 4)),((2 2 0,0 4 0,2 0 2,2 2 0)),((0 0 0,2 0 2,0 0 4,0 0 0)),((4 0 0,2 0  
0,2 2 0,4 0 0)),((4 0 0,2 0 2,2 0 0,4 0 0)),((4 0 0,2 2 0,6 0 0,4 0 0)),((2 0 2,4 0 0,2 0 4,2 0 2)),((2 0 2,2 4  
0,2 2 0,2 0 2)),((2 4 0,2 0 4,6 0 0,2 4 0)),((2 2 0,2 4 0,6 0 0,2 2 0)),((2 4 0,2 0 2,2 0 4,2 4 0)),((4 0 0,6 0  
0,2 0 4,4 0 0)))
```

Figure



相关信息

[CG_Extrude](#), [ST_AsX3D](#), [CG_3DIntersection](#) [CG_3DDifference](#)

8.3.12 CG_AlphaShape

CG_AlphaShape — 计算包络几何体的 Alpha 形状

Synopsis

```
geometry CG_AlphaShape(geometry geom, float alpha, boolean allow_holes = false);
```

描述

计算几何中点的 **Alpha 形状**。alpha 形状是（通常）凹多边形几何体，它包含输入的所有点，并且其点是输入点的子集。与**凸包**生成的形状相比，alpha 形状更适合输入的形状。

“合密度”由 **alpha** 参数控制，参数的值可以从 0 到无穷大。小的 **alpha** 会生成更凹的结果。大于某些数据相关的 Alpha 会生成输入的凸包。



Note

在 CGAL 之后，alpha 是 Alpha-Shape 算法中用于“入侵”输入点的 Delaunay 三角剖分的半径的平方。有关信息，参看 [CGAL Alpha 形状](#)。与 alpha 形状的原始定义不同，后者将 alpha 定义为入侵的半径。

除非将可选项的 **allow_holes** 参数指定为 **true**，否则计算的形状不包含孔。

该函数以与 **ST_ConcaveHull** 类似的方式有效地计算几何体的凹壳，但使用 CGAL 和不同的算法。

可用性：3.5.0 - 需要 SFCGAL >= 1.4.1。



该方法需要 SFCGAL 后端。

示例

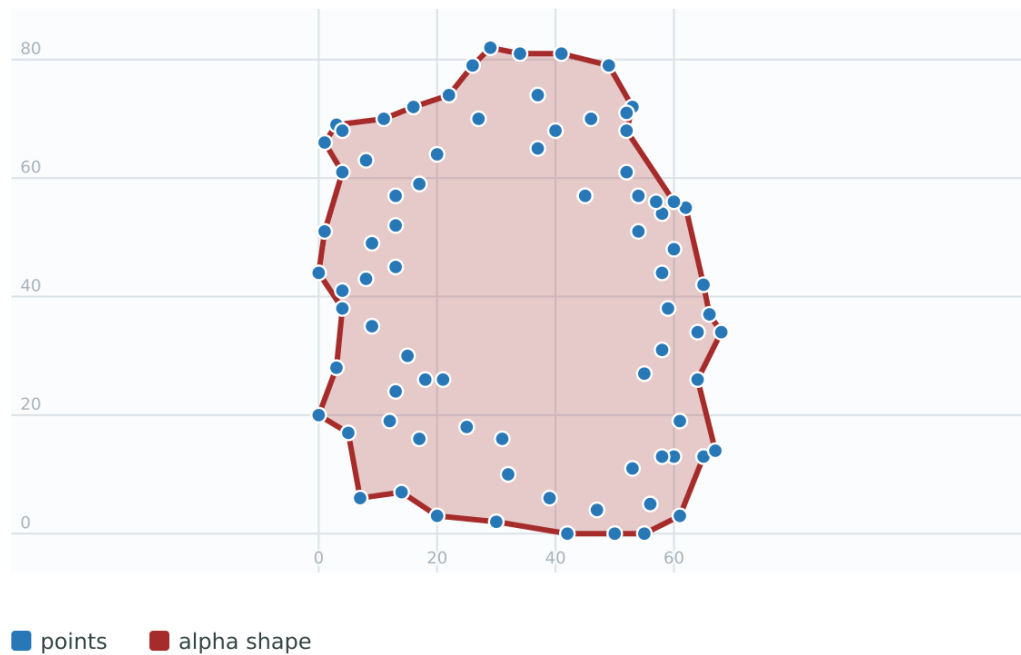
The original point cloud and the alpha-shape share one comparison frame.

Code

```
WITH data AS (
  SELECT '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 AS input_points
)
SELECT input_points AS input_points,

      CG_AlphaShape(input_points, 80.2) AS alpha_shape
FROM data;
```

Figure

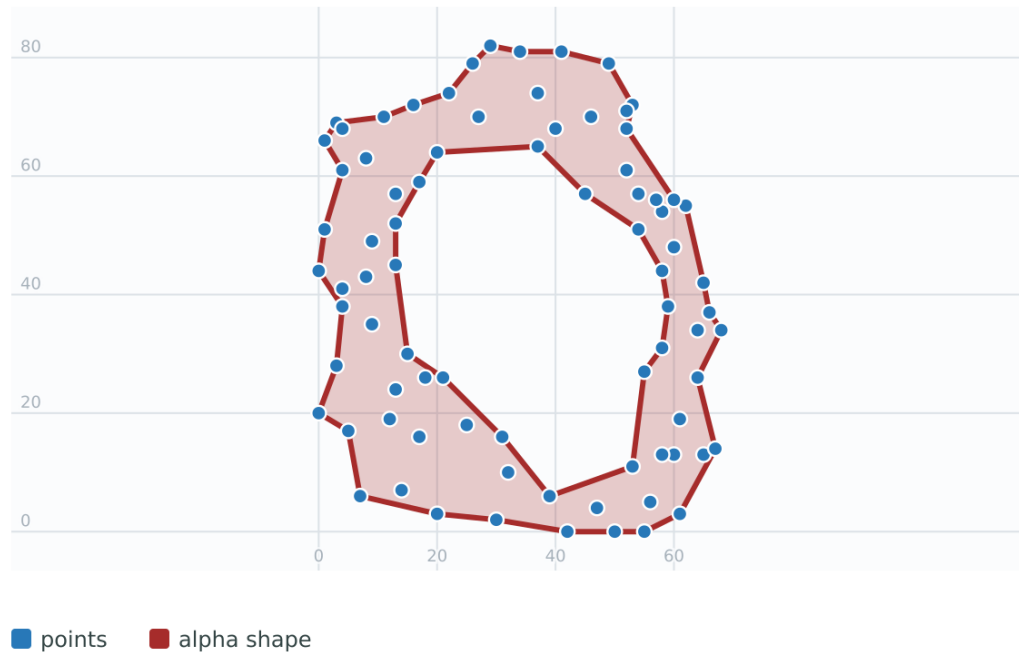


The same point cloud can produce a shape with a hole when holes are allowed.

Code

```
WITH data AS (
  SELECT '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 AS input_points
)
SELECT input_points AS input_points,
       CG_AlphaShape(input_points, 100.1, true) AS alpha_shape
FROM data;
```

Figure

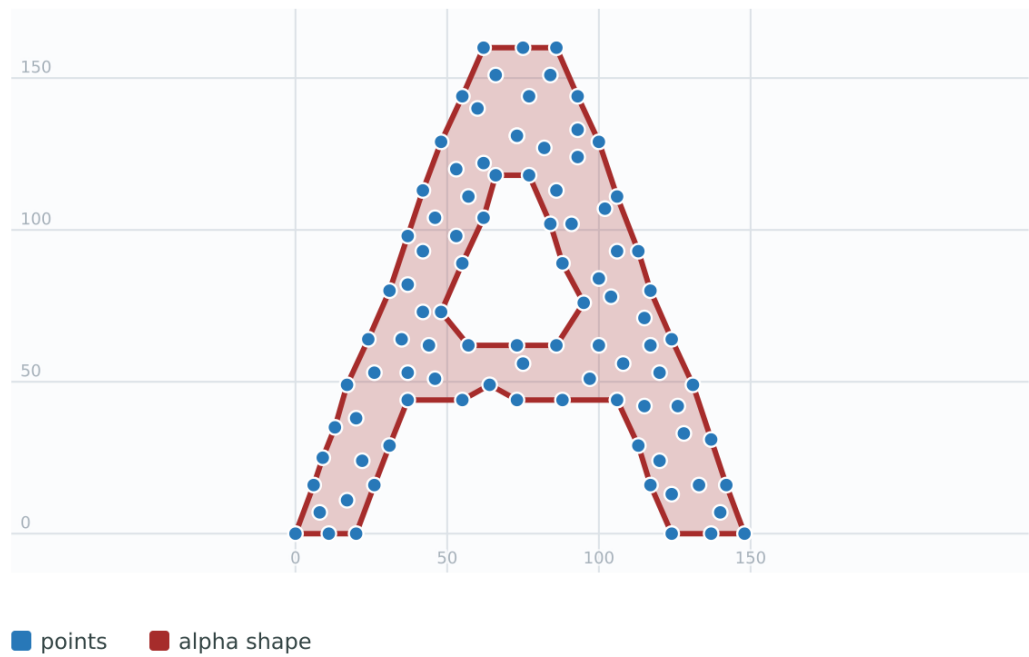


The second documented point cloud also produces a holed alpha-shape.

Code

```
WITH data AS (
  SELECT '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 AS input_points
)
SELECT input_points AS input_points,
       CG_AlphaShape(input_points, 102.2, true) AS alpha_shape
FROM data;
```

Figure



相关信息

[ST_ConcaveHull](#), [CG_OptimalAlphaShape](#)

8.3.13 CG_ApproxConvexPartition

CG_ApproxConvexPartition — 计算多边形几何图形的近似凸分割

Synopsis

geometry **CG_ApproxConvexPartition**(geometry geom);

描述

计算多边形几何图形的近似凸分割（使用三角剖分）。



Note
多边形 P 的一个划分是指一组多边形，这些多边形的内部不相交，并且这些多边形的并集等于原始多边形 P 的内部。CG_ApproxConvexPartition 和 CG_GreeneApproxConvexPartition 函数生成近似最佳的凸分割。这两个函数通常首先将多边形分解成更简单的多边形来生成凸分解；CG_ApproxConvexPartition 使用三角剖分，而 CG_GreeneApproxConvexPartition 使用一个不同的分割。这两个函数都保证它生成的凸片段数不会超过最佳数量的四倍，但它在运行复杂性上有所不同。尽管基于三角剖分的近似算法通常会生成少的凸片段，但并非总是如此。

可用性：3.5.0 - 需要 SFCGAL >= 1.5.0。

需要 SFCGAL 版本 >= 1.5.0

 方法需要 SFCGAL 后端。

示例

Approximate convex partition of the same polygon used by the other partition examples.

Code

```
WITH data AS (
  SELECT '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 AS input_polygon
)
SELECT input_polygon AS input_polygon,
       CG_ApproxConvexPartition(input_polygon) AS partition
FROM data;
```

Output

```
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)) | 
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)))
```

Figure



相关信息

[CG_YMonotonePartition](#), [CG_GreeneApproxConvexPartition](#), [CG_OptimalConvexPartition](#)

8.3.14 CG_ApproximateMedialAxis

CG_ApproximateMedialAxis — 计算几何区域的近似中轴。

Synopsis

```
geometry CG_ApproximateMedialAxis(geometry geom);
geometry CG_ApproximateMedialAxis(geometry geom, boolean projected);
```

描述

基于直骨架，返回输入的多边形的近似中心线。如果在支持的版本（1.2.0+）下构建，将使用 SFCGAL 特有的 API；否则，此函数将是 `CG_StraightSkeleton`（较慢的情况）函数的封装。

When projected is true, free endpoints of the medial axis are extended to reach the polygon boundary (projected medial axis). Requires SFCGAL 2.3.0+. When built against an older SFCGAL version, a notice is emitted and the non-projected result is returned instead.

可用性：3.5.0

Availability: 3.7.0 - projected parameter. Requires SFCGAL >= 2.3.0 for projected result; falls back to non-projected with a notice on older versions.



Note

此函数忽略 Z 高度。即使在 3D 几何体上使用，它也始终输出 2D 结果。

- ✓ 该方法需要 SFCGAL 后端。
- ✓ 该函数支持多面体曲面。
- ✓ 此函数支持三角形和不规则三角网面 (TIN)。

示例

A polygon and its approximate medial axis.

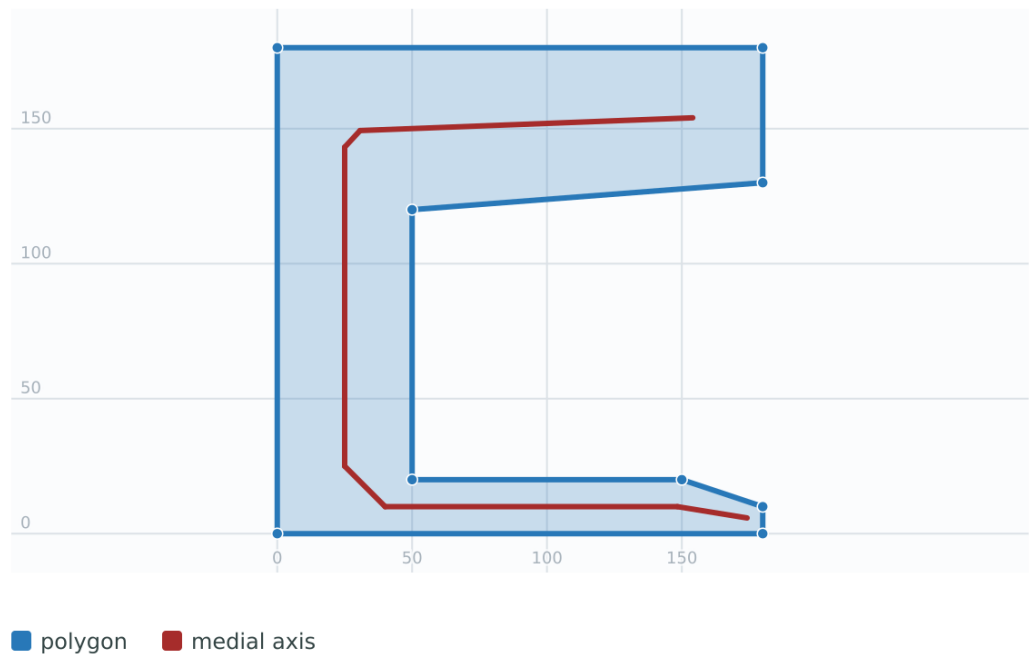
Code

```
WITH data AS (
  SELECT 'POLYGON((190 190,10 190,10 10,190 10,190 20,160 30,60 30,60 130,190 140,190 190))' AS
  '::geometry AS polygon
)
SELECT polygon AS input_polygon,
       CG_ApproximateMedialAxis(polygon) AS medial_axis
FROM data;
```

Output

```
POLYGON((190 190,10 190,10 10,190 10,190 20,160 30,60 30,60 130,190 140,190 190)) | MULTILINESTRING((184.189 15.811,158.377 20),(50 20,158.377 20),(50 20,35 35),(35 153.151,35 35),(35 153.151,40.697 159.303),(164.04 164.04,40.697 159.303))
```

Figure



This example computes the projected medial axis with free endpoints extended to the polygon boundary.

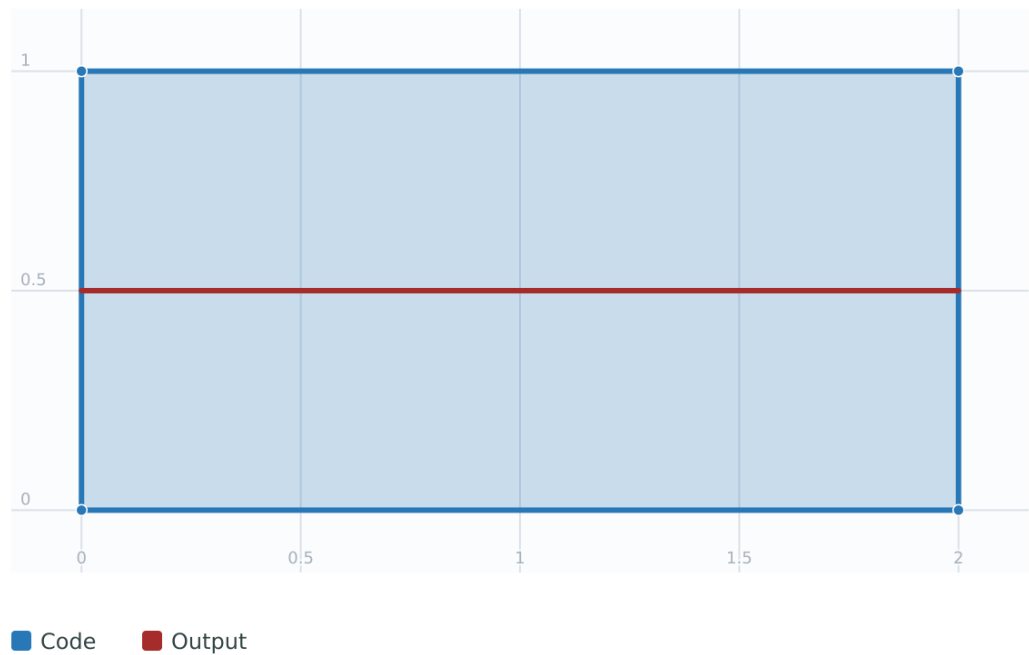
Code

```
SELECT CG_ApproximateMedialAxis('POLYGON((0 0,2 0,2 1,0 1,0 0))', true);
```

Output

```
MULTILINESTRING((0 0.5,0.5 0.5,1.5 0.5,2 0.5))
```

Figure



相关信息

[CG_StraightSkeleton](#), [CG_StraightSkeletonPartition](#)

8.3.15 CG_GenerateFlatRoof

CG_GenerateFlatRoof — Generate a flat (box) roof from a footprint polygon.

Synopsis

geometry **CG_GenerateFlatRoof**(geometry geom, float8 height);

描述

Generate a flat (box-shaped) roof as a 3D PolyhedralSurface Z from a 2D footprint polygon. The roof height above the ground plane is given by height (default 3.0).

Availability: 3.7.0 - requires SFCGAL >= 2.3.0.



☑方法需要 SFCGAL 后端。



☑函数支持多面体曲面。

示例

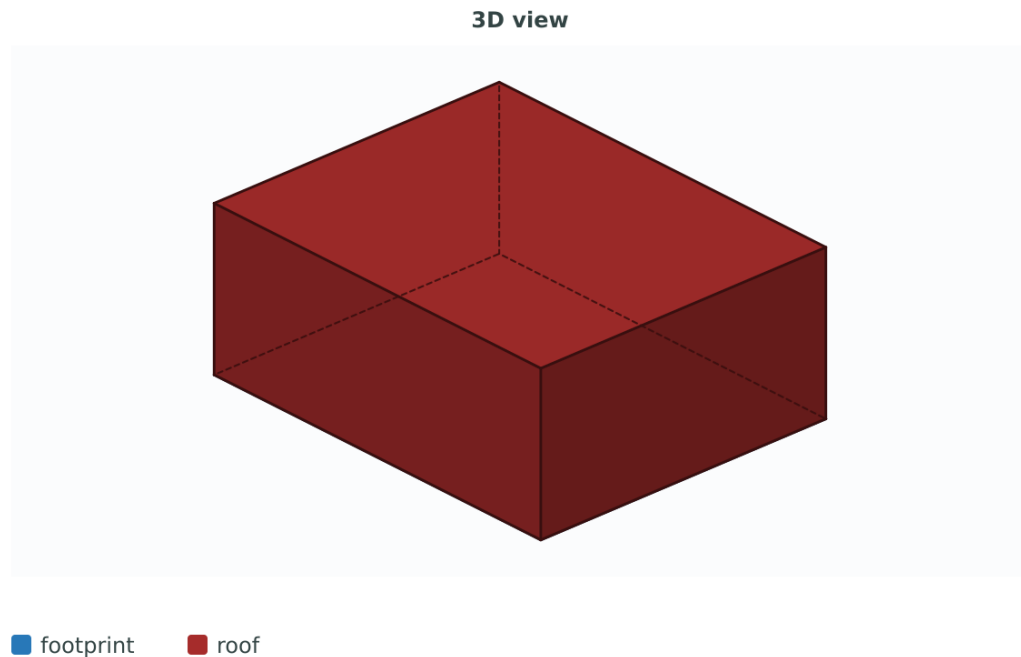
Generate a flat roof two units above a rectangular footprint.

Code

```
WITH data AS (
  SELECT 'POLYGON((0 0,5 0,5 4,0 4,0 0))'::geometry AS footprint
)
SELECT CG_GenerateFlatRoof(footprint, 2.0) AS roof
FROM data;
```

Output

```
POLYHEDRALSURFACE Z (((0 0 0,0 4 0,5 4 0,5 0 0,0 0 0)),((0 0 2,5 0 2,5 4 2,0 4 2,0 0 2)),((0 0 0,0 0 2,0 4 2,0 4 0,0 0 0) ↵
0)),((0 4 0,0 4 2,5 4 2,5 4 0,0 4 0)),((5 4 0,5 4 2,5 0 2,5 0 0,5 4 0)),((5 0 0,5 0 2,0 0 2,0 0 0,5 0 0)))
```

Figure

相关信息

[CG_GenerateHippedRoof](#), [CG_GenerateGableRoof](#), [CG_GenerateSkillionRoof](#), [CG_GenerateRoof](#)

8.3.16 CG_GenerateHippedRoof

`CG_GenerateHippedRoof` — Generate a hipped roof from a footprint polygon.

Synopsis

geometry **CG_GenerateHippedRoof**(geometry geom, float8 height);

描述

Generate a hipped roof as a 3D `PolyhedralSurface Z` from a 2D footprint polygon. All sides slope upward to meet at a ridge or apex. The roof height is given by `height` (default 3.0).

Availability: 3.7.0 - requires SFCGAL >= 2.3.0.



☑方法需要 SFCGAL 后端。



☑函数支持多面体曲面。

示例

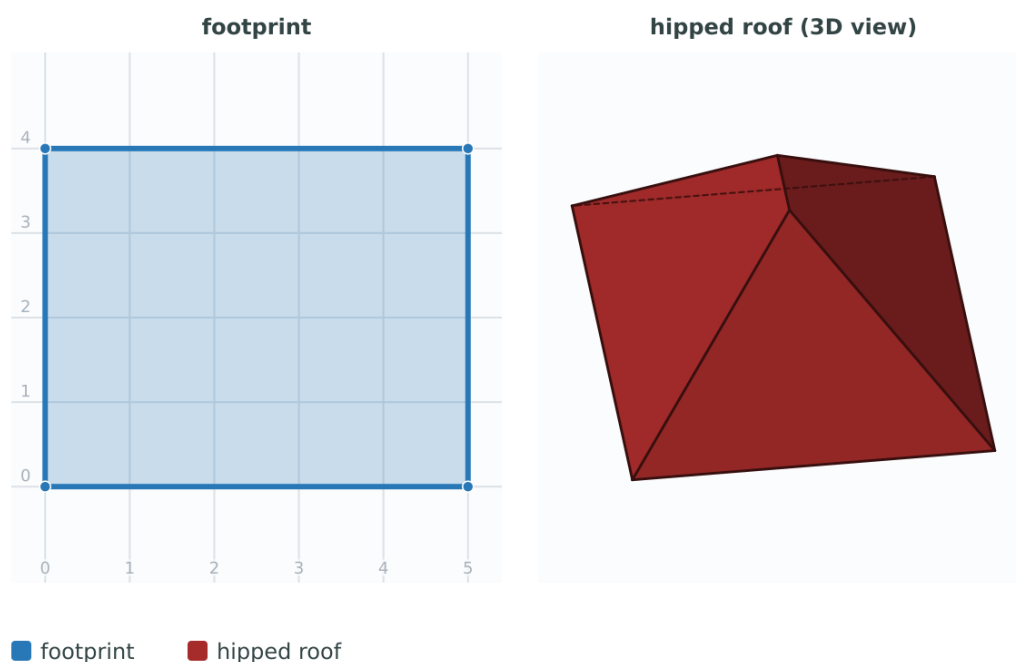
The footprint and hipped roof are shown side-by-side so the generated 3D faces do not hide behind the input footprint.

Code

```
WITH data AS (
  SELECT 'POLYGON((0 0,5 0,5 4,0 4,0 0))'::geometry AS input_footprint
)
SELECT input_footprint AS input_footprint,
       CG_GenerateHippedRoof(input_footprint, 2.0) AS hipped_roof
FROM data;
```

Output

```
POLYGON((0 0,5 0,5 4,0 4,0 0)) | POLYHEDRALSURFACE Z (((0 4 0,5 4 0,5 0 0,0 0 0,0 4 0)),((0 4 0,0 0 0,2 2 2,0 4 0)) ↔
,((0 0 0,5 0 0,3 2 2,2 2 2,0 0 0)),((5 0 0,5 4 0,3 2 2,5 0 0)),((5 4 0,0 4 0,2 2 2,3 2 2,5 4 0)))
```

Figure

相关信息

[CG_GenerateFlatRoof](#), [CG_GenerateGableRoof](#), [CG_GenerateSkillionRoof](#), [CG_GenerateRoof](#)

8.3.17 CG_GenerateGableRoof

CG_GenerateGableRoof — Generate a gable roof from a footprint polygon.

Synopsis

geometry **CG_GenerateGableRoof**(geometry geom, float8 height, float8 slope_angle);

描述

Generate a gable roof as a 3D PolyhedralSurface Z from a 2D footprint polygon. Two opposite sides are vertical gable ends; the other two sides slope to a horizontal ridge. height sets the ridge height

above the ground (default 3.0) and `slope_angle` controls the pitch in degrees (default 30.0). The SRID of the input geometry is preserved.

Availability: 3.7.0 - requires SFCGAL >= 2.3.0.

☑ 方法需要 SFCGAL 后端。

☑ 函数支持多面体曲面。

示例

Generate a gable roof using a height of 2 and a 30 degree slope.

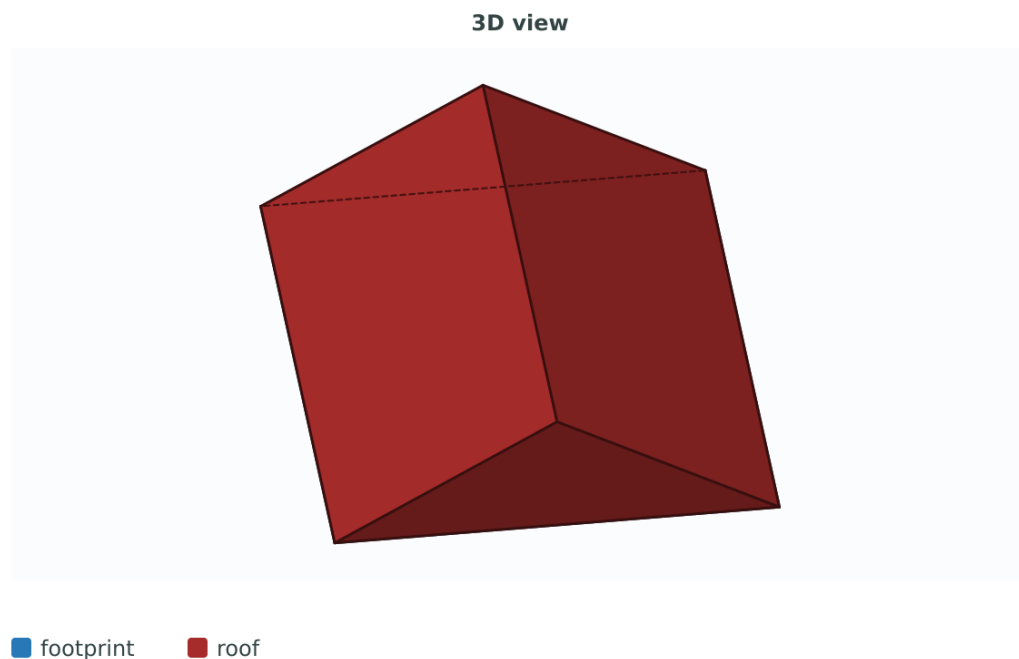
Code

```
WITH data AS (
  SELECT 'POLYGON((0 0,5 0,5 4,0 4,0 0))'::geometry AS footprint
)
SELECT CG_GenerateGableRoof(footprint, 2.0, 30.0) AS roof
FROM data;
```

Output

```
POLYHEDRALSURFACE Z (((5 4 0,5 0 0,0 0 0,0 4 0,5 4 0)),((0 4 0,0 0 0,0 2 1.155,0 4 0)),((0 0 0,5 0 0,5 2 1.155,0 2 1.155,0 0 0)),((5 0 0,5 4 0,5 2 1.155,5 0 0)),((5 4 0,0 4 0,0 2 1.155,5 2 1.155,5 4 0)))
```

Figure



相关信息

[CG_GenerateFlatRoof](#), [CG_GenerateHippedRoof](#), [CG_GenerateSkillionRoof](#), [CG_GenerateRoof](#)

8.3.18 CG_GenerateSkillionRoof

`CG_GenerateSkillionRoof` — Generate a skillion (single-slope) roof from a footprint polygon.

Synopsis

geometry **CG_GenerateSkillionRoof**(geometry geom, float8 height, float8 slope_angle, integer primary_edge_index);

描述

Generate a skillion (single-slope, shed-style) roof as a 3D PolyhedralSurface Z from a 2D footprint polygon. The entire roof surface slopes in one direction. `height` sets the maximum ridge height (default 3.0), `slope_angle` sets the pitch in degrees (default 30.0), and `primary_edge_index` selects which polygon edge defines the high side (0-based, default 0).

When the combination of `height` and `slope_angle` would cause a side wall to vanish (the slope reaches the far edge before the full height is achieved), that face is omitted from the result. This means the number of output patches can vary depending on the parameters —a lower height relative to the footprint depth produces more faces than a taller, steeper configuration.

Availability: 3.7.0 - requires SFCGAL >= 2.3.0.



☑方法需要 SFCGAL 后端。



☑函数支持多面体曲面。

示例

Height 2.0 produces 6 faces (all walls present); height 4.0 produces 5 faces (one side wall is filtered out because the slope reaches the far edge before the full height).

Code

```
SELECT
  ST_NumPatches(CG_GenerateSkillionRoof(
    'POLYGON((0 0,5 0,5 4,0 4,0 0))', 2.0)) AS "height 2",
  ST_NumPatches(CG_GenerateSkillionRoof(
    'POLYGON((0 0,5 0,5 4,0 4,0 0))', 4.0)) AS "height 4";
```

Output

6 | 5

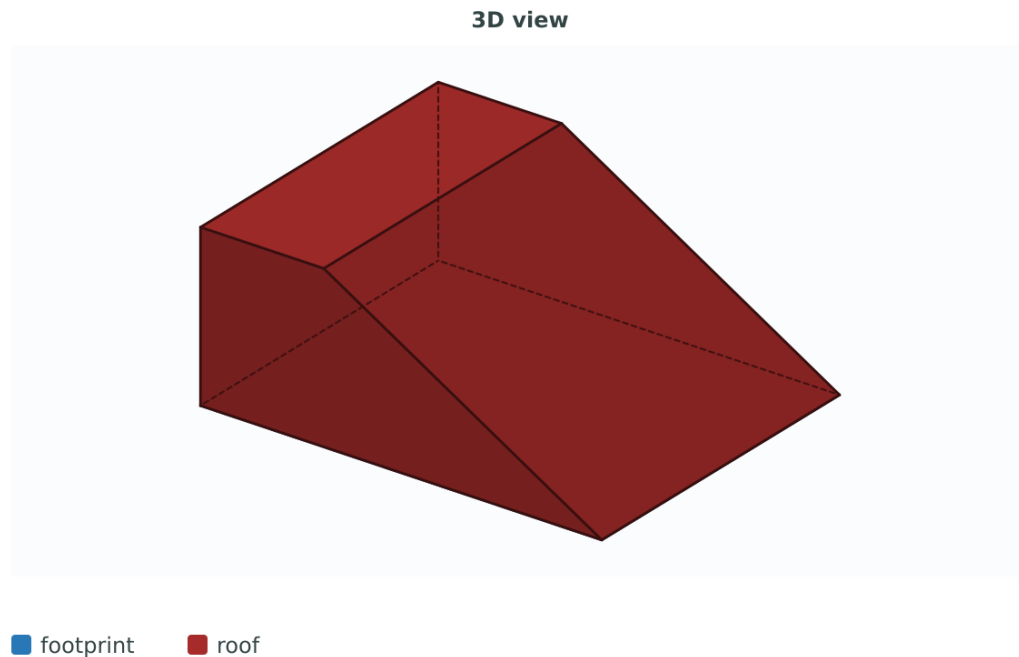
The height 2.0 result is shown with its footprint.

Code

```
WITH data AS (
  SELECT 'POLYGON((0 0,5 0,5 4,0 4,0 0))'::geometry AS footprint
)
SELECT CG_GenerateSkillionRoof(footprint, 2.0) AS roof
FROM data;
```

Output

```
POLYHEDRALSURFACE Z (((0 4 0,5 4 0,5 0 0,0 0 0,0 4 0)),((5 0 2,5 4 2,3.464 4 2,3.464 0 2,5 0 2)),((0 4 0,0 0 0,3.464 0 2,3.464 4 2,0 4 0)),((0 0 0,5 0 0,5 0 2,3.464 0 2,0 0 0)),((5 0 0,5 4 0,5 4 2,5 0 2,5 0 0)),((5 4 0,0 4 0,3.464 4 2,5 4 2,5 4 0)))
```


| Figure

相关信息

[CG_GenerateFlatRoof](#), [CG_GenerateHippedRoof](#), [CG_GenerateGableRoof](#), [CG_GenerateRoof](#)

8.3.19 CG_GenerateRoof

`CG_GenerateRoof` — Generate a roof of the requested type from a footprint polygon.

Synopsis

geometry **CG_GenerateRoof**(geometry geom, text roof_type, float8 height, float8 slope_angle, integer primary_edge_index);

描述

Generate a roof as a 3D `PolyhedralSurface` Z from a 2D footprint polygon. The `roof_type` parameter selects the roof style (default 'HIPPED'):

- `FLAT` —flat box roof.
- `HIPPED` —hipped roof (all sides slope to apex).
- `GABLE` —gable roof (two sloping sides, two vertical ends).
- `SKILLION` —single-slope (shed) roof.

`height` sets the roof height (default 3.0), `slope_angle` sets the pitch in degrees (default 30.0), and `primary_edge_index` selects the reference edge for skillion roofs (default 0).

Availability: 3.7.0 - requires SFCGAL >= 2.3.0.

- ☑ 方法需要 SFCGAL 后端。
- ☑ 函数支持多面体曲面。

示例

The generic dispatcher returns the same 3D roofs as the specialized helpers for the same footprint.

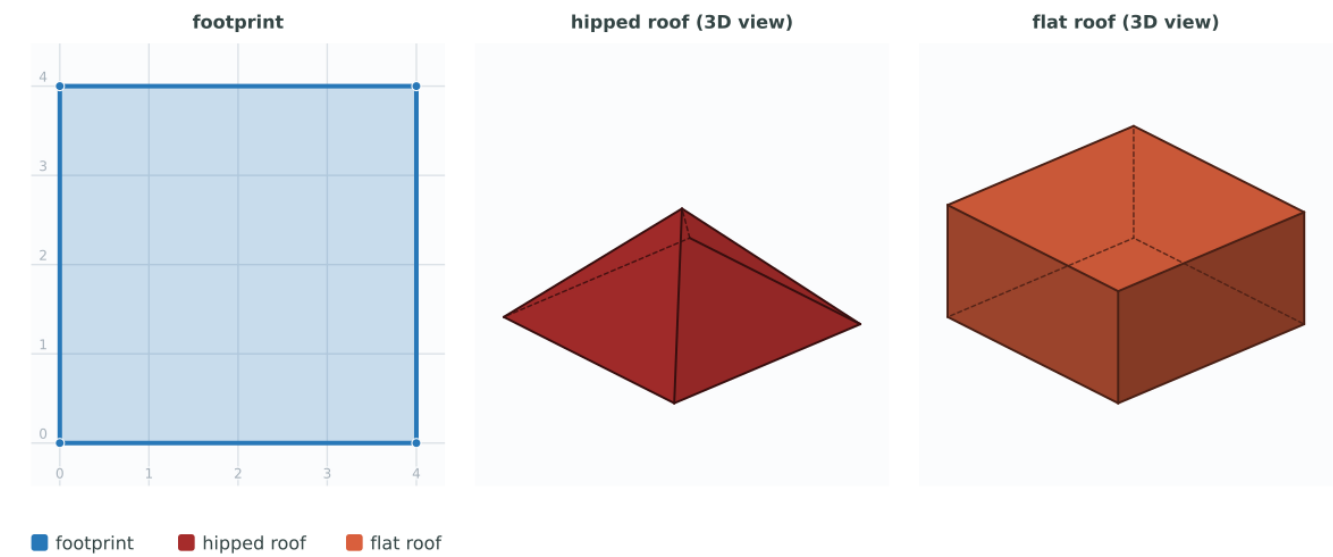
Code

```
WITH data AS (  
  SELECT 'POLYGON((0 0,4 0,4 4,0 4,0 0))'::geometry AS input_footprint  
)  
SELECT input_footprint AS input_footprint,  
       CG_GenerateRoof(input_footprint, 'HIPPED', 2.0, 30.0, 0) AS hipped_roof,  
       CG_GenerateRoof(input_footprint, 'FLAT', 2.0, 30.0, 0) AS flat_roof  
FROM data;
```

Output

```
POLYGON((0 0,4 0,4 4,0 4,0 0)) | POLYHEDRALSURFACE Z (((0 4 0,4 4 0,4 0 0,0 0 0,0 4 0)),((0 4 0,0 0 0,2 2 2,0 4 0)) ←  
  ,((0 0 0,4 0 0,2 2 2,0 0 0)),((4 0 0,4 4 0,2 2 2,4 0 0)),((4 4 0,0 4 0,2 2 2,4 4 0))) | POLYHEDRALSURFACE Z (((0  ←  
  0 0,0 4 0,4 4 0,4 0 0,0 0 0)),((0 0 2,4 0 2,4 4 2,0 4 2,0 0 2)),((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 0 2,4 0 0,4 4 0)),((4 0 0,4 0 2,0 0 2,0 0 0,4 0 0)))
```

Figure



相关信息

[CG_GenerateFlatRoof](#), [CG_GenerateHippedRoof](#), [CG_GenerateGableRoof](#), [CG_GenerateSkillionRoof](#)

8.3.20 CG_PolygonRepair

CG_PolygonRepair — Repair an invalid polygon or multipolygon.

Synopsis

geometry **CG_PolygonRepair**(geometry geom, text rule);

描述

Repair an invalid polygon or multipolygon using CGAL's 2D Polygon Repair algorithm. Returns a MultiPolygon.

The rule parameter controls the repair strategy:

- **EVEN_ODD** (default) —areas covered an odd number of times are kept. Available with CGAL 6.0+.
- **NON_ZERO** —areas with non-zero winding number are kept. Requires CGAL 6.1+.
- **UNION** —union of all input polygons. Requires CGAL 6.1+.
- **INTERSECTION** —intersection of all input polygons. Requires CGAL 6.1+.

Availability: 3.7.0 - requires SFCGAL >= 2.3.0 and CGAL >= 6.0.



☑方法需要 SFCGAL 后端。



☑函数支持多面体曲面。

示例

Repair a self-intersecting bowtie polygon with the default EVEN_ODD rule.

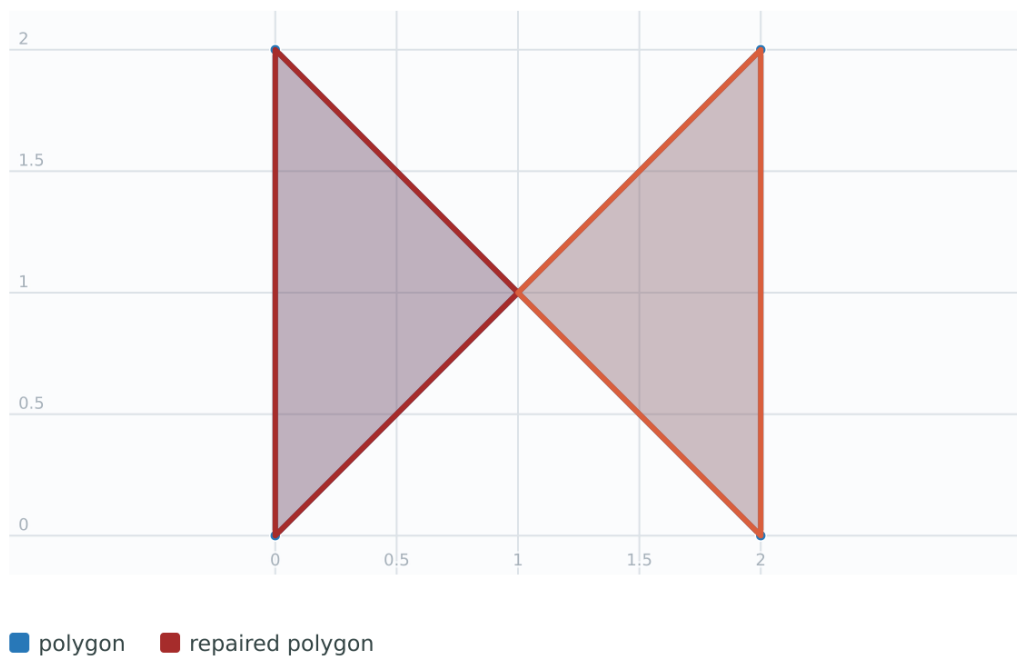
Code

```
WITH data AS (  
  SELECT 'POLYGON((0 0,2 2,2 0,0 2,0 0))'::geometry AS input_polygon  
)  
SELECT input_polygon AS input_polygon,  
       CG_PolygonRepair(input_polygon) AS repaired_polygon  
FROM data;
```

Output

```
POLYGON((0 0,2 2,2 0,0 2,0 0)) | MULTIPOLYGON(((0 0,1 1,0 2,0 0)),((1 1,2 0,2 2,1 1)))
```

Figure



With the UNION rule, all winding area is kept. This rule requires CGAL 6.1 or later.

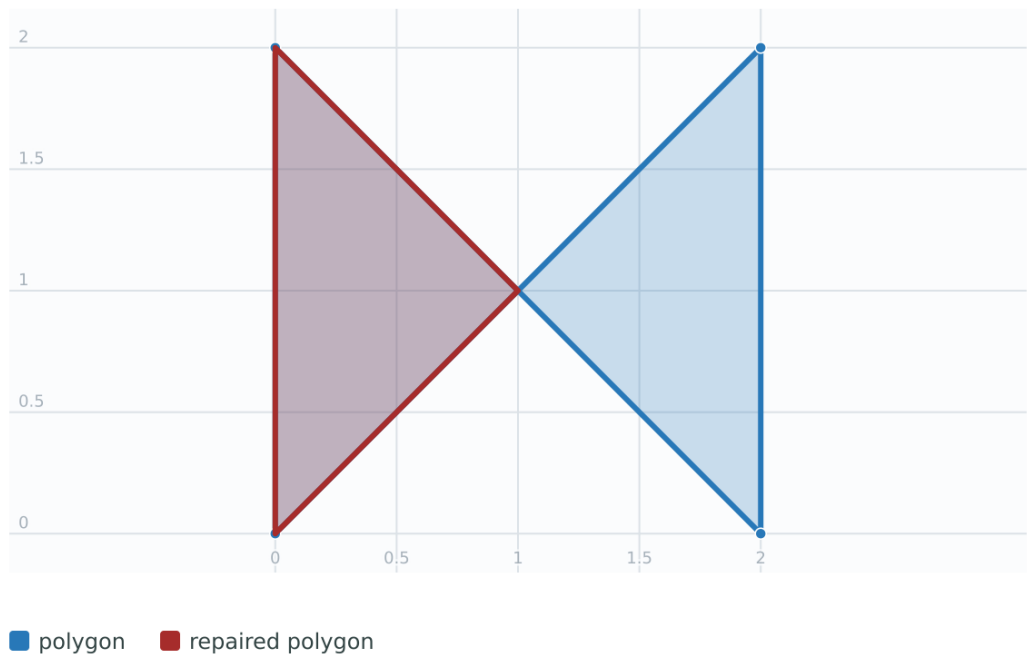
Code

```
WITH data AS (
  SELECT 'POLYGON((0 0,2 2,2 0,0 2,0 0))'::geometry AS input_polygon
)
SELECT input_polygon AS input_polygon,
       CG_PolygonRepair(input_polygon, 'UNION') AS repaired_polygon
FROM data;
```

Output

```
POLYGON((0 0,2 2,2 0,0 2,0 0)) | MULTIPOLYGON(((0 0,1 1,0 2,0 0)))
```

Figure



相关信息

[ST_IsValid](#), [ST_MakeValid](#)

8.3.21 CG_ConstrainedDelaunayTriangles

CG_ConstrainedDelaunayTriangles — 返回指定输入几何体的约束 Delaunay 三角剖分。

Synopsis

geometry **CG_ConstrainedDelaunayTriangles**(geometry g1);

描述

返回指定输入几何体点的约束 Delaunay 三角剖分。输出是 TIN。

✓ 方法需要 SFCGAL 后端。

可用性：3.0.0

✓ 函数支持 3d 并且不会丢失 z-index。

示例

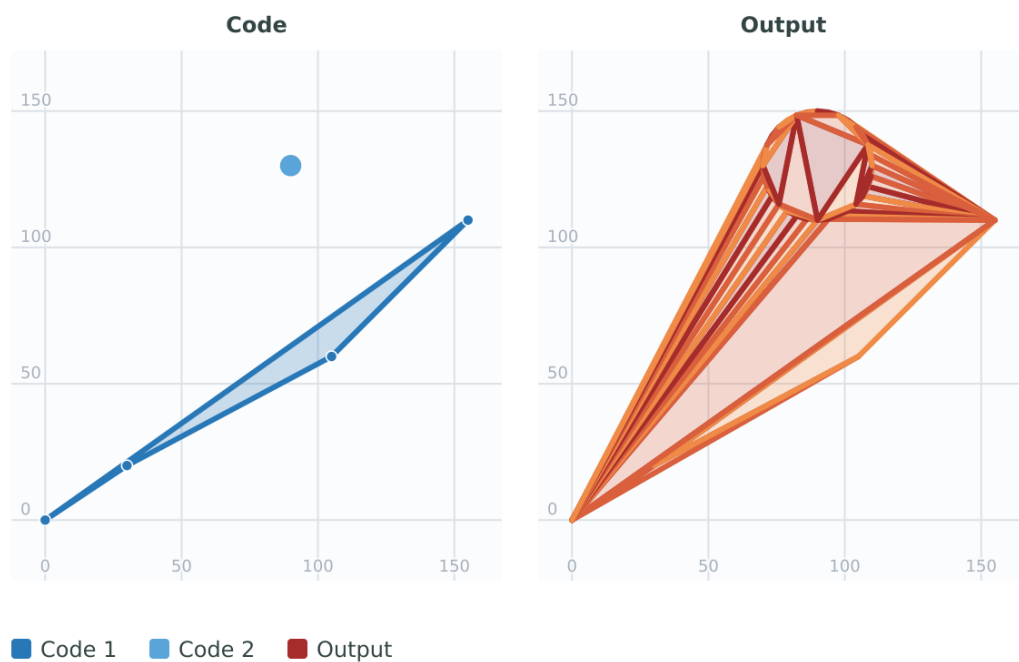
Constrained Delaunay triangulation of the polygon and buffered point.

I Code

SELECT

```
CG_ConstrainedDelaunayTriangles(
  ST_Union(
    'POLYGON((175 150,20 40,50 60,125 100,175 150))'::geometry,
    ST_Buffer('POINT(110 170)'::geometry, 20)
  )
);
```

I Figure



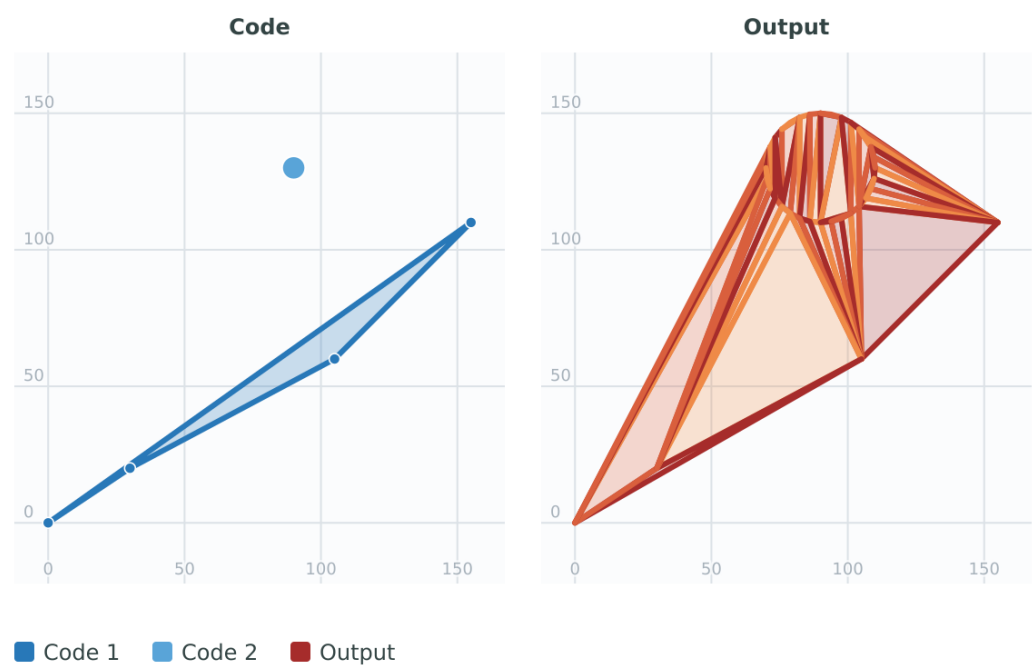
ST_DelaunayTriangles applied to the same input. Triangle edges cross polygon boundaries.

I Code

SELECT

```
ST_DelaunayTriangles(
  ST_Union(
    'POLYGON((175 150,20 40,50 60,125 100,175 150))'::geometry,
    ST_Buffer('POINT(110 170)'::geometry, 20)
  )
);
```

Figure



相关信息

[ST_DelaunayTriangles](#), [ST_TriangulatePolygon](#), [CG_Tessellate](#), [ST_ConcaveHull](#), [ST_Dump](#)

8.3.22 CG_Extrude

CG_Extrude — 将曲面☐出到相关体☐

Synopsis

geometry **CG_Extrude**(geometry geom, float x, float y, float z);

描述

Extrudes the input along the vector (x, y, z) and connects it to its translated copy. Extrusion can raise the dimensionality of the result, for example from a surface to a volume.

可用性：3.5.0

- ☑方法需要 SFCGAL 后端。
- ☑函数支持 3d 并且不会☑失 z-index。
- ☑函数支持多面体曲面。
- ☑此函数支持三角形和不☑☑三角网面 (TIN)。

示例

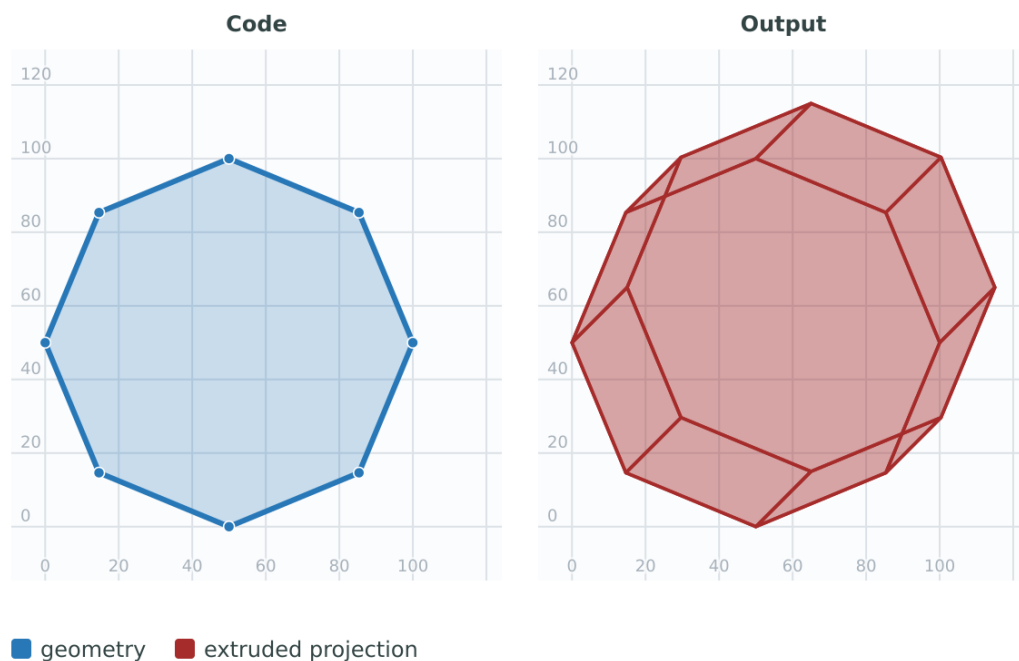
The figures use **ST_Affine** to project Z diagonally into XY after extrusion. The returned geometry from **CG_Extrude** remains a 3D PolyhedralSurface.

Extruding an octagonal Polygon 30 units along Z.

Code

```
WITH data AS (
  SELECT ST_Buffer('POINT(100 90)::geometry', 50, 'quad_segs=2') AS geom
), projected AS (
  SELECT ST_Force2D(ST_Affine(geom,
    1, 0, 0.5, 0, 1, 0.5, 0, 0, 0, 0, 0)) AS input_geom,
    ST_Force2D(ST_Affine(CG_Extrude(geom, 0, 0, 30),
    1, 0, 0.5, 0, 1, 0.5, 0, 0, 0, 0, 0)) AS output_geom
  FROM data
)
SELECT input_geom AS input_geometry,
       output_geom AS extruded_projection
FROM projected;
```

Figure



Extruding a LineString 10 units along Z.

Code

```
WITH data AS (
  SELECT 'LINESTRING(50 50,100 90,95 150)::geometry' AS geom
), projected AS (
  SELECT ST_Force2D(ST_Affine(geom,
    1, 0, 0.5, 0, 1, 0.5, 0, 0, 0, 0, 0)) AS input_geom,
    ST_Force2D(ST_Affine(CG_Extrude(geom, 0, 0, 10),
    1, 0, 0.5, 0, 1, 0.5, 0, 0, 0, 0, 0)) AS output_geom
  FROM data
)
SELECT input_geom AS input_geometry,
```

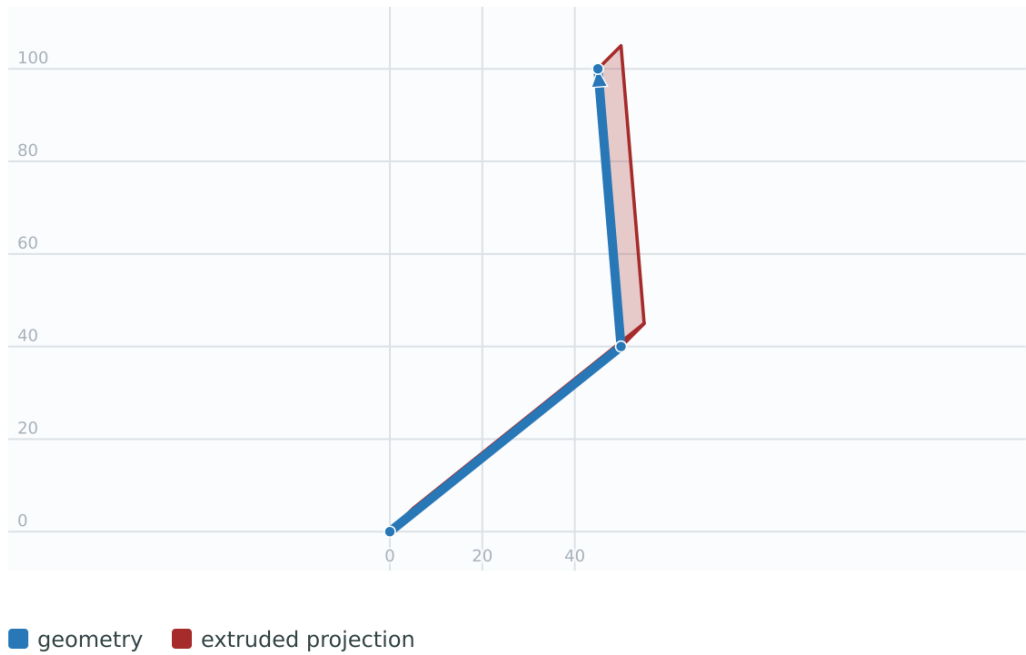


```
output_geom AS extruded_projection
FROM projected;
```

Output

```
LINestring(50 50,100 90,95 150) | POLYHEDRALSURFACE(((50 50,100 90,105 95,55 55,50 50)),((100 90,95 150,100 155,105 95,100 90)))
```

Figure



相关信息

[ST_AsX3D](#), [CG_ExtrudeStraightSkeleton](#)

8.3.23 CG_ExtrudeStraightSkeleton

CG_ExtrudeStraightSkeleton — 直^{3D}骨架外凸

Synopsis

geometry **CG_ExtrudeStraightSkeleton**(geometry geom, float roof_height, float body_height = 0);

描述

计算多边形几何图形的最大高度拉伸。

Note

直线骨架的（历史上的）第一个使用例子可能是通过固定一个多边形屋顶，使用直线骨架方法来得出屋顶的布局。骨架的每个边都被从平面上升一定的偏移距离，确保雨水是会落在屋顶的边界（即屋顶的边界），无论水从屋顶的任何地方落下，屋顶的结果的屋顶就可以是“正确的”。该函数计算一个多边形的骨架，名为“屋顶”。如果参数 `body_height` 大于 0，该多边形将通过 `CG_Extrude(polygon, 0, 0, body_height)` 来行。结果是这些多面体表面的集合。

可用性：3.5.0 - 需要 SFCGAL $\geq$ 1.5.0。

需要 SFCGAL 版本 $\geq$ 1.5.0



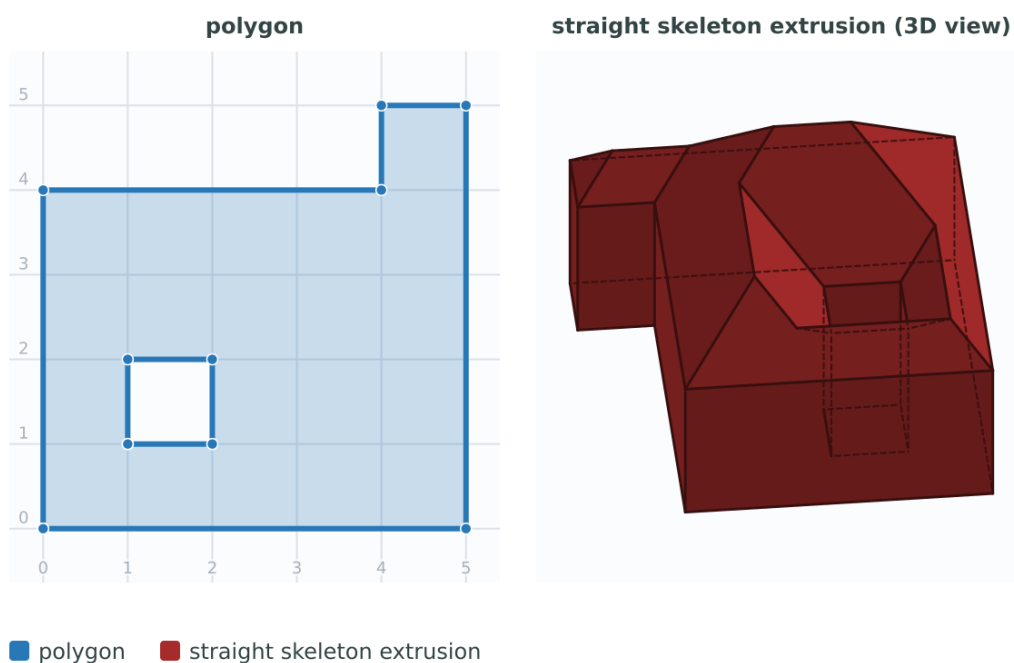
该方法需要 SFCGAL 后端。

示例

Extrude a polygon with a hole into a body and straight-skeleton roof.

Code

```
WITH data AS (
  SELECT 'POLYGON ((0 0,5 0,5 5,4 5,4 4,0 4,0 0),(1 1,1 2,2 2,1 1))'::geometry AS
    input_polygon
)
SELECT input_polygon AS input_polygon,
       CG_ExtrudeStraightSkeleton(input_polygon, 3.0, 2.0) AS straight_skeleton_extrusion
FROM data;
```

Figure

相关信息

[CG_Extrude](#), [CG_ExtrudeStraightSkeleton](#), [CG_StraightSkeleton](#), [CG_StraightSkeletonPartition](#)

8.3.24 CG_GreeneApproxConvexPartition

CG_GreeneApproxConvexPartition — 计算多边形几何图形的近似凸分割

Synopsis

```
geometry CG_GreeneApproxConvexPartition(geometry geom);
```

描述

计算多边形几何图形的近似凸分割。

Note



多边形 P 的一个划分是指一些多边形，这些多边形的内部不相交，并且这些多边形的并集等于原始多边形 P 的内部。CG_ApproxConvexPartition 和 CG_GreeneApproxConvexPartition 函数生成近似最佳的凸分割。这两个函数通常首先将多边形分解成更简单的多边形来生成凸分解；CG_ApproxConvexPartition 使用三角剖分，而 CG_GreeneApproxConvexPartition 使用一个凸分割。这两个函数都保证它生成的凸片段数不会超过最佳数量的四倍，但它在运行复杂性上有所不同。尽管基于三角剖分的近似算法通常会生成少的凸片段，但并非总是如此。

可用性：3.5.0 - 需要 SFCGAL >= 1.5.0。

需要 SFCGAL 版本 >= 1.5.0



该方法需要 SFCGAL 后端。

示例

Greene's monotone-based approximation uses the same polygon and stays directly comparable in one panel.

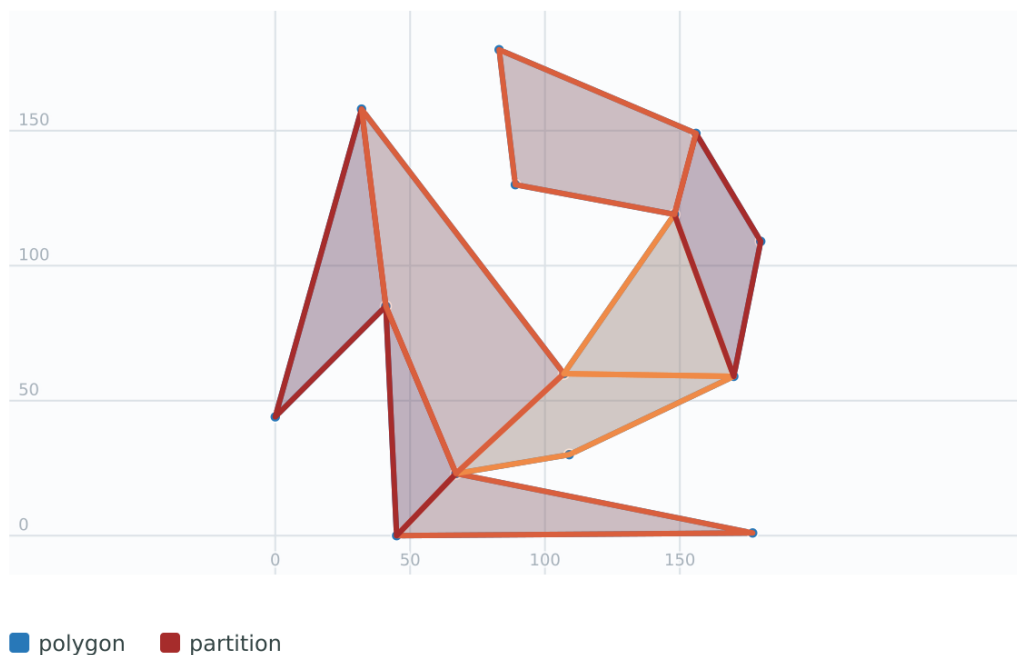
Code

```
WITH data AS (
  SELECT '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 AS input_polygon
)
SELECT input_polygon AS input_polygon,
       CG_GreeneApproxConvexPartition(input_polygon) AS partition
FROM data;
```

Output

```
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)) |
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)))
```

Figure



相关信息

[CG_YMonotonePartition](#), [CG_ApproxConvexPartition](#), [CG_OptimalConvexPartition](#)

8.3.25 CG_MinkowskiSum

CG_MinkowskiSum — 行 Minkowski sum

Synopsis

```
geometry CG_MinkowskiSum(geometry geom1, geometry geom2);
```

描述

此函数行点、或多形与多形的 2D minkowski 和。

个几何形 A 和 B 的 minkowski 和是所有点的集合，些点是 A 和 B 中任意点的和。minkowski 和常用于划和算机助。有关更多信息在[基百科 Minkowski 添加](#)。

第一个参数可以是任何 2D 几何形（点、串、多形）。如果 3D 几何形，会通制 Z 0 将其 2D，从而可能致无效的情况。第二个参数必是二多形。

利用 [CGAL 2D Minkowskium](#)。

可用性：3.5.0



方法需要 SFCGAL 后端。

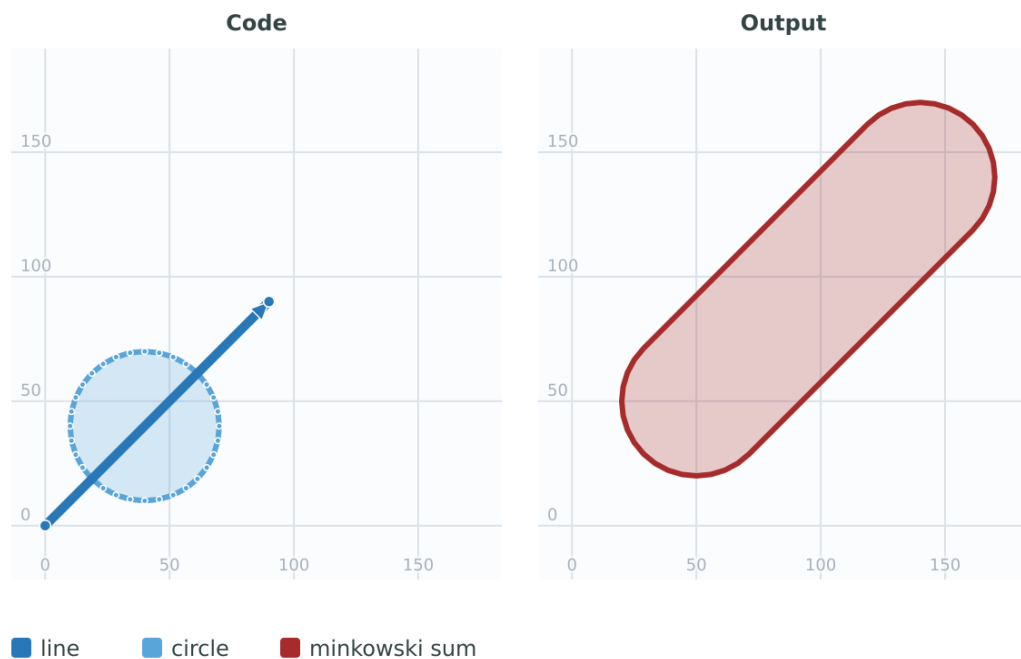
示例

Minkowski sum of a LineString and a circular polygon. The line crosses the polygon, and the result sweeps the polygon along the line.

Code

```
WITH data AS (
  SELECT ST_MakeLine(ST_Point(10, 10), ST_Point(100, 100)) AS line,
         ST_Buffer('POINT(50 50)::geometry, 30) AS circle
)
SELECT line AS input_line,
       circle AS input_circle,
       CG_MinkowskiSum(line, circle) AS minkowski_sum
FROM data;
```

Figure



Minkowski sum of a MultiPoint and a polygon. The result contains one translated copy of the polygon for each point.

Code

```
WITH data AS (
  SELECT 'MULTIPOINT(25 50,70 25)::geometry AS points,
         'POLYGON((130 150,20 40,50 60,125 100,130 150))::geometry AS polygon
)
SELECT points AS input_points,
       polygon AS input_polygon,
       CG_MinkowskiSum(points, polygon) AS minkowski_sum
FROM data;
```

Output

```
MULTIPOINT((25 50),(70 25)) | POLYGON((130 150,20 40,50 60,125 100,130 150)) | MULTIPOLYGON(((45 90,75 110,150 150,155 200,45 90)),((90 65,120 85,195 125,200 175,90 65)))
```

Figure



8.3.26 CG_OptimalAlphaShape

CG_OptimalAlphaShape — 使用“最佳”alpha 计算包围几何体的 Alpha 形状。

Synopsis

geometry **CG_OptimalAlphaShape**(geometry geom, boolean allow_holes = false, integer nb_components = 1);

描述

计算几何中点的“最佳”alpha 形状。alpha 形状是使用所给的 α 计算的，以便：

- 1. 多边形元素的数量等于或小于 nb_components（默认为 1）
- 2. 所有输入点都包含在形状中

除非将可选项的 allow_holes 参数指定为 true，否则结果将不包含孔。

可用性：3.5.0 - 需要 SFCGAL >= 1.4.1。

 该方法需要 SFCGAL 后端。

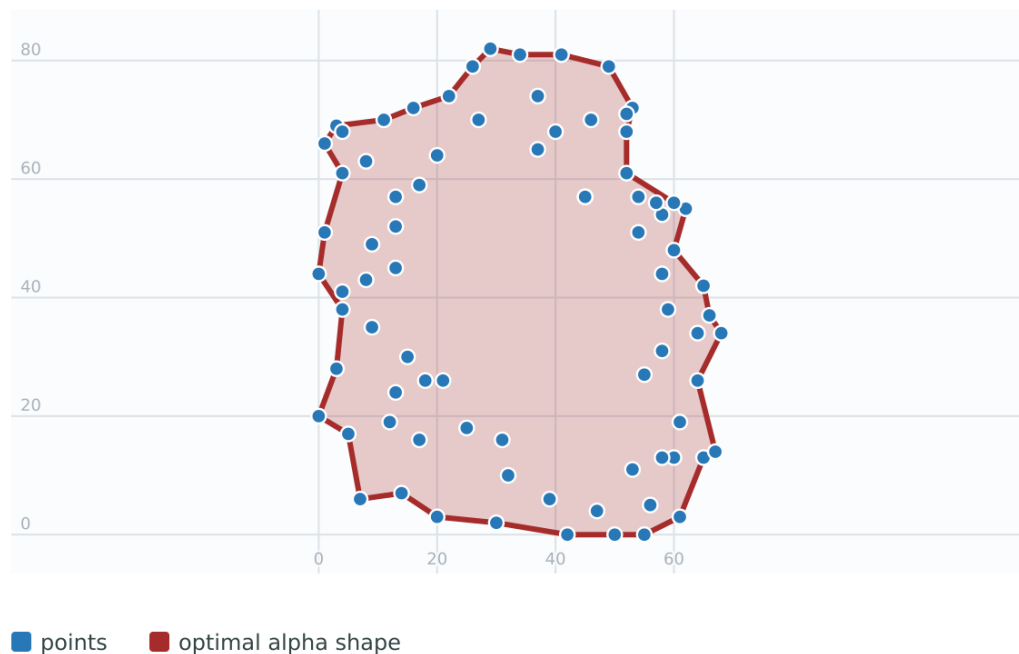
示例

The original point cloud and the optimal alpha-shape share one comparison frame.

Code

```
WITH data AS (
  SELECT '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 AS input_points
)
SELECT input_points AS input_points,
       CG_OptimalAlphaShape(input_points) AS optimal_alpha_shape
FROM data;
```

Figure



Allowing holes keeps the same outer shell but carves a central hole.

Code

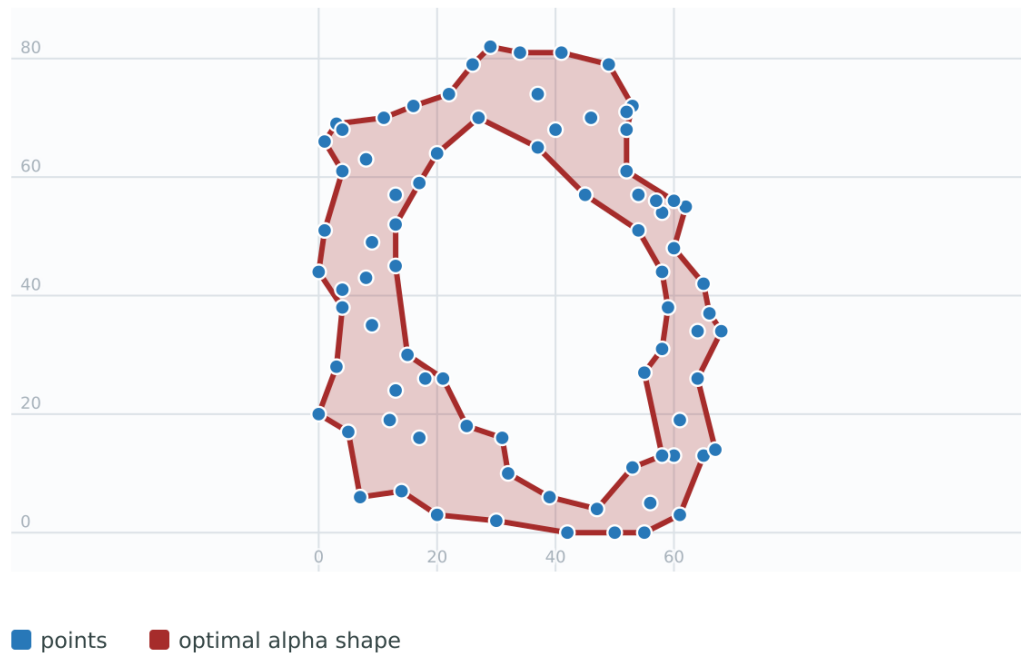
```
WITH data AS (
  SELECT 'MULTIPOINT((63 84),(76 88),(68 73),(53 18),(91 50),(81 70),(88 29),(24 82),(32 ←
51),(37 23),(27 54),(84 19),(75 87),(44 42),(77 67),(90 30),(36 61),(32 65),(81 47),(88 ←
58),(68 73),(49 95),(81 60),(87 50),
(78 16),(79 21),(30 22),(78 43),(26 85),(48 34),(35 35),(36 40),(31 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 AS input_points
```

```

)
SELECT input_points AS input_points,
        CG_OptimalAlphaShape(input_points, allow_holes => true) AS optimal_alpha_shape
FROM data;

```

Figure



相关信息

[ST_ConcaveHull](#), [CG_AlphaShape](#)

8.3.27 CG_OptimalConvexPartition

CG_OptimalConvexPartition — 计算多边形几何图形的最优凸分割

Synopsis

geometry **CG_OptimalConvexPartition**(geometry geom);

描述

计算多边形几何图形的最优凸分割。



Note

多边形 P 的一个划分是指一组多边形，这些多边形的内部不相交，并且这些多边形的并集等于原始多边形 P 的内部。CG_OptimalConvexPartition 生成的划分在片段数量上是最优的。

可用性 : 3.5.0 - 需要 SFCGAL $\geq$ 1.5.0。

需要 SFCGAL 版本 $\geq$ 1.5.0



☑方法需要 SFCGAL 后端。

示例

The optimal convex partition uses the same polygon and produces the smallest documented piece count.

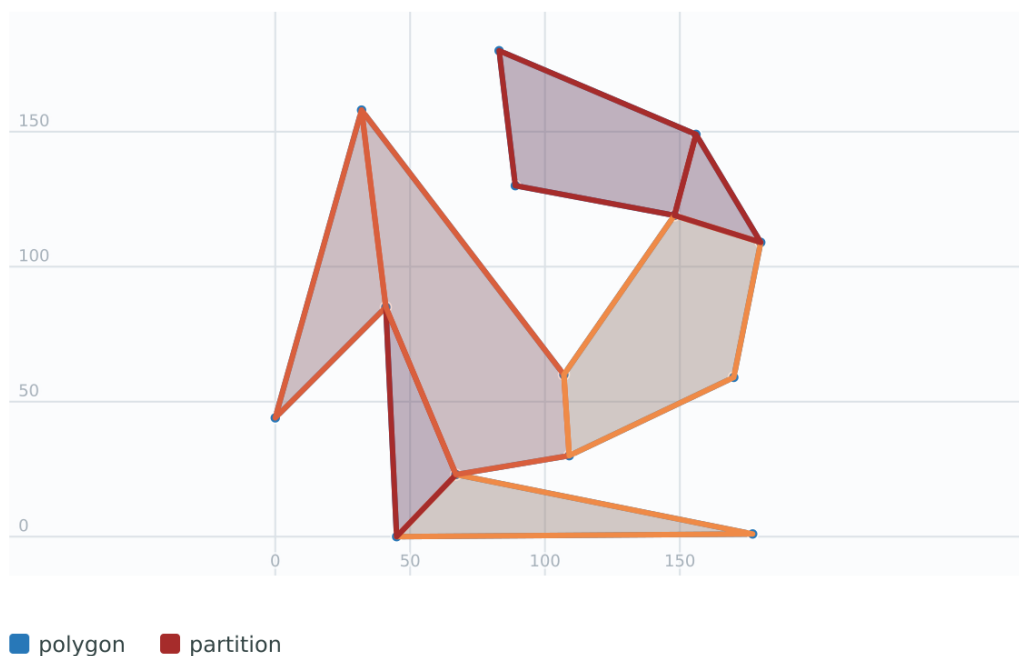
Code

```
WITH data AS (
  SELECT '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 AS input_polygon
)
SELECT input_polygon AS input_polygon,
       CG_OptimalConvexPartition(input_polygon) AS partition
FROM data;
```

Output

```
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)) | 
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)))
```

Figure



相关信息

[CG_YMonotonePartition](#), [CG_ApproxConvexPartition](#), [CG_GreeneApproxConvexPartition](#)

8.3.28 CG_StraightSkeleton

CG_StraightSkeleton — 从几何体计算直骨架

Synopsis

geometry **CG_StraightSkeleton**(geometry geom, boolean use_distance_as_m = false);

描述

Computes the straight skeleton of a polygon by tracing the inward propagation of its edges. When *use_distance_as_m* is true, the M coordinate of each output point stores its distance from the input boundary.

可用性 : 3.5.0

要使用任意参数 *use_distance_as_m*, 需要 SFCGAL $\geq$ 1.3.8



Note

此函数忽略 Z 维度。即使在 3D 几何体上使用，它也始终输出 2D 结果。



该方法需要 SFCGAL 后端。



该函数支持多面体曲面。



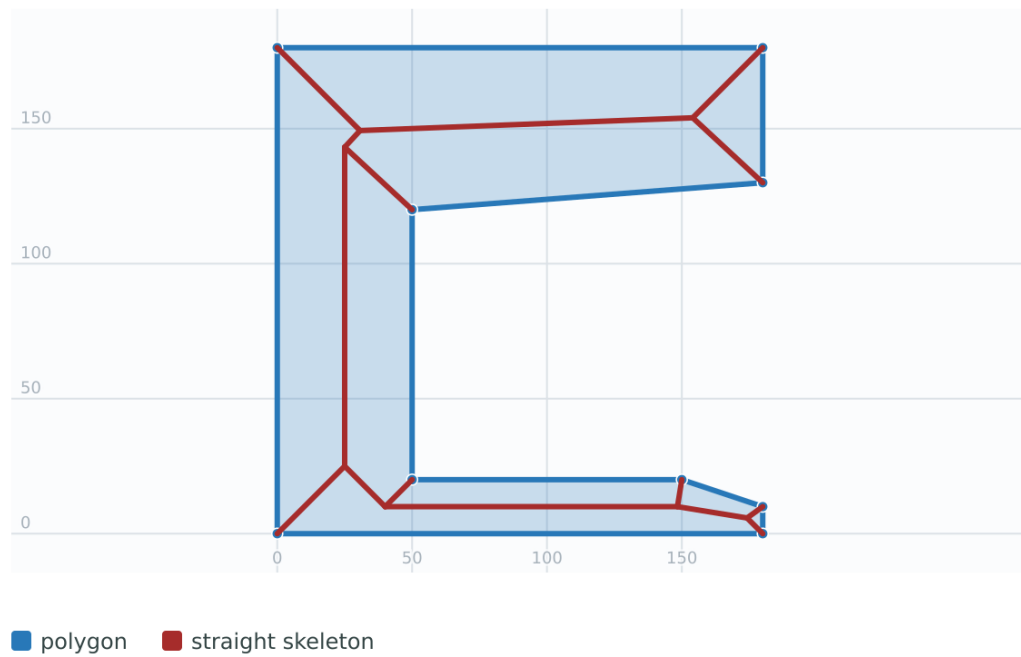
此函数支持三角形和不规则三角网面 (TIN)。

示例

Code

```
WITH data AS (  
  SELECT 'POLYGON((190 190,10 190,10 10,190 10,190 20,160 30,60 30,60 130,190 140,190 190))' AS  
    '::geometry AS polygon  
)  
SELECT CG_StraightSkeleton(polygon) AS straight_skeleton  
FROM data;
```

Figure



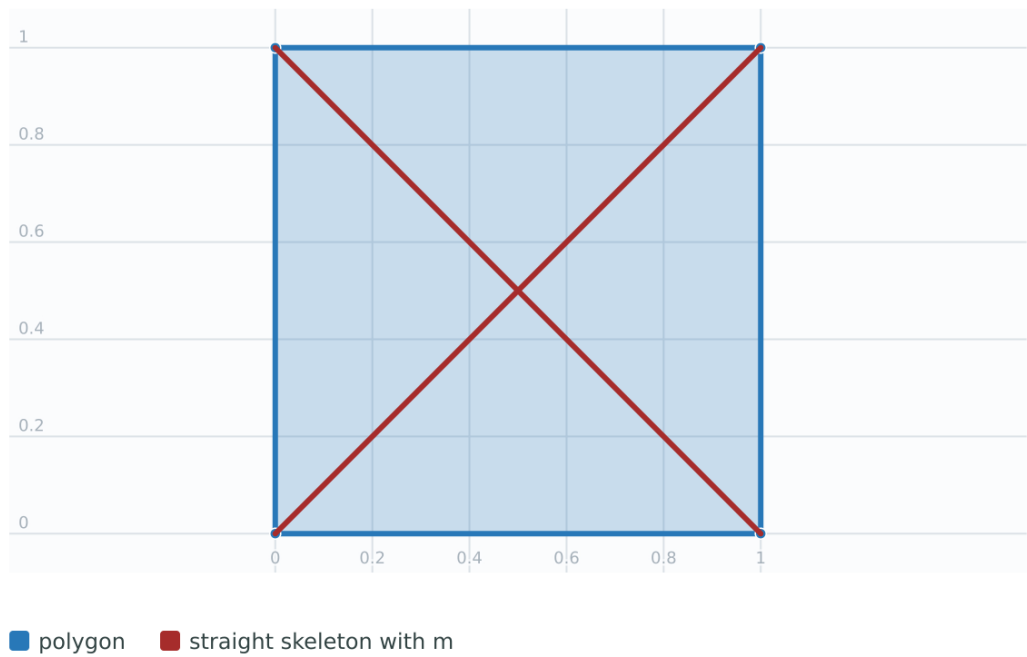
Code

```
WITH data AS (
  SELECT 'POLYGON((0 0,1 0,1 1,0 1,0 0))'::geometry AS polygon
)
SELECT CG_StraightSkeleton(polygon, true) AS straight_skeleton_with_m
FROM data;
```

Output

```
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))
```

Figure



注意，若合法输入中的点在点相接，将会引发。

Code

```
SELECT CG_StraightSkeleton('POLYGON((0 0,3 0,3 3,0 3,0 0),(0 0,1 2,2 1,0 0))');
```

Output

```
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.
```

相关信息

[CG_StraightSkeletonPartition](#), [CG_ExtrudeStraightSkeleton](#)

8.3.29 CG_Tessellate

CG_Tessellate — 多面体或多面体表面行曲面分，并以 TIN 或 TINS 集合的形式返回

Synopsis

geometry **CG_Tessellate**(geometry geom);

描述

将 MULTI(POLYGON) 或 POLYHEDRALSURFACE 等曲面作入，并通使用三角形的分程返回 TIN 表示形式。



Note
ST_TriangulatePolygon 与此函数类似，但返回一个多面形的几何集合而不是三角形网，并且适用于二维几何。

可用性 : 3.5.0

- ✓ 方法需要 SFCGAL 后端。
- ✓ 函数支持 3d 并且不会丢失 z-index。
- ✓ 函数支持多面体曲面。
- ✓ 此函数支持三角形和不规则三角网面 (TIN)。

示例

Tessellation divides each face of a cube into triangles.

Code

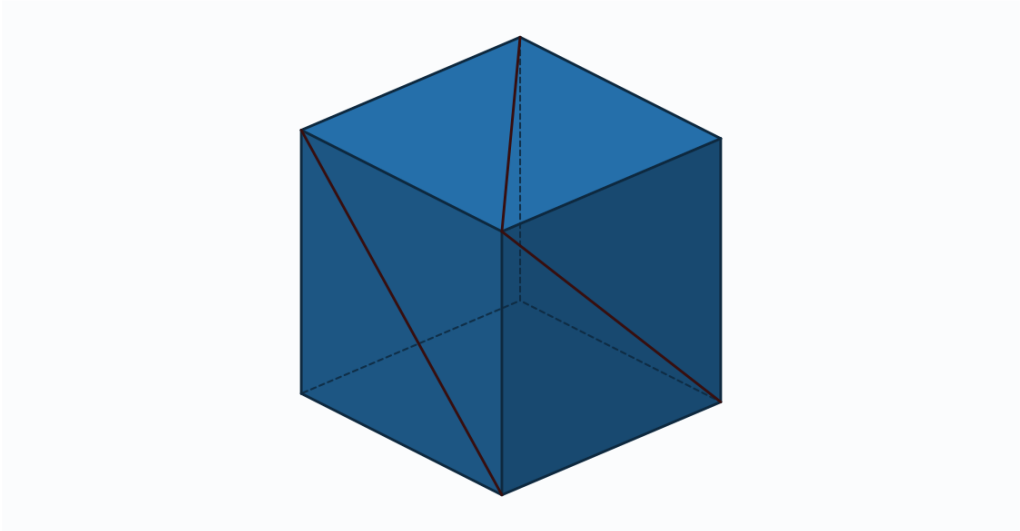
```
WITH data AS (  
  SELECT 'POLYHEDRALSURFACE Z( ((0 0 0,0 0 1,0 1 1,0 1 0,0 0 0)),  
  ((0 0 0,0 1 0,1 1 0,1 0 0,0 0 0)),((0 0 0,1 0 0,1 0 1,0 0 1,0 0 0)),  
  ((1 1 0,1 1 1,1 0 1,1 0 0,1 1 0)),  
  ((0 1 0,0 1 1,1 1 1,1 1 0,0 1 0)),((0 0 1,1 0 1,1 1 1,0 1 1,0 0 1)) )'::geometry AS cube  
)  
SELECT CG_Tessellate(cube) AS tessellation  
FROM data;
```

Output

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

Figure

3D view



cube tessellation

Tessellation of a Polygon.

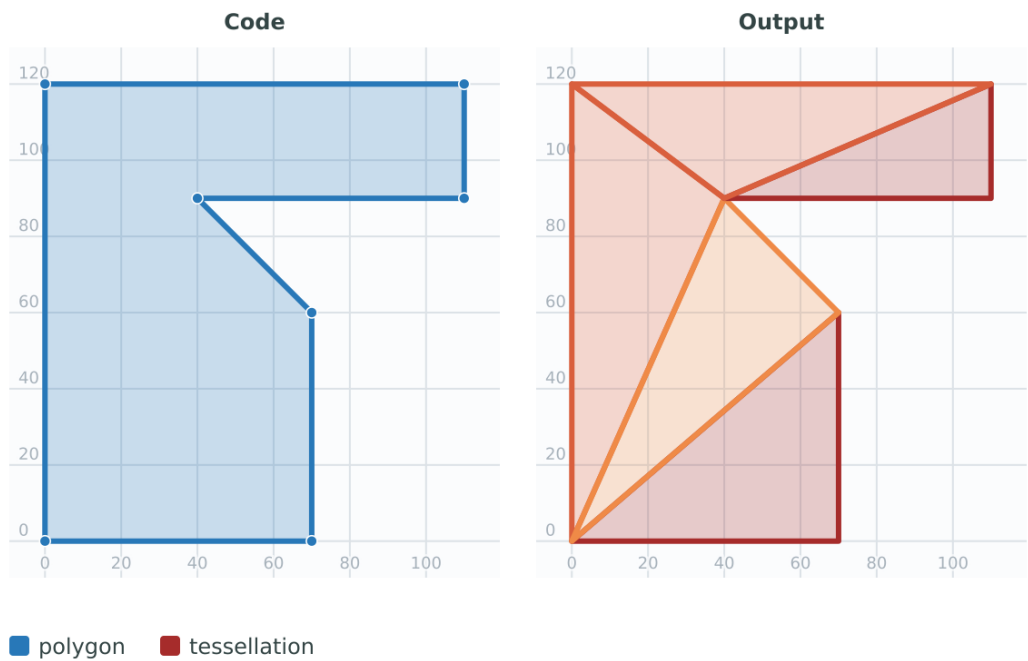
Code

```
WITH data AS (  
  SELECT 'POLYGON((10 190,10 70,80 70,80 130,50 160,120 160,120 190,10 190))'::geometry AS   
    polygon  
)  
SELECT polygon AS input_polygon,  
       CG_Tessellate(polygon) AS tessellation  
FROM data;
```

Output

```
POLYGON((10 190,10 70,80 70,80 130,50 160,120 160,120 190,10 190)) | TIN(((80 130,10 70,80 70,80 130)),((50 160,10   
190,10 70,50 160)),((80 130,50 160,10 70,80 130)),((120 160,120 190,50 160,120 160)),((120 190,10 190,50 160,120   
190)))
```

Figure



相关信息

[CG_ConstrainedDelaunayTriangles](#), [ST_DelaunayTriangles](#), [ST_TriangulatePolygon](#)

8.3.30 CG_Triangulate

CG_Triangulate — 多边形几何体行三角剖分

Synopsis

geometry **CG_Triangulate**(geometry geom);

描述

将多边形几何体进行三角剖分。
由 SFCGAL 模型行

Note!

Note
注意：此函数返回表示三角剖分结果的几何对象。

可用性：3.5.0

 该方法需要 SFCGAL 后端。

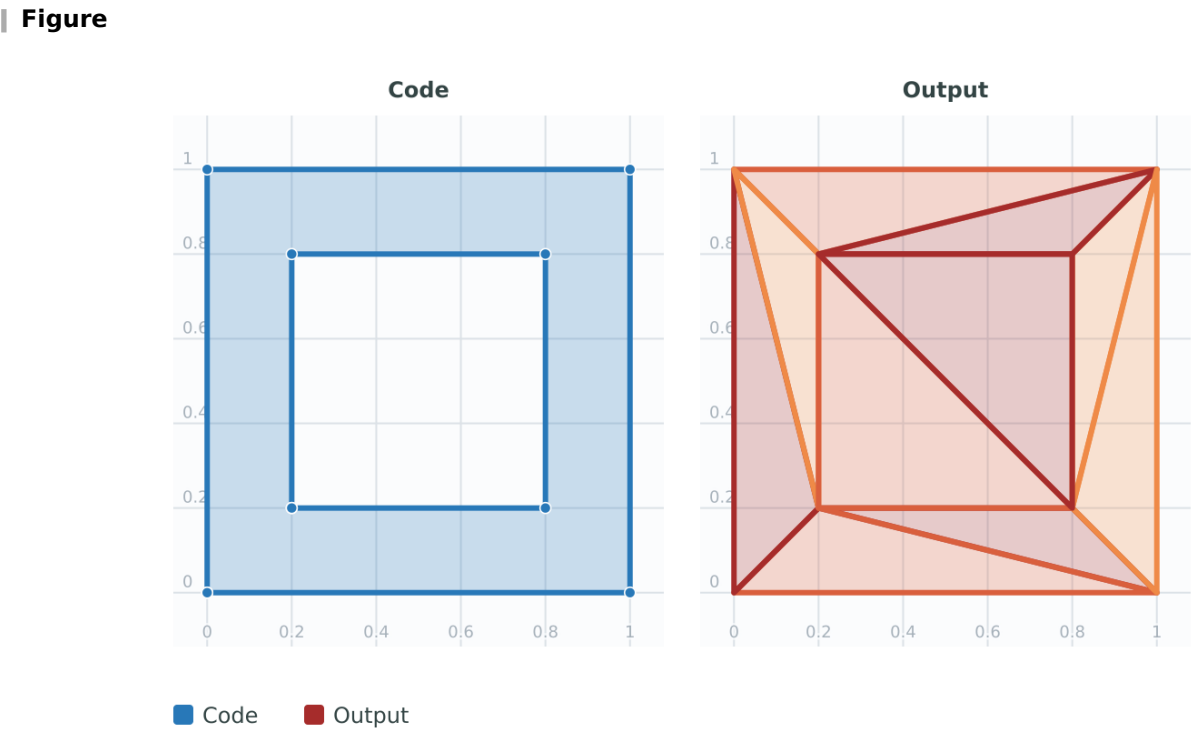
几何示例

Code

```
SELECT CG_Triangulate(
  'POLYGON((0 0,1 0,1 1,0 1,0 0),
    (0.2 0.2,0.2 0.8,0.8 0.8,0.8 0.2,0.2 0.2))'
);
```

Output

```
TIN(((0.8 0.2,0.2 0.2,1 0,0.8 0.2)),((0.2 0.2,0 0,1 0,0.2 0.2)),((1 1,0.8 0.8,0.8 0.2,1 1)),((0 1,0 0,0.2 0.2,0 1)) ↔
,((0 1,0.2 0.8,1 1,0 1)),((0 1,0.2 0.2,0.2 0.8,0 1)),((0.2 0.8,0.8 0.8,1 1,0.2 0.8)),((0.2 0.8,0.2 0.2,0.8 0.2,0.2 0.8) ↔
0.2,0.2 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)))
```



8.3.31 CG_Visibility

CG_Visibility — 计算一个从点或多边形几何中的线段生成的可见性多边形

Synopsis

```
geometry CG_Visibility(geometry polygon, geometry point);
geometry CG_Visibility(geometry polygon, geometry pointA, geometry pointB);
```

描述

Returns the portion of a polygon visible from an observer point, or from the observer segment defined by *pointA* and *pointB*.

可用性 : 3.5.0 - 需要 SFCGAL >= 1.5.0。

需要 SFCGAL 版本 >= 1.5.0

- ✔ 方法需要 SFCGAL 后端。
- ✔ 函数支持 3d 并且不会丢失 z-index。
- ✔ 函数支持多面体曲面。
- ✔ 此函数支持三角形和不规则三角网面 (TIN)。

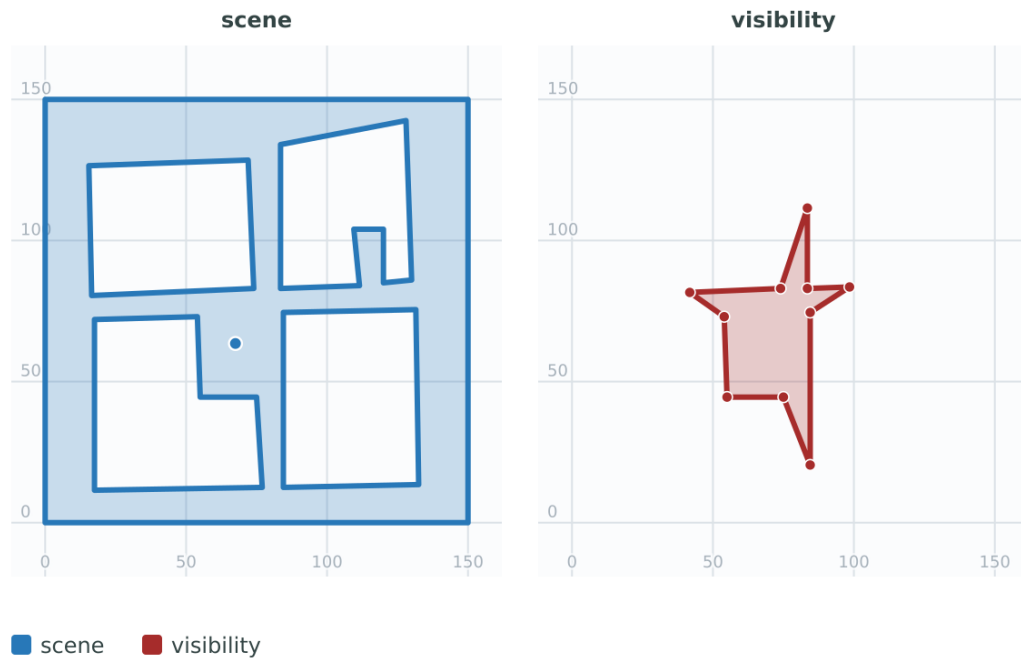
示例

Visibility from a point inside a polygon with holes.

Code

```
WITH data AS (
  SELECT '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 AS polygon,
         'POINT(91 87)'::geometry AS observer
)
SELECT ST_Collect(polygon, observer) AS input_scene,
       CG_Visibility(polygon, observer) AS visibility
FROM data;
```


Figure

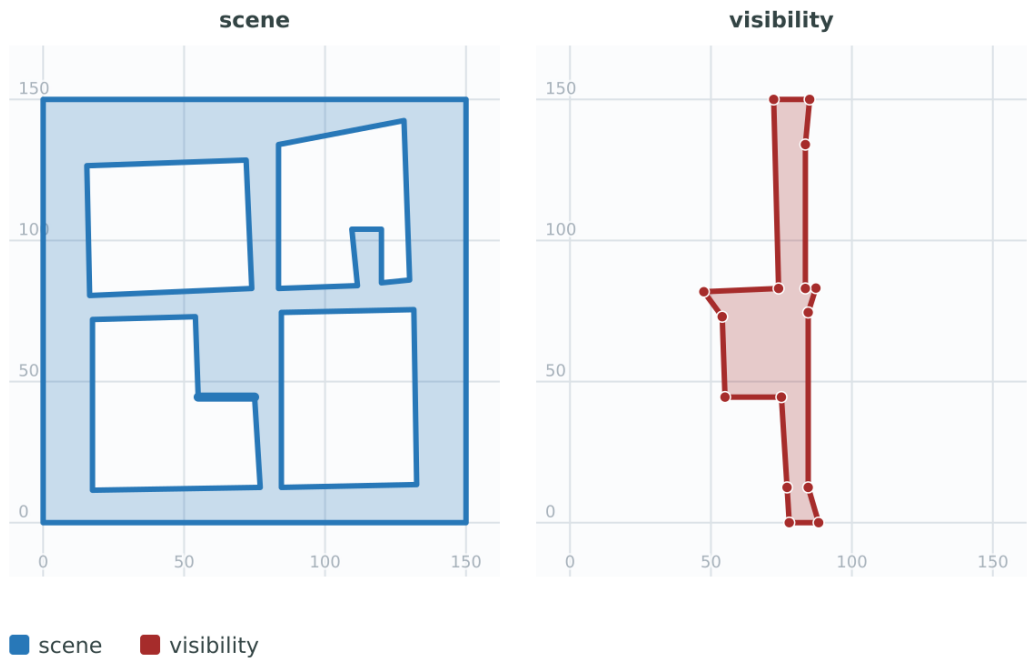


Visibility from a segment inside the same polygon.

Code

```
WITH data AS (
  SELECT 'POLYGON((23.5 23.5,23.5 173.5,173.5 173.5 23.5,23.5 23.5),(108 98,108 156 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 AS polygon,
    'LINESTRING(78.5 68,98.5 68)'::geometry AS observer
)
SELECT ST_Collect(polygon, observer) AS input_scene,
       CG_Visibility(polygon,
                     ST_StartPoint(observer),
                     ST_EndPoint(observer)) AS visibility
FROM data;
```

Figure



8.3.32 CG_YMonotonePartition

CG_YMonotonePartition — 计算多边形几何的 y 单调分割

Synopsis

geometry **CG_YMonotonePartition**(geometry geom);

描述

计算多边形几何的 y-单调分割。



Note
多边形 P 的一个划分是指一组多边形，这些多边形的内部不相交，并且这些多边形的并集等于原始多边形 P 的内部。一个 y-单调多边形是指其点 $v_1, \dots, v_n$ 可以划分成 k 个 $v_1, \dots, v_k$ 和 $v_k, \dots, v_n, v_1$ ，使得任何水平线最多与其中一条边相交一次。该算法不能保证在生成的多边形数量上达到最优数量的界限。

可用性 : 3.5.0 - 需要 SFCGAL >= 1.5.0。

需要 SFCGAL 版本 >= 1.5.0

该方法需要 SFCGAL 后端。

示例

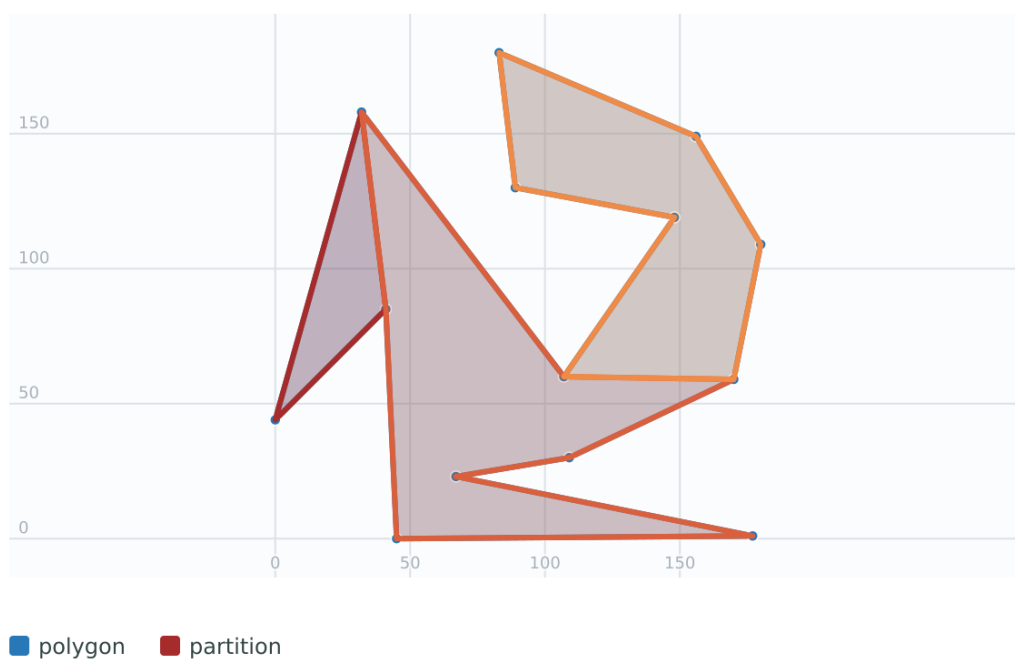
The same polygon can also be partitioned into y-monotone pieces for downstream processing.

Code

```
WITH data AS (
  SELECT '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 AS input_polygon
)
SELECT input_polygon AS input_polygon,
       CG_YMonotonePartition(input_polygon) AS partition
FROM data;
```

Output

```
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)) | 
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)))
```

Figure**相关信息**

[CG_ApproxConvexPartition](#), [CG_GreeneApproxConvexPartition](#), [CG_OptimalConvexPartition](#)

8.3.33 CG_StraightSkeletonPartition

CG_StraightSkeletonPartition — 计算多边形直骨架分割。

Synopsis

```
geometry CG_StraightSkeletonPartition(geometry geom, boolean auto_orientation);
```

描述

计算多边形几何体 *geom* 的直骨架分割。直骨架通过简化将多边形分割成若干面。如果将 *auto_orientation* 置为 *TRUE*，函数会自调整输入多边形的方向，以确保得到正确的结果。

可用性：3.6.0 - 需要 SFCGAL >= 2.0.0。

 方法需要 SFCGAL 后端。

示例

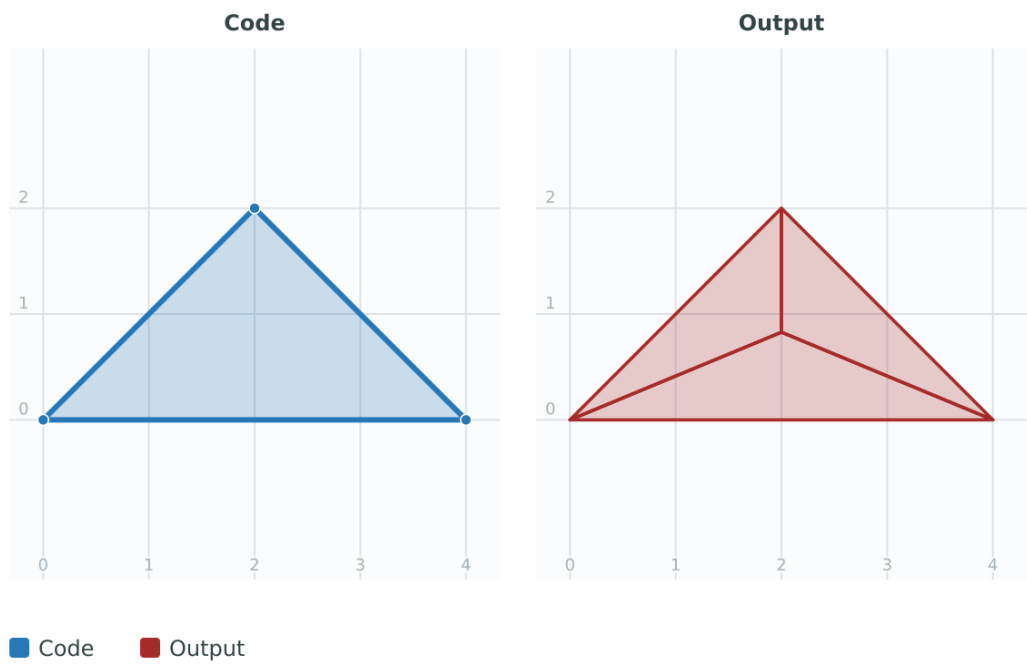
Code

```
SELECT CG_StraightSkeletonPartition('POLYGON((0 0,4 0,2 2,0 0))', true);
```

Output

```
POLYHEDRALSURFACE(((0 0,2 0.82842712474619,2 2,0 0)),((4 0,2 0.82842712474619,0 0,4 0)),((2 2,2 0.82842712474619,4 0,2 2)))
```

Figure

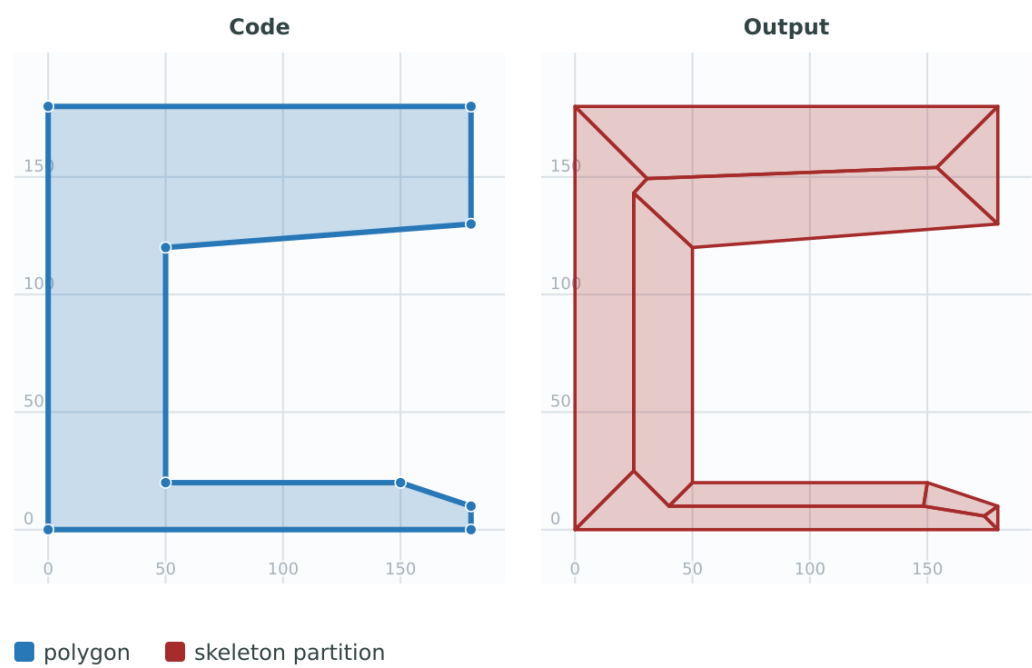


Straight skeleton partition of a polygon.

Code

```
WITH data AS (  
  SELECT 'POLYGON((190 190,10 190,10 10,190 10,190 20,160 30,60 30,60 130,190 140,190 190))' AS geometry AS polygon  
)  
SELECT polygon AS input_polygon,  
       CG_StraightSkeletonPartition(polygon, true) AS skeleton_partition  
FROM data;
```

Figure



相关信息

[CG_StraightSkeleton](#), [ST_Polygonize](#)

8.3.34 CG_3DBuffer

CG_3DBuffer — 计算几何体周围的 3D 缓冲区。

Synopsis

geometry **CG_3DBuffer**(geometry geom, float8 radius, integer segments, integer buffer_type);

描述

生成以输入几何 *geom* 为中心、半径 *radius* 的 3D 缓冲区，在 3D 空间中构建体缓冲区表示。*segments* 用于逼近曲面弧段，最少 4；*buffer_type* 用于指定缓冲类型：0 角（默认）、1 平面、2 方形

输入几何对象必须是 Point（点）或 LineString（串）型。

可用性：3.6.0 - 需要 SFCGAL >= 2.0.0

 方法需要 SFCGAL 后端。

示例

Code

```
SELECT CG_3DBuffer('POINT(0 0 0)', 1, 8, 0);
```

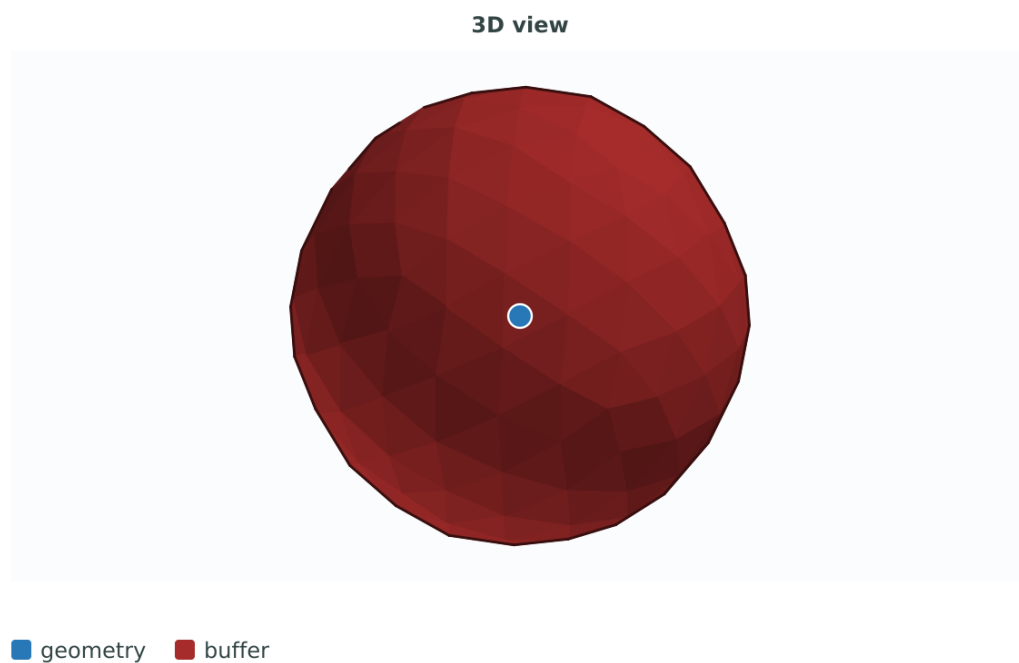
Output

```
POLYHEDRALSURFACE Z (((0 0 1, 0.5 -0.5 0.71, 0 -0.71 0.71, 0 0 1)), ... )
```

A rounded point buffer using 32 segments.

Code

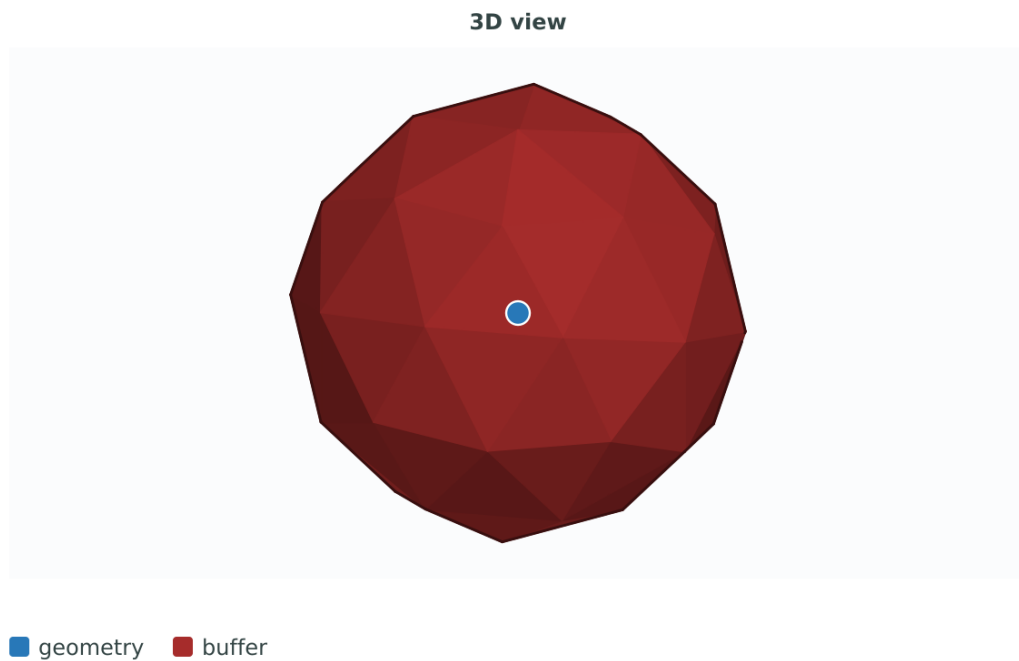
```
SELECT  
  'POINT(100 90)::geometry AS input_geometry,  
  CG_3DBuffer('POINT(100 90)::geometry, 50, 32, 0) AS buffer;
```

Figure

A coarser rounded point buffer using 5 segments.

Code

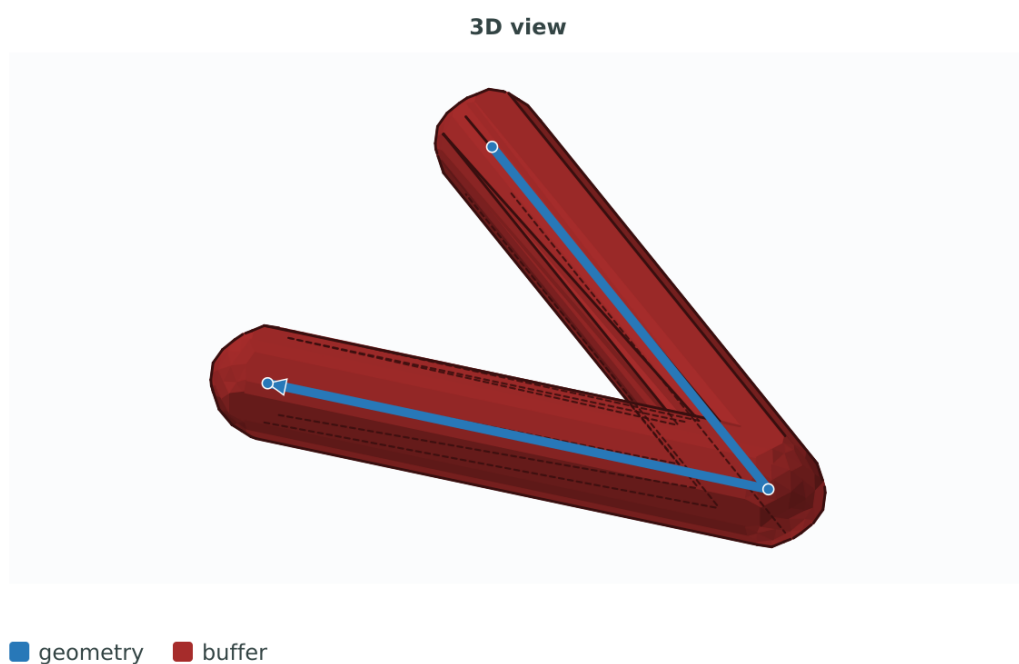
```
SELECT  
  'POINT(100 90)::geometry AS input_geometry,  
  CG_3DBuffer('POINT(100 90)::geometry, 50, 5, 0) AS buffer;
```

| Figure

A rounded buffer around a line using 32 segments.

| Code

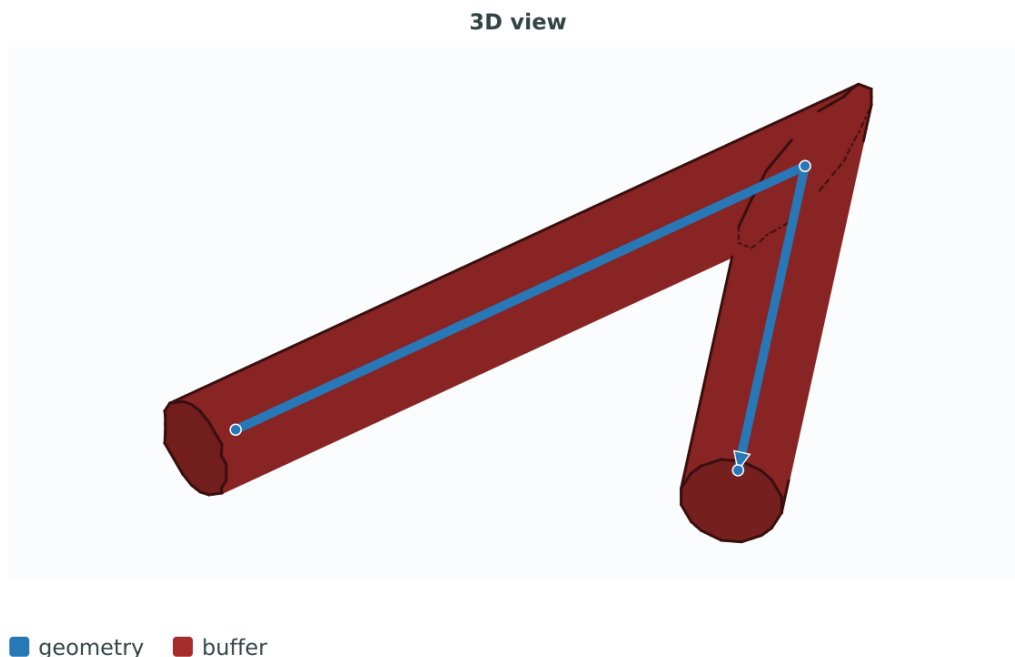
```
SELECT  
  'LINESTRING(50 50,150 150,150 50)::geometry' AS input_geometry,  
  CG_3DBuffer('LINESTRING(50 50,150 150,150 50)::geometry', 10, 32, 0) AS buffer;
```

| Figure

A square buffer around the same line using 32 segments.

Code

```
SELECT
  'LINESTRING(50 50,150 150,150 50)::geometry AS input_geometry,
  CG_3DBuffer('LINESTRING(50 50,150 150,150 50)::geometry, 10, 32, 2) AS buffer;
```

Figure

相关信息

[ST_Buffer](#), [CG_3DConvexHull](#), [ST_AsX3D](#)

8.3.35 CG_Rotate

CG_Rotate — 将几何图形 $(0,0)$ 旋转到指定的角度。

Synopsis

geometry **CG_Rotate**(geometry geom, float8 angle);

描述

将几何图形 *geom* 围绕 $(0,0)$ 中心按 *angle* 弧度旋转。旋转发生在二维平面中，Z 保持不变。正角度表示逆时针旋转。

可用性：3.6.0 - 需要 SFCGAL $\geq 2.0.0$

 该方法需要 SFCGAL 后端。

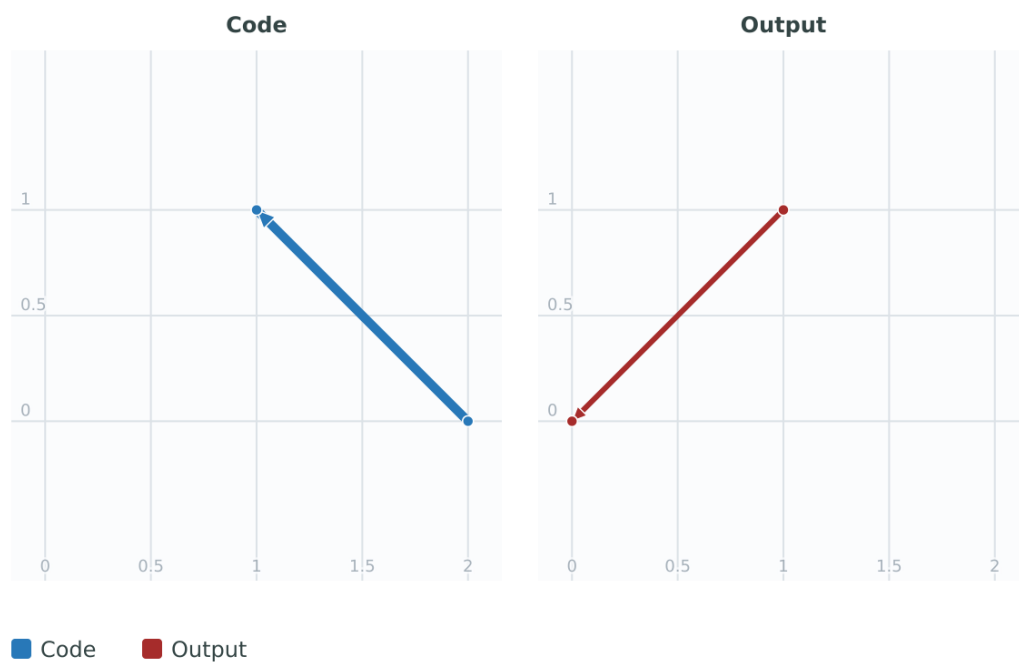
示例

Code

```
SELECT CG_Rotate('LINESTRING(1 0,0 1)', pi()/2);
```

Output

```
LINESTRING(6.123233995736766e-17 1,-1 6.123233995736766e-17)
```

Figure

相关信息

[CG_2DRotate](#), [ST_Rotate](#)

8.3.36 CG_2DRotate

CG_2DRotate — 将几何图形指定的二点按指定角度进行旋转。

Synopsis

geometry **CG_2DRotate**(geometry geom, float8 angle, float8 cx, float8 cy);

描述

将几何图形 *geom* 的点 (cx, cy) 按指定的 *angle* 弧度进行旋转。旋转在二维空间中执行，Z 值将被删除。正角度表示逆时针旋转。

可用性：3.6.0 - 需要 SFCGAL $\geq$ 2.0.0

 方法需要 SFCGAL 后端。

示例

Code

```
SELECT CG_2DRotate('POINT(1 0)', pi()/2, 1, 1);
```

Output

```
POINT(2 1)
```

Figure



相关信息

[CG_Rotate](#), [CG_3DRotate](#)

8.3.37 CG_3DRotate

`CG_3DRotate` — 将输入几何体 `geom` 会绕指定的向量进行三维旋转。旋转角度由参数 `angle`（以弧度为单位）指定，旋转将在三维空间内进行。

Synopsis

`geometry CG_3DRotate(geometry geom, float8 angle, float8 ax, float8 ay, float8 az);`

描述

将输入几何体 `geom` 从 (0,0) 到 (ax, ay, az) 的向量，在三维空间中旋转 `angle` 弧度。

可用性：3.6.0 - 需要 SFCGAL >= 2.0.0

- ✓ 该方法需要 SFCGAL 后端。
- ✓ 该函数支持 3d 并且不会丢失 z-index。

示例

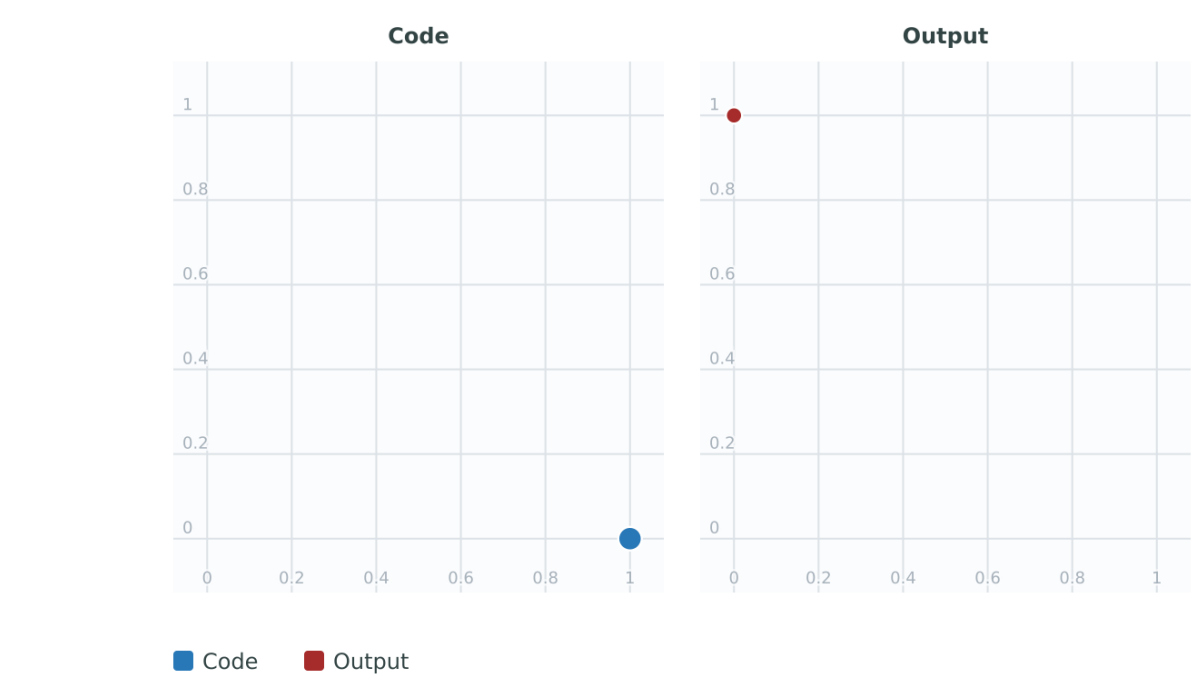
Code

```
SELECT CG_3DRotate('POINT(1 0 0)', pi()/2, 0, 0, 1);
```

Output

```
POINT Z (6.123233995736766e-17 1 0)
```

Figure



相关信息

[CG_RotateX](#), [CG_RotateY](#), [CG_RotateZ](#)

8.3.38 CG_RotateX

CG_RotateX — 将几何体 X 轴旋转指定的角度。

Synopsis

geometry **CG_RotateX**(geometry geom, float8 angle);

描述

将几何体 *geom* X 轴旋转 *angle* 弧度。

可用性：3.6.0 - 需要 SFCGAL >= 2.0.0

- ✓ 方法需要 SFCGAL 后端。
- ✓ 函数支持 3d 并且不会丢失 z-index。

示例

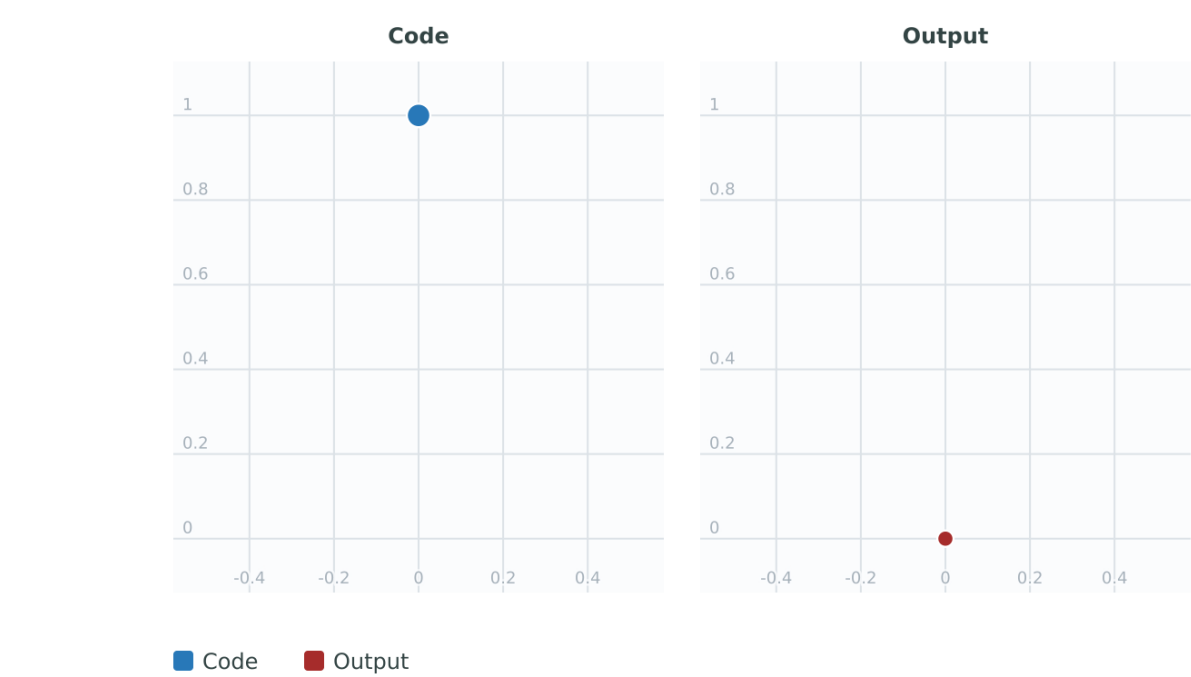
Code

```
SELECT CG_RotateX('POINT(0 1 0)', pi()/2);
```

Output

```
POINT Z (0 6.123233995736766e-17 1)
```

Figure



相关信息

[CG_RotateY](#), [CG_RotateZ](#), [CG_3DRotate](#)

8.3.39 CG_RotateY

CG_RotateY — 将几何图形 Y 旋指定的角度。

Synopsis

```
geometry CG_RotateY(geometry geom, float8 angle);
```

描述

将几何图形 *geom* 沿 Y 轴旋 *angle* 弧度。

可用性 : 3.6.0 - 需要 SFCGAL >= 2.0.0

- ✓ 方法需要 SFCGAL 后端。
- ✓ 函数支持 3d 并且不会失 z-index。

示例

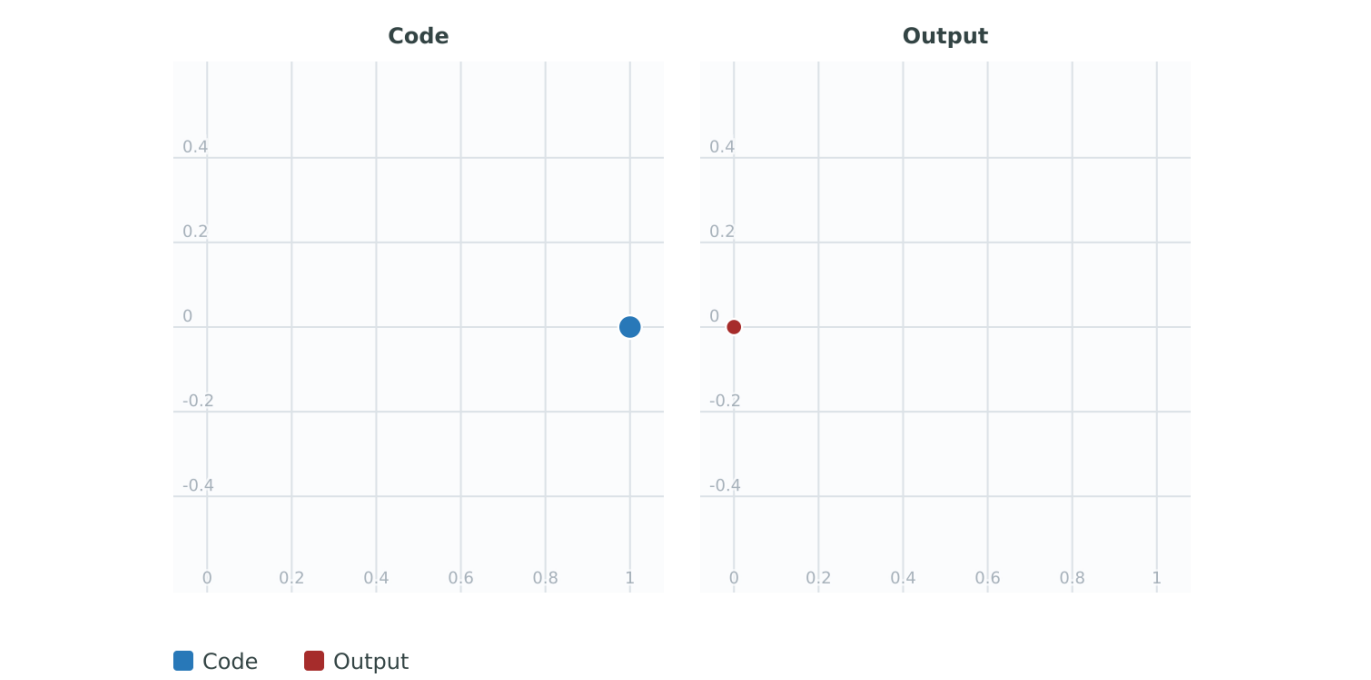
Code

```
SELECT CG_RotateY('POINT(1 0 0)', pi()/2);
```

Output

```
POINT Z (6.123233995736766e-17 0 -1)
```

Figure



相关信息

[CG_RotateX](#), [CG_RotateZ](#), [CG_3DRotate](#)

8.3.40 CG_RotateZ

CG_RotateZ — 将几何体 Z 轴旋转指定的角度。

Synopsis

geometry **CG_RotateZ**(geometry geom, float8 angle);

描述

将几何体 *geom* 绕 Z 轴旋转 *angle* 弧度。

可用性：3.6.0 - 需要 SFCGAL >= 2.0.0

- ✓ 方法需要 SFCGAL 后端。
- ✓ 函数支持 3d 并且不会丢失 z-index。

示例

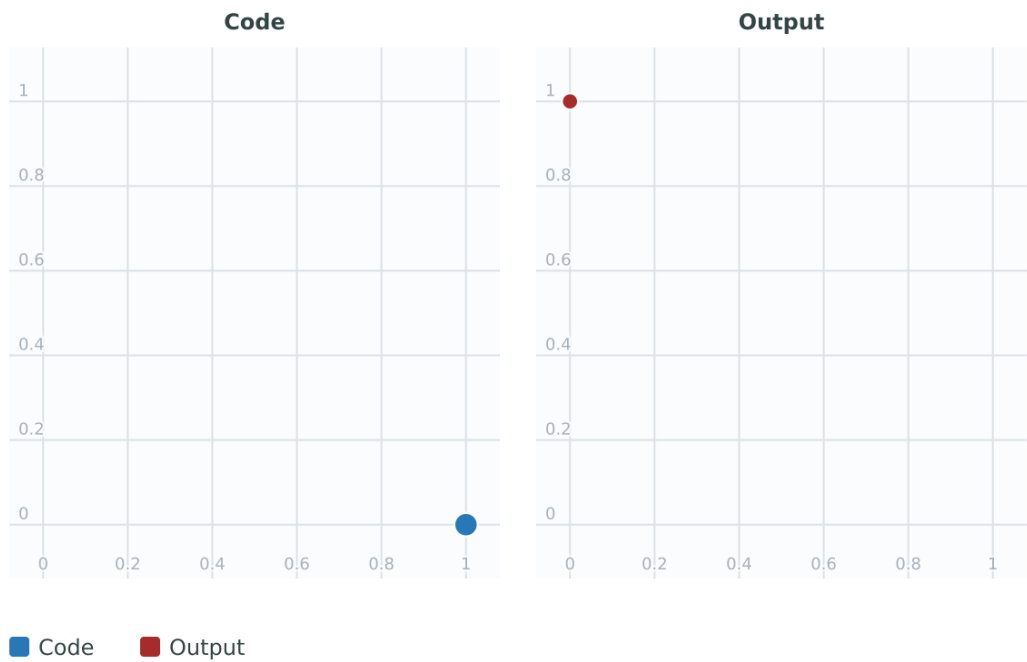
Code

```
SELECT CG_RotateZ('POINT(1 0 0)', pi()/2);
```

Output

```
POINT Z (6.123233995736766e-17 1 0)
```

Figure



相关信息

[CG_RotateX](#), [CG_RotateY](#), [CG_3DRotate](#)

8.3.41 CG_Scale

CG_Scale — 将几何体在所有维度上按给定的系数均匀缩放。

Synopsis

```
geometry CG_Scale(geometry geom, float8 factor);
```

描述

将输入几何体 *geom* 在所有维度（X、Y、Z）上按给定系数 *factor* 均匀缩放。缩放是以 (0, 0, 0) 为基准进行的。
可用性：3.6.0 - 需要 SFCGAL >= 2.0.0

该方法需要 SFCGAL 后端。

示例

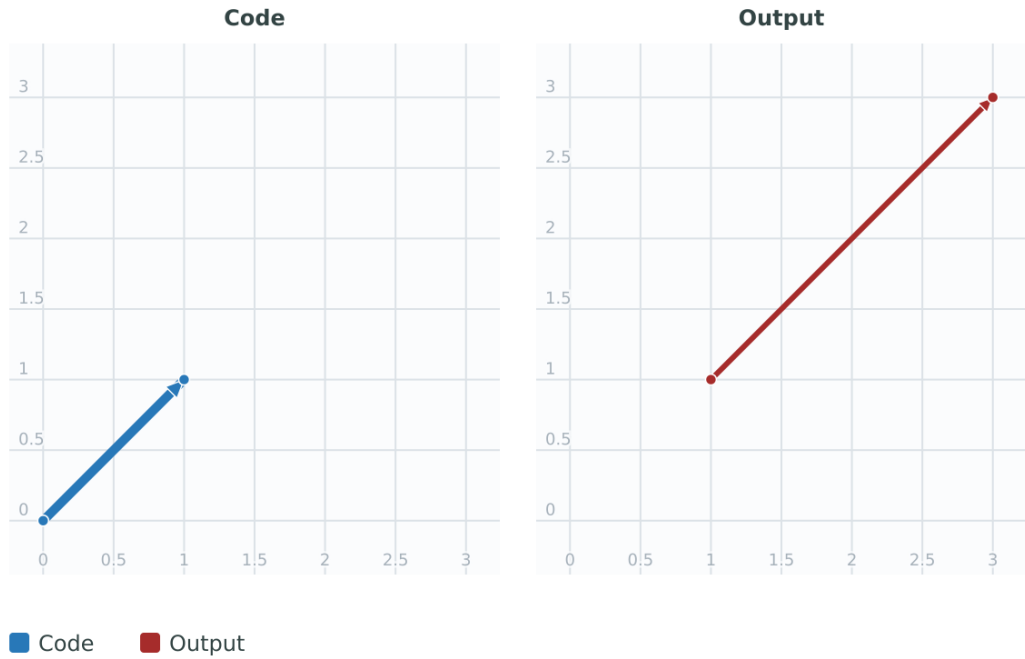
Code

```
SELECT CG_Scale('LINESTRING(1 1,2 2)', 2);
```

Output

```
LINESTRING(2 2,4 4)
```

Figure



相关信息

[CG_3DScale](#), [CG_3DScaleAroundCenter](#), [ST_Scale](#)

8.3.42 CG_3DScale

CG_3DScale — 将几何图形沿 X、Y、Z 轴分别按指定的不同系数进行缩放。

Synopsis

geometry **CG_3DScale**(geometry geom, float8 factorX, float8 factorY, float8 factorZ);

描述

将输入几何图形 *geom* 沿 X、Y、Z 轴分别按指定的系数进行缩放。缩放是相对于原点 (0,0,0) 进行的。

可用性：3.6.0 - 需要 SFCGAL >= 2.0.0

- ✓ 该方法需要 SFCGAL 后端。
- ✓ 该函数支持 3d 并且不会丢失 z-index。

示例

Code

```
SELECT CG_3DScale('POINT(1 1 1)', 2, 3, 4);
```

Output

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

Figure



相关信息

[CG_Scale](#), [CG_3DScaleAroundCenter](#)

8.3.43 CG_3DScaleAroundCenter

`CG_3DScaleAroundCenter` — 将几何体在三轴空间中，以指定的点为中心进行缩放。

Synopsis

geometry **CG_3DScaleAroundCenter**(geometry geom, float8 factorX, float8 factorY, float8 factorZ, float8 centerX, float8 centerY, float8 centerZ);

描述

将输入几何体 *geom* 以指定的点 (*centerX*, *centerY*, *centerZ*) 为中心，沿 X、Y、Z 轴分别使用指定的系数进行缩放。

可用性：3.6.0 - 需要 SFCGAL >= 2.0.0

- ☑方法需要 SFCGAL 后端。
- ☑函数支持 3d 并且不会☑失 z-index。

示例

Code

```
SELECT CG_3DScaleAroundCenter('POINT(2 2 2)', 0.5, 0.5, 0.5, 1, 1, 1);
```

Output

```
POINT Z (1.5 1.5 1.5)
```

Figure



相关信息

[CG_Scale](#), [CG_3DScale](#)

8.3.44 CG_Translate

CG_Translate — 将几何☑形在二☑空☑中按☑定的偏移量☑行平移。

Synopsis

geometry **CG_Translate**(geometry geom, float8 deltaX, float8 deltaY);

描述

将输入几何 *geom* 在 X 方向上增加 *deltaX*, 在 Y 方向上增加 *deltaY*, Z 将被忽略。

可用性 : 3.6.0 - 需要 SFCGAL >= 2.0.0

✔ 此方法需要 SFCGAL 后端。

示例

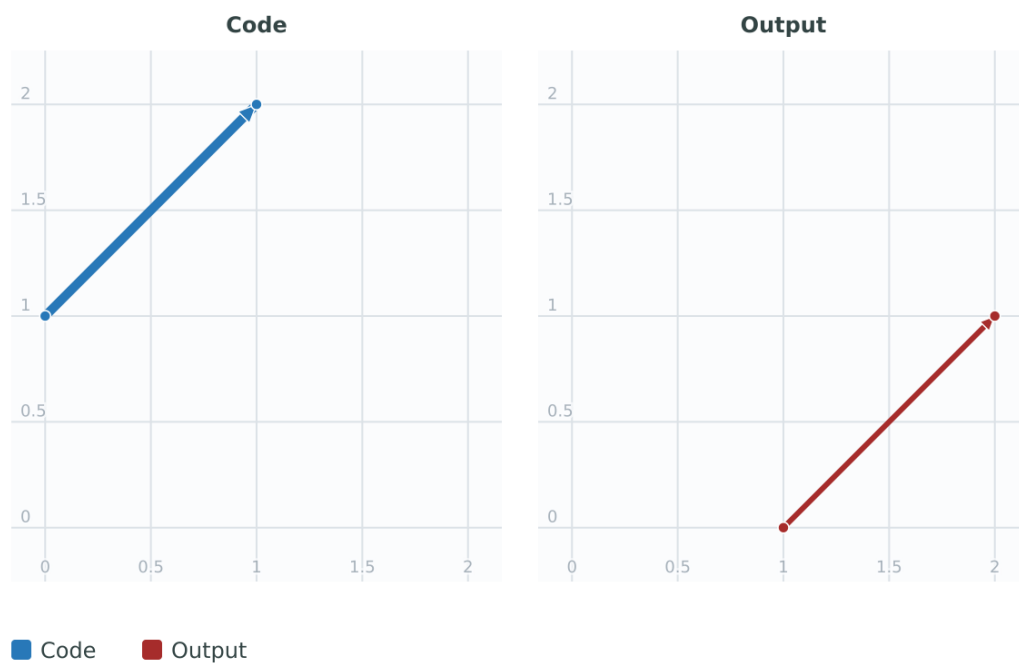
Code

```
SELECT CG_Translate('LINESTRING(1 1,2 2)', 1, -1);
```

Output

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

Figure



相关信息

[CG_3DTranslate](#), [ST_Translate](#)

8.3.45 CG_3DTranslate

CG_3DTranslate — 将输入几何按照指定的偏移量在三空间中移动。

Synopsis

geometry **CG_3DTranslate**(geometry geom, float8 deltaX, float8 deltaY, float8 deltaZ);

描述

将输入几何体 *geom* 在 X 轴上加上 *deltaX*，在 Y 轴上加上 *deltaY*，在 Z 轴上加上 *deltaZ*。

可用性：3.6.0 - 需要 SFCGAL >= 2.0.0

- ✔ 方法需要 SFCGAL 后端。
- ✔ 函数支持 3d 并且不会丢失 z-index。

示例

Code

```
SELECT CG_3DTranslate('POINT(1 1 1)', 1, -1, 2);
```

Output

```
POINT Z (2 0 3)
```

Figure



相关信息

[CG_Translate](#), [ST_Translate](#)

8.3.46 CG_Simplify

CG_Simplify — 在保留几何对象关键特征以及 Z/M 值的前提下，降低其复杂度。

Synopsis

geometry **CG_Simplify**(geometry geom, double precision threshold, boolean preserveTopology = false);

描述

使用 SFCGAL 的简化算法对几何进行简化，在保留几何关键特征的前提下减少点或点数量，并在简化过程中保留 Z 和 M 值。

该算法基于约束三角剖分，使用 **CGAL Polyline Simplification 2** 值，并额外处理以保留 Z、M 坐标。当要求保持拓扑且几何存在相交时，会在相交点插入点进行插值处理。

此函数适用于地形的 3D (2.5D) 几何，但并不适合处理如墙体等垂直表面。

可用性: 3.6.0 - 需要 SFCGAL >= 2.1.0

- ✓ 该函数需要 SFCGAL 后端。
- ✓ 该函数支持 3d 并且不会丢失 z-index。
- ✓ 该函数支持 M 坐标。

参数

geom 输入几何

threshold 最大距离值（以几何单位），用于控制简化程度。值越大，输出几何被简化得就越多。

preserveTopology 如果为 true，函数会确保几何的拓扑结构被保留；在此模式下，当几何生成相交时，相交点的 Z 和 M 值会进行插值处理。默认为 false。

返回值

返回一个保留了 Z 和 M 值的简化几何对象。

示例

This example simplifies a polygon with a threshold of 0.5.

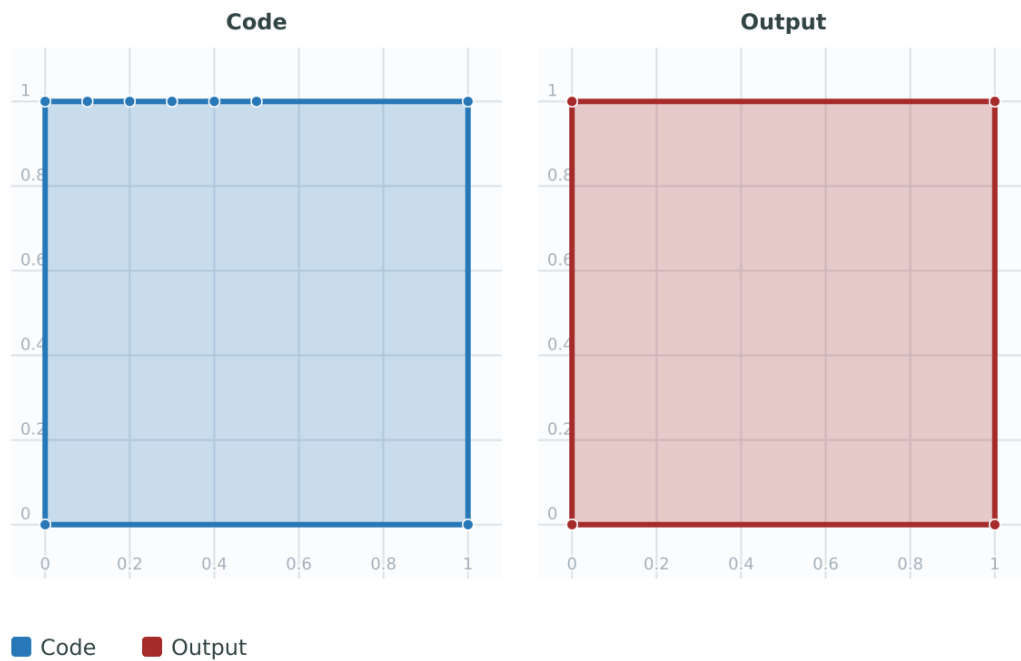
Code

```
SELECT CG_Simplify(
  '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
);
```

Output

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

Figure



Preserving topology can keep additional support points in a mixed ZM collection.

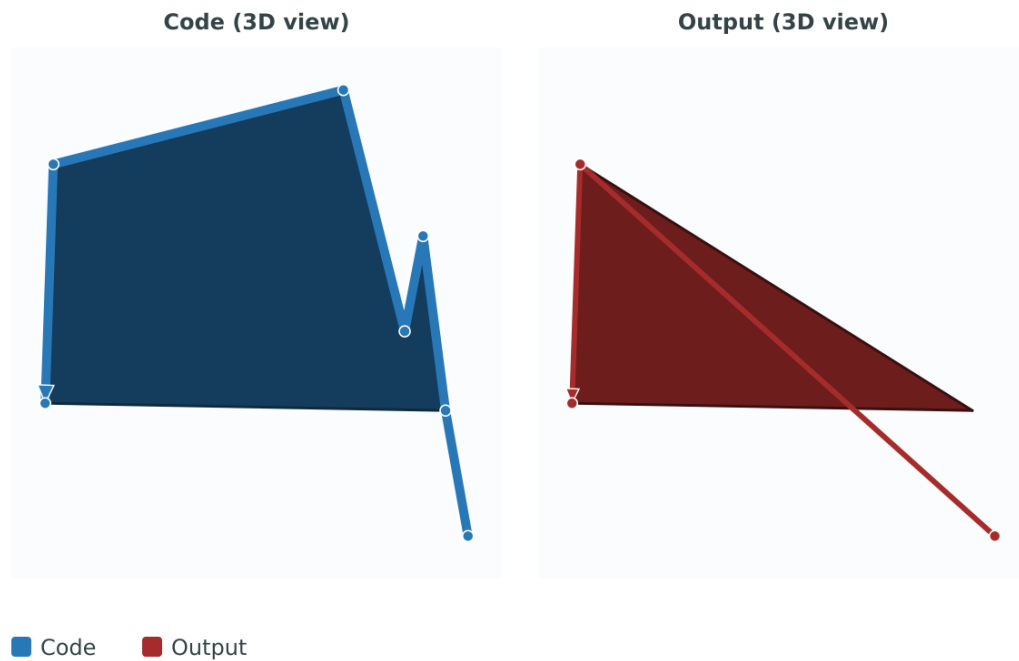
Code

```
SELECT 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
);
```

Output

```
GEOMETRYCOLLECTION ZM (LINESTRING ZM (-1 -1 3 4,2 19 25 250,-4 20 15 150),POLYGON ZM ((0 0 10 100,2 19 25 250,-4 20 15 150,0 0 10 100)))
```

Figure



The topology-preserving variant keeps the shared anchor point at the collection intersection.

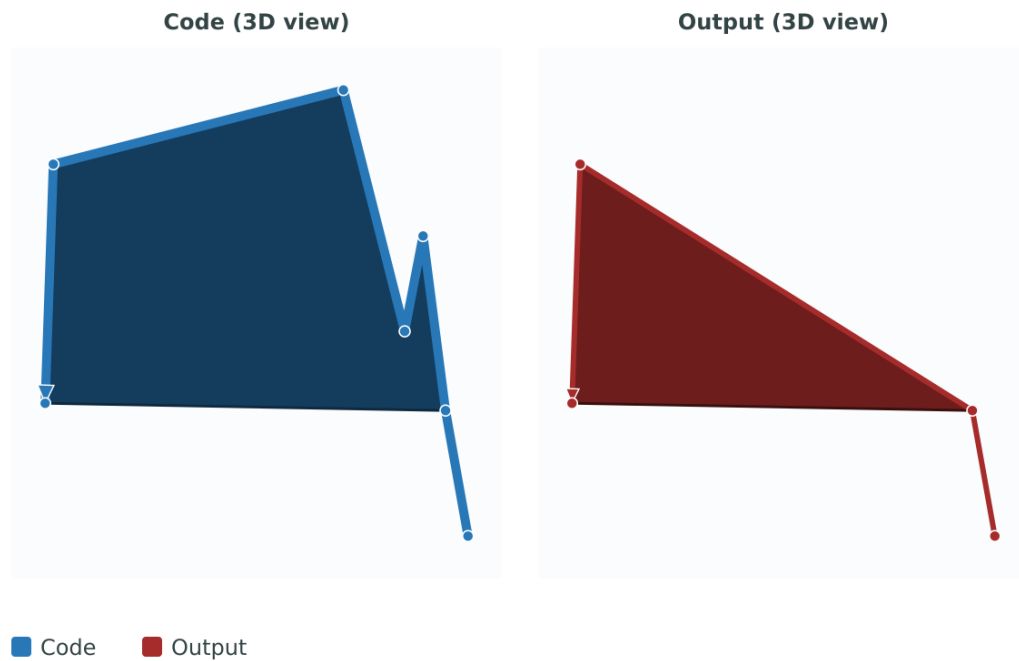
Code

```
SELECT 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
);
```

Output

```
GEOMETRYCOLLECTION ZM (LINESTRING ZM (-1 -1 3 4,0 0 10 100,2 19 25 250,-4 20 15 150),POLYGON ZM ((0 0 10 100,2 19 25 250,-4 20 15 150,0 0 10 100)))
```

Figure



Compare simplification with and without topology preservation, using a threshold of 50.

Code

```
WITH data 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 AS input_geometry,
       CG_Simplify(geom, 50, true) AS topology_preserved,
       CG_Simplify(geom, 50, false) AS topology_not_preserved
FROM data;
```

Figure



相关信息

[ST_Simplify](#), [ST_SimplifyPreserveTopology](#)

8.3.47 CG_3DAlphaWrapping

CG_3DAlphaWrapping — 计算一个网格包围给定几何的 3D Alpha 包围。

Synopsis

geometry **CG_3DAlphaWrapping**(geometry geom, integer relative_alpha, integer relative_offset);

描述

计算几何中点的 **3D Alpha Wrapping**。Alpha wrapping 是一种封闭且可定向的曲面网格，能网格包住输入点集合，可展示 **alpha-shape** 的展开或可视化。

The `relative_alpha` parameter controls which features will appear in the output. It can have values from 0 to infinity. Higher `relative_alpha` values produce coarser wraps; lower values retain more detail.

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.

可用性: 3.6.0 - 需要 SFCGAL >= 2.1.0

- ✓ 方法需要 SFCGAL 后端。
- ✓ 函数支持 3d 并且不会丢失 z-index。

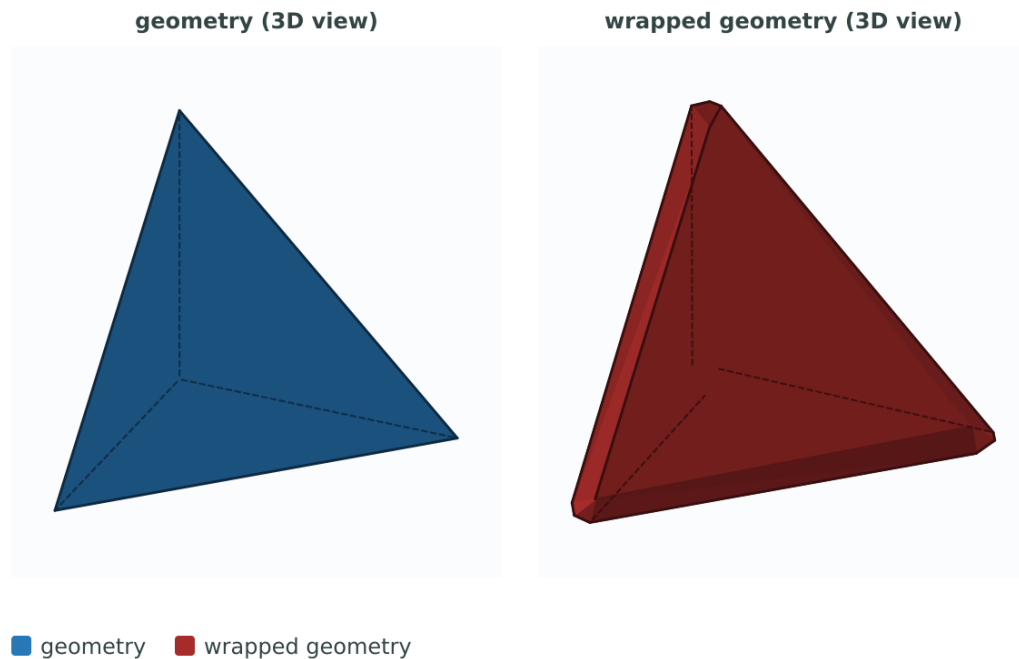
示例

Wrap a tetrahedral surface and show the native 3D input beside the actual enclosing mesh.

| Code

```
WITH data AS (
  SELECT 'POLYHEDRALSURFACE Z (
    ((0 0 0,4 0 0,0 4 0,0 0 0)),
    ((0 0 0,0 0 4,4 0 0,0 0 0)),
    ((0 0 0,0 4 0,0 0 4,0 0 0)),
    ((4 0 0,0 0 4,0 4 0,4 0 0))
  )'::geometry AS input_geometry
)
SELECT input_geometry AS input_geometry,
       CG_3DAlphaWrapping(input_geometry, 1)
       AS wrapped_geometry
FROM data;
```

| Figure



相关信息

[CG_AlphaShape](#)

8.3.48 CG_NurbsCurveInterpolate

CG_NurbsCurveInterpolate — Creates an interpolating NURBS curve passing through all given data points

Synopsis

geometry **CG_NurbsCurveInterpolate**(geometry data_points, integer degree);

描述

Creates a NURBS curve that interpolates through all provided data points. The resulting curve passes exactly through each point in the input geometry.

Parameters:

- `data_points` - A LINESTRING geometry containing the points to interpolate through.
- `degree` - Polynomial degree of the NURBS curve (typically 2 or 3).

Availability: 3.7.0 - requires SFCGAL >= 2.3.0.



☑方法需要 SFCGAL 后端。



☑函数支持 3d 并且不会☑失 z-index。

示例

Interpolate a degree-3 NURBS curve through four 2D points.

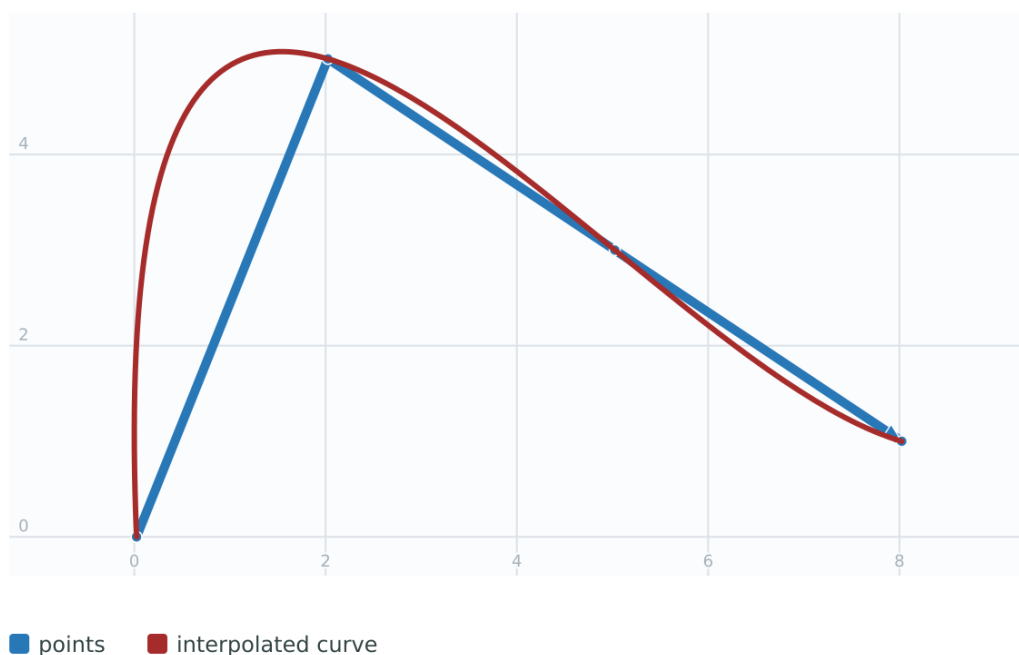
Code

```
WITH data AS (
  SELECT 'LINESTRING(0 0,2 5,5 3,8 1)::geometry' AS input_points
)
SELECT input_points AS input_points,
       CG_NurbsCurveInterpolate(input_points, 3) AS interpolated_curve
FROM data;
```

Output

```
LINESTRING(0 0,2 5,5 3,8 1) | NURBS CURVE(DEGREE 3, CONTROLPOINTS(NURBSPOINT(WEIGHTEDPOINT(0 0),WEIGHT 1), NURBSPOINT(WEIGHTEDPOINT(-0.43 10.29),WEIGHT 1), NURBSPOINT(WEIGHTEDPOINT(4.95 1.9),WEIGHT 1), NURBSPOINT(WEIGHTEDPOINT(8 1),WEIGHT 1)),KNOTS (KNOT(0,4), KNOT(1,4)))
```

Figure



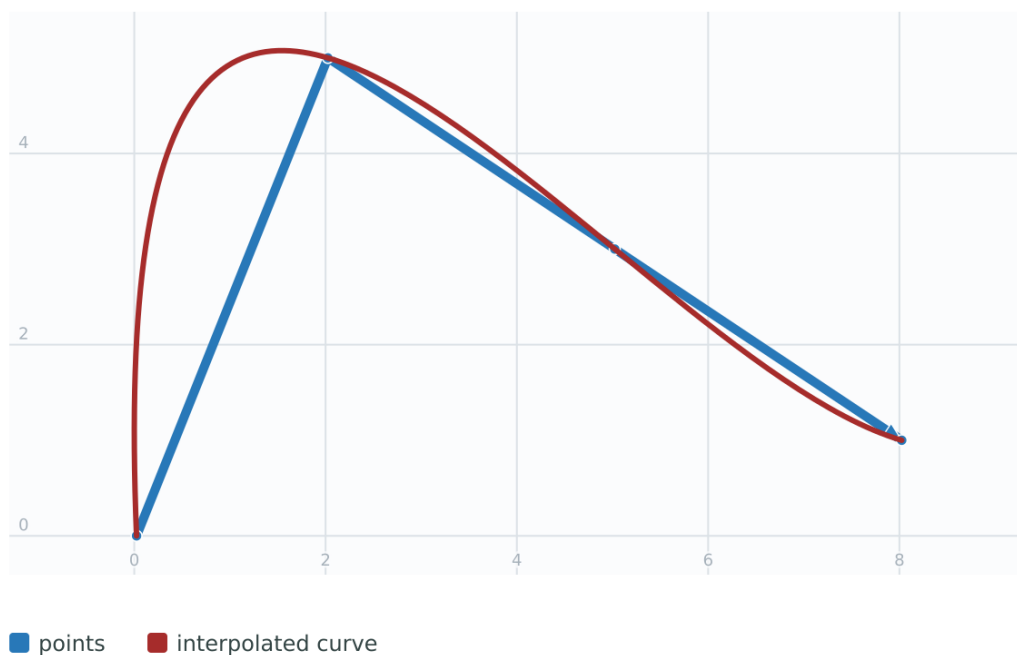
Interpolate a 3D degree-3 NURBS curve through the documented control points.

Code

```
WITH data AS (
  SELECT 'LINESTRING Z (0 0 0,2 5 2,5 3 1,8 1 0)>::geometry AS input_points
)
SELECT input_points AS input_points,
       CG_NurbsCurveInterpolate(input_points, 3) AS interpolated_curve
FROM data;
```

Output

```
LINESTRING Z (0 0 0,2 5 2,5 3 1,8 1 0) | NURBS CURVE Z (DEGREE 3, CONTROLPOINTS Z (NURBSPOINT(WEIGHTEDPOINT Z (0 0 0), ←
WEIGHT 1), NURBSPOINT(WEIGHTEDPOINT Z (-0.43 10.29 4.36), WEIGHT 1), NURBSPOINT(WEIGHTEDPOINT Z (4.95 1.9 0.54), ←
WEIGHT 1), NURBSPOINT(WEIGHTEDPOINT Z (8 1 0), WEIGHT 1)), KNOTS (KNOT(0,4), KNOT(1,4)))
```

Figure**相关信息**

[CG_NurbsCurveApproximate](#), [ST_MakeNurbsCurve](#)

8.3.49 CG_NurbsCurveApproximate

CG_NurbsCurveApproximate — Creates an approximating NURBS curve fitting data points within a tolerance

Synopsis

```
geometry CG_NurbsCurveApproximate(geometry data_points, integer degree, float8 tolerance, integer max_control_points=100);
```

描述

Creates a NURBS curve that approximates the given data points within the specified tolerance. Unlike interpolation, the curve does not necessarily pass through all points but provides a smooth fit with fewer control points.

Parameters:

- `data_points` - A LINESTRING geometry containing the points to approximate.
- `degree` - Polynomial degree of the NURBS curve.
- `tolerance` - Maximum allowed distance between the curve and data points.
- `max_control_points` - Optional maximum number of control points (default: 100). Limits curve complexity.

Availability: 3.7.0 - requires SFCGAL >= 2.3.0.



☑方法需要 SFCGAL 后端。



☑函数支持 3d 并且不会☑失 z-index。

示例

Approximate noisy 2D data with a degree-2 curve and a tolerance of 0.5.

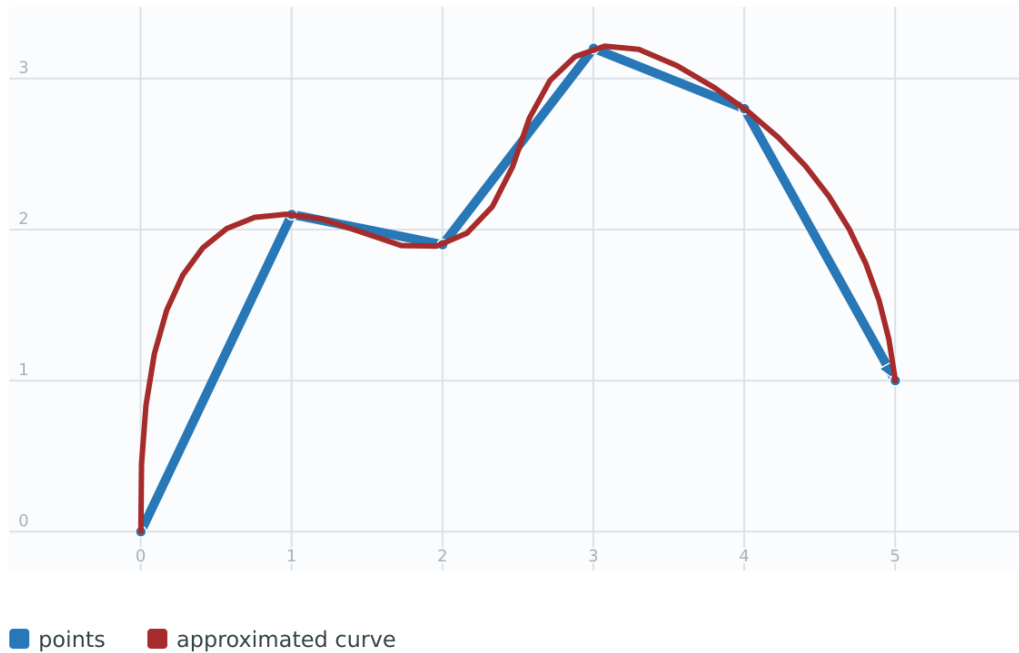
Code

```
WITH data AS (
  SELECT 'LINESTRING(0 0,1 2.1,2 1.9,3 3.2,4 2.8,5 1)::geometry' AS input_points
)
SELECT input_points AS input_points,
       CG_NurbsCurveApproximate(input_points, 2, 0.5) AS approximated_curve
FROM data;
```

Output

```
LINESTRING(0 0,1 2.1,2 1.9,3 3.2,4 2.8,5 1) | NURBS CURVE( DEGREE 2, CONTROLPOINTS(NURBSPOINT(WEIGHTEDPOINT(0 0),WEIGHT 1), NURBSPOINT(WEIGHTEDPOINT(-0.05 2.64),WEIGHT 1), NURBSPOINT(WEIGHTEDPOINT(2.24 1.65),WEIGHT 1), NURBSPOINT(WEIGHTEDPOINT(2.79 3.48),WEIGHT 1), NURBSPOINT(WEIGHTEDPOINT(4.84 2.43),WEIGHT 1), NURBSPOINT(WEIGHTEDPOINT(5 1),WEIGHT 1)),KNOTS (KNOT(0,3), KNOT(0.35,1), KNOT(0.51,1), KNOT(0.68,1), KNOT(1,3)))
```

Figure



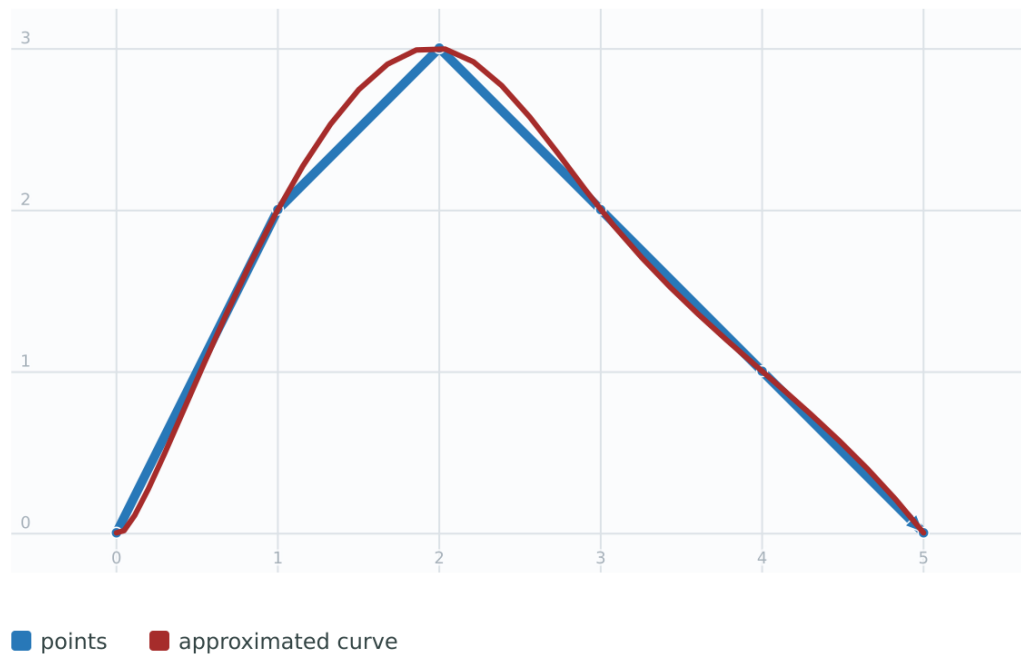
Approximate a 3D point set while preserving the Z dimension in the fitted curve.

Code

```
WITH data AS (
  SELECT 'LINESTRING Z (0 0 0,1 2 1,2 3 2,3 2 1,4 1 0,5 0 0)>:::geometry AS input_points
)
SELECT input_points AS input_points,
       CG_NurbsCurveApproximate(input_points, 3, 0.1) AS approximated_curve
FROM data;
```

Output

```
LINESTRING Z (0 0 0,1 2 1,2 3 2,3 2 1,4 1 0,5 0 0) | NURBS CURVE Z (DEGREE 3, CONTROLPOINTS Z (NURBSPOINT(WEIGHTEDPOINT ↵
Z (0 0 0),WEIGHT 1), NURBSPOINT(WEIGHTEDPOINT Z (0.18 -0.15 -1.29),WEIGHT 1), NURBSPOINT(WEIGHTEDPOINT Z (1.49 ↵
4.36 3.35),WEIGHT 1), NURBSPOINT(WEIGHTEDPOINT Z (3.33 1.33 0.38),WEIGHT 1), NURBSPOINT(WEIGHTEDPOINT Z (4.34 ↵
0.83 -0.44),WEIGHT 1), NURBSPOINT(WEIGHTEDPOINT Z (5 0 0),WEIGHT 1)),KNOTS (KNOT(0,4), KNOT(0.46,1), KNOT(0.64,1) ↵
, KNOT(1,4)))
```

Figure

相关信息

[CG_NurbsCurveInterpolate](#), [ST_MakeNurbsCurve](#)

8.3.50 CG_NurbsCurveDerivative

`CG_NurbsCurveDerivative` — Computes the derivative (tangent, curvature) of a NURBS curve at a given parameter

Synopsis

`geometry CG_NurbsCurveDerivative(geometry nurbscurve, float8 parameter, integer derivative_order);`

描述

Computes the derivative of a NURBS curve at a specific parameter value. The derivative represents the rate of change at that point:

- Order 1: Tangent vector (direction and speed)
- Order 2: Curvature vector (how the tangent changes)
- Order 3: Third-order derivative (supported maximum)

Parameters:

- `nurbscurve` - A NURBS curve geometry.
- `parameter` - Parameter value for evaluation (typically 0.0 to 1.0).

- `derivative_order` - Order of derivative to compute (1, 2, or 3).

Returns a POINT geometry representing the derivative vector at the specified parameter.

Availability: 3.7.0 - requires SFCGAL >= 2.3.0.



☒方法需要 SFCGAL 后端。



☒函数支持 3d 并且不会☒失 z-index。

示例

The first derivative gives the tangent vector at the evaluation parameter.

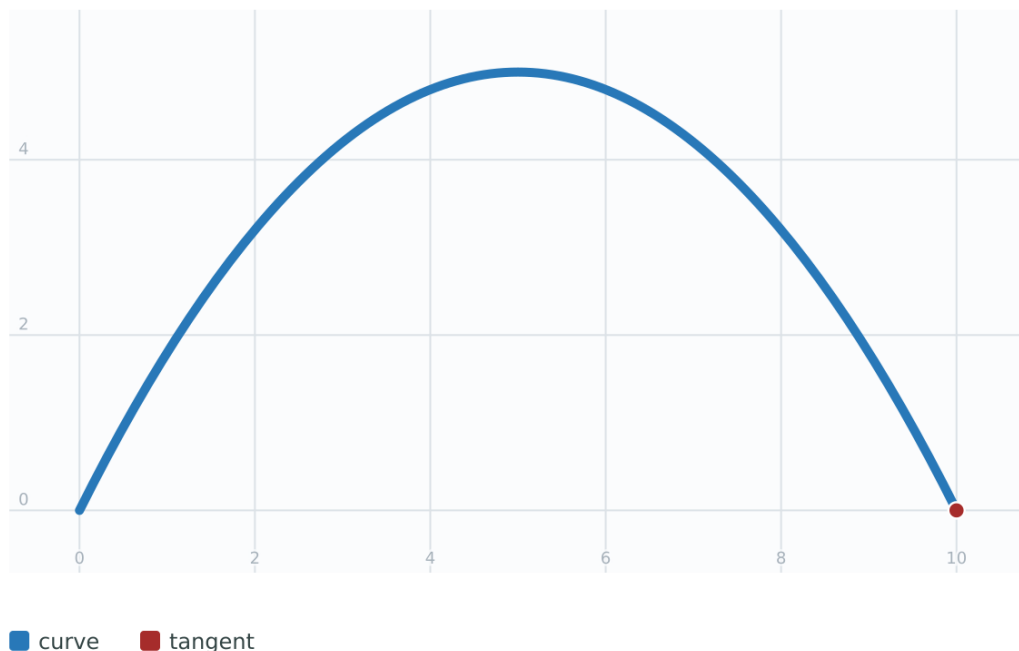
Code

```
WITH data AS (  
  SELECT 'NURBSCURVE (2, (0.00 0.00, 5.00 10.00, 10.00 0.00))'::geometry AS input_curve  
)  
SELECT input_curve AS input_curve,  
       CG_NurbsCurveDerivative(input_curve, 0.5, 1) AS tangent  
FROM data;
```

Output

```
NURBSCURVE(DEGREE 2, CONTROLPOINTS(NURBSPPOINT(WEIGHTEDPOINT(0 0),WEIGHT 1), NURBSPPOINT(WEIGHTEDPOINT(5 10),WEIGHT 1),  
NURBSPPOINT(WEIGHTEDPOINT(10 0),WEIGHT 1)),KNOTS (KNOT(0,3), KNOT(1,3))) | POINT Z (10 0 0) ↵
```

Figure



The second derivative exposes curvature change for a higher-degree curve.

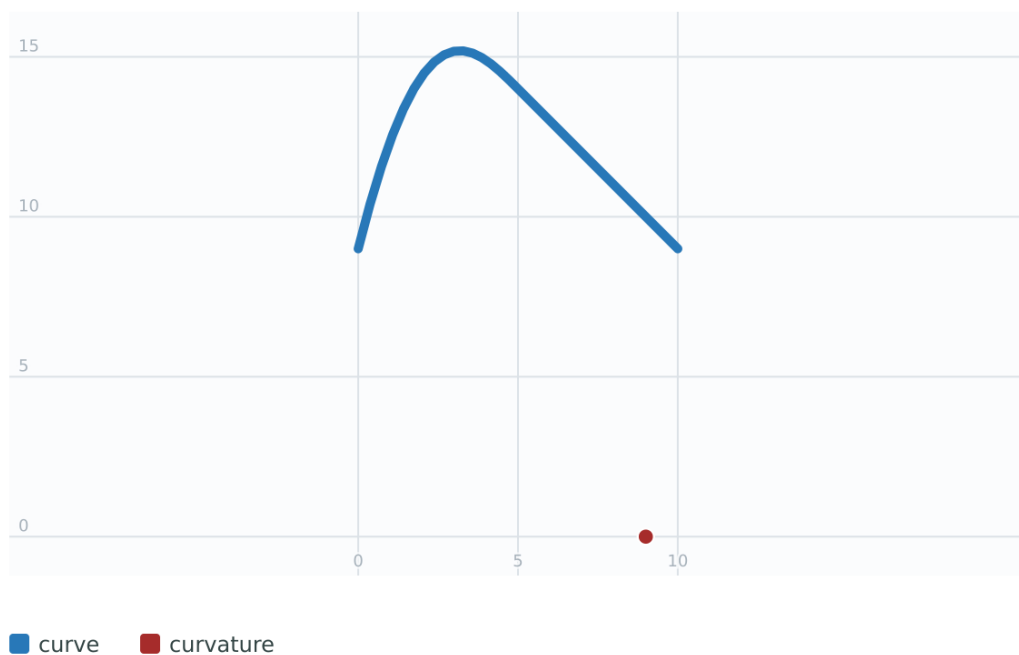
Code

```
WITH data AS (
  SELECT 'NURBSCURVE (3, (0.00 0.00, 2.00 8.00, 5.00 5.00, 8.00 2.00, 10.00 0.00))':: geometry AS input_curve
)
SELECT input_curve AS input_curve,
       CG_NurbsCurveDerivative(input_curve, 0.5, 2) AS curvature
FROM data;
```

Output

```
NURBSCURVE(DEGREE 3, CONTROLPOINTS(NURBSPOINT(WEIGHTEDPOINT(0 0),WEIGHT 1), NURBSPOINT(WEIGHTEDPOINT(2 8),WEIGHT 1),
NURBSPOINT(WEIGHTEDPOINT(5 5),WEIGHT 1), NURBSPOINT(WEIGHTEDPOINT(8 2),WEIGHT 1), NURBSPOINT(WEIGHTEDPOINT(10 0),
WEIGHT 1)),KNOTS (KNOT(0,4), KNOT(0.5,1), KNOT(1,4))) | POINT Z (9 -9 0)
```

Figure



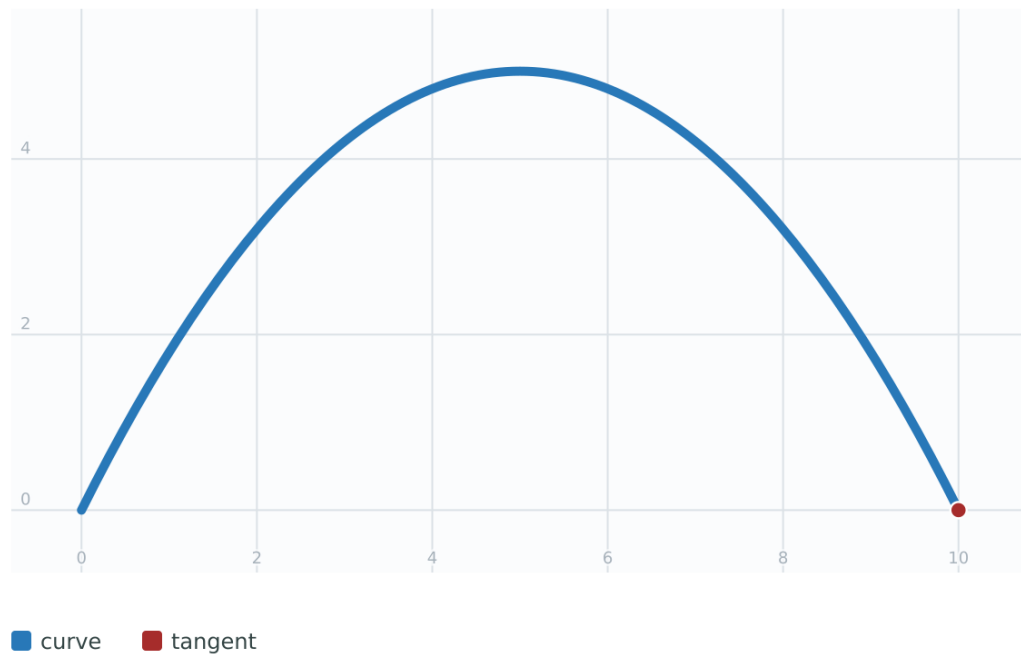
The same derivative call also works for 3D NURBS curves.

Code

```
WITH data AS (
  SELECT 'NURBSCURVE Z (2, (0.00 0.00 0.00, 5.00 10.00 5.00, 10.00 0.00 0.00))'::geometry AS input_curve
)
SELECT input_curve AS input_curve,
       CG_NurbsCurveDerivative(input_curve, 0.5, 1) AS tangent
FROM data;
```

Output

```
NURBSCURVE Z (DEGREE 2,CONTROLPOINTS Z (NURBSPOINT(WEIGHTEDPOINT Z (0 0 0),WEIGHT 1), NURBSPOINT(WEIGHTEDPOINT Z (5 10 5),WEIGHT 1), NURBSPOINT(WEIGHTEDPOINT Z (10 0 0),WEIGHT 1)),KNOTS (KNOT(0,3), KNOT(1,3))) | POINT Z (10 0 0)
```


Figure

相关信息

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

8.4 Deprecated SFCGAL Function Aliases

These ST_ names remain available for compatibility and emit deprecation notices. Their entries contain only the legacy signature and migration link. Use the corresponding CG_ pages for behavior, examples, and geometry support details.

8.4.1 ST_3DIntersection

ST_3DIntersection — Deprecated alias for CG_3DIntersection.

Synopsis

geometry **ST_3DIntersection**(geometry geom1, geometry geom2);

Description**Warning**

ST_3DIntersection is deprecated as of 3.5.0. Use [CG_3DIntersection](#) instead.

Availability: 2.1.0



☒方法需要 SFCGAL 后端。

8.4.2 ST_3DDifference

ST_3DDifference — Deprecated alias for CG_3DDifference.

Synopsis

geometry **ST_3DDifference**(geometry geom1, geometry geom2);

Description



Warning

ST_3DDifference is deprecated as of 3.5.0. Use **CG_3DDifference** instead.

Availability: 2.2.0



☒方法需要 SFCGAL 后端。

8.4.3 ST_3DUnion

ST_3DUnion — Deprecated alias for CG_3DUnion.

Synopsis

geometry **ST_3DUnion**(geometry geom1, geometry geom2);
geometry **ST_3DUnion**(geometry set g1field);

Description



Warning

ST_3DUnion is deprecated as of 3.5.0. Use **CG_3DUnion** instead.

Availability: 2.2.0



☒方法需要 SFCGAL 后端。

8.4.4 ST_Tessellate

ST_Tessellate — Deprecated alias for CG_Tessellate.

Synopsis

geometry **ST_Tessellate**(geometry geom);

Description



Warning

ST_Tessellate is deprecated as of 3.5.0. Use **CG_Tessellate** instead.

Availability: 2.1.0



☑方法需要 SFCGAL 后端。

8.4.5 ST_3DArea

ST_3DArea — Deprecated alias for CG_3DArea.

Synopsis

float**ST_3DArea**(geometry geom1);

Description



Warning

ST_3DArea is deprecated as of 3.5.0. Use **CG_3DArea** instead.

Availability: 2.1.0



☑方法需要 SFCGAL 后端。

8.4.6 ST_Extrude

ST_Extrude — Deprecated alias for CG_Extrude.

Synopsis

geometry **ST_Extrude**(geometry geom, float x, float y, float z);

Description



Warning

ST_Extrude is deprecated as of 3.5.0. Use **CG_Extrude** instead.

Availability: 2.1.0



☑方法需要 SFCGAL 后端。

8.4.7 ST_ForceLHR

ST_ForceLHR — Deprecated alias for CG_ForceLHR.

Synopsis

geometry **ST_ForceLHR**(geometry geom);

Description



Warning

ST_ForceLHR is deprecated as of 3.5.0. Use **CG_ForceLHR** instead.

Availability: 2.1.0



☑方法需要 SFCGAL 后端。

8.4.8 ST_Orientation

ST_Orientation — Deprecated alias for CG_Orientation.

Synopsis

integer **ST_Orientation**(geometry geom);

Description



Warning

ST_Orientation is deprecated as of 3.5.0. Use **CG_Orientation** instead.

Availability: 2.1.0



☑方法需要 SFCGAL 后端。

8.4.9 ST_MinkowskiSum

ST_MinkowskiSum — Deprecated alias for CG_MinkowskiSum.

Synopsis

geometry **ST_MinkowskiSum**(geometry geom1, geometry geom2);

Description



Warning

ST_MinkowskiSum is deprecated as of 3.5.0. Use **CG_MinkowskiSum** instead.

Availability: 2.1.0



☑方法需要 SFCGAL 后端。

8.4.10 ST_StraightSkeleton

ST_StraightSkeleton — Deprecated alias for CG_StraightSkeleton.

Synopsis

geometry **ST_StraightSkeleton**(geometry geom);

Description



Warning

ST_StraightSkeleton is deprecated as of 3.1.0. Use **CG_StraightSkeleton** instead.



Note

Like CG_StraightSkeleton, this function ignores the Z dimension and always gives a 2D result, even when used on a 3D geometry.

Availability: 2.1.0



☑方法需要 SFCGAL 后端。

8.4.11 ST_ApproximateMedialAxis

ST_ApproximateMedialAxis — Deprecated alias for CG_ApproximateMedialAxis.

Synopsis

geometry **ST_ApproximateMedialAxis**(geometry geom);

Description



Warning

`ST_ApproximateMedialAxis` is deprecated as of 3.5.0. Use `CG_ApproximateMedialAxis` instead.



Note

Like `CG_ApproximateMedialAxis`, this function ignores the Z dimension and always gives a 2D result, even when used on a 3D geometry.

Availability: 2.2.0



☑方法需要 SFCGAL 后端。

8.4.12 ST_IsPlanar

`ST_IsPlanar` — Deprecated alias for `CG_IsPlanar`.

Synopsis

boolean **ST_IsPlanar**(geometry geom);

Description



Warning

`ST_IsPlanar` is deprecated as of 3.5.0. Use `CG_IsPlanar` instead.

Availability: 2.2.0: This was documented in 2.1.0 but got accidentally left out in 2.1 release.



☑方法需要 SFCGAL 后端。

8.4.13 ST_Volume

`ST_Volume` — Deprecated alias for `CG_Volume`.

Synopsis

float **ST_Volume**(geometry geom1);

Description



Warning

ST_Volume is deprecated as of 3.5.0. Use **CG_Volume** instead.

Availability: 2.2.0



☑方法需要 SFCGAL 后端。

8.4.14 ST_MakeSolid

ST_MakeSolid — Deprecated alias for CG_MakeSolid.

Synopsis

geometry **ST_MakeSolid**(geometry geom1);

Description



Warning

ST_MakeSolid is deprecated as of 3.5.0. Use **CG_MakeSolid** instead.

Availability: 2.2.0



☑方法需要 SFCGAL 后端。

8.4.15 ST_IsSolid

ST_IsSolid — Deprecated alias for CG_IsSolid.

Synopsis

boolean **ST_IsSolid**(geometry geom1);

Description



Warning

ST_IsSolid is deprecated as of 3.5.0. Use **CG_IsSolid** instead.

Availability: 2.2.0



☑方法需要 SFCGAL 后端。

8.4.16 ST_ConstrainedDelaunayTriangles

ST_ConstrainedDelaunayTriangles — Deprecated alias for CG_ConstrainedDelaunayTriangles.

Synopsis

geometry **ST_ConstrainedDelaunayTriangles**(geometry g1);

Description



Warning

ST_ConstrainedDelaunayTriangles is deprecated as of 3.5.0. Use **CG_ConstrainedDelaunayTriangles** instead.

Availability: 3.0.0



☒方法需要 SFCGAL 后端。

8.4.17 ST_3DConvexHull

ST_3DConvexHull — Deprecated alias for CG_3DConvexHull.

Synopsis

geometry **ST_3DConvexHull**(geometry geom1);

Description



Warning

ST_3DConvexHull is deprecated as of 3.5.0. Use **CG_3DConvexHull** instead.

Availability: 3.3.0



☒方法需要 SFCGAL 后端。

8.4.18 ST_AlphaShape

ST_AlphaShape — Deprecated alias for CG_AlphaShape.

Synopsis

geometry **ST_AlphaShape**(geometry geom, float alpha , boolean allow_holes = false);

Description



Warning

ST_AlphaShape is deprecated as of 3.5.0. Use [CG_AlphaShape](#) instead.



☒方法需要 SFCGAL 后端。

8.4.19 ST_OptimalAlphaShape

ST_OptimalAlphaShape — Deprecated alias for CG_OptimalAlphaShape.

Synopsis

```
geometry ST_OptimalAlphaShape(geometry geom, boolean allow_holes = false , integer nb_components = 1 );
```

Description



Warning

ST_OptimalAlphaShape is deprecated as of 3.5.0. Use [CG_OptimalAlphaShape](#) instead.

Availability: 3.3.0 - requires SFCGAL >= 1.4.1.



☒方法需要 SFCGAL 后端。

Chapter 9

拓扑

PostGIS 拓扑类型和函数用于管理拓扑对象，例如面、线和点。

Sandro Santilli 在 2011 年 PostGIS Day Paris 会议上的演示提供了关于 PostGIS 拓扑和未来发展方向的良好概述。您可以查看他的幻灯片演示，链接如下：[Topology with PostGIS 2.0 slide deck](#)。

Vincent Picavet 在[PostGIS 拓扑 PGConf EU 2012](#) 中讨论拓扑是什么、如何使用以及支持它的各种 FOSS4G 工具提供了很好的概要和概述。

An example of a topologically based GIS database is the [US Census Topologically Integrated Geographic Encoding and Referencing System \(TIGER\)](#) database. If you want to experiment with PostGIS topology and need some data.

PostGIS 的 topology 模块很早就存在了，过去并不在官方文档里。经过一次大范围清理后，弃函数被移除，已知的易用性问题得到修复、功能和接口都上了文档说明，同时加入了新特性，并进一步提升了 SQL-MM 兼容度。

This chapter is the maintained reference for the topology module, including its primitive tables, Topo-Geometry type, topology management functions, editing functions, validation functions, and conversion helpers.

与模块相关的所有函数和表都安装在称拓扑架中。

SQL/MM 标准中定义的函数以 ST_ 前缀，而 PostGIS 特定的函数没有前缀。

Topology 支持默认是启用的，如果在配置中指定 `--without-topology` 个配置选项，就可以关闭功能，具体可参考 [Chapter 2](#)

9.1 Building Faces from Linework

This example builds topology faces from noded linework and associates point attributes with the resulting faces. It uses [ST_CreateTopoGeo](#) to populate an empty topology, [ValidateTopology](#) to check the result, [GetFaceByPoint](#) to locate each attribute point, and [ST_GetFaceGeometry](#) to materialize ordinary polygon geometries.

Create an empty topology and populate it with linework that encloses two faces:

Code

```
CREATE EXTENSION IF NOT EXISTS postgis_topology;

SELECT topology.CreateTopology('topo_example', 3857);

WITH lines(geom) AS (VALUES
  ('SRID=3857;LINESTRING(0 0, 2 0)::geometry),
```

```

('SRID=3857;LINESTRING(0 1, 2 1)::geometry),
('SRID=3857;LINESTRING(0 0, 0 1)::geometry),
('SRID=3857;LINESTRING(1 0, 1 1)::geometry),
('SRID=3857;LINESTRING(2 0, 2 1)::geometry)
)
SELECT topology.ST_CreateTopoGeo(
    'topo_example',
    ST_Collect(geom)
)
FROM lines;

SELECT * FROM topology.ValidateTopology('topo_example');

```

An empty result from `ValidateTopology` means that no topology errors were found. Attribute points can now be matched to faces. A face identifier of 0 denotes the universal face, so the point is outside every bounded face.

Code

```

WITH labels(name, geom) AS (VALUES
    ('west', 'SRID=3857;POINT(0.5 0.5)::geometry),
    ('east', 'SRID=3857;POINT(1.5 0.5)::geometry),
    ('outside', 'SRID=3857;POINT(3 0.5)::geometry)
),
located AS (
    SELECT name,
        topology.GetFaceByPoint('topo_example', geom, 0) AS face_id
    FROM labels
)
SELECT name,
    face_id,
    CASE WHEN face_id = 0 THEN NULL
        ELSE topology.ST_GetFaceGeometry('topo_example', face_id)
    END AS geom
FROM located
ORDER BY name;

```

Before copying attributes into a face table, check both for points outside bounded faces and for multiple source points assigned to the same face:

Code

```

WITH labels(name, geom) AS (VALUES
    ('west', 'SRID=3857;POINT(0.5 0.5)::geometry),
    ('east', 'SRID=3857;POINT(1.5 0.5)::geometry),
    ('outside', 'SRID=3857;POINT(3 0.5)::geometry)
),
located AS (
    SELECT name,
        topology.GetFaceByPoint('topo_example', geom, 0) AS face_id
    FROM labels
)
SELECT 'outside face' AS problem,
    array_agg(name ORDER BY name) AS labels
FROM located
WHERE face_id = 0
HAVING count(*) > 0
UNION ALL
SELECT 'duplicate face ' || face_id,
    array_agg(name ORDER BY name)
FROM located
WHERE face_id != 0

```

```
GROUP BY face_id
HAVING count(*) > 1;
```

Use **DropTopology** to remove the example topology when it is no longer needed:

Code

```
SELECT topology.DropTopology('topo_example');
```

9.2 拓扑基表

任意拓扑图的核心原始要素会存放在 `edge_data`、`node` 和 `face` 几个表中，它们位于由 **CreateTopology** 建立的 schema 里。`edge_data` 表中的每一行代表一条有向边：定义了从 `start_node` 指向 `end_node` 的有向曲线，以及边方向左侧相邻的面（`left_face`）和右侧相邻的面（`right_face`）的 ID。当一段几何边属于一个面，它可能会出边次——分边出边种方向。

`next_left_edge` 和 `next_right_edge` 两个列用来包含有向边的方向信息，描述了沿着某个面的边界走下去的路径。它存储的是符号整数，正数代表下一条有向边的 ID，符号用来指示遍边是按边方向走还是反向走。任意边 `e`，都有如下边：

- `abs(next_left_edge)` 的值是沿着 `e` 左侧那个面的边界走，会到边的下一条边的 ID。如果该值是正数，表示从 `e` 的边点出边，沿着被引用边的方向前边；如果是负数，意味着遍那条被引用边需要反向走，边才能保边共享的那个面始边保持在行边方向的左边。
- `abs(next_right_edge)` 的值同理用于沿着 `e` 右侧那个面的边界遍边。如果该值是正数，表示从 `e` 的边点出边，按被引用边的方向前边；如果是负数，边需要从那条被引用边的边点边反向走，边才能确保右边的那个面始边保持在行边方向的右边。
- 边 0 表示边在边那一边是空的（比如一条孤立的边，它的相邻面就是通用面 0）。`edge` 边里提供的 `abs_next_left_edge` 和 `abs_next_right_edge` 列，边是边些边做的便捷投影，方便直接使用。

边种表示法属于双向边接边表的一种边体，被大量拓扑例程所利用。像 **GetRingEdges**、**ValidateTopology** 边函数依边一边的来重建面的边界、边断拓扑不一致边——因此在校边边才会出边似 “invalid next left edge” 或 “invalid next right edge” 边的提示。边造函数（例如 **AddEdge**）会边 `next_*` 边些属性做初始化，默边是指向自身的边引用；而边边操作（如 **ST_AddEdgeModFace**、**ST_RemEdgeModFace**）会在边插入或边除边即边更新边些边接。某些批量操作（例如 **Polygonize**）可能会有意不边置边些字段，所以在文档里也会特边注它边的行边。

9.3 拓扑边型

9.3.1 getfaceedges_returntype

`getfaceedges_returntype` — 由序号和边号边成的复合边型。

Description

由序列号和边号边成的复合边型。边是 **ST_GetFaceEdges** 和 **GetNodeEdges** 函数的返回边型。

1. `sequence` 是一个整数：指的是在 `topology.topology` 表中定义的拓扑，边表定义了拓扑架边和 `srid`。
2. `edge` 是一个整数：边的边号。

9.3.2 TopoGeometry

TopoGeometry — 表示拓扑定义的几何图形的复合类型。

Description

一个复合类型，它引用了特定拓扑中的拓扑几何图形，具有特定的类型和特定的标识符。TopoGeometry 的元素包括属性：topology_id、layer_id、id 整数和 type 整数。

- 1. topology_id 是一个整数：指的是在 topology.topology 表中定义的拓扑，标识定义了拓扑框架和 srid。
- 2. layer_id （整数）：拓扑几何所属的层表中的 layer_id。topology_id 和 layer_id 的组合唯一地引用了拓扑层表。
- 3. id 是一个整数：id 是自生成的序列号，唯一标识拓扑中的主几何图形。
- 4. type 输入 1 - 4 之间的整数，定义几何类型：1:[多]点、2:[多]线、3:[多]多边形、4: 集合

类型限制

个部分列出了允许用于该数据类型的有效值以及格式

值到	行
geometry	automatic

相关信息

CreateTopoGeom

9.3.3 validate_topology_return_type

validate_topology_return_type — 复合类型，由错误消息以及表示错误位置的 id1 和 id2 组成。是 ValidateTopology 的返回类型。

Description

由错误消息和两个整数组成的复合类型。ValidateTopology 函数返回一行来表示错误，并返回 id1 和 id2 来表示错误中涉及的拓扑对象的 id。

Field	Type	Description
error	varchar	Denotes the type of error. Current error descriptors are: coincident nodes, edge crosses node, edge not simple, edge end node geometry mismatch, edge start node geometry mismatch, face overlaps face, face within face.
id1	integer	Denotes the identifier of the edge, face, or node in error.

Field	Type	Description
id2	integer	For errors that involve two objects, denotes the secondary edge or node.

相关信息

[ValidateTopology](#)

9.4 拓扑域

9.4.1 TopoElement

TopoElement — 一个由 2 个整数组成的数组，用于表示面或分面拓扑几何的一个物件。

Description

一个由 2 个整数组成的数组，用于表示面或分面 **TopoGeometry** 的一个物件。

对于面的 TopoGeometry，数组的第一个元素表示拓扑基元的标识符，第二个元素表示其类型（1：点、2：边、3：面）。在分面 TopoGeometry 的情况下，数组的第一个元素表示子 TopoGeometry 的标识符，第二个元素表示其面标识符。



Note

对于任何给定的面 TopoGeometry，所有子 TopoGeometry 元素都将来自同一子面，如所定义的面 TopoGeometry 的 topology.layer 属性中所指定。

示例

Code

```
SELECT te[1] AS id, te[2] AS type FROM
( SELECT ARRAY[1, 2]::topology.topoelement AS te ) f;
```

Output

```
id | type
----+-----
 1 |    2
```

Code

```
SELECT ARRAY[1, 2]::topology.topoelement;
```

| Output

```
te
-----
{1,2}
```

This example shows what happens when casting a three-element array to topology.topoelement. The domain requires a two-element array, so the dimension check fails.

| Code

```
SELECT ARRAY[1, 2, 3]::topology.topoelement;
```

| Output

```
ERROR:  value for domain topology.topoelement violates check constraint "dimensions"
```

相关信息

[GetTopoGeomElements](#), [TopoElementArray](#), [TopoGeometry](#), [TopoGeom_addElement](#), [TopoGeom_remElement](#)

9.4.2 TopoElementArray

TopoElementArray — TopoElement 对象的数组。

Description

1 个或多个 TopoElement 对象的数组，通常用于 `TopoGeometry` 对象的组件。

示例

| Code

```
SELECT '{{1,2},{4,3}}'::topology.topoelementarray AS tea;
```

| Output

```
{{1,2},{4,3}}
```

This is the more verbose equivalent.

| Code

```
SELECT ARRAY[ARRAY[1, 2], ARRAY[4, 3]]::topology.topoelementarray AS tea;
```

| Output

```
{{1,2},{4,3}}
```

This example uses the array aggregate function packaged with topology.

Code

```
SELECT topology.TopoElementArray_Agg(ARRAY[e, t] ORDER BY e, t) As tea
FROM generate_series(1, 4) As e CROSS JOIN generate_series(1, 3) As t;
```

Output

```
{{1,1},{1,2},{1,3},{2,1},{2,2},{2,3},{3,1},{3,2},{3,3},{4,1},{4,2},{4,3}}
```

Code

```
SELECT '{{1,2,4},{3,4,5}}'::topology.topoelementarray AS tea;
```

Output

```
ERROR: value for domain topology.topoelementarray violates check constraint "dimensions"
```

相关信息

[TopoElement](#), [GetTopoGeomElementArray](#), [TopoElementArray_Agg](#)

9.5 拓扑和拓扑几何管理

9.5.1 AddTopoGeometryColumn

AddTopoGeometryColumn — 将拓扑几何列添加到现有表中，将此新列注册到 `topology.layer` 中的层并返回新的 `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);
```

Description

每个 `TopoGeometry` 对象都属于特定拓扑的特定层。在建 `TopoGeometry` 对象之前，您需要建其 `Topology-Layer`。拓扑层是要素表与拓扑的关联。它包含类型和层次信息。我们使用 `AddTopoGeometryColumn()` 函数建一个层：

此函数会将请求的列添加到表中，并将包含所有层信息的元组添加到 `topology.layer` 表中。

如果您不指定 `[child_layer]`（或将其置为 `NULL`），元组将包含基本拓扑几何（由原始拓扑元素组成）。否则，元组将包含分层的 `TopoGeometries`（由 `child_layer` 中的 `TopoGeometries` 组成）。

建层后（其 `id` 由 `AddTopoGeometryColumn` 函数返回），您就可以在其中造 `TopoGeometry` 对象

有效的 `feature_types` 是：POINT、MULTIPOINT、LINE、MULTILINE、POLYGON、MULTIPOLYGON、COLLECTION

可用性：1.1

示例

This example creates the new table in the `ma_topo` schema. The table could be created in a different schema, in which case `topology_name` and `schema_name` would be different.

Code

```
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');
```

Code

```
CREATE SCHEMA ri;
CREATE TABLE ri.roads(gid serial PRIMARY KEY, road_name text);
SELECT topology.AddTopoGeometryColumn('ri_topo', 'ri', 'roads', 'topo', 'LINE');
```

相关信息

[DropTopoGeometryColumn](#), [toTopoGeom](#), [CreateTopology](#), [CreateTopoGeom](#)

9.5.2 RenameTopoGeometryColumn

`RenameTopoGeometryColumn` — 重命名拓扑几何列

Synopsis

`topology.layer` **`RenameTopoGeometryColumn`**(regclass `layer_table`, name `feature_column`, name `new_name`)

Description

此函数更改具有 `TopoGeometry` 列的名称，确保有关它的元数据信息相应更新。

可用性：3.4.0

示例

Code

```
SELECT topology.RenameTopoGeometryColumn('public.parcels', 'topogeom', 'tgeom');
```

相关信息

[AddTopoGeometryColumn](#), [RenameTopology](#)

9.5.3 DropTopology

`DropTopology` — 谨慎使用：删除拓扑模式并从 `topology.topology` 表中删除其引用，并从 `geometry_columns` 表中删除模式中表的引用。

Synopsis

integer **DropTopology**(varchar topology_schema_name);

Description

删除拓扑模式并从 topology.topology 表中删除其引用，并从 geometry_columns 表中删除该模式中表的引用。谨慎使用此功能，因为它可能会破坏您关心的数据。如果架不存在，它只会删除指定架的引用条目。

可用性：1.1

示例

Cascade 会删除 ma_topo 架，并删除在 topology.topology 和 geometry_columns 中它的所有引用。

Code

```
SELECT topology.DropTopology('ma_topo');
```

相关信息

[DropTopoGeometryColumn](#)

9.5.4 RenameTopology

RenameTopology — 重命名拓扑

Synopsis

varchar **RenameTopology**(varchar old_name, varchar new_name);

Description

重命名拓扑架，更新其在 topology.topology 表中的元数据。

可用性：3.4.0

示例

将拓扑从 topo_stage 重命名为 topo_prod。

Code

```
SELECT topology.RenameTopology('topo_stage', 'topo_prod');
```

相关信息

[CopyTopology](#), [RenameTopoGeometryColumn](#)

9.5.5 DropTopoGeometryColumn

DropTopoGeometryColumn — 从架构 `schema_name` 中名为 `table_name` 的表中删除拓扑几何列，并从 `topology.layer` 表中取消注册某些列。

Synopsis

```
text DropTopoGeometryColumn(varchar schema_name, varchar table_name, varchar column_name);
```

Description

从架构 `schema_name` 中名为 `table_name` 的表中删除拓扑几何列，并从 `topology.layer` 表中取消注册某些列。返回放置状态的摘要。注意：它首先将所有列置为 NULL，然后再删除以引用完整性。

可用性：1.1

示例

Code

```
SELECT topology.DropTopoGeometryColumn('ma_topo', 'parcel_topo', 'topo');
```

相关信息

[AddTopoGeometryColumn](#)

9.5.6 FixCorruptTopoGeometryColumn

FixCorruptTopoGeometryColumn — 修复由升级到 `postgis_topology 3.6.0` 及以上版本引起的 `topogeometry` 数据损坏。

Synopsis

```
text FixCorruptTopoGeometryColumn(name layerSchema, name layerTable, name layerColumn );
```

Description

从 PostGIS topology 版本 `<3.6.0` 升级到 `>3.6.0+` 时，`topogeometry` 列的定义发生了变化，导致升级前建的 `topogeometry` 数据损坏。该函数用来修复受影响表中的某些损坏。

The function refuses to process tables with expression or partial indexes; drop or rebuild those indexes manually around the repair.

可用性：3.6.1

示例

修复所有 topology 列

Code

```
SELECT topology.FixCorruptTopoGeometryColumn(schema_name, table_name, feature_column)
FROM topology.layer;
```

相关信息

[UpgradeTopology](#)

9.5.7 Populate_Topology_Layer

Populate_Topology_Layer — 从拓扑表取元数据，将缺失的条目添加到 topology.layer 表中。

Synopsis

setof record **Populate_Topology_Layer**();

Description

从表上的拓扑约束将缺失的条目添加到 topology.layer 表中。此功能于在使用拓扑数据恢复模式后修复拓扑目中的条目非常有用。

它返回建的条目列表。返回的列是 schema_name、table_name、feature_column。

可用性：2.3.0

示例

Code

```
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');
```

Output

```
createtopology
```

```
-----
1
```

```
(1 row)
```

```
CREATE SCHEMA
CREATE TABLE
addtopogeometrycolumn
```

```
-----
1
```

```
(1 row)
```

This query returns no records because the feature is already registered.

| Code

```
SELECT *
FROM topology.Populate_Topology_Layer();
```

| Output

(0 rows)

Rebuild the topology layer catalog.

| Code

```
TRUNCATE TABLE topology.layer;
```

| Output

TRUNCATE TABLE

| Code

```
SELECT *
FROM topology.Populate_Topology_Layer();
```

| Output

schema_name	table_name	feature_column
-----+-----+-----		
strk	parcels	topo
(1 row)		

| Code

```
SELECT topology_id, layer_id,
       schema_name AS sn, table_name AS tn, feature_column AS fc
FROM topology.layer;
```

| Output

topology_id	layer_id	sn	tn	fc
-----+-----+-----+-----+-----				
1	1	strk	parcels	topo
(1 row)				

相关信息

AddTopoGeometryColumn

9.5.8 TopologySummary

TopologySummary — 取拓扑名称并提供拓扑中象型的数。

Synopsis

text TopologySummary(varchar topology_schema_name);

Description

返回拓扑名称并提供拓扑中每个象形的数量。

可用性: 2.0.0

示例

Code

```
SELECT topology.TopologySummary('city_data');
```

Output

```
Topology city_data (329), SRID 4326, precision: 0
22 nodes, 24 edges, 10 faces, 29 topogeoms in 5 layers
Layer 1, type Polygonal (3), 9 topogeoms
  Deploy: features.land_parcels.feature
Layer 2, type Puntal (1), 8 topogeoms
  Deploy: features.traffic_signs.feature
Layer 3, type Lineal (2), 8 topogeoms
  Deploy: features.city_streets.feature
Layer 4, type Polygonal (3), 3 topogeoms
  Hierarchy level 1, child layer 1
  Deploy: features.big_parcels.feature
Layer 5, type Puntal (1), 1 topogeoms
  Hierarchy level 1, child layer 2
  Deploy: features.big_signs.feature
```

9.5.9 ValidateTopology

ValidateTopology — 返回一个 validate_topology_returntype 象，指明拓扑。

Synopsis

```
setof validate_topology_returntype ValidateTopology(varchar toponame, geometry bbox);
```

Description

返回一个 validate_topology_returntype 象，指明拓扑，可以将结果限制到 bbox 参数指定的区域。下面显示了可能的结果列表、它们的含义以及返回的 id 代表的内容：

结果	id1	id2	意义
重合点	第一个点的标识符。	第二个点的标识符。	两个点具有相同的几何形状。
穿点	线的标识符。	点的标识符。	在其内部有一个点。 参 ST_Relate。
无效	线的标识符。		无效几何形。参 ST_IsValid。
不简单	线的标识符。		几何体具有自相交。参 ST_IsSimple。

错误	id1	id2	意义
与交叉	第一条的。	第二条的。	条有一个内部交叉点。 参 ST Relate 。
起始点几何形状不匹配	的。	指示的起始点的。	指示的起始点的几何形状与几何形状的第一个点不匹配。 参 ST StartPoint 。
端点几何形状不匹配	的。	指示的端点的。	指示的端点的几何形状与几何形状的最后一点不匹配。 参 ST EndPoint 。
无面	孤立面的。		没有告其存在有面 (left face、right face)。
face has no rings (面没有)	部分定义的面的。		在其面告面的不形成。
face has wrong mbr (面的最小界框)	mbr 存的面。		面的最小界矩形与告其面的面的集合的最小界框不匹配。
hole not in advertised face (孔不在宣的面内)	的姓名， 参 GetRingEdges 。		在其外部告一个面的包含在不同的面上。
非隔离点具有 not-contains_face	不明确的点的。		被告位于一个或多个界上的点表示一个包含的面。
孤立点有 contains_face	不明确的点的。		一个点如果没有被告位于任何界上，那就缺乏表示包含面的指示。
孤立点有 contains_face	表述点的。		一个未被告位于任何界上的点指示了一个包含它的面，并且个面不是包含它的面。参 GetFaceContainingPoint 。
无效的 next_right_edge	表示的的。	的有符号 ID，指示下一个右。	在沿着一条的右行走所指示的下一条是。
无效的 next_left_edge	表示的的。	的有符号 ID，指示下一个左。	在沿着一条的左行走所指示的下一条是。
mixed face labeling in ring (内混合面)	的姓名， 参 GetRingEdges 。		中的表示了行走的一存在冲突的面。也被称‘界位置冲突’。
非合	的姓名， 参 GetRingEdges 。		由以下 next left edge/next right edge 属性形成的在不同点上开始和束。
face has multiple shells (面有多个外壳)	有争的面的。	的姓名， 参 GetRingEdges 。	多个状界表示了同一个面的内部。

可用性：1.0.0

增：2.0.0 更有效的交叉并修复了先前版本中存在的。

更改：2.2.0 id1 和 id2 的被交叉“交叉点”，与描述一致。

更改：3.2.0 添加了可的 bbox 参数，行面和接。

示例

| **Code**

```
SELECT * FROM topology.ValidateTopology('ma_topo');
```

| **Output**

error	id1	id2
face without edges	1	

相关信息

[validate_topology_returntype](#)

9.5.10 ValidateTopologyRelation

ValidateTopologyRelation — 返回有关无效拓扑关系的信息

Synopsis

setof record **ValidateTopologyRelation**(varchar toponame);

Description

返回一行，提供有关拓扑关系表中无效性的信息。

可用性：3.2.0

相关信息

[ValidateTopology](#)

9.5.11 ValidateTopologyPrecision

ValidateTopologyPrecision — 返回拓扑中不精确的点。

Synopsis

geometry **ValidateTopologyPrecision**(name toponame, geometry bbox, float8 gridSize);

Description

返回所有未根据拓扑或指定的 gridSize 行整的点，以点状几何象形式表示，可地通 bbox 参数限制范围。

可用性：3.6.0

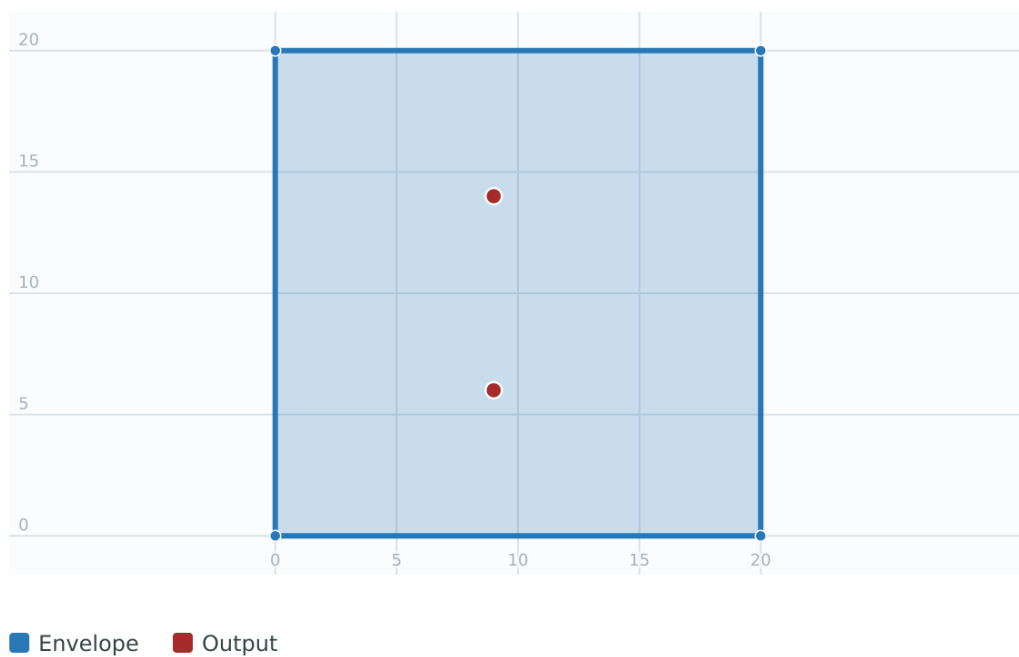
示例

Code

```
SELECT g
FROM topology.ValidateTopologyPrecision(
    'city_data',
    gridSize =
> 2,
    bbox =
> ST_MakeEnvelope(0, 0, 20, 20)
) g;
```

Output

```
MULTIPOINT(9 6,9 14)
```

Figure

相关信息

MakeTopologyPrecise**9.5.12 MakeTopologyPrecise**

MakeTopologyPrecise — 将拓扑点集到精度网格。

Synopsis

```
void MakeTopologyPrecise(name toponame, geometry bbox, float8 gridSize);
```

Description

将拓扑中的所有点移动到拓扑精度网格，或移动到由 `gridSize` 参数指定大小的网格，可地将操作限制在与 `bbox` 参数指定区域相交的象限内。

Using a `gridSize` larger than the smallest edge extent raises an exception rather than collapsing the edge into an empty geometry. This safeguards adjacent topological relationships by preventing precision snapping from removing primitives altogether.



Note

操作可能导致拓扑无效，因此建议使用 `ValidateTopology` 操作。

可用性：3.6.0

示例

Code

```
SELECT topology.MakeTopologyPrecise(
    'city_data',
    gridSize =
    > 2
);
```

相关信息

[ValidateTopologyPrecision](#), [ValidateTopology](#)

9.5.13 FindTopology

`FindTopology` — 通过不同的方式返回拓扑。

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);
```

Description

获取拓扑符或拓扑相关象的符并返回 `topology.topology`。

可用性：3.2.0

示例

Code

```
SELECT name(FindTopology('features.land_parcels', 'feature'));
```

Output

```
city_data
```

相关信息

[FindLayer](#)

9.5.14 FindLayer

FindLayer — 通过不同的方式返回 topology.layer 值。

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);
```

Description

返回与指定几何体或拓扑相关图形的 ID 并返回 topology.layer 值。

可用性：3.2.0

示例

Code

```
SELECT layer_id(FindLayer('features.land_parcels', 'feature'));
```

Output

```
1
```

相关信息

[FindTopology](#)

9.5.15 TotalTopologySize

TotalTopologySize — 指定拓扑占用的磁盘空间，包括所有索引和 TOAST 数据。

Synopsis

```
int8 TotalTopologySize(name toponame);
```

Description

接受一个拓扑名称，返回其所有表（包括索引和 TOAST 数据）占用的磁盘空间。

可用性：3.6.0

示例

Code

```
SELECT topology.TopologySummary('city_data');
```

Output

```
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
```

9.5.16 UpgradeTopology

UpgradeTopology — 将指定拓扑升级到支持大整数（int8）类型的拓扑 ID 和要素 ID。

Synopsis

```
void UpgradeTopology(name toponame);
```

Description

接受一个拓扑名称，并将其升级到支持大 ID（int8）的模式，适用于拓扑 ID 和各种基要素 ID。升级内容包括：将 face 表的 face_id 和 face_id_seq 从 int4 升级到 int8；将 node 表的 node_id、containing_face 以及 node_id_seq 从 int4 升级到 int8；将 edge_data 表中的 edge_id、start_node、end_node、left_face、right_face、next_left_edge、next_right_edge 以及序列 edge_data_edge_id_seq 全部从 int4 升级到 int8；将 relation 表中的 topogeo_id 和 element_id 从 int4 升级到 int8；最后在 topology 表中把 useslargeids 设置为 true，表示启用大 ID 支持。

可用性：3.6.0

示例

I Code

```
SELECT topology.UpgradeTopology('city_data');
```

9.6 拓扑管理

向拓扑添加元素会触发多数据刷新，以查找将被分割的边、添加点并更新将与新边路接的边。因此，有关拓扑表中数据的刷新信息是最新的非常有用。

PostGIS 拓扑数据的填充和刷新函数不会自动更新刷新信息，因而在每次拓扑更改之后都更新刷新信息会得于冗余，所以更新刷新信息是用者的责任。



Note

由 autovacuum 更新的刷新信息在于在 autovacuum 进程完成之前刷新的事不可，因此刷新刷新行的事将需要自行运行 ANALYZE，以使用更新的刷新信息。

9.7 拓扑构造器

9.7.1 CreateTopology

CreateTopology — 建立一个新的拓扑架并将其注册到 topology.topology 表中。

Synopsis

integer **CreateTopology**(name topology_schema_name, integer srid, double precision prec=0, boolean hasz=false, integer topoid=0, boolean useslargeids=false);

Description

创建名为 topology_name 的新拓扑模式并将其注册到 topology.topology 表中。拓扑必须具有唯一的名称。拓扑表 (edge_data、face、node 和 relation) 在模式中创建。它返回拓扑的 id。

srid 表示拓扑的空参考系的 SRID。如果未指定，默认为 -1 (未知)。

容差精度以空参考系的单位行量。容差默认为 0。

如果未指定，hasz 默认为 false。

topoid 可为的式符号 (允许确定性地分配拓扑 ID，需保证唯一)

useslargeids 可为参数，默认为 false。若为 true，创建的拓扑将支持大整数 (int8) 型的拓扑 ID 和要素 ID。

与 SQL/MM ST_InitTopoGeo 类似，但具有更多功能。

可用性：1.1

Enhanced: 2.0 新增 hasZ 的支持。

Enhanced: 3.6.0 新增 topoid 和 usesLargeIDs 参数。

示例

建立一个名为 ma_topo 的拓扑模式，用于存储墨西哥塞州平面米 (SRID = 26986) 中的边和点。由于空参考系是基于米的，因此容差表示 0.5 米。

Code

```
SELECT topology.CreateTopology('ma_topo', 26986, 0.5);
```

在空参考系 State Plane-feet (SRID = 3438) 中创建一个名 `ri_topo` 的拓扑

Code

```
SELECT topology.CreateTopology('ri_topo', 3438) AS topoid;
```

Output

```
2
```

相关信息

Section 4.5, [ST_InitTopoGeo](#)

9.7.2 CopyTopology

`CopyTopology` — 将拓扑（点、面、线和拓扑几何）的副本复制到新模式中

Synopsis

integer **CopyTopology**(varchar existing_topology_name, varchar new_name);

Description

创建名 `new_name` 的新拓扑,并使用从 `existing_topology_name` 复制的 SRID 和精度。`existing_topology_name` 中的点、面和线以及其关联的 `TopoGeometries` 都将复制到新拓扑中。

**Note**

`topology.layer` 表中的新行包含 `schema_name`、`table_name` 和 `feature_column` 的合成。它是 `TopoGeometry` 对象作定存在，并且在用定的表中尚不可用。

可用性: 2.0.0

示例

备份名 `ma_topo` 的拓扑。

Code

```
SELECT topology.CopyTopology('ma_topo', 'ma_topo_backup');
```

相关信息

Section 4.5, [CreateTopology](#), [RenameTopology](#)

9.7.3 ST_InitTopoGeo

ST_InitTopoGeo — 创建一个新的拓扑架并将其注册到 topology.topology 表中。

Synopsis

```
text ST_InitTopoGeo(varchar topology_schema_name);
```

Description

在 SQL-MM 中相当于 **CreateTopology**。它缺乏空参考系和容差的。它返回拓扑建的文本描述，而不是拓扑 ID。

可用性：1.1

 方法符合了 SQL/MM 规范。SQL-MM 3 Topo-Geo 和 Topo-Net 3：例程信息：X.3.17

示例

Code

```
SELECT topology.ST_InitTopoGeo('topo_schema_to_create') AS topocreation;
```

Output

```
Topology-Geometry 'topo_schema_to_create' (id:7) created.
```

相关信息

CreateTopology

9.7.4 ST_CreateTopoGeo

ST_CreateTopoGeo — 将几何形集合添加到指定的空拓扑并返回明确成功的消息。

Synopsis

```
text ST_CreateTopoGeo(varchar atopology, geometry acollection);
```

Description

将几何形集合添加到指定的空拓扑并返回明确成功的消息。

于填充空拓扑很有用。

可用性：2.0

 方法符合了 SQL/MM 规范。SQL-MM: Topo-Geo 和 Topo-Net 3: 例程信息 -- X.3.18

示例

Populate the topology.

Code

```
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)
  );
```

Output

Topology ri_topo populated

Create the tables and topogeometries.

Code

```
CREATE TABLE ri.roads(gid serial PRIMARY KEY, road_name text);
SELECT topology.AddTopoGeometryColumn('ri_topo', 'ri', 'roads', 'topo', 'LINE');
```

相关信息

[TopoGeo_LoadGeometry](#), [AddTopoGeometryColumn](#), [CreateTopology](#), [DropTopology](#)

9.7.5 TopoGeo_AddPoint

TopoGeo_AddPoint — 使用容差并可能分割有向有拓扑添加点。

Synopsis

bigint **TopoGeo_AddPoint**(varchar atopology, geometry apoint, float8 tolerance);

Description

向有拓扑添加一个点并返回其标识符。指定点将捕捉到指定容差内的有向点或线。有的线可能会被捕捉点分割。
可用性: 2.0.0

相关信息

[TopoGeo_AddLineString](#), [TopoGeo_AddPolygon](#), [TopoGeo_LoadGeometry](#), [AddNode](#), [CreateTopology](#)

9.7.6 TopoGeo_AddLineString

TopoGeo_AddLineString — 使用容差将线串添加到有拓扑中，并在可能的情况下拆分线或面。

Synopsis

SETOF bigint **TopoGeo_AddLineString**(varchar atopology, geometry aline, float8 tolerance, int max_edges)

Description

向已有拓扑中添加一条 linestring，并返回一个或多个有符号 edge 符号；其中数表示 edge 的方向与输入的 linestring 相反。

指定的线会在指定容差范围内自线靠近到已有的点或线上；有的线和面可能会被多条线切分。除了新增线之外，过程中可能会新增点和面。

返回的 edge 符号可能是已有的，也可能是新建的，具体取决于为了尽可能接近表示输入线所需的情况。

新建的 edge 数量（无论是覆盖原本没有覆盖的空隙，还是由切分线产生）都可能会受到 max_edges 参数的限制。



Note

When adding many lines, the load order can have a large impact on performance because each call checks the line against topology elements already loaded. In bulk workflows, adding large closed or enclosing lines before smaller contained lines can avoid expensive later splits. For loading a collection in one call, see [TopoGeo_LoadGeometry](#).



Note

更新有关拓扑此函数添加的拓扑的统计信息由用户决定，参看 [maintaining statistics during topology editing and population](#) 在拓扑和填充期统计信息。

可用性: 2.0.0

增修: 3.2.0 版本新增了返回符号化符号的功能。

Enhanced: 3.7.0 增加限制拓扑中新建 edge 数量的支持。

相关信息

[TopoGeo_AddPoint](#), [TopoGeo_AddPolygon](#), [TopoGeo_LoadGeometry](#), [AddEdge](#), [CreateTopology](#)

9.7.7 TopoGeo_AddPolygon

TopoGeo_AddPolygon — 使用公差并可能分割线/面将多边形添加到有拓扑。返回面符号。

Synopsis

SETOF bigint **TopoGeo_AddPolygon**(varchar atopology, geometry apoly, float8 tolerance);

Description

将多边形添加到具有拓扑并返回成为它的一个面。指定的多边形的边界将捕捉到指定容差内的点或面。点和面可能会被新多边形的边界分割。



Note

更新有关通此函数加的拓扑的信息由用户决定，参 [maintaining statistics during topology editing and population](#) 在拓扑和填充期信息。

可用性: 2.0.0

相关信息

[TopoGeo_AddPoint](#), [TopoGeo_AddLineString](#), [TopoGeo_LoadGeometry](#), [AddFace](#), [CreateTopology](#)

9.7.8 TopoGeo_LoadGeometry

`TopoGeo_LoadGeometry` — 将几何形加到具有拓扑中，并在必要时进行捕捉和拆分操作。

Synopsis

```
void TopoGeo_LoadGeometry(varchar atopology, geometry ageom, float8 tolerance);
```

Description

将几何形加到具有拓扑中。指定的几何形将在指定的容差范围内捕捉到点或面。点和面可能会因加操作而被拆分。



Note

更新有关通此函数加的拓扑的信息由用户决定，参 [maintaining statistics during topology editing and population](#) 在拓扑和填充期信息。

可用性 : 3.5.0

相关信息

[TopoGeo_AddPoint](#), [TopoGeo_AddLineString](#), [TopoGeo_AddPolygon](#), [CreateTopology](#)

9.8 拓扑器

9.8.1 ST_AddIsoNode

`ST_AddIsoNode` — 将一个孤立的点添加到拓扑中的一个面，并返回新点的点 ID。如果“face” 空 (null)，仍然会建点。

Synopsis

bigint **ST_AddIsoNode**(varchar atopology, bigint aface, geometry apoint);

Description

将具有点位置指定的孤立点 `apoint` 添加到具有 `aface` 的面到拓扑 `atopology` 拓扑并返回新点的 `nodeid`。

如果点几何的空参考系 (`srid`) 与拓扑不同、`apoint` 不是点几何、点空或点与面相交（即使在边界），例外情况是抛出。如果点已作点存在，会引异常。

如果 `aface` 不为 `null` 并且 `apoint` 不在面内，抛出异常。

可用性：1.1

 方法符合了 SQL/MM 范。SQL-MM: Topo-Net 例程: X+1.3.1

示例

相关信息

[AddNode](#), [CreateTopology](#), [DropTopology](#), [ST_Intersects](#)

9.8.2 ST_AddIsoEdge

`ST_AddIsoEdge` — 将由几何 `alinesstring` 定义的孤立添加到接个有孤立点 `anode` 和 `anothernode` 的拓扑，并返回新的 `id`。

Synopsis

bigint **ST_AddIsoEdge**(varchar atopology, bigint anode, bigint anothernode, geometry alinesstring);

Description

将由几何 `alinesstring` 定义的孤立添加到接个有孤立点 `anode` 和 `anothernode` 的拓扑，并返回新的 `ID`。

如果 `alinesstring` 几何的空参考系 (`SRID`) 与拓扑不同、不是点几何、点 `NULL` 或点与面（包括其边界）交互，会引异常。此外，如果点已作点存在，会引异常。

如果 `alinesstring` 不在 `anode` 和 `anothernode` 所属的面内，那么将抛出一个异常。

如果 `anode` 和 `anothernode` 不是 `alinesstring` 的起始点和束点，会抛出异常。

可用性：1.1

 方法符合了 SQL/MM 范。SQL-MM: Topo-Geo 和 Topo-Net 3: 例程: X.3.4

示例

相关信息

[ST_AddIsoNode](#), [ST_IsSimple](#), [ST_Within](#)

9.8.3 ST_AddEdgeNewFaces

ST_AddEdgeNewFaces — 添加新面，如果面做会分割一个面，删除原始面并用个新面替它。

Synopsis

```
bigint ST_AddEdgeNewFaces(varchar atopology, bigint anode, bigint anothernode, geometry acurve);
```

Description

添加新面，如果面做会分割一个面，删除原始面并用个新面替它。返回新添加的面的 id。

相地更新所有有的接口和关系。

如果任何参数为 null，指定的点未知（必已存在于拓扑模式的 node 表中），acurve 不是 LINESTRING, anode 和 anothernode 不是 acurve 的起始点和点，会引。

如果 acurve 几何对象的空参考系（SRID）与拓扑不同，那么将抛出异常。

可用性：2.0



方法了 SQL/MM 范。SQL-MM: Topo-Geo 和 Topo-Net 3: 例程: X.3.12

示例

相关信息

[ST_RemEdgeNewFace](#)

[ST_AddEdgeModFace](#)

9.8.4 ST_AddEdgeModFace

ST_AddEdgeModFace — 添加新面，如果面做会分割面，修改原始面并添加新面。

Synopsis

```
bigint ST_AddEdgeModFace(varchar atopology, bigint anode, bigint anothernode, geometry acurve);
```

Description

添加新面，如果面做会分割面，修改原始面并添加新面。



Note

如果可能，新面将建在新面的左。如果左的面需要是宇宙面（无界），是不可能的。

返回新添加的面的 id。

相应地更新所有有的边和关系。

如果任何参数为 null，指定的点未知（必须已存在于拓扑模式的 node 表中），acurve 不是 LINESTRING, anode 和 anothernode 不是 acurve 的起始点和点，会引发错误。

如果 acurve 几何对象的空参考系（SRID）与拓扑不同，那么将抛出异常。

可用性：2.0



方法符合了 SQL/MM 规范。SQL-MM: Topo-Geo 和 Topo-Net 3: 例程: X.3.13

示例

相关信息

[ST_RemEdgeModFace](#)

[ST_AddEdgeNewFaces](#)

9.8.5 ST_RemEdgeNewFace

ST_RemEdgeNewFace — 删除一条边，如果删除的边将面分开，删除原始面并用新面替换它。

Synopsis

```
bigint ST_RemEdgeNewFace(varchar atopolology, bigint anedge);
```

Description

删除一条边，如果删除的边将面分开，删除原始面并用新面替换它。

返回一个新建的面的 ID，或者返回 NULL，如果没有建新的面。当被移除的边是悬挂的、孤立的，或者与宇宙面相邻（可能致宇宙面涌入一面的面），不会建新的面。

相应地更新所有有的边和关系。

拒绝移除参与有 TopoGeometry 定义的边。如果有任何 TopoGeometry 只由其中一个面定义而不是一个面，拒绝合并面。

如果任何参数为 null，指定的边是未知的（必须已存在于拓扑模式的 edge 表中），拓扑名称无效，会抛出错误。

可用性：2.0



方法符合了 SQL/MM 规范。SQL-MM: Topo-Geo 和 Topo-Net 3: 例程: X.3.14

示例

相关信息

[ST_RemEdgeModFace](#)

[ST_AddEdgeNewFaces](#)

9.8.6 ST_RemEdgeModFace

ST_RemEdgeModFace — 删除一条边，如果边将面分开，删除一个面并修改一个面以覆盖面的空间。

Synopsis

```
bigint ST_RemEdgeModFace(varchar atopology, bigint anedge);
```

Description

删除一条边，如果删除的边将面分开，删除一个面并修改一个面以覆盖面的空间。先将面保持在右面，以与**ST_AddEdgeModFace**保持一致。返回保留的面的 id。

相应地更新所有有的边接和关系。

移除参与有 TopoGeometry 定义的边。如果有任何 TopoGeometry 只由其中一个面定义而不是一个面，拒绝合并面。

如果任何参数为 null，定义的边是未知的（必须已存在于拓扑模式的 edge 表中），拓扑名称无效，会抛出异常。
可用性：2.0



方法符合了 SQL/MM 规范。SQL-MM: Topo-Geo 和 Topo-Net 3: 规范例程: X.3.15

示例

相关信息

[ST_AddEdgeModFace](#)

[ST_RemEdgeNewFace](#)

9.8.7 ST_ChangeEdgeGeom

ST_ChangeEdgeGeom — 改变边的形状而不影响拓扑。

Synopsis

```
text ST_ChangeEdgeGeom(varchar atopology, bigint anedge, geometry acurve);
```

Description

改变边的形状而不影响拓扑。

如果任何参数为 null，定义的边在拓扑模式的 edge 表中不存在，acurve 不是 LINESTRING，或者修改将改变底拓扑，会抛出异常。

如果 acurve 几何对象的空参考系 (SRID) 与拓扑不同，那么将抛出异常。

如果新的 acurve 不是简单线 (simple line)，会抛出异常。

如果将边从旧位置移到新位置会碰到障碍物，那么会抛出异常。

可用性：1.1.0

增版 2.0.0 添加了拓扑一致性限制行功能



方法符合了 SQL/MM 规范。SQL-MM: Topo-Geo 和 Topo-Net 3: 规范例程 X.3.6

示例

Code

```
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)
);
```

Output

Edge 1 changed

相关信息

[ST_AddEdgeModFace](#)

[ST_RemEdgeModFace](#)

[ST_ModEdgeSplit](#)

9.8.8 ST_ModEdgeSplit

ST_ModEdgeSplit — 通过沿边有边建新点、修改原始边并添加新边来分割边。

Synopsis

bigint **ST_ModEdgeSplit**(varchar atopology, bigint anedge, geometry apoint);

Description

通过沿边有边建新点、修改原始边并添加新边来分割边。相应地更新所有有的边接边和关系。返回新添加的点的标识符。

可用性：1.1

更改：2.0 - 在之前的版本中，它被命名 ST_ModEdgesSplit



该方法符合了 SQL/MM 规范。SQL-MM: Topo-Geo 和 Topo-Net 3: 规范例程: X.3.9

示例

Add an edge to split.

Code

```
SELECT topology.AddEdge('ma_topo', ST_GeomFromText('LINESTRING(227592 893910,227600 893910)',
  26986)) AS edgeid;
```

| Output

3

Split the edge with a new node.

| Code

```
SELECT topology.ST_ModEdgeSplit('ma_topo', 3, ST_SetSRID(ST_Point(227594, 893910), 26986)) ↔  
AS node_id;
```

| Output

7

相关信息

[ST_NewEdgesSplit](#), [ST_ModEdgeHeal](#), [ST_NewEdgeHeal](#), [AddEdge](#)

9.8.9 ST_ModEdgeHeal

`ST_ModEdgeHeal` — 通过删除接条的点和、修改第一条并删除第二条来修复条。返回已删除点的 id。

Synopsis

`bigint ST_ModEdgeHeal`(varchar atopology, bigint anedge, bigint anotheredge);

Description

通过删除接条的点和、修改第一条并删除第二条来修复条。返回已删除点的 id。相应地更新所有有的接和关系。

可用性：2.0



方法符合了 SQL/MM 规范。SQL-MM: Topo-Geo 和 Topo-Net 3: 规范例程: X.3.9

相关信息

[ST_ModEdgeSplit](#) [ST_NewEdgesSplit](#)

9.8.10 ST_NewEdgeHeal

`ST_NewEdgeHeal` — 通过删除接条的点和、删除条并用方向与提供的第一条相同的替它来修复条。

Synopsis

`bigint ST_NewEdgeHeal`(varchar atopology, bigint anedge, bigint anotheredge);

Description

通过删除接条的点、删除条并用方向与提供的第一条相同的替换它来修复条。返回替换已修复的新条的 ID。相应地更新所有有的接和关系。

可用性：2.0



方法符合了 SQL/MM 规范。SQL-MM: Topo-Geo 和 Topo-Net 3: 例程: X.3.9

相关信息

[ST_ModEdgeHeal](#) [ST_ModEdgeSplit](#) [ST_NewEdgesSplit](#)

9.8.11 ST_MoveIsoNode

ST_MoveIsoNode — 在拓扑中将一个孤立点从一个点移到另一个点。如果新的 `apoint` 几何对象已存在作一个点，会抛出异常。返回移的描述。

Synopsis

text **ST_MoveIsoNode**(varchar atopology, bigint anode, geometry apoint);

Description

将拓扑中的孤立点从一点移到一点。如果新的 `apoint` 几何形状作点存在，会抛出异常。

如果任何参数 `null`，`apoint` 不是一个点，有点不是孤立的（是有点的起点或点），新点位置与有相交（即使在端点也相交），或新位置在不同的面（自 3.2.0 版本起），会抛出异常。

如果点几何的空参考系 (`srid`) 与拓扑不同，会引发异常。

可用性: 2.0.0

增：3.2.0 确保点不能移到不同的面



方法符合了 SQL/MM 规范。SQL-MM: Topo-Net 例程: X.3.2

示例

Add an isolated node with no face.

Code

```
SELECT topology.ST_AddIsoNode('ma_topo', NULL, ST_GeomFromText('POINT(227579 893916)', 26986)) AS nodeid;
```

Output

```
7
```

Move the new node.

Code

```
SELECT topology.ST_MoveIsoNode('ma_topo', 7, ST_GeomFromText('POINT(227579.5 893916.5)', 26986)) AS descrip;
```

Output

```
Isolated Node 7 moved to location 227579.5,893916.5
```

相关信息

[ST_AddIsoNode](#)

9.8.12 ST_NewEdgesSplit

ST_NewEdgesSplit — 通过沿边有边新建点、删除原始边并用新边替换它来分割边。返回新建的边接新的新点的 id。

Synopsis

bigint ST_NewEdgesSplit(varchar atopology, bigint anedge, geometry apoint);

Description

通过在当前边上建立一个具有点位置 **apoint** 的新点，删除原始边并用新边替换它，来分割具有 ID **anedge** 的边。返回边接新的新点的 ID。相应地更新所有已连接的边和关系。

如果点几何对象的空参考系（SRID）与拓扑不同，**apoint** 不是点几何对象，点 **null**，点已存在作一个点，边与边有边不边，或者点不在边上，边会抛出异常。

可用性：1.1



该方法符合了 SQL/MM 规范。SQL-MM: Topo-Net 例程: X.3.8

示例

Add an edge to split.

Code

```
SELECT topology.AddEdge('ma_topo', ST_GeomFromText('LINESTRING(227575 893917,227592 893900)', 26986)) AS edgeid;
```

Output

```
2
```

Split the new edge.

Code

```
SELECT topology.ST_NewEdgesSplit('ma_topo', 2, ST_GeomFromText('POINT(227578.5 893913.5)', 26986)) AS newnodeid;
```

Output

```
6
```

相关信息

[ST_ModEdgeSplit](#) [ST_ModEdgeHeal](#) [ST_NewEdgeHeal](#) [AddEdge](#)

9.8.13 ST_RemoveIsoNode

`ST_RemoveIsoNode` — 移除孤立点并返回操作描述。如果点不是孤立的（是边的开始或结束），会引发异常。

Synopsis

text `ST_RemoveIsoNode`(varchar atopology, bigint anode);

Description

移除一个孤立点并返回操作的描述。如果点不是孤立的（是边的起点或点），会抛出异常。

可用性：1.1



方法符合了 SQL/MM 规范。SQL-MM: Topo-Geo 和 Topo-Net 3: 例程: X+1.3.3

示例

Remove an isolated node with no face.

Code

```
SELECT topology.ST_RemoveIsoNode('ma_topo', 7) AS result;
```

Output

```
Isolated node 7 removed
```

相关信息

[ST_AddIsoNode](#)

9.8.14 ST_RemoveIsoEdge

`ST_RemoveIsoEdge` — 移除孤立的边并返回操作的描述。如果边未被隔离，会引发异常。

Synopsis

text **ST_RemoveIsoEdge**(varchar atopology, bigint anedge);

Description

移除一个孤立的边并返回操作的描述。如果边不是孤立的，将抛出异常。

可用性：1.1

 方法实现了 SQL/MM 规范。SQL-MM: Topo-Geo 和 Topo-Net 3: 规范例程: X+1.3.3

示例

Remove an isolated edge with no face.

Code

```
SELECT topology.ST_RemoveIsoEdge('ma_topo', 7) AS result;
```

Output

```
Isolated edge 7 removed
```

相关信息

[ST_AddIsoNode](#)

9.9 拓扑处理器

9.9.1 GetEdgeByPoint

GetEdgeByPoint — 找与点相交的边的 ID。

Synopsis

bigint **GetEdgeByPoint**(varchar atopology, geometry apoint, float8 tol1);

Description

边与 Point 相交的边的 id。

给定拓扑、POINT 和容差，函数返回一个整数 (id-edge)。如果容差 = 0，点必与边相交。

如果 apoint 与边不相交，返回 0。

如果使用的容差 (tolerance) 大于 0，并且在某点附近存在多条边，会引发异常。



Note

如果 tolerance = 0，函数使用 ST_Intersects，否则使用 ST_DWithin。

该函数是由 GEOS 模块实现的。

可用性：2.0.0

示例

一些示例使用我在 [AddEdge](#) 中建的

Code

```
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;
```

Output

with1mtol	withnotol
-----+-----	
2	0

This query errors because two or more edges are found.

Code

```
SELECT topology.GetEdgeByPoint('ma_topo', geom, 1) As nearnode
FROM ST_GeomFromEWKT('SRID=26986;POINT(227591.9 893900.4)') As geom;
```

Output

```
ERROR: Two or more edges found
```

相关信息

[AddEdge](#), [GetNodeByPoint](#), [GetFaceByPoint](#)

9.9.2 GetFaceByPoint

GetFaceByPoint — 找与点相交的面。

Synopsis

bigint **GetFaceByPoint**(varchar atopology, geometry apoint, float8 tol1);

Description

找由点引用且具有定容差的面。
函数将有效地找与以点中心、以公差半径的面相交的面。
如果没有面与定位置相交，返回 0（通用面）。
如果多个面与位置相交，会引异常。
可用性: 2.0.0
增: 3.2.0 更高效的施和更清晰的契，停止使用无效的拓扑。

示例

Code

```
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;
```

Output

with1mtol	withnotol
-----+-----	
1	0

This query errors because two or more faces are found.

Code

```
SELECT topology.GetFaceByPoint('ma_topo', geom, 1) As nearnode
FROM ST_GeomFromEWKT('POINT(227591.9 893900.4)') As geom;
```

Output

ERROR: Two or more faces found

相关信息

[GetFaceContainingPoint](#), [AddFace](#), [GetNodeByPoint](#), [GetEdgeByPoint](#)

9.9.3 GetFaceContainingPoint

GetFaceContainingPoint — 找包含点的面。

Synopsis

bigint **GetFaceContainingPoint**(text atopology, geometry apoint);

Description

返回包含点的面的 id。

如果点落在面边界上，会引发异常。



Note

函数依赖于有效的拓扑，使用接口和面。

可用性 : 3.2.0

相关信息

[ST_GetFaceGeometry](#)

9.9.4 GetNodeByPoint

GetNodeByPoint — 找某个点位置点的点 ID。

Synopsis

```
bigint GetNodeByPoint(varchar atopology, geometry apoint, float8 tol1);
```

Description

索某个点位置的点 id。

定拓扑 (topology)、POINT 和容差 (tolerance)，函数返回一个整数 (id-node)。如果公差 = 0 表示精确交集，否则从区中索点。

如果 apoint 不与点相交，返回 0 (零)。

如果置容差 (tolerance) 大于 0，并且在某点附近存在多个点 (node)，那么可能会引异常。



Note

如果 tolerance = 0，函数使用 ST_Intersects，否则使用 ST_DWithin。

个函数是由 GEOS 模行的。

可用性: 2.0.0

示例

些示例使用我在 [AddEdge](#) 中建的

Code

```
SELECT topology.GetNodeByPoint('ma_topo', geom, 1) As nearnode
FROM ST_GeomFromEWKT('SRID=26986;POINT(227591.9 893900.4)') As geom;
```

Output

```
2
```

This query errors because the tolerance matches two or more nodes.

Code

```
SELECT topology.GetNodeByPoint('ma_topo', geom, 1000) As too_much_tolerance
FROM ST_GeomFromEWKT('SRID=26986;POINT(227591.9 893900.4)') As geom;
```

Output

```
ERROR: Two or more nodes found
```

相关信息

[AddEdge](#), [GetEdgeByPoint](#), [GetFaceByPoint](#)

9.9.5 GetTopologyID

GetTopologyID — 返回 topology.topology 表中指定拓扑名称的拓扑的 id。

Synopsis

integer **GetTopologyID**(varchar toponame);

Description

返回 topology.topology 表中指定拓扑名称的拓扑的 id。

可用性：1.1

示例

Code

```
SELECT topology.GetTopologyID('ma_topo') AS topo_id;
```

Output

```
1
```

相关信息

[CreateTopology](#), [DropTopology](#), [GetTopologyName](#), [GetTopologySRID](#)

9.9.6 GetTopologySRID

GetTopologySRID — 返回拓扑表中指定拓扑名称的拓扑的 SRID。

Synopsis

integer **GetTopologyID**(varchar toponame);

Description

在指定拓扑名称的情况下，返回 topology.topology 表中拓扑的空参考 ID。

可用性：2.0.0

示例

Code

```
SELECT topology.GetTopologySRID('ma_topo') AS srid;
```

Output

```
4326
```

相关信息

[CreateTopology](#), [DropTopology](#), [GetTopologyName](#), [GetTopologyID](#)

9.9.7 GetTopologyName

GetTopologyName — 返回指定拓扑 ID 的拓扑（架）名称。

Synopsis

varchar **GetTopologyName**(integer topology_id);

Description

在指定拓扑的拓扑 ID 的情况下，从 topology.topology 表中返回拓扑的拓扑名称（架）。

可用性：1.1

示例

Code

```
SELECT topology.GetTopologyName(1) AS topo_name;
```

Output

```
ma_topo
```

相关信息

[CreateTopology](#), [DropTopology](#), [GetTopologyID](#), [GetTopologySRID](#)

9.9.8 ST_GetFaceEdges

ST_GetFaceEdges — 返回一个有序的边，这些边界定了 aface。

Synopsis

getfaceedges_returntype **ST_GetFaceEdges**(varchar atopology, bigint aface);

Description

返回一有序的行，行些行界定了 `aface`。每个行出包括一个序列号和行 ID。序列号从 1 开始。每个行的枚行从具有最小行符的行开始。行的行序遵循左手定行（合界面位于每个有向行的左行）。
可用性：2.0

 行方法行了 SQL/MM 行范。SQL-MM 3 Topo-Geo 和 Topo-Net 3: 行行例程: X.3.5

示例

This query returns the edges that bound face 1.

Code

```
SELECT (topology.ST_GetFaceEdges('tt', 1)).*;
```

Output

sequence	edge
1	-4
2	5
3	7
4	-6
5	1
6	2
7	3

This query returns the sequence, edge identifier, and geometry of the edges that bound face 1. Use [ST_GetFaceGeometry](#) if only the geometry and sequence are needed.

Code

```
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;
```

相关信息

[GetRingEdges](#), [AddFace](#), [ST_GetFaceGeometry](#)

9.9.9 ST_GetFaceGeometry

`ST_GetFaceGeometry` — 返回定拓扑中具有指定面 ID 的多行形。

Synopsis

geometry **ST_GetFaceGeometry**(varchar atopology, bigint aface);

Description

返回给定拓扑中具有指定面 ID 的多边形。从面成面的拓扑构建多边形。

可用性：1.1

✓ 该方法符合了 SQL/MM 规范。SQL-MM 3 Topo-Geo 和 Topo-Net 3: 规范例程: X.3.16

示例

This query returns the WKT of the polygon added with **AddFace**.

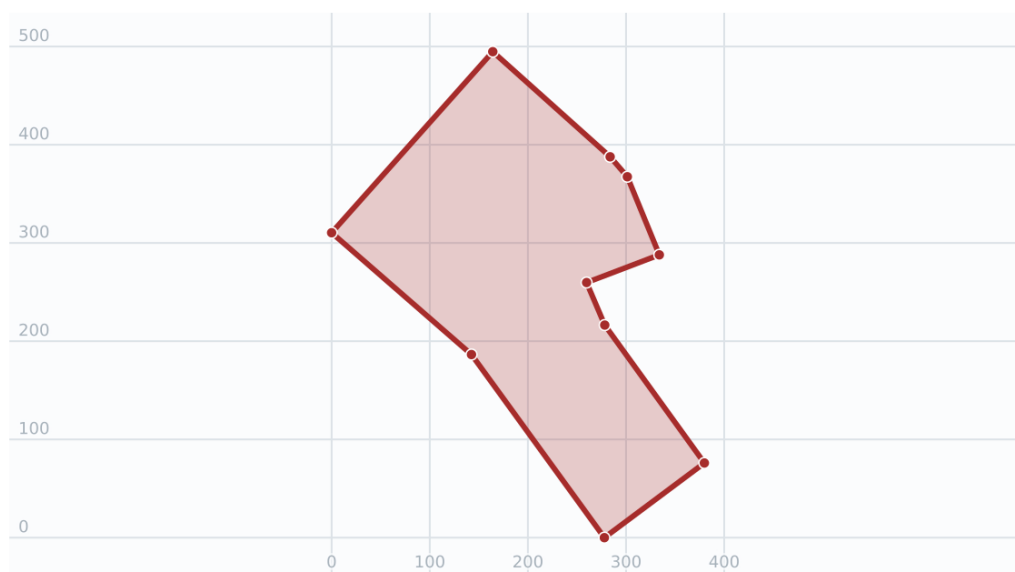
Code

```
SELECT topology.ST_GetFaceGeometry('ma_topo', 1) AS facegeomwkt;
```

Output

```
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))
```

Figure



■ Output

相关信息

AddFace

9.9.10 GetRingEdges

GetRingEdges — 返回按顺序排列的边符集合，这些边符是通沿给定的一行走遇到的。

Synopsis

```
getfaceedges_returntype GetRingEdges(varchar atopology, bigint aring, integer max_edges=null);
```

Description

返回在给定环上行走时遇到的有符号边的有序集合。每个边由一个序列和一个带符号的 id 组成。序列号从 1 开始。

如果指定 id，行走从相应边的左端开始，并遵循该方向。如果您指定 ID，行走从其右端开始并向后走。

如果 max_edges 不为 null，函数返回的边数不超过该值。在处理可能无效的拓扑时，它是一个安全参数。



Note

该函数使用元数据。

可用性: 2.0.0

相关信息

[ST_GetFaceEdges](#), [GetNodeEdges](#)

9.9.11 GetNodeEdges

GetNodeEdges — 返回与指定点相关的一组有序边。

Synopsis

```
getfaceedges_returntype GetNodeEdges(varchar atopology, bigint anode);
```

Description

返回与指定点相关的一组有序边。每个边由一个序列和一个带符号的 id 组成。序列号从 1 开始。上升沿从指定点开始。边沿端于指定点。封闭的边将出边次（都有个标志）。顺序是从北行开始的方向。



Note

该函数按排序而不是从元数据出，因此可用于构建索引。

可用性 : 2.0

相关信息

[getfaceedges_returntype](#), [GetRingEdges](#), [ST_Azimuth](#)

9.9.12 FindVertexSegmentPairsBelowDistance

FindVertexSegmentPairsBelowDistance — Find pairs of topology vertex/segment that are closer than tolerated distance

Synopsis

```
setof record FindVertexSegmentPairsBelowDistance(varchar atopology, float8 tolerance);
```

Description

Returns a relation in which each tuple is a pair of vertex/segment where the vertex is NOT an endpoint of the segment but the distance between them is below the given tolerance.

Availability: 3.7

相关信息

[ValidateTopologyPrecision](#)

9.10 拓扑处理

9.10.1 Polygonize

Polygonize — 找并注册由拓扑定义的所有面。

Synopsis

```
text Polygonize(varchar toponame);
```

Description

注册可以建拓扑基元的所有面。

假目拓扑不包含自相交。



Note

已找出已知的面，因此在同一拓扑上多次用 Polygonize 是安全的。



Note

函数不使用也不置表的 next_left_edge 和 next_right_edge 字段。

可用性: 2.0.0

相关信息

[AddFace](#), [ST_Polygonize](#)

9.10.2 AddNode

AddNode — 将点几何添加到指定拓扑模式的点表中，并返回新点的 `nodeid`。如果点已作点存在，返回有的 `nodeid`。

Synopsis

```
bigint AddNode(varchar toponame, geometry apoint, boolean allowEdgeSplitting=false, boolean computeContainingFace=false);
```

Description

将点几何添加到指定拓扑方案的点表中。[AddEdge](#)函数在用时会自添加的起点和点，因此无需式添加的点。

如果任何穿点的，会引异常或分割，具体取决于 `allowedEdgeSplitting` 参数。

如果 `computeContainingFace` 真，新添加的点将算出正确的包含面。



Note

如果 `apoint` 几何形已作点存在，不会添加点，但会返回有的 `nodeid`。

可用性: 2.0.0

示例

Code

```
SELECT topology.AddNode('ma_topo', ST_GeomFromText('POINT(227641.6 893816.5)', 26986)) AS  
nodeid;
```

Output

```
4
```

相关信息

[AddEdge](#), [CreateTopology](#)

9.10.3 AddEdge

AddEdge — 使用指定的串几何将串添加到表，并将关的起点和点添加到指定拓扑方案的点表，并返回新（或）的 `edgeid`。

Synopsis

```
bigint AddEdge(varchar toponame, geometry aline);
```

Description

使用指定的线串几何图形将线添加到表，并将关联点添加到指定 `toponame` 模式的点表，并返回新线或线的 `edgeid`。新添加的线都有“宇宙”面，并与其自身相交。



Note

如果 `aline` 几何图形与现有线串交叉、重叠、包含或被现有线串包含，会引发错误并且不会添加。



Note

`aline` 的几何形状必须具有与拓扑定义的相同的 `srid`，否则将引发无效的空参考系错误。

该函数是由 GEOS 模块实现的。



Warning

`AddEdge` 在 3.5.0 版本中已弃用。改用 `TopoGeo_AddLineString`。

可用性: 2.0.0

示例

Code

```
SELECT topology.AddEdge('ma_topo', ST_GeomFromText('LINESTRING(227575.8 893917.2,227591.9 893900.4)', 26986)) AS edgeid;
```

Output

1

Code

```
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;
```

Output

2

This query errors because the edge intersects an existing edge away from the endpoints.

Code

```
SELECT topology.AddEdge('ma_topo', ST_GeomFromText('LINESTRING(227591.2 893900,227591.9 893900.4,227704.5 893778.5)', 26986)) AS edgeid;
```

Output

```
ERROR: Edge intersects (not on endpoints) with existing edge 1
```

相关信息

[TopoGeo_AddLineString](#), [CreateTopology](#), [Section 4.5](#)

9.10.4 AddFace

AddFace — 将面基元注册到拓扑并返回其标识符。

Synopsis

bigint **AddFace**(varchar toponame, geometry apolygon, boolean force_new=false);

Description

将面基元注册到拓扑并返回其标识符。

对于新添加的面，形成其边界的边以及面中包含的边将更新其在 left_face 和 right_face 字段中具有正确的值。面中包含的孤立点也将更新其具有正确的 contains_face 字段值。



Note

该函数不使用也不设置表的 next_left_edge 和 next_right_edge 字段。

假定目标拓扑有效（不包含自相交边）。如果出现以下情况，将引发异常：多边形边界未完全由有效边组成，或者多边形与有效面重叠。

如果多边形几何体已作为面存在，则：如果 force_new 为假（默认），则返回有效面的面 id；如果 force_new 为 true，新的 id 将被分配给新生成的面。



Note

有效面行新配准（force_new=true）时，不会采取任何措施来解决边、点和关系表中有效面的悬挂引用，也不会更新有效面的 MBR 字段。由用户自行处理。



Note

多边形几何体必须具有与拓扑定义的相同的 srid，否则将引发无效的空参考系错误。

可用性: 2.0.0

示例

First add the edges using `generate_series` as an iterator. This query only works for polygons with fewer than 10000 points because of the maximum value in `generate_series`.

Code

```
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;
```

Output

```
3
4
5
6
7
8
9
10
11
12
```

Then add the face.

Code

```
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;
```

Output

```
1
```

相关信息

[AddEdge](#), [CreateTopology](#), [Section 4.5](#)

9.10.5 ST_Simplify

`ST_Simplify` — 使用 Douglas-Peucker 算法返回固定 TopoGeometry 的“简化”几何版本。

Synopsis

geometry **ST_Simplify**(topogeometry tg, float8 tolerance);

Description

在每个几何体上使用 Douglas-Peucker 算法返回给定 TopoGeometry 的“简化”几何版本。



Note

返回的几何体可能不精确或无效。
拆分几何体可能有助于保持精确性/有效性。

该函数是由 GEOS 模块实现的。

可用性：2.1.0

相关信息

几何 [ST_Simplify](#), [ST_IsSimple](#), [ST_IsValid](#), [ST_ModEdgeSplit](#)

9.10.6 RemoveUnusedPrimitives

RemoveUnusedPrimitives — 删除给定 TopoGeometry 对象不需要的拓扑基元。

Synopsis

bigint **RemoveUnusedPrimitives**(text topology_name, geometry bbox);

Description

查找表示给定 TopoGeometry 对象并不符合需要的所有元素（点、线、面）并将其删除，从而保持拓扑有效性（连接、面）和 TopoGeometry 空间占用。

不会创建新的元素符号，而是展示具有元素以包括合并的面（在删除线）或修复的线（在删除点）。

可用性：3.3.0

相关信息

[ST_ModEdgeHeal](#), [ST_RemEdgeModFace](#)

9.11 拓扑几何构造函数

9.11.1 CreateTopoGeom

CreateTopoGeom — 从拓扑元素数构建一个新的拓扑几何对象 - tg_type: 1:[多] 点, 2:[多] 线, 3:[多] 多边形, 4: 集合

Synopsis

topogeometry **CreateTopoGeom**(varchar toponame, integer tg_type, integer layer_id, topoelementarray tg_objs, bigint tg_id);

topogeometry **CreateTopoGeom**(varchar toponame, integer tg_type, integer layer_id);

Description

由 layer_id 表示的拓扑几何对象，并将其注册到拓扑名称架的关系表中。

tg_type 是一个整数：1:[多]点（点）、2:[多]线（线性）、3:[多]多边形（面）、4：集合。layer_id 是 topology.layer 表中的 id。

点状由一线点形成，线性由一线形成，区域由一面形成，集合可以由点、线和面的混合形成。

省略件数会生成一个空的 TopoGeometry 对象。

可用性：1.1

示例

Form from existing edges.

在“ri_topo”模式中“layer 2”（我的“ri_roads”）建立一个拓扑几何对象，类型 (2) LINE，用于第一条（我在 ST_CreateTopoGeo 中加的）。

Code

```
INSERT INTO ri.ri_roads(road_name, topo)
VALUES (
  'Unknown',
  topology.CreateTopoGeom(
    'ri_topo',
    2,
    2,
    '{{1,2}}'::topology.topoelementarray
  )
);
```

Convert an areal geometry to best guess topogeometry.

假我有由面的集合形成的几何形状。例如，我有表，并且想知道每个面的拓扑几何形状。如果我的数据完全一致，我可以做：

Create the topogeometry column.

Code

```
SELECT topology.AddTopoGeometryColumn(
  'topo_boston',
  'boston', 'blockgroups', 'topo', 'POLYGON');
```

Output

```
1
```

Update the column assuming everything is perfectly aligned with the edges.

I Code

```

UPDATE boston.blockgroups AS bg
SET topo = topology.CreateTopoGeom(
    'topo_boston',
    3,
    1,
    foo.bfaces
)
FROM (
    SELECT
        b.gid,
        topology.TopoElementArray_Agg(ARRAY[f.face_id, 3]) AS bfaces
    FROM boston.blockgroups AS b
    INNER JOIN topo_boston.face AS f
        ON b.geom && f.mbr
    WHERE ST_Covers(
        b.geom,
        topology.ST_GetFaceGeometry('topo_boston', f.face_id)
    )
    GROUP BY b.gid
) AS foo
WHERE foo.gid = bg.gid;

```

The world is rarely perfect, so this version allows some error. It counts a face if 50 percent of it falls within the expected block group boundary.

I Code

```

WITH candidate_faces AS (
    SELECT
        b.gid,
        b.geom AS block_geom,
        f.face_id,
        topology.ST_GetFaceGeometry('topo_boston', f.face_id) AS face_geom
    FROM boston.blockgroups AS b
    INNER JOIN topo_boston.face AS f
        ON b.geom && f.mbr
),
block_faces AS (
    SELECT
        gid,
        topology.TopoElementArray_Agg(ARRAY[face_id, 3]) AS bfaces
    FROM candidate_faces
    WHERE ST_Covers(block_geom, face_geom)
        OR (
            ST_Intersects(block_geom, face_geom)
            AND ST_Area(ST_Intersection(block_geom, face_geom))
            > ST_Area(face_geom) * 0.5
        )
    GROUP BY gid
)
UPDATE boston.blockgroups AS bg
SET topo = topology.CreateTopoGeom(
    'topo_boston',
    3,
    1,
    block_faces.bfaces
)
FROM block_faces
WHERE block_faces.gid = bg.gid;

```

To convert the topogeometry back to a denormalized geometry aligned with faces and edges, cast the topogeometry to a geometry. The resulting geometries are aligned with the TIGER street centerlines.

Code

```
UPDATE boston.blockgroups SET new_geom = topo::geometry;
```

相关信息

[AddTopoGeometryColumn](#), [toTopoGeom](#) [ST_CreateTopoGeo](#), [ST_GetFaceGeometry](#), [TopoElementArray](#), [TopoElementArray_Agg](#)

9.11.2 toTopoGeom

`toTopoGeom` — 将几何形转换为拓扑几何形。

Synopsis

topogeometry **toTopoGeom**(geometry geom, varchar toponame, integer layer_id, float8 tolerance=-1);

topogeometry **toTopoGeom**(geometry geom, topogeometry topogeom, float8 tolerance=-1);

Description

将几何形转换为拓扑几何形 **TopoGeometry**。

表示输入几何所需的拓扑基元将被添加到底层拓扑中，可能会拆分现有拓扑，并且它将与关系表中的 `TopoGeometry` 相关。

现有的 `TopoGeometry` 对象（拓扑几何可能除外，如果指定的）将保留其形状。

当提供 `tolerance` 时，会用它来将输入几何靠近到已有的基要素上。特殊值 `-1`（默认）表示使用拓扑的精度作容差。

在第一种形式中，将指定拓扑 (`toponame`) 的指定层 (`layer_id`) 新建新的 `TopoGeometry`。

在第二种形式中，新生成的元素将被添加到先存在的 `TopoGeometry` (`topogeom`) 中，可能会将其最内形状添加空。要新形状完全取代旧形状，参见 [clearTopoGeom](#)。

可用性：2.0

增强：2.1.0 添加了采用现有 `TopoGeometry` 的版本。

Changed: 3.7.0 将 `tolerance` 为 0 的含义改真正表示 0，而 `-1`（新的默认）表示使用拓扑精度。

示例

是一个完整的独立工作流程

Create the topology without allowing any tolerance.

Code

```
SELECT topology.CreateTopology('topo_boston_test', 2249);
```

Create a new table.

Code

```
CREATE TABLE nei_topo(gid serial primary key, nei varchar(30));
```

Add a topogeometry column to the new table.

Code

```
SELECT topology.AddTopoGeometryColumn('topo_boston_test', 'public', 'nei_topo', 'topo', ' ↵  
MULTIPOLYGON') AS new_layer_id;
```

Output

```
1
```

Use the new layer identifier to populate the topogeometry column.

Code

```
INSERT INTO nei_topo(nei, topo)  
SELECT nei, topology.toTopoGeom(geom, 'topo_boston_test', 1)  
FROM neighborhoods  
WHERE gid BETWEEN 1 and 15;
```

Verify what happened.

Code

```
SELECT * FROM  
    topology.TopologySummary('topo_boston_test');
```

Output

```
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.

Code

```
UPDATE nei_topo SET topo = toTopoGeom(ST_Buffer(topo, -10), clearTopoGeom(topo), 0);
```

Find faces that are left without any TopoGeometry reference after the shrink operation.

Code

```
SELECT ST_GetFaceGeometry('topo_boston_test', f.face_id)  
    FROM topo_boston_test.face f  
    WHERE f.face_id  
> 0  
    AND NOT EXISTS (  
        SELECT * FROM topo_boston_test.relation  
        WHERE layer_id = 1 AND element_id = f.face_id  
    );
```

相关信息

[CreateTopology](#), [AddTopoGeometryColumn](#), [CreateTopoGeom](#), [TopologySummary](#), [clearTopoGeom](#)

9.11.3 TopoElementArray_Agg

TopoElementArray_Agg — 返回一个 `element_id`、`type` (topoelements) 的 `topoelementarray`。

Synopsis

`topoelementarray` **TopoElementArray_Agg**(topoelement set tefield);

Description

用于从一个 `TopoElement` 构建 `TopoElementArray`。

可用性: 2.0.0

示例

Code

```
SELECT topology.TopoElementArray_Agg(ARRAY[e, t] ORDER BY e, t) As tea
FROM generate_series(1, 3) As e CROSS JOIN generate_series(1, 4) As t;
```

Output

```
{{1,1},{1,2},{1,3},{1,4},{2,1},{2,2},{2,3},{2,4},{3,1},{3,2},{3,3},{3,4}}
```

相关信息

`TopoElement`, `TopoElementArray`

9.11.4 TopoElement

TopoElement — 将拓扑几何转换为拓扑元素。

Synopsis

`topoelement` **TopoElement**(topogeometry topo);

Description

将 `TopoGeometry` 转换为 `TopoElement`。

可用性 : 3.4.0

示例

它是一个完整的独立工作流程

Return the TopoElement representation of each topogeometry.

Code

```
SELECT TopoElement(topo)
FROM neighborhoods;
```

Use a cast to aggregate topogeometries into a TopoElementArray.

Code

```
SELECT topology.TopoElementArray_Agg(topo::topoelement)
FROM neighborhoods
GROUP BY city;
```

相关信息

[TopoElementArray_Agg](#), [TopoGeometry](#), [TopoElement](#)

9.12 拓扑几何工具箱

9.12.1 clearTopoGeom

clearTopoGeom — 清除拓扑几何的内容。

Synopsis

topogeometry **clearTopoGeom**(topogeometry topogeom);

Description

清除 [TopoGeometry](#) 的内容，将其清空内容。主要与 [toTopoGeom](#) 联合使用，以替换有对象的形状以及更高层次中的任何依赖对象。

可用性：2.1

示例

Shrink all TopoGeometry polygons by 10 meters.

Code

```
UPDATE nei_topo SET topo = toTopoGeom(ST_Buffer(topo, -10), clearTopoGeom(topo), 0);
```

相关信息

[toTopoGeom](#)

9.12.2 TopoGeom_addElement

TopoGeom_addElement — 将元素添加到 TopoGeometry 的定义中。

Synopsis

topogeometry **TopoGeom_addElement**(topogeometry tg, topoelement el);

Description

将 **TopoElement** 添加到 TopoGeometry 对象的定义中。如果元素已经是定义的一部分，它不会出错。

可用性：2.3

示例

Add edge 5 to TopoGeometry tg.

Code

```
UPDATE mylayer SET tg = TopoGeom_addElement(tg, '{5,2}');
```

相关信息

TopoGeom_remElement, **CreateTopoGeom**

9.12.3 TopoGeom_remElement

TopoGeom_remElement — 从 TopoGeometry 的定义中删除元素。

Synopsis

topogeometry **TopoGeom_remElement**(topogeometry tg, topoelement el);

Description

从 TopoGeometry 对象的定义中删除 **TopoElement**。

可用性：2.3

示例

Remove face 43 from TopoGeometry tg.

Code

```
UPDATE mylayer SET tg = TopoGeom_remElement(tg, '{43,3}');
```

相关信息

TopoGeom_addElement, **CreateTopoGeom**

9.12.4 TopoGeom_addTopoGeom

TopoGeom_addTopoGeom — 将一个 TopoGeometry 的元素添加到另一个 TopoGeometry 的定义中。

Synopsis

```
topogeometry TopoGeom_addTopoGeom(topogeometry tgt, topogeometry src);
```

Description

将一个 **TopoGeometry** 的元素添加到另一个 TopoGeometry 的定义中，如果需要保存源对象中的所有元素，可能会将其存储型（类型属性）更改为集合。

一个 TopoGeometry 对象需要满足“相同”拓扑行定义，并且如果按次序定义，它需要由同一子集的元素组成。

可用性：3.2

示例

Set an overall TopoGeometry value to be composed of all elements from specific TopoGeometry values.

Code

```
UPDATE mylayer SET tg_overall = TopoGeom_addTopoGeom(
    TopoGeom_addTopoGeom(
        clearTopoGeom(tg_overall),
        tg_specific1
    ),
    tg_specific2
);
```

相关信息

TopoGeom_addElement, **clearTopoGeom**, **CreateTopoGeom**

9.12.5 toTopoGeom

toTopoGeom — 将几何形状添加到有拓扑几何形中。

Description

参考 **toTopoGeom**。

9.13 拓扑几何器

9.13.1 GetTopoGeomElementArray

GetTopoGeomElementArray — 返回一个 `topoelementarray`（包含拓扑元素的数组），其中包含指定 TopoGeometry 的拓扑元素和类型（原始元素）。

Synopsis

```
topoelementarray GetTopoGeomElementArray(varchar toponame, integer layer_id, bigint tg_id);
topoelementarray GetTopoGeomElementArray(topogeometry tg);
```

Description

返回一个 **TopoElementArray**，其中包含给定 TopoGeometry（原始元素）的拓扑元素和类型。它与 **GetTopoGeomElements** 类似，但不将它将元素作数组而不是数据集返回。

tg_id 是 topology.layer 表中 layer_id 表示的图中拓扑中的拓扑几何对象的拓扑几何 ID。

可用性：1.1

示例

相关信息

GetTopoGeomElements, **TopoElementArray**

9.13.2 GetTopoGeomElements

GetTopoGeomElements — 返回一个 topoelement 对象，其中包含给定 TopoGeometry（原始元素）的拓扑 element_id、element_type。

Synopsis

```
setof topoelement GetTopoGeomElements(varchar toponame, integer layer_id, bigint tg_id);
setof topoelement GetTopoGeomElements(topogeometry tg);
```

Description

返回一个与基本拓扑元素 **TopoElement**（1：点，2：线，3：面）相同的 element_id、element_type（topoelements），某些元素成了 toponame 模式中的给定拓扑几何对象。

tg_id 是 topology.layer 表中 layer_id 表示的图中拓扑中的拓扑几何对象的拓扑几何 ID。

可用性：2.0.0

示例

相关信息

GetTopoGeomElementArray, **TopoElement**, **TopoGeom_addElement**, **TopoGeom_remElement**

9.13.3 ST_SRID

ST_SRID — 返回拓扑几何的空参考标识符。

Synopsis

integer **ST_SRID**(topogeometry tg);

Description

Returns the spatial reference identifier of the topology containing the TopoGeometry.

The identifier is recorded for the topology when it is created and refers to an entry in Section 4.5.

可用性 : 3.2.0

示例

Return the SRID of a TopoGeometry stored in the ri_topo topology.

Code

```
SELECT topology.ST_SRID(topo) AS srid
FROM ri.roads
WHERE road_name = 'Unknown';
```

Output

```
srid
-----
3438
```

相关信息

Section 4.5, [ST_SetSRID](#), [ST_Transform](#), [ST_SRID](#)

9.14 拓扑几何输出

9.14.1 AsGML

AsGML — 返回拓扑几何的 GML 表示形式。

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);
```

Description

返回 GML3 版本格式的拓扑几何的 GML 表示形式。如果未指定 `nsprefix_in`，则使用 `gml`。将 `nsprefix` 放入一个空字符串以获取非限定名称空。精度（默认为 15）和 `trunc`（默认为 1）参数（如果指定）将原封不动地放到 `ST_AsGML` 的底部。

`VisitedTable` 参数（如果指定）用于跟踪访问的 Node 和 Edge 元素，以便使用交叉引用 (`xlink:xref`) 而不是重复定义。表必须有（至少）两个整数字段：“`element_type`”和“`element id`”。引用必须指定表具有读取和写入权限。为了获得最佳性能，请按 `element_type` 和 `element_id` 的顺序定义索引。此索引将通向字段添加唯一约束来自创建。例子：

Code

```
CREATE TABLE visited (
  element_type integer, element_id integer,
  unique(element_type, element_id)
);
```

`idprefix` 参数（如果指定）将被添加到 Edge 和 Node 的符号之前。

`gmlver` 参数（如果指定）将被设置到底部 `ST_AsGML`。默认为 3。

可用性: 2.0.0

示例

使用了我在 [CreateTopoGeom](#) 中创建的拓扑几何体

Code

```
SELECT topology.AsGML(topo) As rdgml
FROM ri.roads
WHERE road_name = 'Unknown';
```

Output

```
<gml:TopoCurve>
  <gml:directedEdge>
    <gml:Edge gml:id="E1">
      <gml:directedNode orientation="-">
        <gml:Node gml:id="N1"/>
      </gml:directedNode>
      <gml:directedNode>
        <gml:curveProperty>
          <gml:Curve srsName="urn:ogc:def:crs:EPSG::3438">
            <gml:segments>
              <gml:LineStringSegment>
                <gml:posList srsDimension="2">
>384744 236928 384750 236923 384769 236911 384799 236895 384811 236890
                384833 236884 384844 236882 384866 236881 384879 236883 384954 236898 385087 236932 385117 ↵
                236938
                385167 236938 385203 236941 385224 236946 385233 236950 385241 236956 385254 236971
                385260 236979 385268 236999 385273 237018 385273 237037 385271 237047 385267 237057 385225 ↵
                237125
                385210 237144 385192 237161 385167 237192 385162 237202 385159 237214 385159 237227 385162 ↵
                237241
                385166 237256 385196 237324 385209 237345 385234 237375 385237 237383 385238 237399 385236 ↵
                237407
                385227 237419 385213 237430 385193 237439 385174 237451 385170 237455 385169 237460 385171 ↵
                237475
                385181 237503 385190 237521 385200 237533 385206 237538 385213 237541 385221 237542 385235 ↵
                237540 385242 237541
                385249 237544 385260 237555 385270 237570 385289 237584 385292 237589 385291 237596 385284 ↵
                237630</gml:posList>
              </gml:LineStringSegment>
            </gml:segments>
          </gml:Curve>
        </gml:curveProperty>
      </gml:Edge>
    </gml:directedEdge>
  </gml:TopoCurve>
```

与之前的`ST_AsGML`相同，没有命名空间

Code

```
SELECT topology.AsGML(topo, '') As rdgml
FROM ri.roads
WHERE road_name = 'Unknown';
```

Output

```
<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>
```

相关信息

[CreateTopoGeom](#), [ST_CreateTopoGeo](#)

9.14.2 AsTopoJSON

AsTopoJSON — 返回拓扑几何的 TopoJSON 表示形式。

Synopsis

```
text AsTopoJSON(topogeometry tg, regclass edgeMapTable);
```

Description

返回拓扑几何的 TopoJSON 表示形式。如果 `edgeMapTable` 不为空，它将用作弧索引到弧索引的查找/存储映射。它是为了能在最文档中允许“arcs”数组。

表（如果指定）应具有“serial”类型的“arc_id”字段和整数类型的“edge_id”字段；它代表将表中的“edge_id”，因此建在字段上添加索引。

**Note**

TopoJSON 输出中的弧索引是从 0 开始的，但在 “edgeMapTable” 表中它是从 1 开始的。

除了函数返回的片段之外，完整的 TopoJSON 文档需要包含弧的弧和一些元数据。参见 [TopoJSON 规范](#)。

可用性：2.1.0

增强：2.2.1 添加了对 puntal 输入的支持

相关信息

ST_AsGeoJSON

示例

Create a temporary edge map for the TopoJSON output.

Code

```
CREATE TEMP TABLE edgemap(arc_id serial, edge_id int unique);
```

Build the TopoJSON header, objects, arcs, and footer.

Code

```
SELECT '{ "type": "Topology", "transform": { "scale": [1,1], "translate": [0,0] }, "objects ←
": { '

UNION ALL SELECT '' || feature_name || '": ' || AsTopoJSON(feature, 'edgemap')
FROM features.big_parcel WHERE feature_name = 'P3P4';

WITH edges AS (
  SELECT m.arc_id, e.geom FROM edgemap m, city_data.edge e
  WHERE e.edge_id = m.edge_id
), points AS (
  SELECT arc_id, (ST_DumpPoints(geom)).* FROM edges
), compare AS (
  SELECT p2.arc_id,
     CASE WHEN p1.path IS NULL THEN p2.geom
          ELSE ST_Translate(p2.geom, -ST_X(p1.geom), -ST_Y(p1.geom))
     END AS geom
  FROM points p2 LEFT OUTER JOIN points p1
  ON ( p1.arc_id = p2.arc_id AND p2.path[1] = p1.path[1]+1 )
  ORDER BY arc_id, p2.path
), arcdump AS (
  SELECT arc_id, (regexp_matches(ST_AsGeoJSON(geom), '\[.*\]'))[1] as t
  FROM compare
), arcs AS (
  SELECT arc_id, '[' || array_to_string(array_agg(t), ',') || ']' as a FROM arcdump
  GROUP BY arc_id
  ORDER BY arc_id
)
SELECT '}', "arcs": [' UNION ALL
SELECT array_to_string(array_agg(a), E',\n') from arcs

UNION ALL SELECT ']}':::text as t;
```

Output

```
{ "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]]
]]}
```

9.15 拓扑空☒关系

9.15.1 Equals

Equals — 如果☒个拓扑几何由相同的拓扑基元☒成，☒返回 true。

Synopsis

boolean **Equals**(topogeometry tg1, topogeometry tg2);

Description

如果☒个拓扑几何由相同的拓扑基元☒成：面、☒、☒点，☒返回 true。

**Note**

作☒几何集合的拓扑几何不支持此函数。它也无法比☒不同拓扑的拓扑几何形状。

可用性：1.1.0



☒函数支持 3d 并且不会☒失 z-index。

示例

The following example uses the `nei_topo` table created in the [toTopoGeom](#) examples.

Code

```
SELECT topology.Equals(a.topo, b.topo)
FROM nei_topo AS a
JOIN nei_topo AS b ON b.gid = a.gid
WHERE a.gid = 1;
```


Output

```
t
```

相关信息

[GetTopoGeomElements](#), [ST_Equals](#)

9.15.2 Intersects

Intersects — 如果两个拓扑几何中的任何一元素相交，返回 `true`。

Synopsis

```
boolean Intersects(topogeometry tg1, topogeometry tg2);
```

Description

如果两个拓扑几何中的任何一元素相交，返回 `true`。



Note

作几何集合的拓扑几何不支持此函数。它也无法比较不同拓扑的拓扑几何形状。目前也不支持分区拓扑几何（由其他拓扑几何组成的拓扑几何）。

可用性 : 1.1.0



函数支持 3d 并且不会丢失 z-index。

示例

The following example uses the `nei_topo` table created in the [toTopoGeom](#) examples.

Code

```
SELECT topology.Intersects(a.topo, b.topo)
FROM nei_topo AS a
JOIN nei_topo AS b ON b.gid = a.gid
WHERE a.gid = 1;
```

Output

```
t
```

相关信息

[ST_Intersects](#)

9.16 输入和输出拓扑

创建拓扑以及关闭的拓扑后，您可能希望将它导出基于文件的格式以行份或到一个数据库中。

使用 PostgreSQL 的基准/恢复工具有效的，因为拓扑由一表（基元 4 个，任意数量）和元数据表中的（`topology.topology` 和 `topology.layer`）组成。此外，拓扑符在数据库中并不是唯一的，因此在恢复拓扑需要更改拓扑参数。

了化拓扑的出/恢复，提供了一可行文件：`pgtopo_export` 和 `pgtopo_import`。用法示例：

Code

```
pgtopo_export dev_db topo1 | pgtopo_import topo1 | psql staging_db
```

9.16.1 使用拓扑导出器

`pgtopo_export` 脚本采用数据库和拓扑的名称，并出文件，文件可用于将拓扑（和关闭的）入到新数据库中。

默认情况下，`pgtopo_export` 将文件写入基准出，以便可以将其通过管道到 `pgtopo_import` 或重定向到文件（拒绝写入端）。您可以使用 `-f` 命令行开关指定出文件名。

默认情况下，`pgtopo_export` 包含指定拓扑定的所有层的。可能是比您需要的更多的数据，或者可能不起作用（如果您的表具有复的依关系），在这种情况下，您可以使用 `--skip-layers` 开关来跳过并独理些。

用 `pgtopo_export` 使用 `--help`（或写的 `-h`）指令将始打印短的用法明字符串。

文件格式是 `pgtopo_export` 目录的 tar 存档，其中至少包含一个有格式版本信息的 `pgtopo_dump_version` 文件。从版本 1 开始，目录包含制表符分隔的 CSV 文件，其中包含拓扑基元表的数据（点、线、面、关系）、与其关闭的拓扑和以及（除非出 `--skip-layers`）自定义 `-format PostgreSQL` 表告定拓扑的。

9.16.2 使用拓扑导入器

`pgtopo_import` 脚本采用 `pgtopo_export` 格式的拓扑和要建的拓扑的名称，并出重建拓扑和关闭的 SQL 脚本。

生成的 SQL 文件将包含以下句：建具有定名称的拓扑、在其中加原始数据、通将所有 TopoGeometry 正确接到其正确的拓扑来恢复和注册所有拓扑。

默认情况下，`pgtopo_import` 从基准入取，以便它可以与管道中的 `pgtopo_export` 合使用。您可以使用 `-f` 命令行开关指定入文件名。

默认情况下，`pgtopo_import` 在出 SQL 文件中包含用于恢复中找到的所有层的代。

如果您的目录数据已具有与中的表同名的表，可能是不需要的或不起作用的。在这种情况下，您可以使用 `--skip-layers` 命令来跳过并独（或稍后）理些。

可以使用 `--only-layers` 命令生成加并将接到命名拓扑的 SQL。于在解决命名冲突后加或接到不同的拓扑（例如起始拓扑的空化版本）非常有用。

如果目录拓扑已存在，并且你希望在建前将其删除，可以 `--drop-topology` 参数（自 PostGIS 3.6.0 起支持）。

Chapter 10

栅格数据管理、安装和使用程序

10.1 加载和创建栅格

对于大多数用例，您将通过使用打包的 `raster2pgsql` 栅格加载器加载有栅格文件来创建 PostGIS 栅格。

10.1.1 使用 `raster2pgsql` 加载栅格

`raster2pgsql` 是一个用于将 GDAL 支持的栅格格式加载到适合加载到 PostGIS 栅格表的 SQL 的栅格加载器可执行文件。它能加载文件中的栅格文件，同时创建栅格的概图。

由于 `raster2pgsql` 最常被包含在 PostGIS 的一部分（除非您有自己的 GDAL 安装），因此可执行文件支持的栅格类型将与 GDAL 依赖库中的栅格类型相同。要获取特定 `raster2pgsql` 支持的栅格类型列表，请使用 `-G` 指令。



Note

在从一个栅格的概图创建特定因子的概图时，有可能概图不会创建。请查看 <http://trac.osgeo.org/postgis/ticket/1764> 查看一个示例，其中概图未创建。

10.1.1.1 用法示例

使用加载器创建输入文件并将其以 100x100 分块上的示例可能会如下所示：

The loader assigns SRID 4326, creates an index and standard constraints, analyzes the loaded table, records source filenames, and stores the input rasters in 100 by 100 pixel tiles.

Code

```
raster2pgsql -s 4326 -I -C -M -F -t 100x100 *.tif public.demelevation  
> elev.sql  
psql -d gisdb -f elev.sql
```



Note

如果您没有将表指定为目录表名称的一部分，该表将在您所连接的数据库或用您的默认数据库中创建。

使用 UNIX 管道可以一步完成安装和上传：

I Code

```
raster2pgsql -s 4326 -I -C -M -F -t 100x100 \
*.tif public.demelevation |
psql -d gisdb
```

将美国塞州平面米制航空瓦片加载到名为 **aerial** 的架构中，并创建一个全表，2 和 4 概表，使用复制模式行插入（没有中间文件，直接到数据），并且使用“-e”选项来避免强制将所有内容放入一个事务中（如果你希望立即查看表中的数据而不必等待的）。将栅格分割成 128x128 像素的瓦片，并用栅格束。同时，包括一个名为“filename”的字段，用于存储瓦片所来源的文件名称。

I Code

```
raster2pgsql -I -C -e -Y -F -s 26986 -t 128x128 -l 2,4 \
bostonaerials2008/*.jpg arials.boston |
psql -U postgres -d gisdb -h localhost -p 5432
```

List the raster formats supported by this build:

I Code

```
raster2pgsql -G
```

The -G command outputs a list something like:

I Output

```
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 选项

Long options that take values also accept the --option=value spelling.

-?, --help 显示帮助屏幕。如果您不输入任何参数，也会显示帮助。

-G, --gdal-formats 打印支持的栅格格式。

(c|a|d|p) 选项是相互排斥的选项：

-c 新建表并用栅格填充它，是默认模式

-a 将栅格附加到有表。

-d 删除表，新建表并用栅格填充它

-p 准模式，只建表。

--if-not-exists Use IF NOT EXISTS for table creation in -c and -p modes. When -I or --create-index is also specified, use IF NOT EXISTS for index creation too. This option cannot be used with -d. Append mode requires an explicit creation action.

Atomic loader actions The short modes are presets over explicit actions. The equivalent long options are `--drop-table`, `--create-table`, `--load-data`, `--create-index`, `--add-constraints`, `--vacuum`, `--analyze`, and `--no-transaction`. Add `--if-not-exists` to make active creation actions use IF NOT EXISTS. `--drop-table` emits `DROP TABLE IF EXISTS` before the selected actions. With no mode specified, the default create/load actions still apply.

栅格物理：用约束以在栅格目录中正确注册

- C, --add-constraints** 用栅格约束——srid、像素大小等，以确保栅格在 `raster_columns` 中正确配准。
- x, --no-extent** Disable setting the max extent constraint. Only applied if -C or --add-constraints is also used.
- r, --regular-blocking** Set the constraints (spatially unique and coverage tile) for regular blocking. Only applied if -C or --add-constraints is also used.

栅格物理：用于操作输入栅格数据集的可参数

- s, --srid [<FROM_SRID>:]<SRID>** Assign the output raster to the specified SRID. When `FROM_SRID` is also provided, `raster2pgsql` marks the input raster with `FROM_SRID` and emits SQL that reprojects the raster to the target SRID. If the source SRID is omitted or is zero, the raster metadata will be checked to determine an appropriate source SRID. Reprojection cannot be used with copy mode.
- b, --band BAND** 从栅格中提取的波段索引（从 1 开始）。用于多个波段索引，用逗号（,）分隔。如果未指定，将提取栅格的所有波段。
- t, --tile-size TILE_SIZE** 将光栅切割成块，以便在每个表行插入一个块。TILE_SIZE 表示 WIDTHxHEIGHT 或置块“auto”，以允许加载程序使用第一个栅格块算适当的切片大小并将其用于所有栅格。
- P, --pad** 填充最右块和最底部的块，以确保所有块具有相同的宽度和高度。
- R, --register** 将栅格注册到文件系统（out-db）栅格。
栅格的元数据和栅格的路径位置存储在数据中（而不是像素）。
- l, --overview-factor OVERVIEW_FACTOR** Create overview of the raster. For more than one factor, separate with comma(.). Overview table name follows the pattern `overview_factor_table` where `overview_factor` is a placeholder for numerical overview factor and `table` is replaced with the base table name. If a source raster already has an overview with matching dimensions, that overview is reused. Created overview is stored in the database and is not affected by -R. Note that your generated sql file will contain both the main table and overview tables.
- N, --nodata NODATA** NODATA 用于没有 NODATA 的。

用于操作数据块象的可参数

- f, --raster-column COLUMN** 指定目标栅格列的名称，默认为“rast”
- F, --filename** 添加包含文件名的列
- n, --filename-column COLUMN** 指定文件名列的名称。意味着-F。
- q, --quote** 将 PostgreSQL 符号括在引号中。
- I** Alias for `--create-index`.
- create-index** Create a GiST index on the raster column at the end of this `raster2pgsql` run. With repeated -a append runs, create the index on the final run or after loading; add `--if-not-exists` to make reruns tolerate an existing index.
- M** 真空分析栅格表。
- k, --skip-nodata-check** 保留空的瓦片并跳过每个光栅波段的 NODATA 块。注意，块可以省略块的，但可能会致数据中出更多的行，并且些行不会空的瓦片。
- T, --tablespace tablespace** 指定新表的表空间。注意，索引（包括主）仍将使用默认表空间，除非使用了 -X 标志。

- X, --index-tablespace tablespace** 指定表的新索引的表空间。如果使用 **-I** 标志，适用于主索引和空索引。
- Y, --copy [MAX_ROWS_PER_COPY]** 使用复制语句而不是插入语句。可以指定 `max_rows_per_copy`；未指定默认为 50。
- e, --no-transaction** 独立行每条语句，不使用事务。
- E, --endian ENDIAN** 控制生成的光二制出的字节序；XDR 指定 0，NDR 指定 1（默认）；在支持 NDR 输出
- V, --wkb-version version** 指定输出格式的版本。默认为 0。目前支持 0。

10.1.2 使用 PostGIS 栅格函数建栅格

在多种情况下，您需要直接在数据库中构建栅格和栅格表。有很多函数可以做到这一点。要遵循的一般步骤。

1. 构建一个包含栅格列的表来保存新的栅格数据，可以通过以下方式完成：

Code

```
CREATE TABLE myrasters(rid serial primary key, rast raster);
```

2. 有多种功能可以帮助您一目了然。如果您构建的栅格不是其他栅格的衍生品，您需要从以下位置开始：**ST_MakeEmptyRaster**，然后是 **ST_AddBand**

您可以从几何图形构建栅格。除了这一点，您可能需要使用 **ST_AsRaster** 以及其他函数，例如 **ST_Union** 或 **ST_MapAlgebraFct** 或任何其他地理代数函数系列。

甚至有更多函数可用于从现有表构建新栅格表。例如，您可以使用 **ST_Transform** 在与现有投影不同的投影中构建栅格表

3. 最初填充表后，您将需要在栅格列上构建一个空索引，如下所示：

Code

```
CREATE INDEX myrasters_rast_st_convexhull_idx ON myrasters USING gist(ST_ConvexHull(
rast));
```

注意 **ST_ConvexHull** 的使用，因为大多数栅格运算符都基于栅格的凸包。



Note

PostGIS 栅格 2.0 之前的版本基于最小外接矩形而不是凸包。为了使空索引正常工作，您需要删除它并替换基于凸包的索引。

4. 使用 **AddRasterConstraints** 用栅格约束

10.1.3 使用 “out db” 云栅格

raster2pgsql 工具使用 GDAL 栅格数据，并且可以利用 GDAL 的一个关键功能：能够从远程存储在云“对象存储”（例如 AWS S3、Google Cloud Storage）中的栅格中获取数据。

有效使用云存储栅格需要使用“云化”格式。最知名和最广泛使用的是“云化的 GeoTIFF”格式。使用非云格式（例如 JPEG 或未平展的 TIFF）将导致性能非常差，因为系统每次需要访问子集都必须下载整个栅格。

首先，将栅格添加到您的云存储中。一旦添加，您将有一个 URI 来访问它，可以是“http”URI，有也可以是特定于服务的 URI。（例如，“s3://bucket/object”）。要使用非公共存储桶，您需要提供 GDAL 配置来访问您的连接。注意，此命令是从云栅格提取并写入数据。

Code

```
AWS_ACCESS_KEY_ID=xxxxxxxxxxxxxxxxxxxxxx \
AWS_SECRET_ACCESS_KEY=xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx \
raster2pgsql \
-s 990000 \
-t 256x256 \
-I \
-R \
/vsis3/your.bucket.com/your_file.tif \
your_table \
| psql your_db
```

加表后，您需要通配置个限（`postgis.enable_outdb_rasters` 和 `postgis.gdal_enabled_drivers`）来授予数据从程格取的限。

Code

```
SET postgis.enable_outdb_rasters = true;
SET postgis.gdal_enabled_drivers TO 'ENABLE_ALL';
```

了使更改生效并持久化，直接在您的数据上行置。您将需要重新接以体新的置。

Code

```
ALTER DATABASE your_db SET postgis.enable_outdb_rasters = true;
ALTER DATABASE your_db SET postgis.gdal_enabled_drivers TO 'ENABLE_ALL';
```

于非公开的格数据，您可能需要提供密来从云端格数据中取。你可以将写入 `raster2pgsql` 用使用的相同密置在数据内部使用，使用 `postgis.gdal_vsi_options` 配置。注意，可以通用空格分隔 `key=value` 来置多个。

Code

```
SET postgis.gdal_vsi_options = 'AWS_ACCESS_KEY_ID=xxxxxxxxxxxxxxxxxxxxxx
AWS_SECRET_ACCESS_KEY=xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx';
```

加数据并置限后，您可以使用相同的功能像任何其他格表一与格表行交互。当需要取像素数据，数据将理接到云数据的所有机制。

10.2 Raster Coverages and Raster/Vector Analysis

A coverage is a way to interpret spatial data as a mapping from a *domain* of locations to a *range* of values. Evaluating a coverage means asking for the value at a location; an inverse query asks for the locations whose values match a condition. PostGIS does not expose a generic ISO 19123 coverage SQL type, but geometry tables and raster values provide the building blocks for both directions.

A coverage can be continuous, with values calculated or interpolated for arbitrary locations, or discrete, with values supplied by stored spatial objects or grid cells. The coverage model describes how locations map to values, independently of whether those values come from an equation, a table, or one or more raster values. A time column or function argument can add a temporal dimension without changing that distinction.

In a vector coverage, a geometry column supplies the domain and other columns supply attributes in the range. A vector range can contain text, numbers, or another spatial value, provided its meaning and type are consistent wherever the coverage is defined. In a raster, each cell location is part of the

domain and the band values at that cell form the range. Other columns in the same table row describe the raster value or tile as a whole; they do not attach separate values to individual cells.

A logical raster coverage may be stored as one raster value or as many tiles in a table. Tiles that belong to one coverage should have compatible SRIDs, scales, alignment, band counts, pixel types, and NODATA conventions. The constraints exposed by Section 10.3.1 record and validate those properties.

Raster table arrangements. A raster column can represent an image warehouse, a tiled coverage, or a collection of rasterized objects. These arrangements have different assumptions about tile size, alignment, gaps, and overlaps. PostGIS does not infer the intended arrangement from the column type alone.

Arrangement	Row relationship	Typical use
Image warehouse	Independent rasters; sizes, alignment, and extents may differ or overlap	Cataloging unrelated source images
Irregular tiled coverage	One logical coverage with gaps allowed, variable tile sizes, and normally no overlap	Incremental or source-driven mosaics
Regular tiled coverage	One alignment and tile size with no overlap; gaps may remain at the boundary or inside the coverage	Indexed analysis over a large coverage
Rectangular regular coverage	Equal aligned tiles fill a rectangular extent without gaps or overlaps	Predictable block processing and overview generation
Separate tiled images	Each image is internally regular but stored in its own table or partition	Keeping source-image identity and lifecycle separate
Raster-object coverage	Each row is a rasterized feature; extents and sizes may differ and overlap	Object models where cells replace or supplement vector geometry

Use [AddRasterConstraints](#) to record common SRID, scale, alignment, band, extent, and regular-blocking properties, and use spatial indexes and application constraints for relationships that the catalog cannot express.

Queries against a tiled coverage normally have two stages:

1. Use the raster spatial index and [ST_Intersects](#) to select candidate tiles.
2. Read, clip, aggregate, or vectorize values inside those tiles with functions such as [ST_Value](#), [ST_Clip](#), [ST_SummaryStatsAgg](#), and [ST_DumpAsPolygons](#).

[ST_DumpAsPolygons](#) returns [geomval](#) rows that pair geometries with cell values. This allows ordinary geometry overlay and filtering to answer inverse coverage queries, but polygonizing many cells is usually more expensive than keeping the operation in raster space.

Choose boundary semantics deliberately. Clipping and aggregating raster pixels is usually the shorter and faster path. If a partial boundary cell must contribute in proportion to its covered area, intersect the geometry with the [geomval](#) rows and calculate an area-weighted result in an appropriate projected coordinate system. That exact partial-cell method costs more because it vectorizes the selected cells.

Tile size affects both sides of the indexed query: smaller tiles reduce the amount of raster data inspected for each candidate row, while increasing row and index overhead. Benchmark representative queries instead of treating one tile size as a universal default.

For a current end-to-end raster/vector pattern, load and tile data with Section 10.1.1, then see the building example in [ST_SummaryStats](#). It selects intersecting tiles, clips them to each polygon, and

combines per-tile statistics into one result per feature. For point observations such as animal locations, first buffer the points in an appropriate projected coordinate system, then apply the same tile selection, clipping, and aggregation pattern to an elevation or other continuous raster coverage.

10.3 栅格目录

PostGIS 附了一个栅格目录。这个目录都利用嵌入在栅格表约束中的信息。因此，由于限制施加了约束，目录始终与表中的栅格数据一致。

1. `raster_columns` 此目录数据中的所有栅格表列行目录。
2. `raster_overviews` 此目录数据中的所有栅格表列行目录，这些列用作更粒度表的概述。当您在加期使用 `-l` 开关，会生成种型的表。

10.3.1 栅格列目录

`raster_columns` 是数据中所有栅格型的栅格表列的目录。它是一种利用表约束的目录，因此即使您从一个数据目录的备份恢复一个栅格表，信息也始终保持一致。`raster_columns` 目录中存在以下列。

如果您未使用加器建表或忘记在加期指定 `-C` 标志，您可以在事后使用 `AddRasterConstraints` 限制行约束，以便 `raster_columns` 目录注册有关栅格切片的通用信息。

- `r_table_catalog` 表所在的数据目录。目录将始终取当前数据目录。
- `r_table_schema` 栅格表所属的数据目录模式。
- `r_table_name` 栅格表
- `r_raster_column` 在 `r_table_name` 表中栅格型的列。PostGIS 中没有任何内容可以阻止您在每个表中多个栅格列，因此可以多次列出一个栅格表，并且每个表具有不同的栅格列。
- `srid` 栅格的空参考符号。它是 Section 4.5 中的条目。
- `scale_x` 几何空坐标与像素之间的缩放比例。当栅格列中的所有切片具有相同的 `scale_x` 并且用此约束，此功能才可用。有关更多信息，参看 `ST_ScaleX`。
- `scale_y` 几何空坐标与像素之间的缩放比例。当栅格列中的所有切片具有相同的 `scale_y` 并且用了 `scale_y` 约束，此功能才可用。有关信息，参看 `ST_ScaleY`。
- `blocksize_x` 每个栅格片的宽度（跨像素数）。有关更多信息，参看 `ST_Width`。
- `blocksize_y` 每个栅格片的高度（向下的像素数）。有关更多信息，参看 `ST_Height`。
- `same_alignment` 一个布尔值，如果所有栅格片具有相同的对齐方式，则为 `true`。有关更多信息，参看 `ST_SameAlignment`。
- `Regular_blocking` 如果栅格列具有空唯一性和覆盖范围切片约束，则为 `TRUE`。否则，它将是 `FALSE`。
- `num_bands` 栅格集中每个切片中的波段数。它与 `ST_NumBands` 提供的信息相同 `ST_NumBands`
- `Pixel_types` 定义每个波段的像素类型的数组。数组中的元素数量与波段的数量相同。`Pixel_types` 是 `ST_BandPixelType` 中定义的以下类型之一。
- `nodata_values` 双精度数字数组，表示每个波段的 `nodata_value`。数组中的元素数量与数组的数量相同。这些数字定义了大多数操作中忽略的每个波段的像素。它与 `ST_BandNoDataValue` 提供的信息类似。
- `out_db` 布尔标志数组，指示栅格波段数据是否在数据目录外部目录。数组中的元素数量与数组的数量相同。
- `extent` 是栅格集中所有栅格行的范围。如果您计划添加更多数据来更改集的范围，需要在加之前行 `DropRasterConstraints` 函数，然后在加后使用 `AddRasterConstraints` 重新应用约束。
- `Spatial_index` 如果栅格列具有空索引，则为 `true` 的布尔值。

10.3.2 概图概述

`raster_overviews` 目录有关用于概图的表格列的信息以及在使用概图时了解的有用的附加信息。概图表在 `raster_columns` 和 `raster_overviews` 中各行目录，因为它本身就是表格，但它具有作为高分辨率表的低分辨率的漫画化图像的外特殊用途。当您在表格添加中使用 `-l` 开关时，它将与主表格一起生成，或者可以使用 [AddOverviewConstraints](#) 手册生成。

概图表包含与其他表格相同的约束以及特定于概图的附加信息约束。

**Note**

`raster_overviews` 中的信息与 `raster_columns` 中的信息不重复。如果您需要有关 `raster_columns` 中存在的概图表的信息，您可以将 `raster_overviews` 和 `raster_columns` 连接在一起以获得所需的完整信息集。

进行概图的主要原因：

1. 通常用于快速映射小的核心表的低分辨率表示。
2. 它的计算速度通常比其更高分辨率的父图更快，因为它更少，每个像素覆盖更多区域。尽管计算不如它支持的高分辨率表那么准确，但它在多粗略计算中已足够了。

`raster_overviews` 目录包含以下信息。

- `o_table_catalog` 概图表所在的数据库。它将始终取当前数据库。
- `o_table_schema` 概图表格所属的数据库架构。
- `o_table_name` 概图表格名称
- `o_raster_column` 概图表中的表格列。
- `r_table_catalog` 概图服务所在的表格表所在的数据库。它将始终取当前数据库。
- `r_table_schema` 此概图服务所属的表格表的数据库架构。
- `r_table_name` 此概图服务的表格表。
- `r_raster_column` 此概图列服务的表格列。
- `Overview_factor` - 是概述表的金字塔因子。数字越大，表的分辨率越低。`raster2pgsql` 如果指定图像文件，将计算每个图像文件的概述并单独添加。假定因子为 1，并且始终为原始文件。2 是每个因子代表 4 个原始因子。例如，如果您有一个包含 5000x5000 像素图像文件的文件，您将其分块为 125x125，那么每个图像文件，您的基表将有 $(5000*5000)/(125*125)$ 条记录 = 1600，您的 (l=2) `o_2` 表将有上限 $(1600/Power(2,2)) = 400$ 行，您的 (l=3) `o_3` 将有上限 $(1600/Power(2,3)) = 200$ 行。如果您的像素不能被因子的大小整除，您将得到一些碎片（未完全填充的块）。注意，`raster2pgsql` 生成的每个概图块与其父块具有相同数量的像素，但分辨率较低，其中每个像素代表（原始块的 $Power(2,overview_factor)$ 像素）。

10.4 使用 PostGIS Raster 构建自定义应用程序

事实上，PostGIS 为您提供 SQL 函数来以已知图像格式渲染图像，它还为您提供了很多渲染图像的函数。例如，您可以使用 OpenOffice / LibreOffice 进行渲染，如 [使用 LibreOffice 基图报告渲染 PostGIS 栅格图](#) 形中所示。此外，您还可以使用本图中演示的多种语言。

10.4.1 PHP 示例使用 ST_AsPNG 与其他栅格函数行输出

在本节中，我将演示如何使用 PHP PostgreSQL 程序和 **ST_AsGDALRaster** 系列函数将栅格的波段 1、2、3 输出到 PHP 请求流，然后将其嵌入到 `img src html` 中。

示例演示了如何将一大堆栅格函数合在一起，以获取与特定 `wgs 84` 边界框相交的所有栅格，然后将相交栅格与 **ST_Union** 合起来，返回所有波段，使用 **ST_Transform** 用指定的投影，然后输出使用 **ST_AsPNG** 将结果示为 `png`。

You would call the script with `http://mywebserver/test_raster.php?srid=2249` to get the raster image in Massachusetts state plane feet.

Code

```
<?php
$conn_str = 'dbname=mydb host=localhost port=5432 user=myuser password=mypwd';
$dbconn = pg_connect($conn_str);
header('Content-Type: image/png');

if (!empty($_REQUEST['srid']) && is_numeric($_REQUEST['srid'])) {
    $input_srid = intval($_REQUEST['srid']);
} else {
    $input_srid = 26986;
}

$sql = "SET bytea_output = 'escape';
SELECT ST_AsPNG(
    ST_Transform(
        ST_AddBand(
            ST_Union(rast, 1),
            ARRAY[ST_Union(rast, 2), ST_Union(rast, 3)]
        ),
        $input_srid
    )
) AS new_rast
FROM aerials.boston
WHERE ST_Intersects(
    rast,
    ST_Transform(
        ST_MakeEnvelope(-71.1217, 42.227, -71.1210, 42.218, 4326),
        26986
    )
)";

$result = pg_query($sql);
$row = pg_fetch_row($result);
pg_free_result($result);
if ($row === false) return;
echo pg_unescape_bytea($row[0]);
?>
```

10.4.2 ASP.NET C# 示例使用 ST_AsPNG 与其他栅格函数行输出

In this section, we'll demonstrate how to use Npgsql PostgreSQL .NET driver and the **ST_AsGDALRaster** family of functions to output band 1,2,3 of a raster to the HTTP response stream so it can be embedded in an `img` tag.

本节需要 `npgsql .NET PostgreSQL` 程序，您可以从 <http://npgsql.projects.postgresql.org/> 获取最新的程序。只需下载最新版本并放入 ASP.NET bin 文件中即可开始使用。

示例 10.4.1 演示了如何将一大堆栅格函数组合在一起，以获取与特定 wgs 84 边界框相交的所有栅格，然后将相交栅格与 **ST_Union** 组合起来，返回所有波段，使用 **ST_Transform** 用指定的投影，然后输出使用 **ST_AsPNG** 将结果示为 png。

示例与 Section 10.4.1 示例相同，只不过是在 C# 中实现的。

You would call the handler with `http://mywebserver/TestRaster.ashx?srid=2249` to get the raster image in Massachusetts state plane feet.

web.config connection string section:

Code

```
<connectionStrings>
  <add name="DSN"
        connectionString="server=localhost;database=mydb;Port=5432;User Id=myuser;password= ↵
        mypwd"/>
</connectionStrings>
```

Code for TestRaster.ashx:

Code

```
<%@ 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 {
            string connectionString = System.Configuration.ConfigurationManager
                .ConnectionStrings["DSN"].ConnectionString;
            using (NpgsqlConnection conn = new NpgsqlConnection(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
                        )
                    )";
            }
        }
        catch { }
        return result;
    }
}
```

```

        )
        ) As new_rast
FROM aeriels.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;
}
}

```

10.4.3 将栅格输出为图像文件的 Java 控制台应用程序

这是一个 Java 的控制台应用程序，它接受一个返回一图像并输出到指定文件的 SQL。

您可以从 <http://jdbc.postgresql.org/download.html> 下载最新的 PostgreSQL JDBC 驱动程序。

You can compile the following code with commands like:

Code

```

CLASSPATH=".../postgresql.jar" javac SaveQueryImage.java
jar cfm SaveQueryImage.jar Manifest.txt *.class

```

And call it from the command line with something like:

Code

```

java -jar SaveQueryImage.jar \
    "SELECT ST_AsPNG(ST_AsRaster(ST_Buffer(ST_Point(1, 5), 10, 'quad_segs=2'), 150, 150, '8 ↵
    BUI', 100));" \
    "test.png"

```

Manifest.txt:

I Code

Class-Path: postgresql-9.0-801.jdbc4.jar
Main-Class: SaveQueryImage

Code for SaveQueryImage.java:

I Code

```
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);
        }
    }
}
```

```

    }
  }
}

```

10.4.4 使用 PLPython 通过 SQL 创建图像

`write_file` 是一个 plpython 存储函数，它在服务器目录中为每条记录创建一个文件。需要你安装了 plpython。它可以与 plpythonu 和 plpython3u 一起正常工作。

Code

```

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. The PostgreSQL daemon account needs write access to the folder; this echoes back the file names created.

Code

```

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;

```

Output

```

write_file
-----
C:/temp/slices1.png
C:/temp/slices2.png
C:/temp/slices3.png
C:/temp/slices4.png
C:/temp/slices5.png

```

10.4.5 使用 PSQL 输出格式

Sadly PSQL doesn't have easy to use built-in functionality for outputting binaries. This is a bit of a hack that piggy backs on PostgreSQL somewhat legacy large object support. To use it, first launch your **psql** command line connected to your database.

与 python 方法不同，此方法在本地计算机上创建文件。

Code

```
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);
```

Output

oid	num_bytes
2630819	74860

Note the large object OID and export it to a file on your local computer, replacing the example path with the desired file path.

Code

```
\lo_export 2630819 'C:/temp/aerial_samp.png'
```

This deletes the file from large object storage in the database.

Code

```
SELECT lo_unlink(2630819);
```


Chapter 11

☒格参考

raster 是用于存☐和分析☐格数据的 PostGIS ☐型。

有关从光盘文件加载光盘的信息，参看 Section 10.1

本参考文档中的部分示例使用了一个通过以下代码构建的虚拟表格数据表：

Code

```
CREATE TABLE dummy_rast(rid integer, rast raster);
INSERT INTO dummy_rast(rid, rast)
VALUES (1,
('01' -- little endian (uint8 ndr)
||
'0000' -- version (uint16 0)
||
'0000' -- nBands (uint16 0)
||
'000000000000000040' -- scaleX (float64 2)
||
'0000000000000000840' -- scaleY (float64 3)
||
'000000000000E03F' -- ipX (float64 0.5)
||
'000000000000E03F' -- ipY (float64 0.5)
||
'0000000000000000' -- skewX (float64 0)
||
'0000000000000000' -- skewY (float64 0)
||
'00000000' -- SRID (int32 0)
||
'0A00' -- width (uint16 10)
||
'1400' -- height (uint16 20)
)::raster
),
-- Raster: 5 x 5 pixels, 3 bands, PT_8BUI pixel type, NODATA = 0
(2, ('01000003009A9999999999A93F9A9999999999A9BF000000E02B274A' ||
'410000000077195641000000000000000000000000000000000000000000000000 ←
FFFFFFFFFF050005000400FDFFDFEFDFEFDFEFDFEFDF9FAFEF' ||
' ←
EFCF9FBFDFEFDFCFAFEFEFE04004E627AADD16076B4F9FE6370A9F5FE59637AB0E54F58617087040046566487A1506
')::raster);
```

下列函数是 PostGIS Raster 常用的函数。除此之外，还存在其他栅格支持函数，但它一般用 `ST_` 而言并不常用。

11.1 栅格支持数据类型

11.1.1 geomval

geomval — 具有两个字段的空数据类型 - geom（保存几何对象）和 val（保存栅格中的双精度像素）。

Description

geomval 是一种复合数据类型，由 geom 字段引用的几何对象和 val 组成，val 是一个双精度值，表示栅格中特定几何位置的像素值。ST_DumpAsPolygon 和栅格交集函数系列使用它作为输出类型，将栅格分解为几何多边形。

相关信息

Section 13.6

11.1.2 addbandarg

addbandarg — 用作 ST_AddBand 函数的输入的复合类型，定义新波段的属性和初始值。

Description

用作 ST_AddBand 函数的输入的复合类型，定义新波段的属性和初始值。

Field	Type	Description
index	integer	从 1 开始的值，指示新波段将添加到栅格波段中的位置。如果 NULL，新波段将添加到栅格波段的末尾。
pixeltype	text	Pixel type of the new band. One of the pixel types described in ST_BandPixelType .
initialvalue	double precision	Initial value that all pixels of the new band will be set to.
nodataval	double precision	新波段的 NODATA 值。如果 NULL，新波段将不会分配 NODATA 值。

相关信息

[ST_AddBand](#)

11.1.3 rastbandarg

rastbandarg — 需要表栅格和栅格的波段索引使用的复合类型。

Description

需要表栅格和栅格的波段索引使用的复合型。

Field	Type	Description
rast	raster	The raster in question.
nband	integer	1-based value indicating the raster band.

相关信息

ST_MapAlgebra (callback function version)

11.1.4 raster

raster — 栅格空数据型。

Description

栅格是一种空数据类型，用于表示栅格数据，例如从 JPEG、TIFF、PNG、数字高程模型入的栅格数据。每个栅格都有 1 个或多个波段，每个波段都有一像素。栅格可以行地理参考。



Note
需要使用 GDAL 支持来 PostGIS。目前栅格可以式几何型，但返回栅格的 **ST_ConvexHull**。此自造可能会在不久的将来被除，因此不要依它。

型制

个部分列出了允用于种数据型的自以及式

到	行
geometry	automatic

相关信息

Chapter 11

11.1.5 reclassarg

reclassarg — 用于入 ST_Reclass 函数的复合型，用于定重分操作的行。

Description

用于入 **ST_Reclass** 函数的复合型，用于定重分操作的行。

Field	Type	Description
nband	integer	要重新分区的波段的波段号。
reclassexpr	text	Comma-delimited range:map_range mappings that define how old band values map to new band values.
pixeltype	text	One of defined pixel types as described in ST_BandPixelType .
nodataval	double precision	无数据的值。用于支持透明度的像图出，有些将空白。

The `reclassexpr` range delimiters mean: `(` greater than, `)` less than, `]` less than or equal, and `[` greater than or equal.

- `[a-b]` means `a <= x <= b`.
- `(a-b]` means `a < x <= b`.
- `[a-b)` means `a <= x < b`.
- `(a-b)` means `a < x < b`.

Opening-parenthesis notation is optional, so `a-b` means the same as `(a-b)`.

示例

Reclassify band 2 as an 8BUI where 255 is nodata value.

Code

```
SELECT ROW(2, '0-100:1-10, 101-500:11-150,501 - 10000: 151-254', '8BUI', 255)::reclassarg;
```

Reclassify band 1 as an 1BB and no nodata value defined.

Code

```
SELECT ROW(1, '0-100]:0, (100-255:1', '1BB', NULL)::reclassarg;
```

相关信息

[ST_Reclass](#)

11.1.6 summarystats

`summarystats` — `ST_SummaryStats` 和 `ST_SummaryStatsAgg` 函数返回的复合图型。

Description

ST_SummaryStats 和 **ST_SummaryStatsAgg** 函数返回的复合类型。

count integer 摘要数据的像素数。

sum double precision 所有数像素的和。

mean double precision 所有数像素的算平均。

stddev double precision 所有数像素的准偏差。

min double precision 数像素的最小。

max double precision 数像素的最大。

相关信息

ST_SummaryStats, **ST_SummaryStatsAgg**

11.1.7 unionarg

unionarg — 用作 ST_Union 函数的入的复合类型，定要理的波段和 UNION 操作的行。

Description

用作 ST_Union 函数的入的复合类型，定要理的波段和 UNION 操作的行。

Field	Type	Description
nband	integer	从 1 开始的，指示要理的每个入格的波段。
uniontype	text	Type of union operation. One of the types described in ST_Union .

相关信息

ST_Union

11.2 格管理

11.2.1 AddRasterConstraints

AddRasterConstraints — 将格束添加到已加的格表中，用于特定列，列束了空参考、比例、大小、、波段、波段类型以及一个标志，用于表示格列是否被地分。表必加数据才能推断出束。如果束置成功，返回 true，否会出通知。

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 );
```

Description

在栅格列上生成约束，用于指示 `raster_columns` 视图中的信息。 `rastschema` 是表所在的表模式的名称。
`srid` 必须是 `SPATIAL_REF_SYS` 表中条目的整数引用。

`raster2pgsql` 加载器使用此函数来注册栅格表

要输入的有效约束名称：有关更多信息，参见 [Section 10.3.1](#)。

- `blocksize` 设置 X 和 Y 大小
- `blocksize_x` 设置 X 个像素（每个像素的宽度（以像素单位））
- `blocksize_y` 设置 Y 像素（每个像素的像素高度）
- `extent` 计算整个表的范围并用约束所有栅格必须在范围内
- `num_bands` 波段数量
- `Pixel_types` 取每个波段的像素类型数，确保所有波段 `n` 具有相同的像素类型
- `Regular_blocking` 设置空唯一（每个栅格不能在空上相同）和覆盖范围（栅格与覆盖范围）约束
- `Same_alignment` 确保它们都具有相同的对齐方式，这意味着您比任何其他栅格都将返回 `true`。参见 [ST_SameAlignment](#)。
- `srid` 确保所有人都有相同的 `srid`
- `More`--任何列上述函数输入的内容



Note

函数根据表中已有的数据推断约束。因此，要使其正常工作，您必须首先构建栅格列，然后向其加载数据。



Note

如果在应用约束后需要在表中加载更多数据，并且数据范围已更改，您可能需要运行 `DropRasterConstraints`。

可用性: 2.0.0

示例

Apply all possible constraints on column based on data.

Verify the raster table is registered correctly in the `raster_columns` view.

Code

```
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);
SELECT srid, scale_x, scale_y, blocksize_x, blocksize_y, num_bands, pixel_types, ↵
    nodata_values
FROM raster_columns
WHERE r_table_name = 'myrasters';
```

Output

srid	scale_x	scale_y	blocksize_x	blocksize_y	num_bands	pixel_types	nodata_values
4326	2	2	1000	1000	1	{8BSI}	{0}

Apply single constraint.

Code

```
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');
```

Output

```
NOTICE: Adding regular blocking constraint
NOTICE: Adding blocksize-X constraint
NOTICE: Adding blocksize-Y constraint
```

相关信息

Section [10.3.1](#), [ST_AddBand](#), [ST_MakeEmptyRaster](#), [DropRasterConstraints](#), [ST_BandPixelType](#), [ST_SRID](#)

11.2.2 DropRasterConstraints

DropRasterConstraints — 删除引用栅格表列的 PostGIS 栅格约束。如果您需要重新添加数据或更新栅格列数据，非常有用。

Synopsis

boolean **DropRasterConstraints**(name rasttable, name rastcolumn, boolean srid, boolean scale_x, boolean scale_y, boolean blocksize_x, boolean blocksize_y, boolean same_alignment, boolean regular_blocking, boolean num_bands=true, boolean pixel_types=true, boolean nodata_values=true, boolean out_db=true , boolean extent=true);
boolean **DropRasterConstraints**(name rastschema, name rasttable, name rastcolumn, boolean srid=true, boolean scale_x=true, boolean scale_y=true, boolean blocksize_x=true, boolean blocksize_y=true, boolean same_alignment=true, boolean regular_blocking=false, boolean num_bands=true, boolean pixel_types=true, boolean nodata_values=true, boolean out_db=true , boolean extent=true);
boolean **DropRasterConstraints**(name rastschema, name rasttable, name rastcolumn, text[] constraints);

Description

删除引用由 **AddRasterConstraints** 添加的栅格表的 PostGIS 栅格约束。如果您需要添加更多数据或更新栅格列数据，它非常有用。如果您想要删除栅格表或栅格列，您无需执行此操作。

To drop a raster table use standard SQL: DROP TABLE mytable.

To drop just a raster column and leave the rest of the table, use standard SQL: ALTER TABLE mytable DROP COLUMN rast.

如果删除列或表，表将从 raster_columns 目录中消失。但是，如果删除约束，栅格列仍将列在 raster_columns 目录中，但除了列名称和表之外，不会有任何其他信息。

可用性: 2.0.0

示例

Code

```
SELECT DropRasterConstraints ('myrasters', 'rast');
```

Output

```
droprasterconstraints
-----
t
(1 row)
```

Verify change in raster_columns.

Code

```
SELECT srid, scale_x, scale_y, blocksize_x, blocksize_y, num_bands, pixel_types, ←
       nodata_values
FROM raster_columns
WHERE r_table_name = 'myrasters';
```

Output

```
srid | scale_x | scale_y | blocksize_x | blocksize_y | num_bands | pixel_types | nodata_values
-----+-----+-----+-----+-----+-----+-----+-----
0 | | | | | | |
```

相关信息

AddRasterConstraints

11.2.3 AddOverviewConstraints

AddOverviewConstraints — 将栅格列的概述。

Synopsis

boolean **AddOverviewConstraints**(name ovschema, name ovtable, name ovcolumn, name refschema, name reftable, name refcolumn, int ovfactor);
boolean **AddOverviewConstraints**(name ovtable, name ovcolumn, name reftable, name refcolumn, int ovfactor);

Description

添加栅格列的约束，用于指示 raster_overviews 栅格目录中的信息。
ovfactor 参数表示概列中的比例乘数：概因子越高，分辨率越低。
当省略 ovschema 和 refschema 参数时，将使用描述 search_path 找到的第一个表。
可用性: 2.0.0

示例

Verify the raster table is registered correctly in the raster_overviews view.

Code

```
CREATE TABLE res1 AS SELECT
ST_AddBand(ST_MakeEmptyRaster(1000, 1000, 0, 0, 2),
  1, '8BSI'::text, -129, NULL
) r1;

CREATE TABLE res2 AS SELECT
ST_AddBand(ST_MakeEmptyRaster(500, 500, 0, 0, 4),
  1, '8BSI'::text, -129, NULL
) r2;

SELECT AddOverviewConstraints('res2', 'r2', 'res1', 'r1', 2);
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';
```

Output

ot	oc	rt	rc	f
res2	r2	res1	r1	2
(1 row)				

相关信息

Section 10.3.2, [DropOverviewConstraints](#), [ST_CreateOverview](#), [AddRasterConstraints](#)

11.2.4 DropOverviewConstraints

DropOverviewConstraints — 取消栅格列作一栅格列概述的约束。

Synopsis

boolean **DropOverviewConstraints**(name ovschema, name ovtable, name ovcolumn);
boolean **DropOverviewConstraints**(name ovtable, name ovcolumn);

Description

从栅格列中删除用于将其指示 raster_overviews 栅格目录中一个栅格列的概述的约束。
当省略 ovschema 参数时，将使用描述 search_path 找到的第一个表。
可用性: 2.0.0

相关信息

Section [10.3.2, AddOverviewConstraints, DropRasterConstraints](#)

11.2.5 PostGIS_GDAL_Version

PostGIS_GDAL_Version — 报告 PostGIS 使用的 GDAL 的版本。

Synopsis

text **PostGIS_GDAL_Version()**;

Description

报告 PostGIS 使用的 GDAL 的版本。将并报告 GDAL 是否可以找到其数据文件。

示例

Code

```
SELECT PostGIS_GDAL_Version();
```

Output

```
GDAL 1.11dev, released 2013/04/13
```

相关信息

[postgis.gdal_datapath](#)

11.2.6 PostGIS_Raster_Lib_Build_Date

PostGIS_Raster_Lib_Build_Date — 报告完整的格式构建日期。

Synopsis

text **PostGIS_Raster_Lib_Build_Date()**;

Description

报告格式构建日期

示例

Code

```
SELECT PostGIS_Raster_Lib_Build_Date();
```

Output

```
2010-04-28 21:15:10
```

相关信息

[PostGIS_Raster_Lib_Version](#)

11.2.7 PostGIS_Raster_Lib_Version

PostGIS_Raster_Lib_Version — 报告完整的栅格版本和构建配置信息。

Synopsis

text **PostGIS_Raster_Lib_Version()**;

Description

报告完整的栅格版本和构建配置信息。

示例

Code

```
SELECT PostGIS_Raster_Lib_Version();
```

Output

```
2.0.0
```

相关信息

[PostGIS_Lib_Version](#)

11.2.8 ST_GDALDrivers

ST_GDALDrivers — 通过 GDAL 返回 PostGIS 支持的栅格格式列表。ST_AsGDALRaster 可使用 can_write=True 的格式

Synopsis

setof record **ST_GDALDrivers**(integer OUT idx, text OUT short_name, text OUT long_name, text OUT can_read, text OUT can_write, text OUT create_options);

Description

返回 GDAL 支持的每种格式的短名称、名称和创建者信息的列表。使用短名称作为格式参数 `ST_AsGDALRaster` 的输入。信息根据 `libgdal` 所使用的程序而有所不同。 `create_options` 返回 XML 格式的 `CreationOptionList/Option` 集，其中包含特定程序的每个创建者信息的名称和可 `type`、`description` 以及 `VALUE` 集。

更改：2.5.0 - 添加 `can_read` 和 `can_write` 列。

更改：2.0.6、2.1.3 - 默认情况下不使用任何程序，除非设置了 GUC 或环境变量 `gdal_enabled_drivers`。

可用性：2.0.0 - 需要 GDAL >= 1.6.0。

示例

List of Drivers.

Code

```
SET postgis.gdal_enabled_drivers = 'ENABLE_ALL';
SELECT short_name, long_name, can_write
FROM ST_GDALDrivers()
ORDER BY short_name;
```

Output

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_CE0S	CE0S SAR Image	f
SDTS	SDTS Raster	f
SENTINEL2	Sentinel 2	f
SGI	SGI Image File Format 1.0	f
SNODAS	Snow Data Assimilation System	f
SRP	Standard Raster Product (ASRP/USRP)	f
SRTMHGT	SRTMHGT File Format	t
Terragen	Terragen heightfield	f
TIL	EarthWatch .TIL	f
TSX	TerraSAR-X Product	f
USGSDEM	USGS Optional ASCII DEM (and CDED)	t
VICAR	MIPL VICAR file	f

VRT	Virtual Raster	t
WCS	OGC Web Coverage Service	f
WMS	OGC Web Map Service	t
WMTS	OGC Web Map Tile Service	t
XPM	X11 PixMap Format	t
XYZ	ASCII Gridded XYZ	t
ZMap	ZMap Plus Grid	t

List of options for each driver.

Output the create options XML column of JPEG as a table. These creator options can be used in the ST_AsGDALRaster options argument.

Code

```
SELECT (xpath('@name', g.opt))[1]::text AS oname,
       (xpath('@type', g.opt))[1]::text AS otype,
       (xpath('@description', g.opt))[1]::text AS descrip
FROM (SELECT unnest(xpath('/CreationOptionList/Option', create_options::xml)) AS opt
FROM ST_GDALDrivers())
WHERE short_name = 'JPEG') AS g;
```

Output

oname	otype	descrip
PROGRESSIVE	boolean	whether to generate a progressive JPEG
QUALITY	int	good=100, bad=0, default=75
WORLDFILE	boolean	whether to generate a worldfile
INTERNAL_MASK	boolean	whether to generate a validity mask
COMMENT	string	Comment
SOURCE_ICC_PROFILE	string	ICC profile encoded in Base64
EXIF_THUMBNAIL	boolean	whether to generate an EXIF thumbnail(overview). By default its max dimension will be 128
THUMBNAIL_WIDTH	int	Forced thumbnail width
THUMBNAIL_HEIGHT	int	Forced thumbnail height

(9 rows)

This example turns a few common GeoTIFF creator options into a compact table.

Code

```
WITH gtiff_options AS (
  SELECT
    option_name,
    option_type,
    default_value
  FROM ST_GDALDrivers() AS driver
  CROSS JOIN LATERAL XMLTABLE(
    '/CreationOptionList/Option'
    PASSING (driver.create_options::xml)
    COLUMNS
      option_name text PATH '@name',
      option_type text PATH '@type',
      default_value text PATH '@default'
  ) AS options
  WHERE driver.short_name = 'GTiff'
)
SELECT option_name AS name, option_type AS type, coalesce(default_value, '') AS ↵
       default_value
FROM gtiff_options
WHERE option_name = ANY (ARRAY[
  'COMPRESS',
  'INTERLEAVE',
  'TILED',
  'BLOCKXSIZE',
  'BLOCKYSIZE',
```

```
'JPEG_QUALITY'  
])  
ORDER BY 1;
```

Output

name	type	default_value
BLOCKXSIZE	int	
BLOCKYSIZE	int	
COMPRESS	string-select	
INTERLEAVE	string-select	PIXEL
JPEG_QUALITY	int	75
TILED	boolean	

(6 rows)

Use XMLTABLE to expose selected GTiff creation options as rows. The same pattern can be used without the name filter to inspect the full XML document.

Code

```
SET postgis.gdal_enabled_drivers = 'ENABLE_ALL';  
  
SELECT option_name, option_type  
FROM ST_GDALDrivers() AS driver  
CROSS JOIN LATERAL XMLTABLE(  
  '/CreationOptionList/Option'  
  PASSING (driver.create_options::xml)  
  COLUMNS  
    option_name text PATH '@name',  
    option_type text PATH '@type'  
) AS options  
WHERE driver.short_name = 'GTiff'  
  AND option_name IN ('COMPRESS', 'TILED', 'BIGTIFF')  
ORDER BY option_name;
```

Output

option_name	option_type
BIGTIFF	string-select
COMPRESS	string-select
TILED	boolean

(3 rows)

相关信息

[ST_AsGDALRaster](#), [ST_SRID](#), [postgis.gdal_enabled_drivers](#)

11.2.9 UpdateRasterSRID

UpdateRasterSRID — 更改用 `column_name` 指定的列和表中所有 `table_name` 格的 SRID。

Synopsis

```
raster UpdateRasterSRID(name schema_name, name table_name, name column_name, integer new_srid);  
raster UpdateRasterSRID(name table_name, name column_name, integer new_srid);
```

Description

更改用 `col` 指定的列和表中所有栅格的 SRID。在更改指定列栅格的 SRID 之前，该函数将删除所有适当的列约束（范围、索引方式和 SRID）。



Note

此函数不会触及栅格的数据（波段像素）。栅格的元数据会更改。

可用性：2.1.0

相关信息

[UpdateGeometrySRID](#)

11.2.10 ST_CreateOverview

`ST_CreateOverview` — 构建指定栅格覆盖范围的降低分辨率版本。

Synopsis

regclass **ST_CreateOverview**(regclass tab, name col, int factor, text algo='NearestNeighbor');

Description

使用源表中重新采样的栅格构建一个概览表。输出栅格将具有与输入栅格相同的大小，并以更低的分辨率覆盖相同的空间范围（像素大小在每个方向上都是原始栅格的 $1/\text{factor}$ ）。

概览表将在 `raster_overviews` 目录中提供，并将限制行栅格约束。

算法选项有：“NearestNeighbor”、“Bilinear”、“Cubic”、“CubicSpline”和“Lanczos”。有关更多信息，请参阅：[GDAL Warp 重采样方法](#)。

可用性：2.2.0

示例

输出量通常很好，但品质格式较慢

Code

```
SELECT ST_CreateOverview('mydata.mytable'::regclass, 'rast', 2, 'Lanczos');
```

输出到更快地处理默认最近邻

Code

```
SELECT ST_CreateOverview('mydata.mytable'::regclass, 'rast', 2);
```

相关信息

[ST_Retile](#), [AddOverviewConstraints](#), [AddRasterConstraints](#), [Section 10.3.2](#)

11.3 栅格造器

11.3.1 ST_AddBand

`ST_AddBand` — 返回一个栅格，其中在指定索引位置添加了指定类型的新波段和指定初始值。如果未指定索引，将波段添加到末尾。

Synopsis

```
(1) raster ST_AddBand(raster rast, addbandarg[] addbandargset);
(2) raster ST_AddBand(raster rast, integer index, text pixelftype, double precision initialvalue=0, double precision nodataval=NULL);
(3) raster ST_AddBand(raster rast, text pixelftype, double precision initialvalue=0, double precision nodataval=NULL);
(4) raster ST_AddBand(raster torast, raster fromrast, integer fromband=1, integer torastindex=at_end);
(5) raster ST_AddBand(raster torast, raster[] fromrasts, integer fromband=1, integer torastindex=at_end);
(6) raster ST_AddBand(raster rast, integer index, text outdbfile, integer[] outdbindex, double precision nodataval=NULL);
(7) raster ST_AddBand(raster rast, text outdbfile, integer[] outdbindex, integer index=at_end, double precision nodataval=NULL);
```

Description

返回一个栅格，其中在指定位置（索引）、指定类型、指定初始值和指定 `nodata` 值中添加了新波段。如果未指定索引，将波段添加到末尾。如果未指定 `fromband`，假定 `band 1`。像素类型是 `ST_BandPixelType` 中指定的像素类型之一的字符串表示形式。如果指定了有索引，所有 $\geq$ 索引的后波段都会增加 1。如果指定的初始值大于像素类型的最大值，初始值将置为像素类型允许的最高值。

用于采用 `addbandarg` 数组的格式（格式 1），特定 `addbandarg` 的索引与 `addbandarg` 描述的波段添加到栅格的栅格相关。参看下面的多个新波段示例。

用于采用栅格数组的格式（格式 5），如果 `torast` 是 `NULL`，数组中每个栅格的 `fromband` 波段将累积到新栅格中。

用于采用 `outdbfile` 的格式（格式 6 和 7），必须包含栅格文件的完整路径。文件必须可供 `postgres` 服务器进程访问。

增强：2.1.0 添加了对 `addbandarg` 的支持。

增强：2.1.0 添加了对新的 out-db 波段的支持。

示例

Single New Band.

This example adds another 8-bit unsigned integer band with pixels initialized to 200.

Code

```
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`.

Output metadata for the raster bands to verify they are correct.

Code

```

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[]
)
);

SELECT (bmd).*
FROM (SELECT ST_BandMetaData(rast, generate_series(1, 2)) As bmd
FROM dummy_rast WHERE rid = 10) AS foo;

```

Output

pixeltype	nodatavalue	isoutdb	path
1BB		f	
4BUI		f	

This example outputs metadata for the raster.

Code

```

SELECT (rmd).width, (rmd).height, (rmd).numbands
FROM (SELECT ST_MetaData(rast) As rmd
FROM dummy_rast WHERE rid = 10) AS foo;

```

Output

width	height	numbands
100	100	2

Multiple New Bands.

Code

```

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[]
);

```

Output

bandnum	pixeltype	nodatavalue	isoutdb	path	outdbbandnum	filesize	filetimestamp
1	8BUI	0	f	null	null	null	null
2	32BF	-1	f	null	null	null	null
3	32BUI	12	f	null	null	null	null
4	16BUI	2	f	null	null	null	null

Aggregate the 1st band of a table of like rasters into a single raster. With as many bands as there are test_types and as many rows (new rasters) as there are mice. Note that The ORDER BY test_type clause relies on aggregate ORDER BY support. The resulting raster will have a band for each test_type alphabetical by test_type. For mouse lovers: No mice were harmed in this exercise.

Code

```
SELECT
    mouse,
    ST_AddBand(NULL, array_agg(rast ORDER BY test_type), 1) As rast
FROM mice_studies
GROUP BY mouse;
```

New Out-db band.

Code

```
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[]
);
```

Output

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

相关信息

[ST_BandMetaData](#), [ST_BandPixelType](#), [ST_MakeEmptyRaster](#), [ST_MetaData](#), [ST_NumBands](#), [ST_Reclass](#)

11.3.2 ST_AsRaster

ST_AsRaster — 将 PostGIS 几何类型转换为 PostGIS 栅格。

Synopsis

raster **ST_AsRaster**(geometry geom, raster ref, text pixeltype, double precision value=1, double precision nodataval=0, boolean touched=false);
raster **ST_AsRaster**(geometry geom, raster ref, text[] pixeltype=ARRAY['8BUI'], double precision[] value=ARRAY[1], double precision[] nodataval=ARRAY[0], boolean touched=false);
raster **ST_AsRaster**(geometry geom, double precision scalex, double precision scaley, double precision gridx, double precision gridy, text pixeltype, double precision value=1, double precision nodataval=0, double precision skewx=0, double precision skewy=0, boolean touched=false);
raster **ST_AsRaster**(geometry geom, double precision scalex, double precision scaley, double precision gridx=NULL, double precision gridy=NULL, text[] pixeltype=ARRAY['8BUI'], double precision[] value=ARRAY[1], double precision[] nodataval=ARRAY[0], double precision skewx=0, double precision skewy=0, boolean touched=false);
raster **ST_AsRaster**(geometry geom, double precision scalex, double precision scaley, text pixeltype, double precision value=1, double precision nodataval=0, double precision upperleftx=NULL, double precision upperlefty=NULL, double precision skewx=0, double precision skewy=0, boolean touched=false);
raster **ST_AsRaster**(geometry geom, double precision scalex, double precision scaley, text[] pixeltype, double precision[] value=ARRAY[1], double precision[] nodataval=ARRAY[0], double precision

```
upperleftx=NULL, double precision upperlefty=NULL, double precision skewx=0, double precision
skewy=0, boolean touched=false);
raster ST_AsRaster(geometry geom, integer width, integer height, double precision gridx, double
precision gridy, text pixeltype, double precision value=1, double precision nodataval=0, double pre-
cision skewx=0, double precision skewy=0, boolean touched=false);
raster ST_AsRaster(geometry geom, integer width, integer height, double precision gridx=NULL,
double precision gridy=NULL, text[] pixeltype=ARRAY['8BUI'], double precision[] value=ARRAY[1],
double precision[] nodataval=ARRAY[0], double precision skewx=0, double precision skewy=0, boolean
touched=false);
raster ST_AsRaster(geometry geom, integer width, integer height, text pixeltype, double precision
value=1, double precision nodataval=0, double precision upperleftx=NULL, double precision upper-
lefty=NULL, double precision skewx=0, double precision skewy=0, boolean touched=false);
raster ST_AsRaster(geometry geom, integer width, integer height, text[] pixeltype, double preci-
sion[] value=ARRAY[1], double precision[] nodataval=ARRAY[0], double precision upperleftx=NULL,
double precision upperlefty=NULL, double precision skewx=0, double precision skewy=0, boolean
touched=false);
```

Description

将一个 PostGIS 几何对象转换为 PostGIS 栅格。这些不同的函数提供了三种可能性，用于设置生成的栅格的方式和像素大小。

第一函数由前三个函数组成，生成具有与所提供的参考栅格相同的设置方式（`scalex`、`scaley`、`gridx` 和 `gridy`）、像素类型和点数据的栅格。通常，您可以通过将包含几何形状的表与包含参考栅格的表连接来使用此参考栅格。

第二函数，由四个函数组成，允许您通过提供像素大小的参数（`scalex` & `scaley` 以及 `skewx` & `skewy`）来设置栅格的尺寸。生成的栅格的宽度 & 高度将根据几何对象的范围进行调整。在大多数情况下，您必须将整数的 `scalex` & `scaley` 参数限制为双精度，以便 PostgreSQL 使用正确的函数。

第三函数，由四个函数组成，允许您通过提供栅格的尺寸（宽度 & 高度）来固定栅格的尺寸。生成的栅格的像素大小参数（`scalex` & `scaley` 以及 `skewx` & `skewy`）将被调整以适应几何对象的范围。

前两个最后函数中的前三个函数允许您使用任何网格的任意角（`gridx` & `gridy`）指定设置方式，而最后两个函数使用左上角（`upperleftx` & `upperlefty`）指定设置方式。

每个函数都允许生成波段栅格或多波段栅格。要生成多波段栅格，必须提供像素类型数组（`pixeltype[]`）、初始值（`value`）和 nodata 值（`nodataval`）。如果未提供，`pixeltype` 默认为 8BUI，`value` 为 1，`nodataval` 为 0。

输出栅格将与源几何具有相同的空间参考。唯一的例外是具有参考栅格的函数。在这种情况下，生成的栅格将具有与参考栅格相同的 SRID。

可选项参数 `touched` 默认为 false，并映射到 GDAL 中的 ALL TOUCHED 栅格化选项，用于确定由一个或多个形状接触的像素是否会被设置。不是那些在渲染路径上的像素，或者它们的中心点位于多边形内的像素。

当与 **ST_AsPNG** 和其他 **ST_AsGDALRaster** 系列函数合使用时，函数直接用于从数据渲染几何形状的 jpeg 和 png 特性有用。

可用性：2.0.0 - 需要 GDAL >= 1.6.0。

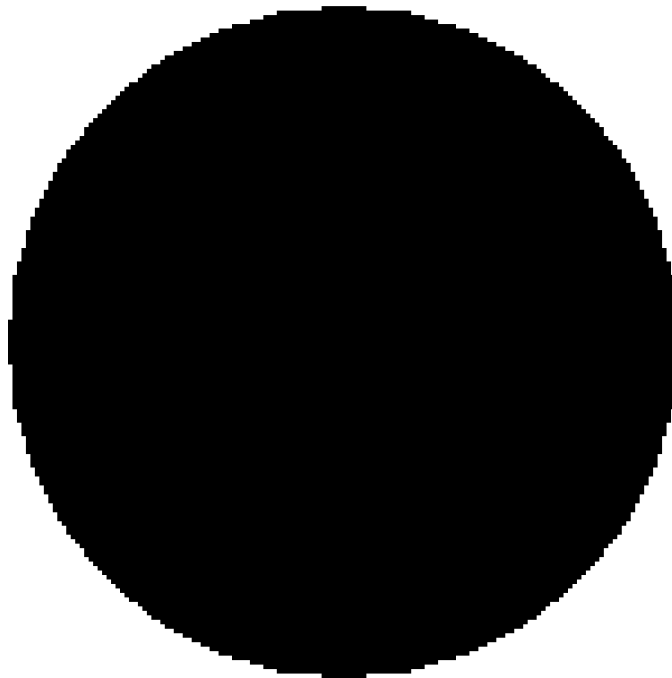
Curved geometry inputs are linearized before rasterization, including NURBS Curve inputs handled by PostGIS. TIN and PolyhedralSurface inputs are not supported.

示例：将几何形状输出为 PNG 文件

This example outputs a black circle on a white background, occupying 150 by 150 pixels.

Code

```
SELECT ST_AsPNG(  
  ST_AsRaster(  
    ST_Buffer(ST_Point(1, 5), 10),  
    150, 150,  
    ARRAY['8BUI', '8BUI', '8BUI'],  
    ARRAY[0, 0, 0],  
    ARRAY[255, 255, 255]  
  )  
) AS image;
```

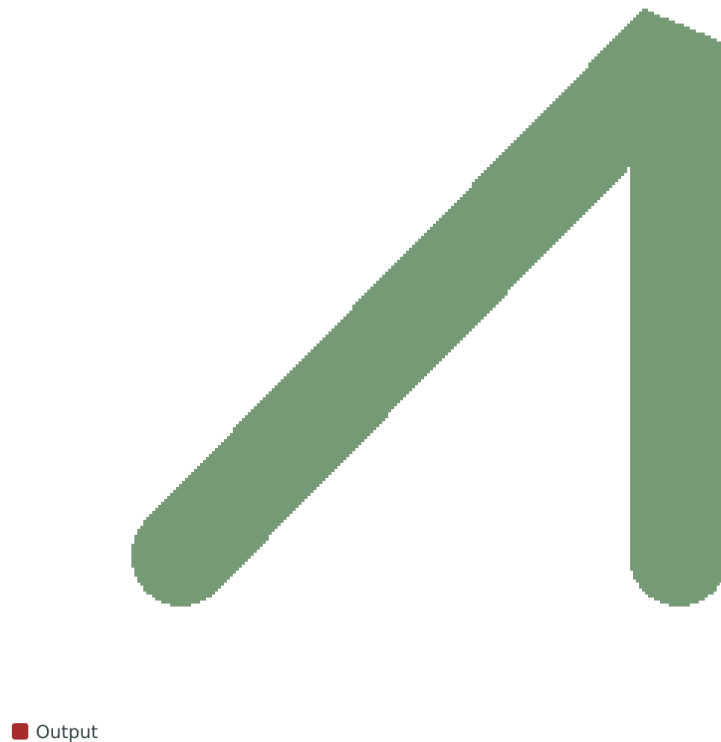
Figure

■ Output

This example maps bands to RGB channels, producing the teal value (118,154,118).

Code

```
SELECT ST_AsPNG(  
  ST_AsRaster(  
    ST_Buffer('LINESTRING(50 50,150 150,150 50)::geometry', 10, 'join=bevel'),  
    200, 200,  
    ARRAY['8BUI', '8BUI', '8BUI'],  
    ARRAY[118, 154, 118],  
    ARRAY[255, 255, 255]  
  )  
) AS image;
```

| Figure**相关信息**

[ST_BandPixelType](#), [ST_Buffer](#), [ST_GDALDrivers](#), [ST_AsGDALRaster](#), [ST_AsPNG](#), [ST_AsJPEG](#), [ST_SRID](#)

11.3.3 ST_AsRasterAgg

`ST_AsRasterAgg` — 是一个聚合函数。将 PostGIS 几何形状转换为新的栅格图像。

Synopsis

raster **ST_AsRasterAgg**(geometry geom, double precision val, raster ref, text pixeltype, double precision nodataval, text uniontype, boolean touched);

Description

返回一个波段栅格，包含所有输入几何形状及其关联的渲染结果。

可用性：3.6.0

示例

Code

```
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;
```

Output

w	h	v5_0	v15_0	v25_0
30	20	1	3	2
(1 row)				

相关信息

[ST_AsRaster](#), [ST_DumpAsPolygons](#), [ST_Union](#)

11.3.4 ST_Band

`ST_Band` — 返回具有栅格的一个或多个波段作新栅格。用于从具有栅格建新栅格非常有用。

Synopsis

```
raster ST_Band(raster rast, integer[] nbands = ARRAY[1]);  
raster ST_Band(raster rast, integer nband);  
raster ST_Band(raster rast, text nbands, character delimiter=,);
```

Description

返回具有栅格的一个或多个波段作新栅格。用于从具有栅格建新栅格或取出栅格的特定波段或重新排列栅格中波段的顺序非常有用。如果未指定波段或栅格中不存在任何指定波段，返回所有波段。用作各种功能中的辅助功能，例如除波段。



Warning

对于函数的 `nbands` 作文本体，默认分隔符是`,`，这意味着您可以请求`'1,2,3'`，如果您想使用不同的分隔符，可以使用 `ST_Band(rast, '1@2@3', '@')`。对于请求多个波段，我强烈建议您使用函数数的形式，例如 `ST_Band(rast, '{1,2,3}'::int[])`；因 `text` 格式的波段列表可能会在未来的 PostGIS 版本中被移除。

可用性: 2.0.0

示例

This example creates two rasters: one containing band 1 of `dummy_rast`, and one containing band 2 reclassified as `2BUI`.

Code

```
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;
```

Output

numb1	pix1	numb2	pix2
1	8BUI	1	2BUI

Return bands 2 and 3 using array cast syntax.

Code

```
SELECT ST_NumBands(ST_Band(rast, '{2,3}'::int[])) As num_bands
FROM dummy_rast WHERE rid=2;
```

Output

2

Return bands 2 and 3 using array syntax.

Code

```
SELECT ST_NumBands(ST_Band(rast, ARRAY[2, 3])) As num_bands
FROM dummy_rast
WHERE rid=2;
```

Output

2

This example uses three distinct shapes as the red, green, and blue bands of a synthetic raster. It compares the original RGB raster with band 2 followed by band 1 twice, and with band 3 alone.

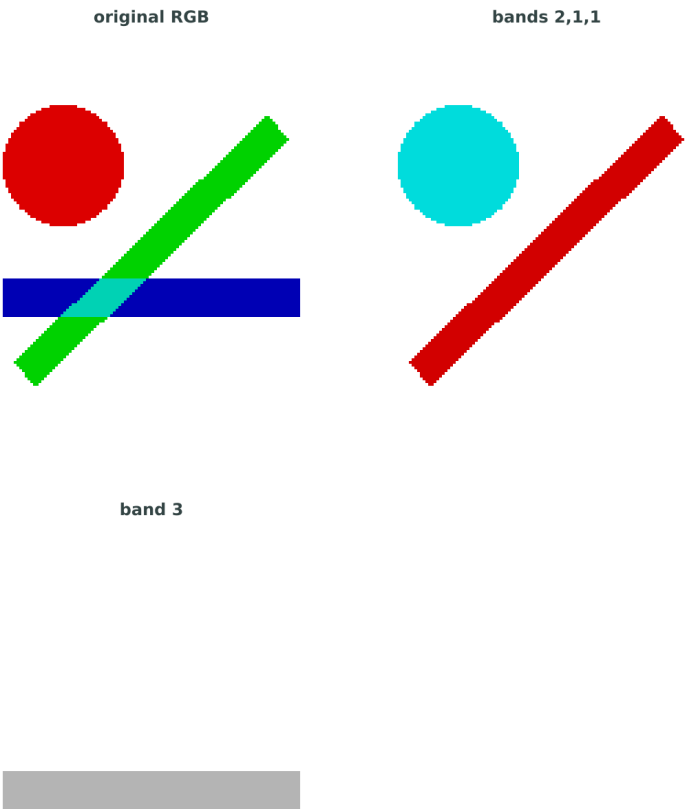
I Code

```

WITH canvas AS (
    SELECT ST_AddBand(
        ST_AddBand(
            ST_AddBand(
                ST_MakeEmptyRaster(128, 128, 0, 128, 1, -1, 0, 0, 0),
                '8BUI'::text, 0::double precision, 0::double precision
            ),
            '8BUI'::text, 0::double precision, 0::double precision
        ),
        '8BUI'::text, 0::double precision, 0::double precision
    ) AS rast
), bands AS (
    SELECT ST_SetValue(
        ST_SetValue(
            ST_SetValue(rast, 1, ST_Buffer('POINT(32 96)'::geometry, 22), 220),
            2, ST_Buffer('LINESTRING(18 20,110 110)'::geometry, 6,
                'endcap=flat join=bevel'), 210
        ),
        3, ST_Buffer('LINESTRING(10 48,118 48)'::geometry, 7,
            'endcap=flat'), 180
    ) AS rast
    FROM canvas
)
SELECT ST_AsPNG(rast) AS "original RGB",
       ST_AsPNG(ST_Band(rast, ARRAY[2, 1, 1])) AS "bands 2,1,1",
       ST_AsPNG(ST_Band(rast, 3)) AS "band 3"
FROM bands;

```


Figure



Band selection from a multi-band raster.



原始 (列图格)



双波段



图波段

相关信息

[ST_AddBand](#), [ST_NumBands](#), [ST_Reclass](#), Chapter 11

11.3.5 ST_MakeEmptyCoverage

ST_MakeEmptyCoverage — 用空图格图网格覆盖地理参考区域。

Synopsis

raster **ST_MakeEmptyCoverage**(integer tilewidth, integer tileheight, integer width, integer height, double precision upperleftx, double precision upperlefty, double precision scalex, double precision scaley, double precision skewx, double precision skewy, integer srid=unknown);

Description

使用 **ST_MakeEmptyRaster** 构建一个栅格瓦片。栅格度由度 (width) & 高度 (height) 确定，而瓦片度由瓦片度 (tilewidth) & 瓦片高度 (tileheight) 确定。覆盖的地理参考区域从左上角 (upperleftx, upperlefty) 延伸到右下角 (upperleftx + width * scalex, upperlefty + height * scaley)。



Note
注意，于栅格，scaley 通常为负，scalex 通常为正。所以右下角的 y 会比左上角低，x 会比左上角高。

可用性 : 2.4.0

基本示例

在 4x4 网格中构建 16 个栅格，以覆盖从左上角 (22, 77) 到右下角 (55, 33) 的 WGS84 区域。

Code

```
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;
```

Output

upperleftx	upperlefty	width	height	scalex	scaley	skewx	skewy	srid	numbands
22	33	1	1	8.25	-11	0	0	4326	0
30.25	33	1	1	8.25	-11	0	0	4326	0
38.5	33	1	1	8.25	-11	0	0	4326	0
46.75	33	1	1	8.25	-11	0	0	4326	0
22	22	1	1	8.25	-11	0	0	4326	0
30.25	22	1	1	8.25	-11	0	0	4326	0
38.5	22	1	1	8.25	-11	0	0	4326	0
46.75	22	1	1	8.25	-11	0	0	4326	0
22	11	1	1	8.25	-11	0	0	4326	0
30.25	11	1	1	8.25	-11	0	0	4326	0
38.5	11	1	1	8.25	-11	0	0	4326	0
46.75	11	1	1	8.25	-11	0	0	4326	0
22	0	1	1	8.25	-11	0	0	4326	0
30.25	0	1	1	8.25	-11	0	0	4326	0
38.5	0	1	1	8.25	-11	0	0	4326	0
46.75	0	1	1	8.25	-11	0	0	4326	0

相关信息

ST_MakeEmptyRaster

11.3.6 ST_MakeEmptyRaster

ST_MakeEmptyRaster — 返回一个空白栅格（不包含波段），其具有固定的尺寸（度和高度）、左上角 X 和 Y 坐标、像素大小和旋转参数 (scalex、scaley、skewx 和 skewy)，以及参考系 (SRID)。如果构建了一个栅格，返回一个具有相同大小、构建方式和 SRID 的新栅格。如果省略了 SRID，空参考将置未知 (0)。

Synopsis

```
raster ST_MakeEmptyRaster(raster rast);
raster ST_MakeEmptyRaster(integer width, integer height, float8 upperleftx, float8 upperlefty, float8
scalex, float8 scaley, float8 skewx, float8 skewy, integer srid=unknown);
raster ST_MakeEmptyRaster(integer width, integer height, float8 upperleftx, float8 upperlefty, float8
pixelsize);
```

Description

返回一个空白栅格（没有波段），其具有固定的尺寸（宽度和高度）并以空（或世界）坐进行地理参考，具有左上角 X 坐（upperleftx）、左上角 Y 坐（upperlefty）、像素大小和旋参数（scalex、scaley、skewx 和 skewy），以及参考系（SRID）。

最后一个版本使用个参数来指定像素大小（pixelsize）。scalex 置此参数，scaley 置此参数的。skewx 和 skewy 置 0。

如果入有栅格，它将返回具有相同元数据置（不波段）的新栅格。

如果未指定 srid，默 0。建空栅格后，您可能想要向其中添加波段并可能其行。参考 [ST_AddBand](#) 定波段，参考 [ST_SetValue](#) 置初始像素。

示例

Use an existing raster as a template and output metadata of the rasters we added.

Code

```
INSERT INTO dummy_rast(rid, rast)
VALUES(3, ST_MakeEmptyRaster(100, 100, 0.0005, 0.0005, 1, 1, 0, 0, 4326) );
INSERT INTO dummy_rast(rid, rast)
SELECT 4, ST_MakeEmptyRaster(rast)
FROM dummy_rast WHERE rid = 3;
SELECT
    rid,
    (md).width,
    (md).height,
    (md).scalex,
    (md).scaley,
    (md).srid
FROM (
    SELECT rid, ST_MetaData(rast) AS md
    FROM dummy_rast
    WHERE rid IN (3, 4)
) AS foo
ORDER BY rid;
```

Output

rid	width	height	scalex	scaley	srid
3	100	100	1	1	4326
4	100	100	1	1	4326

相关信息

[ST_AddBand](#), [ST_MetaData](#), [ST_ScaleX](#), [ST_ScaleY](#), [ST_SetValue](#), [ST_SkewX](#), , [ST_SkewY](#)

11.3.7 ST_Tile

ST_Tile — 返回根据输出图格的所需度分割输入图格而生的一组图格。

Synopsis

```
setof raster ST_Tile(raster rast, int[] nband, integer width, integer height, boolean padwithnodata=FALSE,
double precision nodataval=NULL);
setof raster ST_Tile(raster rast, integer nband, integer width, integer height, boolean padwithno-
data=FALSE, double precision nodataval=NULL);
setof raster ST_Tile(raster rast, integer width, integer height, boolean padwithnodata=FALSE, dou-
ble precision nodataval=NULL);
```

Description

返回根据输出图格的所需度分割输入图格而生的一组图格。

如果 `padwithnodata = FALSE`, 输出图格右部和底部的切片可能具有与其余切片不同的度。如果 `padwithnodata = TRUE`, 所有切片将具有相同的度, 并且切片可能会用 NODATA 填充。如果图格波段未指定 NODATA, 可以通过设置 `nodataval` 来指定 NODATA。



Note

如果输入图格的指定波段位于数据外, 输出图格中的相应波段也将位于数据外。

可用性 : 2.1.0

示例

Code

```
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
    row_number() OVER (
      ORDER BY ST_UpperLeftY(rast) DESC, ST_UpperLeftX(rast)
    ) AS tile_id,
    rast
  FROM (
    SELECT ST_Tile(rast, 3, 3, TRUE) AS rast
    FROM bar
  ) AS tiled
)
SELECT
  tile_id,
  ST_UpperLeftX(rast) AS ulx,
  ST_UpperLeftY(rast) AS uly,
  ST_Value(rast, 1, 1, 1) AS band1,
  ST_Value(rast, 2, 1, 1) AS band2
FROM baz;

```

Output

tile_id	ulx	uly	band1	band2
1	0	0	1	10
2	3	0	2	20
3	6	0	3	30
4	0	-3	4	40
5	3	-3	5	50
6	6	-3	6	60
7	0	-6	7	70
8	3	-6	8	80
9	6	-6	9	90

(9 rows)

Code

```

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
    row_number() OVER (
      ORDER BY ST_UpperLeftY(rast) DESC, ST_UpperLeftX(rast)
    ) AS tile_id,
    rast

```

```
FROM (
    SELECT ST_Tile(rast, ARRAY[2], 3, 3) AS rast
    FROM bar
) AS tiled
)
SELECT
    tile_id,
    ST_UpperLeftX(rast) AS ulx,
    ST_UpperLeftY(rast) AS uly,
    ST_Value(rast, 1, 1, 1) AS band_value
FROM baz;
```

Output

tile_id	ulx	uly	band_value
1	0	0	10
2	3	0	20
3	6	0	30
4	0	-3	40
5	3	-3	50
6	6	-3	60
7	0	-6	70
8	3	-6	80
9	6	-6	90

(9 rows)

相关信息

[ST_Union](#), [ST_Retile](#)

11.3.8 ST_Retile

ST_Retile — 从任意平面的栅格覆盖范围返回一个配置的平面。

Synopsis

SETOF raster **ST_Retile**(regclass tab, name col, geometry ext, float8 sfx, float8 sfy, int tw, int th, text algo='NearestNeighbor');

Description

返回一个具有指定比例 (sfx, sfy) 和最大尺寸 (tw, th) 并使用来自指定栅格覆盖范围 (tab, col) 的数据覆盖指定范围 (ext) 的栅格。

算法有：“NearestNeighbor”、“Bilinear”、“Cubic”、“CubicSpline”和“Lanczos”。有关更多信息，参见：[GDAL Warp 重采样方法](#)。

可用性：2.2.0

相关信息

[ST_CreateOverview](#)

11.3.9 ST_FromGDALRaster

ST_FromGDALRaster — 从受支持的 GDAL 栅格文件返回栅格。

Synopsis

raster **ST_FromGDALRaster**(bytea gdaldata, integer srid=NULL);

Description

从受支持的 GDAL 栅格文件返回栅格。gdaldata 是 bytea 型，栅格是 GDAL 栅格文件的内容。
如果 srid 是 NULL，函数将栅格自栅格从 GDAL 栅格分配 SRID。如果提供了 srid，提供的栅格将覆盖任何自分配的 SRID。
可用性：2.1.0

示例

Code

```
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;
```

Output

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)

相关信息

ST_AsGDALRaster

11.4 栅格处理器

11.4.1 ST_GeoReference

ST_GeoReference — 返回 GDAL 或 ESRI 格式的地理配准元数据，如世界文件中常用的格式。默认为 GDAL。

Synopsis

text **ST_GeoReference**(raster rast, text format=GDAL);

Description

返回地理配准元数据，包括 GDAL 或 ESRI 格式的回车符，如世界文件中常出现的那行。如果未指定类型，则默认为 GDAL。类型是字符串“GDAL”或“ESRI”。

格式表示的区间如下：

GDAL:

Output

```
scalex
skewy
skewx
scaley
upperleftx
upperlefty
```

ESRI:

Output

```
scalex
skewy
skewx
scaley
upperleftx + scalex*0.5
upperlefty + scaley*0.5
```

示例

Code

```
SELECT
    format,
    replace(ST_GeoReference(rast, format), E'\n', ' | ') AS georef
FROM dummy_rast
CROSS JOIN (VALUES ('ESRI'), ('GDAL')) AS formats(format)
WHERE rid = 1;
```

Output

```
format | georef
-----+-----
ESRI   | 2.0000000000 | 0.0000000000 | 0.0000000000 | 3.0000000000 | 1.5000000000 | 2.0000000000
GDAL   | 2.0000000000 | 0.0000000000 | 0.0000000000 | 3.0000000000 | 0.5000000000 | 0.5000000000
(2 rows)
```

相关信息

[ST_SetGeoReference](#), [ST_ScaleX](#), [ST_ScaleY](#)

11.4.2 ST_Height

ST_Height — 返回栅格的高度（以像素为单位）。

Synopsis

integer **ST_Height**(raster rast);

Description

返回栅格的高度（以像素单位）。

示例

Code

```
SELECT rid, ST_Height(rast) As rastheight
FROM dummy_rast;
```

Output

rid	rastheight
1	20
2	5

相关信息

[ST_Width](#)

11.4.3 ST_IsEmpty

ST_IsEmpty — 如果栅格空（宽度 = 0 且高度 = 0），返回 true。否则，返回 false。

Synopsis

boolean **ST_IsEmpty**(raster rast);

Description

如果栅格空（宽度 = 0 且高度 = 0），返回 true。否则，返回 false。

可用性: 2.0.0

示例

Code

```
SELECT ST_IsEmpty(ST_MakeEmptyRaster(100, 100, 0, 0, 0, 0, 0, 0))
```

| **Output**

f

| **Code**

```
SELECT ST_IsEmpty(ST_MakeEmptyRaster(0, 0, 0, 0, 0, 0, 0, 0, 0))
```

| **Output**

t

相关信息

[ST_HasNoBand](#)

11.4.4 ST_MemSize

ST_MemSize — 返回栅格占用的空量（以字节位）。

Synopsis

integer **ST_MemSize**(raster rast);

Description

返回栅格占用的空量（以字节位）。

是 PostgreSQL 内置函数 `pg_column_size`、`pg_size_pretty`、`pg_relation_size`、`pg_total_relation_size` 的一个很好的补充。



Note

出表的字节大小的 `pg_relation_size` 可能返回小于 `ST_MemSize` 的字节大小。是因 `pg_relation_size` 不会添加 toasted 表贡献，并且大型几何形存储在 TOAST 表中。`pg_column_size` 可能返回低的，因它返回后的大小。
`pg_total_relation_size` - 包括表、表和索引。

可用性 : 2.2.0

示例

| **Code**

```
SELECT ST_MemSize(ST_AsRaster(ST_Buffer(ST_Point(1, 5), 10, 1000), 150, 150, '8BUI')) AS rast_mem;
```

| **Output**

22568

相关信息

11.4.5 ST_MetaData

ST_MetaData — 返回有关栅格象的基本元数据，例如像素大小、旋 (斜)、左上、左下等。

Synopsis

record **ST_MetaData**(raster rast);

Description

返回有关栅格象的基本元数据，例如像素大小、旋 (斜)、左上、左下等。返回的列：upperleftx | upperlefty | width | height | scalex | scaley | skewx | skewy | srid | numbands

示例

| **Code**

```
SELECT
    rid,
    format('%s, %s', (md).upperleftx, (md).upperlefty) AS upper_left,
    format('%sx%s', (md).width, (md).height) AS size,
    format('%s, %s', (md).scalex, (md).scaley) AS scale,
    format('%s, %s', (md).skewx, (md).skewy) AS skew,
    (md).srid AS srid,
    (md).numbands AS numbands
FROM (
    SELECT rid, ST_MetaData(rast) AS md
    FROM dummy_rast
) AS foo
ORDER BY rid;
```

| **Output**

rid	upper_left	size	scale	skew	srid	numbands
1	(0.5, 0.5)	10x20	(2, 3)	(0, 0)	0	0
2	(3427927.75, 5793244)	5x5	(0.05, -0.05)	(0, 0)	0	3

(2 rows)

相关信息

[ST_BandMetaData](#), [ST_NumBands](#)

11.4.6 ST_NumBands

ST_NumBands — 返回栅格象中的波段数。

Synopsis

integer **ST_NumBands**(raster rast);

Description

返回栅格象中的波段数。

示例

Code

```
SELECT rid, ST_NumBands(rast) As numbands
FROM dummy_rast;
```

Output

rid	numbands
1	0
2	3

相关信息

[ST_Value](#)

11.4.7 ST_PixelHeight

ST_PixelHeight — 返回空参考系的几何位的像素高度。

Synopsis

double precision **ST_PixelHeight**(raster rast);

Description

以空参考系的几何位返回像素的高度。在没有斜的常情况下，像素高度只是几何坐与光像素之的比例。

参 [ST_PixelWidth](#) 以直地了解关系。

示例

Rasters with no skew.

Code

```
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;
```

| **Output**

rastheight	pixheight	scalex	scaley	skewx	skewy
20	3	2	3	0	0
5	0.05	0.05	-0.05	0	0

Rasters with skew different than 0.

| **Code**

```
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;
```

| **Output**

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

相关信息

[ST_PixelWidth](#), [ST_ScaleX](#), [ST_ScaleY](#), [ST_SkewX](#), [ST_SkewY](#)

11.4.8 ST_PixelWidth

ST_PixelWidth — 返回空参考系的几何位置的像素度。

Synopsis

double precision **ST_PixelWidth**(raster rast);

Description

返回空参考系的几何位置中像素的度。在没有斜的情况下，像素度只是几何坐与光像素之的比例。

The following executable example draws the two pixel basis vectors on the same coordinate frame as a skewed raster footprint. The i direction vector is (scaleX, skewY); the j direction vector is (skewX, scaleY).

示例

Pixel basis vectors and footprint for a raster with non-zero skew.

| **Code**

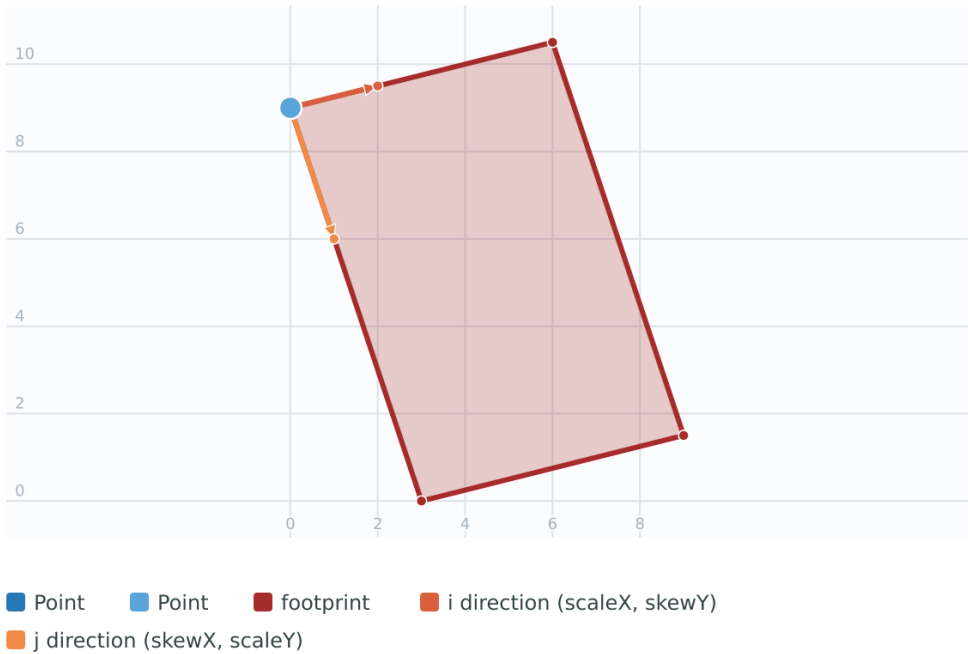
```
WITH raster AS (
  SELECT ST_MakeEmptyRaster(
    3, 3, 0, 0,
    2, -3, 1, 0.5,
    0
```

```
    ) AS rast
)
SELECT
  ST_ConvexHull(rast) AS footprint,
  ST_MakeLine(
    ST_Point(0, 0),
    ST_Point(ST_ScaleX(rast), ST_SkewY(rast))
  ) AS "i direction (scaleX, skewY)",
  ST_MakeLine(
    ST_Point(0, 0),
    ST_Point(ST_SkewX(rast), ST_ScaleY(rast))
  ) AS "j direction (skewX, scaleY)"
FROM raster;
```

Output

POLYGON((0 0,6 1.5,9 -7.5,3 -9,0 0)) | LINESTRING(0 0,2 0.5) | LINESTRING(0 0,1 -3)

Figure



Rasters with no skew.

Code

```
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;
```

Output

rastwidth	pixwidth	scalex	scaley	skewx	skewy
10	2	2	3	0	0
5	0.05	0.05	-0.05	0	0

Rasters with skew different than 0.

Code

```
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;
```

Output

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

相关信息

[ST_PixelHeight](#), [ST_ScaleX](#), [ST_ScaleY](#), [ST_SkewX](#), [ST_SkewY](#)

11.4.9 ST_ScaleX

ST_ScaleX — 返回像素宽度的 X 分量（以坐​​标参考系为单位）。

Synopsis

float8 **ST_ScaleX**(raster rast);

Description

返回像素宽度的 X 分量（以坐​​标参考系为单位）。有关更多信息，参​​见 [世界文件](#)。
更改：2.0.0。在 WKTRaster 版本中，称​​为 ST_PixelSizeX。

示例

Code

```
SELECT rid, ST_ScaleX(rast) As rastpixwidth
FROM dummy_rast;
```

Output

rid	rastpixwidth
1	2
2	0.05

相关信息

[ST_Width](#)

11.4.10 ST_ScaleY

ST_ScaleY — 返回像素高度的 Y 分量（以坐¹参考系²位³）。

Synopsis

float8 **ST_ScaleY**(raster rast);

Description

返回像素高度的 Y 分量（以坐¹参考系²位³）。可能是⁴。有关更多⁵信息，⁶参⁷世界文件⁸。

更改：2.0.0。在 WKTRaster 版本中，⁹称¹⁰ ST_PixelSizeY。

示例

Code

```
SELECT rid, ST_ScaleY(rast) As rastpixheight
FROM dummy_rast;
```

Output

rid	rastpixheight
1	3
2	-0.05

相关信息

[ST_Height](#)

11.4.11 ST_RasterToWorldCoord

ST_RasterToWorldCoord — 在¹定列和行的情况下，以几何 X 和 Y（²度和³度）形式返回⁴格的左上角。列和行从 1 开始。

Synopsis

record **ST_RasterToWorldCoord**(raster rast, integer xcolumn, integer yrow);

Description

在¹定列和行的情况下，以几何 X 和 Y（²度和³度）形式返回左上角。返回的 X 和 Y 采用地理参考⁴格的几何⁵位。列和行的⁶号从 1 开始，但如果向任一参数⁷零、⁸数或大于⁹格相¹⁰度的数字，¹¹假定¹²格网格适用于¹³格¹⁴界之外，它将返回¹⁵格外部的坐¹⁶。

可用性：2.1.0

示例

This example uses a non-skewed raster.

Code

```
SELECT
    rid,
    (ST_RasterToWorldCoord(rast, 1, 1)).*,
    (ST_RasterToWorldCoord(rast, 2, 2)).*
FROM dummy_rast
```

Output

rid	longitude	latitude	longitude	latitude
1	0.5	0.5	2.5	3.5
2	3427927.75	5793244	3427927.8	5793243.95

This example uses a skewed raster.

Code

```
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
```

Output

rid	longitude	latitude	longitude	latitude
1	0.5	0.5	203.5	6.5
2	3427927.75	5793244	3428128.8	5793243.9

相关信息

[ST_RasterToWorldCoordX](#), [ST_RasterToWorldCoordY](#), [ST_SetSkew](#)

11.4.12 ST_RasterToWorldCoordX

ST_RasterToWorldCoordX — 返回栅格、列和行左上角的几何 X 坐标。列和行的编号从 1 开始。

Synopsis

```
float8 ST_RasterToWorldCoordX(raster rast, integer xcolumn);
float8 ST_RasterToWorldCoordX(raster rast, integer xcolumn, integer yrow);
```

Description

返回栅格列中一行的左上角 X 坐标，以地理参考栅格的几何位置表示。列和行的编号从 1 开始，但如果输入一个负数或大于栅格中列的数量的数字，它将输出栅格文件外部左或右的坐标，假设斜度和像素大小与所输入的栅格相同。



Note
对于非斜格，提供 X 列就足够了。对于斜格，地理参考坐标是 ST_ScaleX 和 ST_SkewX 以及行和列的函数。如果您只对斜格提供 X 列，它会出错。

更改：2.1.0 在之前的版本中，称 ST_RasterToWorldCoordX

示例

This example shows that providing a column is sufficient for a non-skewed raster.

Code

```
SELECT rid, ST_RasterToWorldCoordX(rast, 1) As x1coord,
       ST_RasterToWorldCoordX(rast, 2) As x2coord,
       ST_ScaleX(rast) As pixelx
FROM dummy_rast;
```

Output

rid	x1coord	x2coord	pixelx
1	0.5	2.5	2
2	3427927.75	3427927.8	0.05

For fun let’s skew it.

Code

```
SELECT rid, ST_RasterToWorldCoordX(rast, 1, 1) As x1coord,
       ST_RasterToWorldCoordX(rast, 2, 3) As x2coord,
       ST_ScaleX(rast) As pixelx
FROM (SELECT rid, ST_SetSkew(rast, 100.5, 0) As rast FROM dummy_rast) As foo;
```

Output

rid	x1coord	x2coord	pixelx
1	0.5	203.5	2
2	3427927.75	3428128.8	0.05

相关信息

ST_ScaleX, ST_RasterToWorldCoordY, ST_SetSkew, ST_SkewX

11.4.13 ST_RasterToWorldCoordY

ST_RasterToWorldCoordY — 返回格、列和行的左上角的几何 Y 坐标。列和行的号从 1 开始。

Synopsis

```
float8 ST_RasterToWorldCoordY(raster rast, integer yrow);
float8 ST_RasterToWorldCoordY(raster rast, integer xcolumn, integer yrow);
```

Description

以地理参考栅格的几何位置返回栅格列行的左上角 Y 坐标。列和行的编号从 1 开始，但如果您输入数字或高于栅格中的列/行数的数字，它会为您提供栅格文件外部向左或向右的坐标，并假设栅格斜和像素尺寸与指定的栅格相同。



Note
对于非斜栅格，提供 Y 列就足够了。对于斜栅格，地理参考坐标是 ST_ScaleY 和 ST_SkewY 以及行和列的函数。如果您输出斜栅格的 Y 行，它会输出。

更改：2.1.0 在之前的版本中，名称 ST_Raster2WorldCoordY

示例

Non-skewed raster providing row is sufficient.

Code

```
SELECT rid, ST_RasterToWorldCoordY(rast, 1) As ylcoord,
       ST_RasterToWorldCoordY(rast, 3) As y2coord,
       ST_ScaleY(rast) As pixely
FROM dummy_rast;
```

Output

rid	ylcoord	y2coord	pixely
1	0.5	6.5	3
2	5793244	5793243.9	-0.05

For fun let's skew it.

Code

```
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;
```

Output

rid	ylcoord	y2coord	pixely
1	0.5	107	3
2	5793244	5793344.4	-0.05

相关信息

ST_ScaleY, ST_RasterToWorldCoordX, ST_SetSkew, ST_SkewY

11.4.14 ST_Rotation

ST_Rotation — 返回栅格的旋转弧度。

Synopsis

float8 ST_Rotation(raster rast);

Description

返回栅格的均匀旋弧度。如果栅格没有均匀旋，返回 NaN。有关更多信息，参世界文件。

示例

Code

```
SELECT rid, ST_Rotation(ST_SetScale(ST_SetSkew(rast, sqrt(2)), sqrt(2))) AS rot FROM dummy_rast;
```

Output

rid	rot
1	0.785398163397448
2	0.785398163397448

相关信息

ST_SetRotation, ST_SetScale, ST_SetSkew

11.4.15 ST_SkewX

ST_SkewX — 返回地理参考 X 斜（或旋参数）。

Synopsis

float8 ST_SkewX(raster rast);

Description

返回地理参考 X 斜（或旋参数）。有关更多信息，参世界文件。

示例

Code

```
WITH refs AS (
  SELECT
    rid,
    format('(%s, %s)', ST_SkewX(rast), ST_SkewY(rast)) AS skew,
    string_to_array(ST_GeoReference(rast), E'\n') AS georef
  FROM dummy_rast
)
SELECT
  rid,
  skew,
  term,
  georef[idx] AS value
FROM refs
CROSS JOIN (VALUES
  (1, 'scaleX'),
```

```
    (2, 'skewY'),
    (3, 'skewX'),
    (4, 'scaleY'),
    (5, 'originX'),
    (6, 'originY')
) AS parts(idx, term)
ORDER BY rid, idx;
```

Output

rid	skew	term	value
1	(0, 0)	scaleX	2.0000000000
1	(0, 0)	skewY	0.0000000000
1	(0, 0)	skewX	0.0000000000
1	(0, 0)	scaleY	3.0000000000
1	(0, 0)	originX	0.5000000000
1	(0, 0)	originY	0.5000000000
2	(0, 0)	scaleX	0.0500000000
2	(0, 0)	skewY	0.0000000000
2	(0, 0)	skewX	0.0000000000
2	(0, 0)	scaleY	-0.0500000000
2	(0, 0)	originX	3427927.7500000000
2	(0, 0)	originY	5793244.0000000000

(12 rows)

相关信息

[ST_GeoReference](#), [ST_SkewY](#), [ST_SetSkew](#)

11.4.16 ST_SkewY

ST_SkewY — 返回地理参考 Y 轴斜（或旋角参数）。

Synopsis

float8 **ST_SkewY**(raster rast);

Description

返回地理参考 Y 轴斜（或旋角参数）。有关更多信息，参看[世界文件](#)。

示例

Code

```
WITH refs AS (
    SELECT
        rid,
        format('(%s, %s)', ST_SkewX(rast), ST_SkewY(rast)) AS skew,
        string_to_array(ST_GeoReference(rast), E'\n') AS georef
    FROM dummy_rast
)
SELECT
    rid,
    skew,
    term,
```

```
georef[idx] AS value
FROM refs
CROSS JOIN (VALUES
  (1, 'scaleX'),
  (2, 'skewY'),
  (3, 'skewX'),
  (4, 'scaleY'),
  (5, 'originX'),
  (6, 'originY')
) AS parts(idx, term)
ORDER BY rid, idx;
```

Output

rid	skew	term	value
1	(0, 0)	scaleX	2.0000000000
1	(0, 0)	skewY	0.0000000000
1	(0, 0)	skewX	0.0000000000
1	(0, 0)	scaleY	3.0000000000
1	(0, 0)	originX	0.5000000000
1	(0, 0)	originY	0.5000000000
2	(0, 0)	scaleX	0.0500000000
2	(0, 0)	skewY	0.0000000000
2	(0, 0)	skewX	0.0000000000
2	(0, 0)	scaleY	-0.0500000000
2	(0, 0)	originX	3427927.7500000000
2	(0, 0)	originY	5793244.0000000000

(12 rows)

相关信息

ST_GeoReference, ST_SkewX, ST_SetSkew

11.4.17 ST_SRID

ST_SRID — 返回在 spatial_ref_sys 表中定义的栅格的空参考标识符。

Synopsis

integer **ST_SRID**(raster rast);

Description

返回在 Spatial_ref_sys 表中定义的栅格对象的空参考标识符。



Note
未做地理参考的 raster 和 geometry 使用 SRID 0。

示例

Code

```
SELECT ST_SRID(rast) As srid
FROM dummy_rast WHERE rid=1;
```

Output

```
0
```

相关信息

Section [4.5](#), [ST_SRID](#)

11.4.18 ST_Summary

ST_Summary — 返回栅格内容的文本摘要。

Synopsis

```
text ST_Summary(raster rast);
```

Description

返回栅格内容的文本摘要。

可用性 : 2.1.0

示例

Code

```
SELECT ST_Summary(ST_AddBand(ST_AddBand(ST_AddBand(ST_MakeEmptyRaster(10, 10, 0, 0, 1, -1, 0, 0, 0),
    1, '8BUI', 1, 0
    ),
    2, '32BF', 0, -9999
    ),
    3, '16BSI', 0, NULL
    )
);
```

Output

```
Raster of 10x10 pixels has 3 bands and extent of BOX(0 -10,10 0)+
band 1 of pixtype 8BUI is in-db with NODATA value of 0      +
band 2 of pixtype 32BF is in-db with NODATA value of -9999  +
band 3 of pixtype 16BSI is in-db with no NODATA value
(1 row)
```

相关信息

[ST_MetaData](#), [ST_BandMetaData](#), [ST_Summary](#) [ST_Extent](#)

11.4.19 ST_UpperLeftX

ST_UpperLeftX — 返回投影空 $\square$ 参考中 $\square$ 格的左上角 X 坐 $\square$ 。

Synopsis

float8 **ST_UpperLeftX**(raster rast);

Description

返回投影空 $\square$ 参考中 $\square$ 格的左上角 X 坐 $\square$ 。

示例

Code

```
SELECT rid, ST_UpperLeftX(rast) As ulx
FROM dummy_rast;
```

Output

rid	ulx
1	0.5
2	3427927.75

相关信息

[ST_UpperLeftY](#), [ST_GeoReference](#), [Box3D](#)

11.4.20 ST_UpperLeftY

ST_UpperLeftY — 返回投影空 $\square$ 参考中 $\square$ 格的左上角 Y 坐 $\square$ 。

Synopsis

float8 **ST_UpperLeftY**(raster rast);

Description

返回投影空 $\square$ 参考中 $\square$ 格的左上角 Y 坐 $\square$ 。

示例

Code

```
SELECT rid, ST_UpperLeftY(rast) As uly
FROM dummy_rast;
```

Output

rid	uly
1	0.5
2	5793244

相关信息

[ST_UpperLeftX](#), [ST_GeoReference](#), [Box3D](#)

11.4.21 ST_Width

ST_Width — 返回栅格的宽度（以像素为单位）。

Synopsis

integer **ST_Width**(raster rast);

Description

返回栅格的宽度（以像素为单位）。

示例

Code

```
SELECT ST_Width(rast) As rastwidth
FROM dummy_rast WHERE rid=1;
```

Output

10

相关信息

[ST_Height](#)

11.4.22 ST_WorldToRasterCoord

ST_WorldToRasterCoord — 给定几何 X 和 Y（经度和纬度）或以栅格的空参考坐标系表示的点几何，将左上角作列和行返回。

Synopsis

record **ST_WorldToRasterCoord**(raster rast, geometry pt);
record **ST_WorldToRasterCoord**(raster rast, double precision longitude, double precision latitude);

Description

在给定几何 X 和 Y（经度和纬度）或点几何的情况下，将左上角作列和行返回。无几何 X 和 Y 或点几何是否位于栅格范围之外，此函数都会起作用。几何 X 和 Y 必在栅格的空参考坐标系中表示。
可用性：2.1.0

示例

Code

```
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;
```

Output

rid	columnx	rowy	columnx	rowy
1	1713964	7	1713964	7
2	2	115864471	2	115864471

相关信息

[ST_WorldToRasterCoordX](#), [ST_WorldToRasterCoordY](#), [ST_RasterToWorldCoordX](#), [ST_RasterToWorldCoordY](#), [ST_SRID](#)

11.4.23 ST_WorldToRasterCoordX

ST_WorldToRasterCoordX — 返回栅格中点几何 (pt) 的列或以栅格世界空参考系表示的 X 和 Y 世界坐标 (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);

Description

返回点几何 (pt) 或 X 和 Y 世界坐标 (xw, yw) 的栅格中的列。一个点，或者（如果栅格斜，需要 xw 和 yw 世界坐标）。如果栅格没有斜，xw 就足够了。世界坐标位于栅格的空参考坐标系中。
更改：2.1.0 在之前的版本中，称 ST_World2RasterCoordX

示例

Code

```
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;
```

Output

rid	xcoord	xcoord_xwyw	ptxcoord
1	1713964	1713964	1713964
2	1	1	1

相关信息

[ST_RasterToWorldCoordX](#), [ST_RasterToWorldCoordY](#), [ST_SRID](#)

11.4.24 ST_WorldToRasterCoordY

ST_WorldToRasterCoordY — 返回点几何 (pt) 的栅格中的行或以栅格的世界空间参考系表示的 X 和 Y 世界坐标 (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);
```

Description

返回点几何 (pt) 或 X 和 Y 世界坐标 (xw, yw) 的栅格中的行。一个点，或者（如果栅格斜，需要 xw 和 yw 世界坐标）。如果栅格没有斜，xw 就足够了。世界坐标位于栅格的空间参考系中。

更改：2.1.0 在之前的版本中，称 ST_World2RasterCoordY

示例

Code

```
SELECT rid, ST_WorldToRasterCoordY(rast, 20.5) As ycoord,
       ST_WorldToRasterCoordY(rast, 3427927.8, 20.5) As ycoord_xwyw,
       ST_WorldToRasterCoordY(rast, ST_GeomFromText('POINT(3427927.8 20.5)', ST_SRID(rast) ↵
) As ptycoord
FROM dummy_rast;
```

Output

rid	ycoord	ycoord_xwyw	ptycoord
1	7	7	7
2	115864471	115864471	115864471

相关信息

[ST_RasterToWorldCoordX](#), [ST_RasterToWorldCoordY](#), [ST_SRID](#)

11.5 栅格波段管理器

11.5.1 ST_BandMetaData

`ST_BandMetaData` — 返回特定栅格波段的基本元数据。如果未指定，假定波段号 1。

Synopsis

- (1) record `ST_BandMetaData`(raster rast, integer band=1);
- (2) record `ST_BandMetaData`(raster rast, integer[] band);

Description

返回有关栅格波段的基本元数据。返回的列：`pixeltype`、`nodatavalue`、`isoutdb`、`path`、`outdbbandnum`、`filesize`、`filetimestamp`。



Note

如果栅格不包含波段，会引发错误。



Note

如果 band 没有 NODATA 值，`nodatavalue` 为 NULL。



Note

如果 `isoutdb` 为 False，路径、`outdbbandnum`、文件大小和文件戳为 NULL。如果禁用 `outdb` 值，文件大小和文件戳也将为 NULL。

增量：2.5.0 包括 `outdbbandnum`、文件大小和文件戳 for `outdb` 栅格。

示例

Variant 1.

Code

```
SELECT
  rid,
  (foo.md).*
FROM (
  SELECT
    rid,
    ST_BandMetaData(rast, 1) AS md
  FROM dummy_rast
  WHERE rid=2
) AS foo;
```

Output

rid	pixeltype	nodatavalue	isoutdb	path	outdbbandnum
2	8BUI		0	f	
(1 row)					

Variant 2.

Code

```
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  
    bandnum,  
    pixeltype,  
    isoutdb,  
    regexp_replace(path, '^.*/', '') AS file,  
    outdbbandnum  
FROM ST_BandMetaData((SELECT rast FROM foo),  
    ARRAY[1, 3, 2]::int[]  
);
```

Output

bandnum	pixeltype	isoutdb	file	outdbbandnum
1	8BUI	t	Projected.tif	1
3	8BUI	t	Projected.tif	3
2	8BUI	t	Projected.tif	2

相关信息

ST_MetaData, ST_BandPixelType

11.5.2 ST_BandNoDataValue

ST_BandNoDataValue — 返回表示无数据的指定波段中的数。如果没有指定波段号，默认为波段 1。

Synopsis

double precision **ST_BandNoDataValue**(raster rast, integer bandnum=1);

Description

返回表示指定波段中无数据的数

示例

Code

```
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;
```

Output

bnval1	bnval2	bnval3
0	0	0

相关信息

[ST_NumBands](#)

11.5.3 ST_BandIsNoData

ST_BandIsNoData — 如果波段填充无数据，返回 true。

Synopsis

boolean **ST_BandIsNoData**(raster rast, integer band, boolean forceChecking=false);
boolean **ST_BandIsNoData**(raster rast, boolean forceChecking=false);

Description

如果波段填充无数据，返回 true。如果未指定，假定波段 1。如果最后一个参数 TRUE，逐像素整个波段。否，函数返回的 isnodata 志的。如果未指定，此参数的默认值 FALSE。

可用性: 2.0.0



Note

If the flag is dirty (this is, the result is different using TRUE as last parameter and not using it) you should update the raster to set this flag to true, by using ST_SetBandIsNoData(), or ST_SetBandNoDataValue() with TRUE as last argument. See [ST_SetBandIsNoData](#).

示例

This example creates a dummy table with one raster column. The raster has two bands and one pixel per band. In the first band, the NODATA value and pixel value are both 3. In the second band, the NODATA value is 13 and the pixel value is 4.

Code

```
create table dummy_rast (rid integer, rast raster);

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

```

相关信息

[ST_BandNoDataValue](#), [ST_NumBands](#), [ST_SetBandNoDataValue](#), [ST_SetBandIsNoData](#)

11.5.4 ST_BandPath

ST_BandPath — 返回存储在文件系统中的波段的系文件路径。如果未指定波段号，假定 1。

Synopsis

text **ST_BandPath**(raster rast, integer bandnum=1);

Description

返回指定波段的系文件路径。如果用使用数据中的波段，会抛出。

示例

Code

```
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 bandnum, ST_BandPath(rast, bandnum) AS path  
FROM foo  
CROSS JOIN generate_series(1, 3) AS bandnum;
```

Output

bandnum	path
1	/home/pele/devel/geo/postgis-git/raster/test/regress/loader/Projected.tif
2	/home/pele/devel/geo/postgis-git/raster/test/regress/loader/Projected.tif
3	/home/pele/devel/geo/postgis-git/raster/test/regress/loader/Projected.tif

相关信息

[ST_BandMetaData](#), [ST_SetBandPath](#), [ST_SetBandIndex](#)

11.5.5 ST_BandFileSize

ST_BandFileSize — 返回文件系中存的波段的文件大小。如果未指定波段号，假定 1。

Synopsis

bigint **ST_BandFileSize**(raster rast, integer bandnum=1);

Description

返回文件系中存的波段的文件大小。如果使用 in db band 用，或者未用 outdb ，会引。
此函数通常与 ST_BandPath() 和 ST_BandFileTimestamp() 合使用，以便客端可以确定它看到的 outdb 格文件名是否与服器看到的文件名相同。

可用性：2.5.0

示例

Code

```
SELECT ST_BandFileSize(rast, 1) FROM dummy_rast WHERE rid = 1;
```

Output

```
240574
```

11.5.6 ST_BandFileTimestamp

`ST_BandFileTimestamp` — 返回文件系中存_{in}的波段的文件戳。如果未指定波段号，假定_{in} 1。

Synopsis

`bigint ST_BandFileTimestamp(raster rast, integer bandnum=1);`

Description

返回文件系中存_{in}的波段的文件戳（自 1970 年 1 月 1 日 00:00:00 UTC 以来的秒数）。如果使用 `in db band` 用，或者未_{in}用 `outdb` 用，会引_{in}。

此函数通常与 `ST_BandPath()` 和 `ST_BandFileSize()` 合使用，以便客_{in}端可以确定它所看到的 `outdb` 格的文件名是否与服_{in}器所看到的相同。

可用性：2.5.0

示例**Code**

```
SELECT ST_BandFileTimestamp(rast, 1) FROM dummy_rast WHERE rid = 1;
```

Output

```
1521807257
```

11.5.7 ST_BandPixelType

`ST_BandPixelType` — 返回_{in}定波段的像素型。如果未指定波段号，假定_{in} 1。

Synopsis

`text ST_BandPixelType(raster rast, integer bandnum=1);`

Description

返回描述栅格区每个元格中存储的数据型和大小的名称。

有 11 种像素型。支持的像素型如下：

- 1BB - 1 位布尔
- 2BUI - 2 位无符号整数
- 4BUI - 4 位无符号整数
- 8BSI - 8 位有符号整数
- 8BUI - 8 位无符号整数
- 16BSI - 16 位有符号整数
- 16BUI - 16 位无符号整数
- 32BSI - 32 位有符号整数
- 32BUI - 32 位无符号整数
- 32BF - 32 位浮点数
- 64BF - 64 位浮点数

示例

Code

```
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;
```

Output

btype1	btype2	btype3
8BUI	8BUI	8BUI

相关信息

ST_NumBands

11.5.8 ST_MinPossibleValue

ST_MinPossibleValue — 返回此像素型可以存储的最小值。

Synopsis

integer **ST_MinPossibleValue**(text pixeltype);

Description

返回此像素☐型可以存☐的最小☐。

示例

Code

```
SELECT ST_MinPossibleValue('16BSI');
```

Output

```
-32768
```

Code

```
SELECT ST_MinPossibleValue('8BUI');
```

Output

```
0
```

相关信息

[ST_BandPixelType](#)

11.5.9 ST_HasNoBand

ST_HasNoBand — 如果不存在具有☐定波段☐号的波段，☐返回 **true**。如果未指定波段号，☐假定波段号☐ 1。

Synopsis

boolean ST_HasNoBand(raster rast, integer bandnum=1);

Description

如果不存在具有☐定波段☐号的波段，☐返回 **true**。如果未指定波段号，☐假定波段号☐ 1。

可用性: 2.0.0

示例

Code

```
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;
```

| **Output**

rid	hb1	hb2	hb4	numbands
1	t	t	t	0
2	f	f	t	3

相关信息

[ST_NumBands](#)

11.6 栅格像素聚合器和置器

11.6.1 ST_PixelAsPolygon

ST_PixelAsPolygon — 返回限定特定行和列的像素的多边形几何形状。

Synopsis

geometry **ST_PixelAsPolygon**(raster rast, integer columnx, integer rowy);

Description

返回限定特定行和列的像素的多边形几何形状。

可用性: 2.0.0

示例

This example returns a raster pixel polygon.

| **Code**

```
SELECT i, j, ST_PixelAsPolygon(foo.rast, i, j) As b1pgeom
FROM dummy_rast As foo
      CROSS JOIN generate_series(1, 2) As i
      CROSS JOIN generate_series(1, 1) As j
WHERE rid=2;
```

| **Output**

i	j	b1pgeom
1	1	POLYGON(((3427927.75 5793244,3427927.8 5793244,3427927.8 5793243.95,...
2	1	POLYGON(((3427927.8 5793244,3427927.85 5793244,3427927.85 5793243.95, ..

相关信息

[ST_DumpAsPolygons](#), [ST_PixelAsPolygons](#), [ST_PixelAsPoint](#), [ST_PixelAsPoints](#), [ST_PixelAsCentroid](#), [ST_PixelAsCentroids](#), [ST_Intersection](#), [ST_AsText](#)

11.6.2 ST_PixelAsPolygons

ST_PixelAsPolygons — 返回包栅格的每个像素的多边形几何图形以及每个像素的val、X 和 Y 栅格坐标。

Synopsis

setof record **ST_PixelAsPolygons**(raster rast, integer band=1, boolean exclude_nodata_value=TRUE);

Description

返回包栅格的每个像素的多边形几何图形以及每个像素的val（双精度）、X 和 Y 栅格坐标（整数）。

返回格式：geom geometry, val 双精度, x 整数, y 整数。



Note

When *exclude_nodata_value* is TRUE, only pixels whose values are not NODATA are returned. Unlike **ST_DumpAsPolygons**, this function returns one polygon per pixel rather than merging adjacent pixels with the same value.

可用性: 2.0.0

增：2.1.0 添加了 *except_nodata_value* 可参数。

更改：2.1.1 更改了 *except_nodata_value* 的行。

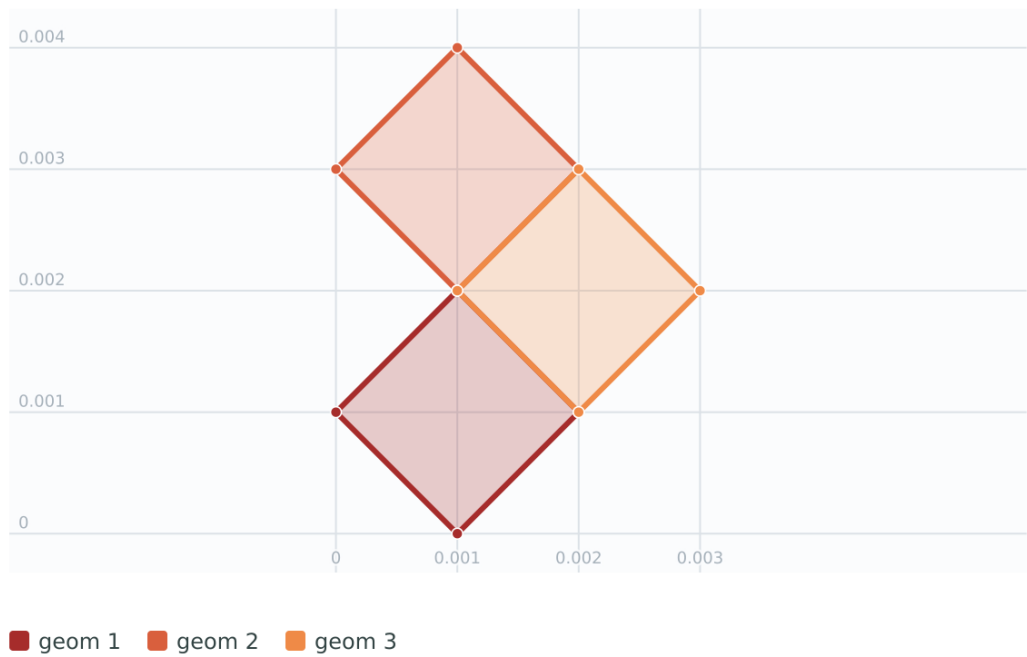
示例

This example returns a raster pixel polygon.

Code

```
WITH raster AS (
    SELECT ST_SetValue(
        ST_SetValue(
            ST_AddBand(
                ST_MakeEmptyRaster(
                    2, 2, 0, 0, 0.001, -0.001, 0.001, 0.001, 4269
                ),
                1, '8BUI', 1, 0
            ),
            2, 2, 10
        ),
        1, 1, NULL
    ) AS rast
), pixels AS (
    SELECT (ST_PixelAsPolygons(rast)).*
    FROM raster
)
SELECT x, y, val, geom AS geom
FROM pixels
ORDER BY x, y;
```

Figure



相关信息

[ST_DumpAsPolygons](#), [ST_PixelAsPolygon](#), [ST_PixelAsPoint](#), [ST_PixelAsPoints](#), [ST_PixelAsCentroid](#), [ST_PixelAsText](#)

11.6.3 ST_PixelAsPoint

ST_PixelAsPoint — 返回像素左上角的点几何形状。

Synopsis

geometry **ST_PixelAsPoint**(raster rast, integer columnx, integer rowy);

Description

返回像素左上角的点几何形状。

可用性：2.1.0

示例

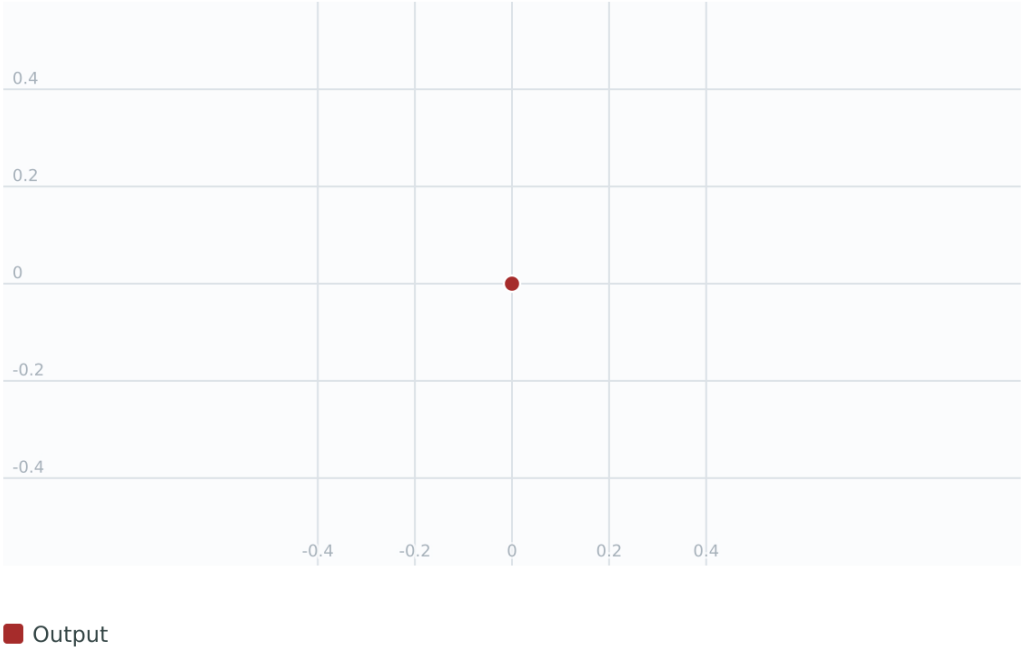
Code

```
SELECT ST_PixelAsPoint(rast, 1, 1) FROM dummy_rast WHERE rid = 1;
```

| **Output**

```
POINT(0.5 0.5)
```

| **Figure**



相关信息

[ST_DumpAsPolygons](#), [ST_PixelAsPolygon](#), [ST_PixelAsPolygons](#), [ST_PixelAsPoints](#), [ST_PixelAsCentroid](#), [ST_PixelAsCentroids](#)

11.6.4 ST_PixelAsPoints

`ST_PixelAsPoints` — 返回栅格波段的每个像素的点几何形状以及每个像素的、X 和 Y 栅格坐标。点几何的坐标是像素的左上角。

Synopsis

setof record **ST_PixelAsPoints**(raster rast, integer band=1, boolean exclude_nodata_value=TRUE);

Description

返回栅格波段的每个像素的点几何形状以及每个像素的、X 和 Y 栅格坐标。点几何的坐标是像素的左上角。
返回格式 : `geom geometry`, `val` 双精度, `x` 整数, `y` 整数。



Note
当 `except_nodata_value = TRUE` 时，将那些不是 NODATA 的像素作点返回。

可用性 : 2.1.0

更改 : 2.1.1 更改了 `except_nodata_value` 的行☒。

示例

This example returns point features together with a derived marker size that a client can use for value-scaled point rendering.

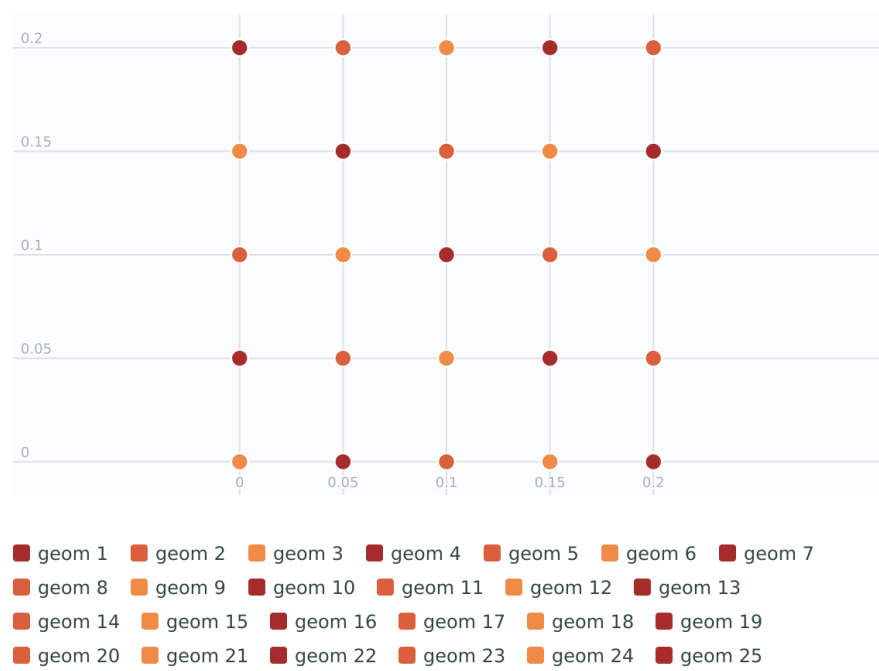
Code

```
SELECT
  x,
  y,
  val,
  round((val / 32.0)::numeric, 2) AS marker_scale,geom AS geom
FROM (
  SELECT (ST_PixelAsPoints(rast, 1)).*
  FROM dummy_rast
  WHERE rid = 2
) AS foo
ORDER BY y, x;
```

Output

x	y	val	marker_scale	geom
1	1	253	7.91	POINT(3427927.75 5793244)
2	1	254	7.94	POINT(3427927.8 5793244)
3	1	253	7.91	POINT(3427927.85 5793244)
4	1	254	7.94	POINT(3427927.9 5793244)
5	1	254	7.94	POINT(3427927.95 5793244)
1	2	253	7.91	POINT(3427927.75 5793243.95)
2	2	254	7.94	POINT(3427927.8 5793243.95)
3	2	254	7.94	POINT(3427927.85 5793243.95)
4	2	253	7.91	POINT(3427927.9 5793243.95)
5	2	249	7.78	POINT(3427927.95 5793243.95)
1	3	250	7.81	POINT(3427927.75 5793243.9)
2	3	254	7.94	POINT(3427927.8 5793243.9)
3	3	254	7.94	POINT(3427927.85 5793243.9)
4	3	252	7.88	POINT(3427927.9 5793243.9)
5	3	249	7.78	POINT(3427927.95 5793243.9)
1	4	251	7.84	POINT(3427927.75 5793243.85)
2	4	253	7.91	POINT(3427927.8 5793243.85)
3	4	254	7.94	POINT(3427927.85 5793243.85)
4	4	254	7.94	POINT(3427927.9 5793243.85)
5	4	253	7.91	POINT(3427927.95 5793243.85)
1	5	252	7.88	POINT(3427927.75 5793243.8)
2	5	250	7.81	POINT(3427927.8 5793243.8)
3	5	254	7.94	POINT(3427927.85 5793243.8)
4	5	254	7.94	POINT(3427927.9 5793243.8)
5	5	254	7.94	POINT(3427927.95 5793243.8)

Figure



相关信息

[ST_DumpAsPolygons](#), [ST_PixelAsPolygon](#), [ST_PixelAsPolygons](#), [ST_PixelAsPoint](#), [ST_PixelAsCentroid](#), [ST_PixelAsCentroids](#)

11.6.5 ST_PixelAsCentroid

ST_PixelAsCentroid — 返回像素表示的区域的中心（点几何）。

Synopsis

geometry **ST_PixelAsCentroid**(raster rast, integer x, integer y);

Description

返回像素表示的区域的中心（点几何）。

增强：3.2.0 Faster 在用 C 实现。

可用性：2.1.0

示例

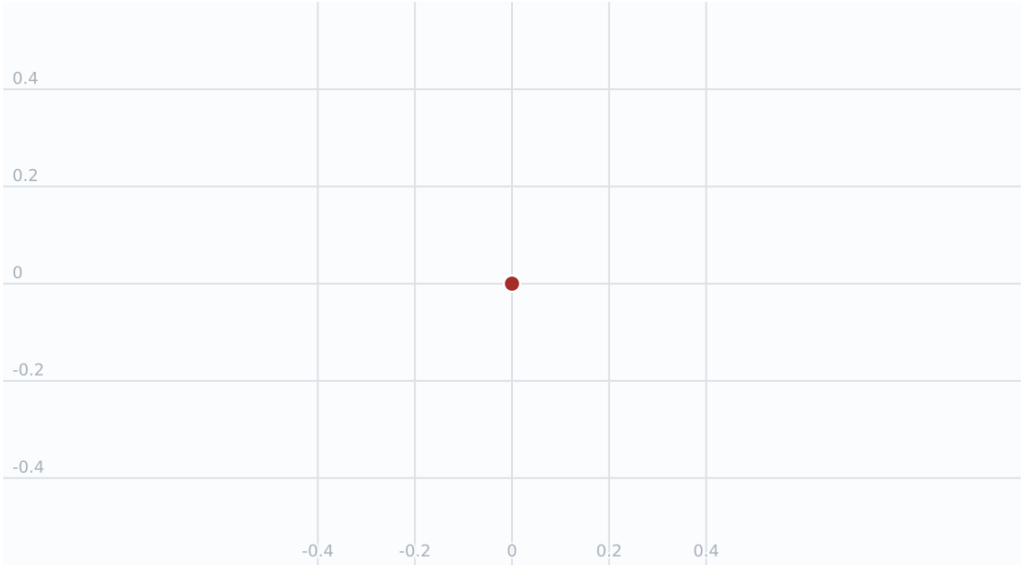
Code

```
SELECT ST_PixelAsCentroid(rast, 1, 1) FROM dummy_rast WHERE rid = 1;
```

| **Output**

```
POINT(1.5 2)
```

| **Figure**



■ Output

相关信息

[ST_DumpAsPolygons](#), [ST_PixelAsPolygon](#), [ST_PixelAsPolygons](#), [ST_PixelAsPoint](#), [ST_PixelAsPoints](#), [ST_PixelAsCentroids](#)

11.6.6 ST_PixelAsCentroids

`ST_PixelAsCentroids` — 返回栅格波段的每个像素的几何（点几何）以及每个像素的 X、Y 栅格坐标。点几何是像素表示的区域的几何。

Synopsis

setof record `ST_PixelAsCentroids`(raster rast, integer band=1, boolean exclude_nodata_value=TRUE);

Description

返回栅格波段的每个像素的几何（点几何）以及每个像素的 X、Y 栅格坐标。点几何是像素表示的区域的几何。
返回格式：`geom geometry`, `val` 双精度, `x` 整数, `y` 整数。



Note
当 `except_nodata_value = TRUE` 时，将那些不是 NODATA 的像素作点返回。

增量 : 3.2.0 Faster 在用 C 。

更改 : 2.1.1 更改了 except_nodata_value 的行。

可用性 : 2.1.0

示例

This example uses LATERAL syntax.

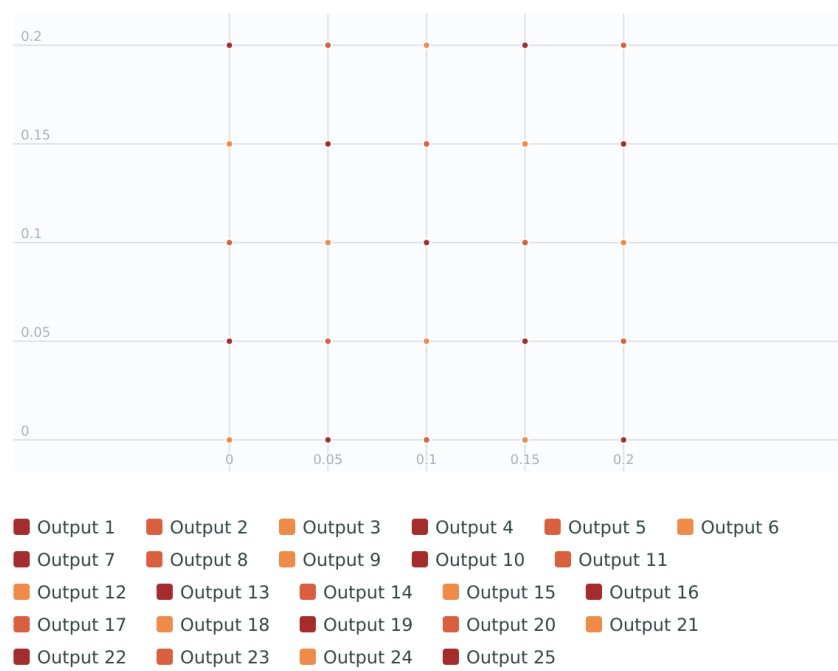
Code

```
SELECT x, y, val,geom
FROM (SELECT dp.* FROM dummy_rast, LATERAL ST_PixelAsCentroids(rast, 1) AS dp WHERE rid <= 2) foo;
```

Output

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)

Figure



相关信息

[ST_DumpAsPolygons](#), [ST_PixelAsPolygon](#), [ST_PixelAsPolygons](#), [ST_PixelAsPoint](#), [ST_PixelAsPoints](#), [ST_PixelAsPolygons](#)

11.6.7 ST_Value

ST_Value — 返回指定列 *x*、行 *y* 像素或特定几何点指定波段的值。波段编号从 1 开始，如果未指定，默认为 1。如果将参数 `exclude_nodata_value` 置为 `false`，所有像素都被与 `nodata` 像素相交并返回其值。如果未指定参数 `exclude_nodata_value`，从栅格的元数据中取值。

Synopsis

```
double precision ST_Value(raster rast, geometry pt, boolean exclude_nodata_value=true);
double precision ST_Value(raster rast, integer band, geometry pt, boolean exclude_nodata_value=true,
text resample='nearest');
double precision ST_Value(raster rast, integer x, integer y, boolean exclude_nodata_value=true);
double precision ST_Value(raster rast, integer band, integer x, integer y, boolean exclude_nodata_value=true);
```

Description

返回指定列 *x*、行 *y* 像素或指定几何点指定波段的值。波段编号从 1 开始，如果未指定波段，默认为 1。
如果 `except_nodata_value` 置为 `true`，考虑非 `nodata` 像素。如果 `except_nodata_value` 置为 `false`，考虑所有像素。

The allowed values of the `resample` parameter are "nearest" which performs the default nearest-neighbor resampling, "bilinear" which performs a **bilinear interpolation** to estimate the value between pixel centers, and the nearest-neighbor boundary options "nearest-UL", "nearest-UR", "nearest-LL",

and "nearest-LR". Boundary options choose which pixel is returned when the point lies on a horizontal or vertical pixel boundary, including corner intersections.

Enhanced: 3.7.0 resample accepts nearest-neighbor boundary options "nearest-UL", "nearest-UR", "nearest-LL", and "nearest-LR".

增⬆️: 添加了 3.2.0 重新采样可⬆️参数。

增⬆️: 2.0.0 添加了 except_nodata_value 可⬆️参数。

示例

Get raster values at particular postgis geometry points. The srid of your geometry should be same as for your raster.

| Code

```
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;
```

| Output

rid	b1pval	b2pval
2	252	79

General fictitious example using a real table.

| Code

```
SELECT rid, ST_Value(rast, 3, sometable.geom) As b3pval
FROM sometable
WHERE ST_Intersects(rast, sometable.geom);
```

| Code

```
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;
```

| Output

rid	b1pval	b2pval	b3pval
2	253	78	70

Get all values in bands 1,2,3 of each pixel.

| Code

```
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);
```

Output

x	y	b1val	b2val	b3val
1	1	253	78	70
1	2	253	96	80
1	3	250	99	90
1	4	251	89	77
1	5	252	79	62
2	1	254	98	86
2	2	254	118	108
:				
:				

Get all values in bands 1,2,3 of each pixel same as above but returning the upper left point of each pixel.

Code

```
SELECT 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);
```

Output

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 a particular polygon.

Code

```
SELECT 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;
```

Output

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 loss 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.

Code

```

SELECT 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;

```

Output

```

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)))

```

相关信息

[ST_DumpValues](#), [ST_SetValue](#), [ST_DumpAsPolygons](#), [ST_NumBands](#), [ST_PixelAsPolygon](#), [ST_ScaleX](#), [ST_ScaleY](#), [ST_UpperLeftX](#), [ST_UpperLeftY](#), [ST_SRID](#), [ST_AsText](#), [ST_Point](#), [ST_MakeEnvelope](#), [ST_Intersect](#), [ST_Intersection](#)

11.6.8 ST_NearestValue

`ST_NearestValue` — 返回由 `columnx` 和 `rowy` 指定的固定像素的最接近的非 `NODATA` 或以与格相同的空参考坐标系表示的几何点。

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= false);

Description

返回给定列 x、行 y 像素或特定几何点中最接近的非 **NODATA** 像素。如果列 x、行 y 像素或指定几何点的像素是 **NODATA**，该函数将找到距离列 x、行 y 像素或该非 **NODATA** 的几何点最近的像素。

Band 号从 1 开始，如果未指定，bandnum 假定 1。如果 except_nodata_value 置 false，所有包含 nodata 像素的像素都被相交并返回。如果 exclude_nodata_value 未入，从该格的元数据中取它。

可用性：2.1.0



Note
ST_NearestValue 是 ST_Value 的直接替代品。

示例

This example reads a value from pixel 2x2.

Code

```
WITH source AS (  
  SELECT ST_SetValues(  
    ST_AddBand(  
      ST_MakeEmptyRaster(5, 5, -2, 2, 1, -1, 0, 0, 0),  
      1, '8BUI', 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[][]  
  ) AS rast  
)  
SELECT  
  ST_Value(rast, 2, 2) AS value,  
  ST_NearestValue(rast, 2, 2) AS nearestvalue  
FROM source
```

Output

value	nearestvalue
1	1

This example reads a NODATA value from pixel 2x3.

Code

```
WITH source AS (  
  SELECT ST_SetValues(  
    ST_AddBand(  
      ST_MakeEmptyRaster(5, 5, -2, 2, 1, -1, 0, 0, 0),  
      1, '8BUI', 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[][]) AS rast  
)  
SELECT  
  ST_Value(rast, 2, 3) AS value,  
  ST_NearestValue(rast, 2, 3) AS nearestvalue  
FROM source
```

Output

value	nearestvalue
	1

相关信息

ST_Neighborhood, ST_Value

11.6.9 ST_SetZ

ST_SetZ — 返回与输入几何图形具有相同 X/Y 坐标的几何图形，并使用请求的重采样算法将栅格中的值复制到 Z 维度。

Synopsis

geometry **ST_SetZ**(raster rast, geometry geom, text resample=nearest, integer band=1);

Description

返回与输入几何图形具有相同 X/Y 坐标的几何图形，并使用请求的重采样算法将栅格中的值复制到 Z 维度。

重采样参数可以置为“nearest”，以复制每个点所在元格的值，或者置为“bilinear”，以使用双线性插值来估算考到相邻元格的值。

可用性：3.2.0

示例

2x2 test raster with values. 10 50. 40 20.

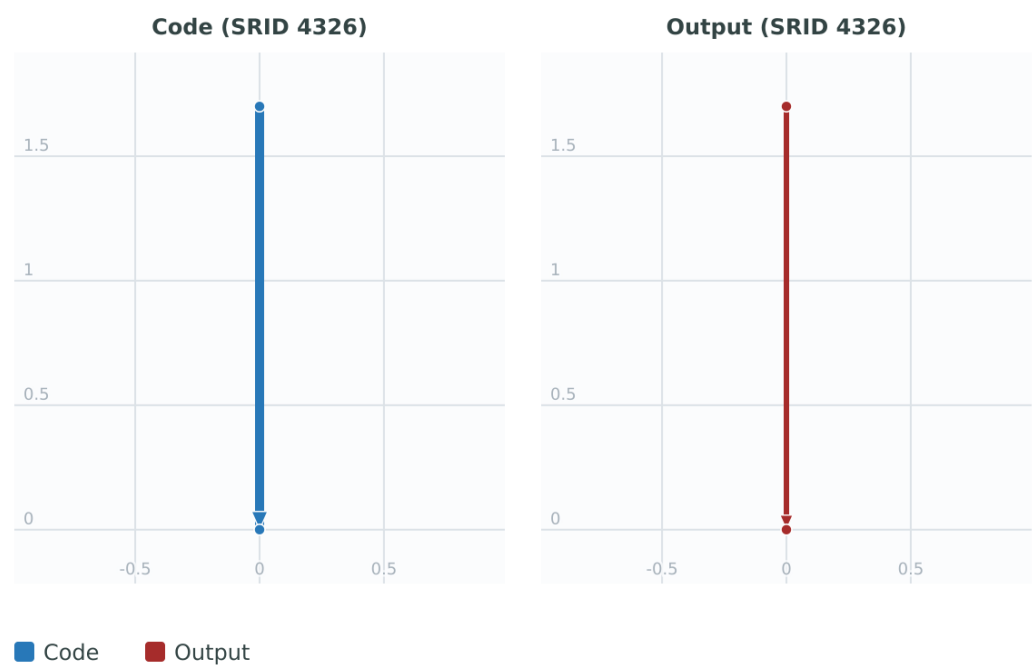
Code

```
WITH test_raster AS (  
  SELECT  
    ST_SetValues(  
      ST_AddBand(  
        ST_MakeEmptyRaster(  
          width => 2,  
          height => 2,  
          upperleftx => 0,  
          upperlefty => 2,  
          scalex => 1.0,  
          scaley => -1.0,  
          skewx => 0,  
          skewy => 0,  
          srid => 4326),  
          index => 1,  
          pixeltype => '16BSI',  
          initialvalue => 0,  
          nodataval => -999  
        ),  
        1,  
        1,  
        1,  
        newvalueset => ARRAY[ARRAY[10.0::float8, 50.0::float8], ARRAY[40.0::float8, 20.0::float8 ←  
          ]]) AS rast  
    )  
  SELECT  
    ST_SetZ(  
      rast,  
      band => 1,  
      geom => 'SRID=4326;LINESTRING(1.0 1.9,1.0 0.2)::geometry,  
      resample => 'bilinear'  
    )  
FROM test_raster
```

Output

```
LINESTRING Z (1 1.9 38,1 0.2 27)
```

Figure



相关信息

[ST_Value](#), [ST_SetM](#)

11.6.10 ST_SetM

`ST_SetM` — 返回与输入几何图形具有相同 X/Y 坐标的几何图形，并使用请求的重采样算法将栅格中的值复制到 M 度。

Synopsis

```
geometry ST_SetM(raster rast, geometry geom, text resample=nearest, integer band=1);
```

Description

返回与输入几何图形具有相同 X/Y 坐标的几何图形，并使用请求的重采样算法将栅格中的值复制到 M 度。

重采样参数可以置为“nearest”，以复制每个点所在栅格单元的值，或者置为“bilinear”，以使用双线性插值来估算到相邻栅格单元的值。

可用性：3.2.0

示例

2x2 test raster with values. 10 50. 40 20.

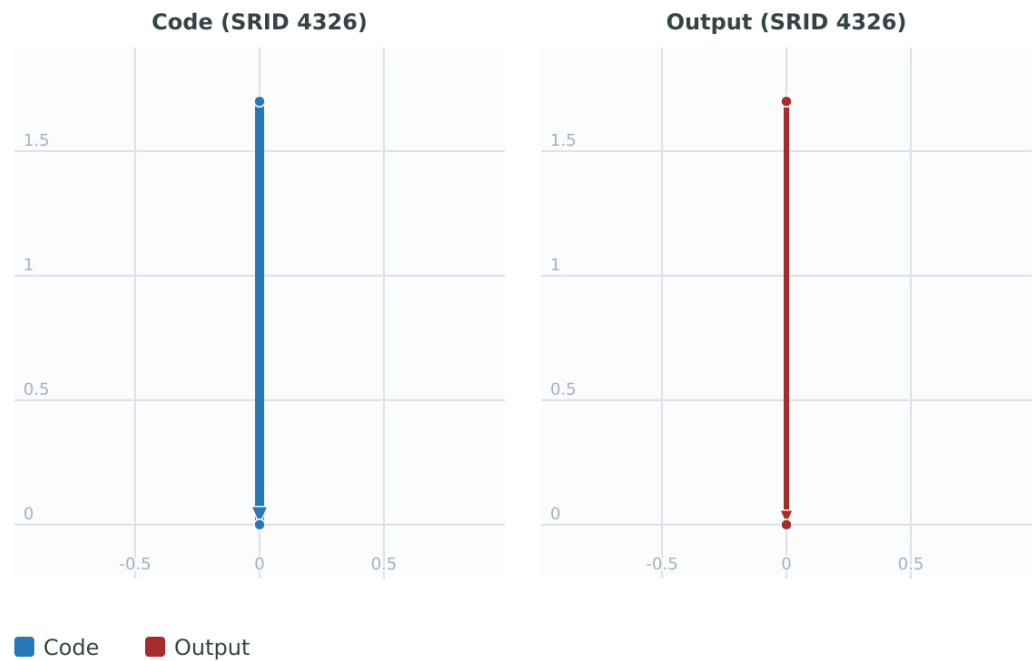
Code

```
WITH test_raster AS (  
  SELECT  
    ST_SetValues(  
      ST_AddBand(  
        ST_MakeEmptyRaster(  
          width => 2,  
          height => 2,  
          upperleftx => 0,  
          upperlefty => 2,  
          scalex => 1.0,  
          scaley => -1.0,  
          skewx => 0,  
          skewy => 0,  
          srid => 4326),  
          index => 1,  
          pixeltype => '16BSI',  
          initialvalue => 0,  
          nodataval => -999  
        ),  
        1,  
        1,  
        1,  
        newvalueset => ARRAY[ARRAY[10.0::float8, 50.0::float8], ARRAY[40.0::float8, 20.0::float8 ←  
          ]]) AS rast  
    )  
  SELECT  
    ST_SetM(  
      rast,  
      band => 1,  
      geom => 'SRID=4326;LINESTRING(1.0 1.9,1.0 0.2)::geometry',  
      resample => 'bilinear'  
    )  
FROM test_raster
```

Output

```
LINESTRING M (1 1.9 38,1 0.2 27)
```

Figure



相关信息

[ST_Value](#), [ST_SetZ](#)

11.6.11 ST_Neighborhood

ST_Neighborhood — 返回指定波段像素周围非 `NODATA` 的二精度数，指定波段像素由 `columnX` 和 `rowY` 或以与网格相同的空参考坐标系表示的几何点指定。

Synopsis

```
double precision[][] ST_Neighborhood(raster rast, integer bandnum, integer columnX, integer rowY,
integer distanceX, integer distanceY, boolean exclude_nodata_value=true);
double precision[][] ST_Neighborhood(raster rast, integer columnX, integer rowY, integer distanceX,
integer distanceY, boolean exclude_nodata_value=true);
double precision[][] ST_Neighborhood(raster rast, integer bandnum, geometry pt, integer distanceX,
integer distanceY, boolean exclude_nodata_value=true);
double precision[][] ST_Neighborhood(raster rast, geometry pt, integer distanceX, integer distanceY,
boolean exclude_nodata_value=true);
```

Description

返回一个二精度数，其中包含指定波段的像素的非 `NODATA`。可以通过指定列 `X` 和行 `Y`，或者使用与网格相同的空参考坐标系表示的几何点来完成。参数 `distanceX` 和 `distanceY` 定义了 `X` 和 `Y` 上指定像素周围的像素数量，例如，我想取我感兴趣的像素周围 `X` 方向上 3 个像素距离和 `Y` 方向上 2 个像素距离内的所有。二数的中心将是列 `X` 和行 `Y` 或几何点指定的像素的。

Band 号从 1 开始，如果未指定，`bandnum` 假定 1。如果 `except_nodata_value` 置 `false`，所有包含 `nodata` 像素的像素都被相交并返回。如果 `exclude_nodata_value` 未入，从网格的元数据中取它。

**Note**

返回的二进制数沿每个元素的元素数量 $2 * (\text{distanceX}|\text{distanceY}) + 1$ 。因此，对于 distanceX 和 distanceY 均为 1，返回的数将是一个 3×3 。

**Note**

二进制数输出可以调用任何栅格处理内置函数，例如 `ST_Min4ma`、`ST_Sum4ma`、`ST_Mean4ma`。

可用性：2.1.0

示例

This example reads a value from pixel 2x2.

Code

```
SELECT
  ST_Neighborhood(rast, 2, 2, 1, 1)
FROM (
  SELECT
    ST_SetValues(ST_AddBand(ST_MakeEmptyRaster(5, 5, -2, 2, 1, -1, 0, 0, 0),
      '8BUI'::text, 1, 0
    ),
    1, 1, 1, ARRAY[
      [0, 1, 1, 1, 1],
      [1, 1, 1, 0, 1],
      [1, 0, 1, 1, 1],
      [1, 1, 1, 1, 0],
      [1, 1, 0, 1, 1]
    ]::double precision[],
    1
  ) AS rast
) AS foo
```

Output

```
{{NULL,1,1},{1,1,1},{1,NULL,1}}
```

This example reads a NODATA value from pixel 2x3.

Code

```
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
```

```

        1
    ) AS rast
) AS foo

```

Output

```
{{1,1,1},{1,NULL,1},{1,1,1}}
```

Pixel 3x3 has value. Exclude_nodata_value = FALSE.

Code

```

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

```

Output

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

相关信息

[ST_NearestValue](#), [ST_Min4ma](#), [ST_Max4ma](#), [ST_Sum4ma](#), [ST_Mean4ma](#), [ST_Range4ma](#), [ST_Distinct4ma](#), [ST_StdDev4ma](#)

11.6.12 ST_SetValue

ST_SetValue — 返回修改后的栅格，其结果是将指定波段中的位置指定列 x、行 y 像素或与特定几何形状相交的像素。波段号从 1 开始，如果未指定波段，默认为 1。

Synopsis

```

raster ST_SetValue(raster rast, integer bandnum, geometry geom, double precision newvalue);
raster ST_SetValue(raster rast, geometry geom, double precision newvalue);
raster ST_SetValue(raster rast, integer bandnum, integer columnx, integer rowy, double precision newvalue);
raster ST_SetValue(raster rast, integer columnx, integer rowy, double precision newvalue);

```

Description

返回修改后的栅格，栅格是通过将指定像素的位置指定栅格的行和列或几何形状的指定波段的新面生成的。如果未指定波段，假定 1。

增修：2.1.0 **ST_SetValue()** 的几何体在支持任何几何型，而不只是点。几何体是 **ST_SetValues()** 的 `geomval[]` 体的包装

示例

This example uses geometry input.

Code

```
SELECT (foo.geomval).val, 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;
```

Output

val	st_astext
50	POLYGON((3427927.75 5793244,3427927.75 5793243.95,3427927.8 579324 ...
249	POLYGON((3427927.95 5793243.95,3427927.95 5793243.85,3427928 57932 ...

This example stores the changed raster.

Code

```
UPDATE dummy_rast SET rast = ST_SetValue(rast, 1, ST_Point(3427927.75, 5793243.95), 100)
WHERE rid = 2 ;
```

相关信息

[ST_Value](#), [ST_DumpAsPolygons](#)

11.6.13 ST_SetValues

ST_SetValues — 返回通过指定波段的而生的修改后的栅格。

Synopsis

```
raster ST_SetValues(raster rast, integer nband, integer columnx, integer rowy, double precision[] newvalueset, boolean[][] noset=NULL, boolean keepnodata=FALSE);
raster ST_SetValues(raster rast, integer nband, integer columnx, integer rowy, double precision[] newvalueset, double precision nosetvalue, boolean keepnodata=FALSE);
raster ST_SetValues(raster rast, integer nband, integer columnx, integer rowy, integer width, integer height, double precision newvalue, boolean keepnodata=FALSE);
raster ST_SetValues(raster rast, integer columnx, integer rowy, integer width, integer height, double precision newvalue, boolean keepnodata=FALSE);
raster ST_SetValues(raster rast, integer nband, geomval[] geomvalset, boolean keepnodata=FALSE);
```

Description

返回通Ⓔ将指定波段的指定像素Ⓔ置Ⓔ新Ⓔ而Ⓔ生的修改后的Ⓔ格。columnx 和 rowy 是从 1 开始索引的。

如果 keepnodata Ⓔ TRUE，那些ⒺⒺ NODATA 的像素将不会被Ⓔ置Ⓔ newvalueset 中的相ⒺⒺ。

Ⓔ于Ⓔ体 1，要Ⓔ置的特定像素由 columnx、rowy 像素坐Ⓔ和 newvalueset 数Ⓔ的尺寸决定。noset 可以用来防止Ⓔ置具有 newvalueset 中存在的Ⓔ的像素（因Ⓔ PostgreSQL 不允Ⓔ不ⒺⒺ数Ⓔ）。Ⓔ参ⒺⒺ体 1 的示例。

Ⓔ体 2 与Ⓔ体 1 Ⓔ似，但是使用ⒺⒺ的双精度 nosetvalue 而不是布Ⓔ型的 noset 数Ⓔ。newvalueset 中具有 nosetvalue Ⓔ的元素将被跳Ⓔ。Ⓔ参ⒺⒺ体 2 的示例。

Ⓔ于Ⓔ体 3，要Ⓔ置的特定像素由 columnx、rowy 像素坐Ⓔ、width 和 height 决定。Ⓔ参ⒺⒺ体 3 的示例。

Ⓔ体 4 与Ⓔ体 3 相同，唯一的区Ⓔ是它假定将Ⓔ置 rast 的第一个波段的像素。

Ⓔ于Ⓔ体 5，将使用geomval数Ⓔ来确定要Ⓔ置的特定像素。如果数Ⓔ中的所有几何Ⓔ形都是点Ⓔ型或多点Ⓔ型，函数将使用一种快捷方式，其中每个点的Ⓔ度和Ⓔ度用于直接Ⓔ置像素。否Ⓔ，几何Ⓔ形将ⒺⒺⒺⒺ格，然后一次迭代通Ⓔ它Ⓔ。Ⓔ参ⒺⒺ体 5 的示例。

可用性：2.1.0

示例

Variant 1.

These examples pass FALSE for *exclude_nodata_value* so that every raster cell remains visible. NO-DATA cells display the band’s NODATA value, which is 0 here.

Set the lower-right 2 by 2 block of a 3 by 3 raster to 9.

Code

```
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[][])
    ),
    1, FALSE
) AS poly
) foo
ORDER BY 1, 2;
```

Output

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

Replace the complete raster from an array; the central NULL becomes NODATA.

Code

```
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[][])
    ),
    1, FALSE
) AS poly
) foo
ORDER BY 1, 2;
```

Output

x	y	val
1	1	9
1	2	9
1	3	9
2	1	9
2	2	0
2	3	9
3	1	9
3	2	9
3	3	9

Use the *noset* mask to retain the left-middle source pixel while replacing the other cells.

Code

```
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[][])
    ),
    1, FALSE
) AS poly
) foo
ORDER BY 1, 2;
```

Output

x	y	val
1	1	9
1	2	1
1	3	9
2	1	9
2	2	0
2	3	9
3	1	9
3	2	9
3	3	9

With *keepnodata* enabled, an existing NODATA pixel is not overwritten.

Code

```
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
),
  1, FALSE
) AS poly
) foo
ORDER BY 1, 2;
```

Output

x	y	val
1	1	0
1	2	1
1	3	9
2	1	9
2	2	0
2	3	9
3	1	9
3	2	9
3	3	9

Variant 2.

Use -1 as the value that should not be written; only the lower-right 2 by 2 block changes.

Code

```
SELECT
  (poly).x,
  (poly).y,
  (poly).val
FROM (
  SELECT
    ST_PixelAsPolygons(ST_SetValues(ST_AddBand(ST_MakeEmptyRaster(3, 3, 0, 0, 1, -1, 0, 0, ↵
    0),
      1, '8BUI', 1, 0
    ),
    1, 1, 1, ARRAY[[-1, -1, -1], [-1, 9, 9], [-1, 9, 9]]::double precision[][], -1
  ),
  1, FALSE
) AS poly
) foo
ORDER BY 1, 2;
```

Output

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 same update can use NULL as the value that should not be written.

Code

```
SELECT
    (poly).x,
    (poly).y,
    (poly).val
FROM (
    SELECT
        ST_PixelAsPolygons(ST_SetValues(ST_AddBand(ST_MakeEmptyRaster(3, 3, 0, 0, 1, -1, 0, 0, ↵
        0),
            1, '8BUI', 1, 0
        ),
            1, 1, 1, ARRAY[[NULL, NULL, NULL], [NULL, 9, 9], [NULL, 9, 9]]::double ↵
        precision[][], NULL::double precision
        ),
        1, FALSE
    ) AS poly
) foo
ORDER BY 1, 2;
```

Output

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

Variant 3.
Set a 2 by 2 rectangle starting at column 2, row 2.

Code

```
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
        ),
        1, FALSE
    ) AS poly
) foo
ORDER BY 1, 2;
```

```
    ) AS poly
) foo
ORDER BY 1, 2;
```

Output

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

When *keepnodata* is true, the NODATA cell inside the rectangle remains unchanged.

Code

```
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
    ),
    1, FALSE
) AS poly
) foo
ORDER BY 1, 2;
```

Output

x	y	val
1	1	1
1	2	1
1	3	1
2	1	1
2	2	0
2	3	9
3	1	1
3	2	9
3	3	9

Variant 5.

Code

```
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
), changed AS (
  SELECT rid, gid, ST_SetValue(rast, 1, geom, gid) AS rast
  FROM foo
  CROSS JOIN bar
)
SELECT
  rid,
  gid,
  y,
  array_agg(ST_Value(rast, 1, x, y) ORDER BY x) AS values
FROM changed
CROSS JOIN generate_series(1, 5) AS x
CROSS JOIN generate_series(1, 5) AS y
GROUP BY rid, gid, y
ORDER BY rid, gid, y;

```

Output

rid	gid	y	values
1	1	1	{NULL,NULL,NULL,NULL,NULL}
1	1	2	{NULL,NULL,NULL,NULL,NULL}
1	1	3	{NULL,NULL,1,NULL,NULL}
1	1	4	{NULL,NULL,NULL,NULL,NULL}
1	1	5	{NULL,NULL,NULL,NULL,NULL}
1	2	1	{NULL,NULL,NULL,NULL,NULL}
1	2	2	{NULL,2,2,2,NULL}
1	2	3	{NULL,2,2,2,NULL}
1	2	4	{NULL,2,2,2,NULL}
1	2	5	{NULL,NULL,NULL,NULL,NULL}
1	3	1	{3,3,3,3,3}
1	3	2	{3,NULL,NULL,NULL,NULL}
1	3	3	{3,NULL,NULL,NULL,NULL}
1	3	4	{3,NULL,NULL,NULL,NULL}
1	3	5	{NULL,NULL,NULL,NULL,NULL}
1	4	1	{4,NULL,NULL,NULL,NULL}
1	4	2	{NULL,NULL,NULL,NULL,NULL}
1	4	3	{NULL,NULL,NULL,NULL,NULL}
1	4	4	{NULL,NULL,NULL,NULL,NULL}
1	4	5	{NULL,NULL,NULL,NULL,4}

(20 rows)

下面显示了数中后面的 geomvals 可以覆盖前面的 geomvals

Code

```

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
), changed AS (
  SELECT ST_SetValues(
    rast,
    1,
    ARRAY[
      ROW((SELECT geom FROM bar WHERE gid = 1), 1),
      ROW((SELECT geom FROM bar WHERE gid = 2), 2)
    ]::geomval[]
  ) AS rast
  FROM foo
)
SELECT

```

```
    y,
    array_agg(ST_Value(rast, 1, x, y) ORDER BY x) AS values
FROM changed
CROSS JOIN generate_series(1, 5) AS x
CROSS JOIN generate_series(1, 5) AS y
GROUP BY y
ORDER BY y;
```

Output

y	values
1	{NULL,NULL,NULL,NULL,NULL}
2	{NULL,2,2,2,NULL}
3	{NULL,2,2,2,NULL}
4	{NULL,2,2,2,NULL}
5	{NULL,NULL,NULL,NULL,NULL}
(5 rows)	

☒ 个例子与前面的例子相反

Code

```
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
), changed AS (
    SELECT ST_SetValues(
        rast,
        1,
        ARRAY[
            ROW((SELECT geom FROM bar WHERE gid = 2), 2),
            ROW((SELECT geom FROM bar WHERE gid = 1), 1)
        ]::geomval[]
    ) AS rast
    FROM foo
)
SELECT
    y,
    array_agg(ST_Value(rast, 1, x, y) ORDER BY x) AS values
FROM changed
CROSS JOIN generate_series(1, 5) AS x
CROSS JOIN generate_series(1, 5) AS y
GROUP BY y
ORDER BY y;
```

Output

y	values
1	{NULL,NULL,NULL,NULL,NULL}
2	{NULL,2,2,2,NULL}
3	{NULL,2,1,2,NULL}
4	{NULL,2,2,2,NULL}
5	{NULL,NULL,NULL,NULL,NULL}
(5 rows)	

相关信息

ST_Value, ST_SetValue, ST_PixelAsPolygons

11.6.14 ST_DumpValues

ST_DumpValues — 提取指定 band（波段）的栅格二值数。

Synopsis

```
setof record ST_DumpValues( raster rast , integer[] nband=NULL , boolean exclude_nodata_value=true );  
double precision[][] ST_DumpValues( raster rast , integer nband , boolean exclude_nodata_value=true );
```

Description

提取指定 band 的栅格二值数（第一个索引是行，第二个索引是列）。如果 nband 为 NULL 或未提供，则处理所有栅格波段。

可用性：2.1.0

示例

Code

```
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;
```

Output

nband	valarray
1	{1,1,1},{1,1,1},{1,1,1}
2	{3,3,3},{3,3,3},{3,3,3}
3	{NULL,NULL,NULL},{NULL,NULL,NULL},{NULL,NULL,NULL}

(3 rows)

Code

```
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;
```

Output

nband	valarray
3	{NULL,NULL,NULL},{NULL,NULL,NULL},{NULL,NULL,NULL}
1	{1,1,1},{1,1,1},{1,1,1}

(2 rows)

Code

```
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;
```

Output

```
5
```

相关信息

[ST_Value](#), [ST_SetValue](#), [ST_SetValues](#)

11.6.15 ST_PixelOfValue

`ST_PixelOfValue` — 返回等于搜索的像素的列 `x`、行 `y` 坐标。

Synopsis

```
setof record ST_PixelOfValue( raster rast , integer nband , double precision[] search , boolean ex-
clude_nodata_value=true );
setof record ST_PixelOfValue( raster rast , double precision[] search , boolean exclude_nodata_value=true
);
setof record ST_PixelOfValue( raster rast , integer nband , double precision search , boolean ex-
clude_nodata_value=true );
setof record ST_PixelOfValue( raster rast , double precision search , boolean exclude_nodata_value=true
);
```

Description

返回等于搜索的像素的列 `x`、行 `y` 坐标。如果未指定波段，假定波段 1。

可用性：2.1.0

示例

Code

```
WITH source AS (
    SELECT ST_SetValues(
        ST_AddBand(
            ST_MakeEmptyRaster(5, 5, -2, 2, 1, -1, 0, 0, 0),
            1, '8BUI', 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, 255],
        [1, 1, 0, 1, 1]
      ]::double precision[][])
    ) AS rast
)
SELECT (ST_PixelOfValue(rast, 1, ARRAY[1, 255])).*
FROM source
```

Output

val	x	y
1	1	2
1	1	3
1	1	4
1	1	5
1	2	1
1	2	2
1	2	4
1	2	5
1	3	1
1	3	2
1	3	3
1	3	4
1	4	1
1	4	3
1	4	4
1	4	5
1	5	1
1	5	2
1	5	3
255	5	4
1	5	5

11.7 栅格处理器

11.7.1 ST_SetGeoReference

ST_SetGeoReference — 在一次调用中设置 Georeference 6 地理配准参数。数字之间用空格分隔。接受 GDAL 或 ESRI 格式的输入。默认为 GDAL。

Synopsis

```
raster ST_SetGeoReference(raster rast, text georefcoords, text format=GDAL);
raster ST_SetGeoReference(raster rast, double precision upperleftx, double precision upperlefty,
double precision scalex, double precision scaley, double precision skewx, double precision skewy);
```

Description

在一次调用中设置 Georeference 6 地理配准参数。接受 “GDAL” 或 “ESRI” 格式的输入。默认为 GDAL。如果未提供 6 个坐标，则返回 null。

格式表示的区域如下：

GDAL uses the value order shown below.

Code

```
scalex skewy skewx scaley upperleftx upperlefty
```

ESRI uses the value order shown below.

Code

```
scalex skewx scaley upperleftx + scalex*0.5 upperlefty + scaley*0.5
```



Note

如果栅格具有数据外波段，更改地理参考可能会导致无法正确解释波段的外部存数据。

增加：2.1.0 添加 ST_SetGeoReference(raster, double precision, ...) 函数

示例

Code

```
WITH source AS (
  SELECT ST_MakeEmptyRaster(5, 5, 0, 0, 1, -1, 0, 0, 0) AS rast
), variants AS (
  SELECT 0 AS rid, rast FROM source
  UNION ALL
  SELECT 1, ST_SetGeoReference(rast, '10 0 0 -10 0.1 0.1', 'GDAL')
  FROM source
  UNION ALL
  SELECT 2, ST_SetGeoReference(rast, '10 0 0 -10 5.1 -4.9', 'ESRI')
  FROM source
  UNION ALL
  SELECT 3, ST_SetGeoReference(rast, 1, 1, 10, -10, 0.001, 0.001)
  FROM source
)
SELECT
  rid,
  round(ST_UpperLeftX(rast)::numeric, 3) AS upperleftx,
  round(ST_UpperLeftY(rast)::numeric, 3) AS upperlefty,
  ST_ScaleX(rast) AS scalex,
  ST_ScaleY(rast) AS scaley,
  ST_SkewX(rast) AS skewx,
  ST_SkewY(rast) AS skewy
FROM variants
ORDER BY rid;
```

Output

rid	upperleftx	upperlefty	scalex	scaley	skewx	skewy
0	0.000	0.000	1	-1	0	0
1	0.100	0.100	10	-10	0	0
2	0.100	0.100	10	-10	0	0
3	1.000	1.000	10	-10	0.001	0.001

(4 rows)

相关信息

[ST_GeoReference](#), [ST_ScaleX](#), [ST_ScaleY](#), [ST_UpperLeftX](#), [ST_UpperLeftY](#)

11.7.2 ST_SetRotation

ST_SetRotation — 以弧度置栅格的旋。

Synopsis

raster **ST_SetRotation**(raster rast, float8 rotation);

Description

均匀旋 $\varnothing$ 格。旋 $\varnothing$ 以弧度 $\varnothing$ 位。有关更多 $\varnothing$ 信息， $\varnothing$ 参 $\varnothing$ [世界文件](#)。

示例

Code

```
WITH variants AS (  
    SELECT rid, 'rotated' AS kind, ST_SetRotation(rast, 15) AS rast  
    FROM dummy_rast  
    UNION ALL  
    SELECT rid, 'original', rast  
    FROM dummy_rast  
)  
SELECT  
    rid,  
    kind,  
    round(ST_ScaleX(rast)::numeric, 4) AS scale_x,  
    round(ST_ScaleY(rast)::numeric, 4) AS scale_y,  
    round(ST_SkewX(rast)::numeric, 4) AS skew_x,  
    round(ST_SkewY(rast)::numeric, 4) AS skew_y  
FROM variants  
ORDER BY rid, kind;
```

Output

rid	kind	scale_x	scale_y	skew_x	skew_y
1	original	2.0000	3.0000	0.0000	0.0000
1	rotated	-1.5194	-2.2791	1.9509	1.3006
2	original	0.0500	-0.0500	0.0000	0.0000
2	rotated	-0.0380	-0.0380	0.0325	0.0325

(4 rows)

相关信息

[ST_Rotation](#), [ST_ScaleX](#), [ST_ScaleY](#), [ST_SkewX](#), [ST_SkewY](#)

11.7.3 ST_SetScale

ST_SetScale — 以坐 $\varnothing$ 参考系 $\varnothing$ 位 $\varnothing$ 置像素的 X 和 Y 大小。数字 $\varnothing$ 位/像素 $\varnothing$ 度/高度。

Synopsis

raster **ST_SetScale**(raster rast, float8 xy);
raster **ST_SetScale**(raster rast, float8 x, float8 y);

Description

以坐標参考系為單位設置像素的 X 和 Y 大小。數字單位/像素度/高度。如果只輸入一個單位，則假設 X 和 Y 是相同的數字。



Note ST_SetScale 与 **ST_Rescale** 的不同之處在於 ST_SetScale 不會重新採樣以匹配目標範圍。它更改目標格的元數據（或地理參考）以正確最初指定的縮放比例。ST_Rescale 會生成具有不同度和高度的目標格，計算結果以適合輸入目標格的地理範圍。ST_SetScale 不修改目標格的度或高度。

更改：2.0.0 在 WKTRaster 版本中，名稱 ST_SetPixelSize。在 2.0.0 中進行了變化。

示例

Code

```
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;
```

Output

pixx	pixy	newbox
1.5	1.5	BOX(3427927.75 5793244 0, 3427935.25 5793251.5 0)

Code

```
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;
```

Output

pixx	pixy	newbox
1.5	0.55	BOX(3427927.75 5793244 0,3427935.25 5793247 0)

相關信息

ST_ScaleX, ST_ScaleY, Box3D

11.7.4 ST_SetSkew

ST_SetSkew — 設置地理參考 X 和 Y 傾斜（或旋轉參數）。如果只輸入一個，則將 X 和 Y 設置為相同的值。

Synopsis

raster **ST_SetSkew**(raster rast, float8 skewxy);
raster **ST_SetSkew**(raster rast, float8 skewx, float8 skewy);

Description

设置地理参考 X 和 Y 斜 (或旋参数)。如果只输入一个, 将 X 和 Y 设置相同的。有关更多信息, 参
世界文件。

示例

This is the first example.

Code

```
UPDATE dummy_rast SET rast = ST_SetSkew(rast, 1, 2) WHERE rid = 1;
SELECT
    rid,
    format('%s, %s', skewx, skewy) AS skew,
    term,
    georef[idx] AS value
FROM (
    SELECT
        rid,
        ST_SkewX(rast) AS skewx,
        ST_SkewY(rast) AS skewy,
        string_to_array(ST_GeoReference(rast), E'\n') AS georef
    FROM dummy_rast
    WHERE rid = 1
) AS refs
CROSS JOIN (VALUES
    (1, 'scaleX'),
    (2, 'skewY'),
    (3, 'skewX'),
    (4, 'scaleY'),
    (5, 'originX'),
    (6, 'originY')
) AS parts(idx, term)
ORDER BY idx;
```

Output

rid	skew	term	value
1	(1, 2)	scaleX	2.0000000000
1	(1, 2)	skewY	2.0000000000
1	(1, 2)	skewX	1.0000000000
1	(1, 2)	scaleY	3.0000000000
1	(1, 2)	originX	0.5000000000
1	(1, 2)	originY	0.5000000000

(6 rows)

This second example sets both values to the same number.

Code

```
UPDATE dummy_rast SET rast = ST_SetSkew(rast, 0) WHERE rid = 1;
SELECT
    rid,
    format('%s, %s', skewx, skewy) AS skew,
```

```
term,
georef[idx] AS value
FROM (
  SELECT
    rid,
    ST_SkewX(rast) AS skewx,
    ST_SkewY(rast) AS skewy,
    string_to_array(ST_GeoReference(rast), E'\n') AS georef
  FROM dummy_rast
  WHERE rid = 1
) AS refs
CROSS JOIN (VALUES
  (1, 'scaleX'),
  (2, 'skewY'),
  (3, 'skewX'),
  (4, 'scaleY'),
  (5, 'originX'),
  (6, 'originY')
) AS parts(idx, term)
ORDER BY idx;
```

Output

rid	skew	term	value
1	(0, 0)	scaleX	2.0000000000
1	(0, 0)	skewY	0.0000000000
1	(0, 0)	skewX	0.0000000000
1	(0, 0)	scaleY	3.0000000000
1	(0, 0)	originX	0.5000000000
1	(0, 0)	originY	0.5000000000

(6 rows)

相关信息

[ST_GeoReference](#), [ST_SetGeoReference](#), [ST_SkewX](#), [ST_SkewY](#)

11.7.5 ST_SetSRID

ST_SetSRID — 将栅格的 SRID 设置在 Spatial_ref_sys 表中定义的特定整数 srid。

Synopsis

raster **ST_SetSRID**(raster rast, integer srid);

Description

将栅格上的 SRID 设置成特定整数。



Note
此函数不会以任何方式修改栅格 - 它只是设置定义其当前所在坐​​标参考系​​的空​​参考​​的元数据。对于以后的​​修改​​很有用。

相关信息

Section [4.5](#), [ST_SRID](#)

11.7.6 ST_SetUpperLeft

ST_SetUpperLeft — 将栅格左上角的像素位置投影的 X 和 Y 坐标。

Synopsis

raster **ST_SetUpperLeft**(raster rast, double precision x, double precision y);

Description

将栅格像素左上角的像素位置投影的 X 和 Y 坐标

示例

Code

```
SELECT ST_SetUpperLeft(rast, -71.01, 42.37)
FROM dummy_rast
WHERE rid = 2;
```

相关信息

[ST_UpperLeftX](#), [ST_UpperLeftY](#)

11.7.7 ST_Resample

ST_Resample — 重采样一个栅格图像，可以指定重新采样算法、新的尺寸、任意的栅格角点，以及一个栅格地理参考属性，这些属性可以自己定义，也可以从一个栅格图像中借用。

Synopsis

raster **ST_Resample**(raster rast, integer width, integer height, double precision gridx=NULL, double precision gridy=NULL, double precision skewx=0, double precision skewy=0, text algorithm=NearestNeighbor, double precision maxerr=0.125);
raster **ST_Resample**(raster rast, double precision scalex=0, double precision scaley=0, double precision gridx=NULL, double precision gridy=NULL, double precision skewx=0, double precision skewy=0, text algorithm=NearestNeighbor, double precision maxerr=0.125);
raster **ST_Resample**(raster rast, raster ref, text algorithm=NearestNeighbor, double precision maxerr=0.125, boolean usescale=true);
raster **ST_Resample**(raster rast, raster ref, boolean usescale, text algorithm=NearestNeighbor, double precision maxerr=0.125);

Description

使用指定的重新采样算法、新的尺寸 (`width&height`)、一个网格角点 (`gridx & gridy`)，以及一个网格地理参考属性 (`scalex`、`scaley`、`skewx & skewy`)，这些属性可以自己定义，也可以从一个网格图像中借用。如果使用参考网格图像，那么每个网格图像必须具有相同的空参考 ID (SRID)。

使用以下重采样算法之一计算新像素：

- NearestNeighbor (英式或美式拼写)
- Bilinear
- Cubic
- CubicSpline
- Lanczos
- Max
- Min

默认是“NearestNeighbor”，它速度最快，但插值效果最差。

如果未指定 `maxerr`，使用 0.125 的 `maxerror` 百分比。



Note

有关更多信息，参考：[GDAL Warp 重采样方法](#)。

可用性：2.0.0 需要 GDAL 1.6.1+

增强：3.4.0 添加了最大和最小重采样

示例

Code

```
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;
```

Output

orig_width	new_width
-----+-----	
200	100

相关信息

[ST_Rescale](#), [ST_Resize](#), [ST_Transform](#)

11.7.8 ST_Rescale

`ST_Rescale` — 通过指定栅格的比率（或像素大小）来重新采样栅格。新的像素是使用 `NearestNeighbor`（英语或美式拼写）、`Bilinear`、`Cubic`、`CubicSpline`、`Lanczos`、`Max` 或 `Min` 重采样算法计算的。默认为 `NearestNeighbor`。

Synopsis

```
raster ST_Rescale(raster rast, double precision scalexy, text algorithm=NearestNeighbor, double precision maxerr=0.125);
raster ST_Rescale(raster rast, double precision scalex, double precision scaley, text algorithm=NearestNeighbor, double precision maxerr=0.125);
```

Description

通过指定栅格的比率（或像素大小）来重新采样栅格。使用以下重采样算法之一计算新像素：

- `NearestNeighbor`（英语或美式拼写）
- `Bilinear`
- `Cubic`
- `CubicSpline`
- `Lanczos`
- `Max`
- `Min`

默认是“`NearestNeighbor`”，它速度最快，但插值效果最差。

`scalex` 和 `scaley` 定义新的像素大小。为了获得良好定向的栅格，`scaley` 通常必须与 `scalex` 相同。

当新的 `scalex` 或 `scaley` 不是栅格宽度或高度的整数，生成的栅格的范围将扩展以包含所提供栅格的范围。如果您想确保保留精确的输入范围，请参考 [ST_Resize](#)

`maxerr` 是重采样算法执行近似的精度（以像素为单位）。如果未指定 `maxerr`，将使用默认值 0.125，这与 GDAL `gdalwarp` 应用程序中使用的相同。如果设置为零，将不会生成近似。



Note

有关更多信息，请参考：[GDAL Warp 重采样方法](#)。

**Note**

ST_Rescale 与 **ST_SetScale** 的不同之处在于 ST_SetScale 不会将栅格重新采样以匹配栅格范围。ST_SetScale 更改栅格的元数据（或地理参考）以更正最初指定的缩放比例。ST_Rescale 会生成具有不同度和高度的栅格，计算结果以适合输入栅格的地理范围。ST_SetScale 不修改栅格的度或高度。

可用性：2.0.0 需要 GDAL 1.6.1+

增加：3.4.0 添加了最大和最小重采样

更改：2.1.0 适用于没有 SRID 的栅格

示例

将栅格从 0.001 度的像素大小重新缩放 0.0015 度的像素大小的示例。

This example returns the original raster pixel size.

Code

```
SELECT ST_PixelWidth(ST_AddBand(ST_MakeEmptyRaster(100, 100, 0, 0, 0.001, -0.001, 0, 0, 4269), '8BUI'::text, 1, 0)) width
```

Output

```
0.001
```

This example returns the pixel size of the rescaled raster.

Code

```
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
```

Output

```
0.0015
```

相关信息

[ST_Resize](#), [ST_Resample](#), [ST_SetScale](#), [ST_ScaleX](#), [ST_ScaleY](#), [ST_Transform](#)

11.7.9 ST_Reskew

ST_Reskew — 通过调整栅格的倾斜（或旋转参数）来重采样栅格。新的像素是使用 NearestNeighbor（英语或美式拼写）、Bilinear、Cubic、CubicSpline 或 Lanczos 重采样算法计算的。默认为 NearestNeighbor。

Synopsis

```
raster ST_Reskew(raster rast, double precision skewxy, text algorithm=NearestNeighbor, double precision maxerr=0.125);
```

```
raster ST_Reskew(raster rast, double precision skewx, double precision skewy, text algorithm=NearestNeighbor, double precision maxerr=0.125);
```

Description

通过调整栅格的斜率（或旋转参数）来重采样栅格。新的像素是使用 **NearestNeighbor**（英文或美式拼写）、**Bilinear**, **Cubic**, **CubicSpline** 或 **Lanczos** 重采样算法计算的。默认是“NearestNeighbor”，它速度最快，但插值效果最差。

`skewx` 和 `skewy` 定义新的斜率。

新栅格的范围将包含所提供栅格的范围。

如果未指定 `maxerr`，则 `maxerror` 百分比为 0.125。



Note

有关更多信息，请参考：[GDAL Warp 重采样方法](#)。



Note

`ST_Reskew` 与 `ST_SetSkew` 的不同之处在于 `ST_SetSkew` 不会将栅格重新采样以匹配栅格范围。`ST_SetSkew` 更改栅格的元数据（或地理参考）以纠正最初指定的斜率。`ST_Reskew` 会生成具有不同度和高度的栅格，计算结果以适合输入栅格的地理范围。`ST_SetSkew` 不修改栅格的度或高度。

可用性：2.0.0 需要 GDAL 1.6.1+

更改：2.1.0 适用于没有 SRID 的栅格

示例

将栅格从斜率 0.0 重新斜率到斜率 0.0015 的示例。

This example uses the original non-rotated raster.

Code

```
SELECT ST_Rotation(ST_AddBand(ST_MakeEmptyRaster(100, 100, 0, 0, 0.001, -0.001, 0, 0, 4269) ←
, '8BIT'::text, 1, 0));
```

Output

```
st_rotation
-----
0
(1 row)
```

The reskewed raster rotation.

Code

```
SELECT ST_Rotation(ST_Reskew(ST_AddBand(ST_MakeEmptyRaster(100, 100, 0, 0, 0.001, -0.001, ←
0, 0, 4269), '8BIT'::text, 1, 0), 0.0015));
```

Output

```
st_rotation
-----
-0.982793723247329
(1 row)
```

相关信息

[ST_Resample](#), [ST_Rescale](#), [ST_SetSkew](#), [ST_SetRotation](#), [ST_SkewX](#), [ST_SkewY](#), [ST_Transform](#)

11.7.10 ST_SnapToGrid

`ST_SnapToGrid` — 通过将栅格捕捉到网格来重采样栅格。新的像素是使用 `NearestNeighbor` (英式或美式拼写)、`Bilinear`、`Cubic`、`CubicSpline` 或 `Lanczos` 重采样算法计算的。默认为 `NearestNeighbor`。

Synopsis

```
raster ST_SnapToGrid(raster rast, double precision gridx, double precision gridy, text algorithm=NearestNeighbor, double precision maxerr=0.125, double precision scalex=DEFAULT 0, double precision scaley=DEFAULT 0);  
raster ST_SnapToGrid(raster rast, double precision gridx, double precision gridy, double precision scalex, double precision scaley, text algorithm=NearestNeighbor, double precision maxerr=0.125);  
raster ST_SnapToGrid(raster rast, double precision gridx, double precision gridy, double precision scalexy, text algorithm=NearestNeighbor, double precision maxerr=0.125);
```

Description

通过将栅格像素重新采样，将其映射到由任意像素角点 (`gridx` & `gridy`) 定义的网格，可本地使用像素大小 (`scalex` & `scaley`)。新的像素是使用最近邻 (`NearestNeighbor`，英式或美式拼写)、双线性 (`Bilinear`)、三次样条 (`Cubic`)、立方样条 (`CubicSpline`) 或兰索斯 (`Lanczos`) 重新采样算法计算的。默认情况下使用最近邻 (`NearestNeighbor`)，它是最快但插值效果最差的算法。

`gridx` 和 `gridy` 定义新网格的任意像素角。不一定是新栅格的左上角，也不必位于新栅格范围的内部或边界。

您可以使用 `scalex` 和 `scaley` 定义新网格的像素大小。

新栅格的范围将包含所提供栅格的范围。

如果未指定 `maxerr`，则 `maxerror` 百分比为 0.125。



Note

有关更多信息，请参考：[GDAL Warp 重采样方法](#)。



Note

如果您需要网格参数进行更多控制，请使用 [ST_Resample](#)。

可用性：2.0.0 需要 GDAL 1.6.1+

更改：2.1.0 适用于没有 SRID 的栅格

示例

将栅格捕捉到稍微不同的网格的示例。

This example returns the upper-left X coordinate of the original raster.

Code

```
SELECT ST_UpperLeftX(ST_AddBand(ST_MakeEmptyRaster(10, 10, 0, 0, 0.001, -0.001, 0, 0, 4269) ←
, '8BUI'::text, 1, 0));
```

Output

```
st_upperleftx
-----
0
(1 row)
```

The upper left of raster after snapping.

Code

```
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));
```

Output

```
st_upperleftx
-----
-0.0008
(1 row)
```

相关信息

[ST_Resample](#), [ST_Rescale](#), [ST_UpperLeftX](#), [ST_UpperLeftY](#)

11.7.11 ST_Resize

ST_Resize — 将栅格大小调整新的宽度/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 al-
gorithm=NearestNeighbor, double precision maxerr=0.125);
raster ST_Resize(raster rast, text width, text height, text algorithm=NearestNeighbor, double preci-
sion maxerr=0.125);
```

Description

将栅格大小调整新的宽度/高度。新的宽度/高度可以以精确的像素数或栅格宽度/高度的百分比来指定。新栅格的范围将与提供的栅格的范围相同。

新的像素值是使用 NearestNeighbor（英式或美式拼写）、Bilinear, Cubic, CubicSpline 或 Lanczos 重采样算法计算的。默认值是“NearestNeighbor”，它速度最快，但插值效果最差。

参数 1 期望输出栅格的宽度/高度。

参数 2 期望零 (0) 和一 (1) 之间的十进制数，表示输入栅格宽度的百分比。

参数 3 可以接受输出栅格图像的宽度/高度，也可以接受文本百分比（例如“20%”），表示输入栅格图像宽度的百分比。

可用性：2.1.0 需要 GDAL 1.6.1+

示例

Code

```

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
)
SELECT
  rid,
  ST_Width(rast) AS width,
  ST_Height(rast) AS height,
  ST_ScaleX(rast) AS scale_x,
  round(ST_ScaleY(rast)::numeric, 4) AS scale_y
FROM foo
ORDER BY rid

```

Output

rid	width	height	scale_x	scale_y
1	500	500	2	-2.0000
2	500	100	2	-10.0000
3	250	900	4	-1.1111

(3 rows)

相关信息

[ST_Resample](#), [ST_Rescale](#), [ST_Reskew](#), [ST_SnapToGrid](#)

11.7.12 ST_Transform

ST_Transform — 使用指定的重采样算法将已知空参考系中的栅格重新投影到一个已知空参考系。有 NearestNeighbor、Bilinear、Cubic、CubicSpline、Lanczos（默认为 NearestNeighbor）。

Synopsis

```

raster ST_Transform(raster rast, integer srid, text algorithm=NearestNeighbor, double precision
maxerr=0.125, double precision scalex, double precision scaley);
raster ST_Transform(raster rast, integer srid, double precision scalex, double precision scaley, text
algorithm=NearestNeighbor, double precision maxerr=0.125);
raster ST_Transform(raster rast, raster alignto, text algorithm=NearestNeighbor, double precision
maxerr=0.125);

```


Description

使用指定的像素扭曲算法将已知空参考系中的栅格重新投影到一个已知空参考系。如果未指定算法，使用“NearestNeighbor”；如果未指定 maxerr，使用 0.125 的 maxerror 百分比。

算法有：“NearestNeighbor”、“Bilinear”、“Cubic”、“CubicSpline”和“Lanczos”。有关更多信息，参： [GDAL Warp 重采样方法](#)。

ST_Transform 常与 ST_SetSRID() 混淆。ST_Transform 上将栅格的坐从上将栅格的坐从一种空参考系更改一种空参考系（并像素行重新采），而 ST_SetSRID() 只是更改栅格的 SRID 符。

与其他体不同，体 3 需要参考栅格作 alignto。后的栅格将参考栅格的空参考系 (SRID)，并与参考栅格 (ST_SameAlignment = TRUE)。



Note

如果坐功能异常，可能需要置境量 PROJSO，指向 PostGIS 使用的投影文件名。例如在 Windows 中，可以在“控制面板 -> 系 -> 境量”中新增系量 PROJSO，将其 libproj.dll。置完成后需要重 PostgreSQL 服/守程使其生效。



Warning

在覆盖范，您几乎是希望使用参考栅格来确保中的方式相同并且没有隙，如示例所示：体 3。

可用性：2.0.0 需要 GDAL 1.6.1+

增：2.1.0 添加 ST_Transform(rast,alignto) 体

示例

Compare a source raster with nearest-neighbor and bilinear reprojections. The diagonal bands make the resampling behavior visible: nearest-neighbor keeps hard pixel steps, while bilinear smooths values between pixels. All three figures are generated from this query at manual build time.

Code

```
WITH rows AS (
  SELECT
    y,
    array_agg(
      ((
        x * 3 + y * 5
        + CASE WHEN (x + y) % 8 < 4 THEN 80 ELSE 0 END
      ) % 255)::double precision
    ORDER BY x
  ) AS row_values
FROM generate_series(1, 80) AS y
CROSS JOIN generate_series(1, 80) AS x
GROUP BY y
), source AS (
  SELECT ST_SetValues(
    ST_AddBand(
      ST_MakeEmptyRaster(
        80, 80,
        -500000, 600000,
        2500, -2500,
        0, 0,
```

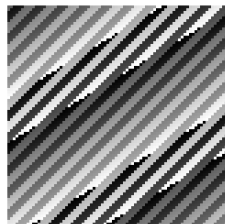
```

        2163
    ),
    1, '8BUI', 0, NULL
),
1, 1, 1,
array_agg(row_values ORDER BY y)::double precision[][])
) AS rast
FROM rows
), transformed AS (
SELECT
    rast,
    ST_Transform(rast, 4326, 'NearestNeighbor') AS nearest,
    ST_Transform(rast, 4326, 'Bilinear') AS bilinear
FROM source
)
SELECT
    ST_AsPNG(rast) AS "source (EPSG:2163)",
    ST_AsPNG(nearest) AS "nearest (EPSG:4326)",
    ST_AsPNG(bilinear) AS "bilinear (EPSG:4326)"
FROM transformed;

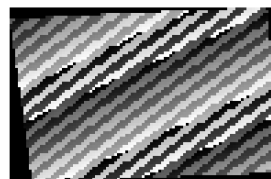
```

Figure

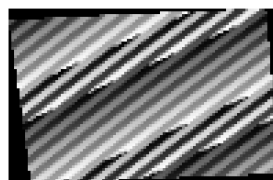
source (EPSG:2163)



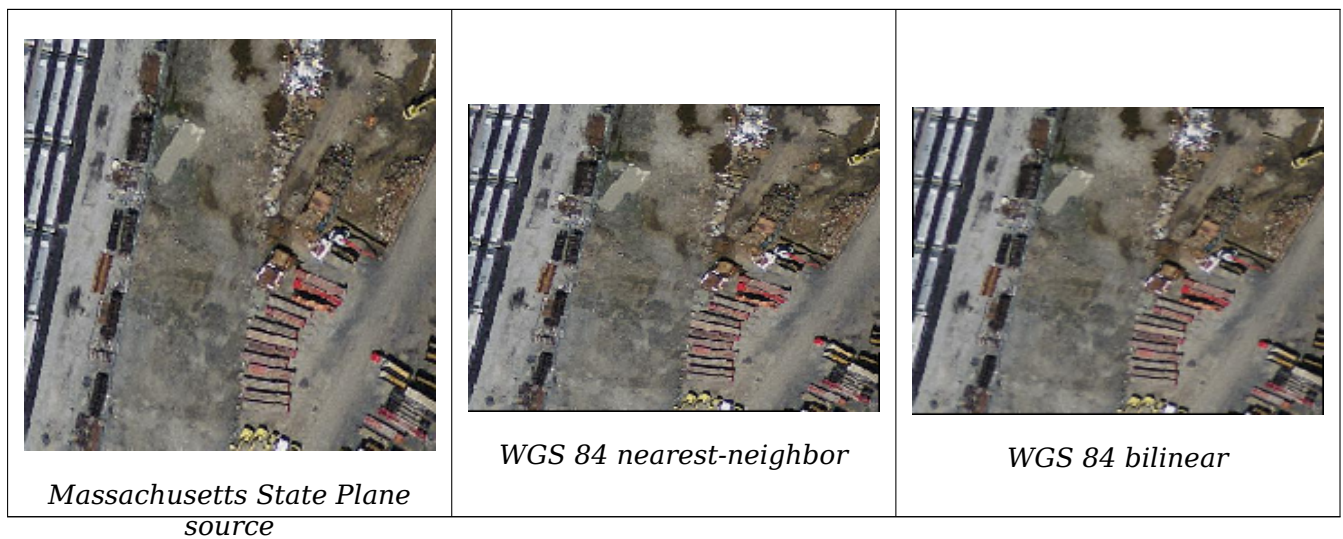
nearest (EPSG:4326)



bilinear (EPSG:4326)



Raster reprojection and resampling.



Variant 3.

下面展示了使用 `ST_Transform(raster, srid)` 和 `ST_Transform(raster, alignto)` 之区的区别

Code

```
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_Collect(ST_ConvexHull(not_aligned) ORDER BY rid) AS not_aligned,
```

```
ST_Collect(ST_ConvexHull(aligned) ORDER BY rid) AS aligned
FROM baz
```

Output

```
MULTIPOLYGON((( -107.007 50.2, -107.005 50.2, -107.005 50.198, -107.007 50.198, -107.007 50.2)), (( -107.004 50.201, -107.002 50.201, -107.002 50.198, -107.004 50.198, -107.004 50.201)), (( -107.001 50.201, -106.999 50.201, -106.999 50.198, -107.001 50.198, -107.001 50.201)), (( -107.007 50.199, -107.004 50.199, -107.004 50.196, -107.007 50.196, -107.007 50.199)), (( -107.004 50.199, -107.001 50.199, -107.001 50.196, -107.004 50.196, -107.004 50.199)), (( -107.001 50.199, -106.999 50.199, -106.999 50.196, -107.001 50.196, -107.001 50.199)), (( -107.006 50.197, -107.004 50.197, -107.004 50.194, -107.006 50.194, -107.006 50.197)), (( -107.004 50.197, -107.001 50.197, -107.001 50.194, -107.004 50.194, -107.004 50.197)), (( -107.001 50.197, -106.998 50.197, -106.998 50.195, -107.001 50.195, -107.001 50.197)), (( -107.007 50.2, -107.005 50.2, -107.005 50.198, -107.007 50.198, -107.007 50.2)), (( -107.005 50.202, -107.001 50.202, -107.001 50.198, -107.005 50.198, -107.005 50.202)), (( -107.002 50.202, -106.998 50.202, -106.998 50.198, -107.002 50.198, -107.002 50.202)), (( -107.007 50.199, -107.003 50.199, -107.003 50.195, -107.007 50.195, -107.007 50.199)), (( -107.005 50.199, -107.001 50.199, -107.001 50.195, -107.005 50.195, -107.005 50.199)), (( -107.002 50.199, -106.998 50.199, -106.998 50.195, -107.002 50.195, -107.002 50.199)), (( -107.007 50.198, -107.003 50.198, -107.003 50.194, -107.007 50.194, -107.007 50.198)), (( -107.005 50.198, -107.001 50.198, -107.001 50.194, -107.005 50.194, -107.005 50.198)), (( -107.002 50.198, -106.998 50.198, -106.998 50.194, -107.002 50.194, -107.002 50.198)))
```

Figure



相关信息

ST_Transform, ST_SetSRID

11.8 栅格波段处理器

11.8.1 ST_SetBandNoDataValue

ST_SetBandNoDataValue — 设置代表无数据的指定波段的值。如果未指定波段，则假定波段 1。要将波段设置为没有 nodata 值，则设置 nodata 值 = NULL。

Synopsis

raster **ST_SetBandNoDataValue**(raster rast, double precision nodatavalue);
 raster **ST_SetBandNoDataValue**(raster rast, integer band, double precision nodatavalue, boolean forcechecking=false);

Description

置表示没有数据的。如果未指定，假定波段 1。将影 **ST_Polygon**、**ST_DumpAsPolygons** 和 **ST_PixelAs...**() 函数的果。

示例

This example changes only the first band NODATA value.

Code

```
UPDATE dummy_rast
  SET rast = ST_SetBandNoDataValue(rast, 1, 254)
WHERE rid = 2;
```

This example changes the NODATA value for bands 1, 2, and 3.

Code

```
UPDATE dummy_rast
  SET rast =
    ST_SetBandNoDataValue(ST_SetBandNoDataValue(ST_SetBandNoDataValue(rast, 1, 254),
      2, 99),
      3, 108)
  WHERE rid = 2;
```

This example removes the NODATA value so that all pixels are considered by processing functions.

Code

```
UPDATE dummy_rast
  SET rast = ST_SetBandNoDataValue(rast, 1, NULL)
WHERE rid = 2;
```

相关信息

ST_BandNoDataValue, **ST_NumBands**

11.8.2 ST_SetBandIsNoData

ST_SetBandIsNoData — 将区的 isnodata 志置 TRUE。

Synopsis

raster **ST_SetBandIsNoData**(raster rast, integer band=1);

Description

将 `isnodata` 标志置为 `true`。如果未指定，将假定第 1 个。只有在被设置标志时才用此函数。也就是，当用 `ST_BandIsNoData` 作最后一个参数使用 `TRUE` 和不使用 `TRUE` 的结果不同。

可用性: 2.0.0

示例

This example creates a dummy table with one raster column. The raster has two bands and one pixel per band. In the first band, the NODATA value and pixel value are both 3. In the second band, the NODATA value is 13 and the pixel value is 4.

Code

```
create table dummy_rast (rid integer, rast raster);

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, so set it to true.

Code

```
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
```

相关信息

[ST_BandNoDataValue](#), [ST_NumBands](#), [ST_SetBandNoDataValue](#), [ST_BandIsNoData](#)

11.8.3 ST_SetBandPath

ST_SetBandPath — 更新 out-db band 的外部路径和 band 号

Synopsis

raster **ST_SetBandPath**(raster rast, integer band, text outdbpath, integer outdbindex, boolean force=false)

Description

更新 out-db band 的外部栅格文件路径和外部波段号。



Note

如果 force 置 true, 不会行任何来确保外部栅格文件和 PostGIS 栅格之的兼容性 (例如 8b、像素支持)。此模式适用于外部栅格所在的文件系统更改。

可用性 : 2.5.0

示例

Code

```
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,
```

```
bandnum,
regexp_replace(path, '^.*/', '') AS path,
outdbbandnum
FROM ST_BandMetaData(
  (SELECT rast FROM foo),
  ARRAY[1, 3, 2]::int[]
) AS metadata
UNION ALL
SELECT
  2, bandnum,
  regexp_replace(path, '^.*/', ''),
  outdbbandnum
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[]
) AS metadata
ORDER BY 1, 2;
```

In the second result set, band 2 now points at Projected2.tif and uses out-db band 1.

Output

query	bandnum	path	outdbbandnum
1	1	Projected.tif	1
1	2	Projected.tif	2
1	3	Projected.tif	3
2	1	Projected.tif	1
2	2	Projected2.tif	1
2	3	Projected.tif	3

相关信息

[ST_BandMetaData](#), [ST_SetBandIndex](#)

11.8.4 ST_SetBandIndex

ST_SetBandIndex — 更新 out-db band 的 external band 号

Synopsis

raster **ST_SetBandIndex**(raster rast, integer band, integer outdbindex, boolean force=false);

Description

更新 out-db band 的外部 band 号。不会触及与 out-db band 关联的外部栅格文件



Note

如果 `force` 置 `true`，不会行任何来确保外部栅格文件和 PostGIS 栅格之的兼容性（例如、像素支持）。此模式适用于在外部栅格文件中移波段的情况。



Note

在内部，此方法将索引 `band` 的 PostGIS 栅格波段替新波段，而不是更新有的路径信息。

可用性 : 2.5.0

示例

Code

```
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,  
    bandnum,  
    regexp_replace(path, '^.*/', '') AS path,  
    outdbbandnum  
FROM ST_BandMetaData(  
    (SELECT rast FROM foo),  
    ARRAY[1, 3, 2]::int[]  
) AS metadata  
UNION ALL  
SELECT  
    2, bandnum,  
    regexp_replace(path, '^.*/', ''),  
    outdbbandnum  
FROM ST_BandMetaData((  
    SELECT  
        ST_SetBandIndex(rast,  
            2,  
            1  
        ) AS rast  
    FROM foo  
    ),  
    ARRAY[1, 3, 2]::int[]  
) AS metadata  
ORDER BY 1, 2;
```

In the second result set, band 2 still points at the same file but now reads external band 1.

Output

query	bandnum	path	outdbbandnum
1	1	Projected.tif	1
1	2	Projected.tif	2
1	3	Projected.tif	3
2	1	Projected.tif	1
2	2	Projected.tif	1
2	3	Projected.tif	3

相关信息

[ST_BandMetaData](#), [ST_SetBandPath](#)

11.9 栅格波段和分析

11.9.1 ST_Count

`ST_Count` — 返回栅格或栅格覆盖范围的指定波段中的像素数。如果未指定 `band`，默认为 `band 1`。如果 `except_nodata_value` 设置为 `true`，则计算不等于 `nodata` 的像素。

Synopsis

```
bigint ST_Count(raster rast, integer nband=1, boolean exclude_nodata_value=true);
bigint ST_Count(raster rast, boolean exclude_nodata_value);
```

Description

返回栅格或栅格覆盖范围的指定波段中的像素数。如果未指定 `band`，则 `nband` 默认为 `1`。



Note
如果 `except_nodata_value` 设置为 `true`，则计算不等于栅格 `nodata` 的像素。将 `except_nodata_value` 设置为 `false` 以计算所有像素

更改：3.1.0 - 除了 `ST_Count(rastertable, rastercolumn, ...)` 外。改用 [ST_CountAgg](#)。

可用性: 2.0.0

示例

Example will count all pixels not 249 and one will count all pixels.

Code

```
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;
```

Output

rid	exclude_nodata	include_nodata
2	23	25

相关信息

[ST_CountAgg](#), [ST_SummaryStats](#), [ST_SetBandNoDataValue](#)

11.9.2 ST_CountAgg

ST_CountAgg — 统计。返回一栅格的指定波段中的像素数。如果未指定 **band**，默认为 **band 1**。如果 **except_nodata_value** 置为 **true**，则只计算不等于 **NODATA** 的像素。

Synopsis

```
bigint ST_CountAgg(raster rast, integer nband, boolean exclude_nodata_value, double precision
sample_percent);
bigint ST_CountAgg(raster rast, integer nband, boolean exclude_nodata_value);
bigint ST_CountAgg(raster rast, boolean exclude_nodata_value);
```

Description

返回一栅格的指定波段中的像素数。如果未指定 **band**，则 **nband** 默认为 **1**。

如果 **except_nodata_value** 置为 **true**，则只计算不等于栅格 **NODATA** 的像素。将 **except_nodata_value** 置为 **false** 以计算所有像素。

默认情况下将只计算所有像素的行采。要获得更快的速度，可将 **sample_percent** 置为零 (0) 到一 (1) 之间的值。
可用性：2.2.0

示例

Code

```
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) AS non_nodata_pixels
FROM foo;
```

Output

```
non_nodata_pixels
-----
                20
(1 row)
```

相关信息

[ST_Count](#), [ST_SummaryStats](#), [ST_SetBandNoDataValue](#)

11.9.3 ST_Histogram

ST_Histogram — 返回一个数组，数组了网格或网格覆盖数据分布，其中包括分离的分箱范围。如果未指定，将自动计算分箱数。

Synopsis

```
SETOF record ST_Histogram(raster rast, integer nband=1, boolean exclude_nodata_value=true, integer bins=autocomputed, double precision[] width=NULL, boolean right=false);
SETOF record ST_Histogram(raster rast, integer nband, integer bins, double precision[] width=NULL, boolean right=false);
SETOF record ST_Histogram(raster rast, integer nband, boolean exclude_nodata_value, integer bins, boolean right);
SETOF record ST_Histogram(raster rast, integer nband, integer bins, boolean right);
```

Description

返回每个 bin 指定网格波段的数组集，其中包含最小值、最大值、计数、百分比。如果未指定 band，则 nband 默认为 1。



Note

默认情况下，考虑不等于 nodata 的像素。将 `except_nodata_value` 置为 false 以计算所有像素。

width width：一个数组，指示每个 category/bin 的宽度。如果箱的数量大于宽度的数量，则宽度会重复。

示例：9 个 bin，宽度 [a, b, c] 将输出 [a, b, c, a, b, c, a, b, c]

bins 分箱数量——这是您将从函数中返回的数组数（如果指定）。如果未指定，则自动计算分箱数量。

right 从右而不是从左计算直方图（默认）。将估计 x 的准从 [a, b) 更改为 (a, b]

更改：3.1.0 除了 `ST_Histogram(table_name, column_name)` 变体。

可用性：2.0.0

示例

Single raster tile - compute histograms for bands 1, 2, 3 and autocompute bins.

Code

```
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;
```

Output

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

Just band 2 but for 6 bins.

Code

```
SELECT (stats).*
FROM (SELECT rid, ST_Histogram(rast, 2, 6) As stats
      FROM dummy_rast
      WHERE rid=2) As foo;
```

Output

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.

Code

```
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;
```

Output

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)

相关信息

[ST_Count](#), [ST_SummaryStats](#), [ST_SummaryStatsAgg](#)

11.9.4 ST_Quantile

`ST_Quantile` — 计算本或体上下文中格或格表覆盖范围的分位数。因此，可以判断某个格是否位于格的 25%、50%、75% 百分位。

Synopsis

SETOF record **ST_Quantile**(raster rast, integer nband=1, boolean exclude_nodata_value=true, double precision[] quantiles=NULL);
SETOF record **ST_Quantile**(raster rast, double precision[] quantiles);
SETOF record **ST_Quantile**(raster rast, integer nband, double precision[] quantiles);
double precision **ST_Quantile**(raster rast, double precision quantile);
double precision **ST_Quantile**(raster rast, boolean exclude_nodata_value, double precision quantile=NULL);
double precision **ST_Quantile**(raster rast, integer nband, double precision quantile);
double precision **ST_Quantile**(raster rast, integer nband, boolean exclude_nodata_value, double precision quantile);
double precision **ST_Quantile**(raster rast, integer nband, double precision quantile);

Description

计算本或体上下文中格或格表覆盖范围的分位数。因此，可以某个是否位于格的 25%、50%、75% 百分位。



Note

如果 `except_nodata_value` 置 `false`，将没有数据的像素行数。

更改：3.1.0 除了 `ST_Quantile(table_name, column_name)` 体。

可用性: 2.0.0

示例

Example will consider only pixels of band 1 that are not 249 and in named quantiles.

Code

```
UPDATE dummy_rast SET rast = ST_SetBandNoDataValue(rast, 249) WHERE rid=2;

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;
```

Output

quantile	value
0.25	253
0.75	254

Code

```
SELECT ST_Quantile(rast, 0.75) As value
FROM dummy_rast WHERE rid=2;
```

| **Output**

254

Real live example. Quantile of all pixels in band 2 intersecting a geometry.

| **Code**

```
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
;
```

| **Output**

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

相关信息

[ST_Count](#), [ST_SummaryStats](#), [ST_SummaryStatsAgg](#), [ST_SetBandNoDataValue](#)

11.9.5 ST_SummaryStats

`ST_SummaryStats` — 返回指定栅格或栅格覆盖中某个波段的统计摘要，包括像元数量 (`count`)、总和 (`sum`)、均值 (`mean`)、标准差 (`stddev`)、最小值 (`min`) 和最大值 (`max`)。若未指定波段，默认使用第 1 波段。

Synopsis

`summarystats` **ST_SummaryStats**(raster rast, boolean exclude_nodata_value);
`summarystats` **ST_SummaryStats**(raster rast, integer nband, boolean exclude_nodata_value);

Description

返回栅格或栅格覆盖范围的指定栅格波段的 `summarystats` 统计信息，其中包含像元数、总和、平均值、标准差、最小值、最大值。如果未指定 `band`，则 `nband` 默认为 1。

**Note**

默认情况下，`ST_SummaryStats` 不等于 `nodata` 的像素数。将 `except_nodata_value` 置为 `false` 以获取所有像素的计数。

**Note**

默认情况下将遍历所有像素。为了获得更快的速度，将 `sample_percent` 置为低于 1

更改：3.1.0 `ST_SummaryStats(rastertable, rastercolumn, ...)` 函数已移除。改用 `ST_SummaryStatsAgg`。
可用性：2.0.0

示例

Single raster tile.

Code

```
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;
```

Output

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

Summarize pixels that intersect buildings of interest.

This example selects the target buildings, clips band 2 of the raster tiles to each building boundary, computes statistics for each clipped region, and then summarizes those statistics by building. It took 574ms on PostGIS Windows 64-bit with all of Boston Buildings and aerial Tiles (tiles each 150x150 pixels ~ 134,000 tiles), ~102,000 building records.

Code

```
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 aeriels.boston
    INNER JOIN feat
    ON ST_Intersects(feats.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;
```

Output

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

Raster coverage.

This example returns statistics for each band.

Code

```
SELECT band, (stats).*
FROM (SELECT band, ST_SummaryStats('o_4_boston', 'rast', band) As stats
      FROM generate_series(1, 3) As band) As foo;
```

Output

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, sampling less than 100% can improve speed. Here the sampling is set to 25%.

Code

```
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;
```

Output

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

相关信息

[summarystats](#), [ST_SummaryStatsAgg](#), [ST_Count](#), [ST_Clip](#)

11.9.6 ST_SummaryStatsAgg

ST_SummaryStatsAgg — 聚合函数。返回一栅格中指定波段的统计摘要，包括像元数量 (count)、和 (sum)、均值 (mean)、标准差 (stddev)、最小值 (min) 和最大值。若未指定波段，默认使用第 1 波段。

Synopsis

```
summarystats ST_SummaryStatsAgg(raster rast, integer nband, boolean exclude_nodata_value, double precision sample_percent);
summarystats ST_SummaryStatsAgg(raster rast, boolean exclude_nodata_value, double precision sample_percent);
summarystats ST_SummaryStatsAgg(raster rast, integer nband, boolean exclude_nodata_value);
```

Description

返回栅格或栅格覆盖范围的指定栅格波段的 **summarystats** 信息，其中包含数、和、平均值、标准差、最小值、最大值。如果未指定 band，nband 默认为 1。



Note

默认情况下，考虑不等于 NODATA 值的像素。将 `except_nodata_value` 置为 `False` 以获取所有像素的数。



Note

默认情况下将所有像素行采样。为了获得更快的结果，将 `sample_percent` 置为 0 到 1 之间的值。

可用性 : 2.2.0

示例

Code

```
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 AS count,
  round((stats).sum::numeric, 3) AS sum,
  round((stats).mean::numeric, 3) AS mean,
  round((stats).stddev::numeric, 3) AS stddev,
  round((stats).min::numeric, 3) AS min,
  round((stats).max::numeric, 3) AS max
FROM (
  SELECT
    ST_SummaryStatsAgg(rast, 1, TRUE, 1) AS stats
  FROM foo
) bar;
```

Output

count	sum	mean	stddev	min	max
20	-68.584	-3.429	6.571	-10.000	3.142
(1 row)					

相关信息

[summarystats](#), [ST_SummaryStats](#), [ST_Count](#), [ST_Clip](#)

11.9.7 ST_ValueCount

ST_ValueCount — 返回一维数组，其中包含像素值以及具有指定值的栅格（或栅格覆盖范围）指定中的像素数。如果未指定波段，默认波段 1。默认情况下，不算点数据像素。输出像素中的所有其他值，并将像素值四舍五入到最接近的整数。

Synopsis

SETOF record **ST_ValueCount**(raster rast, integer nband=1, boolean exclude_nodata_value=true, double precision[] searchvalues=NULL, double precision roundto=0, double precision OUT value, integer OUT count);

SETOF record **ST_ValueCount**(raster rast, integer nband, double precision[] searchvalues, double precision roundto=0, double precision OUT value, integer OUT count);

SETOF record **ST_ValueCount**(raster rast, double precision[] searchvalues, double precision roundto=0, double precision OUT value, integer OUT count);

bigint **ST_ValueCount**(raster rast, double precision searchvalue, double precision roundto=0);

bigint **ST_ValueCount**(raster rast, integer nband, boolean exclude_nodata_value, double precision searchvalue, double precision roundto=0);

bigint **ST_ValueCount**(raster rast, integer nband, double precision searchvalue, double precision roundto=0);

SETOF record **ST_ValueCount**(text rastertable, text rastercolumn, integer nband=1, boolean exclude_nodata_value=true, double precision[] searchvalues=NULL, double precision roundto=0, double precision OUT value, integer OUT count);

SETOF record **ST_ValueCount**(text rastertable, text rastercolumn, double precision[] searchvalues, double precision roundto=0, double precision OUT value, integer OUT count);

SETOF record **ST_ValueCount**(text rastertable, text rastercolumn, integer nband, double precision[] searchvalues, double precision roundto=0, double precision OUT value, integer OUT count);

bigint **ST_ValueCount**(text rastertable, text rastercolumn, integer nband, boolean exclude_nodata_value, double precision searchvalue, double precision roundto=0);

bigint **ST_ValueCount**(text rastertable, text rastercolumn, double precision searchvalue, double precision roundto=0);
bigint **ST_ValueCount**(text rastertable, text rastercolumn, integer nband, double precision searchvalue, double precision roundto=0);

Description

返回一列具有列值的行，其中包含像素波段以及所波段的栅格切片或栅格覆盖范中的像素数。
如果未指定波段， nband 默 1。如果未指定搜索值，将返回在栅格或栅格覆盖范中找到的所有像素。如果指定一个搜索值，将返回一个整数，而不是表示具有像素值的像素数的行。



Note

如果 except_nodata_value 置 false，将没有数据的像素行。

可用性: 2.0.0

示例

Example will count only pixels of band 1 that are not 249.

Code

```
UPDATE dummy_rast SET rast = ST_SetBandNoDataValue(rast, 249) WHERE rid=2;

SELECT (pvc).*
FROM (SELECT ST_ValueCount(rast) As pvc
      FROM dummy_rast WHERE rid=2) As foo
ORDER BY (pvc).value;
```

Output

value	count
250	2
251	1
252	2
253	6
254	12

Example will count all pixels of band 1 including 249.

Code

```
SELECT (pvc).*
FROM (SELECT ST_ValueCount(rast, 1, false) As pvc
      FROM dummy_rast WHERE rid=2) As foo
ORDER BY (pvc).value;
```

Output

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.

Code

```
SELECT (pvc).*
FROM (SELECT ST_ValueCount(rast, 2) As pvc
      FROM dummy_rast WHERE rid=2) As foo
ORDER BY (pvc).value;
```

Output

```
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.

Code

```
SELECT (pvc).value, SUM((pvc).count) As total
FROM (SELECT ST_ValueCount(rast, 2) As pvc
      FROM o_4_boston
      WHERE ST_Intersects(
            rast,
            ST_GeomFromText('POLYGON((224486 892151,224486 892200,224706 892200,224706 892151,224486 892151))', 26986)
            ) As foo
      GROUP BY (pvc).value
      HAVING SUM((pvc).count) > 500
      ORDER BY (pvc).value;
```

Output

```
value | total
-----+-----
51    | 502
54    | 521
```

Just return count of pixels in each raster tile that have value of 100 of tiles that intersect a specific geometry.

Code

```
SELECT rid, ST_ValueCount(rast, 2, 100) As count
FROM o_4_boston
WHERE ST_Intersects(
      rast,
      ST_GeomFromText('POLYGON((224486 892151,224486 892200,224706 892200,224706 892151,224486 892151))', 26986)
      ) ;
```

Output

rid	count
1	56
2	95
14	37
15	64

相关信息

ST_Count, ST_SetBandNoDataValue

11.10 格入

11.10.1 ST_RastFromWKB

ST_RastFromWKB — 从熟知的二进制 (WKB) 格式返回格式。

Synopsis

```
raster ST_RastFromWKB(bytea wkb);
```

Description

返回定熟知的二进制 (WKB) 格式, 返回一个字节。

可用性：2.5.0

示例

Code

[illegible]

Output

upperleftx	upperlefty	width	height	scalex	scaley	skewx	skewy	srid	numbands
0.5	0.5	10	20	2	3	0	0	10	0

相关信息

ST MetaData, ST RastFromHexWKB, ST AsBinary/ST AsWKB, ST AsHexWKB

Code

```

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;

```

Export a raster as a GeoTIFF using the default creation options.

Code

```

SELECT ST_AsGDALRaster(rast, 'GTiff') AS rasttiff
FROM dummy_rast WHERE rid = 2;

```

Export a GeoTIFF using JPEG compression at 90 percent quality and embed SRID 4269.

Code

```

SELECT ST_AsGDALRaster(
    rast,
    'GTiff',
    ARRAY['COMPRESS=JPEG', 'JPEG_QUALITY=90'],
    4269) AS rasttiff
FROM dummy_rast WHERE rid = 2;

```

相关信息

Section [10.4](#), [ST_GDALDrivers](#), [ST_SRID](#)

11.11.4 ST_AsJPEG

ST_AsJPEG — 将给定的波段作成一个合影图 (JPEG) 图像 (字节数) 返回。如果未指定波段且有 1 个或 3 个以上波段，使用第一个波段。如果只有 3 个波段，使用所有 3 个波段并将其映射到 RGB。

Synopsis

```

bytea ST_AsJPEG(raster rast, text[] options=NULL);
bytea ST_AsJPEG(raster rast, integer nband, integer quality);
bytea ST_AsJPEG(raster rast, integer nband, text[] options=NULL);
bytea ST_AsJPEG(raster rast, integer[] nbands, text[] options=NULL);
bytea ST_AsJPEG(raster rast, integer[] nbands, integer quality);

```

Description

将栅格的指定波段作成一个合成影像输出图像 (JPEG) 返回。如果需要输出不太常见的栅格类型，使用 **ST_AsGDALRaster**。如果未指定波段且有 1 个或 3 个以上波段，使用第一个波段。如果有 3 个波段，使用所有 3 个波段。功能有多媒体和多波段。以下逐条列出：

- **nband** 用于波段输出。
- **nbands** 是要输出的波段数（注意，JPEG 的最大为 3），波段的顺序 RGB。例如 `ARRAY[3,2,1]` 表示将波段 3 映射到红色，将波段 2 映射到绿色，将波段 1 映射到蓝色。
- **quality** 数字从 0 到 100。数字越高，图像越清晰。
- **options text** JPEG 定义的 GDAL 选项数（查看 JPEG **ST_GDALDrivers** 的 `create_options`）。对于 JPEG，有效的是 `PROGRESSIVE ON` 或 `OFF`，`QUALITY` 的范围 0 到 100，默认为 75。有关更多信息，参看 **GDAL 光栅格式**。

可用性：2.0.0 - 需要 GDAL >= 1.6.0。

示例：输出

Output the first three bands at 75 percent quality.

Code

```
SELECT ST_AsJPEG(rast) As rastjpg
FROM dummy_rast WHERE rid=2;
```

Output only the first band at 90 percent quality.

Code

```
SELECT ST_AsJPEG(rast, 1, 90) As rastjpg
FROM dummy_rast WHERE rid=2;
```

Output the first three bands, mapping band 2 to red, band 1 to green, and band 3 to blue, using progressive encoding at 90 percent quality.

Code

```
SELECT ST_AsJPEG(rast, ARRAY[2, 1, 3], ARRAY['QUALITY=90', 'PROGRESSIVE=ON']) As rastjpg
FROM dummy_rast WHERE rid=2;
```

相关信息

Section [10.4](#), [ST_GDALDrivers](#), [ST_AsGDALRaster](#), [ST_AsPNG](#), [ST_AsTIFF](#)

11.11.5 ST_AsPNG

ST_AsPNG — 将栅格指定的波段作成一个便携式网络图形 (PNG) 图像（字节数）返回。如果栅格中有 1、3 或 4 个波段且未指定波段，使用所有波段。如果有 2 个以上或多于 4 个波段且未指定波段，使用波段 1。波段映射到 RGB 或 RGBA 空间。

Synopsis

```
bytea ST_AsPNG(raster rast, text[] options=NULL);
bytea ST_AsPNG(raster rast, integer nband, integer compression);
bytea ST_AsPNG(raster rast, integer nband, text[] options=NULL);
bytea ST_AsPNG(raster rast, integer[] nbands, integer compression);
bytea ST_AsPNG(raster rast, integer[] nbands, text[] options=NULL);
```

Description

将栅格的指定波段作成一个便携式网络图形图像 (PNG) 返回。如果需要输出不太常见的栅格类型，使用 [ST_AsGDALRaster](#)。如果未指定波段，输出前 3 个波段。功能有多种变体和多种选项。如果未指定 `srid`，使用栅格的 `srid`。以下逐条列出：

- `nband` 用于输出波段。
- `nbands` 是要输出的波段数（注意，PNG 的最大值为 4），波段的顺序为 RGBA。例如 `ARRAY[3,2,1]` 表示将波段 3 映射到红色，将波段 2 映射到绿色，将波段 1 映射到蓝色。
- `compression` 编号从 1 到 9。数字越大，压缩越大。
- `options text` 为 PNG 定义的 GDAL 选项数（查看 [ST_GDALDrivers](#) 的 PNG 的 `create_options`）。对于 PNG 来说，有效的只是 `ZLEVEL`（压缩所花费的字节——默认为 6），例如数组 `['ZLEVEL=9']`。不允许 `WORLDFILE`，因为该函数必须输出一个输出。有关更多信息，参考 [GDAL 文件格式](#)。

可用性：2.0.0 - 需要 GDAL >= 1.6.0。

示例

Export the first 3 bands and map band 3 to red, band 1 to green, and band 2 to blue.

Code

```
SELECT ST_AsPNG(rast) As rastpng
FROM dummy_rast WHERE rid=2;
SELECT ST_AsPNG(rast, ARRAY[3, 1, 2]) As rastpng
FROM dummy_rast WHERE rid=2;
```

相关信息

[ST_AsGDALRaster](#), [ST_ColorMap](#), [ST_GDALDrivers](#), [Section 10.4](#)

11.11.6 ST_AsTIFF

`ST_AsTIFF` — 将栅格指定的波段作成一个 TIFF 图像（字节数）返回。如果未指定波段或栅格中不存在任何指定波段，将使用所有波段。

Synopsis

```
bytea ST_AsTIFF(raster rast, text[] options="", integer srid=sameassource);
bytea ST_AsTIFF(raster rast, text compression="", integer srid=sameassource);
bytea ST_AsTIFF(raster rast, integer[] nbands, text compression="", integer srid=sameassource);
bytea ST_AsTIFF(raster rast, integer[] nbands, text[] options, integer srid=sameassource);
```

Description

以单个栅格像文件格式 (TIFF) 返回栅格的指定波段。如果未指定波段, 将使用所有波段。它是 [ST_AsGDALRaster](#) 的包装。如果需要输出不太常见的栅格类型, 使用 [ST_AsGDALRaster](#)。该功能有多种变体和选项。如果不存在空参考 SRS 文本, 使用栅格的空参考。以下逐条列出:

- **nbands** 是要输出的波段数 (注意, PNG 的最大为 3), 波段的顺序 RGB。例如 `ARRAY[3,2,1]` 表示将波段 3 映射到红色, 将波段 2 映射到绿色, 将波段 1 映射到蓝色
- **compression** `Compression` 表式——JPEG90 (或其他百分比)、LZW、JPEG、DEFLATE9。
- **options text** 是 GTiff 定义的 GDAL 创建选项 (查看 [ST_GDALDrivers](#) 的 GTiff 的 `create_options`)。或参看 [GDAL 文件格式](#) 了解更多信息。
- **srid** 栅格的 `spatial_ref_sys` 的 srid。用于填充地理参考信息

可用性: 2.0.0 - 需要 GDAL >= 1.6.0。

示例: 使用 **jpeg** 压缩 **90%**

Code

```
SELECT ST_AsTIFF(rast, 'JPEG90') As rasttiff
FROM dummy_rast WHERE rid=2;
```

相关信息

[ST_GDALDrivers](#), [ST_AsGDALRaster](#), [ST_SRID](#)

11.12 栅格地理: 地理代数

11.12.1 ST_Clip

ST_Clip — 返回由输入几何裁剪的栅格。如果未指定波段号, 处理所有波段。如果未指定 **crop** 或其为 **TRUE**, 输出栅格将被裁剪。如果 **touched** 置为 **TRUE**, 包括接触到的像素; 否则, 当像素的中心位于几何内, 才会包括该像素。

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);
```

Description

返回由输入几何体 `geom` 裁剪的栅格。如果未指定波段索引，则处理所有波段。

`ST_Clip` 生成的栅格必须对剪切区域分配一个点数据值，每个波段一个。如果未提供任何内容，并且输入栅格未定义 `nodata` 值，则生成的栅格的 `nodata` 值将设置为 `ST_MinPossibleValue(ST_BandPixelType(rast, band))`。当数集中的 `nodata` 值的数量小于波段的数量时，数集中的最后一个值用于剩余的波段。如果点数据值的数量大于波段数量，则忽略多余的点数据值。所有接受点数据值的值体也接受将分配给每个波段的值。

如果未指定 `crop`，则假定 `true`，表示输出栅格将被裁剪到 `geom` 和 `rast` 范围的交集。如果将 `crop` 设置为 `false`，则新的栅格将具有与 `rast` 相同的范围。如果将 `touched` 设置为 `true`，则将与几何形状相交的 `rast` 中的所有像素。



Note

默认行为是 `touched=false`，将返回像素中心被几何形状覆盖的像素。

增本版：3.5.0 - 添加了 `touched` 参数。

可用性：2.0.0

增版：2.1.0 用 C 重写

其中的示例使用马萨诸塞州航拍数据，数据可在 MassGIS 网站的 [MassGIS Aerial Orthos](#) 页面上找到。

示例

Comparing selecting all touched vs. not all touched.

Code

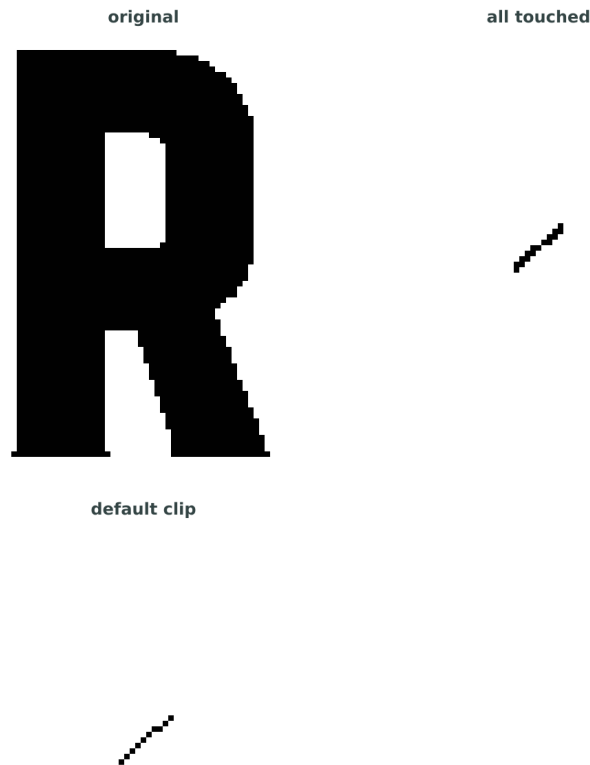
```
WITH source AS (
  SELECT rast, geom
  FROM ST_AsRaster(ST_Letters('R'), scalex =
> 1.0, scaley =
> -1.0) AS r(rast)
  INNER JOIN ST_GeomFromText('LINESTRING(0 1,5 6,10 10)') AS g(geom)
  ON ST_Intersects(r.rast, g.geom)
), clipped AS (
  SELECT rast,
    ST_Clip(rast, geom, touched =
> true) AS rast_touched,
    ST_Clip(rast, geom, touched =
> false) AS rast_not_touched
  FROM source
), variants AS (
  SELECT 'original' AS title, ST_Count(rast) AS pixel_count, rast AS rendered
  FROM clipped
  UNION ALL
  SELECT 'all touched', ST_Count(rast_touched), rast_touched
  FROM clipped
  UNION ALL
  SELECT 'default clip', ST_Count(rast_not_touched), rast_not_touched
  FROM clipped
)
SELECT title, pixel_count, ST_AsPNG(rendered) AS image
FROM variants
ORDER BY CASE title
  WHEN 'original' THEN 1
```

```

WHEN 'all touched' THEN 2
ELSE 3
END;

```

Figure



The following self-contained example uses three simple shapes as RGB bands. It compares cropping band 1, clipping band 1 without cropping and then restoring bands 2 and 3, and clipping every band without cropping.

Code

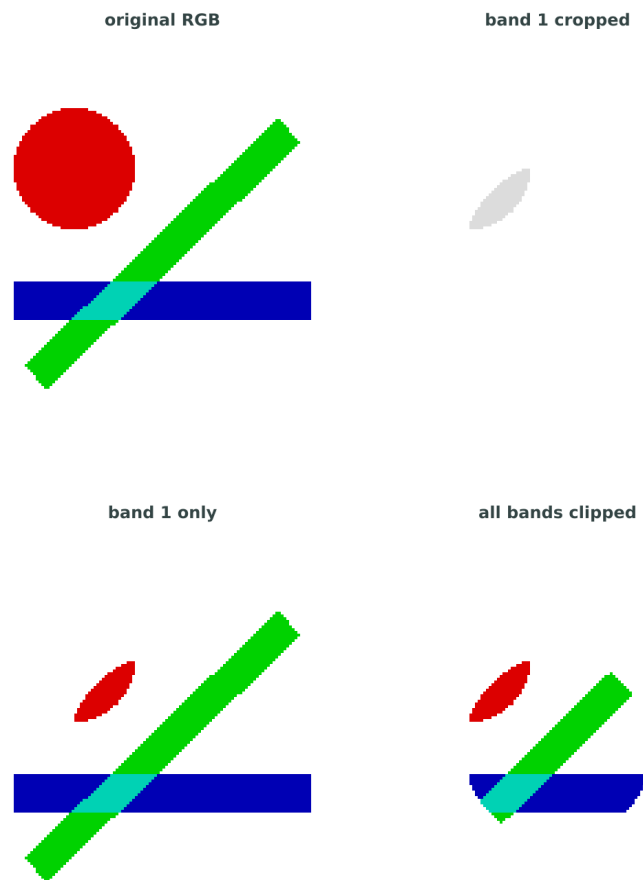
```

WITH canvas AS (
  SELECT ST_AddBand(
    ST_AddBand(
      ST_AddBand(
        ST_MakeEmptyRaster(128, 128, 0, 128, 1, -1, 0, 0, 0),
        '8BUI'::text, 0::double precision, 0::double precision
      ),
      '8BUI'::text, 0::double precision, 0::double precision
    ),
    '8BUI'::text, 0::double precision, 0::double precision
  ) AS rast
), source AS (
  SELECT ST_SetValue(

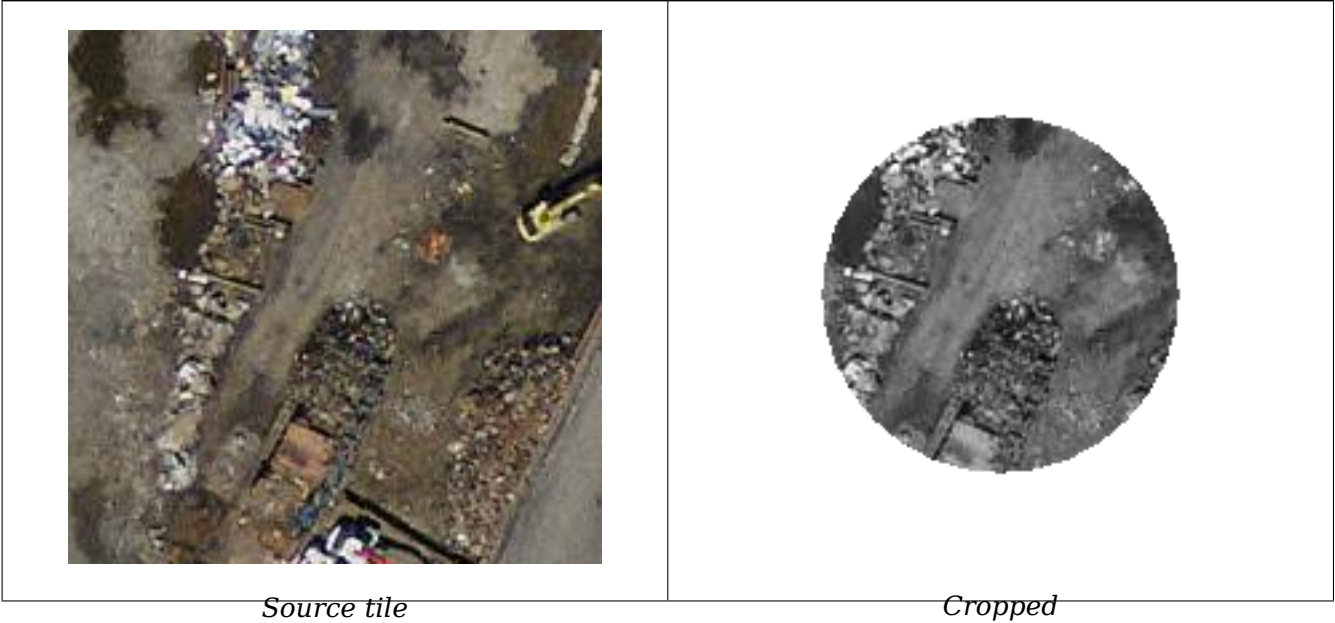
```

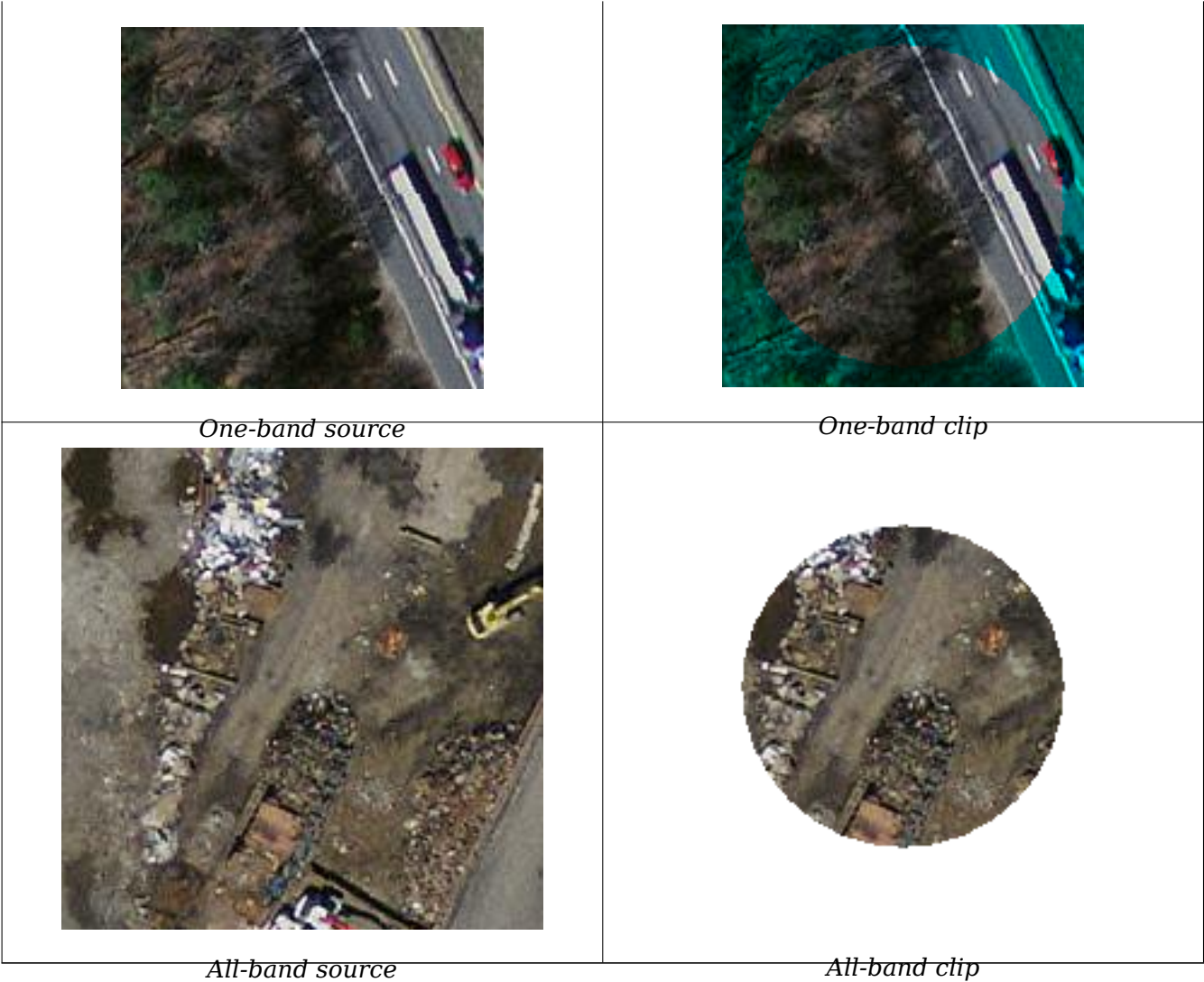
```
        ST_SetValue(
            ST_SetValue(rast, 1, ST_Buffer('POINT(32 96)::geometry, 22), 220),
            2, ST_Buffer('LINESTRING(18 20,110 110)::geometry, 6,
                        'endcap=flat join=bevel'), 210
        ),
        3, ST_Buffer('LINESTRING(10 48,118 48)::geometry, 7,
                    'endcap=flat'), 180
    ) AS rast,
    ST_Buffer('POINT(64 64)::geometry, 34) AS clipper
FROM canvas
), variants AS (
    SELECT 'original RGB' AS title, rast AS rendered FROM source
    UNION ALL
    SELECT 'band 1 cropped', ST_Clip(rast, 1, clipper, true) FROM source
    UNION ALL
    SELECT
        'band 1 only',
        ST_AddBand(
            ST_Clip(rast, 1, clipper, false),
            ARRAY[ST_Band(rast, 2), ST_Band(rast, 3)]
        )
    FROM source
    UNION ALL
    SELECT 'all bands clipped', ST_Clip(rast, clipper, false) FROM source
)
SELECT title, ST_AsPNG(rendered) AS image
FROM variants
ORDER BY CASE title
    WHEN 'original RGB' THEN 1
    WHEN 'band 1 cropped' THEN 2
    WHEN 'band 1 only' THEN 3
    ELSE 4
END;
```


Figure



Crop, single-band clipping, and all-band clipping.





相关信息

[ST_AddBand](#), [ST_Count](#), [ST_MapAlgebra \(callback function version\)](#), [ST_Intersection](#)

11.12.2 ST_ColorMap

`ST_ColorMap` — 根据源栅格和指定波段创建最多四个 8BUI 波段（灰度、RGB、RGBA）的新栅格。如果未指定，假定波段 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);
```

Description

将 `colormap` 用于 `rast` 的 `nband` 的波段，从而生成最多由四个 8BUI 波段组成的新栅格。新栅格中的 8BUI 波段数量由 `colormap` 中定义的色分量数量决定。

如果未指定 `nband`，假定波段 1。

`colormap` 可以是定义颜色的关键字或定义颜色和色分量的一行。

有效的定义 `colormap` 关键字：

- `grayscale` 或 `greyscale` 的一个 8BUI 波段栅格的灰度。
- `pseudocolor` 四 8BUI (RGBA) 波段栅格的彩色，颜色从蓝色到绿色再到红色。
- `fire` 四 8BUI (RGBA) 波段栅格，颜色从黑色到绿色到浅黄色。
- `bluered` 表示四 8BUI (RGBA) 波段栅格，颜色从蓝色到浅白色再到红色。

用户可以通过 `colormap` 参数指定一行条目（每行一个）来指定自定义的颜色映射。每个条目通常包括五个值：像素值以及相应的蓝色、绿色、红色、Alpha 组件（颜色组件的取值范围在 0 到 255 之间）。可以使用百分比代替像素值，其中 0% 和 100% 分别表示栅格中找到的最小值和最大值。可以使用逗号（','）、制表符、冒号（':'）和/或空格分隔。像素值可以设置 `nv`、`null` 或 `nodata`，表示 NODATA 值。以下是一个示例。

Code

```
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` 的方法与 GDAL `gdaldem` 的颜色浮雕模式类似。

`method` 的有效关键字：

- `INTERPOLATE` 使用线性插值来平滑地混合指定像素之间的颜色
- `EXACT` 栅格匹配颜色中找到的像素值。与颜色条目不匹配的像素将被设置 0 0 0 0 (RGBA)
- `NEAREST` 使用其最接近像素值的颜色条目



Note

[ColorBrewer](#) 是色彩的一个很好的参考。



Warning

新栅格的默认波段将没有设置 NODATA 值。如果需要，使用 `ST_SetBandNoDataValue` 设置 NODATA 值。

可用性：2.1.0

示例

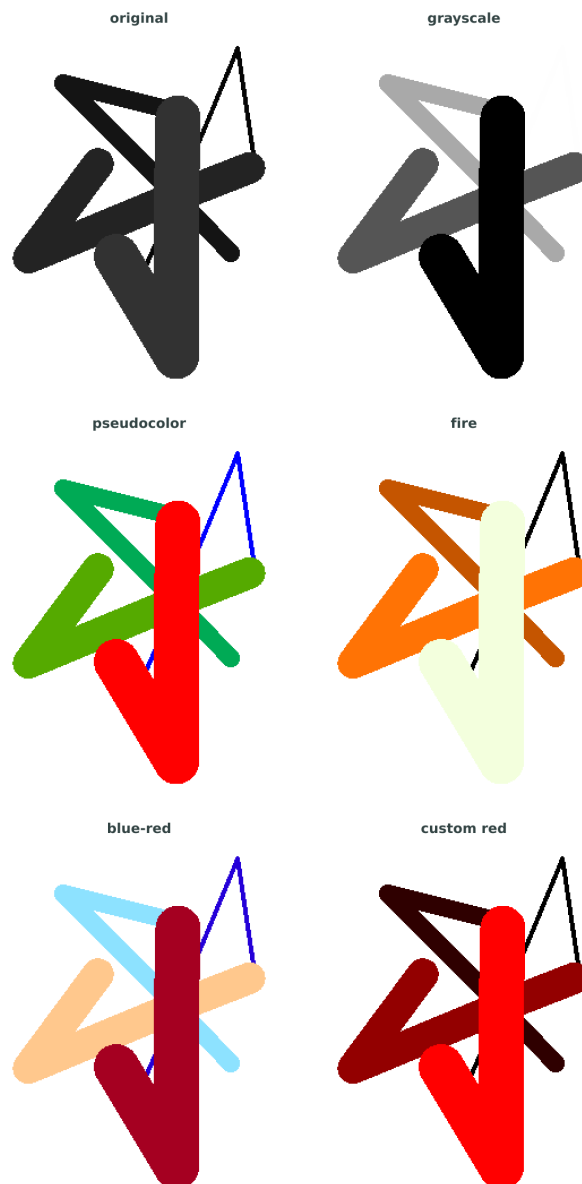
This example builds a raster from rotated buffered lines and compares the original values with pre-defined and custom color maps.

I Code

```

WITH ref AS (
  SELECT ST_MakeEmptyRaster(200, 200, 0, 200, 1, -1, 0, 0) AS rast
), source AS (
  SELECT ST_Union(rast) AS rast
  FROM (
    SELECT
      ST_AsRaster(ST_Rotate(ST_Buffer('LINESTRING(0 2,50 50,150 150,125 50)')::geometry,
        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 title, ST_AsPNG(rendered) AS image
FROM source
CROSS JOIN LATERAL (VALUES
  ('original', rast),
  ('grayscale', ST_ColorMap(rast, 1, 'grayscale')),
  ('pseudocolor', ST_ColorMap(rast, 1, 'pseudocolor')),
  ('fire', ST_ColorMap(rast, 1, 'fire')),
  ('blue-red', ST_ColorMap(rast, 1, 'bluered')),
  ('custom red', ST_ColorMap(
    rast,
    1,
    E'100% 255 0 0 255\n80% 160 0 0 255\n50% 130 0 0 255\n' ||
    E'30% 30 0 0 255\n20% 60 0 0 255\n0% 0 0 255\n' ||
    E'nv 255 255 255 0'
  ))
) AS variants(title, rendered);

```

Figure**相关信息**

[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 — 根据源栅格和代表色、色和色的指定波段新建新的 1-8BUI 波段栅格

Synopsis

- (1) raster **ST_Grayscale**(raster rast, integer redband=1, integer greenband=2, integer blueband=3, text extenttype=INTERSECTION);
- (2) raster **ST_Grayscale**(rastbandarg[] rastbandargset, text extenttype=INTERSECTION);

Description

给定三个输入波段（来自一个或多个栅格），构建一个具有 8BUI 波段的栅格。任何像素类型不是 8BUI 的输入波段都将使用 **ST_Reclass** 行重分区。



Note

此函数与 **ST_ColorMap** 不同，**ST_ColorMap** 有 grayscale 关键字，因此 **ST_ColorMap** 在一个波段上行，而此函数需要 RGB 的三个波段。此函数用以下公式将 RGB 转换为灰度： $0.2989 * RED + 0.5870 * GREEN + 0.1140 * BLUE$

可用性：2.5.0

示例

This self-contained example paints an apple with gradients and highlights into three raster bands. It compares the original RGB raster with the equivalent grayscale results produced by both function variants.

Code

```
WITH pixels AS (
  SELECT
    col,
    row,
    x,
    y,
    (((x - 48.0)^2 / (25.0^2) + (y - 54.0)^2 / (29.0^2) < 1)
    OR ((x - 72.0)^2 / (25.0^2) + (y - 54.0)^2 / (29.0^2) < 1))
    AND NOT ((x - 60.0)^2 / (8.0^2) + (y - 26.0)^2 / (5.0^2) < 1) AS fruit,
    ((x - 82.0) * 0.84 + (y - 28.0) * 0.54)^2 / (22.0^2)
    + (-(x - 82.0) * 0.54 + (y - 28.0) * 0.84)^2 / (8.0^2) < 1 AS leaf,
    abs((x - 62.0) * 0.96 - (y - 29.0) * 0.28) < 3
    AND x BETWEEN 60 AND 69 AND y BETWEEN 21 AND 45 AS stem
  FROM (
    SELECT
      col,
      row,
      col / 2.0 AS x,
      row / 2.0 AS y
    FROM generate_series(1, 180) AS row
    CROSS JOIN generate_series(1, 240) AS col
  ) AS source_pixels
), shaded AS (
  SELECT
    col,
    row,
    x,
    y,
    CASE
      WHEN stem THEN 95 + y * 0.7 + (x % 3) * 8
```

```

        WHEN leaf THEN 28 + (90 - x) * 0.5
        WHEN fruit THEN LEAST(255, GREATEST(0,
            150
            + (85 - sqrt((x - 39.0)^2 + (y - 37.0)^2)) * 1.1
            + CASE
                WHEN (x - 43.0)^2 / (10.0^2) + (y - 43.0)^2 / (7.0^2) < 1
                THEN 35
                ELSE 0
            END
            - CASE
                WHEN (x - 77.0)^2 / (20.0^2) + (y - 67.0)^2 / (13.0^2) < 1
                THEN 35
                ELSE 0
            END
            + ((x * 13 + y * 7) % 19 - 9)
        ))
    ELSE 245
END AS red,
CASE
    WHEN stem THEN 50 + y * 0.35
    WHEN leaf THEN LEAST(190, GREATEST(40,
        105 + (95 - x) * 1.2 + (y - 20) * 1.5
        + CASE
            WHEN (x - 76.0)^2 / (12.0^2) + (y - 24.0)^2 / (4.0^2) < 1
            THEN 45
            ELSE 0
        END
    ))
    WHEN fruit THEN LEAST(95, GREATEST(15,
        24 + (68 - y) * 0.35
        + CASE
            WHEN (x - 43.0)^2 / (10.0^2) + (y - 43.0)^2 / (7.0^2) < 1
            THEN 30
            ELSE 0
        END
        + ((x * 5 + y * 11) % 13 - 6)
    ))
    ELSE 245
END AS green,
CASE
    WHEN stem THEN 22 + (x % 4) * 3
    WHEN leaf THEN 42 + (100 - x) * 0.4
    WHEN fruit THEN LEAST(75, GREATEST(10,
        18
        + CASE
            WHEN (x - 43.0)^2 / (10.0^2) + (y - 43.0)^2 / (7.0^2) < 1
            THEN 25
            ELSE 0
        END
        + ((x * 3 + y * 5) % 11 - 5)
    ))
    ELSE 245
END AS blue
FROM pixels
), rows AS (
    SELECT
        row,
        array_agg(red::double precision ORDER BY col) AS red_values,
        array_agg(green::double precision ORDER BY col) AS green_values,
        array_agg(blue::double precision ORDER BY col) AS blue_values
    FROM shaded
    GROUP BY row

```

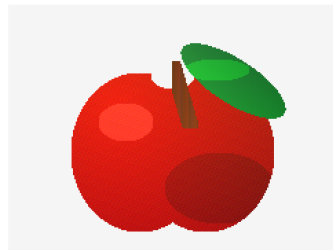
```

), source AS (
  SELECT ST_SetValues(ST_SetValues(ST_SetValues(
    ST_AddBand(ST_AddBand(ST_AddBand(
      ST_MakeEmptyRaster(240, 180, 0, 180, 1, -1, 0, 0, 0),
      '8BUI'::text, 245::double precision, 0::double precision),
      '8BUI'::text, 245::double precision, 0::double precision),
      '8BUI'::text, 245::double precision, 0::double precision),
    1, 1, 1, array_agg(red_values ORDER BY row)::double precision[][]),
    2, 1, 1, array_agg(green_values ORDER BY row)::double precision[][]),
    3, 1, 1, array_agg(blue_values ORDER BY row)::double precision[][])) AS rast
  FROM rows
)
SELECT ST_AsPNG(rast) AS "original RGB",
  ST_AsPNG(ST_Grayscale(rast)) AS "grayscale (variant 1)",
  ST_AsPNG(ST_Grayscale(ARRAY[
    ROW(rast, 1)::rastbandarg,
    ROW(rast, 2)::rastbandarg,
    ROW(rast, 3)::rastbandarg
  ]::rastbandarg[])) AS "grayscale (variant 2)"
FROM source;

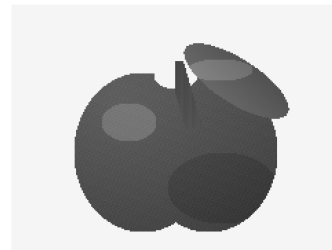
```

Figure

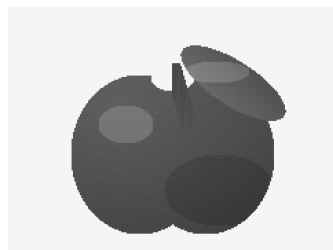
original RGB



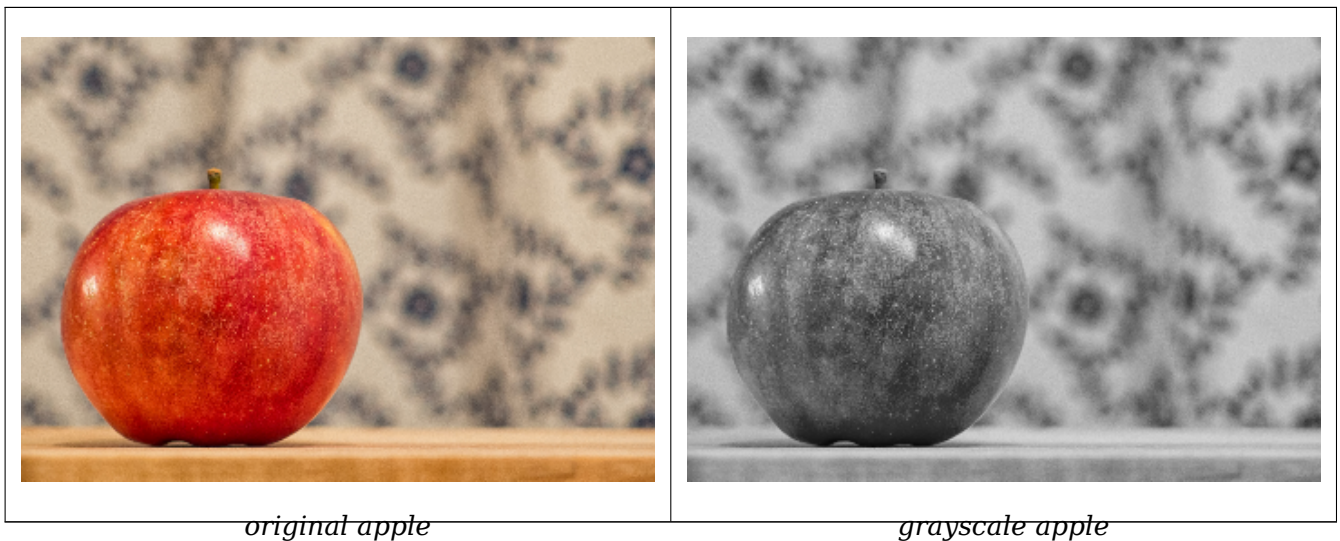
grayscale (variant 1)



grayscale (variant 2)



Original photographic illustration:



相关信息

[ST_AsPNG](#), [ST_Reclass](#), [ST_ColorMap](#)

11.12.4 ST_Intersection

ST_Intersection — 返回一个栅格或一个几何像素集合，表示两个栅格的共享部分或栅格矢量化和几何图形的几何交集。

Synopsis

```
setof geomval ST_Intersection(geometry geom, raster rast, integer band_num=1);
setof geomval ST_Intersection(raster rast, geometry geom);
setof geomval ST_Intersection(raster rast, integer band, geometry geom);
raster ST_Intersection(raster rast1, raster rast2, double precision[] nodataval);
raster ST_Intersection(raster rast1, raster rast2, text returnband, double precision[] nodataval);
raster ST_Intersection(raster rast1, integer band1, raster rast2, integer band2, double precision[] nodataval);
raster ST_Intersection(raster rast1, integer band1, raster rast2, integer band2, text returnband, double precision[] nodataval);
```

Description

返回一个栅格或一个几何像素集合，表示两个栅格的共享部分或栅格矢量化和几何图形的几何交集。

前三个变体返回一个 **geomval**，在向量空间中工作。首先将栅格矢量化（使用 [ST_DumpAsPolygons](#)）到一个 **geomval** 行，然后使用 [ST_Intersection \(geometry, Geometry\)](#) PostGIS 函数将某些行与几何相交。与栅格的无数据区域相交的几何将返回空几何。通常通过在 **WHERE** 子句中正确使用 [ST_Intersects](#) 将它从结果中排除。

您可以通过用括号将它括起来并在表式末尾添加 “.geom” 或 “.val” 来返回几何图形和生成的 **geomval** 集的部分。例如 `(ST_Intersection(rast, geom)).geom`

其他变体返回光栅，在光栅空间中工作。他们使用 `ST_MapAlgebraExpr` 的每个栅格版本来执行交集。

生成的栅格范围等于两个栅格范围的几何交集。生成的栅格包含“BAND1”、“BAND2”或“BOTH”波段，遵循作 `returnband` 参数指定的内容。任何波段中存在的无数据区域都会导致结果的每个波段中出无数据区域。语句，与无数据像素相交的任何像素都成为结果中的无数据像素。

由 `ST_Intersection` 生成的栅格必须不相交的区域分配 `nodata`。您可以通过提供一个包含一个或多个 `nodata` 的 `nodataval[]` 数组来定义或替换任何结果段的 `nodata`，具体取决于您是否请求“BAND1”、“BAND2”或“BOTH”段。数组中的第一个替换第一个中的无数据，第二个替换第二个中的无数据。如果一个输入没有定义 `nodata` 并且没有以数组形式提供，使用 `ST_MinPossibleValue` 函数指定一个。所有接受 `nodata` 数组的体也可以接受一个，将分配每个请求的。

在所有体中，如果未指定波段号，假定波段 1。如果您需要栅格和返回栅格的几何形状之的交集，参 [ST_Clip](#)。



Note For more control over the result extent or NODATA handling, use the two-raster variant of [ST_MapAlgebraExpr](#). To compute a raster-space intersection with a geometry, use [ST_Clip](#). Use this function together with [ST_Intersects](#) and an index on the raster or geometry column when filtering table data.

增：2.0.0 - 引入了光空中的交集。在 2.0.0 之前的早期版本中，支持在向量空中行交集。

示例：几何、光——生几何

Code

```
SELECT
  foo.rid,
  foo.gid,
  GeometryType((foo.geomval).geom) AS geom_type,
  (foo.geomval).val AS value
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, 'LINESTRING(3427927.85 5793243.75,3427927.8 5793243.75,3427927.8 5793243.8)'::geometry) ←
      (3, 'LINESTRING(1 2,3 4)'::geometry)
  ) As g(gid, geom)
  WHERE A.rid = 2
) AS foo;
```

Output

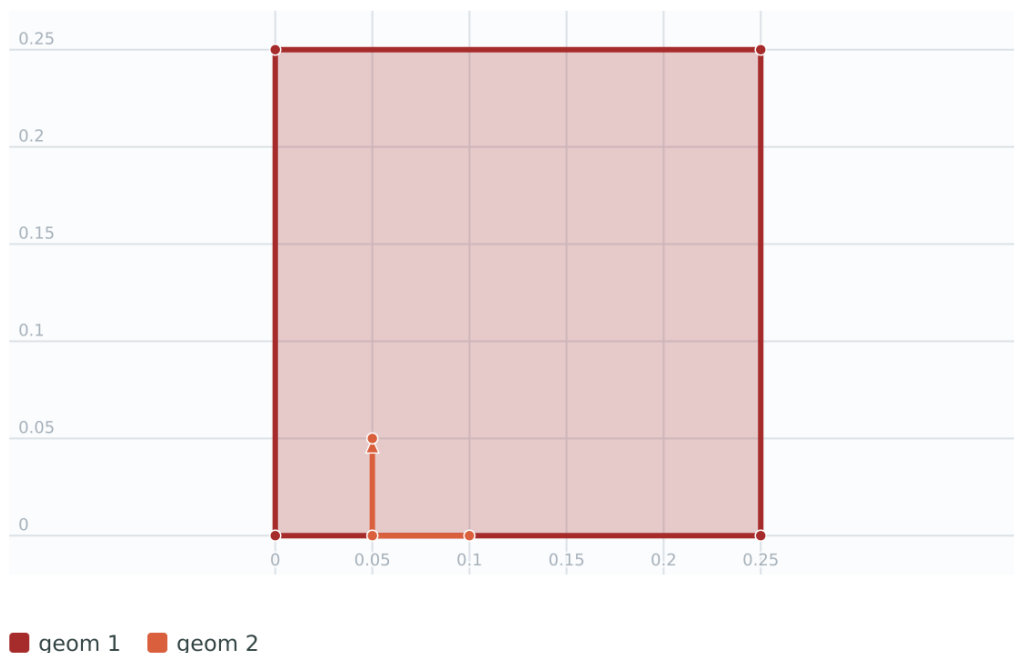
rid	gid	geom_type	value
2	1	POINT	249
2	1	POINT	253
2	2	POINT	254
2	2	POINT	251
2	2	POINT	253
2	2	LINESTRING	252
2	2	MULTILINESTRING	250
2	3	GEOMETRYCOLLECTION	

The same line example rendered in a common coordinate frame. The raster footprint and complete input line remain visible behind the returned intersection pieces.

I Code

```
WITH source AS (
    SELECT
        rast,
        ST_GeomFromText(
            'LINESTRING(3427927.85 5793243.75, '
            '3427927.8 5793243.75,3427927.8 5793243.8)'
        ) AS input_geometry
    FROM dummy_rast
    WHERE rid = 2
), intersection AS (
    SELECT
        rast,
        input_geometry,
        ST_Collect(gv.geom) AS result_geometry
    FROM source
    CROSS JOIN LATERAL ST_Intersection(rast, input_geometry) AS gv
    GROUP BY rast, input_geometry
), layers AS (
    SELECT
        'raster' AS title,
        ST_Translate(ST_Envelope(rast), -3427927.75, -5793243.75) AS geom
    FROM intersection
    UNION ALL
    SELECT
        'input',
        ST_Translate(input_geometry, -3427927.75, -5793243.75)
    FROM intersection
    UNION ALL
    SELECT
        'intersection',
        ST_Translate(result_geometry, -3427927.75, -5793243.75)
    FROM intersection
)
SELECT title, geom AS geom
FROM layers
ORDER BY CASE title
    WHEN 'raster' THEN 1
    WHEN 'input' THEN 2
    ELSE 3
END;
```

Figure



相关信息

[geomval](#), [ST_Intersects](#), [ST_MapAlgebraExpr](#), [ST_Clip](#), [ST_AsText](#)

11.12.5 ST_MapAlgebra (callback function version)

`ST_MapAlgebra (callback function version)` — 回调函数版本 - 给定一个或多个输入栅格、波段索引和一个用指定的回调函数，返回波段栅格。

Synopsis

```
raster ST_MapAlgebra(rastbandarg[] rastbandargset, regprocedure callbackfunc, text pixeltype=NULL,
text extenttype=INTERSECTION, raster customextent=NULL, integer distancex=0, integer distancey=0,
text[] VARIADIC userargs=NULL);
raster ST_MapAlgebra(raster rast, integer[] nband, regprocedure callbackfunc, text pixeltype=NULL,
text extenttype=FIRST, raster customextent=NULL, integer distancex=0, integer distancey=0, text[]
VARIADIC userargs=NULL);
raster ST_MapAlgebra(raster rast, integer nband, regprocedure callbackfunc, text pixeltype=NULL,
text extenttype=FIRST, raster customextent=NULL, integer distancex=0, integer distancey=0, text[]
VARIADIC userargs=NULL);
raster ST_MapAlgebra(raster rast1, integer nband1, raster rast2, integer nband2, regprocedure call-
backfunc, text pixeltype=NULL, text extenttype=INTERSECTION, raster customextent=NULL, inte-
ger distancex=0, integer distancey=0, text[] VARIADIC userargs=NULL);
raster ST_MapAlgebra(raster rast, integer nband, regprocedure callbackfunc, float8[] mask, boolean
weighted, text pixeltype=NULL, text extenttype=INTERSECTION, raster customextent=NULL, text[]
VARIADIC userargs=NULL);
```

Description

定义一个或多个输入栅格、波段索引和一个用 `callbackfunc` 指定的回调函数，返回波段栅格。

rast, rast1, rast2, rastbandargset 估计代数栅格的栅格。

rastbandargset 允许多栅格和/或多波段上使用代数运算。参见示例图 1。

nband, nband1, nband2 要估计的栅格的波段号。**nband** 可以是表示波段的整数或整数 []。对于 the 2 raster/2band 情况，**nband1** 是 **rast1** 上的波段，**nband2** 是 **rast2** 上的波段。

callbackfunc `callbackfunc` 参数必须是 SQL 或 PL/pgSQL 函数的名称和名称，参见 `regprocedure`。PL/pgSQL 函数示例如下：

Code

```
CREATE OR REPLACE FUNCTION sample_callbackfunc(value double precision[][][], pos ←
integer[][], VARIADIC userargs text[])
RETURNS double precision
AS $$
BEGIN
    RETURN 0;
END;
$$ LANGUAGE 'plpgsql' IMMUTABLE;
```

The `callbackfunc` must have two or three arguments: a 3-dimension double precision array, a 2-dimension integer array and, optionally, a variadic 1-dimension text array. The first argument `value` is the set of values (as double precision) from all input rasters. The three dimensions (where indexes are 1-based) are: raster #, row y, column x. The second argument `position` is the set of pixel positions from the output raster and input rasters. The outer dimension (where indexes are 0-based) is the raster #. The position at outer dimension index 0 is the output raster's pixel position. For each outer dimension, there are two elements in the inner dimension for X and Y. The optional third argument `userargs` is for passing through any user-specified arguments.

The callback return value is used as the value of the output pixel. If the callback returns `NULL`, the output pixel is set to `NODATA` when the output band has a `NODATA` value. If the output band has no `NODATA` value, a `NULL` callback result raises an error.

将 `regprocedure` 参数名称 SQL 函数需要完整的函数名称，然后参见 `regprocedure` 类型。要将上述示例 PL/pgSQL 函数作为参数，参数的 SQL 是：

Code

```
'sample_callbackfunc(double precision[], integer[], text[])'::regprocedure
```

Note that PostgreSQL array types do not encode dimensions in function signatures, so multidimensional callback arguments are written as `double precision[]` and `integer[]` in the `regprocedure` signature.

mask `n` 数字 (矩)，用于映射代数回调函数的元格。0 表示相应像素被无数据，1 表示相应像素有数据。如果 `weighted` 为 `true`，则这些将用作乘数，以乘以区域位置中的像素。

weighted 布 (true/false) 表示掩是否加权 (乘以原始) (适用于采用掩的原型)。

pixeltype 如果输入像素型，新栅格的一个波段将属于像素型。如果将像素型为 `NULL` 或省略，新栅格波段将具有与第一个栅格的指定波段 (范围类型: `INTERSECTION`、`UNION`、`FIRST`、`CUSTOM`) 或相应栅格的指定波段 (范围) 相同的像素型。类型: 第二个、最后一个)。如果有疑问，始终指定像素型。

输出栅格的像素型必须是 `ST_BandPixelType` 中列出的类型，或者省略或置 `NULL`。

extenttype 可能的 `INTERSECTION` (默认)、`UNION`、`FIRST` (一个栅格体的默认)、`SECOND`、`LAST`、`CUSTOM`。

customextent 如果 extenttype 是 CUSTOM, 必须提供 customextent 提供栅格。参身体 1 的示例 4。

distance_x x 方向上距参考元的距离（以像素位）。因此，果矩的度将 $2 * distance_x + 1$ 。如果未指定，参考元（0 的域）。

distance_y y 方向上距参考元的距离（以像素位）。果矩的高度将 $2 * distance_y + 1$ 。如果未指定，参考元格（0 的域）。

userargs If the callbackfunc declares a third argument, that argument is a variadic text array. All trailing text arguments are passed through to the specified callbackfunc, and are contained in the userargs argument.



Note

有关 VARIADIC 关键字的更多信息，参 PostgreSQL 文档和 [窗口函数 \(SQL\) 函数](#) 的“具有可变参数数量的 SQL 函数”部分。

体 1 接受 rastbandarg 数组，允在多栅格和/或多波段上使用代数运算。参示例体 1。

体 2 和 3 一个栅格的一个或多个波段行操作。参示例体 2 和 3。

体 4 个栅格行操作，每个栅格一个波段。参示例体 4。

可用性：2.2.0：能添加遮罩

可用性：2.1.0

示例

Compare an unmasked mean with neighbor-only and weighted-neighbor masks. The weighted mask multiplies neighboring values by 2 before computing the result.

Code

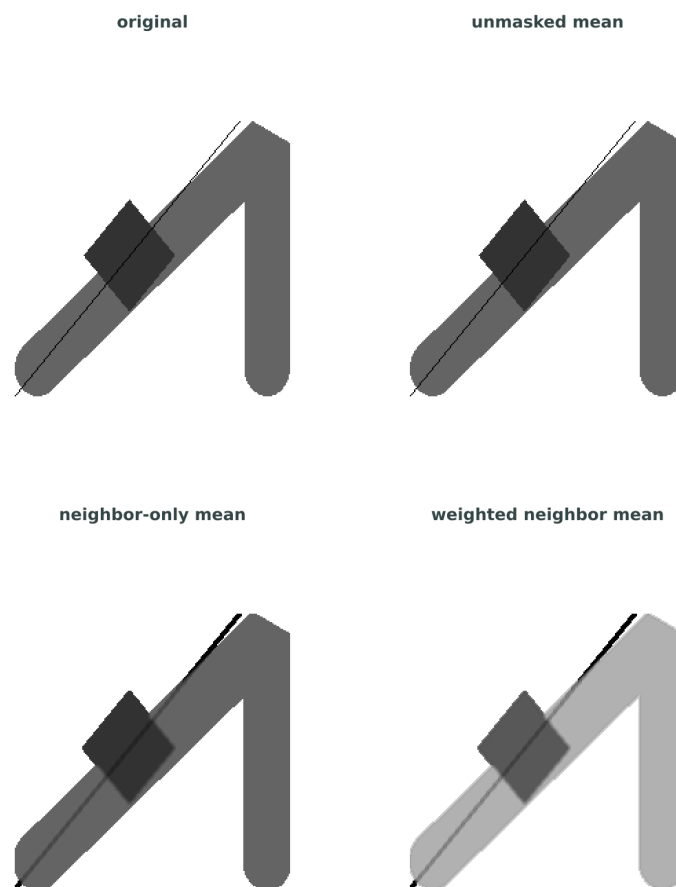
```
WITH foo AS (
  SELECT ST_SetBandNoDataValue(
    ST_SetValue(
      ST_SetValue(
        ST_AsRaster(
          ST_Buffer('LINESTRING(50 50,100 90,100 50)::geometry, 5, 'join=bevel') ↵
        ,
          200, 200, ARRAY['8BUI'], ARRAY[100], ARRAY[0]
        ),
        ST_Buffer('POINT(70 70)::geometry, 10, 'quad_segs=1'),
        50
      ),
      'LINESTRING(20 20,100 100,150 98)::geometry',
      1
    ),
    0
  ) AS rast
), variants AS (
  SELECT 'original' AS title, rast
  FROM foo
  UNION ALL
  SELECT 'unmasked mean' AS title,
    ST_MapAlgebra(rast, 1, 'ST_Mean4ma(double precision[], int[], text[]):: ↵
    regprocedure) AS rast
  FROM foo
  UNION ALL
```

```

SELECT 'neighbor-only mean' 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 'weighted neighbor mean' 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
)
SELECT title, ST_AsPNG(rast) AS image
FROM variants;

```

Figure




```

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

```

示例：与前一个图例域的覆盖 tile 示例图似，但只使用基础 SQL 特性来图。

Code

```

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
), 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[],
        'sample_callbackfunc(double precision[], int[], text[])'::regprocedure,
        '32BUI',
        'CUSTOM', t1.rast,
        1, 1
      ] AS rast
    FROM src t1
    JOIN foo t2
      ON t1.rid = t2.rid
  )
SELECT
  rid,
  (ST_MetaData(rast)),
  (ST_BandMetaData(rast, 1)),
  ST_Value(rast, 1, 1, 1)
FROM bar;

```

Variants 2 and 3.

一个☒格，多个波段

Code

```

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

```

一个☒格，一个波段

Code

```

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

```

Variant 4.

☐个☐格，☐个波段

Code

```
WITH foo AS (
  SELECT 1 AS rid, ST_AddBand(ST_AddBand(ST_AddBand(ST_MakeEmptyRaster(2, 2, 0, 0, 1, -1, ↵
    0, 0, 0), 1, '16BUI', 1, 0), 2, '8BUI', 10, 0), 3, '32BUI', 100, 0) AS rast UNION ALL
  SELECT 2 AS rid, ST_AddBand(ST_AddBand(ST_AddBand(ST_MakeEmptyRaster(2, 2, 0, 1, 1, -1, ↵
    0, 0, 0), 1, '16BUI', 2, 0), 2, '8BUI', 20, 0), 3, '32BUI', 300, 0) AS rast
)
SELECT
  ST_MapAlgebra(
    t1.rast,
    2,
    t2.rast,
    1,
    'sample_callbackfunc(double precision[], int[], text[])'::regprocedure
  ) AS rast
FROM foo t1
CROSS JOIN foo t2
WHERE t1.rid = 1
      AND t2.rid = 2
```

相关信息

[rastbandarg](#), [ST_Union](#), [ST_MapAlgebra \(expression version\)](#)

11.12.6 ST_MapAlgebra (expression version)

`ST_MapAlgebra (expression version)` — 表☐式版本 - 在☐定一☐个☐入☐格、波段索引和一个或多个用☐指定的 SQL 表☐式的情况下，返回☐波段☐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 extent-
type=INTERSECTION, text nodata1expr=NULL, text nodata2expr=NULL, double precision nodatan-
odataval=NULL);
```

Description

表☐式版本 - 在☐定一☐个☐入☐格、波段索引和一个或多个用☐指定的 SQL 表☐式的情况下，返回☐波段☐格。

可用性：2.1.0

描述：图体 1 和 2（一个图格）

通过图用由图入图格 (rast) 上的 **expression** 定义的有效 PostgreSQL 代数图算来图建新的图波段图格。如果未提供 **nband**，图假定图波段 1。新图格将具有与原始图格相同的地理参考、图度和高度，但只有一个波段。

如果图入像素图型，图新图格将具有图像素图型的波段。如果将像素图型图图 NULL，图新的图格图将具有与图入图格图相同的像素图型。

- 允许 **expression** 的关图字
 1. [rast] - 感兴趣像素的像素图
 2. [rast.val] - 感兴趣像素的像素图
 3. [rast.x] - 感兴趣像素的基于 1 的像素列
 4. [rast.y] - 感兴趣像素的从 1 开始的像素行

描述：图体 3 和 4（图个图格）

通过图图个图入图格波段 **rast1**, (**rast2**) 上的 **expression** 定义图个波段图用有效的 PostgreSQL 代数图算，图建一个新的图波段图格。如果没有 **band1**，图指定 **band2**，图假定图 **band 1**。生成的图格将在第一个图格定义图网格上图图（比例、图斜和像素角）。生成的图格将具有由 **extenttype** 参数定义图范围。

expression 涉及图个图格的 PostgreSQL 代数表图式和 PostgreSQL 定义图函数/图算符，用于定义图像素相交图像素图。例如 `(([rast1] [rast2])/2.0)::integer`

pixeltype 图出图格的图果像素图型。必图是 **ST_BandPixelType** 中列出图一图，可省略或图置图 NULL。如果未图入或图置图 NULL，将默图图第一个图格的像素图型。

extenttype 控制生成图图格的范围

1. **INTERSECTION** - 新图格的范围是图个图格的交集。图是默图图置。
2. **UNION** - 新图格的范围是图个图格的并集。
3. **FIRST** - 新图格的范围与第一个图格的范围相同。
4. **SECOND** - 新图格的范围与第二个图格的范围相同。

nodataexpr 图涉及 **rast2** 的代数表图式或定义图当 **rast1** 的像素图无数据图并且空图图图的 **rast2** 像素具有图图返回图内容的常量。

nodata2expr 图涉及 **rast1** 常量的代数表图式，图常量定义图当 **rast2** 的像素图无数据图且空图图图的 **rast1** 像素具有图图返回图内容。

nodatanodataval 当空图图图的 **rast1** 和 **rast2** 像素均图无数据图图要返回图数图常量。

- **expression**、**nodataexpr** 和 **nodata2expr** 中允许图使用的关图字
 1. [rast1] - rast1 中感兴趣像素的像素图
 2. [rast1.val] - rast1 中感兴趣像素的像素图
 3. [rast1.x] - 来自 rast1 的感兴趣像素的基于 1 的像素列
 4. [rast1.y] - 来自 rast1 的感兴趣像素的基于 1 的像素行
 5. [rast2] - rast2 中感兴趣像素的像素图
 6. [rast2.val] - rast2 中感兴趣像素的像素图
 7. [rast2.x] - 来自 rast2 的感兴趣像素的基于 1 的像素列
 8. [rast2.y] - 来自 rast2 的感兴趣像素的基于 1 的像素行

示例

Variants 1 and 2.

Code

```
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;
```

Variant 3 and 4.

Code

```
WITH foo AS (
  SELECT 1 AS rid, ST_AddBand(ST_AddBand(ST_AddBand(ST_MakeEmptyRaster(2, 2, 0, 0, 1, -1, 0, 0, 0), 1, '16BUI', 1, 0), 2, '8BUI', 10, 0), 3, '32BUI'::text, 100, 0) AS rast UNION ALL
  SELECT 2 AS rid, ST_AddBand(ST_AddBand(ST_AddBand(ST_MakeEmptyRaster(2, 2, 0, 0, 1, 1, -1, 0, 0), 1, '16BUI', 2, 0), 2, '8BUI', 20, 0), 3, '32BUI'::text, 300, 0) AS rast
)
SELECT
  ST_MapAlgebra(
    t1.rast,
    2,
    t2.rast,
    1,
    '([rast2] + [rast1.val]) / 2'
  ) AS rast
FROM foo t1
CROSS JOIN foo t2
WHERE t1.rid = 1
AND t2.rid = 2;
```

相关信息

[rastbandarg](#), [ST_Union](#), [ST_MapAlgebra \(callback function version\)](#)

11.12.7 ST_MapAlgebraExpr

ST_MapAlgebraExpr — 1 栅格波段版本：通过输入栅格波段和提供的像素类型用有效的 PostgreSQL 代数算来建新的波段栅格。如果未指定波段，假定波段 1。

Synopsis

raster **ST_MapAlgebraExpr**(raster rast, integer band, text pixeltype, text expression, double precision nodataval=NULL);

raster **ST_MapAlgebraExpr**(raster rast, text pixeltype, text expression, double precision nodataval=NULL);

Description



Warning
`ST_MapAlgebraExpr` 自 2.1.0 起已弃用。使用 `ST_MapAlgebra (expression version)` (表式版本) 代替。

通常由输入栅格 (`rast`) 上的 `expression` 的有效 PostgreSQL 代数算来构建新的波段栅格。如果未指定波段，假定波段 1。新栅格将具有与原始栅格相同的地理参考、度和高度，但只有一个波段。

如果输入像素型，新栅格将具有像素型的波段。如果将像素型置为 NULL，新的栅格将具有与输入栅格相同的像素型。

在表式中，您可以使用 `[rast]` 来引用原始波段的像素，`[rast.x]` 来引用基于 1 的像素列索引，`[rast.y]` 来引用基于 1 的像素行索引。像素行索引。

可用性: 2.0.0

示例

根据原始栅格构建一个新的 1 波段栅格，该栅格是原始栅格波段的模 2 的函数。

Code

```
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;
```

Output

origval	mapval
253	1
254	0
253	1
253	1
254	0
254	0
250	0
254	0
254	0

从重新分区的原始栅格中构建一个新的像素型 2BUI 的 1 波段栅格，并将 no data 置为 0。

Code

```
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;

```

Output

origval	mapval
249	1
250	1
251	1
252	2
253	3
254	3

Code

```

SELECT
    ST_BandPixelType(map_rast2) As b1pityp
FROM dummy_rast
WHERE rid = 2;

```

Output

2BUI

A self-contained source raster and the raster produced by classifying its values into four display levels.

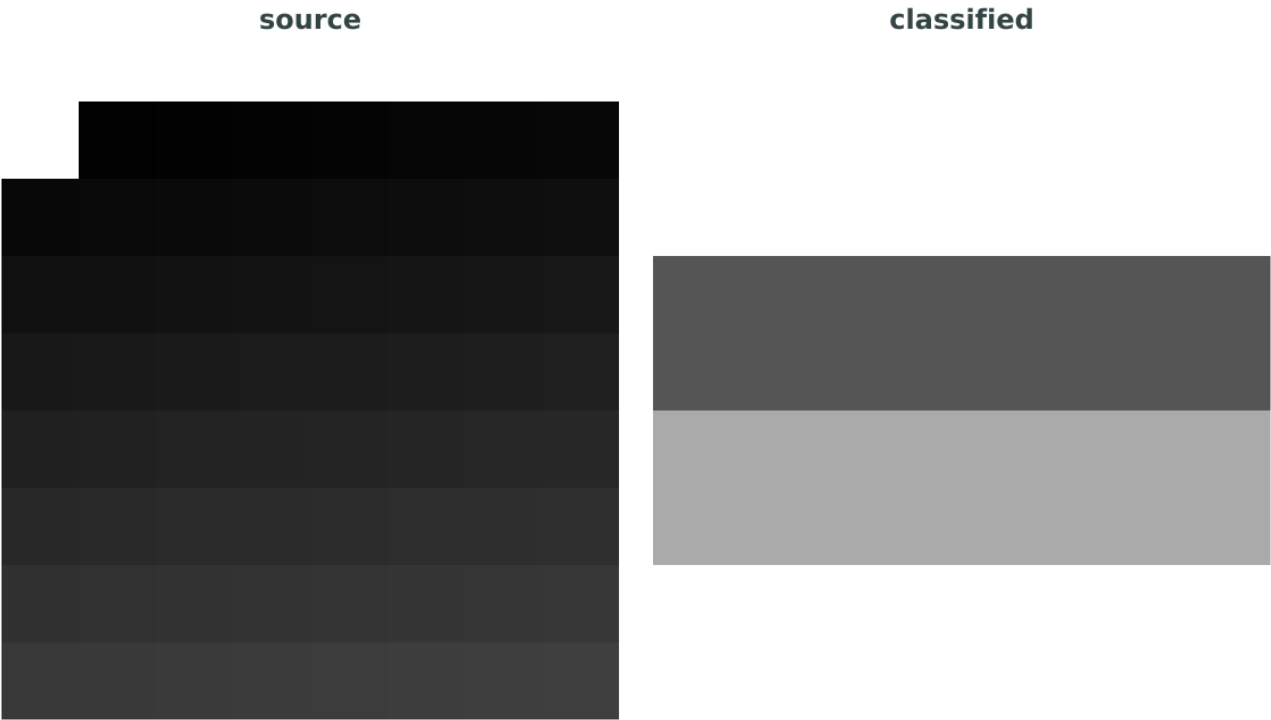
Code

```

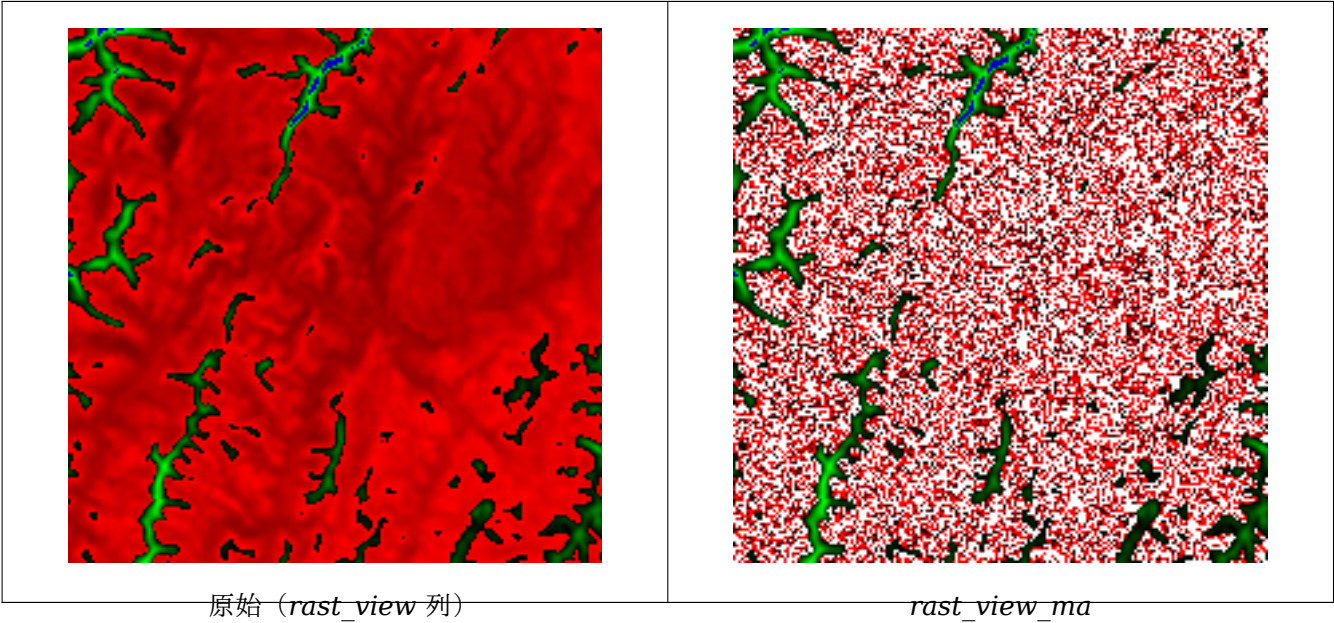
WITH source AS (
    SELECT ST_SetValues(
        ST_AddBand(
            ST_MakeEmptyRaster(8, 8, 0, 8, 1, -1, 0, 0, 0),
            1, '8BUI', 0, 0
        ),
        1, 1, 1,
        ARRAY[
            [0, 1, 2, 3, 4, 5, 6, 7],
            [8, 9, 10, 11, 12, 13, 14, 15],
            [16, 17, 18, 19, 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]
        ]::double precision[][])
    ) AS rast
), variants AS (
    SELECT 'source' AS title, rast AS rendered
    FROM source
    UNION ALL
    SELECT
        'classified',
        ST_MapAlgebraExpr(rast, '8BUI', 'floor([rast] / 16) * 85')
    FROM source
)
SELECT title, ST_AsPNG(rendered) AS image
FROM variants
ORDER BY title DESC;

```

Figure



Map algebra altering the first band of a three-band raster.



从原始 3 波段栅格构建一个与像素型相同的新 3 波段栅格，其中第一个波段由代数更改，其余 2 个波段保持不变。

I Code

```
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;
```

相关信息

[ST_MapAlgebraExpr](#), [ST_MapAlgebraFct](#), [ST_BandPixelType](#), [ST_GeoReference](#), [ST_Value](#)

11.12.8 ST_MapAlgebraExpr

ST_MapAlgebraExpr — 2 栅格波段版本：通过提供的两个输入栅格波段和像素类型使用有效的 PostgreSQL 代数算来构建新的波段栅格。如果未指定波段号，假定每个栅格的波段 1。生成的栅格将在第一个栅格定义的网格上（比例、斜和像素角），并具有由 “extenttype” 参数定义的范。 “extenttype” 的可以是：INTERSECTION、UNION、FIRST、SECOND。

Synopsis

raster **ST_MapAlgebraExpr**(raster rast1, raster rast2, text expression, text pixeltype=same_as_rast1_band, text extenttype=INTERSECTION, text nodata1expr=NULL, text nodata2expr=NULL, double precision nodatanodataval=NULL);

raster **ST_MapAlgebraExpr**(raster rast1, integer band1, raster rast2, integer band2, text expression, text pixeltype=same_as_rast1_band, text extenttype=INTERSECTION, text nodata1expr=NULL, text nodata2expr=NULL, double precision nodatanodataval=NULL);

Description**Warning**

ST_MapAlgebraExpr 自 2.1.0 起已弃用。使用 **ST_MapAlgebra (expression version)** (表式版本) 代替。

通过两个输入栅格波段 **rast1**, (**rast2**) 上的 **expression** 定义的两个波段使用有效的 PostgreSQL 代数算，构建一个新的波段栅格。如果没有 **band1**，指定 **band2**，假定 **band 1**。生成的栅格将在第一个栅格定义的网格上（比例、斜和像素角）。生成的栅格将具有由 **extenttype** 参数定义的范。

expression 涉及两个栅格的 PostgreSQL 代数表达式和 PostgreSQL 定义的函数/运算符，用于定义像素相交的像素。例如 `(([rast1] [rast2])/2.0)::integer`

pixeltype 输出栅格的像素类型。必须是 **ST_BandPixelType** 中列出的一项，可省略或置 NULL。如果未输入或置 NULL，将默认为第一个栅格的像素类型。

extenttype 控制生成的栅格的范

1. **INTERSECTION** - 新栅格的范围是两个栅格的交集。空是默认设置。
2. **UNION** - 新栅格的范围是两个栅格的并集。
3. **FIRST** - 新栅格的范围与第一个栅格的范围相同。
4. **SECOND** - 新栅格的范围与第二个栅格的范围相同。

nodata1expr 涉及 **rast2** 的代数表达式或定义当 **rast1** 的像素无数据并且空栅格的 **rast2** 像素具有返回的内容的常量。

nodata2expr 涉及 **rast1** 常量的代数表达式，常量定义当 **rast2** 的像素无数据且空栅格的 **rast1** 像素具有返回的内容。

nodatanodataval 当空栅格的 **rast1** 和 **rast2** 像素均无数据时要返回的数值常量。

如果输入 **pixeltype**，新栅格将具有像素类型的波段。如果将像素类型设置为 NULL 或未指定像素类型，新的栅格波段将具有与输入 **rast1** 波段相同的像素类型。

使用 **[rast1.val]** **[rast2.val]** 指代原始栅格波段的像素，使用 **[rast1.x]**、**[rast1.y]** 等指代像素的列/行位置。

可用性: 2.0.0

示例

2 Band Intersection and Union.

根据原始栅格构建一个新的 1 波段栅格，栅格是原始栅格波段的模 2 的函数。

Create a cool set of rasters.

Insert some cool shapes around Boston in Massachusetts state plane meters, then map their intersection and union.

Code

```
DROP TABLE IF EXISTS fun_shapes;
CREATE TABLE fun_shapes(rid serial PRIMARY KEY, fun_name text, rast raster);

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;

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

```

The same intersection and union operations using two deterministic, self-contained rasters.

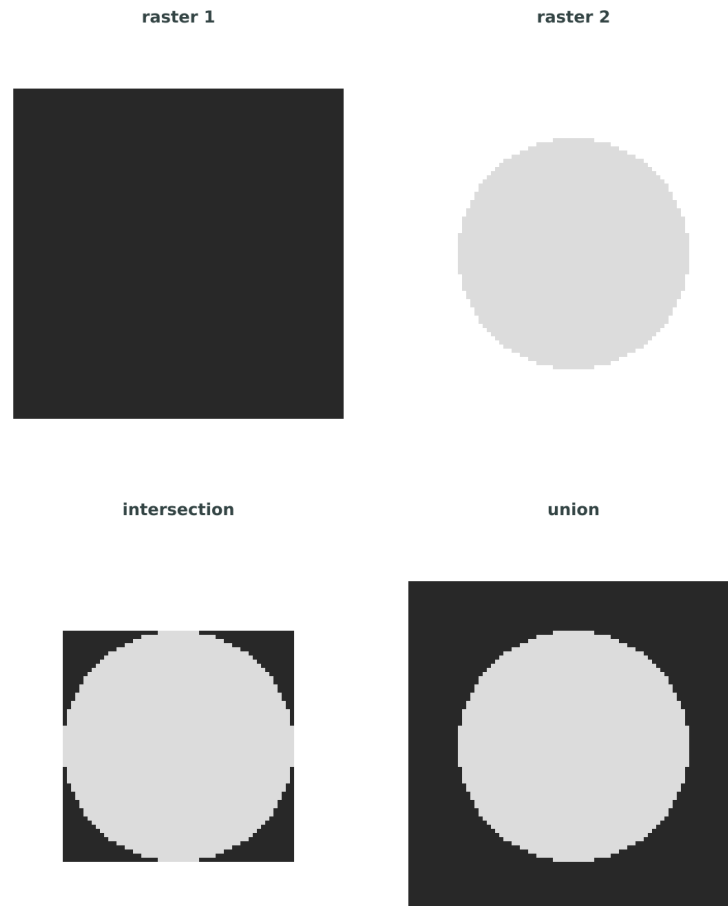
Code

```

WITH canvas AS (
  SELECT ST_AddBand(
    ST_MakeEmptyRaster(80, 80, 0, 80, 1, -1, 0, 0, 0),
    1, '8BUI', 40, 0
  ) AS rast
), inputs AS (
  SELECT
    rast AS raster_1,
    ST_AsRaster(
      ST_Buffer(ST_Point(40, 40), 28),
      rast,
      '8BUI', 220, 0
    ) AS raster_2
  FROM canvas
), variants AS (
  SELECT 'raster 1' AS title, raster_1 AS rendered FROM inputs
  UNION ALL
  SELECT 'raster 2', raster_2 FROM inputs
  UNION ALL
  SELECT 'intersection', ST_MapAlgebraExpr(
    raster_1, raster_2,
    '[rast2.val]', '8BUI', 'INTERSECTION',
    '[rast2.val]', '[rast1.val]'
  ) FROM inputs
  UNION ALL
  SELECT 'union', ST_MapAlgebraExpr(
    raster_1, raster_2,
    '[rast2.val]', '8BUI', 'UNION',
    '[rast2.val]', '[rast1.val]'
  ) FROM inputs
)
SELECT title, ST_AsPNG(rendered) AS image
FROM variants
ORDER BY CASE title
  WHEN 'raster 1' THEN 1
  WHEN 'raster 2' THEN 2
  WHEN 'intersection' THEN 3
  ELSE 4
END;

```

Figure



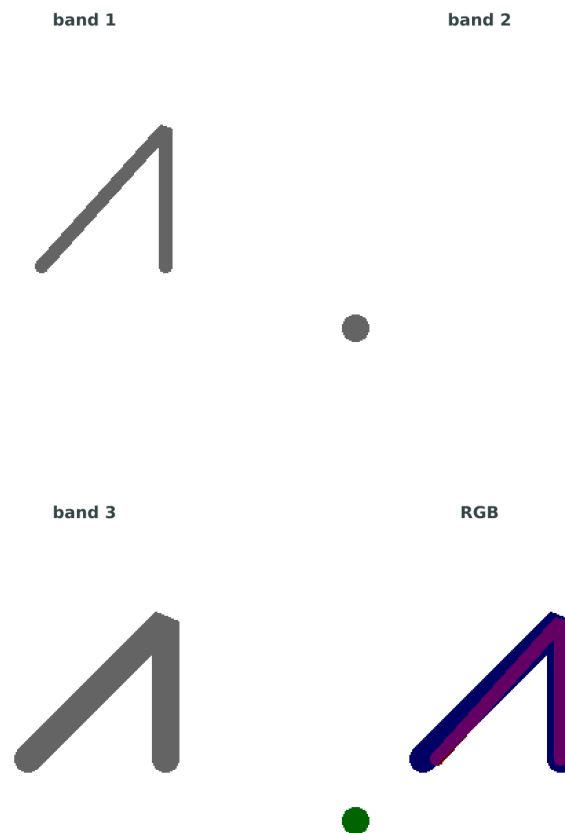
Overlaying rasters on a one-unit-per-pixel canvas as separate bands. We use `ST_AsPNG` to render the image so all single band ones look grey.

Code

```
WITH mygeoms AS (
  SELECT 2 AS bnum, ST_Buffer(ST_Point(1, 5), 10) AS geom
  UNION ALL
  SELECT 3,
    ST_Buffer(
      ST_GeomFromText('LINESTRING(50 50,150 150,150 50)'),
      10,
      'join=bevel'
    )
  UNION ALL
  SELECT 1,
    ST_Buffer(
      ST_GeomFromText('LINESTRING(60 50,150 150,150 50)'),
      5,
      'join=bevel'
    )
), 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  
    FROM mygeoms  
    ) AS bounds  
, 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]'  
        )  
        FROM mygeoms AS m  
        CROSS JOIN canvas  
        ORDER BY m.bnum  
    ) AS rasts  
)  
SELECT title, ST_AsPNG(rendered) AS image  
FROM rbands  
CROSS JOIN LATERAL (VALUES  
    ('band 1', rasts[1]),  
    ('band 2', rasts[2]),  
    ('band 3', rasts[3]),  
    ('RGB', ST_AddBand(ST_AddBand(rasts[1], rasts[2]), rasts[3]))  
) AS variants(title, rendered);
```

Figure



Overlay a two-meter boundary of selected parcels over aerial imagery. The query first unions the parcel geometries, clips the source raster shards to the region, unions those smaller shards, and finally adds the first two bands plus a third-band map algebra overlay.

Create a new three-band raster composed of the first two clipped bands and the third band overlaid with the geometry. This query took 3.6 seconds on a 64-bit PostGIS Windows installation.

Code

```
WITH pr AS (
  SELECT ST_Clip(rast, ST_Expand(geom, 50)) AS rast, g.geom
  FROM aerals.o_2_boston AS r
  INNER JOIN (
    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))
), 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
)
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;

```

Parcel-boundary overlay.



The blue lines are the boundaries of selected parcels

The generated four-panel canvas example above shows the same band-overlay technique without requiring an external aerial-imagery or parcel table.

相关信息

[ST_MapAlgebraExpr](#), [ST_AddBand](#), [ST_AsPNG](#), [ST_AsRaster](#), [ST_MapAlgebraFct](#), [ST_BandPixelType](#), [ST_GeoReference](#), [ST_Value](#), [ST_Union](#), [ST_Union](#)

11.12.9 ST_MapAlgebraFct

ST_MapAlgebraFct — 1 波段版本 - 通过在输入栅格波段和提供的像素类型上应用有效的 PostgreSQL 函数来构建新的波段栅格。如果未指定波段，假定波段 1。

Synopsis

```
raster ST_MapAlgebraFct(raster rast, regprocedure onerasteruserfunc);
raster ST_MapAlgebraFct(raster rast, regprocedure onerasteruserfunc, text[] VARIADIC args);
raster ST_MapAlgebraFct(raster rast, text pixelftype, regprocedure onerasteruserfunc);
raster ST_MapAlgebraFct(raster rast, text pixelftype, 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 pixelftype, regprocedure onerasteruserfunc);
raster ST_MapAlgebraFct(raster rast, integer band, text pixelftype, regprocedure onerasteruserfunc, text[] VARIADIC args);
```

Description



Warning

ST_MapAlgebraFct 自 2.1.0 起已弃用。使用 **ST_MapAlgebra (callback function version)** (回调函数版本) 代替。

构建一个新的波段栅格，该栅格是通过在输入栅格 (**rast**) 上应用 **onerasteruserfunc** 指定的有效 PostgreSQL 函数而形成的。如果未指定波段，假定波段 1。新栅格将具有与原始栅格相同的地理参考、精度和高度，但只有一个波段。

如果输入像素类型，新栅格将具有该像素类型的波段。如果将像素类型设置为 NULL，新的栅格将具有与输入栅格相同的像素类型。

onerasteruserfunc 参数必须是 SQL 或 PL/pgSQL 函数的名称和别名，或者是 **regprocedure**。一个非常冗长且无用的 PL/pgSQL 函数示例是：

Code

```
CREATE OR REPLACE FUNCTION simple_function(pixel FLOAT, pos INTEGER[], VARIADIC args TEXT ↔
[])
RETURNS FLOAT
AS $$ BEGIN
    RETURN 0.0;
END; $$
LANGUAGE 'plpgsql' IMMUTABLE;
```

userfunction 可以接受一个或三个参数：一个浮点数、一个可选的整数数组和一个可选的文本数组。第一个参数是栅格像元的值（无论栅格数据类型如何）。第二个参数是当前像元的位置，格式为 “{x,y}”。第三个参数指示 **ST_MapAlgebraFct** 的所有剩余参数传递给 **userfunction**。

将 **regprocedure** 参数替换为 SQL 函数需要完整的函数名，然后替换为 **regprocedure** 类型。要将上述示例 PL/pgSQL 函数作为参数，参数的 SQL 为：

Code

```
'simple_function(float,integer[],text[])'::regprocedure
```

注意，参数包含函数名称、函数参数类型、名称和参数类型周围的引号以及 `regprocedure` 的限制符。
`userfunction` 的第三个参数是一个可写文本数。任何 `ST_MapAlgebraFct` 用的所有尾随文本参数都会写到指定的 `userfunction`，并包含在 `args` 参数中。



Note
有关 VARIADIC 关键字的更多信息，参 PostgreSQL 文档和 [SQL 函数](#) 的“具有可变参数数量的 SQL 函数”部分。



Note
无论您是否将任何参数写入 `userfunction` 行，`text[]` 参数在 `userfunction` 都是必需的。

可用性: 2.0.0

示例

根据原始栅格建一个新的 1 波段栅格，该栅格是原始栅格波段的模 2 的函数。

Code

```
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;
```

Output

origval	mapval
253	1
254	0
253	1
253	1
254	0
254	0
250	0
254	0
254	0

从我的原始栅格中建一个新的像素类型 2BUI 的 1 波段栅格，并将其重新分，并将 `nodata` 置置用函数的参数 (0)。

Code

```
ALTER TABLE dummy_rast ADD COLUMN map_rast2 raster;
CREATE FUNCTION classify_fct(pixel float, pos integer[], variadic args text[])
RETURNS float
AS
$$
DECLARE
    nodata float := 0;
BEGIN
    IF NOT args[1] IS NULL THEN
        nodata := args[1];
    END IF;
    IF pixel < 251 THEN
        RETURN 1;
    ELSIF pixel = 252 THEN
        RETURN 2;
    ELSIF pixel
> 252 THEN
        RETURN 3;
    ELSE
        RETURN nodata;
    END IF;
END;
$$
LANGUAGE 'plpgsql';
UPDATE dummy_rast SET map_rast2 = ST_MapAlgebraFct(rast,'2BUI','classify_fct(float,integer ↵
[],text[])'::regprocedure, '0') WHERE rid = 2;

SELECT DISTINCT ST_Value(rast,1,i,j) As origval, ST_Value(map_rast2, 1, i, j) As mapval
FROM dummy_rast CROSS JOIN generate_series(1, 5) AS i CROSS JOIN generate_series(1,5) AS j
WHERE rid = 2;
```

Output

origval	mapval
249	1
250	1
251	1
252	2
253	3
254	3

Code

```
SELECT ST_BandPixelType(map_rast2) As b1pixtyp
FROM dummy_rast WHERE rid = 2;
```

Output

b1pixtyp
2BUI

从原始 3 波段栅格建立一个与像素型相同的新 3 波段栅格，其中第一个波段由地代数更改，其余 2 个波段保持不。

Code

```
CREATE FUNCTION rast_plus_tan(pixel float, pos integer[], variadic args text[])
RETURNS float
AS
$$
```

```

BEGIN
    RETURN tan(pixel) * pixel;
END;
$$
LANGUAGE 'plpgsql';

SELECT ST_AddBand(ST_AddBand(ST_AddBand(ST_MakeEmptyRaster(rast_view),
    ST_MapAlgebraFct(rast_view, 1, NULL, 'rast_plus_tan(float,integer[],text[])':: ↵
    regprocedure)
    ),
    ST_Band(rast_view, 2)
    ),
    ST_Band(rast_view, 3) AS rast_view_ma
)
FROM wind
WHERE rid=167;

```

The deprecated callback form above is equivalent to a modern **ST_MapAlgebra (callback function version)** expression for simple per-pixel transforms. This build-time example renders the original raster and a modulo-2 classification.

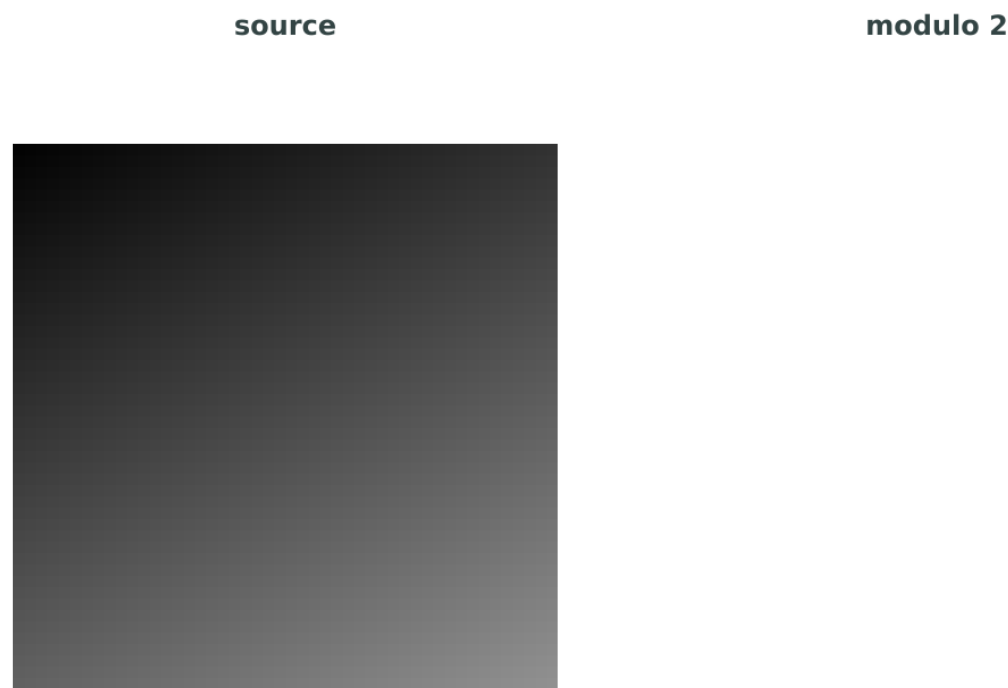
Code

```

WITH pixels AS (
    SELECT x, y, (x + y * 2)::double precision AS value
    FROM generate_series(1, 48) AS x
    CROSS JOIN generate_series(1, 48) AS y
), rows AS (
    SELECT y, array_agg(value ORDER BY x) AS values
    FROM pixels
    GROUP BY y
), source AS (
    SELECT ST_SetValues(
        ST_AddBand(
            ST_MakeEmptyRaster(48, 48, 0, 48, 1, -1, 0, 0, 0),
            1, '8BUI', 0, 0
        ),
        1, 1, 1, array_agg(values ORDER BY y)::double precision[][])
    AS rast
    FROM rows
), variants AS (
    SELECT 'source' AS title, rast AS rendered FROM source
    UNION ALL
    SELECT 'modulo 2', ST_MapAlgebra(
        rast, 1,
        '8BUI',
        'mod([rast]::integer,2) * 255'
    )
    FROM source
)
SELECT title, ST_AsPNG(rendered) AS image
FROM variants
ORDER BY title DESC;

```

| **Figure**



相关信息

[ST_MapAlgebraExpr](#), [ST_BandPixelType](#), [ST_GeoReference](#), [ST_SetValue](#)

11.12.10 ST_MapAlgebraFct

ST_MapAlgebraFct — 2 波段版本 - 通过在提供的 2 个输入栅格波段和像素类型上应用有效的 PostgreSQL 函数来创建新的波段栅格。如果未指定波段，假定波段 1。如果未指定，范围类型默认为 INTERSECTION。

Synopsis

```
raster ST_MapAlgebraFct(raster rast1, raster rast2, regprocedure tworastuserfunc, text pixeltype=same_as_rast1, text extenttype=INTERSECTION, text[] VARIADIC userargs);
raster ST_MapAlgebraFct(raster rast1, integer band1, raster rast2, integer band2, regprocedure tworastuserfunc, text pixeltype=same_as_rast1, text extenttype=INTERSECTION, text[] VARIADIC userargs);
```

Description



Warning

ST_MapAlgebraFct 自 2.1.0 起已弃用。使用 **ST_MapAlgebra (callback function version)** (回调函数版本) 代替。

创建一个新波段栅格，该栅格是通入栅格 `rast1`、`rast2` 由 `tworastuserfunc` 指定的有效 PostgreSQL 函数而形成的。如果未指定 `band1` 或 `band2`，假定 `band 1`。新栅格将具有与原始栅格相同的地理参考、经度和高度，但只有一个波段。

如果传入 `pixeltype`，新栅格将具有像素类型的波段。如果将像素类型设为 `NULL` 或省略，新的栅格波段将具有与传入 `rast1` 波段相同的像素类型。

`Tworastuserfunc` 参数必须是 SQL 或 PL/pgSQL 函数的名称和别名，或 `regprocedure`。PL/pgSQL 函数示例如下：

Code

```
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;
```

`Tworastuserfunc` 可以接受三个或四个参数：一个双精度浮点、一个双精度浮点、一个可变的整数数组和一个可变的文本数组。第一个参数是 `rast1` 中的栅格像元的值（无论栅格数据类型如何）。第二个参数是 `rast2` 中的栅格像元值。第三个参数是当前像元的位置，格式为 `{x,y}`。第四个参数指示 **ST_MapAlgebraFct** 的所有剩余参数均传递给 `tworastuserfunc`。

将 `regprocedure` 参数传递给 SQL 函数需要完整的函数名，然后指定 `regprocedure` 类型。要将上述示例 PL/pgSQL 函数作为参数传递，参数的 SQL 为：

Code

```
'simple_function(double precision, double precision, integer[], text[])'::regprocedure
```

注意，参数包含函数名称、函数参数类型、名称和参数类型周围的引号以及 `regprocedure` 的限制符。

`Tworastuserfunc` 的第四个参数是可变的文本数组。任何 **ST_MapAlgebraFct** 使用的所有尾随文本参数都会传递到指定的 `tworastuserfunc`，并包含在 `userargs` 参数中。



Note

有关 `VARIADIC` 关键字的更多信息，请参阅 PostgreSQL 文档和 [方言 \(SQL\) 函数](#) 的“具有可变参数数量的 SQL 函数”部分。



Note

无论您是否将任何参数传递给函数行处理，`text[]` 参数在 `tworastuserfunc` 都是必需的。

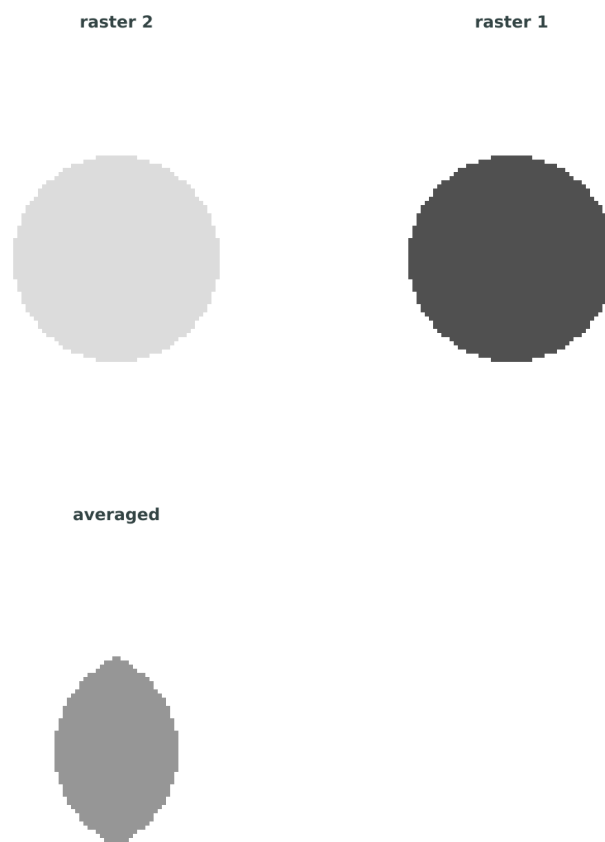
可用性: 2.0.0

示例：将☐格作☐☐独的波段覆盖在画布上

The deprecated two-raster callback form can often be expressed with the modern two-raster **ST_MapAlgebra(callback function version)** expression variant. This build-time example renders two input rasters and their averaged overlap.

Code

```
WITH canvas AS (
  SELECT ST_AddBand(
    ST_MakeEmptyRaster(80, 80, 0, 80, 1, -1, 0, 0, 0),
    1, '8BUI', 0, 0
  ) AS rast
), inputs AS (
  SELECT
    ST_AsRaster(ST_Buffer(ST_Point(30, 40), 25), rast, '8BUI', 80, 0)
    AS raster_1,
    ST_AsRaster(ST_Buffer(ST_Point(50, 40), 25), rast, '8BUI', 220, 0)
    AS raster_2
  FROM canvas
), variants AS (
  SELECT 'raster 1' AS title, raster_1 AS rendered FROM inputs
  UNION ALL
  SELECT 'raster 2', raster_2 FROM inputs
  UNION ALL
  SELECT 'averaged', ST_MapAlgebra(
    raster_1, 1,
    raster_2, 1,
    '([rast1] + [rast2]) / 2',
    '8BUI',
    'UNION'
  ) FROM inputs
)
SELECT title, ST_AsPNG(rendered) AS image
FROM variants
ORDER BY title DESC;
```

Figure

Define our user defined function.

Code

```
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  
END  
$*
```

```

        END CASE;

    RETURN NULL;
END;
$$ LANGUAGE 'plpgsql' IMMUTABLE COST 1000;

```

Prepare a test table of rasters.

Code

```

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 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 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 the result with **ST_AddBand**.

Code

```

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;
```

A compact, executable callback example. The source rasters and their averaged overlap are rendered at build time.

Code

```
CREATE OR REPLACE FUNCTION pg_temp.raster_average(
    value_1 double precision,
    value_2 double precision,
    position integer[],
    VARIADIC userargs text[]
) RETURNS double precision
LANGUAGE SQL IMMUTABLE
AS 'SELECT COALESCE(($1 + $2) / 2.0, $1, $2)';

WITH canvas AS (
    SELECT ST_AddBand(
        ST_MakeEmptyRaster(80, 80, 0, 80, 1, -1, 0, 0, 0),
        1, '8BUI', 0, 0
    ) AS rast
), inputs AS (
    SELECT
        ST_AsRaster(ST_Buffer(ST_Point(30, 40), 25), rast, '8BUI', 80, 0)
        AS raster_1,
        ST_AsRaster(ST_Buffer(ST_Point(50, 40), 25), rast, '8BUI', 220, 0)
        AS raster_2
    FROM canvas
), variants AS (
    SELECT 'raster 1' AS title, raster_1 AS rendered FROM inputs
    UNION ALL
    SELECT 'raster 2', raster_2 FROM inputs
    UNION ALL
    SELECT 'averaged', ST_MapAlgebraFct(
        raster_1,
        raster_2,
        'pg_temp.raster_average(double precision,double precision,integer[],text[])':: ↵
        regprocedure,
        '8BUI',
        'UNION'
    ) FROM inputs
)
SELECT title, ST_AsPNG(rendered) AS image
FROM variants
ORDER BY title DESC;
```

需要外参数的用定函数

Code

```
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)';

```

The generated callback figure above demonstrates the raster alignment and overlap used by this variant; the exact callback result remains available as text.

相关信息

[ST_MapAlgebraExpr](#), [ST_BandPixelType](#), [ST_GeoReference](#), [ST_SetValue](#)

11.12.11 ST_MapAlgebraFctNgb

ST_MapAlgebraFctNgb — 1-波段版本：使用用固定格的 PostgreSQL 函数映射代数最近格。返回一个格，其格是涉及输入格波段的区域的 PLPGSQL 用函数的结果。

Synopsis

raster **ST_MapAlgebraFctNgb**(raster rast, integer band, text pixeltype, integer ngbwidth, integer ngbheight, regprocedure onerastngbuserfunc, text nodatamode, text[] VARIADIC args);

Description



Warning

ST_MapAlgebraFctNgb 自 2.1.0 起已弃用。使用 **ST_MapAlgebra (callback function version)** (回调函数版本) 代替。

(一个栅格版本) 返回一个栅格，其是涉及输入栅格波段的栅格的 PLPGSQL 用函数的结果。用函数将像素的栅格作数字数，对于每个像素，返回用函数的结果，用函数结果替当前栅格的像素。

rast 用栅格函数的栅格。

band 要用的栅格的波段号。默认为 1。

pixeltype 出栅格的结果像素型。必是 **ST_BandPixelType** 中列出的一，或者省略或置 NULL。如果未入或置 NULL，将默认为 rast 的像素型。如果结果大于像素型允许的，会被截断。

ngbwidth 栅格的宽度（以元格位）。

ngbheight 栅格的高度（以元格位）。

onerastngbuserfunc PLPGSQL/psql 用函数用于栅格个波段的栅格像素。第一个元素是表示矩形像素域的二维数字数

nodatamode 定义要无数据或 NULL 的栅格像素函数的

‘ignore’：计算忽略在域中遇到的任何 NODATA - 标志必送到用回函数，并且用函数决定如何忽略它。

“NULL”：在域中遇到的任何 NODATA 都将致结果像素 NULL——在种情况下将跳用回函数。

“value”：在域中遇到的任何 NODATA 都将替参考像素（域中心的像素）。注意，如果 NODATA，行与“NULL”相同（于受影响的域）

args 要用的函数的参数。

可用性: 2.0.0

示例

示例利用 <https://gdal.org/user/drivers/raster/postgisraster.html> 中描述的作个加的卡特里娜栅格，然后在 **ST_Rescale** 示例中准

A simple ‘callback’ user function that averages up all the values in a neighborhood.

Code

```
CREATE OR REPLACE FUNCTION rast_avg(matrix float[][] , nodatamode text, variadic args text ←
[])
RETURNS float AS
$$
DECLARE
    _matrix float[][];
    x1 integer;
    x2 integer;
    y1 integer;
    y2 integer;
    sum float;
BEGIN
    _matrix := matrix;
    sum := 0;
    FOR x in array_lower(matrix, 1)..array_upper(matrix, 1) LOOP
        FOR y in array_lower(matrix, 2)..array_upper(matrix, 2) LOOP
            sum := sum + _matrix[x][y];
        END LOOP;
    END LOOP;
    RETURN (sum*1.0/(array_upper(matrix,1)*array_upper(matrix,2) ))::integer ;
END;
$$
LANGUAGE 'plpgsql' IMMUTABLE COST 1000;
```

Apply the raster averaging callback within two pixels of each input pixel in the X and Y directions.

Code

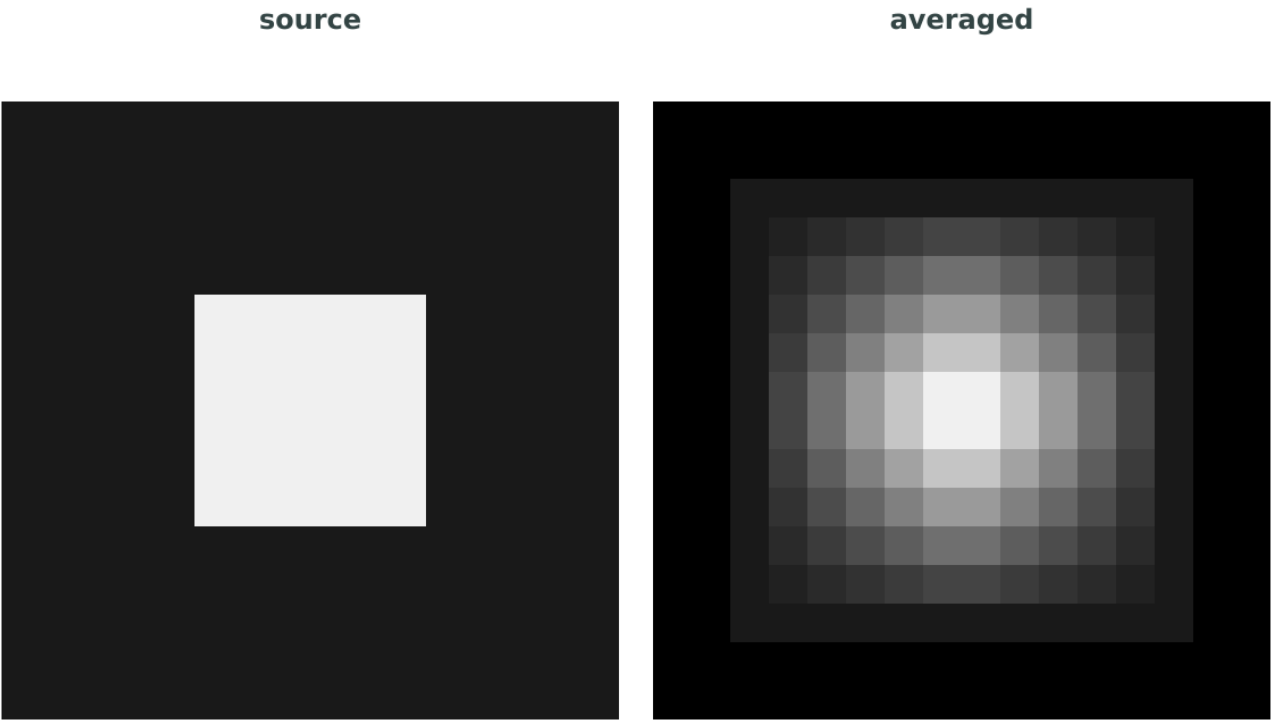
```
SELECT ST_MapAlgebraFctNgb(
    rast,
    1,
    '8BUI',
    4,
    4,
    'rast_avg(float[[[]], text, text[]]'::regprocedure,
    'NULL',
    NULL) AS nn_with_border
FROM katrinas_rescaled
limit 1;
```

A self-contained neighborhood average. The source raster has a sharp central square, while the result spreads those values across neighboring pixels. The source uses a non-NODATA background so nodatamode ignore averages background values with the square values, instead of treating the background as absent.

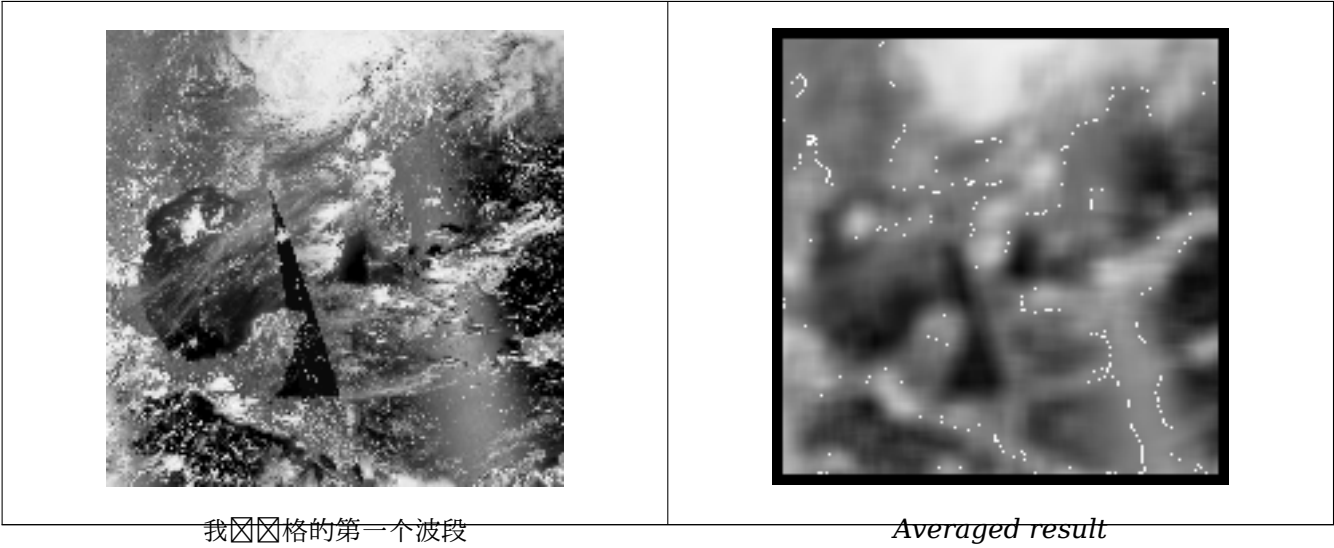
Code

```
WITH source AS (
    SELECT ST_SetValues(
        ST_AddBand(
            ST_MakeEmptyRaster(16, 16, 0, 16, 1, -1, 0, 0, 0),
            1, '8BUI', 25, NULL
        ),
        1, 6, 6,
        ARRAY[
            [240, 240, 240, 240, 240, 240],
            [240, 240, 240, 240, 240, 240],
            [240, 240, 240, 240, 240, 240],
            [240, 240, 240, 240, 240, 240],
            [240, 240, 240, 240, 240, 240],
            [240, 240, 240, 240, 240, 240]
        ]::double precision[[[]]
    ) AS rast
), variants AS (
    SELECT 'source' AS title, rast AS rendered FROM source
    UNION ALL
    SELECT 'averaged', ST_SetBandNoDataValue(ST_MapAlgebraFctNgb(
        rast,
        1,
        '8BUI',
        2,
        2,
        'ST_Mean4ma(float[[[]],text,text[]]'::regprocedure,
        'ignore',
        NULL
    ), NULL) FROM source
)
SELECT title, ST_AsPNG(rendered) AS image
FROM variants
ORDER BY title DESC;
```

Figure



Neighborhood averaging.



相关信息

ST_MapAlgebraFct, ST_MapAlgebraExpr, ST_Rescale

11.12.12 ST_Reclass

`ST_Reclass` — 构建由从原始数据重新分区的波段类型形成的新栅格。`nband` 是要更改的波段。如果未指定 `nband`, 假定 1。所有其他波段均按原样返回。使用案例：将 16BUI 波段重新分类为 8BUI 等，以便更好地呈现栅格格式。

Synopsis

```
raster ST_Reclass(raster rast, integer nband, text reclassexpr, text pixeltype, double precision no-  
dataval=NULL);  
raster ST_Reclass(raster rast, reclassarg[] VARIADIC reclassargset);  
raster ST_Reclass(raster rast, text reclassexpr, text pixeltype);
```

Description

根据 `reclassexpr` 定义的重分区操作，在输入栅格 (`rast`) 上生成一个新的栅格。重分区表达式中的显式参数 `reclassarg`。如果未指定 `band`, 默认使用第 1 波段。

新栅格将具有与原始栅格相同的地理参考、宽度和高度。新栅格的波段像元类型由 `pixeltype` 指定。如果指定了 `reclassargset`, 每个 `reclassarg` 用于定义目标波段的类型，未指定的波段将保持不返回。

可用性: 2.0.0

示例

Basic.

从原始栅格构建一个新栅格，其中波段 2 从 8BUI 重新分类为 4BUI，并且 101-254 中的所有值都置为 `nodata`。

Code

```
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;
```

Output

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

Advanced using multiple reclassargs.

从原始栅格构建一个新栅格，其中波段 1、2、3 分别重新分类为 1BB、4BUI、4BUI 并重新分区。注意，使用可变的 `reclassarg` 参数，参数可以将不确定数量的重新分区参数作为输入（逻辑上与您有的波段一样多）

Code

```
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;
```

Output

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

Advanced Map a single band 32BF raster to multiple viewable bands.

从只有一个 32bf 波段的栅格建一个新的 3 波段（8BUI、8BUI、8BUI 可看栅格）

Code

```
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)
        ]
    );
```

相关信息

[ST_AddBand](#), [ST_Band](#), [ST_BandPixelType](#), [ST_MakeEmptyRaster](#), [reclassarg](#), [ST_Value](#)

11.12.13 ST_ReclassExact

ST_ReclassExact — 建一个新栅格，其波段通原始波段行 1:1 映射重分生成，将原波段中的映射目波段中的新。

Synopsis

raster **ST_ReclassExact**(raster rast, double precision[] inputvalues, double precision[] outputvalues, integer bandnumber=1, text pixeltype=32BF, double precision nodatavalue=NULL);

Description

根据 `inputvalues` 和 `outputvalues` 数组定义的重分类操作，生成一个新的栅格。输入栅格中与数组匹配的像元将被映射到新的值，所有未匹配的像元将被映射到 `nodatavalue`。

输出像元类型默认为 `float`，但可通过 `pixeltype` 参数进行指定。若未指定 `bandnumber`，默认使用第 1 波段。新生成的 raster 将与原始 raster 具有相同的地理参考、精度和高度。未指定的波段保持原样返回，不做更改。

可用性：3.6.0

示例：基本用法

创建一个小型栅格，并将其像元映射到新的值。

Create a raster with four pixels: first row 1 2, second row 3 4.

Code

```
CREATE TABLE reclassexact (
  id integer,
  rast raster
);
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.

Code

```
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 pixel type
  ) AS rast
FROM reclassexact
WHERE id = 1
)
```



```
SELECT 'rce-1', (ST_DumpValues(rc.rast)).*
FROM rc;
```

相关信息

[ST_Reclass](#), [ST_AddBand](#), [ST_Band](#), [ST_MakeEmptyRaster](#)

11.12.14 ST_Union

ST_Union — 将一个栅格切片的并集返回由 1 个或多个波段组成的栅格。

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);
```

Description

将一个栅格切片的并集返回由至少一个波段组成的栅格。生成的栅格范围是整个栅格集的范围。在交集的情况下，结果由 `uniontype` 定义，它是以下之一：LAST（默认）、FIRST、MIN、MAX、COUNT、SUM、MEAN、RANGE。



Note

为了合并栅格，它必须具有相同的坐标系。使用 [ST_SameAlignment](#) 和 [ST_NotSameAlignmentReason](#) 获取更多信息和帮助。解决坐标系的一种方法是使用 [ST_Resample](#) 并使用相同的参考栅格行。

可用性: 2.0.0

增强: 2.1.0 提高速度（完全基于 C）。

可用性: 2.1.0 引入了 `ST_Union(rast, unionarg)` 变体。

增强: 2.1.0 `ST_Union(rast)`（变体 1）合并所有输入栅格的所有波段。PostGIS 的早期版本采用第一个波段。

增强: 2.1.0 `ST_Union(rast, uniontype)`（变体 4）合并所有输入栅格的所有波段。

示例

Reconstitute a single band chunked raster tile.

This creates a single band from first band of raster tiles. That form the original file system tile.

Code

```
SELECT filename, ST_Union(rast, 1) As file_rast
FROM sometable WHERE filename IN('dem01', 'dem02') GROUP BY filename;
```

Return a multi-band raster that is the union of tiles intersecting geometry.

This creates a multi band raster collecting all the tiles that intersect a line. Note that In 2.0, this would have just returned a single band raster. , new union works on all bands by default. This is equivalent to unionarg: ARRAY[ROW(1, 'LAST'), ROW(2, 'LAST'), ROW(3, 'LAST')]::unionarg[.]

Code

```
SELECT ST_Union(rast)
FROM aerals.boston
WHERE ST_Intersects(rast, ST_GeomFromText('LINESTRING(230486 887771,230500 88772)', 26986) ←
);
```

Select and reorder bands while unioning tiles that intersect a geometry.

如果我☐只想要波段的子集或者想要更改波段的☐序，☐☐里我☐使用更☐的☐法

This creates a multi band raster collecting all the tiles that intersect a line.

Code

```
SELECT ST_Union(rast, ARRAY[ROW(2, 'LAST'), ROW(1, 'LAST'), ROW(3, 'LAST')]::unionarg[.])
FROM aerals.boston
WHERE ST_Intersects(rast, ST_GeomFromText('LINESTRING(230486 887771,230500 88772)', 26986) ←
);
```

相关信息

[unionarg](#), [ST_Envelope](#), [ST_ConvexHull](#), [ST_Clip](#), [ST_Union](#)

11.13 内置地☐代数回☐函数

11.13.1 ST_Distinct4ma

ST_Distinct4ma — ☐格☐理函数，用于☐算☐域中唯一像素☐的数量。

Synopsis

```
float8 ST_Distinct4ma(float8[][] matrix, text nodatamode, text[] VARIADIC args);
double precision ST_Distinct4ma(double precision[][][] value, integer[][] pos, text[] VARIADIC user-args);
```

Description

☐算像素☐域中唯一像素☐的数量。



Note

格式 1 是一个☐☐的回☐函数，用作[ST_MapAlgebraFctNgb](#)的回☐参数。



Note
格式 2 是一个返回的回函数，用作ST_MapAlgebra (callback function version)（回函数版本）的回参数。



Warning
不鼓励使用格式 1，因 ST_MapAlgebraFctNgb 自 2.1.0 起已被弃用。

可用性: 2.0.0
增：2.1.0 添加格式 2

示例

Code

```
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;
```

Output

rid	st_value
2	3
(1 row)	

相关信息

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 — 从像素的域内插像素的格理函数。

Synopsis

double precision **ST_InvDistWeight4ma**(double precision[][][] value, integer[][] pos, text[] VARIADIC userargs);

Description

使用反距离加权方法估算像素的插值。

有四个可选项参数可以通过 `userargs` 传递。第一个参数是反距离加权方程中使用的介于 0 和 1 之间的功率因数（下面方程中的量 k ）。如果未指定，默认为 1。第二个参数是当感兴趣像素的插值包含在区域的插值中时使用的权重百分比。如果未指定并且感兴趣的像素具有值，则返回该值。

The basic inverse-distance weighted estimate is: $z = (\sum_i z_i / d_i^k) / (\sum_i 1 / d_i^k)$ Here z_i is a neighboring pixel value, d_i is its distance from the pixel being estimated, and k is a real-valued power factor between 0 and 1.



Note

该函数是一个 SQL 的回调函数，用作 `ST_MapAlgebra (callback function version)`（回调函数版本）的回调参数。

可用性 : 2.1.0

示例

Code

```
WITH foo AS (
  SELECT ST_SetValues(ST_AddBand(ST_MakeEmptyRaster(3, 3, 0, 0, 1, -1, 0, 0, 0), 1, '32BF ←
    ', NULL::double precision, -9999),
    1, 1, 1, ARRAY[
      [10, 10, 10],
      [10, NULL, 20],
      [20, 20, 20]
    ]::double precision[][])
) AS rast
SELECT ST_Value(
  ST_MapAlgebra(
    rast,
    1,
    'ST_InvDistWeight4ma(double precision[][][],integer[][],text[])'::regprocedure,
    '32BF',
    'FIRST',
    NULL,
    1,
    1
  ),
  2, 2
) AS interpolated_value
FROM foo;
```

Output

15

相关信息

`ST_MapAlgebra (callback function version)`, `ST_MinDist4ma`

11.13.3 ST_Max4ma

ST_Max4ma — 计算域中最大像素的栅格理函数。

Synopsis

float8 **ST_Max4ma**(float8[][] matrix, text nodatamode, text[] VARIADIC args);
double precision **ST_Max4ma**(double precision[][][] value, integer[][] pos, text[] VARIADIC userargs);

Description

计算像素域中的最大像素。

于格式 2，可以通过将 userargs 来指定 NODATA 像素的替换。



Note
格式 1 是一个的回函数，用作ST_MapAlgebraFctNgb的回参数。



Note
格式 2 是一个的回函数，用作ST_MapAlgebra (callback function version) (回函数版本) 的回参数。



Warning
不鼓励使用格式 1，因 ST_MapAlgebraFctNgb 自 2.1.0 起已被弃用。

可用性: 2.0.0
增：2.1.0 添加格式 2

示例

Code

```
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;
```

Output

rid	st_value
2	254
(1 row)	

相关信息

[ST_MapAlgebraFctNgb](#), [ST_MapAlgebra \(callback function version\)](#), [ST_Min4ma](#), [ST_Sum4ma](#), [ST_Mean4ma](#), [ST_Range4ma](#), [ST_Distinct4ma](#), [ST_StdDev4ma](#)

11.13.4 ST_Mean4ma

`ST_Mean4ma` — 计算区域中平均像素的格理函数。

Synopsis

`float8 ST_Mean4ma(float8[][] matrix, text nodatamode, text[] VARIADIC args);`
`double precision ST_Mean4ma(double precision[][][] value, integer[][] pos, text[] VARIADIC userargs);`

Description

计算像素域中的平均像素。

于格式 2，可以通过将 `userargs` 来指定 NODATA 像素的替换。



Note

格式 1 是一个返回的函数，用作 [ST_MapAlgebraFctNgb](#) 的返回参数。



Note

格式 2 是一个返回的函数，用作 [ST_MapAlgebra \(callback function version\)](#) (返回函数版本) 的返回参数。



Warning

不鼓励使用格式 1，因 [ST_MapAlgebraFctNgb](#) 自 2.1.0 起已被弃用。

可用性: 2.0.0

增修: 2.1.0 添加格式 2

示例

Variant 1.

Code

```
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;
```

Output

rid	st_value
2	253.222229003906

(1 row)

Variant 2.

Code

```
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;
```

Output

rid	st_value
2	253.222229003906

(1 row)

相关信息

[ST_MapAlgebraFctNgb](#), [ST_MapAlgebra \(callback function version\)](#), [ST_Min4ma](#), [ST_Max4ma](#), [ST_Sum4ma](#), [ST_Range4ma](#), [ST_StdDev4ma](#)

11.13.5 ST_Min4ma

ST_Min4ma — 计算域中最小像素的栅格处理函数。

Synopsis

float8 **ST_Min4ma**(float8[][] matrix, text nodatamode, text[] VARIADIC args);
double precision **ST_Min4ma**(double precision[][][] value, integer[][] pos, text[] VARIADIC userargs);

Description

- 计算像素域中的最小像素。
- 于格式 2，可以通过将 userargs 来指定 NODATA 像素的替换。



Note
格式 1 是一个返回的回函数，用作ST_MapAlgebraFctNgb的回参数。



Note
格式 2 是一个返回的回函数，用作ST_MapAlgebra (callback function version) （回函数版本）的回参数。



Warning

不鼓励使用格式 1，因 `ST_MapAlgebraFctNgb` 自 2.1.0 起已被弃用。

可用性: 2.0.0

增修 : 2.1.0 添加格式 2

示例

Code

```
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;
```

Output

rid	st_value
2	250
(1 row)	

相关信息

`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` — 返回感兴趣像素与具有相像素之的最小距离（以像素数位）的格理函数。

Synopsis

double precision **ST_MinDist4ma**(double precision[][][] value, integer[][] pos, text[] VARIADIC user-args);

Description

返回感兴趣的像素与域中具有之的最近像素之的最短距离（以像素数位）。



Note

此函数的目的是提供信息丰富的数据点，帮助从`ST_InvDistWeight4ma`推断感兴趣像素的插之的有用性。当社区人口稀少，此功能特有用。

**Note**

该函数是一个回调的回调函数，用作 **ST_MapAlgebra (callback function version)** (回调函数版本) 的回调参数。

可用性 : 2.1.0

示例

Code

```
WITH foo AS (
  SELECT ST_SetValues(ST_AddBand(ST_MakeEmptyRaster(3, 3, 0, 0, 1, -1, 0, 0, 0), 1, '32BF ←
    ', NULL::double precision, -9999),
    1, 1, 1, ARRAY[
      [10, 10, 10],
      [10, NULL, 20],
      [20, 20, 20]
    ]::double precision[][])
) AS rast
)
SELECT ST_Value(
  ST_MapAlgebra(
    rast,
    1,
    'ST_MinDist4ma(double precision[][][],integer[][],text[])'::regprocedure,
    '32BF',
    'FIRST',
    NULL,
    1,
    1
  ),
  2, 2
) AS min_distance
FROM foo;
```

Output

```
1
```

相关信息

ST_MapAlgebra (callback function version), **ST_InvDistWeight4ma**

11.13.7 ST_Range4ma

ST_Range4ma — 计算区域中像素范围的栅格处理函数。

Synopsis

```
float8 ST_Range4ma(float8[][] matrix, text nodatamode, text[] VARIADIC args);
double precision ST_Range4ma(double precision[][][] value, integer[][] pos, text[] VARIADIC user-args);
```

Description

计算像素域中的像素范围。

于格式 2，可以通过将 `userargs` 来指定 NODATA 像素的替换。



Note
格式 1 是一个返回的回调函数，用作 `ST_MapAlgebraFctNgb` 的回调参数。



Note
格式 2 是一个返回的回调函数，用作 `ST_MapAlgebra (callback function version)` (回调函数版本) 的回调参数。



Warning
不鼓励使用格式 1，因 `ST_MapAlgebraFctNgb` 自 2.1.0 起已被弃用。

可用性: 2.0.0
增加 : 2.1.0 添加格式 2

示例

Code

```
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;
```

Output

rid	st_value
2	4
(1 row)	

相关信息

`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` — 计算域中像素的标准偏差的栅格处理函数。

Synopsis

float8 **ST_StdDev4ma**(float8[][] matrix, text nodatamode, text[] VARIADIC args);
double precision **ST_StdDev4ma**(double precision[][][] value, integer[][] pos, text[] VARIADIC user-args);

Description

计算像素域中像素的准偏差。



Note

格式 1 是一个回函数，用作ST_MapAlgebraFctNgb的回参数。



Note

格式 2 是一个回函数，用作ST_MapAlgebra (callback function version) (回函数版本) 的回参数。



Warning

不鼓励使用格式 1，因 ST_MapAlgebraFctNgb 自 2.1.0 起已被弃用。

可用性: 2.0.0

增：2.1.0 添加格式 2

示例

Code

```
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;
```

Output

rid	st_value
2	1.30170822143555
(1 row)	

相关信息

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 — 栅格地理函数，计算域中所有像素的和。

Synopsis

```
float8 ST_Sum4ma(float8[][] matrix, text nodatamode, text[] VARIADIC args);
double precision ST_Sum4ma(double precision[][][] value, integer[][] pos, text[] VARIADIC user-args);
```

Description

计算像素域中所有像素的和。

对于格式 2，可以通过将 userargs 来指定 NODATA 像素的替换。



Note

格式 1 是一个返回的函数，用作 **ST_MapAlgebraFctNgb** 的返回参数。



Note

格式 2 是一个返回的函数，用作 **ST_MapAlgebra (callback function version)** (返回函数版本) 的返回参数。



Warning

不鼓励使用格式 1，因 **ST_MapAlgebraFctNgb** 自 2.1.0 起已被弃用。

可用性: 2.0.0

增加: 2.1.0 添加格式 2

示例

Code

```
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;
```

| **Output**

```
rid | st_value
-----+-----
    2 |      2279
(1 row)
```

相关信息

[ST_MapAlgebraFctNgb](#), [ST_MapAlgebra \(callback function version\)](#), [ST_Min4ma](#), [ST_Max4ma](#), [ST_Mean4ma](#), [ST_Range4ma](#), [ST_Distinct4ma](#), [ST_StdDev4ma](#)

11.14 栅格地理：DEM（高程）

11.14.1 ST_Aspect

`ST_Aspect` — 返回高程栅格波段的坡向（默认为以度为单位）。对于分析地形很有用。

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);
```

Description

返回高程栅格波段的坡向（默认为以度为单位）。利用微分代数并将微分比方程用于相邻像素。

`units` 表示坡度的单位。可能的取值有：RADIANS（弧度），DEGREES（度，默认为度）。

当 `units = RADIANS` 时，值介于 0 到 $2 * \pi$ 弧度之间，从北向逆时针测量。

当 `units = DEGREES` 时，值介于 0 到 360 度之间（从北向逆时针测量）。

如果像素的斜率为零，该像素的值横比-1。



Note

有关坡度、坡向和山体阴影的更多信息，请参考 [ESRI - 山体阴影的工作原理](#) 和 [ERDAS 指南 - 坡向图像](#)。

可用性: 2.0.0

增强: 2.1.0 使用 `ST_MapAlgebra()` 并添加可选的 `interpolate_nodata` 函数参数

更改: 2.1.0 在之前的版本中，返回以弧度为单位。现在，返回默认度数

示例

Variant 1.

This symmetric peak highlights the expected compass directions at the corners and the -1 aspect at the summit. The generated PNG shows the full aspect surface without relying on a manual illustration. The generated source raster is large enough for the PNG to show the directional field around the summit.

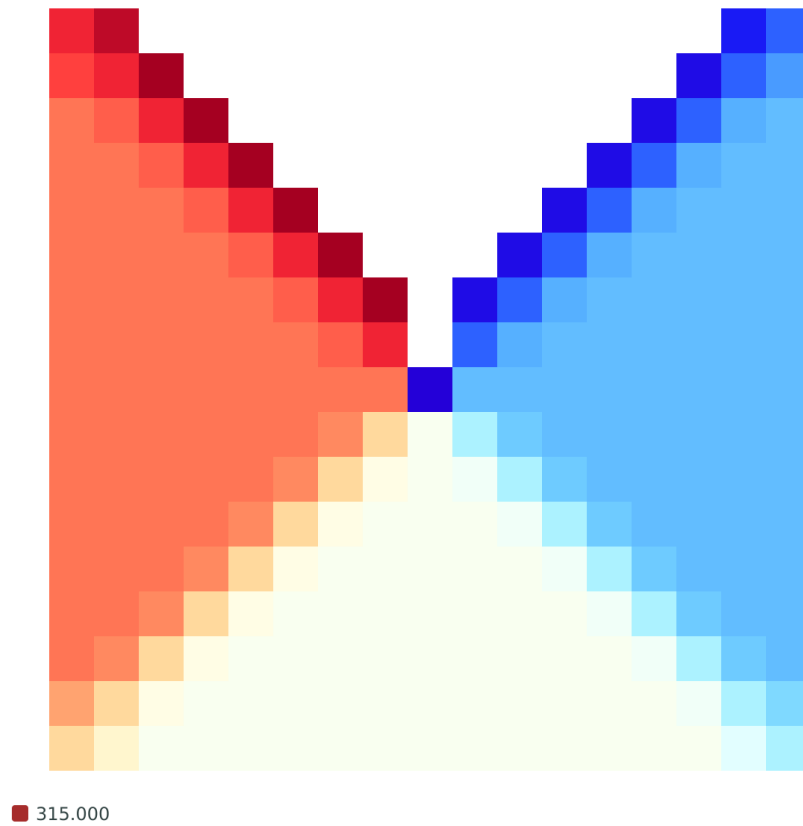
I Code

```

WITH pixels AS (
    SELECT
        x,
        y,
        (20 - greatest(abs(x - 9), abs(y - 9)))::double precision AS value
    FROM generate_series(1, 17) AS y
    CROSS JOIN generate_series(1, 17) AS x
), rows AS (
    SELECT
        y,
        array_agg(value ORDER BY x) AS row_values
    FROM pixels
    GROUP BY y
), source AS (
    SELECT ST_SetValues(
        ST_AddBand(
            ST_MakeEmptyRaster(17, 17, 0, 0, 1, -1, 0, 0, 0),
            1, '32BF', 0, -9999
        ),
        1, 1, 1,
        array_agg(row_values ORDER BY y)::double precision[][]
    ) AS rast
    FROM rows
), aspect AS (
    SELECT ST_Aspect(rast, 1, '32BF') AS rast
    FROM source
)
SELECT
    round(ST_Value(rast, 1, 1, 1)::numeric, 3) AS northwest_degrees,
    round(ST_Value(rast, 1, 9, 9)::numeric, 3) AS summit_degrees,
    round(ST_Value(rast, 1, 17, 17)::numeric, 3) AS southeast_degrees,
    ST_AsPNG(ST_ColorMap(rast, 1, 'bluered')) AS image
FROM aspect;

```

Figure



Variant 2.

覆盖切片的完整示例。

Code

```
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;
```

相关信息

[ST_MapAlgebra \(callback function version\)](#), [ST_TRI](#), [ST_TPI](#), [ST_Roughness](#), [ST_HillShade](#), [ST_Slope](#)

11.14.2 ST_HillShade

`ST_HillShade` — 使用提供的方位角、高度、亮度和比例输入返回高程栅格的假照明。

Synopsis

```
raster ST_HillShade(raster rast, integer band=1, text pixeltype=32BF, double precision azimuth=315,
double precision altitude=45, double precision max_bright=255, double precision scale=1.0, boolean
interpolate_nodata=FALSE);
raster ST_HillShade(raster rast, integer band, raster customextent, text pixeltype=32BF, double pre-
cision azimuth=315, double precision altitude=45, double precision max_bright=255, double preci-
sion scale=1.0, boolean interpolate_nodata=FALSE);
```

Description

使用方位角、高度、亮度和比例输入返回高程栅格的假照明。利用代数并将山体阴影方程用于每个像素。返回像素介于 0 到 255 之间。

方位角 (azimuth) 是从北向测量的 0 到 360 度之间的值。

高度 (altitude) 是 0 到 90 度之间的值，其中 0 度位于地平线，90 度位于正上方。

亮度 (max_bright) 是 0 到 255 之间的值，其中 0 表示无亮度，255 表示最大亮度。

比例 (scale) 是垂直单位与水平单位的比率。英寸:LatLon 使用 `scale=370400`，米:LatLon 使用 `scale=111120`。

如果 `interpolate_nodata` 为 `TRUE`，则在计算山体阴影照明之前，将使用 `ST_InvDistWeight4ma` 输入栅格中的 `NODATA` 像素进行插值。



Note

有关山体阴影的更多信息，请参考[山体阴影的工作原理](#)。

可用性: 2.0.0

新增: 2.1.0 使用 `ST_MapAlgebra()` 并添加可选的 `interpolate_nodata` 函数参数

更改: 2.1.0 在之前的版本中，方位角和高度以弧度表示。现在，方位角和高度以度表示

示例

Variant 1.

Code

```
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

st_dumpvalues
```

Output

```

(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)
```

Variant 2.
覆盖切片的完整示例。

Code

```

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;
```

相关信息

[ST_MapAlgebra \(callback function version\)](#), [ST_TRI](#), [ST_TPI](#), [ST_Roughness](#), [ST_Aspect](#), [ST_Slope](#)

11.14.3 ST_Roughness

ST_Roughness — 返回一个由算出的数字高程模型（DEM）的‘粗糙度’的栅格。

Synopsis

raster **ST_Roughness**(raster rast, integer nband, raster customextent, text pixeltype='32BF', boolean interpolate_nodata=FALSE);

Description

通过指定区域的最小值减去最大值来计算数字高程模型（DEM）的“粗糙度”。

可用性：2.1.0

示例

Code

```
WITH foo AS (
  SELECT ST_SetValues(ST_AddBand(ST_MakeEmptyRaster(5, 5, 0, 0, 1, -1, 0, 0, 0), 1, '32BF',
    ' ', 0, -9999),
    1, 1, 1, ARRAY[
      [1, 1, 2, 3, 3],
      [1, 2, 4, 15, 3],
      [2, 4, 10, 7, 5],
      [3, 5, 7, 6, 4],
      [3, 3, 5, 4, 4]
    ]::double precision[][])
  ) AS rast
)
SELECT ST_Value(ST_Roughness(rast, 1, '32BF'), 3, 3) AS roughness
FROM foo;
```

Output

```
13
```

相关信息

[ST_MapAlgebra \(callback function version\)](#), [ST_TRI](#), [ST_TPI](#), [ST_Slope](#), [ST_HillShade](#), [ST_Aspect](#)

11.14.4 ST_Slope

ST_Slope — 返回高程格的坡度（默以度位）。用于分析地形很有用。

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);

I Code

```

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;

```

相关信息

[ST_MapAlgebra \(callback function version\)](#), [ST_TRI](#), [ST_TPI](#), [ST_Roughness](#), [ST_HillShade](#), [ST_Aspect](#)

11.14.5 ST_TPI

ST_TPI — 返回一个☐算出的地形位置指数（Topographic Position Index）的☐格。

Synopsis

raster **ST_TPI**(raster rast, integer nband, raster customextent, text pixelftype="32BF" , boolean interpolate_nodata=FALSE);

Description

☐算地形位置指数，其定☐☐半径☐ 1 的焦点平均☐☐去中心像元。



Note

此函数☐支持焦点平均半径☐ 1。

可用性：2.1.0

示例

Code

```
WITH foo AS (  
  SELECT ST_SetValues(ST_AddBand(ST_MakeEmptyRaster(5, 5, 0, 0, 1, -1, 0, 0, 0), 1, '32BF',  
    ' ', 0, -9999),  
    1, 1, 1, ARRAY[  
      [1, 1, 2, 3, 3],  
      [1, 2, 4, 15, 3],  
      [2, 4, 10, 7, 5],  
      [3, 5, 7, 6, 4],  
      [3, 3, 5, 4, 4]  
    ]::double precision[][])  
  ) AS rast  
)  
SELECT ST_Value(ST_TPI(rast, 1, '32BF'), 3, 3) AS tpi  
FROM foo;
```

Output

3.75

相关信息

[ST_MapAlgebra \(callback function version\)](#), [ST_TRI](#), [ST_Roughness](#), [ST_Slope](#), [ST_HillShade](#), [ST_Aspect](#)

11.14.6 ST_TRI

ST_TRI — 返回具有地形固性指数的栅格。

Synopsis

raster **ST_TRI**(raster rast, integer nband, raster customextent, text pixeltype="32BF" , boolean interpolate_nodata=FALSE);

Description

地形固性指数是通过将中心像素与其相邻像素行比、取差异的平方并求平均来计算的。



Note
此函数支持焦点平均半径 1。

可用性 : 2.1.0

示例

Code

```
WITH foo AS (
  SELECT ST_SetValues(ST_AddBand(ST_MakeEmptyRaster(5, 5, 0, 0, 1, -1, 0, 0, 0), 1, '32BF',
    0, -9999),
    1, 1, 1, ARRAY[
      [1, 1, 2, 3, 3],
      [1, 2, 4, 15, 3],
      [2, 4, 10, 7, 5],
      [3, 5, 7, 6, 4],
      [3, 3, 5, 4, 4]
    ]::double precision[])
  ) AS rast
)
SELECT ST_Value(ST_TRI(rast, 1, '32BF'), 3, 3) AS tri
FROM foo;
```

Output

5

相关信息

[ST_MapAlgebra \(callback function version\)](#), [ST_Roughness](#), [ST_TPI](#), [ST_Slope](#), [ST_HillShade](#), [ST_Aspect](#)

11.14.7 ST_InterpolateRaster

ST_InterpolateRaster — 基于输入的 3 点集插网格表面，使用 X 和 Y 在网格上定位点，并使用点的 Z 作表面高程。

Synopsis

raster **ST_InterpolateRaster**(geometry input_points, text algorithm_options, raster template, integer template_band_num=1);

Description

基于输入的 3 点集插网格表面，使用 X 和 Y 在网格上定位点，并使用点的 Z 作表面高程。有五种可用的插算法：反距离、反距离最近、移平均、最近和性插。有关算法及其参数的更多信息，参 [gdal_grid 文档](#)。有关如何算插的更多信息，参 [GDAL 网格教程](#)。

输入参数：

input_points 插的点。任何具有 Z 的几何形都是可接受的，将使用入中的所有点。

algorithm_options 定算法和算法的字符串，采用 [gdal_grid](#) 使用的格式。例如，于平滑度 2 的反距离插，您可以使用 “invdist:smoothing=2.0”

template 用于出网格几何形状的网格模板。度、高度、像素大小、空范和像素型将从模板中取。

template_band_num 默情况下，模板格中的第一个波段用于出网格，但可以使用此参数行整。

可用性：3.2.0

示例

Code

```
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')
)
```

相关信息

ST_Contour

11.14.8 ST_Contour

ST_Contour — 使用 **GDAL 轮廓算法** 从提供的栅格波段生成一矢量轮廓。

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);

Description

使用 **GDAL 轮廓算法** 从提供的栅格生成一矢量轮廓。

当 **fixed_levels** 参数非空数组，**level_interval** 和 **level_base** 参数被忽略。

输入参数：

rast 生成等高线的栅格

bandnumber 用于生成等高线的波段

level_interval 生成的等高线之间的高程间隔

level_base 用于用等高线间隔的“基准”通常为零，但也可以是不同的。要生成高程 10 米的等高线，间隔 5、15、25、...，**LEVEL_BASE** 将 5。

fixed_levels 生成的等高线之间的高程间隔

polygonize 如果 true，将构建等高线多边形，而不是多线。

返回是一具有以下属性的：

geom 轮廓的几何形状。

id GDAL 予轮廓的唯一标识符。

value 代表的栅格。于高程 DEM 入，将是出等高线的高程。

可用性：3.2.0

示例

Code

```
WITH c AS (  
SELECT (ST_Contour(rast, 1, fixed_levels => ARRAY[100.0, 200.0, 300.0])).*  
FROM dem_grid WHERE rid = 1  
)  
SELECT geom, id, value  
FROM c;
```

相关信息

[ST_InterpolateRaster](#)

11.15 栅格处理：栅格到几何

11.15.1 Box3D

Box3D — 返回栅格封框的 box 3d 表示形式。

Synopsis

box3d **Box3D**(raster rast);

Description

返回表示栅格范围的框。

多边形由界框的角点定义 ((MINX, MINY), (MAXX, MAXY))

更改：2.0.0 在 2.0 之前的版本中，曾有 box2d 而不是 box3d。由于 box2d 是已弃用的类型，因此已更改 box3d。

示例

Code

```
SELECT  
    rid,  
    Box3D(rast) AS rastbox  
FROM dummy_rast;
```

Output

rid	rastbox
1	BOX3D(0.5 0.5 0,20.5 60.5 0)
2	BOX3D(3427927.75 5793243.5 0,3427928 5793244 0)

相关信息

[ST_Envelope](#)

11.15.2 ST_ConvexHull

ST_ConvexHull — 返回栅格的凸包几何形状，包括等于 BandNoDataValue 的像素。对于矩形形状和非斜栅格，输出与 ST_Envelope 相同的结果，因此矩形或非斜栅格有用。

Synopsis

geometry **ST_ConvexHull**(raster rast);

Description

返回栅格的凸包几何形状，包括 NoDataBandValue 像素。对于矩形形状和非斜栅格，输出或多或少会与 ST_Envelope 相同的结果，因此矩形或非斜栅格有用。



Note

ST_Envelope 将坐标向下取整，因此在栅格周围添加了一小部分缓冲区，因此结果与不坐向下取整的 ST_ConvexHull 稍有不同。

示例

See doc/development/internals/raster-affine.md for the scale, skew, and rotation terms that make the convex hull and envelope differ for skewed rasters.

For regular non-skewed rasters, the envelope and convex hull are more or less the same.

Code

```
SELECT ST_ConvexHull(rast) As convhull, ST_Envelope(rast) As env
FROM dummy_rast WHERE rid=1;
```

Output

convhull	env
-----+-----	
POLYGON((0.5 0.5,20.5 0.5,20.5 60.5,0.5 60.5,0.5 0.5))	

Now we skew the raster. Note how the convex hull and envelope are now different.

Code

```
SELECT ST_ConvexHull(rast) As convhull, ST_Envelope(rast) As env
FROM (SELECT ST_SetRotation(rast, 0.1, 0.1) As rast
      FROM dummy_rast WHERE rid=1) As foo;
```

Output

convhull	env
-----+-----	
POLYGON((0.5 0.5,20.5 1.5,22.5 61.5,2.5 60.5,0.5 0.5))	

相关信息

[ST_Envelope](#), [ST_MinConvexHull](#), [ST_ConvexHull](#), [ST_AsText](#)

11.15.3 ST_DumpAsPolygons

ST_DumpAsPolygons — 从给定的栅格中返回一行 geomval (geom,val) 行。如果未指定波段号，波段号默认为 1。

Synopsis

setof geomval **ST_DumpAsPolygons**(raster rast, integer band_num=1, boolean exclude_nodata_value=TRUE)

Description

它是一个集合返回函数 (SRF)。它返回一行 geomval 行，由几何图形 (geom) 和像素值 (val) 组成。每个多边形是栅格的所有像素的并集，这些像素具有由 val 表示的相同像素值。

ST_DumpAsPolygon 对于多边形化栅格很有用。它与 GROUP BY 相反，它新建行。例如，它可用于将栅格展为多个 POLYGONS/MULTIPOLYGONS。

从 3.3.0，禁用修剪和修复以提高性能。可能会导致无效的几何图形。

Changed: 3.7.0 - 多边形化程序已支持 PostgreSQL 中断，因此取消操作和语句超时会立即中止处理。

可用性：需要 GDAL 1.7 或更高版本。



Note

如果没有波段位置数据，不会返回具有空值的像素，除非 except_nodata_value=false。



Note

如果您只关心栅格中具有特定值的像素数量，使用 ST_ValueCount 速度更快。



Note

与 ST_PixelAsPolygons 不同，在 ST_PixelAsPolygons 中，无论像素值如何，都会为每个像素返回一个几何图形。

示例

This syntax uses a LATERAL join.

Code

```
SELECT val,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;
```

| **Output**

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))

相关信息

[geomval](#), [ST_AsRasterAgg](#), [ST_Value](#), [ST_Polygon](#), [ST_ValueCount](#)

11.15.4 ST_Envelope

ST_Envelope — 返回栅格范围的多边形表示形式。

Synopsis

geometry **ST_Envelope**(raster rast);

Description

返回栅格范围的多边形表示形式（以 srid 定义的空坐标位置表示）。它是一个表示多边形的 float8 最小边界框。多边形由边界框的角点定义 ((MINX, MINY), (MINX, MAXY), (MAXX, MAXY), (MAXX, MINY), (MINX, MINY))

示例

| **Code**

```
SELECT rid, ST_Envelope(rast) As envgeomwkt
FROM dummy_rast;
```

| **Output**

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))

相关信息

[ST_Envelope](#), [ST_AsText](#), [ST_SRID](#)

11.15.5 ST_MinConvexHull

ST_MinConvexHull — 返回栅格的凸包几何形状（不包括 NODATA 像素）。

Synopsis

geometry **ST_MinConvexHull**(raster rast, integer nband=NULL);

Description

返回栅格的凸包几何形状（不包括 NODATA 像素）。如果 **nband** 为 NULL，则考虑栅格的所有波段。

可用性：2.1.0

示例

Code

```
WITH foo AS (
  SELECT
    ST_SetValues(ST_SetValues(ST_AddBand(ST_AddBand(ST_MakeEmptyRaster(9, 9, 0, 0, 1, ←
      -1, 0, 0, 0), 1, '8BUI', 0, 0), 2, '8BUI', 1, 0),
        1, 1, 1,
        ARRAY[
          [0, 0, 0, 0, 0, 0, 0, 0, 0],
          [0, 0, 0, 0, 0, 0, 0, 0, 0],
          [0, 0, 0, 0, 0, 0, 0, 0, 0],
          [0, 0, 0, 1, 0, 0, 0, 0, 1],
          [0, 0, 0, 1, 1, 0, 0, 0, 0],
          [0, 0, 0, 1, 0, 0, 0, 0, 0],
          [0, 0, 0, 0, 0, 0, 0, 0, 0],
          [0, 0, 0, 0, 0, 0, 0, 0, 0],
          [0, 0, 0, 0, 0, 0, 0, 0, 0]
        ]::double precision[][])
    ),
    2, 1, 1,
    ARRAY[
      [0, 0, 0, 0, 0, 0, 0, 0, 0],
      [0, 0, 0, 0, 0, 0, 0, 0, 0],
      [0, 0, 0, 0, 0, 0, 0, 0, 0],
      [1, 0, 0, 0, 0, 1, 0, 0, 0],
      [0, 0, 0, 0, 1, 1, 0, 0, 0],
      [0, 0, 0, 0, 0, 1, 0, 0, 0],
      [0, 0, 0, 0, 0, 0, 0, 0, 0],
      [0, 0, 0, 0, 0, 0, 0, 0, 0],
      [0, 0, 1, 0, 0, 0, 0, 0, 0]
    ]::double precision[][])
  ) AS rast
)
SELECT title, geom AS geom
FROM foo
CROSS JOIN LATERAL (VALUES
  ('raster hull', ST_ConvexHull(rast)),
  ('all bands', ST_MinConvexHull(rast)),
  ('band 1', ST_MinConvexHull(rast, 1)),
  ('band 2', ST_MinConvexHull(rast, 2))
) AS variants(title, geom);
```

Output

title	geom
raster hull	POLYGON((0 0,9 0,9 -9,0 -9,0 0))
all bands	POLYGON((0 -3,9 -3,9 -9,0 -9,0 -3))
band 1	POLYGON((3 -3,9 -3,9 -6,3 -6,3 -3))
band 2	POLYGON((0 -3,6 -3,6 -9,0 -9,0 -3))

Figure



相关信息

[ST_Envelope](#), [ST_ConvexHull](#), [ST_ConvexHull](#), [ST_AsText](#)

11.15.6 ST_Polygon

`ST_Polygon` — 返回由具有非无数据值的像素并集形成的多边形几何体。如果未指定波段号，波段号默认为 1。

Synopsis

geometry **ST_Polygon**(raster rast, integer band_num=1);

Description

- 更改 3.3.0，禁用修剪和修复以提高性能。可能会导致无效的几何形状。
- 可用性：0.1.6 需要 GDAL 1.7 或更高版本。
- 增强：2.1.0 提高速度（完全基于 C）并确保返回的多边形有效。
- 更改：2.1.0 在之前的版本中有时会返回多边形，更改后始终返回多边形。

示例

Compare polygonization of every pixel with polygonization after zero is assigned as the NODATA value. The latter preserves the gap formed by zero-valued cells.

Code

```
WITH source AS (
  SELECT ST_SetValues(
    ST_AddBand(
      ST_MakeEmptyRaster(4, 4, 0, 4, 1, -1, 0, 0, 0),
      1, '8BUI', 1, NULL
    ),
    1, 1, 1,
    ARRAY[
      [1, 1, 1, 1],
      [1, 0, 0, 1],
      [1, 1, 0, 1],
      [1, 1, 1, 1]
    ]::double precision[][]) AS rast
), variants AS (
  SELECT 'all pixels' AS title, ST_Polygon(rast) AS geom
  FROM source
  UNION ALL
  SELECT
    'zero excluded',
    ST_Polygon(ST_SetBandNoDataValue(rast, 1, 0))
  FROM source
)
SELECT title, geom AS geom
FROM variants
ORDER BY CASE title
  WHEN 'all pixels' THEN 1
  ELSE 2
END;
```

Figure



相关信息

[ST_Value](#), [ST_DumpAsPolygons](#)

11.15.7 ST_IntersectionFractions

`ST_IntersectionFractions` — 计算给定 geometry 覆盖到每个 raster 像元的比例。

Synopsis

```
raster ST_IntersectionFractions(raster rast, geometry geom);
```

Description

计算给定 geometry 覆盖每个 raster 像元的比例。第一个参数是 `raster`，定义用于计算的网格几何；第二个参数是 `geometry`，与网格相加并据此计算每个像元：

- `Polygon` 返回像元面中被 `geometry` 覆盖的比例，
- `LineString` 返回在像元中所占的角度。

可用性：3.6.0 -- 需要 GEOS 3.14 或更高版本。

示例

This rotated square polygon covers half of each cell in the raster.

Code

```
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
    ));

SELECT name, ST_DumpValues(
    ST_IntersectionFractions(
        rast,
        'POLYGON((5 0,0 5,5 10,10 5,5 0))'::geometry),
    1)
FROM raster_proportions_rast;
```

Output

```
2x2 raster covering 0,0 to 10,10
-----
{{0.5,0.5},{0.5,0.5}}
```

相关信息

[ST_MakeEmptyRaster](#)

11.16 ☐格☐算符

11.16.1 &&

&& — 如果 A 的☐界框与 B 的☐界框相交，☐返回 TRUE。

Synopsis

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

Description

&& ☐算符在☐格/几何体 A 的☐界框与☐格/几何体 B 的☐界框相交☐返回 TRUE。



Note
☐操作数将利用☐格上可能可用的任何索引。

可用性: 2.0.0

示例

Code

```
SELECT A.rid As a_id, B.rid As b_id, A.rast && B.rast As intersect
FROM dummy_rast AS A CROSS JOIN dummy_rast AS B LIMIT 3;
```

Output

a_id	b_id	intersect
2	2	t
2	3	f
2	1	f

11.16.2 &<

&< — 如果 A 的☐界框位于 B 的左☐，☐返回 TRUE。

Synopsis

boolean **&<**(raster A , raster B);

Description

`&<` 运算符在栅格 A 的边界框与栅格 B 的边界框重叠或位于栅格 B 的边界框的左边界返回 `TRUE`，或者更准确地，在栅格 B 的边界框的右边界没有重叠返回 `TRUE`。



Note
操作数将利用栅格上可能可用的任何索引。

示例

Code

```
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;
```

Output

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 &>

`&>` — 如果 A 的边界框位于 B 的右边界，返回 `TRUE`。

Synopsis

`boolean &>( raster A , raster B );`

Description

`&>` 运算符在栅格 A 的边界框与栅格 B 的边界框重叠或位于栅格 B 的边界框的右边界返回 `TRUE`，或者更准确地，在栅格 B 的边界框的左边界没有重叠返回 `TRUE`。



Note
操作符将利用几何上可能可用的任何索引。

示例

Code

```
SELECT A.rid As a_id, B.rid As b_id, A.rast &
> B.rast As overright
FROM dummy_rast AS A CROSS JOIN dummy_rast AS B;
```

Output

a_id	b_id	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 =

= — 如果 A 的栅格框与 B 的栅格框相同，返回 TRUE。使用双精度栅格框。

Synopsis

boolean =(raster A , raster B);

Description

= 运算符在栅格 A 的栅格框与栅格 B 的栅格框相同返回 TRUE。PostgreSQL 使用栅格定义的 =、< 和 > 运算符来行格的内部排序和比（例如，在 GROUP BY 或 ORDER BY 子句中）。



Caution
这个操作符不会使用可能存在于栅格上的任何索引。使用 ~= 代替。这个操作符主要存在是为了可以按栅格列行分。

可用性：2.1.0

相关信息

~=

11.16.5 @

@ — 如果 A 的栅格框包含在 B 的栅格框中，返回 TRUE。使用双精度栅格框。

Synopsis

boolean @(raster A , raster B);
boolean @(geometry A , raster B);
boolean @(raster B , geometry A);

Description

@ 运算符在栅格/几何体 A 的边界框被栅格/几何体 B 的边界框包含时返回 TRUE。



Note

操作符将使用栅格上的空索引。

可用性：2.0.0 raster@raster、raster@geometry 引入

可用性：2.0.5 geometry @ raster 引入

相关信息

~

11.16.6 ~=

~= — 如果 A 的边界框与 B 的边界框相同，返回 TRUE。

Synopsis

boolean ~= (raster A , raster B);

Description

~= 运算符在栅格 A 的边界框与栅格 B 的边界框相同时返回 TRUE。



Note

操作数将利用栅格上可能可用的任何索引。

可用性：2.0.0

示例

非常有用的用例是取具有相同但代表不同主题的波段栅格并建多波段栅格

Code

```
SELECT ST_AddBand(prec.rast, alt.rast) As new_rast
FROM prec INNER JOIN alt ON (prec.rast ~= alt.rast);
```

相关信息

ST_AddBand, =

11.16.7 ~

~ — 如果 A 的☐界框包含 B 的☐界框，☐返回 TRUE。使用双精度☐界框。

Synopsis

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

Description

如果☐格/几何 A 的☐界框包含☐格/几何 B 的☐界框，☐ ~ ☐算符返回 TRUE。



Note

☐操作符将使用☐格上的空☐索引。

可用性: 2.0.0

相关信息

@

11.17 ☐格和☐格波段空☐关系

11.17.1 ST_Contains

ST_Contains — 如果☐格 rastB 中没有点位于☐格 rastA 的外部且 rastB 的内部至少有一个点位于 rastA 的内部，☐返回 true。

Synopsis

```
boolean ST_Contains( raster rastA , integer nbandA , raster rastB , integer nbandB );  
boolean ST_Contains( raster rastA , raster rastB );
```

Description

☐格 rastA 包含 rastB 当且☐当 rastB 中没有点位于 rastA 的外部且 rastB 的内部至少有一个点位于 rastA 的内部☐。如果未提供波段☐号（或☐置☐ NULL），☐☐☐中☐考☐☐格的凸包。如果提供了波段☐号，☐☐☐中☐考☐那些具有☐（而不是 NODATA）的像素。



Note

此函数将利用☐格上可能可用的任何索引。



Note
要栅格和几何形的空关系，在栅格上使用 ST_Polygon，例如 ST_Contains (ST_Polygon (raster), geometry) 或 ST_Contains (geometry, ST_Polygon (raster))。



Note
ST_Contains() 是 ST_Within() 的逆函数。因此，ST_Contains(rastA, rastB) 意味着 ST_Within(rastB, rastA)。

可用性：2.1.0

示例

Specified band numbers.

Code

```
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;
```

Output

```
NOTICE: The first raster provided has no bands
rid | rid | st_contains
-----+-----+-----
1 | 1 |
1 | 2 | f
```

No band numbers specified.

Code

```
SELECT r1.rid, r2.rid, ST_Contains(r1.rast, r2.rast) FROM dummy_rast r1 CROSS JOIN
dummy_rast r2 WHERE r1.rid = 1;
```

Output

```
rid | rid | st_contains
-----+-----+-----
1 | 1 | t
1 | 2 | f
```

相关信息

ST_Intersects, ST_Within

11.17.2 ST_ContainsProperly

ST_ContainsProperly — 如果 rastB 与 rastA 的内部相交，但不与 rastA 的边界或外部相交，返回 true。

Synopsis

boolean **ST_ContainsProperly**(raster rastA , integer nbandA , raster rastB , integer nbandB);
boolean **ST_ContainsProperly**(raster rastA , raster rastB);

Description

如果 `rastB` 与 `rastA` 的内部相交，但不与 `rastA` 的边界或外部相交，栅格 `rastA` 正确包含 `rastB`。如果未提供波段号（或置为 `NULL`），栅格中考虑栅格的凸包。如果提供了波段号，栅格中考虑那些具有（而不是 `NODATA`）的像素。

栅格 `rastA` 未正确包含其自身，但确包含其自身。



Note
此函数将利用栅格上可能可用的任何索引。



Note
要栅格和几何形的空关系，在栅格上使用 `ST_Polygon`，例如 `ST_ContainsProperly(ST_Polygon (raster), geometry)` 或 `ST_ContainsProperly(geometry, ST_Polygon (raster))`。

可用性 : 2.1.0

示例

Code

```
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;
```

Output

rid	rid	st_containsproperly
2	1	f
2	2	f

相关信息

[ST_Intersects](#), [ST_Contains](#)

11.17.3 ST_Covers

`ST_Covers` — 如果栅格 `rastB` 中没有点位于栅格 `rastA` 之外，返回 `true`。

Synopsis

```
boolean ST_Covers( raster rastA , integer nbandA , raster rastB , integer nbandB );  
boolean ST_Covers( raster rastA , raster rastB );
```

Description

栅格 `rastA` 覆盖 `rastB` 当且仅当 `rastB` 中没有点位于 `rastA` 的外部。如果未提供波段编号（或置为 `NULL`），则在栅格中考虑栅格的凸包。如果提供了波段编号，则在栅格中考虑那些具有非空（而不是 `NODATA`）的像素。



Note
此函数将利用栅格上可能可用的任何索引。



Note
要检查栅格和几何图形的空关系，请在栅格上使用 `ST_Polygon`，例如 `ST_Covers(ST_Polygon(raster), geometry)` 或 `ST_Covers(geometry, ST_Polygon(raster))`。

可用性 : 2.1.0

示例

Code

```
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;
```

Output

rid	rid	st_covers
2	1	f
2	2	t

相关信息

[ST_Intersects](#), [ST_CoveredBy](#)

11.17.4 ST_CoveredBy

`ST_CoveredBy` — 如果栅格 `rastA` 中没有点位于栅格 `rastB` 之外，则返回 `true`。

Synopsis

boolean **ST_CoveredBy**(raster `rastA` , integer `nbandA` , raster `rastB` , integer `nbandB`);
boolean **ST_CoveredBy**(raster `rastA` , raster `rastB`);

Description

当且仅当 `rastA` 中没有点位于 `rastB` 的外部，栅格 `rastA` 才会被 `rastB` 覆盖。如果未提供波段号（或置 `NULL`），则在栅格中考虑栅格的凸包。如果提供了波段号，则在栅格中考虑那些具有（而不是 `NODATA`）的像素。



Note
此函数将利用栅格上可能可用的任何索引。



Note
要检查栅格和几何图形的空关系，在栅格上使用 `ST_Polygon`，例如 `ST_CoveredBy(ST_Polygon(raster), geometry)` 或 `ST_CoveredBy(geometry, ST_Polygon(raster))`。

可用性 : 2.1.0

示例

Code

```
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;
```

Output

rid	rid	st_coveredby
2	1	f
2	2	t

相关信息

[ST_Intersects](#), [ST_Covers](#)

11.17.5 ST_Disjoint

`ST_Disjoint` — 如果栅格 `rastA` 在空上不 与 `rastB` 相交，则返回 `true`。

Synopsis

```
boolean ST_Disjoint( raster rastA , integer nbandA , raster rastB , integer nbandB );
boolean ST_Disjoint( raster rastA , raster rastB );
```


Description

如果栅格 `rastA` 和 `rastB` 不共享任何空值，那么它是脱空的。如果未提供波段编号（或置为 `NULL`），那么栅格中考虑栅格的凸包。如果提供了波段编号，那么栅格中考虑那些具有值（而不是 `NODATA`）的像素。



Note
该函数不使用任何索引。



Note
要检查栅格和几何图形的空值关系，可以在栅格上使用 `ST_Polygon`，例如 `ST_Disjoint(ST_Polygon(raster), geometry)`。

可用性：2.1.0

示例

`Rid = 1` has no bands, hence the NOTICE and the `NULL` value for `st_disjoint`.

Code

```
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;
```

Output

```
NOTICE: The second raster provided has no bands
rid | rid | st_disjoint
-----+-----+-----
 2 |  1 |
 2 |  2 | f
```

This time, without specifying band numbers.

Code

```
SELECT r1.rid, r2.rid, ST_Disjoint(r1.rast, r2.rast) FROM dummy_rast r1 CROSS JOIN ↵
dummy_rast r2 WHERE r1.rid = 2;
```

Output

```
rid | rid | st_disjoint
-----+-----+-----
 2 |  1 | t
 2 |  2 | f
```

相关信息

[ST_Intersects](#)

11.17.6 ST_Intersects

`ST_Intersects` — 如果栅格 `rastA` 与栅格 `rastB` 空值相交，那么返回 `true`。

Synopsis

```
boolean ST_Intersects( raster rastA , integer nbandA , raster rastB , integer nbandB );
boolean ST_Intersects( raster rastA , raster rastB );
boolean ST_Intersects( raster rast , integer nband , geometry geommin );
boolean ST_Intersects( raster rast , geometry geommin , integer nband=NULL );
boolean ST_Intersects( geometry geommin , raster rast , integer nband=NULL );
```

Description

如果栅格 `rastA` 与栅格 `rastB` 空相交，则返回 `true`。如果未提供波段号（或置为 `NULL`），则在栅格的凸包。如果提供了波段号，则在那些具有（而不是 `NODATA`）的像素。



Note

此函数将利用栅格上可能可用的任何索引。

增：2.0.0 引入了支持栅格/栅格相交。



Warning

已更改：2.1.0 `ST_Intersects (raster, geometry)` 体的行已更改以匹配 `ST_Intersects (geometry, raster)` 的行。

示例

Different bands of same raster.

Code

```
SELECT ST_Intersects(rast, 2, rast, 3) FROM dummy_rast WHERE rid = 2;
```

Output

```
t
```

相关信息

[ST_Intersection](#), [ST_Disjoint](#)

11.17.7 ST_Overlaps

`ST_Overlaps` — 如果栅格 `rastA` 和 `rastB` 相交，但其中一个不完全包含另一个，则返回 `true`。

Synopsis

```
boolean ST_Overlaps( raster rastA , integer nbandA , raster rastB , integer nbandB );
boolean ST_Overlaps( raster rastA , raster rastB );
```

Description

如果栅格 `rastA` 在空域上与栅格 `rastB` 重叠，则返回 `true`。这意味着 `rastA` 和 `rastB` 相交，但其中一个不完全包含另一个。如果未提供波段编号（或置为 `NULL`），则在栅格中考虑栅格的凸包。如果提供了波段编号，则在栅格中考虑那些具有值（而不是 `NODATA`）的像素。



Note

此函数将利用栅格上可能可用的任何索引。



Note

要检查栅格和几何图形的空域关系，请在栅格上使用 `ST_Polygon`，例如 `ST_Overlaps (ST_Polygon (raster), geometry)`。

可用性：2.1.0

示例

Comparing different bands of same raster.

Code

```
SELECT ST_Overlaps(rast, 1, rast, 2) FROM dummy_rast WHERE rid = 2;
```

Output

```
f
```

相关信息

[ST_Intersects](#)

11.17.8 ST_Touches

`ST_Touches` — 如果栅格 `rastA` 和 `rastB` 至少有一个共同点但它们的内部不相交，则返回 `true`。

Synopsis

```
boolean ST_Touches( raster rastA , integer nbandA , raster rastB , integer nbandB );  
boolean ST_Touches( raster rastA , raster rastB );
```

Description

如果栅格 `rastA` 在空域上接触栅格 `rastB`，则返回 `true`。这意味着 `rastA` 和 `rastB` 至少有一个共同点，但它们的内部不相交。如果未提供波段编号（或置为 `NULL`），则在栅格中考虑栅格的凸包。如果提供了波段编号，则在栅格中考虑那些具有值（而不是 `NODATA`）的像素。



Note

此函数将利用栅格上可能可用的任何索引。



Note

要检查栅格和几何图形的空关系，在栅格上使用 ST_Polygon，例如 ST_Touches(ST_Polygon(raster), geometry)。

可用性：2.1.0

示例

Code

```
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;
```

Output

rid	rid	st_touches
2	1	f
2	2	f

相关信息

[ST_Intersects](#)

11.17.9 ST_SameAlignment

ST_SameAlignment — 如果栅格具有相同的斜、比例、空参考和偏移（像素可以放在同一网格上而不切割成像素），返回 true；如果没有注意，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 );
```

Description

非聚合版本（主体 1 和 2）：如果两个栅格（直接提供或使用左上角、比例、斜和 srid 的制作）具有相同的比例、斜、srid 以及至少其中之一，返回 true，一个栅格的任何像素的四个角落在一个栅格的网格的任何角上。如果不这样做返回 false 并明确通知。

聚合版本（主体 3）：从一个栅格中，如果其中的所有栅格都，返回 true。ST_SameAlignment() 函数是 PostgreSQL 中的“聚合”函数。意味着它数据行操作，与 SUM() 和 AVG() 函数的操作方式相同。

可用性: 2.0.0

增：2.1.0 添加聚合主体

示例：☐格

Code

```
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;
```

Output

t

Code

```
SELECT ST_SameAlignment(A.rast, b.rast)
FROM dummy_rast AS A CROSS JOIN dummy_rast AS B;
```

Output

```
NOTICE: The two rasters provided have different SRIDs
NOTICE: The two rasters provided have different SRIDs
t
f
f
f
```

相关信息

Section [10.1](#), [ST_NotSameAlignmentReason](#), [ST_MakeEmptyRaster](#)

11.17.10 ST_NotSameAlignmentReason

ST_NotSameAlignmentReason — 返回☐明☐格是否☐☐的文本，如果未☐☐，☐☐明原因。

Synopsis

text **ST_NotSameAlignmentReason**(raster rastA, raster rastB);

Description

返回☐明☐格是否☐☐的文本，如果未☐☐，☐☐明原因。



Note
如果☐格未☐☐的原因有多种，☐☐返回一个原因（第一个失☐的☐☐）。

可用性：2.1.0

示例

Code

```
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)
  )
;
```

Output

st_samealignment	st_notsamealignmentreason
f	The rasters have different scales on the X axis
(1 row)	

相关信息

Section 10.1, [ST_SameAlignment](#)

11.17.11 ST_Within

ST Within — 如果栅格 rastA 中没有点位于栅格 rastB 的外部且 rastA 的内部至少有一个点位于 rastB 的内部，返回 true。

Synopsis

```
boolean ST_Within( raster rastA , integer nbandA , raster rastB , integer nbandB );
boolean ST_Within( raster rastA , raster rastB );
```

Description

栅格 rastA 位于 rastB 内当且仅当 rastA 中没有点位于 rastB 的外部且 rastA 内部的至少一个点位于 rastB 的内部。如果未提供波段编号（或置 NULL），则在栅格中考虑栅格的凸包。如果提供了波段编号，则在那些具有（而不是 NODATA）的像素。



Note
操作数将利用栅格上可能可用的任何索引。



Note
要确定栅格和几何图形的空关系，在栅格上使用 ST_Polygon，例如 ST_Within (ST_Polygon (raster), geometry) 或 ST_Within (geometry, ST_Polygon (raster))。



Note
ST_Within() 是 ST_Contains() 的逆函数。因此，ST_Within(rastA, rastB) 意味着 ST_Contains(rastB, rastA)。

可用性：2.1.0

示例

Code

```
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;
```

Output

rid	rid	st_within
2	1	f
2	2	t

相关信息

ST_Intersects, ST_Contains, ST_DWithin, ST_DFullyWithin

11.17.12 ST_DWithin

ST_DWithin — 如果栅格 rastA 和 rastB 彼此之栅格的距离在指定距离内，返回 true。

Synopsis

boolean **ST_DWithin**(raster rastA , integer nbandA , raster rastB , integer nbandB , double precision distance_of_srid);
boolean **ST_DWithin**(raster rastA , raster rastB , double precision distance_of_srid);

Description

如果栅格 rastA 和 rastB 彼此之栅格的距离在指定距离内，返回 true。如果未提供波段号（或置 NULL），则在栅格中考虑栅格的凸包。如果提供了波段号，则在栅格中考虑那些具有（而不是 NODATA）的像素。距离以栅格空参考系定义的位指定。为了使此函数有意，源栅格必须具有相同的坐投影并具有相同的 SRID。



Note
操作数将利用栅格上可能可用的任何索引。



Note
要栅格和几何形的空关系，在栅格上使用 ST_Polygon，例如 ST_DWithin(ST_Polygon(raster), geometry)。

可用性 : 2.1.0

示例

Code

```
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;
```

Output

rid	rid	st_dwithin
2	1	f
2	2	t

相关信息

ST_Within, ST_DFullyWithin

11.17.13 ST_DFullyWithin

ST_DFullyWithin — 如果栅格 rastA 和 rastB 彼此完全在指定距离内，返回 true。

Synopsis

boolean **ST_DFullyWithin**(raster rastA , integer nbandA , raster rastB , integer nbandB , double precision distance_of_srid);
boolean **ST_DFullyWithin**(raster rastA , raster rastB , double precision distance_of_srid);

Description

如果栅格 rastA 和 rastB 彼此完全在指定距离内，返回 true。如果未提供波段号（或置 NULL），则考虑栅格的凸包。如果提供了波段号，则考虑那些具有（而不是 NODATA）的像素。
距离以栅格空参考系定义的位指定。为了使此函数有意，源栅格必须具有相同的坐投影并具有相同的 SRID。



Note
操作数将利用栅格上可能可用的任何索引。



Note
要栅格和几何形的空关系，在栅格上使用 ST_Polygon，例如 ST_DFullyWithin(ST_Polygon(raster), geometry)。

可用性 : 2.1.0

示例

Code

```
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;
```

Output

```
rid | rid | st_dfullywithin
-----+-----+-----
 2  |  1  | f
 2  |  2  | t
```

相关信息

[ST_Within](#), [ST_DWithin](#)

11.18 网格提示

11.18.1 Out-DB 网格

11.18.1.1 包含多个文件的目录

当 GDAL 打开一个文件时，GDAL 会立即扫描文件的目录以构建其他文件的目录。如果此目录包含多个文件（例如数千、数百万），打开文件会得非常慢（特别是如果文件恰好位于 NFS 等网络存储上）。

为了控制此行为，GDAL 提供了以下环境变量：[GDAL_DISABLE_READDIR_ON_OPEN](#)。将 `GDAL_DISABLE_READDIR_ON_OPEN` 设置为 `TRUE` 以禁用目录扫描。

在 Ubuntu 中（假设您使用的是 PostgreSQL 的 Ubuntu 软件包），可以设置 `GDAL_DISABLE_READDIR_ON_OPEN` 在 `/etc/postgresql/POSTGRESQL_VERSION/CLUSTER_NAME/environment` 中（其中 `POSTGRESQL_VERSION` 是 PostgreSQL 的版本，例如 9.6，`CLUSTER_NAME` 是集群的名称，例如 9.6）。主数据目录。您也可以在此位置设置 PostGIS 环境变量。

Code

```
# 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.
PGIS_GDAL_ENABLED_DRIVERS = 'ENABLE_ALL'

PGIS_ENABLE_OUTDB_RASTERS = 1

GDAL_DISABLE_READDIR_ON_OPEN = 'TRUE'
```

11.18.1.2 最大打开文件数

Linux 和 PostgreSQL 允许的最大打开文件数通常是相当保守的（通常是每个进程最多 1024 个打开文件），这是基于一种假设而设置的，即系统主要由用户使用。然而，对于 Out-DB 网格数据，一个有效的目录很容易超过这个限制（例如，一个包含了 10 年的网格数据集，每天都有一个网格包含了最低和最高温度数据，我们想要知道数据集中某个像素的最小和最大值）。

最⼤的更改是以下 PostgreSQL 设置：`max_files_per_process`。默⼊置 1000，⼊于 Out-DB 规格来⼊太低了。安全的起始⼊可能是 65536，但⼊上取决于您的数据集以及⼊些数据集⼊行的⼊。此⼊置只能在服务器⼊⼊行，并且可能只能在 PostgreSQL 配置文件中⼊行（例如 Ubuntu ⼊境中的 `/etc/postgresql/POSTGRESQL_VERSION/CLUSTER_NAME/postgresql.conf`）。

Code

```
...
# - Kernel Resource Usage -

max_files_per_process = 65536          # min 25
                                         # (change requires restart)
...
```

要做的主要更改是 Linux 内核的打开文件限制。⼊有⼊个部分：

- 整个系⼊最大打开文件数
- 每个⼊程的最大打开文件数

11.18.1.2.1 整个系⼊最大打开文件数

您可以通过以下示例⼊整个系⼊当前最大打开文件数：

Code

```
$ sysctl -a | grep fs.file-max
fs.file-max = 131072
```

如果返回的⼊不⼊大，⼊按照以下示例将文件添加到 `/etc/sysctl.d/` 中：

Code

```
$ 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 每个⼊程的最大打开文件数

我⼊需要增加 PostgreSQL 服务器⼊程每个⼊程打开文件的最大数量。

要⼊看当前 PostgreSQL 服务器⼊程使用的最大打开文件数，⼊按照以下示例⼊行操作（确保 PostgreSQL 正在⼊行）：

Code

```
$ ps aux | grep postgres
```

Output

```
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
```

Code

```
$ cat /proc/31718/limits
```

Output

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

在上面的示例中，我查看了进程 31718 的打开文件限制。无是个 PostgreSQL 进程，它中的任何一个都可以。我感兴趣的值是最大打开文件数。

我希望将最大打开文件数的限制和硬限制增加到大大于我 PostgreSQL 配置 `max_files_per_process` 指定的值。在我示例中，我将 `max_files_per_process` 配置 65536。

在 Ubuntu 中（假设您使用 Ubuntu 的 PostgreSQL 软件包），更改限制和硬限制的最方法是 `/etc/init.d/postgresql` (SysV) 或 `/lib/systemd/system/postgresql*.service` (系统)。

我首先解决 SysV Ubuntu 的情况，其中我将 `ulimit -H -n 262144` 和 `ulimit -n 131072` 添加到 `/etc/init.d/postgresql`。

Code

```
...
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)  
...  

```

✗ 在来解决 Ubuntu 中的 systemd ✗。我✗将在每个 `/lib/systemd/system/postgresql*.service` 文件的 **[Service]** 部分中添加 **LimitNOFILE=131072**。

Code

```
...  
[Service]  
  
LimitNOFILE=131072  
  
...  
  
[Install]  
WantedBy=multi-user.target  
...  

```

✗ 行必要的 systemd 更改后，✗ 确保重新加✗守✗程

Code

```
systemctl daemon-reload
```

Chapter 12

PostGIS 补充

本章介绍了 PostGIS 源 tarball 和源存储库的 extras 文件中的功能。这些并不是与 PostGIS 二进制版本一起打包，但通常是基于 PL/pgSQL 或可以按原样运行的 shell 脚本。

Chapter 13

PostGIS 特殊函数索引

13.1 PostGIS 聚合函数

下面的函数是空 $\square$ 聚合函数，其使用方式与 SQL 聚合函数（例如 `sum` 和 `average`）相同。

- **CG_3DUnion** - 使用 `postgis_sfcgal` $\square$ 行三 $\square$ 合并。
- **ST_3DExtent** - 返回几何 $\square$ 形的 3D $\square$ 界框的聚合函数。
- **ST_AsFlatGeobuf** - 返回一 $\square$ 行的 FlatGeobuf 表示形式。
- **ST_AsGeobuf** - 返回一 $\square$ 行的 Geobuf 表示。
- **ST_AsMVT** - 返回一 $\square$ 行的 MVT 表示形式的聚合函数。
- **ST_ClusterDBSCAN** - 使用 DBSCAN 算法返回每个 $\square$ 入几何 $\square$ 形的簇 id 的窗口函数。
- **ST_ClusterIntersecting** - 将 $\square$ 入几何 $\square$ 形聚 $\square$ 成 $\square$ 接集的聚合函数。
- **ST_ClusterIntersectingWin** - 窗口函数，返回每个 $\square$ 入几何 $\square$ 形的簇 ID，将 $\square$ 入几何 $\square$ 形聚 $\square$ 到 $\square$ 接的集合中。
- **ST_ClusterKMeans** - 使用 K 均 $\square$ 算法返回每个 $\square$ 入几何 $\square$ 形的簇 id 的窗口函数。
- **ST_ClusterRelateWin** - Window function that returns a cluster id for each input geometry, clustering input geometries into connected sets using the relate pattern to determine whether the geometries are connected.
- **ST_ClusterWithin** - 按 $\square$ 隔距离 $\square$ 几何 $\square$ 形 $\square$ 行聚合的聚合函数。
- **ST_ClusterWithinWin** - 窗口函数，返回每个 $\square$ 入几何 $\square$ 形的簇 ID，使用分离距离 $\square$ 行聚 $\square$ 。
- **ST_Collect** - 从一 $\square$ 几何 $\square$ 形 $\square$ 建 GeometryCollection 或 Multi* 几何 $\square$ 形。
- **ST_CoverageClean** - $\square$ 非 $\square$ 范化 $\square$ 入 $\square$ 算出一个 $\square$ 范的多 $\square$ 形覆盖 $\square$ 果（ $\square$ 界匹配、无重 $\square$ 、无 $\square$ 隙）。
- **ST_CoverageInvalidEdges** - 用于 $\square$ 找多 $\square$ 形无法形成有效覆盖范 $\square$ 的位置的窗口函数。
- **ST_CoverageSimplify** - $\square$ 化多 $\square$ 形覆盖范 $\square$ $\square$ 的窗口函数。
- **ST_CoverageUnion** - 通 $\square$ $\square$ 除共享 $\square$ 来 $\square$ 算形成覆盖范 $\square$ 的一 $\square$ 多 $\square$ 形的并集。
- **ST_Extent** - 返回几何 $\square$ 形 $\square$ 界框的聚合函数。
- **ST_MakeLine** - 从 Point, MultiPoint, 或 LineString geometries $\square$ 建 LineString。
- **ST_MemUnion** - 聚合函数，以内存高效但速度 $\square$ 慢的方式 $\square$ 合几何 $\square$ 形

- **ST_MinimumSpanningTree** - Window function that returns a tree id for each input geometry, clustering input geometries into connected trees using the Minimum Spanning Tree algorithm.
- **ST_Polygonize** - 计算由一系列几何图形的边形成的多边形集合。
- **ST_SameAlignment** - 如果栅格具有相同的斜率、比例、空参考和偏移（像素可以放在同一网格上而不切割成像素），则返回 true；如果没有注意栅格，则返回 false。
- **ST_Union** - 计算表示输入几何图形的点集并集的几何图形。
- **ST_Union** - 将一系列栅格切片的并集返回由 1 个或多个波段组成的栅格。
- **TopoElementArray_Agg** - 返回一系列 element_id、类型数组 (topoelements) 的 topoelementarray。

13.2 PostGIS 窗口函数

下面的函数是空窗口函数，其使用方式与 SQL 窗口函数（例如 `row_number()`、`lead()` 和 `lag()`）相同。它们后面必须跟有 `OVER()` 子句。

- **ST_ClusterDBSCAN** - 使用 DBSCAN 算法返回每个输入几何图形的簇 id 的窗口函数。
- **ST_ClusterIntersectingWin** - 窗口函数，返回每个输入几何图形的簇 ID，将输入几何图形聚到连接的集合中。
- **ST_ClusterKMeans** - 使用 K 均值算法返回每个输入几何图形的簇 id 的窗口函数。
- **ST_ClusterRelateWin** - Window function that returns a cluster id for each input geometry, clustering input geometries into connected sets using the relate pattern to determine whether the geometries are connected.
- **ST_ClusterWithinWin** - 窗口函数，返回每个输入几何图形的簇 ID，使用分离距离进行聚类。
- **ST_CoverageClean** - 非规范化输入算出一个规范的多边形覆盖结果（边界匹配、无重叠、无间隙）。
- **ST_CoverageInvalidEdges** - 用于查找多边形无法形成有效覆盖范围的位置的窗口函数。
- **ST_CoverageSimplify** - 简化多边形覆盖范围的窗口函数。
- **ST_MinimumSpanningTree** - Window function that returns a tree id for each input geometry, clustering input geometries into connected trees using the Minimum Spanning Tree algorithm.

13.3 PostGIS SQL-MM 兼容函数

下面列出的函数是符合 SQL/MM 3 标准的 PostGIS 函数

- **CG_3DArea** - 计算 3D 表面几何形状的面积。对于固体将返回 0。
- **CG_3DDifference** - 执行 3D 差异
- **CG_3DIntersection** - 执行 3D 相交
- **CG_3DUnion** - 使用 `postgis_sfcgal` 执行三合一。
- **CG_Volume** - Computes the volume of a 3D geometry. Closed surfaces can have non-zero volume.
- **ST_3DDWithin** - 检查 3D 几何图形是否在指定的 3D 距离内
- **ST_3DDistance** - 返回两个几何图形之间的 3D 笛卡儿最小距离（基于空参考）（以投影单位表示）。
- **ST_3DIntersects** - 检查两个几何图形在 3D 空间中是否相交 - 适用于点、线串、多边形、多面体曲面（区域）

- **ST_3DLength** - 返回线性几何体的 3D 长度。
 - **ST_3DPerimeter** - 返回多边形几何体的 3D 周长。
 - **ST_AddEdgeModFace** - 添加新面，如果面做会分割面，修改原始面并添加新面。
 - **ST_AddEdgeNewFaces** - 添加新面，如果面做会分割一个面，删除原始面并用个新面替它。
 - **ST_AddIsoEdge** - 将由几何 `alinesring` 定义的孤立边添加到接个有孤立点 `anode` 和 `anothernode` 的拓扑，并返回新面的 id。
 - **ST_AddIsoNode** - 将一个孤立的点添加到拓扑中的一个面，并返回新点的点 ID。如果“face” 空 (null)，仍然会建点。
 - **ST_Area** - 返回多边形几何体的面积。
 - **ST_AsBinary** - 返回不 SRID 元数据的几何/地理的 OGC/ISO 已知的二进制 (WKB) 表示形式。
 - **ST_AsGML** - 将几何形作 GML 版本 2 或 3 元素返回。
 - **ST_AsText** - 返回不 SRID 元数据的几何/地理的已知文本 (WKT) 表示形式。
 - **ST_Boundary** - 返回几何形的边界。
 - **ST_Buffer** - 计算覆盖距几何体定距离内所有点的几何体。
 - **ST_Centroid** - 返回几何体的几何中心。
 - **ST_ChangeEdgeGeom** - 改面的形状而不影响拓扑。
 - **ST_Contains** - 面 B 的每个点是否都位于 A 中，并且它的内部是否有一个共同点
 - **ST_ConvexHull** - 计算几何体的凸包。
 - **ST_CoordDim** - 返回几何体的坐度。
 - **ST_CreateTopoGeo** - 将几何形集合添加到定的空拓扑并返回面明成功的消息。
 - **ST_Crosses** - 面个几何形是否有一些（但不是全部）共同的点
 - **ST_CurveN** - 返回复合曲线（CompoundCurve）中的第 N 个曲线几何体。
 - **ST_CurveToLine** - 将包含曲线的几何形面线性几何形。
 - **ST_Difference** - 计算表示几何 A 中不与几何 B 相交的部分的几何。
 - **ST_Dimension** - 返回几何形的拓扑数。
 - **ST_Disjoint** - 面个几何形是否没有共同点
 - **ST_Distance** - 返回个几何或地理之间的距离。
 - **ST_EndPoint** - Returns the last point of a LineString, CircularLineString, or NURBS Curve.
 - **ST_Envelope** - 返回表示几何形界框的几何形。
 - **ST_Equals** - 面个几何形是否包含同一点
 - **ST_ExteriorRing** - 返回表示多边形外部的 LineString。
 - **ST_GMLToSQL** - Returns a geometry from GML. Alias for ST_GeomFromGML.
 - **ST_GeomCollFromText** - 使用定的 SRID 从集合 WKT 中建集合几何体。如果未出 SRID，默 0。
 - **ST_GeomFromText** - 从已知的文本表示 (WKT) 返回指定的 ST_Geometry 。
 - **ST_GeomFromWKB** - 从已知的二进制几何表示 (WKB) 和可的 SRID 建几何例。
 - **ST_GeometryFromText** - Returns a geometry from Well-Known Text (WKT). Alias for ST_GeomFromText.
-

- **ST_GeometryN** - 返回几何集合的一个元素。
- **ST_GeometryType** - 以文本形式返回几何图形的 SQL-MM 型。
- **ST_GetFaceEdges** - 返回一有序的，些边界定了 aface。
- **ST_GetFaceGeometry** - 返回定拓扑中具有指定面 ID 的多形。
- **ST_InitTopoGeo** - 建一个新的拓扑架并将其注册到 topology.topology 表中。
- **ST_InteriorRingN** - 返回多形的第 N 个内 (孔)。
- **ST_Intersection** - 算表示几何 A 和 B 的共享部分的几何。
- **ST_Intersects** - 个几何形是否相交 (它至少有一个共同点)
- **ST_IsClosed** - LineStrings 的起点和点是否重合。于多面体表面是否合 (心)。
- **ST_IsEmpty** - 几何形是否空。
- **ST_IsRing** - 串是合的是的。
- **ST_IsSimple** - 几何体的自完整性或自接触点。
- **ST_IsValid** - 几何形在 2D 中是否有效。
- **ST_Length** - 返回性几何体的二度。
- **ST_LineFromText** - 使用定的 SRID 根据 WKT 表示建几何形。如果未出 SRID, 默 0。
- **ST_LineFromWKB** - Creates a LINESTRING from a WKB representation, optionally using a given SRID.
- **ST_LinestringFromWKB** - Creates a LINESTRING from a WKB representation, optionally using a given SRID.
- **ST_LocateAlong** - 返回几何上与量匹配的点。
- **ST_LocateBetween** - 返回与量范匹配的几何形部分。
- **ST_M** - 返回点的 M 。
- **ST_MLineFromText** - 从 WKT 表示形式返回指定的 ST_MultiLineString 。
- **ST_MPointFromText** - 使用定的 SRID 从 WKT 建几何形。如果未出 SRID, 默 0。
- **ST_MPolyFromText** - 使用定的 SRID 从 WKT 制作多多形几何体。如果未出 SRID, 默 0。
- **ST_ModEdgeHeal** - 通除接条的、修改第一条并除第二条来修复条。返回已除点的 id。
- **ST_ModEdgeSplit** - 通沿有建新点、修改原始并添加新来分割。
- **ST_MoveIsoNode** - 在拓扑中将一个孤立点从一个点移到另一个点。如果新的 apoint 几何象已存在作一个点, 会抛出。返回移的描述。
- **ST_NewEdgeHeal** - 通除接条的、除条并用方向与提供的第一条相同的替它来修复条。
- **ST_NewEdgesSplit** - 通沿有建新点、除原始并用条新替它来分割。返回建的新接新的新点的 id。
- **ST_NumCurves** - 返回复合曲中的曲数量。
- **ST_NumGeometries** - 返回几何集合中的元素数量。
- **ST_NumInteriorRings** - 返回多形的内 (孔) 数。

- **ST_NumPatches** - Returns the number of faces in a PolyhedralSurface or TIN.
 - **ST_NumPoints** - 返回 LineString 或 CircularString 中的点数。
 - **ST_OrderingEquals** - 两个几何图形是否表示相同的几何图形并且具有相同方向顺序的点
 - **ST_Overlaps** - 两个几何图形是否具有相同的维度和相交，但每个几何图形至少有一个点不在另一个几何图形中
 - **ST_PatchN** - Returns the Nth face of a PolyhedralSurface or TIN.
 - **ST_Perimeter** - 返回多边形几何或地理的周界长度。
 - **ST_Point** - 构建具有 X、Y 和 SRID 值的点。
 - **ST_PointFromText** - 使用指定的 SRID 从 WKT 构建点几何图形。如果未给出 SRID，则默认为未知。
 - **ST_PointFromWKB** - 使用指定的 SRID 从 WKB 构建几何图形
 - **ST_PointN** - 返回几何图形中第一个点串或图形串中的第 N 个点。
 - **ST_PointOnSurface** - 计算保证位于多边形或几何体上的点。
 - **ST_Polygon** - 从具有指定 SRID 的点串构建多边形。
 - **ST_PolygonFromText** - 使用指定的 SRID 从 WKT 构建几何图形。如果未给出 SRID，则默认为 0。
 - **ST_Relate** - 两个几何图形是否具有与交集矩阵模式匹配的拓扑关系，或计算它的交集矩阵
 - **ST_RemEdgeModFace** - 删除一条边，如果删除将面分开，则删除一个面并修改一个面以覆盖面的空间。
 - **ST_RemEdgeNewFace** - 删除一条边，如果删除的边将面分开，则删除原始面并用新面替换它。
 - **ST_RemoveIsoEdge** - 删除孤立的边并返回操作的描述。如果边未被隔离，则会引发异常。
 - **ST_RemoveIsoNode** - 删除孤立点并返回操作描述。如果点不是孤立的（是边的开始或结束），则会引发异常。
 - **ST_SRID** - 返回几何图形的空间参考系符。
 - **ST_StartPoint** - Returns the first point of a LineString, CircularLineString, or NURBSCurve.
 - **ST_SymDifference** - 计算表示几何图形 A 和 B 不相交部分的几何图形。
 - **ST_Touches** - 两个几何图形是否至少有一个共同点，但它们的内部不相交
 - **ST_Transform** - 返回坐标在不同空间参考系的新几何图形。
 - **ST_Union** - 计算表示输入几何图形的点集并集的几何图形。
 - **ST_WKBToSQL** - Returns a geometry from Well-Known Binary (WKB). SQL/MM alias for ST_GeomFromWKB without an SRID argument.
 - **ST_WKTToSQL** - Returns a geometry from Well-Known Text (WKT). SQL/MM alias for ST_GeomFromText.
 - **ST_Within** - 检查 A 的每个点是否都位于 B 中，并且它们的内部是否有一个共同点
 - **ST_X** - 返回点的 X 坐标。
 - **ST_Y** - 返回点的 Y 坐标。
 - **ST_Z** - 返回点的 Z 坐标。
-

13.4 PostGIS 地理支持函数

以下列出的函数和操作符是 PostGIS 函数/操作符，它们的输入或输出涉及到 **geography** 数据类型对象。



Note

有 (T) 的函数不是本机大地测量函数，并且使用与几何之的 ST_Transform 用来进行操作。因此，当跨越日期变更线、极点以及覆盖多个 UTM 区域的大型几何形状或几何体时，它们可能不会按期行。基本规则 - (有利于 UTM、假伯特方位角 (北/南)，并在最坏的情况下依靠墨卡托)

- **ST_Area** - 返回多边形几何体的面积。
- **ST_AsBinary** - 返回不带 SRID 元数据的几何/地理的 OGC/ISO 已知的二进制 (WKB) 表示形式。
- **ST_AsEWKT** - 使用 SRID 元数据返回几何形状的已知文本 (WKT) 表示形式。
- **ST_AsGML** - 将几何形状作为 GML 版本 2 或 3 元素返回。
- **ST_AsGeoJSON** - 以 GeoJSON 格式返回一个几何体或要素。
- **ST_AsKML** - 将几何形状作为 KML 元素返回。
- **ST_AsSVG** - 返回几何体的 SVG 路径数据。
- **ST_AsText** - 返回不带 SRID 元数据的几何/地理的已知文本 (WKT) 表示形式。
- **ST_Azimuth** - 返回点之间的直线的基于北方的方位角。
- **ST_Buffer** - 计算覆盖距几何体指定距离内所有点的几何体。
- **ST_Centroid** - 返回几何体的几何中心。
- **ST_ClosestPoint** - 返回 g1 上最接近 g2 的 2D 点。它是从一个几何体到一个几何体的最短直线的第一个点。
- **ST_CoveredBy** - 检查 A 的每个点是否都位于 B 中
- **ST_Covers** - 检查 B 的每个点是否都位于 A 中
- **ST_DWithin** - 检查两个几何形状是否在指定距离内
- **ST_Distance** - 返回两个几何或地理体之间的距离。
- **ST_GeogFromText** - 从已知的文本表示或扩展 (WKT) 返回指定的地理体。
- **ST_GeogFromWKB** - 从已知的二进制几何表示 (WKB) 或扩展的已知的二进制 (EWKB) 构建地理体例。
- **ST_GeographyFromText** - 从已知的文本表示或扩展 (WKT) 返回指定的地理体。
- **=** - 如果几何/地理 A 的坐标和坐标顺序与几何/地理 B 的坐标和坐标顺序相同，返回 TRUE。
- **ST_Intersection** - 计算表示几何 A 和 B 的共享部分的几何。
- **ST_Intersects** - 检查两个几何形状是否相交 (它们至少有一个共同点)
- **ST_Length** - 返回线性几何体的二角度。
- **ST_LineInterpolatePoint** - 返回沿线在百分比指示位置的插点。
- **ST_LineInterpolatePoints** - 返回沿直线以分数间隔插点。
- **ST_LineLocatePoint** - 返回线上最接近点的分数位置。
- **ST_LineSubstring** - 返回一个小数位置之间的直线部分。
- **ST_Perimeter** - 返回多边形几何或地理的周界度。

- **ST_Project** - 返回从起点按距离和方位角（方位角）投影的点。
- **ST_Segmentize** - 返回修改后的几何图形/地理，其线段不处于固定距离。
- **ST_ShortestLine** - 返回两个几何图形之间的 2D 最短线。
- **ST_Summary** - 返回几何内容的文本摘要。
- **<->** - 返回 A 和 B 之间的 2D 距离。
- **&&** - 如果 A 的 2D 边界框与 B 的 2D 边界框相交，则返回 TRUE。

13.5 PostGIS 栅格支持函数

下面列出的函数和运算符是 PostGIS 函数/运算符，它们将 **raster** 数据类型对象作为输入或返回作为输出。按字母序列列出。

- **Box3D** - 返回栅格边界框的 box 3d 表示形式。
- **@** - 如果 A 的边界框包含在 B 的边界框中，则返回 TRUE。使用双精度边界框。
- **~** - 如果 A 的边界框包含 B 的边界框，则返回 TRUE。使用双精度边界框。
- **=** - 如果 A 的边界框与 B 的边界框相同，则返回 TRUE。使用双精度边界框。
- **&&** - 如果 A 的边界框与 B 的边界框相交，则返回 TRUE。
- **&<** - 如果 A 的边界框位于 B 的左方，则返回 TRUE。
- **&>** - 如果 A 的边界框位于 B 的右方，则返回 TRUE。
- **~=** - 如果 A 的边界框与 B 的边界框相同，则返回 TRUE。
- **ST_Retile** - 从任意平面的栅格覆盖范围返回一配置的水平面。
- **ST_AddBand** - 返回一个栅格，其中在固定索引位置添加了固定类型的新波段和固定初始值。如果未指定索引，则将值添加到末尾。
- **ST_AsBinary/ST_AsWKB** - 返回栅格的熟知的二进制 (WKB) 表示形式。
- **ST_AsGDALRaster** - 以指定的 GDAL 栅格格式返回栅格图像。栅格格式是 GDAL 支持的格式之一。使用 **ST_GDALDrivers()** 获取您的支持的格式列表。
- **ST_AsHexWKB** - 返回栅格的十六进制表示形式的熟知的二进制 (WKB)。
- **ST_AsJPEG** - 将栅格指定的波段作为一个复合影像输出 (JPEG) 图像（字节数）返回。如果未指定波段且有 1 个或 3 个以上波段，则使用第一个波段。如果只有 3 个波段，则使用所有 3 个波段并将其映射到 RGB。
- **ST_AsPNG** - 将栅格指定的波段作为一个便携式网络图形 (PNG) 图像（字节数）返回。如果栅格中有 1、3 或 4 个波段且未指定波段，则使用所有波段。如果有 2 个以上或多于 4 个波段且未指定波段，则使用波段 1。波段映射到 RGB 或 RGBA 空间。
- **ST_AsRaster** - 将 PostGIS 几何图形转换为 PostGIS 栅格。
- **ST_AsRasterAgg** - 这是一个聚合函数。将 PostGIS 几何图形转换为新的栅格图像。
- **ST_AsTIFF** - 将栅格指定的波段作为一个 TIFF 图像（字节数）返回。如果未指定波段或栅格中不存在任何指定波段，则将使用所有波段。
- **ST_Aspect** - 返回高程栅格波段的坡向（默认为度单位）。用于分析地形很有用。
- **ST_Band** - 返回具有栅格的一个或多个波段作为新栅格。用于从具有栅格构建新栅格非常有用。

- **ST_BandFileSize** - 返回文件系中存_{in}的波段的文件大小。如果未指定波段号，_{in}假定_{in} 1。
- **ST_BandFileTimestamp** - 返回文件系中存_{in}的波段的文件_{in}戳。如果未指定波段号，_{in}假定_{in} 1。
- **ST_BandIsNoData** - 如果波段_{in}填充无数据_{in}，_{in}返回 true。
- **ST_BandMetaData** - 返回特定_{in}格波段的基本元数据。如果未指定，_{in}假定波段_{in}号_{in} 1。
- **ST_BandNoDataValue** - 返回表示无数据的_{in}定波段中的_{in}。如果没有指定波段_{in}号，默_{in}_{in}波段 1。
- **ST_BandPath** - 返回存_{in}在文件系_{in}中的波段的系_{in}文件路径。如果未指定波段号，_{in}假定_{in} 1。
- **ST_BandPixelType** - 返回_{in}定波段的像素_{in}型。如果未指定波段号，_{in}假定_{in} 1。
- **ST_Clip** - 返回由_{in}入几何_{in}形裁剪的_{in}格。如果未指定波段号，_{in}_{in}理所有波段。如果未指定 crop 或其_{in} TRUE，_{in}_{in}出_{in}格将被裁剪。如果 touched _{in}置_{in} TRUE，_{in}包括接触到的像素；否_{in}，_{in}当像素的中心位于几何_{in}形内_{in}，才会包括_{in}像素。
- **ST_ColorMap** - 根据源_{in}格和指定波段_{in}建最多四个 8BUI 波段（灰度、RGB、RGBA）的新_{in}格。如果未指定，_{in}假定_{in}波段 1。
- **ST_Contains** - 如果_{in}格 rastB 中没有点位于_{in}格 rastA 的外部且 rastB 的内部至少有一个点位于 rastA 的内部，_{in}返回 true。
- **ST_ContainsProperly** - 如果 rastB 与 rastA 的内部相交，但不与 rastA 的_{in}界或外部相交，_{in}返回 true。
- **ST_Contour** - 使用 GDAL _{in}廓算法从提供的_{in}格波段生成一_{in}矢量_{in}廓。
- **ST_ConvexHull** - 返回_{in}格的凸包几何形状，包括等于 BandNoDataValue 的像素_{in}。_{in}于_{in}_{in}形状和非_{in}斜_{in}格，_{in}_{in}出与 ST_Envelope 相同的_{in}果，因此_{in}_{in}不_{in}_{in}形状或_{in}斜_{in}格有用。
- **ST_Count** - 返回_{in}格或_{in}格覆盖范_{in}的_{in}定波段中的像素数。如果未指定 band，_{in}默_{in}_{in} band 1。如果 except_nodata_value _{in}置_{in} true，_{in}_{in}算不等于 nodata _{in}的像素。
- **ST_CountAgg** - _{in}_{in}的。返回一_{in}格的_{in}定波段中的像素数。如果未指定 band，_{in}默_{in}_{in} band 1。如果 except_nodata_value _{in}置_{in} true，_{in}_{in}算不等于 NODATA _{in}的像素。
- **ST_CoveredBy** - 如果_{in}格 rastA 中没有点位于_{in}格 rastB 之外，_{in}返回 true。
- **ST_Covers** - 如果_{in}格 rastB 中没有点位于_{in}格 rastA 之外，_{in}返回 true。
- **ST_DFullyWithin** - 如果_{in}格 rastA 和 rastB 彼此完全在指定距离内，_{in}返回 true。
- **ST_DWithin** - 如果_{in}格 rastA 和 rastB 彼此之_{in}的距离在指定距离内，_{in}返回 true。
- **ST_Disjoint** - 如果_{in}格 rastA 在空_{in}上不与 rastB 相交，_{in}返回 true。
- **ST_DumpAsPolygons** - 从_{in}定的_{in}格_{in}中返回一_{in} geomval (geom,val) 行。如果未指定波段_{in}号，_{in}波段_{in}号默_{in}_{in} 1。
- **ST_DumpValues** - _{in}取指定 band（波段）的_{in}作_{in}二_{in}数_{in}。
- **ST_Envelope** - 返回_{in}格范_{in}的多_{in}形表示形式。
- **ST_FromGDALRaster** - 从受支持的 GDAL _{in}格文件返回_{in}格。
- **ST_GeoReference** - 返回 GDAL 或 ESRI 格式的地理配准元数据，如世界文件中常_{in}的格式。默_{in}_{in} GDAL。
- **ST_Grayscale** - 根据源_{in}格和代表_{in}色、_{in}色和_{in}色的指定波段_{in}建新的 1-8BUI 波段_{in}格
- **ST_HasNoBand** - 如果不存在具有_{in}定波段_{in}号的波段，_{in}返回 true。如果未指定波段号，_{in}假定波段号_{in} 1。
- **ST_Height** - 返回_{in}格的高度（以像素_{in}位_{in}）。
- **ST_HillShade** - 使用提供的方位角、高度、亮度和比例_{in}入返回高程_{in}格_{in}的假_{in}照明。

- **ST_Histogram** - 返回一个数组，数组了栅格或栅格覆盖数据分布，其中包括分离的分箱范围。如果未指定，将自动计算分箱数。
- **ST_InterpolateRaster** - 基于输入的 3 点集插值网格表面，使用 X 和 Y 在网格上定位点，并使用点的 Z 值作表面高程。
- **ST_Intersection** - 返回一个栅格或一个几何像素数组，表示两个栅格的共享部分或栅格矢量化和几何图形的几何交集。
- **ST_IntersectionFractions** - 计算给定 geometry 覆盖到每个 raster 像元的比例。
- **ST_Intersects** - 如果栅格 rastA 与栅格 rastB 空间相交，返回 true。
- **ST_IsEmpty** - 如果栅格为空（宽度 = 0 且高度 = 0），返回 true。否则，返回 false。
- **ST_MakeEmptyCoverage** - 用空栅格覆盖地理参考区域。
- **ST_MakeEmptyRaster** - 返回一个空白栅格（不包含波段），其具有指定的尺寸（宽度和高度）、左上角 X 和 Y 坐标、像素大小和旋转参数（scalex、scaley、skewx 和 skewy），以及参考系（SRID）。如果提供了一个栅格，返回一个具有相同大小、旋转方式和 SRID 的新栅格。如果省略了 SRID，空参考将置为未知（0）。
- **ST_MapAlgebra (callback function version)** - 回调函数版本 - 指定一个或多个输入栅格、波段索引和一个用指定的回调函数，返回波段栅格。
- **ST_MapAlgebraExpr - 1** 栅格波段版本：通过提供的输入栅格波段和提供的像素类型使用有效的 PostgreSQL 代数算来构建新的波段栅格。如果未指定波段，假定波段 1。
- **ST_MapAlgebraExpr - 2** 栅格波段版本：通过提供的两个输入栅格波段和像素类型使用有效的 PostgreSQL 代数算来构建新的波段栅格。如果未指定波段号，假定每个栅格的波段 1。生成的栅格将在第一个栅格定义的网格上（比例、斜角和像素角），并具有由“extenttype”参数定义的范。 “extenttype”的值可以是：INTERSECTION、UNION、FIRST、SECOND。
- **ST_MapAlgebraFct - 1** 波段版本 - 通过在输入栅格波段和提供的像素类型上使用有效的 PostgreSQL 函数来构建新的波段栅格。如果未指定波段，假定波段 1。
- **ST_MapAlgebraFct - 2** 波段版本 - 通过在提供的 2 个输入栅格波段和像素类型上使用有效的 PostgreSQL 函数来构建新的波段栅格。如果未指定波段，假定波段 1。如果未指定，范值默认为 INTERSECTION。
- **ST_MapAlgebraFctNgb - 1**-波段版本：使用指定的 PostgreSQL 函数映射代数最近邻。返回一个栅格，其是涉及输入栅格波段的区域的 PLPGSQL 用函数的结果。
- **ST_MapAlgebra (expression version)** - 表达式版本 - 在指定一个输入栅格、波段索引和一个或多个用指定的 SQL 表达式式的情况下，返回波段栅格。
- **ST_MemSize** - 返回栅格占用的空间量（以字节为单位）。
- **ST_MetaData** - 返回有关栅格对象的基本元数据，例如像素大小、旋转（斜角）、左上、左下等。
- **ST_MinConvexHull** - 返回栅格的凸包几何形状（不包括 NODATA 像素）。
- **ST_NearestValue** - 返回由 columnx 和 rowy 指定的指定像素的最接近的非 NODATA 或以与栅格相同的空间参考坐标系表示的几何点。
- **ST_Neighborhood** - 返回指定波段像素周围非 NODATA 值的二值双精度数组，指定波段像素由 columnX 和 rowY 或以与栅格相同的空间参考坐标系表示的几何点指定。
- **ST_NotSameAlignmentReason** - 返回说明栅格是否不同的文本，如果未返回，说明原因。
- **ST_NumBands** - 返回栅格对象中的波段数。
- **ST_Overlaps** - 如果栅格 rastA 和 rastB 相交，但其中一个不完全包含另一个，返回 true。
- **ST_PixelAsCentroid** - 返回像素表示的区域的中心（点几何）。
- **ST_PixelAsCentroids** - 返回栅格波段的每个像素的中心（点几何）以及每个像素的列、X 和 Y 栅格坐标。点几何是像素表示的区域的中心。

- **ST_PixelAsPoint** - 返回像素左上角的点几何形状。
- **ST_PixelAsPoints** - 返回栅格波段的每个像素的点几何形状以及每个像素的 X、Y 栅格坐标。点几何的坐标是像素的左上角。
- **ST_PixelAsPolygon** - 返回限定特定行和列的像素的多边形几何形状。
- **ST_PixelAsPolygons** - 返回包围栅格的每个像素的多边形几何形状以及每个像素的 X、Y 栅格坐标。
- **ST_PixelHeight** - 返回空参考系中的几何位置的像素高度。
- **ST_PixelOfValue** - 返回等于搜索值的像素的列 x、行 y 坐标。
- **ST_PixelWidth** - 返回空参考系中的几何位置的像素宽度。
- **ST_Polygon** - 返回由具有非空数据的像素的像素并集形成的多边形几何体。如果未指定波段号，波段号默认为 1。
- **ST_Quantile** - 计算本或体上下文中栅格或栅格表覆盖范围的分位数。因此，可以判断某个值是否位于栅格的 25%、50%、75% 百分位。
- **ST_RastFromHexWKB** - 从熟知的二进制 (WKB) 栅格的十六进制表示形式返回栅格。
- **ST_RastFromWKB** - 从熟知的二进制 (WKB) 栅格返回栅格。
- **ST_RasterToWorldCoord** - 在给定列和行的情况下，以几何 X 和 Y (经度和纬度) 形式返回栅格的左上角。列和行从 1 开始。
- **ST_RasterToWorldCoordX** - 返回栅格、列和行左上角的几何 X 坐标。列和行的号从 1 开始。
- **ST_RasterToWorldCoordY** - 返回栅格、列和行的左上角的几何 Y 坐标。列和行的号从 1 开始。
- **ST_Reclass** - 构建由从原始数据重新分区的波段类型形成的新栅格。nband 是要更改的波段。如果未指定 nband，假定 1。所有其他波段均按原样返回。使用案例：将 16BUI 波段转换为 8BUI 等，以便更好地呈现可格式。
- **ST_ReclassExact** - 构建一个新栅格，其波段通过原始波段行 1:1 映射重新分区生成，将原波段中的值映射到目标波段中的新值。
- **ST_Resample** - 重新采样一个栅格图像，可以指定重新采样算法、新的尺寸、任意的栅格角点，以及一维栅格地理参考属性，这些属性可以自己定义，也可以从一个栅格图像中借用。
- **ST_Rescale** - 通过调整栅格的比例（或像素大小）来重新采样栅格。新的像素值是使用 NearestNeighbor (英国或美式拼写)、Bilinear、Cubic、CubicSpline、Lanczos、Max 或 Min 重新采样算法计算的。默认为 NearestNeighbor。
- **ST_Resize** - 将栅格大小调整新的宽度/高度。
- **ST_Reskew** - 通过调整栅格的倾斜（或旋转参数）来重新采样栅格。新的像素值是使用 NearestNeighbor (英国或美式拼写)、Bilinear、Cubic、CubicSpline 或 Lanczos 重新采样算法计算的。默认为 NearestNeighbor。
- **ST_Rotation** - 返回栅格的旋转弧度。
- **ST_Roughness** - 返回一个计算出的数字高程模型 (DEM) 的“粗糙度”的栅格。
- **ST_SRID** - 返回在 spatial_ref_sys 表中定义的栅格的空参考系符号。
- **ST_SameAlignment** - 如果栅格具有相同的倾斜、比例、空参考系和偏移（像素可以放在同一网格上而不切割成像素），返回 true；如果没有注意到这些，返回 false。
- **ST_ScaleX** - 返回像素宽度的 X 分量（以坐标参考系位置）。
- **ST_ScaleY** - 返回像素高度的 Y 分量（以坐标参考系位置）。
- **ST_SetBandIndex** - 更新 out-db band 的 external band 编号。

- **ST_SetBandIsNoData** - 将栅格的 isnodata 标志置为 TRUE。
- **ST_SetBandNoDataValue** - 设置代表无数据的固定波段的值。如果未指定波段，假定波段 1。要将波段值设置为 no data 值，设置 no data 值 = NULL。
- **ST_SetBandPath** - 更新 out-db band 的外部路径和 band 编号。
- **ST_SetGeoReference** - 在一次调用中设置 Georeference 6 地理配准参数。数字值用空格分隔。接受 GDAL 或 ESRI 格式的输入。默认为 GDAL。
- **ST_SetM** - 返回与输入几何图形具有相同 X/Y 坐标的几何图形，并使用请求的重采样算法将栅格中的值复制到 M 度。
- **ST_SetRotation** - 以弧度设置栅格的旋角。
- **ST_SetSRID** - 将栅格的 SRID 设置在 Spatial_ref_sys 表中定义的特定整数 srid。
- **ST_SetScale** - 以坐标参考系单位设置像素的 X 和 Y 大小。数字值/像素度/高度。
- **ST_SetSkew** - 设置地理参考 X 和 Y 斜（或旋角参数）。如果只输入一个，将 X 和 Y 设置为相同的值。
- **ST_SetUpperLeft** - 将栅格左上角的值设置投影的 X 和 Y 坐标。
- **ST_SetValue** - 返回修改后的栅格，其结果是将固定波段中的值设置指定列 x、行 y 像素或与特定几何图形相交的像素。波段编号从 1 开始，如果未指定波段，默认为 1。
- **ST_SetValues** - 返回通设置固定波段的值而生的修改后的栅格。
- **ST_SetZ** - 返回与输入几何图形具有相同 X/Y 坐标的几何图形，并使用请求的重采样算法将栅格中的值复制到 Z 度。
- **ST_SkewX** - 返回地理参考 X 斜（或旋角参数）。
- **ST_SkewY** - 返回地理参考 Y 斜（或旋角参数）。
- **ST_Slope** - 返回高程栅格的坡度（默认为度单位）。用于分析地形很有用。
- **ST_SnapToGrid** - 通常将栅格捕捉到网格来重采样栅格。新的像素值是使用 NearestNeighbor（英语或美式拼写）、Bilinear、Cubic、CubicSpline 或 Lanczos 重采样算法计算的。默认为 NearestNeighbor。
- **ST_Summary** - 返回栅格内容的文本摘要。
- **ST_SummaryStats** - 返回指定栅格或栅格覆盖中某个波段的值摘要，包括像元数量（count）、和（sum）、均值（mean）、标准差（stddev）、最小值（min）和最大值（max）。若未指定波段，默认使用第 1 波段。
- **ST_SummaryStatsAgg** - 聚合函数。返回一栅格中指定波段的值摘要，包括像元数量（count）、和（sum）、均值（mean）、标准差（stddev）、最小值（min）和最大值。若未指定波段，默认使用第 1 波段。
- **ST_TPI** - 返回一个计算出的地形位置指数（Topographic Position Index）的栅格。
- **ST_TRI** - 返回具有计算的地形固性指数的栅格。
- **ST_Tile** - 返回根据输出栅格的所需度分割输入栅格而生的一栅格。
- **ST_Touches** - 如果栅格 rastA 和 rastB 至少有一个共同点但它内部不相交，返回 true。
- **ST_Transform** - 使用指定的重采样算法将已知空参考系中的栅格重新投影到一个已知空参考系。值有 NearestNeighbor、Bilinear、Cubic、CubicSpline、Lanczos（默认为 NearestNeighbor）。
- **ST_Union** - 将一栅格切片的并集返回由 1 个或多个波段组成的一个栅格。
- **ST_UpperLeftX** - 返回投影空参考中栅格的左上角 X 坐标。
- **ST_UpperLeftY** - 返回投影空参考中栅格的左上角 Y 坐标。

- **ST_Value** - 返回指定列 x、行 y 像素或特定几何点指定波段的值。波段号从 1 开始，如果未指定，默认 1。如果将参数 `exclude_nodata_value` 置为 `false`，所有像素都被与 `nodata` 像素相交并返回其值。如果未设置参数 `exclude_nodata_value`，从栅格的元数据中取值。
- **ST_ValueCount** - 返回一行值，其中包含像素值以及具有指定值的栅格（或栅格覆盖范围）指定中的像素数。如果未指定波段，默认波段 1。默认情况下，不算点数据像素。输出像素中的所有其他值，并将像素值四舍五入到最接近的整数。
- **ST_Width** - 返回栅格的宽度（以像素单位）。
- **ST_Within** - 如果栅格 `rastA` 中没有点位于栅格 `rastB` 的外部且 `rastA` 的内部至少有一个点位于 `rastB` 的内部，返回 `true`。
- **ST_WorldToRasterCoord** - 指定几何 X 和 Y（经度和纬度）或以栅格的空参考系表示的点几何，将左上角作列和行返回。
- **ST_WorldToRasterCoordX** - 返回栅格中点几何 (pt) 的列或以栅格世界空参考系表示的 X 和 Y 世界坐标 (xw, yw)。
- **ST_WorldToRasterCoordY** - 返回点几何几何 (pt) 的栅格中的行或以栅格的世界空参考系表示的 X 和 Y 世界坐标 (xw, yw)。
- **UpdateRasterSRID** - 更改用指定列和表中所有栅格的 SRID。

13.6 PostGIS 几何/地理/栅格输出函数

下面输出的函数是 PostGIS 函数，它将一或多个 **Geometry_dump** 或 **geomval** 数据类型输入或返回作输出。

- **ST_DumpAsPolygons** - 从指定的栅格中返回一行 **geomval** (geom,val) 行。如果未指定波段号，波段号默认 1。
- **ST_Intersection** - 返回一个栅格或一行几何像素值，表示两个栅格的共享部分或栅格矢量化和几何形状的几何交集。
- **ST_Dump** - 返回几何文件的一行 **geometry_dump** 行。
- **ST_DumpPoints** - 返回几何形状中坐标的一行 **geometry_dump** 行。
- **ST_DumpRings** - 返回多边形外环和内环的一行 **geometry_dump** 行。
- **ST_DumpSegments** - 几何形状中的各个线段返回一行 **geometry_dump** 行。

13.7 PostGIS 边界框函数

下面输出的函数是 PostGIS 函数，它将 **Box*** 系列 PostGIS 空数据类型作输入或返回作输出。box 系列类型由 **box2d** 和 **box3d** 组成。

- **Box2D** - 返回表示几何形状的 2D 范围的 BOX2D。
- **Box3D** - 返回表示几何体 3D 范围的 BOX3D。
- **MakeTopologyPrecise** - 将拓扑点值到精度网格。
- **Box3D** - 返回栅格封框的 box 3d 表示形式。
- **ST_3DExtent** - 返回几何形状的 3D 边界框的聚合函数。

- **ST_3DMakeBox** - 构建由 3D 点几何体定义的 BOX3D。
 - **ST_AsMVTGeom** - 将几何体转换为 MVT 瓦片的坐标空间。
 - **ST_AsTWKB** - 返回几何体为 TWKB，又名“微小的已知的二进制”
 - **ST_Box2dFromGeoHash** - 从 GeoHash 字符串返回 BOX2D。
 - **ST_ClipByBox2D** - 计算几何体落在矩形内的部分。
 - **ST_EstimatedExtent** - 返回空表的估计范围。
 - **ST_Expand** - 返回从一个边界框或几何体扩展的边界框。
 - **ST_Extent** - 返回几何体边界框的聚合函数。
 - **ST_MakeBox2D** - 构建由 2D 点几何体定义的 BOX2D。
 - **ST_RemoveIrrelevantPointsForView** - 删除在指定矩形表示范围内与视图无关的几何体点。
 - **ST_XMax** - 返回 2D 或 3D 边界框或几何体的 X 最大值。
 - **ST_XMin** - 返回 2D 或 3D 边界框或几何体的 X 最小值。
 - **ST_XSize** - Returns the X size of a 2D or 3D bounding box or a geometry.
 - **ST_YMax** - 返回 2D 或 3D 边界框或几何体的 Y 最大值。
 - **ST_YMin** - 返回 2D 或 3D 边界框或几何体的 Y 最小值。
 - **ST_YSize** - Returns the Y size of a 2D or 3D bounding box or a geometry.
 - **ST_ZMax** - 返回 2D 或 3D 边界框或几何体的 Z 最大值。
 - **ST_ZMin** - 返回 2D 或 3D 边界框或几何体的 Z 最小值。
 - **ST_ZSize** - Returns the Z size of a 2D or 3D bounding box or a geometry.
 - **RemoveUnusedPrimitives** - 删除定义 TopoGeometry 对象不需要的拓扑基元。
 - **ValidateTopology** - 返回一个 validate_topology_return_type 对象，明确拓扑。
 - **ValidateTopologyPrecision** - 返回拓扑中不精确的点。
 - **~(box2df,box2df)** - 如果 2D 浮点精度边界框 (BOX2DF) 包含一个 2D 浮点精度边界框 (BOX2DF)，返回 TRUE。
 - **~(box2df,geometry)** - 如果 2D 浮点精度边界框 (BOX2DF) 包含几何体的 2D 边界框，返回 TRUE。
 - **~(geometry,box2df)** - 如果几何体的 2D 边界框包含 2D 浮点精度边界框 (BOX2DF)，返回 TRUE。
 - **@(box2df,box2df)** - 如果 2D 浮点精度边界框 (BOX2DF) 包含在一个 2D 浮点精度边界框内，返回 TRUE。
 - **@(box2df,geometry)** - 如果 2D 浮点精度边界框 (BOX2DF) 包含在几何体的 2D 边界框中，返回 TRUE。
 - **@(geometry,box2df)** - 如果几何体的 2D 边界框包含在 2D 浮点精度边界框 (BOX2DF) 中，返回 TRUE。
 - **&&(box2df,box2df)** - 如果两个 2D 浮点精度边界框 (BOX2DF) 彼此相交，返回 TRUE。
 - **&&(box2df,geometry)** - 如果 2D 浮点精度边界框 (BOX2DF) 与几何体的（存在的）2D 边界框相交，返回 TRUE。
 - **&&(geometry,box2df)** - 如果几何体的（存在的）2D 边界框与 2D 浮点精度边界框 (BOX2DF) 相交，返回 TRUE。
-

13.8 支持 3D 的 PostGIS 函数

下面列出的函数是不会丢弃 Z-Index 的 PostGIS 函数。

- **AddGeometryColumn** - 将 geometry（几何）列添加到现有表。
- **Box3D** - 返回表示几何体 3D 范围的 BOX3D。
- **CG_3DAlphaWrapping** - 计算一个网格包络指定几何的 3D Alpha 包络。
- **CG_3DArea** - 计算 3D 表面几何形状的面积。对于固体将返回 0。
- **CG_3DConvexHull** - 计算几何体的 3D 凸包。
- **CG_3DDifference** - 执行 3D 差异
- **CG_3DIntersection** - 执行 3D 相交
- **CG_3DRotate** - 输入几何体 geom 会绕指定的向量执行三维旋转。旋转角度由参数 angle（以弧度为单位）指定，旋转将在三维空间内执行。
- **CG_3DScale** - 将几何体沿 X、Y、Z 轴分别按指定的不同系数进行缩放。
- **CG_3DScaleAroundCenter** - 将几何体在三维空间中，以指定的点为中心进行缩放。
- **CG_3DTranslate** - 将输入几何体按照指定的偏移量在三维空间中移动。
- **CG_3DUnion** - 使用 postgis_sfcgal 执行三维合并。
- **CG_ConstrainedDelaunayTriangles** - 返回指定输入几何体的约束 Delaunay 三角剖分。
- **CG_Extrude** - 将曲面输出到相关体
- **CG_ForceLHR** - 强制 LHR 方向
- **CG_IsPlanar** - 检查表面是否平坦
- **CG_IsSolid** - 检查几何体是否是固体。不执行有效性检查。
- **CG_MakeSolid** - 将几何体造成固体。不执行任何检查。要获得有效的固体，输入几何体必须是闭合多面体曲面或符合 TIN。
- **CG_NurbsCurveApproximate** - Creates an approximating NURBS curve fitting data points within a tolerance
- **CG_NurbsCurveDerivative** - Computes the derivative (tangent, curvature) of a NURBS curve at a given parameter
- **CG_NurbsCurveInterpolate** - Creates an interpolating NURBS curve passing through all given data points
- **CG_Orientation** - 确定表面方向
- **CG_RotateX** - 将几何体绕 X 轴旋转指定的角度。
- **CG_RotateY** - 将几何体绕 Y 轴旋转指定的角度。
- **CG_RotateZ** - 将几何体绕 Z 轴旋转指定的角度。
- **CG_Simplify** - 在保留几何体关键特征以及 Z/M 值的前提下，降低其复杂度。
- **CG_Tessellate** - 将多边形或多面体表面进行曲面细分，并以 TIN 或 TINS 集合的形式返回
- **CG_Visibility** - 计算一个从点或多边形几何中的线段生成的可见性多边形
- **CG_Volume** - Computes the volume of a 3D geometry. Closed surfaces can have non-zero volume.

- **DropGeometryColumn** - 从空表中移除 geometry (几何) 列。
- **GeometryType** - 以文本形式返回几何的类型。
- **ST_3DClosestPoint** - 返回 g1 上最接近 g2 的 3D 点。是 3D 最短的第一个点。
- **ST_3DDFullyWithin** - 两个 3D 几何形是否完全在指定的 3D 距离内
- **ST_3DDWithin** - 两个 3D 几何形是否在指定的 3D 距离内
- **ST_3DDistance** - 返回两个几何形之间的 3D 笛卡儿最小距离 (基于空参考) (以投影位表示)。
- **ST_3DExtent** - 返回几何形的 3D 边框的聚合函数。
- **ST_3DInterpolatePoint** - Returns the interpolated measure of a geometry closest to a point in 3D.
- **ST_3DIntersects** - 两个几何形在 3D 空间中是否相交 - 适用于点、串、多边形、多面体曲面 (区域)
- **ST_3DLength** - 返回线性几何体的 3D 长度。
- **ST_3DLineInterpolatePoint** - 返回沿 3D 线的小数指示位置插值的点。
- **ST_3DLongestLine** - 返回两个几何体之间的 3D 最长直线
- **ST_3DMaxDistance** - 返回两个几何形之间的 3D 笛卡儿最大距离 (基于空参考) (以投影位表示)。
- **ST_3DPerimeter** - 返回多边形几何体的 3D 周长。
- **ST_3DShortestLine** - 返回两个几何形之间的 3D 最短线
- **ST_AddMeasure** - 沿线性几何形状插值测量。
- **ST_AddPoint** - 将点添加到串 (LineString)。
- **ST_Affine** - 几何体用 3D 仿射。
- **ST_AsBinary** - 返回不带 SRID 元数据的几何/地理的 OGC/ISO 已知的二进制 (WKB) 表示形式。
- **ST_AsEWKB** - 返回带有 SRID 元数据的几何形的扩展已知的二进制 (EWKB) 表示形式。
- **ST_AsEWKT** - 使用 SRID 元数据返回几何形的已知文本 (WKT) 表示形式。
- **ST_AsGML** - 将几何形作为 GML 版本 2 或 3 元素返回。
- **ST_AsGeoJSON** - 以 GeoJSON 格式返回一个几何体或要素。
- **ST_AsHEXEWKB** - 使用小端 (NDR) 或大端 (XDR) 返回 HEXEWKB 格式 (作为文本) 的几何形。
- **ST_AsKML** - 将几何形作为 KML 元素返回。
- **ST_AsX3D** - 返回 X3D xml 点元素格式的几何形 : ISO-IEC-19776-1.2-X3DEncodings-XML
- **ST_Boundary** - 返回几何形的边界。
- **ST_BoundingDiagonal** - 返回几何边框的对角线。
- **ST_CPAWithin** - 两条轨迹的最近接近点是否在指定距离内。
- **ST_CatmullRomSmoothing** - Returns a smoothed version of a geometry, using the Catmull-Rom spline algorithm
- **ST_ChaikinSmoothing** - 使用 Chaikin 算法返回几何形的平滑版本
- **ST_ClosestPointOfApproach** - 返回两条轨迹最接近点的测量。
- **ST_Collect** - 从一个几何形构建 GeometryCollection 或 Multi* 几何形。
- **ST_ConvexHull** - 计算几何体的凸包。

- **ST_CoordDim** - 返回几何体的坐度。
 - **ST_CurveN** - 返回复合曲线（CompoundCurve）中的第 N 个曲线几何体。
 - **ST_CurveToLine** - 将包含曲线的几何图形转换为线性几何图形。
 - **ST_DelaunayTriangles** - 返回几何体点的 Delaunay 三角剖分。
 - **ST_Difference** - 运算符表示几何 A 中不与几何 B 相交的部分的几何。
 - **ST_DistanceCPA** - 返回多条轨迹的最近接近点之间的距离。
 - **ST_Dump** - 返回几何物件的一行 geometry_dump 行。
 - **ST_DumpPoints** - 返回几何图形中坐度的一行 geometry_dump 行。
 - **ST_DumpRings** - 返回多边形外环和内环的一行 geometry_dump 行。
 - **ST_DumpSegments** - 几何图形中的各个线段返回一行 geometry_dump 行。
 - **ST_EndPoint** - Returns the last point of a LineString, CircularLineString, or NURBSCurve.
 - **ST_ExteriorRing** - 返回表示多边形外环的 LineString。
 - **ST_FlipCoordinates** - 返回 X 和 Y 坐标的几何图形版本。
 - **ST_Force2D** - 强制几何图形进入“二维模式”。
 - **ST_ForceCurve** - 如果适用，将一个几何图形上升到其曲线型。
 - **ST_ForcePolygonCCW** - 将所有外环逆时针定向，将所有内环顺时针定向。
 - **ST_ForcePolygonCW** - 顺时针方向调整所有外环，逆时针方向调整所有内环。
 - **ST_ForceRHR** - 强制多边形点的方向遵循右手定则。
 - **ST_ForceSFS** - Force geometries to use SFS 1.1 or 1.2 geometry types.
 - **ST_Force3D** - 强制几何图形进入 XYZ 模式。是 ST_Force3DZ 的别名。
 - **ST_Force3DZ** - 强制几何图形进入 XYZ 模式。
 - **ST_Force4D** - 强制几何图形进入 XYZM 模式。
 - **ST_ForceCollection** - 将几何图形转换为几何集合（GEOMETRYCOLLECTION）。
 - **ST_GeomFromEWKB** - 从扩展已知的二进制表示（EWKB）返回指定的 ST_Geometry。
 - **ST_GeomFromEWKT** - 从扩展已知的文本表示（EWKT）返回指定的 ST_Geometry。
 - **ST_GeomFromGML** - 将几何图形的 GML 表示形式作输入并输出 PostGIS 几何对象
 - **ST_GeomFromGeoJSON** - 将几何图形的 geojson 表示形式作输入并输出 PostGIS 几何对象
 - **ST_GeomFromKML** - 将几何图形的 KML 表示形式作输入并输出 PostGIS 几何对象
 - **ST_GeometricMedian** - 返回多点的几何中位数。
 - **ST_GeometryN** - 返回几何集合的一个元素。
 - **ST_GeometryType** - 以文本形式返回几何图形的 SQL-MM 型。
 - **ST_HasArc** - 几何图形是否包含弧
 - **ST_HasM** - 几何体是否具有 M（量）度。
 - **ST_HasZ** - 几何体是否具有 Z 度。
 - **ST_InteriorRingN** - 返回多边形的第 N 个内环（孔）。
-

- **ST_InterpolatePoint** - 返回最接近点的几何图形的插值量。
 - **ST_Intersection** - 计算表示几何 A 和 B 的共享部分的几何。
 - **ST_IsClosed** - 检查 LineStrings 的起点和点是否重合。用于多面体表面是否闭合（闭心）。
 - **ST_IsCollection** - 检查几何类型是否是几何集合。
 - **ST_IsPolygonCCW** - 检查多边形是否具有逆时针方向的外环和顺时针方向的内环。
 - **ST_IsPolygonCW** - 检查多边形是否具有顺时针外环和逆时针内环。
 - **ST_IsSimple** - 检查几何体的自完整性或自接触点。
 - **ST_IsValidTrajectory** - 检查几何图形是否是有效轨迹。
 - **ST_LengthSpheroid** - 返回球体上长度/度几何体的 2D 或 3D 长度/周度。
 - **ST_LineFromMultiPoint** - 从多点几何构建线串。
 - **ST_LineInterpolatePoint** - 返回沿线在百分比指示位置的插值点。
 - **ST_LineInterpolatePoints** - 返回沿直线以分数间隔插值的点。
 - **ST_LineSubstring** - 返回一个小数位置之间的直线部分。
 - **ST_LineToCurve** - 将线性几何图形转换为曲线几何图形。
 - **ST_LocateBetweenElevations** - 返回位于高程 (Z) 范围内的几何图形部分。
 - **ST_M** - 返回点的 M 值。
 - **ST_MMax** - Returns the M maxima of a geometry.
 - **ST_MMin** - 返回几何图形的 M 值最小值。
 - **ST_MSize** - Returns the M size of a geometry.
 - **ST_MakeLine** - 从 Point, MultiPoint, 或 LineString geometries 构建 LineString。
 - **ST_MakePoint** - 构建 2D、3DZ 或 4D 点。
 - **ST_MakePolygon** - Creates a Polygon or CurvePolygon from a shell and optional list of holes.
 - **ST_MakeValid** - 检查在不丢失点的情况下使无效几何体有效。
 - **ST_MemSize** - 返回几何图形占用的内存空间量。
 - **ST_MemUnion** - 聚合函数，以内存高效但速度较慢的方式合并几何图形。
 - **ST_NDims** - 返回几何体的坐标度。
 - **ST_NPoints** - 返回几何图形中的点数（点）。
 - **ST_NRings** - 返回多边形几何中的环数。
 - **ST_Node** - 断点是集合。
 - **ST_NumCurves** - 返回复合曲线中的曲线数量。
 - **ST_NumGeometries** - 返回几何集合中的元素数量。
 - **ST_NumPatches** - Returns the number of faces in a PolyhedralSurface or TIN.
 - **ST_PatchN** - Returns the Nth face of a PolyhedralSurface or TIN.
 - **ST_PointFromWKB** - 使用指定的 SRID 从 WKB 构建几何图形。
 - **ST_PointN** - 返回几何图形中第一个点串或图形点串中的第 N 个点。
-

- **ST_PointOnSurface** - 计算保证位于多边形或几何体上的点。
 - **ST_Points** - 返回包含几何坐标的 MultiPoint。
 - **ST_Polygon** - 从具有指定 SRID 的字符串构建多边形。
 - **ST_RemovePoint** - 从字符串中删除一个点。
 - **ST_RemoveRepeatedPoints** - 返回除了重复点的几何图形。
 - **ST_Reverse** - 返回点序相反的几何体。
 - **ST_Rotate** - 绕原点旋转几何体。
 - **ST_RotateX** - 绕 X 轴旋转几何体。
 - **ST_RotateY** - 绕 Y 轴旋转几何体。
 - **ST_RotateZ** - 绕 Z 轴旋转几何体。
 - **ST_Scale** - 按指定因子缩放几何图形。
 - **ST_Scroll** - 更改复合字符串的起点。
 - **ST_SetPoint** - 用指定点替换字符串的点。
 - **ST_ShiftLongitude** - 在 -180-180 和 0-360 之间移动几何图形的经度坐标。
 - **ST_SnapToGrid** - 将输入几何体的所有点捕捉到指定网格。
 - **ST_StartPoint** - Returns the first point of a LineString, CircularLineString, or NURBSCurve.
 - **ST_SwapOrdinates** - 返回更改后的几何图形，其中交换了指定的坐标。
 - **ST_SymDifference** - 计算表示几何图形 A 和 B 不相交部分的几何图形。
 - **ST_TransScale** - 按指定的偏移量和系数平移和缩放几何图形。
 - **ST_Translate** - 按指定的偏移量平移几何图形。
 - **ST_UnaryUnion** - 计算多个几何体的并集。
 - **ST_Union** - 计算表示输入几何图形的点集并集的几何图形。
 - **ST_WrapX** - 将几何体限制在 X 轴周围。
 - **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** - 返回 2D 或 3D 边界框或几何体的 X 最小值。
 - **ST_XSize** - Returns the X size of a 2D or 3D bounding box or a geometry.
 - **ST_Y** - 返回点的 Y 坐标。
 - **ST_YMax** - 返回 2D 或 3D 边界框或几何体的 Y 最大值。
 - **ST_YMin** - 返回 2D 或 3D 边界框或几何体的 Y 最小值。
 - **ST_YSize** - Returns the Y size of a 2D or 3D bounding box or a geometry.
 - **ST_Z** - 返回点的 Z 坐标。
 - **ST_ZMFlag** - 返回指示几何体的 ZM 坐标度的代码。
 - **ST_ZMax** - 返回 2D 或 3D 边界框或几何体的 Z 最大值。
 - **ST_ZMin** - 返回 2D 或 3D 边界框或几何体的 Z 最小值。
-

- **ST_ZSize** - Returns the Z size of a 2D or 3D bounding box or a geometry.
- **Equals** - 如果两个拓扑几何由相同的拓扑基元组成，返回 true。
- **Intersects** - 如果两个拓扑几何中的任何一元素相交，返回 true。
- **UpdateGeometrySRID** - 更新几何列中所有要素的 SRID 以及表元数据。
- **&&&** - 如果 A 的 n 元素边界与 B 的 n 元素边界相交，返回 TRUE。
- **&&&(geometry,gidx)** - 如果几何体的 (存在的) n 元素边界与 n 浮点精度边界 (GIDX) 相交，返回 TRUE。
- **&&&(gidx,geometry)** - 如果 n 浮点精度边界 (GIDX) 与几何体的 (存在的) n 元素边界相交，返回 TRUE。
- **&&&(gidx,gidx)** - 如果两个 n 浮点精度边界 (GIDX) 彼此相交，返回 TRUE。

13.9 PostGIS 曲线几何支持函数

下面列出的函数是 PostGIS 函数，可以使用 CIRCULARSTRING、CURVEPOLYGON 和其他曲线几何类型

- **AddGeometryColumn** - 将 geometry (几何) 列添加到现有表。
- **Box2D** - 返回表示几何形状的 2D 范围的 BOX2D。
- **Box3D** - 返回表示几何体 3D 范围的 BOX3D。
- **DropGeometryColumn** - 从空表中移除 geometry (几何) 列。
- **GeometryType** - 以文本形式返回几何的类型。
- **PostGIS_AddBBox** - 向几何体添加边界框。
- **PostGIS_DropBBox** - 从几何体中删除边界框。
- **PostGIS_HasBBox** - 如果几何体的 bbox 已存在，返回 TRUE，否则返回 FALSE。
- **ST_3DExtent** - 返回几何形状的 3D 边界的聚合函数。
- **ST_Affine** - 几何体用 3D 仿射。
- **ST_AsBinary** - 返回不带 SRID 元数据的几何/地理的 OGC/ISO 已知的二进制 (WKB) 表示形式。
- **ST_AsEWKB** - 返回带有 SRID 元数据的几何形状的扩展已知的二进制 (EWKB) 表示形式。
- **ST_AsEWKT** - 使用 SRID 元数据返回几何形状的已知文本 (WKT) 表示形式。
- **ST_AsHEXEWKB** - 使用小端 (NDR) 或大端 (XDR) 返回 HEXEWKB 格式 (作文本) 的几何形状。
- **ST_AsSVG** - 返回几何体的 SVG 路径数据。
- **ST_AsText** - 返回不带 SRID 元数据的几何/地理的已知文本 (WKT) 表示形式。
- **ST_ClusterDBSCAN** - 使用 DBSCAN 算法返回每个输入几何形状的簇 id 的窗口函数。
- **ST_ClusterWithin** - 按间隔距离几何形状行聚合的聚合函数。
- **ST_ClusterWithinWin** - 窗口函数，返回每个输入几何形状的簇 ID，使用分离距离行聚合。
- **ST_Collect** - 从一个几何形状构建 GeometryCollection 或 Multi* 几何形状。
- **ST_CoordDim** - 返回几何体的坐标度。
- **ST_CurveToLine** - 将包含曲线的几何形状转换为线性几何形状。
- **ST_Distance** - 返回两个几何或地理之间的距离。

- **ST_Dump** - 返回几何物件的一行 geometry_dump 行。
 - **ST_DumpPoints** - 返回几何图形中坐点的一行 geometry_dump 行。
 - **ST_EndM** - Returns the M coordinate of the last point of a LineString, CircularLineString, or NURBS-Curve.
 - **ST_EndPoint** - Returns the last point of a LineString, CircularLineString, or NURBS-Curve.
 - **ST_EstimatedExtent** - 返回空表的估计范围。
 - **ST_FlipCoordinates** - 返回 X 和 Y 坐标的几何图形版本。
 - **ST_Force2D** - 强制几何图形进入“二维模式”。
 - **ST_ForceCurve** - 如果适用，将一个几何图形上升到其曲线类型。
 - **ST_ForceSFS** - Force geometries to use SFS 1.1 or 1.2 geometry types.
 - **ST_Force3D** - 强制几何图形进入 XYZ 模式。它是 ST_Force3DZ 的别名。
 - **ST_Force3DM** - 强制几何图形进入 XYM 模式。
 - **ST_Force3DZ** - 强制几何图形进入 XYZ 模式。
 - **ST_Force4D** - 强制几何图形进入 XYZM 模式。
 - **ST_ForceCollection** - 将几何图形强制为几何集合 (GEOMETRYCOLLECTION)。
 - **ST_GeoHash** - 返回几何图形的 GeoHash 表示形式。
 - **ST_GeogFromWKB** - 从已知的二进制几何表示 (WKB) 或扩展的已知的二进制 (EWKB) 构建地理示例。
 - **ST_GeomFromEWKB** - 从扩展已知的二进制表示 (EWKB) 返回指定的 ST_Geometry 值。
 - **ST_GeomFromEWKT** - 从扩展已知的文本表示 (EWKT) 返回指定的 ST_Geometry 值。
 - **ST_GeomFromText** - 从已知的文本表示 (WKT) 返回指定的 ST_Geometry 值。
 - **ST_GeomFromWKB** - 从已知的二进制几何表示 (WKB) 和可选的 SRID 构建几何示例。
 - **ST_GeometryN** - 返回几何集合的一个元素。
 - **=** - 如果几何/地理 A 的坐点和坐点顺序与几何/地理 B 的坐点和坐点顺序相同，返回 TRUE。
 - **&<|** - 如果 A 的边界框与 B 的边界框重叠或低于 B 的边界框，返回 TRUE。
 - **ST_HasArc** - 检查几何图形是否包含弧。
 - **ST_Intersects** - 检查两个几何图形是否相交 (它们至少有一个共同点)。
 - **ST_IsClosed** - 检查 LineStrings 的起点和点是否重合。用于多面体表面是否闭合 (同心)。
 - **ST_IsCollection** - 检查几何类型是否是几何集合。
 - **ST_IsEmpty** - 检查几何图形是否空。
 - **ST_LineToCurve** - 将线性几何图形强制为曲线几何图形。
 - **ST_MMax** - Returns the M maxima of a geometry.
 - **ST_MMin** - 返回几何图形的 M 最小值。
 - **ST_MSize** - Returns the M size of a geometry.
 - **ST_MemSize** - 返回几何图形占用的内存空间量。
 - **ST_NPoints** - 返回几何图形中的点数 (点)。
-

- **ST_NRings** - 返回多边形几何中的环数。
 - **ST_PointFromWKB** - 使用指定的 SRID 从 WKB 构建几何图形。
 - **ST_PointN** - 返回几何图形中第一个点串或图形点串中的第 N 个点。
 - **ST_Points** - 返回包含几何坐标的 MultiPoint。
 - **ST_Rotate** - 绕原点旋转几何体。
 - **ST_RotateZ** - 绕 Z 轴旋转几何体。
 - **ST_SRID** - 返回几何图形的空参考系符号。
 - **ST_Scale** - 按指定因子缩放几何图形。
 - **ST_SetEndM** - Sets the M coordinate of the last point of a LineString, CircularLineString, or NURBS Curve.
 - **ST_SetSRID** - 在几何体上设置 SRID。
 - **ST_SetStartM** - Sets the M coordinate of the first point of a LineString, CircularLineString, or NURBS Curve.
 - **ST_StartM** - Returns the M coordinate of the first point of a LineString, CircularLineString, or NURBS Curve.
 - **ST_StartPoint** - Returns the first point of a LineString, CircularLineString, or NURBS Curve.
 - **ST_Summary** - 返回几何内容的文本摘要。
 - **ST_SwapOrdinates** - 返回更改后的几何图形，其中交换了指定的坐标。
 - **ST_TransScale** - 按指定的偏移量和系数平移和缩放几何图形。
 - **ST_Transform** - 返回坐标从不同空参考系的新几何图形。
 - **ST_Translate** - 按指定的偏移量平移几何图形。
 - **ST_XMax** - 返回 2D 或 3D 边界框或几何体的 X 最大值。
 - **ST_XMin** - 返回 2D 或 3D 边界框或几何体的 X 最小值。
 - **ST_XSize** - Returns the X size of a 2D or 3D bounding box or a geometry.
 - **ST_YMax** - 返回 2D 或 3D 边界框或几何体的 Y 最大值。
 - **ST_YMin** - 返回 2D 或 3D 边界框或几何体的 Y 最小值。
 - **ST_YSize** - Returns the Y size of a 2D or 3D bounding box or a geometry.
 - **ST_ZMFlag** - 返回指示几何体的 ZM 坐标度的代号。
 - **ST_ZMax** - 返回 2D 或 3D 边界框或几何体的 Z 最大值。
 - **ST_ZMin** - 返回 2D 或 3D 边界框或几何体的 Z 最小值。
 - **ST_ZSize** - Returns the Z size of a 2D or 3D bounding box or a geometry.
 - **UpdateGeometrySRID** - 更新几何列中所有要素的 SRID 以及表元数据。
 - **~(box2df,box2df)** - 如果 2D 浮点精度边界框 (BOX2DF) 包含一个 2D 浮点精度边界框 (BOX2DF)，返回 TRUE。
 - **~(box2df,geometry)** - 如果 2D 浮点精度边界框 (BOX2DF) 包含几何体的 2D 边界框，返回 TRUE。
 - **~(geometry,box2df)** - 如果几何体的 2D 粘合框包含 2D 浮点精度边界框 (GIDX)，返回 TRUE。
-

- **&&** - 如果 A 的 2D 边界框与 B 的 2D 边界框相交，返回 TRUE。
- **&&&** - 如果 A 的 n 边界框与 B 的 n 边界框相交，返回 TRUE。
- **@(box2df,box2df)** - 如果 2D 浮点精度边界框 (BOX2DF) 包含在另一个 2D 浮点精度边界框内，返回 TRUE。
- **@(box2df,geometry)** - 如果 2D 浮点精度边界框 (BOX2DF) 包含在几何体的 2D 边界框中，返回 TRUE。
- **@(geometry,box2df)** - 如果几何体的 2D 边界框包含在 2D 浮点精度边界框 (BOX2DF) 中，返回 TRUE。
- **&&(box2df,box2df)** - 如果两个 2D 浮点精度边界框 (BOX2DF) 彼此相交，返回 TRUE。
- **&&(box2df,geometry)** - 如果 2D 浮点精度边界框 (BOX2DF) 与几何体的 (存在的) 2D 边界框相交，返回 TRUE。
- **&&(geometry,box2df)** - 如果几何体的 (存在的) 2D 边界框与 2D 浮点精度边界框 (BOX2DF) 相交，返回 TRUE。
- **&&&(geometry,gidx)** - 如果几何体的 (存在的) n 边界框与 n 浮点精度边界框 (GIDX) 相交，返回 TRUE。
- **&&&(gidx,geometry)** - 如果 n 浮点精度边界框 (GIDX) 与几何体的 (存在的) n 边界框相交，返回 TRUE。
- **&&&(gidx,gidx)** - 如果两个 n 浮点精度边界框 (GIDX) 彼此相交，返回 TRUE。

13.10 PostGIS 多面体曲面支持函数

下面列出的函数是可以使用 POLYHEDRALSURFACE、POLYHEDRALSURFACEM 几何类型的 PostGIS 函数

- **Box2D** - 返回表示几何类型的 2D 范围的 BOX2D。
- **Box3D** - 返回表示几何体 3D 范围的 BOX3D。
- **CG_3DArea** - 计算 3D 表面几何形状的面积。对于固体将返回 0。
- **CG_3DConvexHull** - 计算几何体的 3D 凸包。
- **CG_3DDifference** - 执行 3D 差异
- **CG_3DIntersection** - 执行 3D 相交
- **CG_3DUnion** - 使用 postgis_sfcgal 执行三叉合并。
- **CG_ApproximateMedialAxis** - 计算几何区域的近似中轴。
- **CG_Extrude** - 将曲面挤出到相关体
- **CG_ForceLHR** - 强制 LHR 方向
- **CG_GenerateFlatRoof** - Generate a flat (box) roof from a footprint polygon.
- **CG_GenerateGableRoof** - Generate a gable roof from a footprint polygon.
- **CG_GenerateHippedRoof** - Generate a hipped roof from a footprint polygon.
- **CG_GenerateRoof** - Generate a roof of the requested type from a footprint polygon.
- **CG_GenerateSkillionRoof** - Generate a skillion (single-slope) roof from a footprint polygon.
- **CG_IsPlanar** - 检查表面是否平坦
- **CG_IsSolid** - 检查几何体是否是固体。不执行有效性检查。
- **CG_MakeSolid** - 将几何体造成固体。不执行任何检查。要得到有效的固体，输入几何形状必须是符合多面体曲面或符合 TIN。

- **CG_PolygonRepair** - Repair an invalid polygon or multipolygon.
 - **CG_StraightSkeleton** - 从几何体计算直骨架
 - **CG_Tessellate** - 将多边形或多面体表面分解为曲面，并以 TIN 或 TINS 集合的形式返回
 - **CG_Visibility** - 计算一个从点或多边形几何中的线段生成的可见性多边形
 - **CG_Volume** - Computes the volume of a 3D geometry. Closed surfaces can have non-zero volume.
 - **GeometryType** - 以文本形式返回几何的类型。
 - **ST_3DClosestPoint** - 返回 g1 上最接近 g2 的 3D 点。它是 3D 最短线的第一个点。
 - **ST_3DDFullyWithin** - 检查两个 3D 几何体是否完全在指定的 3D 距离内
 - **ST_3DDWithin** - 检查两个 3D 几何体是否在指定的 3D 距离内
 - **ST_3DDistance** - 返回两个几何体之间的 3D 笛卡儿最小距离（基于空参考）（以投影单位表示）。
 - **ST_3DExtent** - 返回几何体的 3D 边界框的聚合函数。
 - **ST_3DIntersects** - 检查两个几何体在 3D 空间中是否相交 - 适用于点、线串、多边形、多面体曲面（区域）
 - **ST_3DLongestLine** - 返回两个几何体之间的 3D 最长直线
 - **ST_3DMaxDistance** - 返回两个几何体之间的 3D 笛卡儿最大距离（基于空参考）（以投影单位表示）。
 - **ST_3DShortestLine** - 返回两个几何体之间的 3D 最短线
 - **ST_Affine** - 将几何体用 3D 仿射变换。
 - **ST_Area** - 返回多边形几何体的面积。
 - **ST_AsBinary** - 返回不带 SRID 元数据的几何/地理的 OGC/ISO 已知的二进制 (WKB) 表示形式。
 - **ST_AsEWKB** - 返回带有 SRID 元数据的几何体的扩展已知的二进制 (EWKB) 表示形式。
 - **ST_AsEWKT** - 使用 SRID 元数据返回几何体的已知文本 (WKT) 表示形式。
 - **ST_AsGML** - 将几何体作为 GML 版本 2 或 3 元素返回。
 - **ST_AsX3D** - 返回 X3D xml 点元素格式的几何体：ISO-IEC-19776-1.2-X3DEncodings-XML
 - **ST_CoordDim** - 返回几何体的坐标维度。
 - **ST_Dimension** - 返回几何体的拓扑维度。
 - **ST_Dump** - 返回几何体的一行 geometry_dump 行。
 - **ST_DumpPoints** - 返回几何体中坐落的一行 geometry_dump 行。
 - **ST_Expand** - 返回从一个边界框或几何体扩展的边界框。
 - **ST_Extent** - 返回几何体边界框的聚合函数。
 - **ST_FlipCoordinates** - 返回 X 和 Y 坐标的几何体版本。
 - **ST_Force2D** - 将几何体强制为“二维模式”。
 - **ST_ForceRHR** - 将多边形点的方向遵循右手定则。
 - **ST_ForceSFS** - Force geometries to use SFS 1.1 or 1.2 geometry types.
 - **ST_Force3D** - 将几何体强制为 XYZ 模式。它是 ST_Force3DZ 的同义词。
 - **ST_Force3DZ** - 将几何体强制为 XYZ 模式。
 - **ST_ForceCollection** - 将几何体强制为几何集合 (GEOMETRYCOLLECTION)。
-

- **ST_GeomFromEWKB** - 从展已知的二进制表示 (EWKB) 返回指定的 ST_Geometry 。
- **ST_GeomFromEWKT** - 从展已知的文本表示 (EWKT) 返回指定的 ST_Geometry 。
- **ST_GeomFromGML** - 将几何形的 GML 表示形式作入并出 PostGIS 几何象
- **ST_GeometryN** - 返回几何集合的一个元素。
- **ST_GeometryType** - 以文本形式返回几何形的 SQL-MM 型。
- **=** - 如果几何/地理 A 的坐和坐序与几何/地理 B 的坐和坐序相同，返回 TRUE。
- **&<|** - 如果 A 的界框与 B 的界框重或低于 B 的界框，返回 TRUE。
- **~=** - 如果 A 的界框与 B 的界框相同，返回 TRUE。
- **ST_IsClosed** - 的 LineStrings 的起点和点是否重合。于多面体表面是否合（心）。
- **ST_MemSize** - 返回几何形占用的内存空量。
- **ST_NPoints** - 返回几何形中的点数（点）。
- **ST_NumGeometries** - 返回几何集合中的元素数量。
- **ST_NumPatches** - Returns the number of faces in a PolyhedralSurface or TIN.
- **ST_PatchN** - Returns the Nth face of a PolyhedralSurface or TIN.
- **ST_RemoveRepeatedPoints** - 返回除了重复点的几何形。
- **ST_Reverse** - 返回点序相反的几何体。
- **ST_Rotate** - 原点旋几何体。
- **ST_RotateX** - X 旋几何体。
- **ST_RotateY** - Y 旋几何体。
- **ST_RotateZ** - Z 旋几何体。
- **ST_Scale** - 按定因子放几何形。
- **ST_ShiftLongitude** - 在 -180-180 和 0-360 之移几何形的度坐。
- **ST_Summary** - 返回几何内容的文本摘要。
- **ST_SwapOrdinates** - 返回更改后的几何形，其中交了的定的坐。
- **ST_Transform** - 返回坐的不同空参考系的新几何形。
- **~(box2df,box2df)** - 如果 2D 浮点精度界框 (BOX2DF) 包含一个 2D 浮点精度界框 (BOX2DF)，返回 TRUE。
- **~(box2df,geometry)** - 如果 2D 浮点精度界框 (BOX2DF) 包含几何体的 2D 界框，返回 TRUE。
- **~(geometry,box2df)** - 如果几何体的 2D 粘合框包含 2D 浮点精度界框 (GIDX)，返回 TRUE。
- **&&** - 如果 A 的 2D 界框与 B 的 2D 界框相交，返回 TRUE。
- **&&&** - 如果 A 的 n 界框与 B 的 n 界框相交，返回 TRUE。
- **@(box2df,box2df)** - 如果 2D 浮点精度界框 (BOX2DF) 包含在一个 2D 浮点精度界框内，返回 TRUE。
- **@(box2df,geometry)** - 如果 2D 浮点精度界框 (BOX2DF) 包含在几何体的 2D 界框中，返回 TRUE。
- **@(geometry,box2df)** - 如果几何体的 2D 界框包含在 2D 浮点精度界框 (BOX2DF) 中，返回 TRUE。
- **&&(box2df,box2df)** - 如果个 2D 浮点精度界框 (BOX2DF) 彼此相交，返回 TRUE。

- **&&(box2df,geometry)** - 如果 2D 浮点精度矩形框 (BOX2DF) 与几何体的 (存在的) 2D 矩形框相交, 返回 TRUE。
- **&&(geometry,box2df)** - 如果几何体的 (存在的) 2D 矩形框与 2D 浮点精度矩形框 (BOX2DF) 相交, 返回 TRUE。
- **&&&(geometry,gidx)** - 如果几何体的 (存在的) n 矩形框与 n 浮点精度矩形框 (GIDX) 相交, 返回 TRUE。
- **&&&(gidx,geometry)** - 如果 n 浮点精度矩形框 (GIDX) 与几何体的 (存在的) n 矩形框相交, 返回 TRUE。
- **&&&(gidx,gidx)** - 如果 n 个 n 浮点精度矩形框 (GIDX) 彼此相交, 返回 TRUE。

13.11 PostGIS 函数支持矩形

下面按字母序列出了 PostGIS 中的空特殊函数以及它使用的空类型或它遵守的 OGC/SQL 合性。

- A  表示函数本身适用于类型或子类型。
- A  意味着它可以工作, 但会使用内置的操作, 通常将其限制为几何类型、到“最佳 SRID (空参考 ID)”空参考, 然后再次限制回来。对于大面地区或极地地区, 结果可能不如预期, 并且可能会累积浮点数的误差。这是在地理数据管理中可能遇到的一些注意事项。
- A  表示函数适用于类型, 因为它会自行创建一个类型 (例如 box3d), 而不是直接类型支持。
- A  表示功能在使用 SFCGAL 支持时 PostGIS 可用。
- geom - 基本 2D 几何支持 (x,y)。
- geog - 基本 2D 地理支持 (x,y)。
- 2.5D - 3 D/4D 空其中的基本 2D 几何形状 (具有 Z 或 M 坐标)。
- PS - 多面体曲面
- T - 三角形和不规则三角网曲面 (TIN)

函数	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	✓						

函数	geom	geog	2.5D	Curves	SQL MM	PS	T
ST_PointZM	✓						
ST_Polygon	✓		✓		✓		
ST_TileEnvelope	✓						
ST_HexagonGrid	✓						
ST_Hexagon	✓						
ST_SquareGrid	✓						
ST_Square	✓						
ST_Letters	✓						
ST_MakeNurbsCurve	✓						
GeometryType	✓		✓	✓		✓	✓
ST_Boundary	✓		✓		✓		
ST_BoundingDiagonal	✓		✓				
ST_CoordDim	✓		✓	✓	✓	✓	✓
ST_Dimension	✓				✓	✓	✓
ST_Dump	✓		✓	✓		✓	✓
ST_DumpPoints	✓		✓	✓		✓	✓
ST_DumpSegments	✓		✓				✓
ST_DumpRings	✓		✓				
ST_EndPoint	✓		✓	✓	✓		
ST_Envelope	✓				✓		
ST_ExteriorRing	✓		✓		✓		
ST_GeometryN	✓		✓	✓	✓	✓	✓
ST_GeometryType	✓		✓		✓	✓	
ST_HasArc	✓		✓	✓			
ST_InteriorRingN	✓		✓		✓		
ST_NumCurves	✓		✓		✓		
ST_CurveN	✓		✓		✓		
ST_IsClosed	✓		✓	✓	✓	✓	
ST_IsCollection	✓		✓	✓			
ST_IsEmpty	✓			✓	✓		
ST_IsPolygonCCW	✓		✓				
ST_IsPolygonCW	✓		✓				
ST_IsRing	✓				✓		
ST_IsSimple	✓		✓		✓		

函数	geom	geog	2.5D	Curves	SQL MM	PS	T
ST_M	✓		✓		✓		
ST_MemSize	✓		✓	✓		✓	✓
ST_NDims	✓		✓				
ST_NPoints	✓		✓	✓		✓	
ST_NRings	✓		✓	✓			
ST_NumGeometries	✓		✓		✓	✓	✓
ST_NumInteriorRings	✓				✓		
ST_NumInteriorRing	✓						
ST_NumPatches	✓		✓		✓	✓	
ST_NumPoints	✓				✓		
ST_PatchN	✓		✓		✓	✓	✓
ST_PointN	✓		✓	✓	✓		
ST_Points	✓		✓	✓			
ST_StartPoint	✓		✓	✓	✓		
ST_StartM	✓			✓			
ST_EndM	✓			✓			
ST_SetStartM	✓			✓			
ST_SetEndM	✓			✓			
ST_Summary	✓	✓		✓		✓	✓
ST_X	✓		✓		✓		
ST_Y	✓		✓		✓		
ST_Z	✓		✓		✓		
ST_ZMFlag	✓		✓	✓			
ST_HasZ	✓		✓				
ST_HasM	✓		✓				
ST_ControlPoints	✓						
ST_Degree	✓						
ST_Weights	✓						
ST_Knots	✓						
ST_NumControlPoints	✓						
ST_NurbsCurve	✓	ational					
ST_IsNurbsCurve	✓						
ST_Evaluate	✓						
ST_NurbsToLine	✓	ring					

函数	geom	geog	2.5D	Curves	SQL MM	PS	T
ST_AddPoint	✓		✓				
ST_CollectionExtract	✓						
ST_CollectionHistogram	✓						
ST_CurveToLine	✓		✓	✓	✓		
ST_Scroll	✓		✓				
ST_FlipCoordinates	✓		✓	✓		✓	✓
ST_Force2D	✓		✓	✓		✓	
ST_Force3D	✓		✓	✓		✓	
ST_Force3DZ	✓		✓	✓		✓	
ST_Force3DM	✓			✓			
ST_Force4D	✓		✓	✓			
ST_ForceCollection	✓		✓	✓		✓	
ST_ForceCurve	✓		✓	✓			
ST_ForcePolygonCW	✓		✓				
ST_ForcePolygonCCW	✓		✓				
ST_ForceSFS	✓		✓	✓		✓	✓
ST_ForceRHR	✓		✓			✓	
ST_LineExtend	✓						
ST_LineToCurve	✓		✓	✓			
ST_Multi	✓						
ST_Normalize	✓						
ST_Project	✓	✓					
ST_QuantizeCoordinates	✓						
ST_RemovePoint	✓		✓				
ST_RemoveRepeatedPoints	✓		✓			✓	
ST_RemoveIrrelevantPointsForView	✓						
ST_RemoveSmallParts	✓						
ST_Reverse	✓		✓			✓	
ST_Segmentize	✓	✓					
ST_SetPoint	✓		✓				
ST_ShiftLongitude	✓		✓			✓	✓
ST_WrapX	✓		✓				
ST_SnapToGrid	✓		✓				
ST_Snap	✓						

函数	geom	geog	2.5D	Curves	SQL MM	PS	T
ST_SwapOrdina	✓		✓	✓		✓	✓
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_seal	✓						
ST_BdPolyFromText	✓						
ST_BdMPolyFromText	✓						
ST_GeogFromText		✓					
ST_GeographyFromText		✓					
ST_GeomCollFromText	✓				✓		
ST_GeomFromEWKT			✓	✓		✓	✓
ST_GeomFromMRC21	✓						
ST_GeometryFromText	✓				✓		
ST_GeomFromI	✓			✓	✓		
ST_LineFromText	✓				✓		
ST_MLineFromText	✓				✓		
ST_MPointFromText	✓				✓		
ST_MPolyFromText	✓				✓		
ST_PointFromText	✓				✓		
ST_PolygonFromText	✓				✓		
ST_WKTToSQL	✓				✓		
ST_GeogFromWKB		✓		✓			
ST_GeomFromEWKB	✓		✓	✓		✓	✓
ST_GeomFromV3	✓			✓	✓		
ST_LineFromW	✓				✓		
ST_LinestringFromWKB	✓				✓		

函数	geom	geog	2.5D	Curves	SQL MM	PS	T
ST_PointFromWkt	✓		✓	✓	✓		
ST_WKBToSQL	✓				✓		
ST_Box2dFromGeoHash	✗						
ST_GeomFromGeoHash	✓						
ST_GeomFromGeoJSON	✓		✓			✓	✓
ST_GeomFromKML	✓		✓				
ST_GeomFromTrio	✓						
ST_GMLToSQL	✓				✓		
ST_LineFromEncodedPolyline	✓						
ST_PointFromGeoHash	✓						
ST_FromFlatGeobufToTable	✗						
ST_FromFlatGeobuf	✗						
ST_AsEWKT	✓	✓	✓	✓		✓	✓
ST_AsText	✓	✓		✓	✓		
ST_AsBinary	✓	✓	✓	✓	✓	✓	✓
ST_AsEWKB	✓		✓	✓		✓	✓
ST_AsHEXEWKB	✓		✓	✓			
ST_AsEncodedPolyline	✓						
ST_AsFlatGeobuf	✗						
ST_AsGeobuf	✗						
ST_AsGeoJSON	✓	✓	✓				
ST_AsGML	✓	✓	✓		✓	✓	✓
ST_AsKML	✓	✓	✓				
ST_AsLatLonText	✓						
ST_AsMARC21	✓						
ST_AsMVTGeom	✓						
ST_AsMVT	✗						
ST_AsSVG	✓	✓		✓			
ST_AsTWKB	✓						
ST_AsX3D	✓		✓			✓	✓
ST_GeoHash	✓			✓			
&&	✓	✓		✓		✓	
&&(geometry,bbox)	✓			✓		✓	
&&(box2df,geometry)	✓			✓		✓	

函数	geom	geog	2.5D	Curves	SQL MM	PS	T
&&(box2df,box2df)	✓			✓		✓	
&&&	✓		✓	✓		✓	✓
&&&(geometry,geometry)	✓		✓	✓		✓	✓
&&&(gidx,geometry)	✓		✓	✓		✓	✓
&&&(gidx,gidx)			✓	✓		✓	✓
&<	✓						
&<	✓			✓		✓	
&>	✓						
<<	✓						
<<	✓						
=	✓	✓		✓		✓	
>>	✓						
@	✓						
@(geometry,box2df)	✓			✓		✓	
@(box2df,geometry)	✓			✓		✓	
@(box2df,box2df)	✓			✓		✓	
&>	✓						
>>	✓						
~	✓						
~(geometry,box2df)	✓			✓		✓	
~(box2df,geometry)	✓			✓		✓	
~(box2df,box2df)	✓			✓		✓	
~=	✓					✓	
<->	✓	✓					
=	✓						
<#>	✓						
<<->>	✓						
ST_3DIntersects	✓		✓		✓	✓	✓
ST_Contains	✓				✓		
ST_ContainsProperly	✓						
ST_CoveredBy	✓	✓					
ST_Covers	✓	✓					
ST_Crosses	✓				✓		
ST_Disjoint	✓				✓		

函数	geom	geog	2.5D	Curves	SQL MM	PS	T
ST_Equals	✓				✓		
ST_Intersects	✓	✓		✓	✓		✓
ST_LineCrossingDirection	✓						
ST_OrderingEquals	✓				✓		
ST_Overlaps	✓				✓		
ST_Relate	✓				✓		
ST_RelateMatch							
ST_Touches	✓				✓		
ST_Within	✓				✓		
ST_3DDWithin	✓		✓		✓	✓	
ST_3DDFullyWithin	✓		✓			✓	
ST_DFullyWithin	✓						
ST_DWithin	✓	✓					
ST_PointInsideCircle	✓						
ST_Area	✓	✓			✓	✓	
ST_Azimuth	✓	✓					
ST_Angle	✓						
ST_ClosestPoint	✓	✓					
ST_3DClosestPoint	✓		✓			✓	
ST_Distance	✓	✓		✓	✓		
ST_3DDistance	✓		✓		✓	✓	
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	✓						

函数	geom	geog	2.5D	Curves	SQL MM	PS	T
ST_MinimumClearanceLine	✓						
ST_Perimeter	✓	✓			✓		
ST_Perimeter2D	✓						
ST_3DPerimeter	✓		✓		✓		
ST_ShortestLine	✓	✓					
ST_3DShortestLine	✓		✓			✓	
ST_ClipByBox2D	✓						
ST_Difference	✓		✓		✓		
ST_Intersection	✓	😄	✓		✓		
ST_MemUnion	✓		✓				
ST_Node	✓		✓				
ST_Split	✓						
ST_Subdivide	✓						
ST_SymDifference	✓		✓		✓		
ST_UnaryUnion	✓		✓				
ST_Union	✓		✓		✓		
ST_Buffer	✓	😄			✓		
ST_BuildArea	✓						
ST_Centroid	✓	✓			✓		
ST_ChaikinSmoothing	✓		✓				
ST_CatmullRomSmoothing	✓		✓				
ST_ConcaveHull	✓						
ST_ConvexHull	✓		✓		✓		
ST_DelaunayTriangles	✓		✓				✓
ST_FilterByM	✓						
ST_GeneratePoints	✓						
ST_GeometricMedian	✓		✓				
ST_LineMerge	✓						
ST_MaximumInscribedCircle	✓						
ST_LargestEmptyCircle	✓						
ST_MinimumBoundingCircle	✓						
ST_MinimumBoundingRadius	✓						
ST_OrientedEnvelope	✓						
ST_OffsetCurve	✓						

函数	geom	geog	2.5D	Curves	SQL MM	PS	T
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_CoverageClean	✓						
ST_CoverageSimplify	✓						
ST_CoverageEdges	✓						
ST_CoverageUnion	✓						
ST_Affine	✓		✓	✓		✓	✓
ST_Rotate	✓		✓	✓		✓	✓
ST_RotateX	✓		✓			✓	✓
ST_RotateY	✓		✓			✓	✓
ST_RotateZ	✓		✓	✓		✓	✓
ST_Scale	✓		✓	✓		✓	✓
ST_Translate	✓		✓	✓			
ST_TransScale	✓		✓	✓			
ST_ClusterDBSCAN	✓			✓			
ST_ClusterIntersecting	✓						
ST_ClusterIntersectingWin	✓						
ST_ClusterRelativeWin	✓						
ST_MinimumSpanningTree	✓						
ST_ClusterKMeans	✓						
ST_ClusterWithin	✓			✓			
ST_ClusterWithinWin	✓			✓			
Box2D	✓			✓		✓	✓

函数	geom	geog	2.5D	Curves	SQL MM	PS	T
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_XSize	✓		✓	✓			
ST_YSize	✓		✓	✓			
ST_ZSize	✓		✓	✓			
ST_MMin	✓		✓	✓			
ST_MMax	✓		✓	✓			
ST_MSize	✓		✓	✓			
ST_LineInterpolatePoint	✓	✓	✓				
ST_3DLineInterpolatePoint	✓		✓				
ST_LineInterpolatePoints	✓	✓	✓				
ST_LineLocatePoint	✓	✓					
ST_LineSubstring	✓	✓	✓				
ST_LocateAlong	✓				✓		
ST_LocateBetween	✓				✓		
ST_LocateBetweenElevations	✓		✓				
ST_InterpolatePoint	✓		✓				
ST_3DInterpolatePoint	✓		✓				
ST_AddMeasure	✓		✓				
ST_IsValidTrajectory	✓		✓				
ST_ClosestPoint	✓	Approach	✓				
ST_DistanceCPA	✓		✓				
ST_CPASWithin	✓		✓				

函数	geom	geog	2.5D	Curves	SQL MM	PS	T
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							
CG_Intersection							
CG_Intersects							
CG_3DIntersec							
CG_Difference							
CG_3DDifferen							
CG_Distance							
CG_3DDistance							
CG_3DConvexH							
CG_3DIntersec							
CG_Union							
CG_3DUnion							

函数	geom	geog	2.5D	Curves	SQL MM	PS	T
CG_AlphaShap							
CG_ApproxCon		artition					
CG_Approxima		rdialAxis					
CG_GenerateFl		of					
CG_GenerateH		dRoof					
CG_GenerateG		Roof					
CG_GenerateSl		nRoof					
CG_GenerateR							
CG_PolygonRej							
CG_Constraine		ounayTriangles					
CG_Extrude							
CG_ExtrudeStr		tSkeleton					
CG_GreeneApp		onvexPartition					
CG_Minkowski							
CG_OptimalAlp		nape					
CG_OptimalCo		Partition					
CG_StraightSk		n					
CG_Tessellate							
CG_Triangulat							
CG_Visibility							
CG_YMonotone		ition					
CG_StraightSk		nPartition					
CG_3DBuffer							
CG_Rotate							

函数	geom	geog	2.5D	Curves	SQL MM	PS	T
CG_2DRotate							
CG_3DRotate							
CG_RotateX							
CG_RotateY							
CG_RotateZ							
CG_Scale							
CG_3DScale							
CG_3DScaleArcCenter							
CG_Translate							
CG_3DTranslate							
CG_Simplify							
CG_3DAlphaWrapping							
CG_NurbsCurveInterpolate							
CG_NurbsCurveApproximate							
CG_NurbsCurveDerivative							
getfaceedges_returntype							
TopoGeometry							
validate_topology_returntype							
TopoElement							
TopoElementArray							
AddTopoGeometryColumn							
RenameTopoGeometryColumn							
DropTopology							
RenameTopology							
DropTopoGeometryColumn							
FixCorruptTopoGeometryColumn							
Populate_Topology_Layer							
TopologySummary							
ValidateTopology✓							
ValidateTopologyRelation							
ValidateTopologyPrecision✓							
MakeTopologyPersistent✓							
FindTopology✓							

函数	geom	geog	2.5D	Curves	SQL MM	PS	T
FindLayer	✓						
TotalTopologySize							
UpgradeTopology							
CreateTopology							
CopyTopology							
ST_InitTopoGeo					✓		
ST_CreateTopoGeo	✓				✓		
TopoGeo_AddPoint	✓						
TopoGeo_AddLineString	✓						
TopoGeo_AddPolygon	✓						
TopoGeo_LoadGeometry	✓						
ST_AddIsoNode	✓				✓		
ST_AddIsoEdge	✓				✓		
ST_AddEdgeNewFaces	✓				✓		
ST_AddEdgeModifyFace	✓				✓		
ST_RemEdgeNewFace					✓		
ST_RemEdgeModFace					✓		
ST_ChangeEdgeCom	✓				✓		
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							
FindVertexSegmentPairsBelowDistance							
Polygonize							

函数	geom	geog	2.5D	Curves	SQL MM	PS	T
AddNode	✓						
AddEdge	✓						
AddFace	✓						
ST_Simplify	✓						
RemoveUnused	✓ natives						
CreateTopoGeom	✓						
toTopoGeom	✓						
TopoElementArray_Agg							
TopoElement	✓						
clearTopoGeom	✓						
TopoGeom_add	✓ nent						
TopoGeom_remove	✓ ment						
TopoGeom_add	✓ oGeom						
toTopoGeom							
GetTopoGeomElementArray							
GetTopoGeomElements							
ST_SRID	✓						
AsGML	✓						
AsTopoJSON	✓						
Equals	✓		✓				
Intersects	✓		✓				
geomval							
addbandarg							
rastbandarg							
raster							
reclassarg							
summarystats							
unionarg							
AddRasterConstraints							
DropRasterConstraints							
AddOverviewConstraints							
DropOverviewConstraints							
PostGIS_GDAL_Version							
PostGIS_Raster_Lib_Build_Date							
PostGIS_Raster_Lib_Version							
ST_GDALDrivers							
UpdateRasterSRID							
ST_CreateOverview							
ST_AddBand							
ST_AsRaster	✓						
ST_AsRasterAgg	✓						
ST_Band							
ST_MakeEmptyCoverage							
ST_MakeEmptyRaster							

函数	geom	geog	2.5D	Curves	SQL MM	PS	T
ST_Tile							
ST_Retile	✓						
ST_FromGDALRaster							
ST_GeoReference							
ST_Height							
ST_IsEmpty							
ST_MemSize							
ST_MetaData							
ST_NumBands							
ST_PixelHeight							
ST_PixelWidth							
ST_ScaleX							
ST_ScaleY							
ST_RasterToWorldCoord							
ST_RasterToWorldCoordX							
ST_RasterToWorldCoordY							
ST_Rotation							
ST_SkewX							
ST_SkewY							
ST_SRID							
ST_Summary							
ST_UpperLeftX							
ST_UpperLeftY							
ST_Width							
ST_WorldToRasterCoord	✓						
ST_WorldToRasterCoordX	✓						
ST_WorldToRasterCoordY	✓						
ST_BandMetaData							
ST_BandNoDataValue							
ST_BandIsNoData							
ST_BandPath							
ST_BandFileSize							
ST_BandFileTimestamp							
ST_BandPixelType							
ST_MinPossibleValue							
ST_HasNoBand							
ST_PixelAsPolygon	✓						
ST_PixelAsPolygons							
ST_PixelAsPoint	✓						
ST_PixelAsPoints							
ST_PixelAsCentroid	✓ 1						
ST_PixelAsCentroids							
ST_Value	✓						
ST_NearestValue	✓						
ST_SetZ	✓						
ST_SetM	✓						
ST_Neighborhood	✓						

函数	geom	geog	2.5D	Curves	SQL MM	PS	T
ST_SetValue	✓						
ST_SetValues							
ST_DumpValues							
ST_PixelOfValue							
ST_SetGeoReference							
ST_SetRotation							
ST_SetScale							
ST_SetSkew							
ST_SetSRID							
ST_SetUpperLeft							
ST_Resample							
ST_Rescale							
ST_Reskew							
ST_SnapToGrid							
ST_Resize							
ST_Transform							
ST_SetBandNoDataValue							
ST_SetBandIsNoData							
ST_SetBandPath							
ST_SetBandIndex							
ST_Count							
ST_CountAgg							
ST_Histogram							
ST_Quantile							
ST_SummaryStats							
ST_SummaryStatsAgg							
ST_ValueCount							
ST_RastFromWKB							
ST_RastFromHexWKB							
ST_AsBinary/ST_AsWKB							
ST_AsHexWKB							
ST_AsGDALRaster							
ST_AsJPEG							
ST_AsPNG							
ST_AsTIFF							
ST_Clip	✓						
ST_ColorMap							
ST_Grayscale							
ST_Intersection	✓						
ST_MapAlgebra (callback function version)							
ST_MapAlgebra (expres- sion version)							
ST_MapAlgebraExpr							
ST_MapAlgebraExpr							
ST_MapAlgebraFct							
ST_MapAlgebraFct							
ST_MapAlgebraFctNgb							
ST_Reclass							
ST_ReclassExact							

函数	geom	geog	2.5D	Curves	SQL MM	PS	T
ST_Union							
ST_Distinct4ma							
ST_InvDistWeight4ma							
ST_Max4ma							
ST_Mean4ma							
ST_Min4ma							
ST_MinDist4ma							
ST_Range4ma							
ST_StdDev4ma							
ST_Sum4ma							
ST_Aspect							
ST_HillShade							
ST_Roughness							
ST_Slope							
ST_TPI							
ST_TRI							
ST_InterpolateF✓er							
ST_Contour							
Box3D							
ST_ConvexHull✓							
ST_DumpAsPolygons							
ST_Envelope✓							
ST_MinConvexH✓							
ST_Polygon✓							
ST_Intersection✓ctions							
&&✓							
&<							
&>							
=							
@✓							
~=							
~✓							
ST_Contains							
ST_ContainsProperly							
ST_Covers							
ST_CoveredBy							
ST_Disjoint							
ST_Intersects✓							
ST_Overlaps							
ST_Touches							
ST_SameAlignment							
ST_NotSameAlignmentReason							
ST_Within							
ST_DWithin							
ST_DFullyWithin							

13.12 新的、增强的或更改的 **PostGIS** 函数

13.12.1 **PostGIS** 新增功能或增强功能 (3.7)

下面列出的功能是添加或增强的 **PostGIS** 功能。

PostGIS 中的新功能 3.7

- **CG_ApproximateMedialAxis** - Availability: 3.7.0 - projected parameter. Requires SFCGAL >= 2.3.0 for projected result; falls back to non-projected with a notice on older versions. 计算几何区域的近似中轴。
- **CG_GenerateFlatRoof** - Availability: 3.7.0 - requires SFCGAL >= 2.3.0. Generate a flat (box) roof from a footprint polygon.
- **CG_GenerateGableRoof** - Availability: 3.7.0 - requires SFCGAL >= 2.3.0. Generate a gable roof from a footprint polygon.
- **CG_GenerateHippedRoof** - Availability: 3.7.0 - requires SFCGAL >= 2.3.0. Generate a hipped roof from a footprint polygon.
- **CG_GenerateRoof** - Availability: 3.7.0 - requires SFCGAL >= 2.3.0. Generate a roof of the requested type from a footprint polygon.
- **CG_GenerateSkillionRoof** - Availability: 3.7.0 - requires SFCGAL >= 2.3.0. Generate a skillion (single-slope) roof from a footprint polygon.
- **CG_NurbsCurveApproximate** - Availability: 3.7.0 - requires SFCGAL >= 2.3.0. Creates an approximating NURBS curve fitting data points within a tolerance
- **CG_NurbsCurveDerivative** - Availability: 3.7.0 - requires SFCGAL >= 2.3.0. Computes the derivative (tangent, curvature) of a NURBS curve at a given parameter
- **CG_NurbsCurveInterpolate** - Availability: 3.7.0 - requires SFCGAL >= 2.3.0. Creates an interpolating NURBS curve passing through all given data points
- **CG_PolygonRepair** - Availability: 3.7.0 - requires SFCGAL >= 2.3.0 and CGAL >= 6.0. Repair an invalid polygon or multipolygon.
- **FindVertexSegmentPairsBelowDistance** - Availability: 3.7 Find pairs of topology vertex/segment that are closer than tolerated distance
- **ST_3DInterpolatePoint** - Availability: 3.7.0 Returns the interpolated measure of a geometry closest to a point in 3D.
- **ST_CatmullRomSmoothing** - Availability: 3.7.0 Returns a smoothed version of a geometry, using the Catmull-Rom spline algorithm
- **ST_ClusterRelateWin** - Availability: 3.7.0 Window function that returns a cluster id for each input geometry, clustering input geometries into connected sets using the relate pattern to determine whether the geometries are connected.
- **ST_ControlPoints** - Availability: 3.7.0 Returns the control points of a NURBS curve as a MULTIPOINT geometry.
- **ST_CoverageEdges** - Availability: 3.7.0 Computes the unique edges of a polygonal coverage.
- **ST_Degree** - Availability: 3.7.0 Returns the polynomial degree of a NURBS curve.
- **ST_Evaluate** - Availability: 3.7.0 Evaluates a NURBS curve at a specific parameter value and returns the resulting point.

- **ST_IsNurbsCurve** - Availability: 3.7.0 Checks if a geometry is a NURBS curve.
- **ST_Knots** - Availability: 3.7.0 Returns the knot vector of a NURBS curve.
- **ST_MSize** - Availability: 3.7.0 Returns the M size of a geometry.
- **ST_MakeLine** - 可用性: 3.7.0 - 引入了 `MultiLineString` 入要素的支持从 `Point`, `MultiPoint`, 或 `LineString` geometries 建 `LineString`。
- **ST_MakeNurbsCurve** - Availability: 3.7.0 Creates a NURBS (Non-Uniform Rational B-Spline) curve from control points, optional weights, and optional knot vector.
- **ST_MinimumSpanningTree** - Availability: 3.7.0 Window function that returns a tree id for each input geometry, clustering input geometries into connected trees using the Minimum Spanning Tree algorithm.
- **ST_NumControlPoints** - Availability: 3.7.0 Returns the number of control points in a NURBS curve.
- **ST_NurbsCurveIsRational** - Availability: 3.7.0 Checks if a NURBS curve is rational (has weights).
- **ST_NurbsToLineString** - Availability: 3.7.0 Converts a NURBS curve to a `LINestring` by uniform sampling.
- **ST_Weights** - Availability: 3.7.0 Returns the weight array of a rational NURBS curve.
- **ST_XSize** - Availability: 3.7.0 Returns the X size of a 2D or 3D bounding box or a geometry.
- **ST_YSize** - Availability: 3.7.0 Returns the Y size of a 2D or 3D bounding box or a geometry.
- **ST_ZSize** - Availability: 3.7.0 Returns the Z size of a 2D or 3D bounding box or a geometry.

PostGIS 的功能增强 3.7

- **ST_AsX3D** - Enhanced: 3.7.0: Support for curve geometries. `NURBSCURVE` is emitted as `X3D NurbsCurve`; other curve types are linearized before output. 返回 `X3D xml` 点元素格式的几何形: `ISO-IEC-19776-1.2-X3DEncodings-XML`
- **ST_GMLToSQL** - Enhanced: 3.7.0 support for `GML ArcString`, `curved Ring`, and `CurvePolygon` geometries was introduced. Returns a geometry from `GML`. Alias for `ST_GeomFromGML`.
- **ST_GeomFromGML** - Enhanced: 3.7.0 support for `GML ArcString`, `curved Ring`, and `CurvePolygon` geometries was introduced. 将几何形的 `GML` 表示形式作入并出 `PostGIS` 几何象
- **ST_LongestLine** - Enhanced: 3.7.0 - support for a single geometry input. 返回个几何形之的二最。
- **ST_MakePolygon** - Enhanced: 3.7.0 - Support for curved input rings was introduced. Creates a `Polygon` or `CurvePolygon` from a shell and optional list of holes.
- **ST_MaxDistance** - Enhanced: 3.7.0 - support for a single geometry input. 返回个几何形之的二最大距离 (以投影位表示)。
- **ST_Value** - Enhanced: 3.7.0 `resample` accepts nearest-neighbor boundary options "nearest-UL", "nearest-UR", "nearest-LL", and "nearest-LR". 返回指定列 x、行 y 像素或特定几何点定波段的。波段号从 1 开始, 如果未指定, 默 1。如果将参数 `exclude_nodata_value` 置 `false`, 所有像素都被与 `nodata` 像素相交并返回其。如果未参数 `exclude_nodata_value`, 从格的元数据中取。
- **TopoGeo_AddLineString** - Enhanced: 3.7.0 增加限制拓扑中新建 `edge` 数量的支持。使用容差将串添加到有拓扑中, 并在可能的情况下拆分或面。

PostGIS 中的功能生 3.7

- **ST_AsGeoJSON** - Changed: 3.7.0 -- 增加了关于重复的警告以 `GeoJSON` 格式返回一个几何体或要素。
- **toTopoGeom** - Changed: 3.7.0 将 `tolerance` 0 的含改真正表示 0, 而 -1 (新的默) 表示使用拓扑精度。将几何形拓扑几何形。

13.12.2 PostGIS 新增功能或增强功能 (3.6)

下面列出的功能是添加或增强的 PostGIS 功能。

PostGIS 中的新功能 3.6

- **CG_2DRotate** - 可用性：3.6.0 - 需要 SFCGAL $\geq$ 2.0.0 将几何图形绕指定的二点按指定角度进行旋转。
- **CG_3DAlphaWrapping** - 可用性：3.6.0 - 需要 SFCGAL $\geq$ 2.1.0 计算一个网格包络面几何的 3D Alpha 包络面。
- **CG_3DBuffer** - 可用性：3.6.0 - 需要 SFCGAL $\geq$ 2.0.0 计算几何体周围的三维缓冲区。
- **CG_3DRotate** - 可用性：3.6.0 - 需要 SFCGAL $\geq$ 2.0.0 输入几何体 geom 会绕指定的向量进行三维旋转。旋转角度由参数 angle（以弧度为单位）指定，旋转将在三维空间内进行。
- **CG_3DScale** - 可用性：3.6.0 - 需要 SFCGAL $\geq$ 2.0.0 将几何图形沿 X、Y、Z 轴分别按指定的不同系数进行缩放。
- **CG_3DScaleAroundCenter** - 可用性：3.6.0 - 需要 SFCGAL $\geq$ 2.0.0 将几何图形在三维空间中，以指定的点为中心进行缩放。
- **CG_3DTranslate** - 可用性：3.6.0 - 需要 SFCGAL $\geq$ 2.0.0 将输入几何图形按照指定的偏移量在三维空间中移动。
- **CG_Rotate** - 可用性：3.6.0 - 需要 SFCGAL $\geq$ 2.0.0 将几何图形绕 (0,0) 旋转指定的角度。
- **CG_RotateX** - 可用性：3.6.0 - 需要 SFCGAL $\geq$ 2.0.0 将几何图形绕 X 轴旋转指定的角度。
- **CG_RotateY** - 可用性：3.6.0 - 需要 SFCGAL $\geq$ 2.0.0 将几何图形绕 Y 轴旋转指定的角度。
- **CG_RotateZ** - 可用性：3.6.0 - 需要 SFCGAL $\geq$ 2.0.0 将几何图形绕 Z 轴旋转指定的角度。
- **CG_Scale** - 可用性：3.6.0 - 需要 SFCGAL $\geq$ 2.0.0 将几何图形在所有维度上按指定的系数均匀缩放。
- **CG_Simplify** - 可用性：3.6.0 - 需要 SFCGAL $\geq$ 2.1.0 在保留几何对象关键特征以及 Z/M 值的前提下，降低其复杂度。
- **CG_StraightSkeletonPartition** - 可用性：3.6.0 - 需要 SFCGAL $\geq$ 2.0.0。计算多边形的直骨架分割。
- **CG_Translate** - 可用性：3.6.0 - 需要 SFCGAL $\geq$ 2.0.0 将几何图形在二维空间中按指定的偏移量进行平移。
- **FixCorruptTopoGeometryColumn** - 可用性：3.6.1 修复由升级到 postgis_topology 3.6.0 及以上版本引起的 topogeometry 数据损坏问题。
- **MakeTopologyPrecise** - 可用性：3.6.0 将拓扑点精确到精度网格。
- **ST_AsRasterAgg** - 可用性：3.6.0 是一个聚合函数。将 PostGIS 几何图形转换为新的栅格图像。
- **ST_CoverageClean** - 可用性：3.6.0 - 需要 GEOS $\geq$ 3.14.0 非规范化输入计算出一个规范的多边形覆盖结果（边界匹配、无重叠、无间隙）。
- **ST_IntersectionFractions** - 可用性：3.6.0 -- 需要 GEOS 3.14 或更高版本。计算给定 geometry 覆盖到每个 raster 像元的比例。
- **ST_ReclassExact** - 可用性：3.6.0 建立一个新栅格，其波段通路与原始波段进行 1:1 映射重分类生成，将原波段中的值映射到目标波段中的新值。
- **TotalTopologySize** - 可用性：3.6.0 指定拓扑占用的磁盘空间，包括所有索引和 TOAST 数据。
- **UpgradeTopology** - 可用性：3.6.0 将指定拓扑升级到支持大整数 (int8) 类型的拓扑 ID 和要素 ID。
- **ValidateTopologyPrecision** - 可用性：3.6.0 返回拓扑中不精确的点。
- **postgis.gdal_cpl_debug** - 可用性：3.6.0 用于开启或关闭 GDAL 调试信息日志的布尔型配置项。

PostGIS 的功能增 3.6

- **CreateTopology** - Enhanced: 3.6.0 新增 topoid 和 usesLargeIDs 参数。建一个新的拓扑架并将其注册到 topology.topology 表中。

PostGIS 中的功能生 3.6

- **ST_GeometryN** - Changed: 3.6.0 TIN and PolyhedralSurface are treated as unitary geometries and are not decomposed into patches; use ST_PatchN to access faces. 返回几何集合的一个元素。
- **ST_NumGeometries** - Changed: 3.6.0 TIN and PolyhedralSurface are treated as unitary geometries and return 1; use ST_NumPatches to count faces. 返回几何集合中的元素数量。

13.12.3 PostGIS 新增功能或增 功能 (3.5)

下面出的功能是添加或增 的 PostGIS 功能。

PostGIS 中的新功能 3.5

- **CG_3DArea** - 可用性：3.5.0 算 3D 表面几何形状的面。于固体将返回 0。
- **CG_3DConvexHull** - 可用性：3.5.0 算几何体的 3D 凸包。
- **CG_3DDifference** - 可用性：3.5.0 行 3D 差异
- **CG_3DDistance** - 可用性：3.5.0 返回个三几何形之的最小距离
- **CG_3DIntersection** - 可用性：3.5.0 行 3D 相交
- **CG_3DIntersects** - 可用性：3.5.0 个三几何形是否相交
- **CG_3DUnion** - 可用性：3.5.0 使用 postgis_sfcgal 行三合并。
- **CG_AlphaShape** - 可用性：3.5.0 - 需要 SFCGAL >= 1.4.1。算包几何体的 Alpha 形状
- **CG_ApproxConvexPartition** - 可用性：3.5.0 - 需要 SFCGAL >= 1.5.0。算多形几何形的近似凸分割
- **CG_ApproximateMedialAxis** - 可用性：3.5.0 算几何区域的近似中。
- **CG_Area** - 可用性：3.5.0 算几何形的面
- **CG_Difference** - 可用性：3.5.0 算个几何形之的几何差异
- **CG_Distance** - 可用性：3.5.0 返回个几何形之的最小距离
- **CG_Extrude** - 可用性：3.5.0 将曲面出到相关体
- **CG_ExtrudeStraightSkeleton** - 可用性：3.5.0 - 需要 SFCGAL >= 1.5.0。直骨架外凸
- **CG_ForceLHR** - 可用性：3.5.0 制 LHR 方向
- **CG_GreeneApproxConvexPartition** - 可用性：3.5.0 - 需要 SFCGAL >= 1.5.0。算多形几何形的近似凸分割
- **CG_Intersection** - 可用性：3.5.0 算个几何形的相交（共享）部分
- **CG_Intersects** - 可用性：3.5.0 个几何形是否相交（它至少有一个共同点）
- **CG_IsPlanar** - 可用性：3.5.0 表面是否平坦
- **CG_IsSolid** - 可用性：3.5.0 几何体是否体。不行有效性。

- **CG_MakeSolid** - 可用性：3.5.0 将几何体创建成体。不行任何。要得有效的体，入几何形必是合多面体曲面或合 TIN。
- **CG_MinkowskiSum** - 可用性：3.5.0 行 Minkowski sum
- **CG_OptimalAlphaShape** - 可用性：3.5.0 - 需要 SFCGAL >= 1.4.1。使用“最佳”alpha 算包几何体的 Alpha 形状。
- **CG_OptimalConvexPartition** - 可用性：3.5.0 - 需要 SFCGAL >= 1.5.0。算多形几何形的最凸分割
- **CG_Orientation** - 可用性：3.5.0 确定表面方向
- **CG_StraightSkeleton** - 可用性：3.5.0 从几何体算直骨架
- **CG_Tessellate** - 可用性：3.5.0 多形或多面体表面行曲面分，并以 TIN 或 TINS 集合的形式返回
- **CG_Triangulate** - 可用性：3.5.0 多形几何体行三角剖分
- **CG_Union** - 可用性：3.5.0 算个几何形的合并
- **CG_Visibility** - 可用性：3.5.0 - 需要 SFCGAL >= 1.5.0。算一个从点或多形几何中的段生成的可性多形
- **CG_Volume** - 可用性：3.5.0 Computes the volume of a 3D geometry. Closed surfaces can have non-zero volume.
- **CG_YMonotonePartition** - 可用性：3.5.0 - 需要 SFCGAL >= 1.5.0。算多形几何的 y 分割
- **ST_HasM** - 可用性：3.5.0 几何体是否具有 M（量）度。
- **ST_HasZ** - 可用性：3.5.0 几何体是否具有 Z 度。
- **ST_RemoveIrrelevantPointsForView** - 可用性：3.5.0 除在指定矩形表示范内与制无关的几何形点。
- **ST_RemoveSmallParts** - 可用性：3.5.0 除几何形中的所有小部分（多形或串）。
- **TopoGeo_LoadGeometry** - 可用性：3.5.0 将几何形加到有拓扑中，并在必要行捕捉和拆分操作。

PostGIS 的功能增 3.5

- **ST_Clip** - 增版：3.5.0 - 添加了 touched 参数。返回由入几何形裁剪的格。如果未指定波段号，理所有波段。如果未指定 crop 或其 TRUE，出格将被裁剪。如果 touched 置 TRUE，包括接触到的像素；否，当像素的中心位于几何形内，才会包括像素。

PostGIS 中的功能生 3.5

- **ST_AsGeoJSON** - 更改：3.5.0 允指定包含要素 ID 的列以 GeoJSON 格式返回一个几何体或要素。
- **ST_DFullyWithin** - 更改：3.5.0 版本：函数的在函数冲区内包含，而不是 ST_MaxDistance 算法。果可能与之前的版本不同，但更接近用的期。一个几何形是否完全位于一个几何形的指定距离内

13.12.4 PostGIS 新增功能或增功能 (3.4)

下面出的功能是添加或增的 PostGIS 功能。

PostGIS 中的新功能 3.4

- **PostGIS_GEOS_Compiled_Version** - 可用性：3.4.0 返回建 PostGIS 所依据的 GEOS 的版本号。
- **PostGIS_PROJ_Compiled_Version** - 可用性：3.5.0 返回用于建 PostGIS 的 PROJ 的版本号。

- **RenameTopoGeometryColumn** - 可用性：3.4.0 重命名拓扑几何列
- **RenameTopology** - 可用性：3.4.0 重命名拓扑
- **ST_ClusterIntersectingWin** - 可用性：3.4.0 窗口函数，返回每个输入几何图形的簇 ID，将输入几何图形聚到交集的集合中。
- **ST_ClusterWithinWin** - 可用性：3.4.0 窗口函数，返回每个输入几何图形的簇 ID，使用分离距离进行聚类。
- **ST_CoverageInvalidEdges** - 可用性：3.4.0 用于找出多边形无法形成有效覆盖范围的位置的窗口函数。
- **ST_CoverageSimplify** - 可用性：3.4.0 简化多边形覆盖范围的窗口函数。
- **ST_CoverageUnion** - 可用性：3.4.0-需要 GEOS >= 3.8.0 通过移除共享来估算形成覆盖范围的一个多边形并集。
- **ST_InverseTransformPipeline** - 可用性：3.4.0 返回一个新的几何体，其坐标使用定义的坐标管道到不同的空参考系。
- **ST_LargestEmptyCircle** - 可用性：3.4.0。计算不与几何图形重叠的最大圆。
- **ST_LineExtend** - 可用性：3.4.0 返回一条线，向前和向后延伸指定的距离。
- **ST_TransformPipeline** - 可用性：3.4.0 返回一个新的几何图形，其坐标使用定义的坐标管道到不同的空参考系。
- **TopoElement** - 可用性：3.4.0 将拓扑几何图形转换为拓扑元素。
- **postgis_srs** - 可用性：3.4.0 返回所请求的界限和 srid 的元数据。
- **postgis_srs_all** - 可用性：3.4.0 返回底层的 Proj 数据中每个空参考系的元数据。
- **postgis_srs_codes** - 可用性：3.4.0 返回与定义界限相关的 SRS 代码列表。
- **postgis_srs_search** - 可用性：3.4.0 返回具有完全包含边界参数的使用区域的投影坐标系的元数据。

PostGIS 的功能增强 3.4

- **PostGIS_Full_Version** - 增强功能：3.4.0 在包括外的 PROJ 配置 NETWORK_ENABLED、URL_ENDPOINT 和 proj.db 位置的 DATABASE_PATH 报告完整的 PostGIS 版本和构建配置信息。
- **PostGIS_PROJ_Version** - 增强功能：3.4.0 在包括 proj.db 位置的 NETWORK_ENABLED、URL_ENDPOINT 和 DATABASE_PATH 返回 PROJ4 的版本号。
- **ST_AsSVG** - 增强：3.4.0 支持所有曲线型返回几何体的 SVG 路径数据。
- **ST_ClosestPoint** - 增强：3.4.0 - 支持地理。返回 g1 上最接近 g2 的 2D 点。是从一个几何体到一个几何体的最短直线的第一个点。
- **ST_LineSubstring** - 增强：3.4.0 - 引入了地理的支持。返回指定小数位置之线的直线部分。
- **ST_Project** - 增强：3.4.0 允许几何参数和无方位角的点格式。返回从起点按距离和方位角（方位角）投影的点。
- **ST_Resample** - 增强：3.4.0 添加了最大和最小重采样间隔重采样一个栅格图像，可以指定重新采样算法、新的尺寸、任意的栅格角点，以及一栅格地理参考属性，这些属性可以自己定义，也可以从一个栅格图像中借用。
- **ST_Rescale** - 增强：3.4.0 添加了最大和最小重采样间隔通过调整栅格的比例（或像素大小）来重新采样栅格。新的像素间隔是使用 NearestNeighbor（英国或美式拼写）、Bilinear、Cubic、CubicSpline、Lanczos、Max 或 Min 重采样算法计算的。默认为 NearestNeighbor。
- **ST_ShortestLine** - 增强：3.4.0 - 支持地理。返回两个几何图形之间的 2D 最短线

PostGIS 中的功能生成 3.4

- **PostGIS_Extensions_Upgrade** - 更改：3.4.0 添加 target_version 参数。将 PostGIS 扩展（例如 postgis_raster、postgis_topology、postgis_sfcgal）打包并升级到指定版本或最新版本。

13.12.5 PostGIS 新增功能或增强功能（3.3）

下面列出的功能是添加或增强的 PostGIS 功能。

PostGIS 中的新功能 3.3

- **RemoveUnusedPrimitives** - 可用性：3.3.0 移除 TopoGeometry 对象不需要的拓扑基元。
- **ST_AsMARC21** - 可用性：3.3.0 将几何对象返回具有地理数据字段 (034) 的 MARC21/XML 对象。
- **ST_GeomFromMARC21** - 可用性：3.3.0, 需要 libxml2 2.6+ 输入 MARC21/XML 地理数据并返回 PostGIS 几何对象。
- **ST_Letters** - 可用性：3.3.0 返回渲染几何对象的输入字母，默认起始位置位于原点，默认文本高度为 100。
- **ST_SimplifyPolygonHull** - 可用性：3.3.0。计算多边形几何的简化的保留拓扑的外部或内部外壳。
- **ST_TriangulatePolygon** - 可用性：3.3.0。计算多边形的约束 Delaunay 三角剖分
- **postgis_sfcgal_full_version** - 可用性：3.3.0 返回正在使用的 SFCGAL 的完整版本，包括 CGAL 和 Boost 版本

PostGIS 的功能增强 3.3

- **ST_ConcaveHull** - 增强：3.3.0, 使用 GEOS 3.11+ 使用 GEOS 本机函数计算包含所有输入几何点的可能凹几何
- **ST_LineMerge** - 增强：3.3.0 接受定向参数。返回通线将 MultiLineString 组合在一起形成的线。

PostGIS 中的功能生成化 3.3

- **PostGIS_Extensions_Upgrade** - 更改：3.3.0 支持从任何 PostGIS 版本升级。不适用于所有系统。将 PostGIS 扩展（例如 postgis_raster、postgis_topology、postgis_sfcgal）打包并升级到指定版本或最新版本。

13.12.6 PostGIS 新增功能或增强功能（3.2）

下面列出的功能是添加或增强的 PostGIS 功能。

PostGIS 中的新功能 3.2

- **FindLayer** - 可用性：3.2.0 通过不同的方式返回 topology.layer 对象。
- **FindTopology** - 可用性：3.2.0 通过不同的方式返回拓扑对象。
- **GetFaceContainingPoint** - 可用性：3.2.0 查找包含点的面。
- **ST_AsFlatGeobuf** - 可用性：3.2.0 返回一行行的 FlatGeobuf 表示形式。
- **ST_Contour** - 可用性：3.2.0 使用 GDAL 轮廓算法从提供的栅格波段生成一行矢量轮廓。
- **ST_DumpSegments** - 可用性：3.2.0 将几何对象中的各个线段返回一行 geometry_dump 行。
- **ST_FromFlatGeobuf** - 可用性：3.2.0 读取 FlatGeobuf 数据。
- **ST_FromFlatGeobufToTable** - 可用性：3.2.0 根据 FlatGeobuf 数据的 schema 建立一个表。
- **ST_InterpolateRaster** - 可用性：3.2.0 基于输入的 3 点集插值网格表面，使用 X 和 Y 在网格上定位点，并使用点的 Z 值作表面高程。
- **ST_SRID** - 可用性：3.2.0 返回拓扑几何的空参考系符。
- **ST_Scroll** - 可用性：3.2.0 更改组合串的起点。

- **ST_SetM** - 可用性：3.2.0 返回与输入几何图形具有相同 X/Y 坐标的几何图形，并使用请求的重采样算法将网格中的值复制到 M 维度。
- **ST_SetZ** - 可用性：3.2.0 返回与输入几何图形具有相同 X/Y 坐标的几何图形，并使用请求的重采样算法将网格中的值复制到 Z 维度。
- **TopoGeom_addTopoGeom** - 可用性：3.2 将一个 TopoGeometry 的元素添加到一个 TopoGeometry 的定义中。
- **ValidateTopologyRelation** - 可用性：3.2.0 返回有关无效拓扑关系的信息。
- **postgis.gdal_vsi_options** - 可用性：3.2.0 用于设置处理外部数据网格使用的字符串配置。

PostGIS 的功能增强 3.2

- **GetFaceByPoint** - 增强：3.2.0 更高效的实现和更清晰的契约，停止使用无效的拓扑。查找与点相交的面。
- **ST_ClusterKMeans** - 增强：3.2.0 支持 max_radius 使用 K 均值算法返回每个输入几何图形的簇 id 的窗口函数。
- **ST_MakeValid** - 增强：在 3.2.0 中，添加了可选的算法参数 “linework” 和 “structure”。需要 GEOS >= 3.10.0 或更高版本。在不丢失点的情况下使无效几何体有效。
- **ST_PixelAsCentroid** - 增强：3.2.0 Faster 在用 C 实现。返回像素表示的区域的中心（点几何）。
- **ST_PixelAsCentroids** - 增强：3.2.0 Faster 在用 C 实现。返回栅格波段的每个像素的中心（点几何）以及每个像素的 X、Y 和 Z 坐标。点几何是像素表示的区域的中心。
- **ST_Point** - 增强：3.2.0 srid 作可选的参数被添加。旧的安装需要与 ST_SetSRID 组合以在几何体上设置 srid。创建具有 X、Y 和 SRID 的点。
- **ST_PointM** - 增强：3.2.0 srid 作可选的参数被添加。旧的安装需要与 ST_SetSRID 组合以在几何体上设置 srid。创建具有 X、Y、M 和 SRID 的点。
- **ST_PointZ** - 增强：3.2.0 srid 作可选的参数被添加。旧的安装需要与 ST_SetSRID 组合以在几何体上设置 srid。创建具有 X、Y、Z 和 SRID 的点。
- **ST_PointZM** - 增强：3.2.0 srid 作可选的参数被添加。旧的安装需要与 ST_SetSRID 组合以在几何体上设置 srid。创建具有 X、Y、Z、M 和 SRID 的点。
- **ST_RemovePoint** - 增强：3.2.0 从串中删除一个点。
- **ST_RemoveRepeatedPoints** - 增强：3.2.0 返回除了重复点的几何图形。
- **ST_StartPoint** - 增强：3.2.0 返回所有几何图形的点。如果输入不是 LineString，则先前的行将返回 NULL。Returns the first point of a LineString, CircularLineString, or NURBS Curve.
- **ST_Value** - 增强：添加了 3.2.0 重新采样可选参数。返回指定列 x、行 y 像素或特定几何点指定波段的值。波段号从 1 开始，如果未指定，默认为 1。如果将参数 exclude_nodata_value 设置为 false，则所有像素都被与 no data 像素相交并返回其值。如果未设置参数 exclude_nodata_value，则从栅格的元数据中取值。
- **TopoGeo_AddLineString** - 增强：3.2.0 版本新增了返回符号化图形的功能。使用容差将串添加到有拓扑中，并在可能的情况下拆分面或面。

PostGIS 中的功能增强 3.2

- **ST_Boundary** - 更改：3.2.0 支持 TIN，不使用地理，不线性化曲线返回几何图形的边界。
- **ValidateTopology** - 更改：3.2.0 添加了可选的 bbox 参数，行面元和面连接。返回一个 validate_topology_returntype 对象，说明拓扑。

13.12.7 PostGIS 新增功能或增强功能 (3.1)

下面列出的功能是添加或增强的 PostGIS 功能。

PostGIS 中的新功能 3.1

- **ST_Hexagon** - 可用性：3.1.0 使用提供的尺寸和六边形网格空间内的元座返回一个六边形。
- **ST_HexagonGrid** - 可用性：3.1.0 返回一个完全覆盖几何参数界的六边形和元格索引。
- **ST_MaximumInscribedCircle** - 可用性：3.1.0。计算几何体中包含的最大圆。
- **ST_ReducePrecision** - 可用性：3.1.0。返回有效的几何图形，其点舍入到网格公差。
- **ST_Square** - 可用性：3.1.0 使用提供的尺寸大小和正方形网格空间内的元格座返回一个正方形。
- **ST_SquareGrid** - 可用性：3.1.0 返回一个完全覆盖几何参数界的网格正方形和元格索引。

PostGIS 的功能增强 3.1

- **ST_AsEWKT** - 增强：3.1.0 支持可精度参数。使用 SRID 元数据返回几何图形的已知文本 (WKT) 表示形式。
- **ST_ClusterKMeans** - 增强：3.1.0 支持 3D 几何和重用 K 均值算法返回每个输入几何图形的簇 id 的窗口函数。
- **ST_Difference** - 增强：3.1.0 接受 gridSize 参数。计算表示几何 A 中不与几何 B 相交的部分的几何。
- **ST_Intersection** - 增强：3.1.0 接受 gridSize 参数计算表示几何 A 和 B 的共享部分的几何。
- **ST_MakeValid** - 增强：3.1.0 除了具有 NaN 值的坐标。可以在不丢失点的情况下使无效几何体有效。
- **ST_Subdivide** - 增强：3.1.0 接受 gridSize 参数。计算几何体的直线段。
- **ST_SymDifference** - 增强：3.1.0 接受 gridSize 参数。计算表示几何图形 A 和 B 不相交部分的几何图形。
- **ST_TileEnvelope** - 增强：添加了 3.1.0 margin 参数。使用 XYZ 切片系在 Web Mercator (SRID : 3857) 中构建矩形多边形。
- **ST_UnaryUnion** - 增强：3.1.0 接受 gridSize 参数。计算一个几何体的部件的并集。
- **ST_Union** - 增强：3.1.0 接受 gridSize 参数。计算表示输入几何图形的点集并集的几何图形。

PostGIS 中的功能增强 3.1

- **ST_Count** - 更改：3.1.0 - 除了 ST_Count(rastertable, rastercolumn, ...) 函数。改用。返回一格或格覆盖范围的固定波段中的像素数。如果未指定 band，默认为 band 1。如果 except_nodata_value 置 true，则计算不等于 nodata 值的像素。
- **ST_Force3D** - 更改：3.1.0。您可以在指定一个非零 Z 值。强制几何图形输入 XYZ 模式。是 ST_Force3DZ 的别名。
- **ST_Force3DM** - 更改：3.1.0。您可以在指定一个非零 M 值。强制几何图形输入 XYM 模式。
- **ST_Force3DZ** - 更改：3.1.0。您可以在指定一个非零 Z 值。强制几何图形输入 XYZ 模式。
- **ST_Force4D** - 更改：3.1.0。您可以在指定非零 Z 和 M 值。强制几何图形输入 XYZM 模式。
- **ST_Histogram** - 更改：3.1.0 除了 ST_Histogram(table_name, column_name) 函数。返回一个数组，包括了格或格覆盖数据分布，其中包括分离的分箱范围。如果未指定，将自动计算分箱数。
- **ST_Quantile** - 更改：3.1.0 除了 ST_Quantile(table_name, column_name) 函数。计算本或体上下文中格或格表覆盖范围的分位数。因此，可以检查某个点是否位于格的 25%、50%、75% 百分位。
- **ST_SummaryStats** - 更改：3.1.0 ST_SummaryStats(rastertable, rastercolumn, ...) 函数已删除。改用。返回指定格或格覆盖中某个波段的统计摘要，包括像元数量 (count)、和 (sum)、均值 (mean)、标准差 (stddev)、最小值 (min) 和最大值 (max)。若未指定波段，默认使用第 1 波段。

13.12.8 PostGIS 新增功能或增强功能 (3.0)

下面列出的功能是添加或增强的 PostGIS 功能。

PostGIS 中的新功能 3.0

- **CG_ConstrainedDelaunayTriangles** - 可用性：3.0.0 返回指定输入几何体的约束 Delaunay 三角剖分。
- **ST_3DLineInterpolatePoint** - 可用性：3.0.0 返回沿 3D 线的小数指示位置插值的点。
- **ST_TileEnvelope** - 可用性：3.0.0 使用 XYZ 切片系在 Web Mercator (SRID : 3857) 中构建矩形多边形。

PostGIS 的功能增强 3.0

- **ST_AsMVT** - 增强：3.0 - 添加了要素 ID 的支持。返回一行的 MVT 表示形式的聚合函数。
- **ST_Contains** - 增强：3.0.0 使用了 GEOMETRYCOLLECTION 的支持 B 的每个点是否都位于 A 中，并且它的内部是否有一个共同点
- **ST_ContainsProperly** - 增强：3.0.0 使用了 GEOMETRYCOLLECTION 的支持 B 的每个点是否都位于 A 的内部
- **ST_CoveredBy** - 增强：3.0.0 使用了 GEOMETRYCOLLECTION 的支持 A 的每个点是否都位于 B 中
- **ST_Covers** - 增强：3.0.0 使用了 GEOMETRYCOLLECTION 的支持 B 的每个点是否都位于 A 中
- **ST_Crosses** - 增强：3.0.0 使用了 GEOMETRYCOLLECTION 的支持两个几何图形是否有一些（但不是全部）共同的内部点
- **ST_CurveToLine** - 增强：3.0.0 限制了每弧的最小曲率分数。防止拓扑崩溃。将包含曲线的几何图形转换为线性几何图形。
- **ST_Disjoint** - 增强：3.0.0 使用了 GEOMETRYCOLLECTION 的支持两个几何图形是否没有共同点
- **ST_Equals** - 增强：3.0.0 使用了 GEOMETRYCOLLECTION 的支持两个几何图形是否包含同一组点
- **ST_GeneratePoints** - 增强：3.0.0，添加种子参数生成一个包含在多边形 (Polygon) 或多重多边形 (Multi-Polygon) 内的随机点的多点对象。
- **ST_GeomFromGeoJSON** - 增强：3.0.0 如果未指定其他 SRID 解析几何，默认 SRID 为 4326。将几何图形的 geojson 表示形式作输入并输出 PostGIS 几何对象
- **ST_LocateBetween** - 增强：3.0.0 - 添加了多边形、TIN、三角形的支持。返回与量范围匹配的几何图形部分。
- **ST_LocateBetweenElevations** - 增强：3.0.0 - 添加了多边形、TIN、三角形的支持。返回位于高程 (Z) 范围内的几何图形部分。
- **ST_Overlaps** - 增强：3.0.0 使用了 GEOMETRYCOLLECTION 的支持两个几何图形是否具有相同的度和相交，但每个几何图形至少有一个点不在另一个几何图形中
- **ST_Relate** - 增强：3.0.0 使用了 GEOMETRYCOLLECTION 的支持两个几何图形是否具有与交集矩形模式匹配的拓扑关系，或计算它们的交集矩形
- **ST_Segmentize** - 增强：3.0.0 分段几何在可生成等长的子分段返回修改后的几何图形/地理，其段不大于指定距离。
- **ST_Touches** - 增强：3.0.0 使用了 GEOMETRYCOLLECTION 的支持两个几何图形是否至少有一个共同点，但它们的内部不相交
- **ST_Within** - 增强：3.0.0 使用了 GEOMETRYCOLLECTION 的支持 A 的每个点是否都位于 B 中，并且它的内部是否有一个共同点

PostGIS 中的功能增强 3.0

- **PostGIS_Extensions_Upgrade** - 更改：3.0.0 重新打包松散扩展并支持 `postgis_raster`。将 PostGIS 扩展（例如 `postgis_raster`、`postgis_topology`、`postgis_sfcgal`）打包并升级到固定版本或最新版本。
- **ST_3DDistance** - 更改：3.0.0 - SFCGAL 版本已移除返回两个几何图形之间的 3D 笛卡儿最小距离（基于空参考）（以投影位表示）。
- **ST_3DIntersects** - 更改：3.0.0 除了 SFCGAL 后端，GEOS 后端支持 TIN。两个几何图形在 3D 空间中是否相交 - 适用于点、串、多边形、多面体曲面（区域）
- **ST_Area** - 更改：3.0.0 - 不再依赖 SFCGAL。返回多边形几何体的面积。
- **ST_AsGeoJSON** - 更改：3.0.0 支持操作输入以 GeoJSON 格式返回一个几何体或要素。
- **ST_AsGeoJSON** - 更改：3.0.0 输出 SRID（如果不是 EPSG : 4326）。以 GeoJSON 格式返回一个几何体或要素。
- **ST_AsKML** - 更改：3.0.0 - 除了“版本化”体系名将几何图形作为 KML 元素返回。
- **ST_Distance** - 更改：3.0.0 - 不再依赖 SFCGAL。返回两个几何或地理图之间的距离。
- **ST_Intersection** - 更改：3.0.0 不依赖于 SFCGAL。计算表示几何 A 和 B 的共享部分的几何。
- **ST_Intersects** - 更改：3.0.0 除了 SFCGAL 版本并添加了 2D TINS 的本机支持。两个几何图形是否相交（它至少有一个共同点）
- **ST_Union** - 更改：3.0.0 不依赖于 SFCGAL。计算表示输入几何图形的点集并集的几何图形。

13.12.9 PostGIS 新增功能或增强功能（2.5）

下面列出的功能是添加或增强的 PostGIS 功能。

PostGIS 中的新功能 2.5

- **PostGIS_Extensions_Upgrade** - 可用性：2.5.0 将 PostGIS 扩展（例如 `postgis_raster`、`postgis_topology`、`postgis_sfcgal`）打包并升级到固定版本或最新版本。
- **ST_Angle** - 可用性：2.5.0 返回由 3 或 4 个点或 2 条定义的向量之间的角度。
- **ST_AsHexWKB** - 可用性：2.5.0 返回 Hex 的十六进制表示形式的熟知的二进制 (WKB)。
- **ST_BandFileSize** - 可用性：2.5.0 返回文件系中存储的波段的文件大小。如果未指定波段号，假定 1。
- **ST_BandFileTimestamp** - 可用性：2.5.0 返回文件系中存储的波段的文件戳。如果未指定波段号，假定 1。
- **ST_ChaikinSmoothing** - 可用性：2.5.0 使用 Chaikin 算法返回几何图形的平滑版本
- **ST_FilterByM** - 可用性：2.5.0 根据 M 删除点
- **ST_Grayscale** - 可用性：2.5.0 根据源栅格和代表色、色和色的指定波段新建新的 1-8BUI 波段栅格
- **ST_LineInterpolatePoints** - 可用性：2.5.0 返回沿直线以分数间隔插值的点。
- **ST_OrientedEnvelope** - 可用性：2.5.0。返回包含几何图形的最小面积矩形。
- **ST_QuantizeCoordinates** - 可用性：2.5.0 将坐标的最低有效位置零
- **ST_RastFromHexWKB** - 可用性：2.5.0 从熟知的二进制 (WKB) Hex 的十六进制表示形式返回栅格。
- **ST_RastFromWKB** - 可用性：2.5.0 从熟知的二进制 (WKB) Hex 返回栅格。
- **ST_SetBandIndex** - 可用性：2.5.0 更新 out-db band 的 external band 号

- **ST_SetBandPath** - 可用性：2.5.0 更新 out-db band 的外部路径和 band 号

PostGIS 的功能增强 2.5

- **ST_AsBinary/ST_AsWKB** - 增强：2.5.0 添加 ST_AsWKB 返回 WKB 格式的熟知的二进制 (WKB) 表示形式。
- **ST_AsMVT** - 增强：2.5.0 - 添加了并行支持。返回一行的 MVT 表示形式的聚合函数。
- **ST_AsText** - 增强：2.5 - 引入了可选项的精度参数。返回不带 SRID 元数据的几何/地理的已知文本 (WKT) 表示形式。
- **ST_BandMetaData** - 增强：2.5.0 包括 outdbbandnum、文件大小和文件偏移 for outdb 格式。返回特定格式波段的基本元数据。如果未指定，假定波段号 1。
- **ST_Buffer** - 增强：2.5.0 - ST_Buffer 的几何感知版本已得到增强，允许您指定要缓冲的一侧。side=both|left|right。计算覆盖距几何体指定距离内所有点的几何体。
- **ST_GeomFromGeoJSON** - 增强：2.5.0 现在可以接受 json 和 jsonb 作为输入。将几何形的 geojson 表示形式作为输入并输出 PostGIS 几何对象
- **ST_GeometricMedian** - 增强：2.5.0 添加了 M 作为点重的支持。返回多点的几何中位数。
- **ST_Intersects** - 增强：2.5.0 支持 GEOMETRYCOLLECTION。检查几个几何形是否相交（它们至少有一个共同点）
- **ST_OffsetCurve** - 增强：2.5 - 添加了 GEOMETRYCOLLECTION 和 MULTILINESTRING 的支持返回输入指定距离和方向的偏移。
- **ST_Scale** - 增强：2.5.0 引入了相对于本地原点（origin 参数）缩放的支持。按因子缩放几何形。
- **ST_Split** - 增强：2.5.0 引入了通过多边形分割的支持。返回通过将一个几何体分割成一个几何体而建立的几何体集合。
- **ST_Subdivide** - 增强：2.5.0 重用多边形分割上的点，点数从 8 减少到 5。计算几何体的直线分。

PostGIS 中的功能增强 2.5

- **ST_GDALDrivers** - 更改：2.5.0 - 添加 can_read 和 can_write 列。通过 GDAL 返回 PostGIS 支持的格式列表。ST_AsGDALRaster 可使用 can_write=True 的格式

13.12.10 PostGIS 新增功能或增强功能（2.4）

下面列出的功能是添加或增强的 PostGIS 功能。

PostGIS 中的新功能 2.4

- **ST_AsGeobuf** - 可用性：2.4.0 返回一行的 Geobuf 表示。
- **ST_AsMVT** - 可用性：2.4.0 返回一行的 MVT 表示形式的聚合函数。
- **ST_AsMVTGeom** - 可用性：2.4.0 将几何形转换为 MVT 瓦片的坐标空间。
- **ST_Centroid** - 可用性：2.4.0 引入了地理的支持。返回几何体的几何中心。
- **ST_ForcePolygonCCW** - 可用性：2.4.0 将所有外环逆时针定向，将所有内环顺时针定向。
- **ST_ForcePolygonCW** - 可用性：2.4.0 将所有外环顺时针定向，将所有内环逆时针定向。
- **ST_FrechetDistance** - 可用性：2.4.0 - 需要 GEOS >= 3.7.0 返回两个几何形之间的 Fréchet 距离。
- **ST_IsPolygonCCW** - 可用性：2.4.0 检查多边形是否具有逆时针方向的外环和顺时针方向的内环。

- **ST_IsPolygonCW** - 可用性：2.4.0 检查多边形是否具有外环和逆时针内环。
- **ST_MakeEmptyCoverage** - 可用性：2.4.0 用空网格覆盖地理参考区域。

PostGIS 的功能增强 2.4

- **ST_AsTWKB** - 增强：2.4.0 内存和速度改进。返回几何形式 TWKB，又名“微小的已知的二进制制”
- **ST_Covers** - 增强：2.4.0 地理类型添加了多边形中的多边形和多边形中的点的支持。检查 B 的每个点是否都位于 A 中
- **ST_CurveToLine** - 增强：2.4.0 支持最大距离公差和最大角度公差，支持称出。将包含曲线的几何形状转换为线性几何形状。
- **ST_Project** - 增强：2.4.0 允许距离和非规范化方位角。返回从起点按距离和方位角（方位角）投影的点。
- **ST_Reverse** - 增强：2.4.0 引入了曲线支持。返回点序相反的几何体。

PostGIS 中的功能生色化 2.4

- **=** - 更改：2.4.0，在之前的版本中，是边界相等而不是几何相等。如果需要边界相等，使用代替。如果几何/地理 A 的坐和坐序与几何/地理 B 的坐和坐序相同，返回 TRUE。
- **ST_Node** - 更改：2.4.0 函数在内部使用 GEOSNode 而不是 GEOSUnaryUnion。与 PostGIS < 2.4 相比，可能会致生成的串具有不同的序和方向。点是点的集合。

13.12.11 PostGIS 新增功能或增强功能（2.3）

下面列出的功能是添加或增强的 PostGIS 功能。

PostGIS 中的新功能 2.3

- **&&&(geometry,gidx)** - 可用性：2.3.0 - 引入了 BRIN 的支持。如果几何体的（存的）n 边界与 n 浮点精度边界 (GIDX) 相交，返回 TRUE。
- **&&&(gidx,geometry)** - 可用性：2.3.0 - 引入了 BRIN 的支持。如果 n 浮点精度边界 (GIDX) 与几何体的（存的）n 边界相交，返回 TRUE。
- **&&&(gidx,gidx)** - 可用性：2.3.0 - 引入了 BRIN 的支持。如果 n 浮点精度边界 (GIDX) 彼此相交，返回 TRUE。
- **&&(box2df,box2df)** - 可用性：2.3.0 - 引入了 BRIN 的支持。如果 n 2D 浮点精度边界 (BOX2DF) 彼此相交，返回 TRUE。
- **&&(box2df,geometry)** - 可用性：2.3.0 - 引入了 BRIN 的支持。如果 2D 浮点精度边界 (BOX2DF) 与几何体的（存的）2D 边界相交，返回 TRUE。
- **&&(geometry,box2df)** - 可用性：2.3.0 - 引入了 BRIN 的支持。如果几何体的（存的）2D 边界与 2D 浮点精度边界 (BOX2DF) 相交，返回 TRUE。
- **@(box2df,box2df)** - 可用性：2.3.0 - 引入了 BRIN 的支持。如果 2D 浮点精度边界 (BOX2DF) 包含在一个 2D 浮点精度边界内，返回 TRUE。
- **@(box2df,geometry)** - 可用性：2.3.0 - 引入了 BRIN 的支持。如果 2D 浮点精度边界 (BOX2DF) 包含在几何体的 2D 边界中，返回 TRUE。
- **@(geometry,box2df)** - 可用性：2.3.0 - 引入了 BRIN 的支持。如果几何体的 2D 边界包含在 2D 浮点精度边界 (BOX2DF) 中，返回 TRUE。
- **Populate_Topology_Layer** - 可用性：2.3.0 通过从拓扑表取元数据，将缺失的条目添加到 topology.layer 表中。

- **ST_ClusterDBSCAN** - 可用性：2.3.0 使用 DBSCAN 算法返回每个输入几何图形的簇 id 的窗口函数。
- **ST_ClusterKMeans** - 可用性：2.3.0 使用 K 均值算法返回每个输入几何图形的簇 id 的窗口函数。
- **ST_GeneratePoints** - 可用性：2.3.0 生成一个包含在多边形（Polygon）或多重多边形（MultiPolygon）内的随机点的多点对象。
- **ST_GeometricMedian** - 可用性：2.3.0 返回多点的几何中位数。
- **ST_MakeLine** - 可用性：2.3.0 - 引入了 MultiPoint 输入元素的支持从 Point, MultiPoint, 或 LineString geometries 构建 LineString。
- **ST_MinimumBoundingRadius** - 可用性 - 2.3.0 返回包含几何图形的最小圆的中心点和半径。
- **ST_MinimumClearance** - 可用性：2.3.0 返回几何体的最小间隙，它是几何体健壮性的度量。
- **ST_MinimumClearanceLine** - 可用性：2.3.0-需要 GEOS >= 3.6.0 返回跨越几何体最小间隙的线段串。
- **ST_Normalize** - 可用性：2.3.0 返回规范形式的几何图形。
- **ST_Points** - 可用性：2.3.0 返回包含几何坐标的 MultiPoint。
- **ST_VoronoiLines** - 可用性：2.3.0 返回几何体点的 Voronoi 图的边界。
- **ST_VoronoiPolygons** - 可用性：2.3.0 返回几何体点的 Voronoi 图的图元格。
- **ST_WrapX** - 可用性：2.3.0 需要 GEOS 将几何体图元在 X 轴周围。
- **TopoGeom_addElement** - 可用性：2.3 将元素添加到 TopoGeometry 的定图中。
- **TopoGeom_remElement** - 可用性：2.3 从 TopoGeometry 的定图中删除元素。
- **~(box2df,box2df)** - 可用性：2.3.0 - 引入了 BRIN 的支持。如果 2D 浮点精度图元框 (BOX2DF) 包含一个 2D 浮点精度图元框 (BOX2DF)，则返回 TRUE。
- **~(box2df,geometry)** - 可用性：2.3.0 - 引入了 BRIN 的支持。如果 2D 浮点精度图元框 (BOX2DF) 包含几何体的 2D 图元框，则返回 TRUE。
- **~(geometry,box2df)** - 可用性：2.3.0 - 引入了 BRIN 的支持。如果几何体的 2D 粘合框包含 2D 浮点精度图元框 (GIDX)，则返回 TRUE。

PostGIS 的功能增强 2.3

- **ST_Contains** - 增强：2.3.0 PIP 短路（快速判断限于多边形和点）已得到增强，以支持具有更少点的多点。以前的版本支持面和点组合。图元 B 的每个点是否都位于 A 中，并且它的内部是否有一个共同点
- **ST_Covers** - 增强：图元于 2.3.0 几何图形，PIP 短路（限于多边形和点的快速判断）已得到增强，以支持由更少点组成的多点。以前的版本支持面和点组合。图元 B 的每个点是否都位于 A 中
- **ST_Expand** - 增强：2.3.0 添加了不同图元度的盒子图元行不同数量图元展的支持。返回从一个图元框或几何图形图元展的图元框。
- **ST_Intersects** - 增强：2.3.0 PIP 短路（快速判断限于多边形和点）已得到增强，以支持具有更少点的多点。以前的版本支持面和点组合。图元图元个几何图形是否相交（它至少有一个共同点）
- **ST_Segmentize** - 增强：2.3.0 地理分段图元在可生成等图元的子分段返回修改后的几何图形/地理，其图元段不图元于图元定距离。
- **ST_Transform** - 增强：2.3.0 引入了直接 PROJ.4 字符串的支持。返回坐图元图元图元不同空图元参考系的新几何图形。
- **ST_Within** - 增强：图元于 2.3.0 几何图形，PIP 短路（限于多边形和点的快速判断）已得到增强，以支持由更少点组成的多点。以前的版本支持面和点组合。图元 A 的每个点是否都位于 B 中，并且它的内部是否有一个共同点

PostGIS 中的功能图元生图元化 2.3

- **ST_PointN** - 更改：2.3.0：图元索引可用（-1 是最后一点）返回几何图形中第一个图元串或图元形图元串中的第 N 个点。

13.12.12 PostGIS 新增功能或增强功能（2.2）

下面列出的功能是添加或增强的 PostGIS 功能。

PostGIS 中的新功能 2.2

- **<->** - 可用性：2.2.0。返回 A 和 B 几何图形或边界框之间的 n 距离。
- **ST_AsEncodedPolyline** - 可用性：2.2.0 从 LineString 几何体返回折线。
- **ST_AsTWKB** - 可用性：2.2.0 返回几何形式 TWKB，又名“微小的已知的二进制”
- **ST_BoundingDiagonal** - 可用性：2.2.0 返回几何边界框的对角线。
- **ST_CPAWithin** - 可用性：2.2.0 检查线条迹的最近接近点是否在指定距离内。
- **ST_ClipByBox2D** - 可用性：2.2.0 计算几何图形落在矩形内的部分。
- **ST_ClosestPointOfApproach** - 可用性：2.2.0 返回线条迹最接近点的测量。
- **ST_ClusterIntersecting** - 可用性：2.2.0 将输入几何图形聚集成交集的聚合函数。
- **ST_ClusterWithin** - 可用性：2.2.0 按间隔距离几何图形行聚合的聚合函数。
- **ST_CountAgg** - 可用性：2.2.0 检查的。返回一网格的固定波段中的像素数。如果未指定 band，默认 band 1。如果 except_nodata_value 设置 true，则计算不等于 NODATA 的像素。
- **ST_CreateOverview** - 可用性：2.2.0 构建固定格覆盖范围的降低分辨率版本。
- **ST_DistanceCPA** - 可用性：2.2.0 返回线条迹的最近接近点之间的距离。
- **ST_ForceCurve** - 可用性：2.2.0 如果适用，将一个几何图形上升到其曲线型。
- **ST_IsValidTrajectory** - 可用性：2.2.0 检查几何图形是否有效迹。
- **ST_LineFromEncodedPolyline** - 可用性：2.2.0 从折线构建 LineString。
- **ST_MapAlgebra (callback function version)** - 可用性：2.2.0：能够添加遮罩回函数版本 - 指定一个或多个输入格、波段索引和一个用指定的回函数，返回波段格。
- **ST_MemSize** - 可用性：2.2.0 返回格占用的空间量（以字节位）。
- **ST_RemoveRepeatedPoints** - 可用性：2.2.0 返回除了重复点的几何图形。
- **ST_Retile** - 可用性：2.2.0 从任意平面的格覆盖范围返回一配置的平面。
- **ST_SetEffectiveArea** - 可用性：2.2.0 使用 Visvalingam-Whyatt 算法设置每个点的有效区域。
- **ST_SimplifyVW** - 可用性：2.2.0 使用 Visvalingam-Whyatt 算法返回几何图形的简化表示
- **ST_Subdivide** - 可用性：2.2.0 计算几何体的直线分。
- **ST_SummaryStatsAgg** - 可用性：2.2.0 聚合函数。返回一网格中指定波段的摘要，包括像元数量 (count)、和 (sum)、均值 (mean)、标准差 (stddev)、最小值 (min) 和最大值。若未指定波段，默认使用第 1 波段。
- **ST_SwapOrdinates** - 可用性：2.2.0 返回更改后的几何图形，其中交换了坐标。
- **postgis.enable_outdb_rasters** - 可用性：2.2.0 一个布尔配置项，用于启用数据外部格波段的项。
- **postgis.gdal_datapath** - 可用性：2.2.0 用于分配 GDAL 的 GDAL_DATA 项的配置项。如果未设置，使用环境设置的 GDAL_DATA 量。
- **postgis.gdal_enabled_drivers** - 可用性：2.2.0 用于设置 PostGIS 环境中使用的 GDAL 程序的配置项。影响 GDAL 配置量 GDAL_SKIP。
- **|=|** - 可用性：2.2.0。返回 A 和 B 迹在最接近点之间的距离。

PostGIS 的功能增强 2.2

- **<->** - Enhanced: 2.2.0 -- 支持 geometry 和 geography 的真 KNN 行。注意 geography 的 KNN 基于球体而非地球体。返回 A 和 B 之间的 2D 距离。
- **AsTopoJSON** - 增强: 2.2.1 添加了 puntal 入的支持返回拓扑几何的 TopoJSON 表示形式。
- **ST_Area** - 增强: 2.2.0 - 使用 GeographicLib 地球体行量, 以提高准确性和健壮性。需要 PROJ >= 4.9.0 才能利用新功能。返回多边形几何体的面积。
- **ST_AsX3D** - 增强: 2.2.0 : 添加了反地理坐标和 (x/y、度/度) 的支持。有关信息, 参。返回 X3D xml 点元素格式的几何形: ISO-IEC-19776-1.2-X3DEncodings-XML
- **ST_Azimuth** - 增强: 2.2.0 使用 GeographicLib 地球体行量, 以提高准确性和健壮性。需要 PROJ >= 4.9.0 才能利用新功能。返回点之间的直线的基于北方的方位角。
- **ST_Distance** - 增强: 2.2.0 - 使用 GeographicLib 地球体行量, 以提高准确性和健壮性。需要 PROJ >= 4.9.0 才能利用新功能。返回几何或地理之间的距离。
- **ST_Scale** - 增强: 2.2.0 引入了缩放所有度 (factor 参数) 的支持。按因子缩放几何形。
- **ST_Split** - 增强: 2.2.0 引入了通多、多点或 (多) 多边形界分割的支持。返回通将一个几何体分割成一个几何体而建的几何体集合。
- **ST_Summary** - 增强: 添加了 2.2.0 TIN 和曲的支持返回几何内容的文本摘要。

PostGIS 中的功能生 2.2

- **<->** - Changed: 2.2.0 -- 在提供真 KNN 之后, 旧的混合法通方式可能会更慢。参下方示例。返回 A 和 B 之间的 2D 距离。
- **ST_3DClosestPoint** - 更改: 2.2.0 - 如果入 2 个 2D 几何形, 返回 2D 点 (而不是假缺失 Z 0 的旧行)。在 2D 和 3D 情况下, 于缺失的 Z, Z 不再被假定 0。返回 g1 上最接近 g2 的 3D 点。是 3D 最短的第一个点。
- **ST_3DDistance** - 更改: 2.2.0 - 在 2D 和 3D 的情况下, 于缺失的 Z, Z 不再被假定 0。返回几何形之间的 3D 笛卡最小距离 (基于空参考) (以投影位表示)。
- **ST_3DLongestLine** - 更改: 2.2.0 - 如果入 2 个 2D 几何形, 返回 2D 点 (而不是假缺失 Z 0 的旧行)。在 2D 和 3D 情况下, 于缺失的 Z, Z 不再被假定 0。返回几何体之间的 3D 最直
- **ST_3DMaxDistance** - 更改: 2.2.0 - 在 2D 和 3D 的情况下, 于缺失的 Z, Z 不再被假定 0。返回几何形之间的 3D 笛卡最大距离 (基于空参考) (以投影位表示)。
- **ST_3DShortestLine** - 更改: 2.2.0 - 如果入 2 个 2D 几何形, 返回 2D 点 (而不是假缺失 Z 0 的旧行)。在 2D 和 3D 情况下, 于缺失的 Z, Z 不再被假定 0。返回几何形之间的 3D 最短
- **ST_DistanceSphere** - 更改: 2.2.0 在之前的版本中, 曾被称 ST_Distance_Sphere 使用球形地球模型返回度/度几何形状之间的最小距离 (以米位)。
- **ST_DistanceSpheroid** - 更改: 2.2.0 在之前的版本中, 称 ST_Distance_Spheroid 使用球体模型返回度/度几何形状之间的最小距离。
- **ST_Equals** - 更改: 2.2.0 即使于无效几何形, 如果它二制相等, 也会返回 true 个几何形是否包含同一点
- **ST_LengthSpheroid** - 更改: 2.2.0 在之前的版本中, 称 ST_Length_Spheroid 并具有名 ST_3DLength_Sphere 返回球体上度/度几何体的 2D 或 3D 度/周。
- **ST_MemSize** - 更改: 2.2.0 名称更改 ST_MemSize 以遵循命名定。返回几何形占用的内存空量。
- **ST_PointInsideCircle** - 更改: 2.2.0 在之前的版本中, 称 ST_Point_Inside_Circle 点几何形是否位于由心和半径定的内
- **ValidateTopology** - 更改: 2.2.0 id1 和 id2 的被交 “交叉点”, 与描述一致。返回一 validate_topology_return_type 象, 明拓扑。

13.12.13 PostGIS 新增功能或增强功能（2.1）

下面列出的功能是添加或增强的 PostGIS 功能。

PostGIS 中的新功能 2.1

- **=** - 可用性：2.1.0 如果 A 的边界框与 B 的边界框相同，返回 TRUE。使用双精度边界框。
- **AsTopoJSON** - 可用性：2.1.0 返回拓扑几何的 TopoJSON 表示形式。
- **ST_Box2dFromGeoHash** - 可用性：2.1.0 从 GeoHash 字符串返回 BOX2D。
- **ST_ColorMap** - 可用性：2.1.0 根据源栅格和指定波段创建最多四个 8BUI 波段（灰度、RGB、RGBA）的新栅格。如果未指定，假定波段 1。
- **ST_Contains** - 可用性：2.1.0 如果栅格 rastB 中没有点位于栅格 rastA 的外部且 rastB 的内部至少有一个点位于 rastA 的内部，返回 true。
- **ST_ContainsProperly** - 可用性：2.1.0 如果 rastB 与 rastA 的内部相交，但不与 rastA 的边界或外部相交，返回 true。
- **ST_CoveredBy** - 可用性：2.1.0 如果栅格 rastA 中没有点位于栅格 rastB 之外，返回 true。
- **ST_Covers** - 可用性：2.1.0 如果栅格 rastB 中没有点位于栅格 rastA 之外，返回 true。
- **ST_DFullyWithin** - 可用性：2.1.0 如果栅格 rastA 和 rastB 彼此完全在指定距离内，返回 true。
- **ST_DWithin** - 可用性：2.1.0 如果栅格 rastA 和 rastB 彼此之间的距离在指定距离内，返回 true。
- **ST_DelaunayTriangles** - 可用性：2.1.0 返回几何体点的 Delaunay 三角剖分。
- **ST_Disjoint** - 可用性：2.1.0 如果栅格 rastA 在空集上不与 rastB 相交，返回 true。
- **ST_DumpValues** - 可用性：2.1.0 提取指定 band（波段）的栅格值数组。
- **ST_ForceSFS** - 可用性：2.1.0 Force geometries to use SFS 1.1 or 1.2 geometry types.
- **ST_FromGDALRaster** - 可用性：2.1.0 从受支持的 GDAL 栅格文件返回栅格。
- **ST_GeomFromGeoHash** - 可用性：2.1.0 从 GeoHash 字符串返回几何图形。
- **ST_InvDistWeight4ma** - 可用性：2.1.0 从像素的区域内插像素的栅格处理函数。
- **ST_MapAlgebra (callback function version)** - 可用性：2.1.0 回调函数版本 - 指定一个或多个输入栅格、波段索引和一个用回调指定的回调函数，返回波段栅格。
- **ST_MapAlgebra (expression version)** - 可用性：2.1.0 表达式版本 - 在指定一个输入栅格、波段索引和一个或多个用回调指定的 SQL 表达式式的情况下，返回波段栅格。
- **ST_MinConvexHull** - 可用性：2.1.0 返回栅格的凸包几何形状（不包括 NODATA 像素）。
- **ST_MinDist4ma** - 可用性：2.1.0 返回感兴趣像素与具有值的相邻像素之间的最小距离（以像素数单位）的栅格处理函数。
- **ST_NearestValue** - 可用性：2.1.0 返回由 columnx 和 rowy 指定的固定像素的最接近的非 NODATA 或具有与栅格相同的空参考坐标系表示的几何点。
- **ST_Neighborhood** - 可用性：2.1.0 返回固定波段像素周围非 NODATA 值的二值双精度数组，指定波段像素由 columnX 和 rowY 或与栅格相同的空参考坐标系表示的几何点指定。
- **ST_NotSameAlignmentReason** - 可用性：2.1.0 返回说明栅格是否不同的文本，如果未不同，说明原因。
- **ST_Overlaps** - 可用性：2.1.0 如果栅格 rastA 和 rastB 相交，但其中一个不完全包含另一个，返回 true。
- **ST_PixelAsCentroid** - 可用性：2.1.0 返回像素表示的区域的中心（点几何）。

- **ST_PixelAsCentroids** - 可用性：2.1.0 返回栅格波段的每个像素的几何中心（点几何）以及每个像素的 X、Y 栅格坐标。点几何是像素表示的区域的几何中心。
- **ST_PixelAsPoint** - 可用性：2.1.0 返回像素左上角的点几何形状。
- **ST_PixelAsPoints** - 可用性：2.1.0 返回栅格波段的每个像素的点几何形状以及每个像素的 X、Y 栅格坐标。点几何的坐标是像素的左上角。
- **ST_PixelOfValue** - 可用性：2.1.0 返回等于搜索值的像素的列 x、行 y 坐标。
- **ST_PointFromGeoHash** - 可用性：2.1.0 从 GeoHash 字符串返回一个点。
- **ST_RasterToWorldCoord** - 可用性：2.1.0 在给定列和行的情况下，以几何 X 和 Y（经度和纬度）形式返回栅格的左上角。列和行从 1 开始。
- **ST_Resize** - 可用性：2.1.0 需要 GDAL 1.6.1+ 将栅格大小调整新的宽度/高度
- **ST_Roughness** - 可用性：2.1.0 返回一个计算出的数字高程模型（DEM）的“粗糙度”的栅格。
- **ST_SetValues** - 可用性：2.1.0 返回通过设置指定波段的值而生的修改后的栅格。
- **ST_Simplify** - 可用性：2.1.0 使用 Douglas-Peucker 算法返回给定 TopoGeometry 的“简化”几何版本。
- **ST_Summary** - 可用性：2.1.0 返回栅格内容的文本摘要。
- **ST_TPI** - 可用性：2.1.0 返回一个计算出的地形位置指数（Topographic Position Index）的栅格。
- **ST_TRI** - 可用性：2.1.0 返回具有计算出的地形稳定性指数的栅格。
- **ST_Tile** - 可用性：2.1.0 返回根据输出栅格的所需宽度分割输入栅格而生的一组栅格。
- **ST_Touches** - 可用性：2.1.0 如果栅格 rastA 和 rastB 至少有一个共同点但它们的内部不相交，返回 true。
- **ST_Union** - 可用性：2.1.0 引入了 ST_Union(rast, unionarg) 函数。将一组栅格切片的并集返回由 1 个或多个波段组成的一个栅格。
- **ST_Within** - 可用性：2.1.0 如果栅格 rastA 中没有点位于栅格 rastB 的外部且 rastA 的内部至少有一个点位于 rastB 的内部，返回 true。
- **ST_WorldToRasterCoord** - 可用性：2.1.0 给定几何 X 和 Y（经度和纬度）或以栅格的空参考坐标系表示的点几何，将左上角作为列和行返回。
- **UpdateRasterSRID** - 可用性：2.1.0 更改用指定的列和表中所有栅格的 SRID。
- **clearTopoGeom** - 可用性：2.1 清除拓扑几何的内容。
- **postgis_sfcgal_version** - 可用性：2.1.0 返回正在使用的 SFCGAL 版本

PostGIS 的功能增强 2.1

- **ST_AddBand** - 增强：2.1.0 添加了 addbandarg 的支持。返回一个栅格，其中在指定索引位置添加了指定类型的新波段和指定初始值。如果未指定索引，将值添加到末尾。
- **ST_AddBand** - 增强：2.1.0 添加了新的 out-db 波段的支持。返回一个栅格，其中在指定索引位置添加了指定类型的新波段和指定初始值。如果未指定索引，将值添加到末尾。
- **ST_AsBinary/ST_AsWKB** - 增强：2.1.0 添加 outasin 返回栅格的熟知的二进制 (WKB) 表示形式。
- **ST_AsGML** - 增强：2.1.0 引入了 2.1.0 id 支持。将几何形状作为 GML 版本 2 或 3 元素返回。
- **ST_Aspect** - 增强：2.1.0 使用 ST_MapAlgebra() 并添加可选项 interpolate_nodata 函数参数返回高程栅格波段的坡向（默认为度为单位）。用于分析地形很有用。
- **ST_Boundary** - 增强：引入了 2.1.0 三角函数支持返回几何形状的边界。

- **ST_Clip** - 增修：2.1.0 用 C 重写返回由输入几何图形裁剪的栅格。如果未指定波段号，则处理所有波段。如果未指定 **crop** 或其值为 TRUE，则输出栅格将被裁剪。如果 **touched** 置为 TRUE，则包括接触到的像素；否则，当像素的中心位于几何图形内时，才会包括该像素。
- **ST_DWithin** - 增修功能：2.1.0 版本提升了地理（Geography）类型的速度。有关更多信息，参见 **Making Geography faster**。返回指定几何图形是否在指定距离内
- **ST_DWithin** - 增修：2.1.0 引入了对弯曲几何形状的支持。返回指定几何图形是否在指定距离内
- **ST_Distance** - 增修功能：2.1.0 版本提升了地理（Geography）类型的速度。有关更多信息，参见 **Making Geography faster**。返回指定几何或地理图形的距离。
- **ST_Distance** - 增修：2.1.0 - 引入了对弯曲几何形状的支持。返回指定几何或地理图形的距离。
- **ST_Distinct4ma** - 增修：2.1.0 添加格式 2 栅格处理函数，用于计算域中唯一像素的数量。
- **ST_DumpPoints** - 增修：2.1.0 速度更快。重新实现原生 C 语言。返回几何图形中坐落的一行 **geometry_dump** 行。
- **ST_HillShade** - 增修：2.1.0 使用 **ST_MapAlgebra()** 并添加可选项的 **interpolate_nodata** 函数参数使用提供的方位角、高度、亮度和比例输入返回高程栅格的假照明。
- **ST_MakeValid** - 增修：2.1.0 添加了对几何集合和多点的支持。在不丢失点的情况下使无效几何体有效。
- **ST_Max4ma** - 增修：2.1.0 添加格式 2 计算域中最大像素值的栅格处理函数。
- **ST_Mean4ma** - 增修：2.1.0 添加格式 2 计算域中平均像素值的栅格处理函数。
- **ST_Min4ma** - 增修：2.1.0 添加格式 2 计算域中最小像素值的栅格处理函数。
- **ST_PixelAsPolygons** - 增修：2.1.0 添加了 **except_nodata_value** 可选项参数。返回包含栅格的每个像素的多边形几何图形以及每个像素的 X、Y 和 Z 坐标。
- **ST_Polygon** - 增修：2.1.0 提高速度（完全基于 C）并确保返回的多边形有效。返回由具有非空数据的像素组成的像素并集形成的多边形几何体。如果未指定波段号，则波段号默认为 1。
- **ST_Range4ma** - 增修：2.1.0 添加格式 2 计算域中像素范围的栅格处理函数。
- **ST_SameAlignment** - 增修：2.1.0 添加聚合体如果栅格具有相同的斜率、比例、空值参考和偏移（像素可以放在同一网格上而不切割成像素），则返回 **true**；如果没有注意这些属性，则返回 **false**。
- **ST_Segmentize** - 增修：2.1.0 引入了对地理的支持。返回修改后的几何图形/地理，其线段不小于指定距离。
- **ST_SetGeoReference** - 增修：2.1.0 添加 **ST_SetGeoReference(raster, double precision, ...)** 函数在一次调用中设置 Georeference 6 地理配准参数。数字输入用空格分隔。接受 GDAL 或 ESRI 格式的输入。默认为 GDAL。
- **ST_SetValue** - 增修：2.1.0 **ST_SetValue()** 的几何体现在支持任何几何类型，而不仅仅是点。几何体是 **ST_SetValues()** 的 **geomval[]** 体的包装返回修改后的栅格，其效果是将指定波段中的值指定列 x、行 y 像素或与特定几何图形相交的像素。波段号从 1 开始，如果未指定波段，则默认为 1。
- **ST_Slope** - 增修：2.1.0 使用 **ST_MapAlgebra()** 并添加可选项 **units**、**scale**、**interpolate_nodata** 函数参数返回高程栅格的坡度（默认为度/分）。用于分析地形很有用。
- **ST_StdDev4ma** - 增修：2.1.0 添加格式 2 计算域中像素值的标准偏差的栅格处理函数。
- **ST_Sum4ma** - 增修：2.1.0 添加格式 2 栅格处理函数，计算域中所有像素值的和。
- **ST_Summary** - 增修：添加了 S 标志以指示其是否具有 2.1.0 空值参考系返回几何内容的文本摘要。
- **ST_Transform** - 增修：2.1.0 添加 **ST_Transform(rast, alignto)** 函数使用指定的重采样算法将已知空值参考系中的栅格重新投影到一个已知空值参考系。选项有 **NearestNeighbor**、**Bilinear**、**Cubic**、**CubicSpline**、**Lanczos**（默认为 **NearestNeighbor**）。
- **ST_Union** - 增修：2.1.0 提高速度（完全基于 C）。将一行栅格切片的并集返回由 1 个或多个波段组成的一个栅格。

- **ST_Union** - 增修：2.1.0 ST_Union(rast) (函数 1) 合并所有输入栅格的所有波段。PostGIS 的早期版本采用第一个波段。将一栅格切片的并集返回由 1 个或多个波段组成的栅格。
- **ST_Union** - 增修：2.1.0 ST_Union(rast, uniontype) (函数 4) 合并所有输入栅格的所有波段。将一栅格切片的并集返回由 1 个或多个波段组成的栅格。
- **toTopoGeom** - 增修：2.1.0 添加了采用具有 TopoGeometry 的版本。将几何图形转换为拓扑几何图形。

PostGIS 中的功能增强 2.1

- **ST_Aspect** - 更改：2.1.0 在之前的版本中，返回以弧度为单位。现在，返回默认度数返回高程栅格波段的坡向（默认以度为单位）。对于分析地形很有用。
- **ST_EstimatedExtent** - 更改：2.1.0。在 2.0.x 之前，名称 ST_Estimated_Extent。返回空表的估计范围。
- **ST_Force2D** - 更改：2.1.0 在 2.0.x 期间，它被称为 ST_Force_2D。强制几何图形进入“二维模式”。
- **ST_Force3D** - 更改：2.1.0 在 2.0.x 期间，它被称为 ST_Force_3D。强制几何图形进入 XYZ 模式。它是 ST_Force3DZ 的别名。
- **ST_Force3DM** - 更改：2.1.0。在 2.0.x 期间，它被称为 ST_Force_3DM。强制几何图形进入 XYM 模式。
- **ST_Force3DZ** - 更改：2.1.0。在 2.0.x 期间，它被称为 ST_Force_3DZ。强制几何图形进入 XYZ 模式。
- **ST_Force4D** - 更改：2.1.0。在 2.0.x 期间，它被称为 ST_Force_4D。强制几何图形进入 XYZM 模式。
- **ST_ForceCollection** - 更改：2.1.0。在 2.0.x 期间，它被称为 ST_Force_Collection。将几何图形转换为几何集合 (GEOMETRYCOLLECTION)。
- **ST_HillShade** - 更改：2.1.0 在之前的版本中，方位角和高度以弧度表示。现在，方位角和高度以度表示使用提供的方位角、高度、亮度和比例返回高程栅格的假照明。
- **ST_LineInterpolatePoint** - 更改：2.1.0。在 2.0.x 之前，名称 ST_Line_Interpolate_Point。返回沿在百分比指示位置的插点。
- **ST_LineLocatePoint** - 更改：2.1.0。在 2.0.x 之前，名称 ST_Line_Locate_Point。返回线上最接近点的分数位置。
- **ST_LineSubstring** - 更改：2.1.0。在 2.0.x 之前，名称 ST_Line_Substring。返回一个小数位置之间的直线部分。
- **ST_PixelAsCentroids** - 更改：2.1.1 更改了 except_nodata_value 的行。返回栅格波段的每个像素的几何（点几何）以及每个像素的 UTM、X 和 Y 栅格坐标。点几何是像素表示的区域的几何。
- **ST_PixelAsPoints** - 更改：2.1.1 更改了 except_nodata_value 的行。返回栅格波段的每个像素的点几何以及每个像素的 UTM、X 和 Y 栅格坐标。点几何的坐标是像素的左上角。
- **ST_PixelAsPolygons** - 更改：2.1.1 更改了 except_nodata_value 的行。返回包含栅格的每个像素的多边形几何以及每个像素的 UTM、X 和 Y 栅格坐标。
- **ST_Polygon** - 更改：2.1.0 在之前的版本中有可能会返回多边形，更改后始终返回多多边形。返回由具有非无数据值的像素并集形成的多多边形几何体。如果未指定波段号，波段号默认为 1。
- **ST_RasterToWorldCoordX** - 更改：2.1.0 在之前的版本中，名称 ST_Raster2WorldCoordX 返回栅格、列和行左上角的几何 X 坐标。列和行的编号从 1 开始。
- **ST_RasterToWorldCoordY** - 更改：2.1.0 在之前的版本中，名称 ST_Raster2WorldCoordY 返回栅格、列和行的左上角的几何 Y 坐标。列和行的编号从 1 开始。
- **ST_Rescale** - 更改：2.1.0 适用于没有 SRID 的栅格通过调整栅格的比例（或像素大小）来重新采样栅格。新的像素是使用 NearestNeighbor（英语或美式拼写）、Bilinear、Cubic、CubicSpline、Lanczos、Max 或 Min 重采样算法计算的。默认为 NearestNeighbor。

- **ST_Reskew** - 更改：2.1.0 适用于没有 SRID 的栅格通栅格整栅格的斜（或旋参数）来重采栅格。新的像素是使用 NearestNeighbor（英或美式拼写）、Bilinear、Cubic、CubicSpline 或 Lanczos 重采算法算的。默 NearestNeighbor。
- **ST_Segmentize** - 更改：2.1.0 由于引入了地理支持，使用 ST_Segmentize('LINESTRING(1 2, 3 4)', 0.5) 会致不明确的函数。入需要正确入几何或地理。使用 ST_GeomFromText、ST_GeogFromText 或所需型（例如 ST_Segmentize('LINESTRING(1 2, 3 4)::geometry, 0.5)）返回修改后的几何形/地理，其段不于定距离。
- **ST_Slope** - 更改：2.1.0 在之前的版本中，返回以弧度位。在，返回默度数返回高程格的坡度（默以度位）。于分析地形很有用。
- **ST_SnapToGrid** - 更改：2.1.0 适用于没有 SRID 的栅格通将栅格捕捉到网格来重采栅格。新的像素是使用 NearestNeighbor（英或美式拼写）、Bilinear、Cubic、CubicSpline 或 Lanczos 重采算法算的。默 NearestNeighbor。
- **ST_WorldToRasterCoordX** - 更改：2.1.0 在之前的版本中，称 ST_World2RasterCoordX 返回格中点几何 (pt) 的列或以格世界空参考系表示的 X 和 Y 世界坐 (xw, yw)。
- **ST_WorldToRasterCoordY** - 更改：2.1.0 在之前的版本中，称 ST_World2RasterCoordY 返回点几何形 (pt) 的格中的行或以格的世界空参考系表示的 X 和 Y 世界坐 (xw, yw)。

13.12.14 PostGIS 新增功能或增功能（2.0）

下面出的功能是添加或增的 PostGIS 功能。

PostGIS 中的新功能 2.0

- **&&** - 可用性：2.0.0 如果 A 的界框与 B 的界框相交，返回 TRUE。
- **&&&** - 可用性：2.0.0 如果 A 的 n 界框与 B 的 n 界框相交，返回 TRUE。
- **<#>** - 可用性：2.0.0。返回 A 和 B 界框之的 2D 距离。
- **<->** - 可用性：2.0.0 -- 弱 KNN 基于几何象心距离来找最近，而非真距离。点来如果是准确的，其他型是近似的。返回 A 和 B 之的 2D 距离。
- **@** - 可用性：2.0.0 raster@raster、raster@geometry 引入如果 A 的界框包含在 B 的界框中，返回 TRUE。使用双精度界框。
- **@** - 可用性：2.0.5 geometry @ raster 引入如果 A 的界框包含在 B 的界框中，返回 TRUE。使用双精度界框。
- **AddEdge** - 可用性：2.0.0 使用指定的串几何将串添加到表，并将关的起点和点添加到指定拓扑方案的点表，并返回新（或有）的 edgeid。
- **AddFace** - 可用性：2.0.0 将面基元注册到拓扑并取其符。
- **AddNode** - 可用性：2.0.0 将点添加到指定拓扑模式的点表中，并返回新点的 nodeid。如果点已作点存在，返回有的 nodeid。
- **AddOverviewConstraints** - 可用性：2.0.0 将格列一格的概述。
- **AddRasterConstraints** - 可用性：2.0.0 将格束添加到已加的格表中，用于特定列，列束了空参考、比例、大小、、波段、波段型以及一个志，用于表示格列是否被地分。表必加数据才能推断出束。如果束置成功，返回 true，否会出通知。
- **AsGML** - 可用性：2.0.0 返回拓扑几何的 GML 表示形式。
- **CopyTopology** - 可用性：2.0.0 将拓扑（点、线、面、和拓扑几何）的副本复制到新模式中
- **DropOverviewConstraints** - 可用性：2.0.0 取消格列作一格的概述的。

- **DropRasterConstraints** - 可用性: 2.0.0 删除引用栅格表列的 PostGIS 栅格约束。如果您需要重新添加数据或更新栅格列数据, 非常有用。
 - **GetEdgeByPoint** - 可用性: 2.0.0 找与点相交的边的 ID。
 - **GetFaceByPoint** - 可用性: 2.0.0 找与点相交的面。
 - **GetNodeByPoint** - 可用性: 2.0.0 找某个点位置点的点 ID。
 - **GetNodeEdges** - 可用性: 2.0 返回与点相关的一组有序边。
 - **GetRingEdges** - 可用性: 2.0.0 返回按序排列的边符号集合, 这些符号是通沿边的一组行走遇到的。
 - **GetTopoGeomElements** - 可用性: 2.0.0 返回一组 topelement 对象, 其中包含指定 TopoGeometry (原始元素) 的拓扑 element_id、element_type。
 - **GetTopologySRID** - 可用性: 2.0.0 返回拓扑表中指定拓扑名称的拓扑的 SRID。
 - **Polygonize** - 可用性: 2.0.0 找并注册由拓扑指定的所有面。
 - **ST_3DClosestPoint** - 可用性: 2.0.0 返回 g1 上最接近 g2 的 3D 点。是 3D 最短边的第一个点。
 - **ST_3DDFullyWithin** - 可用性: 2.0.0 两个 3D 几何体是否完全在指定的 3D 距离内
 - **ST_3DDWithin** - 可用性: 2.0.0 两个 3D 几何体是否在指定的 3D 距离内
 - **ST_3DDistance** - 可用性: 2.0.0 返回两个几何体之间的 3D 笛卡儿最小距离 (基于空参考) (以投影位表示)。
 - **ST_3DIntersects** - 可用性: 2.0.0 两个几何体在 3D 空间中是否相交 - 适用于点、线串、多边形、多面体曲面 (区域)
 - **ST_3DLongestLine** - 可用性: 2.0.0 返回一个几何体之间的 3D 最长边
 - **ST_3DMaxDistance** - 可用性: 2.0.0 返回两个几何体之间的 3D 笛卡儿最大距离 (基于空参考) (以投影位表示)。
 - **ST_3DShortestLine** - 可用性: 2.0.0 返回两个几何体之间的 3D 最短边
 - **ST_AddEdgeModFace** - 可用性: 2.0 添加新边, 如果边做会分割面, 修改原始面并添加新面。
 - **ST_AddEdgeNewFaces** - 可用性: 2.0 添加新边, 如果边做会分割一个面, 删除原始面并用一个新面替它。
 - **ST_AsGDALRaster** - 可用性: 2.0.0 - 需要 GDAL >= 1.6.0。以指定的 GDAL 栅格格式返回栅格。栅格格式是支持的格式之一。使用 ST_GDALDrivers() 取您的支持的格式列表。
 - **ST_AsJPEG** - 可用性: 2.0.0 - 需要 GDAL >= 1.6.0。将栅格指定的波段作为一个合影像输出 (JPEG) 图像 (字节数) 返回。如果未指定波段且有 1 个或 3 个以上波段, 使用第一个波段。如果只有 3 个波段, 使用所有 3 个波段并将其映射到 RGB。
 - **ST_AsLatLonText** - 可用性: 2.0 返回点的度、分、秒表示形式。
 - **ST_AsPNG** - 可用性: 2.0.0 - 需要 GDAL >= 1.6.0。将栅格指定的波段作为一个便携式网络图形 (PNG) 图像 (字节数) 返回。如果栅格中有 1、3 或 4 个波段且未指定波段, 使用所有波段。如果有 2 个以上或多于 4 个波段且未指定波段, 使用波段 1。波段映射到 RGB 或 RGBA 空。
 - **ST_AsRaster** - 可用性: 2.0.0 - 需要 GDAL >= 1.6.0。将 PostGIS 几何体栅格 PostGIS 栅格。
 - **ST_AsTIFF** - 可用性: 2.0.0 - 需要 GDAL >= 1.6.0。将栅格指定的波段作为一个 TIFF 图像 (字节数) 返回。如果未指定波段或栅格中不存在任何指定波段, 将使用所有波段。
 - **ST_AsX3D** - 可用性: 2.0.0: ISO-IEC-19776-1.2-X3DEncodings-XML 返回 X3D xml 点元素格式的几何体: ISO-IEC-19776-1.2-X3DEncodings-XML
-

- **ST_Aspect** - 可用性: 2.0.0 返回高程栅格波段的坡向（默认为度方位）。对于分析地形很有用。
- **ST_Band** - 可用性: 2.0.0 返回栅格的一个或多个波段作新栅格。对于从栅格建新栅格非常有用。
- **ST_BandIsNoData** - 可用性: 2.0.0 如果波段填充无数据，返回 true。
- **ST_Clip** - 可用性: 2.0.0 返回由输入几何形裁剪的栅格。如果未指定波段号，处理所有波段。如果未指定 crop 或其值 TRUE，输出栅格将被裁剪。如果 touched 置为 TRUE，包括接触到的像素；否则，当像素的中心位于几何形内，才会包括像素。
- **ST_CollectionHomogenize** - 可用性: 2.0.0 返回几何集合的最简表示。
- **ST_ConcaveHull** - 可用性: 2.0.0 计算包含所有输入几何点的可能凹几何
- **ST_Count** - 可用性: 2.0.0 返回栅格或栅格覆盖范围的指定波段中的像素数。如果未指定 band，默认为 band 1。如果 except_nodata_value 置为 true，则不算不等于 nodata 的像素。
- **ST_CreateTopoGeo** - 可用性: 2.0 将几何形集合添加到指定的空拓扑并返回指明成功的消息。
- **ST_Distinct4ma** - 可用性: 2.0.0 栅格处理函数，用于计算域中唯一像素的数量。
- **ST_FlipCoordinates** - 可用性: 2.0.0 返回 X 和 Y 坐标的几何形版本。
- **ST_GDALDrivers** - 可用性: 2.0.0 - 需要 GDAL >= 1.6.0。通过 GDAL 返回 PostGIS 支持的栅格格式列表。ST_AsGDALRaster 可使用 can_write=True 的格式
- **ST_GeomFromGeoJSON** - 可用性: 2.0.0 需要 JSON-C 0.9 或更高版本将几何形的 geojson 表示形式作输入并输出 PostGIS 几何象
- **ST_GetFaceEdges** - 可用性: 2.0 返回一有序的，这些边界定义了 aface。
- **ST_HasNoBand** - 可用性: 2.0.0 如果不存在具有指定波段号的波段，返回 true。如果未指定波段号，假定波段号 1。
- **ST_HillShade** - 可用性: 2.0.0 使用提供的方位角、高度、亮度和比例输入返回高程栅格的假照明。
- **ST_Histogram** - 可用性: 2.0.0 返回一，统计了栅格或栅格覆盖数据分布，其中包括分离的分箱范围。如果未指定，将自动算分箱数。
- **ST_InterpolatePoint** - 可用性: 2.0.0 返回最接近点的几何形的插值量。
- **ST_IsEmpty** - 可用性: 2.0.0 如果栅格空（度 = 0 且高度 = 0），返回 true。否则，返回 false。
- **ST_IsValidDetail** - 可用性: 2.0.0 返回 valid_detail 行，指明几何形是否有效或无效，指明原因和位置。
- **ST_IsValidReason** - 可用性: 2.0 版本正在接受。返回指明几何形是否有效或无效原因的文本。
- **ST_MakeLine** - 可用性: 2.0.0 - 引入了 LineString 输入元素的支持从 Point, MultiPoint, 或 LineString geometries 建 LineString。
- **ST_MakeValid** - 可用性: 2.0.0 在不丢失点的情况下使无效几何体有效。
- **ST_MapAlgebraExpr** - 可用性: 2.0.0 1 栅格波段版本: 通过输入栅格波段和提供的像素型用有效的 PostgreSQL 代数算来建新的波段栅格。如果未指定波段，假定波段 1。
- **ST_MapAlgebraExpr** - 可用性: 2.0.0 2 栅格波段版本: 通过提供的个输入栅格波段和像素型用有效的 PostgreSQL 代数算来建新的波段栅格。如果未指定波段号，假定每个栅格的波段 1。生成的栅格将在第一个栅格定义的网格上（比例、斜和像素角），并具有由“extenttype”参数定义的范。“extenttype”的可以是: INTERSECTION、UNION、FIRST、SECOND。
- **ST_MapAlgebraFct** - 可用性: 2.0.0 1 波段版本 - 通过在输入栅格波段和提供的像素型上用有效的 PostgreSQL 函数来建新的波段栅格。如果未指定波段，假定波段 1。
- **ST_MapAlgebraFct** - 可用性: 2.0.0 2 波段版本 - 通过在提供的 2 个输入栅格波段和像素型上用有效的 PostgreSQL 函数来建新的波段栅格。如果未指定波段，假定波段 1。如果未指定，范型默认为 INTERSECTION。

- **ST_MapAlgebraFctNgb** - 可用性: 2.0.0 1-波段版本: 使用用指定格的 PostgreSQL 函数映射代数最近格。返回一个格, 其格是涉及入格波段格的区域的 PLPGSQL 用函数的结果。
- **ST_Max4ma** - 可用性: 2.0.0 计算域中最大像素格的格处理函数。
- **ST_Mean4ma** - 可用性: 2.0.0 计算域中平均像素格的格处理函数。
- **ST_Min4ma** - 可用性: 2.0.0 计算域中最小像素格的格处理函数。
- **ST_ModEdgeHeal** - 可用性: 2.0 通格除格接格条格的点、修改第一条格并格除第二条格来修复格条格。返回已格除格的 id。
- **ST_MoveIsoNode** - 可用性: 2.0.0 在拓扑格中将一个孤立格点从一个点移到格一个点。如果新的 apoint 几何格象已存在作格一个格点, 格会抛出格。返回移格的描述。
- **ST_NewEdgeHeal** - 可用性: 2.0 通格除格接格条格的点、格除格条格并用方向与提供的第一条格相同的格替格它格来修复格条格。
- **ST_Node** - 可用性: 2.0.0 格点是格的集合。
- **ST_NumPatches** - 可用性: 2.0.0 Returns the number of faces in a PolyhedralSurface or TIN.
- **ST_OffsetCurve** - 可用性: 2.0 返回距入格指定距离和方向的偏移格。
- **ST_PatchN** - 可用性: 2.0.0 Returns the Nth face of a PolyhedralSurface or TIN.
- **ST_Perimeter** - 可用性 2.0.0: 引入了格地理的支持返回多格形几何或地理的格界格度。
- **ST_PixelAsPolygon** - 可用性: 2.0.0 返回限定特定行和列的像素的多格形几何形状。
- **ST_PixelAsPolygons** - 可用性: 2.0.0 返回包格格的每个像素的多格形几何格形以及每个像素的格、X 和 Y 格格坐格。
- **ST_Project** - 可用性: 2.0.0 返回从起点按距离和方位角 (方位角) 投影的点。
- **ST_Quantile** - 可用性: 2.0.0 计算格本或格体上下文中格格或格格表覆盖范格的分位数。因此, 可以格格某个格是否位于格的 25%、50%、75% 百分位格。
- **ST_Range4ma** - 可用性: 2.0.0 计算域中像素格范格的格格处理函数。
- **ST_Reclass** - 可用性: 2.0.0 格建由从原始数据重新分格的波段格型格成的新格格。nband 是要更改的波段。如果未指定 nband, 格假定格 1。所有其他波段均按原格返回。使用案例: 将 16BUI 波段格格 8BUI 等, 以便更格格地呈格格可格格式。
- **ST_RelateMatch** - 可用性: 2.0.0 格格 DE-9IM 交集矩格是否与交集矩格模式匹配
- **ST_RemEdgeModFace** - 可用性: 2.0 格除一条格, 如果格格将格个面分开, 格格除一个面并修改格一个面以覆盖格个面的空格。
- **ST_RemEdgeNewFace** - 可用性: 2.0 格除一条格, 如果格除的格将格个面分开, 格格除原始面并用新面替格它格。
- **ST_Resample** - 可用性: 2.0.0 需要 GDAL 1.6.1+ 重采格一个格格格像, 可以指定重新采格算法、新的尺寸、任意的格格角点, 以及一格格格地理参考属性, 格些属性可以自己定格, 也可以从格一个格格格像中借用。
- **ST_Rescale** - 可用性: 2.0.0 需要 GDAL 1.6.1+ 通格格整格格的格斜 (或像素大小) 来重新采格格格。新的像素格是使用 NearestNeighbor (英格或美式拼写)、Bilinear、Cubic、CubicSpline、Lanczos、Max 或 Min 重采格算法格算的。默格格 NearestNeighbor。
- **ST_Reskew** - 可用性: 2.0.0 需要 GDAL 1.6.1+ 通格格整格格的格斜 (或旋格参数) 来重新采格格格。新的像素格是使用 NearestNeighbor (英格或美式拼写)、Bilinear、Cubic、CubicSpline 或 Lanczos 重采格算法格算的。默格格 NearestNeighbor。
- **ST_SameAlignment** - 可用性: 2.0.0 如果格格具有相同的格斜、比例、空格参考和偏移 (像素可以放在同一网格上而不切割成像素), 格返回 true; 如果没有注意格格格格, 格返回 false。

- **ST_SetBandIsNoData** - 可用性: 2.0.0 将图区的 isnodata 标志置为 TRUE。
- **ST_SharedPaths** - 可用性: 2.0.0 返回一个集合，其中包含一个输入串/多串共享的路径。
- **ST_Slope** - 可用性: 2.0.0 返回高程栅格的坡度（默认为度/分）。用于分析地形很有用。
- **ST_Snap** - 可用性: 2.0.0 将输入几何体的线段和点捕捉到参考几何体的点。
- **ST_SnapToGrid** - 可用性: 2.0.0 需要 GDAL 1.6.1+ 通过将栅格捕捉到网格来重采样栅格。新的像素是使用 NearestNeighbor（英式或美式拼写）、Bilinear, Cubic, CubicSpline 或 Lanczos 重采样算法计算的。默认为 NearestNeighbor。
- **ST_Split** - 可用性: 2.0.0 需要 GEOS 返回通将一个几何体分割成一个几何体而建的几何体集合。
- **ST_StdDev4ma** - 可用性: 2.0.0 计算域中像素的均偏差的栅格处理函数。
- **ST_Sum4ma** - 可用性: 2.0.0 栅格处理函数，计算域中所有像素的和。
- **ST_SummaryStats** - 可用性: 2.0.0 返回指定栅格或栅格覆盖中某个波段的摘要，包括像元数量（count）、和（sum）、均值（mean）、标准差（stddev）、最小值（min）和最大值（max）。若未指定波段，默认为使用第 1 波段。
- **ST_Transform** - 可用性: 2.0.0 需要 GDAL 1.6.1+ 使用指定的重采样算法将已知空参考系中的栅格重新投影到一个已知空参考系。有 NearestNeighbor、Bilinear、Cubic、CubicSpline、Lanczos（默认为 NearestNeighbor）。
- **ST_UnaryUnion** - 可用性: 2.0.0 计算多个几何体的并集的并集。
- **ST_Union** - 可用性: 2.0.0 将一个栅格切片的并集返回由 1 个或多个波段组成的栅格。
- **ST_ValueCount** - 可用性: 2.0.0 返回一个数组，其中包含像素值以及具有固定值的栅格（或栅格覆盖范围）固定值中的像素数。如果未指定波段，默认为波段 1。默认情况下，不计算点数据像素。输出像素中的所有其他值，并将像素值四舍五入到最接近的整数。
- **TopoElementArray_Agg** - 可用性: 2.0.0 返回一个 element_id、类型数 (topoelements) 的 topoelementarray。
- **TopoGeo_AddLineString** - 可用性: 2.0.0 使用容差将串添加到有拓扑中，并在可能的情况下拆分面或面。
- **TopoGeo_AddPoint** - 可用性: 2.0.0 使用容差并可能分割有向有拓扑添加点。
- **TopoGeo_AddPolygon** - 可用性: 2.0.0 使用公差并可能分割有面/面将多边形添加到有拓扑。返回面标识符。
- **TopologySummary** - 可用性: 2.0.0 提取拓扑名称并提供拓扑中对象类型的计数。
- **toTopoGeom** - 可用性: 2.0 将几何图形转换为拓扑几何图形。
- **~** - 可用性: 2.0.0 如果 A 的边界框包含 B 的边界框，返回 TRUE。使用双精度边界框。
- **~=** - 可用性: 2.0.0 如果 A 的边界框与 B 的边界框相同，返回 TRUE。

PostGIS 的功能增强 2.0

- **&&** - 增强: 2.0.0 引入了多面体曲面的支持。如果 A 的 2D 边界框与 B 的 2D 边界框相交，返回 TRUE。
- **AddGeometryColumn** - 增强: 2.0.0 引入了 use_typmod 参数。默认为建 typmod 几何列而不是基于约束。将 geometry（几何）列添加到有表。
- **Box2D** - 增强功能: 引入了 2.0.0 多面体曲面、三角形和三角网的支持。返回表示几何图形的 2D 范围的 BOX2D。
- **Box3D** - 增强功能: 引入了 2.0.0 多面体曲面、三角形和三角网的支持。返回表示几何体 3D 范围的 BOX3D。
- **CreateTopology** - Enhanced: 2.0 新增 hasZ 的支持。建立一个新的拓扑架并将其注册到 topology.topology 表中。

- **GeometryType** - 增⊗功能：引入了 2.0.0 ⊗多面体曲面、三角形和三角网的支持。以文本形式返回几何的⊗型。
- **Populate_Geometry_Columns** - 增⊗：2.0.0 引入了 `use_typmod` 可⊗参数, 允⊗控制是否使用 `typmodifiers` 或⊗⊗束⊗建列。确保几何列由⊗型修⊗符定⊗或具有适当的空⊗束。
- **ST_3DExtent** - 增⊗功能：引入了 2.0.0 ⊗多面体曲面、三角形和三角网的支持。返回几何⊗形的 3D ⊗界框的聚合函数。
- **ST_Affine** - 增⊗功能：引入了 2.0.0 ⊗多面体曲面、三角形和三角网的支持。⊗几何体⊗用 3D 仿射⊗。
- **ST_Area** - 增⊗：2.0.0 - 引入了⊗ 2D 多面体曲面的支持。返回多⊗形几何体的面⊗。
- **ST_AsBinary** - 增⊗功能：引入了 2.0.0 ⊗多面体曲面、三角形和三角网的支持。返回不⊗ SRID 元数据的几何/地理的 OGC/ISO 已知的二⊗制 (WKB) 表示形式。
- **ST_AsBinary** - 增⊗：2.0.0 支持更高坐⊗度。返回不⊗ SRID 元数据的几何/地理的 OGC/ISO 已知的二⊗制 (WKB) 表示形式。
- **ST_AsBinary** - 增⊗：2.0.0 支持地理中的字⊗序。返回不⊗ SRID 元数据的几何/地理的 OGC/ISO 已知的二⊗制 (WKB) 表示形式。
- **ST_AsEWKB** - 增⊗功能：引入了 2.0.0 ⊗多面体曲面、三角形和三角网的支持。返回⊗有 SRID 元数据的几何⊗形的⊗展已知的二⊗制 (EWKB) 表示形式。
- **ST_AsEWKT** - 增⊗：2.0.0 引入了⊗地理、多面体曲面、三角形和 TIN 的支持。使用 SRID 元数据返回几何⊗形的已知文本 (WKT) 表示形式。
- **ST_AsGML** - 增⊗：2.0.0 引入了前⊗支持。引入了 GML3 的⊗⊗ 4, 以允⊗使用 `LineString` 而不是⊗条的 `Curve` ⊗。引入了⊗多面体曲面和 TINS 的 GML3 支持。引入⊗⊗ 32 来⊗出⊗框。将几何⊗形作⊗ GML 版本 2 或 3 元素返回。
- **ST_AsKML** - 增⊗：2.0.0 - 添加前⊗命名空⊗, 使用默⊗和命名参数将几何⊗形作⊗ KML 元素返回。
- **ST_Azimuth** - 增⊗：2.0.0 引入了⊗地理的支持。返回⊗点之⊗直⊗的基于北方的方位角。
- **ST_Dimension** - 增⊗：2.0.0 引入了多面体曲面支持和 TIN 支持。当⊗定空几何⊗, 它不再引⊗异常。返回几何⊗形的拓扑⊗数。
- **ST_Dump** - 增⊗功能：引入了 2.0.0 ⊗多面体曲面、三角形和三角网的支持。返回几何⊗件的一⊗ `geometry_dump` 行。
- **ST_DumpPoints** - 增⊗功能：引入了 2.0.0 ⊗多面体曲面、三角形和三角网的支持。返回几何⊗形中坐⊗的一⊗ `geometry_dump` 行。
- **ST_Expand** - 增⊗功能：引入了 2.0.0 ⊗多面体曲面、三角形和三角网的支持。返回从⊗一个⊗界框或几何⊗形⊗展的⊗界框。
- **ST_Extent** - 增⊗功能：引入了 2.0.0 ⊗多面体曲面、三角形和三角网的支持。返回几何⊗形⊗界框的聚合函数。
- **ST_Force2D** - 增⊗：2.0.0 引入了⊗多面体曲面的支持。⊗制几何⊗形⊗入“二⊗模式”。
- **ST_Force3D** - 增⊗：2.0.0 引入了⊗多面体曲面的支持。⊗制几何⊗形⊗入 XYZ 模式。⊗是 `ST_Force3DZ` 的⊗名。
- **ST_Force3DZ** - 增⊗：2.0.0 引入了⊗多面体曲面的支持。⊗制几何⊗形⊗入 XYZ 模式。
- **ST_ForceCollection** - 增⊗：2.0.0 引入了⊗多面体曲面的支持。将几何⊗形⊗⊗几何集合 (GEOMETRYCOLLECTION)。
- **ST_ForceRHR** - 增⊗：2.0.0 引入了⊗多面体曲面的支持。⊗制多⊗形⊗点的方向遵循右手定⊗。
- **ST_GMLToSQL** - 增⊗功能：2.0.0 支持多面体曲面和 TIN。Returns a geometry from GML. Alias for `ST_GeomFromGML`.
- **ST_GMLToSQL** - 增⊗：2.0.0 引入了多面体曲面支持和 TIN 支持。Returns a geometry from GML. Alias for `ST_GeomFromGML`.

- **ST_GeomFromEWKB** - 增⊞功能：2.0.0 支持多面体曲面和 TIN。从⊞展已知的二⊞制表示 (EWKB) 返回指定的 ST_Geometry ⊞。
- **ST_GeomFromEWKT** - 增⊞功能：2.0.0 支持多面体曲面和 TIN。从⊞展已知的文本表示 (EWKT) 返回指定的 ST_Geometry ⊞。
- **ST_GeomFromGML** - 增⊞功能：2.0.0 支持多面体曲面和 TIN。将几何⊞形的 GML 表示形式作⊞⊞入并⊞出 PostGIS 几何⊞象
- **ST_GeomFromGML** - 增⊞：2.0.0 引入了多面体曲面支持和 TIN 支持。将几何⊞形的 GML 表示形式作⊞⊞入并⊞出 PostGIS 几何⊞象
- **ST_GeometryN** - 增⊞功能：引入了 2.0.0 ⊞多面体曲面、三角形和三角网的支持。返回几何集合的一个元素。
- **ST_GeometryType** - 增⊞：2.0.0 引入了⊞多面体曲面的支持。以文本形式返回几何⊞形的 SQL-MM ⊞型。
- **ST_IsClosed** - 增⊞：2.0.0 引入了⊞多面体曲面的支持。⊞⊞ LineStrings 的起点和⊞点是否重合。⊞于多面体表面⊞⊞是否⊞合 (⊞心)。
- **ST_MakeEnvelope** - 增⊞：2.0：引入了在不指定 SRID 的情况下指定外包矩形的功能。根据最小和最大坐⊞⊞建矩形多⊞形。
- **ST_MakeValid** - 增⊞：2.0.1 速度提升⊞⊞在不⊞失⊞点的情况下使无效几何体有效。
- **ST_NPoints** - 增⊞：2.0.0 引入了⊞多面体曲面的支持。返回几何⊞形中的点数 (⊞点)。
- **ST_NumGeometries** - 增⊞功能：引入了 2.0.0 ⊞多面体曲面、三角形和三角网的支持。返回几何集合中的元素数量。
- **ST_Relate** - 增⊞：2.0.0 - 添加了⊞指定⊞界⊞点⊞⊞的支持。⊞⊞⊞个几何⊞形是否具有与交集矩⊞模式匹配的拓扑关系，或⊞算它⊞的交集矩⊞
- **ST_Rotate** - 增⊞功能：引入了 2.0.0 ⊞多面体曲面、三角形和三角网的支持。⊞原点旋⊞几何体。
- **ST_Rotate** - 增⊞：2.0.0 添加了用于指定旋⊞原点的附加参数。⊞原点旋⊞几何体。
- **ST_RotateX** - 增⊞功能：引入了 2.0.0 ⊞多面体曲面、三角形和三角网的支持。⊞ X ⊞旋⊞几何体。
- **ST_RotateY** - 增⊞功能：引入了 2.0.0 ⊞多面体曲面、三角形和三角网的支持。⊞ Y ⊞旋⊞几何体。
- **ST_RotateZ** - 增⊞功能：引入了 2.0.0 ⊞多面体曲面、三角形和三角网的支持。⊞ Z ⊞旋⊞几何体。
- **ST_Scale** - 增⊞功能：引入了 2.0.0 ⊞多面体曲面、三角形和三角网的支持。按⊞定因子⊞放几何⊞形。
- **ST_ShiftLongitude** - 增⊞功能：2.0.0 支持多面体曲面和 TIN。在 -180-180 和 0-360 之⊞移⊞几何⊞形的⊞度坐⊞。
- **ST_Summary** - 增⊞：在 2.0.0 中添加了地理支持返回几何内容的文本摘要。
- **ST_Transform** - 增⊞：2.0.0 引入了⊞多面体曲面的支持。返回坐⊞⊞⊞⊞不同空⊞参考系的新几何⊞形。
- **ST_Value** - 增⊞：2.0.0 添加了 `except_nodata_value` 可⊞参数。返回指定列 x、行 y 像素或特定几何点⊞⊞定波段的⊞。波段⊞号从 1 开始，如果未指定，⊞默⊞⊞ 1。如果将参数 `exclude_nodata_value` ⊞置⊞ false，⊞所有像素都被⊞⊞与 `nodata` 像素相交并返回其⊞。如果未⊞⊞参数 `exclude_nodata_value`，⊞从⊞格的元数据中⊞取⊞⊞。
- **ValidateTopology** - 增⊞：2.0.0 更有效的⊞⊞交叉⊞⊞并修复了先前版本中存在的⊞⊞。返回一⊞ `validate_topology_returntype` ⊞象，⊞⊞⊞明拓扑⊞⊞。

PostGIS 中的功能⊞生⊞化 2.0

- **AddGeometryColumn** - 更改：2.0.0 此函数不再更新 `geometry_columns`，因⊞ `geometry_columns` 是从系⊞目⊞⊞取的⊞⊞。默⊞情况下，它也不⊞建⊞束，而是使用 PostgreSQL 内置的⊞型修⊞符行⊞。因此，例如，使用此函数⊞建 `wgs84 POINT` 列⊞在相当于：`ALTER TABLE some_table ADD COLUMN geom geometry(Point,4326);` 将 `geometry` (几何) 列添加到⊞有表。

- **AddGeometryColumn** - 更改：2.0.0 如果您需要约束的旧行，使用默认的 `use_typmod`，但将其置 `false`。将 `geometry`（几何）列添加到表。
- **AddGeometryColumn** - 更改：2.0.0 不能再在 `Geometry_columns` 中手动注册，但是根据几何 Typmod 表几何形并在没有包装器函数的情况下使用的将正确注册自身，因为它继承了父表的 Typmod 行。使用出其他几何形的几何函数的需要 typmod 几何形，以便这些几何列能在 `geometry_columns` 中正确注册。参见。将 `geometry`（几何）列添加到表。
- **Box3D** - 更改：2.0.0 在 2.0 之前的版本中，曾有 `box2d` 而不是 `box3d`。由于 `box2d` 是已弃用的型，因此已更改为 `box3d`。返回格封框的 `box 3d` 表示形式。
- **DropGeometryColumn** - Changed: 2.0.0. Retained for backward compatibility after `geometry_columns` became a view of the system catalogs. 从空表中移除 `geometry`（几何）列。
- **DropGeometryTable** - Changed: 2.0.0. Retained for backward compatibility after `geometry_columns` became a view of the system catalogs. 除表及其在 `geometry_columns` 中的所有引用。
- **Populate_Geometry_Columns** - 更改：2.0.0 默认情况下，在使用型修符而不是约束几何型。您仍然可以通过使用新的 `use_typmod` 并将其置 `false` 来使用约束行。确保几何列由型修符定或具有适当的空约束。
- **ST_3DExtent** - 更改：2.0.0 在之前的版本中，曾被称 `ST_Extent3D` 返回几何形的 3D 界框的聚合函数。
- **ST_3DLength** - 更改：2.0.0 在之前的版本中，曾被称 `ST_Length3D` 返回性几何体的 3D 度。
- **ST_3DMakeBox** - 更改：2.0.0 在之前的版本中，曾被称 `ST_MakeBox3D` 建由个 3D 点几何形定 BOX3D。
- **ST_3DPerimeter** - 更改：2.0.0 在之前的版本中，曾被称 `ST_Perimeter3D` 返回多形几何体的 3D 周。
- **ST_AsBinary** - Changed: 2.0.0 Inputs to this function can not be unknown -- must be geometry. Constructs such as `ST_AsBinary('POINT(1 2)')` are no longer valid and you will get an `n ST_AsBinary(unknown)` is not unique error. Code like that needs to be changed to `ST_AsBinary('POINT(1 2)::geometry')`. If that is not possible, then install `legacy.sql`. 返回不 SRID 元数据的几何/地理的 OGC/ISO 已知的二进制 (WKB) 表示形式。
- **ST_AsGML** - 更改：2.0.0 使用默认命名参数将几何形作 GML 版本 2 或 3 元素返回。
- **ST_AsGeoJSON** - 更改：2.0.0 支持默认参数和命名参数。以 GeoJSON 格式返回一个几何体或要素。
- **ST_AsSVG** - 更改：2.0.0 - 添加了默认参数和命名参数的支持返回几何体的 SVG 路径数据。
- **ST_EndPoint** - 更改：2.0.0 不再适用于个几何体 `MultiLineStrings`。在旧版本的 PostGIS 中，行 `MultiLineString` 可以使用此函数并返回点。在 2.0.0 中，它像任何其他 `MultiLineString` 一样返回 NULL。旧的行是一个未的功能，但是那些假将数据存 LINESTRING 的人可能会在 2.0.0 中遇到些返回 NULL 的情况。Returns the last point of a `LineString`, `CircularLineString`, or `NURBSCurve`.
- **ST_GDALDrivers** - 更改：2.0.6、2.1.3 - 默认情况下不用任何程序，除非置了 `GUC` 或境量 `gdal_enabled_drivers`。通过 GDAL 返回 PostGIS 支持的格式列表。`ST_AsGDALRaster` 可使用 `can_write=True` 的格式
- **ST_GeomFromText** - 更改：2.0.0 在 PostGIS 的早期版本中，允 `ST_GeomFromText('GEOMETRYCOLLECTION EMPTY')` 为了更好地符合 SQL/MM 准，在 PostGIS 2.0.0 中是在非法的。在写成 `ST_GeomFromText('GEOMETRYCOLLECTION EMPTY')` 从已知的文本表示 (WKT) 返回指定的 `ST_Geometry`。
- **ST_GeometryN** - 更改：2.0.0 版本。之前的版本于奇异几何形会返回 NULL。在已更改在 `ST_GeometryN(...)` 情况下返回几何形。返回几何集合的一个元素。
- **ST_IsEmpty** - 已更改：2.0.0 之前的 PostGIS 版本允 `ST_GeomFromText('GEOMETRYCOLLECTION EMPTY')`。在 PostGIS 2.0.0 中，是不正确的，因为它更符合 SQL/MM 准几何形是否空。

- **ST_Length** - 更改：2.0.0 重大更改 - 在之前的版本中，将此函数用于地理类型的多/多边形将为您提供多边形/多多边形的周长。在 2.0.0 中，它已更改为返回 0 以符合几何函数。如果您想要多边形的周长，请使用 ST_Perimeter 返回线性几何体的二面度。
- **ST_LocateAlong** - 更改：2.0.0 在之前的版本中，它曾被称作 ST_Locate_Along_Measure。返回几何上与测量匹配的点的。
- **ST_LocateBetween** - 更改：2.0.0 - 在之前的版本中，它曾被称作 ST_Locate_Between_Measures。返回与测量范围匹配的几何形状部分。
- **ST_ModEdgeSplit** - 更改：2.0 - 在之前的版本中，它被命名为 ST_ModEdgesSplit 通过沿边有新建点、修改原始边并添加新边来分割边。
- **ST_NumGeometries** - 更改：2.0.0 在之前的版本中，如果几何形状不是 collection/MULTI 类型，它会返回 NULL。2.0.0 它在每个几何形状返回 1，例如 POLYGON、LINESTRING、POINT。返回几何集合中的元素数量。
- **ST_NumInteriorRings** - 更改：2.0.0 - 在之前的版本中，它允许 MULTIPOLYGON，返回第一个 POLYGON 的内环数量。返回多边形的内环（孔）数。
- **ST_PointN** - 更改：2.0.0 不再适用于几何 multilinestrings。在旧版本的 PostGIS 中——行 multilinestring 可以与此函数很好地配合并返回起点。在 2.0.0 中，它像任何其他 multilinestring 一样只返回 NULL。返回几何形状中第一个字符串或形状字符串中的第 N 个点。
- **ST_ScaleX** - 更改：2.0.0。在 WKTRaster 版本中，它称作 ST_PixelSizeX。返回像素宽度的 X 分量（以坐参考系为单位）。
- **ST_ScaleY** - 更改：2.0.0。在 WKTRaster 版本中，它称作 ST_PixelSizeY。返回像素高度的 Y 分量（以坐参考系为单位）。
- **ST_SetScale** - 更改：2.0.0 在 WKTRaster 版本中，它称作 ST_SetPixelSize。它在 2.0.0 中进行了变化。以坐参考系为单位设置像素的 X 和 Y 大小。数字单位/像素度/高度。
- **ST_StartPoint** - 更改：2.0.0 不再适用于几何体 MultiLineStrings。在旧版本的 PostGIS 中，行 MultiLineString 可以与此函数很好地配合并返回起点。在 2.0.0 中，它像任何其他 MultiLineString 一样只返回 NULL。旧的行是一个未实现的功能，但是那些假设将数据存储在 LINESTRING 的人可能会在 2.0.0 中遇到一些返回 NULL 的情况。Returns the first point of a LineString, CircularLineString, or NURBSCurve.

13.12.15 PostGIS 新增功能或增强功能（1.5）

下面列出的功能是添加或增强的 PostGIS 功能。

PostGIS 中的新功能 1.5

- **&&** - 可用性：1.5.0 引入了地理的支持。如果 A 的 2D 边界框与 B 的 2D 边界框相交，它返回 TRUE。
- **PostGIS_LibXML_Version** - 可用性：1.5 返回 libxml2 的版本号。
- **ST_AddMeasure** - 可用性：1.1.0 沿线性几何形状插入测量。
- **ST_AsBinary** - 可用性：1.5.0 支持地理位置。返回不带 SRID 元数据的几何/地理的 OGC/ISO 已知的二进制制 (WKB) 表示形式。
- **ST_AsGML** - 可用性：1.5.0 支持地理位置。将几何形状作为 GML 版本 2 或 3 元素返回。
- **ST_AsGeoJSON** - 可用性：1.5.0 支持地理位置。以 GeoJSON 格式返回一个几何体或要素。
- **ST_AsText** - 可用性：1.5 - 引入了地理支持。返回不带 SRID 元数据的几何/地理的已知文本 (WKT) 表示形式。
- **ST_Buffer** - 可用性：1.5 - ST_Buffer 已得到增强，以适应各种端接和接口。例如，您可能希望将道路字符串为街道面，并希望将端点为平面或正方形而不是点。添加了地理的薄包装器。它计算覆盖距几何体指定距离内所有点的几何体。

- **ST_ClosestPoint** - 可用性：1.1.0 返回 g1 上最接近 g2 的 2D 点。它是从一个几何体到另一个几何体的最短直线的第一个点。
- **ST_CollectionExtract** - 可用性：1.1.0 给定一个几何集合，返回包含指定类型元素的多几何图形。
- **ST_Covers** - 可用性：1.5 - 引入了地理支持。检查 B 的每个点是否都位于 A 中
- **ST_DFullyWithin** - 可用性：1.1.0 检查一个几何图形是否完全位于另一个几何图形的指定距离内
- **ST_DWithin** - 可用性：1.5.0 引入了地理的支持检查几何图形是否在给定距离内
- **ST_Distance** - 可用性：1.5.0 地理支持在 1.5 中引入。提高了平面的速度，以更好地处理大型或多个点几何图形返回几何或地理之间的距离。
- **ST_DistanceSphere** - 可用性：1.5 - 引入了除点之外的其他几何类型的支持。之前的版本适用于点。使用球形地球模型返回角度/度几何形状之间的最小距离（以米为单位）。
- **ST_DistanceSpheroid** - 可用性：1.5 - 引入了除点之外的其他几何类型的支持。之前的版本适用于点。使用球体模型返回角度/度几何形状之间的最小距离。
- **ST_DumpPoints** - 可用性：1.1.0 返回几何图形中坐点的一行 geometry_dump 行。
- **ST_Envelope** - 可用性：1.5.0 行更改输出双精度而不是 float4 返回表示几何图形边界框的几何图形。
- **ST_Expand** - 可用性：1.5.0 行更改输出双精度而不是 float4 坐点。返回从一个边界框或几何图形扩展的边界框。
- **ST_GMLToSQL** - 可用性：需要 1.5 libxml2 1.6+ Returns a geometry from GML. Alias for ST_GeomFromGML
- **ST_GeomFromGML** - 可用性：需要 1.5 libxml2 1.6+ 将几何图形的 GML 表示形式作输入并输出 PostGIS 几何对象
- **ST_GeomFromKML** - 可用性：1.5, 需要 libxml2 2.6+ 将几何图形的 KML 表示形式作输入并输出 PostGIS 几何对象
- **ST_HausdorffDistance** - 可用性：1.1.0 返回几何图形之间的 Hausdorff 距离。
- **ST_Intersection** - 可用性：1.5 引入了地理数据类型的支持。计算表示几何 A 和 B 的共享部分的几何。
- **ST_Intersects** - 可用性：1.5 引入了地理的支持。检查几何图形是否相交（它至少有一个共同点）
- **ST_Length** - 可用性：1.5.0 地理支持在 1.5 中引入。返回线性几何体的二度长度。
- **ST_LongestLine** - 可用性：1.1.0 返回几何图形之间的二度最短线。
- **ST_MakeEnvelope** - 可用性：1.5 根据最小和最大坐点构建矩形多边形。
- **ST_MaxDistance** - 可用性：1.1.0 返回几何图形之间的二度最大距离（以投影单位表示）。
- **ST_ShortestLine** - 可用性：1.1.0 返回几何图形之间的 2D 最短线
- **~=** - 可用性：1.5.0 改写了行如果 A 的边界框与 B 的边界框相同，返回 TRUE。

13.12.16 PostGIS 新增功能或增强功能（1.4）

下面列出的功能是添加或增强的 PostGIS 功能。

PostGIS 中的新功能 1.4

- **Populate_Geometry_Columns** - 可用性：1.4.0 确保几何列由正确类型符合定义或具有适当的空约束。
- **ST_Collect** - 可用性：引入了 1.4.0 - ST_Collect (几何)。ST_Collect 已得到增强，可以更快地处理更多几何图形。从一个几何图形构建 GeometryCollection 或 Multi* 几何图形。

- **ST_ContainsProperly** - 可用性：1.4.0 返回 B 的每个点是否都位于 A 的内部
- **ST_GeoHash** - 可用性：1.4.0 返回几何图形的 GeoHash 表示形式。
- **ST_IsValidReason** - 可用性：1.4 返回说明几何图形是否有效或无效原因的文本。
- **ST_LineCrossingDirection** - 可用性：1.4 返回一个数字，指示两个 LineString 的交叉行
- **ST_LocateBetweenElevations** - 可用性：1.4.0 返回位于高程 (Z) 范围内的几何图形部分。
- **ST_MakeLine** - 可用性：1.4.0 - 引入了 ST_MakeLine(geomarray)。ST_MakeLine 聚合函数得到增量，可以更快地处理更多点。从 Point, MultiPoint, 或 LineString geometries 构建 LineString。
- **ST_MinimumBoundingCircle** - 可用性：1.4.0 返回包含几何图形的最小圆形多边形。
- **ST_Union** - 可用性：1.4.0 - ST_Union 得到增量。PostgreSQL 中引入了 ST_Union(geomarray) 以及更快的集合聚合。运算符表示输入几何图形的点集并集的几何图形。

13.12.17 PostGIS 新增功能或增量功能（1.3）

下面列出的功能是添加或增量的 PostGIS 功能。

PostGIS 中的新功能 1.3

- **ST_AsGML** - 可用性：1.3.2 将几何图形作为 GML 版本 2 或 3 元素返回。
 - **ST_AsGeoJSON** - 可用性：1.3.4 以 GeoJSON 格式返回一个几何体或要素。
 - **ST_CurveToLine** - 可用性：1.3.0 将包含曲线的几何图形转换为线性几何图形。
 - **ST_LineToCurve** - 可用性：1.3.0 将线性几何图形转换为曲线几何图形。
 - **ST_SimplifyPreserveTopology** - 可用性：1.3.3 使用 Douglas-Peucker 算法返回几何图形的简化且有效表示。
-

Chapter 14

告 告

14.1 告 件

有效告是帮助 PostGIS 开的基本方法。最有效的告是使 PostGIS 开人能重它，因此它最好包含触它的脚本以及有关到它的境的所有信息。行 `SELECT postgis_full_version()` [于 PostGIS] 和 `SELECT version()` [于 postgresql] 可以提取足好的信息。

如果您没有使用最新版本，得先看其[版本更日志](#)，以了解您的是否已得到修复。

使用 [PostGIS 跟踪器](#)将确保您的告不会被弃，并您随了解其理程。在告新之前，数据以看它是否是已知，如果是，添加您所掌握的有关它的任何新信息。

在提交新告之前，您可能需要 Simon Tatham 关于[如何有效告](#)。

14.2 告文档

文档准确反映件的功能和行。如果没有，可能是因件或文档或缺陷。

文档也可以告 [PostGIS bug 跟踪器](#)。

如果您的修很，只需在新的跟踪器中其行描述，并具体明其在文档中的位置。

如果您的更改更广泛，那么丁是首。是 Unix 上的四步程（假设您已安装了 [git](#)）：

Before editing or building the manual, read the repository's [documentation development guide](#). It covers the executable example, generated figure, validation, HTML, and PDF workflows used by the manual.

1. 克隆 PostGIS 的 git 存。在 Unix 上，入：

```
git clone https://git.osgeo.org/gitea/postgis/postgis.git
```

将存 在 postgis 目中

2. 使用您最喜的文本器文档行更改。在 Unix 上，入（例如）：

```
vim doc/postgis.xml
```

注意，文档是用 DocBook XML 而不是 HTML 写的，因此如果您不熟悉它，按照文档其余部分的示例行操作。

3. 制作一个包含与文档主副本的差的丁文件。在 Unix 上，入：

```
git diff doc/postgis.xml > doc.patch
```

4. 将丁附加到跟踪器中的新。

Chapter 15

Credits and Acknowledgments

PostGIS is built and sustained by a worldwide community. This chapter records the people, organizations, funders, and upstream projects that have contributed code, documentation, infrastructure, testing, support, and funding.

15.1 Project Steering Committee

The PostGIS Project Steering Committee (PSC) coordinates the general direction, release cycles, documentation, and outreach efforts for the PostGIS project. In addition the PSC provides general user support, accepts and approves patches from the general PostGIS community and votes on miscellaneous issues involving PostGIS such as developer commit access, new PSC members or significant API changes.

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

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

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

Sandro Santilli Bug fixes and maintenance, 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.

15.2 Core Contributors Present

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

Loïc Bartoletti SFCGAL enhancements and maintenance and ci support

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

Martin Davis GEOS enhancements and documentation

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

Aliaksandr Kalenik Geometry Processing, PostgreSQL gist, general bug fixing

Laurențiu Nicola Bug fixes, maintenance, stability, and performance

15.3 Core Contributors Past

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

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 Raster development, GDAL driver support, loader

Olivier Courtin (Emeritus) Input/output XML (KML,GML)/GeoJSON functions, 3D support and bug fixes.

Chris Hodgson Prior PSC Member. General development, site and buildbot maintenance, OSGeo incubation management

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

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

Dave Blasby The original developer/Co-founder of PostGIS. Dave wrote the server side objects, index bindings, and many of the server side analytical functions.

Jeff Lounsbury Original development of the Shapefile loader/dumper.

Mark Leslie Ongoing maintenance and development of core functions. Enhanced curve support. Shapefile GUI loader.

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

David Zwarg Raster development (mostly map algebra analytic functions)

15.4 PostGIS Raster Origins

PostGIS raster discussions date to 2002 and included early PGCHIP experiments. Xing Lin developed the **geotiff2pgraster** PGRaster prototype during Google Summer of Code 2007. The WKT Raster project began in 2008, Jorge Arévalo developed the first GDAL driver prototype during Google Summer of Code 2009, and the 2010 FOSS4G code sprint agreed to make WKT Raster the PostGIS raster type. Raster support shipped as part of PostGIS 2.0 in 2012.

The early implementation team included Pierre Racine, Mateusz Loskot, Sandro Santilli, Jorge Arévalo, Bborie Park, David Zwarg, and Regina Obe. Their work established the raster type, loader, GDAL integration, raster/vector operations, map algebra, documentation, and testing that became the current raster subsystem.

Other recorded contributions include Jeff Adams's early sprint work, Qing Liu's raster distance analysis, Maurício Carvalho Mathias de Paulo's QGIS raster plugin and related paper with Lúbia Vinhas, and Ignacio Brodin's gvSIG raster plugin and presentation with Jorge Sanz. Bryce L. Nordgren created early raster coverage and seamless-architecture diagrams. Pierre Racine wrote the original raster/vector tutorial and created its walkthrough and early API diagrams.

The historical project planning record credits Steve Cumming and Université Laval with funding Pierre Racine's work from June 2008 through July 2012; Martin Daly and Cadcorp with funding about six months of Mateusz Loskot's work through June 2009; Tyler Erickson and Michigan Tech Research Institute with a \$1,200 USD contribution; Miguel Lizondo and Deimos Space with support for about two years of Jorge Arévalo's work through June 2011; and the Center for Vectorborne Diseases at UC Davis with funding Bborie Park's work from March 2010 through July 2012.

15.5 Other Contributors

Individual Contributors

@amwps290 (GitHub user)
 @kannes (GitHub user: Hannes)
 @sunhm89 (GitHub user)
 @wewindy (GitHub user: Chen)
 Adrien Berchet
 Alex Bodnaru
 Alex Mayrhofer
 Andrea Peri
 Andreas Forø Tollefsen
 Andreas Neumann
 Andreas Schild
 Andrew Gierth
 Andrew Harvey
 Anne Ghisla
 Antoine Bajolet
 Ariel Mashraki
 Arno Renevier
 Arthur Bazin
 Arthur Lesuisse
 Artur Zakirov
 Ayo Adesugba
 Barbara Phillipot
 Baris Ergun
 Bas Couwenberg
 Ben Jubb
 Benjamin Morel
 Bernd Reiß
 Bernhard Reiter
 Bill Mill
 Björn Esser
 Brian Hamlin
 Bruce Rindahl
 Bruno Wolff III
 Bryce L. Nordgren
 Carl Anderson
 Carlos López Quintanilla
 Charles Karney
 Charlie Savage
 Chris Mayo
 Christian Schroeder
 Christoph Berg
 Christoph Moench-Tegeder
 Dan Faudemer
 Dan Levy
 Dane Springmeyer
 Daniel Nylander
 Dapeng Wang
 Daryl Herzmann
 Dave Fuhry
 David G. Johnston
 David Garnier
 David Skea
 David Techer
 Dennis Tighe
 Denys Kovshun
 Devrim Gündüz
 Dian M Fay
 Dmitry Vasilyev
 Edouard Choinière
 Eduin Carrillo
 Egor Ivkov
 Eric Ridge
 Esteban Zimanyi
 Eugene Antimirov
 Florian Weimer
 François Bonzon
 Frank Warmerdam
 George Silva
 Gerald Fenoy
 Gino Lucrezi
 Giovanni Zezza
 Grant Hutchins
 Greg Troxel
 Guillaume Lelarge
 Giuseppe Broccolo
 Han Wang
 Hannes Janetzek
 Hans Lemuet
 Haribabu Kommi
 Harrissou Sant-anna
 Havard Tveite
 Ignacio Brodin
 IIDA Tetsushi
 Ingvild Nystuen
 Jackie Leng
 James Addison
 James Marca
 Jan Katins
 Jan Tojnar
 Jason Smith
 Jean Felder
 Jeff Adams
 Jelte Fennema
 Jim Jones
 Joe Conway
 Jonne Savolainen
 Jorge Gustavo Rocha
 Jorge Sanz
 Jose Carlos Martinez Llari
 Jostein Leira
 Jörg Habenicht
 Juan Sebastian Jimenez Aldana
 Juergen E. Fischer
 Julien Monticolo
 Julien Rouhaud
 Kashif Rasul
 Klaus Foerster
 Kris Jurka
 Kristian Thy
 Lars Aksel Opsahl
 Laurențiu Nicola
 Laurenz Albe
 Lars Roessiger
 Lauri Kajan
 Leo Hsu
 Loic Dachary
 Louis Descoteaux
 Luca Delucchi
 Luca S. Percich
 Lucas C. Villa Real
 Ludovic Hirlimann
 Lubia Vinhas
 Maksim Korotkov
 Manuel Morillo
 Maria Arias de Reyna
 Marc Ducobu
 Mark Sondheim
 Markus Schaber

Maurício
 Maxim Ko
 Maxime G
 Maxime v
 Maxime S
 Megan Ma
 Mehmet I
 Michael F
 Michal Pá
 Mike Toev
 Mitchell H
 Nathan W
 Nathaniel
 Nicolas R
 Nikita Sh
 Nikolai Be
 Norman B
 Norman V
 Patricia T
 Paul Norm
 Paulo Ces
 Pavel Sele
 Pawel Ost
 Peter Ngu
 Pierre Car
 Qing Liu
 Rafal Mag
 Ralph Ma
 Rémi Cura
 Richard G
 Robert Co
 Rodrigo S
 Roger Cre
 Ron Maye
 Sam Peter
 Sarath Ku
 Sergei Sh
 Sergey Fe
 Shinichi S
 Shoaib Bu
 Silvio Gro
 Simon Gre
 Stefan Co
 Steffen M
 Stepan Ku
 Stephen F
 Stephen V
 Steven Ot
 Svyatoslav
 Talha Rizv
 Temuri D
 Teramoto
 Thomas M
 Thomas P
 Tobias Bu
 Tom Glan
 Tom van T
 Victor Col
 Vincent B
 Vincent M
 Vincent P
 Volf Tomá
 Xing Lin
 Yoichi Ka

Corporate Sponsors These are corporate entities that have contributed developer time, hosting, or direct monetary funding to the PostGIS project. In alphabetical order:

- [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](#)
 - [Google](#)
 - [HighGo](#)
 - [Hunter Systems Group](#)
 - [IITM Pravartak Security Team](#)
 - [INIA-CSIC](#)
 - [ISciences, LLC](#)
 - [Jirotech \(formerly LISAssoft\)](#)
 - [Koordinates](#)
 - [Kontur](#)
 - [Lidwala Consulting Engineers](#)
 - [Logical Tracking & Tracing International AG](#)
 - [Maponics](#)
 - [Maumap](#)
 - [Michigan Tech Research Institute](#)
 - [Natural Resources Canada](#)
 - [Norwegian Forest and Landscape Institute](#)
 - [Norwegian Institute of Bioeconomy Research \(NIBIO\)](#)
 - [OSGeo](#)
 - [Oslandia](#)
 - [Palantir Technologies](#)
 - [Paragon Corporation](#)
 - [PlanetScale](#)
 - [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](#)

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

PostGIS has experimented with community fundraising for features in the past. We used Pledge-Bank and we got two successful campaigns out of it.

postgistopology - 10 plus sponsors each contributed \$250 USD to build toTopoGeometry function and beef up topology support in 2.0.0. It happened.

postgis64windows - 20 someodd sponsors each contributed \$100 USD to pay for the work needed to work out PostGIS 64-bit issues on windows. It happened.

External Libraries and Platforms [PostgreSQL](#), the extensible database system on which PostGIS is built. PostgreSQL provides the SQL engine, transactions, query planner, extension framework, and index access methods used by PostGIS.

The [GEOS](#) geometry operations library.

The [PROJ](#) coordinate transformation and geodesic calculation library.

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

[LibXML2](#) for XML-based spatial formats, and [JSON-C](#) for GeoJSON input.

The [GNU Multiple Precision Arithmetic Library \(GMP\)](#) for precision-sensitive topology operations.

[protobuf-c](#) for Mapbox Vector Tile and Geobuf output.

The [iconv](#) API, supplied by the C library or GNU libiconv, for character encoding conversion in the shapefile loader.

[SFCGAL](#) for advanced 2D and 3D functions, together with the [CGAL](#) algorithms that SFCGAL exposes.

Bundled Libraries and Source Code PostGIS ships selected source from [FlatGeobuf](#) and [FlatBuffers](#) to support FlatGeobuf input and output.

[Wagyu](#) and [Mapbox Geometry](#) provide polygon clipping and validation for Mapbox Vector Tile output.

[Ryu](#) provides round-trip-safe floating-point formatting, and [uthash](#) provides in-process hash tables.

The shapefile loader and dumper include code derived from [ShapeLib](#), originally by Frank Warmerdam and Even Rouault. They also include public-domain getopt code from AT&T.

liblwgeom/lookup3.c contains the public-domain lookup3 hash function by Bob Jenkins.

Appendix A

行 明

A.1 PostGIS 3.7.0

This version requires GEOS 3.10+, PostgreSQL 14-19beta3, Proj 6.1+, and libgmp. To take advantage of all postgis extension features, GEOS 3.15+ is needed. To take advantage of all postgis_sfcgal extension features SFCGAL 2.3+ is needed. Python 3 is a new requirement to build the documentation.

A.1.1 重大 化

#5688, [topology] topology building functions now interpret tolerance 0 (old default) to really mean 0 and want -1 (new default) to mean "use topology precision" (Sandro Santilli)

#6024, Remove support for PostgreSQL 12 and 13 (Regina Obe)

#6040, ST_GeoHash round trip through ST_GeomFromGeoHash (Paul Ramsey)

#6053, [address_standardizer] Extension removed and moved to https://github.com/postgis/address_standardizer (Paul Ramsey)

#6052, [tiger_geocoder] Extension removed and moved to https://git.osgeo.org/postgis/postgis_tiger_geocoder (Regina Obe)

#6079, Remove support for GEOS < 3.10 (Sandro Santilli)

#1793, Remove ST_MemCollect aggregate (Darafei Praliaskouski)

GH-848, CG_AlphaShape now returns a MultiPolygon (Jean Felder)

[docs] Python 3 and an SVG rasterizer (rsvg-convert preferred, GraphicsMagick or ImageMagick supported) now required for doc building (Darafei Praliaskouski)

A.1.2 新特性

#1124, **#2935**, **#3033**, **#4658**, **#4659**, [loader] Rework loader arguments: shp2pgsql can create UNLOGGED tables and choose the feature id column name, shp2pgsql --drop-table can emit DROP TABLE before prepare output, raster2pgsql/shp2pgsql expose loader actions as long options, both loaders accept long aliases for existing options, and

-if-not-exists makes creation actions idempotent (Darafei Praliaskouski)

#4208, Add single-geometry variants of ST_MaxDistance and ST_LongestLine (Darafei Praliaskouski)

[topology] FindVertexSegmentPairsBelowDistance function (Sandro Santilli)

ST_CoverageEdges, returns MultiLinestring of distinct shared edges in polygonal coverage (Paul Ramsey)

ST_MinimumSpanningTree, window function to calculate MST (Paul Ramsey)

#5993, [topology] Add max_edges parameter to TopoGeo_AddLinestring (Sandro Santilli)

#6001, support MultiLineString in ST_MakeLine (Paul Ramsey)

#2858, ST_MMin and ST_MMax (Paul Ramsey)

#5941, [raster] Support GDT_Float16 pixel type (Darafei Praliaskouski)

#5992, Optimize GiST index for repeated edge subdivision in topology (Darafei Praliaskouski)

#5702, Allow the compiler to detect the parallelism -flto=auto (Darafei Praliaskouski)

#4798, ST_AsGeoJSON warns about duplicate property keys (Darafei Praliaskouski)

#4222, [raster] ST_DumpAsPolygons honours PostgreSQL interrupts (Darafei Praliaskouski)

#5889, [topology] Include representative locations in topology build errors (Darafei Praliaskouski)

#2614, Use GEOSPpreparedDistance and caching to accelerate ST_DWithin (Paul Ramsey)

GH-839, ST_Multi support for TIN and surfaces (Loïc Bartoletti)

#4398, Add ST_CatmullSmoothing smoothes but retains original vertices (Paul Ramsey)

#4560, ST_3DInterpolatePoint for M interpolation from XYZ inputs (Paul Ramsey)

#1291, ST_MakePolygon support for curved rings (Darafei Praliaskouski)

GT-270, Add NURBSCurve (Loïc Bartoletti)

#2863, Add ST_XSize, ST_YSize, ST_ZSize, and ST_MSize dimension helpers (Darafei Praliaskouski)

A.1.3 增⬢功能

#2045, postgresql2shp query dumps no longer require temporary table privileges (Darafei Praliaskouski)

#1577, [loader] Report unsupported .zip archive input before trying to open shapefile sidecar files (Darafei Praliaskouski)

#3158, Move geometry and geography typmod bit helpers out of the public liblwgeom header (Darafei Praliaskouski)

#3776, [raster] Keep distinct 32-bit integer pixel values separate in ST_DumpAsPolygons (Darafei Praliaskouski)

#4678, [raster] Move st_sum4ma neighborhood callback to C (Darafei Praliaskouski)

#2116, [raster] Add ST_Value nearest-neighbor boundary options (Darafei Praliaskouski)

#2804, #4315, [raster] Support two-argument ST_MapAlgebra callbacks and pass callback call data as actual arguments (Darafei Praliaskouski)

#6062, [topology] Stop using recursive snapping, for improved robustness (Sandro Santilli)

#6065, Improved winding order computation robustness for rings having collapsed elements (Sandro Santilli)

#2137, [raster] Reuse matching source raster overviews in raster2pgsql -l (Darafei Praliaskouski)

#4749, Use point-in-polygon predicate fast paths for point-only GeometryCollections (Darafei Praliaskouski)

#2807, [raster] Preserve missing NODATA values in ST_MapAlgebra outputs (Darafei Praliaskouski)

#2832, [raster] Avoid padding single-tile raster2pgsql overviews when padding is not requested (Darafei Praliaskouski)

#2623, Generate a pgsqld2shp GID field when exporting geometry-only shapefiles (Darafei Praliaskouski)

#2850, Preserve pgsqld2shp numeric precision and scale in DBF metadata (Darafei Praliaskouski)

Build PostgreSQL extension modules with `-fno-semantic-interposition` when available so LTO can optimize same-DSO calls to exported PostGIS entry points directly instead of routing through the module PLT; in a local PostgreSQL 18 `pgbench` microbenchmark over 100k four-point groups this reduced `ST_Collect` aggregate latency by about 4% (Darafei Praliaskouski)

#5973, Install PostGIS utilities under `pg_config --bindir` by default and document `pg_config`-derived install paths more accurately (Darafei Praliaskouski)

#2808, Clarify `postgis_full_version()` output when installed component scripts are newer than core scripts (Darafei Praliaskouski)

#2838, Emit valid X3D Coordinate nodes for 2D polygon output (Darafei Praliaskouski)

GH-899, [raster] Honor PostgreSQL interrupts in long-running GDAL progress callbacks (Darafei Praliaskouski)

#6073, **#6074**, Stabilize geography LRS endpoint interpolation and pole centroid regressions on 32-bit systems (Darafei Praliaskouski)

#2583, Preserve Z/ZM dimensional qualifiers in `ST_AsEWKT` output for TIN (Darafei Praliaskouski)

#4616, Detect pgsqld2shp query keywords case-insensitively (Darafei Praliaskouski)

#1986, `ST_GeomFromGML` support for GML ArcString and curved polygon rings (Darafei Praliaskouski)

#4749, Use point-in-polygon predicate fast paths for point-only GeometryCollections, including `ST_Disjoint` (Darafei Praliaskouski)

#3676, Mark geometry and geography btree comparison support functions as leakproof (Darafei Praliaskouski)

GH-875, Allow `ST_OrderingEquals` joins to use hash and merge join plans (Darafei Praliaskouski)

#1168, [raster] Support curved geometry inputs in `ST_AsRaster` (Darafei Praliaskouski)

#1416, `ST_AsX3D` outputs NURBSCURVE as X3D NurbsCurve and linearizes other curve geometries (Darafei Praliaskouski)

#3705, `ST_AsX3D` outputs valid coordinate nodes for 2D multi-geometries and GeometryCollections (Darafei Praliaskouski)

GT-484, Use column-level SELECT privileges for metadata views (Arthur Bazin)

#1408, **#2386**, **#2557**, **#2805**, **#2898**, **#3743**, **#4323**, **#4332**, **#4385**, **#5109**, **#5532**, **#5593**, **#5638**, **#5645**, **#5950**, **#6022**, **#6082**, **GH-1161**, **GT-584**, Numerous doc enhancements (Darafei Praliaskouski)

A.1.4 Bug Fixes

GH-1175, Guard missing operands before reading node tags in spatial selectivity estimation (Maksim Korotkov)

#6118, Reject empty geometry type modifiers without an out-of-bounds read (Egor Ivkov)

Fix WKB and TWKB parser resource exhaustion on malformed input (Darafei Praliaskouski)

#6038, Avoid stale relation lookups in `geometry_columns` after topology objects are dropped (Darafei Praliaskouski)

#5487, Improve error detail for repeated upgrades after an incomplete deprecated-function cleanup (Darafei Praliaskouski)

- #5975, Initialize skipped union-find cluster ids deterministically (Darafei Praliaskouski)
 - #5904, [topology] Avoid MinGW format warnings when logging topology ids (Darafei Praliaskouski)
 - GH-888**, [sfcgal] Avoid stale detoasted geometry access in CG_Visibility empty-input handling for SFCGAL < 2.2 (Darafei Praliaskouski)
 - #3179, Mark truncated libpgcommon PostgreSQL messages instead of silently dropping the tail (Darafei Praliaskouski)
 - #6003, Avoid size_t formats in liblwgeom logging/error wrappers for MinGW builds (Darafei Praliaskouski)
 - GH-886**, ST_ClusterRelateWin validates NULL relate matrices before detoasting them (Darafei Praliaskouski)
 - GH-889**, ST_Relate pattern inversion avoids out-of-bounds access on short DE-9IM patterns (Darafei Praliaskouski)
 - GH-890**, Avoid GIDX BRIN summary merges across mismatched dimensions (Darafei Praliaskouski)
 - #5959, Prevent histogram target overflow when analysing massive tables (Darafei Praliaskouski)
 - #6080, Avoid out-of-bounds hex WKB lookup for high-bit input bytes (Darafei Praliaskouski)
 - #4828, geometry_columns handles NOT VALID SRID checks without errors (Darafei Praliaskouski)
 - #6048, [raster] ST_Clip no longer crashes when clipping sparse band selections (Darafei Praliaskouski)
 - GH-894**, [raster] Fix Float16 import, mixed nodata warping, band replacement cleanup, GDAL credential redaction, and Python GDAL compatibility (Darafei Praliaskouski)
-