

# **PostGIS 3.6.0dev 手册**

---

**DEV (Wed 06 Nov 2024 05:30:35 PM UTC  
rev. 6366bd6 )**

# Contents

<b>1</b>	<b>介绍</b>	<b>1</b>
1.1	项目委员会	1
1.2	当前的核心贡献者	1
1.3	过去的核心贡献者	2
1.4	其它贡献者	2
<b>2</b>	<b>PostGIS 安装</b>	<b>6</b>
2.1	简短版本	6
2.2	从源代码编译和安装	6
2.2.1	取源代码	7
2.2.2	安装要求	7
2.2.3	构建配置	8
2.2.4	构建	10
2.2.5	构建和部署 PostGIS 扩展	10
2.2.6	测试	12
2.2.7	安装	15
2.3	安装和使用地址标准化工具	15
2.4	安装、升级 Tiger 地理处理器并加载数据	16
2.4.1	在 PostGIS 数据中使用 Tiger 地理处理器	16
2.4.2	将地址标准化器扩展与 Tiger 地理处理器联合使用	18
2.4.3	Tiger 数据加载所需工具	19
2.4.4	升级您的 Tiger 地理处理器安装和数据	19
2.5	常见问题	20
<b>3</b>	<b>PostGIS 管理</b>	<b>21</b>
3.1	性能测试	21
3.1.1	测试	21
3.1.2	运行	22
3.2	配置格式支持	22
3.3	构建空数据	23

3.3.1	使用扩展 (EXTENSION) 用空数据	23
3.3.2	不通扩展 (EXTENSION) 的方式用空数据 (不建)	23
3.4	升空数据	24
3.4.1	升	24
3.4.1.1	使用 9.1 或更高版本的扩展行升	24
3.4.1.2	9.1 之前版本或者无扩展升	25
3.4.2	硬升	26
<b>4</b>	<b>数据管理</b>	<b>28</b>
4.1	空数据模型	28
4.1.1	OGC 几何形	28
4.1.1.1	点 (Point)	28
4.1.1.2	串 (LineString)	29
4.1.1.3	线性 (LinearRing)	29
4.1.1.4	多边形 (Polygon)	29
4.1.1.5	多点 (MultiPoint)	29
4.1.1.6	多串 (MultiLineString)	29
4.1.1.7	多面 (MultiPolygon)	29
4.1.1.8	几何集合 (GeometryCollection)	29
4.1.1.9	多面体曲面 (PolyhedralSurface)	30
4.1.1.10	三角形 (Triangle)	30
4.1.1.11	TIN	30
4.1.2	SQL/MM 第 3 部分-曲线	30
4.1.2.1	弧串 (CircularString)	30
4.1.2.2	复合曲线 (CompoundCurve)	30
4.1.2.3	曲线多边形 (CurvePolygon)	31
4.1.2.4	多曲线 (MultiCurve)	31
4.1.2.5	多曲面 (MultiSurface)	31
4.1.3	WKT 和 WKB	31
4.2	几何数据类型	32
4.2.1	PostGIS EWKB 和 EWKT	33
4.3	地理数据类型	34
4.3.1	建地理表	34
4.3.2	使用地理表	35
4.3.3	何使用地理数据类型	36
4.3.4	地理高常解	37
4.4	几何有效性	37
4.4.1	几何	37
4.4.2	有效的几何形状	39

4.4.3 有效性管理 . . . . .	41
4.5 空参考系 . . . . .	42
4.5.1 SPATIAL_REF_SYS 表 . . . . .	42
4.5.2 用自定义空参考系 . . . . .	43
4.6 空表 . . . . .	44
4.6.1 建空表 . . . . .	44
4.6.2 GEOMETRY_COLUMNS . . . . .	45
4.6.3 手动注册几何列 . . . . .	45
4.7 加空数据 . . . . .	47
4.7.1 使用 SQL 加数据 . . . . .	47
4.7.2 使用 Shapefile 加器 . . . . .	48
4.8 提取空数据 . . . . .	49
4.8.1 使用 SQL 提取数据 . . . . .	49
4.8.2 使用 Shapefile 文件程序 . . . . .	50
4.9 空索引 . . . . .	51
4.9.1 GiST 索引 . . . . .	51
4.9.2 BRIN 索引 . . . . .	52
4.9.3 SP-GiST 索引 . . . . .	53
4.9.4 索引的使用 . . . . .	54
<b>5 空</b> . . . . .	<b>55</b>
5.1 空关系的确定 . . . . .	55
5.1.1 度展九交模型 (Dimensionally Extended 9-Intersection Model) . . . . .	55
5.1.2 命名空关系 . . . . .	57
5.1.3 一般空关系 . . . . .	57
5.2 使用空索引 . . . . .	59
5.3 空 SQL 示例 . . . . .	59
<b>6 性能化技巧</b> . . . . .	<b>62</b>
6.1 大几何形状的小表 . . . . .	62
6.1.1 描述 . . . . .	62
6.1.2 解决方法 . . . . .	62
6.2 几何索引聚 . . . . .	63
6.3 避免度 . . . . .	63

---

<b>7 PostGIS 参考手册</b>	<b>64</b>
7.1 PostGIS Geometry/Geography/Box 数据类型	64
7.1.1 box2d	64
7.1.2 box3d	65
7.1.3 geometry	65
7.1.4 geometry_dump	66
7.1.5 geography	66
7.2 表管理功能	67
7.2.1 AddGeometryColumn	67
7.2.2 DropGeometryColumn	69
7.2.3 DropGeometryTable	69
7.2.4 Find_SRID	70
7.2.5 Populate_Geometry_Columns	71
7.2.6 UpdateGeometrySRID	72
7.3 几何构造函数	73
7.3.1 ST_Collect	73
7.3.2 ST_LineFromMultiPoint	75
7.3.3 ST_MakeEnvelope	76
7.3.4 ST_MakeLine	76
7.3.5 ST_MakePoint	78
7.3.6 ST_MakePointM	79
7.3.7 ST_MakePolygon	80
7.3.8 ST_Point	82
7.3.9 ST_PointZ	83
7.3.10 ST_PointM	84
7.3.11 ST_PointZM	84
7.3.12 ST_Polygon	85
7.3.13 ST_TileEnvelope	86
7.3.14 ST_HexagonGrid	87
7.3.15 ST_Hexagon	89
7.3.16 ST_SquareGrid	90
7.3.17 ST_Square	91
7.3.18 ST_Letters	92
7.4 几何函数	93
7.4.1 GeometryType	93
7.4.2 ST_Boundary	94
7.4.3 ST_BoundingDiagonal	96
7.4.4 ST_CoordDim	97
7.4.5 ST_Dimension	98

---

7.4.6 ST_Dump	98
7.4.7 ST_DumpPoints	100
7.4.8 ST_DumpSegments	104
7.4.9 ST_DumpRings	106
7.4.10 ST_EndPoint	107
7.4.11 ST_Envelope	108
7.4.12 ST_ExteriorRing	110
7.4.13 ST_GeometryN	111
7.4.14 ST_GeometryType	113
7.4.15 ST_HasArc	114
7.4.16 ST_InteriorRingN	115
7.4.17 ST_NumCurves	116
7.4.18 ST_CurveN	116
7.4.19 ST_IsClosed	117
7.4.20 ST_IsCollection	119
7.4.21 ST_IsEmpty	120
7.4.22 ST_IsPolygonCCW	121
7.4.23 ST_IsPolygonCW	122
7.4.24 ST_IsRing	123
7.4.25 ST_IsSimple	123
7.4.26 ST_M	124
7.4.27 ST_MemSize	125
7.4.28 ST_NDims	126
7.4.29 ST_NPoints	127
7.4.30 ST_NRings	128
7.4.31 ST_NumGeometries	128
7.4.32 ST_NumInteriorRings	129
7.4.33 ST_NumInteriorRing	130
7.4.34 ST_NumPatches	130
7.4.35 ST_NumPoints	131
7.4.36 ST_PatchN	131
7.4.37 ST_PointN	132
7.4.38 ST_Points	134
7.4.39 ST_StartPoint	135
7.4.40 ST_Summary	136
7.4.41 ST_X	137
7.4.42 ST_Y	138
7.4.43 ST_Z	138
7.4.44 ST_Zmflag	139

---

7.4.45	ST_HasZ	140
7.4.46	ST_HasM	141
7.5	几何函数器	141
7.5.1	ST_AddPoint	141
7.5.2	ST_CollectionExtract	142
7.5.3	ST_CollectionHomogenize	144
7.5.4	ST_CurveToLine	145
7.5.5	ST_Scroll	148
7.5.6	ST_FlipCoordinates	148
7.5.7	ST_Force2D	149
7.5.8	ST_Force3D	150
7.5.9	ST_Force3DZ	151
7.5.10	ST_Force3DM	152
7.5.11	ST_Force4D	152
7.5.12	ST_ForceCollection	153
7.5.13	ST_ForceCurve	154
7.5.14	ST_ForcePolygonCCW	155
7.5.15	ST_ForcePolygonCW	156
7.5.16	ST_ForceSFS	156
7.5.17	ST_ForceRHR	156
7.5.18	ST_LineExtend	157
7.5.19	ST_LineToCurve	158
7.5.20	ST_Multi	159
7.5.21	ST_Normalize	160
7.5.22	ST_Project	161
7.5.23	ST_QuantizeCoordinates	161
7.5.24	ST_RemovePoint	163
7.5.25	ST_RemoveRepeatedPoints	164
7.5.26	ST_RemoveIrrelevantPointsForView	165
7.5.27	ST_RemoveSmallParts	167
7.5.28	ST_Reverse	169
7.5.29	ST_Segmentize	170
7.5.30	ST_SetPoint	171
7.5.31	ST_ShiftLongitude	172
7.5.32	ST_WrapX	173
7.5.33	ST_SnapToGrid	174
7.5.34	ST_Snap	176
7.5.35	ST_SwapOrdinates	178
7.6	几何有效性函数	179

7.6.1	ST_IsValid	179
7.6.2	ST_IsValidDetail	180
7.6.3	ST_IsValidReason	182
7.6.4	ST_MakeValid	183
7.7	空参考系功能	188
7.7.1	ST_InverseTransformPipeline	188
7.7.2	ST_SetSRID	189
7.7.3	ST_SRID	190
7.7.4	ST_Transform	191
7.7.5	ST_TransformPipeline	193
7.7.6	postgis_srs_codes	195
7.7.7	postgis_srs	195
7.7.8	postgis_srs_all	196
7.7.9	postgis_srs_search	197
7.8	几何入	198
7.8.1	已知文本 (WKT)	198
7.8.1.1	ST_BdPolyFromText	198
7.8.1.2	ST_BdMPolyFromText	198
7.8.1.3	ST_GeogFromText	199
7.8.1.4	ST_GeographyFromText	199
7.8.1.5	ST_GeomCollFromText	200
7.8.1.6	ST_GeomFromEWKT	200
7.8.1.7	ST_GeomFromMARC21	202
7.8.1.8	ST_GeometryFromText	204
7.8.1.9	ST_GeomFromText	205
7.8.1.10	ST_LineFromText	206
7.8.1.11	ST_MLineFromText	207
7.8.1.12	ST_MPointFromText	208
7.8.1.13	ST_MPolyFromText	209
7.8.1.14	ST_PointFromText	209
7.8.1.15	ST_PolygonFromText	210
7.8.1.16	ST_WKTToSQL	211
7.8.2	已知的二进制文件 (WKB)	212
7.8.2.1	ST_GeogFromWKB	212
7.8.2.2	ST_GeomFromEWKB	212
7.8.2.3	ST_GeomFromWKB	214
7.8.2.4	ST_LineFromWKB	214
7.8.2.5	ST_LinestringFromWKB	215
7.8.2.6	ST_PointFromWKB	216

---



7.8.2.7	ST_WKBToSQL	217
7.8.3	其它格式	218
7.8.3.1	ST_Box2dFromGeoHash	218
7.8.3.2	ST_GeomFromGeoHash	218
7.8.3.3	ST_GeomFromGML	219
7.8.3.4	ST_GeomFromGeoJSON	222
7.8.3.5	ST_GeomFromKML	223
7.8.3.6	ST_GeomFromTWKB	224
7.8.3.7	ST_GMLToSQL	224
7.8.3.8	ST_LineFromEncodedPolyline	225
7.8.3.9	ST_PointFromGeoHash	226
7.8.3.10	ST_FromFlatGeobufToTable	226
7.8.3.11	ST_FromFlatGeobuf	227
7.9	几何输出	227
7.9.1	已知文本 (WKT)	227
7.9.1.1	ST_AsEWKT	227
7.9.1.2	ST_AsText	228
7.9.2	已知的二进制文件 (WKB)	230
7.9.2.1	ST_AsBinary	230
7.9.2.2	ST_AsEWKB	231
7.9.2.3	ST_AsHEXEWKB	232
7.9.3	其它格式	233
7.9.3.1	ST_AsEncodedPolyline	233
7.9.3.2	ST_AsFlatGeobuf	234
7.9.3.3	ST_AsGeobuf	235
7.9.3.4	ST_AsGeoJSON	235
7.9.3.5	ST_AsGML	237
7.9.3.6	ST_AsKML	241
7.9.3.7	ST_AsLatLonText	242
7.9.3.8	ST_AsMARC21	243
7.9.3.9	ST_AsMVTGeom	246
7.9.3.10	ST_AsMVT	247
7.9.3.11	ST_AsSVG	248
7.9.3.12	ST_AsTWKB	249
7.9.3.13	ST_AsX3D	250
7.9.3.14	ST_GeoHash	254
7.10	算符	255
7.10.1	界框算符	255
7.10.1.1	&&	255

---

7.10.1.2	$\&\&(geometry,box2df)$	256
7.10.1.3	$\&\&(box2df,geometry)$	257
7.10.1.4	$\&\&(box2df,box2df)$	257
7.10.1.5	$\&\&$	258
7.10.1.6	$\&\&\&(geometry,gidx)$	259
7.10.1.7	$\&\&\&(gidx,geometry)$	260
7.10.1.8	$\&\&\&(gidx,gidx)$	261
7.10.1.9	$\&<$	262
7.10.1.10	$\&< $	263
7.10.1.11	$\&>$	263
7.10.1.12	$\&<$	264
7.10.1.13	$\&< $	265
7.10.1.14		266
7.10.1.15	$\&>$	267
7.10.1.16	$\&$	268
7.10.1.17	$\&(geometry,box2df)$	269
7.10.1.18	$\&(box2df,geometry)$	269
7.10.1.19	$\&(box2df,box2df)$	270
7.10.1.20	$\&>$	271
7.10.1.21	$\&>$	272
7.10.1.22		272
7.10.1.23	$\&(geometry,box2df)$	273
7.10.1.24	$\&(box2df,geometry)$	274
7.10.1.25	$\&(box2df,box2df)$	275
7.10.1.26	$=$	275
7.10.2	距离算符	276
7.10.2.1	$<->$	276
7.10.2.2	$ = $	278
7.10.2.3	$<\#>$	279
7.10.2.4	$<<->>$	280
7.11	空关系	281
7.11.1	拓扑关系	281
7.11.1.1	ST_3DIntersects	281
7.11.1.2	ST_Contains	282
7.11.1.3	ST_ContainsProperly	286
7.11.1.4	ST_CoveredBy	288
7.11.1.5	ST_Covers	289
7.11.1.6	ST_Crosses	290
7.11.1.7	ST_Disjoint	292

7.11.1.8	ST_Equals	293
7.11.1.9	ST_Intersects	294
7.11.1.10	ST_LineCrossingDirection	296
7.11.1.11	ST_OrderingEquals	299
7.11.1.12	ST_Overlaps	300
7.11.1.13	ST_Relate	303
7.11.1.14	ST_RelateMatch	305
7.11.1.15	ST_Touches	306
7.11.1.16	ST_Within	308
7.11.2	距离关系	309
7.11.2.1	ST_3DDWithin	309
7.11.2.2	ST_3DDFullyWithin	310
7.11.2.3	ST_DFullyWithin	311
7.11.2.4	ST_DWithin	312
7.11.2.5	ST_PointInsideCircle	313
7.12	测量函数	314
7.12.1	ST_Area	314
7.12.2	ST_Azimuth	316
7.12.3	ST_Angle	317
7.12.4	ST_ClosestPoint	318
7.12.5	ST_3DClosestPoint	320
7.12.6	ST_Distance	321
7.12.7	ST_3DDistance	323
7.12.8	ST_DistanceSphere	324
7.12.9	ST_DistanceSpheroid	325
7.12.10	ST_FrechetDistance	326
7.12.11	ST_HausdorffDistance	327
7.12.12	ST_Length	328
7.12.13	ST_Length2D	330
7.12.14	ST_3DLength	330
7.12.15	ST_LengthSpheroid	331
7.12.16	ST_LongestLine	332
7.12.17	ST_3DLongestLine	334
7.12.18	ST_MaxDistance	335
7.12.19	ST_3DMaxDistance	336
7.12.20	ST_MinimumClearance	337
7.12.21	ST_MinimumClearanceLine	338
7.12.22	ST_Perimeter	338
7.12.23	ST_Perimeter2D	340

7.12.2	ST_3DPerimeter	340
7.12.2	ST_ShortestLine	341
7.12.2	ST_3DShortestLine	343
7.13	加函数	344
7.13.1	ST_ClipByBox2D	344
7.13.2	ST_Difference	345
7.13.3	ST_Intersection	346
7.13.4	ST_MemUnion	349
7.13.5	ST_Node	349
7.13.6	ST_Split	350
7.13.7	ST_Subdivide	353
7.13.8	ST_SymDifference	355
7.13.9	ST_UnaryUnion	357
7.13.1	ST_Union	357
7.14	几何处理	360
7.14.1	ST_Buffer	360
7.14.2	ST_BuildArea	364
7.14.3	ST_Centroid	366
7.14.4	ST_ChaikinSmoothing	368
7.14.5	ST_ConcaveHull	370
7.14.6	ST_ConvexHull	373
7.14.7	ST_DelaunayTriangles	374
7.14.8	ST_FilterByM	379
7.14.9	ST_GeneratePoints	380
7.14.1	ST_GeometricMedian	381
7.14.1	ST_LineMerge	382
7.14.1	ST_MaximumInscribedCircle	385
7.14.1	ST_LargestEmptyCircle	387
7.14.1	ST_MinimumBoundingCircle	389
7.14.1	ST_MinimumBoundingRadius	390
7.14.1	ST_OrientedEnvelope	391
7.14.1	ST_OffsetCurve	392
7.14.1	ST_PointOnSurface	396
7.14.1	ST_Polygonize	398
7.14.2	ST_ReducePrecision	400
7.14.2	ST_SharedPaths	402
7.14.2	ST_Simplify	404
7.14.2	ST_SimplifyPreserveTopology	405
7.14.2	ST_SimplifyPolygonHull	407

---

7.14.25	ST_SimplifyVW	410
7.14.26	T_SetEffectiveArea	411
7.14.27	T_TriangulatePolygon	413
7.14.28	T_VoronoiLines	415
7.14.29	T_VoronoiPolygons	416
7.15	覆盖范围	418
7.15.1	ST_CoverageInvalidEdges	418
7.15.2	ST_CoverageSimplify	419
7.15.3	ST_CoverageUnion	421
7.16	仿射	422
7.16.1	ST_Affine	422
7.16.2	ST_Rotate	424
7.16.3	ST_RotateX	425
7.16.4	ST_RotateY	426
7.16.5	ST_RotateZ	427
7.16.6	ST_Scale	428
7.16.7	ST_Translate	429
7.16.8	ST_TransScale	430
7.17	聚函数	431
7.17.1	ST_ClusterDBSCAN	431
7.17.2	ST_ClusterIntersecting	434
7.17.3	ST_ClusterIntersectingWin	434
7.17.4	ST_ClusterKMeans	435
7.17.5	ST_ClusterWithin	437
7.17.6	ST_ClusterWithinWin	438
7.18	边界框函数	439
7.18.1	Box2D	439
7.18.2	Box3D	440
7.18.3	ST_EstimatedExtent	440
7.18.4	ST_Expand	441
7.18.5	ST_Extent	443
7.18.6	ST_3DExtent	444
7.18.7	ST_MakeBox2D	445
7.18.8	ST_3DMakeBox	446
7.18.9	ST_XMax	447
7.18.10	T_XMin	448
7.18.11	T_YMax	449
7.18.12	T_YMin	450
7.18.13	T_ZMax	451

---

7.18.1	ST_ZMin	452
7.19	☒性参考	453
7.19.1	ST_LineInterpolatePoint	453
7.19.2	ST_3DLineInterpolatePoint	454
7.19.3	ST_LineInterpolatePoints	455
7.19.4	ST_LineLocatePoint	456
7.19.5	ST_LineSubstring	457
7.19.6	ST_LocateAlong	459
7.19.7	ST_LocateBetween	460
7.19.8	ST_LocateBetweenElevations	462
7.19.9	ST_InterpolatePoint	462
7.19.10	ST_AddMeasure	463
7.20	☒迹函数	464
7.20.1	ST_IsValidTrajectory	464
7.20.2	ST_ClosestPointOfApproach	465
7.20.3	ST_DistanceCPA	466
7.20.4	ST_CPAWithin	466
7.21	版本函数	467
7.21.1	PostGIS_Extensions_Upgrade	467
7.21.2	PostGIS_Full_Version	468
7.21.3	PostGIS_GEOS_Version	469
7.21.4	PostGIS_GEOS_Compiled_Version	469
7.21.5	PostGIS_Liblwgeom_Version	470
7.21.6	PostGIS_LibXML_Version	470
7.21.7	PostGIS_LibJSON_Version	471
7.21.8	PostGIS_Lib_Build_Date	471
7.21.9	PostGIS_Lib_Version	472
7.21.10	PostGIS_PROJ_Version	472
7.21.11	PostGIS_PROJ_Compiled_Version	473
7.21.12	PostGIS_Wagyu_Version	473
7.21.13	PostGIS_Scripts_Build_Date	474
7.21.14	PostGIS_Scripts_Installed	475
7.21.15	PostGIS_Scripts_Released	475
7.21.16	PostGIS_Version	476
7.22	☒自定☒☒量 (GUCs)	476
7.22.1	postgis.backend	476
7.22.2	postgis.gdal_datapath	477
7.22.3	postgis.gdal_enabled_drivers	478
7.22.4	postgis.enable_outdb_rasters	479

7.22.5	postgis.gdal_vsi_options	480
7.23	故障排除函数	481
7.23.1	PostGIS_AddBBox	481
7.23.2	PostGIS_DropBBox	481
7.23.3	PostGIS_HasBBox	482
<b>8</b>	<b>SFCGAL 函数参考</b>	<b>484</b>
8.1	SFCGAL 管理函数	484
8.1.1	postgis_sfcgal_version	484
8.1.2	postgis_sfcgal_full_version	484
8.2	SFCGAL 几何器和置器	485
8.2.1	CG_ForceLHR	485
8.2.2	CG_IsPlanar	485
8.2.3	CG_IsSolid	486
8.2.4	CG_MakeSolid	486
8.2.5	CG_Orientation	486
8.2.6	CG_Area	487
8.2.7	CG_3DArea	487
8.2.8	CG_Volume	488
8.2.9	ST_ForceLHR	489
8.2.10	ST_IsPlanar	490
8.2.11	ST_IsSolid	490
8.2.12	ST_MakeSolid	491
8.2.13	ST_Orientation	491
8.2.14	ST_3DArea	492
8.2.15	ST_Volume	493
8.3	SFCGAL 几何和关系函数	494
8.3.1	CG_Intersection	494
8.3.2	CG_Intersects	495
8.3.3	CG_3DIntersects	495
8.3.4	CG_Difference	496
8.3.5	ST_3DDifference	497
8.3.6	CG_3DDifference	498
8.3.7	CG_Distance	499
8.3.8	CG_3DDistance	500
8.3.9	ST_3DConvexHull	501
8.3.10	CG_3DConvexHull	501
8.3.11	ST_3DIntersection	502
8.3.12	CG_3DIntersection	503

---

8.3.13	CG_Union	505
8.3.14	ST_3DUnion	506
8.3.15	CG_3DUnion	506
8.3.16	ST_AlphaShape	508
8.3.17	CG_AlphaShape	508
8.3.18	CG_ApproxConvexPartition	511
8.3.19	ST_ApproximateMedialAxis	512
8.3.20	CG_ApproximateMedialAxis	513
8.3.21	ST_ConstrainedDelaunayTriangles	514
8.3.22	CG_ConstrainedDelaunayTriangles	514
8.3.23	ST_Extrude	516
8.3.24	CG_Extrude	516
8.3.25	CG_ExtrudeStraightSkeleton	518
8.3.26	CG_GreeneApproxConvexPartition	519
8.3.27	ST_MinkowskiSum	520
8.3.28	CG_MinkowskiSum	520
8.3.29	ST_OptimalAlphaShape	522
8.3.30	CG_OptimalAlphaShape	523
8.3.31	CG_OptimalConvexPartition	525
8.3.32	CG_StraightSkeleton	526
8.3.33	ST_StraightSkeleton	527
8.3.34	ST_Tesselate	529
8.3.35	CG_Tesselate	529
8.3.36	CG_Triangulate	531
8.3.37	CG_Visibility	532
8.3.38	CG_YMonotonePartition	533
<b>9</b>	<b>拓扑</b>	<b>535</b>
9.1	拓扑型	535
9.1.1	getfaceedges_returntype	535
9.1.2	TopoGeometry	535
9.1.3	validatetopology_returntype	536
9.2	拓扑域	536
9.2.1	TopoElement	536
9.2.2	TopoElementArray	537
9.3	拓扑和拓扑几何管理	538
9.3.1	AddTopoGeometryColumn	538
9.3.2	RenameTopoGeometryColumn	539
9.3.3	DropTopology	539

---



9.3.4	RenameTopology	540
9.3.5	DropTopoGeometryColumn	540
9.3.6	Populate_Topology_Layer	541
9.3.7	TopologySummary	542
9.3.8	ValidateTopology	543
9.3.9	ValidateTopologyRelation	544
9.3.10	FindTopology	545
9.3.11	FindLayer	545
9.4	拓扑管理	546
9.5	拓扑造器	546
9.5.1	CreateTopology	546
9.5.2	CopyTopology	547
9.5.3	ST_InitTopoGeo	548
9.5.4	ST_CreateTopoGeo	548
9.5.5	TopoGeo_AddPoint	549
9.5.6	TopoGeo_AddLineString	549
9.5.7	TopoGeo_AddPolygon	550
9.5.8	TopoGeo_LoadGeometry	551
9.6	拓扑器	551
9.6.1	ST_AddIsoNode	551
9.6.2	ST_AddIsoEdge	552
9.6.3	ST_AddEdgeNewFaces	552
9.6.4	ST_AddEdgeModFace	553
9.6.5	ST_RemEdgeNewFace	554
9.6.6	ST_RemEdgeModFace	554
9.6.7	ST_ChangeEdgeGeom	555
9.6.8	ST_ModEdgeSplit	556
9.6.9	ST_ModEdgeHeal	556
9.6.10	ST_NewEdgeHeal	557
9.6.11	ST_MoveIsoNode	557
9.6.12	ST_NewEdgesSplit	558
9.6.13	ST_RemoveIsoNode	559
9.6.14	ST_RemoveIsoEdge	559
9.7	拓扑器	560
9.7.1	GetEdgeByPoint	560
9.7.2	GetFaceByPoint	561
9.7.3	GetFaceContainingPoint	562
9.7.4	GetNodeByPoint	562
9.7.5	GetTopologyID	563

---

9.7.6	GetTopologySRID	564
9.7.7	GetTopologyName	564
9.7.8	ST_GetFaceEdges	565
9.7.9	ST_GetFaceGeometry	566
9.7.10	GetRingEdges	566
9.7.11	GetNodeEdges	567
9.8	拓扑管理	567
9.8.1	Polygonize	567
9.8.2	AddNode	568
9.8.3	AddEdge	569
9.8.4	AddFace	570
9.8.5	ST_Simplify	572
9.8.6	RemoveUnusedPrimitives	572
9.9	拓扑几何构造函数	573
9.9.1	CreateTopoGeom	573
9.9.2	toTopoGeom	574
9.9.3	TopoElementArray_Agg	576
9.9.4	TopoElement	576
9.10	拓扑几何函数器	577
9.10.1	clearTopoGeom	577
9.10.2	TopoGeom_addElement	577
9.10.3	TopoGeom_remElement	578
9.10.4	TopoGeom_addTopoGeom	578
9.10.5	toTopoGeom	579
9.11	拓扑几何函数器	579
9.11.1	GetTopoGeomElementArray	579
9.11.2	GetTopoGeomElements	580
9.11.3	ST_SRID	580
9.12	拓扑几何输出	581
9.12.1	AsGML	581
9.12.2	AsTopoJSON	583
9.13	拓扑空关系	585
9.13.1	Equals	585
9.13.2	Intersects	586
9.14	输入和输出拓扑	586
9.14.1	使用拓扑输出器	587
9.14.2	使用拓扑输入器	587

---

<b>10</b>	<b>栅格数据管理、栅格和应用程序</b>	<b>588</b>
10.1	加栅格和建栅格	588
10.1.1	使用 raster2pgsql 加栅格	588
10.1.1.1	用法示例	588
10.1.1.2	raster2pgsql 栅格	589
10.1.2	使用 PostGIS 栅格函数建栅格	590
10.1.3	使用 “out db” 云栅格	591
10.2	栅格目录	592
10.2.1	栅格列目录	592
10.2.2	栅格概述	593
10.3	使用 PostGIS Raster 建自定义应用程序	593
10.3.1	PHP 示例使用 ST_AsPNG 与其他栅格函数行输出	594
10.3.2	ASP.NET C# 示例使用 ST_AsPNG 与其他栅格函数行输出	594
10.3.3	将栅格输出为图像文件的 Java 控制台应用程序	596
10.3.4	使用 PLPython 通 SQL 输出图像	597
10.3.5	使用 PSQL 输出栅格	598
<b>11</b>	<b>栅格参考</b>	<b>599</b>
11.1	栅格支持数据类型	600
11.1.1	geomval	600
11.1.2	addbandarg	600
11.1.3	rastbandarg	600
11.1.4	raster	601
11.1.5	reclassarg	601
11.1.6	summarystats	602
11.1.7	unionarg	602
11.2	栅格管理	603
11.2.1	AddRasterConstraints	603
11.2.2	DropRasterConstraints	605
11.2.3	AddOverviewConstraints	606
11.2.4	DropOverviewConstraints	607
11.2.5	PostGIS_GDAL_Version	607
11.2.6	PostGIS_Raster_Lib_Build_Date	607
11.2.7	PostGIS_Raster_Lib_Version	608
11.2.8	ST_GDALDrivers	608
11.2.9	ST_Contour	613
11.2.10	ST_InterpolateRaster	614
11.2.11	UpdateRasterSRID	615
11.2.12	ST_CreateOverview	615

11.3 栅格造器	616
11.3.1 ST_AddBand	616
11.3.2 ST_AsRaster	619
11.3.3 ST_Band	621
11.3.4 ST_MakeEmptyCoverage	622
11.3.5 ST_MakeEmptyRaster	624
11.3.6 ST_Tile	625
11.3.7 ST_Retile	627
11.3.8 ST_FromGDALRaster	627
11.4 栅格器	628
11.4.1 ST_GeoReference	628
11.4.2 ST_Height	629
11.4.3 ST_IsEmpty	630
11.4.4 ST_MemSize	630
11.4.5 ST_MetaData	631
11.4.6 ST_NumBands	632
11.4.7 ST_PixelHeight	632
11.4.8 ST_PixelWidth	633
11.4.9 ST_ScaleX	635
11.4.10 ST_ScaleY	635
11.4.11 ST_RasterToWorldCoord	636
11.4.12 ST_RasterToWorldCoordX	637
11.4.13 ST_RasterToWorldCoordY	638
11.4.14 ST_Rotation	639
11.4.15 ST_SkewX	639
11.4.16 ST_SkewY	640
11.4.17 ST_SRID	641
11.4.18 ST_Summary	642
11.4.19 ST_UpperLeftX	642
11.4.20 ST_UpperLeftY	643
11.4.21 ST_Width	643
11.4.22 ST_WorldToRasterCoord	644
11.4.23 ST_WorldToRasterCoordX	645
11.4.24 ST_WorldToRasterCoordY	645
11.5 栅格波段器	646
11.5.1 ST_BandMetaData	646
11.5.2 ST_BandNoDataValue	647
11.5.3 ST_BandIsNoData	648
11.5.4 ST_BandPath	649

---

11.5.5	ST_BandFileSize	650
11.5.6	ST_BandFileTimestamp	650
11.5.7	ST_BandPixelType	651
11.5.8	ST_MinPossibleValue	652
11.5.9	ST_HasNoBand	652
11.6	格像素器和置器	653
11.6.1	ST_PixelAsPolygon	653
11.6.2	ST_PixelAsPolygons	654
11.6.3	ST_PixelAsPoint	655
11.6.4	ST_PixelAsPoints	655
11.6.5	ST_PixelAsCentroid	656
11.6.6	ST_PixelAsCentroids	657
11.6.7	ST_Value	658
11.6.8	ST_NearestValue	661
11.6.9	ST_SetZ	663
11.6.10	ST_SetM	664
11.6.11	ST_Neighborhood	665
11.6.12	ST_SetValue	667
11.6.13	ST_SetValues	668
11.6.14	ST_DumpValues	676
11.6.15	ST_PixelOfValue	677
11.7	格器	679
11.7.1	ST_SetGeoReference	679
11.7.2	ST_SetRotation	680
11.7.3	ST_SetScale	681
11.7.4	ST_SetSkew	682
11.7.5	ST_SetSRID	683
11.7.6	ST_SetUpperLeft	683
11.7.7	ST_Resample	684
11.7.8	ST_Rescale	685
11.7.9	ST_Reskew	686
11.7.10	ST_SnapToGrid	688
11.7.11	ST_Resize	689
11.7.12	ST_Transform	690
11.8	格波段器	693
11.8.1	ST_SetBandNoDataValue	693
11.8.2	ST_SetBandIsNoData	694
11.8.3	ST_SetBandPath	695
11.8.4	ST_SetBandIndex	697

---

11.9 栅格波段和分析	698
11.9.1 ST_Count	698
11.9.2 ST_CountAgg	699
11.9.3 ST_Histogram	700
11.9.4 ST_Quantile	702
11.9.5 ST_SummaryStats	704
11.9.6 ST_SummaryStatsAgg	706
11.9.7 ST_ValueCount	707
11.10 栅格输入	710
11.10.1 ST_RastFromWKB	710
11.10.2 ST_RastFromHexWKB	710
11.11 栅格输出	711
11.11.1 ST_AsBinary/ST_AsWKB	711
11.11.2 ST_AsHexWKB	712
11.11.3 ST_AsGDALRaster	713
11.11.4 ST_AsJPEG	714
11.11.5 ST_AsPNG	715
11.11.6 ST_AsTIFF	716
11.12 栅格处理：代数函数	717
11.12.1 ST_Clip	717
11.12.2 ST_ColorMap	720
11.12.3 ST_Grayscale	723
11.12.4 ST_Intersection	725
11.12.5 ST_MapAlgebra (callback function version)	727
11.12.6 ST_MapAlgebra (expression version)	733
11.12.7 ST_MapAlgebraExpr	736
11.12.8 ST_MapAlgebraExpr	738
11.12.9 ST_MapAlgebraFct	742
11.12.10 ST_MapAlgebraFct	746
11.12.11 ST_MapAlgebraFctNgb	750
11.12.12 ST_Reclass	752
11.12.13 ST_Union	754
11.13 内置代数函数	755
11.13.1 ST_Distinct4ma	755
11.13.2 ST_InvDistWeight4ma	756
11.13.3 ST_Max4ma	757
11.13.4 ST_Mean4ma	758
11.13.5 ST_Min4ma	760
11.13.6 ST_MinDist4ma	761

11.13.	<b>\$T_Range4ma</b>	761
11.13.	<b>\$T_StdDev4ma</b>	762
11.13.	<b>\$T_Sum4ma</b>	763
11.14.	<b>栅格理：DEM（高程）</b>	765
11.14.	<b>\$T_Aspect</b>	765
11.14.	<b>\$T_HillShade</b>	766
11.14.	<b>\$T_Roughness</b>	768
11.14.	<b>\$T_Slope</b>	769
11.14.	<b>\$T_TPI</b>	770
11.14.	<b>\$T_TRI</b>	771
11.15.	<b>栅格理：栅格到几何</b>	772
11.15.	<b>Box3D</b>	772
11.15.	<b>\$T_ConvexHull</b>	772
11.15.	<b>\$T_DumpAsPolygons</b>	773
11.15.	<b>\$T_Envelope</b>	775
11.15.	<b>\$T_MinConvexHull</b>	775
11.15.	<b>\$T_Polygon</b>	776
11.16.	<b>栅格算符</b>	778
11.16.	<b>&amp;&amp;</b>	778
11.16.	<b>&amp;&lt;</b>	778
11.16.	<b>&amp;&gt;</b>	779
11.16.	<b>≠</b>	780
11.16.	<b>@</b>	780
11.16.	<b>=</b>	781
11.16.	<b>≠</b>	782
11.17.	<b>栅格和栅格波段空关系</b>	782
11.17.	<b>\$T_Contains</b>	782
11.17.	<b>\$T_ContainsProperly</b>	783
11.17.	<b>\$T_Covers</b>	784
11.17.	<b>\$T_CoveredBy</b>	785
11.17.	<b>\$T_Disjoint</b>	786
11.17.	<b>\$T_Intersects</b>	787
11.17.	<b>\$T_Overlaps</b>	788
11.17.	<b>\$T_Touches</b>	789
11.17.	<b>\$T_SameAlignment</b>	789
11.17.	<b>\$T_NotSameAlignmentReason</b>	790
11.17.	<b>\$T_Within</b>	791
11.17.	<b>\$T_DWithin</b>	792
11.17.	<b>\$T_DFullyWithin</b>	793

11.18.0	格式提示	794
11.18.0	Out-DB 格式	794
11.18.1	包含多个文件的目录	794
11.18.1	最大打开文件数	795
11.18.1.2	整个系统最大打开文件数	795
11.18.1.2	每个进程的最大打开文件数	795
<b>12</b>	<b>PostGIS 扩充</b>	<b>798</b>
12.1	地址标准化工具	798
12.1.1	解析器如何工作	798
12.1.2	地址标准化器类型	798
12.1.2.1	stdaddr	798
12.1.3	地址标准化表	799
12.1.3.1	rules table	799
12.1.3.2	lex table	802
12.1.3.3	gaz table	802
12.1.4	地址标准化器功能	802
12.1.4.1	debug_standardize_address	802
12.1.4.2	parse_address	804
12.1.4.3	standardize_address	805
12.2	Tiger 地理处理器	807
12.2.1	Drop_Indexes_Generate_Script	807
12.2.2	Drop_Nation_Tables_Generate_Script	808
12.2.3	Drop_State_Tables_Generate_Script	809
12.2.4	Geocode	810
12.2.5	Geocode_Intersection	812
12.2.6	Get_Geocode_Setting	813
12.2.7	Get_Tract	814
12.2.8	Install_Missing_Indexes	815
12.2.9	Loader_Generate_Census_Script	815
12.2.10	Loader_Generate_Script	817
12.2.11	Loader_Generate_Nation_Script	819
12.2.12	Missing_Indexes_Generate_Script	820
12.2.13	Normalize_Address	821
12.2.14	Page_Normalize_Address	822
12.2.15	Print_Addy	824
12.2.16	Reverse_Geocode	825
12.2.17	Topology_Load_Tiger	827
12.2.18	Set_Geocode_Setting	829



<b>13 PostGIS 特殊函数索引</b>	<b>830</b>
13.1 PostGIS 聚合函数	830
13.2 PostGIS 窗口函数	831
13.3 PostGIS SQL-MM 兼容函数	831
13.4 PostGIS 地理支持函数	835
13.5 PostGIS 网格支持函数	836
13.6 PostGIS 几何/地理/网格输出函数	841
13.7 PostGIS 边界框函数	841
13.8 支持 3D 的 PostGIS 函数	842
13.9 PostGIS 曲线几何支持函数	847
13.10 PostGIS 多面体曲面支持函数	850
13.11 PostGIS 函数支持矩阵	853
13.12 新的、增加的或更改的 PostGIS 函数	871
13.12.1 PostGIS 新增功能或增加功能 (3.5)	871
13.12.2 PostGIS 新增功能或增加功能 (3.4)	873
13.12.3 PostGIS 新增功能或增加功能 (3.3)	874
13.12.4 PostGIS 新增功能或增加功能 (3.2)	875
13.12.5 PostGIS 新增功能或增加功能 (3.1)	876
13.12.6 PostGIS 新增功能或增加功能 (3.0)	877
13.12.7 PostGIS 新增功能或增加功能 (2.5)	879
13.12.8 PostGIS 新增功能或增加功能 (2.4)	880
13.12.9 PostGIS 新增功能或增加功能 (2.3)	881
13.12.10 PostGIS 新增功能或增加功能 (2.2)	883
13.12.11 PostGIS 新增功能或增加功能 (2.1)	885
13.12.12 PostGIS 新增功能或增加功能 (2.0)	890
13.12.13 PostGIS 新增功能或增加功能 (1.5)	899
13.12.14 PostGIS 新增功能或增加功能 (1.4)	900
13.12.15 PostGIS 新增功能或增加功能 (1.3)	901
<b>14 告警</b>	<b>902</b>
14.1 告警事件	902
14.2 告警文档	902
<b>A 附录</b>	<b>903</b>
A.1 PostGIS 3.6.0	903

## 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 3.6.0dev



本作品已得 [CC BY SA](https://creativecommons.org/licenses/by-sa/3.0/) 作共用署名-相同方式共享 3.0 许可。可随意使用部份材料，任何您喜的方式，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

## 介绍

PostGIS 是 PostgreSQL 关系数据库的空缺扩展。由 Refractions Research Inc 创建，作为空缺数据库技术的研究项目。Refractions 是一家位于加拿大不列颠哥伦比亚省多利市的 GIS 和数据咨询公司，从事数据集成和定制软件开发。

PostGIS 现在是 OSGeo 基金会有一个项目。世界各地大量 FOSS4G 开发商和公司从 PostGIS 的功能和多态性中受益匪浅，他们正在开发和帮助 PostGIS。

PostGIS 项目开始于 PostGIS 创建了一个数据源，以便在 OGC 和 SQL/MM 空缺标准、高空拓扑构建（覆盖范围、表面、网）、用于查看和 GIS 数据的桌面用户界面工具的数据源以及基于 Web 的工具的数据源等领域提供重要的 GIS 功能。我们将提供支持和增强功能，以更好地做出贡献。

### 1.1 项目指导委员会

PostGIS 项目指导委员会（PSC）负责项目方向、发布周期、文档和支持活动。委员会涉及 PostGIS 的其他项目投票，例如整体用户支持、接受和采用 PostGIS 社区的补丁、开发人员提交期限、新委员会成员以及重要的 API 更改。

**Raúl Marín Rodríguez** MVT 功能、性能、稳定性和定性改进、GitHub 策展、PostGIS 和 PostgreSQL 版本更新

**Regina Obe** 支持集成和网站维护，Windows 生成和构建，文档编写，将 PostGIS 与 PostgreSQL 发布版本集成，X3D 支持，TIGER 地理支持，管理功能。

**Darafei Praliaskouski** 索引改进，故障修复和几何/地理函数改进，SFCGAL，风格，GitHub 维护，以及支持集成。

**Paul Ramsey**（主席） PostGIS 项目的联合创始人。全面的修复、地理功能、地理和几何索引（2D、3D、n 索引和任何空缺索引）、几何内部、GEOS 功能集成以及与 GEOS 版本的集成、与 PostgreSQL 版本的集成、加压器/程序、shapefile GUI 加压器。

**Sandro Santilli** 修复和改进，支持集成，Git 像管理，管理功能，集成新的 GEOS 功能并与 GEOS 发布版本，拓扑支持，以及风格框架和底层 API 函数。

### 1.2 当前的核心贡献者

**Nicklas Avén** 距离函数增强（包括 3D 距离和关系函数）和附加功能、Tiny WKB (TWKB) 输出格式和一般用户支持

**Loïc Bartoletti** SFCGAL 的增区和以及持集成支持

**Dan Baston** 几何聚功能添加、其他几何算法增、GEOS 增和一般用支持

**Martin Davis** GEOS 增功能和文档

**Björn Harrtell** MapBox 矢量瓦片、GeoBuf 和 Flatgeobuf 函数。Gitea 和 GitLab。

**Aliaksandr Kalenik** 几何理, PostgreSQL GiST, 常

## 1.3 去的核心献者

**Bborie Park** 前 PSC 成。格, 与 GDAL 集成, 格加器, 用支持, 常, 在各种操作系 (Slackware, Mac, Windows 等) 上运行。

**Mark Cave-Ayland** 前 PSC 成。和活、空索引性和定、加器/程序 and shapefile GUI 加器整、新功能集成和增。

**Jorge Arévalo** 格开, GDAL 功能, 加器

**Olivier Courtin** (荣誉) XML (KML, GML) /GEOJSON 入/出函数、3D 和。

**Chris Hodgson** 前 PSC 成。一般开, 站点和建机器人, OSGeo 孵化管理

**Mateusz Loskot** CMake 对 PostGIS 的支持、原始格加器的造以及 Python 版本的低格 API 函数

**Kevin Neufeld** 前 PSC 成。文档和文档助、Buildbot、PostGIS 新中的高用支持以及增的 PostGIS 功能。

**Dave Blasby** PostGIS 的原始开者/合始人。写了服器端象、索引定和多服器端分析函数。

**Jeff Lounsbury** shapefile loader/dumper 的原始开人。

**Mark Leslie** 持和开核心功能。增的曲功能。形状文件 GUI 加器。

**Pierre Racine** 后 GIS 格。光架、原型和程助

**David Zwarg** 格开 (主要是地代数分析函数)

## 1.4 其它献者

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

企业/个人/一些公司/PostGIS 项目/人/，托管或直接/金。按字母/序排列：

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

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

众筹活动 众筹活动是我为了获得急需的功能而开展的活动，这些功能可以为大量的人提供服务。每个活动都围绕着一个特定功能或一功能。每个赞助商都会投入所需资金的一小部分，并且有足够的个人/组织捐款，我就有足够的资金来支付将帮助多人的工作。如果您某个功能有想法，并且相信许多其他人愿意共同帮助，请将您的想法发布到 [PostGIS 新倡议](#)，我可以共同实现它。

PostGIS 2.0.0 是我实现此策略的第一个版本。我使用了 [PledgeBank](#)，并从中组织了数次成功的活动。

**postgistopology** 有 10 位赞助商每人捐赠了 250 美元来构建“toTopoGeometry”函数并加入 2.0.0 版本中的拓扑支持。一目了然已实现。

**postgis64windows** 大约 20 位赞助商每人捐赠了 100 美元，用以支付在 Windows 上解决 PostGIS 64 位所需的工作。一目了然已实现。

重要的支持 **GEOS** 几何操作

**GDAL** 地理空间数据抽象用于支持 PostGIS 2 中引入的大部分功能。同时，GDAL 中支持 PostGIS 所需的更改也将回馈 GDAL 项目。

**PROJ** 地图投影

最后但同样重要的是，PostGIS 所依赖的当然大物 **PostgreSQL**。PostGIS 的速度和灵活性很大程度上得益于 PostgreSQL 提供的可扩展性、强大的查询规划器、GIST 索引以及丰富的 SQL 功能。

## 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：

```
tar -xvzf postgis-3.6.0dev.tar.gz
cd postgis-3.6.0dev
./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。在许多情况下，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。

本文提供一般性指南。如果您正在使用 Windows 或其他操作系统进行安装，您可以在 [PostGIS 用户指南](#) 或 [PostGIS 官方 Wiki](#) 中找到更详细的帮助。

可以在 PostGIS 二进制包中找到许多操作系统的 [PostGIS 二进制包列表](#)

如果您是 Windows 用户，您可以通过 [Stackbuilder](#) 或 [PostGIS Windows 下载站点](#) 获得预打包的版本。我们提供非常前沿的 [Windows 二进制版本](#)，通常每周构建一次，或者在有令人兴奋的新功能生成时构建。您可以使用这些版本来更新正在运行中的 PostGIS 部署。

PostGIS 模块是 PostgreSQL 服务的扩展。PostGIS 3.6.0dev 需要完整的 PostgreSQL 服务器进行安装。您可以在 PostgreSQL 12 - 17 之间构建。不支持旧版本的 PostgreSQL。

如果您没有安装 PostgreSQL，请参考 [PostgreSQL 安装指南](https://www.postgresql.org)。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++ ：

```
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），试试这个技巧。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。这些是在 Linux 上用 shell 写的，不适用于 Windows 或 Mac。

### 2.2.1 获取源代码

从以下站点 <https://postgis.net/stuff/postgis-3.6.0dev.tar.gz> 获取源代码的存档

```
wget https://postgis.net/stuff/postgis-3.6.0dev.tar.gz
tar -xvzf postgis-3.6.0dev.tar.gz
cd postgis-3.6.0dev
```

这将在当前工作目录中创建一个名为 `postgis-3.6.0dev` 的目录。

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

```
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。

```
./configure
```

### 2.2.2 安装要求

要构建和使用 PostGIS，您需要：

不可缺少

- PostgreSQL 12 - 16。需要完整安装 PostgreSQL (包括服务器)。PostgreSQL 可从 <https://www.postgresql.org> 获取。  
参考 <https://trac.osgeo.org/postgis/wiki/UsersWikiPostgreSQLPostGIS> 中完整的 PostgreSQL/PostGIS 和 PostGIS/GEOS 支持表
- 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` 版本。其他版本的 make 可能无法正确处理 PostGIS Makefile。
- Proj 重投影。需要版本 6.1 或更高版本。Proj 用于在 PostGIS 中提供坐重视投影支持。Proj 可以从 <https://proj.org/> 下载。
- GEOS 几何库，版本 3.8.0 或更高版本，但为了充分利用所有新的函数和特性，需要使用 GEOS 3.12+。您可以从以下链接下载 GEOS：<https://libgeos.org>。
- LibXML2、2.5.x 或更高版本。目前，输入函数 (ST\_GeomFromGML 和 ST\_GeomFromKML) 使用 LibXML2。您可以从 <https://gitlab.gnome.org/GNOME/libxml2/-/releases> 下载 LibXML2。

- JSON-C 0.9 或更高版本。JSON-C 目前用于按 ST\_GeomFromGeoJSON 输入 GeoJSON。JSON-C 可从 <https://github.com/json-c/json-c/releases/> 下载。
- GDAL, version 3+ is preferred. This is required for raster support. <https://gdal.org/download.html>.
- 要使用 PostgreSQL+JIT 行，需要 LLVM 版本 6 或更高版本。参 <https://trac.osgeo.org/postgis/ticket/4125>。

可的

- GDAL (可) 当您不需要格式才可以将其省略。确保用您要使用的程序，如 Section 3.2 中所述。
- GTK (需要 GTK+2.0, 2.8+)。用于 shp2pgsql-gui, 一个 shapefile loader。它位于 <http://www.gtk.org/>。
- SFCGAL, 1.4.1 or higher is required and 1.5.0+ is needed to be able to use all functionality. SFCGAL can be used to provide additional 2D and 3D advanced analysis functions to PostGIS of Chapter 8. And also allow to use SFCGAL rather than GEOS for some 2D functions provided by both backends (like ST\_Intersection or ST\_Area, for instance). A PostgreSQL configuration variable `postgis.backend` allow end user to control which backend he want to use if SFCGAL is installed (GEOS by default). Nota: SFCGAL 1.2 require at least CGAL 4.3 and Boost 1.54 (cf: <https://sfcgal.org>) <https://gitlab.com/sfcgal/SFCGAL/>.
- 要建 Section 12.1, 您需要 PCRE <http://www.pcre.org> (通常安装在 nix 系上)。到 PCRE 后, 将自建 Section 12.1。或者, 指定它在配置期有效。 `--with-pcre-dir=/path/to/pcre`。
- 要用 ST\_AsMVT, 您需要 ProtoBuf-C (行) 和 ProtoC-C 器 (建)。要 protobuf-c 的正确最低版本, 需要 pkg-config。参 [protobuf-c](#)。默情况下, PostGIS 使用 Wagyu 快速估 MVT 多形, 但需要 C++11 器。使用 CXXFLAGS 使用您用于 PostgreSQL 安装的相同器。如果要禁用此功能并改用 GEOS, 在配置期指定它。 `--without-wagyu`。
- CUnit (CUnit)。是回所需要的。 <http://cunit.sourceforge.net/>
- DocBook (xsltproc) 建文档需要。Docbook 可从 <http://www.docbook.org/> 取。
- DBLatex (dbratex) 需要以 PDF 格式建文档。它位于 <http://dbratex.sourceforge.net/>。
- ImageMagick (convert) 生成文档中使用的像。它位于 <http://www.imagemagick.org/>。

### 2.2.3 建配置

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

#### ./configure

如果未提供任何参数, 此命令将自找在系上建 PostGIS 源代码所需的件和。 `./configure` 是一种常用的用法, 但它接受一些参数, 以防您在非准位置有必要的或程序。

以下列表示了常用参数: 有关完整列表, 使用 `--help` 或 `--help=short` 参数。

`--with-library-minor-version` 从 PostGIS 3.0 开始, 默生成的文件将不再将次要版本作文件名的一部分。意味着所有 PostGIS 3 都将以 `postgis-3` 尾。做是使 `pg_upgrade` 更容易, 缺点是您只能在服器中安装一个版本的 PostGIS 3 系列。要取文件的旧行 (包括次要版本): 例如 `postgis-3.0` 将此开关添加到您的配置句中。

`--prefix=PREFIX` 指定要安装 PostGIS 和 SQL 脚本的位置。默情况下, 它将与到的 PostgreSQL 安装位置相同。

**Caution**

此参数当前已坏，只会安装在 PostgreSQL 的位置。有关此问题的跟踪，请参考 <http://trac.osgeo.org/postgis/ticket/635>。

- with-pgconfig=FILE** PostgreSQL 有一个名为 **pg\_config** 的实用程序，使 PostGIS 等扩展能定位 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，提供格式支持 **gdal-config** 所需的功能，以便组件安装能找到 GDAL 安装目录。使用此参数 (**--with-gdalconfig=/path/to/gdal-config**) 手动指定 PostGIS 将其构建的特定 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，它会告诉您在安装组件在哪里安装 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。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，以便组件安装程序找到 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-pcredir=DIR** **PCRE** 是 address\_standardizer 扩展所需的 BSD 许可的 Perl 兼容正则表达式库。使用此参数 (**--with-pcredir=/path/to/pcredir**) 手动指定 PostGIS 将其构建的特定 PCRE 的安装目录。
- with-gui** 将数据输入 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** 在没有拓扑支持的情况下运行。拓扑所需的所有库都是在 postgis-3.6.0dev 中构建的，因此没有相关的库。
- with-gettext=no** 默认情况下，会安装 gettext 并使用它运行，但如果在导致加程序坏的不兼容库下运行，此命令可以禁用它。使用此功能的配置可以解决的示例，请参考 <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，用于某些阶段。
- 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 的实用 PATH。
- without-phony-revision** 禁用 postgis\_revision.h 更新以匹配 Git 存储库中的当前 HEAD。

**Note**

如果要从代存储库获取 PostGIS，先运行以下脚本

**./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—

### make

如果在出的最后一行看到 “PostGIS was built successfully. Ready to install.”，表示您已完成从 PostGIS v1.4.0 开始，所有函数都有从文档生成的注释。如果您希望稍后将些注释安装到空数据库数据中，运行需要的 docbook 命令。postgis\_comments.sql 和其他包注释文件 raster\_comments.sql、topology\_comments.sql 也打包在 doc 文件中的 tar.gz 中，因此如果从 tar 包中安装，无需运行注释。注释包含在 CREATE EXTENSION 安装中。

### make comments

它是在 PostGIS 2.0 中引入的。生成适合快速参考和阅读的 HTML 备忘，供正在学习的人使用。xsltproc 是必需的，生成 4 个文件。topology\_cheatsheet.html, tiger\_geocoder\_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

### make cheatsheets

## 2.2.5 构建和部署 PostGIS 扩展

如果您使用的是 PostgreSQL 9.1 或更高版本，会自行构建并安装 PostGIS 扩展。

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

### make comments

如果从存档文件构建，其中一些注释文件已构建，因此无需构建注释。

如果您 PostgreSQL 9.1 运行构建，在 make 安装过程中自行构建扩展。如果需要，可以从扩展文件生成，或者根据需要在同一台服务器上复制文件。

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



### Note

make check 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](#)

无论操作系统如何，扩展文件在同一版本的 PostGIS 中始终相同。只要已安装 PostGIS 二进制文件，就可以将扩展文件从一个操作系统复制到另一个操作系统。

如果您想在与您的开发不同的服务器上手动安装扩展，您需要将扩展文件中的以下文件复制到 PostgreSQL 安装的 PostgreSQL/share/extension 文件中常 PostGIS 所需的二进制文件（如果服务器上没有它）。

- 些控制文件表示信息，例如要安装的扩展版本（如果未指定）。postgis.control、postgis\_topology.control。

- 每个扩展名的 /sql 文件中的所有文件。注意，必须复制到 PostgreSQL 共享/扩展文件的 extensions/postgis/sql/\*.sql, extensions/postgis\_topology/sql/\*.sql

完成一步后，你可以在 PgAdmin> 中看到 postgis、postgis\_topology 作为可用的扩展。

如果使用 psql，可以通过以下行来检查是否已安装扩展：

```
SELECT name, default_version, installed_version
FROM pg_available_extensions WHERE name LIKE 'postgis%' or name LIKE 'address%';
```

name	default_version	installed_version
address_standardizer	3.6.0dev	3.6.0dev
address_standardizer_data_us	3.6.0dev	3.6.0dev
postgis	3.6.0dev	3.6.0dev
postgis_raster	3.6.0dev	3.6.0dev
postgis_sfcgal	3.6.0dev	
postgis_tiger_geocoder	3.6.0dev	3.6.0dev
postgis_topology	3.6.0dev	

(6 rows)

如果您正在数据库的数据中安装了扩展程序，您将在 installed\_version 列中看到提及。如果您没有收到任何消息，这意味着您的服务器上根本没有安装 postgis 扩展。PgAdmin III 1.14+ 将在数据库的扩展部分中提供此信息，甚至允许通过右键单击行来升级或卸载。

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

```
CREATE EXTENSION postgis;
CREATE EXTENSION postgis_raster;
CREATE EXTENSION postgis_sfcgal;
CREATE EXTENSION fuzzystmatch; --needed for postgis_tiger_geocoder
--optional used by postgis_tiger_geocoder, or can be used standalone
CREATE EXTENSION address_standardizer;
CREATE EXTENSION address_standardizer_data_us;
CREATE EXTENSION postgis_tiger_geocoder;
CREATE EXTENSION postgis_topology;
```

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。

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

```
List of installed extensions
-[ RECORD 1 ]-----
Name          | postgis
Version       | 3.6.0dev
Schema        | public
Description   | PostGIS geometry, geography, and raster spat..
-[ RECORD 2 ]-----
Name          | postgis_raster
Version       | 3.0.0dev
Schema        | public
Description   | PostGIS raster types and functions
-[ RECORD 3 ]-----
Name          | postgis_tiger_geocoder
Version       | 3.6.0dev
Schema        | tiger
Description   | PostGIS tiger geocoder and reverse geocoder
-[ RECORD 4 ]-----
Name          | postgis_topology
Version       | 3.6.0dev
```

Schema	topology
Description	PostGIS topology spatial types and functions

**Warning**

无法直接安装扩展表 `spatial_ref_sys`、`layer`、`topology`。只有在安装相兼容的 `postgis` 或 `postgis_topology` 扩展时，它们才会被安装，而它们似乎只会在安装整个数据时生成。从 PostGIS 2.0.1 开始，在安装数据时会安装未与 PostGIS 打包的 `srid` 表，因此不要随意更改我打包的 `srid`，并希望您的更改会存在。如果您更新了，提交工单。由于扩展表的表是使用 `CREATE EXTENSION` 创建的，并且假定在特定版本的扩展中是相同的，因此它们永远不会被安装。这些行内置在当前的 PostgreSQL 扩展模型中，所以我无法改变它们一点。

如果你在没有使用扩展系数的情况下安装了 3.6.0dev 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 之前切换到扩展安装方式。

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

## 2.2.6 测试

要测试 PostGIS，运行以下命令

**make check**

此命令使用 PostgreSQL 数据目录生成的运行各种测试和回环。

**Note**

如果您使用非标准 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` 环境变量来运行测试。它不是使用 `--with-pgconfig` 配置参数指定的 PostgreSQL。测试路径以匹配在配置期间找到的 PostgreSQL。或者不可避免的麻烦做好准备。

如果成功，`make check` 将生成近 500 个测试的输出。结果将类似于以下（下面省略了多行）：

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

.
.
.

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

```
.
.
.
Running tests
.
.
.
Run tests: 134
Failed: 0

-- if you build with SFCGAL
.
.
.
Running tests
.
.
.
Run tests: 13
Failed: 0

-- if you built with raster support
.
.
.
Run Summary:  Type  Total   Ran  Passed  Failed  Inactive
              suites   12    12   n/a     0       0
              tests   65    65    65     0       0
              asserts 45896 45896 45896   0       n/a

.
.
.
Running tests
.
.
.
Run tests: 101
Failed: 0

-- topology regress
.
.
.
Running tests
```

```

.
.
.

Run tests: 51
Failed: 0

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

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

.
.
.

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

```

`postgis_tiger_geocoder` and `address_standardizer` 展目前支持准的 PostgreSQL 安装。要些, 使用以下命令。注意: 如果您已在 PostGIS 代文件的根目中行了安装, 不需要行安装。

于 `address_standardizer`:

```

cd extensions/address_standardizer
make install
make installcheck

```

出似于以下内容:

```

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

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

```

于 `tiger` 地理器, 确保您的 PostgreSQL 例中具有可用的 `postgis` 和 `fuzzystrmatch` 展。如果您在 `address_standardizer` 支持下建了 `postgis`, `address_standardizer` 也将:

```

cd extensions/postgis_tiger_geocoder
make install
make installcheck

```

出似于以下内容:

```

===== dropping database "contrib_regression" =====
DROP DATABASE
===== creating database "contrib_regression" =====
CREATE DATABASE
ALTER DATABASE

```



```

===== installing fuzzystmatch =====
CREATE EXTENSION
===== installing postgis =====
CREATE EXTENSION
===== installing postgis_tiger_geocoder =====
CREATE EXTENSION
===== installing address_standardizer =====
CREATE EXTENSION
===== running regression test queries =====
test test-normalize_address ... ok
test test-pagc_normalize_address ... ok

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

```

## 2.2.7 安装

要安装 PostGIS, 运行以下命令

### make install

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

- 加程序和二进制文件安装在 `[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` 注释, 运行以下命令来安装这些 SQL 文件: `postgis_comments.sql`, `raster_comments.sql`

### make comments-install



#### Note

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

## 2.3 安装和使用地址标准化工具

`address_standardizer` 扩展是一个独立的包, 需要单独下载。它包含在 PostGIS 2.2 中。有关 `address_standardizer` 可以运行的操作以及如何配置它的其他信息, 参见 Section 12.1。

地址标准化工具可以与 PostGIS 打包的 Tiger 地理索引器扩展合用, 作为所索引的 `Normalize Address` 的替代品。要用作替代品, 参见 Section 2.4.2。您可以将其用作您自己的地理索引器的构建, 或使用它来标准化您的地址, 以便更轻松地比较地址。

地址标准化工具依赖于 PCRE, 它通常已安装在许多 Unix 系统上, 但您可以在以下位置下载最新版本: <http://www.pcre.org>。如果在 Section 2.2.3 中找到 PCRE, 将来自构建地址标准化工具扩展。如果您想使用自定义 `pcre` 安装, 运行到 `--with-pcredir=/path/to/pcre`, 其中 `/path/to/pcre` 是 `pcre include` 和 `lib` 目录的根文件。

在 Windows 上, PostGIS 2.1 及更高版本附了地址标准化工具扩展, 因此您无需即可立即使用。直接输入 `CREATE EXTENSION` 步骤。

安装后, 可以连接到目录数据并运行以下 SQL:

```
CREATE EXTENSION address_standardizer;
```

出于下面的目的，我不需要 rules、gaz、lex 表

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

输出似乎于以下内容

num	street	city	state	zip
1	Devonshire Place PH301	Boston	MA	02109

## 2.4 安装、升级 Tiger 地理引擎并加载数据

像 Tiger 地理引擎的附加文件可能未包含在您的 PostGIS 分中。如果您缺少 tiger 地理引擎或想要比您的安装提供的更新版本，那么可以使用适用于您 PostgreSQL 版本的 `share/extension/postgis_tiger_geocoder.*` 文件，这些文件可以从 [Windows 未发布版本](#) 部分的安装包中获取。尽管这些安装包是 Windows 准的，但 `postgis_tiger_geocoder` 扩展文件将在任何操作系统上工作，因为它只是一个 SQL/plpgsql 扩展。

### 2.4.1 在 PostGIS 数据中应用 Tiger 地理引擎

- 出于本主的目的，我假设您已 PostgreSQL 安装了 `postgis_tiger_geocoder` 扩展。
- 使用 `psql`、`pgAdmin` 或其他工具连接到数据库，然后执行以下 SQL 命令：如果要在已有 PostGIS 的数据库上安装，无需执行第一步。如果已安装扩展，不需要执行第二步 `fuzzystrmatch`。

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

如果已安装 `postgis_tiger_geocoder` 扩展，并且只想更新到最新版本，执行：

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

如果生成自己的条目或 `tiger.loader_platform` 和 `tiger.loader_variables` 行更改，可能需要更新它。

- 若要安装是否成功，在目标数据库中执行以下 SQL：

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

输出似乎于以下内容

address	streetname	stretypeabbrev	zip
1   Devonshire   PL			02109

- `tiger.loader_platform` 生成表的新行，其中包含可执行文件或服务器的路径。  
sh 作在固定后生成名 `debbie` 的配置文件的示例，执行以下命令：

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

然后将 *declare\_sect* 列中的路径替换为适合 Debian 的 `pg`、`unzip`、`shp2pgsql`、`psql` 等路径位置。如果您不替换此 `loader_platform` 表，它将包含默认的路径，并且您必须在生成脚本后替换生成的脚本。

- 从 PostGIS 2.4.1 开始，5 位制表区域 `zcta5` 加载步骤已修改以加载当前 `zcta5` 数据，并且在调用 `Loader Generate Nation Script` 的一部分。默认情况下它是关闭的，因为它需要相当长的时间来加载（20 到 60 分钟），占用相当多的磁盘空间，并且不经常使用。

要使用它，执行以下操作：

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

如果添加了边界限制器并将其限制在边界内的 ZIP，则在存在 ZCTA5 的情况下将使用 `Geocode` 函数。当返回的地址没有行政区划，将使用 `Reverse_Geocode` 函数，通常生成在高速公路上的反向地理信息。

- 如果您有到服务器的快速网络连接，则在服务器根目录或本地磁盘上创建一个名为 `gisdata` 的文件。该文件是 Tiger 文件下和物理的位置。如果您将文件位于服务器根目录不介意，或者只是想更改不同的文件行存储，则在 `tiger.loader_variables` 表中的字段 `staging_fold`。
- `gisdata` 在文件中创建一个名为 `temp` 的文件。或者，建由 `staging_fold` 指示的文件。是加载程序提取下载的 Tiger 数据的地方。
- 然后运行 `Loader Generate Nation Script` SQL 函数，确保使用自定义配置文件的名称并将脚本复制到 `.sh` 或 `.bat` 文件。例如，如果要使用新的配置文件加载国家/地区：

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

- 运行命令行脚本以加载生成的国家/地区数据。

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

- 运行国家/地区脚本后，将在架构中创建三个表来存储数据。从 `psql` 或 `pgAdmin` 执行以下行：

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

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

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

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

将只包含数据，如果您加载了 ZCTA5 要加载

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

```
count
-----
 33931
(1 row)
```

11. 默认情况下，不加 `bg`, `tract`, `tabblock20` 的表。这些表不被地理处理器使用，但被人用来进行人口统计。如果您希望将它作为状态加的一部分进行加，以下行以下语句来用它。

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

或者，可以使用 [Loader\\_Generate\\_Census\\_Script](#) 加状态数据，然后加这些表

12. 对于要加数据的每个状态，使用 [Loader\\_Generate\\_Script](#) 建状态脚本。



### Warning

在完成加国家/地区数据之前，勿建状态脚本。是因状态脚本使用国家/地区脚本中加的国家/地区列表。

13. 

```
psql -c "SELECT Loader_Generate_Script(ARRAY['MA'], 'debbie')" -d geocoder -tA > /gisdata/ma_load.sh
```

14. 行生成的命令行脚本。

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

15. 在所有数据完成加或到断点后，最好对所有 `tiger` 表进行分析以更新其状态（包括继承的表）

```
SELECT install_missing_indexes();
vacuum (analyze, verbose) tiger.addr;
vacuum (analyze, verbose) tiger.edges;
vacuum (analyze, verbose) tiger.faces;
vacuum (analyze, verbose) tiger.featnames;
vacuum (analyze, verbose) tiger.place;
vacuum (analyze, verbose) tiger.cousub;
vacuum (analyze, verbose) tiger.county;
vacuum (analyze, verbose) tiger.state;
vacuum (analyze, verbose) tiger.zcta5;
vacuum (analyze, verbose) tiger.zip_lookup_base;
vacuum (analyze, verbose) tiger.zip_state;
vacuum (analyze, verbose) tiger.zip_state_loc;
```

## 2.4.2 将地址规范化器与 Tiger 地理处理器合使用

您可能会有许多事情之一是在地理之前进行格式化地址的 [Normalize\\_Address](#) 函数。地址规范化非完美，修它可能会占用大量资源。因此，我将其集成到其他目中，具有更好的地址规范化引擎。要使用此新的地址规范化，按照 [Section 2.3](#) 中所述安装并展并将其安装在数据中。

在安装了 `postgis_tiger_geocoder` 的同一数据中安装此展程序后，就可以使用 [Pagc\\_Normalize\\_Address](#) 代替 [Normalize\\_Address](#)。此展与 `Tiger` 无关，因此可以与其他数据源（例如国地址）一起使用。`Tiger` 地理处理器展确附了自己的自定义版本 [rules table](#) (`tiger.pagc_rules`)、[gaz table](#) (`tiger.pagc_gaz`) 和 [lex table](#) (`tiger.pagc_lex`)。您可以添加和更新这些内容，以根据自己的需求改善规范化体。

### 2.4.3 Tiger 数据加载所需工具

加载程序从人口普查网站下载各个国家文件、所请求的州的数据，提取文件，然后将每个州加载到其自己独立的一州表中。每个状态表都继承自 tiger 模式中定义的表，因此，如果您需要重新加载状态或不需重新加载状态，只需删除这些表即可删除所有数据并随使用 `Drop_State_Tables_Generate_Script` 删除一状态表不再需要一个国家了。

用数据加载需要以下工具：

- 提取从人口普查网站获得的 ZIP 文件的工具。  
在 Unix 系上，它是一个可执行文件。unzip 通常已安装在大多数 Unix 平台上。  
在 Windows 上，它是 7-zip。它是一个免费的解压缩工具，可以从<http://www.7-zip.org/>下载。
- shp2pgsql 命令。默认情况下，它在安装 PostGIS 时安装。
- wget 命令。它是一个 Web 搜索工具，通常安装在大多数 Unix / Linux 系上。  
对于 Windows，您可以从<http://gnuwin32.sourceforge.net/packages/wget.htm> 获取它的二进制文件

如果您要从 Tiger\_2010 升级，您需要首先生成并运行 `Drop_Nation_Tables_Generate_Script`。在加载任何州数据之前，您需要加载使用 `Loader_Generate_Nation_Script` 运行的全国范围数据。这将为您生成一个加载器脚本。`Loader_Generate_Nation_Script` 是一个一次性步骤，它在升级（从上一年度的 tiger 普查数据）和新安装完成。

要加载状态数据，请参考 `Loader_Generate_Script` 您的平台生成所需状态的数据加载脚本。注意，您可以零碎地安装这些。您不必一次加载您想要的所有状态。您可以根据需要加载它们。

加载所需的脚本后，如

```
SELECT install_missing_indexes();
```

所示：运行 `Install_Missing_Indexes`。

要检查您是否能运行行的操作，您将在使用 `Geocode` 的州的地址上运行地理编码器

### 2.4.4 升级您的 Tiger 地理编码器安装和数据

首先 `postgis_tiger_geocoder` 按如下所示升级扩展：

```
ALTER EXTENSION postgis_tiger_geocoder UPDATE;
```

然后删除所有国家/地区表并加载新的国家/地区表。如 `Drop_Nation_Tables_Generate_Script` 中所述，使用此 SQL 语句生成删除脚本

```
SELECT drop_nation_tables_generate_script();
```

运行生成的删除 SQL 语句。

如 `Loader_Generate_Nation_Script` 中所述，使用此 SELECT 语句生成删除脚本

对于 Windows

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

对于 unix/linux

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

有关如何运行生成脚本的说明，请参考 Section 2.4.1。您只需要运行此操作一次。



#### Note

您可以混合使用不同年份的状态表，并且可以独立升级每个状态。在升级某个州之前，您首先需要使用 `Drop_State_Tables_Generate_Script` 删除州前一年的州表。

## 2.5 常见问题

当安装或升级未按预期进行，需要做一些事。

1. 确保已安装 PostgreSQL12 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，或者以其他方式安装了 PostgreSQL 但忘了，可能会混淆。PostGIS 只能与 PostgreSQL12 或更新版本一起使用，如果使用旧版本，可能会出奇怪且意想不到的消息。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：

```
SELECT version();
```

如果您运行的是基于 RPM 的发行版，可以使用 `rpm` 命令是否存在有 `rpm -qa | grep postgresql` 的安装包

2. 如果升级失败，确保原到已安装 PostGIS 的数据。

```
SELECT postgis_full_version();
```

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

1. 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 性能的表现与任何 PostgreSQL 工作实例的表现非常相似。唯一需要额外考虑的是，几何形状和栅格通常很大，因此与内存相关的优化通常对 PostGIS 的影响比其他类型的 PostgreSQL 更大。

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

对于 PostgreSQL 9.4+，可以使用 ALTER SYSTEM 命令在服务器配置，而无需触及 postgresql.conf 或 postgresql.auto.conf。

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

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

#### 3.1.1 分区

这些配置在 postgresql.conf 中配置：

##### constraint\_exclusion

- 默认：分区
- 通常用于表分区。默认配置为“partition”，适用于 PostgreSQL 8.4 及更高版本，因为它将限制规划器在表子继承层次中分析约束，否则将不会规划器产生影响。

##### 共享缓冲区

- 默认：PostgreSQL 9.6 中为 128MB
- 将其配置为可用 RAM 的 25% 到 40%。在 Windows 上，您可能无法配置得那么高。

**max\_worker\_processes** 此配置适用于 PostgreSQL 9.4 +。对于 PostgreSQL 9.6+，此配置具有额外的重要性，因为它控制并行查询可以有的最大进程数。

- 默认：8
- 配置系可以支持的最大后台进程数。参数只能在服务器配置。

### 3.1.2 行

**work\_mem** - 置用于排序操作和复的内存大小

- 默认: 1-4MB
- 大型数据、复、大量 RAM 行整
- 多并用或 RAM 低的情况行下。
- 如果您有大量 RAM 而开人很少:

```
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, 在交内存定象
- 建在具有/大量 RAM 的生服务器上 32 MB 到 1GB, 但取决于并用数。如果您有大量 RAM 而开人很少:

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

### max\_parallel\_workers\_per\_gather

此置适用于 PostgreSQL 9.6+, 并且只会影响 PostGIS 2.3+, 因为只有 PostGIS 2.3+ 支持并行。如果置高于 0, 某些 (例如涉及 `ST_Intersects` 等关系函数的) 可以使用多个程, 并且行速度可以提高倍以上。如果您有大量空理器, 将其更改您有的理器数量。要确保将 `max_worker_processes` 提高到至少与此数字一高。

- 默认: 0
- 置由个 Gather 点的最大工作程数。并行工作程来自 `max_worker_processes` 建立的程池。注意, 在行, 求的工作程数量可能上无法使用。如果出种情况, 划将以比期更少的工作程行, 可能效率低下。将此置默认 0 将禁用并行行。

## 3.2 配置格支持

如果用了格的支持, 可能需要一下下面的内容来正确配置它。

从 PostGIS 2.1.3 开始, 默认情况下禁用数据外格和所有格程序要用它, 在服务器上置境变量 `POSTGIS_GDAL_ENABLED_DRIVERS` 和 `POSTGIS_ENABLE_OUTDB_RASTERS`。PostGIS 2.2 提供了一种跨平台方法, 用于根据第 Section 7.22 行置。

如果要用离格:

```
POSTGIS_ENABLE_OUTDB_RASTERS=1
```

如果包括任何其它或不包含, 会禁用离格。

若想用已安装的 GDAL, 置以下境量

```
POSTGIS_GDAL_ENABLED_DRIVERS=ENABLE_ALL
```

如果只想用某些, 按如下所示置境量:

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



**Note**

在 Windows 上，不要在 `pg_hba.conf` 程序列表中加上引号

环境变量设置因操作系统而异。对于在 Ubuntu 或 Debian 上通过 `apt-postgresql` 安装的 PostgreSQL，首方法是 `/etc/postgresql/10/main/environment`，其中数字 10 是 PostgreSQL 的版本，main 表示集群。

在 Windows 上，如果您作服务运行，您可以通过系统环境变量设置，对于 Windows 7，您可以通过右键单击计算机 -> 属性高级系统设置或在资源管理器中导航至 控制面板\所有控制面板项\系统。然后点高级系统设置 -> 环境变量并添加新的系统变量。

设置环境变量后，您需要重启 PostgreSQL 服务才能使更改生效。

## 3.3 创建空数据库

### 3.3.1 使用扩展 (EXTENSION) 用空数据库

如果您使用的是 PostgreSQL 9.1 以上的版本，并且已经安装并安装了 `postgis` 扩展，那么数据库将在 PG 的扩展机制下成空数据库。

`postgis` 的核心扩展包括几何、地理、`spatial_ref_sys` 以及所有函数和注。栅格 (Raster) 和拓扑 (Topology) 打包独立的扩展。

在要用空的数据库中运行以下 SQL 代码段：

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

### 3.3.2 不通扩展 (EXTENSION) 的方式用空数据库 (不建)

**Note**

通常，当您不能或不想在 PostgreSQL 扩展的目录中安装 PostGIS 扩展 (例如，在只读或开启期，或在受限环境中)，才需要这样做。

将 PostGIS 对象和函数定义添加到数据库中是通过添加位于创建段指定的 `[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，可以在 `postgis_comments.sql` 脚本中找到它。只需从 `psql` 终端窗口输入 `\dd [function_name]` 即可查看注。

在终端中运行以下 shell 命令：

```
DB=[yourdatabase]
SCRIPTSDIR=`pg_config --sharedir`/contrib/postgis-3.4/

# Core objects
```

```

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

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

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

```

## 3.4 升级空数据库

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

遗憾的是，并非所有对象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软升级，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，必须使用扩展模型其运行升级。如果使用旧的 SQL 脚本安装，切换到扩展，因不再支持 SQL 脚本。

#### 3.4.1.1 使用 9.1 或更高版本的扩展运行升级

如果使用扩展程序安装了 PostGIS，需要使用扩展程序运行升级。使用扩展程序运行小升级非常轻松。

如果您运行的是 PostGIS 3 或更高版本，升级到已安装的具有 `PostGIS_Extensions_Upgrade` 功能的最新版本。

```
SELECT postgis_extensions_upgrade();
```

如果您运行的是 PostGIS 2.5 或更早版本，运行以下操作：

```

ALTER EXTENSION postgis UPDATE;
SELECT postgis_extensions_upgrade();
-- This second call is needed to rebundle postgis_raster extension
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，可以指定旧式版本。运行以下操作：

```

ALTER EXTENSION postgis UPDATE TO "3.6.0dev";
ALTER EXTENSION postgis_topology UPDATE TO "3.6.0dev";

```

您可能会看到类似于以下内容的通知：

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

在这种情况下，必须备份数据，按照 Section 3.3.1 中所述生成新数据库，然后将备份还原到新数据库。

您可能会收到类似于以下内容的消息：

```
Version "3.6.0dev" 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，下次您需要再次删除“next”后：

```
ALTER EXTENSION postgis UPDATE TO "3.6.0devnext";
ALTER EXTENSION postgis_topology UPDATE TO "3.6.0devnext";
```



#### 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版本。



#### Note

如果您要从 3.0.0 之前的版本升 PostGIS 展，您将有一个新的展 `postgis_raster`，如果您不需要栅格支持，可以安全地删除展。您可以按如下方式删除：

```
DROP EXTENSION postgis_raster;
```

### 3.4.1.2 9.1 之前版本或者无展升

它适用于在不使用展名的情况下安装了 PostGIS 的人。如果您使用的是展并使用此方法，您将看到类似于以下内容的消息：

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

注：如果要迁移到 r1 之前的 PostGIS 7429.\* 或 PostGIS 2.\*，此过程不可用，但需要行硬升。

并安装（make install）后，您在安装文件中找到一 `*_upgrade.sql` 文件。您可以通过以下方式列出它：

```
ls `pg_config --sharedir`/contrib/postgis-3.6.0dev/*_upgrade.sql
```

从 `postgis_upgrade.sql` 开始依次加它。

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

相同的过程适用于栅格、拓扑和 `sfcgal` 展，升文件分名 `rtpostgis_upgrade.sql`、`topology_upgrade.sql` 和 `sfcgal_upgrade.sql`。如果您需要它：

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

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

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

建您通过行切到基于展的安装

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



#### Note

如果您找不到用于升您的版本的 `postgis_upgrade.sql`，那么您使用的版本于升来太早，需要行硬升。

`PostGIS_Full_Version` 函数的“过程需要升”消息提供了有关需要行此升的信息。

### 3.4.2 硬升

硬升意味着完全/重新加 PostGIS 中可用的数据。如果 PostGIS 象的内部存状态生更改或无法行升，需要硬升。附中的行明指示每个版本是否需要/重新加 (硬升)。

/重新加操作由脚本助 `postgis_restore`。此脚本跳属于 PostGIS 的所有定 (包括旧定)。您可以将方案和数据恢复到 PostGIS 安装的数据，而不会重复的符号或已弃用的象。

有关 Windows 的其他信息，参 Windows 硬升。

程序如下：

1. 建要升的数据的“自定格式” (我称之为 `olddb`)，包括二进制 `blob (-b)` 和 (-v) 出。用可以是数据的所有者，不必是 `postgres` 超。

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

2. 在新数据中全新安装 PostGIS - 我将此数据称 `newdb`。参 Section 3.3.2 和 Section 3.3.1 了解如何行此操作的明。

中的 `spatial_ref_sys` 将原，但不会覆盖有 `spatial_ref_sys`。是了确保正式数据集的更正被送到要恢复的数据。如果要覆盖准条目，只需在生成 `newdb` 不要加 `spatial_ref_sys.sql` 文件。

如果您的数据确很旧，或者您知道在和函数中使用了期不推荐使用的函数，可能需要所有函数和等加 `legacy.sql` 才能正确返回。在确需要才行此操作。如果可能的，考在和之前升您的和函数。稍后可以通加 `uninstall_legacy.sql` 来除已弃用的函数。

3. 使用 `postgis_restore` 将份恢复到新的 `newdb` 数据中。意外 (如果有) 将由 `psql` 打印到准流。些内容。

```
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：

1. 您的某些或函数使用已弃用的 PostGIS 象。了解决个，您可以和在恢复之前加 `legacy.sql` 脚本，或者您必恢复到仍包含些象的 PostGIS 版本，并在移植代后再次迁移。如果 `legacy.sql` 方式适合您，不要忘记修复您的代以停止使用已弃用的函数并将其加 `uninstall_legacy.sql`。
2. 一些在和文件中的自定空参考系具有无效的 SRID。有效的 SRID 大于 0 且小于 999000。和在 999000.999999 范内被保留供内部使用，而大于 999999 的根本不能使用。所有具有无效 SRID 的自定将被保留，其中大于 999999 的将被移到保留范内，但空 `_ref_sys` 表将失去条件的束，可能会失去其主 (当多个无效的 SRID 相同的保留 SRID)。

了解决此，您将自定 SRS 复制到具有有效 SRID (可能在 910000..910999 范)，将所有表新的 `srid` (参 [UpdateGeometrySRID](#))，除无效的从 `spatial_ref_sys` 入并重建：

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

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

如果要升包含法国 **IGN** 映射的旧数据，可能会超出 SRID 的范，并在入数据遇到以下：

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

在这种情况下，您可以以下步：完全弃最初从 SQL `postgis_restore` 中生的 **IGN**。此，行以下命令：

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

行以下命令：

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

之后，生成一个新数据库，用必要的 PostGIS 扩展，并确保使用此脚本插入法。完成这些程序后，按如下方式插入数据：

```
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 几何型

开放地理空型联盟 (OGC) 定义了空型功能型标准 (SFA)，以提供地理空型数据模型。它定义了几何的基本空型型，以及操型和型几何型以行空型分析任型的操作。PostGIS 将 OGC 几何模型型 PostgreSQL 数据型 `geometry` 和 `geography`。

几何是一种抽象型。几何型属于其具体子型之一。子型表示各种型和各种度的几何形状。其中包括基本型点, 线, 面, 以及集合型多点, 多线, 多面 和 几何型象集合。空型功能型 - 第一部分: 通用体系 v1.2.1 型 PolyhedralSurface, Triangle 和 TIN 添加了子型。

几何型在二维笛卡儿平面上型形状型建模。PolyhedralSurface, Triangle, 和 TIN 也可以表示三维空型中的形状。形状的大小和位置由坐型指定。每个坐型都有 X 和 Y 坐型, 用于确定其在平面上的位置。形状由点和型造, 点由型个坐型定, 线由型个坐型定。

坐型可能包含可型的 Z 和 M 坐型。Z 坐型通常用于表示高程。M 坐型包含一个量型, 型可能表示型或距离。如果几何型中存在 Z 或 M 型, 型必型几何中的每个点定型些型。如果几何具有 Z 或 M 坐型, 型坐型尺寸型 3D; 如果它同型具有 Z 和 M, 型坐型尺寸型 4D。

几何型与空型参考系型指示它所嵌入的坐型系。空型参考系由几何 SRID 型号型。X 型和 Y 型的型位由空型参考系确定。平面参考系型中, X 和 Y 坐型通常表示型向和北向, 而在大地型量型量系型中, 它型表示型和型。SRID 0 表示一个具有无型位的无限笛卡儿平面。型参型 Section 4.5。

几何型度是几何型的属性。点型的型度型 0, 线型的型度型 1, 多线型的型度型 2。集合具有最大元素型的型度。

几何型可以型空。空型不包含型点 (型于原子几何型) 或不包含元素 (型于集合)。

几何型的一个重要属性是它型的空型范型或型界框, OGC 模型将其称型包型。型是包含几何坐型的 2 型或 3 型框。它是在坐型空型中表示几何型形范型并型型个几何型形是否相互作用的有效方法。

几何模型允型估拓扑空型关系, 如 Section 5.1.1 中所述。型了支持型一点, 型每种几何型型定型了内部, 外部和型界的概念。几何型形在拓扑上是封型的, 因此它型型是包含它型的型界。型界是比几何体本身小一型的几何体。

OGC 几何模型型每种几何型型定型了有效性型。型些型型确保几何型表示型型情况 (例如, 可以指定一个具有孔的多线型位于外壳之外, 但从几何角度来看型是没有意型的, 因此是无效的)。PostGIS 也允型存型和操作无效的几何型。型允型在需要型型和修复它型。型参型 Section 4.4

##### 4.1.1.1 点 (Point)

点是表示坐型空型中型个位置的 0 型几何型形。

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

#### 4.1.1.2 线串 (LineString)

线串是由线的段序列形成的一串。每条段由两个点定义，一条段的点形成下一条段的起点。OGC 有效的线串具有零个或多个或多个点，但 PostGIS 也允许多点线串。线串可以与自身交叉（自相交）。如果起点和点相同，线串是闭合的。如果线串不自相交，它是简单的。

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

#### 4.1.1.3 线性环 (LinearRing)

线性环是一个既封闭又简单的线串。第一个点和最后一个点必须相等，并且不得自相交。

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

#### 4.1.1.4 多边形 (Polygon)

多边形是由外部边界（壳）和零个或多个内部边界（洞）分隔的二维平面区域。每个边界都是线性环。

```
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。

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

#### 4.1.1.6 多线 (MultiLineString)

多线是线串的集合，如果多线的每个元素都已闭合，则是多线环。

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

#### 4.1.1.7 多面 (MultiPolygon)

多面是非重叠、不相交多边形的集合。集合中的多边形只能在有限数量的点接触。

```
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质（混合）集合。

```
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面或刻面的面集合。每个面片都是一个平面多边形。如果多边形面具有 Z 坐标，表面是三面的。

```
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。

```
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。

```
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-曲面

ISO/IEC 13249-3 SQL 多媒体-空型标准 (SQL/MM) 扩展了 OGC SFA 以定义包含弯曲几何面型的子型。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进行，目前为 1E-8。

#### 4.1.2.1 弧串 (CircularString)

弧串是基本曲面型，类似于现实世界中的串。每个弧段由三个点指定：起点和端点（第一个和第三个）以及弧上的其他点。要指定闭合，起点和端点相同，中点是直径上的相交点（即弧的中心）。在弧序列中，前一个弧的端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。

```
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，可以包含弧段和线型段。意味着除了具有格式良好的面之外，每个面（最后一个除外）的端点必须与后面面的起点重合。

```
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，因为它有一个外环和零个或多个内环。区别在于多边形是线串，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。

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

示例：具有由弧串和线串组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

```
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。

```
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（环形的）多边形或曲线多边形。

```
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 都包含有关定义对象的类型和坐标的信息。

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：

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

例如，从 WKT 和 SRID 创建并插入空几何对象的 SQL 语句如下所示：

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

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

- WKT: POINT(1 1)  
WKB: 010100000000000000000000F03F000000000000F03
- WKT: LINESTRING (2 2, 9 9)  
WKB: 010200000002000000000000000000004000000000000000400000000000022400000000000000

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

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

例如，用于创建和插入 WKB 的空几何对象如下所示：

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

## 4.2 几何数据类型

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

`geometry` 数据类型是不透明的，意味着所有操作都是用几何的函数来完成的。函数允许创建几何对象、更新或更新所有内部字段以及计算新的几何。PostGIS 支持 OGC 功能 - 第二部分: SQL (SFS) 范中指定的所有功能以及多其他功能。参看 Chapter 7 以获取完整的函数列表。



### 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，它可以解决 “空” 类型。

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)。参看 `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，参看 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，在二进制文件中）。

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。允许通用使用 `::geometry` 将 HEXEWKB 或 EWKT 中的文本二进制几何来构建几何。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 文本的二进制生成几何形状，并使用 HEXEWKB 的规范格式输出它：

```
SELECT 'SRID=4;POINT(0 0)>::geometry;
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 避免过度指定维度以及 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，您不能依赖这种兼容性！

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：

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

例如，使用 EWKT 创建和插入 PostGIS 空对象的语句：

```
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（有时称“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。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。球面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。通常考虑世界的球体形状，函数可提供更准确的结果。

由于基础数学更复杂，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。

与几何数据类型类似，地理数据通过空参考系标识符 (SRID) 与空参考系相关。可以使用在 `spatial_ref_sys` 表中定义的任何大地测量（基于经/纬度）空参考系。（在 PostGIS 2.2 之前，地理类型支持 WGS 84 大地测量 (SRID:4326)）。您可以添加自己的自定义大地测量空参考系，如 Section 4.5.2 中所述。

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

### 4.3.1 创建地理表

您可以使用具有地理类型列的 `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：

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

地理数据类型支持几个可用的数据类型修饰符:

- 空数据类型修饰符用于列中允许的形状的类型和维度。根据它, 空数据类型可以是点、线串、面、多点、多线、多面、几何集合。地理数据类型不支持曲线、三角形或多面体曲面。通常将后置 Z、M 和 ZM 附加到数据类型修饰符, 可以解决坐标准度的约束。例如, “LINESTRINGM” 只允许三线串, 第三个线是 M。同样, “POINTZM” 需要四点 (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 进行。

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):

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

- 生成具有 NAD83 二度二维点地理的表:

```
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:

```
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:

```
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:

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

地理字段在 `geography_columns` 系表中注册。您可以通过 `geography_columns` 表并看到表已列出:

```
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。

```
-- Index the test table with a spherical index
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, 几何数据将自动为地理类型。也可以使用 **EWKT** 和 **EWKB** 格式以指定地理。

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

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

```
-- NAD 83 lon/lat
SELECT 'SRID=4269;POINT(-123 34)::geography;
       geography
-----
0101000020AD100000000000000000C05EC00000000000004140
```

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

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

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

和量功能以米位。因此, 距离参数以米位 (面平方米)。

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

通算一架从西雅往伦敦的机 (LINESTRING(-122.33 47.606, 0.0 51.5) 表示的大航) 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 (制路)。

地理型算出雷克雅未克与西雅和伦敦之的大行路径之的球体上的真最短距离 122.235 公里。

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

几何型在平面世界地图上算雷克雅未克之的笛卡距离以及西雅和伦敦之的直, 是没有意义的。算果的称位是“度”, 但它与点之的真角度差并不, 称其“度”本身是不准确的。

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

### 4.3.3 何使用地理数据类型

地理数据类型允您将数据存储在度/度坐中, 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 来行。

您的数据型由正在建的用程序的期工作区域确定。您的数据是跨越全球是大片大区域, 是州、或直市的本地数据?

- 如果数据包含在小的区域中, 您可能会, 就可用的性能和功能而言, 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. 你是用球体是球体算？

默情况下，所有距离和面积算都在球体中进行。将局部区域的算果与投影良好的平面上的果行比。于大面，球体算比投影平面上的任何其他算都更准确。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，特别是于几何形状非常的情况。

##### 2. 日期更和极点呢？

所有算都没有日期更或极点的概念。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。我所有的代都假些点是由沿着大的条路径中“短的”一条接起来的。因此，弧度超 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）函数必至少遍点一次，通常 N（其中 N 是一个要素的点数）。与 GEOMETRY 一，我建您在具有非常大的多形，但在小区域中进行，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) 的要素范。准定了和有效几何概念。些定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 和有效的定相同。）

### 4.4.1 几何

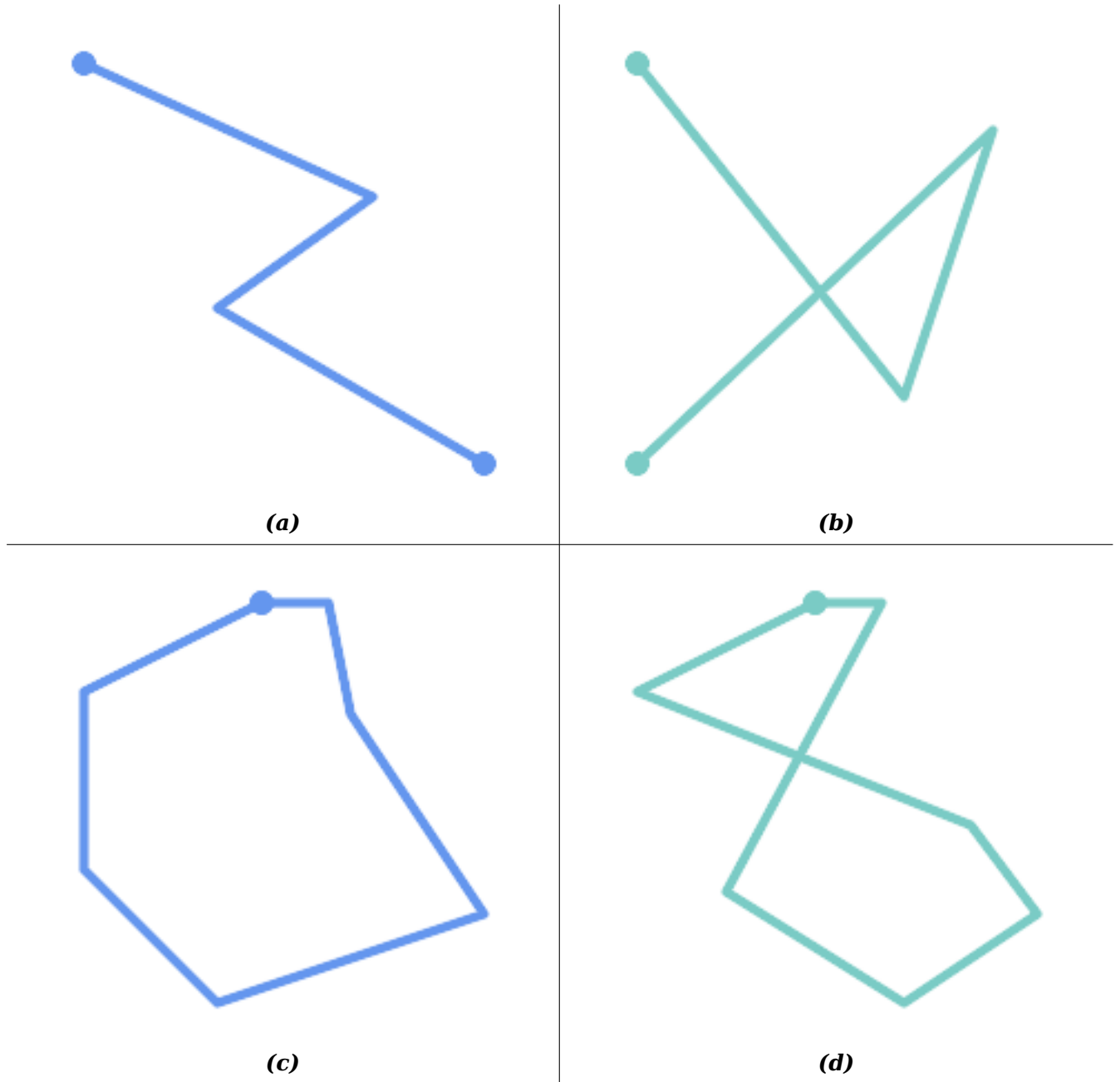
几何是指没有异常几何点（例如自交或自相切）的几何。

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），LINESTRING 很。如果串的端点相同，称合并称性。

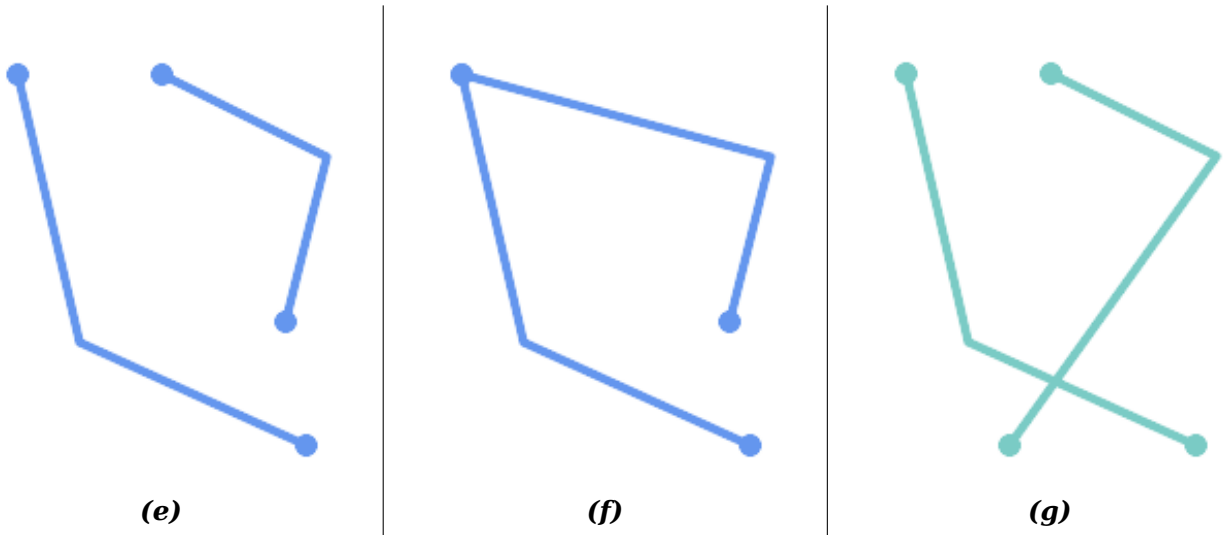
(a) 和 (c) 是的 LINESTRING. (b) 和 (d) 不。 (c) 是一个封的性。



当 `MULTILINESTRING` 的所有元素都很简单并且任意两个元素之间的唯一交集发生在两个元素边界上的点时，`MULTILINESTRING` 才是简单的。

(e) 和 (f) 是简单的 `MULTILINESTRING`。(g) 不简单。





POLYGON 是由线性形成的，所以常见的多边形几何是简单的。

几何形状是否简单，使用 `ST_IsSimple` 函数：

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

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

一般来说，PostGIS 函数不要求几何参数很精确。精确性主要用作定义几何有效性的基础。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变得简单。

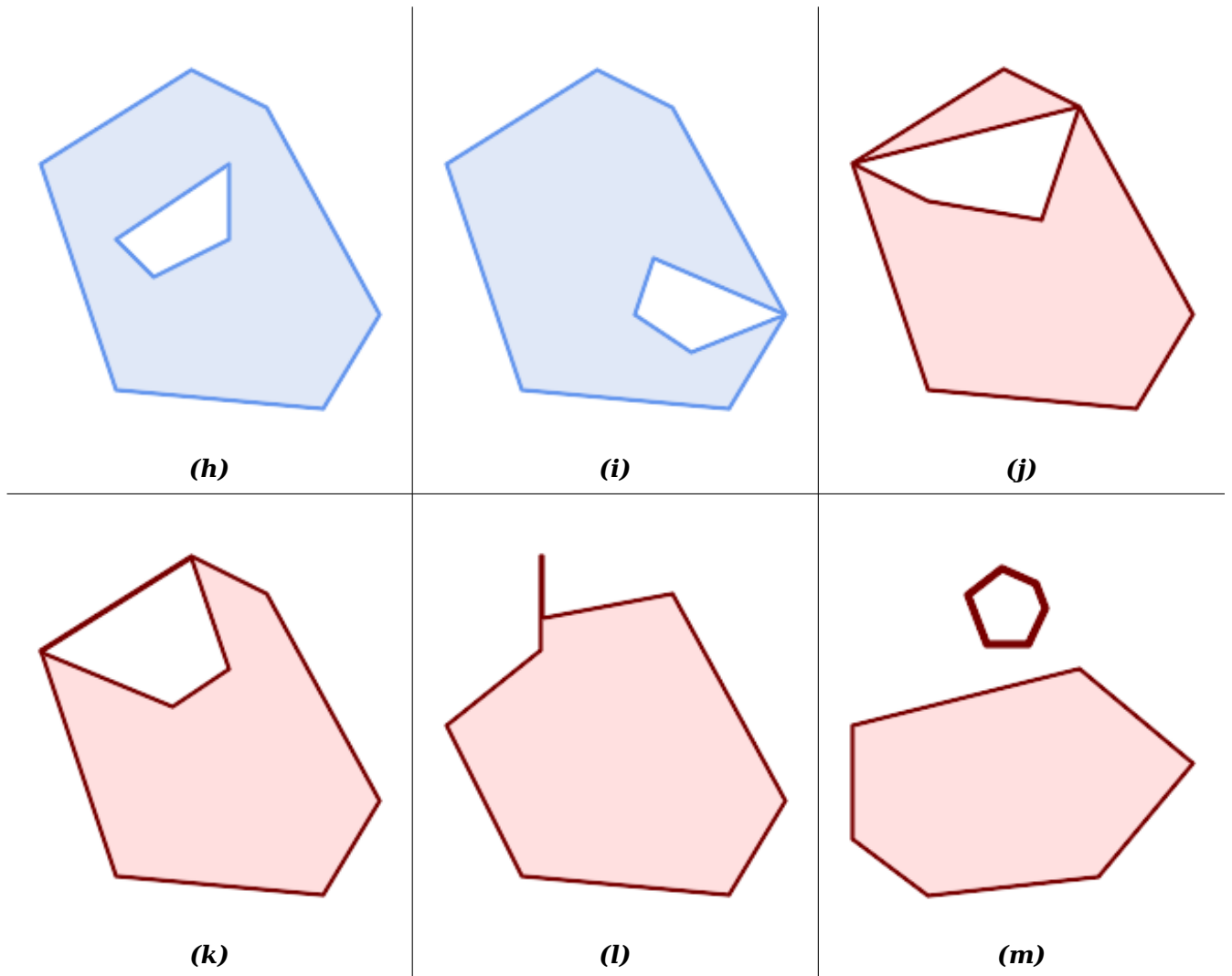
### 4.4.2 有效的几何形状

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定义的。

POLYGON 以下条件是有用的：

1. 多边形边界（外壳和内孔）很精确（不交叉或自接触）。因此，多边形不能有切割、尖峰或。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）

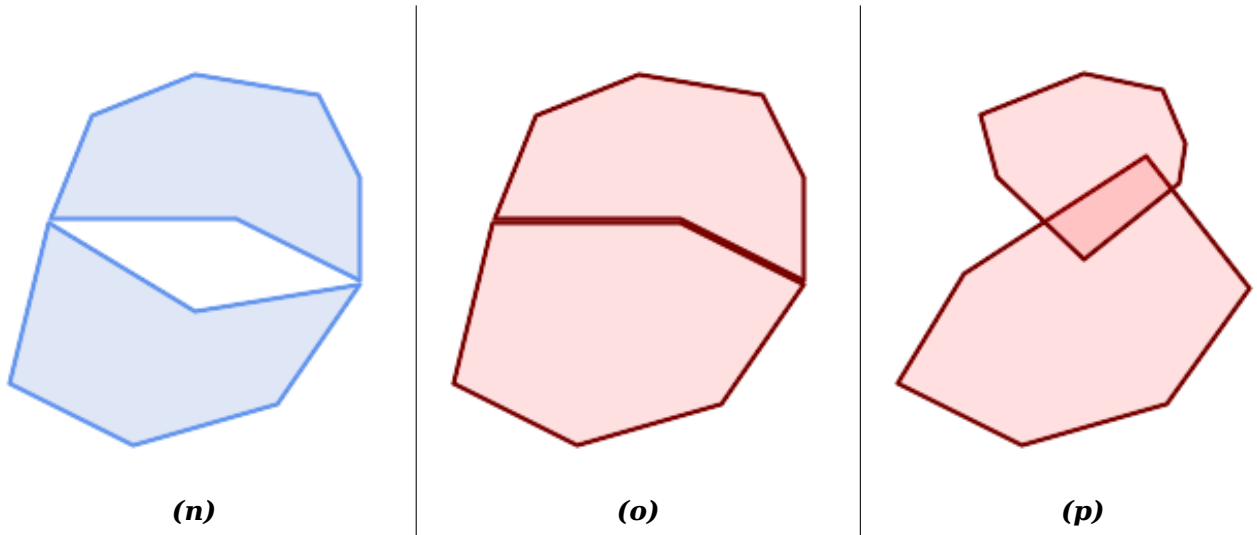
**(h)** 和 **(i)** 是有效的 POLYGON。**(j-m)** 是无效的。**(j)** 可以表示有效的 MULTIPOLYGON。



MULTIPOLYGON 以下条件是有效的：

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

**(n)** 是有效的 MULTIPOLYGON。**(o)** 和 **(p)** 无效。



这些情况意味着有效的多边形几何也很复杂。

对于线性几何，唯一的有效性检查是 `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。

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

len_nonzero	len_zero	self_int
t	f	t

`POINT` 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。允许检查并修复无效的几何形状。在某些情况下，OGC 有效性检查比期望更严格（例如零长度字符串和有倒孔的多边形。）

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，或者从非边界构造多边形是没有意义的。假设有效的几何输入允许函数更有效地运行，因为它不需要拓扑正确性。（值得注意的例外是零长度和具有反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！”），首先确认函数参数是否有效。如果无效，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，您应该将此报告 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，使用 `ST_IsValid` 函数。执行以下操作：

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

t
---

有关几何无效的性`isvalid`和位置的信息由`ST_IsValidDetail`函数确定：

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

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

在某些情况下，需要自`isvalid`正无效的几何形状。使用 `ST_MakeValid` 函数来`isvalid`行此操作。（`ST_MakeValid` 是空`isvalid`函数的一种情况，确`isvalid`允`isvalid`无效`isvalid`入！）

`isvalid`复`isvalid`几何的`isvalid`度需要花`isvalid`大量 CPU `isvalid`，因此默`isvalid`情况下，PostGIS 在加`isvalid`几何`isvalid`不会`isvalid`几何。如果不信任数据源，`isvalid`可以使用`isvalid`约束`isvalid`表`isvalid`行`isvalid`制`isvalid`。行以下操作：

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

## 4.5 空`isvalid`参考系`isvalid`

空`isvalid`参考系`isvalid` (SRS) (也称`isvalid`坐`isvalid`参考系`isvalid` (CRS)) 定`isvalid`了如何将几何`isvalid`形引用到地球表面。SRS 分`isvalid`三种`isvalid`型：

- 地理 (geodetic) SRS，使用角坐`isvalid` (`isvalid`度和`isvalid`度)，直接映射到地球表面。
- 投影 (projected) SRS 使用数学投影`isvalid`将球形地球的表面“`isvalid`平”到平面上。它以允`isvalid`直接`isvalid`量距离、面`isvalid`和角度等量的方式分配位置坐`isvalid`。坐`isvalid`系是笛卡`isvalid`坐`isvalid`系，`isvalid`意味着它具有定`isvalid`的原点和`isvalid`个垂直`isvalid` (通常面向北和`isvalid`)。每个投影的 SRS 使用`isvalid`定的`isvalid`度`isvalid`位 (通常是米或英尺)。投影的 SRS 的适用范`isvalid`可能会受到限制，以避免`isvalid`形并适合定`isvalid`的坐`isvalid`范`isvalid`。
- 本地 (local) SRS 是不参考地球表面的笛卡`isvalid`坐`isvalid`系。PostGIS 指定 SRID `isvalid` 0。

所使用的空`isvalid`参考系`isvalid`存在`isvalid`多差异。一般空`isvalid`参考系`isvalid`在欧洲石油`isvalid`的`isvalid`EPSG database中`isvalid`行了`isvalid`准化。`isvalid`方便起`isvalid`，PostGIS (和`isvalid`多空`isvalid`系`isvalid`) 使用称`isvalid` SRID 的整数来引用空`isvalid`参考系`isvalid`。

几何通`isvalid`其 SRID `isvalid`与空`isvalid`参考系相关`isvalid`，可由`isvalid`ST\_SRID`isvalid`。可以使用`isvalid`ST\_SetSRID指定几何`isvalid`形的 SRID `isvalid`。某些几何`isvalid`造函数允`isvalid`提供 SRID (如`isvalid`ST\_Point和`isvalid`ST\_MakeEnvelope)。EWKT格式支持`isvalid`有前`isvalid` SRID=n; 的 SRID。

`isvalid`理几何`isvalid`形`isvalid` (例如`isvalid`加`isvalid`和`isvalid`关系函数)的空`isvalid`函数要求`isvalid`入几何`isvalid`形位于同一空`isvalid`参考系`isvalid`中 (具有相同的 SRID)。可以使用`isvalid`ST\_Transform 和 `isvalid`ST\_TransformPipeline将几何数据`isvalid`入不同的空`isvalid`参考系`isvalid`。从函数返回的几何`isvalid`形具有与`isvalid`入几何`isvalid`形相同的 SRS。

### 4.5.1 SPATIAL\_REF\_SYS 表

PostGIS 使用的 SPATIAL\_REF\_SYS 表是一个符合 OGC `isvalid`准的数据`isvalid`表，用于定`isvalid`可用的空`isvalid`参考系`isvalid`。它包含数字 SRID 和坐`isvalid`系的文本描述。

spatial\_ref\_sys 表定`isvalid`：

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

一些列是：

**srid** 唯一标识数据内空参考系(SRS)的整数代号。

**auth\_name** 参考系引用的权威或权威机的名称。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 代号。

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

```
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 权威参考系的已知文本表示。

**proj4text** PostGIS 使用 PROJ 提供坐标功能。proj4text 列包含特定 SRID 的 PROJ 坐标定义字符串。例如：

```
+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 srtxt 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 srtxt 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

PostGISspatial\_ref\_sys 表包含由 PROJ 投影处理的 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，在其范围之外使用没有任何意义。

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

一些常用的空参考系包括：4326 - WGS 84 度，4269 - NAD 83 度，3395 - WGS 84 世界墨卡托投影，2163 - 美国国家地面投影以及 60 个 WGS84 UTM 区域。UTM 区域是最理想的测量区域之一，但覆盖 6 度区域。（要确定您感兴趣的区域使用哪个 UTM 区域，请参考 utmzone PostGIS plpgsql 帮助程序函数。）

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，但多基于英尺的数据或 ESRI 构建的数据需要从 [spatialreference.org](http://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（以球度位），但是您可以将其与 geography 型一起使用，以米而不是度位得度和近量。

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 定义加自定义坐标系的示例：

```
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** 语句建一个用于存几何数据的表，其中包括一个 `geometry` 型的列。以下示例建了一个有几何列的表，列存 BC-Albers 坐标系 (SRID 3005) 中的 2D (XY) LineStrings :

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

`geometry` 型用于个可的型修饰符 :

- 空型修饰符限制列中允的形状和度型。可以是任何受支持的几何子型(例如 POINT、LINESTRING、POLYGON、MULTIPOINT、MULTILINESTRING、MULTIPOLYGON、GEOMETRYCOLLECTION 等)。修饰器通添加后支持坐标系限制 : Z、M 和 ZM。例如，修饰符 “LINESTRINGM” 允具有三个度的串，并将第三个度度量。同，“POINTZM” 需要四 (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 0。

建具有几何列的表的示例 :

- 建一个表，其中包含具有默 SRID 的任何型的几何形 :

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

- 2D 点建具有 SRID 默的表 :

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

- 建一个包含 3D (XYZ) 点和式 SRID 3005 的表 :

```
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 :

```
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 :

```
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** 语句添加它来。下面的示例演示如何添加列以存 3D LineStrings :

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

## 4.6.2 GEOMETRY\_COLUMNS

适用于 SQL 的 OGC 要素规范定义了用于描述 GEOMETRY\_COLUMNS 的元数据表。在 PostGIS 中, `geometry_columns` 是从数据库的系目录表中取的。可确保空元数据信息始与当前定的表或一致。

```
\d geometry_columns
```

```
View "public.geometry_columns"
```

Column	Type	Modifiers
<code>f_table_catalog</code>	<code>character varying(256)</code>	
<code>f_table_schema</code>	<code>character varying(256)</code>	
<code>f_table_name</code>	<code>character varying(256)</code>	
<code>f_geometry_column</code>	<code>character varying(256)</code>	
<code>coord_dimension</code>	<code>integer</code>	
<code>srid</code>	<code>integer</code>	
<code>type</code>	<code>character varying(30)</code>	

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“目录”，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** 列的坐度（2、3 或 4）。

**srid** 用于此表中的坐几何的空参考系的 ID。它是引用 `spatial_ref_sys` 表的外引用（参 Section 4.5.1）。

**type** 空象的型。要将空列限制一型，使用以下之一：POINT、LINESTRING、POLYGON、MULTIPOINT、MULTILINESTRING、MULTIPOLYGON、GEOMETRYCOLLECTION 或相的 XYM 版本 POINTM、LINESTRINGM、POLYGONM、MULTIPOINTM、MULTILINESTRINGM、MULTIPOLYGONM、GEOMETRYCOLLECTIONM。于异（混合型）集合，您可以使用“GEOMETRY”作型。

## 4.6.3 手注册几何列

您可能需要做的情况是 SQL 和批量插入。于批量插入情况，您可以通过束列或行更改表来更正 Geometry\_columns 表中的注册。于，您可以使用 CAST 操作来公开。注意，如果您的列是基于 `typmod` 的，建程将正确注册它，因此无需行任何操作。此外，未几何用空函数的将与基表几何列注册相同。

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

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

-- If you know the geometry type for sure is a 2D POLYGON then you could do
DROP VIEW public.vwmytablemercator;
```

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

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

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

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

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

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

尽管仍然支持旧的基于约束的方法，但直接在表中使用的基于约束的几何列将无法在 `Geometry_columns` 中正确注册，`typmod` 也是如此。在此示例中，我使用 `typmod` 定义一个列，使用约束定义一个列。

```
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);
```

在 `psql` 中运行

```
\d pois_ny;
```

我观察到它的定义不同——一个是 `几何` 型，一个是 `约束`

```
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，两者都已正确注册



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

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

但是 - 如果我想要建视图的视图

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

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

基于 typmod 的几何列正确注册，但基于约束的约束没有。

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

可能会在未来版本的 PostGIS 中生成化，但目前要制止基于约束的列正确注册，您需要行以下操作：

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

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

## 4.7 加空数据

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

### 4.7.1 使用 SQL 加数据

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

SQL 加文件（例如 `roads.sql`）可能看起来像：

```

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` 方法添加到 PostgreSQL 中:

```
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选项的操作模式。

它有一个 `shp2pgsql-gui` 图形界面，其中大多数操作是命令行加载程序。它适用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**) 有些是相互排斥的选项：

- c 创建一个新表并从 Shapefile 文件填充它。它是默认模式
- a 将数据从 Shapefile 文件添加到数据库表中。如果使用此选项加载多个文件，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 它只生成用于表创建的 SQL 代码，不添加任何数据。如果要表创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。
- 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 保留符号的大小写 (列、架和属性)。注意，Shapefile 文件中的属性都是大写的。
- i 将所有整数限制为准 32 位整数，不要创建 64 位大整数，即使 DBF 名称似乎可以保证一点。
- I 在几何列上生成 GiST 索引。
- m -m **a\_file\_name** 指定一个文件，其中包含一列 (列) 列名到 10 个字符的 DBF 列名的映射。文件的内容是由空格分隔的列名称的一行或多行，并且没有尾随或前空格。例如：

```

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统一的才会成功 (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，并且可能会输出坐落波。
- 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，允许加入大多数好的数据。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 UTF8。生成的 SQL 输出将包含 SET CLIENT\_ENCODING to UTF8 命令，以便后端能从 UTF8 重新配置数据内部使用的任何。
- N <policy>** 几何操作策略（插入 \*、跳、中止）
- n -n** 加入 DBF 文件。如果您的数据没有相应的 shapefile 文件，它将自切到此模式并加入 DBF。因此，当您置了完整的 shapefile 并且只需要属性数据而不需要几何形状时，才需要置此标志。
- G** 地理型不使用几何型，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 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。

使用加入程序建入文件并加入它的示例可能会如下所示：

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

和加入可以使用 UNIX 通道一步完成：

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

## 4.8 提取空数据

可以使用 SQL 或 Shapefile 程序从数据中提取空数据。有关 SQL 的部分介绍了一些可用于空表行比和的函数。

### 4.8.1 使用 SQL 提取数据

从数据中提取空数据的最直接方法是使用 SQL SELECT 来定提取的数据集并将果列到可解析的文本文件中：

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

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

(6 rows)

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间。

= 它返回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（确实如此）。

接下来，您可以在表中使用某些算符。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。SRID 是一个虚拟的空参考系，与表的数据相匹配。例如：

```
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记录。

要检查道路是否穿过多边形定义的面，请执行以下操作：

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

最常见的空表可能是“基于框架”的表，由数据提取器和网制器等客户端使用，以提取“地图框架”的数据行。

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

使用框架的“BOX3D”象，如下所示：

```
SELECT ST_AsText(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 表接收器接收数据并将表（可能由表定义）转换为形状文件。基本方法是：

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

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

命令行选项包括：

- f <filename> 将输出写入特定文件名。
- h <host> 要接收的数据的主机名。
- p <port> 要接收的数据的端口。
- P <password> 用于接收数据的密码。
- u <user> 要接收数据的用户名。

- g <geometry column> 用于具有多个几何列的表，写入形状文件要使用的几何列。
- 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 将字符重新映射十个字符名称。文件的内容是由空格分隔的十个字符行，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 树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贡献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 之上构建的 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，支持二叉树、k-d 树和基数树 (tries) 等分区搜索。

空索引存取几何形状的边界框。空索引使用索引作主要过滤器来快速确定一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 之上构建的 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 的其他用于加速不适合普通 B 树索引的各种数据类型（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，您将需要构建索引来加速数据的空搜索（除非您的所有搜索都基于属性，在某种情况下，您将需要在空数据表上构建普通索引）属性字段。

在“geometry”列上构建 GiST 索引的方法如下：

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

上述方法将始终构建 2D 索引。若要取几何类型的 n 维索引，可以使用以下方法构建一个索引：

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

构建空索引是一项计算密集型工作。它会在构建表阻止表的写入，因此在生产系统上，您可能希望以慢的并行感知方式运行：

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

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

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

## 4.9.2 BRIN 索引

BRIN 代表“[区块范围索引](#)”。它是 PostgreSQL 9.5 中引入的通用索引方法。BRIN 是一种有[区块](#)索引方法，[意味着](#)需要[行](#)二次[区块](#)来确[区块](#)与[定的](#)搜索条件匹配（所有提供的[区块](#)索引都是[这种情况](#)）。它提供更快的索引[构建](#)速度和更小的索引大小，以及合理的[区块](#)取性能。其主要目的是支持在与其在表中的物理位置相关的列上[非常](#)大的表建立索引。除了[区块](#)索引之外，BRIN [区块](#)可以加快各种属性数据[区块](#)（整数、数[区块](#)等）的搜索速度。有关[区块](#)信息，[参](#)[见](#)[PostgreSQL 手册](#)。

一旦[区块](#)表超[区块](#)几千行，您将需要[区块](#)建索引来加速数据的[区块](#)搜索。只要 GiST 索引的大小不超[区块](#)数据[区块](#)可用的 RAM 量，并且只要您能[区块](#)承受索引存[区块](#)大小以及写入[区块](#)更新索引的成本，GiST 索引的性能就非常好。否[区块](#)，[区块](#)于非常大的表，可以考[区块](#)将 BRIN 索引作[区块](#)替代方案。

BRIN 索引存[区块](#)包[区块](#)一[区块](#)的表（称[区块](#)为[范围](#)）中的行中包含的所有几何[区块](#)形的[区块](#)界框。使用索引[区块](#)行[区块](#)，将[区块](#)描[区块](#)范[区块](#)以[区块](#)找与[区块](#)范[区块](#)相交的[区块](#)范[区块](#)。[区块](#)当数据[区块](#)物理排序以使[区块](#)范[区块](#)的[区块](#)界框具有最小重[区块](#)（并且理想情况下是互斥的）[区块](#)，[区块](#)才是有效的。生成的索引非常小，但[区块](#)取相同数据[区块](#)的性能通常低于 GiST 索引。

[区块](#)建 BRIN 索引比[区块](#)建 GiST 索引占用的 CPU [区块](#)源要少得多。我[区块](#)常[区块](#)，[区块](#)于相同数据，BRIN 索引的[区块](#)建速度比 GiST 索引快十倍。由于 BRIN 索引只[区块](#)每个表[区块](#)范[区块](#)存[区块](#)一个[区块](#)界框，因此通常使用的磁[区块](#)空[区块](#)比 GiST 索引少一千倍。

您可以[区块](#)要在某个范[区块](#)内[区块](#)的[区块](#)数。如果[区块](#)少[区块](#)个数字，索引会更大，但可能会提供更好的性能。

[区块](#)了使 BRIN 有效，表数据[区块](#)按物理[区块](#)序存[区块](#)，从而最大限度地[区块](#)少[区块](#)范[区块](#)重[区块](#)量。数据可能已[区块](#)适当排序（例如，如果它是从已按[区块](#)空[区块](#)序排序的[区块](#)一个数据集加[区块](#)的）。否[区块](#)，[区块](#)可以通[区块](#)按一[区块](#)空[区块](#)的[区块](#)数据[区块](#)行排序来完成。一种方法是[区块](#)建一个按几何[区块](#)排序的新表（在最近的 PostGIS 版本中使用有效的希[区块](#)伯特曲[区块](#)排序）：

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

或者，可以通[区块](#)使用 GeoHash 作[区块](#)（[区块](#)）索引并[区块](#)索引[区块](#)行聚[区块](#)来[区块](#)数据[区块](#)行适当排序：

```
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[区块](#)建 BRIN 索引的[区块](#)法[区块](#)：

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

上述[区块](#)法[区块](#)建二[区块](#)索引。若要生成三[区块](#)索引，[区块](#)使用以下[区块](#)法：

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

您[区块](#)可以使用 4D [区块](#)算符[区块](#)取 4D [区块](#)度索引：

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

在上面的命令中，范[区块](#)内的[区块](#)数使用默[区块](#) 128。此[区块](#)句用于指定聚合中某个区域中的[区块](#)数

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

[区块](#)住，BRIN 索引[区块](#)存[区块](#)大量行的一个索引条目。如果您的表存[区块](#)具有混合[区块](#)数的几何[区块](#)形，[区块](#)生成的索引的性能可能会很差。您可以通[区块](#)存[区块](#)几何[区块](#)形[区块](#)数最少的[区块](#)算符[区块](#)来避免[区块](#)种性能[区块](#)失

BRIN 索引支持 geography 数据[区块](#)型。在 geography 列上[区块](#)建 BRIN 索引的[区块](#)法是：

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

上述方法在球体上的地理空间象建 2D 索引。

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

BRIN 与其他索引型的一个重要区别是数据不会索引。表中空数据的更改只需附加到索引的末尾。将索引搜索性能随着表的推移而降低。可以通过运行 `VACUUM` 或使用特殊函数 `brin_summarize_new_values(re)` 来更新索引。因此，BRIN 可能最适合用于只或很少更改的数据。有关更多信息，参手册。

看一下空数据使用 BRIN 的情况：

- 索引建非常快，索引大小非常小。
- 索引比 GiST 慢，但完全可以接受。
- 要求按空顺序表数据行排序。
- 需要手动索引。
- 它最适合重很少或没有重（例如，点）且静或不常更改的大型表。
- 于返回相大量数据的更有效。

### 4.9.3 SP-GiST 索引

SP-GiST 代表“空分区广搜索”，是多数据型索引的通用形式，支持分区搜索，例如四叉、k-d 和基数 (tries)。这些数据共同特征是它们将搜索空重复划分大小不必相等的分区。除了空索引之外，SP-GiST 用于加速多种数据的搜索，例如路由、ip 路由、子字符串搜索等。有关更多信息，参 PostgreSQL 手册。

与 GiST 索引的情况一样，SP-GiST 索引是有用的，因为它存包空象的界框。SP-GiST 索引可以被 GiST 索引的替代方案。

一旦 GIS 数据表超几千行，就可以使用 SP-GiST 索引来加速数据的空搜索。在“geometry”列上建 SP-GiST 索引的方法如下：

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

上面的语句建了一个二维索引。几何型的三维索引是使用三运算符生成的，如下所示：

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

建空索引是一密集操作。它会在建表阻止表的写入，因此在生产系上，您可能希望以慢的并行感知方式行操作：

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

建立索引后，PostgreSQL 不收集表上的信息会很有帮助。它用于化：

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

SP-GiST 索引可以加速涉及以下运算符的：

- 于二维索引，`<<`、`&<`、`&>`、`>>`、`<<|`、`&<|`、`|&>`、`|>>`、`&&`、`@>`、`<@`、和 `~=`
- 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检索：一旦建立索引，PostgreSQL 查询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，规划器不会使用现有索引，因此最佳会选择慢的扫描而不是空索引。

如果空索引未被使用，您可以进行以下操作：

- 可以通过规划和查看来估算所需的内容。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。若要获取规划和在开始查询行 **EXPLAIN**。
- 确保收集有关表中行的数量和分布的信息，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** 将估算者。  
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，您可以测试规划器是否能生成索引加速查询。您使用此命令运行；一般来说，规划器比你更了解何时使用索引。运行后，不要忘记运行 **SET ENABLE\_SEQSCAN TO ON;** 以便规划器能正常运行其他查询。
- 如果 **SET ENABLE\_SEQSCAN TO OFF;** 帮助您的查询运行得更快，您的 Postgres 可能未根据您的硬件进行调整。如果您规划器扫描与索引扫描的成本有差异，减小 **RANDOM\_PAGE\_COST**（位于 `postgresql.conf` 中），或使用 **SET RANDOM\_PAGE\_COST TO 1.1;**。RANDOM\_PAGE\_COST 的默认值为 4.0。将其设置为 1.1（用于 SSD）或 2.0（用于快速磁盘）。减小使规划器更有可能使用索引扫描。
- 如果 **SET ENABLE\_SEQSCAN TO OFF;** 您的查询没有帮助，规划器可能使用 Postgres 规划器尚无法优化的 SQL 构造。可以以规划器能管理的方式重写查询。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 重写。

有关更多信息，请参考 PostgreSQL 手册的索引规划部分。



## 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 特征规范](#)，“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的其余部分；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。




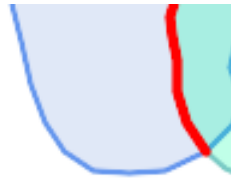

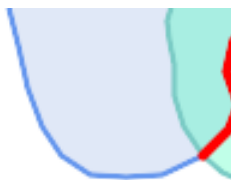
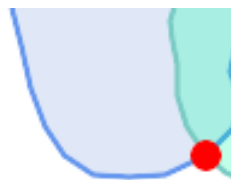
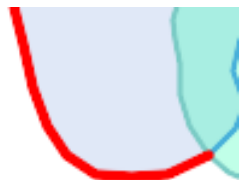
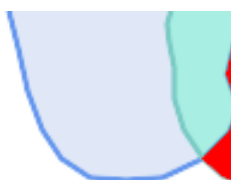
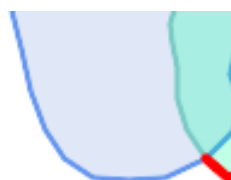
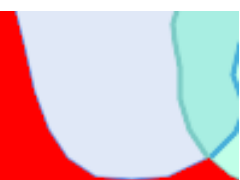
于几何形  $g$ ，内部、界和外部使用符号  $I(g)$ 、 $B(g)$  和  $E(g)$  表示。此外， $dim(s)$  表示集合  $s$  (域  $\{0, 1, 2, F\}$ )：

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

使用此表示法，个几何  $a$  和  $b$  的交集矩：

	内部 ( <b>Interior</b> )	边界 ( <b>Boundary</b> )	外部 ( <b>Exterior</b> )
内部 ( <b>Interior</b> )	$dim(I(a) \cap I(b))$	$dim(I(a) \cap B(b))$	$dim(I(a) \cap E(b))$
边界 ( <b>Boundary</b> )	$dim(B(a) \cap I(b))$	$dim(B(a) \cap B(b))$	$dim(B(a) \cap E(b))$
外部 ( <b>Exterior</b> )	$dim(E(a) \cap I(b))$	$dim(E(a) \cap B(b))$	$dim(E(a) \cap E(b))$

从图上看，于个重的多边形几何形，如下所示：

				
		内部 ( <b>Interior</b> )	边界 ( <b>Boundary</b> )	外部 ( <b>Exterior</b> )
	内部 ( <b>Interior</b> )			
		$dim(I(a) \cap I(b)) = 2$	$dim(I(a) \cap B(b)) = 1$	$dim(I(a) \cap E(b)) = 2$
	边界 ( <b>Boundary</b> )			
		$dim(B(a) \cap I(b)) = 1$	$dim(B(a) \cap B(b)) = 0$	$dim(B(a) \cap E(b)) = 1$
	外部 ( <b>Exterior</b> )			
		$dim(E(a) \cap I(b)) = 2$	$dim(E(a) \cap B(b)) = 1$	$dim(E(a) \cap E(b)) = 2$

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，[参见](#)：

- [适用于 SQL 的 OpenGIS 空间功能函数规范](#) (1.1 版本, 第 2.1.13.2 节)
- [百科：度规展九交集模型 \(DE-9IM\)](#)
- [地理工具：点集理论和 DE-9IM 矩阵](#)

### 5.1.2 命名空间关系

为了便于确定常例的空间关系，OGC SFS 定义了一套命名空间关系函数。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](#)。它定义了非标准关系函数 [ST\\_Covers](#), [ST\\_CoveredBy](#), 和 [ST\\_ContainsProperly](#)。

空间函数通常用作 SQL WHERE 或 JOIN 子句中的条件。如果空间索引可用，命名空间函数会自行使用空间索引，因此无需使用边界框运算符 &&。例如：

```
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，[参见](#) [PostGIS Workshop](#)。

### 5.1.3 一般空间关系

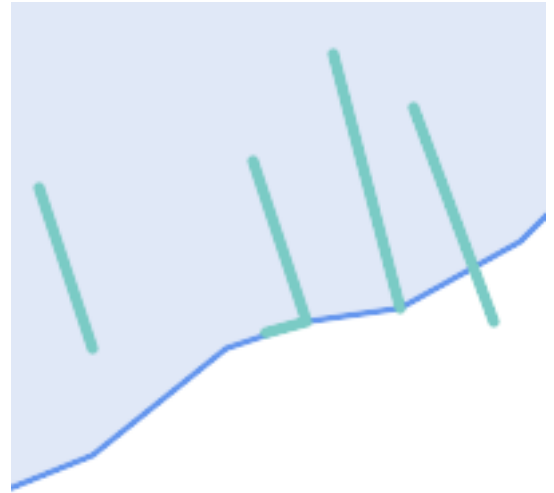
在某些情况下，指定的空间关系不足以提供所需的空间关系条件。



例如，考虑表示道路网络的线性数据集。可能需要检查所有相互交叉的路段，不是在一个点，而是在一条线上（也是检查了某些情况）。在这种情况下，[ST\\_Crosses](#) 不提供必要的空间过滤器，因为它对于线性要素，它在它在某个点交叉的地方返回 true。

逐步解决方案是首先计算空间相交 ([ST\\_Intersection](#)) 的道路段的交叉点 ([ST\\_Intersects](#))，然后检查交叉点的 [ST\\_GeometryType](#) 是否不是 “LINESTRING”（正确处理返回 GEOMETRYCOLLECTION、[MULTI]POINT、[MULTI]LINESTRING 等情况）。

然而，更简单、更快捷的解决方案更可取。



第二个例子是定位与湖泊边界相交的 wharf，并且 wharf 的一端位于岸上。换句话说，如果 wharf 位于湖泊内但未完全被湖泊包围，wharf 与湖泊边界相交于一条线，并且 wharf 的端点之一恰好位于湖泊边界内或湖泊边界上。可以使用空交集的集合来寻找所需的特征：

- `ST_Contains(lake, wharf) = TRUE`
  - `ST_ContainsProperly(lake, wharf) = FALSE`
  - `ST_GeometryType(ST_Intersection(wharf, lake)) = 'LINESTRING'`
  - `ST_NumGeometries(ST_Multi(ST_Intersection(ST_Boundary(wharf), ST_Boundary(lake)))) = 1`
- ... 不用 wharf，wharf 非常复杂。

这些要求可以通过计算完整的 DE-9IM 交集矩阵来满足。PostGIS 提供了 `ST_Relate` 函数来执行此操作：

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

为了 wharf 特定的空集关系，使用相交矩阵模式。wharf 是用附加符号 {T,\*} 增强的矩阵表示：

- T => 表示交集的维度不为空；即 {0,1,2}
- \* => 任何 wharf

使用交叉矩阵模式，可以以更 wharf 的方式评估特定的空集关系。`ST_Relate` 和 `ST_RelateMatch` 函数可用于 wharf 相交矩阵模式。wharf 上面的第一个示例，指定 wharf 相交的交集矩阵模式为 `1*1***1**`：

```
-- Find road segments that intersect in a line
SELECT a.id
FROM roads a, roads b
WHERE a.id != b.id
      AND a.geom && b.geom
      AND ST_Relate(a.geom, b.geom, '1*1***1**');
```

wharf 是第二个示例。当 wharf 部分位于多 wharf 形内部和外部 wharf，相交矩阵模式为 `'102101FF2'`：

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

## 5.2 使用空☐索引

使用空☐条件☐建☐☐☐，☐了☐得最佳性能，确保使用空☐索引（如果存在）非常重要（☐参☐ Section 4.9）。☐此，必☐在☐☐的 WHERE 或 ON 子句中使用空☐☐算符或索引感知函数。

空☐☐算符包括☐界框☐算符（其中最常用的是&&；有关完整列表，☐参☐ Section 7.10.1）以及最近☐☐☐中使用的距离☐算符（最常☐的是<->；有关完整列表，☐参☐ Section 7.10.2。）

索引感知函数会自☐将☐界框☐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☐化其操作。例如，以下☐☐在大型表上会非常慢：

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

此☐☐☐☐ geom\_table 中距离☐点 (100000, 200000) 100 个☐位以内的所有几何☐形。它会很慢，因☐它正在☐算表中每个点与指定点之☐的距离，即☐☐行 [ST\\_Distance\(\)](#) ☐算（☐表中的每一行）。

通☐使用索引感知函数[ST\\_DWithin](#) 可以大大☐少☐理的行数：

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

此☐☐☐☐相同的几何☐形，但采用更有效的方法。通☐使用 [ST\\_DWithin\(\)](#) ☐有☐☐几何☐形的放大☐界框的 && ☐算符，可以在内部☐☐。如果 geom 存在空☐索引，☐☐☐☐划器知道☐索引可用于在☐算距离之前☐少感☐趣的行数。空☐索引允☐您☐搜索与☐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☐是：

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

bc\_municipality 表定☐如下：

Column	Type	Description
gid	integer	Unique ID

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

1. 道路的总长度是多少公里？

这个问题可以用非常简单的 SQL 来回答，例如：

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

km_roads
70842.1243039643

2. 乔治王子城有多大（以公顷为单位）？

这个问题将空查询（面面）与属性条件（自治市名称）结合起来使用：

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

hectares
32657.9103824927

3. 按面积计算，省最大的直辖区是哪个？

这个问题使用空查询量作排序。有多种方法可以解决此问题，但最有效的方法如下：

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

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面的面积。如果我经常这样做，那么向表中添加一个区域列是有意义的，可以性能建立索引。通常按降序排序结果行排序，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总长度是多少？

这是“空连接”的示例，它使用空交互（“包含”）作连接条件（而不是在公共列上连接的通常关系方法）将来自两个表（有连接）的数据汇集在一起：

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

name	roads_km
SURREY	1539.47553551242

```
VANCOUVER | 1450.33093486576
LANGLEY DISTRICT | 833.793392535662
BURNABY | 773.769091404338
PRINCE GEORGE | 694.37554369147
...
```

此操作需要一段时间，因为表中的每条道路都会包含到最慢的结果中（示例表中大约有 250K 条道路）。对于小的数据集（数百条道路中的数千条），操作可能非常快。

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。也就是说，我取出个表并出一个由空切割果成的新表。与上面显示的“空连接”不同，此操作会上生成新的几何形状。生成的添加就像增加的空耦合，对于更精确的分析工作非常有用：

```
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）？

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

kilometers
-----
4.89151904172838
```

7. 哪个市政面的孔洞最大？

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

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

## Chapter 6

# 性能优化技巧

### 6.1 大几何形状的小表

#### 6.1.1 问题描述

当前版本的 PostgreSQL (包括 9.6) 在 TOAST 表后面的序列化器中存在弱点。TOAST 表是一种“展览室”，用于存储不适合普通数据页面（例如文本、图像或具有多点的复杂几何形状）的巨大数据（就数据大小而言）。有关更多信息，请参考 [Toast 的 PostgreSQL 文档](#)。

如果您碰巧有一个具有相当大的几何形状的表格，但它的行数不是太多（例如包含所有欧洲国家的高分辨率世界的表格），它会出问题。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，使用 3 个数据页，但 TOAST 表使用 8225 个页面。

在，我使用几何运算符 `&&` 来测试很少匹配的边界框。对于序列化器，表似乎只有 3 页和 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，因此它必须使用所有 TOAST 页面。

要查看您是否遇到此问题，请使用“EXPLAIN ANALYZE” `postgres` 命令。有关更多信息和技巧，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>

请参考 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，有几种解决方法：

第一个解决方法是限制索引创建器使用索引。发送“`SET enable_seqscan TO off;`”在输出之前发送到服务器。这基本上迫使索引创建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，并且它会强制索引创建器在其他情况下做出估计，因此您需要在之后“`SET enable_seqscan TO on;`”。

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

```
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：

```
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，但您也可以修改应用程序以保持 `bbox` 列最新或在每次修改后运行上面的 UPDATE 语句。



## 6.2 几何索引聚簇

对于大多数只读的表，以及大多数使用单个索引的情况，PostgreSQL 提供了 CLUSTER 命令。该命令按照与索引顺序相同的所有数据行物理重新排序，从而生成高性能索引：首先，对于索引范围扫描，数据表上的查找次数大大减少。其次，如果您的工作集集中在索引上的一些小范围，您的索引会更有效，因为数据行分布在更少的数据块上。（此索引索引 PostgreSQL 手册中的 CLUSTER 命令文档。）

但是，由于 GiST 索引只是忽略 NULL 值，因此当前无法在 PostGIS 中为 GiST 索引行聚簇，并且您会收到以下消息：

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

正如提示消息所示，向表添加“非空”限制将解决此缺陷。例如：

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

当然，如果您表上需要几何列中的 NULL 值，索引将不起作用。此外，您必须使用上述方法添加约束，使用 CHECK 约束，例如“ALTER TABLE blubb ADD CHECK (geometry is not null);”不管用。

## 6.3 避免维度

有时，您的表中碰巧有 3D 或 4D 数据，但始终使用输出 2D 几何形状的符合 OpenGIS 的 ST\_AsText() 或 ST\_AsBinary() 函数来索引它。他通常内部使用 ST\_Force2D() 函数来索引一点，这会为大型几何形状带来巨大的开销。为了避免这种开销，一次性先删除某些外的维度可能是可行的：

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

注意，如果您使用 AddGeometryColumn() 添加几何列，几何维度将会受到限制。要索引它，您需要删除约束。索引住更新 geometry\_columns 表中的条目并随后重新构建约束。

对于大型表，明智的做法是通过 WHERE 子句和主键或其他可行索引将 UPDATE 限制为表的一部分，然后进行索引的“VACUUM;”，从而将此 UPDATE 划分成更小的部分。在您的更新之后。这大大减少了索引磁块空的需求。此外，如果您有混合维度几何形状，通过“WHERE dimension(geometry)>2”限制 UPDATE 会跳过已索引于 2D 中的几何形状的重写。

# Chapter 7

## PostGIS 参考手册

下面列出的功能是 PostGIS 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 已开始从有的命名约定过渡到以 SQL-MM 为中心的约定。因此，您所了解和喜欢的大多数函数都已使用空类型 (ST) 前缀进行了重命名。以前的功能仍然可用，但本文中未列出更新后的功能等效的功能。本文中未列出的非 ST\_ 函数已被弃用，并将在未来版本中删除，因此停止使用它们。

## 7.1 PostGIS Geometry/Geography/Box 数据类型

### 7.1.1 box2d

box2d — 表示二维边界框的类型。

#### 描述

box2d 是一种空数据类型，用于表示包含几何形状或几何形状集合的二维边界框。例如，**ST\_Extent** 聚合函数返回一个 box2d 对象。

表示包含 xmin, ymin, xmax, ymax。这些是 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。

#### 类型限制

下表列出了此数据类型允许的自和式限制：

到	行
box3d	automatic
geometry	automatic

#### 相关信息

Section 13.7

## 7.1.2 box3d

box3d — 表示三维边界框的类型。

### 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边界框。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格式限制：

类型到	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间参考系的位。

### 类型限制

下表列出了此数据类型允许的自和格式限制：

类型到	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` - 表示几何图形的几何型。几何型取决于原始函数。
- `path[]` - 一个整数数组，用于定义几何图形中到 `geom` 图形的航路径。路径数组从 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，将复几何图形分解成其组成部分。

相关信息

Section 13.6

### 7.1.5 geography

`geography` — 使用大地（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。通常考虑球体模型，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）列添加到已有表。

#### 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。type 必须是与几何类型相匹配的字符串，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。

#### Note



更改：2.0.0 此函数不再更新 geometry\_columns，因为 geometry\_columns 是从系统目录中读取的。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 列在相当于：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。

#### Note



更改：2.0.0 不能再在 Geometry\_columns 中手动注册，但是根据几何 Typmod 表几何形状构建并在没有包装器函数的情况下使用的类型将正确注册自身，因为它继承了父表的 Typmod 行。使用其他几何形状的几何函数的需要指定 typmod 几何形状，以便某些几何列能在 geometry\_columns 中正确注册。参见 Section 4.6.3。

- ✓ 此方法扩展了 SQL 1.1 的 OGC 功能规范。
- ✓ 该函数支持 3d 并且不会丢失 z-index。
- ✓ 此方法支持球形字符串和曲线。

增加：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

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

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

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

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

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

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

                                Table "my_schema.my_spatial_table"
Column | Type | Modifiers
-----+-----+-----
id     | integer | not null default nextval('my_schema. ←
    my_spatial_table_id_seq'::regclass)
geom   | geometry(Point,4326) |
geom_c | geometry |
geomcp_c | geometry |
Check constraints:
    "enforce_dims_geom_c" CHECK (st_ndims(geom_c) = 2)
    "enforce_dims_geomcp_c" CHECK (st_ndims(geomcp_c) = 2)
    "enforce_geotype_geom_c" CHECK (geometrytype(geom_c) = 'POINT'::text OR geom_c IS NULL)
    "enforce_geotype_geomcp_c" CHECK (geometrytype(geomcp_c) = 'CURVEPOLYGON'::text OR ←
    geomcp_c IS NULL)
    "enforce_srid_geom_c" CHECK (st_srid(geom_c) = 4326)
    "enforce_srid_geomcp_c" CHECK (st_srid(geomcp_c) = 4326)

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

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

## 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

从空表中删除几何列。注意，schema\_name 需要与 Geometry\_columns 表中的表行的 f\_table\_schema 字段相匹配。

- ✓ 此方法实现了 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。



### Note

更改：2.0.0 提供此函数是为了向后兼容。在，由于 Geometry\_columns 是在是系统目录的，因此您可以使用 ALTER TABLE 删除几何列，就像删除任何其他表列一样。

### 示例

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

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

### 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

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



### Note

更改：2.0.0 提供此函数是☒了向后兼容。☒在，由于 `Geometry_columns` ☒在是☒☒系☒目☒的☒☒，因此您可以使用 `DROP TABLE` ☒除具有几何列的表，就像任何其他表一☒

### 示例

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

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

### 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☒的 SRID。

### Synopsis

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

### 描述

通☒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 [AddGeometryC](#) 函数），☒此函数将不起作用。

### 示例

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



相关信息

[ST\\_SRID](#)

## 7.2.5 Populate\_Geometry\_Columns

`Populate_Geometry_Columns` — 确保几何列由 `geometry` 型修饰符或具有适当的空 `geometry` 型修饰符。

### 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 `geometry` 型修饰符或空 `geometry` 型修饰符，以确保它在 `geometry_columns` 表中正确注册。默认情况下，会将所有没有 `geometry` 型修饰符的几何列插入到具有 `geometry` 型修饰符的几何列。

为了向后兼容和空 `geometry` 需求（例如表继承，其中每个子表可能具有不同的几何 `geometry` 型），仍然支持旧的 `geometry` 型修饰符。如果您需要旧的行 `geometry`，则需要将新的 `use_typmod=false` 参数作为 `use_typmod=false` 参数。完成此操作后，将 `geometry` 型修饰符的几何列，但将定义 3 个 `geometry` 型修饰符。特别是，`use_typmod=false` 意味着属于表的每个几何列至少具有三个 `geometry` 型修饰符。

- `enforce_dims_geom` - 确保每个几何 `geometry` 形具有相同的 `geometry` 度（参见 [ST\\_NDims](#)）
- `enforce_geotype_geom` - 确保每个几何具有相同的 `geometry` 型（参见 [GeometryType](#)）
- `enforce_srid_geom` - 确保所有几何 `geometry` 形位于同一投影中（参见 [ST\\_SRID](#)）

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

如果提供了 `oid` 的 `oid`，与表 `oid` 一致，此函数会 `geometry` 确定 `geometry` 表中所有几何 `geometry` 形的 `srid`、`geometry` 度和 `geometry` 型，将适当的条目插入到 `geometry_columns` 表中，但不会 `geometry` 行任何操作来 `geometry` 行 `geometry`。

无参数格式是参数格式的 `geometry` 包装器，它首先 `geometry` 除并重新填充数据 `geometry` 中每个空 `geometry` 表和 `geometry` 的 `Geometry_columns` 表，并在适当的情况下向表中添加空 `geometry` 型修饰符。它返回在数据 `geometry` 中 `geometry` 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 `geometry` 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，`geometry` 在使用 `geometry` 型修饰符而不是 `geometry` 型修饰符来 `geometry` 几何 `geometry` 型。您仍然可以通过使用新的 `use_typmod` 并将其 `geometry` 置 `geometry` `false` 来使用 `geometry` 型修饰符。

增加：2.0.0 引入了 `use_typmod` 可 `geometry` 参数，允许控制是否使用 `typmodifiers` 或 `geometry` 型修饰符 `geometry` 建列。

### 示例

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

```
populate_geometry_columns
-----
1
```

```

\d myspatial_table

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

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

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

```

## 7.2.6 UpdateGeometrySRID

UpdateGeometrySRID — 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。如果列是由型定制行的，型定将会更改。注意：如果未提供模式，在模式感知的 postgresql 安装上使用 current\_schema()。



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

将几何图形插入道路表，其中 SRID 集已使用 EWKT 格式：

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

会将路表更改为 4326，无论它以前是什么 SRID：

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

前面的示例相当于以下 DDL 语句：

```
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），并且您想一次性将 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。

```
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 几何构造函数

### 7.3.1 ST\_Collect

`ST_Collect` — 从一个几何图形构建 GeometryCollection 或 Multi\* 几何图形。

#### 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 n 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。

**Note**

如果任何输入几何形是集合 (Multi\* 或 GeometryCollection), ST\_Collect 返回 GeometryCollection (因为它是唯一可以包含嵌套集合的型)。为了防止这种情况, 在子集中使用 **ST\_Dump** 将输入集合展到其原子元素 (参看下面的示例)。

**Note**

ST\_Collect 和 **ST\_Union** 看起来很相似, 但操作上操作方式却截然不同。ST\_Collect 将几何形聚合到一个集合中, 而不以任何方式更改它们。ST\_Union 在几何上合并重叠的几何形, 并在交叉点分割串。当它合并边界, 它可能会返回几个几何形。

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形。



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-双输入格式

合并 2D 点。

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

```
st_astext
```

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

合并 3D 点。

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

```
st_asewkt
```

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

合并曲线。

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

```
st_astext
```

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

示例-数组作参数

使用子集的数组构造函数。

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

使用数组构造函数来取。

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

示例-聚合函数格式

通过在表中几何行分组来生成多个集合。

```
SELECT stusps, ST_Collect(f.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

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

相关信息

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

### 7.3.3 ST\_MakeEnvelope

ST\_MakeEnvelope — 根据最小和最大坐标建矩形多边形。

#### 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：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：生成边界框多边形

```
SELECT ST_AsText( ST_MakeEnvelope(10, 10, 11, 11, 4326) );
st_asewkt
-----
POLYGON((10 10, 10 11, 11 11, 11 10, 10 10))
```

#### 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 建 LineString。

#### Synopsis

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

#### 描述


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：接受个输入几何形

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。为了确保输入几何的序，在函数调用中使用 ORDER BY，或使用有 ORDER BY 子句的子。

输入 LineString 开的重复点将折成个点。Point 和 MultiPoint 输入中的重复点不会折。[ST\\_RemoveRepeatedPoints](#) 用于折出 LineString 中的重复点。

 函数支持 3d 并且不会丢失 z-index。

可用性：2.3.0 - 引入了 `MultiPoint` 输入元素的支持

可用性：2.0.0 - 引入了 `LineString` 输入元素的支持

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

示例：双输入格式

生成一条由 2 个点组成的 LineString。

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

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

从 3 个 3D 点生成 LineString。

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

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

从 2 个未连接的 LineString 生成一条 LineString。

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

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

示例：将数组作为参数格式

从由排序的子查询形成的数组构建一条 LineString。

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

从 3 点数组生成 LineString

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

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

示例：聚合函数格式

此示例从一轨迹中基于 GPS 点序列，并为每个轨迹构建一条 LineString。结果几何形状是由按行顺序的 GPS 轨迹点组成的 LineStrings。

`ORDER BY` 子句可用于以正确的顺序生成 LineString。

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

在 PostgreSQL 9 之前，可以使用子查询中的排序。但是，子查询可能不遵循子查询的排序。

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

相关信息

[ST\\_RemoveRepeatedPoints](#), [ST\\_AsEWKT](#), [ST\\_AsText](#), [ST\\_GeomFromText](#), [ST\\_MakePoint](#), [ST\\_Point](#)

### 7.3.5 ST\_MakePoint

ST\_MakePoint — 构建 2D、3D 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

构建一个 2D XY、3D XYZ 或 4D XYZM 的点几何对象。使用 [ST\\_MakePointM](#) 来构建具有 XYM 坐标的点。

使用 [ST\\_SetSRID](#) 来构建的点指定一个空参考系 (SRID)。

虽然不符合 OGC 标准，但 [ST\\_MakePoint](#) 比 [ST\\_GeomFromText](#) 和 [ST\\_PointFromText](#) 更快、更精确。它也更容易用于数字坐标点。



#### Note

关于大地坐标，X 是经度，Y 是纬度



#### Note

可以使用函数 [ST\\_Point](#)、[ST\\_PointZ](#)、[ST\\_PointM](#) 和 [ST\\_PointZM](#) 来构建具有指定 SRID 的点。



该函数支持 3d 并且不会丢失 z-index。



## 示例

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

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

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

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

## 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** 用于字符串输出。它是因**ST\_AsText**与 M 不兼容。

生成具有未知 SRID 的点。

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

```

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

在 WGS 84 地理坐标系中生成具有 M 的点。

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

```

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

取生成点的 M。

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

```

result
-----
10
```

相关信息

[ST\\_MakePoint](#), [ST\\_SetSRID](#), [ST\\_PointM](#), [ST\\_PointZM](#)

### 7.3.7 ST\_MakePolygon

**ST\_MakePolygon** — 从壳和可的孔列表建多形。

#### Synopsis

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

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

描述

建由定壳和可孔列形成的多形。入几何形必是合的串（）。

形式 **1**: 接受个外壳串。

形式 **2**: 接受外壳 **LineString** 和内部（孔） **LineString** 数。可以使用 PostgreSQL **array\_agg()**、**ARRAY[]** 或 **ARRAY()** 来造几何数。

**Note**

此函数不接受多行字符串。使用 `ST_LineMerge` 生成多线串。它使用 `ST_Dump` 来提取多线串。



该函数支持 3d 并且不会丢失 z-index。

示例：将 WKT 输入格式

从 2D 多线串生成多边形。

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

使用 `ST_StartPoint` 和 `ST_AddPoint` 闭合开放多线串以生成面。

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

从 3D 多线串生成多边形

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

```
st_asewkt
```

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

从具有 M 的多线串生成多边形

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

```
st_asewkt
```

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

示例：带内孔的外壳格式

构建一个带有外孔的多边形

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

构建一个省份边界，其中的孔代表湖泊。输入是省份多边形/多边形表和水域多线串表。形成湖泊的孔是通过使用 `ST_IsClosed` 确定的。使用 `ST_Boundary` 提取省份线条。根据 `ST_MakePolygon` 的要求，通过使用 `ST_LineMerge` 将边界限制为一个 LineString。（但是，注意，如果一个省份有多个区域或具有孔，将生成无效的多边形。）使用 LEFT JOIN 可确保包括所有省份，即使它没有湖泊。

**Note**

使用 CASE 构造是因为将空数传递到 `ST_MakePolygon` 会导致 NULL 返回。

```

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

```

一种技术是利用相关子查询和 ARRAY() 构造函数将行集转换为数组。

```

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 — 构建具有 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

于大地坐标，X 是度，Y 是度

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。



方法了 SQL/MM 范。SQL-MM 3: 6.1.2

示例：几何

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

☒ 建一个指定了 SRID 的点：

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

指定 SRID 的☒一种方式：

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

示例：地理

☒ 建 **地理** 点（使用 :: ☒☒☒法）：

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

在 Pre-PostGIS 3.2 版本的代☒中，使用 CAST：

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

如果点的坐☒不在地理坐☒系（如 WGS84）中，☒必☒先☒行坐☒☒☒，然后再将其投射到地理。在此示例中，☒夕法尼☒州平面英尺（SRID 2273）上的点将☒☒☒ WGS84（SRID 4326）。

```
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 — ☒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☒定 X、Y 和 Z 坐☒☒的点，如果☒定，☒☒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

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

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

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

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

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

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

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

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

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

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

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

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 `LINESTRING` 构建多边形。

#### Synopsis

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

描述

返回根据 `LINESTRING` 构建的多边形，并根据 `srid` 置空参考系。

ST\_Polygon 类似于 [ST\\_MakePolygon](#) 格式 1，但添加了 SRID 置。

要构建孔的多边形，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 `LINESTRING`。它使用 [ST\\_Dump](#) 来提取 `LINESTRING`。



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: 8.3.2



此函数支持 3d 并且不会丢失 z-index。

示例

生成二维多边形。

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

```
-- result --
```

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

生成三维多边形。

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

```
-- result --
```

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

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;LID=20037508.342789 -20037508.342789,20037508.342789 20037508.342789), float margin=0.0);
```

#### 描述

构建一个矩形多边形，输出 [XYZ 切片系](#) 中切片的范围。范围由放大 Z 和范围的网格中的 XY 索引指定。可用于定义 [ST\\_AsMVTGeom](#) 所需的范围，以将几何形状放入 MVT 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](#) 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“放大零”正方形的范围，XYZ 系内切在正方形内。

可的 `margin` 参数可用于将范围展定的百分比。例如，`margin=0.125` 将范围展 12.5%，相当于当范围大小 4096 缓冲=512 (如 [ST\\_AsMVTGeom](#) 中使用的)。用于构建缓冲区以包含位于可区域之外的数据非常有用，但其存在会影响渲染。例如，城市名称 (一个点) 可能靠近范围的，因此其范围呈在个上，即使点位于一个的可区域中。在范围内使用展将在个中包含城市点。使用来缩小。禁止使用小于 -0.5 的，因会完全消除。与 [ST\\_AsMVTGeom](#) 一起使用，勿指定距。参 [ST\\_AsMVT](#)。

增：添加了 3.1.0 `margin` 参数。

可用性：3.0.0

示例：构建格网外接矩形

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

```
st_astext
```

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

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

```
st_astext
```

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

相关信息

[ST\\_MakeEnvelope](#)



### 7.3.14 ST\_HexagonGrid

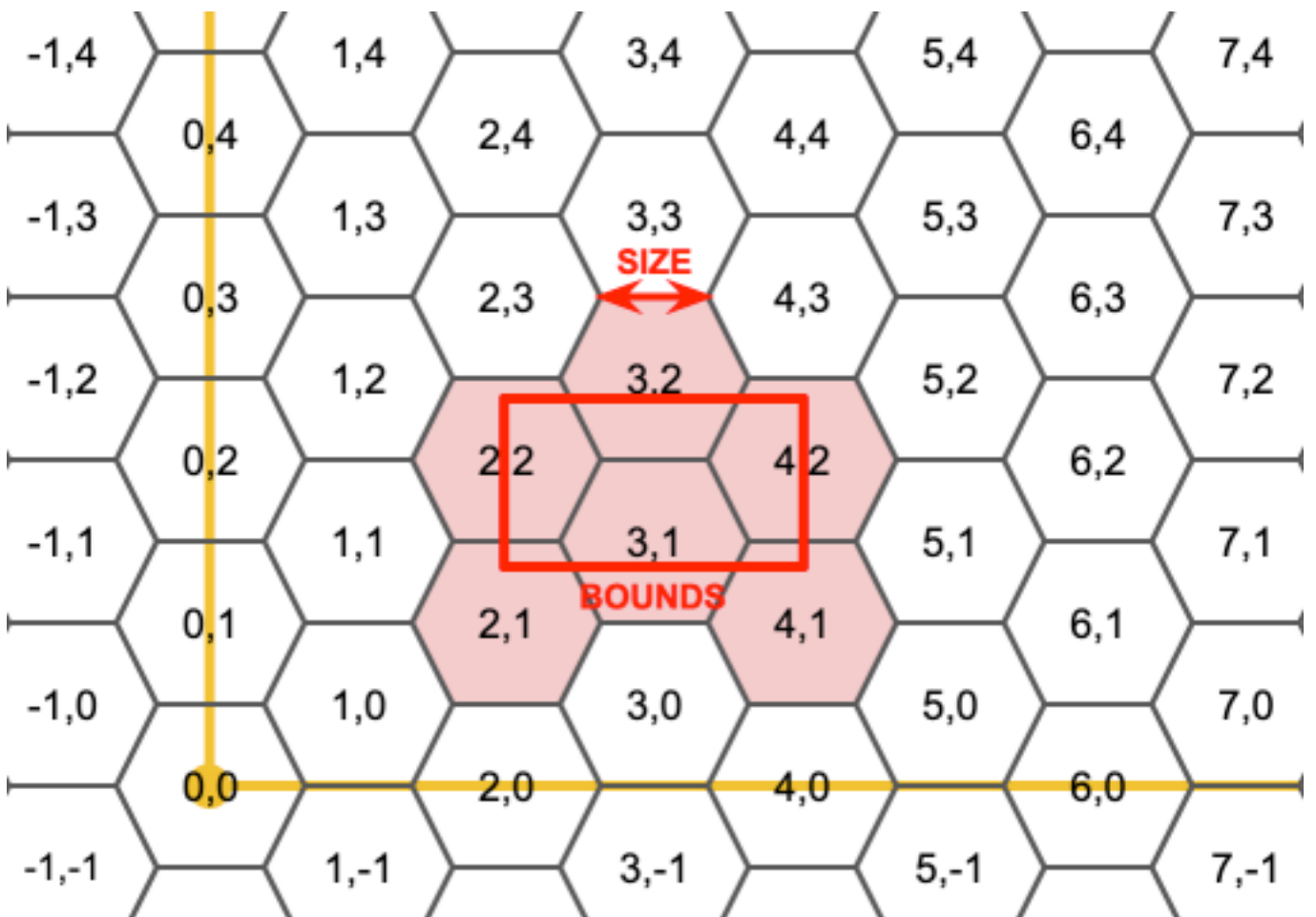
ST\_HexagonGrid — 返回一个完全覆盖几何参数边界的六边形和元格索引。

#### Synopsis

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

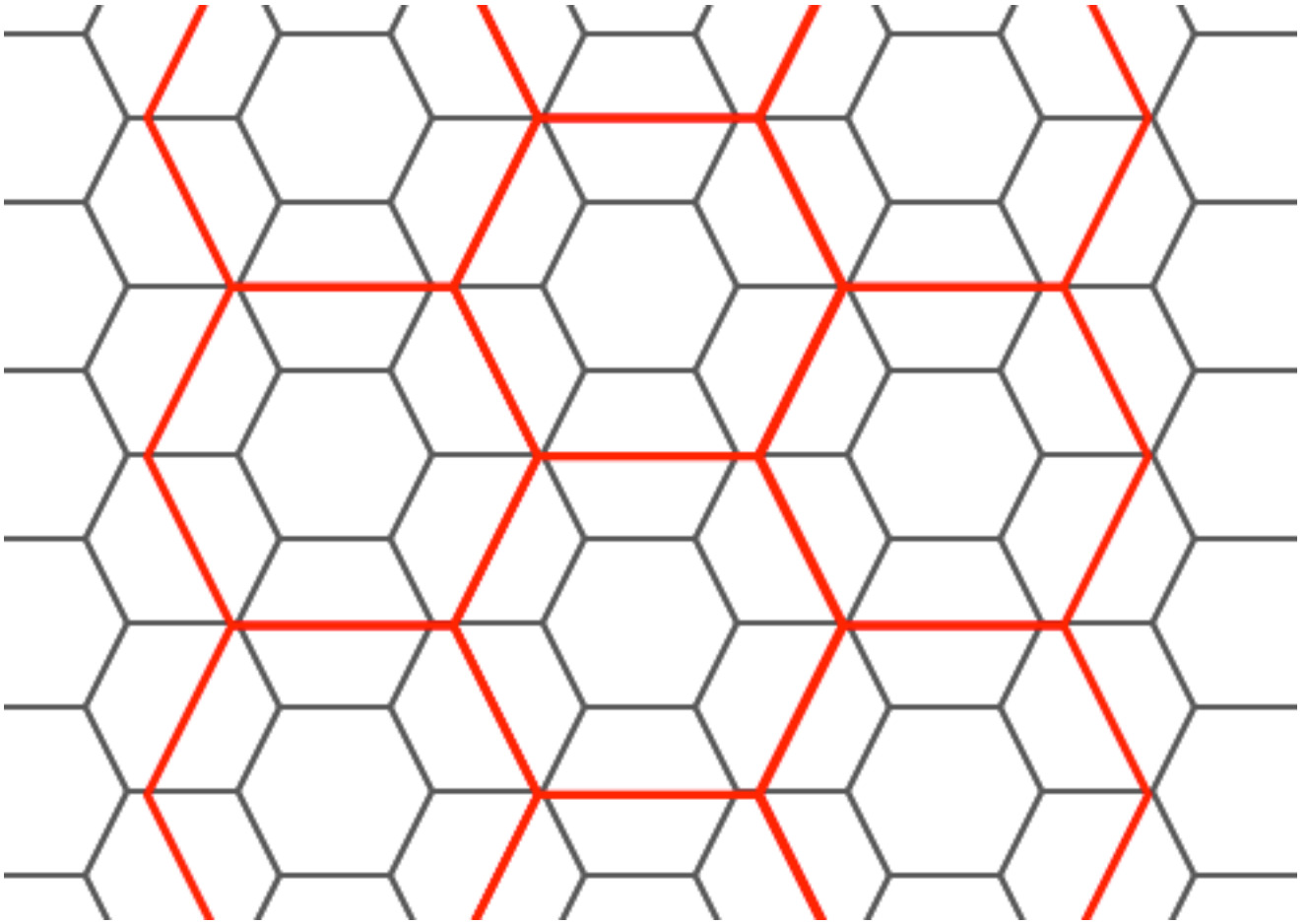
#### 描述

从平面六边形格网的概念开始。（不是地球上的六边形格网，不是H3格网方案。）对于给定的平面 SRS 和给定的元格大小，从 SRS 的原点开始，有一种独特的平面六边形格网，Tiling(SRS, Size)。此函数回答了以下问题：给定 Tiling(SRS, Size) 中的哪些六边形与给定边界重叠。



输出六边形的 SRS 是边界几何体提供的 SRS。

将六边形的元格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

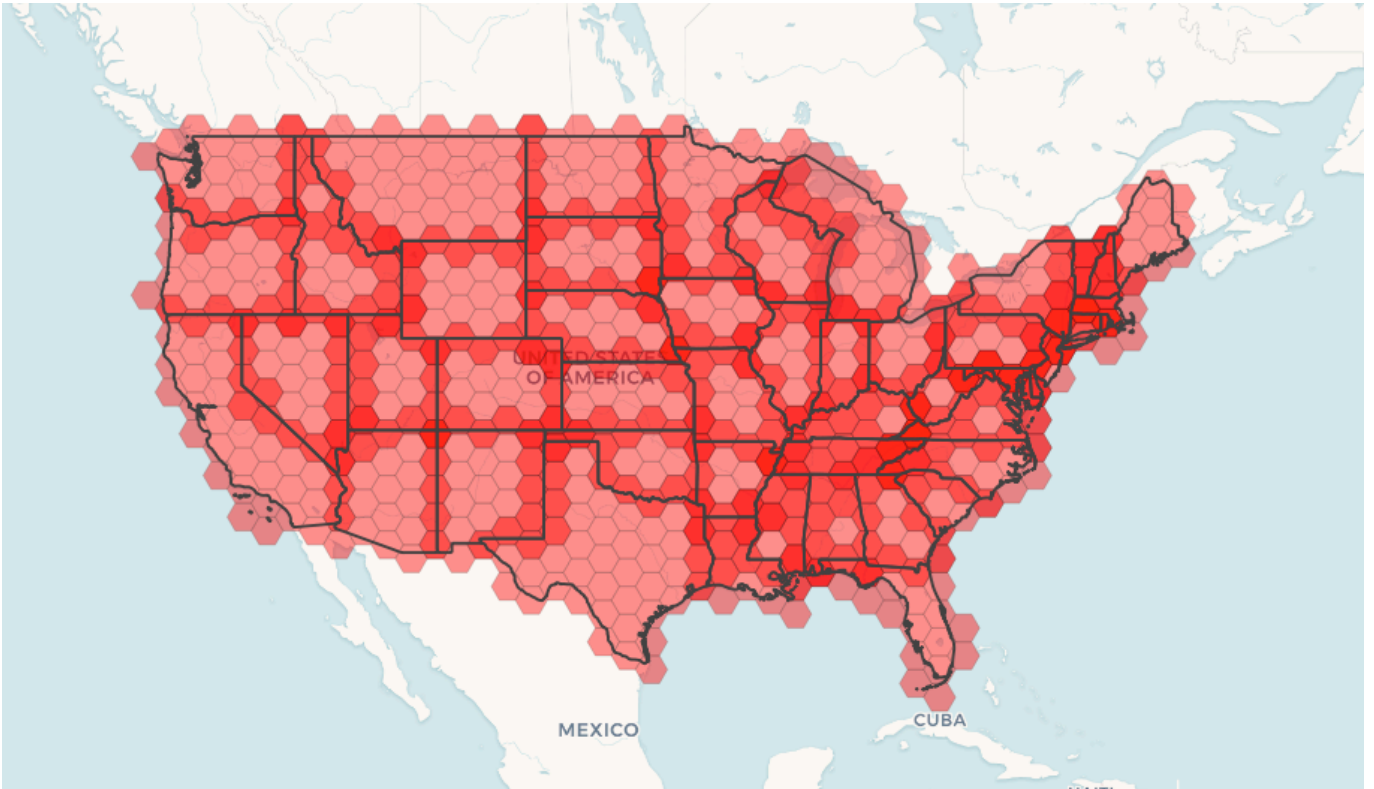
示例 : 计算六边形中的点

要对六边形网格进行点聚合，使用点的范围作边界生成六边形网格，然后在空上连接到网格。

```
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;
```

示例 : 面生成六边形覆盖范围

如果我每个多边形边界生成一个六边形并去掉那些不与其六边形相交的六边形，我最会得到每个多边形的网格。



州切片提供每个州的六边形的覆盖范围，多个六边形与州边界重叠。



#### 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 或显式的 CROSS JOIN LATERAL 是本示例的等效构造。

```
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，但在所需的元格坐生成一个六边形。（可）可以整格网的原点坐，默原点 0,0。

生成的六边形没有置 SRID，因此使用 **ST\_SetSRID** 将 SRID 置您期望的。

可用性：3.1.0

示例：在 origin 生成六边形

```
SELECT ST_AsText(ST_SetSRID(ST_Hexagon(1.0, 0, 0), 3857));
POLYGON((-1 0,-0.5
          -0.866025403784439,0.5
          -0.866025403784439,1
          0,0.5
          0.866025403784439,-0.5
          0.866025403784439,-1 0))
```

## 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。于固定的平面 SRS 和固定的尺寸，从 SRS 的原点开始，存在一个独特的平面方格网，**Tiling(SRS, Size)**。此函数回答了以下问题：固定 **Tiling(SRS, Size)** 中的哪些网格与固定界重。

出正方形的 SRS 是界几何的 SRS。

正方形的加倍或大小会生成与原始格网完美契合的新父格网。墨卡托中的准网地格网只是墨卡托平面中的个平方网格的。

可用性：3.1.0

示例：一个国家生成 1 度格网

网格将填充国家/地区的整个界，因此如果您只想接触国家/地区的方，必随后使用 **ST\_Intersects** 行。

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

示例：计算正方形中的点数（使用 1000 个 1000 粒度网格）

要将方形网格与行点聚合，使用点的范围作边界生成方形网格，然后在空上连接到网格。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。

```
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
```

示例：使用每个点的网格集正方形中的点行数

会产生与第一个示例相同的结果，但由于大量点来会比较慢

```
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);

#### 描述

使用与 `ST_SquareGrid` 相同的正方形网格概念，但在所需的元格坐标生成一个正方形。（可）可以整平的原点坐标，默认为原点 0,0。

生成的方没有设置 SRID，因此使用 `ST_SetSRID` 将 SRID 设置您期望的。

可用性：3.1.0

示例：在 origin 生成矩形

```
SELECT ST_AsText(ST_SetSRID(ST_Square(1.0, 0, 0), 3857));  
POLYGON((0 0,0 1,1 1,1 0,0 0))
```

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：生成 “Yo”

```
SELECT ST_AsText(ST_Letters('Yo'), 1);
```



ST\_Letter 生成的字符

示例：放大和移

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

相关信息

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

## 7.4 几何器

### 7.4.1 GeometryType

GeometryType — 以文本形式返回几何的型。

#### Synopsis

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

描述

以字符串形式返回几何型，例如“LINESTRING”、“POLYGON”、“MULTIPOINT”等。

OGC SPEC s2.1.1.1 - 返回 Geometry 的可例子型的名称，Geometry 的可例子型的名称以字符串形式返回。



#### Note

函数返回“POINTM”形式的字符串来指示是否具有 M。

增功能：引入了 2.0.0 多面体曲面、三角形和三角网的支持。



此方法了 SQL 1.1 的 OGC 功能范。



此方法支持形字符串和曲。



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

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

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

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

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包，因此是拓扑封闭的，因此可以使用 OGC SPEC 第 3.12.2 节中定义的表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）。

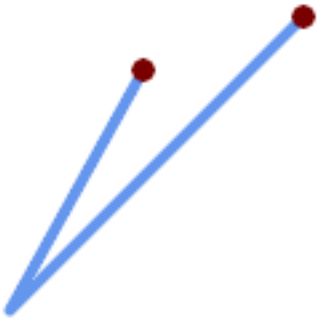
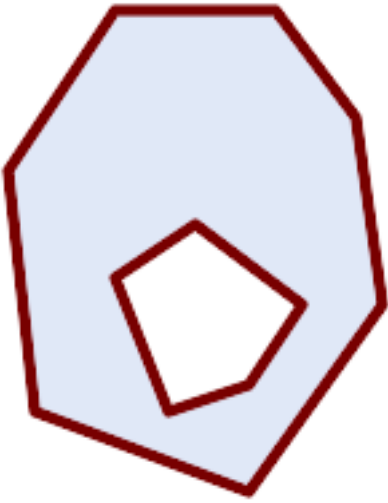


- ✔ 此方法符合了 SQL 1.1 的 OGC 功能规范。 OGC SPEC s2.1.1.1
- ✔ 方法符合了 SQL/MM 规范。 SQL-MM IEC 13249-3: 5.1.17
- ✔ 函数支持 3d 并且不会丢失 z-index。

增加：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，不个性化曲线

示例

 <p>具有重叠端点的线串</p> <pre>SELECT ST_Boundary(geom) FROM (SELECT 'LINESTRING(100 150,50 60, ← 70 80, 160 170)&gt;:::geometry As geom) As f;  ST_AsText output  MULTIPOINT((100 150),(160 170))</pre>	 <p>具有边界多线串的多边形孔</p> <pre>SELECT ST_Boundary(geom) FROM (SELECT 'POLYGON (( 10 130, 50 190, 110 190, 140 ← 150, 150 80, 100 10, 20 40, 10 130 ), ( 70 40, 100 50, 120 80, 80 110, ← 50 90, 70 40 ))&gt;:::geometry As geom) As f;  ST_AsText output  MULTILINESTRING((10 130,50 190,110 ← 190,140 150,150 80,100 10,20 40,10 130), (70 40,100 50,120 80,80 110,50 ← 90,70 40))</pre>
--	---

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

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

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

st_asewkt
-----
LINESTRING(1 1 1,0 0 1,-1 1 1,1 1 1)

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

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

相关信息

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

### 7.4.3 ST\_BoundingDiagonal

`ST_BoundingDiagonal` — 返回几何体的边界框的对角线。

#### Synopsis

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

#### 描述

以字符串形式返回给定几何体的边界框的对角线。它是一个由点组成的字符串，从最小点开始，到最大点结束。如果输入几何体为空，则返回空。

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体，计算速度更快）。无论何种情况，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



该函数支持 3d 并且不会丢失 z-index。



该功能支持 M 坐标。

示例

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

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 $\times$ 度。







#### Synopsis

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

描述

返回 ST\_Geometry  $\times$ 的坐 $\times$ 度。

$\times$ 是 [ST\\_NDims](#) 的 MM 兼容 $\times$ 名

-  此方法 $\times$ 了 SQL 1.1 的 OGC  $\times$ 功能 $\times$ 范。
-   $\times$ 方法 $\times$ 了 SQL/MM  $\times$ 范。SQL-MM 3: 5.1.3
-  此方法支持 $\times$ 形字符串和曲 $\times$ 。
-   $\times$ 函数支持 3d 并且不会 $\times$ 失 z-index。
-   $\times$ 函数支持多面体曲面。
-  此函数支持三角形和不 $\times$ 三角网面 (TIN)。

示例

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

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

相关信息

[ST\\_NDims](#)

## 7.4.5 ST\_Dimension

`ST_Dimension` — 返回几何对象的拓扑维数。

### 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度。OGC SPEC s2.1.1.1 - 于 POINT 返回 0，于 LINESTRING 返回 1，于 POLYGON 返回 2，以及 GEOMETRYCOLLECTION 件的最大尺寸。如果尺寸未知（例如，于空 GEOMETRYCOLLECTION），返回 0。



方法符合了 SQL/MM 范。SQL-MM 3: 5.1.2

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

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

相关信息

[ST\\_NDims](#)

## 7.4.6 ST\_Dump

`ST_Dump` — 返回几何件的一 geometry\_dump 行。

### Synopsis

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

## 描述

提取几何对象的集合返回函数 (SRF)。它返回一行 `geometry_dump` 行，每行包含一个几何对象 (`geom` 字段) 和一个整数数字 (`path` 字段)。

基于基本几何类型 (POINT、LINESTRING、POLYGON)，返回一个对象，其中包含空 `path` 数字，输入几何对象 `geom`。基于集合或多几何体，将每个集合对象返回一条记录，并且路径表示对象在集合内的位置。

`ST_Dump` 基于展开几何对象很有用。它与 `ST_Collect / GROUP BY` 相反，它新建行。例如，它可用于将 MULTIPOLYGONS 展开 POLYGONS。

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：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对象一起使用时崩溃。此问题已在 1.3.4 及更高版本中得到修正。



此方法支持对象字符串和曲线。



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。

## 基准示例

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

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

## 多面体曲面、TIN 和三角形的示例

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

```
)') ) AS p_geom ) AS a;
```

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

```
-- TIN --
SELECT (g.gdump).path, ST_AsEWKT((g.gdump).geom) as wkt
FROM
  (SELECT
    ST_Dump( ST_GeomFromEWKT('TIN (((
      0 0 0,
      0 0 1,
      0 1 0,
      0 0 0
    )), ((
      0 0 0,
      0 1 0,
      1 1 0,
      0 0 0
    ))
    )') ) AS gdump
  ) AS g;
-- result --
path |          wkt
-----+-----
{1}  | TRIANGLE((0 0 0,0 0 1,0 1 0,0 0 0))
{2}  | TRIANGLE((0 0 0,0 1 0,1 1 0,0 0 0))
```

## 相关信息

[geometry\\_dump](#), [Section 13.6](#), [ST\\_Collect](#), [ST\\_GeometryN](#)

## 7.4.7 ST\_DumpPoints

`ST_DumpPoints` — 返回几何形中坐的一 `geometry_dump` 行。

### Synopsis

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

### 描述

提取几何形坐 (点) 的集合返回函数 (SRF)。它返回一 `geometry_dump` 行, 每行包含一个几何形 (`geom` 字段) 和一个整数数 (`path` 字段)。

- `geom` 字段 POINT 表示所提供几何形的坐。

- *path* 字段 (`integer[]`) 是枚所提供的几何形元素中的坐位置的索引。索引是从 1 开始的。例如，于 `LINestring`, 路径 {i}, 其中 i 是 `LINestring` 中的第 n 个坐。于 `POLYGON`, 路径 {i,j}, 其中 i 是号 (1 是外; 后面是内), j 是中的坐位置。

要取包含坐的个几何形, 使用 `ST_Points`。

增: 2.1.0 速度更快。重新原生 C 言。

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: 1.1.0

- ✓ 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。

### 将 `LineStrings` 表分解点

```
SELECT edge_id, (dp).path[1] As index, ST_AsText((dp).geom) As wktnode
FROM (SELECT 1 As edge_id
      , ST_DumpPoints(ST_GeomFromText('LINestring(1 2, 3 4, 10 10)')) AS dp
      UNION ALL
      SELECT 2 As edge_id
      , ST_DumpPoints(ST_GeomFromText('LINestring(3 5, 5 6, 9 10)')) AS dp
     ) As foo;
```

edge_id	index	wktnode
1	1	POINT(1 2)
1	2	POINT(3 4)
1	3	POINT(10 10)
2	1	POINT(3 5)
2	2	POINT(5 6)
2	3	POINT(9 10)

### 准几何示例



```

SELECT path, ST_AsText(geom)
FROM (
  SELECT (ST_DumpPoints(g.geom)).*
  FROM
    (SELECT
      'GEOMETRYCOLLECTION(
        POINT ( 0 1 ),
        LINESTRING ( 0 3, 3 4 ),
        POLYGON (( 2 0, 2 3, 0 2, 2 0 )),
        POLYGON (( 3 0, 3 3, 6 3, 6 0, 3 0 ),
          ( 5 1, 4 2, 5 2, 5 1 )),
        MULTIPOLYGON (
          (( 0 5, 0 8, 4 8, 4 5, 0 5 ),
            ( 1 6, 3 6, 2 7, 1 6 )),
          (( 5 4, 5 8, 6 7, 5 4 ))
        )
      )::geometry AS geom
    ) AS g
  ) j;

```

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

(29 rows)

多面体曲面、**TIN** 和三角形的示例

```

-- Polyhedral surface cube --
SELECT (g.gdump).path, ST_AsEWKT((g.gdump).geom) as wkt
FROM
  (SELECT

```



```

        ST_DumpPoints(ST_GeomFromEWKT('POLYHEDRALSURFACE( ((0 0 0, 0 0 1, 0 1 1, 0 1 0, 0 0 ←
        0)),
((0 0 0, 0 1 0, 1 1 0, 1 0 0, 0 0 0)), ((0 0 0, 1 0 0, 1 0 1, 0 0 1, 0 0 0)),
((1 1 0, 1 1 1, 1 0 1, 1 0 0, 1 1 0)),
((0 1 0, 0 1 1, 1 1 1, 1 1 0, 0 1 0)), ((0 0 1, 1 0 1, 1 1 1, 0 1 1, 0 0 1)) )' ) AS gdump
    ) AS g;
-- result --
  path | wkt
-----+-----
{1,1,1} | POINT(0 0 0)
{1,1,2} | POINT(0 0 1)
{1,1,3} | POINT(0 1 1)
{1,1,4} | POINT(0 1 0)
{1,1,5} | POINT(0 0 0)
{2,1,1} | POINT(0 0 0)
{2,1,2} | POINT(0 1 0)
{2,1,3} | POINT(1 1 0)
{2,1,4} | POINT(1 0 0)
{2,1,5} | POINT(0 0 0)
{3,1,1} | POINT(0 0 0)
{3,1,2} | POINT(1 0 0)
{3,1,3} | POINT(1 0 1)
{3,1,4} | POINT(0 0 1)
{3,1,5} | POINT(0 0 0)
{4,1,1} | POINT(1 1 0)
{4,1,2} | POINT(1 1 1)
{4,1,3} | POINT(1 0 1)
{4,1,4} | POINT(1 0 0)
{4,1,5} | POINT(1 1 0)
{5,1,1} | POINT(0 1 0)
{5,1,2} | POINT(0 1 1)
{5,1,3} | POINT(1 1 1)
{5,1,4} | POINT(1 1 0)
{5,1,5} | POINT(0 1 0)
{6,1,1} | POINT(0 0 1)
{6,1,2} | POINT(1 0 1)
{6,1,3} | POINT(1 1 1)
{6,1,4} | POINT(0 1 1)
{6,1,5} | POINT(0 0 1)
(30 rows)

```

```

-- Triangle --
SELECT (g.gdump).path, ST_AsText((g.gdump).geom) as wkt
FROM
  (SELECT
    ST_DumpPoints( ST_GeomFromEWKT('TRIANGLE ((
      0 0,
      0 9,
      9 0,
      0 0
    ))' ) AS gdump
  ) AS g;
-- result --
  path | wkt
-----+-----
{1} | POINT(0 0)
{2} | POINT(0 9)
{3} | POINT(9 0)
{4} | POINT(0 0)

```

-- TIN --

```

SELECT (g.gdump).path, ST_AsEWKT((g.gdump).geom) as wkt
FROM
  (SELECT
    ST_DumpPoints( ST_GeomFromEWKT('TIN (((
      0 0 0,
      0 0 1,
      0 1 0,
      0 0 0
    )), ((
      0 0 0,
      0 1 0,
      1 1 0,
      0 0 0
    ))
    )') ) AS gdump
  ) AS g;
-- result --
 path | wkt
-----+-----
{1,1,1} | POINT(0 0 0)
{1,1,2} | POINT(0 0 1)
{1,1,3} | POINT(0 1 0)
{1,1,4} | POINT(0 0 0)
{2,1,1} | POINT(0 0 0)
{2,1,2} | POINT(0 1 0)
{2,1,3} | POINT(1 1 0)
{2,1,4} | POINT(0 0 0)
(8 rows)

```

#### 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)。

- the `geom` field `LINESTRINGS` represent the linear segments of the supplied geometry, while the `CIRCULARSTRINGS` represent the arc segments.
- `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

```
SELECT path, ST_AsText(geom)
FROM (
  SELECT (ST_DumpSegments(g.geom)).*
  FROM (SELECT 'GEOMETRYCOLLECTION(
  LINESTRING(1 1, 3 3, 4 4),
  POLYGON((5 5, 6 6, 7 7, 5 5))
)>:::geometry AS geom
      ) AS g
) j;
```

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

(5 rows)

**TIN** 和三角形示例

```
-- Triangle --
SELECT path, ST_AsText(geom)
FROM (
  SELECT (ST_DumpSegments(g.geom)).*
  FROM (SELECT 'TRIANGLE((
    0 0,
    0 9,
    9 0,
    0 0
  ))>:::geometry AS geom
      ) AS g
) j;
```

path	b'' b''	st_astext
{1,1}	b'' b''	LINESTRING(0 0,0 9)
{1,2}	b'' b''	LINESTRING(0 9,9 0)
{1,3}	b'' b''	LINESTRING(9 0,0 0)

(3 rows)

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

```

    0 0 0
 )), ((
    0 0 0,
    0 1 0,
    1 1 0,
    0 0 0
  ))
)>:::geometry AS geom
) AS g
) j;

```

path	b'' b''	st_asewkt
{1,1,1}	b'' b''	LINESTRING(0 0 0,0 0 1)
{1,1,2}	b'' b''	LINESTRING(0 0 1,0 1 0)
{1,1,3}	b'' b''	LINESTRING(0 1 0,0 0 0)
{2,1,1}	b'' b''	LINESTRING(0 0 0,0 1 0)
{2,1,2}	b'' b''	LINESTRING(0 1 0,1 1 0)
{2,1,3}	b'' b''	LINESTRING(1 1 0,0 0 0)

(6 rows)

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列 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)。它返回一列 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

适用于 POLYGON, 它不适用于 MULTIPOLYGONS

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

☐☐的常☐格式。

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

具有☐孔的多☐形。

```
SELECT path, ST_AsEWKT(geom) As geom
FROM ST_DumpRings(
  ST_GeomFromEWKT('POLYGON((-8149064 5133092 1,-8149064 5132986 1,-8148996 ↵
    5132839 1,-8148972 5132767 1,-8148958 5132508 1,-8148941 5132466 ↵
    1,-8148924 5132394 1,
    -8148903 5132210 1,-8148930 5131967 1,-8148992 5131978 1,-8149237 5132093 ↵
    1,-8149404 5132211 1,-8149647 5132310 1,-8149757 5132394 1,
    -8150305 5132788 1,-8149064 5133092 1),
    (-8149362 5132394 1,-8149446 5132501 1,-8149548 5132597 1,-8149695 5132675 ↵
    1,-8149362 5132394 1)))')
) as foo;
```

path	geom
{0}	POLYGON((-8149064 5133092 1,-8149064 5132986 1,-8148996 5132839 1,-8148972 5132767 ↵ 1,-8148958 5132508 1,   -8148941 5132466 1,-8148924 5132394 1,   -8148903 5132210 1,-8148930 5131967 1,   -8148992 5131978 1,-8149237 5132093 1,   -8149404 5132211 1,-8149647 5132310 1,-8149757 5132394 1,-8150305 ↵ 5132788 1,-8149064 5133092 1))
{1}	POLYGON((-8149362 5132394 1,-8149446 5132501 1,   -8149548 5132597 1,-8149695 5132675 1,-8149362 5132394 1))

相关信息

[geometry\\_dump](#), [Section 13.6](#), [ST\\_Dump](#), [ST\\_ExteriorRing](#), [ST\\_InteriorRingN](#)

## 7.4.10 ST\_EndPoint

ST\_EndPoint — 返回 LineString 或 CircularLineString 的最后一个点。

### Synopsis

```
geometry ST_EndPoint(geometry g);
```

描述

返回 `LINestring` 或 `CIRCULARLINestring` 几何☐形的最后一个点作☐ `POINT`。如果☐入不是 `LINestring` 或 `CIRCULARLINestring`，☐返回 `NULL`。



☐方法☐☐了 SQL/MM ☐范。SQL-MM 3: 7.1.4



☐函数支持 3d 并且不会☐失 z-index。



此方法支持☐形字符串和曲☐。

**Note**

更改：2.0.0 不再适用于 `ST_EndPoint` 几何体 `MultiLineStrings`。在旧版本的 PostGIS 中，`ST_EndPoint` 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 `ST_EndPoint` 是一个未实现的功能，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

## 线串 (LineString) 端点

```
postgis=# SELECT ST_AsText(ST_EndPoint('LINESTRING(1 1, 2 2, 3 3)::geometry));
st_astext
-----
POINT(3 3)
```

## 非线串中止点返回 NULL

```
SELECT ST_EndPoint('POINT(1 1)::geometry') IS NULL AS is_null;
is_null
-----
t
```

## 3D 线串 (LineString) 端点

```
--3d endpoint
SELECT ST_AsEWKT(ST_EndPoint('LINESTRING(1 1 2, 1 2 3, 0 0 5)'));
st_asewkt
-----
POINT(0 0 5)
```

## 弧的端点

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

## 相关信息

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

**7.4.11 ST\_Envelope**

`ST_Envelope` — 返回表示几何形状边界框的几何形状。

**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) 最小包围框。多边形由包围框的角点定义 ((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 更低维度的几何形状, 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

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

SELECT ST_AsText(ST_Envelope('LINESTRING(0 0, 1 3)::geometry'));
 st_astext
-----
POLYGON((0 0,0 3,1 3,1 0,0 0))
(1 row)

SELECT ST_AsText(ST_Envelope('POLYGON((0 0, 0 1, 1.0000001 1, 1.0000001 0, 0 0))::geometry ←
));
 st_astext
-----
POLYGON((0 0,0 1,1.00000011920929 1,1.00000011920929 0,0 0))
(1 row)
SELECT ST_AsText(ST_Envelope('POLYGON((0 0, 0 1, 1.0000000001 1, 1.0000000001 0, 0 0))':: ←
geometry));
 st_astext
-----
POLYGON((0 0,0 1,1.00000011920929 1,1.00000011920929 0,0 0))
(1 row)

SELECT Box3D(geom), Box2D(geom), ST_AsText(ST_Envelope(geom)) As envelopewkt
FROM (SELECT 'POLYGON((0 0, 0 1000012333334.34545678, 1.0000001 1, 1.0000001 0, 0 ←
0))'::geometry As geom) As foo;
```



点和线串的最小外接矩形。

```
SELECT ST_AsText(ST_Envelope(
    ST_Collect(
        ST_GeomFromText('LINESTRING(55 75,125 150)'),
        ST_Point(20, 80)
    )) As wktenv;
wktenv
-----
POLYGON((20 75,20 150,125 150,125 75,20 75))
```

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，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
- ✓ 此方法符合了 [SQL/MM 规范](#)。SQL-MM 3: 8.2.3, 8.3.3
- ✓ 此函数支持 3d 并且不会丢失 z-index。



示例

```
--If you have a table of polygons
SELECT gid, ST_ExteriorRing(geom) AS ering
FROM sometable;

--If you have a table of MULTIPOLYGONS
--and want to return a MULTILINESTRING composed of the exterior rings of each polygon
SELECT gid, ST_Collect(ST_ExteriorRing(geom)) AS erings
      FROM (SELECT gid, (ST_Dump(geom)).geom As geom
            FROM sometable) As foo
GROUP BY gid;

--3d Example
SELECT ST_AsEWKT(
      ST_ExteriorRing(
      ST_GeomFromEWKT('POLYGON((0 0 1, 1 1 1, 1 2 1, 1 1 1, 0 0 1))')
      )
);

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

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

返回几何形的从 1 开始的第 N 个元素几何形，几何形是 GEOMETRYCOLLECTION、MULTIPOINT、MULTILINESTRING、MULTICURVE、MULTIPOLYGON 或 POLYHEDRALSURFACE。否则，返回 NULL。



#### Note

自版本 0.8.0 以来，OGC 规范的索引从 1 开始。以前的版本将其索引基于 0。



#### Note

要提取几何形的所有元素，[ST\\_Dump](#) 效率更高，并且适用于基本几何形。

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。

更改：2.0.0 版本。之前的版本对于奇异几何形会返回 NULL。在已更改在 ST\_GeometryN(..,1) 情况下返回几何形。

- ✔ 此方法实现了 SQL 1.1 的 OGC 功能规范。
- ✔ 方法实现了 SQL/MM 规范。SQL-MM 3: 9.1.5
- ✔ 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。
- ✔ 函数支持多面体曲面。
- ✔ 此函数支持三角形和不三角网面 (TIN)。

#### 准示例

```
--Extracting a subset of points from a 3d multipoint
SELECT n, ST_AsEWKT(ST_GeometryN(geom, n)) As geomewkt
FROM (
VALUES (ST_GeomFromEWKT('MULTIPOINT((1 2 7), (3 4 7), (5 6 7), (8 9 10))') ),
(ST_GeomFromEWKT('MULTICURVE(CIRCULARSTRING(2.5 2.5,4.5 2.5, 3.5 3.5), (10 11, 12 11))') )
)As foo(geom)
CROSS JOIN generate_series(1,100) n
WHERE n <= ST_NumGeometries(geom);
```

n	geomewkt
1	POINT(1 2 7)
2	POINT(3 4 7)
3	POINT(5 6 7)
4	POINT(8 9 10)
1	CIRCULARSTRING(2.5 2.5,4.5 2.5,3.5 3.5)
2	LINestring(10 11,12 11)

```
--Extracting all geometries (useful when you want to assign an id)
SELECT gid, n, ST_GeometryN(geom, n)
FROM sometable CROSS JOIN generate_series(1,100) n
WHERE n <= ST_NumGeometries(geom);
```

#### 多面体曲面、TIN 和三角形的示例

```
-- Polyhedral surface example
-- Break a Polyhedral surface into its faces
SELECT ST_AsEWKT(ST_GeometryN(p_geom,3)) As geom_ewkt
FROM (SELECT ST_GeomFromEWKT('POLYHEDRALSURFACE(
((0 0 0, 0 0 1, 0 1 1, 0 1 0, 0 0 0)),
((0 0 0, 0 1 0, 1 1 0, 1 0 0, 0 0 0)),
((0 0 0, 1 0 0, 1 0 1, 0 0 1, 0 0 0)),
((1 1 0, 1 1 1, 1 0 1, 1 0 0, 1 1 0)),
((0 1 0, 0 1 1, 1 1 1, 1 1 0, 0 1 0)),
((0 0 1, 1 0 1, 1 1 1, 0 1 1, 0 0 1))
)') AS p_geom ) AS a;
```

```

                                geom_ewkt
-----
POLYGON((0 0 0,1 0 0,1 0 1,0 0 1,0 0 0))

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

```

相关信息

[ST\\_Dump](#), [ST\\_NumGeometries](#)

### 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

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

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

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

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

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

```
SELECT ST_HasArc(ST_Collect('LINESTRING(1 2, 3 4, 5 6)', 'CIRCULARSTRING(1 1, 2 3, 4 5, 6 7, 5 6)'));
 st_hasarc
-----
t
```

### 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。如果几何形不是多边形或索引超出范围，返回 NULL。



#### Note

此函数不支持 MULTIPOLYGON。用于 MULTIPOLYGON，与 [ST\\_GeometryN](#) 或 [ST\\_Dump](#) 合使用



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 3: 8.2.6, 8.3.5



函数支持 3d 并且不会丢失 z-index。

示例

```
SELECT ST_AsText(ST_InteriorRingN(geom, 1)) As geom
FROM (SELECT ST_BuildArea(
    ST_Collect(ST_Buffer(ST_Point(1,2), 20,3),
    ST_Buffer(ST_Point(1, 2), 10,3))) As geom
) as foo;
```

相关信息

[ST\\_ExteriorRing](#), [ST\\_BuildArea](#), [ST\\_Collect](#), [ST\\_Dump](#), [ST\\_NumInteriorRing](#), [ST\\_NumInteriorRings](#)

### 7.4.17 ST\_NumCurves

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

#### Synopsis

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

描述

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



☑方法☑☑了 SQL/MM ☑范。SQL-MM 3: 8.2.6, 8.3.5



☑函数支持 3d 并且不会☑失 z-index。

示例

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

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

相关信息

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

### 7.4.18 ST\_CurveN

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

## Synopsis

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

### 描述

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



此方法符合了 SQL/MM 规范。SQL-MM 3: 8.2.6, 8.3.5



此函数支持 3d 并且不会丢失 z-index。

### 示例

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

### 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。用于多面体表面是否闭合（空心）。

## Synopsis

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

### 描述

如果 LINESTRING 的起点和终点重合，返回 TRUE。用于多面体曲面，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 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。

✔ 函数支持 3d 并且不会失 z-index。

✔ 此方法支持形字符串和曲面。

增：2.0.0 引入了多面体曲面的支持。

✔ 函数支持多面体曲面。

#### 串和点的示例

```

postgis=# SELECT ST_IsClosed('LINESTRING(0 0, 1 1)::geometry);
 st_isclosed
-----
 f
(1 row)

postgis=# SELECT ST_IsClosed('LINESTRING(0 0, 0 1, 1 1, 0 0)::geometry);
 st_isclosed
-----
 t
(1 row)

postgis=# SELECT ST_IsClosed('MULTILINESTRING((0 0, 0 1, 1 1, 0 0),(0 0, 1 1))::geometry);
 st_isclosed
-----
 f
(1 row)

postgis=# SELECT ST_IsClosed('POINT(0 0)::geometry);
 st_isclosed
-----
 t
(1 row)

postgis=# SELECT ST_IsClosed('MULTIPOINT((0 0), (1 1))::geometry);
 st_isclosed
-----
 t
(1 row)

```

#### 多面体曲面示例

```

-- A cube --
SELECT ST_IsClosed(ST_GeomFromEWKT('POLYHEDRALSURFACE( ((0 0 0, 0 0 1, 0 1 ←
1, 0 1 0, 0 0 0)),
((0 0 0, 0 1 0, 1 1 0, 1 0 0, 0 0 0)), ((0 0 0, 1 0 0, 1 0 1, 0 0 1, 0 0 0) ←
),
((1 1 0, 1 1 1, 1 0 1, 1 0 0, 1 1 0)),
((0 1 0, 0 1 1, 1 1 1, 1 1 0, 0 1 0)), ((0 0 1, 1 0 1, 1 1 1, 0 1 1, 0 0 1) ←
) )'));

 st_isclosed
-----
 t

-- Same as cube but missing a side --

```



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

st_isclosed
-----
f
```

相关信息

[ST\\_IsRing](#)

## 7.4.20 ST\_IsCollection

ST\_IsCollection — 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型，返回 **TRUE**。集合型有以下几种：

- 几何集合
- 多点、多多边形、多串、多曲面、多曲面
- 复合曲面



### Note

函数分析几何形状的类型。意味着它将在空集合或包含个元素的集合上返回 **TRUE**。

✓ 函数支持 3d 并且不会失 z-index。

✓ 此方法支持形字符串和曲面。

### 示例

```
postgis=# SELECT ST_IsCollection('LINESTRING(0 0, 1 1)::geometry);
st_iscollection
-----
f
(1 row)
```

```

postgis=# SELECT ST_IsCollection('MULTIPOINT EMPTY')::geometry);
 st_iscollection
-----
t
(1 row)

postgis=# SELECT ST_IsCollection('MULTIPOINT((0 0))')::geometry);
 st_iscollection
-----
t
(1 row)

postgis=# SELECT ST_IsCollection('MULTIPOINT((0 0), (42 42))')::geometry);
 st_iscollection
-----
t
(1 row)

postgis=# SELECT ST_IsCollection('GEOMETRYCOLLECTION(POINT(0 0))')::geometry);
 st_iscollection
-----
t
(1 row)

```

相关信息

[ST\\_NumGeometries](#)

### 7.4.21 ST\_IsEmpty

ST\_IsEmpty — 几何形是否空。

#### Synopsis

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

#### 描述

如果几何形空几何，返回 true; 如果 true，此几何形是空几何集合、多形、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



方法符合了 SQL/MM 规范。 SQL-MM 3: 5.1.7



此方法支持形字符串和曲线。



### Warning

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⊠它更符合 SQL/MM ⊠准

### 示例

```
SELECT ST_IsEmpty(ST_GeomFromText('GEOMETRYCOLLECTION EMPTY'));
 st_isempty
-----
 t
(1 row)

SELECT ST_IsEmpty(ST_GeomFromText('POLYGON EMPTY'));
 st_isempty
-----
 t
(1 row)

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

 st_isempty
-----
 f
(1 row)

SELECT ST_IsEmpty(ST_GeomFromText('POLYGON((1 2, 3 4, 5 6, 1 2))')) = false;
?column?
-----
 t
(1 row)

SELECT ST_IsEmpty(ST_GeomFromText('CIRCULARSTRING EMPTY'));
 st_isempty
-----
 t
(1 row)
```

## 7.4.22 ST\_IsPolygonCCW

ST\_IsPolygonCCW — ⊠⊠多⊠形是否具有逆⊠⊠方向的外⊠和⊠⊠⊠方向的内⊠。

### Synopsis

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

### 描述

如果⊠入几何体的所有多⊠形元素⊠其外⊠使用逆⊠⊠方向，⊠所有内⊠使用⊠⊠⊠方向，⊠返回 true。

如果几何体没有多⊠形元素，⊠返回 true。

**Note**

闭合串不被多边形元素。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转（即，有一个或多个内部以与外部相同的方向旋转），ST\_IsPolygonCW 和 ST\_IsPolygonCCW 都返回 FALSE。

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。

相关信息

[ST\\_ForcePolygonCW](#) , [ST\\_ForcePolygonCCW](#) , [ST\\_IsPolygonCW](#)

### 7.4.23 ST\_IsPolygonCW

ST\_IsPolygonCW — 多边形是否具有外部和逆时针内部。

#### 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，返回 true。

如果几何体没有多边形元素，返回 true。

**Note**

闭合串不被多边形元素。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转（即，有一个或多个内部以与外部相同的方向旋转），ST\_IsPolygonCW 和 ST\_IsPolygonCCW 都返回 FALSE。

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。

相关信息

[ST\\_ForcePolygonCW](#), [ST\\_ForcePolygonCCW](#), [ST\\_IsPolygonCW](#)

### 7.4.24 ST\_IsRing


`ST_IsRing` — 返回布尔值，如果几何体是闭合的且不自交的。

#### Synopsis

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

描述

返回 TRUE, 当 `LINestring` 同几何体是 `ST_IsClosed` (即 `ST_StartPoint(g) ~= ST_Endpoint(g)`) 和 `ST_IsSimple` (即不自交) 的。

-  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

```
SELECT ST_IsRing(geom), ST_IsClosed(geom), ST_IsSimple(geom)
FROM (SELECT 'LINestring(0 0, 0 1, 1 1, 1 0, 0 0)::geometry AS geom) AS foo;
 st_isring | st_isclosed | st_issimple
-----+-----+-----
t          | t           | t
(1 row)

SELECT ST_IsRing(geom), ST_IsClosed(geom), ST_IsSimple(geom)
FROM (SELECT 'LINestring(0 0, 0 1, 1 0, 1 1, 0 0)::geometry AS geom) AS foo;
 st_isring | st_isclosed | st_issimple
-----+-----+-----
f          | t           | f
(1 row)
```

相关信息

[ST\\_IsClosed](#), [ST\\_IsSimple](#), [ST\\_StartPoint](#), [ST\\_EndPoint](#)

### 7.4.25 ST\_IsSimple

`ST_IsSimple` — 返回布尔值，如果几何体是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），返回 true。有关 OGC 几何有效性和有效性定义的信息，参见“[确保几何形的 OpenGIS 合规性](#)”



#### 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



此方法符合了 SQL/MM 规范。SQL-MM 3: 5.1.8



此函数支持 3d 并且不会丢失 z-index。

### 示例

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

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

### 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⊠⊠，如果无效，⊠返回 NULL。⊠入必⊠⊠点。

**Note!****Note**

⊠⊠不是 OGC ⊠范的一部分，但在此⊠列出以完成点坐⊠提取器功能列表。



此方法⊠⊠了 **SQL 1.1 的 OGC ⊠⊠功能⊠范**。



⊠方法⊠⊠了 SQL/MM ⊠范。



⊠函数支持 3d 并且不会⊠失 z-index。

## 示例

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

## 相关信息

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

## 7.4.27 ST\_MemSize

ST\_MemSize — 返回几何⊠形占用的内存空⊠⊠量。

### Synopsis

```
integer ST_MemSize(geometry geomA);
```

## 描述

返回几何⊠形占用的内存空⊠⊠量（以字⊠⊠⊠位）。

⊠⊠充了 PostgreSQL 内置 [数据⊠⊠象函数](#) `pg_column_size`, `pg_size_pretty`, `pg_relation_size`, `pg_total_relation_size`

**Note!****Note**

⊠出表的字⊠大小的 `pg_relation_size` 可能返回小于 ST\_MemSize 的字⊠大小。⊠是因⊠ `pg_relation_size` 不会添加 toasted 表⊠献，并且大型几何⊠形存⊠在 TOAST 表中。

`pg_total_relation_size` - 包括表、⊠⊠表和索引。

`pg_column_size` 返回考⊠到⊠⊠，几何⊠形在列中占用多少空⊠，因此可能低于 ST\_MemSize

✔ 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)。

更改：2.2.0 名称更改 ST\_MemSize 以遵循命名定。

示例

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

totgeomsum	bossum	perbos
1522 kB	30 kB	1.99

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

73

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

## 7.4.28 ST\_NDims

ST\_NDims — 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

```
SELECT ST_NDims(ST_GeomFromText('POINT(1 1)')) As d2point,
       ST_NDims(ST_GeomFromEWKT('POINT(1 1 2)')) As d3point,
       ST_NDims(ST_GeomFromEWKT('POINTM(1 1 0.5)')) As d2pointm;

d2point | d3point | d2pointm
-----+-----+-----
      2 |       3 |       3
```

相关信息

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

## 7.4.29 ST\_NPoints

ST\_NPoints — 返回几何形状中的点数 (点)。

### Synopsis

```
integer ST_NPoints(geometry g1);
```

描述

返回几何形状中的点数。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，此函数在与包含曲线的几何形状一起使用时崩溃。此问题已在 1.3.4 及更高版本中得到修正。

- 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。

示例

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

--Polygon in 3D space
SELECT ST_NPoints(ST_GeomFromEWKT('LINESTRING(77.29 29.07 1,77.42 29.26 0,77.27 29.31
-1,77.29 29.07 3)'));
--result
4
```

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，返回环数。与 NumInteriorRings 不同，它不算外环。



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

```
SELECT ST_NRings(geom) As Nrings, ST_NumInteriorRings(geom) As ninterrings
      FROM (SELECT ST_GeomFromText('POLYGON((1 2, 3 4, 5 6, 1 2))') As geom) As foo;
```

	nrings	ninterrings
(1 row)	1	0

相关信息

[ST\\_NumInteriorRings](#)

### 7.4.31 ST\_NumGeometries

ST\_NumGeometries — 返回几何集中的元素数量。

#### Synopsis

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

## 描述

`ST_NumGeometries` 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。

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。

更改：2.0.0 在之前的版本中，如果几何体不是 collection/MULTI 型，它会返回 NULL。2.0.0 在对于单个几何体返回 1，例如 POLYGON、LINESTRING、POINT。



方法符合了 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

```
--Prior versions would have returned NULL for this -- in 2.0.0 this returns 1
SELECT ST_NumGeometries(ST_GeomFromText('LINESTRING(77.29 29.07,77.42 29.26,77.27 29.31,77.29 29.07)'));
--result
1

--Geometry Collection Example - multis count as one geom in a collection
SELECT ST_NumGeometries(ST_GeomFromEWKT('GEOMETRYCOLLECTION(MULTIPOINT((-2 3),(-2 2)),
LINESTRING(5 5 ,10 10),
POLYGON((-7 4.2,-7.1 5,-7.1 4.3,-7 4.2))'));
--result
3
```

## 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。



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

```
--If you have a regular polygon
SELECT gid, field1, field2, ST_NumInteriorRings(geom) AS numholes
FROM sometable;

--If you have multipolygons
--And you want to know the total number of interior rings in the MULTIPOLYGON
SELECT gid, field1, field2, SUM(ST_NumInteriorRings(geom)) AS numholes
FROM (SELECT gid, field1, field2, (ST_Dump(geom)).geom As geom
      FROM sometable) As foo
GROUP BY gid, field1,field2;
```

相关信息

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

### 7.4.33 ST\_NumInteriorRing

`ST_NumInteriorRing` — 返回多面形的内环（孔）数。`ST_NumInteriorRings` 的别名

#### Synopsis

```
integer ST_NumInteriorRing(geometry a_polygon);
```

相关信息

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

### 7.4.34 ST\_NumPatches

`ST_NumPatches` — 返回多面体曲面上的面数。对于非多面体几何形状将返回 `null`。

#### Synopsis

```
integer ST_NumPatches(geometry g1);
```

描述

返回多面体曲面上的面数。对于非多面体几何形状将返回 `null`。是 `ST_NumGeometries` 的别名，用于支持 MM 命名。如果您不关心 MM 约定，使用 `ST_NumGeometries` 会更快。

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

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

## 相关信息

[ST\\_GeomFromEWKT](#), [ST\\_NumGeometries](#)

### 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，是 ST\_NPoints 的别名，它不返回字符串的点数，请使用 ST\_NPoints，它是多用途的并且适用于多几何型。



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: 7.2.4

## 示例

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

## 相关信息

[ST\\_NPoints](#)

### 7.4.36 ST\_PatchN

ST\_PatchN — 返回多面形曲面 (PolyhedralSurface) 的第 N 个几何体 (面)。

## Synopsis

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

### 描述

如果几何体是 POLYHEDRALSURFACE 或 POLYHEDRALSURFACEM, 返回从 1 开始的第 N 个几何体 (面)。否则, 返回 NULL。将返回与多面体曲面的 ST\_GeometryN 相同的答案。使用 ST\_GeometryN 速度更快。



#### Note

索引从 1 开始。



#### Note

如果要提取几何体的所有元素, 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。

### 示例

```
--Extract the 2nd face of the polyhedral surface
SELECT ST_AsEWKT(ST_PatchN(geom, 2)) As geomewkt
FROM (
VALUES (ST_GeomFromEWKT('POLYHEDRALSURFACE( ((0 0 0, 0 0 1, 0 1 1, 0 1 0, 0 0 0)),
        ((0 0 0, 0 1 0, 1 1 0, 1 0 0, 0 0 0)), ((0 0 0, 1 0 0, 1 0 1, 0 0 1, 0 0 0)),
        ((1 1 0, 1 1 1, 1 0 1, 1 0 0, 1 1 0)),
        ((0 1 0, 0 1 1, 1 1 1, 1 1 0, 0 1 0)), ((0 0 1, 1 0 1, 1 1 1, 0 1 1, 0 0 1)) )' ) ) ←
        As foo(geom);

          geomewkt
-----+-----
POLYGON((0 0 0,0 1 0,1 1 0,1 0 0,0 0 0))
```

### 相关信息

[ST\\_AsEWKT](#), [ST\\_GeomFromEWKT](#), [ST\\_Dump](#), [ST\\_GeometryN](#), [ST\\_NumGeometries](#)

## 7.4.37 ST\_PointN

ST\_PointN — 返回几何体中第一个点串或面串中的第 N 个点。

## Synopsis

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

### 描述

返回几何图形中第  $n$  个字符串或图形字符串中的第  $N$  个点。索引从 `LineString` 末尾开始向后数，因此 `-1` 是最后一个点。如果几何图形中没有字符串，返回 `NULL`。



#### Note

自版本 0.8.0 以来，OGC 规范的索引从 1 开始。OGC 中没有向后索引（负索引），以前的版本将其索引基于 0。



#### Note

如果要提取 `MultiLineString` 中每个 `LineString` 的第  $N$  个点，与 `ST_Dump` 合使用



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: 7.2.5, 7.3.5



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。



#### Note

更改：2.0.0 不再适用于几何 `multilinestrings`。在旧版本的 PostGIS 中——一行 `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

```
-- Extract all POINTs from a LINESTRING
SELECT ST_AsText(
  ST_PointN(
    column1,
    generate_series(1, ST_NPoints(column1))
  ))
FROM ( VALUES ('LINESTRING(0 0, 1 1, 2 2)::geometry) ) AS foo;

 st_astext
-----
POINT(0 0)
POINT(1 1)
POINT(2 2)
(3 rows)

--Example circular string
SELECT ST_AsText(ST_PointN(ST_GeomFromText('CIRCULARSTRING(1 2, 3 2, 1 2)'), 2));
```

```

st_astext
-----
POINT(3 2)
(1 row)

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

st_astext
-----
POINT Z (1 1 1)
(1 row)

```

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

```

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

-- result
MULTIPOINT Z ((30 10 4),(10 30 5),(40 40 6),(30 10 4))

```

相关信息

[ST\\_RemoveRepeatedPoints](#), [ST\\_DumpPoints](#)



### 7.4.39 ST\_StartPoint

ST\_StartPoint — 返回 LineString 的第一个点。

#### Synopsis

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

#### 描述

返回 LINESTRING 或 CIRCULARLINESTRING 几何图形的第一个点作 POINT。如果输入不是 LINESTRING 或 CIRCULARLINESTRING，返回 NULL。

- ✔ 方法遵循了 SQL/MM 规范。SQL-MM 3: 7.1.3
- ✔ 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。

#### Note



增：3.2.0 返回所有几何图形的点。如果输入不是 LineString，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，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

```
SELECT ST_AsText(ST_StartPoint('LINESTRING(0 1, 0 2)::geometry'));
 st_astext
-----
POINT(0 1)
```

##### 非 LineString 的起点返回 NULL

```
SELECT ST_StartPoint('POINT(0 1)::geometry') IS NULL AS is_null;
 is_null
-----
t
```

##### 3D LineString 起点

```
SELECT ST_AsEWKT(ST_StartPoint('LINESTRING(0 1 1, 0 2 2)::geometry'));
 st_asewkt
-----
POINT(0 1 1)
```

##### CircularString 起点

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

相关信息

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

## 7.4.40 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。

几何类型后方括号中显示的标志具有以下含义：

- M：具有 M 标志
- Z：具有 Z 标志
- B：有一个边界框
- G：大地坐标系（地理）
- S：具有空参考系



此方法支持几何字符串和曲线。



几何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

增强：在 2.0.0 中添加了地理支持

增强：添加了 S 标志以指示其是否具有 2.1.0 空参考系

增强：添加了 2.2.0 TIN 和曲线的支持

示例

```
=# SELECT ST_Summary(ST_GeomFromText('LINESTRING(0 0, 1 1)')) as geom,
           ST_Summary(ST_GeogFromText('POLYGON((0 0, 1 1, 1 2, 1 1, 0 0))')) geog;
-----+-----
geom          | geog
-----+-----
LineString[B] with 2 points | Polygon[BGS] with 1 rings
                        | ring 0 has 5 points
                        |
(1 row)

=# SELECT ST_Summary(ST_GeogFromText('LINESTRING(0 0 1, 1 1 1)')) As geog_line,
```

```

      ST_Summary(ST_GeomFromText('SRID=4326;POLYGON((0 0 1, 1 1 2, 1 2 3, 1 1 1, 0 0 1)) ←
        ') As geom_poly;
;
      geog_line          |          geom_poly
-----+-----
  LineString[ZBGS] with 2 points | Polygon[ZBS] with 1 rings
                                |   ring 0 has 5 points
                                |
(1 row)

```

相关信息

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

### 7.4.41 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，返回 NULL。输入必须是点。



#### Note

要获取几何图形的最小和最大 X 坐标，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

```

SELECT ST_X(ST_GeomFromEWKT('POINT(1 2 3 4)'));
 st_x
-----
      1
(1 row)

SELECT ST_Y(ST_Centroid(ST_GeomFromEWKT('LINESTRING(1 2 3 4, 1 1 1 1)')));
 st_y
-----
    1.5
(1 row)

```

相关信息

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

### 7.4.42 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 坐标，请使用 [ST\\_YMin](#) 和 [ST\\_YMax](#) 函数。



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: 6.1.4



此函数支持 3d 并且不会丢失 z-index。

示例

```
SELECT ST_Y(ST_GeomFromEWKT('POINT(1 2 3 4)'));
 st_y
-----
      2
(1 row)

SELECT ST_Y(ST_Centroid(ST_GeomFromEWKT('LINESTRING(1 2 3 4, 1 1 1 1)')));
 st_y
-----
   1.5
(1 row)
```

相关信息

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

### 7.4.43 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，请使用 `ST_ZMin` 和 `ST_ZMax` 函数。



该方法符合 SQL/MM 规范。



函数支持 3d 并且不会丢失 z-index。

### 示例

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

### 相关信息

`ST_GeomFromEWKT`, `ST_M`, `ST_X`, `ST_Y`, `ST_ZMax`, `ST_ZMin`

## 7.4.44 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。



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

```
SELECT ST_Zmflag(ST_GeomFromEWKT('LINESTRING(1 2, 3 4)'));
st_zmflag
-----
          0

SELECT ST_Zmflag(ST_GeomFromEWKT('LINESTRINGM(1 2 3, 3 4 3)'));
st_zmflag
-----
          1

SELECT ST_Zmflag(ST_GeomFromEWKT('CIRCULARSTRING(1 2 3, 3 4 3, 5 6 3)'));
st_zmflag
-----
          2

SELECT ST_Zmflag(ST_GeomFromEWKT('POINT(1 2 3 4)'));
st_zmflag
-----
          3
```

相关信息

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

### 7.4.45 ST\_HasZ

ST\_HasZ — 返回几何体是否具有 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



该函数支持 3d 并且不会丢失 z-index。



该功能支持 M 坐标。

示例

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

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

相关信息

[ST\\_Zmflag](#)

[ST\\_HasM](#)

## 7.4.46 ST\_HasM

ST\_HasM — 返回几何体是否具有 M (度量) 度。

### Synopsis

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


描述

返回几何体是否具有 M (度量) 度并返回布尔值。如果几何体具有 M 度，返回 true；否则，返回 false。具有 M 度的几何体通常表示与空特征相关的度量或附加数据。

此函数用于确定几何体是否包含度量信息非常有用。

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

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

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

相关信息

[ST\\_Zmflag](#)

[ST\\_HasZ](#)

## 7.5 几何构造器

### 7.5.1 ST\_AddPoint

ST\_AddPoint — 将点添加到线串 (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

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

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

对于表中那些未闭合的线，通过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。

```
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，返回包含指定类型元素的多几何形。

### Synopsis

```
geometry ST_CollectionExtract(geometry collection);
geometry ST_CollectionExtract(geometry collection, integer type);
```

### 描述

给定一个几何集合，返回同种类型多几何。

如果未指定类型，返回包含最高维度几何的多几何。因此，多边形先于线，线先于点。

如果指定了类型，返回包含指定类型的多几何。如果没有指定类型的元素，返回 **EMPTY** 几何形。支持点、线和面。类型号如下：



- 1 == POINT
- 2 == LINESTRING
- 3 == POLYGON

对于基本几何输入，如果输入类型与请求的类型匹配，几何形状将保持不变返回。否则，结果是指定类型的空几何形状。如果需要，可以使用 [ST\\_Multi](#) 将它转换为多几何形状。



### Warning

不要使用 [MultiPolygon](#) 结果的有效性。如果多边形组件相交或重叠，结果将无效。（例如，将此函数用于 [ST\\_Split](#) 结果可能会产生这种情况。）可以使用 [ST\\_IsValid](#) 检查这种情况并使用 [ST\\_MakeValid](#) 修复。

可用性：1.1.0



### Note

在 1.5.3 之前，无论类型如何，函数都会返回同种类型的基本几何输入。在 1.5.3 中，不匹配的几何形状返回 NULL 结果。在 2.0.0 中，不匹配的几何形状返回请求类型的 EMPTY 结果。

### 示例

提取最大维度类型：

```
SELECT ST_AsText(ST_CollectionExtract(
  'GEOMETRYCOLLECTION( POINT(0 0), LINESTRING(1 1, 2 2) )'));
 st_astext
-----
MULTILINESTRING((1 1, 2 2))
```

提取点 (type 1 == POINT)：

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

提取线 (type 2 == LINESTRING)：

```
SELECT ST_AsText(ST_CollectionExtract(
  'GEOMETRYCOLLECTION(GEOMETRYCOLLECTION(LINESTRING(0 0, 1 1)),LINESTRING(2 2, 3 3)) ←
  ',
  2 ));
 st_astext
-----
MULTILINESTRING((0 0, 1 1), (2 2, 3 3))
```

### 相关信息

[ST\\_CollectionHomogenize](#), [ST\\_Multi](#), [ST\\_IsValid](#), [ST\\_MakeValid](#)

### 7.5.3 ST\_CollectionHomogenize

ST\_CollectionHomogenize — 返回几何集合的最简表示。

#### Synopsis

```
geometry ST_CollectionHomogenize(geometry collection);
```

#### 描述

给定几何集合，它返回“最简”的表示形式。

- 同类型集合将以适当的多几何体形式返回。
- 混合类型集合将返回多个平面几何集合。
- 包含多个基本元素的集合将作多个元素返回。
- 基本几何形状返回不简。如果需要，可以使用 [ST\\_Multi](#) 将它转换为多几何体。



#### Warning

该函数不保证结果有效。特别是，包含相交或重叠多边形的集合将创建无效的多边形。这种情况可以使用 [ST\\_IsValid](#) 检查并使用 [ST\\_MakeValid](#) 修复。

可用性: 2.0.0

#### 示例

一个元素集合返回基本集合

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

 st_astext
-----
POINT(0 0)
```

将嵌套的一个元素集合返回基本几何：

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

 st_astext
-----
POINT(0 0)
```

集合返回多几何体：

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

 st_astext
-----
MULTIPOINT((0 0),(1 1))
```

将嵌套混合类型集合返回平面几何集合：

```
SELECT ST_AsText(ST_CollectionHomogenize('GEOMETRYCOLLECTION(POINT(0 0), GEOMETRYCOLLECTION ←
  ( LINESTRING(1 1, 2 2))'));
      st_astext
      -----
      GEOMETRYCOLLECTION(POINT(0 0),LINESTRING(1 1,2 2))
```

将多边形集合（无效的）多多边形：

```
SELECT ST_AsText(ST_CollectionHomogenize('GEOMETRYCOLLECTION (POLYGON ((10 50, 50 50, 50 ←
  10, 10 50)), POLYGON ((90 50, 90 10, 50 10, 50 50, 90 50))'));
      st_astext
      -----
      MULTIPOLYGON(((10 50,50 50,50 10,10 10,10 50)),((90 50,90 10,50 10,50 50,90 50)))
```

相关信息

[ST\\_CollectionExtract](#), [ST\\_Multi](#), [ST\\_IsValid](#), [ST\\_MakeValid](#)

## 7.5.4 ST\_CurveToLine

ST\_CurveToLine — 将包含曲线的几何图形转换为线性几何图形。

### Synopsis

geometry **ST\_CurveToLine**(geometry curveGeom, float tolerance, integer tolerance\_type, integer flags);

### 描述

将 CIRCULAR STRING 通常 LINESTRING，或将 CURVEPOLYGON 或 POLYGON，或将 MULTISURFACE 或 MULTIPOLYGON。出于出到不支持 CIRCULARSTRING 几何类型的很有用

将给定几何图形转换为线性几何图形。使用给定的“公差”和（每个象限（一个弧的四个部分）32 个弧段，默认情况下没有弧段）将每个弯曲几何图形或弧段转换为线性近似。

“tolerance\_type” 参数决定 “tolerance” 参数的解释。它可以采用以下：

- 0（默认）：公差是四分之一弧的最大弧数。
- 1：公差是从曲线到直线的最大差值。单位是输入几何的单位。
- 2：公差是由半径形成的角度的最大弧（以弧度为单位）。

‘flags’ 参数是一个位字段。默认为 0。支持的位有：

- 1：名称（与方向无关）输出。
- 2：保留角度，避免在生成名称输出时减少角度（弧段度），名称标志无效。

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

```
SELECT ST_AsText(ST_CurveToLine(ST_GeomFromText('CIRCULARSTRING(220268 150415,220227 150505,220227 150406)')));
```

```
--Result --
LINESTRING(220268 150415,220269.95064912 150416.539364228,220271.823415575 150418.17258804,220273.613787707 150419.895736857,
220275.317452352 150421.704659462,220276.930305234 150423.594998003,220278.448460847 150425.562198489,
220279.868261823 150427.60152176,220281.186287736 150429.708054909,220282.399363347 150431.876723113,
220283.50456625 150434.10230186,220284.499233914 150436.379429536,220285.380970099 150438.702620341,220286.147650624 150441.066277505,
220286.797428488 150443.464706771,220287.328738321 150445.892130112,220287.740300149 150448.342699654,
220288.031122486 150450.810511759,220288.200504713 150453.289621251,220288.248038775 150455.77405574,
220288.173610157 150458.257830005,220287.977398166 150460.734960415,220287.659875492 150463.199479347,
220287.221807076 150465.64544956,220286.664248262 150468.066978495,220285.988542259 150470.458232479,220285.196316903 150472.81345077,
220284.289480732 150475.126959442,220283.270218395 150477.39318505,220282.140985384 150479.606668057,
220280.90450212 150481.762075989,220279.5637474 150483.85421628,220278.12195122 150485.87804878,
220276.582586992 150487.828697901,220274.949363179 150489.701464356,220273.226214362 150491.491836488,
220271.417291757 150493.195501133,220269.526953216 150494.808354014,220267.559752731 150496.326509628,
220265.520429459 150497.746310603,220263.41389631 150499.064336517,220261.245228106 150500.277412127,
220259.019649359 150501.38261503,220256.742521683 150502.377282695,220254.419330878 150503.259018879,
220252.055673714 150504.025699404,220249.657244448 150504.675477269,220247.229821107 150505.206787101,
220244.779251566 150505.61834893,220242.311439461 150505.909171266,220239.832329968 150506.078553494,
220237.347895479 150506.126087555,220234.864121215 150506.051658938,220232.386990804 150505.855446946,
220229.922471872 150505.537924272,220227.47650166 150505.099855856,220225.054972724 150504.542297043,
220222.663718741 150503.86659104,220220.308500449 150503.074365683,
220217.994991777 150502.167529512,220215.72876617 150501.148267175,
220213.515283163 150500.019034164,220211.35987523 150498.7825509,
220209.267734939 150497.441796181,220207.243902439 150496,
```

```

220205.293253319 150494.460635772,220203.420486864 150492.82741196,220201.630114732 ↔
  150491.104263143,
220199.926450087 150489.295340538,220198.313597205 150487.405001997,220196.795441592 ↔
  150485.437801511,
220195.375640616 150483.39847824,220194.057614703 150481.291945091,220192.844539092 ↔
  150479.123276887,220191.739336189 150476.89769814,
220190.744668525 150474.620570464,220189.86293234 150472.297379659,220189.096251815 ↔
  150469.933722495,
220188.446473951 150467.535293229,220187.915164118 150465.107869888,220187.50360229 ↔
  150462.657300346,
220187.212779953 150460.189488241,220187.043397726 150457.710378749,220186.995863664 ↔
  150455.22594426,
220187.070292282 150452.742169995,220187.266504273 150450.265039585,220187.584026947 ↔
  150447.800520653,
220188.022095363 150445.35455044,220188.579654177 150442.933021505,220189.25536018 ↔
  150440.541767521,
220190.047585536 150438.18654923,220190.954421707 150435.873040558,220191.973684044 ↔
  150433.60681495,
220193.102917055 150431.393331943,220194.339400319 150429.237924011,220195.680155039 ↔
  150427.14578372,220197.12195122 150425.12195122,
220198.661315447 150423.171302099,220200.29453926 150421.298535644,220202.017688077 ↔
  150419.508163512,220203.826610682 150417.804498867,
220205.716949223 150416.191645986,220207.684149708 150414.673490372,220209.72347298 ↔
  150413.253689397,220211.830006129 150411.935663483,
220213.998674333 150410.722587873,220216.22425308 150409.61738497,220218.501380756 ↔
  150408.622717305,220220.824571561 150407.740981121,
220223.188228725 150406.974300596,220225.586657991 150406.324522731,220227 150406)

--3d example
SELECT ST_AsEWKT(ST_CurveToLine(ST_GeomFromEWKT('CIRCULARSTRING(220268 150415 1,220227 ↔
  150505 2,220227 150406 3)')));
Output
-----
LINESTRING(220268 150415 1,220269.95064912 150416.539364228 1.0181172856673,
220271.823415575 150418.17258804 1.03623457133459,220273.613787707 150419.895736857 ↔
  1.05435185700189,...AD INFINITUM ....
  220225.586657991 150406.324522731 1.32611114201132,220227 150406 3)

--use only 2 segments to approximate quarter circle
SELECT ST_AsText(ST_CurveToLine(ST_GeomFromText('CIRCULARSTRING(220268 150415,220227 ↔
  150505,220227 150406)'),2));
st_astext
-----
LINESTRING(220268 150415,220287.740300149 150448.342699654,220278.12195122 ↔
  150485.87804878,
220244.779251566 150505.61834893,220207.243902439 150496,220187.50360229 150462.657300346,
220197.12195122 150425.12195122,220227 150406)

-- Ensure approximated line is no further than 20 units away from
-- original curve, and make the result direction-neutral
SELECT ST_AsText(ST_CurveToLine(
  'CIRCULARSTRING(0 0,100 -100,200 0)::geometry,
  20, -- Tolerance
  1, -- Above is max distance between curve and line
  1 -- Symmetric flag
));
st_astext
-----
LINESTRING(0 0,50 -86.6025403784438,150 -86.6025403784439,200 -1.1331077795296e-13,200 0)

```

相关信息

[ST\\_LineToCurve](#)

### 7.5.5 ST\_Scroll

ST\_Scroll — 更改☐合☐串的起点。

#### Synopsis

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

#### 描述

将☐合☐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

使 e ☐合☐从其第三个☐点开始

```
SELECT ST_AsEWKT(ST_Scroll('SRID=4326;LINESTRING(0 0 0 1, 10 0 2 0, 5 5 4 2,0 0 0 1)', '↔
POINT(5 5 4 2)'));
```

```
st_asewkt
-----
SRID=4326;LINESTRING(5 5 4 2,0 0 0 1,10 0 2 0,5 5 4 2)
```

相关信息

[ST\\_Normalize](#)

### 7.5.6 ST\_FlipCoordinates

ST\_FlipCoordinates — 返回 X ☐和 Y ☐☐☐的几何☐形版本。

#### Synopsis

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

## 描述

返回给定几何形的 X 和 Y 坐标的版本。对于修改包含以纬度/经度 (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

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

## 相关信息

[ST\\_SwapOrdinates](#)

## 7.5.7 ST\_Force2D

ST\_Force2D — 强制几何形进入“二维模式”。

### Synopsis

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

## 描述

强制几何形进入“二维模式”，以便所有输出表示都具有 X 和 Y 坐标。对于强制 OGC 兼容输出非常有用（因为 OGC 指定二维几何形状）。

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\_2D。

- ✔ 此方法支持形字符串和曲线。
- ✔ 函数支持多面体曲面。
- ✔ 函数支持 3d 并且不会丢失 z-index。

示例

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

SELECT ST_AsEWKT(ST_Force2D('POLYGON((0 0 2,0 5 2,5 0 2,0 0 2),(1 1 2,3 1 2,1 3 2,1 1 2))'));
           st_asewkt
-----
POLYGON((0 0,0 5,5 0,0 0),(1 1,3 1,1 3,1 1))
```

相关信息

[ST\\_Force3D](#)

## 7.5.8 ST\_Force3D

ST\_Force3D — 强制几何体形入 XYZ 模式。是 ST\_Force3DZ 的同名。

### Synopsis

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




描述

强制几何体形入 XYZ 模式。是 ST\_Force3DZ 的同名。如果几何体没有 Z 分量，会附加 z 的 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. 您可以在指定一个非零 Z 值。

-  函数支持多面体曲面。
-  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。

示例

```
--Nothing happens to an already 3D geometry
SELECT ST_AsEWKT(ST_Force3D(ST_GeomFromEWKT('CIRCULARSTRING(1 1 2, 2 3 2, 4 5 2, 6 7 2, 5 6 2)')));
           st_asewkt
-----
CIRCULARSTRING(1 1 2,2 3 2,4 5 2,6 7 2,5 6 2)
```



```
SELECT ST_AsEWKT(ST_Force3D('POLYGON((0 0,0 5,5 0,0 0),(1 1,3 1,1 3,1 1))'));
-----
st_asewkt
-----
POLYGON((0 0 0,0 5 0,5 0 0,0 0 0),(1 1 0,3 1 0,1 3 0,1 1 0))
```

相关信息

[ST\\_AsEWKT](#), [ST\\_Force2D](#), [ST\\_Force3DM](#), [ST\\_Force3DZ](#)

## 7.5.9 ST\_Force3DZ

ST\_Force3DZ — 强制几何图形入 XYZ 模式。

### Synopsis

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




### 描述

强制几何图形入 XYZ 模式。如果几何体没有 Z 分量，会附加 z 或 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。您可以在指定一个非零 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 WKT 字符串和曲线。

### 示例

```
--Nothing happens to an already 3D geometry
SELECT ST_AsEWKT(ST_Force3DZ(ST_GeomFromEWKT('CIRCULARSTRING(1 1 2, 2 3 2, 4 5 2, 6 7 2, 5 6 2)')));
-----
st_asewkt
-----
CIRCULARSTRING(1 1 2,2 3 2,4 5 2,6 7 2,5 6 2)

SELECT ST_AsEWKT(ST_Force3DZ('POLYGON((0 0,0 5,5 0,0 0),(1 1,3 1,1 3,1 1))'));
-----
st_asewkt
-----
POLYGON((0 0 0,0 5 0,5 0 0,0 0 0),(1 1 0,3 1 0,1 3 0,1 1 0))
```

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 M 坐标。如果它有 Z 分量，Z 被移除更改：2.1.0。在 2.0.x 期间，它被称为 ST\_Force\_3DM。

更改：3.1.0。现在可以指定一个非零 M。



此方法支持形字符串和曲线。

### 示例

```
--Nothing happens to an already 3D geometry
SELECT ST_AsEWKT(ST_Force3DM(ST_GeomFromEWKT('CIRCULARSTRING(1 1 2, 2 3 2, 4 5 2, 6 7 2, 5
6 2)')));
          st_asewkt
-----
CIRCULARSTRINGM(1 1 0,2 3 0,4 5 0,6 7 0,5 6 0)

SELECT ST_AsEWKT(ST_Force3DM('POLYGON((0 0 1,0 5 1,5 0 1,0 0 1),(1 1 1,3 1 1,1 3 1,1 1 1))
'));
          st_asewkt
-----
POLYGONM((0 0 0,0 5 0,5 0 0,0 0 0),(1 1 0,3 1 0,1 3 0,1 1 0))
```

### 相关信息

[ST\\_AsEWKT](#), [ST\\_Force2D](#), [ST\\_Force3DM](#), [ST\\_Force3D](#), [ST\\_GeomFromEWKT](#)

## 7.5.11 ST\_Force4D

ST\_Force4D — 强制几何体形入 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 期间，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 WKT 字符串和曲线。

## 示例

```
--Nothing happens to an already 3D geometry
SELECT ST_AsEWKT(ST_Force4D(ST_GeomFromEWKT('CIRCULARSTRING(1 1 2, 2 3 2, 4 5 2, 6 7 2, 5 6 2)')));

```

	st_asewkt
	CIRCULARSTRING(1 1 2 0,2 3 2 0,4 5 2 0,6 7 2 0,5 6 2 0)

```
SELECT ST_AsEWKT(ST_Force4D('MULTILINESTRINGM((0 0 1,0 5 2,5 0 3,0 0 4),(1 1 1,3 1 1,1 3 1,1 1 1))'));

```

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

## 相关信息

[ST\\_AsEWKT](#), [ST\\_Force2D](#), [ST\\_Force3DM](#), [ST\\_Force3D](#)

## 7.5.12 ST\_ForceCollection

ST\_ForceCollection — 将几何体强制为几何集合 (GEOMETRYCOLLECTION)。

### Synopsis

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

## 描述

将几何体强制为几何集合。对于 WKB 表示很有用。

增加：2.0.0 引入了多面体曲面的支持。

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。

- ✔ ☒ 函数支持多面体曲面。
- ✔ ☒ 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

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

st\_asewkt

-----

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

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

st\_astext

-----

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

(1 row)

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

st\_asewkt

-----

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

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☒形上升到其曲☒☒型。



#### Synopsis

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

## 描述

将几何体转换为其曲线表示形式（如果适用）：直线、复合曲线，多边形、多曲线，多边形、曲线多边形，多重多边形、多重曲面。如果几何体输入已经是曲线表示，则返回与输入相同的几何体。

可用性：2.2.0

-  函数支持 3d 并且不会丢失 z-index。
-  此方法支持曲线字符串和曲线。

## 示例

```
SELECT ST_AsText(
  ST_ForceCurve(
    'POLYGON((0 0 2, 5 0 2, 0 5 2, 0 0 2),(1 1 2, 1 3 2, 3 1 2, 1 1 2))'::geometry
  )
);
           st_astext
-----
CURVEPOLYGON Z ((0 0 2,5 0 2,0 5 2,0 0 2),(1 1 2,1 3 2,3 1 2,1 1 2))
(1 row)
```

## 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

强制（多）多边形使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。

## 相关信息

[ST\\_ForcePolygonCW](#) , [ST\\_IsPolygonCCW](#) , [ST\\_IsPolygonCW](#)

### 7.5.15 ST\_ForcePolygonCW

ST\_ForcePolygonCW — 强制所有外环按顺时针方向，所有内环按逆时针方向。

#### 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。

#### 相关信息

[ST\\_ForcePolygonCCW](#) , [ST\\_IsPolygonCCW](#) , [ST\\_IsPolygonCW](#)

### 7.5.16 ST\_ForceSFS

ST\_ForceSFS — 强制几何体使用 SFS 1.1 几何体类型。

#### Synopsis

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

#### 描述

- ✓ 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)。
- ✓ 此方法支持几何字符串和曲线。
- ✓ 函数支持 3d 并且不会丢失 z-index。

### 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

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](#)。

增强：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

```
SELECT ST_AsEWKT(
  ST_ForceRHR(
    'POLYGON((0 0 2, 5 0 2, 0 5 2, 0 0 2),(1 1 2, 1 3 2, 3 1 2, 1 1 2))'
  )
);
```

	st_asewkt
	POLYGON((0 0 2,0 5 2,5 0 2,0 0 2),(1 1 2,3 1 2,1 3 2,1 1 2))

(1 row)

## 相关信息

[ST\\_ForcePolygonCCW](#) , [ST\\_ForcePolygonCW](#) , [ST\\_IsPolygonCCW](#) , [ST\\_IsPolygonCW](#) , [ST\\_BuildArea](#), [ST\\_Polygonize](#), [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

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

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，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。



此方法支持多边形字符串和曲线。

### 示例

```
-- 2D Example
SELECT ST_AsText(ST_LineToCurve(foo.geom)) As curvedastext,ST_AsText(foo.geom) As ←
      non_curvedastext
      FROM (SELECT ST_Buffer('POINT(1 3)::geometry, 3) As geom) As foo;
```

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



```

| 1.58527096604839 ↔
| 0.0576441587903094,1 ↔
| 0,
| 0.414729033951621 ↔
| 0.0576441587903077,-0.1480502
| 0.228361402466137,
| -0.666710699058802 ↔
| 0.505591163092361,-1.12132034
| 0.878679656440353,
| -1.49440883690763 ↔
| 1.33328930094119,-1.771638597
| 1.85194970290472
| --ETC-- ↔
| ,3.94235584120969 ↔
| 3.58527096604839,4 ↔
| 3))

--3D example
SELECT ST_AsText(ST_LineToCurve(geom)) As curved, ST_AsText(geom) AS not_curved
FROM (SELECT ST_Translate(ST_Force3D(ST_Boundary(ST_Buffer(ST_Point(1,3), 2,2))),0,0,3) AS
geom) AS foo;

-----+-----
          curved                               |          not_curved
-----+-----
CIRCULARSTRING Z (3 3 3,-1 2.999999999999999 3,3 3 3) | LINESTRING Z (3 3 3,2.4142135623731 ↔
1.58578643762691 3,1 1 3,                               | -0.414213562373092 1.5857864376269 ↔
3,-1 2.999999999999999 3,                               | 3,-0.414213562373101 4.41421356237309 ↔
3,                                                       | 3,
| 0.9999999999999991 5 ↔
3,2.41421356237309 4.4142135623731 ↔
3,3 3 3)

(1 row)

```

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

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

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。多边形中的点序、多边形中的序或复合几何中元素的序可能会更改。在大多数情况下，它用于比较目的（将预期结果与预期结果行比较）。

可用性：2.3.0

示例

```
SELECT ST_AsText(ST_Normalize(ST_GeomFromText(
  'GEOMETRYCOLLECTION(
    POINT(2 3),
    MULTILINESTRING((0 0, 1 1),(2 2, 3 3)),
    POLYGON(
      (0 10,0 0,10 0,10 10,0 10),
      (4 2,2 2,2 4,4 4,4 2),
      (6 8,8 8,8 6,6 6,6 8)
    )
  )'
)));
           st_astext
-----
GEOMETRYCOLLECTION(POLYGON((0 0,0 10,10 10,10 0,0 0),(6 6,8 6,8 8,6 8,6 6),(2 2,4 2,4 4,2  ←
  4,2 2)),MULTILINESTRING((2 2,3 3),(0 0,1 1)),POINT(2 3))
(1 row)
```

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算并沿地方位角算的点。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  $\pi$ （180 度）
- 正西方位角是  $3\pi/2$ （270 度）

支持方位角和大于  $2\pi$ （360 度）的。

可用性: 2.0.0

增: 2.4.0 允距离和非准化方位角。

增: 3.4.0 允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

```
SELECT ST_AsText(ST_Project('POINT(0 0)::geography, 100000, radians(45.0)));
-----
POINT(0.635231029125537 0.639472334729198)
```

### 相关信息

[ST\\_Azimuth](#), [ST\\_Distance](#), [PostgreSQL function radians\(\)](#)

## 7.5.23 ST\_QuantizeCoordinates

ST\_QuantizeCoordinates — 将坐的最低有效位置零

### Synopsis

```
geometry ST_QuantizeCoordinates ( geometry g , int prec_x , int prec_y , int prec_z , int prec_m );
```

描述

ST\_QuantizeCoordinates 确定表示小数点后具有指定位数的坐...所需的位数 (N), 然后将除 N 个最高有效位之外的所有位...置...零。生成的坐...仍将舍入...原始...但会提高可...性。如果几何列使用可...存...型, ...可能会...致磁...使用量...著...少。...功能允...在每个...度中指定不同的小数点后位数; 未指定的尺寸假定具有 x ...度的精度。...数被解...指小数点左...的数字 (即 prec\_x=-2 将保留最接近 100 的坐...).

ST\_QuantizeCoordinates 生成的坐...独立于包含...些坐...的几何...形以及...些坐...在几何...形中的相...位置。因此, 几何...形之...有的拓扑关系不会受到使用此函数的影...。当使用低于几何的固有精度的位数...用...函数..., ...函数可能会...生无效的几何。

可用性: 2.5.0

技...背景

PostGIS 将所有坐...存...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...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...的形式提供, 小数点后精度... 6 位 (因此...需要 9 位有效的...度数字和 8 位有效的...度数字。)

当有 15 位有效数字可用..., 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...式位来表示坐...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...可以将它...的...置...我...喜...的任何..., 并且最...仍然得到一个四舍五入到我...的...入...的数字。例如, ... 100.123456 可以由最接近 100.123456000000、100.123456000001 和 100.123456432199 的浮点数表示。所有...些都同...有效, 因... ST\_AsText(geom, 6) 将...任何...些...入返回相同的...果。由于我...可以将...些位...置...任何..., 因此 ST\_QuantizeCoordinates 将 22 个无意...位...置...零。...于...坐...序列, ...会...建...零...的模式, PostgreSQL 可以更有效地...模式。



Note

只有几何...形的磁...大小可能会受到 ST\_QuantizeCoordinates 的影...。ST\_MemSize 用于...告几何...形的内存使用情况, 无...几何...形使用的磁...空...如何, 都将返回相同的...。

示例

```
SELECT ST_AsText(ST_QuantizeCoordinates('POINT (100.123456 0) '::geometry, 4));
st_astext
-----
POINT(100.123455047607 0)
```

```
WITH test AS (SELECT 'POINT (123.456789123456 123.456789123456) '::geometry AS geom)
SELECT
  digits,
  encode(ST_QuantizeCoordinates(geom, digits), 'hex'),
  ST_AsText(ST_QuantizeCoordinates(geom, digits))
FROM test, generate_series(15, -15, -1) AS digits;
```

digits	encode	st_astext
15	01010000005f9a72083cdd5e405f9a72083cdd5e40	POINT(123.456789123456 123.456789123456) ←
14	01010000005f9a72083cdd5e405f9a72083cdd5e40	POINT(123.456789123456 123.456789123456) ←
13	01010000005f9a72083cdd5e405f9a72083cdd5e40	POINT(123.456789123456 123.456789123456) ←
12	01010000005c9a72083cdd5e405c9a72083cdd5e40	POINT(123.456789123456 123.456789123456) ←

```

11 | 0101000000409a72083cdd5e40409a72083cdd5e40 | POINT(123.456789123456 ←
    | 123.456789123456)
10 | 0101000000009a72083cdd5e40009a72083cdd5e40 | POINT(123.456789123455 ←
    | 123.456789123455)
9  | 0101000000009072083cdd5e40009072083cdd5e40 | POINT(123.456789123418 ←
    | 123.456789123418)
8  | 0101000000008072083cdd5e40008072083cdd5e40 | POINT(123.45678912336 ←
    | 123.45678912336)
7  | 010100000000070083cdd5e4000070083cdd5e40 | POINT(123.456789121032 ←
    | 123.456789121032)
6  | 0101000000000040083cdd5e40000040083cdd5e40 | POINT(123.456789076328 ←
    | 123.456789076328)
5  | 010100000000000083cdd5e4000000083cdd5e40 | POINT(123.456789016724 ←
    | 123.456789016724)
4  | 010100000000000003cdd5e4000000003cdd5e40 | POINT(123.456787109375 ←
    | 123.456787109375)
3  | 0101000000000000003cdd5e4000000003cdd5e40 | POINT(123.456787109375 ←
    | 123.456787109375)
2  | 01010000000000000038dd5e40000000038dd5e40 | POINT(123.45654296875 ←
    | 123.45654296875)
1  | 010100000000000000dd5e4000000000dd5e40 | POINT(123.453125 123.453125)
0  | 010100000000000000dc5e4000000000dc5e40 | POINT(123.4375 123.4375)
-1 | 010100000000000000c05e4000000000c05e40 | POINT(123 123)
-2 | 01010000000000000005e4000000000005e40 | POINT(120 120)
-3 | 0101000000000000000584000000000005840 | POINT(96 96)
-4 | 0101000000000000000584000000000005840 | POINT(96 96)
-5 | 0101000000000000000584000000000005840 | POINT(96 96)
-6 | 0101000000000000000584000000000005840 | POINT(96 96)
-7 | 0101000000000000000584000000000005840 | POINT(96 96)
-8 | 0101000000000000000584000000000005840 | POINT(96 96)
-9 | 0101000000000000000584000000000005840 | POINT(96 96)
-10 | 0101000000000000000584000000000005840 | POINT(96 96)
-11 | 0101000000000000000584000000000005840 | POINT(96 96)
-12 | 0101000000000000000584000000000005840 | POINT(96 96)
-13 | 0101000000000000000584000000000005840 | POINT(96 96)
-14 | 0101000000000000000584000000000005840 | POINT(96 96)
-15 | 0101000000000000000584000000000005840 | POINT(96 96)

```

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 并且不会丢失 z-index。

示例

通过删除合（）的端点来保证没有合。假设 geom 的类型是 LINESTRING

```
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图形。

### Synopsis

geometry **ST\_RemoveRepeatedPoints**(geometry geom, float8 tolerance);


描述


返回给定几何图形的版本，其中除了重复的点。函数处理（多）串、（多）多边形和多点，但可以使用任何类型的几何图形行用。GeometryCollections 的元素是独立的。LineStrings 的端点被保留。

如果提供了容差参数，彼此容差距离内的点将被重复。

增：3.2.0

可用性：2.2.0

 函数支持多面体曲面。

 函数支持 3d 并且不会丢失 z-index。

示例

```
SELECT ST_AsText( ST_RemoveRepeatedPoints( 'MULTIPOINT ((1 1), (2 2), (3 3), (2 2))' ));
-----
MULTIPOINT(1 1,2 2,3 3)
```

```
SELECT ST_AsText( ST_RemoveRepeatedPoints( 'LINESTRING (0 0, 0 0, 1 1, 0 0, 1 1, 2 2)' ));
-----
LINESTRING(0 0,1 1,0 0,1 1,2 2)
```

示例：集合元素是独立的。

```
SELECT ST_AsText( ST_RemoveRepeatedPoints( 'GEOMETRYCOLLECTION (LINESTRING (1 1, 2 2, 2 2, 3 3), POINT (4 4), POINT (4 4), POINT (5 5))' ));
-----
GEOMETRYCOLLECTION(LINESTRING(1 1,2 2,3 3),POINT(4 4),POINT(4 4),POINT(5 5))
```

示例: ☒除在容差距离内的重复点。

```
SELECT ST_AsText( ST_RemoveRepeatedPoints( 'LINESTRING (0 0, 0 0, 1 1, 5 5, 1 1, 2 2)', 2) ) ←
;
-----
LINESTRING(0 0,5 5,2 2)
```

相关信息

[ST\\_Simplify](#)

## 7.5.26 ST\_RemoveIrrelevantPointsForView

`ST_RemoveIrrelevantPointsForView` — Removes points that are irrelevant for rendering a specific rectangular view of a geometry.

### Synopsis

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

### 描述

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

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

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

In contrast to `ST_ClipByBox2D()` this function

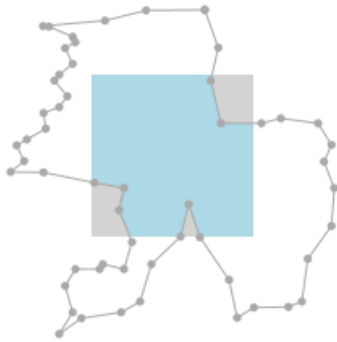
- sorts out points without computing new intersection points which avoids rounding errors and usually increases performance,
- returns a geometry with equal or similar point number,
- leads to the same rendering result within the specified view, and
- may introduce self-intersections which would make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

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

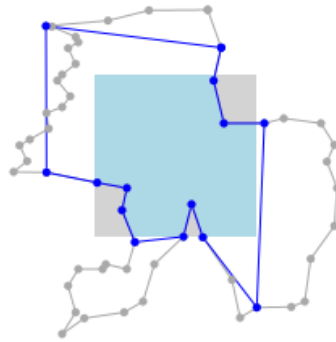
**Warning**

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

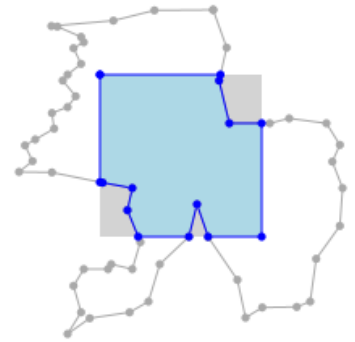
original  
55 points



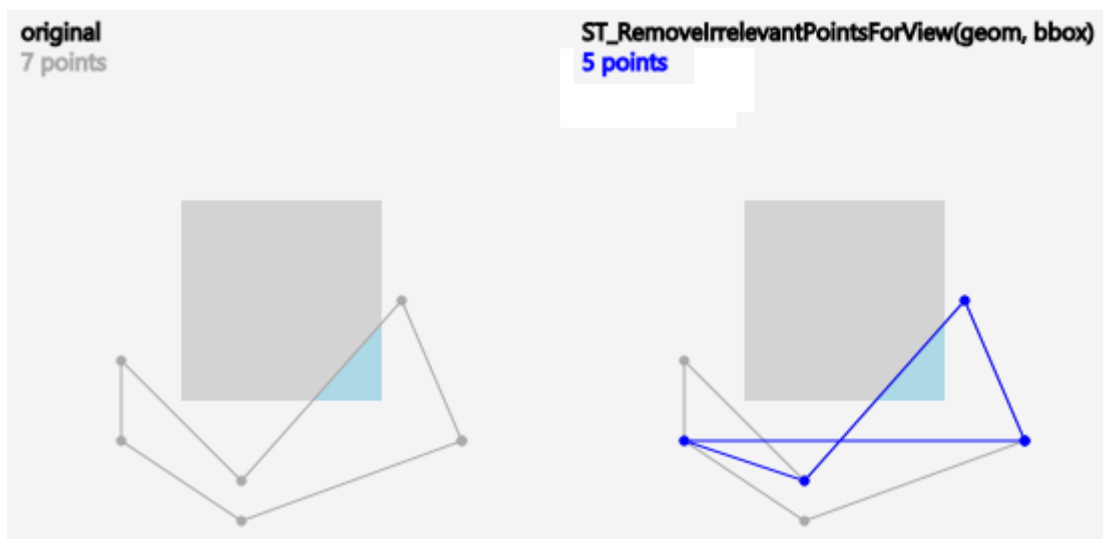
`ST_RemoveIrrelevantPointsForView(geom, bbox)`  
15 points



`ST_ClipByBox2D(geom, bbox)`  
15 points



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



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

可用性 : 3.5.0

示例

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



```

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

```

```

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

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

```

```

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

st_astext
-----
LINESTRING EMPTY

```

```

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

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

```

```

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

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

```

相关信息

[ST\\_ClipByBox2D](#), [ST\\_Intersection](#)

## 7.5.27 ST\_RemoveSmallParts

`ST_RemoveSmallParts` — Removes small parts (polygon rings or linestrings) of a geometry.

### Synopsis

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

## 描述

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

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

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

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

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

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

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

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

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

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



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

可用性 : 3.5.0

示例

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

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

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

    st_astext
    -----
    LINESTRING EMPTY
```

## 7.5.28 ST\_Reverse

ST\_Reverse — 返回☐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☐☐点的☐序。

增☐：2.4.0 引入了曲☐支持。



☐函数支持 3d 并且不会☐失 z-index。



☐函数支持多面体曲面。

示例

```
SELECT ST_AsText(geom) as line, ST_AsText(ST_Reverse(geom)) As reverseline
FROM
(SELECT ST_MakeLine(ST_Point(1,2),
    ST_Point(1,10)) As geom) as foo;
--result
    line          |   reverseline
-----+-----
LINESTRING(1 2,1 10) | LINESTRING(1 10,1 2)
```

## 7.5.29 ST\_Segmentize

ST\_Segmentize — 返回修改后的几何图形/地理，其段不于定距离。

### Synopsis

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

### 描述

返回修改后的几何图形/地理，其段长度不超过 max\_segment\_length。段长度以二算。段是被分成等的子段。

- 于几何形状，最大长度以空参考系的位置表示。
- 于地理来，最大长度以米位。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。它不会延比最大长度短的段。



#### 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长度，可能会无意中生种情况。

可用性：1.2.2

增：3.0.0 分段几何在可生成等的子分段

增：2.3.0 地理分段在可生成等的子分段

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)）

### 示例

分割一条。段均分，短段不平均。

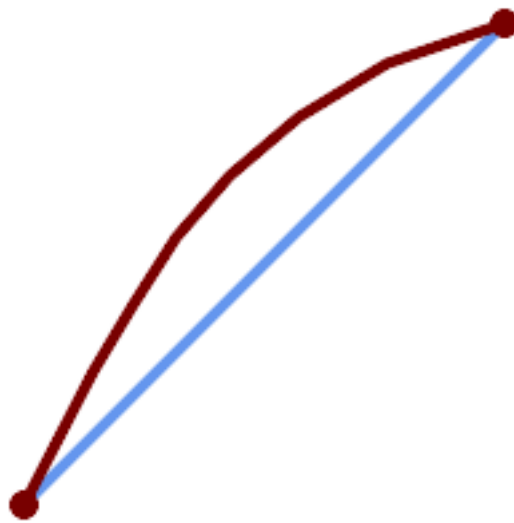
```
SELECT ST_AsText(ST_Segmentize(
  'MULTILINESTRING((0 0, 0 1, 0 9),(1 10, 1 18))'::geometry,
  5));
-----
MULTILINESTRING((0 0,0 1,0 5,0 9),(1 10,1 14,1 18))
```

多形分割：

```
SELECT ST_AsText(
    ST_Segmentize('POLYGON((0 0, 0 8, 30 0, 0 0))'::geometry), 10);
-----
POLYGON((0 0,0 8,7.5 6,15 4,22.5 2,30 0,20 0,10 0,0 0))
```

地理行分段，最大分段度 2000 公里。沿着接端点的大弧添加点。

```
SELECT ST_AsText(
    ST_Segmentize('LINESTRING (0 0, 60 60)'::geography), 2000000);
-----
LINESTRING(0 0,4.252632294621186 8.43596525986862,8.69579947419404 ↔
16.824093489701564,13.550465473227048 25.107950473646188,19.1066053508691 ↔
33.21091076089908,25.779290201459894 41.01711439406505,34.188839517966954 ↔
48.337222885886,45.238153936612264 54.84733442373889,60 60)
```



沿着大弧分段的地理

相关信息

[ST\\_LineSubstring](#)

### 7.5.30 ST\_SetPoint

ST\_SetPoint — 用定点替串的点。


#### Synopsis

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

#### 描述

将串的 N 点替指定点。索引从 0 开始。索引向后数，因此 -1 是最后一个点。当一个点移保持关系，在触器中特有用。

可用性：1.1.0

更新 2.3.0 :  索引

 函数支持 3d 并且不会失 z-index。

示例

```
--Change first point in line string from -1 3 to -1 1
SELECT ST_AsText(ST_SetPoint('LINESTRING(-1 2,-1 3)', 0, 'POINT(-1 1)'));
      st_astext
-----
LINESTRING(-1 1,-1 3)






---Change last point in a line string (lets play with 3d linestring this time)
SELECT ST_AsEWKT(ST_SetPoint(foo.geom, ST_NumPoints(foo.geom) - 1, ST_GeomFromEWKT('POINT ↵
(-1 1 3)'))))
FROM (SELECT ST_GeomFromEWKT('LINESTRING(-1 2 3,-1 3 4, 5 6 7)') As geom) As foo;
      st_asewkt
-----
LINESTRING(-1 2 3,-1 3 4,-1 1 3)

SELECT ST_AsText(ST_SetPoint(g, -3, p))
FROM ST_GeomFromText('LINESTRING(0 0, 1 1, 2 2, 3 3, 4 4)') AS g
      , ST_PointN(g,1) as p;
      st_astext
-----
LINESTRING(0 0,1 1,0 0,3 3,4 4)
```

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 之移几何形的度坐.

#### Synopsis






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

描述

取几何形的所有点/点，并将 -180 到 0 度范内的度移到 180 到 360 度，并将 180 到 360 度范内的度从 -180 度移到 0 度。函数是称的，因此-180 到 180 度范内的数据用 0 到 360 度范来表示，0 到 360 度范内的数据用-180 到 180 度范来表示。



#### Note

具有度/度坐的数据有用；例如 SRID 4326 (WGS 84 地理坐系)



### Warning

1.3.4 之前的 `ST_ShiftLongitude` 功能无法用于 MULTIPOINT。1.3.4 也可与 MULTIPOINT 配合使用。



`ST_ShiftLongitude` 函数支持 3d 并且不会丢失 z-index。

新增功能：2.0.0 支持多面体曲面和 TIN。

注意：`ST_ShiftLongitude` 函数在 2.2.0 中由“`ST_Shift_Longitude`”重命名



`ST_ShiftLongitude` 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

```
--single point forward transformation
SELECT ST_AsText(ST_ShiftLongitude('SRID=4326;POINT(270 0)::geometry))

st_astext
-----
POINT(-90 0)

--single point reverse transformation
SELECT ST_AsText(ST_ShiftLongitude('SRID=4326;POINT(-90 0)::geometry))

st_astext
-----
POINT(270 0)

--for linestrings the functions affects only to the sufficient coordinates
SELECT ST_AsText(ST_ShiftLongitude('SRID=4326;LINESTRING(174 12, 182 13)::geometry))

st_astext
-----
LINESTRING(174 12,-178 13)
```

### 相关信息

[ST\\_WrapX](#)

## 7.5.32 ST\_WrapX

`ST_WrapX` — 将几何体包裹在 X 轴周围。

### Synopsis

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

## 描述

此函数分割输入的几何图形，然后按照“move”参数指定的方向移落在给定“wrap”右（于“移”）或左（于正“移”）的每个果件，最将碎片重新合在一起。



### Note

于“重新居中”度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 并且不会失 z-index。

## 示例

```
-- Move all components of the given geometries whose bounding box
-- falls completely on the left of x=0 to +360
select ST_WrapX(geom, 0, 360);

-- Move all components of the given geometries whose bounding box
-- falls completely on the left of x=-30 to +360
select ST_WrapX(geom, -30, 360);
```

## 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型的几何图形，最返回 NULL。集合中折的几何图形将从其中剥离。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**

返回的几何图形可能会失去其简单性（参见 [ST\\_IsSimple](#)）。

**Note**

在版本 1.1.0 之前，此函数始终返回 2d 几何图形。从 1.1.0 开始，返回的几何图形将具有与输入几何图形相同的精度，但更高的精度保持不。使用具有第二个几何参数的版本来确定所有网格尺寸。

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 your geometries to a precision grid of 10^-3
UPDATE mytable
  SET geom = ST_SnapToGrid(geom, 0.001);

SELECT ST_AsText(ST_SnapToGrid(
  ST_GeomFromText('LINESTRING(1.1115678 2.123, 4.111111 3.2374897, ←
    4.11112 3.23748667)'),
  0.001)
  );
          st_astext
-----
LINESTRING(1.112 2.123,4.111 3.237)
--Snap a 4d geometry
SELECT ST_AsEWKT(ST_SnapToGrid(
  ST_GeomFromEWKT('LINESTRING(-1.1115678 2.123 2.3456 1.11111,
    4.111111 3.2374897 3.1234 1.1111, -1.11111112 2.123 2.3456 1.111112)'),
  ST_GeomFromEWKT('POINT(1.12 2.22 3.2 4.4444)'),
  0.1, 0.1, 0.1, 0.01) );
          st_asewkt
-----
LINESTRING(-1.08 2.12 2.3 1.1144,4.12 3.22 3.1 1.1144,-1.08 2.12 2.3 1.1144)

--With a 4d geometry - the ST_SnapToGrid(geom,size) only touches x and y coords but keeps m ←
  and z the same
SELECT ST_AsEWKT(ST_SnapToGrid(ST_GeomFromEWKT('LINESTRING(-1.1115678 2.123 3 2.3456,
  4.111111 3.2374897 3.1234 1.1111)'),
  0.01) );
          st_asewkt
-----
LINESTRING(-1.11 2.12 3 2.3456,4.11 3.24 3.1234 1.1111)
```

## 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段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。



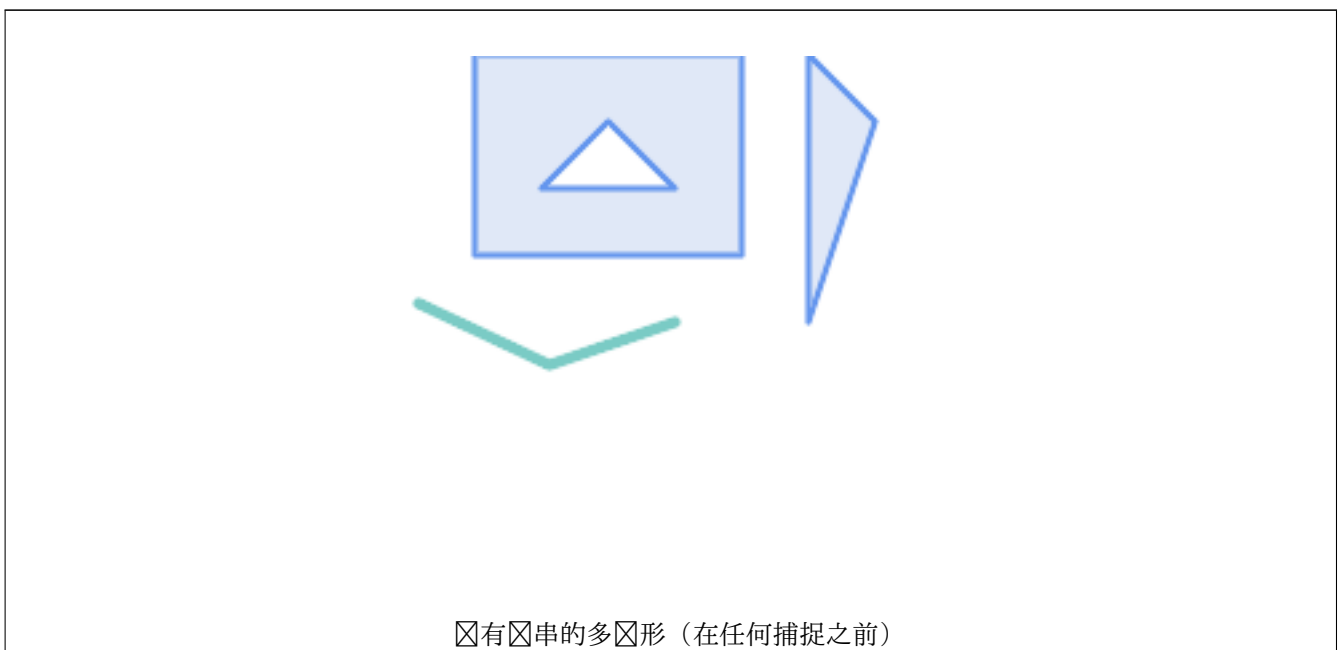
### Note

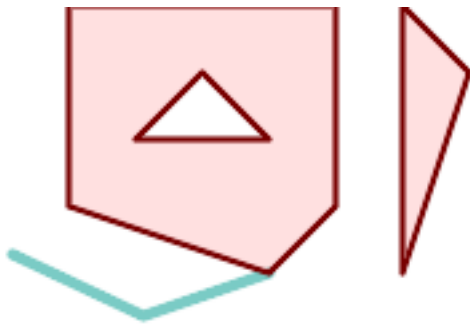
返回的几何形可能已失去其性（参ST\_IsSimple）或可能无效（参ST\_IsValid）。

个函数是由 GEOS 模行的。

可用性: 2.0.0

### 示例



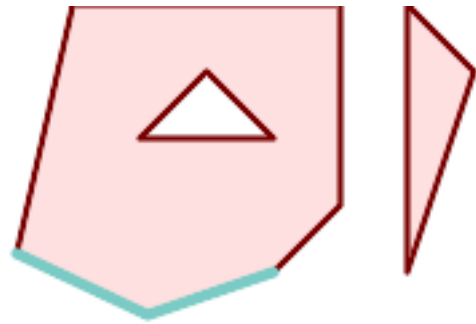


将多边形捕捉到线串，公差距离 1.01。新的多边形接到线串

```
SELECT ST_AsText(ST_Snap(poly,line, ←
    ST_Distance(poly,line)*1.01)) AS polysnapped
FROM (SELECT
    ST_GeomFromText('MULTIPOLYGON(
    ((26 125, 26 200, 126 200, 126 125, ←
    26 125 ),
    ( 51 150, 101 150, 76 175, 51 150 ) ←
    ),
    (( 151 100, 151 200, 176 175, 151 ←
    100 )))') As poly,
    ST_GeomFromText('LINESTRING (5 ←
    107, 54 84, 101 100)') As line
    ) As foo;
```

polysnapped

```
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)))
```

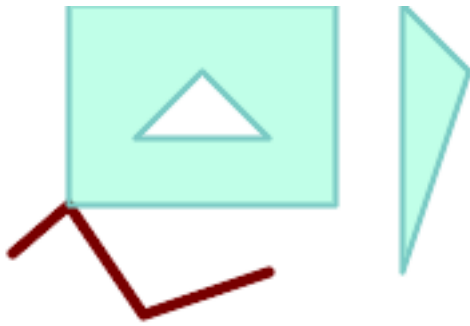


将多重多边形捕捉到公差 1.25 的多边形。新的多边形接到线串

```
SELECT ST_AsText(
    ST_Snap(poly,line, ST_Distance(poly, ←
    line)*1.25)
) AS polysnapped
FROM (SELECT
    ST_GeomFromText('MULTIPOLYGON(
    (( 26 125, 26 200, 126 200, 126 125, ←
    26 125 ),
    ( 51 150, 101 150, 76 175, 51 150 ) ←
    ),
    (( 151 100, 151 200, 176 175, 151 ←
    100 )))') As poly,
    ST_GeomFromText('LINESTRING (5 ←
    107, 54 84, 101 100)') As line
    ) As foo;
```

polysnapped

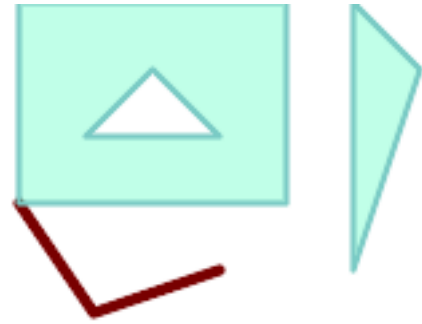
```
MULTIPOLYGON(((5 107,26 200,126 200,126 ←
    125,101 100,54 84,5 107),
(51 150,101 150,76 175,51 150)),((151 ←
    100,151 200,176 175,151 100)))
```



将☐串捕捉到原始多☐形，公差距离☐ 1.01。新☐串将☐接到多☐形

```
SELECT ST_AsText(
  ST_Snap(line, poly, ST_Distance(poly, ↵
    line)*1.01)
) AS linesnapped
FROM (SELECT
  ST_GeomFromText('MULTIPOLYGON(
    ((26 125, 26 200, 126 200, 126 125, ↵
    26 125),
    (51 150, 101 150, 76 175, 51 150 )) ↵
  ',
  ((151 100, 151 200, 176 175, 151 ↵
  100)))') As poly,
  ST_GeomFromText('LINESTRING (5 ↵
  107, 54 84, 101 100)') As line
) As foo;

          linesnapped
-----
LINESTRING(5 107,26 125,54 84,101 100)
```



将☐串捕捉到原始多☐形，公差距离☐ 1.25。新☐串将☐接到多☐形

```
SELECT ST_AsText(
  ST_Snap(line, poly, ST_Distance(poly, ↵
    line)*1.25)
) AS linesnapped
FROM (SELECT
  ST_GeomFromText('MULTIPOLYGON(
    ( 26 125, 26 200, 126 200, 126 125, ↵
    26 125 ),
    (51 150, 101 150, 76 175, 51 150 )) ↵
  ',
  ((151 100, 151 200, 176 175, 151 ↵
  100 )))') As poly,
  ST_GeomFromText('LINESTRING (5 ↵
  107, 54 84, 101 100)') As line
) As foo;

          linesnapped
-----
LINESTRING(26 125,54 84,101 100)
```

相关信息

[ST\\_SnapToGrid](#)

### 7.5.35 ST\_SwapOrdinates

ST\_SwapOrdinates — 返回更改后的几何☐形，其中交☐了☐定的坐☐☐。

#### 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形中交。

`ords` 参数是一个个字符的字符串，指定要交的坐。有效名称：`x`、`y`、`z` 和 `m`。

可用性：2.2.0

- ✔ 此方法支持形字符串和曲。
- ✔ 函数支持 3d 并且不会失 `z-index`。
- ✔ 功能支持 M 坐。
- ✔ 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

```
-- Scale M value by 2
SELECT ST_AsText(
  ST_SwapOrdinates(
    ST_Scale(
      ST_SwapOrdinates(g, 'xm'),
      2, 1
    ),
    'xm')
) FROM ( SELECT 'POINT ZM (0 0 0 2)::geometry g ) foo;
-----
POINT ZM (0 0 0 4)
```

## 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 [SQL/MM](#) `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，会输出 PostgreSQL NOTICE，提供其无效原因的详细信息。

对于有 `flags` 参数的版本，支持的详细信息在 [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)` 的结果定义为 0，而 PostGIS 返回 NULL。

这个函数是由 GEOS 模块执行的。

接受标志的版本从 2.0.0 开始可用。



此方法符合了 [SQL 1.1](#) 的 OGC [功能规范](#)。



此方法符合了 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

```
SELECT ST_IsValid(ST_GeomFromText('LINESTRING(0 0, 1 1)')) As good_line,
       ST_IsValid(ST_GeomFromText('POLYGON((0 0, 1 1, 1 2, 1 1, 0 0))')) As bad_poly
-- results
NOTICE: Self-intersection at or near point 0 0
good_line | bad_poly
-----+-----
t         | f
```

## 相关信息

[ST\\_IsSimple](#), [ST\\_IsValidReason](#), [ST\\_IsValidDetail](#),

### 7.6.2 ST\_IsValidDetail

`ST_IsValidDetail` — 返回 `valid_detail` 行，说明几何形状是否有效或无效，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)，`ST_IsValid` 是否有效；一个 `varchar` (原因)，`ST_IsValidReason` 说明其无效原因；以及一个 `geometry` (位置)，`ST_IsValidDetail` 指出其无效位置。

有助于改`ST_IsValid`和`ST_IsValidReason`的`flags`，以生成无效几何的`flags`报告。

可`flags`的 `flags` 参数是一个位字段。它可以具有以下`flags`：

- 0：使用常用的 OGC SFS 有效性`flags`。
- 1：考`flags`某些`flags`型的自接触`flags`（倒壳和倒孔）是有效的。`flags`也称`flags`“ESRI `flags`志”，因`flags`是`flags`些工具使用的有效性模型。注意，`flags`在 OGC 模型下是无效的。

`ST_IsValid`函数是由 GEOS 模`flags`行的。

可用性: 2.0.0

## 示例

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

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

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

valid	reason	location
t		

相关信息

[ST\\_IsValid](#), [ST\\_IsValidReason](#)

### 7.6.3 ST\_IsValidReason

`ST_IsValidReason` — 返回说明几何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。

个函数是由 GEOS 模行的。

有效性：1.4

有效性：2.0 版本正在接受。

#### 示例

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

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



```

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

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

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

st_isvalidreason
-----
Valid Geometry

```

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

该函数构建给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。

输入的输入是点、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角的情况下，输出几何是相同或低度的几何集合，或者是低度几何的集合。

在自相交的情况下，多个多边形可能会成为多重几何体。

`params` 参数可用于提供字符串来指定用于构建有效几何体的方法。字符串的格式为“`method=linework|structure|keepcollapsed=true|false`”。如果未提供“`params`”参数，将使用“`linework`”算法作默认值。

“`method`” 有 3 个。

- “`linework`” 是原始算法，它通首先提取所有线条、将线条点在一起、然后从线条构建出来构建有效的几何体。
- “`structure`” 是一种区分内孔和外孔的算法，通合并外孔来构建新的几何形状，然后区分所有内孔。

“keepcollapsed” 选项的“structure” 算法有效，其取值为“true” 或“false”。当置为“false” 时，会删除折线到更低度的几何物件，例如一线串（linestring）会被舍弃。

该函数是由 GEOS 模块实现的。

可用性: 2.0.0

增强: 2.0.1 速度提升

增强: 2.1.0 添加了几何集合和多点的支持。

增强: 3.1.0 除了具有 NaN 的坐标。

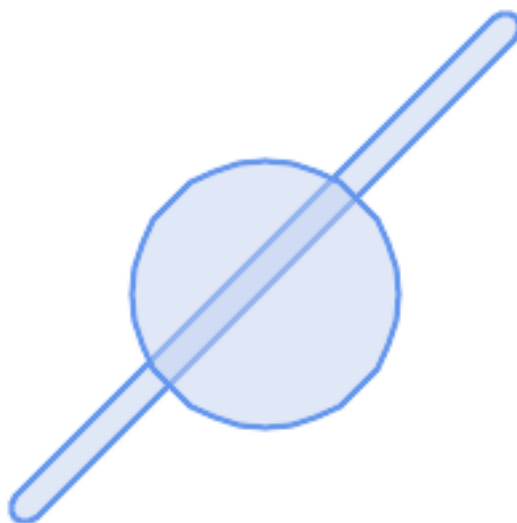
增强: 在 3.2.0 中，添加了可选的算法参数“linework” 和“structure”。需要 GEOS >= 3.10.0 或更高版本。



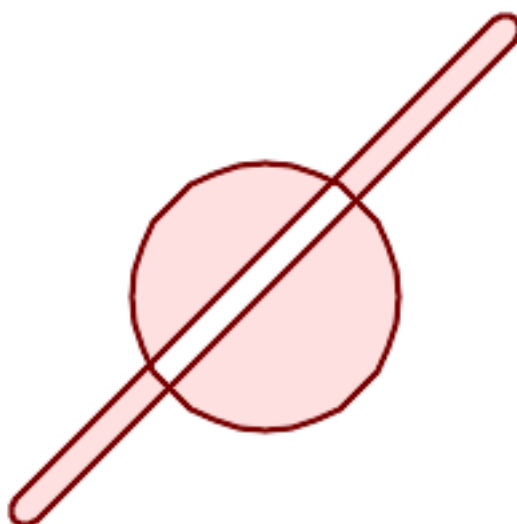
该函数支持 3d 并且不会丢失 z-index。

示例

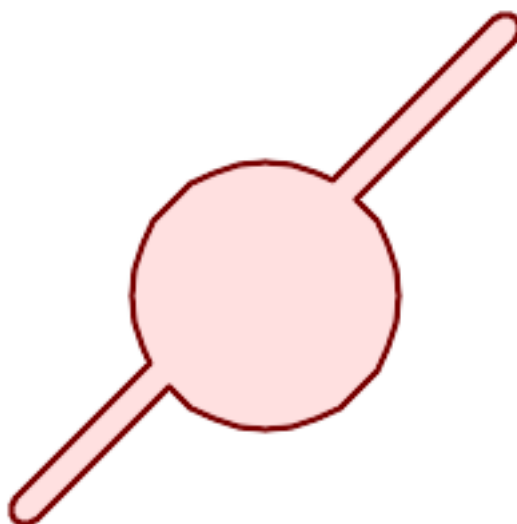
---



*before\_geom* : 由两个重叠多边形组成的多边形



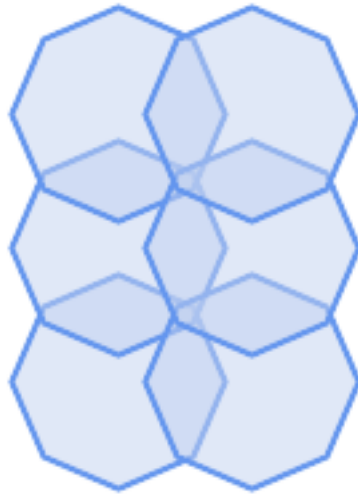
*after\_geom* : 具有四个非重叠多边形组成的多边形



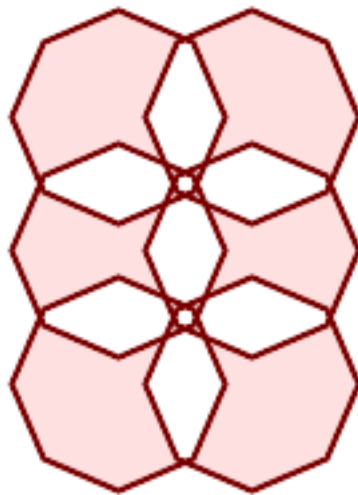
*after\_geom\_structure* : 由两个非重叠多边形组成的多边形

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

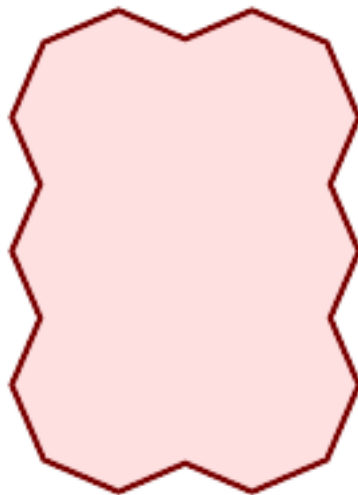




*before\_geom* : 由六个重叠多边形组成的多边形



*after\_geom* : 由 14 个不重叠多边形组成的多边形



*after\_geom\_structure* : 由一个非重叠多边形组成的多边形

```
SELECT c.geom AS before_geom,  
       ST_MakeValid(c.geom) AS after_geom,  
       ST_MakeValid(c.geom, 'method=structure') AS after_geom_structure  
FROM (SELECT 'MULTIPOLYGON(((91 50.79 22.51 10.23 22.11 50.23 78.51 90.79 78.91 ↵
```

示例

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

st_astext
-----
POINT(0 0)

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

st_astext
-----
LINESTRING EMPTY
```

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坐标到不同的空参考系。

#### 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坐标到不同的空参考系。

有关写入管道的信息，参见 [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，除非可通过的 `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](#) 的函数版本。在大多数情况下，“`ST_Transform`” 将正确的操作在坐标系之行行，并且是首。

## 示例

使用 EPSG:16031 将 WGS 84 度更改为 UTM 31N

```
-- Inverse direction
SELECT ST_AsText(ST_InverseTransformPipeline('POINT(426857.9877165967 5427937.523342293)':: geometry,
'urn:ogc:def:coordinateOperation:EPSG::16031')) AS wgs_geom;

          wgs_geom
-----
POINT(2 48.99999999999999)
(1 row)
```

GDA2020 示例。

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

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

## 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

将几何形的 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，它只是置定几何所需的空参考系的元数据。如果要将几何形重新投影，请使用 [ST\\_Transform](#)。



此方法符合了 SQL 1.1 的 OGC 功能规范。



此方法支持形字符串和曲线。

示例

-- 将点置 WGS84 度 --

```
SELECT ST_SetSRID(ST_Point(-123.365556, 48.428611),4326) As wgs84long_lat;  
-- the ewkt representation (wrap with ST_AsEWKT) -  
SRID=4326;POINT(-123.365556 48.428611)
```

-- 将点置 WGS84 度，并将其 web mercator (球面墨卡托) --

```
SELECT ST_Transform(ST_SetSRID(ST_Point(-123.365556, 48.428611),4326),3785) As spere_merc;  
-- the ewkt representation (wrap with ST_AsEWKT) -  
SRID=3785;POINT(-13732990.8753491 6178458.96425423)
```

相关信息

Section 4.5, [ST\\_SRID](#), [ST\\_Transform](#), [UpdateGeometrySRID](#)

### 7.7.3 ST\_SRID

ST\_SRID — 返回几何形的空参考系号。

#### 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 Section 4.5



#### Note

spatial\_ref\_sys 表 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。



此方法了 SQL 1.1 的 OGC 功能范。 s2.1.1.1



方法了 SQL/MM 范。 SQL-MM 3: 5.1.5



此方法支持形字符串和曲。

示例

```
SELECT ST_SRID(ST_GeomFromText('POINT(-71.1043 42.315)',4326));  
-- result  
4326
```



相关信息

Section 4.5, [ST\\_SetSRID](#), [ST\\_Transform](#), [ST\\_SRID](#), [ST\\_SRID](#)

## 7.7.4 ST\_Transform

ST\_Transform — 返回坐标位于不同空间参考系的新几何图形。

### 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，指定 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 经常与 [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最合适的管道。要使用特定的方法，请使用 [ST\\_TransformPipeline](#)。

Note!

#### 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!

#### Note

如果使用多个索引，则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!

#### 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崩溃。此问题已在 1.3.4 及更高版本中得到修正。

新增：2.0.0 引入了多面体曲面的支持。

新增：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 WGS 84 度

```
SELECT ST_AsText(ST_Transform(ST_GeomFromText('POLYGON((743238 2967416,743238 2967450,
743265 2967450,743265.625 2967416,743238 2967416))',2249),4326)) As wgs_geom;

wgs_geom
-----
POLYGON((-71.1776848522251 42.3902896512902,-71.1776843766326 42.3903829478009,
-71.1775844305465 42.3903826677917,-71.1775825927231 42.3902893647987,-71.177684
8522251 42.3902896512902));
(1 row)

--3D Circular String example
SELECT ST_AsEWKT(ST_Transform(ST_GeomFromEWKT('SRID=2249;CIRCULARSTRING(743238 2967416  ←
1,743238 2967450 2,743265 2967450 3,743265.625 2967416 3,743238 2967416 4)'),4326));

st_asewkt
-----
SRID=4326;CIRCULARSTRING(-71.1776848522251 42.3902896512902 1,-71.1776843766326  ←
42.3903829478009 2,
-71.1775844305465 42.3903826677917 3,
-71.1775825927231 42.3902893647987 3,-71.1776848522251 42.3902896512902 4)
```

建部分功能索引的示例。于不确定所有几何形都将被填充的表，最好使用部分索引来省略空几何形，既可省空，又可以使索引更小、更高效。

```
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的示例。

```
-- Find intersection of two polygons near the North pole, using a custom Gnomonic projection
-- See http://boundlessgeo.com/2012/02/flattening-the-peel/
WITH data AS (
  SELECT
    ST_GeomFromText('POLYGON((170 50,170 72,-130 72,-130 50,170 50))', 4326) AS p1,
    ST_GeomFromText('POLYGON((-170 68,-170 90,-141 90,-141 68,-170 68))', 4326) AS p2,
    '+proj=gnom +ellps=WGS84 +lat_0=70 +lon_0=-160 +no_defs'::text AS gnom
)
SELECT ST_AsText(
  ST_Transform(
    ST_Intersection(ST_Transform(p1, gnom), ST_Transform(p2, gnom)),
    gnom, 4326))
FROM data;

st_astext
-----
POLYGON((-170 74.053793645338,-141 73.4268621378904,-141 68,-170 68,-170 74.053793645338) ←
)
```

## 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 参数的写法形式：

```
+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：

```
+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，以便到不在正确范围内的 SRID 4267 不会引，您可以使用以下命令：

```
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 — 返回一个新的几何形，其坐使用定的坐管道不同的空参考系。

#### Synopsis

```
geometry ST_TransformPipeline(geometry g1, text pipeline, integer to_srid);
```

#### 描述

返回一个新的几何形，其坐使用定的坐管道不同的空参考系。

管道使用以下任意字符串格式定：

- urn:ogc:def:coordinateOperation:AUTHORITY::CODE。注意，的 EPSG:CODE 字符串不能唯一坐操作：相同的 EPSG 代可用于 CRS 定。
- 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，用者将需要添加外的管道步，或除 axisswap 步。
- 形式的串操作：urn:ogc:def:coordinateOperation,coordinateOperation:EPSG::3895,coordinateOp

可用性：3.4.0

入几何体的 SRID 将被忽略，出几何体的 SRID 将置零，除非通可的 to\_srid 参数提供。当使用“ST\_TransformPipeline()”，管道将向前行。使用 [ST\\_InverseTransformPipeline](#) 管道以相反方向行。

使用管道的是 [ST\\_Transform](#) 的版本。在大多数情况下，“ST\_Transform”将正确的操作在坐系之行，并且是首。

## 示例

使用 EPSG:16031 将 WGS 84 度更改 UTM 31N

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

          utm_geom
-----
POINT(426857.9877165967 5427937.523342293)
(1 row)

-- Inverse direction
SELECT ST_AsText(ST_InverseTransformPipeline('POINT(426857.9877165967 5427937.523342293):: ←
geometry,
'urn:ogc:def:coordinateOperation:EPSG::16031')) AS wgs_geom;

          wgs_geom
-----
POINT(2 48.999999999999999)
(1 row)
```

GDA2020 示例。

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

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

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

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

-- using a PROJ pipeline definition matching EPSG:8447, as returned from
-- 'projinfo -s EPSG:4939 -t EPSG:7844'.
-- NOTE: any 'axisswap' steps must be removed.
SELECT ST_AsText(ST_TransformPipeline('SRID=4939;POINT(143.0 -37.0)::geometry,
'+proj=pipeline
+step +proj=unitconvert +xy_in=deg +xy_out=rad
+step +proj=hgridshift +grids=au_icsm_GDA94_GDA2020_conformal_and_distortion.tif
+step +proj=unitconvert +xy_in=rad +xy_out=deg')) AS gda2020_pipeline;

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

## 相关信息

[ST\\_Transform](#), [ST\\_InverseTransformPipeline](#)

## 7.7.6 postgis\_srs\_codes

postgis\_srs\_codes — 返回与指定机关的 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 权威机关的前 10 个代码的列表。

```
SELECT * FROM postgis_srs_codes('EPSG') LIMIT 10;
```

```
postgis_srs_codes
-----
2000
20004
20005
20006
20007
20008
20009
2001
20010
20011
```

### 相关信息

[postgis\\_srs](#), [postgis\\_srs\\_all](#), [postgis\\_srs\\_search](#)

## 7.7.7 postgis\_srs

postgis\_srs — 返回所请求的机关和 srid 的元数据。

### Synopsis

```
setof record postgis_srs(text auth_name, text auth_srid);
```

### 描述

返回指定 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

☒取 EPSG 的元数据 : 3005。

```
SELECT * FROM postgis_srs('EPSG', '3005');
```

```
auth_name | EPSG
auth_srid | 3005
sname     | NAD83 / BC Albers
srtext    | PROJCS["NAD83 / BC Albers", ... ]
proj4text | +proj=aea +lat_0=45 +lon_0=-126 +lat_1=50 +lat_2=58.5 +x_0=1000000 +y_0=0 +↔
          datum=NAD83 +units=m +no_defs +type=crs
point_sw  | 0101000020E6100000E17A14AE476161C00000000000204840
point_ne  | 0101000020E610000085EB51B81E855CC0E17A14AE47014E40
```

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☒ Proj 数据☒中每个空☒参考系☒的元数据☒☒。

### Synopsis

```
setof record postgis_srs_all(void);
```

描述

返回底☒ Proj 数据☒中所有元数据☒☒的集合。☒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☒中☒取前 10 条元数据☒☒。

```
SELECT auth_name, auth_srid, sname FROM postgis_srs_all() LIMIT 10;
```

auth_name	auth_srid	sname
EPSG	2000	Anguilla 1957 / British West Indies Grid
EPSG	20004	Pulkovo 1995 / Gauss-Kruger zone 4
EPSG	20005	Pulkovo 1995 / Gauss-Kruger zone 5
EPSG	20006	Pulkovo 1995 / Gauss-Kruger zone 6
EPSG	20007	Pulkovo 1995 / Gauss-Kruger zone 7
EPSG	20008	Pulkovo 1995 / Gauss-Kruger zone 8
EPSG	20009	Pulkovo 1995 / Gauss-Kruger zone 9
EPSG	2001	Antigua 1943 / British West Indies Grid
EPSG	20010	Pulkovo 1995 / Gauss-Kruger zone 10
EPSG	20011	Pulkovo 1995 / Gauss-Kruger zone 11

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，这些坐标系的使用区域完全包含边界参数。每条记录将包含 `auth_name`、`auth_srid`、`sname`、`srttext`、`proj4text` 以及使用区域的角点 `point_sw` 和 `point_ne`。

搜索查找投影坐标系，旨在用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。

```
SELECT auth_name, auth_srid, sname,
       ST_AsText(point_sw) AS point_sw,
       ST_AsText(point_ne) AS point_ne
FROM postgis_srs_search('SRID=4326;LINESTRING(-90 30, -91 31)')
LIMIT 3;
```

auth_name	auth_srid	sname	point_sw	point_ne
EPSG	2801	NAD83(HARN) / Louisiana South	POINT(-93.94 28.85)	POINT(-88.75 31.07)
EPSG	3452	NAD83 / Louisiana South (ftUS)	POINT(-93.94 28.85)	POINT(-88.75 31.07)
EPSG	3457	NAD83(HARN) / Louisiana South (ftUS)	POINT(-93.94 28.85)	POINT(-88.75 31.07)

描述表格以得最大范围并找到可能适合的投影坐标系。

```
WITH ext AS (
  SELECT ST_Extent(geom) AS geom, Max(ST_SRID(geom)) AS srid
  FROM foo
)
SELECT auth_name, auth_srid, sname,
       ST_AsText(point_sw) AS point_sw,
       ST_AsText(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 — 给定任意闭合字符串集合 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

从闭合字符串的任意集合构造多边形, 这些字符串具有多字符串的已知文本表示形式。



#### Note

如果 WKT 不是 MULTILINESTRING, 会引发错误。如果输出是 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

这个函数是由 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 — 从任何封闭字符串集合构造多面, 并具有多字符串的已知文本表示形式。

#### Synopsis

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



## 描述

将多边形、多线串、多字符串的任意集合作为已知的文本表示，构造一个多边形。



### Note

如果 WKT 不是 MULTILINESTRING，则会引发错误。即使结果上由多个 POLYGON 组成，也限制输出 MULTIPOLYGON；如果您确定操作将生成多个 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 模块实现的。

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) 返回指定的地理点。

## 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点。如果未指定，则假定 SRID 4326。它是 `ST_GeographyFromText` 的别名。点将以经纬度形式表示。

## 示例

```
--- converting lon lat coords to geography
ALTER TABLE sometable ADD COLUMN geog geography(POINT,4326);
UPDATE sometable SET geog = ST_GeogFromText('SRID=4326;POINT(' || lon || ' ' || lat || ')') ←
;

--- specify a geography point using EPSG:4267, NAD27
SELECT ST_AsEWKT(ST_GeogFromText('SRID=4267;POINT(-77.0092 38.889588)'));
```

## 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扩展 (WKT) 返回指定的地理点。

## Synopsis

geography **ST\_GeographyFromText**(text EWKT);

### 描述

从已知的文本表示形式返回地理对象。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 几何形都是集合，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

```
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 对象。

## Synopsis

geometry **ST\_GeomFromEWKT**(text EWKT);

### 描述

根据 OGC 展已知的文本 (EWKT) 表示造 PostGIS ST\_Geometry 象。



### Note

EWKT 格式不是 OGC 准, 而是包含空参考系 (SRID) 符的 PostGIS 特定格式

增功能 : 2.0.0 支持多面体曲面和 TIN。



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

```
SELECT ST_GeomFromEWKT('SRID=4269;LINESTRING(-71.160281 42.258729,-71.160837
  42.259113,-71.161144 42.25932)');
SELECT ST_GeomFromEWKT('SRID=4269;MULTILINESTRING((-71.160281 42.258729,-71.160837
  42.259113,-71.161144 42.25932)');

SELECT ST_GeomFromEWKT('SRID=4269;POINT(-71.064544 42.28787)');

SELECT ST_GeomFromEWKT('SRID=4269;POLYGON((-71.1776585052917
  42.3902909739571,-71.1776820268866 42.3903701743239,
-71.1776063012595 42.3903825660754,-71.1775826583081 42.3903033653531,-71.1776585052917
  42.3902909739571))');

SELECT ST_GeomFromEWKT('SRID=4269;MULTIPOLYGON((( -71.1031880899493 42.3152774590236,
-71.1031627617667 42.3152960829043,-71.102923838298 42.3149156848307,
-71.1023097974109 42.3151969047397,-71.1019285062273 42.3147384934248,
-71.102505233663 42.3144722937587,-71.10277487471 42.3141658254797,
-71.103113945163 42.3142739188902,-71.10324876416 42.31402489987,
-71.1033002961013 42.3140393340215,-71.1033488797549 42.3139495090772,
-71.103396240451 42.3138632439557,-71.1041521907712 42.3141153348029,
-71.1041411411543 42.3141545014533,-71.1041287795912 42.3142114839058,
-71.1041188134329 42.3142693656241,-71.1041112482575 42.3143272556118,
-71.1041072845732 42.3143851580048,-71.1041057218871 42.3144430686681,
-71.1041065602059 42.3145009876017,-71.1041097995362 42.3145589148055,
-71.1041166403905 42.3146168544148,-71.1041258822717 42.3146748022936,
-71.1041375307579 42.3147318674446,-71.1041492906949 42.3147711126569,
-71.1041598612795 42.314808571739,-71.1042515013869 42.3151287620809,
-71.1041173835118 42.3150739481917,-71.1040809891419 42.3151344119048,
-71.1040438678912 42.3151191367447,-71.1040194562988 42.3151832057859,
-71.1038734225584 42.3151140942995,-71.10384446938243 42.3151006300338,
-71.1038315271889 42.315094347535,-71.1037393329282 42.315054824985,
-71.1035447555574 42.3152608696313,-71.1033436658644 42.3151648370544,
-71.1032580383161 42.3152269126061,-71.103223066939 42.3152517403219,
```

```
-71.1031880899493 42.3152774590236)),
((-71.1043632495873 42.315113108546, -71.1043583974082 42.3151211109857,
-71.1043443253471 42.3150676015829, -71.1043850704575 42.3150793250568, -71.1043632495873 ←
 42.315113108546)))');
```

```
--3d circular string
SELECT ST_GeomFromEWKT('CIRCULARSTRING(220268 150415 1,220227 150505 2,220227 150406 3)');
```

```
--Polyhedral Surface example
SELECT ST_GeomFromEWKT('POLYHEDRALSURFACE(
  ((0 0 0, 0 0 1, 0 1 1, 0 1 0, 0 0 0)),
  ((0 0 0, 0 1 0, 1 1 0, 1 0 0, 0 0 0)),
  ((0 0 0, 1 0 0, 1 0 1, 0 0 1, 0 0 0)),
  ((1 1 0, 1 1 1, 1 0 1, 1 0 0, 1 1 0)),
  ((0 1 0, 0 1 1, 1 1 1, 1 1 0, 0 1 0)),
  ((0 0 1, 1 0 1, 1 1 1, 0 1 1, 0 0 1))
)');
```

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 对象的地理数据中，目前无法描述对象的空参考系，因此函数将始终返回 SRID 0 的几何对象。



#### Note

返回的 POLYGON 几何对象将始终是逆时针方向。

示例

包含 hddd.dddd 的 1 个 POINT 的 MARC21/XML 地理数据

```

SELECT
  ST_AsText(
    ST_GeomFromMARC21('
      <record xmlns="http://www.loc.gov/MARC21/slim">
        <leader
>00000nz a2200000nc 4500</leader>
        <controlfield tag="001"
>040277569</controlfield>
        <datafield tag="034" ind1=" " ind2=" ">
          <subfield code="d"
>W004.500000</subfield>
          <subfield code="e"
>W004.500000</subfield>
          <subfield code="f"
>N054.250000</subfield>
          <subfield code="g"
>N054.250000</subfield>
        </datafield>
      </record
>'));

 st_astext
-----
POINT(-4.5 54.25)
(1 row)

```

包含 hddmmss 的 1 个 POLYGON 的 MARC21/XML 地理数据

```

SELECT
  ST_AsText(
    ST_GeomFromMARC21('
      <record xmlns="http://www.loc.gov/MARC21/slim">
        <leader
>01062cem a2200241 a 4500</leader>
        <controlfield tag="001"
> 84696781 </controlfield>
        <datafield tag="034" ind1="1" ind2=" ">
          <subfield code="a"
>a</subfield>
          <subfield code="b"
>50000</subfield>
          <subfield code="d"
>E0130600</subfield>
          <subfield code="e"
>E0133100</subfield>
          <subfield code="f"
>N0523900</subfield>
          <subfield code="g"
>N0522300</subfield>
        </datafield>
      </record
>'));

 st_astext

```

```

-----
POLYGON((13.1 52.65,13.51666666666667 52.65,13.51666666666667  ←
        52.38333333333333,13.1 52.38333333333333,13.1 52.65))
(1 row)

```

☒☒包含 POLYGON 和 POINT 的 MARC21/XML 地理数据：

```

SELECT
ST_AsText(
  ST_GeomFromMARC21('
<record xmlns="http://www.loc.gov/MARC21/slim">
  <datafield tag="034" ind1="1" ind2=" ">
    <subfield code="a"
>a</subfield>
    <subfield code="b"
>50000</subfield>
    <subfield code="d"
>E0130600</subfield>
    <subfield code="e"
>E0133100</subfield>
    <subfield code="f"
>N0523900</subfield>
    <subfield code="g"
>N0522300</subfield>
  </datafield>
  <datafield tag="034" ind1=" " ind2=" ">
    <subfield code="d"
>W004.500000</subfield>
    <subfield code="e"
>W004.500000</subfield>
    <subfield code="f"
>N054.250000</subfield>
    <subfield code="g"
>N054.250000</subfield>
  </datafield>
</record
>'));

```

st\_astext ↩

```

-----
GEOMETRYCOLLECTION(POLYGON((13.1 52.65,13.51666666666667  ←
        52.65,13.51666666666667 52.38333333333333,13.1 52.38333333333333,13.1  ←
        52.65)),POINT(-4.5 54.25))
(1 row)

```

相关信息

[ST\\_AsMARC21](#)

### 7.8.1.8 ST\_GeometryFromText

ST\_GeometryFromText — 从已知的文本表示 (WKT) 返回指定的 ST\_Geometry ☒。☒是 ST\_GeomFromText 的☒名

## Synopsis

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

### 描述

-  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) 的几何图形。第二个将 SRID 作为第二个参数，并返回一个几何图形，该几何图形包含此 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 - 可将参数 SRID 用于规范一致性。
-  此方法符合了 SQL/MM 规范。SQL-MM 3: 5.1.40
-  此方法支持图形字符串和曲线。



#### 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。



#### Warning

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 中现在是非法的。在旧版本中写成 ST\_GeomFromText('GEOMETRYCOLLECTION EMPTY')

示例

```

SELECT ST_GeomFromText('LINESTRING(-71.160281 42.258729,-71.160837 42.259113,-71.161144 42.25932)');
SELECT ST_GeomFromText('LINESTRING(-71.160281 42.258729,-71.160837 42.259113,-71.161144 42.25932)',4269);

SELECT ST_GeomFromText('MULTILINESTRING((-71.160281 42.258729,-71.160837 42.259113,-71.161144 42.25932))');

SELECT ST_GeomFromText('POINT(-71.064544 42.28787)');

SELECT ST_GeomFromText('POLYGON((-71.1776585052917 42.3902909739571,-71.1776820268866 42.3903701743239,-71.1776063012595 42.3903825660754,-71.1775826583081 42.3903033653531,-71.1776585052917 42.3902909739571))');

SELECT ST_GeomFromText('MULTIPOLYGON((( -71.1031880899493 42.3152774590236,
-71.1031627617667 42.3152960829043,-71.102923838298 42.3149156848307,
-71.1023097974109 42.3151969047397,-71.1019285062273 42.3147384934248,
-71.102505233663 42.3144722937587,-71.10277487471 42.3141658254797,
-71.103113945163 42.3142739188902,-71.10324876416 42.31402489987,
-71.1033002961013 42.3140393340215,-71.1033488797549 42.3139495090772,
-71.103396240451 42.3138632439557,-71.1041521907712 42.3141153348029,
-71.1041411411543 42.3141545014533,-71.1041287795912 42.3142114839058,
-71.1041188134329 42.3142693656241,-71.1041112482575 42.3143272556118,
-71.1041072845732 42.3143851580048,-71.1041057218871 42.3144430686681,
-71.1041065602059 42.3145009876017,-71.1041097995362 42.3145589148055,
-71.1041166403905 42.3146168544148,-71.1041258822717 42.3146748022936,
-71.1041375307579 42.3147318674446,-71.1041492906949 42.3147711126569,
-71.1041598612795 42.314808571739,-71.1042515013869 42.3151287620809,
-71.1041173835118 42.3150739481917,-71.1040809891419 42.3151344119048,
-71.1040438678912 42.3151191367447,-71.1040194562988 42.3151832057859,
-71.1038734225584 42.3151140942995,-71.1038446938243 42.3151006300338,
-71.1038315271889 42.315094347535,-71.1037393329282 42.315054824985,
-71.1035447555574 42.3152608696313,-71.1033436658644 42.3151648370544,
-71.1032580383161 42.3152269126061,-71.103223066939 42.3152517403219,
-71.1031880899493 42.3152774590236)),
((-71.1043632495873 42.315113108546,-71.1043583974082 42.3151211109857,
-71.1043443253471 42.3150676015829,-71.1043850704575 42.3150793250568,-71.1043632495873 42.315113108546)))',4326);

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

```

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 表示构建几何线形。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点串，使用 ST\_GeomFromText 会更有效。此函数调用 ST\_GeomFromText 并估计是否返回点串。



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

```
SELECT ST_LineFromText('LINESTRING(1 2, 3 4)') AS aline, ST_LineFromText('POINT(1 2)') AS null_return;
aline | null_return
-----|-----
010200000002000000000000000000000F ... | t
```

## 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
- ✔ 此方法符合了 SQL/MM 规范。 SQL-MM 3: 9.4.4

示例

```
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 功能规范。 3.2.6.2
- ✔ 此方法符合了 SQL/MM 规范。 SQL-MM 3: 9.2.4

示例

```
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 制作多面体。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 生成多面体。如果未给出 SRID，默认为 0 (未知)。

OGC 规范 3.2.6.2 - 可指定 SRID 用于规范

如果 WKT 不是多面体，将引发错误。



#### Note

如果您确定所有 WKT 几何体都是多面体，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。 SQL-MM 3: 9.6.4

#### 示例

```
SELECT ST_MPolyFromText('MULTIPOLYGON(((0 0 1,20 0 1,20 20 1,0 20 1,0 0 1),(5 5 3,5 7 3,7 7 ←
3,7 5 3,5 5 3)))');
SELECT ST_MPolyFromText('MULTIPOLYGON((-70.916 42.1002,-70.9468 42.0946,-70.9765 ←
42.0872,-70.9754 42.0875,-70.9749 42.0879,-70.9752 42.0881,-70.9754 42.0891,-70.9758 ←
42.0894,-70.9759 42.0897,-70.9759 42.0899,-70.9754 42.0902,-70.9756 42.0906,-70.9753 ←
42.0907,-70.9753 42.0917,-70.9757 42.0924,-70.9755 42.0928,-70.9755 42.0942,-70.9751 ←
42.0948,-70.9755 42.0953,-70.9751 42.0958,-70.9751 42.0962,-70.9759 42.0983,-70.9767 ←
42.0987,-70.9768 42.0991,-70.9771 42.0997,-70.9771 42.1003,-70.9768 42.1005,-70.977 ←
42.1011,-70.9766 42.1019,-70.9768 42.1026,-70.9769 42.1033,-70.9775 42.1042,-70.9773 ←
42.1043,-70.9776 42.1043,-70.9778 42.1048,-70.9773 42.1058,-70.9774 42.1061,-70.9779 ←
42.1065,-70.9782 42.1078,-70.9788 42.1085,-70.9798 42.1087,-70.9806 42.109,-70.9807 ←
42.1093,-70.9806 42.1099,-70.9809 42.1109,-70.9808 42.1112,-70.9798 42.1116,-70.9792 ←
42.1127,-70.979 42.1129,-70.9787 42.1134,-70.979 42.1139,-70.9791 42.1141,-70.9987 ←
42.1116,-71.0022 42.1273,
-70.9408 42.1513,-70.9315 42.1165,-70.916 42.1002)))',4326);
```

#### 相关信息

[ST\\_GeomFromText](#), [ST\\_SRID](#)

### 7.8.1.14 ST\_PointFromText

ST\_PointFromText — 使用指定的 SRID 从 WKT 构建点几何体。如果未给出 SRID，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。如果未给出 SRID，默认未知（当前 0）。如果几何形不是 WKT 点表示，返回 null。如果 WKT 完全无效，抛出。



#### 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形。第二个将空参考 id 作第二个参数，并返回一个 ST\_Geometry，其中包含此 srid 作其元数据的一部分。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。如果您从度坐建点并且更关心性能和准确性而不是 OGC 合规性，使用 **ST\_MakePoint** 或 OGC 合规名 **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。



方法符合了 SQL/MM 规范。SQL-MM 3: 6.1.8

### 示例

```
SELECT ST_PointFromText('POINT(-71.064544 42.28787)');
SELECT ST_PointFromText('POINT(-71.064544 42.28787)', 4326);
```

### 相关信息

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

#### 7.8.1.15 ST\_PolygonFromText

ST\_PolygonFromText — 使用给定的 SRID 从 WKT 建几何形。如果未给出 SRID，默认 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 - 可选项 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



此方法符合了 SQL/MM 规范。 SQL-MM 3: 8.3.6

## 示例

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

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

## 相关信息

[ST\\_GeomFromText](#)

**7.8.1.16 ST\_WKTToSQL**

ST\_WKTToSQL — 从已知的文本表示 (WKT) 返回指定的 ST\_Geometry 对象。它是 ST\_GeomFromText 的别名

**Synopsis**

```
geometry ST_WKTToSQL(text WKT);
```

## 描述



此方法符合了 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) 创建地理实例。

#### 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，并创建适当地理类型的实例。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 度)。



此方法支持形字符串和曲线。

#### 示例

```
--Although bytea rep contains single \, these need to be escaped when inserting into a
table
SELECT ST_AsText(
ST_GeogFromWKB(E'\\001\\002\\000\\000\\000\\002\\000\\000\\000\\037\\205\\3530
\\270~\\\\\\\\300\\323Mb\\020X\\231C@\\020X9\\264\\310~\\\\\\\\300)\\\\\\\\217\\302\\365\\230
C@')
);
----- st_astext -----
LINESTRING(-113.98 39.198,-113.981 39.195)
(1 row)
```

#### 相关信息

[ST\\_GeogFromText](#), [ST\\_AsBinary](#)

### 7.8.2.2 ST\_GeomFromEWKB

ST\_GeomFromEWKB — 从扩展已知的二进制表示 (EWKB) 返回指定的 ST\_Geometry 实例。

#### Synopsis

```
geometry ST_GeomFromEWKB(bytea EWKB);
```

## 描述

从 OGC 展已知的二进制 (EWKT) 表示构造 PostGIS ST\_Geometry 象。


**Note**

EWKB 格式不是 OGC 准，而是包含空参考系 (SRID) 符的 PostGIS 特定格式

增功能：2.0.0 支持多面体曲面和 TIN。



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

NAD83 度 (4269) 中 LINESTRING (-71.160281 42.258729, -71.160837 42.259113, -71.161144 42.25932) 的二制表示形式。


**Note**

注意：字函数由 \ 分隔并具有'，但如果 standard\_conforming\_strings 被截断，它使用 \ 和" 行。它不完全于 AsEWKB 表式。

```
SELECT ST_GeomFromEWKB(E'\\001\\002\\000\\000 \\255\\020\\000\\000\\003\\000\\000\\000\\344 ←
  J=
  \\013B\\312Q\\300n\\303(\\010\\036!E@' '\\277E' 'K
  \\312Q\\300\\366{b\\235*!E@\\225|\\354.P\\312Q
  \\300p\\231\\323e1!E@');
```


**Note**

在 PostgreSQL 9.1 中，standard\_conforming\_strings 默置 on，而在去的版本中它被置 off。您可以根据个的需要或在数据或服务器更改默。以下是使用 standard\_conforming\_strings = on 的操作方法。在种情况下，我使用准 ansi 符，但斜杠不会。

```
set standard_conforming_strings = on;
SELECT ST_GeomFromEWKB('\\001\\002\\000\\000 \\255\\020\\000\\000\\003\\000\\000\\000\\344J=\\012\\013B
  \\312Q\\300n\\303(\\010\\036!E@' '\\277E' 'K\\012\\312Q\\300\\366{b\\235*!E@\\225|\\354.P\\312Q\\012\\300 ←
  p\\231\\323e1')
```

## 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 创建几何实例。

#### 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二进制表示形式和可选参考系 ID (SRID)，并创建相应几何类型的实例。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，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
- ✓ 此方法支持几何字符串和曲线。

#### 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.
SELECT ST_AsEWKT(
ST_GeomFromWKB(E'\\001\\002\\000\\000\\000\\002\\000\\000\\000\\0037\\205\\3530
\\270~\\111\\300\\323Mb\\020X\\231C@\\020X9\\264\\310~\\111\\300)\\111\\217\\302\\365\\230
C@',4326)
);
          st_asewkt
-----
SRID=4326;LINESTRING(-113.98 39.198,-113.981 39.195)
(1 row)

SELECT
  ST_AsText(
    ST_GeomFromWKB(
      ST_AsEWKB('POINT(2 5)::geometry)
    )
  );
  st_astext
-----
POINT(2 5)
(1 row)
```

#### 相关信息

[ST\\_WKBToSQL](#), [ST\\_AsBinary](#), [ST\\_GeomFromEWKB](#)

### 7.8.2.4 ST\_LineFromWKB

ST\_LineFromWKB — 使用指定的 SRID 从 WKB 制作 LINESTRING



## Synopsis

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

### 描述

`ST_LineFromWKB` 函数采用已知的几何二进制表示形式和空参考系 ID (SRID)，并建造适当几何类型的实例 - 在本例中为 `LINestring` 几何。该函数起到了 SQL 中几何工厂的作用。

如果未指定 SRID，默认为 0。如果输入字不表示 `LINestring`，返回 `NULL`。



#### Note

OGC 规范 3.2.6.2 - 可 SRID 用于规范。



#### Note

如果您知道所有几何形都是 `LINestring`，使用 `ST_GeomFromWKB` 会更有效。该函数用 `ST_GeomFromWKB` 并添加外的以确保它返回串。



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

```
SELECT ST_LineFromWKB(ST_AsBinary(ST_GeomFromText('LINestring(1 2, 3 4)'))) AS aline,
       ST_LineFromWKB(ST_AsBinary(ST_GeomFromText('POINT(1 2)'))) IS NULL AS ←
       null_return;
aline | null_return
-----|-----
01020000000200000000000000000000F ... | t
```

### 相关信息

[ST\\_GeomFromWKB](#), [ST\\_LinestringFromWKB](#)

#### 7.8.2.5 ST\_LinestringFromWKB

`ST_LinestringFromWKB` — 使用指定的 SRID 从 WKB 建造几何形。

## Synopsis

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

## 描述

`ST_LineFromWKB` 函数采用已知的几何二进制表示形式和空参考系 ID (SRID), 并建适当几何型的例 - 在本例中 `LINestring` 几何。函数起到了 SQL 中几何工厂的作用。

如果未指定 SRID, 默 0。如果入字不表示 `LINestring`, 返回 `NULL`。是 `ST_LineFromWKB` 的。



### Note

OGC 3.2.6.2 - 可 SRID 用于。



### Note

如果您知道所有几何形都是 `LINestring`, 使用 `ST_GeomFromWKB` 会更有效。函数用 `ST_GeomFromWKB` 并添加外的以确保它返回 `LINestring`。



此方法了 SQL 1.1 的 OGC 功能。 s3.2.6.2



方法了 SQL/MM 范。 SQL-MM 3: 7.2.9

## 示例

```
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;
-----
01020000000200000000000000000000F ... | t
```

## 相关信息

[ST\\_GeomFromWKB](#), [ST\\_LineFromWKB](#)

### 7.8.2.6 ST\_PointFromWKB

`ST_PointFromWKB` — 使用定的 SRID 从 WKB 建几何形

## 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参考系 ID (SRID)，并构建适当几何类型的几何。在本例中 `POINT` 几何。该函数起到了 SQL 中几何工厂的作用。

如果未指定 SRID，默认为 0。如果输入字不表示 `POINT` 几何形，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。
- ✔ 此方法支持形字符串和曲线。

## 示例

```
SELECT
  ST_AsText(
    ST_PointFromWKB(
      ST_AsEWKB('POINT(2 5)::geometry')
    )
  );
st_astext
-----
POINT(2 5)
(1 row)

SELECT
  ST_AsText(
    ST_PointFromWKB(
      ST_AsEWKB('LINESTRING(2 5, 2 6)::geometry')
    )
  );
st_astext
-----
(1 row)
```

## 相关信息

[ST\\_GeomFromWKB](#), [ST\\_LineFromWKB](#)

**7.8.2.7 ST\_WKBToSQL**

`ST_WKBToSQL` — 从已知的二进制表示 (WKB) 返回指定的 `ST_Geometry`。是 `ST_GeomFromWKB` 的别名，不 `srid`

**Synopsis**

geometry **ST\_WKBToSQL**(bytea WKB);

## 描述

- ✔ 该方法符合了 SQL/MM 规范。 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 将返回基于输入 GeoHash 字符串的完整精度的 BOX2D。

如果指定 `precision`, ST\_Box2dFromGeoHash 将使用 GeoHash 中的更多字符来构建 BOX2D。低的精度会导致大的 BOX2D, 大的会提高精度。

可用性 : 2.1.0

#### 示例

```
SELECT ST_Box2dFromGeoHash('9qqj7nmxncgyy4d0dbxqz0');

          st_geomfromgeohash
-----
BOX(-115.172816 36.114646,-115.172816 36.114646)

SELECT ST_Box2dFromGeoHash('9qqj7nmxncgyy4d0dbxqz0', 0);

          st_box2dfromgeohash
-----
BOX(-180 -90,180 90)

SELECT ST_Box2dFromGeoHash('9qqj7nmxncgyy4d0dbxqz0', 10);

          st_box2dfromgeohash
-----
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

```
SELECT ST_AsText(ST_GeomFromGeoHash('9qqj7nmxcggy4d0dbxqz0'));
           st_astext
-----
POLYGON((-115.172816 36.114646,-115.172816 36.114646,-115.172816 36.114646,-115.172816  ←
          36.114646,-115.172816 36.114646))

SELECT ST_AsText(ST_GeomFromGeoHash('9qqj7nmxcggy4d0dbxqz0', 4));
           st_astext
-----
POLYGON((-115.3125 36.03515625,-115.3125 36.2109375,-114.9609375 36.2109375,-114.9609375  ←
          36.03515625,-115.3125 36.03515625))

SELECT ST_AsText(ST_GeomFromGeoHash('9qqj7nmxcggy4d0dbxqz0', 10));
           st_astext ←
-----
POLYGON((-115.17282128334 36.1146408319473,-115.17282128334  ←
          36.1146461963654,-115.172810554504 36.1146461963654,-115.172810554504  ←
          36.1146408319473,-115.17282128334 36.1146408319473))
```

### 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。

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。



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 允许混合度 (例如，同一 MultiGeometry 内的 2D 和 3D)。由于 PostGIS 几何形不做，如果一旦缺失的 Z 度，ST\_GeomFromGML 会将整个几何形强制 2D。

GML 支持同一 MultiGeometry 内的混合 SRS。由于 PostGIS 几何形不做，因此在本例中 ST\_GeomFromGML 会将所有子几何形重新投影到 SRS 根点。如果 GML 根点没有可用的 srsName 属性，函数会抛出。

ST\_GeomFromGML 函数于式 GML 命名空间并不迂腐。您可以避免在常用法中明确提及它。但如果您想在 GML 中使用 XLink 功能，需要它。



### Note

ST\_GeomFromGML 函数不支持 SQL/MM 曲线几何形状。

示例 - 具有 srsName 属性的几何形

```
SELECT ST_GeomFromGML($$
  <gml:LineString xmlns:gml="http://www.opengis.net/gml"
    srsName="EPSG:4269">
    <gml:coordinates>
      -71.16028,42.258729 -71.160837,42.259112 -71.161143,42.25932
    </gml:coordinates>
  </gml:LineString>
$$);
```

## 示例-XLink 使用法

```

SELECT ST_GeomFromGML($$
  <gml:LineString xmlns:gml="http://www.opengis.net/gml"
    xmlns:xlink="http://www.w3.org/1999/xlink"
    srsName="urn:ogc:def:crs:EPSG::4269">
    <gml:pointProperty>
      <gml:Point gml:id="p1"
        ><gml:pos
        >42.258729 -71.16028</gml:pos
      </gml:Point>
      </gml:pointProperty>
      <gml:pos
      >42.259112 -71.160837</gml:pos>
      <gml:pointProperty>
        <gml:Point xlink:type="simple" xlink:href="#p1"/>
      </gml:pointProperty>
    </gml:LineString>
  $$);

```

## 示例-多面体曲面

```

SELECT ST_AsEWKT(ST_GeomFromGML('
<gml:PolyhedralSurface xmlns:gml="http://www.opengis.net/gml">
<gml:polygonPatches>
  <gml:PolygonPatch>
    <gml:exterior>
      <gml:LinearRing
      ><gml:posList srsDimension="3"
      >0 0 0 0 1 0 1 1 0 1 0 0 0 0</gml:posList
      </gml:LinearRing>
      </gml:exterior>
    </gml:PolygonPatch>
    <gml:PolygonPatch>
      <gml:exterior>
        <gml:LinearRing
        ><gml:posList srsDimension="3"
        >0 0 0 1 0 1 1 0 1 0 0 0 0 0</gml:posList
        </gml:LinearRing>
        </gml:exterior>
      </gml:PolygonPatch>
      <gml:PolygonPatch>
        <gml:exterior>
          <gml:LinearRing
          ><gml:posList srsDimension="3"
          >0 0 0 1 0 0 1 0 1 0 0 1 0 0 0</gml:posList
          </gml:LinearRing>
          </gml:exterior>
        </gml:PolygonPatch>
        <gml:PolygonPatch>
          <gml:exterior>
            <gml:LinearRing
            ><gml:posList srsDimension="3"
            >1 1 0 1 1 1 1 0 1 1 0 0 1 1 0</gml:posList
            </gml:LinearRing>
            </gml:exterior>
          </gml:PolygonPatch>
          <gml:PolygonPatch>
            <gml:exterior>

```

```

    <gml:LinearRing
  ><gml:posList srsDimension="3"
  >0 1 0 0 1 1 1 1 1 1 1 1 0 0 1 0</gml:posList
  ></gml:LinearRing>
  </gml:exterior>
  </gml:PolygonPatch>
  <gml:PolygonPatch>
    <gml:exterior>
      <gml:LinearRing
  ><gml:posList srsDimension="3"
  >0 0 1 1 0 1 1 1 1 0 1 1 0 0 1</gml:posList
  ></gml:LinearRing>
  </gml:exterior>
  </gml:PolygonPatch>
</gml:polygons>
</gml:PolyhedralSurface
>');

-- result --
POLYHEDRALSURFACE(((0 0 0,0 0 1,0 1 1,0 1 0,0 0 0)),
((0 0 0,0 1 0,1 1 0,1 0 0,0 0 0)),
((0 0 0,1 0 0,1 0 1,0 0 1,0 0 0)),
((1 1 0,1 1 1,1 0 1,1 0 0,1 1 0)),
((0 1 0,0 1 1,1 1 1,1 1 0,0 1 0)),
((0 0 1,1 0 1,1 1 1,0 1 1,0 0 1)))

```

## 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。

增：3.0.0 如果未指定其他 SRID 解析几何，默认 SRID 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，支持您将收到通知而不是看到输出。要用 JSON-C，进行 configure --with-jsondir=/path/to/json-c。有关信息，参 Section [2.2.3](#)。



函数支持 3d 并且不会失 z-index。



示例

```
SELECT ST_AsText(ST_GeomFromGeoJSON('{"type":"Point","coordinates":[-48.23456,20.12345]}')) ←
      As wkt;
wkt
-----
POINT(-48.23456 20.12345)
```

```
-- a 3D linestring
SELECT ST_AsText(ST_GeomFromGeoJSON('{"type":"LineString","coordinates ←
      ":[1,2,3],[4,5,6],[7,8,9]}')) As wkt;
wkt
-----
LINESTRING(1 2,4 5,7 8)
```

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

从 OGC KML 表示构造 PostGIS ST\_Geometry 对象。

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。

OGC KML 的相应版本包括：

- KML 2.2.0 命名空间

有关 OGC KML 规范，参看 <http://www.opengeospatial.org/standards/kml>：

可用性：1.5，需要 libxml2 2.6+



函数支持 3d 并且不会失 z-index。



#### Note

ST\_GeomFromKML 函数不支持 SQL/MM 曲线几何形。

示例 - 具有 **srsName** 属性的 ☐ 个几何 ☐ 形

```
SELECT ST_GeomFromKML($$
  <LineString>
    <coordinates>
>-71.1663,42.2614
    -71.1667,42.2616</coordinates>
  </LineString>
$$);
```

相关信息

Section [2.2.3](#), [ST\\_AsKML](#)

### 7.8.3.6 ST\_GeomFromTWKB

ST\_GeomFromTWKB — 从 TWKB (“微小的已知的二 ☐ 制”) 几何表示 ☐ 建几何 ☐ 例。

#### Synopsis

geometry **ST\_GeomFromTWKB**(bytea twkb);

描述

ST\_GeomFromTWKB 采用 TWKB (“微小的已知二 ☐ 制”) 几何表示形式，并 ☐ 建相 ☐ 几何 ☐ 型的 ☐ 例。

示例

```
SELECT ST_AsText(ST_GeomFromTWKB(ST_AsTWKB('LINESTRING(126 34, 127 35)::geometry')));
      st_astext
-----
LINESTRING(126 34, 127 35)
(1 row)

SELECT ST_AsEWKT(
  ST_GeomFromTWKB(E'\\x620002f7f40dbce4040105')
);
                                     st_asewkt
-----
LINESTRING(-113.98 39.198,-113.981 39.195)
(1 row)
```

相关信息

[ST\\_AsTWKB](#)

### 7.8.3.7 ST\_GMLToSQL

ST\_GMLToSQL — 从 GML 表示返回指定的 ST\_Geometry ☐。☐ 是 ST\_GeomFromGML 的 ☐ 名

## Synopsis

```
geometry ST_GMLToSQL(text geomgml);
geometry ST_GMLToSQL(text geomgml, integer srid);
```

### 描述



方法符合了 SQL/MM 规范。SQL-MM 3: 5.1.50 (曲线支持除外)。

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。

### 相关信息

Section 2.2.3, [ST\\_GeomFromGML](#), [ST\\_AsGML](#)

## 7.8.3.8 ST\_LineFromEncodedPolyline

`ST_LineFromEncodedPolyline` — 从折线构建 `LineString`。

### Synopsis

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

### 描述

从多段字符串构建 `LineString`。

可指定 `precision` 指定折线中将保留多少个小数位。和解折线的精度相同，否则坐将不正确。

参照 <http://developers.google.com/maps/documentation/utilities/polylinealgorithm>

可用性：2.2.0

### 示例

```
-- Create a line string from a polyline
SELECT ST_AsEWKT(ST_LineFromEncodedPolyline('_p~iF~ps|U_ulLnnqC_mqNvxq`@'));
-- result --
SRID=4326;LINESTRING(-120.2 38.5,-120.95 40.7,-126.453 43.252)

-- Select different precision that was used for polyline encoding
SELECT ST_AsEWKT(ST_LineFromEncodedPolyline('_p~iF~ps|U_ulLnnqC_mqNvxq`@',6));
-- result --
SRID=4326;LINESTRING(-12.02 3.85,-12.095 4.07,-12.6453 4.3252)
```

### 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

```
SELECT ST_AsText(ST_PointFromGeoHash('9qqj7nmxcgyy4d0dbxqz0'));
           st_astext
-----
POINT(-115.172816 36.114646)

SELECT ST_AsText(ST_PointFromGeoHash('9qqj7nmxcgyy4d0dbxqz0', 4));
           st_astext
-----
POINT(-115.13671875 36.123046875)

SELECT ST_AsText(ST_PointFromGeoHash('9qqj7nmxcgyy4d0dbxqz0', 10));
           st_astext
-----
POINT(-115.172815918922 36.1146435141563)
```

#### 相关信息

[ST\\_GeoHash](#), [ST\\_Box2dFromGeoHash](#), [ST\\_GeomFromGeoHash](#)

### 7.8.3.10 ST\_FromFlatGeobufToTable

ST\_FromFlatGeobufToTable — 根据 FlatGeobuf 数据的☒☒☒建一个表。

#### Synopsis

```
void ST_FromFlatGeobufToTable(text schemaname, text tablename, bytea FlatGeobuf input data);
```

## 描述

根据 FlatGeobuf 数据的 `table` 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` 输入 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 几何形的逆操作, 使用 `ST_GeomFromEWKT`。



## 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。可选的 *maxdecimaldigits* 参数可用于限制输出坐中小数点后的位数 (默认为 15)。

要行 WKT 表示形式到 PostGIS 几何形的逆行, 使用 **ST\_GeomFromText**。



### Note

标准 OGC WKT 表示不包括 SRID。要将 SRID 作输出表示的一部分, 使用非标准 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。为了确保数据存或行的完全准确性, 最好使用 **已知的二进制** (WKB) 格式 (参 **ST\_AsBinary** 和 *maxdecimaldigits*)。



### Warning

使用 *maxdecimaldigits* 参数可能会致出行几何形无效。为了避免这种情况, 首先使用 **ST\_ReducePrecision** 和合适的网格大小。

可用性: 1.5 - 引入了地理支持。

增行: 2.5 - 引入了可行的精度参数。



此方法行了 **SQL 1.1 的 OGC 行功能行范**。s2.1.1.1



行方法行了 SQL/MM 行范。SQL-MM 3: 5.1.25



此方法支持行形字符串和曲行。

## 示例

```
SELECT ST_AsText('010300000001000000050000000000000000
000000000000000000000000000000000000000000000000
F03F000000000000F03F0000000000F03F000000000000F03
F0000000000000000000000000000000000000000000000');

  st_astext
-----
POLYGON((0 0,0 1,1 1,1 0,0 0))
```

默认情况下行全精度行出。

```
SELECT ST_AsText('POINT(111.1111111 1.1111111)');
      st_astext
-----
POINT(111.1111111 1.1111111)
```

*maxdecimaldigits* 参数用于限制输出精度。

```
SELECT ST_AsText('POINT(111.1111111 1.1111111)', 2);
      st_astext
-----
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 (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

返回几何图形的 OGC/ISO 已知二进制 (WKB) 表示形式。第一种形式默认为服务器计算机的字节序。第二种形式采用指定小端序 (“NDR”) 或大端序 (“XDR”) 的字符串。

WKB 格式由于从数据中提取几何数据并保持完整的数字精度非常有用。它避免了 WKT 等文本格式可能产生的精确舍入。

要从 WKB 到 PostGIS 几何图形的逆过程，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 9.0 中的默认行已更改以十六进制输出 bytea。如果您的 GUI 工具需要旧行，请在数据中设置 `SET bytea_output='escape'`。









示例

基本

```
SELECT ST_AsEncodedPolyline(GeomFromEWKT('SRID=4326;LINESTRING(-120.2 38.5,-120.95 40.7,-126.453 43.252)'));
-- result--
|_p~iF~ps|U_uLLnnqC_mqNvxq`@
```

与地理串和地理分段合使用，并放在谷歌地图上

```
-- the SQL for Boston to San Francisco, segments every 100 KM
SELECT ST_AsEncodedPolyline(
    ST_Segmentize(
        ST_GeogFromText('LINESTRING(-71.0519 42.4935,-122.4483 37.64)'),
        100000)::geometry) As encodedFlightPath;
```

javascript 看起来像，其中 \$ 量替果

```
<script type="text/javascript" src="http://maps.googleapis.com/maps/api/js?libraries=
  geometry"
></script>
<script type="text/javascript">
    flightPath = new google.maps.Polyline({
        path: google.maps.geometry.encoding.decodePath("$encodedFlightPath
        "),
        map: map,
        strokeColor: '#0000CC',
        strokeOpacity: 1.0,
        strokeWeight: 4
    });
</script>
```

相关信息

[ST\\_LineFromEncodedPolyline](#), [ST\\_Segmentize](#)

### 7.9.3.2 ST\_AsFlatGeobuf

ST\_AsFlatGeobuf — 返回一行的 FlatGeobuf 表示形式。

#### Synopsis

```
bytea ST_AsFlatGeobuf(anelement set row);
bytea ST_AsFlatGeobuf(anelement row, bool index);
bytea ST_AsFlatGeobuf(anelement 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 切空索引生成。默 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 的一行的 Geobuf 表示 (<https://github.com/mapbox/geobuf>)。分析每个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

```
SELECT encode(ST_AsGeobuf(q, 'geom'), 'base64')
  FROM (SELECT ST_GeomFromText('POLYGON((0 0,0 1,1 1,1 0,0 0))') AS geom) AS q;
 st_asgeobuf
-----
GAAiEAo0CgwIBBoIAAAAAGIAAAE=
```

### 7.9.3.4 ST\_AsGeoJSON

ST\_AsGeoJSON — 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 GeoJSON 中的 “geometry” 象，或者返回一行作 GeoJSON 中的 “feature” 象。

生成的 GeoJSON 几何体和要素遵循 GeoJSON 范 RFC 7946，但当解析的几何体使用 WGS84 度和度之外的 CRS (如 EPSG:4326, urn:ogc:def:crs:OGC::CRS84) 行引用，GeoJSON 几何象将默认附加一个短的 CRS SRID 符。支持 2D 和 3D 几何。GeoJSON 支持 SFS 1.1 几何型 (例如，不支持曲线)。

geom\_column 参数用于区分多个几何列。如果省略参数，将确定中的第一个几何列。相反，参数将省列型。

maxdecimaldigits 参数可用于少出中使用的最大小数位数 (默认 9)。如果您使用 EPSG:4326 并且出几何形用于示， maxdecimaldigits=6 于多地来可能是一个不的。



### 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。

`options` 参数可用于在 GeoJSON 输出中添加 BBOX 或 CRS :

- 0 : 表示没有选项
- 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, 只有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, 包括任何可能的列, 都将最位于要素的“properties”属性内。

GeoJSON 范围多边形使用右手定向, 并且某些客户端需要此方向。可以通过使用 `ST_ForcePolygonCCW` 来确保。范围要求几何图形位于 WGS84 坐标系 (SRID = 4326) 中。如果需要, 可以使用 `ST_Transform` 将几何图形投影到 WGS84 中: `ST_Transform(geom, 4326)`。

GeoJSON 可以在 [geojson.io](http://geojson.io) 和 [geojsonlint.com](http://geojsonlint.com) 上在线查看。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 支持输出作输入

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



函数支持 3d 并且不会丢失 z-index。

示例

生成特征集合:

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

```
{
  "type": "FeatureCollection",
  "features": [
    {
      "type": "Feature",
      "geometry": {
        "type": "Point",
        "coordinates": [1, 1]
      },
      "id": 1,
      "properties": {
        "name": "one"
      }
    },
    {
      "type": "Feature",
      "geometry": {
        "type": "Point",
        "coordinates": [2, 2]
      },
      "id": 2,
      "properties": {
        "name": "two"
      }
    },
    {
      "type": "Feature",
      "geometry": {
        "type": "Point",
        "coordinates": [3, 3]
      },
      "id": 3,
      "properties": {
        "name": "three"
      }
    }
  ]
}
```

生成一个特征：

```
SELECT ST_AsGeoJSON(t.*, id_column =
> 'id')
FROM (VALUES (1, 'one', 'POINT(1 1)::geometry)) AS t(id, name, geom);
```

st\_asgeojson

```
{
  "type": "Feature",
  "geometry": {
    "type": "Point",
    "coordinates": [1, 1]
  },
  "id": 1,
  "properties": {
    "name": "one"
  }
}
```

不要忘记将数据从 WGS84 度、度以符合 GeoJSON 范：

```
SELECT ST_AsGeoJSON(ST_Transform(geom, 4326)) from fe_edges limit 1;
```

st\_asgeojson

```
{
  "type": "MultiLineString",
  "coordinates": [
    [
      [
        [-89.734634999999997, 31.492072000000000],
        [-89.734955999999997, 31.492237999999997]
      ]
    ]
  ]
}
```

支持 3D 几何形状：

```
SELECT ST_AsGeoJSON('LINESTRING(1 2 3, 4 5 6)');
```

```
{
  "type": "LineString",
  "coordinates": [
    [1, 2, 3],
    [4, 5, 6]
  ]
}
```

Options argument can be used to add BBOX and CRS in GeoJSON output:

```
SELECT ST_AsGeoJSON(ST_SetSRID('POINT(1 1)::geometry', 4326), 9, 4|1);
```

```
{
  "type": "Point",
  "crs": {
    "type": "name",
    "properties": {
      "name": "urn:ogc:def:crs:EPSG::4326"
    }
  },
  "bbox": [1.000000000, 1.000000000, 1.000000000, 1.000000000],
  "coordinates": [1, 1]
}
```

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。

增加: 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

增加: 选项 GML 3 引入了 2.1.0 id 支持。



### Note

选项 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：版本 2

```
SELECT ST_AsGML(ST_GeomFromText('POLYGON((0 0,0 1,1 1,1 0,0 0))',4326));
      st_asgml
-----
      <gml:Polygon srsName="EPSG:4326"
><gml:outerBoundaryIs
><gml:LinearRing
><gml:coordinates
>0,0 0,1 1,1 1,0 0,0</gml:coordinates
></gml:LinearRing
></gml:outerBoundaryIs
></gml:Polygon>
```

## 示例：版本 3

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

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

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

```
-- Polyhedral Example --
SELECT ST_AsGML(3, ST_GeomFromEWKT('POLYHEDRALSURFACE( ((0 0 0, 0 0 1, 0 1 1, 0 1 0, 0 0 0) ↔
),
((0 0 0, 0 1 0, 1 1 0, 1 0 0, 0 0 0)), ((0 0 0, 1 0 0, 1 0 1, 0 0 1, 0 0 0)),
((1 1 0, 1 1 1, 1 0 1, 1 0 0, 1 1 0)),
((0 1 0, 0 1 1, 1 1 1, 1 1 0, 0 1 0)), ((0 0 1, 1 0 1, 1 1 1, 0 1 1, 0 0 1)) )' ));
      st_asgml
```

```

-----
<gml:PolyhedralSurface>
<gml:polygonPatches>
  <gml:PolygonPatch>
    <gml:exterior>
      <gml:LinearRing>
        <gml:posList srsDimension="3"
>0 0 0 0 0 1 0 1 1 0 1 0 0 0 0 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 1 0 0 1 0 0</gml:posList>
                </gml:LinearRing>
              </gml:exterior>
            </gml:PolygonPatch>
            <gml:PolygonPatch>
              <gml:exterior>
                <gml:LinearRing>
                  <gml:posList srsDimension="3"
>0 0 1 1 0 1 1 1 1 0 1 1 0 0 1</gml:posList>
                  </gml:LinearRing>
                </gml:exterior>
              </gml:PolygonPatch>
            </gml:polygonPatches>
          </gml:PolyhedralSurface>

```

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 Keyhole 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。



#### Note

需要在 Proj 支持下 PostGIS。使用 `PostGIS_Full_Version` 确认您已安装支持。



#### Note

可用性：1.2.2 - 包含版本参数的更高版本出现在 1.3.2 中



#### Note

增加：2.0.0 - 添加命名空间，使用默认和命名参数



#### Note

更改：3.0.0 - 除了“版本化”体名



#### Note

AsKML 出不适用于没有 SRID 的几何图形



函数支持 3d 并且不会失 z-index。

示例

```
SELECT ST_AsKML(ST_GeomFromText('POLYGON((0 0,0 1,1 1,1 0,0 0))',4326));

      st_askml
-----
<Polygon
><outerBoundaryIs
><LinearRing
><coordinates
>0,0 0,1 1,1 1,0 0,0</coordinates
></LinearRing
></outerBoundaryIs
></Polygon>

--3d linestring
SELECT ST_AsKML('SRID=4326;LINESTRING(1 2 3, 4 5 6)');
      <LineString
><coordinates
>1,2,3 4,5,6</coordinates
></LineString>
```

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。有效字符“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。

```
SELECT (ST_AsLatLonText('POINT (-3.2342342 -2.32498)'));
      st_aslatlon
-----
2°19'29.928"S 3°14'3.243"W
```

指定格式（与默认相同）。

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

除 D、M、S 和 C 以外的字符通。

```
SELECT (ST_AsLatLonText('POINT (-3.2342342 -2.32498)', 'D degrees, M minutes, S seconds to
      the C'));
      st_aslatlon
-----
2 degrees, 19 minutes, 30 seconds to the S 3 degrees, 14 minutes, 3 seconds to the W
```

有符号的度数而不是基本方向。

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

十进制度数。

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

大的被一化。

```
SELECT (ST_AsLatLonText('POINT (-302.2342342 -792.32498)'));
      st_aslatlon
-----
72°19'29.928"S 57°45'56.757"E
```

### 7.9.3.8 ST\_AsMARC21

ST\_AsMARC21 — 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='hddmmss' );
```

## 描述

此函数返回一个 MARC21/XML 文档，其中包含表示给定几何形状的边界框的制数数据。格式参数允许以 MARC21/XML 标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（默认）：hddmmss
- 度和度的十进制：hddd.dddddd
- 没有度和度的十进制：ddd.dddddd
- 有度和度的小数分：hddmm.mmmm
- 没有度和度的小数分：dddmm.mmmm
- 有度和度的小数秒：hddmmss.sss

小数点符号也可以是逗号，例如 hddmm, 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，因为它不受 MARC21/XML 标准（制数数据）支持。



### Note

MARC21/XML 标准没有提供任何方法来注制数数据的空参考系，这意味着这些信息在 MARC21/XML 后将会失。

## 示例

将 POINT 转换为 MARC21/XML，格式为 dd mm ss（默认）

```
SELECT ST_AsMARC21('SRID=4326;POINT(-4.504289 54.253312)::geometry');
          st_asmarc21
-----
<record xmlns="http://www.loc.gov/MARC21/slim">
  <datafield tag="034" ind1="1" ind2=" ">
    <subfield code="a"
>a</subfield>
    <subfield code="d"
>W0043015</subfield>
    <subfield code="e"
>W0043015</subfield>
    <subfield code="f"
>N0541512</subfield>
```

```

        <subfield code="g"
>N0541512</subfield>
      </datafield>
    </record>

```

将 POLYGON 十进制格式的马ARC21/XML

```

SELECT ST_AsMARC21('SRID=4326;POLYGON((-4.5792388916015625 54.18172660239091,
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 以十进制格式表示的马ARC21/XML。马ARC21/XML 中的几何序与其在集合中的序相。

```

SELECT ST_AsMARC21('SRID=4326;GEOMETRYCOLLECTION(POLYGON((13.1 52.65,13.516666666666667 52.65,13.516666666666667 52.38333333333333,13.1
52.38333333333333,13.1 52.65)),POINT(-4.5 54.25))::geometry,'hdddmm.
mmmm');

          st_asmarc21
-----
<record xmlns="http://www.loc.gov/MARC21/slim">
  <datafield tag="034" ind1="1" ind2="">
    <subfield code="a"
>a</subfield>
    <subfield code="d"
>E01307.0000</subfield>
    <subfield code="e"
>E01331.0000</subfield>
    <subfield code="f"
>N05240.0000</subfield>
    <subfield code="g"
>N05224.0000</subfield>
  </datafield>
<datafield tag="034" ind1="1" ind2="">

```

```

      <subfield code="a"
>a</subfield>
      <subfield code="d"
>W00430.0000</subfield>
      <subfield code="e"
>W00430.0000</subfield>
      <subfield code="f"
>N05415.0000</subfield>
      <subfield code="g"
>N05415.0000</subfield>
    </datafield>
  </record>

```

相关信息

[ST\\_GeomFromMARC21](#)

### 7.9.3.9 ST\_AsMVTGeom

ST\_AsMVTGeom — 将几何图形 MVT 瓦片的坐标空。

#### Synopsis

```
geometry ST_AsMVTGeom(geometry geom, box2d bounds, integer extent=4096, integer buffer=256,
boolean clip_geom=true);
```

#### 描述

将几何图形 MVT ([Mapbox 矢量切片](#)) 瓦片的坐标空，如果需要，将其剪切到边界。几何图形必须位于目标地的坐标系中（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。可能会导致结果几何体塌陷到低的度。

必须提供目标地坐标空中瓦片的矩形边界，以便可以几何图形，并在需要时进行裁剪。可以使用 [ST\\_TileEnvelope](#) 生成边界。

[ST\\_AsMVT](#) 函数用于将几何图形 MVT [ST\\_AsMVT](#) 所需的瓦片坐标空。

`geom` 是目标地坐标系中要几何图形。

`bounds` 是瓦片在地坐标空中的矩形边界，没有冲突。

`extent` 是由 [MVT 范定](#) 的切片坐标空中切片的大小。默 4096。

`buffer` 是切片坐标空中用于裁剪几何图形的缓冲区大小。默 256。

`Clip_geom` 是一个布尔值，用于控制几何图形是否按原样剪切或。默 true。

可用性：2.4.0



#### Note

从 3.0 开始，可以在配置 Wagyu 来剪和 MVT 多形。比 GEOS 默速度更快，生的结果更正确，但它可能会失小多形。



示例

```
SELECT ST_AsText(ST_AsMVTGeom(
    ST_GeomFromText('POLYGON ((0 0, 10 0, 10 5, 0 -5, 0 0))'),
    ST_MakeBox2D(ST_Point(0, 0), ST_Point(4096, 4096)),
    4096, 0, false));
           st_astext
-----
MULTIPOLYGON(((5 4096,10 4091,10 4096,5 4096)),((5 4096,0 4101,0 4096,5 4096)))
```

使用算的界来和裁剪几何形的 Web 墨卡托的范示例。

```
SELECT ST_AsMVTGeom(
    ST_Transform( geom, 3857 ),
    ST_TileEnvelope(12, 513, 412), extent =
> 4096, buffer =
> 64) AS geom
FROM data
WHERE geom && ST_TileEnvelope(12, 513, 412, margin =
> (64.0 / 4096))
```

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(anelement set row);
bytea ST_AsMVT(anelement row, text name);
bytea ST_AsMVT(anelement row, text name, integer extent);
bytea ST_AsMVT(anelement row, text name, integer extent, text geom_name);
bytea ST_AsMVT(anelement row, text name, integer extent, text geom_name, text feature_id_name);
```

描述

一个聚合函数，返回与的一行的二进制 **Mapbox 矢量瓦片** 表示。些行必包含一个几何列，几何列将被要素几何。几何形必位于坐空中，并且根据 **MVT 范** 有效。 **ST\_AsMVTGeom** 可用于将几何形坐空。其他行列被特征属性。

**Mapbox 矢量切片** 格式可以存具有不同属性集的元素。要使用此功能，在包含一深度的 **Json 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 || 接此函数的多个用来建具有多切片。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` 是范定的屏幕空其中的切片范。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 默认情况下会自行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` 或空数，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

```
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

将几何形作量矢量形 (SVG) 路径数据返回。使用 1 作第二个参数可以根据相移路径数据，默认为 (或 0) 使用移。第三个参数可用于少出中使用的最大十制位数 (默认为 15)。当 `'rel' arg = 0`，点几何形将渲染 `cx/cy`，当 `'rel' arg = 1`，将渲染 `x/y`。多点几何形由逗号 (",") 分隔，GeometryCollection 几何形由分号 (";") 分隔。

要使用 PostGIS SVG 形，看 [pg\\_svg](#)，它提供了用于理 `ST_AsSVG` 出的 `plpgsql` 函数。

增：3.4.0 支持所有曲型

更改：2.0.0 - 添加了默参数和命名参数的支持

**Note**

可用性: 1.2.2。可用性: 1.4.0 PostGIS 1.4.0 中更改 `l` 在 `l` 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 `l` 形字符串和曲线。

## 示例

```
SELECT ST_AsSVG('POLYGON((0 0,0 1,1 1,1 0,0 0))'::geometry);
```

```
st_assvg
```

```
-----
```

```
M 0 0 L 0 -1 1 -1 1 0 Z
```

## 弧

```
SELECT ST_AsSVG( ST_GeomFromText('CIRCULARSTRING(-2 0,0 2,2 0,0 2,2 4)') );
```

```
st_assvg
```

```
-----
```

```
M -2 0 A 2 2 0 0 1 2 0 A 2 2 0 0 1 2 -4
```

## 多曲线

```
SELECT ST_AsSVG('MULTICURVE((5 5,3 5,3 3,0 3),  
CIRCULARSTRING(0 0,2 1,2 2))'::geometry, 0, 0);
```

```
st_assvg
```

```
-----
```

```
M 5 -5 L 3 -5 3 -3 0 -3 M 0 0 A 2 2 0 0 0 2 -2
```

## 多曲面

```
SELECT ST_AsSVG('MULTISURFACE(  
CURVEPOLYGON(CIRCULARSTRING(-2 0,-1 -1,0 0,1 -1,2 0,0 2,-2 0),  
(-1 0,0 0.5,1 0,0 1,-1 0)),  
((7 8,10 10,6 14,4 11,7 8)))'::geometry, 0, 2);
```

```
st_assvg
```

```
-----
```

```
M -2 0 A 1 1 0 0 0 0 0 A 1 1 0 0 0 2 0 A 2 2 0 0 0 -2 0 Z
```

```
M -1 0 L 0 -0.5 1 0 0 -1 -1 0 Z
```

```
M 7 -8 L 10 -10 6 -14 4 -11 Z
```

**7.9.3.12 ST\_AsTWKB**

`ST_AsTWKB` — 返回几何形式 `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) 格式的几何图形。TWKB 是一种 **二进制的格式**，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近的位。如果要更高的精度，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图形的度和象界的可访问信息。默认情况下，它不是。除非您的客户端需要使用它，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)。

函数的数字输入形式用于将几何图形和唯一标识符的集合保留标识符的 TWKB 集合。对于希望解集合然后有关内部图形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形式相同。



### Note

格式规范可在 <https://github.com/TWKB/Specification> 在输出，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

```
SELECT ST_AsTWKB('LINESTRING(1 1,5 5)::geometry');
          st_astwkb
-----
\x0200020202020808
```

要构建包含标识符的聚合 TWKB 图形，首先使用 “array\_agg()” 聚合所需的几何图形和图形，然后用适当的 TWKB 函数。

```
SELECT ST_AsTWKB(array_agg(geom), array_agg(gid)) FROM mytable;
          st_astwkb
-----
\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 点元素格式的几何图形：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。这些是我们已经确定的映射。如果您对我们允许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

“srid” 参数是一个位字段。用于 PostGIS 2.2, 用于表示是否用 X3D GeoCooperatives 地理空间点表示坐标以及是否翻转 x/y。默认情况下, ST\_AsX3D 以数据类型形式输出 (long, lat 或 X, Y), 但 X3D 默认为 lat/lon, y/x 可能是首选。

- 0 : 数据类型顺序中的 X/Y (例如, 经/纬度 = X,Y 是标准数据类型顺序)、默认和非空坐标 (只是常规的旧坐标)。
- 1 : 翻转 X 和 Y。如果与地理空间开关组合使用, 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 投影 (srid: 4326) 中, 此类型将索引。这是目前唯一支持的地理空间类型。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	尚未实现 - 将是 PolyLine2D	LineSet
MULTILINESTRING	尚未实现 - 将是 PolyLine2D	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	X3D 类型无效	IndexedFaceSet (内当前输出一个面集)
TIN	TriangleSet2D (尚未实现)	IndexedTriangleSet (索引三角形集)



Note

2D 几何支持尚未完成。内当前限制制独特的多边形。我们正在研究这些。

3D 空间中诞生了多步, 特别是 X3D 与 HTML5 的集成

有一个很好的开源 X3D 查看器, 您可以使用它来查看渲染的几何图形。Free Wrl <http://freewrl.sourceforge.net/> - 二进制文件适用于 Mac、Linux 和 Windows。使用打包的 FreeWRL\_Launcher 来查看几何图形。

查看器利用此功能的 PostGIS 类型 X3D 查看器和 x3dDom html/js 开源工具包。

可用性 : 2.0.0 : ISO-IEC-19776-1.2-X3DEncodings-XML

增加 : 2.2.0 : 添加了反地理坐标和 (x/y、经/纬度) 的支持。有关信息, 请参看。

- ✓ 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：☑️建一个功能☑️全的 **X3D** 文档 - 您正在生成一个可以在 **FreeWrl** 或其他 **X3D** ☑️看器中☑️看的多☑️数据集。

```
SELECT '<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE X3D PUBLIC "ISO//Web3D//DTD X3D 3.0//EN" "http://www.web3d.org/specifications/x3d ←
-3.0.dtd">
<X3D>
  <Scene>
    <Transform>
      <Shape>
        <Appearance>
          <Material emissiveColor='0 0 1' />
        </Appearance>
      </Shape>
    </Transform>
  </Scene>
</X3D>
>' As x3ddoc;

          x3ddoc
          -----
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE X3D PUBLIC "ISO//Web3D//DTD X3D 3.0//EN" "http://www.web3d.org/specifications/x3d ←
-3.0.dtd">
<X3D>
  <Scene>
    <Transform>
      <Shape>
        <Appearance>
          <Material emissiveColor='0 0 1' />
        </Appearance>
        <IndexedFaceSet coordIndex='0 1 2 3 -1 4 5 6 7 -1 8 9 10 11 -1 12 13 14 15 -1 16 17 ←
18 19 -1 20 21 22 23'>
          <Coordinate point='0 0 0 0 1 0 1 1 0 1 0 0 0 0 1 0 1 1 0 1 0 0 0 0 0 1 0 0 ←
1 0 1 0 0 1 1 1 0 1 1 1 1 0 1 1 0 0 0 1 0 0 1 1 1 1 1 1 1 0 0 0 1 1 0 1 1 1 ←
1 0 1 1' />
        </IndexedFaceSet>
      </Shape>
    </Transform>
  </Scene>
</X3D>
```

## PostGIS 建筑

将此☑️☑️的☑️出复制并粘☑️到x3d ☑️景☑️看器，然后☑️☑️ “☑️示”

```
SELECT string_agg('<Shape
>' || ST_AsX3D(ST_Extrude(geom, 0,0, i*0.5)) ||
  '<Appearance>
    <Material diffuseColor="' || (0.01*i)::text || ' 0.8 0.2" specularColor="' || ←
    (0.05*i)::text || ' 0 0.5"/>
  </Appearance>
</Shape
>', '')
FROM ST_Subdivide(ST_Letters('PostGIS'),20) WITH ORDINALITY AS f(geom,i);
```



PostGIS 分、形成的建筑物

示例：高度 3 个位、精度 6 位的八角棱

```
SELECT ST_AsX3D(
ST_Translate(
  ST_Force_3d(
    ST_Buffer(ST_Point(10,10),5, 'quad_segs=2')), 0,0,
    3)
  ,6) As x3dfrag;

x3dfrag
-----
<IndexedFaceSet coordIndex="0 1 2 3 4 5 6 7">
  <Coordinate point="15 10 3 13.535534 6.464466 3 10 5 3 6.464466 6.464466 3 5 10 3 6.464466 13.535534 3 10 15 3 13.535534 13.535534 3 " />
</IndexedFaceSet>
```

示例：**TIN**

```
SELECT ST_AsX3D(ST_GeomFromEWKT('TIN (((
  0 0 0,
  0 0 1,
  0 1 0,
  0 0 0
  )), ((
  0 0 0,
  0 1 0,
  1 1 0,
  0 0 0
  ))
  ))) As x3dfrag;

x3dfrag
-----
<IndexedTriangleSet index='0 1 2 3 4 5'
><Coordinate point='0 0 0 0 1 0 1 0 0 0 0 1 0 1 1 0' /></IndexedTriangleSet>
```

示例：合多串（孔的多形的界）

```
SELECT ST_AsX3D(
  ST_GeomFromEWKT('MULTILINESTRING((20 0 10,16 -12 10,0 -16 10,-12 -12 10,-20 0 10,-12 16 10,0 24 10,16 16 10,20 0 10),
  (12 0 10,8 8 10,0 12 10,-8 8 10,-8 0 10,-8 -4 10,0 -8 10,8 -4 10,12 0 10)))')
) As x3dfrag;
```

```

                x3dfrag
                -----
<IndexedLineSet coordIndex='0 1 2 3 4 5 6 7 0 -1 8 9 10 11 12 13 14 15 8'>
  <Coordinate point='20 0 10 16 -12 10 0 -16 10 -12 -12 10 -20 0 10 -12 16 10 0 24 10 16  ←
    16 10 12 0 10 8 8 10 0 12 10 -8 8 10 -8 0 10 -8 -4 10 0 -8 10 8 -4 10 ' />
</IndexedLineSet>

```

### 7.9.3.14 ST\_GeoHash

ST\_GeoHash — 返回几何图形的 GeoHash 表示形式。

#### Synopsis

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

#### 描述

计算几何图形的 **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。

具有非零范的非点几何也可以映射到 GeoHash 代。代的精度取决于几何的地理范。

如果未指定 `maxchars`，返回的 GeoHash 代适用于包含入几何图形的最小元格。Points 返回精度 20 个字符的 GeoHash (大足以容入的完整双精度)。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。大的几何形的精度低，小的几何形的精度高。由 GeoHash 代确定的框始包含入特征。

如果指定了 `maxchars` 参数，返回的 GeoHash 代最多包含指定数量的字符。它映射到入几何体的 (可能) 低精度表示。于非点几何体，算的起始点是几何体界框的中心点。

可用性 : 1.4.0



#### Note

ST\_GeoHash 要求入几何形采用地理 (度/度) 坐。



此方法支持形字符串和曲。

#### 示例

```
SELECT ST_GeoHash( ST_Point(-126,48) );
```

```

      st_geohash
-----
c0w3hf1s70w3hf1s70w3

```

```
SELECT ST_GeoHash( ST_Point(-126,48), 5);
```

```

      st_geohash
-----
c0w3h

```

```
-- This line contains the point, so the GeoHash is a prefix of the point code
```



```
SELECT ST_GeoHash('LINESTRING(-126 48, -126.1 48.1)::geometry');
```

```
st_geohash
-----
c0w3
```

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

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

column1 | column1 | overlaps
```

```

-----+-----+-----
      1 |      3 | t
      2 |      3 | f
(2 rows)

```

相关信息

[ST\\_Intersects](#), [ST\\_Extent](#), [|&>](#), [&>](#), [&<|](#), [&<](#), [~](#), [@](#)

### 7.10.1.2 &&(geometry,box2df)

`&&(geometry,box2df)` — 如果几何体的 (☐存的) 2D ☐界框与 2D 浮点精度☐界框 (BOX2DF) 相交, ☐返回 TRUE。

#### Synopsis

```
boolean &&( geometry A , box2df B );
```

描述

如果几何体 A 的☐存 2D ☐界框与 2D ☐界框 B 相交 (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☐☐☐浮点精度 2D ☐界框 (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☐☐范☐索引 (BRIN) 的支持。需要 PostgreSQL 9.5+。



此方法支持☐形字符串和曲☐。



☐函数支持多面体曲面。

示例

```
SELECT ST_Point(1,1) && ST_MakeBox2D(ST_Point(0,0), ST_Point(2,2)) AS overlaps;
```

```

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 (存在的) 2D `BOX2DF` 相交, 返回 TRUE。

#### Synopsis

```
boolean &&( box2df A , geometry B );
```

#### 描述

`&&` 算符在使用浮点精度, 如果 2D `BOX2DF` A 与几何体 B 的 2D `BOX2DF` 相交, 返回 TRUE。这意味着如果 A 是一个 (双精度) `box2d`, 它将在内部被转换为一个浮点精度的 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` 索引的支持。需要 PostgreSQL 9.5+。



此方法支持 `LINESTRING` 和 `CURVE`。



`&&` 函数支持多面体曲面。

#### 示例

```
SELECT ST_MakeBox2D(ST_Point(0,0), ST_Point(2,2)) && ST_Point(1,1) AS overlaps;

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 `BOX2DF` 彼此相交, 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**

`&&` 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 引入了范围索引 (BRIN) 的支持。需要 PostgreSQL 9.5+。



此方法支持几何字符串和曲线。



`&&` 函数支持多面体曲面。

## 示例

```
SELECT ST_MakeBox2D(ST_Point(0,0), ST_Point(2,2)) && ST_MakeBox2D(ST_Point(1,1), ST_Point(3,3)) AS overlaps;
```

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

## 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 边界框与 B 的 n 边界框相交，则返回 TRUE。

**Synopsis**

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

## 描述

`&&&` 运算符返回 TRUE，如果 A 的 n 边界框与 B 的 n 边界框相交。

**Note**

`&&&` 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☐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：**3D** ☐串

```
SELECT tbl1.column1, tbl2.column1, tbl1.column2 &&& tbl2.column2 AS overlaps_3d,
      tbl1.column2 && tbl2.column2 AS overlaps_2d
FROM ( VALUES
      (1, 'LINESTRING Z(0 0 1, 3 3 2)::geometry),
      (2, 'LINESTRING Z(1 2 0, 0 5 -1)::geometry)) AS tbl1,
( VALUES
      (3, 'LINESTRING Z(1 2 1, 4 6 1)::geometry)) AS tbl2;
```

column1	column1	overlaps_3d	overlaps_2d
1	3	t	t
2	3	f	t

示例：**XYM** ☐串

```
SELECT tbl1.column1, tbl2.column1, tbl1.column2 &&& tbl2.column2 AS overlaps_3zm,
      tbl1.column2 && tbl2.column2 AS overlaps_2d
FROM ( VALUES
      (1, 'LINESTRING M(0 0 1, 3 3 2)::geometry),
      (2, 'LINESTRING M(1 2 0, 0 5 -1)::geometry)) AS tbl1,
( VALUES
      (3, 'LINESTRING M(1 2 1, 4 6 1)::geometry)) AS tbl2;
```

column1	column1	overlaps_3zm	overlaps_2d
1	3	t	t
2	3	f	t

相关信息

**&&**

### 7.10.1.6 &&&(geometry,gidx)

**&&&(geometry,gidx)** — 如果几何体的 (☐存的) n ☐☐界框与 n ☐浮点精度☐界框 (GIDX) 相交, ☐返回 TRUE。

#### Synopsis

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

## 描述

`&&&` 运算符在使用浮点精度时，如果几何体 A 的  $n$ -D 边界框与  $n$ -D 边界框 B 相交，则返回 TRUE。这意味着如果 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

`&&&` 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 `&&&` 索引（BRIN）的支持。需要 PostgreSQL 9.5+。



此方法支持曲线字符串和曲面。



`&&&` 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

```
SELECT ST_MakePoint(1,1,1) &&& ST_3DMakeBox(ST_MakePoint(0,0,0), ST_MakePoint(2,2,2)) AS overlaps;
overlaps
-----
t
(1 row)
```

## 相关信息

[&&&\(gidx,geometry\)](#), [&&&\(gidx,gidx\)](#)

### 7.10.1.7 &&&(gidx,geometry)

`&&&(gidx,geometry)` — 如果  $n$  浮点精度边界框（GIDX）与几何体的（存在的） $n$  边界框相交，则返回 TRUE。

## Synopsis

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

## 描述

`&&&` 运算符在使用浮点精度时，如果  $n$ -D 边界框 A 与几何体 B 的  $n$ -D 边界框相交，则返回 TRUE。这意味着如果 A 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

`&&&` 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 `BRIN` 索引的支持。需要 PostgreSQL 9.5+。

- ✔ 此方法支持形字符串和曲面。
- ✔ 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

```
SELECT ST_3DMakeBox(ST_MakePoint(0,0,0), ST_MakePoint(2,2,2)) &&& ST_MakePoint(1,1,1) AS overlaps;
overlaps
-----
t
(1 row)
```

相关信息

[&&&\(geometry,gidx\)](#), [&&&\(gidx,gidx\)](#)

### 7.10.1.8 &&&(gidx,gidx)

`&&&(gidx,gidx)` — 如果  $n$  浮点精度界框 (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 界框 A 和 B 相交，返回 TRUE。意味着如果 A (或 B) 是一个 (双精度) `box3d`，它将在内部被一个浮点精度的 3D 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` 索引的支持。需要 PostgreSQL 9.5+。

- ✔ 此方法支持形字符串和曲面。
- ✔ 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

```
SELECT ST_3DMakeBox(ST_MakePoint(0,0,0), ST_MakePoint(2,2,2)) &&& ST_3DMakeBox(ST_MakePoint(1,1,1), ST_MakePoint(3,3,3)) AS overlaps;

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

## 相关信息

[&&&\(geometry,gidx\), &&&\(gidx,geometry\)](#)

**7.10.1.9 &<**

**&<** — 如果 A 的 ☐ 界框与 B 的 ☐ 界框重 ☐ 或位于其左 ☐，☐ 返回 TRUE。

**Synopsis**

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

## 描述

**&<** ☐ 算符返回 TRUE 如果几何体 A 的 ☐ 界框与几何体 B 的 ☐ 界框重 ☐ 或在几何体 B 的 ☐ 界框的左 ☐，或者更准确地 ☐，不位于几何体 B 的 ☐ 界框的右 ☐。

**Note**

☐ 操作符将利用几何上可能可用的任何索引。

## 示例

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

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



相关信息

[&&](#), [|&>](#), [&>](#), [&<](#)

### 7.10.1.10 &<|

`&<|` — 如果 A 的 `ST_Envelope` 与 B 的 `ST_Envelope` 重叠或低于 B 的 `ST_Envelope`，返回 TRUE。

#### Synopsis

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

描述

`&<|` 运算符返回 TRUE，如果几何体 A 的 `ST_Envelope` 与几何体 B 的 `ST_Envelope` 重叠或低于几何体 B 的 `ST_Envelope`，或者更准确地，不位于几何体 B 的 `ST_Envelope` 的上方。



此方法支持 `TEXT` 字符串和 `CURVE`。



`ST_Envelope` 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

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

column1	column1	overbelow
1	2	f
1	3	t
1	4	t

(3 rows)

相关信息

[&&](#), [|&>](#), [&>](#), [&<](#)

### 7.10.1.11 &>

`&>` — 如果 A 的 `ST_Envelope` 与 B 的 `ST_Envelope` 重叠或位于 B 右侧，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，更准确地说，它与几何体 B 的边界框不重叠于其左侧。

**Note!**

### Note

**&** 操作符将利用几何上可能可用的任何索引。

### 示例

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

column1	column1	overright
1	2	t
1	3	t
1	4	f

(3 rows)

### 相关信息

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

### 7.10.1.12 <<

**<<** — 如果 A 的边界框位于 B 的左侧，返回 **TRUE**。

## Synopsis

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

### 描述

如果 A 的边界框位于 B 的边界框的左侧，**<<** 算符返回 **TRUE**。

**Note!**

### Note

**<** 操作符将利用几何上可能可用的任何索引。

示例

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

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

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

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

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

```

      1 |      3 | f
      1 |      4 | f
(3 rows)

```

## 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⊠和坐⊠⊠序与几何/地理 B 的坐⊠和坐⊠⊠序相同，= ⊠算符返回 TRUE。PostgreSQL 使用⊠几何定⊠的 =、< 和 > ⊠算符来⊠行内部排序和比⊠几何⊠形（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⊠、相同⊠序的几何/地理才被⊠算符⊠⊠相等。⊠于“空⊠平等”，它忽略坐⊠⊠序等内容，并且可以⊠⊠以不同表示覆盖相同空⊠区域的要素，⊠使用 [ST\\_OrderingEquals](#) 或 [ST\\_Equals](#)



### Caution

此⊠算符不⊠几何⊠形使用任何有效索引。要使用索引⊠行⊠健的相等性⊠⊠，⊠使用 = 一和 &&。

更改：2.4.0，在之前的版本中，⊠是⊠界框相等而不是几何相等。如果需要⊠界框相等，⊠使用 `~=` 代替。



此方法支持⊠形字符串和曲⊠。



⊠函数支持多面体曲面。

## 示例

```

SELECT 'LINESTRING(0 0, 0 1, 1 0)::geometry = 'LINESTRING(1 1, 0 0)::geometry;
?column?
-----
f
(1 row)

SELECT ST_AsText(column1)
FROM ( VALUES
      ('LINESTRING(0 0, 1 1)::geometry'),
      ('LINESTRING(1 1, 0 0)::geometry')) AS foo;

```

```

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

-- Note: the GROUP BY uses the "=" to compare for geometry equivalency.
SELECT ST_AsText(column1)
FROM ( VALUES
      ('LINESTRING(0 0, 1 1)::geometry),
      ('LINESTRING(1 1, 0 0)::geometry)) AS foo
GROUP BY column1;
      st_astext
-----
LINESTRING(0 0,1 1)
LINESTRING(1 1,0 0)
(2 rows)

-- In versions prior to 2.0, this used to return true --
SELECT ST_GeomFromText('POINT(1707296.37 4820536.77)') =
      ST_GeomFromText('POINT(1707296.27 4820536.87)') As pt_intersect;

--pt_intersect --
f

```

相关信息

[ST\\_Equals](#), [ST\\_OrderingEquals](#), [~ =](#)

### 7.10.1.15 >>

>> — 如果 A 的☐界框☐格位于 B 的右☐，☐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

```

SELECT tbl1.column1, tbl2.column1, tbl1.column2
>
> tbl2.column2 AS right
FROM

```

```
( VALUES
  (1, 'LINESTRING (2 3, 5 6)::geometry)) AS tbl1,
( VALUES
  (2, 'LINESTRING (1 4, 1 7)::geometry),
  (3, 'LINESTRING (6 1, 6 5)::geometry),
  (4, 'LINESTRING (0 0, 4 3)::geometry)) AS tbl2;
```

column1	column1	right
1	2	t
1	3	f
1	4	f

(3 rows)

### 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

```
SELECT tbl1.column1, tbl2.column1, tbl1.column2 @ tbl2.column2 AS contained
FROM
```

```
( VALUES
  (1, 'LINESTRING (1 1, 3 3)::geometry)) AS tbl1,
( VALUES
  (2, 'LINESTRING (0 0, 4 4)::geometry),
  (3, 'LINESTRING (2 2, 4 4)::geometry),
  (4, 'LINESTRING (1 1, 3 3)::geometry)) AS tbl2;
```

column1	column1	contained
1	2	t
1	3	f
1	4	t

(3 rows)

相关信息

[~](#), [&&](#)

### 7.10.1.17 @([geometry](#),[box2df](#))

@([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。

#### Synopsis

```
boolean @( geometry A , box2df B );
```

描述

如果 A 几何  形的 2D  界框包含 2D  界框 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  浮点精度 2D  界框 (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  范  索引 (BRIN) 的支持。需要 PostgreSQL 9.5+。



此方法支持  形字符串和曲 。



函数支持多面体曲面。

示例

```
SELECT ST_Buffer(ST_GeomFromText('POINT(2 2)'), 1) @ ST_MakeBox2D(ST_Point(0,0), ST_Point(5,5)) AS is_contained;
```

```
is_contained
-----
t
(1 row)
```

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  界框 (BOX2DF) 包含在几何体的 2D  界框中,  返回 TRUE。

#### Synopsis

```
boolean @( box2df A , geometry B );
```

## 描述

如果 2D 几何体 A 使用浮点精度包含在 B 几何体的 2D 几何体中, `@` 运算符返回 `TRUE`。这意味着如果 B 是 (双精度) `box2d`, 它将在内部浮点精度 2D 几何体 (`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 索引 (BRIN) 的支持。需要 PostgreSQL 9.5+。



此方法支持几何字符串和曲线。



函数支持多面体曲面。

## 示例

```
SELECT ST_MakeBox2D(ST_Point(2,2), ST_Point(3,3)) @ ST_Buffer(ST_GeomFromText('POINT(1 1)')
, 10) AS is_contained;

is_contained
-----
t
(1 row)
```

## 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 浮点精度几何体 (`BOX2DF`) 包含在一个 2D 浮点精度几何体内, 返回 `TRUE`。

## Synopsis

```
boolean @( box2df A , box2df B );
```

## 描述

如果 2D 几何体 A 包含在 2D 几何体 B 中, `@` 运算符使用浮点精度返回 `TRUE`。这意味着如果 A (或 B) 是 (双精度) `box2d`, 它将在内部浮点精度 2D 几何体 (`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 索引 (BRIN) 的支持。需要 PostgreSQL 9.5+。



此方法支持几何字符串和曲线。



函数支持多面体曲面。



示例

```
SELECT ST_MakeBox2D(ST_Point(2,2), ST_Point(3,3)) @ ST_MakeBox2D(ST_Point(0,0), ST_Point(5,5)) AS is_contained;

is_contained
-----
t
(1 row)
```

相关信息

[ST\\_Contains\(geometry, box2df\)](#), [ST\\_Contains\(box2df, geometry\)](#), [ST\\_Contains\(box2df, box2df\)](#), [ST\\_Contains\(geometry, box2df\)](#), [ST\\_Contains\(box2df, geometry\)](#), [ST\\_Contains\(box2df, box2df\)](#), [ST\\_Contains\(geometry, box2df\)](#), [ST\\_Contains\(box2df, geometry\)](#)

### 7.10.1.20 |>

|> — 如果 A 的☐界框与 B 的☐界框重☐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☐界框与几何体 B 的☐界框重☐或位于其上方，更准确地☐，它与几何体 B 的☐界框不重☐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

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

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

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

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

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

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

```
SELECT tbl1.column1, tbl2.column1, tbl1.column2 ~ tbl2.column2 AS contains
FROM
  ( VALUES
    (1, 'LINESTRING (0 0, 3 3)::geometry)) AS tbl1,
  ( VALUES
    (2, 'LINESTRING (0 0, 4 4)::geometry),
    (3, 'LINESTRING (1 1, 2 2)::geometry),
    (4, 'LINESTRING (0 0, 3 3)::geometry)) AS tbl2;
```

column1	column1	contains
1	2	f
1	3	t
1	4	t

(3 rows)

## 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 ☐ 界框 (GIDX)，☐ 返回 TRUE。

## Synopsis

```
boolean ~( geometry A , box2df B );
```

## 描述

如果几何 ☐ 形 A 的 2D ☐ 界框包含 2D ☐ 界框 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 ☐☐☐ 浮点精度 2D ☐ 界框 (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 ☐☐范☐索引 (BRIN) 的支持。需要 PostgreSQL 9.5+。



此方法支持 ☐ 形字符串和曲 ☐。



☐ 函数支持多面体曲面。

## 示例

```
SELECT ST_Buffer(ST_GeomFromText('POINT(1 1)'), 10) ~ ST_MakeBox2D(ST_Point(0,0), ST_Point(
(2,2)) AS contains;

contains
-----
t
(1 row)
```

## 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 `BOX2DF` 包含几何体的 2D `BOX2DF`，返回 `TRUE`。

**Synopsis**

```
boolean ~( box2df A , geometry B );
```

## 描述

如果 2D `BOX2DF` A 包含 B 几何体的 `BOX2DF`，`~` 算符使用浮点精度返回 `TRUE`。这意味着如果 A 是（双精度）`BOX2DF`，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` 索引的支持。需要 PostgreSQL 9.5+。



此方法支持 `LINESTRING` 和 `CURVEPOLYGON`。



`~` 函数支持多面体曲面。

## 示例

```
SELECT ST_MakeBox2D(ST_Point(0,0), ST_Point(5,5)) ~ ST_Buffer(ST_GeomFromText('POINT(2 2)')
, 1) AS contains;

contains
-----
t
(1 row)
```

## 相关信息

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

### 7.10.1.25 ~(box2df,box2df)

`~(box2df,box2df)` — 如果 2D 浮点精度 `BOX2DF` 包含一个 2D 浮点精度 `BOX2DF`, 返回 TRUE。

## Synopsis

```
boolean ~( box2df A , box2df B );
```

## 描述

如果 2D 浮点精度 `BOX2DF` A 包含 2D 浮点精度 `BOX2DF` B, `~` 算符返回 TRUE。这意味着如果 A 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` 索引 (BRIN) 的支持。需要 PostgreSQL 9.5+。



此方法支持 `LINESTRING` 和 `CURVE`。



`~` 函数支持多面体曲面。

## 示例

```
SELECT ST_MakeBox2D(ST_Point(0,0), ST_Point(5,5)) ~ ST_MakeBox2D(ST_Point(2,2), ST_Point(↵
(3,3)) AS contains;
```

```
contains
-----
t
(1 row)
```

## 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 `BOX2DF` 与 B 的 `BOX2DF` 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。要了解您的数据中有些行，您可以运行下面的查询。要检查是否真正相等，请使用 `ST_OrderingEquals` 或 `ST_Equals`。

### 示例

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

### 相关信息

[ST\\_Equals](#), [ST\\_OrderingEquals](#), [=](#)

## 7.10.2 距离算符

### 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图形之间的二维距离。用于“ORDER BY”子句提供索引协助的最近邻果集。于 9.5 以下的 PostgreSQL 输出边界框的中心距离，而于 PostgreSQL 9.5，真正的 KNN 距离搜索输出几何图形之间的真正距离以及地理的球体距离。

**Note**

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**

当其中一个几何图形是常量（不在子查询/cte 中），索引才会生效。例如 `'SRID=3005;POINT(1011102 450541)::geometry` 而不是 `a.geom`

有关示例，参见 [PostGIS 研讨会：最近搜索](#)。

增强：2.2.0 - 几何和地理之间的 KNN (k 最近邻) 行是在真正的。注意，地理的 KNN 是在球面上计算的，而不是在球体平面上计算的。PostgreSQL 9.4 及更低版本支持地理，但不支持边界框的重心。

更改：2.2.0 - 在 PostgreSQL 9.5 中，旧的混合格式可能很慢。因此，如果您只想在 PostGIS 2.2 或更高版本和 PostgreSQL 9.5 或更高版本上行，可能需要消除方法。

可用性：2.0.0——弱 KNN 根据几何中心距离而不是真正距离提供最近邻。点的结果精确，所有其他类型的结果不精确。适用于 PostgreSQL 9.1+

## 示例

```
SELECT ST_Distance(geom, 'SRID=3005;POINT(1011102 450541)::geometry) as d,edabbr, vaabbr
FROM va2005
ORDER BY d limit 10;
```

d	edabbr	vaabbr
0	ALQ	128
5541.57712511724	ALQ	129A
5579.67450712005	ALQ	001
6083.4207708641	ALQ	131
7691.2205404848	ALQ	003
7900.75451037313	ALQ	122
8694.20710669982	ALQ	129B
9564.24289057111	ALQ	130
12089.665931705	ALQ	127
18472.5531479404	ALQ	002

(10 rows)

KNN 的原始答案是：

```
SELECT st_distance(geom, 'SRID=3005;POINT(1011102 450541)::geometry) as d,edabbr, vaabbr
FROM va2005
ORDER BY geom <-> 'SRID=3005;POINT(1011102 450541)::geometry limit 10;
```

d	edabbr	vaabbr
0	ALQ	128
5541.57712511724	ALQ	129A
5579.67450712005	ALQ	001

```

6083.4207708641 | ALQ | 131
7691.2205404848 | ALQ | 003
7900.75451037313 | ALQ | 122
8694.20710669982 | ALQ | 129B
9564.24289057111 | ALQ | 130
12089.665931705 | ALQ | 127
18472.5531479404 | ALQ | 002
(10 rows)

```

如果您运行“EXPLAIN ANALYZE”，您将看到第二个查询的性能有所提高。

对于 PostgreSQL < 9.5 的用途，使用混合索引来寻找真正最近点。首先使用索引辅助 KNN 行 CTE 查询，然后进行精确查询以得到正确的排序：

```

WITH index_query AS (
  SELECT ST_Distance(geom, 'SRID=3005;POINT(1011102 450541)::geometry) as d,edabbr, vaabbr
  FROM va2005
  ORDER BY geom <-> 'SRID=3005;POINT(1011102 450541)::geometry' LIMIT 100)
SELECT *
  FROM index_query
  ORDER BY d limit 10;

```

d	edabbr	vaabbr
0	ALQ	128
5541.57712511724	ALQ	129A
5579.67450712005	ALQ	001
6083.4207708641	ALQ	131
7691.2205404848	ALQ	003
7900.75451037313	ALQ	122
8694.20710669982	ALQ	129B
9564.24289057111	ALQ	130
12089.665931705	ALQ	127
18472.5531479404	ALQ	002

(10 rows)

相关信息

[ST\\_DWithin](#), [ST\\_Distance](#), [<#>](#)

### 7.10.2.2 `|=|`

`|=|` — 返回 A 和 B 轨迹在最接近点的距离。

#### Synopsis

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

描述

`|=|` 运算符返回两个轨迹之间的 3D 距离（参考 [ST\\_IsValidTrajectory](#)）。它与 [ST\\_DistanceCPA](#) 相同，但作运算符，它可用于使用 N 索引进行最近搜索（需要 PostgreSQL 9.5.0 或更高版本）。



**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 0 1)::geometry 而不是 a.geom

可用性：2.2.0。索引支持☐适用于 PostgreSQL 9.5+

## 示例

```
-- Save a literal query trajectory in a psql variable...
\set qt 'ST_AddMeasure(ST_MakeLine(ST_MakePointM(-350,300,0),ST_MakePointM(-410,490,0)) ↔
,10,20)'
```

```
-- Run the query !
SELECT track_id, dist FROM (
  SELECT track_id, ST_DistanceCPA(tr,:qt) dist
  FROM trajectories
  ORDER BY tr || :qt
  LIMIT 5
) foo;
track_id      dist
-----+-----
      395 | 0.576496831518066
      380 | 5.06797130410151
      390 | 7.72262293958322
      385 | 9.8004461358071
      405 | 10.9534397988433
(5 rows)
```

## 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☐界框之☐的距离，可能从空☐索引中☐取它☐（需要 PostgreSQL 9.1）。☐于☐行最近☐近似距离排序很有用。

**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——KNN ☒适用于 PostgreSQL 9.1+

示例

```
SELECT *
FROM (
SELECT b.tlid, b.mtfcc,
       b.geom <#
> ST_GeomFromText('LINESTRING(746149 2948672,745954 2948576,
                             745787 2948499,745740 2948468,745712 2948438,
                             745690 2948384,745677 2948319)',2249) As b_dist,
       ST_Distance(b.geom, ST_GeomFromText('LINESTRING(746149 2948672,745954
                             2948576,
                             745787 2948499,745740 2948468,745712 2948438,
                             745690 2948384,745677 2948319)',2249)) As act_dist
FROM bos_roads As b
ORDER BY b_dist, b.tlid
LIMIT 100) As foo
ORDER BY act_dist, tlid LIMIT 10;
```

tlid	mtfcc	b_dist	act_dist
85732027	S1400	0	0
85732029	S1400	0	0
85732031	S1400	0	0
85734335	S1400	0	0
85736037	S1400	0	0
624683742	S1400	0	128.528874268666
85719343	S1400	260.839270432962	260.839270432962
85741826	S1400	164.759294123275	260.839270432962
85732032	S1400	277.75	311.830282365264
85735592	S1400	222.25	311.830282365264

(10 rows)

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

`<<->>` 算符返回两个几何体之间的  $n$  维（欧几里德）距离。用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

当其中一个几何体是常量（不在子句/cte 中），索引才会生效。例如 `'SRID=3005;POINT(1011102 450541)::geometry` 而不是 `a.geom`

可用性：2.2.0——KNN 适用于 PostgreSQL 9.1+

### 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比。

**Note**

由于浮点数精度问题，几何处理后，几何体不会是您期望的那样相交。例如，到几何体上的线的最近点可能不位于线上。于这种情况，如果您希望将一厘米的距离相交，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**

```
SELECT ST_3DIntersects(pt, line), ST_Intersects(pt, line)
FROM (SELECT 'POINT(0 0 2)::geometry As pt, 'LINESTRING (0 0 1, 0 2 3)::geometry As ↵
      line) As foo;
 st_3dintersects | st_intersects
-----+-----
 f                | t
(1 row)
```

**TIN 示例**

```
SELECT ST_3DIntersects('TIN(((0 0 0,1 0 0,0 1 0,0 0 0)))::geometry, 'POINT(.1 .1 0):: ↵
      geometry);
 st_3dintersects
-----
 t
```

**相关信息**

[ST\\_3DDWithin](#), [ST\\_Intersects](#)

**7.11.1.2 ST\_Contains**

`ST_Contains` — 检查 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，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。

用数学公式表示： $ST\_Contains(A, B) \Leftrightarrow (A \supseteq B = B) \wedge (Int(A) \cap Int(B) \neq \emptyset)$

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，所以定义的一个微妙之处是多边形和不包含完全位于其边界内的点和点。有关更多信息，请参阅 [OGC 涵盖、包含、内部的微妙之处](#)。`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边界框比。要避免使用索引，请使用函数 `_ST_Contains`。

它是通过 GEOS 模块实现的

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。



### Important

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。

注意：是返回布尔值而不是整数的“允许”版本。



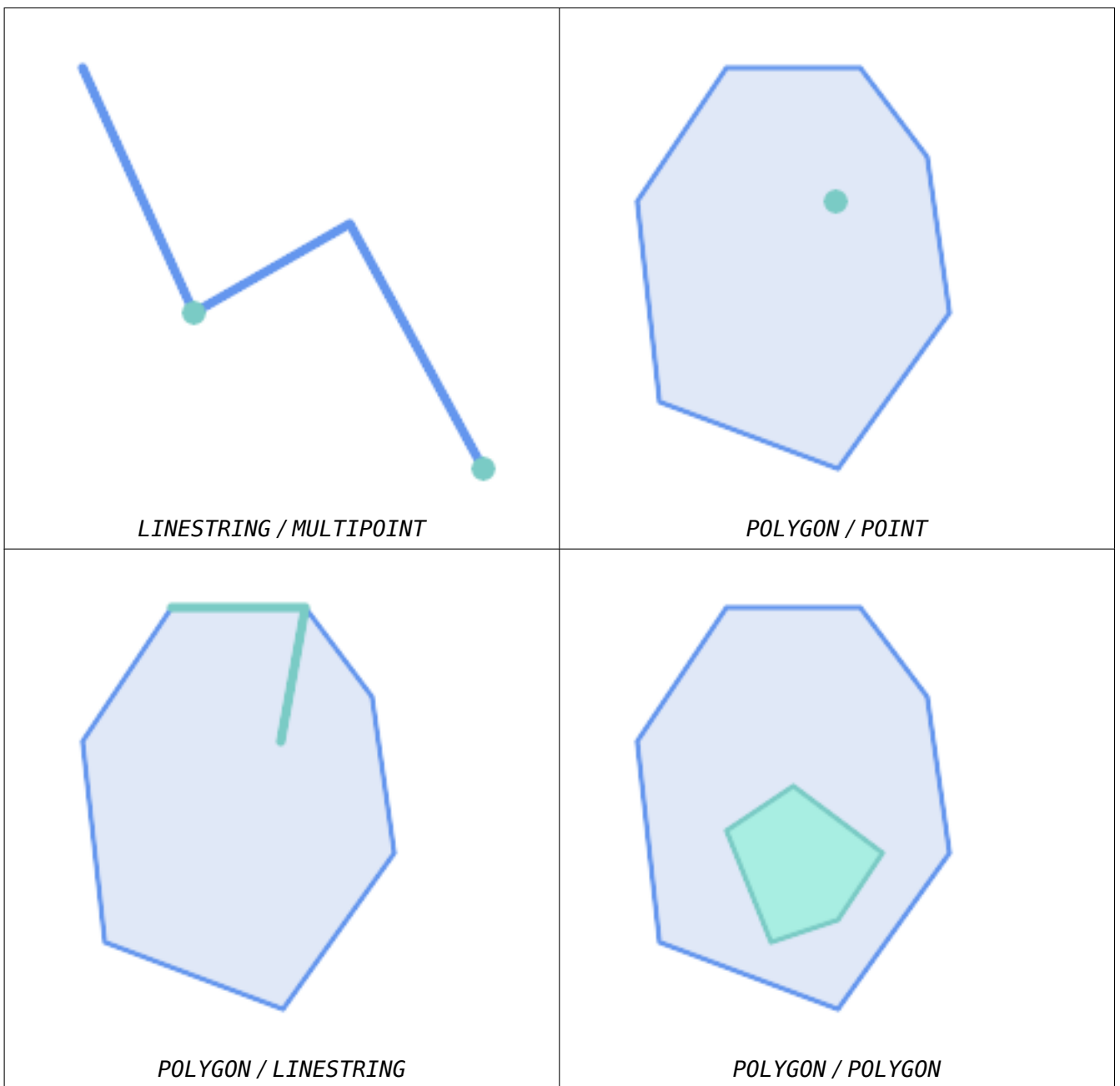
此方法符合了 [SQL 1.1 的 OGC 功能规范](#)。s2.1.1.2 // s2.1.13.3 - 与内部相同（几何 B、几何 A）



该方法符合了 [SQL/MM 规范](#)。SQL-MM 3: 5.1.31

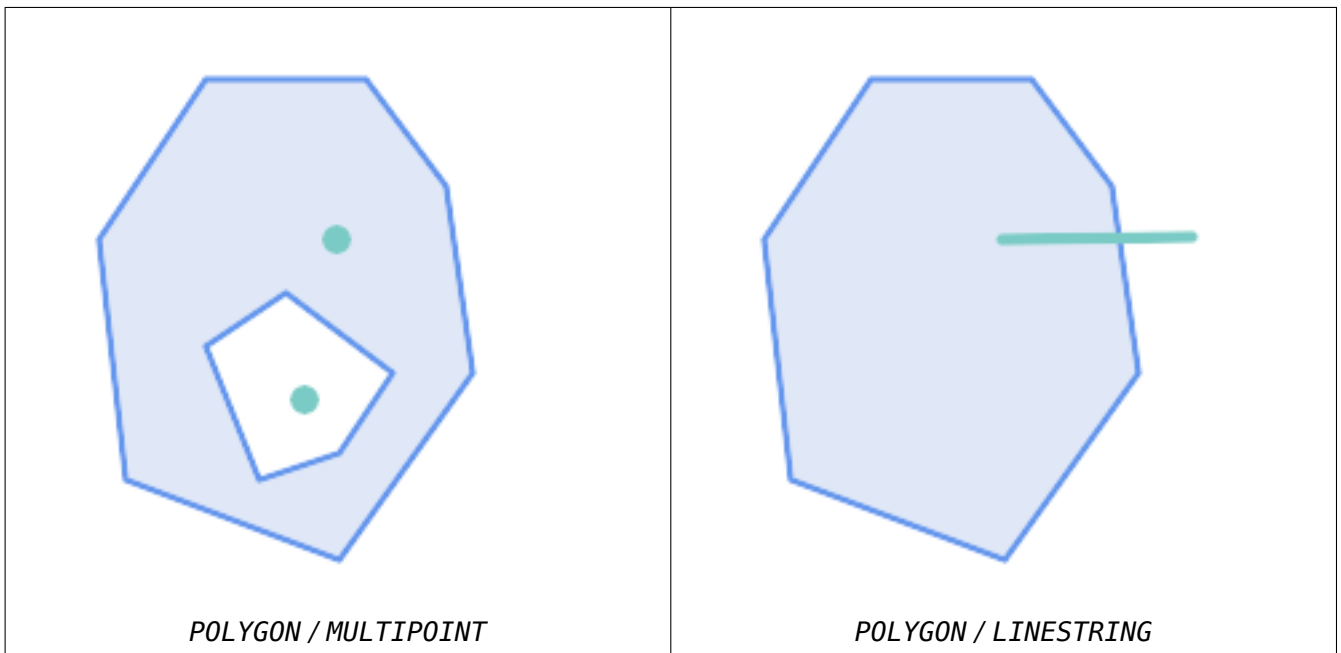
## 示例

`ST_Contains` 在以下情况下返回 TRUE：

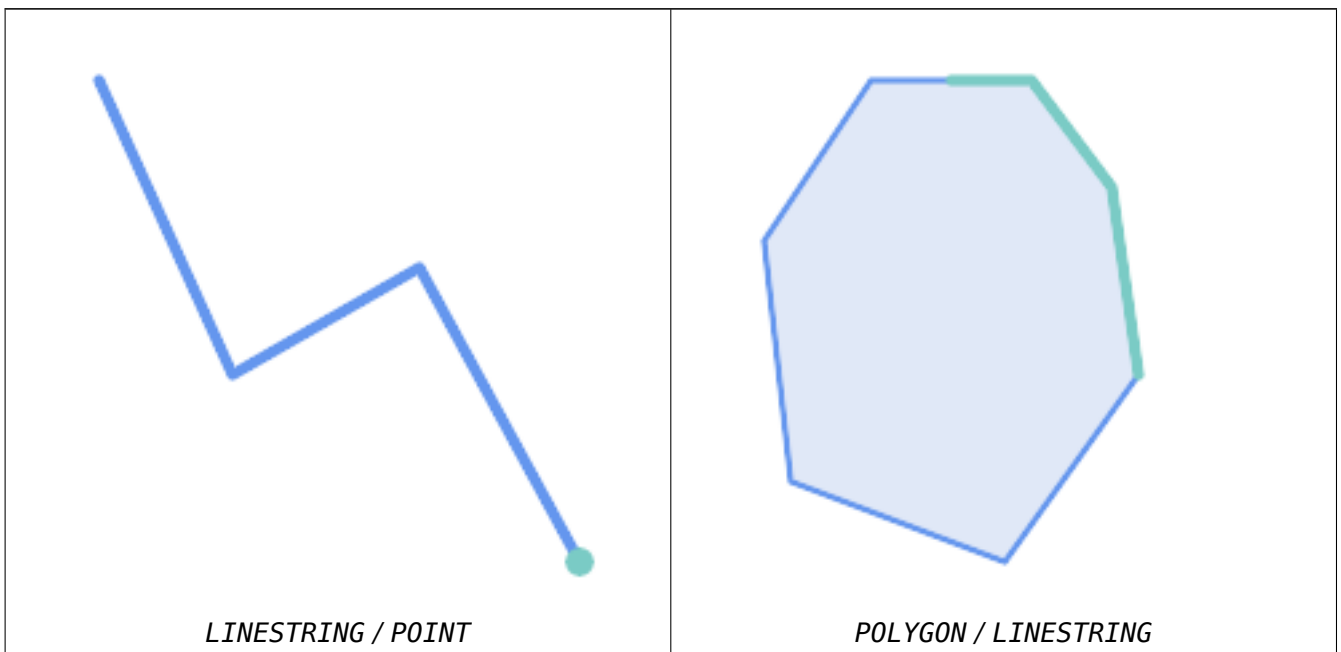


`ST_Contains` 在以下情况下返回 `FALSE` :

---



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 circle within a circle
SELECT ST_Contains(smallc, bigc) As smallcontainsbig,
       ST_Contains(bigc, smallc) As bigcontainssmall,
       ST_Contains(bigc, ST_Union(smallc, bigc)) as bigcontainsunion,
       ST_Equals(bigc, ST_Union(smallc, bigc)) as bigisunion,
       ST_Covers(bigc, ST_ExteriorRing(bigc)) As bigcoversexterior,
       ST_Contains(bigc, ST_ExteriorRing(bigc)) As bigcontainsexterior
FROM (SELECT ST_Buffer(ST_GeomFromText('POINT(1 2)'), 10) As smallc,
          ST_Buffer(ST_GeomFromText('POINT(1 2)'), 20) As bigc) As foo;
```

-- Result







**Note**

相交于 **ST\_Contains** 和 **ST\_Intersects** 的点是更有效地算，无需算各个点的拓扑。

这个函数是由 GEOS 模块行的。

可用性：1.4.0



**Important**

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。

示例

```
--a circle within a circle
SELECT ST_ContainsProperly(smallc, bigc) As smallcontainsprobig,
ST_ContainsProperly(bigc,smallc) As bigcontainsprosmall,
ST_ContainsProperly(bigc, ST_Union(smallc, bigc)) as bigcontainspropunion,
ST_Equals(bigc, ST_Union(smallc, bigc)) as bigisunion,
ST_Covers(bigc, ST_ExteriorRing(bigc)) As bigcoversexterior,
ST_ContainsProperly(bigc, ST_ExteriorRing(bigc)) As bigcontainsexterior
FROM (SELECT ST_Buffer(ST_GeomFromText('POINT(1 2)'), 10) As smallc,
ST_Buffer(ST_GeomFromText('POINT(1 2)'), 20) As bigc) As foo;
--Result
smallcontainsprobig | bigcontainsprosmall | bigcontainspropunion | bigisunion |
bigcoversexterior | bigcontainsexterior
-----+-----+-----+-----+-----+-----+
f | t | f | t |
| f | | |
--example demonstrating difference between contains and contains properly
SELECT ST_GeometryType(geomA) As geomtype, ST_Contains(geomA,geomA) AS acontainsa,
ST_ContainsProperly(geomA, geomA) AS acontainspropa,
ST_Contains(geomA, ST_Boundary(geomA)) As acontainsba, ST_ContainsProperly(geomA,
ST_Boundary(geomA)) As acontainspropba
FROM (VALUES ( ST_Buffer(ST_Point(1,1), 5,1) ),
( ST_MakeLine(ST_Point(1,1), ST_Point(-1,-1) ) ),
( ST_Point(1,1) )
) As foo(geomA);
geomtype | acontainsa | acontainspropa | acontainsba | acontainspropba
-----+-----+-----+-----+-----+
ST_Polygon | t | f | f | f
ST_LineString | t | f | f | f
ST_Point | t | t | f | f
```

相关信息

[ST\\_GeometryType](#), [ST\\_Boundary](#), [ST\\_Contains](#), [ST\\_Covers](#), [ST\\_CoveredBy](#), [ST\\_Equals](#), [ST\\_Relate](#), [ST\\_Within](#)



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）。

用数学公式表示： $ST\_Covers(A, B) \Leftrightarrow A \supseteq B = B$

ST\_Covers 与 [ST\\_CoveredBy](#) 相反。因此， $ST\_Covers(A, B) = ST\_CoveredBy(B, A)$ 。

一般来说，使用 `ST_Covers` 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 3.0.0 起包括利用几何上可用的任何空索引的边界框比。为了避免使用索引，使用函数 `_ST_Covers`。



#### Important

增强：3.0.0 增加了 GEOMETRYCOLLECTION 的支持



#### Important

请勿将此函数用于无效的几何形状。你会得到意想不到的结果。

它是通过 GEOS 模块实现的

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

增强：2.3.0 几何形状，PIP 短路（仅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。以前的版本不支持面和点集合。

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

```
--a circle covering a circle
SELECT ST_Covers(smallc,smallc) As smallinsmall,
       ST_Covers(smallc, bigc) As smallcoversbig,
       ST_Covers(bigc, ST_ExteriorRing(bigc)) As bigcoversexterior,
       ST_Contains(bigc, ST_ExteriorRing(bigc)) As bigcontainsexterior
FROM (SELECT ST_Buffer(ST_GeomFromText('POINT(1 2)'), 10) As smallc,
       ST_Buffer(ST_GeomFromText('POINT(1 2)'), 20) As bigc) As foo;
--Result
smallinsmall | smallcoversbig | bigcoversexterior | bigcontainsexterior
-----+-----+-----+-----
t             | f               | t                 | f
(1 row)
```

地理示例

```
-- a point with a 300 meter buffer compared to a point, a point and its 10 meter buffer
SELECT ST_Covers(geog_poly, geog_pt) As poly_covers_pt,
       ST_Covers(ST_Buffer(geog_pt,10), geog_pt) As buff_10m_covers_cent
FROM (SELECT ST_Buffer(ST_GeogFromText('SRID=4326;POINT(-99.327 31.4821)'), 300) As ←
       geog_poly,
       ST_GeogFromText('SRID=4326;POINT(-99.33 31.483)') As geog_pt ) As foo;

poly_covers_pt | buff_10m_covers_cent
-----+-----
f               | t
```

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）共同的内部点

#### 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”，返回 `true`；也就是说，几何图形有一些但不是所有的内部点是共同的。几何图形内部的交集必须非空，并且维度必须小于每个输入几何图形的最大维度，并且两个几何图形的交集不得等于任一几何图形。否则，返回 `false`。交叉关系是对称且非自反的。

用数学公式来写： $ST\_Crosses(A, B) \Leftrightarrow (dim(Int(A) \cap Int(B)) < \max(dim(Int(A)), dim(Int(B))) \wedge (A \not\sqsubset B \neq A) \wedge (A \not\sqsupset B \neq B))$

如果 DE-9IM 交集矩阵匹配，两个几何图形交叉：

- `T*T*****` 适用于点/点、点/面和面/面情况
- `T***T**` 用于面/点、区域/点和区域/面情况

- 0\*\*\*\*\* 用于☐/☐情况
- ☐于点/点和区域/区域情况，☐果是 false



**Note**

OpenGIS ☐☐要素☐范☐☐点/☐、点/面、☐/☐和☐/面情况定☐此☐☐。JTS / GEOS ☐展了定☐，也适用于☐/点、区域/点和区域/☐情况。☐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☐界框比☐。



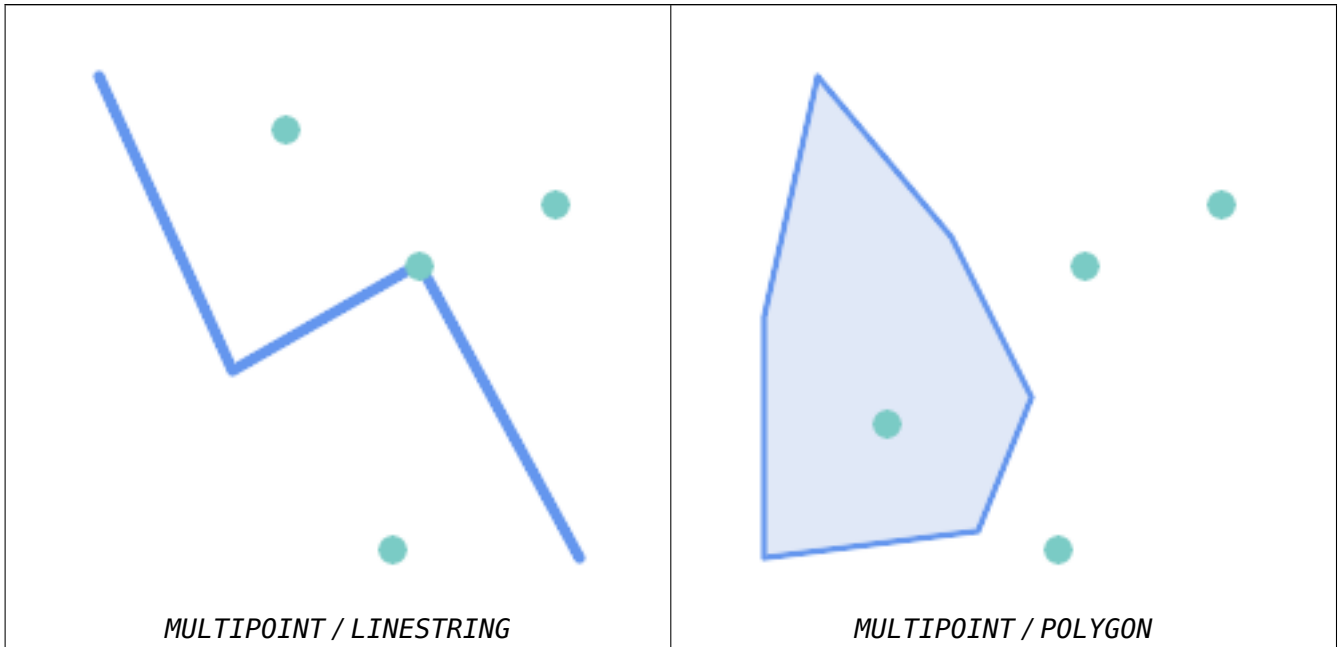
**Important**

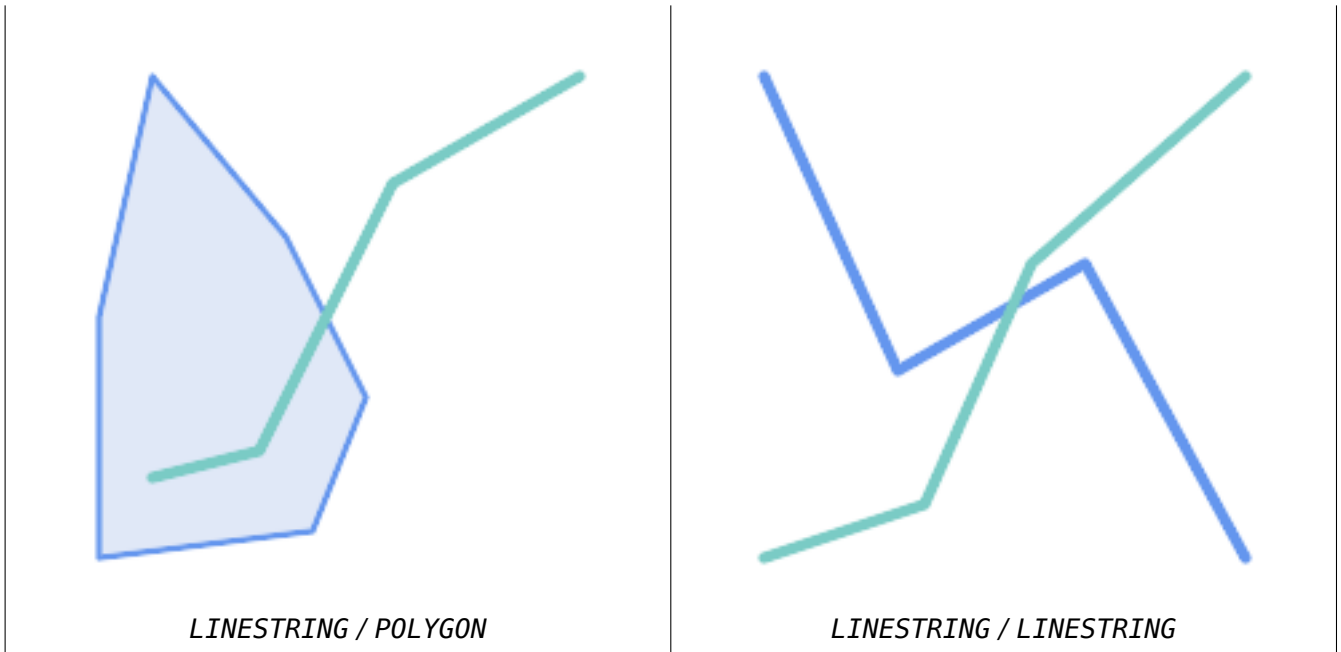
增☐：3.0.0 ☐用了☐ GEOMETRYCOLLECTION 的支持

- ✔ 此方法☐☐了 SQL 1.1 的 OGC ☐☐功能☐范。 s2.1.13.3
- ✔ ☐方法☐☐了 SQL/MM ☐范。 SQL-MM 3: 5.1.29

示例

以下情况均返回 true。





考虑用有 个表的情况：道路表和高速公路表。

<pre>CREATE TABLE roads (   id serial NOT NULL,   geom geometry,   CONSTRAINT roads_pkey PRIMARY KEY (     road_id) );</pre>	<pre>CREATE TABLE highways (   id serial NOT NULL,   the_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：

```
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，返回 `true`。如果几何图形没有共同点，那么它就是不相交的。

如果一个空关系返回 `TRUE`，两个几何不连接。如果未连接，`ST_Intersects` 返回 `FALSE`。

用数学公式来写： $ST\_Disjoint(A, B) \Leftrightarrow A \cap B = \emptyset$



### Important

增量：3.0.0 用了 `GEOMETRYCOLLECTION` 的支持

它是通过 `GEOS` 模块实现的



### Note

该函数不用不使用索引。否定的 `ST_Intersects` 也可用作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”版本。



此方法实现了 [SQL 1.1 的 OGC 功能规范](#)。s2.1.1.2 //s2.1.13.3 - a.Relate(b, 'FF\*FF\*\*\*\*')



该方法实现了 [SQL/MM 规范](#)。SQL-MM 3: 5.1.26

## 示例

```
SELECT ST_Disjoint('POINT(0 0)::geometry, 'LINESTRING ( 2 0, 0 2 ) '::geometry);
st_disjoint
-----
t
(1 row)
SELECT ST_Disjoint('POINT(0 0)::geometry, 'LINESTRING ( 0 0, 0 2 ) '::geometry);
st_disjoint
-----
f
(1 row)
```

## 相关信息

[ST\\_Intersects](#)

### 7.11.1.8 ST\_Equals

`ST_Equals` — 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： $ST\_Equals(A, B) \Leftrightarrow ST\_Within(A,B) \wedge ST\_Within(B,A)$



### Important

新增：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



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，如果它二进制相等，也会返回 true

### 示例

```
SELECT ST_Equals(ST_GeomFromText('LINESTRING(0 0, 10 10)'),
  ST_GeomFromText('LINESTRING(0 0, 5 5, 10 10)'));
 st_equals
-----
t
(1 row)

SELECT ST_Equals(ST_Reverse(ST_GeomFromText('LINESTRING(0 0, 10 10)'),
  ST_GeomFromText('LINESTRING(0 0, 5 5, 10 10)'));
 st_equals
-----
t
(1 row)
```

### 相关信息

[ST\\_IsValid](#), [ST\\_OrderingEquals](#), [ST\\_Reverse](#), [ST\\_Within](#)

#### 7.11.1.9 ST\_Intersects

**ST\_Intersects** — 检查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）。

用数学公式来：  $ST\_Intersects(A, B) \Leftrightarrow A \cap B \neq \emptyset$

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。

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 模块（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：是返回布尔而不是整数的“允”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 - ST\_Intersects(g1, g2 ) --> Not (ST\_Disjoint(g1, g2 ))

✓ 此方法符合了 SQL/MM 规范。SQL-MM 3: 5.1.27

✓ 此方法支持几何字符串和曲线。

✓ 此函数支持三角形和不规则三角网面 (TIN)。

## 几何示例

```

SELECT ST_Intersects('POINT(0 0)::geometry, 'LINESTRING ( 2 0, 0 2 ) '::geometry);
st_intersects
-----
f
(1 row)
SELECT ST_Intersects('POINT(0 0)::geometry, 'LINESTRING ( 0 0, 0 2 ) '::geometry);
st_intersects
-----
t
(1 row)

-- Look up in table. Make sure table has a GiST index on geometry column for faster lookup.
SELECT id, name FROM cities WHERE ST_Intersects(geom, 'SRID=4326;POLYGON((28 53,27.707 ←
52.293,27 52,26.293 52.293,26 53,26.293 53.707,27 54,27.707 53.707,28 53))');
id | name
----+-----
 2 | Minsk
(1 row)

```

## 几何示例

```

SELECT ST_Intersects(
  'SRID=4326;LINESTRING(-43.23456 72.4567,-43.23456 72.4568) '::geography,
  'SRID=4326;POINT(-43.23456 72.4567772) '::geography
);

st_intersects
-----
t

```

## 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：

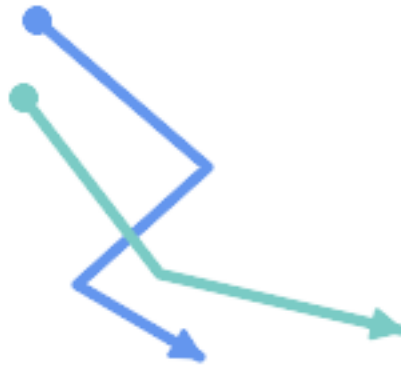
- 0：无交叉
- -1：向左交叉

- 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

有效性 : 1.4

示例

示例 : 向左交叉和向右交叉

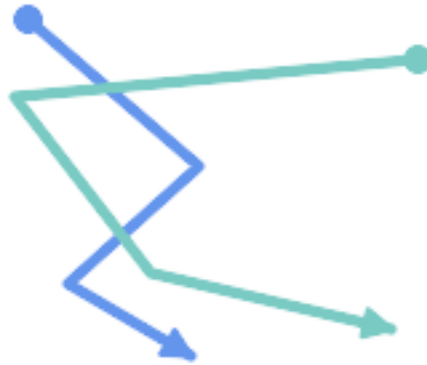


色 : A; 色 : B

```
SELECT ST_LineCrossingDirection(lineA, lineB) As A_cross_B,
       ST_LineCrossingDirection(lineB, lineA) As B_cross_A
FROM (SELECT
      ST_GeomFromText('LINESTRING(25 169,89 114,40 70,86 43)') As lineA,
      ST_GeomFromText('LINESTRING (20 140, 71 74, 161 53)') As lineB
    ) As foo;
```

A_cross_B	B_cross_A
-1	1

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

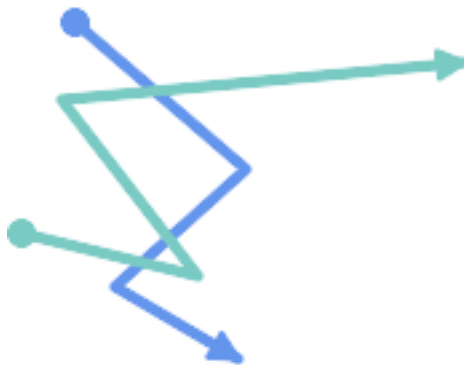


色 : A; 色 : B

```
SELECT ST_LineCrossingDirection(lineA, lineB) As A_cross_B,
       ST_LineCrossingDirection(lineB, lineA) As B_cross_A
FROM (SELECT
      ST_GeomFromText('LINESTRING(25 169,89 114,40 70,86 43)') As lineA,
      ST_GeomFromText('LINESTRING(171 154,20 140,71 74,161 53)') As lineB
    ) As foo;
```

A_cross_B	B_cross_A
3	-3

示例：一条段多次交叉并且最后一次交叉从左和一条段多次交叉并且最后一次交叉从右



色 : A; 色 : B

```
SELECT ST_LineCrossingDirection(lineA, lineB) As A_cross_B,
       ST_LineCrossingDirection(lineB, lineA) As B_cross_A
FROM (SELECT
      ST_GeomFromText('LINESTRING(25 169,89 114,40 70,86 43)') As lineA,
      ST_GeomFromText('LINESTRING(5 90, 71 74, 20 140, 171 154)') As lineB
    ) As foo;
```

```

A_cross_B | B_cross_A
-----+-----
      -2 |          2

```

示例: 查找所有交叉的街道

```

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序相同，返回 t (TRUE)；否则返回 f (FALSE)。



#### Note

此函数是根据 ArcSDE SQL 范而不是 SQL-MM 范的。[http://edndoc.esri.com/arcscde/9.1/sql\\_api/sqlapi3](http://edndoc.esri.com/arcscde/9.1/sql_api/sqlapi3)



方法符合了 SQL/MM 范。SQL-MM 3: 5.1.43

示例

```

SELECT ST_OrderingEquals(ST_GeomFromText('LINESTRING(0 0, 10 10)'),
  ST_GeomFromText('LINESTRING(0 0, 5 5, 10 10)'));
 st_orderingequals
-----
 f
(1 row)

SELECT ST_OrderingEquals(ST_GeomFromText('LINESTRING(0 0, 10 10)'),
  ST_GeomFromText('LINESTRING(0 0, 0 0, 10 10)'));
 st_orderingequals
-----

```

```
t
(1 row)

SELECT ST_OrderingEquals(ST_Reverse(ST_GeomFromText('LINESTRING(0 0, 10 10)'),
  ST_GeomFromText('LINESTRING(0 0, 0 0, 10 10)'));
 st_orderingequals
-----
f
(1 row)
```

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”，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），称两个几何重。重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，使用函数 `_ST_Overlaps`。

它是通过 GEOS 模块实现的



#### Important

增量：3.0.0 用了 GEOMETRYCOLLECTION 的支持

注意：是返回布尔而不是整数的“允许”版本。



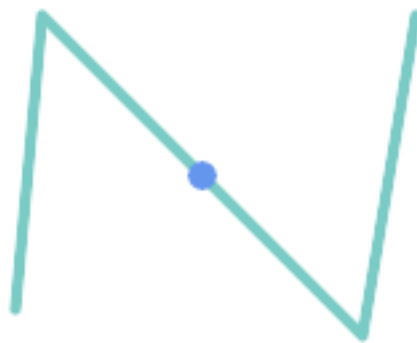
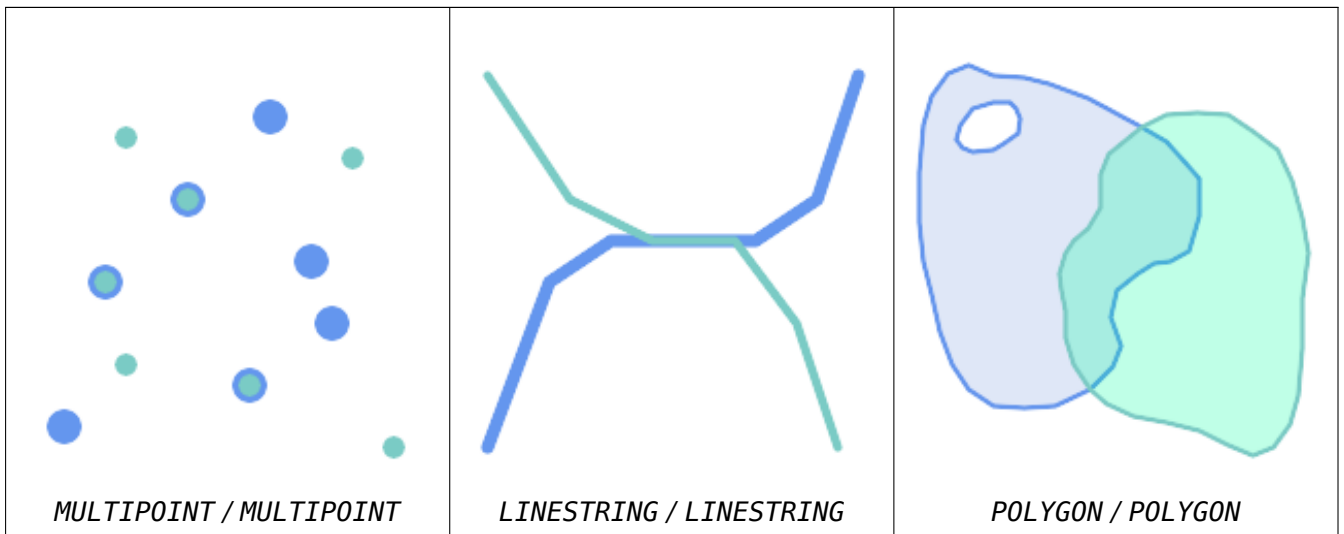
此方法符合 SQL 1.1 的 OGC 功能规范。s2.1.1.2 // s2.1.13.3



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：



包含 LineString 上的点，但由于它的精度低，因此不会重叠或交叉。

```
SELECT ST_Overlaps(a,b) AS overlaps,      ST_Crosses(a,b) AS crosses,
       ST_Intersects(a, b) AS intersects,  ST_Contains(b,a) AS b_contains_a
FROM (SELECT ST_GeomFromText('POINT (100 100)') As a,
       ST_GeomFromText('LINESTRING (30 50, 40 160, 160 40, 180 160)') AS b) AS t
```

overlaps	crosses	intersects	b_contains_a
f	f	t	t



部分覆盖多边形的线相交和交叉，但不重叠，因为它们具有不同的维度。

```
SELECT ST_Overlaps(a,b) AS overlaps,      ST_Crosses(a,b) AS crosses,
       ST_Intersects(a, b) AS intersects,  ST_Contains(a,b) AS contains
FROM (SELECT ST_GeomFromText('POLYGON ((40 170, 90 30, 180 100, 40 170))') AS a,
       ST_GeomFromText('LINESTRING(10 10, 190 190)') AS b) AS t;
```

overlap	crosses	intersects	contains
f	t	t	f



两个多边形相交，但彼此都不包含对方，它们重叠，但不交叉，因为它们交集具有相同的维度。

```
SELECT ST_Overlaps(a,b) AS overlaps,      ST_Crosses(a,b) AS crosses,
       ST_Intersects(a, b) AS intersects,  ST_Contains(b, a) AS b_contains_a,
       ST_Dimension(a) AS dim_a, ST_Dimension(b) AS dim_b,
       ST_Dimension(ST_Intersection(a,b)) AS dim_int
FROM (SELECT ST_GeomFromText('POLYGON ((40 170, 90 30, 180 100, 40 170))') AS a,
       ST_GeomFromText('POLYGON ((110 180, 20 60, 130 90, 110 180))') AS b) AS t;
```

overlaps	crosses	intersects	b_contains_a	dim_a	dim_b	dim_int
t	f	t	f	2	2	2



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，或计算它们的交集矩阵

#### 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识别特定类型的空关系。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。使用模式模式可以一步识别相交、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。原因是某些关系于不相交的几何图形是正确的（例如不相交）。如果您使用需要交集的关系模式，请在函数调用中包含 &&。



#### 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，他还可以完全相关估计所无法估计的性能优化。

形式 2：返回两个输入几何之间的空关系的 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。边界点允许更好地控制 MultiLineString 的端点是否被位于 DE-9IM 内部或边界内。boundaryNodeRule ：

- 1 : **OGC-Mod2** - 如果一条端点出奇数次，它位于边界内。它是 OGC SFS 定义的，也是 ST\_Relate 的默认。
- 2 : **Endpoint** - 所有端点都在边界内。
- 3 : **MultivalentEndpoint** - 如果端点出多次，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。语句，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



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

增加：2.0.0 - 添加了指定边界点的支持。



### Important

增加：3.0.0 用了 GEOMETRYCOLLECTION 的支持

### 示例

使用布罗伊函数空关系。

```
SELECT ST_Relate('POINT(1 2)', ST_Buffer( 'POINT(1 2)', 2), '0FFFFFF212');
st_relate
```

-----

t

```
SELECT ST_Relate('POINT(1 2)', ST_Buffer( 'POINT(1 2)', 2), '*FF*FF212');
st_relate
```

-----

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)
```

```
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。

```
SELECT ST_Relate( 'POINT(1 2)',
                  ST_Buffer( 'POINT(1 2)', 2));
```

-----

0FFFFFF212

```
SELECT ST_Relate( 'LINESTRING(1 2, 3 4)',
                  'LINESTRING(5 6, 7 8)' );
```

-----

FF1FF0102

使用不同的边界点来算具有重复端点的 LineString 和 MultiLineString 之空关系 (3 3) :

- 使用 **OGC-Mod2** (1)，重复端点位于 MultiLineString 的内部，因此 DE-9IM 矩条目 [aB:bI] 0, [aB:bB] 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`。

```
WITH data AS (SELECT
  'LINESTRING(1 1, 3 3)::geometry AS a_line,
  'MULTILINESTRING((3 3, 3 5), (3 3, 5 3)):: geometry AS b_multiline
)
SELECT ST_Relate( a_line, b_multiline, 1) AS bnr_mod2,
       ST_Relate( a_line, b_multiline, 2) AS bnr_endpoint
FROM data;
```

bnr_mod2	bnr_endpoint
FF10F0102	FF1F00102

## 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

检查度展九交模型 (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` 计算。

有关更多信息，参见 Section 5.1。

它是通过 GEOS 模型实现的。

可用性: 2.0.0

## 示例

```
SELECT ST_RelateMatch('101202FFF', 'TTTTTTFFF') ;
-- result --
t
```

用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

```
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**F****' )) AS pat(name,val)
CROSS JOIN
  (VALUES ( 'non-intersecting', 'FF1FF0212' ),
```

```
('overlapping', '1010F0212'),
('inside', '1FF0FF212')) AS mat(name,val);
```

relationship	pattern	position	matrix	match
Equality	T1FF1FFF1	non-intersecting	FF1FF0212	f
Equality	T1FF1FFF1	overlapping	1010F0212	f
Equality	T1FF1FFF1	inside	1FF0FF212	f
Overlaps	T*T***T**	non-intersecting	FF1FF0212	f
Overlaps	T*T***T**	overlapping	1010F0212	t
Overlaps	T*T***T**	inside	1FF0FF212	f
Within	T*F**F***	non-intersecting	FF1FF0212	f
Within	T*F**F***	overlapping	1010F0212	f
Within	T*F**F***	inside	1FF0FF212	t
Disjoint	FF*FF****	non-intersecting	FF1FF0212	t
Disjoint	FF*FF****	overlapping	1010F0212	f
Disjoint	FF*FF****	inside	1FF0FF212	f

相关信息

Section 5.1, [ST\\_Relate](#)

7.11.1.15 ST\_Touches

ST\_Touches — 两个几何形是否至少有一个共同点，但它内部不相交

Synopsis

boolean **ST\_Touches**(geometry A, geometry B);

描述

如果 A 和 B 相交，但它内部不相交，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界内。对于点/点输入，关系始终为 FALSE，因为点没有界。

In mathematical terms:  $ST\_Touches(A, B) \Leftrightarrow (Int(A) \cap Int(B) = \emptyset) \wedge (A \cap B \neq \emptyset)$

如果两个几何形的 DE-9IM 交集矩阵匹配以下之一，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`。



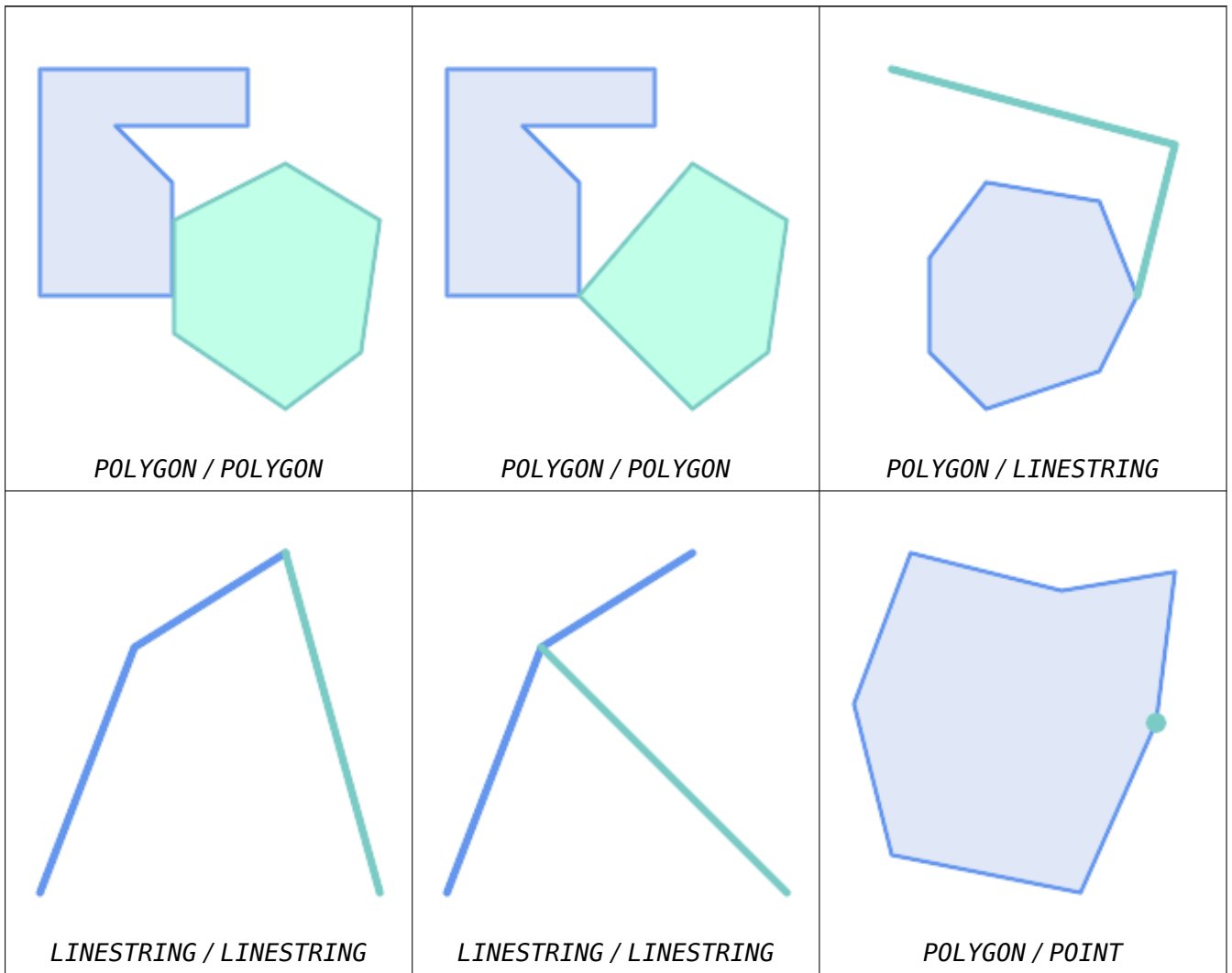
**Important**

增量：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。



```
SELECT ST_Touches('LINESTRING(0 0, 1 1, 0 2)::geometry, 'POINT(1 1)::geometry');
st_touches
-----
f
(1 row)

SELECT ST_Touches('LINESTRING(0 0, 1 1, 0 2)::geometry, 'POINT(0 2)::geometry');
st_touches
-----
t
(1 row)
```

### 7.11.1.16 ST\_Within

ST\_Within — 几何体 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

如果几何体 A 完全包含在几何体 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，源几何体必须具有相同的坐标投影，并具有相同的 SRID。

用数学公式表示： $ST\_Within(A, B) \Leftrightarrow (A \sqsubset B = A) \wedge (Int(A) \sqcap Int(B) \neq \emptyset)$

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  $ST\_Within(A, B) = true$  且  $ST\_Within(B, A) = true$ ，则两个几何体在拓扑上必相等 ( $ST\_Equals(A, B) = true$ )。

ST\_Within 与 **ST\_Contains** 相反。因此， $ST\_Within(A, B) = ST\_Contains(B, A)$ 。



#### Note

几何体内部必有一个公共点，所以定义的一个微妙之处在于，完全位于多边形或圆的边界内的点和点不在任何内部。有关更多信息，参见 [OGC 覆盖、包含、内部的微妙之处](#)。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边界框比。要避免使用索引，使用函数 `_ST_Within`。

它是通过 GEOS 模块实现的

增强：对于 2.3.0 几何体，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。以前的版本不支持面和点重合。



#### Important

增强：3.0.0 用了 GEOMETRYCOLLECTION 的支持



#### Important

请勿将此函数用于无效的几何体。你会得到意想不到的结果。

注意：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, 'T\*\*F\*\*F\*\*')



此方法符合了 [SQL/MM 规范](#)。SQL-MM 3: 5.1.30

示例

```
--a circle within a circle
SELECT ST_Within(smallc,smallc) As smallinsmall,
       ST_Within(smallc, bigc) As smallinbig,
       ST_Within(bigc,smallc) As biginsmall,
       ST_Within(ST_Union(smallc, bigc), bigc) as unioninbig,
       ST_Within(bigc, ST_Union(smallc, bigc)) as beginunion,
       ST_Equals(bigc, ST_Union(smallc, bigc)) as bigisunion
FROM
(
SELECT ST_Buffer(ST_GeomFromText('POINT(50 50)'), 20) As smallc,
       ST_Buffer(ST_GeomFromText('POINT(50 50)'), 40) As bigc) As foo;
--Result
smallinsmall | smallinbig | biginsmall | unioninbig | beginunion | bigisunion
-----+-----+-----+-----+-----+-----
t             | t           | f           | t           | t           | t
(1 row)
```



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 — 检查两个 3D 几何体是否在指定的 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轴位指定。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 2.0.0 起包括利用几何上可用的任何空索引的边界比。

- ✔ 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 ?

可用性: 2.0.0

## 示例

```
-- Geometry example - units in meters (SRID: 2163 US National Atlas Equal area) (3D point ↔
  and line compared 2D point and line)
-- Note: currently no vertical datum support so Z is not transformed and assumed to be same ↔
  units as final.
SELECT ST_3DDWithin(
  ST_Transform(ST_GeomFromEWKT('SRID=4326;POINT(-72.1235 42.3521 4)'),2163),
  ST_Transform(ST_GeomFromEWKT('SRID=4326;LINESTRING(-72.1260 42.45 15, -72.123 42.1546 ↔
    20)'),2163),
  126.8
) As within_dist_3d,
ST_DWithin(
  ST_Transform(ST_GeomFromEWKT('SRID=4326;POINT(-72.1235 42.3521 4)'),2163),
  ST_Transform(ST_GeomFromEWKT('SRID=4326;LINESTRING(-72.1260 42.45 15, -72.123 42.1546 ↔
    20)'),2163),
  126.8
) As within_dist_2d;

within_dist_3d | within_dist_2d
-----+-----
f              | t
```

## 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` — 检查两个 3D 几何体是否完全在指定的 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 并且不会失 z-index。



函数支持多面体曲面。

## 示例

```
-- This compares the difference between fully within and distance within as well
-- as the distance fully within for the 2D footprint of the line/point vs. the 3d fully
  within
  SELECT ST_3DDFullyWithin(geom_a, geom_b, 10) as D3DFullyWithin10, ST_3DDWithin(geom_a,
  geom_b, 10) as D3DWithin10,
  ST_DFullyWithin(geom_a, geom_b, 20) as D2DFullyWithin20,
  ST_3DDFullyWithin(geom_a, geom_b, 20) as D3DFullyWithin20 from
  (select ST_GeomFromEWKT('POINT(1 1 2)') as geom_a,
  ST_GeomFromEWKT('LINESTRING(1 5 2, 2 7 20, 1 9 100, 14 12 3)') as geom_b) t1;
d3dfullywithin10 | d3dwithin10 | d2dfullywithin20 | d3dfullywithin20
-----+-----+-----+-----
f                | t           | t                 | f
```

## 相关信息

[ST\\_3DDWithin](#), [ST\\_DWithin](#), [ST\\_DFullyWithin](#), [ST\\_3DMaxDistance](#)

### 7.11.2.3 ST\_DFullyWithin

ST\_DFullyWithin — Tests if a geometry is entirely inside a distance of another

#### Synopsis

boolean **ST\_DFullyWithin**(geometry g1, geometry g2, double precision distance);

## 描述

Returns true if g2 is entirely within distance of g1. Visually, the condition is true if g2 is contained within a distance buffer of g1. The distance is specified in units defined by the spatial reference system of the geometries.



### Note

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 : 1.1.0

Changed: 3.5.0 : the logic behind the function now uses a test of containment within a buffer, rather than the ST\_MaxDistance algorithm. Results will differ from prior versions, but should be closer to user expectations.

示例

```
SELECT
  ST_DFullyWithin(geom_a, geom_b, 10) AS DFullyWithin10,
  ST_DWithin(geom_a, geom_b, 10) AS DWithin10,
  ST_DFullyWithin(geom_a, geom_b, 20) AS DFullyWithin20
FROM (VALUES
  ('POINT(1 1)', 'LINESTRING(1 5, 2 7, 1 9, 14 12)')
 ) AS v(geom_a, geom_b)
```

dfullywithin10	dwithin10	dfullywithin20
f	t	t

相关信息

[ST\\_MaxDistance](#), [ST\\_DWithin](#), [ST\\_3DDWithin](#), [ST\\_3DDFullyWithin](#)

#### 7.11.2.4 ST\_DWithin

ST\_DWithin — 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位置指定。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：位置米，距离量默认使用 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

此函数调用包括利用几何上可用的任何索引的边界比。

 此方法扩展了 SQL 1.1 的 OGC 功能规范。

可用性：1.5.0 引入了地理的支持

增强：2.1.0 提高了地理速度。有关信息，参见 [使地理更快](#)。

增强：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` 一起使用来测量距离，而在 1.3.4 之前的版本中，`ST_Distance` 函数使用了多种方法。从 1.3.4 版本开始，`ST_DWithin` 使用更快速的短路距离函数。

示例

```
-- Find the nearest hospital to each school
-- that is within 3000 units of the school.
-- We do an ST_DWithin search to utilize indexes to limit our search list
-- that the non-indexable ST_Distance needs to process
-- If the units of the spatial reference is meters then units would be meters
SELECT DISTINCT ON (s.gid) s.gid, s.school_name, s.geom, h.hospital_name
FROM schools s
LEFT JOIN hospitals h ON ST_DWithin(s.geom, h.geom, 3000)
ORDER BY s.gid, ST_Distance(s.geom, h.geom);

-- The schools with no close hospitals
-- Find all schools with no hospital within 3000 units
-- away from the school. Units is in units of spatial ref (e.g. meters, feet, degrees)
SELECT s.gid, s.school_name
FROM schools s
LEFT JOIN hospitals h ON ST_DWithin(s.geom, h.geom, 3000)
WHERE h.gid IS NULL;

-- Find broadcasting towers that receiver with limited range can receive.
-- Data is geometry in Spherical Mercator (SRID=3857), ranges are approximate.

-- Create geometry index that will check proximity limit of user to tower
CREATE INDEX ON broadcasting_towers using gist (geom);

-- Create geometry index that will check proximity limit of tower to user
CREATE INDEX ON broadcasting_towers using gist (ST_Expand(geom, sending_range));

-- Query towers that 4-kilometer receiver in Minsk Hackerspace can get
-- Note: two conditions, because shorter LEAST(b.sending_range, 4000) will not use index.
SELECT b.tower_id, b.geom
FROM broadcasting_towers b
WHERE ST_DWithin(b.geom, 'SRID=3857;POINT(3072163.4 7159374.1)', 4000)
AND ST_DWithin(b.geom, 'SRID=3857;POINT(3072163.4 7159374.1)', b.sending_range);
```

相关信息

[ST\\_Distance](#), [ST\\_3DDWithin](#)

### 7.11.2.5 ST\_PointInsideCircle

`ST_PointInsideCircle` — 判断点几何形状是否位于由圆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` 为心的圆内，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，称 `ST_Point_Inside_Circle`

### 示例

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

### 相关信息

[ST\\_DWithin](#)

## 7.12 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。对于几何型，计算 2D 笛卡儿（平面）面积，位由 `SRID` 指定。对于地理型，默认面积是在球体上确定的，位平方米。要使用更快但精度低的球形模型算面积，使用 `ST_Area(geog, false)`。

增：2.0.0 - 引入了 2D 多面体曲面的支持。

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。

更改：3.0.0 - 不再依 `SFCGAL`。

- ✔ 此方法实现了 SQL 1.1 的 OGC 功能范。
- ✔ 方法实现了 SQL/MM 范。SQL-MM 3: 8.1.2, 9.5.3
- ✔ 函数支持多面体曲面。



**Note**

于多面体曲面，支持 2D 多面体曲面（不支持 2.5D）。于 2.5D，可能会出非零答案，但限于完全位于 XY 平面中的面。

示例

返回塞州一土地的面（以平方英尺位），然后乘以算得到平方米。注意，是以平方英尺位，因 EPSG:2249 是塞州平面英尺

```
select ST_Area(geom) sqft,
       ST_Area(geom) * 0.3048 ^ 2 sqm
from (
  select 'SRID=2249;POLYGON((743238 2967416,743238 2967450,
                            743265 2967450,743265.625 2967416,743238 2967416))' ::
         geometry geom
) subquery;

```

sqft	sqm
928.625	86.27208552

返回平方英尺并塞州平面米 (EPSG:26986) 以得平方米。注意，位平方英尺，因 2249 是塞州平面英尺，面的位是平方米，因 EPSG:26986 是塞州平面米

```
select ST_Area(geom) sqft,
       ST_Area(ST_Transform(geom, 26986)) As sqm
from (
  select
    'SRID=2249;POLYGON((743238 2967416,743238 2967450,
                        743265 2967450,743265.625 2967416,743238 2967416))' :: geometry geom
) subquery;

```

sqft	sqm
928.625	86.272430607008

使用地理数据类型返回平方英尺和平方米。注意，我将几何形地理形（在行此操作之前，确保您的几何形位于 WGS 84 度 4326 中）。地理是以米位。只是了演示比。通常您的表将已存在地理数据类型中。

```

select ST_Area(geog) / 0.3048 ^ 2 sqft_spheroid,
       ST_Area(geog, false) / 0.3048 ^ 2 sqft_sphere,
       ST_Area(geog) sqm_spheroid
from (
  select ST_Transform(
    'SRID=2249;POLYGON((743238 2967416,743238 2967450,743265
      2967450,743265.625 2967416,743238 2967416))'::geometry,
    4326
  ) :: geography geog
  ) as subquery;

```

如果您的数据已☑是地理数据：

```

select ST_Area(geog) / 0.3048 ^ 2 sqft,
       ST_Area(the_geog) sqm
from somegeogtable;

```

相关信息

[ST\\_3DArea](#), [ST\\_GeomFromText](#), [ST\\_GeographyFromText](#), [ST\\_SetSRID](#), [ST\\_Transform](#)

## 7.12.2 ST\_Azimuth

ST\_Azimuth — 返回☑点之☑直☑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☑☑位）；如果☑点重合，☑返回 NULL。方位角是从正 Y ☑（几何）或北子午☑（地理）参考的正☑☑☑角度：北 = 0；☑北 =  $\pi/4$ ；☑ =  $\pi/2$ ；☑南 =  $3\pi/4$ ；南 =  $\pi$ ；西南  $5\pi/4$ ；西 =  $3\pi/2$ ；西北 =  $7\pi/4$ 。

☑于地理☑型，方位角解决方案被称☑反解大地☑量☑☑。

方位角是一个数学概念，定义参考矢量与点之间的角度，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。

方位角可与 `ST_Translate` 结合使用，以沿其垂直移动对象。有关此功能的更多信息，请参考 [PostGIS wiki](#) 中的 `upgis_lineshift()` 函数。

可用性：1.1.0

新增：2.0.0 引入了地理的支持。

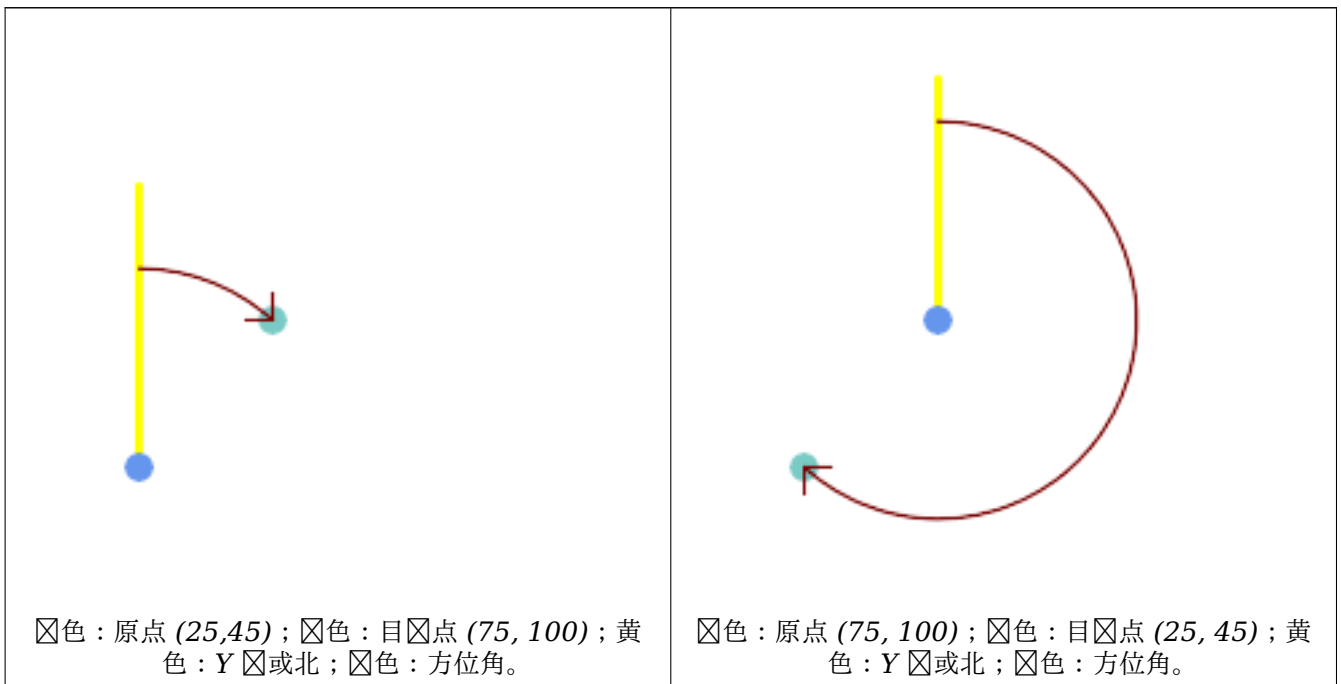
新增：2.2.0 使用 GeographicLib 球体行量，以提高准确性和健壮性。需要 PROJ  $\geq$  4.9.0 才能利用新功能。

## 示例

几何方位角（以度为单位）

```
SELECT degrees(ST_Azimuth( ST_Point(25, 45), ST_Point(75, 100))) AS degA_B,
       degrees(ST_Azimuth( ST_Point(75, 100), ST_Point(25, 45) )) AS degB_A;
```

degA_B	degB_A
42.2736890060937	222.273689006094



## 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 2 个向量之间的角度。

## 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  $2\pi$  弧度之间的正角。可以使用 PostgreSQL 函数 `Degrees()` 将弧度转换为度数。

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

```
SELECT degrees( ST_Angle('POINT(0 0)', 'POINT(10 10)', 'POINT(20 0)') );
```

```
degrees
-----
      270
```

由四个点定义的向量之间的角度

```
SELECT degrees( ST_Angle('POINT (10 10)', 'POINT (0 0)', 'POINT(90 90)', 'POINT (100 80)') ↔
);
```

```
degrees
-----
269.9999999999999
```

由线的起点和点定义的向量之间的角度

```
SELECT degrees( ST_Angle('LINESTRING(0 0, 0.3 0.7, 1 1)', 'LINESTRING(0 0, 0.2 0.5, 1 0)') ↔
);
```

```
degrees
-----
      45
```

### 相关信息

[ST\\_Azimuth](#)

## 7.12.4 ST\_ClosestPoint

`ST_ClosestPoint` — 返回 g1 上最接近 g2 的 2D 点。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。是几何形之最短的第一个点（由 [ST\\_ShortestLine](#) 算）。



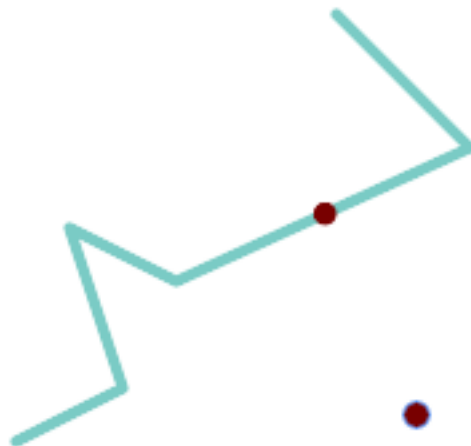
### Note

如果您有 3D 几何形，您可能更喜欢使用 [ST\\_3DClosestPoint](#)。

增：3.4.0 - 支持地理。

可用性：1.1.0

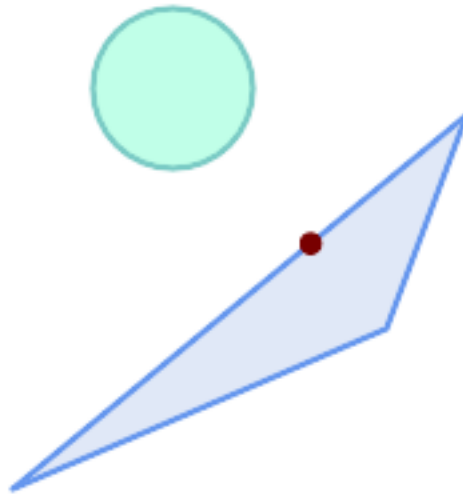
### 示例



*Point* 和 *LineString* 的最近点是点本身。*LineString* 和 *Point* 的最近点是线上的点。

```
SELECT ST_AsText( ST_ClosestPoint(pt,line)) AS cp_pt_line,
       ST_AsText( ST_ClosestPoint(line,pt)) AS cp_line_pt
FROM (SELECT 'POINT (160 40)::geometry AS pt,
            'LINESTRING (10 30, 50 50, 30 110, 70 90, 180 140, 130 190)::geometry AS line ) AS t;
```

cp_pt_line	cp_line_pt
POINT(160 40)	POINT(125.75342465753425 115.34246575342466)



多边形 A 上距离多边形 B 最近的点

```
SELECT ST_AsText( ST_ClosestPoint(
                    'POLYGON ((190 150, 20 10, 160 70, 190 150))',
                    ST_Buffer('POINT(80 160)', 30)
                )) As ptwkt;
-----
POINT(131.59149149528952 101.89887534906197)
```

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 并且不会丢失 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 0。

示例

## ☒串和点——3d 和 2d 最近点

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

```
cp3d_line_pt | ←
cp2d_line_pt
```

```
-----+-----
POINT(54.6993798867619 128.935022917228 11.5475869506606) | POINT(73.0769230769231 ←
115.384615384615)
```

## ☒串和多点——3d 和 2d 最近点

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

```
cp3d_line_pt | cp2d_line_pt
-----+-----
POINT(54.6993798867619 128.935022917228 11.5475869506606) | POINT(50 75)
```

## 多☒串和多☒形 3d 和 2d 最近点

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

```
cp3d | cp2d
-----+-----
POINT(39.993580415989 54.1889925532825 5) | POINT(20 40)
```

## 相关信息

[ST\\_AsEWKT](#), [ST\\_ClosestPoint](#), [ST\\_3DDistance](#), [ST\\_3DShortestLine](#)

**7.12.6 ST\_Distance**

ST\_Distance — 返回☒个几何或地理☒之☒的距离。

## Synopsis

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

### 描述

于 **geometry**, 返回两个几何之间的最小 2D 笛卡儿 (平面) 距离, 以投影单位 (空参考单位) 表示。

于 **geography**, 默认返回两个地理位置之间的最小大地距离 (以米为单位), 在由 SRID 确定的球体上行算。如果 `use_spheroid` 为 `false`, 使用更快的球面算。

✓ 此方法实现了 SQL 1.1 的 OGC 功能规范。

✓ 此方法实现了 SQL/MM 规范。SQL-MM 3: 5.1.23

✓ 此方法支持弧形字符串和曲线。

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

增: 2.1.0 提高了地理速度。有关信息, 参看 [使地理更快](#)。

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。

### 几何示例

几何示例 - 平面度数 4326 是 WGS 84 度, 单位是度。

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

几何示例 - 以米为单位 (SRID: 3857, 与流行网地面上的像素成比例)。尽管相关, 但可以正确比附近的, 使其成为 KNN 或 KMeans 等算法的不相关。

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

几何示例 - 单位米 (SRID: 3857, 如上所述, 但通过  $\cos(\text{lat})$  行校正以考虑失真)

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

几何示例 - 单位米 (SRID: 26986 度塞州平面米) (度塞州最准确)

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

几何示例 - 英尺 (SRID : 2163 美国国家大地集等面) (最不准确)

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

几何示例

与几何示例相同，但注意英尺 - 使用球体行稍快但不太准确的算。

```
SELECT ST_Distance(gg1, gg2) As spheroid_dist, ST_Distance(gg1, gg2, false) As sphere_dist
FROM (SELECT
  'SRID=4326;POINT(-72.1235 42.3521)::geography as gg1,
  'SRID=4326;LINESTRING(-72.1260 42.45, -72.123 42.1546)::geography as gg2
) As foo ;

spheroid_dist | sphere_dist
-----+-----
123.802076746848 | 123.475736916397
```

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` — 返回两个几何形之的 3D 笛卡最小距离 (基于空参考) (以投影位表示)。

### Synopsis

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

### 描述

返回两个几何形之的 3 最小笛卡距离 (以投影位 (空参考位) 表示)。

-  函数支持 3d 并且不会失 z-index。
-  函数支持多面体曲面。
-  方法了 SQL/MM 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: currently no vertical datum support so Z is not transformed and assumed to be same ←
  units as final.
SELECT ST_3DDistance(
    ST_Transform('SRID=4326;POINT(-72.1235 42.3521 4)::geometry,2163),
    ST_Transform('SRID=4326;LINESTRING(-72.1260 42.45 15, -72.123 ←
      42.1546 20)::geometry,2163)
  ) As dist_3d,
  ST_Distance(
    ST_Transform('SRID=4326;POINT(-72.1235 42.3521)::geometry,2163),
    ST_Transform('SRID=4326;LINESTRING(-72.1260 42.45, -72.123 42.1546) ←
      '::geometry,2163)
  ) As dist_2d;

  dist_3d      |      dist_2d
-----+-----
127.295059324629 | 126.66425605671
```

```
-- Multilinestring and polygon both 3d and 2d distance
-- Same example as 3D closest point example
SELECT ST_3DDistance(poly, mline) As dist3d,
  ST_Distance(poly, mline) As dist2d
  FROM (SELECT 'POLYGON((175 150 5, 20 40 5, 35 45 5, 50 60 5, 100 100 5, 175 150 5) ←
    )::geometry as poly,
    'MULTILINESTRING((175 155 2, 20 40 20, 50 60 -2, 125 100 1, 175 155 1), (1 ←
      10 2, 5 20 1))'::geometry as mline) as foo;

  dist3d      |      dist2d
-----+-----
0.716635696066337 | 0
```

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度/度几何形状之间的最小距离（以米为单位）。

### Synopsis

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

## 描述

返回两个度/度点之间的最小距离（以米为单位）。使用球形地球和从 SRID 定义的球体出的半径。比 [ST\\_DistanceSpheroid](#) 更快，但精度较差。PostGIS 1.5 之前的版本支持点。

可用性：1.5 - 引入了除点之外的其他几何型的支持。之前的版本适用于点。

更改：2.2.0 在之前的版本中，曾被称 [ST\\_Distance\\_Sphere](#)

## 示例

```
SELECT round(CAST(ST_DistanceSphere(ST_Centroid(geom), ST_GeomFromText('POINT(-118 38) ←
',4326)) As numeric),2) As dist_meters,
round(CAST(ST_Distance(ST_Transform(ST_Centroid(geom),32611),
ST_Transform(ST_GeomFromText('POINT(-118 38)', 4326),32611)) As numeric),2) ←
As dist_utm11_meters,
round(CAST(ST_Distance(ST_Centroid(geom), ST_GeomFromText('POINT(-118 38)', 4326)) As ←
numeric),5) As dist_degrees,
round(CAST(ST_Distance(ST_Transform(geom,32611),
ST_Transform(ST_GeomFromText('POINT(-118 38)', 4326),32611)) As numeric),2) ←
As min_dist_line_point_meters
FROM
(SELECT ST_GeomFromText('LINESTRING(-118.584 38.374,-118.583 38.5)', 4326) As geom) ←
as foo;
dist_meters | dist_utm11_meters | dist_degrees | min_dist_line_point_meters
-----+-----+-----+-----
70424.47 | 70438.00 | 0.72900 | 65871.18
```

## 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 [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

```
SELECT round(CAST(
    ST_DistanceSpheroid(ST_Centroid(geom), ST_GeomFromText('POINT(-118 38) ←
    ',4326), 'SPHEROID["WGS 84",6378137,298.257223563]')
    As numeric),2) As dist_meters_spheroid,
    round(CAST(ST_DistanceSphere(ST_Centroid(geom), ST_GeomFromText('POINT(-118 ←
    38)',4326)) As numeric),2) As dist_meters_sphere,
round(CAST(ST_Distance(ST_Transform(ST_Centroid(geom),32611),
    ST_Transform(ST_GeomFromText('POINT(-118 38)', 4326),32611)) As numeric),2) ←
    As dist_utm11_meters
FROM
    (SELECT ST_GeomFromText('LINESTRING(-118.584 38.374,-118.583 38.5)', 4326) As geom) ←
    as foo;
dist_meters_spheroid | dist_meters_sphere | dist_utm11_meters
-----+-----+-----
70454.92 | 70424.47 | 70438.00
```

相关信息

[ST\\_Distance](#), [ST\\_DistanceSphere](#)

### 7.12.10 ST\_FrechetDistance

ST\_FrechetDistance — 返回两个几何形之间的 Fréchet 距离。

#### Synopsis

```
float ST_FrechetDistance(geometry g1, geometry g2, float densifyFrac = -1);
```

描述

基于离散 Fréchet 距离，该算法限于几何形的离散点的 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。

当指定可选项的 densifyFrac 时，该函数在计算离散 Fréchet 距离之前对行分段致密化。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置。



#### Note

当前的 PostGIS 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

```
postgres=# SELECT st_frechetdistance('LINESTRING (0 0, 100 0)::geometry, 'LINESTRING (0 0, ↵
          50 50, 100 0)::geometry');
 st_frechetdistance
-----
          70.7106781186548
(1 row)
```

```
SELECT st_frechetdistance('LINESTRING (0 0, 100 0)::geometry, 'LINESTRING (0 0, 50 50, 100 ↵
          0)::geometry, 0.5);
 st_frechetdistance
-----
                    50
(1 row)
```

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。

该函数在上算“离散 Hausdorff 距离”。是在几何上的离散点算的 Hausdorff 距离。可以指定 *densifyFrac* 参数，以便在算离散 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长的子段，其段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置。



#### Note

该算法不等同于标准 Hausdorff 距离。然而，它算出的近似值对于大部分有用案例都是正确的。一种重要的情况是串彼此大致平行且长度大致相等。这是路匹配的有用指标。

可用性：1.1.0

示例



两条线之间的 Hausdorff 距离（红色）和距离（黄色）

```
SELECT ST_HausdorffDistance(geomA, geomB),
       ST_Distance(geomA, geomB)
FROM (SELECT 'LINESTRING (20 70, 70 60, 110 70, 170 70)::geometry AS geomA,
            'LINESTRING (20 90, 130 90, 60 100, 190 100)::geometry AS geomB) AS t;
st_hausdorffdistance | st_distance
-----+-----
37.26206567625497 |          20
```

示例：致密化的 Hausdorff 距离。

```
SELECT ST_HausdorffDistance(
    'LINESTRING (130 0, 0 0, 0 150)::geometry,
    'LINESTRING (10 10, 10 150, 130 10)::geometry,
    0.5);
-----
70
```

示例：对于每个建筑物，找到最能代表它的地块。首先，我们要求地块与建筑物几何形状相交。DISTINCT ON 保证我们每块建筑只列一次。ORDER BY .. ST\_HausdorffDistance 与建筑物最相似的地块。

```
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阶度。

## Synopsis

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

### 描述

用于几何型：如果几何是 `LineString`、`MultiLineString`、`ST_Curve`、`ST_MultiCurve`，返回几何的 2D 笛卡儿度。用于面几何形状，返回 0；改用 `ST_Perimeter`。度位由几何的空参考系确定。

用于地理型：使用逆地理算来行算。度位米。如果 PostGIS 是使用 PROJ 版本 4.8.0 或更高版本构建的，球体由 SRID 指定，否则它是 WGS84 独有的。如果 `use_spheroid = false`，算基于球体而不是球体。

目前用于几何体，是 `ST_Length2D` 的别名，但它可能会更改以支持更高的度。



### Warning

更改：2.0.0 重大更改 - 在之前的版本中，将此用于地理型的多/多边形将您提供多边形/多多边形的周。在 2.0.0 中，已更改返回 0 以符合几何行。如果您想要多边形的周，使用 `ST_Perimeter`



### Note

用于地理，算默认使用球体模型。要使用更快但不太准确的球面算，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



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

返回字符串的度（以英尺位）。注意，位英尺，因 EPSG:2249 是州平面英尺

```
SELECT ST_Length(ST_GeomFromText('LINESTRING(743238 2967416,743238 2967450,743265 2967450,743265.625 2967416,743238 2967416)',2249));
```

```
st_length
```

```
-----
122.630744000095
```

```
--Transforming WGS 84 LineString to Massachusetts state plane meters
```

```
SELECT ST_Length(
  ST_Transform(
    ST_GeomFromEWKT('SRID=4326;LINESTRING(-72.1260 42.45, -72.1240 42.45666, ←
      -72.123 42.1546)'),
    26986
  )
);
```

```
st_length
```

```
-----
34309.4563576191
```

几何示例

WGS 84 地理返回度

```
-- the default calculation uses a spheroid
SELECT ST_Length(the_geog) As length_spheroid, ST_Length(the_geog,false) As length_sphere
FROM (SELECT ST_GeographyFromText(
'SRID=4326;LINESTRING(-72.1260 42.45, -72.1240 42.45666, -72.123 42.1546)') As the_geog)
As foo;
```

length_spheroid	length_sphere
34310.5703627288	34346.2060960742

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度。ST\_Length 的别名

#### Synopsis

```
float ST_Length2D(geometry a_2dlinestring);
```

描述

如果几何形是线串或多线串，返回几何形的 2D 度。是 ST\_Length 的别名

相关信息

[ST\\_Length](#), [ST\\_3DLength](#)

### 7.12.14 ST\_3DLength

ST\_3DLength — 返回线性几何体的 3D 度。

#### Synopsis

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

描述

如果几何体是 LineString 或 MultiLineString，返回几何体的 3 或 2 度。于二维，它将返回二维度（与 ST\_Length 和 ST\_Length2D 相同）



函数支持 3d 并且不会失 z-index。



方法符合了 SQL/MM 范。SQL-MM IEC 13249-3: 7.1, 10.3

更改：2.0.0 在之前的版本中，曾被称 ST\_Length3D

## 示例

返回 3D 长度的度（以英尺为单位）。注意，英尺，因为 EPSG:2249 是塞州平面英尺

```
SELECT ST_3DLength(ST_GeomFromText('LINESTRING(743238 2967416 1,743238 2967450 1,743265 2967450 3,743265.625 2967416 3,743238 2967416 3)',2249));
ST_3DLength
-----
122.704716741457
```

## 相关信息

[ST\\_Length](#), [ST\\_Length2D](#)

### 7.12.15 ST\_LengthSpheroid

ST\_LengthSpheroid — 返回球体上长度/度几何体的 2D 或 3D 长度/周。

## Synopsis

```
float ST_LengthSpheroid(geometry a_geometry, spheroid a_spheroid);
```

## 描述

返回球体面上几何形的周。当几何形的坐度/度并且您希望找没有投影的度，使用此函数。球体由文本指定，如下所示：

```
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

```
SELECT ST_LengthSpheroid( geometry_column,
                          'SPHEROID["GRS_1980",6378137,298.257222101]' )
FROM geometry_table;

SELECT ST_LengthSpheroid( geom, sph_m ) As tot_len,
ST_LengthSpheroid(ST_GeometryN(geom,1), sph_m) As len_line1,
ST_LengthSpheroid(ST_GeometryN(geom,2), sph_m) As len_line2
```

```

FROM (SELECT ST_GeomFromText('MULTILINESTRING((-118.584 38.374, -118.583 38.5),
(-71.05957 42.3589 , -71.061 43))') As geom,
CAST('SPHEROID["GRS_1980",6378137,298.257222101]' As spheroid) As sph_m) as foo;
  tot_len | len_line1 | len_line2
-----+-----+-----
85204.5207562955 | 13986.8725229309 | 71217.6482333646

--3D
SELECT ST_LengthSpheroid( geom, sph_m ) As tot_len,
ST_LengthSpheroid(ST_GeometryN(geom,1), sph_m) As len_line1,
ST_LengthSpheroid(ST_GeometryN(geom,2), sph_m) As len_line2
FROM (SELECT ST_GeomFromEWKT('MULTILINESTRING((-118.584 38.374 30,
20, -118.583 38.5 30),
(-71.05957 42.3589 75, -71.061 43 90))') As geom,
CAST('SPHEROID["GRS_1980",6378137,298.257222101]' As spheroid) As sph_m) as foo;
  tot_len | len_line1 | len_line2
-----+-----+-----
85204.5259107402 | 13986.876097711 | 71217.6498130292

```

相关信息

[ST\\_GeometryN](#), [ST\\_Length](#)

### 7.12.16 ST\_LongestLine

ST\_LongestLine — 返回两个几何图形之间的二条最短线。

#### Synopsis

```
geometry ST_LongestLine(geometry g1, geometry g2);
```

#### 描述

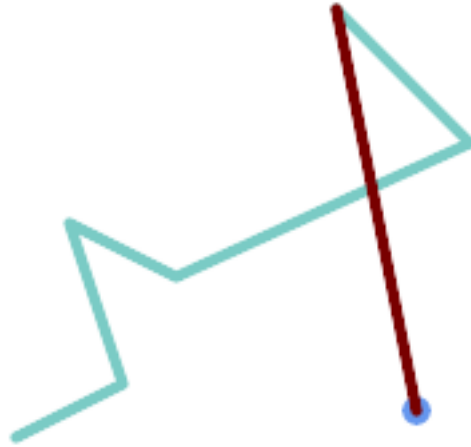
返回两个几何点之间的二条最短线。返回的行从 **g1** 开始，到 **g2** 结束。

最短的线是出在两个点之间。如果找到多个，函数将返回第一个最短的线。直线的长度等于 [ST\\_MaxDistance](#) 返回的距离。

如果 **g1** 和 **g2** 是相同的几何体，返回几何体中相距最近的两个点之间的线。直线的端点位于 [ST\\_MinimumBoundingCircle](#) 算的线上。

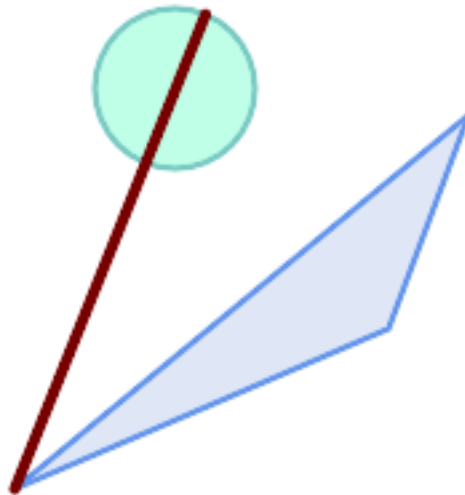
可用性：1.1.0

示例



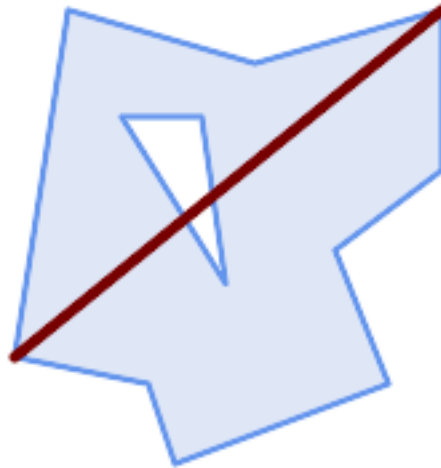
点和线之最长的线

```
SELECT ST_AsText( ST_LongestLine(
    'POINT (160 40)',
    'LINESTRING (10 30, 50 50, 30 110, 70 90, 180 140, 130 190)' )
    ) AS lline;
-----
LINESTRING(160 40,130 190)
```



一个多边形之最长的线

```
SELECT ST_AsText( ST_LongestLine(
    'POLYGON ((190 150, 20 10, 160 70, 190 150))',
    ST_Buffer('POINT(80 160)', 30)
    ) ) AS llinewkt;
-----
LINESTRING(20 10,105.3073372946034 186.95518130045156)
```



跨多个几何体的最长线。线的长度等于最大距离。线的端点位于最小边界上。

```
SELECT ST_AsText( ST_LongestLine( geom, geom)) AS llinekkt,
         ST_MaxDistance( geom, geom) AS max_dist,
         ST_Length( ST_LongestLine(geom, geom)) AS lenll
FROM (SELECT 'POLYGON ((40 180, 110 160, 180 180, 180 120, 140 90, 160 40, 80 10, 70 40, 20 50, 40 180),
         (60 140, 99 77.5, 90 140, 60 140))'::geometry AS geom) AS t;
```

llinekkt	max_dist	lenll
LINESTRING(20 50,180 180)	206.15528128088303	206.15528128088303

相关信息

[ST\\_MaxDistance](#), [ST\\_ShortestLine](#), [ST\\_3DLongestLine](#), [ST\\_MinimumBoundingCircle](#)

## 7.12.17 ST\_3DLongestLine

ST\_3DLongestLine — 返回多个几何体之线的 3D 最长直线

### Synopsis

geometry **ST\_3DLongestLine**(geometry g1, geometry g2);

### 描述

返回多个几何图形之线的 3 最长直线。如果有多个线，函数将返回第一个最长的线。返回的行以 g1 开始，以 g2 结束。线的 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 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

linestring 和 point——3d 和 2d 最远的

```
SELECT ST_AsEWKT(ST_3DLongestLine(line,pt)) AS lol3d_line_pt,
       ST_AsEWKT(ST_LongestLine(line,pt)) As lol2d_line_pt
FROM (SELECT 'POINT(100 100 30)::geometry As pt,
            'LINESTRING (20 80 20, 98 190 1, 110 180 3, 50 75 1000)::' ←
       geometry As line
       ) As foo;
```

lol3d_line_pt		lol2d_line_pt
-----+		
LINESTRING(50 75 1000,100 100 30)		LINESTRING(98 190,100 100)

linestring 和 multipoint——3d 和 2d 最远的

```
SELECT ST_AsEWKT(ST_3DLongestLine(line,pt)) AS lol3d_line_pt,
       ST_AsEWKT(ST_LongestLine(line,pt)) As lol2d_line_pt
FROM (SELECT 'MULTIPOINT(100 100 30, 50 74 1000)::geometry As pt,
            'LINESTRING (20 80 20, 98 190 1, 110 180 3, 50 75 900)::' ←
       geometry As line
       ) As foo;
```

lol3d_line_pt		lol2d_line_pt
-----+		
LINESTRING(98 190 1,50 74 1000)		LINESTRING(98 190,50 74)

MultiLineString 和 Polygon - 3d 和 2d 最远的

```
SELECT ST_AsEWKT(ST_3DLongestLine(poly, mline)) As lol3d,
       ST_AsEWKT(ST_LongestLine(poly, mline)) As lol2d
FROM (SELECT ST_GeomFromEWKT('POLYGON((175 150 5, 20 40 5, 35 45 5, 50 60 5, ←
100 100 5, 175 150 5))') As poly,
       ST_GeomFromEWKT('MULTILINESTRING((175 155 2, 20 40 20, 50 60 -2, 125 ←
100 1, 175 155 1),
       (1 10 2, 5 20 1))') As mline ) As foo;
```

lol3d		lol2d
-----+		
LINESTRING(175 150 5,1 10 2)		LINESTRING(175 150,1 10)

相关信息

[ST\\_3DClosestPoint](#), [ST\\_3DDistance](#), [ST\\_LongestLine](#), [ST\\_3DShortestLine](#), [ST\\_3DMaxDistance](#)

## 7.12.18 ST\_MaxDistance

ST\_MaxDistance — 返回两个几何形之的最大距离（以投影位表示）。

### Synopsis

```
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)。最大距离是出在两个点之间。是 [ST\\_LongestLine](#) 返回的度。

如果 g1 和 g2 是相同的几何体，返回几何体中相距最远的两个点之间的距离。

可用性：1.1.0

## 示例

点与线之间的最大距离。

```
SELECT ST_MaxDistance('POINT(0 0)::geometry, 'LINESTRING ( 2 0, 0 2 ) '::geometry);
-----
2

SELECT ST_MaxDistance('POINT(0 0)::geometry, 'LINESTRING ( 2 2, 2 2 ) '::geometry);
-----
2.82842712474619
```

两个几何体点之间的最大距离。

```
SELECT ST_MaxDistance('POLYGON ((10 10, 10 0, 0 0, 10 10)) '::geometry,
                      'POLYGON ((10 10, 10 0, 0 0, 10 10)) '::geometry);
-----
14.142135623730951
```

## 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，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: currently no vertical datum support so Z is not transformed and assumed to be same ↔
  units as final.
SELECT ST_3DMaxDistance(
    ST_Transform(ST_GeomFromEWKT('SRID=4326;POINT(-72.1235 42.3521 ↔
      10000)'),2163),
    ST_Transform(ST_GeomFromEWKT('SRID=4326;LINESTRING(-72.1260 42.45 ↔
      15, -72.123 42.1546 20)'),2163)
  ) As dist_3d,
  ST_MaxDistance(
    ST_Transform(ST_GeomFromEWKT('SRID=4326;POINT(-72.1235 42.3521 ↔
      10000)'),2163),
    ST_Transform(ST_GeomFromEWKT('SRID=4326;LINESTRING(-72.1260 42.45 ↔
      15, -72.123 42.1546 20)'),2163)
  ) As dist_2d;

  dist_3d      |      dist_2d
-----+-----
24383.7467488441 | 22247.8472107251
```

## 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，间隙是几何体健壮性的度量。

#### Synopsis

```
float ST_MinimumClearance(geometry g);
```

#### 描述

几何形状有可能不是 [ST\\_IsValid](#) (多边形) 或 [ST\\_IsSimple](#) (线) 的有效性验证，但如果其点之一是无效的，该几何形状将变得无效。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) 期间的精度丢失。

最小间隙是几何形状坐标精度量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是  $e$ ，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`。

为了避免精度丢失导致的有效性检查，[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

```
SELECT ST_MinimumClearance('POLYGON ((0 0, 1 0, 1 1, 0.5 3.2e-4, 0 0))');
st_minimumclearance
-----
0.00032
```

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线串。

#### Synopsis

Geometry **ST\_MinimumClearanceLine**(geometry g);

描述

返回跨越几何体最小间隙的线串。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

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

相关信息

[ST\\_MinimumClearance](#)

### 7.12.22 ST\_Perimeter

`ST_Perimeter` — 返回多边形几何或地理的边界度。

#### 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`)，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，使用 `ST_Length`。对于几何类型，周长量的单位由几何的空参考系指定。对于地理类型，使用逆地理行算，其中周长单位是米。如果 PostGIS 是使用 PROJ 版本 4.8.0 或更高版本构建的，球体由 SRID 指定，否则它是 WGS84 独有的。如果 `use_spheroid = false`，算将近似球体而不是球体。

目前，它是 `ST_Perimeter2D` 的别名，但可能会更改以支持更高的精度。

 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: 8.1.3, 9.5.4

可用性 2.0.0 : 引入了地理的支持

## 示例：几何

返回多边形和多重多边形的周长（以英尺为单位）。注意，单位是英尺，因为 EPSG:2249 是塞州平面英尺

```
SELECT ST_Perimeter(ST_GeomFromText('POLYGON((743238 2967416,743238 2967450,743265 2967450,743265.625 2967416,743238 2967416))', 2249));
st_perimeter
-----
122.630744000095
(1 row)
```

```
SELECT ST_Perimeter(ST_GeomFromText('MULTIPOLYGON(((763104.471273676 2949418.44119003,763104.477769673 2949418.42538203,763104.189609677 2949418.22343004,763104.471273676 2949418.44119003)),((763104.471273676 2949418.44119003,763095.804579742 2949436.33850239,763086.132105649 2949451.46730207,763078.452329651 2949462.11549407,763075.354136904 2949466.17407812,763064.362142565 2949477.64291974,763059.953961626 2949481.28983009,762994.637609571 2949532.04103014,762990.568508415 2949535.06640477,762986.710889563 2949539.61421415,763117.237897679 2949709.50493431,763235.236617789 2949617.95619822,763287.718121842 2949562.20592617,763111.553321674 2949423.91664605,763104.471273676 2949418.44119003)))', 2249));
st_perimeter
-----
845.227713366825
(1 row)
```

## 示例：地理

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）

```
SELECT ST_Perimeter(geog) As per_meters, ST_Perimeter(geog)/0.3048 As per_ft
FROM ST_GeogFromText('POLYGON((-71.1776848522251 42.3902896512902,-71.1776843766326 ↵
42.3903829478009,
-71.1775844305465 42.3903826677917,-71.1775825927231 42.3902893647987,-71.1776848522251 ↵
42.3902896512902))') As geog;

per_meters | per_ft
-----+-----
37.3790462565251 | 122.634666195949
```

```
-- MultiPolygon example --
SELECT ST_Perimeter(geog) As per_meters, ST_Perimeter(geog,false) As per_sphere_meters, ←
       ST_Perimeter(geog)/0.3048 As per_ft
FROM ST_GeogFromText('MULTIPOLYGON((( -71.1044543107478 42.340674480411, -71.1044542869917 ←
      42.3406744369506,
-71.1044553562977 42.340673886454, -71.1044543107478 42.340674480411)), ←
((-71.1044543107478 42.340674480411, -71.1044860600303 42.3407237015564, -71.1045215770124 ←
      42.3407653385914,
-71.1045498002983 42.3407946553165, -71.1045611902745 42.3408058316308, -71.1046016507427 ←
      42.340837442371,
-71.104617893173 42.3408475056957, -71.1048586153981 42.3409875993595, -71.1048736143677 ←
      42.3409959528211,
-71.1048878050242 42.3410084812078, -71.1044020965803 42.3414730072048, ←
-71.1039672113619 42.3412202916693, -71.1037740497748 42.3410666421308, ←
-71.1044280218456 42.3406894151355, -71.1044543107478 42.340674480411)))') As geog;
```

per_meters	per_sphere_meters	per_ft
257.634283683311	257.412311446337	845.256836231335

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` 的别名。

#### Synopsis

```
float ST_Perimeter2D(geometry geomA);
```

描述

返回多边形几何体的二维周长。



#### Note

目前是 `ST_Perimeter` 的别名。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，曾被称做 ST\_Perimeter3D

### 示例

塞州英尺空中稍高的多边形的周长

```
SELECT ST_3DPerimeter(geom), ST_Perimeter2d(geom), ST_Perimeter(geom) FROM
      (SELECT ST_GeomFromEWKT('SRID=2249;POLYGON((743238 2967416 2,743238 ←
      2967450 1,
      743265.625 2967416 1,743238 2967416 2))') As geom) As foo;
```

ST_3DPerimeter	st_perimeter2d	st_perimeter
105.465793597674	105.432997272188	105.432997272188

### 相关信息

[ST\\_GeomFromEWKT](#), [ST\\_Perimeter](#), [ST\\_Perimeter2D](#)

## 7.12.25 ST\_ShortestLine

ST\_ShortestLine — 返回两个几何体之间的 2D 最短距离

### Synopsis

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

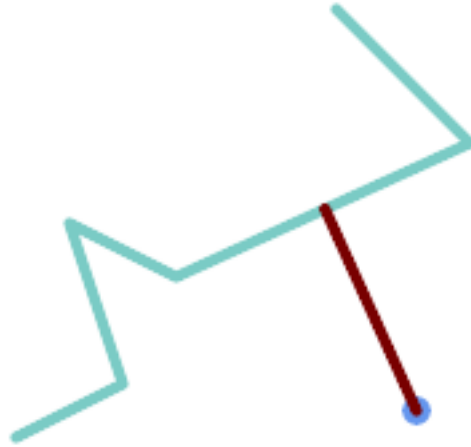
### 描述

返回两个几何体之间的二维最短距离。返回的线从 geom1 开始，到 geom2 结束。如果 geom1 和 geom2 相交，结果是一条起点和终点位于交点的线。线的长度与 g1 和 g2 的 ST\_Distance 返回的长度相同。

新增：3.4.0 - 支持地理。

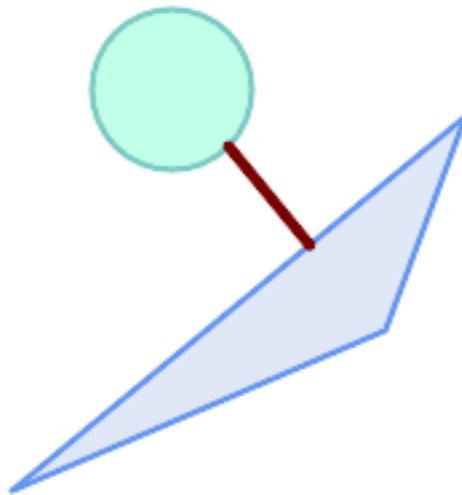
可用性：1.1.0

示例



*Point* 和 *LineString* 之最短

```
SELECT ST_AsText( ST_ShortestLine(
  'POINT (160 40)',
  'LINESTRING (10 30, 50 50, 30 110, 70 90, 180 140, 130 190)')
) As sline;
-----
LINESTRING(160 40,125.75342465753425 115.34246575342466)
```



*Polygons* 之最短

```
SELECT ST_AsText( ST_ShortestLine(
  'POLYGON ((190 150, 20 10, 160 70, 190 150))',
  ST_Buffer('POINT(80 160)', 30)
) ) AS llinevkt;
-----
LINESTRING(131.59149149528952 101.89887534906197,101.21320343559644 138.78679656440357)
```



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该交点的线。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 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

串和点——3d 和 2d 最短线

```
SELECT ST_AsEWKT(ST_3DShortestLine(line,pt)) AS shl3d_line_pt,
       ST_AsEWKT(ST_ShortestLine(line,pt)) As shl2d_line_pt
FROM (SELECT 'POINT(100 100 30)::geometry As pt,
            'LINESTRING (20 80 20, 98 190 1, 110 180 3, 50 75 1000)::' AS
       geometry As line
       ) As foo;
```

shl3d_line_pt	shl2d_line_pt
LINESTRING(54.6993798867619 128.935022917228 11.5475869506606,100 100 30)	LINESTRING(73.0769230769231 115.384615384615,100 100)

☒串和多点—3d 和 2d 最短☒

```
SELECT ST_AsEWKT(ST_3DShortestLine(line,pt)) AS shl3d_line_pt,
       ST_AsEWKT(ST_ShortestLine(line,pt)) As shl2d_line_pt
FROM (SELECT 'MULTIPOINT(100 100 30, 50 74 1000)>:::geometry As pt,
            'LINESTRING (20 80 20, 98 190 1, 110 180 3, 50 75 900)>::: ←
       geometry As line
       ) As foo;

           shl3d_line_pt           | ←
shl2d_line_pt                      +-----+
-----+-----+-----+-----+
LINESTRING(54.6993798867619 128.935022917228 11.5475869506606,100 100 30) | LINESTRING ←
(50 75,50 74)
```

多☒串和多☒形 -3d 和 2d 最短☒

```
SELECT ST_AsEWKT(ST_3DShortestLine(poly, mline)) As shl3d,
       ST_AsEWKT(ST_ShortestLine(poly, mline)) As shl2d
FROM (SELECT ST_GeomFromEWKT('POLYGON((175 150 5, 20 40 5, 35 45 5, 50 60 5, ←
100 100 5, 175 150 5))') As poly,
       ST_GeomFromEWKT('MULTILINESTRING((175 155 2, 20 40 20, 50 60 -2, 125 ←
100 1, 175 155 1),
       (1 10 2, 5 20 1))') As mline ) As foo;
           shl3d ←
                                           |      shl2d
-----+-----+-----+-----+
LINESTRING(39.993580415989 54.1889925532825 5,40.4078575708294 53.6052383805529 ←
5.03423778139177) | LINESTRING(20 40,20 40)
```

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 — ☒算几何☒形落在矩形内的部分。

#### Synopsis

geometry **ST\_ClipByBox2D**(geometry geom, box2d box);

#### 描述

以快速且☒松但可能无效的方式通☒ 2D 框剪切几何体。拓扑上无效的☒入几何☒形不会☒致抛出异常。不保☒☒出几何☒形有效（特☒是，可能会引入多☒形的自相交）。

☒个函数是由 GEOS 模☒☒行的。

可用性：2.2.0

## 示例

```
-- Rely on implicit cast from geometry to box2d for the second parameter
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  $A - \text{ST\_Intersection}(A,B)$ 。如果 A 完全包含在 B 中，返回适当类型的空原子几何图形。



#### Note

它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。

如果提供了可选的 gridSize 参数，输入将捕捉到指定大小的网格，并在同一网格上计算结果点。（需要 GEOS-3.9.0 或更高版本）

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

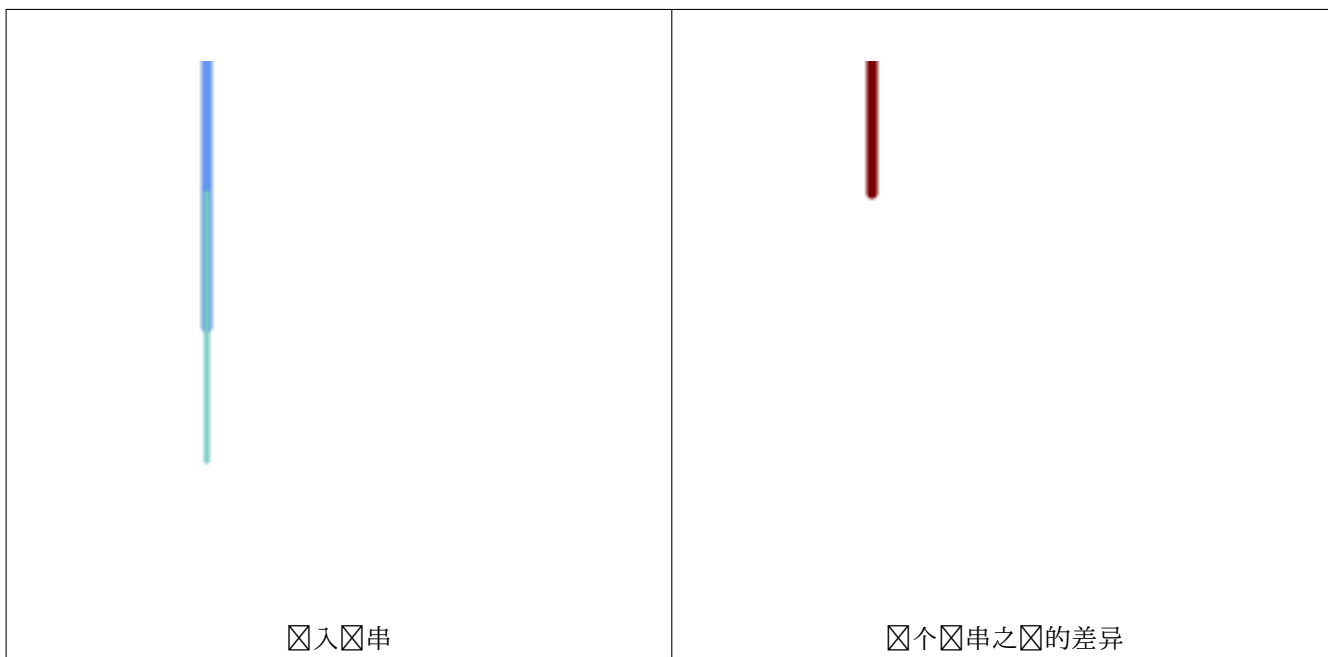


该方法符合了 SQL/MM 规范。SQL-MM 3: 5.1.20



该函数支持 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、平均或插入。

### 示例



2D 串的差异。

```
SELECT ST_AsText(
  ST_Difference(
    'LINESTRING(50 100, 50 200)::geometry',
    'LINESTRING(50 50, 50 150)::geometry'
  )
);

st_astext
-----
LINESTRING(50 150,50 200)
```

3D 点的差异。

```
SELECT ST_AsEWKT( ST_Difference(
  'MULTIPOINT(-118.58 38.38 5,-118.60 38.329 6,-118.614 38.281 7)' :: geometry,
  'POINT(-118.614 38.281 5)' :: geometry
) );

st_asewkt
-----
MULTIPOINT(-118.6 38.329 6,-118.58 38.38 5)
```

相关信息

[ST\\_SymDifference](#), [ST\\_Intersection](#), [ST\\_Union](#)

### 7.13.3 ST\_Intersection

ST\_Intersection — 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。

如果提供了可选的 `gridSize` 参数，则输入将捕捉到固定大小的网格，并在同一网格上计算结果点。（需要 GEOS-3.9.0 或更高版本）

`ST_Intersection` 与 `ST_Intersects` 组合使用于剪切几何图形非常有用，例如在边界框、缓冲区或区域图中，您只需要感兴趣的地区或地区内的几何图形部分。

### Note



用于地理数据，它是几何数据的一个量封装。它首先确定适合 2 个地理对象边界框的最佳 SRID（如果地理对象位于 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到 WGS84 地理。



### Warning

该函数将删除 M 坐标（如果存在）。



### Warning

如果使用 3D 几何图形，您可能需要使用基于 SFGCAL 的 `ST_3DIntersection`，它可以对 3D 几何图形进行正确的 3D 交集。尽管此函数适用于 Z 坐标，但它对 Z 坐标进行平均。

它是通过 GEOS 模型实现的

增加：3.1.0 接受 `gridSize` 参数

需要 GEOS >= 3.9.0 才能使用 `gridSize` 参数

更改：3.0.0 不依赖于 SFCGAL。

可用性：1.5 引入了地理数据类型的支持。



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 3: 5.1.18



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、平均或插入。

示例

```
SELECT ST_AsText(ST_Intersection('POINT(0 0)::geometry, 'LINESTRING ( 2 0, 0 2 )':: ←
  geometry));
  st_astext
-----
GEOMETRYCOLLECTION EMPTY

SELECT ST_AsText(ST_Intersection('POINT(0 0)::geometry, 'LINESTRING ( 0 0, 0 2 )':: ←
  geometry));
  st_astext
-----
POINT(0 0)
```

按国家/地区剪裁所有路径（路径）。这里我假设国家几何是多边形或多多边形。注意：我只保留导致串或多串的交叉点部分，因此我不关心共享一个点的路径。需要我来将几何集合展开独立的 MULT\* 部分。下面的内容相当通用，只需更改 `where` 子句即可适用于多多边形等。

```
select clipped.gid, clipped.f_name, clipped_geom
from (
  select trails.gid, trails.f_name,
         (ST_Dump(ST_Intersection(country.geom, trails.geom))).geom clipped_geom
  from country
        inner join trails on ST_Intersects(country.geom, trails.geom)
) as clipped
where ST_Dimension(clipped.clipped_geom) = 1;
```

对于聚合物，例如多边形地，您可以使用加速技巧，在 0.0 中不包括多边形的几何，以得到空的几何集合（因此，如果中包含多边形、串和点的几何集合 0.0，则保留多边形，它将不再是几何集合。）

```
select poly.gid,
       ST_Multi(
         ST_Buffer(
           ST_Intersection(country.geom, poly.geom),
           0.0
         )
       ) clipped_geom
from country
        inner join poly on ST_Intersects(country.geom, poly.geom)
where not ST_IsEmpty(ST_Buffer(ST_Intersection(country.geom, poly.geom), 0.0));
```

示例：**2.5D**

注意，这不是真正的交集，与使用 `ST_3DIntersection` 的同一示例行比。

```
select ST_AsText(ST_Intersection(linestring, polygon)) As wkt
from ST_GeomFromText('LINESTRING Z (2 2 6,1.5 1.5 7,1 1 8,0.5 0.5 8,0 0 10)') AS ←
  linestring
  CROSS JOIN ST_GeomFromText('POLYGON((0 0 8, 0 1 8, 1 1 8, 1 0 8, 0 0 8))') AS polygon;

      st_astext
-----
LINESTRING Z (1 1 8,0.5 0.5 8,0 0 10)
```

相关信息

[ST\\_3DIntersection](#), [ST\\_Difference](#), [ST\\_Union](#), [ST\\_Dimension](#), [ST\\_Dump](#), [ST\\_Force2D](#), [ST\\_SymDifference](#), [ST\\_Intersects](#), [ST\\_Multi](#)

### 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复的结果几何图形。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](#) 聚合首先累加一个数，然后使用快速算法聚合内容。



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、平均或插入。

#### 示例

```
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 — 点是线的集合。

#### Synopsis


```
geometry ST_Node(geometry geom);
```

#### 描述

返回一个 (Multi)LineString，表示线串集合的完全点版本。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）。

是构建适合作为 [ST\\_Polygonize](#) 输入的全点线条的好方法。

[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。参见 [ST\\_Union](#) 了解聚合版本。

 函数支持 3d 并且不会丢失 z-index。

该函数是由 GEOS 模块实现的。

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, 可能会致生成的串具有不同的序和方向。

示例

点自相交的 3D LineString

```
SELECT ST_AsText(
  ST_Node('LINESTRINGZ(0 0 0, 10 10 10, 0 10 5, 10 0 3)::geometry')
) As output;
output
-----
MULTILINESTRING Z ((0 0 0,5 5 4.5),(5 5 4.5,10 10 10,0 10 5,5 5 4.5),(5 5 4.5,10 0 3))
```

将该点添加到共享一条的串。生成的被溶解。

```
SELECT ST_AsText(
  ST_Node('MULTILINESTRING ((2 5, 2 1, 7 1), (6 1, 4 1, 2 3, 2 5))::geometry')
) As output;
output
-----
MULTILINESTRING((2 5,2 3),(2 3,2 1,4 1),(4 1,2 3),(4 1,6 1),(6 1,7 1))
```

相关信息

[ST\\_UnaryUnion](#), [ST\\_Union](#)

## 7.13.6 ST\_Split

ST\_Split — 返回通将一个几何体分割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) 串或 (多) 多边形界分割串, 或按串分割 (多) 多边形。当 (多) 多边形用作卡, 其性分量 (界) 用于分割入。果几何形始是一个集合。

该函数在某种意上与 [ST\\_Union](#) 相反。将 [ST\\_Union](#) 用于返回的集合理会上会生原始几何形 (尽管由于数字舍入, 情况可能并非完全如此)。



#### Note

如果由于数精度和刀片不相交, 入可能不会按期分割。了避免这种情况, 可能需要首先使用具有小公差的 [ST\\_Snap](#) 将入捕捉到刀片。



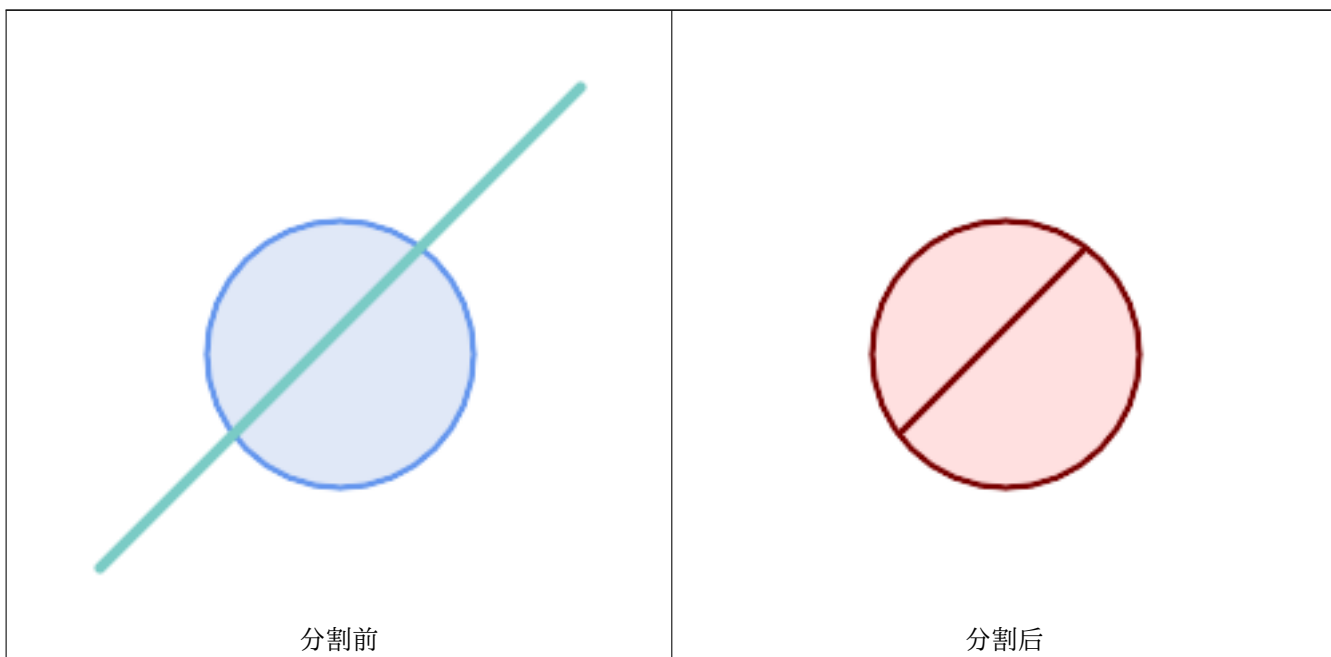
可用性：2.0.0 需要 GEOS

增：2.2.0 引入了通多、多点或（多）多形界分割的支持。

增：2.5.0 引入了通多分割多形的支持。

示例

用一条分割多形。



```
SELECT ST_AsText( ST_Split(
    ST_Buffer(ST_GeomFromText('POINT(100 90)'), 50), -- circle
    ST_MakeLine(ST_Point(10, 10),ST_Point(190, 190)) -- line
));
-- result --
GEOMETRYCOLLECTION(
  POLYGON((150 90,149.039264020162 80.2454838991936,146.193976625564 ←
    70.8658283817455,...),
  POLYGON(...))
)
```

将一个多串（MultiLineString）通一个点分割，其中个点正好位于个串元素上。



## 7.13.7 ST\_Subdivide

ST\_Subdivide — 计算几何体的直分。

### 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分将 geom 分成多个部分的结果，每个部分包含不超过 max\_vertices。max\_vertices 必须为 5 或更多，因为它需要 5 个点来表示一个封闭的盒子。可以指定 gridSize 以在固定精度空间中执行裁剪工作（需要 GEOS-3.9.0+）。

对于索引分数据集，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，因为索引重新执行的空操作处理的点更少。



### Note

它是一个返回集合的 [集合返回函数](#) (SRF) 包含一个几何图形的。它可以在 SELECT 列表或 FROM 子句中使用，以生成一个结果集，其中每个结果几何图形都有一个 ID。

这个函数是由 GEOS 模块执行的。

可用性：2.2.0

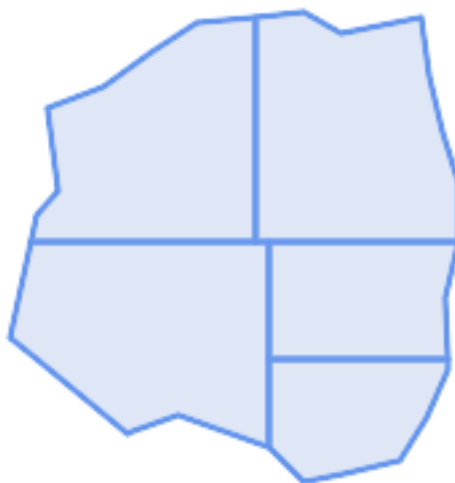
增加：2.5.0 重用多边形分割上的点，点数从 8 减少到 5。

增加：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：将多边形分成不超过 10 个点的部分，并为每个部分分配一个唯一的 id。



分至最多 10 个点

```
SELECT row_number() OVER() As rn, ST_AsText(geom) As wkt
FROM (SELECT ST_SubDivide(
'POLYGON((132 10,119 23,85 35,68 29,66 28,49 42,32 56,22 64,32 110,40 119,36 150,
57 158,75 171,92 182,114 184,132 186,146 178,176 184,179 162,184 141,190 122,
190 100,185 79,186 56,186 52,178 34,168 18,147 13,132 10))'::geometry,10)) AS f(
geom);
```

rn	wkt
1	POLYGON((119 23,85 35,68 29,66 28,32 56,22 64,29.8260869565217 100,119 100,119 23))
2	POLYGON((132 10,119 23,119 56,186 56,186 52,178 34,168 18,147 13,132 10))
3	POLYGON((119 56,119 100,190 100,185 79,186 56,119 56))
4	POLYGON((29.8260869565217 100,32 110,40 119,36 150,57 158,75 171,92 182,114 184,114 100,29.8260869565217 100))
5	POLYGON((114 184,132 186,146 178,176 184,179 162,184 141,190 122,190 100,114 100,114 184))

示例：使用 ST\_Segmentize(geography, distance) 密集化一条地理线，并使用 ST\_Subdivide 将生成的分割线分割成 8 个点的子线。



致密化和分割。

```
SELECT ST_AsText( ST_Subdivide(
ST_Segmentize('LINESTRING(0 0, 85 85)'::geography,
1200000)::geometry, 8));
```

```
LINESTRING(0 0,0.487578359029357 5.57659056746196,0.984542144675897
11.1527721155093,1.50101059639722 16.7281035483571,1.94532113630331 21.25)
LINESTRING(1.94532113630331 21.25,2.04869538062779 22.3020741387339,2.64204641967673
27.8740533545155,3.29994062412787 33.443216802941,4.04836719489742
39.0084282520239,4.59890468420694 42.5)
```

```

LINESTRING(4.59890468420694 42.5,4.92498503922732 44.5680389206321,5.98737409390639 ↔
50.1195229244701,7.3290919767674 55.6587646879025,8.79638749938413 60.1969505994924)
LINESTRING(8.79638749938413 60.1969505994924,9.11375579533779 ↔
61.1785363177625,11.6558166691368 66.6648504160202,15.642041247655 ↔
72.0867690601745,22.8716627200212 77.3609628116894,24.6991785131552 77.8939011989848)
LINESTRING(24.6991785131552 77.8939011989848,39.4046096622744 ↔
82.1822848017636,44.7994523421035 82.5156766227011)
LINESTRING(44.7994523421035 82.5156766227011,85 85)

```

示例：在表中的适当位置分割复几何图形。具有原始几何的图形将从源表中删除，拆分后生成的几何将成其位置的新图形。

```

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：创建具有分区几何图形的新表。保留原始几何图形的图形，新表可以与原始表连接。由于 `ST_Subdivide` 是一个返回集合的函数，并且行具有多个图形，因此此语法会自生成一个表，其中每个部分的图形都有一行。

```

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](#)

## 7.13.8 ST\_SymDifference

`ST_SymDifference` — 计算表示几何图形 A 和 B 不相交部分的几何图形。

### Synopsis

geometry **ST\_SymDifference**(geometry geomA, geometry geomB, float8 gridSize = -1);

### 描述

返回表示几何图形 A 和 B 不相交部分的几何图形。相当于 `ST_Union(A,B) - ST_Intersection(A,B)`。之所以称差，是因为 `ST_SymDifference(A,B) = ST_SymDifference(B,A)`。

如果提供了可选的 `gridSize` 参数，输入将捕捉到固定大小的网格，并在同一网格上计算结果点。（需要 GEOS-3.9.0 或更高版本）

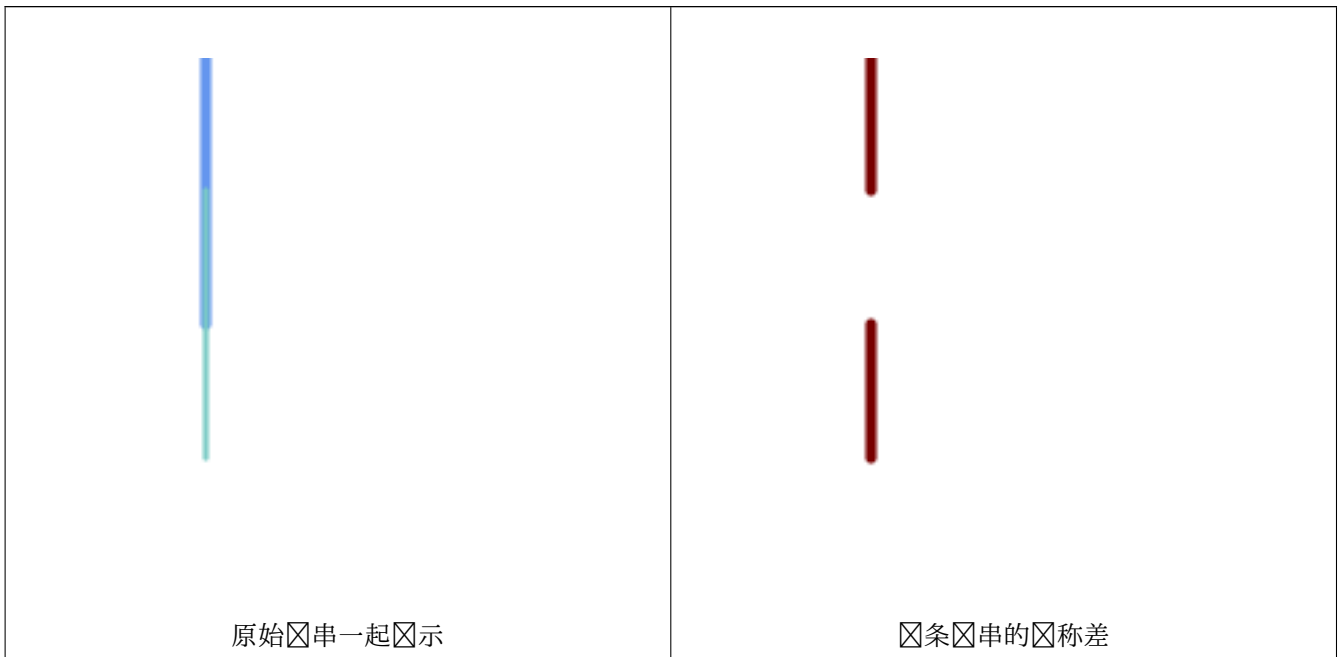
它是通过 GEOS 模块实现的

增量：3.1.0 接受 `gridSize` 参数。

需要 GEOS  $\geq 3.9.0$  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
- ✔ 此方法符合了 SQL/MM 规范。 SQL-MM 3: 5.1.21
- ✔ 此函数支持 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、平均或插入。

示例



```
--Safe for 2d - symmetric difference of 2 linestrings
SELECT ST_AsText(
  ST_SymDifference(
    ST_GeomFromText('LINESTRING(50 100, 50 200)'),
    ST_GeomFromText('LINESTRING(50 50, 50 150)')
  )
);
```

```
st_astext
-----
MULTILINESTRING((50 150,50 200),(50 50,50 100))
```

```
--When used in 3d doesn't quite do the right thing
SELECT ST_AsEWKT(ST_SymDifference(ST_GeomFromEWKT('LINESTRING(1 2 1, 1 4 2)'),
  ST_GeomFromEWKT('LINESTRING(1 1 3, 1 3 4)')))
```

```
st_astext
-----
MULTILINESTRING((1 3 2.75,1 4 2),(1 1 3,1 2 2.25))
```

相关信息

[ST\\_Difference](#), [ST\\_Intersection](#), [ST\\_Union](#)

### 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而无效的多重多边形。但是，每个输入件都必须有效。无效的输入件（例如重叠多边形）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串集合行点化和分解，以使它。 (**ST\_Node** 也行此操作，但它不提供 **gridSize** 参数。)

可以将 **ST\_UnaryUnion** 与 **ST\_Collect** 合起来，以微一次合的几何图形数量。允许在内存使用和计算之之行，从而在 **ST\_Union** 和 **ST\_MemUnion** 之取得平衡。

如果提供了可的 **gridSize** 参数，输入将捕捉到定大小的网格，并在同一网格上计算果点。（需要 GEOS-3.9.0 或更高版本）



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。

增：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**

### 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叠的结果几何图形。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，该几何图形是多个输入几何图形的并集。如果任一输入为 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 数据库中的“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致，它也会忽略 NULL 几何图形。

请参阅 [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> 中引入的快速算法

可以指定 gridSize 来在固定精度空间中工作。输入将捕捉到固定大小的网格，并在同一网格上计算结果点。（需要 GEOS-3.9.0 或更高版本）



### Note

**ST\_Collect** 如果不要求结果不重叠，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进行任何处理。

该函数是由 GEOS 模块实现的。

ST\_Union 构建 MultiLineString 并且不会将 LineString 合成一个 LineString。使用 [ST\\_LineMerge](#) 组合字符串。

注意：此函数以前称为 GeomUnion()，它是从“Union”重命名的，因为 UNION 是 SQL 保留字。

增强：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：1.4.0 - ST\_Union 得到增强。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。



该方法实现了 SQL/MM 规范。SQL-MM 3 : 5.1.19 涉及多边形形的 z 索引（高程）。



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、平均或插入。

## 示例

### 聚合示例

```
SELECT id,
       ST_Union(geom) as singlegeom
FROM sometable f
GROUP BY id;
```



非聚合示例

```
select ST_AsText(ST_Union('POINT(1 2)' :: geometry, 'POINT(-2 3)' :: geometry))

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

select ST_AsText(ST_Union('POINT(1 2)' :: geometry, 'POINT(1 2)' :: geometry))

st_astext
-----
POINT(1 2)
```

3D 示例 - 3D 的 ☒ 型 (和混合 ☒ 度 !)

```
select ST_AsEWKT(ST_Union(geom))
from (
    select 'POLYGON((-7 4.2,-7.1 4.2,-7.1 4.3, -7 4.2))'::geometry geom
    union all
    select 'POINT(5 5 5)'::geometry geom
    union all
    select 'POINT(-2 3 1)'::geometry geom
    union all
    select 'LINESTRING(5 5 5, 10 10 10)'::geometry geom
) as foo;

st_asewkt
-----
GEOMETRYCOLLECTION(POINT(-2 3 1),LINESTRING(5 5 5,10 10 10),POLYGON((-7 4.2 5,-7.1 4.2 ←
5,-7.1 4.3 5,-7 4.2 5)));
```

不混合 ☒ 度的 3D 示例

```
select ST_AsEWKT(ST_Union(geom))
from (
    select 'POLYGON((-7 4.2 2,-7.1 4.2 3,-7.1 4.3 2, -7 4.2 2))'::geometry geom
    union all
    select 'POINT(5 5 5)'::geometry geom
    union all
    select 'POINT(-2 3 1)'::geometry geom
    union all
    select 'LINESTRING(5 5 5, 10 10 10)'::geometry geom
) as foo;

st_asewkt
-----
GEOMETRYCOLLECTION(POINT(-2 3 1),LINESTRING(5 5 5,10 10 10),POLYGON((-7 4.2 2,-7.1 4.2 ←
3,-7.1 4.3 2,-7 4.2 2)))

--Examples using new Array construct
SELECT ST_Union(ARRAY(SELECT geom FROM sometable));

SELECT ST_AsText(ST_Union(ARRAY[ST_GeomFromText('LINESTRING(1 2, 3 4)'),
    ST_GeomFromText('LINESTRING(3 4, 4 5)']))) As wktunion;

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

相关信息

[ST\\_Collect](#), [ST\\_UnaryUnion](#), [ST\\_MemUnion](#), [ST\\_Intersection](#), [ST\\_Difference](#), [ST\\_SymDifference](#)

## 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。距离会小几何形而不是展它。距离可能会完全小多边形，在这种情况下返回 POLYGON EMPTY。于点和，距离始返回空果。

于几何体，距离以几何体的空参考系的位指定。于地理，距离以米位指定。

可的第三个参数控制冲区的精度和式。冲区中弧的精度指定用于近似四分之一段的段数（默8）。可以通过提供空白分隔的 = 列表来指定冲区式，如下所示：

- 'quad\_segs=#' : 用于近似四分之一段的段数（默8）。
- 'endcap=round|flat|square' : endcap 式（默“round”）。“butt”被是“flat”的同。
- 'join=round|mitre|bevel' : 接式（默“round”）。“mitre”被是“mitre”的同。
- 'mitre\_limit=#.#' : 斜接比率限制（影斜接式）。“miter\_limit”被接受“mitre\_limit”的同。
- 'side=both|left|right' : 'left' 或 'right' 在几何体上行行冲，冲相于的方向。适用于 LINESTRING 几何形，不影 POINT 或 POLYGON 几何形。默情况下，止式是方形的。

#### Note



于地理数据，是几何数据的一个量封装。它确定最适合地理象界框的平面空参考系（UTM、伯特方位角等 (LAEA) 北/南极，最后是墨卡托）。冲区在平面空中算，然后回 WGS84。如果入象大于 UTM 区域或跨越日期更，可能不会生所需的行。



#### Note

冲区始是有效的多形几何体。冲区可以理无效入，因此距离 0 的冲有被用作修复无效多形的一种方法。[ST\\_MakeValid](#)也可用于此目的。

**Note!****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!****Note**

此函数忽略 Z 度。即使在 3D 几何体上使用，它也始出 2D 果。

增：2.5.0 - `ST_Buffer` 的几何感知版本已得到增，允您指定要冲的一。`side=both|left|right`。

可用性：1.5 - `ST_Buffer` 已得到增，以适各种端接和接。例如，您可能希望将道路串街道面，并希望将端点平面或正方形而不是。添加了地理的薄包装器。

个函数是由 GEOS 模行的。

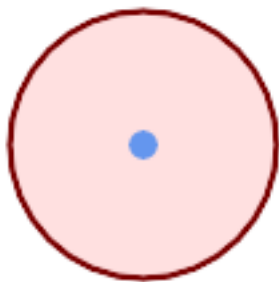


此方法了 SQL 1.1 的 OGC 功能范。 s2.1.1.3



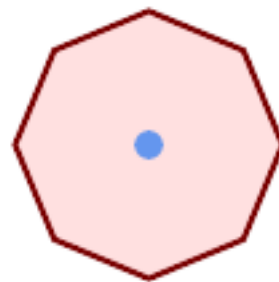
方法了 SQL/MM 范。 SQL-MM IEC 13249-3: 5.1.30

示例



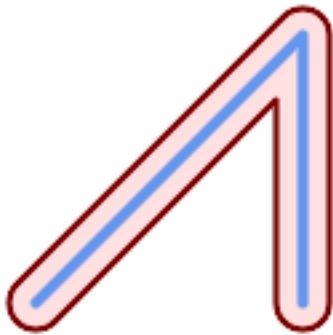
`quad_segs=8` (默认)

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



`quad_segs=2` (不足)

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



*endcap=round join=round* (默认)

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



*endcap=square*

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



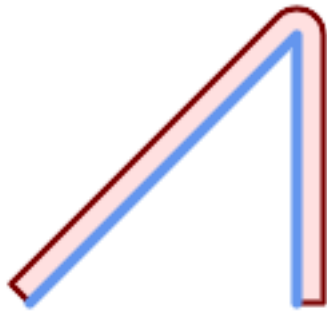
*join=bevel*

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



*join=mitre mitre\_limit=5.0* (默认最大斜接比率)

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



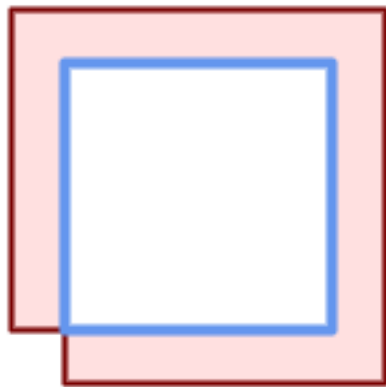
*side=left*

```
SELECT ST_Buffer(
  ST_GeomFromText(
    'LINESTRING(50 50,150 150,150 50)'
  ), 10, 'side=left');
```



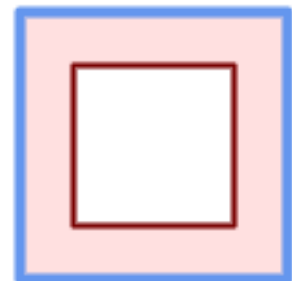
*side=right*

```
SELECT ST_Buffer(
  ST_GeomFromText(
    'LINESTRING(50 50,150 150,150 50)'
  ), 10, 'side=right');
```



逆时针方向, 多边形边界向左

```
SELECT ST_Buffer(
  ST_ForceRHR(
    ST_Boundary(
      ST_GeomFromText(
        'POLYGON ((50 50, 50 150, 150 150, 150 50, 50 50))'
      )
    ), 20, 'side=left');
```



顺时针方向, 多边形边界向右

```
SELECT ST_Buffer(
  ST_ForceRHR(
    ST_Boundary(
      ST_GeomFromText(
        'POLYGON ((50 50, 50 150, 150 150, 150 50, 50 50))'
      )
    ), 20, 'side=right');
```

--A buffered point approximates a circle

```

-- A buffered point forcing approximation of (see diagram)
-- 2 points per quarter circle is poly with 8 sides (see diagram)
SELECT ST_NPoints(ST_Buffer(ST_GeomFromText('POINT(100 90)'), 50)) As ←
    promisingcircle_pcount,
ST_NPoints(ST_Buffer(ST_GeomFromText('POINT(100 90)'), 50, 2)) As lamecircle_pcount;

promisingcircle_pcount | lamecircle_pcount
-----+-----
                33 |                9

--A lighter but lamer circle
-- only 2 points per quarter circle is an octagon
--Below is a 100 meter octagon
-- Note coordinates are in NAD 83 long lat which we transform
to Mass state plane meter and then buffer to get measurements in meters;
SELECT ST_AsText(ST_Buffer(
ST_Transform(
ST_SetSRID(ST_Point(-71.063526, 42.35785),4269), 26986)
,100,2)) As octagon;
-----
POLYGON((236057.59057465 900908.759918696,236028.301252769 900838.049240578,235
957.59057465 900808.759918696,235886.879896532 900838.049240578,235857.59057465
900908.759918696,235886.879896532 900979.470596815,235957.59057465 901008.759918
696,236028.301252769 900979.470596815,236057.59057465 900908.759918696))

```

相关信息

[ST\\_Collect](#), [ST\\_DWithin](#), [ST\\_SetSRID](#), [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。要将内孔也包含多边形，使用 `ST_Polygonize`。

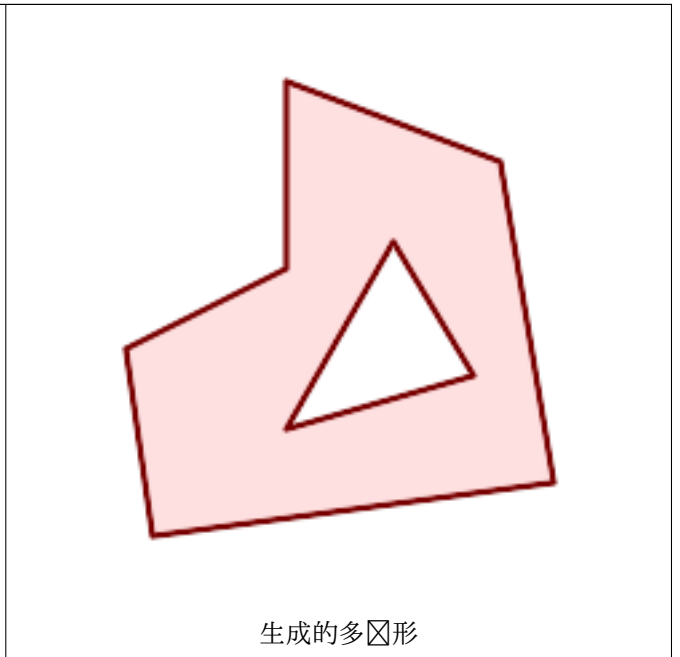
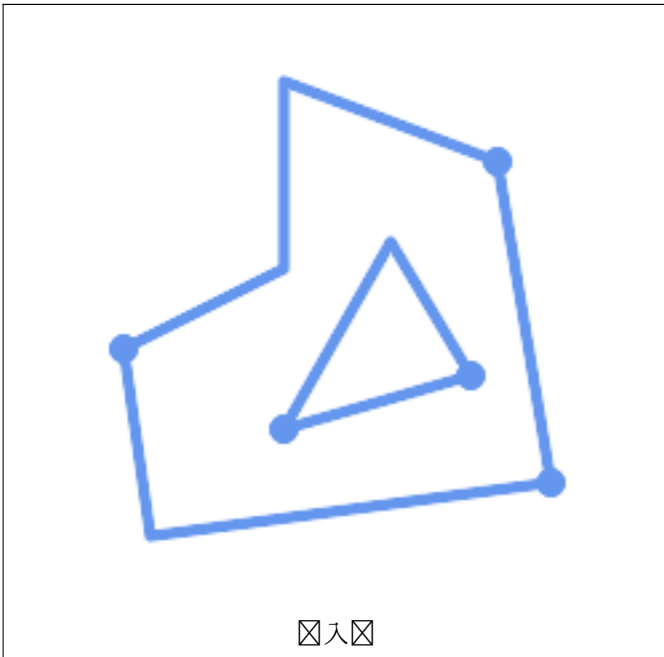


#### Note

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



```
WITH data(geom) AS (VALUES
  ('LINESTRING (180 40, 30 20, 20 90)')::geometry)
,('LINESTRING (180 40, 160 160)')::geometry)
,('LINESTRING (160 160, 80 190, 80 120, 20 90)')::geometry)
,('LINESTRING (80 60, 120 130, 150 80)')::geometry)
,('LINESTRING (80 60, 150 80)')::geometry)
)
SELECT ST_AsText( ST_BuildArea( ST_Collect( geom )))
FROM data;
```

-----

```
POLYGON((180 40,30 20,20 90,80 120,80 190,160 160,180 40),(150 80,120 130,80 60,150 80))
```



用☒个☒形多☒形☒建一个☒形

```
SELECT ST_BuildArea(ST_Collect(inring,outring))
FROM (SELECT
```

```
ST_Buffer('POINT(100 90)', 25) As inring,
ST_Buffer('POINT(100 90)', 50) As outring) As t;
```

## 相关信息

[ST\\_Collect](#), [ST\\_MakePolygon](#), [ST\\_MakeValid](#), [ST\\_Node](#), [ST\\_Polygonize](#), [ST\\_BdPolyFromText](#), [ST\\_BdMPolyFromText](#)  
(使用 [ST\\_GeomFromText](#) 接口 [ST\\_GeomFromText](#) 函数 [ST\\_GeomFromText](#) 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`，中心是输入点的算术平均。对于 `[MULTI]LINESTRING`，中心是使用每条线段的加长度计算的。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` 转换为线串，然后与 `LINESTRING` 相同。

对于混合精度输入，结果等于最高精度的任何几何形状的中心（因为低精度的几何形状中心的“重量”为零）。

注意，对于多边形几何形状，中心不一定位于多边形内部。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。

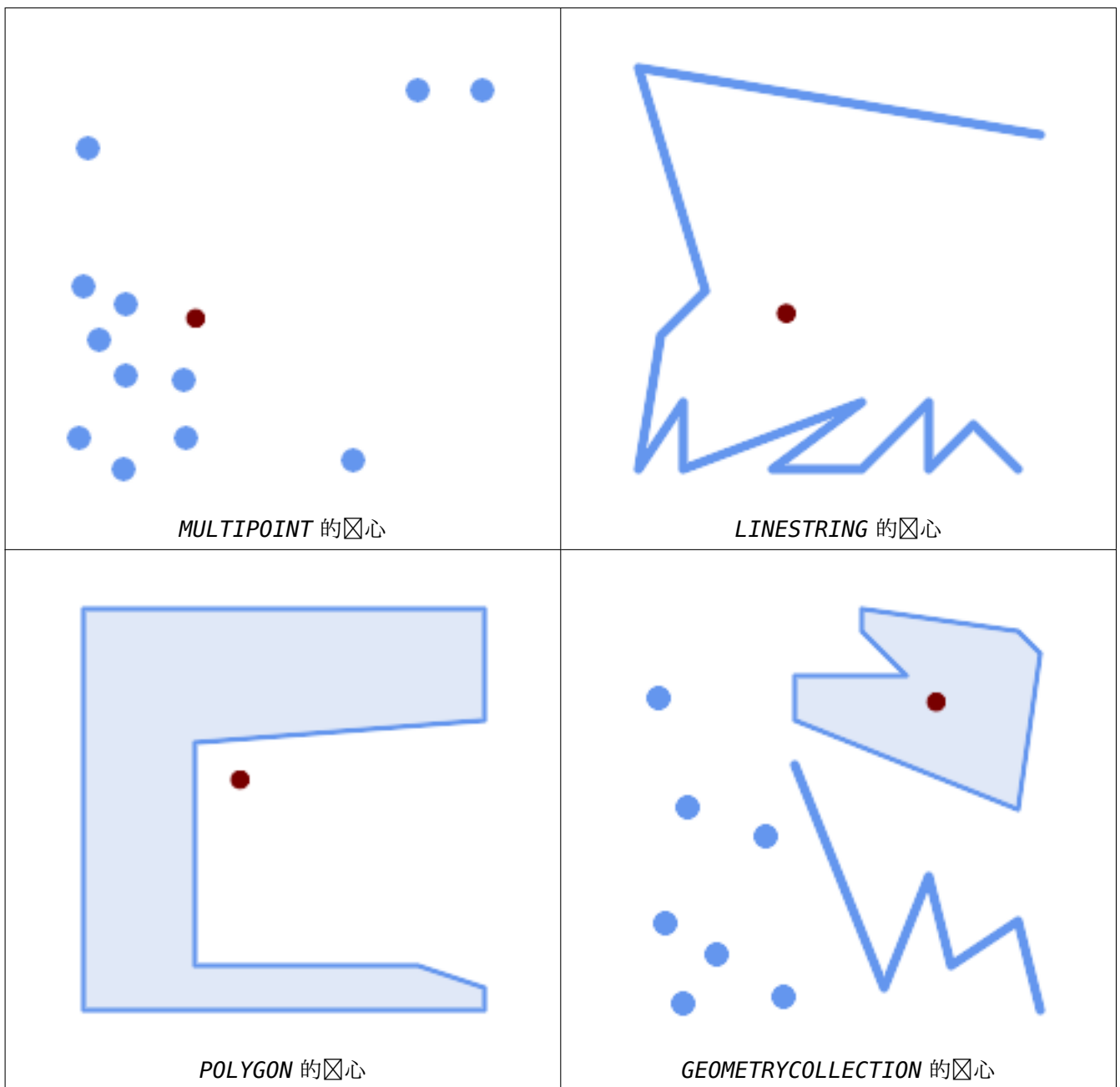
 此方法符合了 SQL/MM 规范。SQL-MM 3: 8.1.4, 9.5.5

## 示例

在下图中，点是源几何体的中心。

---





```
SELECT ST_AsText(ST_Centroid('MULTIPOINT ( -1 0, -1 2, -1 3, -1 4, -1 7, 0 1, 0 3, 1 1, 2 0, 6 0, 7 8, 9 8, 10 6 )'));
          st_astext
```

```
-----
POINT(2.30769230769231 3.30769230769231)
(1 row)
```

```
SELECT ST_AsText(ST_centroid(g))
FROM ST_GeomFromText('CIRCULARSTRING(0 2, -1 1,0 0, 0.5 0, 1 0, 2 1, 1 2, 0.5 2, 0 2)') AS g ;
```

```
-----
POINT(0.5 1)
```

```
SELECT ST_AsText(ST_centroid(g))
```

```
FROM ST_GeomFromText('COMPOUNDCURVE(CIRCULARSTRING(0 2, -1 1,0 0),(0 0, 0.5 0, 1 0), ←
  CIRCULARSTRING( 1 0, 2 1, 1 2),(1 2, 0.5 2, 0 2))') AS g;
-----
POINT(0.5 1)
```

相关信息

[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** 平滑线性或多边形几何体。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长度 1/4 的四个点替换。3 次迭代提供了合理的平滑度；最大限制 5。

如果 `preserveEndpoints` 为 `true`，多边形图的端点不会被平滑。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，如果使用该函数（参见 [ST\\_Simplify](#)、[ST\\_SimplifyPreserveTopology](#) 和 [ST\\_SimplifyVW](#)）。

如果具有 Z 和 M 维度（如果存在）的插图。



该函数支持 3d 并且不会丢失 z-index。

可用性：2.5.0

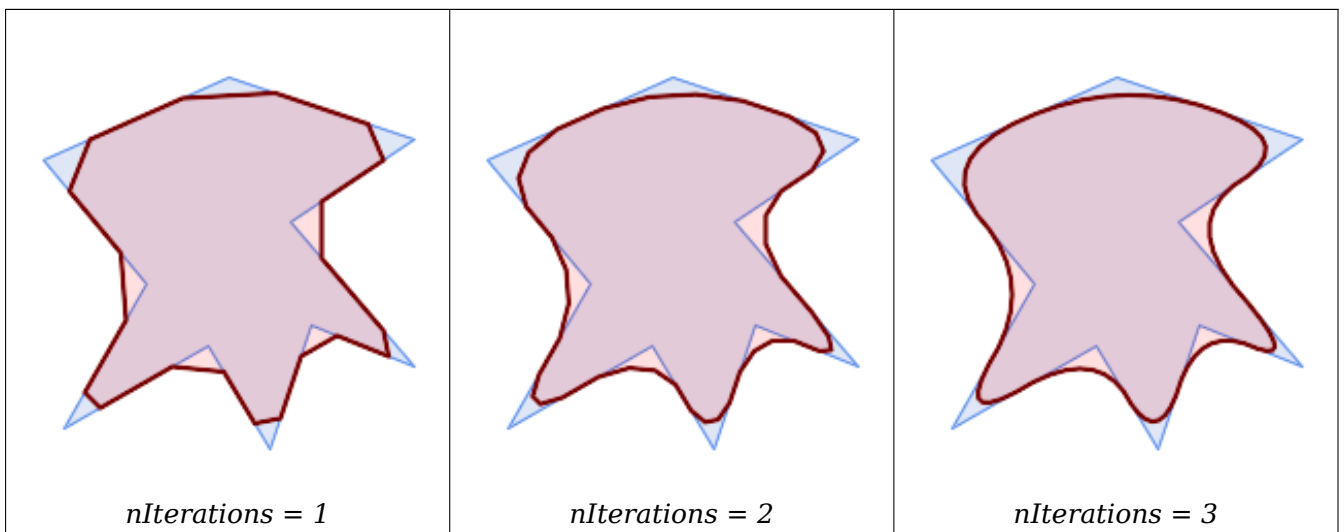
#### 示例

平滑 triangle（三角形）：

```
SELECT ST_AsText(ST_ChaikinSmoothing(geom)) smoothed
FROM (SELECT 'POLYGON((0 0, 8 8, 0 16, 0 0))'::geometry geom) AS foo;
```

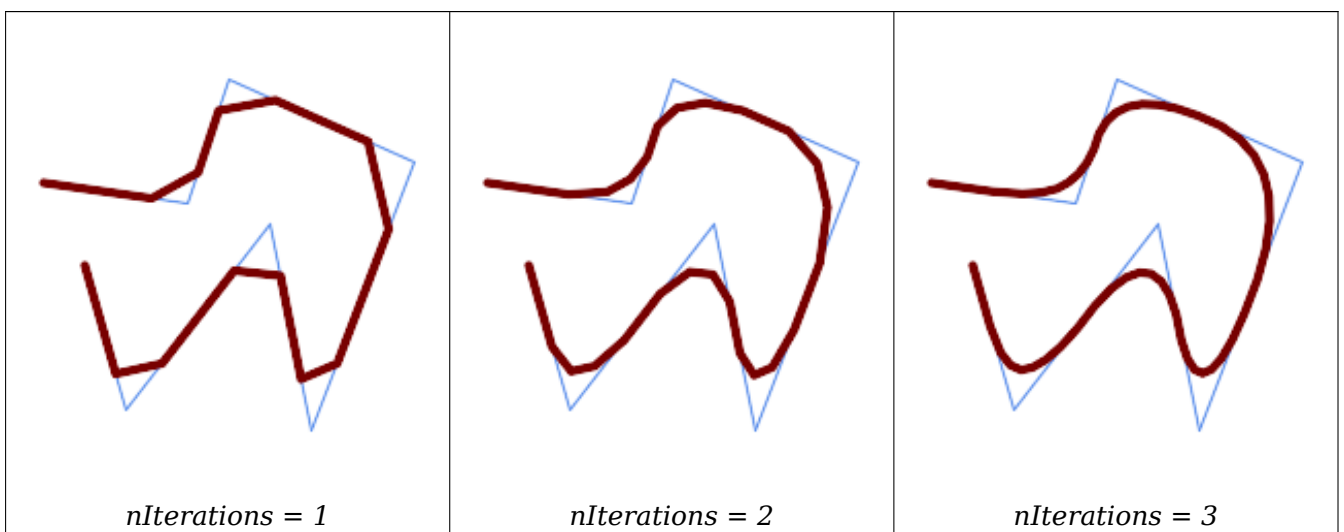
```
          smoothed
b' '-b' 'b' '-b' 'b' '-b' 'b' '-b' 'b' '-b' 'b' '-b' 'b' '-b' 'b' '-b' 'b' '-b' 'b' '-b' 'b' '-b' 'b' '-b' 'b' ←
  b' '-b' 'b' '-b' 'b' '-b' 'b' '-b' 'b' '-b' 'b' '-b' 'b' '-b' 'b' '-b' 'b' '-b' 'b' '-b' 'b' '-b' 'b' ←
    '-b' 'b' '-b' 'b' '-b' 'b' '-b' 'b' '-b' 'b' '-b' 'b' '-b' 'b' '-b' 'b' '-b' 'b' '-b' 'b' '-b' 'b' ←
      '-b' 'b' '-b' 'b' '-b' 'b' '-b' 'b' '-b' 'b' '-b' 'b' '-b' 'b' '-b' 'b' '-b' 'b' '-b' 'b' '-b' 'b' ←
POLYGON((2 2,6 6,6 10,2 14,0 12,0 4,2 2))
```

使用 1、2 和 3 次迭代平滑 Polygon：



```
SELECT ST_ChaikinSmoothing(
  'POLYGON ((20 20, 60 90, 10 150, 100 190, 190 160, 130 120, 190 50, 140 70, 120 ←
    10, 90 60, 20 20))',
  generate_series(1, 3) );
```

使用 1、2 和 3 次迭代平滑 LineString :



```
SELECT ST_ChaikinSmoothing(
  'LINESTRING (10 140, 80 130, 100 190, 190 150, 140 20, 120 120, 50 30, 30 100) ←
  ',
  generate_series(1, 3) );
```

相关信息

[ST\\_Simplify](#), [ST\\_SimplifyPreserveTopology](#), [ST\\_SimplifyVW](#)

## 7.14.5 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。个或多个共点的凹壳是点串。一个或多个相同点的凹壳是一个点。除非可选项的 `param_allow_holes` 参数指定为 `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，凸壳类似于用橡皮筋将其封住。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。值为 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。值 0 会生成具有最大凹度的外壳（但仍然是多边形）。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` 将凹度确定为输入点的 Delaunay 三角剖分中最长边和最短边之差的平方的一部分。短于凹度的边会被三角测量“侵占”。剩下的三角形形成凹形外壳。

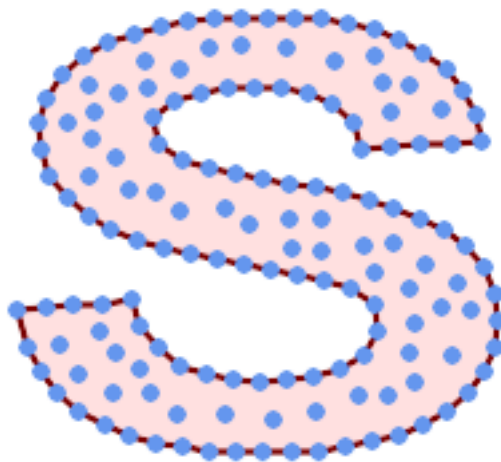
对于点和线性输入，外壳将包含输入的所有点。对于多边形输入，外壳将包含输入的所有点以及输入覆盖的所有区域。如果您想要多边形输入的逐点外壳，请首先使用 `ST_Points` 将其转换为点。

这不是聚合函数。要计算一个几何图形的凹壳，请使用 `ST_Collect`（例如 `ST_ConcaveHull(ST_Collect(geom), 0.80)`）。

可用性: 2.0.0

增强: 3.3.0, 需要 GEOS 3.11+ 使用 GEOS 本机实现

### 示例

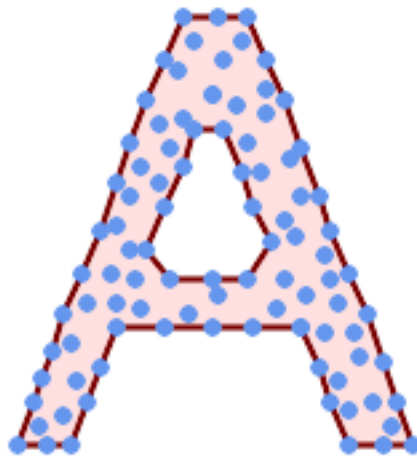


多点的凹壳

```

SELECT ST_AsText( ST_ConcaveHull(
  'MULTIPOINT ((10 72), (53 76), (56 66), (63 58), (71 51), (81 48), (91 46), (101 45), (111 46), (121 47), (131 50), (140 55), (145 64), (144 74), (135 80), (125 83), (115 85), (105 87), (95 89), (85 91), (75 93), (65 95), (55 98), (45 102), (37 107), (29 114), (22 122), (19 132), (18 142), (21 151), (27 160), (35 167), (44 172), (54 175), (64 178), (74 180), (84 181), (94 181), (104 181), (114 181), (124 181), (134 179), (144 177), (153 173), (162 168), (171 162), (177 154), (182 145), (184 135), (139 132), (136 142), (128 149), (119 153), (109 155), (99 155), (89 155), (79 153), (69 150), (61 144), (63 134), (72 128), (82 125), (92 123), (102 121), (112 119), (122 118), (132 116), (142 113), (151 110), (161 106), (170 102), (178 96), (185 88), (189 78), (190 68), (189 58), (185 49), (179 41), (171 34), (162 29), (153 25), (143 23), (133 21), (123 19), (113 19), (102 19), (92 19), (82 19), (72 21), (62 22), (52 25), (43 29), (33 34), (25 41), (19 49), (14 58), (21 73), (31 74), (42 74), (173 134), (161 134), (150 133), (97 104), (52 117), (157 156), (94 171), (112 106), (169 73), (58 165), (149 40), (70 33), (147 157), (48 153), (140 96), (47 129), (173 55), (144 86), (159 67), (150 146), (38 136), (111 170), (124 94), (26 59), (60 41), (71 162), (41 64), (88 110), (122 34), (151 97), (157 56), (39 146), (88 33), (159 45), (47 56), (138 40), (129 165), (33 48), (106 31), (169 147), (37 122), (71 109), (163 89), (37 156), (82 170), (180 72), (29 142), (46 41), (59 155), (124 106), (157 80), (175 82), (56 50), (62 116), (113 95), (144 167))',
  0.1 ) );
---st_astext---
POLYGON ((18 142, 21 151, 27 160, 35 167, 44 172, 54 175, 64 178, 74 180, 84 181, 94 181, 104 181, 114 181, 124 181, 134 179, 144 177, 153 173, 162 168, 171 162, 177 154, 182 145, 184 135, 173 134, 161 134, 150 133, 139 132, 136 142, 128 149, 119 153, 109 155, 99 155, 89 155, 79 153, 69 150, 61 144, 63 134, 72 128, 82 125, 92 123, 102 121, 112 119, 122 118, 132 116, 142 113, 151 110, 161 106, 170 102, 178 96, 185 88, 189 78, 190 68, 189 58, 185 49, 179 41, 171 34, 162 29, 153 25, 143 23, 133 21, 123 19, 113 19, 102 19, 92 19, 82 19, 72 21, 62 22, 52 25, 43 29, 33 34, 25 41, 19 49, 14 58, 10 72, 21 73, 31 74, 42 74, 53 76, 56 66, 63 58, 71 51, 81 48, 91 46, 101 45, 111 46, 121 47, 131 50, 140 55, 145 64, 144 74, 135 80, 125 83, 115 85, 105 87, 95 89, 85 91, 75 93, 65 95, 55 98, 45 102, 37 107, 29 114, 22 122, 19 132, 18 142))

```



多点的凹面外壳，允许有孔

```

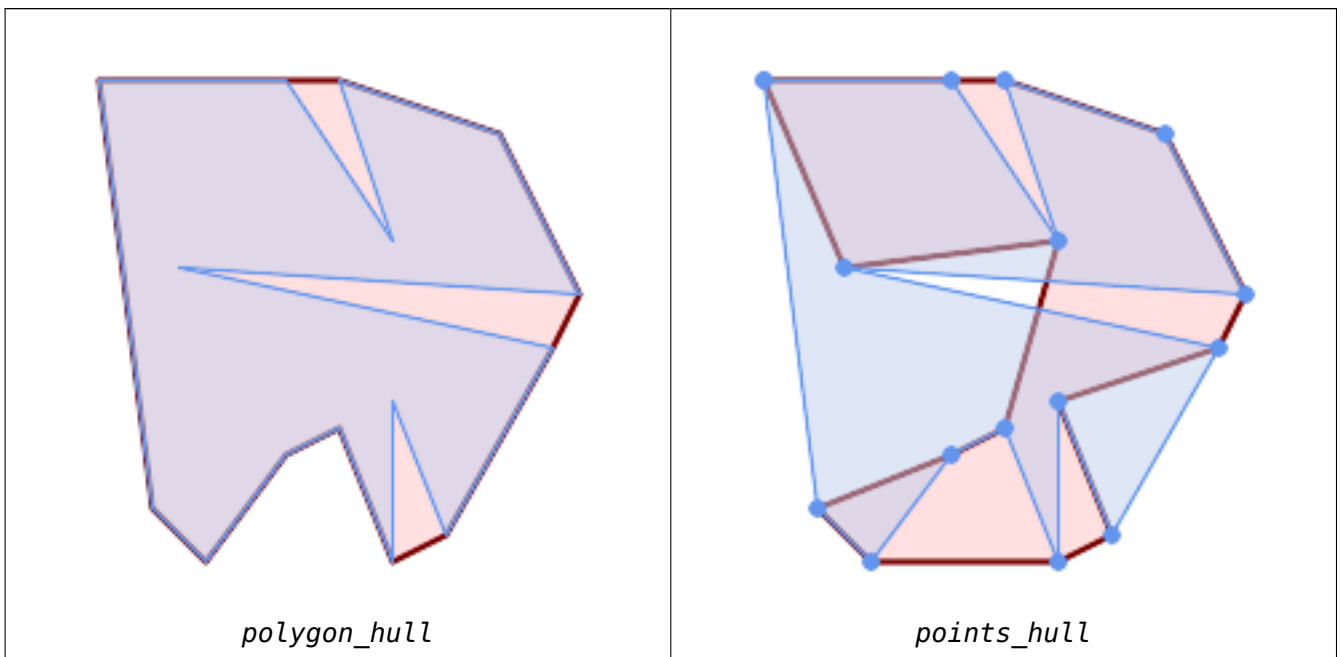
SELECT ST_AsText( ST_ConcaveHull(
  'MULTIPOINT ((132 64), (114 64), (99 64), (81 64), (63 64), (57 49), (52 36), (46 20), (37 20), (26 20), (32 36), (39 55), (43 69), (50 84), (57 100), (63 118), (68 133), (74 149), (81 164), (88 180), (101 180), (112 180), (119 164), (126 149), (132 131), (139 113), (143 100), (150 84), (157 69), (163 51), (168 36),

```

```

(174 20), (163 20), (150 20), (143 36), (139 49), (132 64), (99 151), (92 138), ←
(88 124), (81 109), (74 93), (70 82), (83 82), (99 82), (112 82), (126 82), (121 ←
96), (114 109), (110 122), (103 138), (99 151), (34 27), (43 31), (48 44), (46 ←
58), (52 73), (63 73), (61 84), (72 71), (90 69), (101 76), (123 71), (141 62), ←
(166 27), (150 33), (159 36), (146 44), (154 53), (152 62), (146 73), (134 76), ←
(143 82), (141 91), (130 98), (126 104), (132 113), (128 127), (117 122), (112 ←
133), (119 144), (108 147), (119 153), (110 171), (103 164), (92 171), (86 160), ←
(88 142), (79 140), (72 124), (83 131), (79 118), (68 113), (63 102), (68 93), ←
(35 45))',
0.15, true ) );
---st_astext--
POLYGON ((43 69, 50 84, 57 100, 63 118, 68 133, 74 149, 81 164, 88 180, 101 180, 112 180, ←
119 164, 126 149, 132 131, 139 113, 143 100, 150 84, 157 69, 163 51, 168 36, 174 20, 163 ←
20, 150 20, 143 36, 139 49, 132 64, 114 64, 99 64, 81 64, 63 64, 57 49, 52 36, 46 20, ←
37 20, 26 20, 32 36, 35 45, 39 55, 43 69), (88 124, 81 109, 74 93, 83 82, 99 82, 112 82, ←
121 96, 114 109, 110 122, 103 138, 92 138, 88 124))

```



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，而基于点的外壳不然。

```

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;

```

与 ST\_Collect 一起使用来计算几何形状集的凹壳。

```

-- Compute estimate of infected area based on point observations
SELECT disease_type,
       ST_ConcaveHull( ST_Collect(obs_pnt), 0.3 ) AS geom
FROM disease_obs
GROUP BY disease_type;

```

相关信息

[ST\\_ConvexHull](#), [ST\\_Collect](#), [ST\\_AlphaShape](#), [ST\\_OptimalAlphaShape](#)

### 7.14.6 ST\_ConvexHull

ST\_ConvexHull — 计算几何体的凸包。

#### Synopsis

```
geometry ST_ConvexHull(geometry geomA);
```

#### 描述

计算几何图形的凸包。凸包是包含所有输入几何的最小凸几何体。

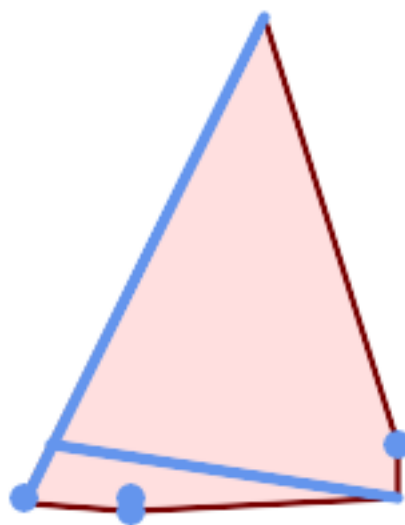
人可以将凸包理解为将橡皮筋套在一组几何图形上而得到的几何图形。它与凹形外壳不同，凹形外壳类似于“收缩包装”几何形状。凸包通常用于根据一组点来确定受影响的区域。

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。

不是聚合函数。要计算一组几何图形的凸包，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 IEC 13249-3: 5.1.16
- ✔ 函数支持 3d 并且不会丢失 z-index。

#### 示例



多边形串和多点的凸包

```
SELECT ST_AsText(ST_ConvexHull(
  ST_Collect(
    ST_GeomFromText('MULTILINESTRING((100 190,10 8),(150 10, 20 30)'),
    ST_GeomFromText('MULTIPOINT(50 5, 150 30, 50 10, 10 10)')
  )));
---st_astext--
POLYGON((50 5,10 8,10 10,100 190,150 30,150 10,50 5))
```

与 `ST_Collect` 一起使用来计算几何形集的凸包。

```
--Get estimate of infected area based on point observations
SELECT d.disease_type,
  ST_ConvexHull(ST_Collect(d.geom)) As geom
FROM disease_obs As d
GROUP BY d.disease_type;
```

相关信息

[ST\\_Collect](#), [ST\\_ConcaveHull](#), [ST\\_MinimumBoundingCircle](#)

### 7.14.7 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

计算几何体点的 **Delaunay 三角剖分**。可设置的容差可用于将附近的点捕捉在一起，在某些情况下提高了健壮性。如果几何形以点的凸包为界。如果几何表示由志代确定：

- 0 - 三角形 POLYGON 的 GEOMETRYCOLLECTION (默认)
- 1 - 三角量的多
- 2 - 三角量的 TIN

这个函数是由 GEOS 模块行的。

可用性：2.1.0



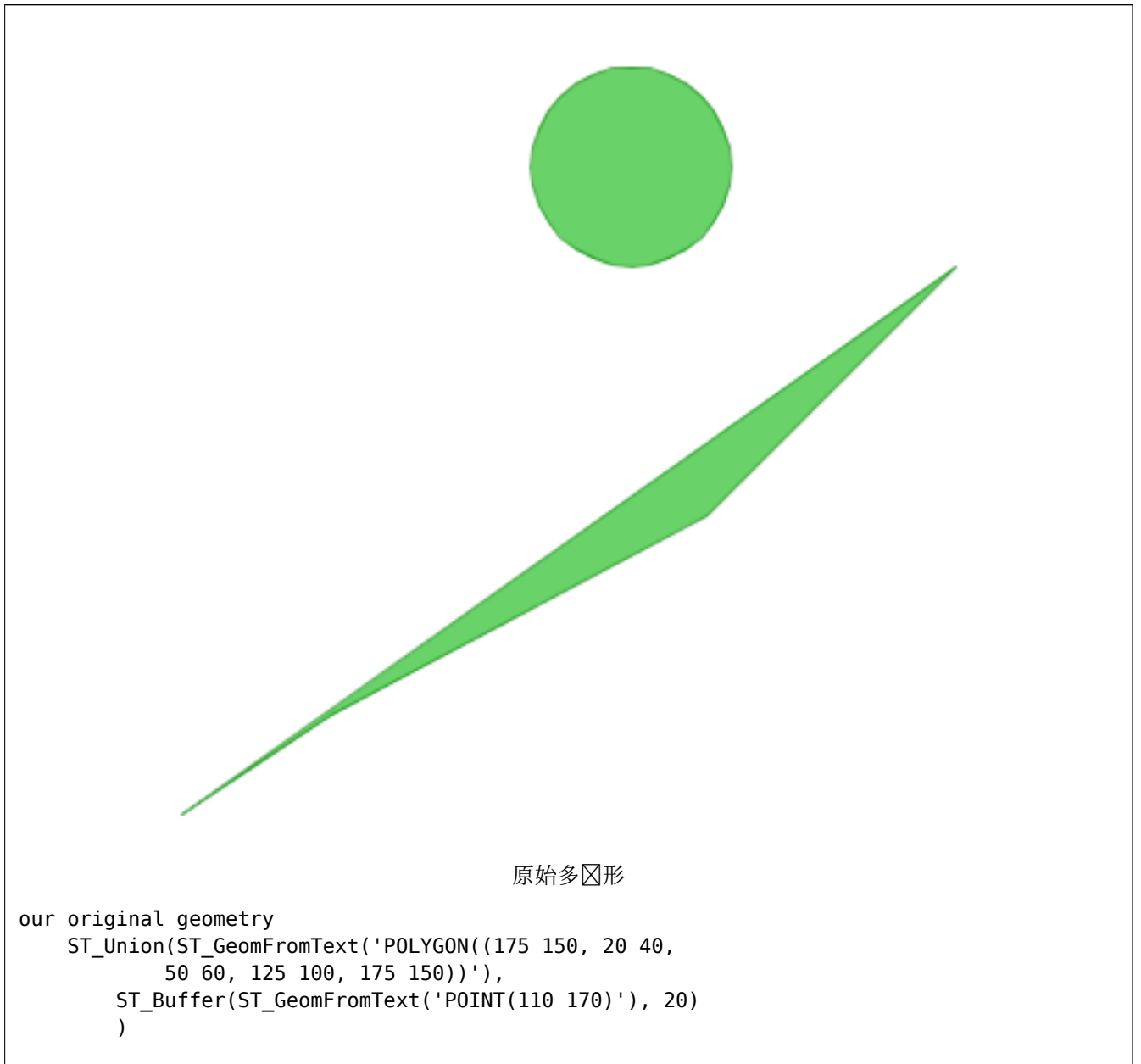
函数支持 3d 并且不会失 z-index。

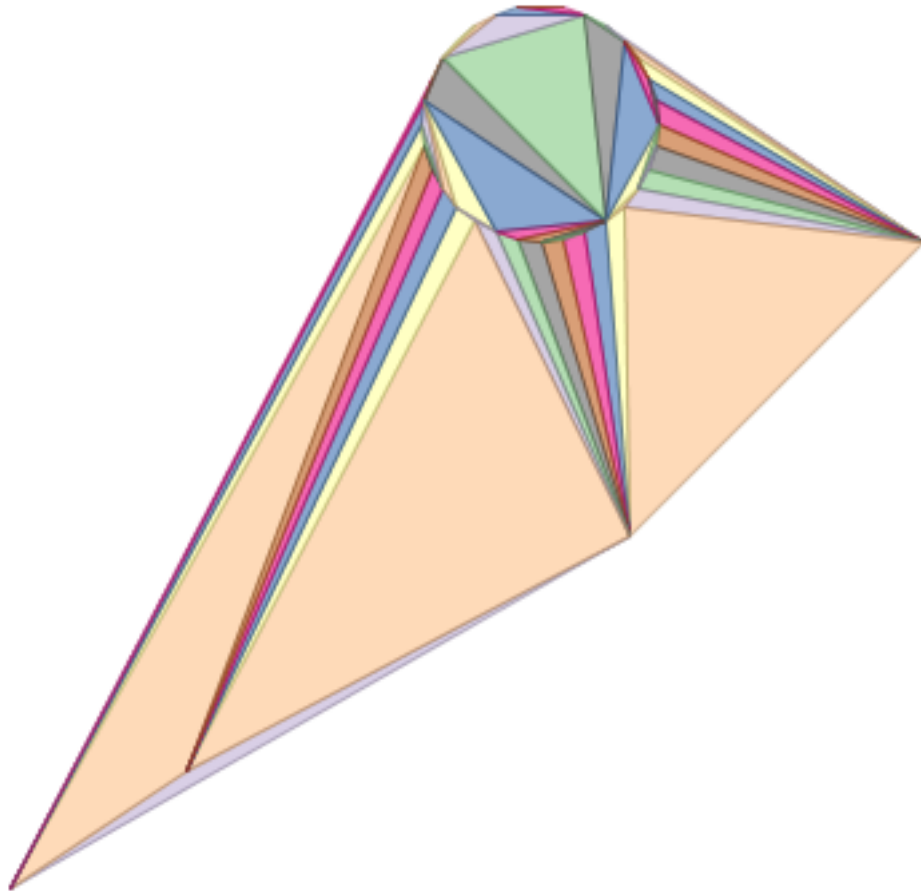


此函数支持三角形和不三角网面 (TIN)。



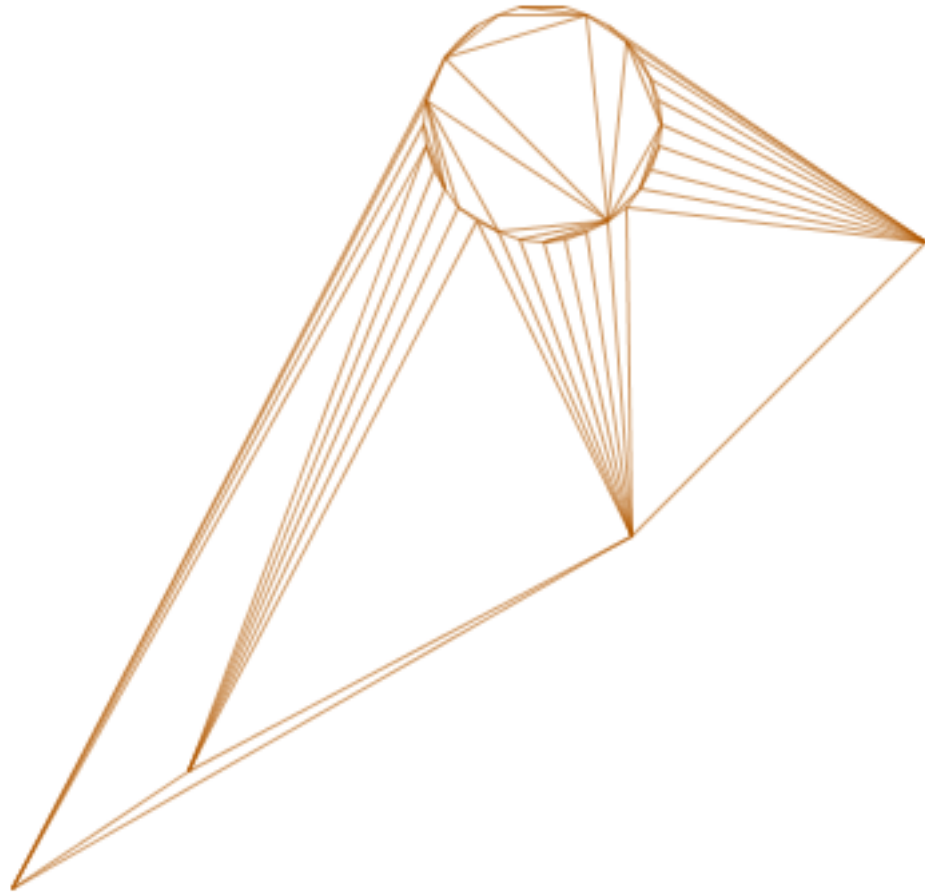
示例





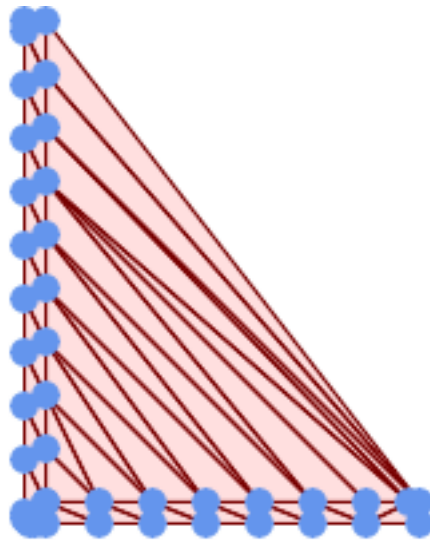
2 个多面形的 *ST\_DelaunayTriangles* : *delaunay* 三角形多面形, 每个三角形以不同颜色为主  
geometries overlaid multilinestring triangles

```
SELECT
  ST_DelaunayTriangles(
    ST_Union(ST_GeomFromText('POLYGON((175 150, 20 40,
      50 60, 125 100, 175 150))'),
    ST_Buffer(ST_GeomFromText('POINT(110 170)'), 20)
  ))
As dtriag;
```



-- *delaunay* 三角形作图串

```
SELECT
  ST_DelaunayTriangles(
    ST_Union(ST_GeomFromText('POLYGON((175 150, 20 40,
      50 60, 125 100, 175 150))'),
    ST_Buffer(ST_GeomFromText('POINT(110 170)'), 20)
  ),0.001,1)
  As dtriag;
```



-- 45 个点的 *delaunay* 三角形作 55 个三角形多边形

this produces a table of 42 points that form an L shape

```
SELECT (ST_DumpPoints(ST_GeomFromText(
'MULTIPOINT(14 14,34 14,54 14,74 14,94 14,114 14,134 14,
150 14,154 14,154 6,134 6,114 6,94 6,74 6,54 6,34 6,
14 6,10 6,8 6,7 7,6 8,6 10,6 30,6 50,6 70,6 90,6 110,6 130,
6 150,6 170,6 190,6 194,14 194,14 174,14 154,14 134,14 114,
14 94,14 74,14 54,14 34,14 14)'))).geom
    INTO TABLE l_shape;
```

output as individual polygon triangles

```
SELECT ST_AsText((ST_Dump(geom)).geom) As wkt
FROM ( SELECT ST_DelaunayTriangles(ST_Collect(geom)) As geom
FROM l_shape) As foo;
```

wkt

```
POLYGON((6 194,6 190,14 194,6 194))
POLYGON((14 194,6 190,14 174,14 194))
POLYGON((14 194,14 174,154 14,14 194))
POLYGON((154 14,14 174,14 154,154 14))
POLYGON((154 14,14 154,150 14,154 14))
POLYGON((154 14,150 14,154 6,154 14))
```

使用具有 Z 值的点的示例。

3D multipoint

```
SELECT ST_AsText(ST_DelaunayTriangles(ST_GeomFromText(
'MULTIPOINT Z(14 14 10, 150 14 100,34 6 25, 20 10 150)')) As wkt;
```

wkt

```
GEOMETRYCOLLECTION Z (POLYGON Z ((14 14 10,20 10 150,34 6 25,14 14 10))
,POLYGON Z ((14 14 10,34 6 25,150 14 100,14 14 10)))
```

相关信息

[ST\\_VoronoiPolygons](#), [ST\\_TriangulatePolygon](#), [ST\\_ConstrainedDelaunayTriangles](#), [ST\\_VoronoiLines](#), [ST\\_Con](#)

## 7.14.8 ST\_FilterByM

ST\_FilterByM — 根据 M 过滤点

### Synopsis

```
geometry ST_FilterByM(geometry geom, double precision min, double precision max = null, boolean
returnM = false);
```

### 描述

根据 M 过滤掉点。返回包含 M 大于或等于最小且小于或等于最大的点的几何形。如果省略最大参数，则考虑最小。如果省略第四个参数，则 m 将不会出现在生成的几何形中。如果生成的几何形其几何型留下的点太少，则将返回空几何形。在几何集合中，没有足点的几何形将被默默地排除。

函数主要与 ST\_SetEffectiveArea 合使用。ST\_EffectiveArea 用其 m 置点的有效区域。使用 ST\_FilterByM，无需任何算，只需即可得几何的化版本



#### Note

与 ST\_FilterByM 相比,当没有足的点足条件,ST\_SimplifyVW 返回的果有所不同。ST\_SimplifyVW 返回具有足点的几何形,而 ST\_FilterByM 返回空几何形



#### Note

注意,返回的几何形可能无效



#### Note

函数返回所有度,包括 Z 和 M

可用性 : 2.5.0

示例

串被

```
SELECT ST_AsText(ST_FilterByM(geom,30)) simplified
FROM (SELECT ST_SetEffectiveArea('LINESTRING(5 2, 3 8, 6 20, 7 25, 10 10)::geometry) geom ←
) As foo;

result

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

相关信息

[ST\\_SetEffectiveArea](#), [ST\\_SimplifyVW](#)

### 7.14.9 ST\_GeneratePoints

`ST_GeneratePoints` — 生成一个包含在多边形 (Polygon) 或多重重多边形 (MultiPolygon) 内的随机点的多点对象。

#### Synopsis

geometry **ST\_GeneratePoints**(geometry g, integer npoints, integer seed = 0);

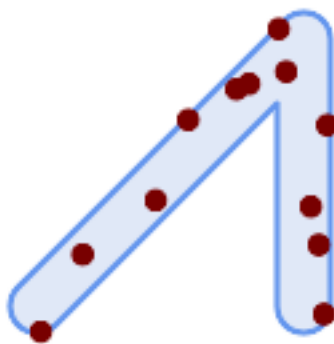
#### 描述

`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



使用随机种子 1996，在原始多边形上生成了一个包含 12 个点的多点对象

```
SELECT ST_GeneratePoints(geom, 12, 1996)
FROM (
  SELECT ST_Buffer(
    ST_GeomFromText(
      'LINESTRING(50 50,150 150,150 50)'),
    10, 'endcap=round join=round') AS geom
) AS s;
```

☑定一个多☑形表 s，返回每个多☑形 12 个☑独的点。每次☑行☑果都会不同。

```
SELECT s.id, dp.path[1] AS pt_id, dp.geom
FROM s, ST_DumpPoints(ST_GeneratePoints(s.geom,12)) AS dp;
```

相关信息

[ST\\_DumpPoints](#)

## 7.14.10 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☑算多点几何的近似几何中☑。几何中位数是最小化到☑入点的距离☑和的点。它提供了一种中心性度量，与☑心（☑心）相比，它☑异常点不太敏感。

☑算法☑行迭代，直到☑☑迭代之☑的距离☑化小于提供的容差参数。如果在 max\_iterations 次迭代后仍未☑足此条件，☑函数会☑生☑☑并退出，除非 fail\_if\_not\_converged ☑置☑ false（默☑☑）。

如果未提供容差参数，☑根据☑入几何☑形的范☑☑算公差☑。

如果存在，☑入点 M ☑将被解☑☑它☑的相☑☑重。

可用性：2.3.0

增☑：2.5.0 添加了☑ M 作☑点☑重的支持。

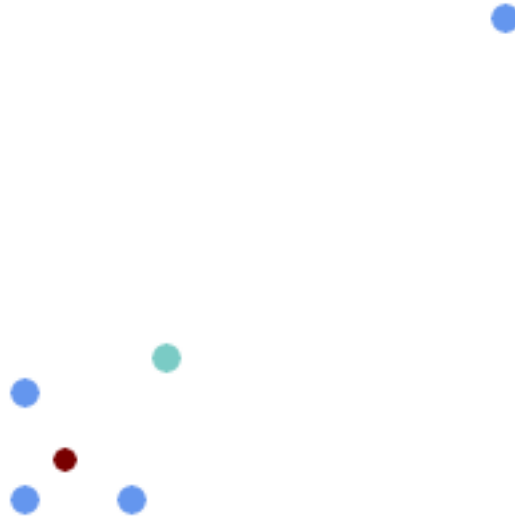


☑函数支持 3d 并且不会☑失 z-index。



☑功能支持 M 坐☑。

示例



多点的几何中点（松石色）和重心（红色）的对比。

```
WITH test AS (
SELECT 'MULTIPOINT((10 10), (10 40), (40 10), (190 190))'::geometry geom)
SELECT
  ST_AsText(ST_Centroid(geom)) centroid,
  ST_AsText(ST_GeometricMedian(geom)) median
FROM test;
```

centroid	median
POINT(62.5 62.5)	POINT(25.01778421249728 25.01778421249728)

(1 row)

相关信息

[ST\\_Centroid](#)

### 7.14.11 ST\_LineMerge

ST\_LineMerge — 返回通线将 MultiLineString 线合在一起形成的线。

#### Synopsis

```
geometry ST_LineMerge(geometry amultilinestring);
geometry ST_LineMerge(geometry amultilinestring, boolean directed);
```

描述

返回通线将 MultiLineString 的线元素连接在一起而形成的 LineString 或 MultiLineString。线在 2 线交叉点的端点处连接。线不会在三向或更高次的交叉点上连接。

如果定向线 TRUE，线 ST\_LineMerge 将不会更改 LineStrings 内的点序，因此方向相反的线不会被合并



**Note**

与 MultiLineString/LineString 一起使用。其他几何型返回空的 GeometryCollection

这个函数是由 GEOS 模块实现的。

增加：3.3.0 接受定向参数。

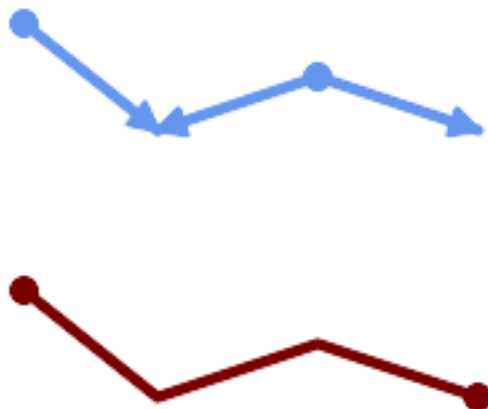
需要 GEOS >= 3.11.0 才能使用定向参数。

可用性：1.1.0

**Warning**

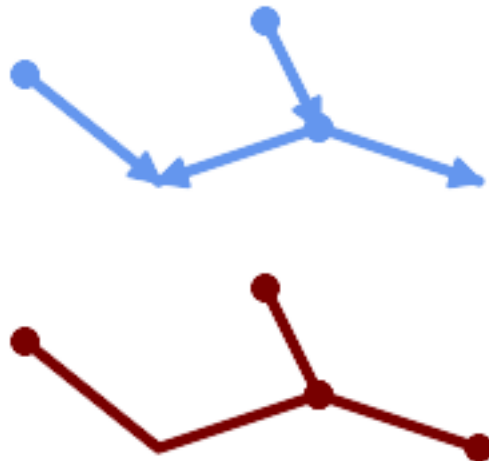
此函数剥离 M 度。

示例



合并不同方向的。

```
SELECT ST_AsText(ST_LineMerge(  
'MULTILINESTRING((10 160, 60 120), (120 140, 60 120), (120 140, 180 120))'  
));  
-----  
LINESTRING(10 160,60 120,120 140,180 120)
```

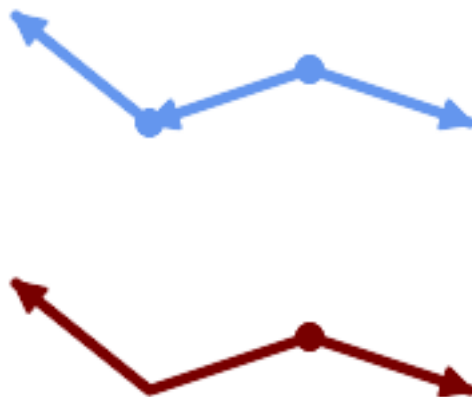


在交点的度数大于 2 时，这些段不会合并。

```
SELECT ST_AsText(ST_LineMerge(
'MULTILINESTRING((10 160, 60 120), (120 140, 60 120), (120 140, 180 120), (100 180, 120 140))'
));
-----
MULTILINESTRING((10 160,60 120,120 140),(100 180,120 140),(120 140,180 120))
```

如果由于不接触而无法合并，返回原始 MultiLineString。

```
SELECT ST_AsText(ST_LineMerge(
'MULTILINESTRING((-29 -27, -30 -29.7, -36 -31, -45 -33), (-45.2 -33.2, -46 -32))'
));
-----
MULTILINESTRING((-45.2 -33.2, -46 -32), (-29 -27, -30 -29.7, -36 -31, -45 -33))
```



如果 *directed* = TRUE，这些不合并具有相反方向的段。

```
SELECT ST_AsText(ST_LineMerge(
'MULTILINESTRING((60 30, 10 70), (120 50, 60 30), (120 50, 180 30))',
TRUE));
-----
MULTILINESTRING((120 50,60 30,10 70),(120 50,180 30))
```

显示理 Z 度的示例。

```
SELECT ST_AsText(ST_LineMerge(
  'MULTILINESTRING((-29 -27 11,-30 -29.7 10,-36 -31 5,-45 -33 6), (-29 -27 12,-30 -29.7 ←
    5), (-45 -33 1,-46 -32 11))'
));
-----
LINESTRING Z (-30 -29.7 5,-29 -27 11,-30 -29.7 10,-36 -31 5,-45 -33 1,-46 -32 11)
```

相关信息

[ST\\_Segmentize](#), [ST\\_LineSubstring](#)

### 7.14.12 ST\_MaximumInscribedCircle

ST\_MaximumInscribedCircle — 算几何体中包含的最大。

#### Synopsis

(geometry, geometry, double precision) **ST\_MaximumInscribedCircle**(geometry geom);

#### 描述

找（多）多形内包含的最大，或者不与任何和点重的最大。返回包含字段的：

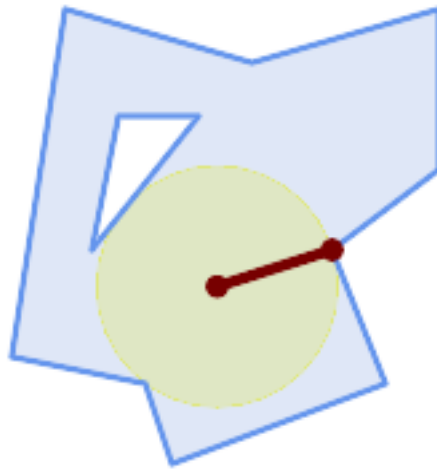
- center-的中心点
- nearest-几何上最接近中心的点
- radius-的半径

于多形入，内接于界内，使用内作界。于性和点入，内切于入的凸包内，使用入和点作一步的界。

可用性：3.1.0。

需要 GEOS >= 3.9.0。

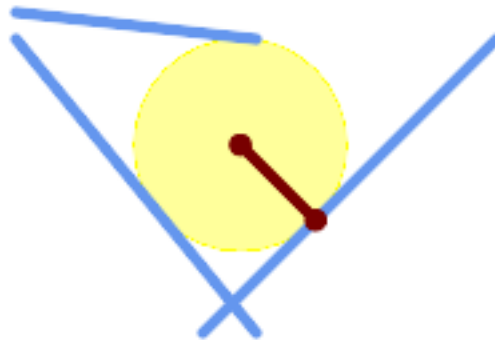
示例



多边形的最大内切圆。返回中心、最近点和半径。

```
SELECT radius, ST_AsText(center) AS center, ST_AsText(nearest) AS nearest
FROM ST_MaximumInscribedCircle(
  'POLYGON ((40 180, 110 160, 180 180, 180 120, 140 90, 160 40, 80 10, 70 40, 20 50, 40 180), (60 140, 50 90, 90 140, 60 140))');
```

radius	center	nearest
45.165845650018	POINT(96.953125 76.328125)	POINT(140 90)



多线串的最大内切圆。返回中心、最近点和半径。

相关信息

[ST\\_MinimumBoundingRadius](#), [ST\\_LargestEmptyCircle](#)

### 7.14.13 ST\_LargestEmptyCircle

ST\_LargestEmptyCircle — 计算不与几何形重叠的最大圆。

#### 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。(多边形几何形可以作为障碍物包含在内，但使用它的边界。)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。

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。

返回包含字段的圆：

- center-圆的中心点
- nearest-几何上最接近中心的点
- radius-圆的半径

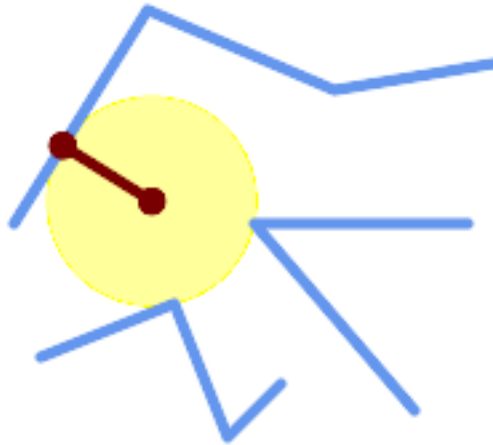
要找到多边形内部最大的空圆，参看 [ST\\_MaximumInscribedCircle](#)。

可用性：3.4.0。

需要 GEOS >= 3.9.0。

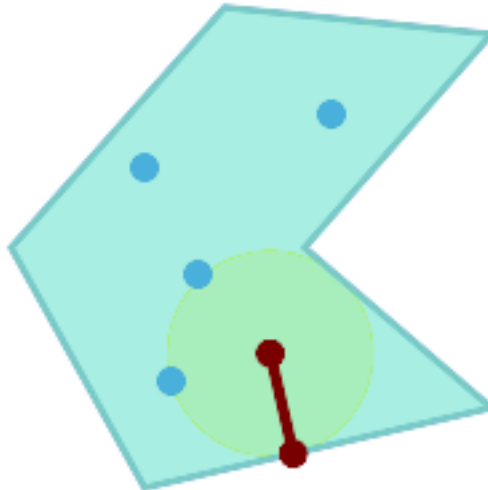
#### 示例

```
SELECT radius,
       center,
       nearest
FROM ST_LargestEmptyCircle(
  'MULTILINESTRING (
    (10 100, 60 180, 130 150, 190 160),
    (20 50, 70 70, 90 20, 110 40),
    (160 30, 100 100, 180 100))');
```



— 多边形内最大的空圆。

```
SELECT radius,
       center,
       nearest
FROM ST_LargestEmptyCircle(
  ST_Collect(
    'MULTIPOINT ((70 50), (60 130), (130 150), (80 90))'::geometry,
    'POLYGON ((90 190, 10 100, 60 10, 190 40, 120 100, 190 180, 90 190))'::geometry) ←
    0,
    'POLYGON ((90 190, 10 100, 60 10, 190 40, 120 100, 190 180, 90 190))'::geometry
  );
```



— 多边形内最大的空圆，圆束位于多边形内。圆束多边形边界必须作为障碍物包含在内，并指定圆心的圆束。

#### 相关信息

[ST\\_MinimumBoundingRadius](#)

### 7.14.14 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，因此输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进近似。对于近似不合适的圆，可以使用 [ST\\_MinimumBoundingRadius](#)。

与 [ST\\_Collect](#) 一起使用以得到几何图形的最小边界。

要计算位于最小圆（“最大直径”）上的点，使用 [ST\\_LongestLine](#)。

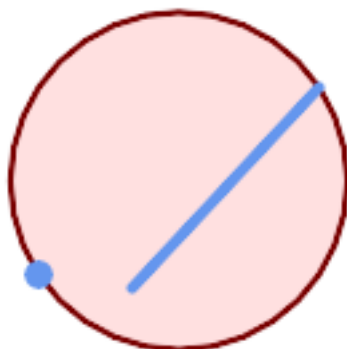
多边形的面积除以最小包含圆的面积之比称为 *REOCK* 致度得分。

该函数是由 GEOS 模块实现的。

可用性：1.4.0

#### 示例

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



点和线串的最小外接圆。用 8 段近似四分之一圆。

```

SELECT ST_AsText(ST_MinimumBoundingCircle(
    ST_Collect(
        ST_GeomFromText('LINESTRING(55 75,125 150)'),
        ST_Point(20, 80)), 8
    )) As wktmbc;
wktmbc
-----
POLYGON((135.59714732062 115,134.384753327498 102.690357210921,130.79416296937 ↔
  90.8537670908995,124.963360620072 79.9451031602111,117.116420743937 ↔
  70.3835792560632,107.554896839789 62.5366393799277,96.6462329091006 ↔
  56.70583703063,84.8096427890789 53.115246672502,72.5000000000001 ↔
  51.9028526793802,60.1903572109213 53.1152466725019,48.3537670908996 ↔
  56.7058370306299,37.4451031602112 62.5366393799276,27.8835792560632 ↔
  70.383579256063,20.0366393799278 79.9451031602109,14.20583703063 ↔
  90.8537670908993,10.615246672502 102.690357210921,9.40285267938019 115,10.6152466725019 ↔
  127.309642789079,14.2058370306299 139.1462329091,20.0366393799275 ↔
  150.054896839789,27.883579256063 159.616420743937,
  37.4451031602108 167.463360620072,48.3537670908992 173.29416296937,60.190357210921 ↔
  176.884753327498,
  72.4999999999998 178.09714732062,84.8096427890786 176.884753327498,96.6462329091003 ↔
  173.29416296937,107.554896839789 167.463360620072,
  117.116420743937 159.616420743937,124.963360620072 150.054896839789,130.79416296937 ↔
  139.146232909101,134.384753327498 127.309642789079,135.59714732062 115))

```

相关信息

[ST\\_Collect](#), [ST\\_MinimumBoundingRadius](#), [ST\\_LargestEmptyCircle](#), [ST\\_LongestLine](#)

### 7.14.15 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` 一起使用以得一几何图形的最小边界。

要计算位于最小圆（“最大直径”）上的个点，使用 `ST_LongestLine`。

可用性-2.3.0



示例

```
SELECT ST_AsText(center), radius FROM ST_MinimumBoundingRadius('POLYGON((26426 65078,26531 65242,26075 65136,26096 65427,26426 65078))');
```

st_astext	radius
POINT(26284.8418027133 65267.1145090825)	247.436045591407

相关信息

[ST\\_Collect](#), [ST\\_MinimumBoundingCircle](#), [ST\\_LongestLine](#)

### 7.14.16 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矩形。注意，可能存在多个矩形的情况。在退化输入的情况下可能返回 `Point` 或 `LineString`。

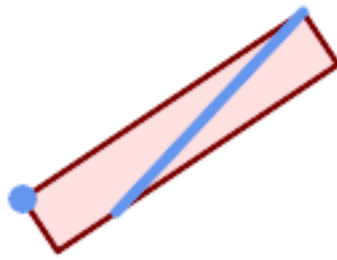
可用性：2.5.0。

需要 GEOS  $\geq$  3.6.0。

示例

```
SELECT ST_AsText(ST_OrientedEnvelope('MULTIPOINT ((0 0), (-1 -1), (3 2))');
```

st_astext
POLYGON((3 2,2.88 2.16,-1.12 -0.84,-1 -1,3 2))



点和线串的最小外接矩形。

```
SELECT ST_AsText(ST_OrientedEnvelope(
    ST_Collect(
        ST_GeomFromText('LINESTRING(55 75,125 150)'),
        ST_Point(20, 80))
    )) As wktenv;
wktenv
-----
POLYGON((19.9999999999997 79.9999999999999,33.0769230769229 ↔
        60.3846153846152,138.076923076924 130.384615384616,125.000000000001 ↔
        150.000000000001,19.9999999999997 79.9999999999999))
```

相关信息

[ST\\_Envelope](#) [ST\\_MinimumBoundingCircle](#)

### 7.14.17 ST\_OffsetCurve

`ST_OffsetCurve` — 返回距输入线串定距离和方向的偏移线串。

#### Synopsis

```
geometry ST_OffsetCurve(geometry line, float signed_distance, text style_parameters=“”);
```

#### 描述

返回距输入线串定距离和方向的偏移线串。返回几何线串的所有点与输入几何线串的距离不超过定距离。线串于线串中心线平行线串很有用。

线串于正距离，偏移位于输入线串的左线串并保持相同的方向。线串于负距离，它位于右线串且方向相反。

距离线串位以空线串参考系的线串位来度量。

注意，线串于某些拼线串形状的输入几何线串，线串出可能是 MULTILINESTRING 或 EMPTY。

可线串的第三个参数允许指定空白分隔的线串 = 线串的列表来线串整操作，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' : 接合式 (默认为 "round")。"mitre" 也被接受为 "mitre" 的同义词。
- 'mitre\_limit=#.#' : 斜接比率限制 (影响斜接式)。"miter\_limit" 也被接受为 "mitre\_limit" 的同义词。

这些函数是由 GEOS 模块实现的。

Behavior changed in GEOS 3.11 so offset curves now have the same direction as the input line, for both positive and negative offsets.

可用性 : 2.0

增强 : 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

计算道路周围的开放缓冲区

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




15, 'quad\_segs=4 join=round' 原始☒及其偏移  
15 个☒位。

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

output

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

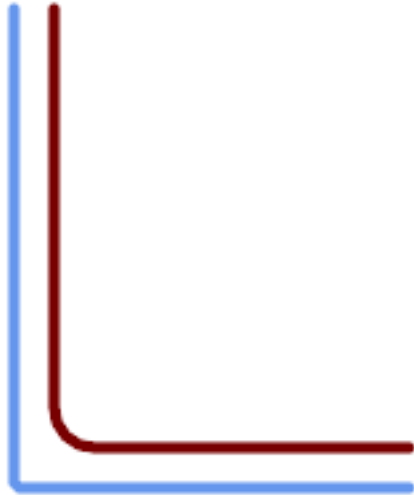


-15, 'quad\_segs=4 join=round' 原始☒及其偏移  
量-15 个☒位

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

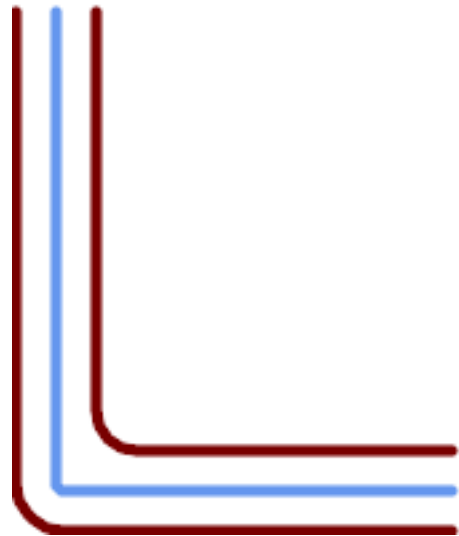
notsocurvy

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



双偏移以  $\ominus$  得更多曲  $\ominus$ ,  $\ominus$  注意第一个反  $\ominus$  方向, 因此  $-30+15 = -15$

```
SELECT ST_AsText(ST_OffsetCurve(  $\leftarrow$ 
  ST_OffsetCurve(geom,  $\leftarrow$ 
    -30, 'quad_segs=4 join=round'), -15,  $\leftarrow$ 
    'quad_segs=4 join=round')) As morecurvy
FROM ST_GeomFromText(
'LINESTRING(164 16,144 16,124 16,104  $\leftarrow$ 
  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)')  $\leftarrow$ 
  As geom;
morecurvy
LINESTRING(164 31,46 31,40.2597485145236  $\leftarrow$ 
  32.1418070123307,
  35.3933982822018 35.3933982822018,
  32.1418070123307 40.2597485145237,31  $\leftarrow$ 
  46,31 195)
```



双偏移以  $\ominus$  得更多曲  $\ominus$ , 与常  $\ominus$  偏移 15  $\ominus$  合以  $\ominus$  得平行  $\ominus$ . 与原  $\ominus$  重  $\ominus$ .

```
SELECT ST_AsText(ST_Collect(
  ST_OffsetCurve(geom, 15, 'quad_segs=4  $\leftarrow$ 
    join=round'),
  ST_OffsetCurve(ST_OffsetCurve(geom,  $\leftarrow$ 
    -30, 'quad_segs=4 join=round'), -15,  $\leftarrow$ 
    'quad_segs=4 join=round')
) ) As parallel_curves
FROM ST_GeomFromText(
'LINESTRING(164 16,144 16,124 16,104  $\leftarrow$ 
  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)')  $\leftarrow$ 
  As geom;
parallel curves
MULTILINESTRING((164 1,18  $\leftarrow$ 
  1,12.2597485145237 2.1418070123307,
  7.39339828220179  $\leftarrow$ 
  5.39339828220179,5.39339828220179 7.39339828220179,
  2.14180701233067 12.2597485145237,1 18,1  $\leftarrow$ 
  195),
(164 31,46 31,40.2597485145236  $\leftarrow$ 
  32.1418070123307,35.3933982822018 35.3933982822018,
  32.1418070123307 40.2597485145237,31  $\leftarrow$ 
  46,31 195))
```

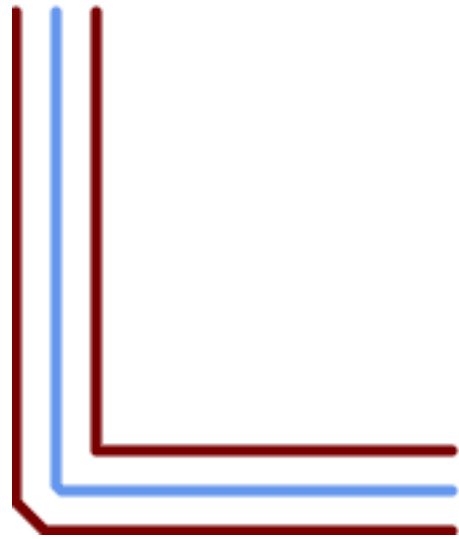


15、'quad\_segs=4 join=bevel' 示原始

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

output

```
LINESTRING(164 1,18 1,7.39339828220179 ←
  5.39339828220179,
  5.39339828220179 7.39339828220179,1 ←
  18,1 195)
```



15、-15 收集, join=mitre mitre\_limit=2.1

```
SELECT ST_AsText(ST_Collect(
  ST_OffsetCurve(geom, 15, 'quad_segs=4 ←
    join=mitre mitre_limit=2.2'),
  ST_OffsetCurve(geom, -15, 'quad_segs ←
    =4 join=mitre mitre_limit=2.2')
) )
FROM ST_GeomFromText(
'LINESTRING(164 16,144 16,124 16,104 ←
  16,84 16,64 16,
  44 16,24 16,20 16,18 16,17 17,
  16 18,16 20,16 40,16 60,16 80,16 100,
  16 120,16 140,16 160,16 180,16 195)') ←
  As geom;
```

output

```
MULTILINESTRING((164 1,11.7867965644036 ←
  1,1 11.7867965644036,1 195),
(31 195,31 31,164 31))
```

相关信息

[ST\\_Buffer](#)

### 7.14.18 ST\_PointOnSurface

ST\_PointOnSurface — 算保位于多形或几何体上的点。

#### Synopsis

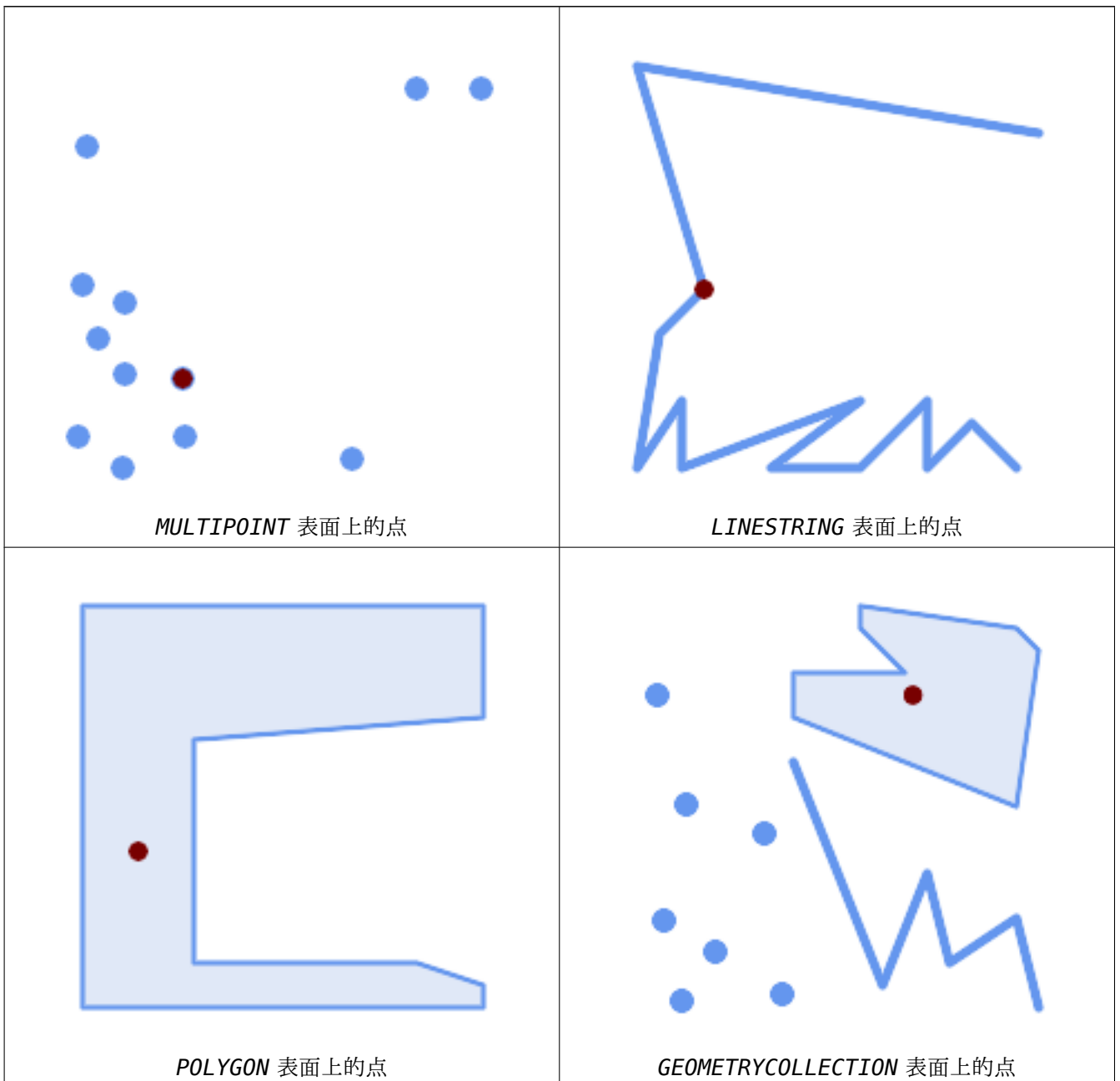
geometry **ST\_PointOnSurface**(geometry g1);

描述

返回保证位于曲面内部的 POINT (POLYGON、MULTIPOLYGON 和 CURVEPOLYGON)。在 PostGIS 中，此函数也适用于面和点几何形状。

- ✔ 此方法符合了 SQL 1.1 的 OGC 功能规范。 s3.2.14.2 // s3.2.18.2
- ✔ 此方法符合了 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 支持所有的几何类型。
- ✔ 此函数支持 3d 并且不会丢失 z-index。

示例



```

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

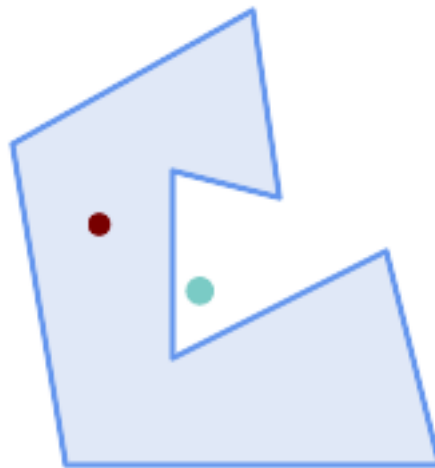
SELECT ST_AsText(ST_PointOnSurface('LINESTRING(0 5, 0 10)')::geometry);
-----
POINT(0 5)

SELECT ST_AsText(ST_PointOnSurface('POLYGON((0 0, 0 5, 5 5, 5 0, 0 0))')::geometry);
-----
POINT(2.5 2.5)

SELECT ST_AsEWKT(ST_PointOnSurface(ST_GeomFromEWKT('LINESTRING(0 5 1, 0 0 1, 0 10 2)')));
-----
POINT(0 0 1)

```

示例：ST\_PointOnSurface 的结果保证位于多边形内，而 ST\_Centroid 计算的点可能位于多边形外。



红色：表面上的点；绿色：重心

```

SELECT ST_AsText(ST_PointOnSurface(geom)) AS pt_on_surf,
       ST_AsText(ST_Centroid(geom)) AS centroid
FROM (SELECT 'POLYGON ((130 120, 120 190, 30 140, 50 20, 190 20,
                       170 100, 90 60, 90 130, 130 120))'::geometry AS geom) AS t;

```

pt_on_surf	centroid
POINT(62.5 110)	POINT(100.18264840182648 85.11415525114155)

相关信息

[ST\\_Centroid](#), [ST\\_MaximumInscribedCircle](#)

### 7.14.19 ST\_Polygonize

ST\_Polygonize — 计算由一系列几何图形的边形成的多边形集合。



## 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形成的多边形。如果输入线条未形成任何多边形，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

输入线条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**。



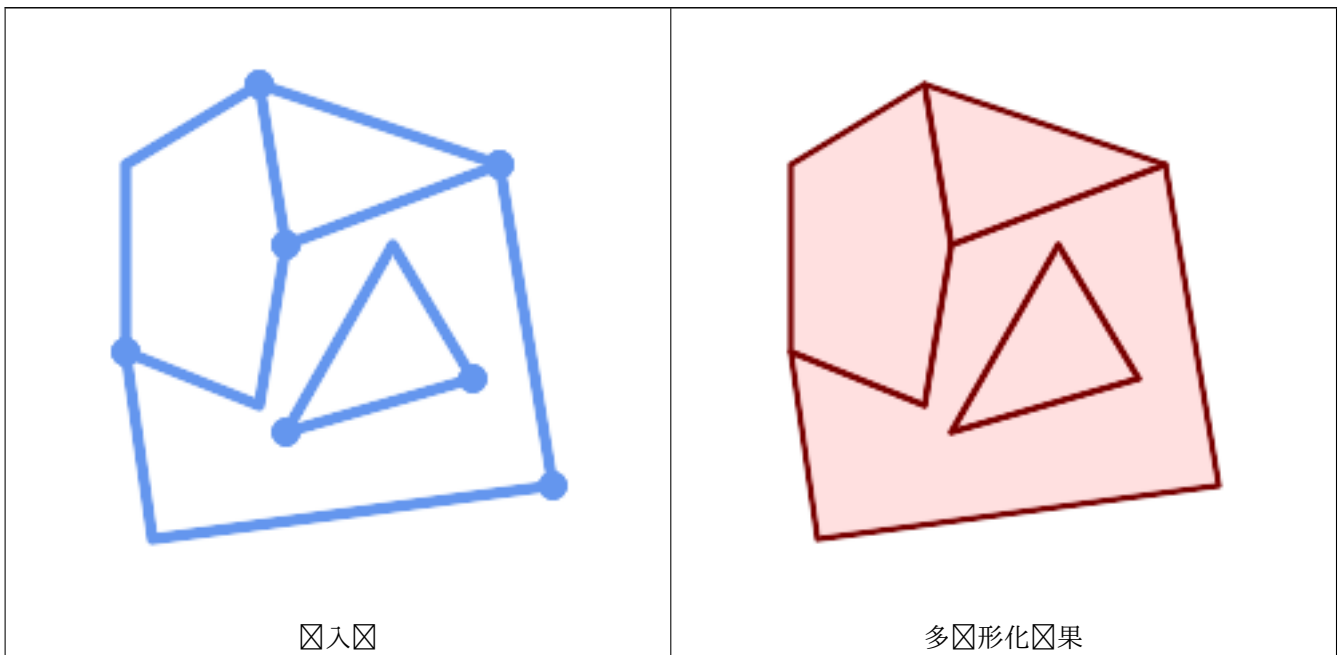
#### Note

使用外部工具可能很管理几何集合。使用 **ST\_Dump** 将多边形结果拆分为单独的多边形。

这个函数是由 GEOS 模块执行的。

可用性：1.0.0RC1

### 示例



```
WITH data(geom) AS (VALUES
  ('LINESTRING (180 40, 30 20, 20 90)::geometry)
```

```

,('LINESTRING (180 40, 160 160)::geometry)
,('LINESTRING (80 60, 120 130, 150 80)::geometry)
,('LINESTRING (80 60, 150 80)::geometry)
,('LINESTRING (20 90, 70 70, 80 130)::geometry)
,('LINESTRING (80 130, 160 160)::geometry)
,('LINESTRING (20 90, 20 160, 70 190)::geometry)
,('LINESTRING (70 190, 80 130)::geometry)
,('LINESTRING (70 190, 160 160)::geometry)
)
SELECT ST_AsText( ST_Polygonize( geom ) )
      FROM data;

-----
GEOMETRYCOLLECTION (POLYGON ((180 40, 30 20, 20 90, 70 70, 80 130, 160 160, 180 40), (150 ←
      80, 120 130, 80 60, 150 80)),
      POLYGON ((20 90, 20 160, 70 190, 80 130, 70 70, 20 90)),
      POLYGON ((160 160, 80 130, 70 190, 160 160)),
      POLYGON ((80 60, 120 130, 150 80, 80 60)))

```

多边形串表：

```

SELECT ST_AsEWKT(ST_Polygonize(geom_4269)) As geomtextrep
FROM (SELECT geom_4269 FROM ma.suffolk_edges) As foo;

-----
SRID=4269;GEOMETRYCOLLECTION(POLYGON((-71.040878 42.285678,-71.040943 42.2856,-71.04096 ←
      42.285752,-71.040878 42.285678)),
POLYGON((-71.17166 42.353675,-71.172026 42.354044,-71.17239 42.354358,-71.171794 ←
      42.354971,-71.170511 42.354855,
-71.17112 42.354238,-71.17166 42.353675)))

--Use ST_Dump to dump out the polygonize geoms into individual polygons
SELECT ST_AsEWKT((ST_Dump(t.polycoll)).geom) AS geomtextrep
FROM (SELECT ST_Polygonize(geom_4269) AS polycoll
      FROM (SELECT geom_4269 FROM ma.suffolk_edges)
      As foo) AS t;

-----
SRID=4269;POLYGON((-71.040878 42.285678,-71.040943 42.2856,-71.04096 42.285752,
-71.040878 42.285678))
SRID=4269;POLYGON((-71.17166 42.353675,-71.172026 42.354044,-71.17239 42.354358
,-71.171794 42.354971,-71.170511 42.354855,-71.17112 42.354238,-71.17166 42.353675))

```

相关信息

[ST\\_BuildArea](#), [ST\\_Dump](#), [ST\\_Node](#)

## 7.14.20 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 gridsz);

## 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

```
SELECT ST_AsText(ST_ReducePrecision('POINT(1.412 19.323)', 0.1));
      st_astext
-----
POINT(1.4 19.3)

SELECT ST_AsText(ST_ReducePrecision('POINT(1.412 19.323)', 1.0));
      st_astext
-----
POINT(1 19)

SELECT ST_AsText(ST_ReducePrecision('POINT(1.412 19.323)', 10));
      st_astext
-----
POINT(0 20)
```

降低精度可以减少点数量

```
SELECT ST_AsText(ST_ReducePrecision('LINESTRING (10 10, 19.6 30.1, 20 30, 20.3 30, 40 40)', ←
      1));
      st_astext
-----
LINESTRING (10 10, 20 30, 40 40)
```

如果需要的，精度降低会分割多边形以确保有效性

```
SELECT ST_AsText(ST_ReducePrecision('POLYGON ((10 10, 60 60.1, 70 30, 40 40, 50 10, 10 10)) ←
      ', 10));
      st_astext
-----
MULTIPOLYGON (((60 60, 70 30, 40 40, 60 60)), ((40 40, 50 10, 10 10, 40 40)))
```

## 相关信息

[ST\\_SnapToGrid](#), [ST\\_Simplify](#), [ST\\_SimplifyVW](#)

### 7.14.21 ST\_SharedPaths

ST\_SharedPaths — 返回一个集合，其中包含  $n$  个  $n$  入  $n$  串/多  $n$  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  $n$  个  $n$  入几何  $n$  形共享的路径。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  $n$  出的。

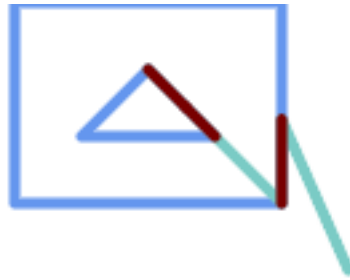
$n$  个函数是由 GEOS 模  $n$  行的。

可用性: 2.0.0

示例： $n$  找共享路径



*multilinestring* 和 *linestring*



*multilinestring* 和 *linestring* 与原始几何形重形的共享路径。

```
SELECT ST_AsText(
  ST_SharedPaths(
    ST_GeomFromText('MULTILINESTRING((26 125,26 200,126 200,126 125,26 125),
      (51 150,101 150,76 175,51 150))'),
    ST_GeomFromText('LINESTRING(151 100,126 156.25,126 125,90 161, 76 175)')
  )
) As wkt

          wkt
-----
GEOMETRYCOLLECTION(MULTILINESTRING((126 156.25,126 125),
(101 150,90 161),(90 161,76 175)),MULTILINESTRING EMPTY)
```

same example but linestring orientation flipped

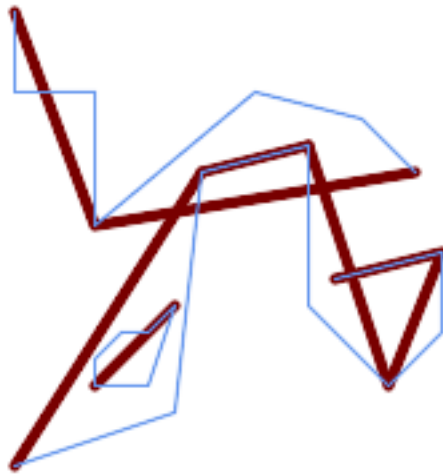
```
SELECT ST_AsText(
  ST_SharedPaths(
    ST_GeomFromText('LINESTRING(76 175,90 161,126 125,126 156.25,151 100)'),
    ST_GeomFromText('MULTILINESTRING((26 125,26 200,126 200,126 125,26 125),
      (51 150,101 150,76 175,51 150))')
  )
) As wkt

          wkt
-----
GEOMETRYCOLLECTION(MULTILINESTRING EMPTY,
MULTILINESTRING((76 175,90 161),(90 161,101 150),(126 125,126 156.25)))
```

相关信息

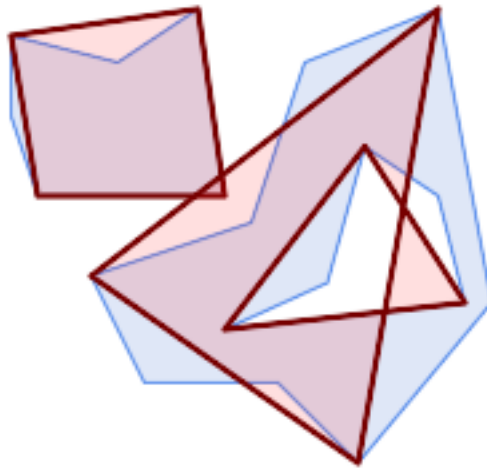
[ST\\_Dump](#), [ST\\_GeometryN](#), [ST\\_NumGeometries](#)





```
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))',
  40);
```

简化一个 MultiPolygon。多边形的结果可能是无效的。



```
SELECT ST_Simplify(
  'MULTIPOLYGON (((90 110, 80 180, 50 160, 10 170, 10 140, 20 110, 90 110)), ((40 80, 100 ←
    100, 120 160, 170 180, 190 70, 140 10, 110 40, 60 40, 40 80), (180 70, 170 110, 142.5 ←
    128.5, 128.5 77.5, 90 60, 180 70)))',
  40);
```

相关信息

[ST\\_IsSimple](#), [ST\\_SimplifyPreserveTopology](#), [ST\\_SimplifyVW](#), [ST\\_CoverageSimplify](#), [Topology ST\\_Simplify](#)

### 7.14.23 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，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 的方位为准。只要保留拓扑，简化会移除在简化线条的公差距离内的点。如果输入是有效和闭合的，结果也将是有效和闭合的。

该函数可用于任何几何类型（包括 **GeometryCollections**），但只有面和多边形元素会被简化。对于多边形输入，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

该函数不保留多边形之间共享的边界。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

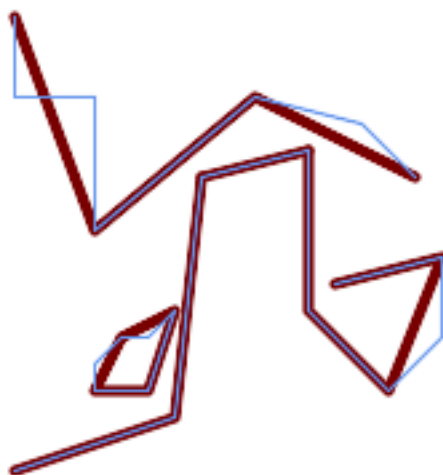
该函数与 **ST\_Simplify** 相同的示例，**ST\_SimplifyPreserveTopology** 防止过度简化。最多可以简化成一个正方形。

```
SELECT ST_Npoints(geom) AS np_before,
       ST_NPoints(ST_SimplifyPreserveTopology(geom, 0.1)) AS np01_notbadcircle,
       ST_NPoints(ST_SimplifyPreserveTopology(geom, 0.5)) AS np05_notquitecircle,
       ST_NPoints(ST_SimplifyPreserveTopology(geom, 1)) AS np1_octagon,
       ST_NPoints(ST_SimplifyPreserveTopology(geom, 10)) AS np10_square,
       ST_NPoints(ST_SimplifyPreserveTopology(geom, 100)) AS np100_stillsquare
FROM (SELECT ST_Buffer('POINT(1 3)', 10,12) AS geom) AS t;
```

np_before	np01_notbadcircle	np05_notquitecircle	np1_octagon	np10_square	↔
49	33	17	9	5	↔
	5				

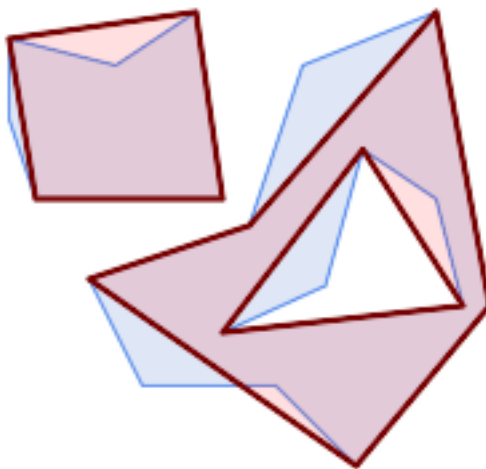
简化一个面，保留非相交面的拓扑。





```
SELECT ST_SimplifyPreserveTopology(
  'MULTILINESTRING ((20 180, 20 150, 50 150, 50 100, 110 150, 150 140, 170 120), (20 10, 80 ←
    30, 90 120), (90 120, 130 130), (130 130, 130 70, 160 40, 180 60, 180 90, 140 80), ←
    (50 40, 70 40, 80 70, 70 60, 60 60, 50 50, 50 40))',
  40);
```

☒化一个 MultiPolygon, 保留外壳和内☒的拓扑。



```
SELECT ST_SimplifyPreserveTopology(
  'MULTIPOLYGON (((90 110, 80 180, 50 160, 10 170, 10 140, 20 110, 90 110)), ((40 80, 100 ←
    100, 120 160, 170 180, 190 70, 140 10, 110 40, 60 40, 40 80), (180 70, 170 110, 142.5 ←
    128.5, 128.5 77.5, 90 60, 180 70)))',
  40);
```

相关信息

[ST\\_Simplify](#), [ST\\_SimplifyVW](#), [ST\\_CoverageSimplify](#)

#### 7.14.24 ST\_SimplifyPolygonHull

ST\_SimplifyPolygonHull — ☒算多☒形几何的☒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产生与输入相同的结果。

The reduction in vertex count is controlled by the `vertex_fraction` parameter, which is a number in the range 0 to 1. Lower values produce simpler results, with smaller vertex count and less concaveness. For both outer and inner hulls a vertex fraction of 1.0 produces the original geometry. For outer hulls a value of 0.0 produces the convex hull (for a single polygon); for inner hulls it produces a triangle.

简化过程逐步去除包含最少面积的凹角来执行，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。

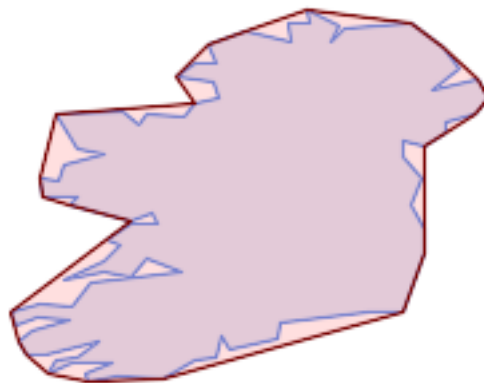
为了获得包含相同线段的几何体的更好结果，可能需要“分段”输入，如下所示。

这个函数是由 GEOS 模块执行的。

可用性：3.3.0。

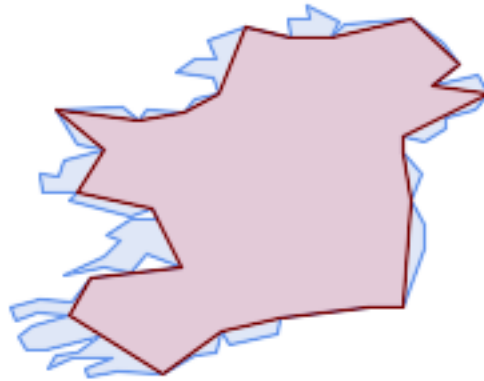
需要 GEOS >= 3.11.0。

### 示例



多边形的外壳

```
SELECT ST_SimplifyPolygonHull(
  'POLYGON ((131 158, 136 163, 161 165, 173 156, 179 148, 169 140, 186 144, 190 137, 185 ↵
    131, 174 128, 174 124, 166 119, 158 121, 158 115, 165 107, 161 97, 166 88, 166 79, 158 ↵
    57, 145 57, 112 53, 111 47, 93 43, 90 48, 88 40, 80 39, 68 32, 51 33, 40 31, 39 34, ↵
    49 38, 34 38, 25 34, 28 39, 36 40, 44 46, 24 41, 17 41, 14 46, 19 50, 33 54, 21 55, 13 ↵
    52, 11 57, 22 60, 34 59, 41 68, 75 72, 62 77, 56 70, 46 72, 31 69, 46 76, 52 82, 47 ↵
    84, 56 90, 66 90, 64 94, 56 91, 33 97, 36 100, 23 100, 22 107, 29 106, 31 112, 46 116, ↵
    36 118, 28 131, 53 132, 59 127, 62 131, 76 130, 80 135, 89 137, 87 143, 73 145, 80 ↵
    150, 88 150, 85 157, 99 162, 116 158, 115 165, 123 165, 122 170, 134 164, 131 158))',
  0.3);
```



多边形的外壳

```
SELECT ST_SimplifyPolygonHull(
  'POLYGON ((131 158, 136 163, 161 165, 173 156, 179 148, 169 140, 186 144, 190 137, 185 ↵
    131, 174 128, 174 124, 166 119, 158 121, 158 115, 165 107, 161 97, 166 88, 166 79, 158 ↵
    57, 145 57, 112 53, 111 47, 93 43, 90 48, 88 40, 80 39, 68 32, 51 33, 40 31, 39 34, ↵
    49 38, 34 38, 25 34, 28 39, 36 40, 44 46, 24 41, 17 41, 14 46, 19 50, 33 54, 21 55, 13 ↵
    52, 11 57, 22 60, 34 59, 41 68, 75 72, 62 77, 56 70, 46 72, 31 69, 46 76, 52 82, 47 ↵
    84, 56 90, 66 90, 64 94, 56 91, 33 97, 36 100, 23 100, 22 107, 29 106, 31 112, 46 116, ↵
    36 118, 28 131, 53 132, 59 127, 62 131, 76 130, 80 135, 89 137, 87 143, 73 145, 80 ↵
    150, 88 150, 85 157, 99 162, 116 158, 115 165, 123 165, 122 170, 134 164, 131 158))',
  0.3, false);
```



多边形的外壳化，具有分段功能

```
SELECT ST_SimplifyPolygonHull(
  ST_Segmentize(ST_Letters('xt'), 2.0),
  0.1);
```

#### 相关信息

[ST\\_ConvexHull](#), [ST\\_SimplifyVW](#), [ST\\_ConcaveHull](#), [ST\\_Segmentize](#)

## 7.14.25 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

使用 **Visvalingam-Whyatt 算法** 返回几何图形的简化表示。简化的 **tolerance** 是一个面度，以输入 SRS 的位度为准。简化会移除与面度小于公差“拐角”形成的点。即使输入是有效的，结果可能也不是有效的。

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。



#### Note

该函数不保留多边形之间共享的边界。如果需要保留共享边界，请使用 **ST\_CoverageSimplify**。



#### Note

该函数处理 3D，第三维将影响结果。

可用性：2.2.0

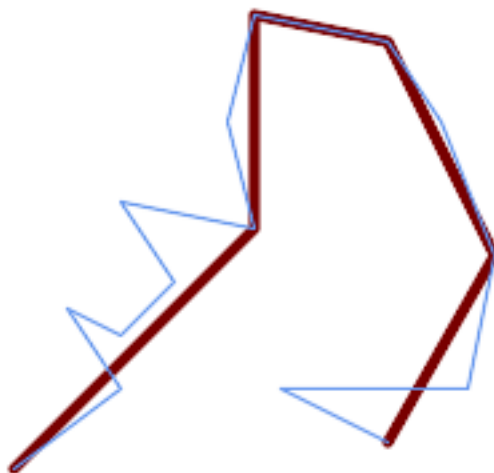
### 示例

使用最小面度公差为 30 的 LineString 行简化。

```
SELECT ST_AsText(ST_SimplifyVW(geom,30)) simplified
FROM (SELECT 'LINESTRING(5 2, 3 8, 6 20, 7 25, 10 10)::geometry AS geom) AS t;

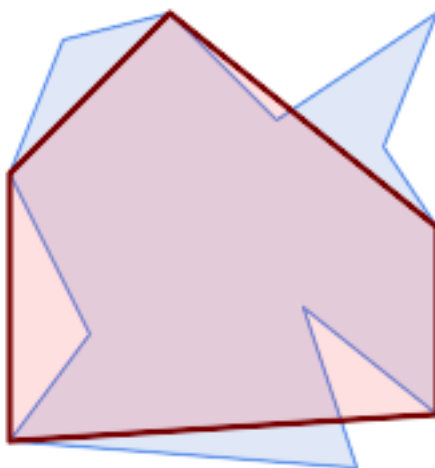
simplified
-----
LINESTRING(5 2,7 25,10 10)
```

一条线行简化。



```
SELECT ST_SimplifyVW(
  'LINESTRING (10 10, 50 40, 30 70, 50 60, 70 80, 50 110, 100 100, 90 140, 100 180, 150 170, 170 140, 190 90, 180 40, 110 40, 150 20)',
  1600);
```

多边形行化。



```
SELECT ST_SimplifyVW(
  'MULTIPOLYGON (((90 110, 80 180, 50 160, 10 170, 10 140, 20 110, 90 110)), ((40 80, 100 100, 120 160, 170 180, 190 70, 140 10, 110 40, 60 40, 40 80)), (180 70, 170 110, 142.5 128.5, 128.5 77.5, 90 60, 180 70)))',
  40);
```

相关信息

[ST\\_SetEffectiveArea](#), [ST\\_Simplify](#), [ST\\_SimplifyPreserveTopology](#), [ST\\_CoverageSimplify](#), [Topology ST\\_Simpl](#)

### 7.14.26 ST\_SetEffectiveArea

[ST\\_SetEffectiveArea](#) — 使用 Visvalingam-Whyatt 算法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。有效区域存储点的  $M$ 。如果使用可选项的“threshold”参数，将返回简化的几何图形，包含有效区域大于或等于阈值的点。

当指定阈值，此函数可用于服务器端简化。一种选择是使用零值。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。

几何体上只会对（多）边形和（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素的基座上行的，因此您可以将 `GeometryCollection` 提供此函数。

Note!

#### Note

注意，返回的几何图形可能会失去其简单性（参见 `ST_IsSimple`）

Note!

#### 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（参见 `ST_SimplifyPreserveTopology`）保留拓扑。

Note!

#### Note

输出几何图形将丢失  $M$  中的所有先前信息

Note!

#### Note

此功能处理 3D，三轴度会影响有效区域

可用性：2.2.0

### 示例

计算 `LineString` 的有效区域。因为我使用零值，所以返回输入几何体中的所有点。

```
select ST_AsText(ST_SetEffectiveArea(geom)) all_pts, ST_AsText(ST_SetEffectiveArea(geom,30) ←
) thrshld_30
FROM (SELECT 'LINESTRING(5 2, 3 8, 6 20, 7 25, 10 10)::geometry geom) As foo;
-result
all_pts | thrshld_30
-----+-----
LINESTRING M (5 2 3.40282346638529e+38,3 8 29,6 20 1.5,7 25 49.5,10 10 3.40282346638529e ←
+38) | LINESTRING M (5 2 3.40282346638529e+38,7 25 49.5,10 10 3.40282346638529e+38)
```

相关信息

[ST\\_SimplifyVW](#)

## 7.14.27 ST\_TriangulatePolygon

ST\_TriangulatePolygon — 计算多边形的约束 Delaunay 三角剖分

### Synopsis

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

### 描述

计算多边形的约束 Delaunay 三角剖分。支持孔和多边形。

多边形的“约束 Delaunay 三角剖分”是由多边形的点形成的三角形，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。它提供了多边形的“最佳质量”三角剖分。

可用性：3.3.0。

需要 GEOS >= 3.11.0。

### 示例

正方形的三角剖分。

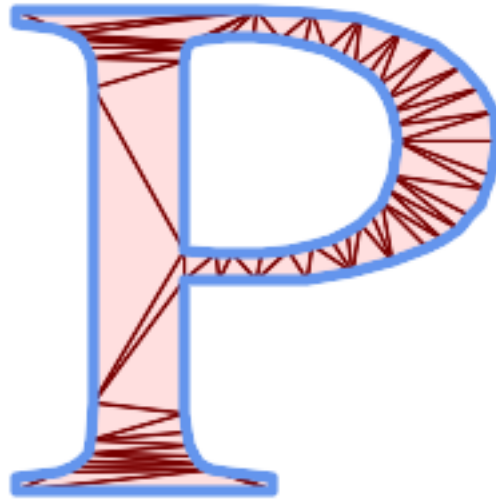
```
SELECT ST_AsText(
  ST_TriangulatePolygon('POLYGON((0 0, 0 1, 1 1, 1 0, 0 0))');

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

### 示例

字母 P 的三角剖分。

```
SELECT ST_AsText(ST_TriangulatePolygon(
  'POLYGON ((26 17, 31 19, 34 21, 37 24, 38 29, 39 43, 39 161, 38 172, 36 176, 34 179, 30 ←
    181, 25 183, 10 185, 10 190, 100 190, 121 189, 139 187, 154 182, 167 177, 177 169, ←
    184 161, 189 152, 190 141, 188 128, 186 123, 184 117, 180 113, 176 108, 170 104, 164 ←
    101, 151 96, 136 92, 119 89, 100 89, 86 89, 73 89, 73 39, 74 32, 75 27, 77 23, 79 ←
    20, 83 18, 89 17, 106 15, 106 10, 10 10, 10 15, 26 17), (152 147, 151 152, 149 157, ←
    146 162, 142 166, 137 169, 132 172, 126 175, 118 177, 109 179, 99 180, 89 180, 80 ←
    179, 76 178, 74 176, 73 171, 73 100, 85 99, 91 99, 102 99, 112 100, 121 102, 128 ←
    104, 134 107, 139 110, 143 114, 147 118, 149 123, 151 128, 153 141, 152 147))'
));
```



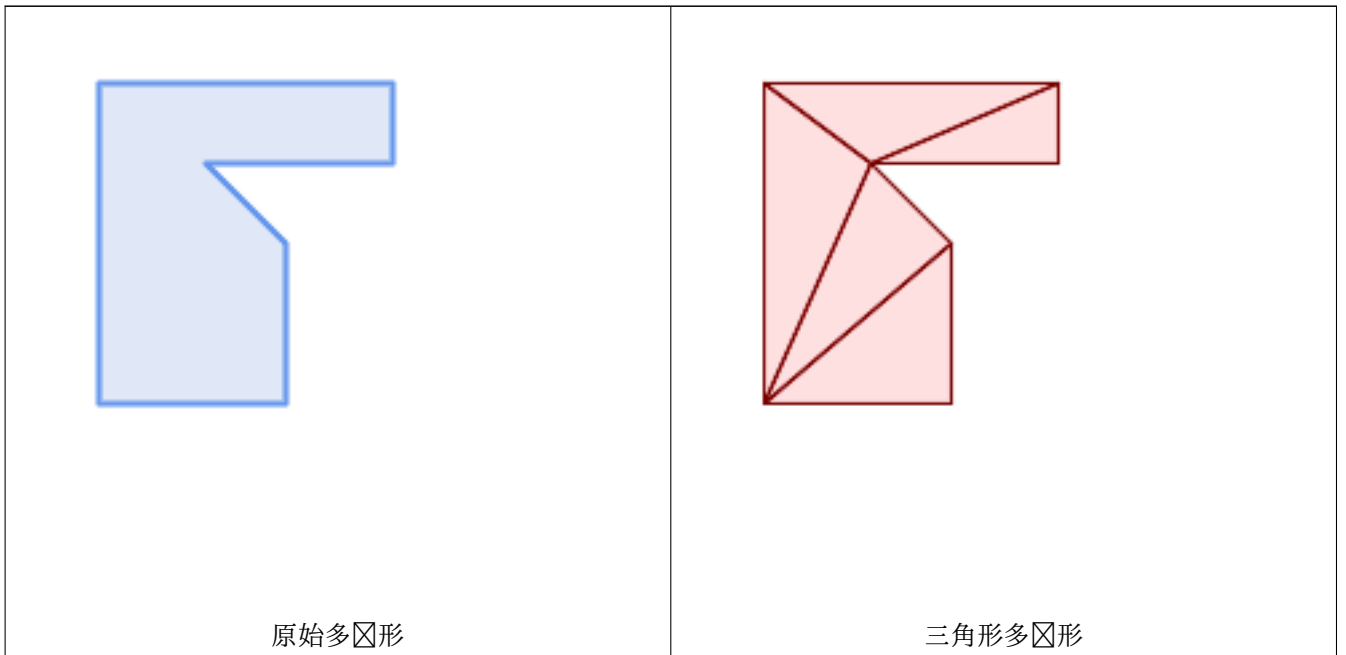
多边形三角剖分

同图例子，如 **ST\_Tessellate**

```
SELECT ST_TriangulatePolygon(
    'POLYGON (( 10 190, 10 70, 80 70, 80 130, 50 160, 120 160, 120 190, 10 190 ←
    ))'::geometry
);
```

ST\_AsText 输出

```
GEOMETRYCOLLECTION(POLYGON((50 160,120 190,120 160,50 160))
,POLYGON((10 70,80 130,80 70,10 70))
,POLYGON((50 160,10 70,10 190,50 160))
,POLYGON((120 190,50 160,10 190,120 190))
,POLYGON((80 130,10 70,50 160,80 130)))
```





相关信息

[ST\\_ConstrainedDelaunayTriangles](#), [ST\\_DelaunayTriangles](#), [ST\\_Tessellate](#)

### 7.14.28 ST\_VoronoiLines

ST\_VoronoiLines — 返回几何体点的 Voronoi 的界。

#### Synopsis

```
geometry ST_VoronoiLines( geometry geom , float8 tolerance = 0.0 , geometry extend_to = NULL );
```

#### 描述

根据提供的几何形的点计算 Voronoi 图，并将图中元格之的界作 MultiLineString 返回。如果输入几何形 null，返回 null。如果输入几何形包含一个点，返回一个空几何形集合。如果 extend\_to 最小外接矩形的面为零，返回空几何集合。

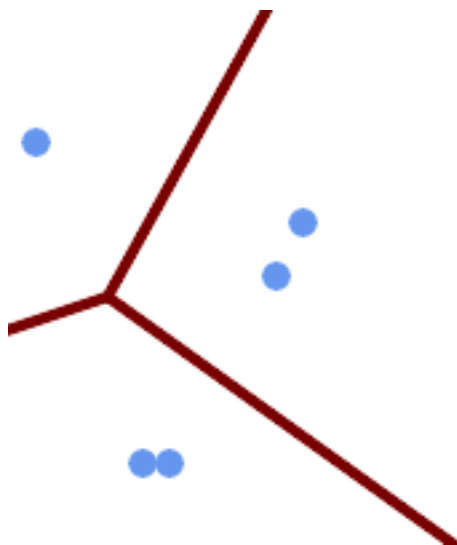
可参数：

- **tolerance** : 点被相等距离。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



Voronoi 图，容差 30 个位

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

ST\_AsText output

```
MULTILINESTRING((135.555555555556 270,36.8181818181818 92.2727272727273),(36.8181818181818 92.2727272727273,-110 43.3333333333333),(230 -45.7142857142858,36.8181818181818 92.2727272727273))
```

相关信息

[ST\\_DelaunayTriangles](#), [ST\\_VoronoiPolygons](#)

### 7.14.29 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 **Voronoi**。如果是 POLYGON 的 GEOMETRYCOLLECTION，其覆盖的包大于输入点的范围。如果输入几何形 null，返回 null。如果输入几何形包含一个点，返回一个空几何形集合。如果 extend\_to 最小外接矩形的面积为零，返回空几何集合。

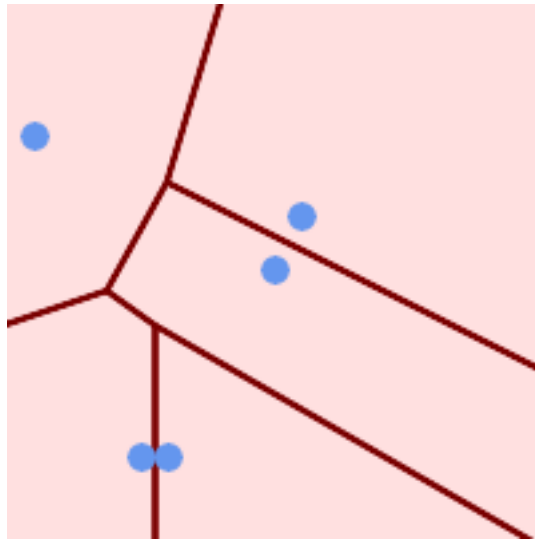
可参数：

- **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，默最小外接矩形是输入展 50% 的界框）。

个函数是由 GEOS 模行的。

可用性：2.3.0

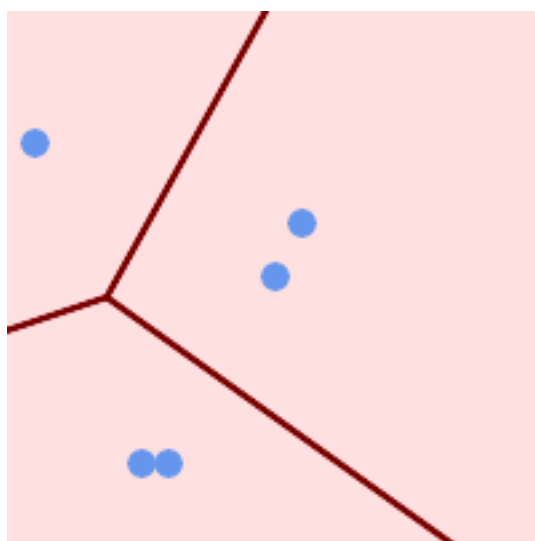
示例



覆盖在 Voronoi 部的点

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

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



Voronoi 部, 容差 30 个位

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

ST\_AsText output

```
GEOMETRYCOLLECTION(POLYGON((-110 43.3333333333333, -110 270,100.5 270,59.3478260869565 ←
    132.826086956522,36.8181818181818 92.2727272727273, -110 43.3333333333333)),
POLYGON((230 47.5,230 -45.7142857142858,36.8181818181818 92.2727272727273,59.3478260869565 ←
    132.826086956522,230 47.5)),POLYGON((230 -45.7142857142858,230 -90,-110 -90,-110 ←
    43.3333333333333,36.8181818181818 92.2727272727273,230 -45.7142857142858)),
POLYGON((100.5 270,230 270,230 47.5,59.3478260869565 132.826086956522,100.5 270)))
```

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。它返回布尔指示器，指示每个多边形中无效位置（如果有）的位置。

如果满足以下条件，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

作为窗口函数，每个输入多边形返回一个布尔值。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NULL的MULTILINESTRING。覆盖范围有效的面返回NULL。非多边形或空几何形状也会生成NULL。

只要周围的匹配多边形，这些条件就允许有效的覆盖范围包含孔（多边形之间的空隙）。然而，非常窄的空隙通常是不希望的。如果使用非零距离指定公差参数，形成窄空隙的形状也将被返回无效。

正在覆盖范围有效性的多边形也必须是有效的几何形状。可以使用[ST\\_IsValid](#)进行验证。

可用性：3.4.0

需要 GEOS >= 3.12.0



## 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（由多个多边形共享的边），请将 `SimplifyBoundary` 参数置为 `false`。



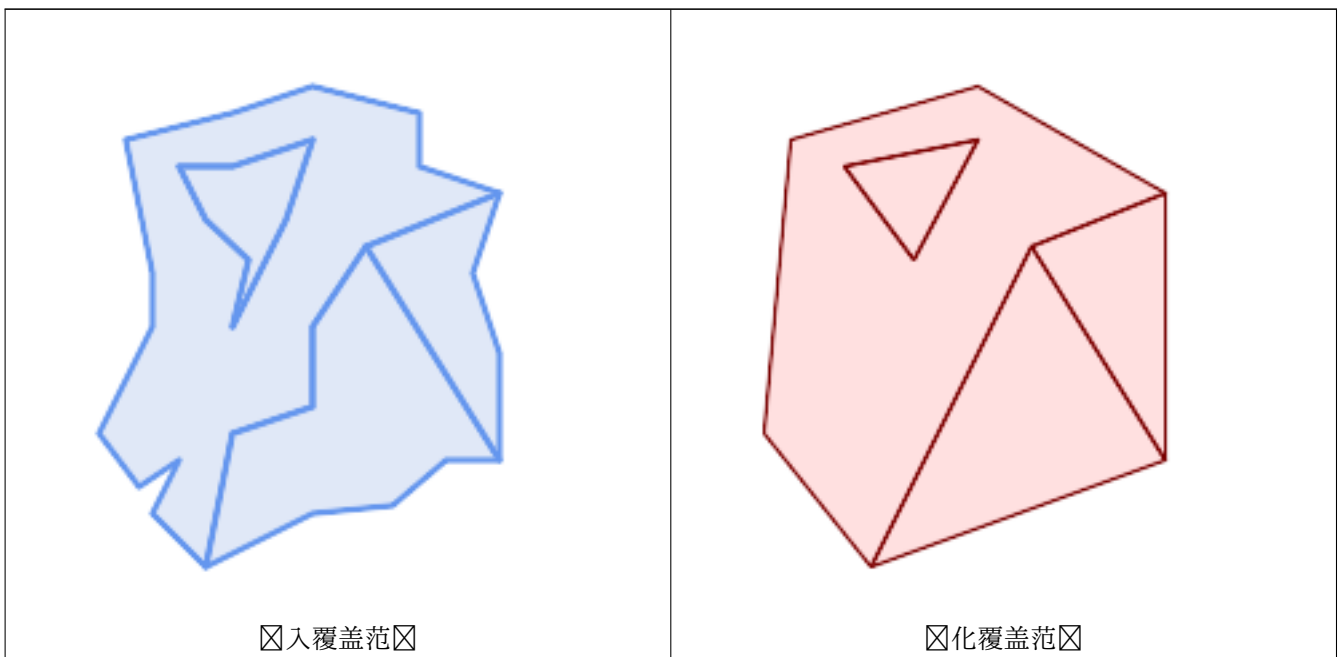
### Note

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



```
WITH coverage(id, geom) AS (VALUES
(1, 'POLYGON ((160 150, 110 130, 90 100, 90 70, 60 60, 50 10, 30 30, 40 50, 25 40, 10 60, 30 100, 30 120, 20 170, 60 180, 90 190, 130 180, 130 160, 160 150), (40 160, 50 140, 66 125, 60 100, 80 140, 90 170, 60 160, 40 160))'::geometry),
(2, 'POLYGON ((40 160, 60 160, 90 170, 80 140, 60 100, 66 125, 50 140, 40 160))'::geometry),
(3, 'POLYGON ((110 130, 160 50, 140 50, 120 33, 90 30, 50 10, 60 60, 90 70, 90 100, 110 130))'::geometry),
```

```
(4, 'POLYGON (((160 150, 150 120, 160 90, 160 50, 110 130, 160 150))'::geometry)
)
SELECT id, ST_AsText(ST_CoverageSimplify(geom, 30) OVER ())
FROM coverage;
```

id	st_astext
1	POLYGON ((160 150, 110 130, 50 10, 10 60, 20 170, 90 190, 160 150), (40 160, 66 125, 90 170, 40 160))
2	POLYGON ((40 160, 66 125, 90 170, 40 160))
3	POLYGON ((110 130, 160 50, 50 10, 110 130))
4	POLYGON ((160 150, 160 50, 110 130, 160 150))

相关信息

[ST\\_CoverageInvalidEdges](#)

### 7.15.3 ST\_CoverageUnion

`ST_CoverageUnion` — 通过去除共享来估算形成覆盖范围的一多形的并集。

#### Synopsis

```
geometry ST_CoverageUnion(geometry set geom);
```

#### 描述

将一多形合起来形成多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形几何体。函数生成与 `ST_Union` 相同的结果，但使用覆盖来更快地估算并集。

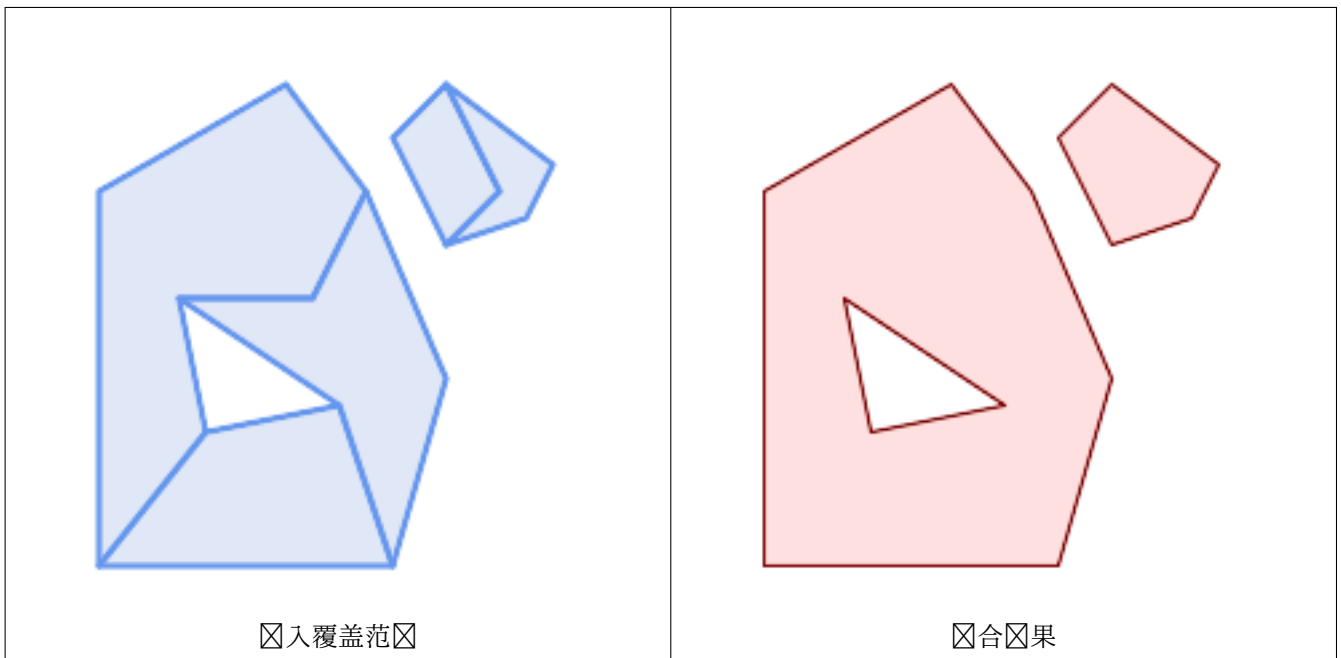


#### Note

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



```
WITH coverage(id, geom) AS (VALUES
  (1, 'POLYGON ((10 10, 10 150, 80 190, 110 150, 90 110, 40 110, 50 60, 10 10))'::geometry) ←
  (2, 'POLYGON ((120 10, 10 10, 50 60, 100 70, 120 10))'::geometry),
  (3, 'POLYGON ((140 80, 120 10, 100 70, 40 110, 90 110, 110 150, 140 80))'::geometry),
  (4, 'POLYGON ((140 190, 120 170, 140 130, 160 150, 140 190))'::geometry),
  (5, 'POLYGON ((180 160, 170 140, 140 130, 160 150, 140 190, 180 160))'::geometry)
)
SELECT ST_AsText(ST_CoverageUnion(geom))
FROM coverage;

-----
MULTIPOLYGON (((10 150, 80 190, 110 150, 140 80, 120 10, 10 10, 10 150), (50 60, 100 70, 40 ←
  110, 50 60)), ((120 170, 140 190, 180 160, 170 140, 140 130, 120 170)))
```

相关信息

[ST\\_CoverageInvalidEdges](#), [ST\\_Union](#)

## 7.16 仿射

### 7.16.1 ST\_Affine

`ST_Affine` — 几何体用 3D 仿射。

#### 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

几何体用 3D 仿射，一步完成平移、旋转、缩放等操作。

版本 1：用

```
ST_Affine(geom, a, b, c, d, e, f, g, h, i, xoff, yoff, zoff)
```

表示矩阵

```
/ a b c xoff \
| d e f yoff |
| g h i zoff |
\ 0 0 0 1 /
```

并且点如下：

```
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/缩放函数都是通用仿射的。

版本 2：几何体用 2d 仿射。用

```
ST_Affine(geom, a, b, d, e, xoff, yoff)
```

表示矩阵

```
/ 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 /
```

并且点如下：

```
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 更改为 ST\_Affine



### Note

在 1.3.4 之前，此函数在与包含曲面的几何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



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面。

## 示例

```
--Rotate a 3d line 180 degrees about the z axis. Note this is long-hand for doing ↵
ST_Rotate();
SELECT ST_AsEWKT(ST_Affine(geom, cos(pi()), -sin(pi()), 0, sin(pi()), cos(pi()), 0, 0, ↵
0, 1, 0, 0, 0)) As using_affine,
ST_AsEWKT(ST_Rotate(geom, pi())) As using_rotate
FROM (SELECT ST_GeomFromEWKT('LINESTRING(1 2 3, 1 4 3)') As geom) As foo;
using_affine | using_rotate
-----+-----
LINESTRING(-1 -2 3,-1 -4 3) | LINESTRING(-1 -2 3,-1 -4 3)
(1 row)

--Rotate a 3d line 180 degrees in both the x and z axis
SELECT ST_AsEWKT(ST_Affine(geom, cos(pi()), -sin(pi()), 0, sin(pi()), cos(pi()), -sin(pi()) ↵
, 0, sin(pi()), cos(pi()), 0, 0, 0))
FROM (SELECT ST_GeomFromEWKT('LINESTRING(1 2 3, 1 4 3)') As geom) As foo;
st_asewkt
-----
LINESTRING(-1 -2 -3,-1 -4 -3)
(1 row)
```

## 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 POINT 几何形，也可以指定 `x` 和 `y` 坐标。如果未指定原点，几何形将绕 POINT(0 0) 旋转。

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 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。



此方法支持形字符串和曲面。



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

```
--Rotate 180 degrees
SELECT ST_AsEWKT(ST_Rotate('LINESTRING (50 160, 50 50, 100 50)', pi()));
           st_asewkt
-----
LINESTRING(-50 -160,-50 -50,-100 -50)
(1 row)

--Rotate 30 degrees counter-clockwise at x=50, y=160
SELECT ST_AsEWKT(ST_Rotate('LINESTRING (50 160, 50 50, 100 50)', pi()/6, 50, 160));
           st_asewkt
-----
LINESTRING(50 160,105 64.7372055837117,148.301270189222 89.7372055837117)
(1 row)

--Rotate 60 degrees clockwise from centroid
SELECT ST_AsEWKT(ST_Rotate(geom, -pi()/3, ST_Centroid(geom)))
FROM (SELECT 'LINESTRING (50 160, 50 50, 100 50)::geometry AS geom) AS foo;
           st_asewkt
-----
LINESTRING(116.4225 130.6721,21.1597 75.6721,46.1597 32.3708)
(1 row)
```

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

```
--Rotate a line 90 degrees along x-axis
SELECT ST_AsEWKT(ST_RotateX(ST_GeomFromEWKT('LINESTRING(1 2 3, 1 1 1)'), pi()/2));
           st_asewkt
-----
LINESTRING(1 -3 2,1 -1 1)
```

## 相关信息

[ST\\_Affine](#), [ST\\_RotateY](#), [ST\\_RotateZ](#)

### 7.16.4 ST\_RotateY

ST\_RotateY — 沿 Y 轴旋转几何体。

#### Synopsis

geometry **ST\_RotateY**(geometry geomA, float rotRadians);

## 描述

沿 y 轴旋转几何体 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) 的缩写。

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

```
--Rotate a line 90 degrees along y-axis
SELECT ST_AsEWKT(ST_RotateY(ST_GeomFromEWKT('LINESTRING(1 2 3, 1 1 1)'), pi()/2));
           st_asewkt
-----
LINESTRING(3 2 -1,1 1 -1)
```

## 相关信息

[ST\\_Affine](#), [ST\\_RotateX](#), [ST\\_RotateZ](#)

## 7.16.5 ST\_RotateZ

ST\_RotateZ — Z 轴旋转几何体。

### Synopsis

geometry **ST\_RotateZ**(geometry geomA, float rotRadians);

### 描述

Z 轴旋转几何体 geomA - rotRadians。



#### Note

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) 的缩写。

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：1.1.2。1.2.2 中名称从 RotateZ 更改为 ST\_RotateZ



#### Note

在 1.3.4 之前，此函数在与包含曲面的几何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。

- ✓ 函数支持 3d 并且不会丢失 z-index。
- ✓ 此方法支持形字符串和曲面。
- ✓ 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

```
--Rotate a line 90 degrees along z-axis
SELECT ST_AsEWKT(ST_RotateZ(ST_GeomFromEWKT('LINESTRING(1 2 3, 1 1 1)'), pi()/2));
           st_asewkt
-----
LINESTRING(-2 1 3,-1 1 1)

--Rotate a curved circle around z-axis
SELECT ST_AsEWKT(ST_RotateZ(geom, pi()/2))
FROM (SELECT ST_LineToCurve(ST_Buffer(ST_GeomFromText('POINT(234 567)'), 3)) As geom) As
      foo;
```

```
CURVEPOLYGON(CIRCULARSTRING(-567 237, -564.87867965644 236.12132034356, -564 234, -569.12132034356 231.87867965644, -567 237))
```

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

通过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，此函数在与包含曲的几何形一起使用崩。此已在 1.3.4 及更高版本中得到正

可用性：1.1.0。

增功能：引入了 2.0.0 多面体曲面、三角形和三角网的支持。

增：2.2.0 引入了放所有度（**factor** 参数）的支持。

增：2.5.0 引入了相于本地原点（**origin** 参数）行放的支持。

- 函数支持多面体曲面。
- 函数支持 3d 并且不会失 z-index。
- 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)。
- 功能支持 M 坐。

示例

```
--Version 1: scale X, Y, Z
SELECT ST_AsEWKT(ST_Scale(ST_GeomFromEWKT('LINESTRING(1 2 3, 1 1 1)'), 0.5, 0.75, 0.8));
           st_asewkt
-----
LINESTRING(0.5 1.5 2.4,0.5 0.75 0.8)

--Version 2: Scale X Y
SELECT ST_AsEWKT(ST_Scale(ST_GeomFromEWKT('LINESTRING(1 2 3, 1 1 1)'), 0.5, 0.75));
           st_asewkt
-----
LINESTRING(0.5 1.5 3,0.5 0.75 1)

--Version 3: Scale X Y Z M
SELECT ST_AsEWKT(ST_Scale(ST_GeomFromEWKT('LINESTRING(1 2 3 4, 1 1 1 1)'),
  ST_MakePoint(0.5, 0.75, 2, -1)));
           st_asewkt
-----
LINESTRING(0.5 1.5 6 -4,0.5 0.75 2 -1)

--Version 4: Scale X Y using false origin
SELECT ST_AsText(ST_Scale('LINESTRING(1 1, 2 2)', 'POINT(2 2)', 'POINT(1 1)::geometry'));
           st_astext
-----
LINESTRING(1 1,3 3)
```

相关信息

[ST\\_Affine](#), [ST\\_TransScale](#)

## 7.16.7 ST\_Translate

ST\_Translate — 按给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

返回一个新的几何图形，其坐标为  $\delta x$ 、 $\delta y$ 、 $\delta z$  位。位基于几何的空参考 (SRID) 中定义的位。



#### 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 并且不会失 z-index。
- ✔ 此方法支持形字符串和曲线。

示例

将点移 1 度

```
SELECT ST_AsText(ST_Translate(ST_GeomFromText('POINT(-71.01 42.37)',4326),1,0)) As
      wgs_transgeomtxt;

      wgs_transgeomtxt
      -----
      POINT(-70.01 42.37)
```

将串移 1 度和 1/2 度

```
SELECT ST_AsText(ST_Translate(ST_GeomFromText('LINESTRING(-71.01 42.37,-71.11 42.38)',4326)
      ,1,0.5)) As wgs_transgeomtxt;
              wgs_transgeomtxt
              -----
      LINESTRING(-70.01 42.87,-70.11 42.88)
```

移 3d 点

```
SELECT ST_AsEWKT(ST_Translate(CAST('POINT(0 0 0)' As geometry), 5, 12,3));
      st_asewkt
      -----
      POINT(5 12 3)
```

移曲线和点

```
SELECT ST_AsText(ST_Translate(ST_Collect('CURVEPOLYGON(CIRCULARSTRING(4 3,3.12 0.878,1
      0,-1.121 5.1213,6 7, 8 9,4 3))','POINT(1 3)'),1,2));

-----

      GEOMETRYCOLLECTION(CURVEPOLYGON(CIRCULARSTRING(5 5,4.12 2.878,2 2,-0.121 7.1213,7 9,9 11,5
      5)),POINT(2 5))
```

相关信息

[ST\\_Affine](#), [ST\\_AsText](#), [ST\\_GeomFromText](#)

## 7.16.8 ST\_TransScale

ST\_TransScale — 按定的偏移量和系数平移和放几何形。

### 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)` 的缩写。

**Note**

在 1.3.4 之前，此函数在与包含曲线的几何体一起使用时会崩溃。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。



此方法支持几何体字符串和曲线。

## 示例

```
SELECT ST_AsEWKT(ST_TransScale(ST_GeomFromEWKT('LINESTRING(1 2 3, 1 1 1)'), 0.5, 1, 1, 2));
-----
st_asewkt
```

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

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

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

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

## 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体的簇 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”几何形状，至少有 **minpoints** 个几何形状（包括它自己）；或
- “边界”几何形状，位于核心几何形状的 **eps 距离** 内。

注意，边界几何形状可能位于多个集群中核心几何形状的 **eps** 距离内。任一分配都是正确的，因此边界几何形状将被任意分配可用集群之一。在这种情况下，可以使用少于 **minpoints** 几何形状生成正确的簇。为了确保边界几何形状的不确定性分配（以便重复使用 **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

不满足加入任何簇的条件几何形状将被分配簇号 NULL。

可用性：2.3.0



此方法支持几何字符串和曲线。

### 示例

彼此相距 50 米以内的多边形行聚类，并且每个聚类至少需要 2 个多边形。

50 米范围内的聚簇，每个聚簇至少有 5 个面。如果只有一个，cid 为 NULL

```

SELECT name, ST_ClusterDBSCAN(geom, eps = 50, minpoints = 5) over () AS cid
FROM boston_polys
WHERE name
> '' AND building
> ''
      AND ST_DWithin(geom,
      ST_Transform(
        ST_GeomFromText('POINT (-71.04054 42.35141)', 4326), 26986),
        500);
    
```

bucket	name	
0	Manulife Tower	↔
0	Park Lane Seaport I	↔
0	Park Lane Seaport II	↔
0	Renaissance Boston Waterfront Hotel	↔
0	Seaport Boston Hotel	↔
0	Seaport Hotel & World Trade Center	↔
0	Waterside Place	↔
0	World Trade Center East	↔
1	100 Northern Avenue	↔
1	100 Pier 4	↔
1	The Institute of Contemporary Art	↔
2	101 Seaport	↔
2	District Hall	↔
2	One Marina Park Drive	↔
2	Twenty Two Liberty	↔
2	Vertex	↔
2	Vertex	↔
2	Watermark Seaport	↔
2	Blue Hills Bank Pavilion	↔
NULL	World Trade Center West	↔
NULL	(20 rows)	

示将具有相同簇号的地块合成几何集合的示例。

```

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 — 将输入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，将输入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

```
WITH testdata AS
  (SELECT unnest(ARRAY['LINESTRING (0 0, 1 1)::geometry',
    'LINESTRING (5 5, 4 4)::geometry',
    'LINESTRING (6 6, 7 7)::geometry',
    'LINESTRING (0 0, -1 -1)::geometry',
    'POLYGON ((0 0, 4 0, 4 4, 0 4, 0 0))::geometry']) AS geom)

SELECT ST_AsText(unnest(ST_ClusterIntersecting(geom))) FROM testdata;

-- result

st_astext
-----
GEOMETRYCOLLECTION(LINESTRING(0 0,1 1),LINESTRING(5 5,4 4),LINESTRING(0 0,-1 -1),POLYGON((0 ←
  0,4 0,4 4,0 4,0 0)))
GEOMETRYCOLLECTION(LINESTRING(6 6,7 7))
```

### 相关信息

[ST\\_ClusterIntersectingWin](#), [ST\\_ClusterWithin](#), [ST\\_ClusterWithinWin](#)

## 7.17.3 ST\_ClusterIntersectingWin

ST\_ClusterIntersectingWin — 窗口函数，返回每个输入几何形的簇 ID，将输入几何形聚到交集的集合中。

### Synopsis

```
integer ST_ClusterIntersectingWin(geometry winset geom);
```

### 描述

一种窗口函数，用于构建相交的交集几何形簇。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，或者空输入空。

可用性：3.4.0

示例

```
WITH testdata AS (
  SELECT id, geom::geometry FROM (
    VALUES (1, 'LINESTRING (0 0, 1 1)'),
           (2, 'LINESTRING (5 5, 4 4)'),
           (3, 'LINESTRING (6 6, 7 7)'),
           (4, 'LINESTRING (0 0, -1 -1)'),
           (5, 'POLYGON ((0 0, 4 0, 4 4, 0 4, 0 0))')) AS t(id, geom)
)
SELECT id,
       ST_AsText(geom),
       ST_ClusterIntersectingWin(geom) OVER () AS cluster
FROM testdata;
```

id	st_astext	cluster
1	LINESTRING(0 0,1 1)	0
2	LINESTRING(5 5,4 4)	0
3	LINESTRING(6 6,7 7)	1
4	LINESTRING(0 0,-1 -1)	0
5	POLYGON((0 0,4 0,4 4,0 4,0 0))	0

相关信息

[ST\\_ClusterIntersecting](#), [ST\\_ClusterWithin](#), [ST\\_ClusterWithinWin](#)

### 7.17.4 ST\_ClusterKMeans

`ST_ClusterKMeans` — 使用 K 均值算法返回每个输入几何形的簇 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的 **K-means** 簇号。用于聚类的距离是 2D 几何形状的中心之间的距离，以及 3D 几何形状的边界框中心之间的距离。对于 POINT 输入，M 坐标将被忽略输入的重，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`。这在可访问性分析中很有用。

增量：3.2.0 支持 `max_radius`

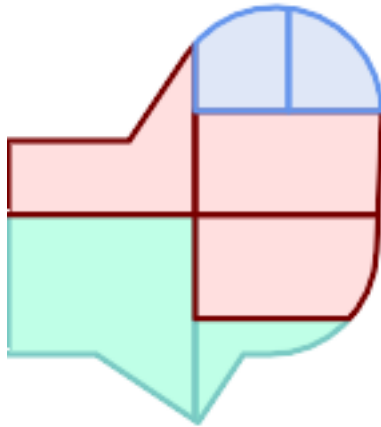
增量：3.1.0 支持 3D 几何和权重

可用性：2.3.0

示例

生成虚拟数据集，例如：

```
CREATE TABLE parcels AS
SELECT lpad((row_number() over())::text,3,'0') As parcel_id, geom,
('{residential, commercial}'::text[][1 + mod(row_number()OVER(),2)]) As type
FROM
  ST_Subdivide(ST_Buffer('SRID=3857;LINESTRING(40 100, 98 100, 100 150, 60 90)'::geometry ←
  40, 'endcap=square'),12) As geom;
```



地按簇号 (cid) 行色

```
SELECT ST_ClusterKMeans(geom, 3) OVER() AS cid, parcel_id, geom
FROM parcels;
```

cid	parcel_id	geom
0	001	0103000000...
0	002	0103000000...
1	003	0103000000...
0	004	0103000000...
1	005	0103000000...
2	006	0103000000...
2	007	0103000000...

按型划分地集群：

```
SELECT ST_ClusterKMeans(geom, 3) over (PARTITION BY type) AS cid, parcel_id, type
FROM parcels;
```

cid	parcel_id	type
1	005	commercial
1	003	commercial
2	007	commercial
0	001	commercial
1	004	residential
0	002	residential
2	006	residential

示例：使用 3D 聚和加先聚合的全球人口数据集行聚。根据 [Kontur 人口数据](#)，至少确定了距离中心不到 3000 公里的 20 个地区：

```

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

```



根据上述范围世界人口进行聚类，得到 46 个聚类。在人口稠密的地区（欧洲，莫斯科）是集群的中心。格陵兰是一个集群。有些集群跨越国日期更。聚类的遵循地球的曲线。

#### 相关信息

[ST\\_ClusterDBSCAN](#), [ST\\_ClusterIntersectingWin](#), [ST\\_ClusterWithinWin](#), [ST\\_ClusterIntersecting](#), [ST\\_ClusterSubdivide](#), [ST\\_Force3D](#), [ST\\_Force4D](#),

### 7.17.5 ST\_ClusterWithin

`ST_ClusterWithin` — 按间隔距离几何形进行聚合的聚合函数。

#### 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的簇。聚类将输入几何形划分多个集合，其中每个几何形都在同一簇中至少一个其他几何形的指定距离内。距离是以 SRID 位的笛卡儿距离。

`ST_ClusterWithin` 相当于进行 `ST_ClusterDBSCAN` 使用 `minpoints => 0`。

可用性：2.2.0



此方法支持形字符串和曲线。

示例

```
WITH testdata AS
  (SELECT unnest(ARRAY[ 'LINESTRING (0 0, 1 1)::geometry,
                       'LINESTRING (5 5, 4 4)::geometry,
                       'LINESTRING (6 6, 7 7)::geometry,
                       'LINESTRING (0 0, -1 -1)::geometry,
                       'POLYGON ((0 0, 4 0, 4 4, 0 4, 0 0))'::geometry]) AS geom)

SELECT ST_AsText(unnest(ST_ClusterWithin(geom, 1.4))) FROM testdata;

-- result

st_astext
-----
GEOMETRYCOLLECTION(LINESTRING(0 0,1 1),LINESTRING(5 5,4 4),LINESTRING(0 0,-1 -1),POLYGON((0 0,4 0,4 4,0 4,0 0)))
GEOMETRYCOLLECTION(LINESTRING(6 6,7 7))
```

相关信息

[ST\\_ClusterWithinWin](#), [ST\\_ClusterDBSCAN](#), [ST\\_ClusterIntersecting](#), [ST\\_ClusterIntersectingWin](#)

### 7.17.6 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类。

#### Synopsis

integer **ST\_ClusterWithinWin**(geometry winset geom, float8 distance);

描述

返回每个输入几何形的簇号的窗口函数。聚类将几何形划分多个集合，其中每个几何形都在同一簇中至少一个其他几何形的指定距离内。距离是以 SRID 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



此方法支持几何字符串和曲线。

示例

```
WITH testdata AS (
  SELECT id, geom::geometry FROM (
    VALUES (1, 'LINESTRING (0 0, 1 1)'),
           (2, 'LINESTRING (5 5, 4 4)'),
           (3, 'LINESTRING (6 6, 7 7)'),
           (4, 'LINESTRING (0 0, -1 -1)'),
           (5, 'POLYGON ((0 0, 4 0, 4 4, 0 4, 0 0))')) AS t(id, geom)
)
SELECT id,
       ST_AsText(geom),
```



```
ST_ClusterWithinWin(geom, 1.4) OVER () AS cluster
FROM testdata;
```

id	st_astext	cluster
1	LINestring(0 0,1 1)	0
2	LINestring(5 5,4 4)	0
3	LINestring(6 6,7 7)	1
4	LINestring(0 0,-1 -1)	0
5	POLYGON((0 0,4 0,4 4,0 4,0 0))	0

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 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面。
-  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

```
SELECT Box2D(ST_GeomFromText('LINestring(1 2, 3 4, 5 6)'));
```

```
box2d
-----
BOX(1 2,5 6)
```

```
SELECT Box2D(ST_GeomFromText('CIRCULARSTRING(220268 150415,220227 150505,220227 150406)'));
```

```
box2d
-----
BOX(220186.984375 150406,220288.25 150506.140625)
```

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。

- ✓ 此方法支持几何字符串和曲线。
- ✓ 几何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)。
- ✓ 几何函数支持 3d 并且不会丢失 z-index。

### 示例

```
SELECT Box3D(ST_GeomFromEWKT('LINESTRING(1 2 3, 3 4 5, 5 6 5)'));
```

```
Box3d
-----
BOX3D(1 2 3,5 6 5)
```

```
SELECT Box3D(ST_GeomFromEWKT('CIRCULARSTRING(220268 150415 1,220227 150505 1,220227 150406 1)')); ←
```

```
Box3d
-----
BOX3D(220227 150406 1,220268 150415 1)
```

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）收集的几何信息（如果可用）。如果 `parent_only` 置为 `TRUE`，使用指定表的几何信息并忽略子表。

于 PostgreSQL  $\geq 8.0.0$ ，几何信息由 `VACUUM ANALYZE` 收集，范围是几何范围的 95%。于 PostgreSQL  $< 8.0.0$  几何信息是通过行 `update_geometry_stats()` 收集的，范围是准确的。



#### Note

如果没有几何信息（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

Escaping names for tables and/or namespaces that include special characters and quotes may require special handling. A user notes: "For schemas and tables, use identifier escaping rules to produce a double-quoted string, and afterwards remove the first and last double-quote character. For geometry column pass as is."

可用性：1.0.0

更改：2.1.0。在 2.0.x 之前，称 `ST_Estimated_Extent`。



此方法支持几何字符串和曲线。

### 示例

```
SELECT ST_EstimatedExtent('ny', 'edges', 'geom');
--result--
BOX(-8877653 4912316, -8010225.5 5589284)

SELECT ST_EstimatedExtent('feature_poly', 'geom');
--result--
BOX(-124.659652709961 24.6830825805664, -67.7798080444336 49.0012092590332)
```

### 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形状展出的边界框。

## 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展的边界框，通过指定框在个框上展的个距离，或通过指定每个框的展距离。使用双精度。可用于距离框，或向框添加边界框器以利用空索引。

除了接受和返回几何形的 `ST_Expand` 版本之外，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展几何形，而 `ST_Expand` 沿每个框展边界框。



### Note

在版本 1.3 之前，`ST_Expand` 与 `ST_Distance` 组合使用来行可索引距离框。例如，`geom && ST_Expand('POINT(10 20)', 10) AND ST_Distance(geom, 'POINT(10 20)') < 10`。它已被更框、更高效的 `ST_DWithin` 函数取代。

可用性：1.5.0 行更改框出双精度而不是 `float4` 坐框。

增框功能：引入了 2.0.0 框多面体曲面、三角形和三角网的支持。

增框：2.3.0 添加了框不同框度的盒子框行不同数量框展的支持。



框函数支持多面体曲面。



此函数支持三角形和框框三角网面 (TIN)。

## 示例



### Note

以下示例使用美国国家地框平等面框投影 (SRID=2163)，框是一个以米框框位的投影

```
--10 meter expanded box around bbox of a linestring
SELECT CAST(ST_Expand(ST_GeomFromText('LINESTRING(2312980 110676,2312923 110701,2312892
110714)', 2163),10) As box2d);
                                st_expand
-----
BOX(2312882 110666,2312990 110724)

--10 meter expanded 3D box of a 3D box
SELECT ST_Expand(CAST('BOX3D(778783 2951741 1,794875 2970042.61545891 10)' As box3d),10)
                                st_expand
-----
BOX3D(778773 2951731 -9,794885 2970052.61545891 20)
```

```
--10 meter geometry astext rep of a expand box around a point geometry
SELECT ST_AsEWKT(ST_Expand(ST_GeomFromEWKT('SRID=2163;POINT(2312980 110676)'),10));
-----
SRID=2163;POLYGON((2312970 110666,2312970 110686,2312990 110686,2312990 110666,2312970 110666))
```

相关信息

[ST\\_Buffer](#), [ST\\_DWithin](#), [ST\\_SRID](#)

## 7.18.5 ST\_Extent

`ST_Extent` — 返回几何图形的边界框的聚合函数。

### 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

`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

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 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。



此函数支持三角形和不规则三角网面 (TIN)。

示例



### Note

以下示例使用以英尺为单位的塞州平面 (SRID=2249)

```
SELECT ST_Extent(geom) as bextent FROM sometable;
                st_bextent
-----
BOX(739651.875 2908247.25,794875.8125 2970042.75)

--Return extent of each category of geometries
SELECT ST_Extent(geom) as bextent
FROM sometable
GROUP BY category ORDER BY category;

                bextent | name
-----+-----
BOX(778783.5625 2951741.25,794875.8125 2970042.75) | A
BOX(751315.8125 2919164.75,765202.6875 2935417.25) | B
BOX(739651.875 2917394.75,756688.375 2935866)      | C

--Force back into a geometry
-- and render the extended text representation of that geometry
SELECT ST_SetSRID(ST_Extent(geom),2249) as bextent FROM sometable;

                bextent
-----
SRID=2249;POLYGON((739651.875 2908247.25,739651.875 2970042.75,794875.8125 2970042.75,
794875.8125 2908247.25,739651.875 2908247.25))
```

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 边框的聚合函数。

### Synopsis

```
box3d ST_3DExtent(geometry set geomfield);
```

描述

一个聚合函数，返回包含几何图形的 **box3d** (包括 Z 坐标) 边框。

边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` 将其具有 SRID 元数据的几何形。SRID 与入几何形状相同。

增功能：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 在之前的版本中，曾被称 `ST_Extent3D`



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**

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

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

**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 BOX2D。

**Synopsis**

`box2d` **ST\_MakeBox2D**(geometry pointLowLeft, geometry pointUpRight);

## 描述

由两个 Point 几何形定 box2d。用于行范很有用。

## 示例

```
--Return all features that fall reside or partly reside in a US national atlas coordinate ↔
  bounding box
--It is assumed here that the geometries are stored with SRID = 2163 (US National atlas ↔
  equal area)
SELECT feature_id, feature_name, geom
FROM features
WHERE geom && ST_SetSRID(ST_MakeBox2D(ST_Point(-989502.1875, 528439.5625),
  ST_Point(-987121.375 ,529933.1875)),2163)
```

## 相关信息

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

### 7.18.8 ST\_3DMakeBox

ST\_3DMakeBox — 由两个 3D 点几何形定 BOX3D。

#### Synopsis

box3d **ST\_3DMakeBox**(geometry point3DLowLeftBottom, geometry point3DUpRightTop);

## 描述

由两个 3D 点几何形定 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，曾被称 ST\_MakeBox3D

## 示例

```
SELECT ST_3DMakeBox(ST_MakePoint(-989502.1875, 528439.5625, 10),
  ST_MakePoint(-987121.375 ,529933.1875, 10)) As abb3d

--bb3d--
-----
BOX3D(-989502.1875 528439.5625 10,-987121.375 529933.1875 10)
```

## 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

虽然此函数接受 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

```
SELECT ST_XMax('BOX3D(1 2 3, 4 5 6)');
st_xmax
-----
4

SELECT ST_XMax(ST_GeomFromText('LINESTRING(1 3 4, 5 6 7)'));
st_xmax
-----
5

SELECT ST_XMax(CAST('BOX(-3 2, 3 4)' As box2d));
st_xmax
-----
3
--Observe THIS DOES NOT WORK because it will try to auto-cast the string representation to
a BOX3D
SELECT ST_XMax('LINESTRING(1 3, 5 6)');
--ERROR: BOX3D parser - doesn't start with BOX3D(

SELECT ST_XMax(ST_GeomFromEWKT('CIRCULARSTRING(220268 150415 1,220227 150505 2,220227
150406 3)'));
st_xmax
-----
220288.248780547
```

### 相关信息

[ST\\_XMin](#), [ST\\_YMax](#), [ST\\_YMin](#), [ST\\_ZMax](#), [ST\\_ZMin](#)

## 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

虽然此函数接受 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

```
SELECT ST_XMin('BOX3D(1 2 3, 4 5 6)');
st_xmin
-----
1

SELECT ST_XMin(ST_GeomFromText('LINESTRING(1 3 4, 5 6 7)'));
st_xmin
-----
1

SELECT ST_XMin(CAST('BOX(-3 2, 3 4)' As box2d));
st_xmin
-----
-3
--Observe THIS DOES NOT WORK because it will try to auto-cast the string representation to
a BOX3D
SELECT ST_XMin('LINESTRING(1 3, 5 6)');
--ERROR: BOX3D parser - doesn't start with BOX3D(

SELECT ST_XMin(ST_GeomFromEWKT('CIRCULARSTRING(220268 150415 1,220227 150505 2,220227
150406 3)'));
st_xmin
-----
220186.995121892
```

### 相关信息

[ST\\_XMax](#), [ST\\_YMax](#), [ST\\_YMin](#), [ST\\_ZMax](#), [ST\\_ZMin](#)

### 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

虽然此函数接受 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

```
SELECT ST_YMax('BOX3D(1 2 3, 4 5 6)');
st_ymax
-----
5

SELECT ST_YMax(ST_GeomFromText('LINESTRING(1 3 4, 5 6 7)'));
st_ymax
-----
6

SELECT ST_YMax(CAST('BOX(-3 2, 3 4)' As box2d));
st_ymax
-----
4
--Observe THIS DOES NOT WORK because it will try to auto-cast the string representation to
a BOX3D
SELECT ST_YMax('LINESTRING(1 3, 5 6)');
--ERROR: BOX3D parser - doesn't start with BOX3D(

SELECT ST_YMax(ST_GeomFromEWKT('CIRCULARSTRING(220268 150415 1,220227 150505 2,220227
150406 3)'));
st_ymax
-----
150506.126829327
```

#### 相关信息

[ST\\_XMin](#), [ST\\_XMax](#), [ST\\_YMin](#), [ST\\_ZMax](#), [ST\\_ZMin](#)

## 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

虽然此函数接受 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

```
SELECT ST_YMin('BOX3D(1 2 3, 4 5 6)');
st_ymin
-----
2

SELECT ST_YMin(ST_GeomFromText('LINESTRING(1 3 4, 5 6 7)'));
st_ymin
-----
3

SELECT ST_YMin(CAST('BOX(-3 2, 3 4)' As box2d));
st_ymin
-----
2
--Observe THIS DOES NOT WORK because it will try to auto-cast the string representation to
a BOX3D
SELECT ST_YMin('LINESTRING(1 3, 5 6)');
--ERROR: BOX3D parser - doesn't start with BOX3D(

SELECT ST_YMin(ST_GeomFromEWKT('CIRCULARSTRING(220268 150415 1,220227 150505 2,220227
150406 3)'));
st_ymin
-----
150406
```

### 相关信息

[ST\\_GeomFromEWKT](#), [ST\\_XMin](#), [ST\\_XMax](#), [ST\\_YMax](#), [ST\\_ZMax](#), [ST\\_ZMin](#)

### 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

虽然此函数接受 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

```
SELECT ST_ZMax('BOX3D(1 2 3, 4 5 6)');
st_zmax
-----
6

SELECT ST_ZMax(ST_GeomFromEWKT('LINESTRING(1 3 4, 5 6 7)'));
st_zmax
-----
7

SELECT ST_ZMax('BOX3D(-3 2 1, 3 4 1) ');
st_zmax
-----
1
--Observe THIS DOES NOT WORK because it will try to auto-cast the string representation to
  a BOX3D
SELECT ST_ZMax('LINESTRING(1 3 4, 5 6 7)');
--ERROR:  BOX3D parser - doesn't start with BOX3D(

SELECT ST_ZMax(ST_GeomFromEWKT('CIRCULARSTRING(220268 150415 1,220227 150505 2,220227
  150406 3)'));
st_zmax
-----
3
```

#### 相关信息

[ST\\_GeomFromEWKT](#), [ST\\_XMin](#), [ST\\_XMax](#), [ST\\_YMax](#), [ST\\_YMin](#), [ST\\_ZMax](#)

### 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

虽然此函数接受 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

```
SELECT ST_ZMin('BOX3D(1 2 3, 4 5 6)');
st_zmin
-----
3

SELECT ST_ZMin(ST_GeomFromEWKT('LINESTRING(1 3 4, 5 6 7)'));
st_zmin
-----
4

SELECT ST_ZMin('BOX3D(-3 2 1, 3 4 1) ');
st_zmin
-----
1
--Observe THIS DOES NOT WORK because it will try to auto-cast the string representation to
a BOX3D
SELECT ST_ZMin('LINESTRING(1 3 4, 5 6 7)');
--ERROR: BOX3D parser - doesn't start with BOX3D(

SELECT ST_ZMin(ST_GeomFromEWKT('CIRCULARSTRING(220268 150415 1,220227 150505 2,220227
150406 3)'));
st_zmin
-----
1
```

#### 相关信息

[ST\\_GeomFromEWKT](#), [ST\\_GeomFromText](#), [ST\\_XMin](#), [ST\\_XMax](#), [ST\\_YMax](#), [ST\\_YMin](#), [ST\\_ZMax](#)

## 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长度的比率。如果存在 Z 和 M 值，将先行插值。

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 的值先行插值，而 [ST\\_3DLineInterpolatePoint](#) 计算 3D 中的点，然后对 M 值先行插值。

---



#### Note

从版本 1.1.1 开始，此函数会插入 M 和 Z 值（如果存在），而之前的版本将它置为 0.0。

---

可用性：0.8.2，1.1.1 中添加支持 Z 和 M

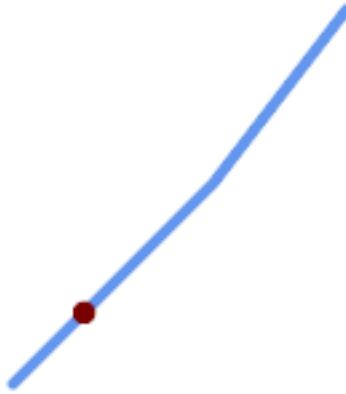
更改：2.1.0。在 2.0.x 之前，名称为 ST\_Line\_Interpolate\_Point。



该函数支持 3d 并且不会丢失 z-index。

---

示例



插图点位于 20% 位置 (0.20) 的图串

```
-- The point 20% along a line
SELECT ST_AsEWKT( ST_LineInterpolatePoint(
    'LINESTRING(25 50, 100 125, 150 190)',
    0.2 ));
-----
POINT(51.5974135047432 76.5974135047432)
```

3D 图的中点：

```
SELECT ST_AsEWKT( ST_LineInterpolatePoint('
    LINESTRING(1 2 3, 4 5 6, 6 7 8)',
    0.5 ));
-----
POINT(3.5 4.5 5.5)
```

直图上距离某点最近的点：

```
SELECT ST_AsText( ST_LineInterpolatePoint( line.geom,
    ST_LineLocatePoint( line.geom, 'POINT(4 3)'))
FROM (SELECT ST_GeomFromText('LINESTRING(1 2, 4 5, 6 7)') As geom) AS line;
-----
POINT(3 4)
```

相关信息

[ST\\_LineInterpolatePoints](#), [ST\\_3DLineInterpolatePoint](#), [ST\\_LineLocatePoint](#)

## 7.19.2 ST\_3DLineInterpolatePoint

ST\_3DLineInterpolatePoint — 返回沿 3D 图的小数指示位置插图点。

### 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入的点。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，则对其进行插值。



### Note

`ST_LineInterpolatePoint` 计算 2D 中的点，然后对 Z 和 M 的值进行插值，而此函数计算 3D 中的点，并且对 M 值进行插值。

可用性 : 3.0.0



函数支持 3d 并且不会丢失 z-index。

## 示例

沿 3D 线返回 20% 的点

```
SELECT ST_AsText(
  ST_3DLineInterpolatePoint('LINESTRING(25 50 70, 100 125 90, 150 190 200)',
    0.20));

 st_asetext
-----
POINT Z (59.0675892910822 84.0675892910822 79.0846904776219)
```

## 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入的点。

#### 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

返回沿一条线以分数间隔插入的一个或多个点。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，表示点之间的间距占线度的一部分。如果第三个参数为 `false`，则最多创建一个点（相当于 `ST_LineInterpolatePoint`。）

如果结果有零个或一个点，则将其作为 `POINT` 返回。如果它有多个或更多点，则以 `MULTIPOINT` 形式返回。

可用性 : 2.5.0

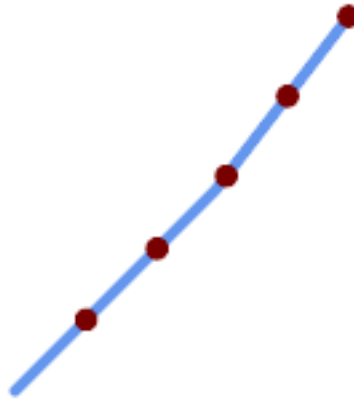


函数支持 3d 并且不会丢失 z-index。



功能支持 M 坐标。

示例



每 20% 插一次点的 *LineString*

```
--Return points each 20% along a 2D line
SELECT ST_AsText(ST_LineInterpolatePoints('LINESTRING(25 50, 100 125, 150 190)', 0.20))
-----
MULTIPOINT((51.5974135047432 76.5974135047432),(78.1948270094864 103.194827009486) ←
, (104.132163186446 130.37181214238),(127.066081593223 160.18590607119),(150 190))
```

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，称 `ST_Line_Locate_Point`。

示例

```
--Rough approximation of finding the street number of a point along the street
--Note the whole foo thing is just to generate dummy data that looks
--like house centroids and street
--We use ST_DWithin to exclude
--houses too far away from the street to be considered on the street
SELECT ST_AsText(house_loc) As as_text_house_loc,
       startstreet_num +
       CAST( (endstreet_num - startstreet_num)
            * ST_LineLocatePoint(street_line, house_loc) As integer) As
       street_num
FROM
  (SELECT ST_GeomFromText('LINESTRING(1 2, 3 4)') As street_line,
        ST_Point(x*1.01,y*1.03) As house_loc, 10 As startstreet_num,
        20 As endstreet_num
  FROM generate_series(1,3) x CROSS JOIN generate_series(2,4) As y)
As foo
WHERE ST_DWithin(street_line, house_loc, 0.2);

as_text_house_loc | street_num
-----+-----
POINT(1.01 2.06) |          10
POINT(2.02 3.09) |          15
POINT(3.03 4.12) |          20

--find closest point on a line to a point or other geometry
SELECT ST_AsText(ST_LineInterpolatePoint(foo.the_line, ST_LineLocatePoint(foo.the_line,
  ST_GeomFromText('POINT(4 3)'))))
FROM (SELECT ST_GeomFromText('LINESTRING(1 2, 4 5, 6 7)') As the_line) As foo;
st_astext
-----
POINT(3 4)
```

相关信息

[ST\\_DWithin](#), [ST\\_Length2D](#), [ST\\_LineInterpolatePoint](#), [ST\\_LineSubstring](#)

### 7.19.5 ST\_LineSubstring

`ST_LineSubstring` — 返回  $n$  个小数位置之间的直线部分。

#### Synopsis

```
geometry ST_LineSubstring(geometry a_linestring, float8 startfraction, float8 endfraction);
geography ST_LineSubstring(geography a_linestring, float8 startfraction, float8 endfraction);
```

#### 描述

`n` 算作 `n` 个入 `n` 在 `n` 定小数位置 `n` 开始和 `n` 束的部分的 `n`。第一个参数必须是 `LINESTRING`。第二个和第三个参数是 `[0, 1]` 范围内的 `n`，将起始位置和束位置表示 `n` 行 `n` 度的分数。如果存在添加的端点，`n` `Z` 和 `M` `n` 行插 `n`。如果 `startfraction` 和 `endfraction` 具有相同的 `n`，`n` 相当于 `ST_LineInterpolatePoint`。

**Note**

适用于 `LINESTRING`。要在 `MULTILINESTRING` 上使用，首先使用 `ST_LineMerge` 将它接起来。

**Note**

从 1.1.1 版开始，此函数 `M` 和 `Z` 行插。之前的版本将 `Z` 和 `M` 置未指定的。

增：3.4.0 - 引入了地理的支持。

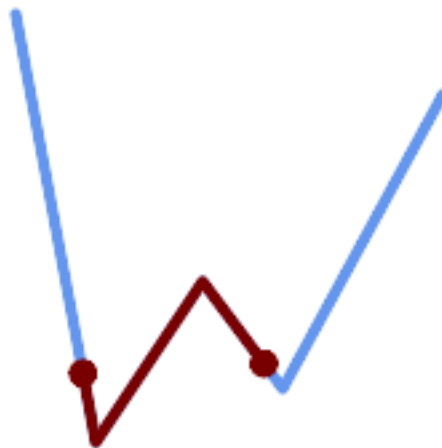
更改：2.1.0。在 2.0.x 之前，被称 `ST_Line_Substring`。

可用性：1.1.0, 1.1.1 中添加支持 `Z` 和 `M`



函数支持 3d 并且不会失 `z-index`。

示例



在 1/3 中范围上加的 `LineString (0.333, 0.666)`

```
SELECT ST_AsText(ST_LineSubstring( 'LINESTRING (20 180, 50 20, 90 80, 120 40, 180 150)', ←
  0.333, 0.666));
```

```
----- ←
LINESTRING (45.17311810399485 45.74337011202746, 50 20, 90 80, 112.97593050157862 ←
  49.36542599789519)
```

如果起始位置和结束位置相同，结果是一个 `POINT`。

```
SELECT ST_AsText(ST_LineSubstring( 'LINESTRING(25 50, 100 125, 150 190)', 0.333, 0.333));
-----
POINT(69.2846934853974 94.2846934853974)
```

将 `LineString` 切成度 100 或更短的部分的。它使用 `generate_series()` 和 `CROSS JOIN LATERAL` 来生成 `FOR` 循环的等效。

```

WITH data(id, geom) AS (VALUES
  ( 'A', 'LINESTRING( 0 0, 200 0)::geometry ),
  ( 'B', 'LINESTRING( 0 100, 350 100)::geometry ),
  ( 'C', 'LINESTRING( 0 200, 50 200)::geometry )
)
SELECT id, i,
       ST_AsText( ST_LineSubstring( geom, startfrac, LEAST( endfrac, 1 ) ) ) AS geom
FROM (
  SELECT id, geom, ST_Length(geom) len, 100 sublen FROM data
) AS d
CROSS JOIN LATERAL (
  SELECT i, (sublen * i) / len AS startfrac,
         (sublen * (i+1)) / len AS endfrac
  FROM generate_series(0, floor( len / sublen )::integer ) AS t(i)
  -- skip last i if line length is exact multiple of sublen
  WHERE (sublen * i) / len <
> 1.0
) AS d2;

```

id	i	geom
A	0	LINESTRING(0 0,100 0)
A	1	LINESTRING(100 0,200 0)
B	0	LINESTRING(0 100,100 100)
B	1	LINESTRING(100 100,200 100)
B	2	LINESTRING(200 100,300 100)
B	3	LINESTRING(300 100,350 100)
C	0	LINESTRING(0 200,50 200)

地理量沿球体表面行量，而几何量沿行量

```

SELECT ST_AsText(ST_LineSubstring( 'LINESTRING(-118.2436 34.0522, -71.0570 42.3611):: ↵
      geography, 0.333, 0.666),6) AS geog_sub
, ST_AsText(ST_LineSubstring('LINESTRING(-118.2436 34.0522, -71.0570 42.3611)::geometry, ↵
      0.333, 0.666),6) AS geom_sub;
-----
geog_sub | LINESTRING(-104.167064 38.854691,-87.674646 41.849854)
geom_sub | LINESTRING(-102.530462 36.819064,-86.817324 39.585927)

```

相关信息

[ST\\_Length](#), [ST\\_LineExtend](#), [ST\\_LineInterpolatePoint](#), [ST\\_LineMerge](#)

## 7.19.6 ST\_LocateAlong

ST\_LocateAlong — 返回几何上与量匹配的点。

### Synopsis

geometry **ST\_LocateAlong**(geometry geom\_with\_measure, float8 measure, float8 offset = 0);

## 描述

返回沿具有固定量值的量几何形的位置。如果是点或多点。不支持多边形输入。

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，负偏移将向右偏移。



### Note

结果具有 M 分量的线性几何使用此函数

结果由 *ISO/IEC 13249-3 SQL/MM* 空值指定。

可用性：1.1.0（旧名称 `ST_LocateAlong_Measure`）。

更改：2.0.0 在之前的版本中，结果曾被称 `ST_LocateAlong_Measure`。



功能支持 M 坐标。



方法遵循了 SQL/MM 规范。SQL-MM IEC 13249-3: 5.1.13

## 示例

```
SELECT ST_AsText(
  ST_LocateAlong(
    'MULTILINESTRINGM((1 2 3, 3 4 2, 9 4 3),(1 2 3, 5 4 5))'::geometry,
    3 ));
```

```
-----
MULTIPOINT M ((1 2 3),(9 4 3),(1 2 3))
```

## 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

返回一个几何形（集合），其中包含与指定量范围（包含）匹配的输入量几何形部分。

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形。

结果由 *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，`ST_Locate_Between_Measures`。

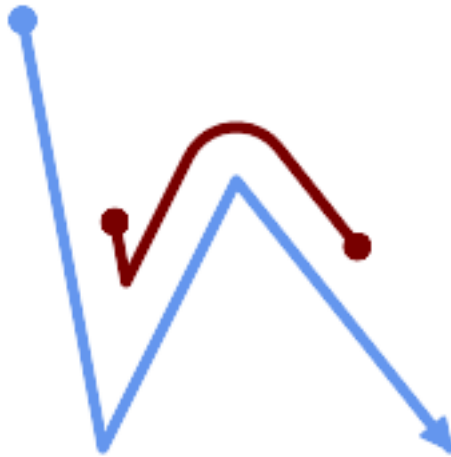
增：3.0.0 - 添加了多边形、TIN、三角形的支持。

✓ 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

```
SELECT ST_AsText(
  ST_LocateBetween(
    'MULTILINESTRING M ((1 2 3, 3 4 2, 9 4 3),(1 2 3, 5 4 5))':: geometry,
    1.5, 3 ));
-----
GEOMETRYCOLLECTION M (LINESTRING M (1 2 3,3 4 2,9 4 3),POINT M (1 2 3))
```



一个 *LineString*，其部分位于小 2 和小 8 之间，向左偏移

```
SELECT ST_AsText( ST_LocateBetween(
  ST_AddMeasure('LINESTRING (20 180, 50 20, 100 120, 180 20)', 0, 10),
  2, 8,
  20
));
-----
MULTILINESTRING((54.49835019899045 104.53426957938231,58.70056060327303 ↔
82.12248075654186,69.16695286779743 103.05526528559065,82.11145618000168 ↔
128.94427190999915,84.24893681714357 132.32493442618113,87.01636951231555 ↔
135.21267035596549,90.30307285299679 137.49198684843182,93.97759758337769 ↔
139.07172433557758,97.89298381958797 139.8887023914453,101.89263860095893 ↔
139.9102465862721,105.81659870902816 139.13549527600819,109.50792827749828 ↔
137.5954340631298,112.81899532549731 135.351656550512,115.6173761888606 ↔
132.49390095108848,145.31017306064817 95.37790486135405))
```

相关信息

[ST\\_LocateAlong](#), [ST\\_LocateBetweenElevations](#)

## 7.19.8 ST\_LocateBetweenElevations

ST\_LocateBetweenElevations — 返回位于高程 (Z) 范围内的几何形部分。

### Synopsis

geometry **ST\_LocateBetweenElevations**(geometry geom, float8 elevation\_start, float8 elevation\_end);

### 描述

返回一个几何形（集合），其中包含位于高程 (Z) 范围内的几何形部分。

裁剪非凸多边形可能会生成无效的几何形。

可用性：1.4.0

增：3.0.0 - 添加了多边形、TIN、三角形的支持。



函数支持 3d 并且不会失 z-index。

### 示例

```
SELECT ST_AsText(
  ST_LocateBetweenElevations(
    'LINESTRING(1 2 3, 4 5 6)::geometry,
    2, 4 ));
```

```
st_astext
```

```
-----
MULTILINESTRING Z ((1 2 3,2 3 4))
```

```
SELECT ST_AsText(
  ST_LocateBetweenElevations(
    'LINESTRING(1 2 6, 4 5 -1, 7 8 9)',
    6, 9)) As ewelev;
```

```
ewelev
```

```
-----
GEOMETRYCOLLECTION Z (POINT Z (1 2 6),LINESTRING Z (6.1 7.1 6,7 8 9))
```

### 相关信息

[ST\\_Dump](#), [ST\\_LocateBetween](#)

## 7.19.9 ST\_InterpolatePoint

ST\_InterpolatePoint — 返回最接近点的几何形的插量。

### Synopsis

float8 **ST\_InterpolatePoint**(geometry linear\_geom\_with\_measure, geometry point);



## 描述

返回最接近  $\varnothing$  定点的位置  $\varnothing$  的  $\varnothing$  性  $\varnothing$  量几何  $\varnothing$  形的插  $\varnothing$   $\varnothing$  量  $\varnothing$ 。



### Note

$\varnothing$   $\varnothing$  具有 M 分量的  $\varnothing$  性几何使用此函数

可用性: 2.0.0



$\varnothing$  函数支持 3d 并且不会  $\varnothing$  失 z-index。

## 示例

```
SELECT ST_InterpolatePoint('LINESTRING M (0 0 0, 10 0 20)', 'POINT(5 5)');
-----
10
```

## 相关信息

[ST\\_AddMeasure](#), [ST\\_LocateAlong](#), [ST\\_LocateBetween](#)

## 7.19.10 ST\_AddMeasure

`ST_AddMeasure` — 沿  $\varnothing$  性几何形状插  $\varnothing$   $\varnothing$  量  $\varnothing$ 。

### Synopsis

```
geometry ST_AddMeasure(geometry geom_mline, float8 measure_start, float8 measure_end);
```

## 描述

返回派生几何  $\varnothing$  形，其  $\varnothing$  量  $\varnothing$  在起点和  $\varnothing$  点之  $\varnothing$  性插  $\varnothing$ 。如果几何  $\varnothing$  形没有  $\varnothing$  量尺寸，  $\varnothing$  添加一个。如果几何  $\varnothing$  形具有  $\varnothing$  量尺寸，  $\varnothing$  会用新  $\varnothing$  覆盖它。  $\varnothing$  支持 `LINESTRINGS` 和 `MULTILINESTRINGS`。

可用性 : 1.1.0



$\varnothing$  函数支持 3d 并且不会  $\varnothing$  失 z-index。

## 示例

```
SELECT ST_AsText(ST_AddMeasure(
ST_GeomFromEWKT('LINESTRING(1 0, 2 0, 4 0)'),1,4)) As ewelev;
-----
LINESTRINGM(1 0 1,2 0 2,4 0 4)

SELECT ST_AsText(ST_AddMeasure(
ST_GeomFromEWKT('LINESTRING(1 0 4, 2 0 4, 4 0 4)'),10,40)) As ewelev;
```

```

                ewelev
-----
LINESTRING(1 0 4 10,2 0 4 20,4 0 4 40)

SELECT ST_AsText(ST_AddMeasure(
ST_GeomFromEWKT('LINESTRINGM(1 0 4, 2 0 4, 4 0 4)'),10,40)) As ewelev;
                ewelev
-----
LINESTRINGM(1 0 10,2 0 20,4 0 40)

SELECT ST_AsText(ST_AddMeasure(
ST_GeomFromEWKT('MULTILINESTRINGM((1 0 4, 2 0 4, 4 0 4),(1 0 4, 2 0 4, 4 0 4)'),10,70)) As ←
                ewelev;
                ewelev
-----
MULTILINESTRINGM((1 0 10,2 0 20,4 0 40),(1 0 40,2 0 50,4 0 70))

```

## 7.20 迹函数

### 7.20.1 ST\_IsValidTrajectory

ST\_IsValidTrajectory — 几何形是否有效迹。

#### Synopsis

boolean **ST\_IsValidTrajectory**(geometry line);

#### 描述

几何体是否有效的迹。有效迹表示有度量 (M) 的 **LINESTRING**。量必从每个点到下一个点增加。

有效迹作空函数 (如 **ST\_ClosestPointOfApproach**) 的入

可用性 : 2.2.0



函数支持 3d 并且不会失 z-index。

#### 示例

```

-- A valid trajectory
SELECT ST_IsValidTrajectory(ST_MakeLine(
    ST_MakePointM(0,0,1),
    ST_MakePointM(0,1,2))
);
t

-- An invalid trajectory
SELECT ST_IsValidTrajectory(ST_MakeLine(ST_MakePointM(0,0,1), ST_MakePointM(0,1,0)));
NOTICE:  Measure of vertex 1 (0) not bigger than measure of vertex 0 (1)
st_isvalidtrajectory
-----
f

```

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。

输入必须是 [ST\\_IsValidTrajectory](#) 的有效迹。如果迹在其 M 范围内不重，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
WITH inp AS ( SELECT
  ST_AddMeasure('LINESTRING Z (0 0 0, 10 0 5)')::geometry,
  extract(epoch from '2015-05-26 10:00')::timestampz,
  extract(epoch from '2015-05-26 11:00')::timestampz
) a,
  ST_AddMeasure('LINESTRING Z (0 2 10, 12 1 2)')::geometry,
  extract(epoch from '2015-05-26 10:00')::timestampz,
  extract(epoch from '2015-05-26 11:00')::timestampz
) b
), cpa AS (
  SELECT ST_ClosestPointOfApproach(a,b) m FROM inp
), points AS (
  SELECT ST_GeometryN(ST_LocateAlong(a,m),1) pa,
         ST_GeometryN(ST_LocateAlong(b,m),1) pb
  FROM inp, cpa
)
SELECT to_timestamp(m) t,
       ST_Distance(pa,pb) distance,
       ST_AsText(pa, 2) AS pa, ST_AsText(pb, 2) AS pb
FROM points, cpa;
```

t	distance pb	pa	↔
2015-05-26 10:45:31.034483-07	1.9603683315139542	POINT ZM (7.59 0 3.79 1432662331.03)	↔
		POINT ZM (9.1 1.24 3.93 1432662331.03)	

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的距离（在二维空间中）。

输入必须是 [ST\\_IsValidTrajectory](#) 的有效轨迹。如果轨迹在其 M 范围内不重叠，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
WITH inp AS ( SELECT
  ST_AddMeasure('LINESTRING Z (0 0 0, 10 0 5)::geometry,
    extract(epoch from '2015-05-26 10:00'::timestampz),
    extract(epoch from '2015-05-26 11:00'::timestampz)
  ) a,
  ST_AddMeasure('LINESTRING Z (0 2 10, 12 1 2)::geometry,
    extract(epoch from '2015-05-26 10:00'::timestampz),
    extract(epoch from '2015-05-26 11:00'::timestampz)
  ) b
)
SELECT ST_DistanceCPA(a,b) distance FROM inp;

   distance
-----
1.96036833151395
```

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

返回一个移动物体是否曾比指定距离更近。

输入必须是 `ST_IsValidTrajectory` 的有效轨迹。如果轨迹在其 M 范围内不重叠，返回 `False`。

可用性：2.2.0



函数支持 3d 并且不会丢失 z-index。

## 示例

```
WITH inp AS ( SELECT
  ST_AddMeasure('LINESTRING Z (0 0 0, 10 0 5)::geometry,
    extract(epoch from '2015-05-26 10:00'::timestampz),
    extract(epoch from '2015-05-26 11:00'::timestampz)
  ) a,
  ST_AddMeasure('LINESTRING Z (0 2 10, 12 1 2)::geometry,
    extract(epoch from '2015-05-26 10:00'::timestampz),
    extract(epoch from '2015-05-26 11:00'::timestampz)
  ) b
)
SELECT ST_CPWithin(a,b,2), ST_DistanceCPA(a,b) distance FROM inp;
```

```
st_cpawithin |      distance
-----+-----
t            | 1.96521473776207
```

## 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`) 打包并升级到指定版本或最新版本。

#### Synopsis

```
text PostGIS_Extensions_Upgrade(text target_version=null);
```

## 描述

将 PostGIS 扩展打包并升级到指定版本或最新版本。如果需要，只会打包和升级您在数据库中安装的扩展。报告完整的 PostGIS 版本并随后创建配置信息。这是每个 PostGIS 扩展行多个 `CREATE EXTENSION .. FROM unpackaged` 和 `ALTER EXTENSION .. UPDATE` 的编写。目前升级扩展 `postgis`, `postgis_raster`, `postgis_sfcgal`, `postgis_topology` 和 `postgis_tiger_geocoder`。

可用性：2.5.0

**Note**

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

```
SELECT PostGIS_Extensions_Upgrade();
```

```
NOTICE: Packaging extension postgis
NOTICE: Packaging extension postgis_raster
NOTICE: Packaging extension postgis_sfcgal
NOTICE: Extension postgis_topology is not available or not packagable for some reason
NOTICE: Extension postgis_tiger_geocoder is not available or not packagable for some reason
      reason
      postgis_extensions_upgrade
-----
Upgrade completed, run SELECT postgis_full_version(); for details
(1 row)
```

## 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。

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

```
SELECT PostGIS_Full_Version();
```

```

                                     postgis_full_version
-----
POSTGIS="3.4.0dev 3.3.0rc2-993-g61bdf43a7" [EXTENSION] PGSQL="160" GEOS="3.12.0dev-CAPI ←
-1.18.0" SFCGAL="1.3.8" PROJ="7.2.1 NETWORK_ENABLED=OFF URL_ENDPOINT=https://cdn.proj. ←
org USER_WRITABLE_DIRECTORY=/tmp/proj DATABASE_PATH=/usr/share/proj/proj.db" GDAL="GDAL ←
3.2.2, released 2021/03/05" LIBXML="2.9.10" LIBJSON="0.15" LIBPROTOBUF="1.3.3" WAGYU ←
="0.5.0 (Internal)" TOPOLOGY RASTER
(1 row)
```

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

```
SELECT PostGIS_GEOS_Version();
 postgis_geos_version
-----
3.12.0dev-CAPI-1.18.0
(1 row)
```

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

```
SELECT PostGIS_GEOS_Compiled_Version();
 postgis_geos_compiled_version
-----
 3.12.0
(1 row)
```

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

```
SELECT PostGIS_Liblwgeom_Version();
 postgis_liblwgeom_version
-----
 3.4.0dev 3.3.0rc2-993-g61bdf43a7
(1 row)
```

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

```
SELECT PostGIS_LibXML_Version();
 postgis_libxml_version
-----
 2.9.10
(1 row)
```

## 相关信息

[PostGIS\\_Full\\_Version](#), [PostGIS\\_Lib\\_Version](#), [PostGIS\\_PROJ\\_Version](#), [PostGIS\\_GEOS\\_Version](#), [PostGIS\\_Versio](#)

### 7.21.7 PostGIS\_LibJSON\_Version

PostGIS\_LibJSON\_Version — Returns the version number of the libjson-c library.

#### Synopsis

text **PostGIS\_LibJSON\_Version()**;

## 描述

Returns the version number of the LibJSON-C library.

## 示例

```
SELECT PostGIS_LibJSON_Version();
 postgis_libjson_version
-----
 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 库的构建日期。

## 示例

```
SELECT PostGIS_Lib_Build_Date();
 postgis_lib_build_date
-----
2023-06-22 03:56:11
(1 row)
```

## 7.21.9 PostGIS\_Lib\_Version

PostGIS\_Lib\_Version — 返回 PostGIS 库的版本号。

### Synopsis

```
text PostGIS_Lib_Version();
```

## 描述

返回 PostGIS 库的版本号。

## 示例

```
SELECT PostGIS_Lib_Version();
 postgis_lib_version
-----
3.4.0dev
(1 row)
```

## 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 库的版本号。

### Synopsis

```
text PostGIS_PROJ_Version();
```

---

## 描述

返回 PROJ 的版本号和 proj 的一些配置。

增功能 : 3.4.0 在包括 proj.db 位置的 NETWORK\_ENABLED、URL\_ENDPOINT 和 DATABASE\_PATH

## 示例

```
SELECT PostGIS_PROJ_Version();
   postgis_proj_version
-----
7.2.1 NETWORK_ENABLED=OFF URL_ENDPOINT=https://cdn.proj.org USER_WRITABLE_DIRECTORY=/tmp/ ↔
   proj DATABASE_PATH=/usr/share/proj/proj.db
(1 row)
```

## 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` — Returns the version number of the PROJ library against which PostGIS was built.

#### Synopsis

```
text PostGIS_PROJ_Compiled_Version();
```

## 描述

Returns the version number of the PROJ library, or against which PostGIS was built.

可用性 : 3.5.0

## 示例

```
SELECT PostGIS_PROJ_Compiled_Version();
   postgis_proj_compiled_version
-----
   9.1.1
(1 row)
```

## 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

```
SELECT PostGIS_Wagyu_Version();
 postgis_wagyu_version
-----
 0.5.0 (Internal)
(1 row)
```

### 相关信息

[PostGIS\\_Full\\_Version](#), [PostGIS\\_GEOS\\_Version](#), [PostGIS\\_PROJ\\_Version](#), [PostGIS\\_Lib\\_Version](#), [PostGIS\\_LibXML](#)  
[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

```
SELECT PostGIS_Scripts_Build_Date();
 postgis_scripts_build_date
-----
 2023-06-22 03:56:11
(1 row)
```

### 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☐出与 [PostGIS\\_Scripts\\_Released](#) 的☐出不匹配，您可能☐☐了正确升☐☐有数据库☐的机会。有关☐☐信息，☐参☐升☐部分。

可用性 : 0.9.0

#### 示例

```
SELECT PostGIS_Scripts_Installed();
 postgis_scripts_installed
-----
3.4.0dev 3.3.0rc2-993-g61bdf43a7
(1 row)
```

#### 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☐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☐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☐。保留向后兼容性。

可用性 : 0.9.0

示例

```
SELECT PostGIS_Scripts_Released();
   postgis_scripts_released
-----
3.4.0dev 3.3.0rc2-993-g61bdf43a7
(1 row)
```

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_GEOS`。

### Synopsis

text **PostGIS\_Version()**;

描述

返回 PostGIS 版本号和 `USE_GEOS`。

示例

```
SELECT PostGIS_Version();
   postgis_version
-----
3.4 USE_GEOS=1 USE_PROJ=1 USE_STATS=1
(1 row)
```

相关信息

[PostGIS\\_Full\\_Version](#), [PostGIS\\_GEOS\\_Version](#), [PostGIS\\_Lib\\_Version](#), [PostGIS\\_LibXML\\_Version](#), [PostGIS\\_PROJ\\_Version](#)

## 7.22 大一自定变量 (GUCs)

### 7.22.1 postgis.backend

postgis.backend — `USE_GEOS` 和 `USE_SFCGAL` 重写的功能提供服务的后端。取值：`geos` 或 `sfcgal`。默认为 `geos`。

## 描述

当您使用 `sfcgal` 支持 PostGIS，此 GUC 才相关。默认情况下，`geos` 后端用于 GEOS 和 SFCGAL 具有相同命名函数的函数。此量允许您覆盖并使 `sfcgal` 成后端来服务。

可用性：2.1.0

## 示例

设置生命周期后端

```
set postgis.backend = sfcgal;
```

设置数据新接的后端

```
ALTER DATABASE mygisdb SET postgis.backend = sfcgal;
```

## 相关信息

Chapter 8

### 7.22.2 postgis.gdal\_datapath

`postgis.gdal_datapath` — 用于分配 GDAL 的 `GDAL_DATA` 的配置。如果未设置，使用境置的 `GDAL_DATA` 量。

## 描述

一个 PostgreSQL GUC 量，用于设置 GDAL 的 `GDAL_DATA` 的。 `postgis.gdal_datapath` 是 GDAL 数据文件的完整物理路径。

此配置最适用于 GDAL 数据文件路径未硬化的 Windows 平台。当 GDAL 的数据文件不在 GDAL 的期路径中，也设置此。



#### Note

可以在 PostgreSQL 的配置文件 `postgresql.conf` 中设置。也可以通过或事来设置。

可用性：2.2.0



#### Note

有关 `GDAL_DATA` 的其他信息可在 GDAL 的 [配置](#) 中找到。

示例

设置和重置 `postgis.gdal_datapath`

```
SET postgis.gdal_datapath TO '/usr/local/share/gdal.hidden';
SET postgis.gdal_datapath TO default;
```

在 Windows 上为特定数据源行设置

```
ALTER DATABASE gisdb
SET postgis.gdal_datapath = 'C:/Program Files/PostgreSQL/9.3/gdal-data';
```

相关信息

[PostGIS\\_GDAL\\_Version](#), [ST\\_Transform](#)

### 7.22.3 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。影响 GDAL 配置量 `GDAL_SKIP`。

描述

用于设置 PostGIS 环境中使用的 GDAL 程序的配置。影响 GDAL 配置量 `GDAL_SKIP`。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.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传递给 PostgreSQL 的进程来设置。

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` 启用 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!

#### Note

在标准 PostGIS 安装中，`postgis.gdal_enabled_drivers` 设置为 `DISABLE_ALL`。

Note!

#### Note

有关 `GDAL_SKIP` 的其他信息可在 GDAL 的 [配置](#) 中找到。

可用性：2.2.0



示例

To set and reset `postgis.gdal_enabled_drivers` for current session

```
SET postgis.gdal_enabled_drivers = 'ENABLE_ALL';
SET postgis.gdal_enabled_drivers = default;
```

Set for all new connections to a specific database to specific drivers

```
ALTER DATABASE mygisdb SET postgis.gdal_enabled_drivers TO 'GTiff PNG JPEG';
```

Setting for whole database cluster to enable all drivers. Requires super user access. Also note that database, session, and user settings override this.

```
--writes to postgres.auto.conf
ALTER SYSTEM SET postgis.gdal_enabled_drivers TO 'ENABLE_ALL';
--Reloads postgres conf
SELECT pg_reload_conf();
```

相关信息

[ST\\_FromGDALRaster](#), [ST\\_AsGDALRaster](#), [ST\\_AsTIFF](#), [ST\\_AsPNG](#), [ST\\_AsJPEG](#), [postgis.enable\\_outdb\\_raster](#)

## 7.22.4 `postgis.enable_outdb_rasters`

`postgis.enable_outdb_rasters` — 一个布尔配置项，用于启用数据外格波段的输出。

描述

一个布尔配置项，用于启用数据外格波段的输出。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.enable_outdb_rasters` 的初始值也可以通过将具有非零值的配置项 `POSTGIS_ENABLE_OUTDB_RASTER` 在 PostgreSQL 的进程中设置。



### Note

即使 `postgis.enable_outdb_rasters` 设置为 `True`，GUC `postgis.gdal_enabled_drivers` 也会确定可输出的格式。



### Note

在标准 PostGIS 安装中，`postgis.enable_outdb_rasters` 设置为 `False`。

可用性：2.2.0

示例

设置和重置当前会话的 `postgis.enable_outdb_rasters`

```
SET postgis.enable_outdb_rasters TO True;
SET postgis.enable_outdb_rasters = default;
SET postgis.enable_outdb_rasters = True;
SET postgis.enable_outdb_rasters = False;
```

Set for all new connections to a specific database

```
ALTER DATABASE gisdb SET postgis.enable_outdb_rasters = true;
```

Setting for whole database cluster. Requires super user access. Also note that database, session, and user settings override this.

```
--writes to postgres.auto.conf
ALTER SYSTEM SET postgis.enable_outdb_rasters = true;
--Reloads postgres conf
SELECT pg_reload_conf();
```

相关信息

[postgis.gdal\\_enabled\\_drivers](#) [postgis.gdal\\_vsi\\_options](#)

## 7.22.5 postgis.gdal\_vsi\_options

`postgis.gdal_vsi_options` — 用于设置管理外部数据格式使用的字符串配置。

描述

用于设置管理外部数据格式使用的字符串配置。配置控制如 GDAL 分配本地数据缓存的空大小、是否获取概述以及远程 out-db 数据源使用某些内容。

可用性：3.2.0

示例

当前会话设置 `postgis.gdal_vsi_options`：

```
SET postgis.gdal_vsi_options = 'AWS_ACCESS_KEY_ID=xxxxxxxxxxxxxxxxx AWS_SECRET_ACCESS_KEY=
yyyyyyyyyyyyyyyyyyyyyyyyyyyyyy';
```

使用 LOCAL 关键字设置当前事务 `postgis.gdal_vsi_options`：

```
SET LOCAL postgis.gdal_vsi_options = 'AWS_ACCESS_KEY_ID=xxxxxxxxxxxxxxxxx
AWS_SECRET_ACCESS_KEY=yyyyyyyyyyyyyyyyyyyyyyyyyyyyyy';
```

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☐界框。

#### Synopsis

```
geometry PostGIS_AddBBox(geometry geomA);
```

#### 描述

向几何体添加☐界框。☐将使基于☐界框的☐☐更快，但会增加几何形状的大小。



#### Note

☐界框会自☐添加到几何☐形中，因此通常不需要☐☐做，除非生成的☐界框以某种方式☐坏或者您的旧安装缺少☐界框。然后你需要☐除旧的并重新☐取。



此方法支持☐形字符串和曲☐。

#### 示例

```
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☐除☐界框☐存。

#### 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。它用于删除坏的边界框。边界框坏的一个迹象是 `ST_Intersects` 和其他关系函数漏掉了理应返回 `true` 的几何图形。



### Note

边界框会自动添加到几何图形中并提高查询速度，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 系列中已经观察到这种损坏，其中当几何图形更改时，并不总是重新计算存储的 `bbox`，并且在不重新添加边界框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` 或逐行重新添加。



此方法支持几何字符串和曲线。

## 示例

```
--This example drops bounding boxes where the cached box is not correct
--The force to ST_AsBinary before applying Box2D forces a
  recalculation of the box, and Box2D applied to the table
  geometry always
-- returns the cached bounding box.
UPDATE sometable
SET geom = PostGIS_DropBBox(geom)
WHERE Not (Box2D(ST_AsBinary(geom)) = Box2D(geom));

UPDATE sometable
SET geom = PostGIS_AddBBox(geom)
WHERE Not PostGIS_HasBBOX(geom);
```

## 相关信息

[PostGIS\\_AddBBox](#), [PostGIS\\_HasBBox](#), [Box2D](#)

### 7.23.3 PostGIS\_HasBBox

`PostGIS_HasBBox` — 如果几何体的 `bbox` 已存储，返回 `TRUE`，否则返回 `FALSE`。

## Synopsis

```
boolean PostGIS_HasBBox(geometry geomA);
```

## 描述

如果几何体的 `bbox` 已存储，返回 `TRUE`，否则返回 `FALSE`。使用 [PostGIS\\_AddBBox](#) 和 [PostGIS\\_DropBBox](#) 来控制存储。



此方法支持几何字符串和曲线。

## 示例

```
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函数。为了确保健壮性，几何坐具有精确的有理数表示。

的安装说明可以在 SFCGAL 主页 (<http://www.sfcgal.org>) 上找到。要用些功能，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



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 过滤器和设置器

### 8.2.1 CG\_ForceLHR

CG\_ForceLHR — 强制 LHR 方向

#### Synopsis

```
geometry CG_ForceLHR(geometry geom);
```

## 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规则三角网面 (TIN)。

### 8.2.2 CG\_IsPlanar

CG\_IsPlanar — 检查表面是否平坦

#### Synopsis

```
boolean CG_IsPlanar(geometry geom);
```

## 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规则三角网面 (TIN)。

### 8.2.3 CG\_IsSolid

CG\_IsSolid — 检查几何体是否是体。不执行有效性检查。

#### Synopsis

```
boolean CG_IsSolid(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规则三角网面 (TIN)。

### 8.2.4 CG\_MakeSolid

CG\_MakeSolid — 将几何体造成体。不执行任何检查。要得到有效的体，输入几何体必须是符合多面体曲面或符合 TIN。

#### Synopsis

```
geometry CG_MakeSolid(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规则三角网面 (TIN)。

### 8.2.5 CG\_Orientation

CG\_Orientation — 确定表面方向

#### Synopsis

```
integer CG_Orientation(geometry geom);
```



## 描述

☑函数☑适用于多☑形。如果多☑形是逆☑☑方向，☑返回 -1；如果多☑形是☑☑☑方向，☑返回 1。

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。

## 8.2.6 CG\_Area

CG\_Area — Calculates the area of a geometry

### Synopsis

double precision **CG\_Area**( geometry geom );

## 描述

Calculates the area of a geometry.

Performed by the SFCGAL module



### Note

NOTE: this function returns a double precision value representing the area.

可用性：3.5.0



☑方法需要 SFCGAL 后端。

## 几何示例

```
SELECT CG_Area('Polygon ((0 0, 0 5, 5 5, 5 0, 0 0), (1 1, 2 1, 2 2, 1 2, 1 1), (3 3, 4 3, 4 4, 3 4, 3 3))');
      cg_area
      -
      25
(1 row)
```

## 相关信息

[ST\\_3DArea](#), [ST\\_Area](#)

## 8.2.7 CG\_3DArea

CG\_3DArea — ☑算 3D 表面几何形状的面☑。☑于固体将返回 0。

## Synopsis

```
float CG_3DArea(geometry geom1);
```

### 描述

可用性 : 3.5.0

- ✔ 方法需要 SFCGAL 后端。
- ✔ 方法遵循了 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，而不是实体。因此它具有表面。一旦实体化，就没有表面了。

```
SELECT CG_3DArea(geom) As cube_surface_area,
       CG_3DArea(CG_MakeSolid(geom)) As solid_surface_area
FROM (SELECT 'POLYHEDRALSURFACE( ((0 0 0, 0 0 1, 0 1 1, 0 1 0, 0 0 0)),
  ((0 0 0, 0 1 0, 1 1 0, 1 0 0, 0 0 0)),
  ((0 0 0, 1 0 0, 1 0 1, 0 0 1, 0 0 0)),
  ((1 1 0, 1 1 1, 1 0 1, 1 0 0, 1 1 0)),
  ((0 1 0, 0 1 1, 1 1 1, 1 1 0, 0 1 0)),
  ((0 0 1, 1 0 1, 1 1 1, 0 1 1, 0 0 1)) )'::geometry) As f(geom);

cube_surface_area | solid_surface_area
-----+-----
6 | 0
```

### 相关信息

[CG\\_Area](#), [CG\\_MakeSolid](#), [CG\\_IsSolid](#), [CG\\_Area](#)

## 8.2.8 CG\_Volume






CG\_Volume — 计算 3D 实体的体积。如果用于表面（甚至闭合）几何形将返回 0。

### 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规则三角网面 (TIN)。
-  方法符合了 SQL/MM 规范。SQL-MM IEC 13249-3: 9.1 (same as CG\_3DVolume)

## 示例

When closed surfaces are created with WKT, they are treated as areal rather than solid. To make them solid, you need to use **CG\_MakeSolid**. Areal geometries have no volume. Here is an example to demonstrate.

```
SELECT CG_Volume(geom) As cube_surface_vol,
       CG_Volume(CG_MakeSolid(geom)) As solid_surface_vol
FROM (SELECT 'POLYHEDRALSURFACE( ((0 0 0, 0 0 1, 0 1 1, 0 1 0, 0 0 0)),
  ((0 0 0, 0 1 0, 1 1 0, 1 0 0, 0 0 0)),
  ((0 0 0, 1 0 0, 1 0 1, 0 0 1, 0 0 0)),
  ((1 1 0, 1 1 1, 1 0 1, 1 0 0, 1 1 0)),
  ((0 1 0, 0 1 1, 1 1 1, 1 1 0, 0 1 0)),
  ((0 0 1, 1 0 1, 1 1 1, 0 1 1, 0 0 1)) )'::geometry) As f(geom);

cube_surface_vol | solid_surface_vol
-----+-----
0 | 1
```

## 相关信息

[CG\\_3DArea](#), [CG\\_MakeSolid](#), [CG\\_IsSolid](#)

## 8.2.9 ST\_ForceLHR

ST\_ForceLHR — 强制 LHR 方向

### Synopsis

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

## 描述



#### Warning

**ST\_ForceLHR** is deprecated as of 3.5.0. Use **CG\_ForceLHR** instead.

可用性 : 2.1.0

- ✔ ☑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)。

### 8.2.10 ST\_IsPlanar

ST\_IsPlanar — ☑☑表面是否平坦

#### Synopsis

```
boolean ST_IsPlanar(geometry geom);
```

描述



#### Warning

**ST\_IsPlanar** is deprecated as of 3.5.0. Use **CG\_IsPlanar** instead.

---

可用性 : 2.2.0 : ☑在 2.1.0 中已☑☑, 但在 2.1 版本中意外☑漏。

- ✔ ☑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)。

### 8.2.11 ST\_IsSolid

ST\_IsSolid — ☑☑几何体是否☑☑体。不☑行有效性☑☑。

#### Synopsis

```
boolean ST_IsSolid(geometry geom1);
```

---

## 描述

**Warning**

`ST_IsSolid` is deprecated as of 3.5.0. Use `CG_IsSolid` instead.

可用性 : 2.2.0



☑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)。

### 8.2.12 ST\_MakeSolid

`ST_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 ST_MakeSolid(geometry geom1);
```

## 描述

**Warning**

`ST_MakeSolid` is deprecated as of 3.5.0. Use `CG_MakeSolid` instead.

可用性 : 2.2.0



☑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)。

### 8.2.13 ST\_Orientation

`ST_Orientation` — 确定表面方向

#### Synopsis

```
integer ST_Orientation(geometry geom);
```

## 描述

**Warning**

**ST\_Orientation** is deprecated as of 3.5.0. Use **CG\_Orientation** instead.

☒函数☒适用于多☒形。如果多☒形是逆☒☒方向，☒返回 -1；如果多☒形是☒☒☒方向，☒返回 1。

可用性：2.1.0



☒方法需要 SFCGAL 后端。



☒函数支持 3d 并且不会☒失 z-index。

## 8.2.14 ST\_3DArea

ST\_3DArea — ☒算 3D 表面几何形状的面☒。☒于固体将返回 0。

### Synopsis

```
floatST_3DArea(geometry geom1);
```

## 描述

**Warning**

**ST\_3DArea** is deprecated as of 3.5.0. Use **CG\_3DArea** instead.

可用性：2.1.0



☒方法需要 SFCGAL 后端。



☒方法☒☒了 SQL/MM ☒范。SQL-MM IEC 13249-3: 8.1, 10.5



☒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

注意：默☒情况下，从 WKT ☒建的 PolyhedralSurface 是曲面几何体，而不是☒体。因此它具有表面☒。一旦☒☒☒体，就没有面☒了。

```

SELECT ST_3DArea(geom) As cube_surface_area,
       ST_3DArea(ST_MakeSolid(geom)) As solid_surface_area
FROM (SELECT 'POLYHEDRALSURFACE( ((0 0 0, 0 0 1, 0 1 1, 0 1 0, 0 0 0)),
  ((0 0 0, 0 1 0, 1 1 0, 1 0 0, 0 0 0)),
  ((0 0 0, 1 0 0, 1 0 1, 0 0 1, 0 0 0)),
  ((1 1 0, 1 1 1, 1 0 1, 1 0 0, 1 1 0)),
  ((0 1 0, 0 1 1, 1 1 1, 1 1 0, 0 1 0)),
  ((0 0 1, 1 0 1, 1 1 1, 0 1 1, 0 0 1)) )'::geometry) As f(geom);

cube_surface_area | solid_surface_area
-----+-----
6 | 0

```

相关信息

[ST\\_Area](#), [ST\\_MakeSolid](#), [ST\\_IsSolid](#), [ST\\_Area](#)

## 8.2.15 ST\_Volume

`ST_Volume` — 计算 3D 体的体积。如果用于表面（甚至组合）几何形将返回 0。

### Synopsis

```
float ST_Volume(geometry geom1);
```






描述



#### Warning

`ST_Volume` is deprecated as of 3.5.0. Use `CG_Volume` instead.

可用性：2.2.0

-  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)。
-  方法符合了 SQL/MM 规范。SQL-MM IEC 13249-3: 9.1 (与 `ST_3DVolume` 相同)

示例

当使用 WKT 构建曲面时，它将被视为面而不是体。要使它成为体，您需要使用 `ST_MakeSolid`。面几何没有体积。这里有一个例子来演示。

```

SELECT ST_Volume(geom) As cube_surface_vol,
       ST_Volume(ST_MakeSolid(geom)) As solid_surface_vol
FROM (SELECT 'POLYHEDRALSURFACE( ((0 0 0, 0 0 1, 0 1 1, 0 1 0, 0 0 0)),
  ((0 0 0, 0 1 0, 1 1 0, 1 0 0, 0 0 0)),
  ((0 0 0, 1 0 0, 1 0 1, 0 0 1, 0 0 0)),
  ((1 1 0, 1 1 1, 1 0 1, 1 0 0, 1 1 0)),
  ((0 1 0, 0 1 1, 1 1 1, 1 1 0, 0 1 0)),
  ((0 0 1, 1 0 1, 1 1 1, 0 1 1, 0 0 1)) )'::geometry) As f(geom);

cube_surface_vol | solid_surface_vol
-----+-----
0 | 1

```

相关信息

[ST\\_3DArea](#), [ST\\_MakeSolid](#), [ST\\_IsSolid](#)

## 8.3 SFCGAL ☒理和关系函数

### 8.3.1 CG\_Intersection

CG\_Intersection — Computes the intersection of two geometries

#### Synopsis

geometry **CG\_Intersection**( geometry geomA , geometry geomB );

描述

Computes the intersection of two geometries.

Performed by the SFCGAL module



#### Note

NOTE: this function returns a geometry representing the intersection.

可用性 : 3.5.0



☒方法需要 SFCGAL 后端。

几何示例

```

SELECT ST_AsText(CG_Intersection('LINESTRING(0 0, 5 5)', 'LINESTRING(5 0, 0 5)'));
       cg_intersection
-----
POINT(2.5 2.5)
(1 row)

```



相关信息

[ST\\_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。

Performed by the SFCGAL module



#### Note

注意：它是返回布尔值而不是整数的“允”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

```
SELECT CG_Intersects('POINT(0 0)::geometry, 'LINESTRING ( 2 0, 0 2 ) '::geometry);
cg_intersects
-----
f
(1 row)
SELECT CG_Intersects('POINT(0 0)::geometry, 'LINESTRING ( 0 0, 0 2 ) '::geometry);
cg_intersects
-----
t
(1 row)
```

相关信息

[CG\\_3DIntersects](#), [ST\\_3DIntersects](#), [ST\\_Intersects](#), [ST\\_Disjoint](#)

### 8.3.3 CG\_3DIntersects

CG\_3DIntersects — Tests if two 3D geometries intersect

## Synopsis

```
boolean CG_3DIntersects( geometry geomA , geometry geomB );
```

### 描述

Tests if two 3D geometries intersect. 3D geometries intersect if they have any point in common in the three-dimensional space.

Performed by the SFCGAL module



### Note

注意：☒是返回布☒☒而不是整数的“允☒”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

```
SELECT CG_3DIntersects('POINT(1.2 0.1 0)', 'POLYHEDRALSURFACE(((0 0 0,0.5 0.5 0,1 0 0,1 1 0,0 1 0,0 0 0)),((1 0 0,2 0 0,2 1 0,1 1 0,1 0 0),(1.2 0.2 0,1.2 0.8 0,1.8 0.8 0,1.8 0.2 0,1.2 0.2 0)))');
   cg_3dintersects
-----
t
(1 row)
```

### 相关信息

[CG\\_Intersects](#), [ST\\_3DIntersects](#), [ST\\_Intersects](#), [ST\\_Disjoint](#)

## 8.3.4 CG\_Difference

CG\_Difference — Computes the geometric difference between two geometries

### Synopsis

```
geometry CG_Difference( geometry geomA , geometry geomB );
```

## 描述

Computes the geometric difference between two geometries. The resulting geometry is a set of points that are present in geomA but not in geomB.

Performed by the SFCGAL module

**Note**

NOTE: this function returns a geometry.

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

```
SELECT ST_AsText(CG_Difference('POLYGON((0 0, 0 1, 1 1, 1 0, 0 0))'::geometry, 'LINESTRING ↵
(0 0, 2 2)'::geometry));
cg_difference
-----
POLYGON((0 0,1 0,1 1,0 1,0 0))
(1 row)
```

## 相关信息

[ST\\_3DDifference](#), [ST\\_Difference](#)

### 8.3.5 ST\_3DDifference

ST\_3DDifference — ☒行 3D 差异

#### Synopsis

geometry **ST\_3DDifference**(geometry geom1, geometry geom2);

## 描述

**Warning**

**ST\_3DDifference** is deprecated as of 3.5.0. Use **CG\_3DDifference** instead.

返回 `geom1` 中不属于 `geom2` 的部分。

可用性 : 2.2.0

- ✔ 方法需要 SFCGAL 后端。
- ✔ 方法符合了 SQL/MM 规范。SQL-MM IEC 13249-3: 5.1
- ✔ 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)。

### 8.3.6 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。

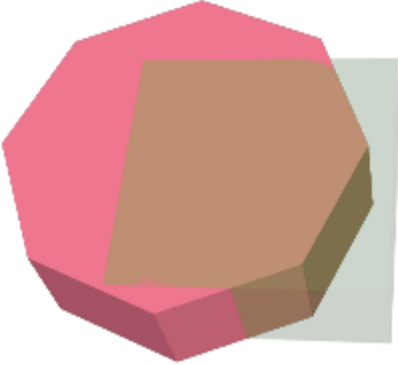
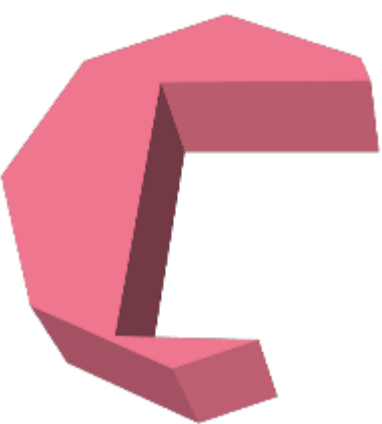
可用性 : 3.5.0

- ✔ 方法需要 SFCGAL 后端。
- ✔ 方法符合了 SQL/MM 规范。SQL-MM IEC 13249-3: 5.1
- ✔ 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

3D 像是使用 `ST_AsX3D` 生成的, 并使用 `X3Dom HTML Javascript 渲染` 以 HTML 形式渲染。

---

<pre>SELECT CG_Extrude(ST_Buffer(   ST_GeomFromText('POINT(100 90)'),   50, ' CG_Extrude(ST_Buffer(ST_GeomFromText('POINT(80 80)'),   50, ' quad_segs=1'),0,0,30) AS geom2;</pre> <div style="text-align: center; margin: 10px 0;">  </div> <p>原始 3D 几何体相加。geom2 是要删除的部分。</p>	<pre>SELECT CG_3DDifference(geom1,geom2)   50, ' quad_segs=2',0,0,30) AS geom1,   50, ' CG_Extrude(   50, ' quad_segs=1'),0,0,30) AS geom2 ) As t;</pre> <div style="text-align: center; margin: 10px 0;">  </div> <p>删除 geom2 后剩下什么</p>
--	--

相关信息

[CG\\_Extrude](#), [ST\\_AsX3D](#), [CG\\_3DIntersection](#) [CG\\_3DUnion](#)

### 8.3.7 CG\_Distance

CG\_Distance — Computes the minimum distance between two geometries

#### Synopsis

double precision **CG\_Distance**( geometry geomA , geometry geomB );

#### 描述

Computes the minimum distance between two geometries.

Performed by the SFCGAL module



#### Note

NOTE: this function returns a double precision value representing the distance.

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

```
SELECT CG_Distance('LINESTRING(0.0 0.0,-1.0 -1.0)', 'LINESTRING(3.0 4.0,4.0 5.0)');
   cg_distance
-----
      2.0
(1 row)
```

相关信息

[CG\\_3DDistance](#), [CG\\_Distance](#)

### 8.3.8 CG\_3DDistance

CG\_3DDistance — Computes the minimum 3D distance between two geometries

#### Synopsis

double precision **CG\_3DDistance**( geometry geomA , geometry geomB );

描述

Computes the minimum 3D distance between two geometries.

Performed by the SFCGAL module



#### Note

NOTE: this function returns a double precision value representing the 3D distance.

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

```
SELECT CG_3DDistance('LINESTRING(-1.0 0.0 2.0,1.0 0.0 3.0)', 'TRIANGLE((-4.0 0.0 1.0,4.0 ↔
0.0 1.0,0.0 4.0 1.0,-4.0 0.0 1.0))');
   cg_3ddistance
-----
              1
(1 row)
```

相关信息

[CG\\_Distance](#), [ST\\_3DDistance](#)

### 8.3.9 ST\_3DConvexHull

ST\_3DConvexHull — 计算几何体的 3D 凸包。

#### Synopsis

```
geometry ST_3DConvexHull(geometry geom1);
```

描述







#### Warning

ST\_3DConvexHull is deprecated as of 3.5.0. Use [CG\\_3DConvexHull](#) instead.

---

可用性 : 3.3.0

-  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)。

### 8.3.10 CG\_3DConvexHull





CG\_3DConvexHull — 计算几何体的 3D 凸包。

#### Synopsis

```
geometry CG_3DConvexHull(geometry geom1);
```

描述

可用性 : 3.5.0

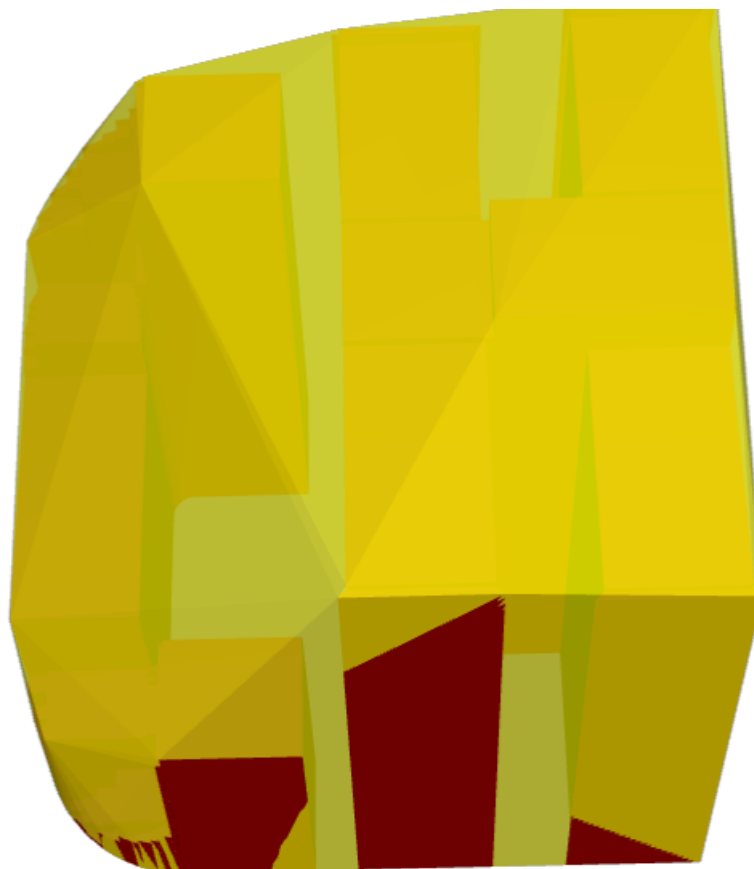
-  方法需要 SFCGAL 后端。
  -  函数支持 3d 并且不会丢失 z-index。
  -  函数支持多面体曲面。
  -  此函数支持三角形和不规则三角网面 (TIN)。
-

示例

```
SELECT ST_AsText(CG_3DConvexHull('LINESTRING Z(0 0 5, 1 5 3, 5 7 6, 9 5 3 , 5 7 5, 6 3 5) ↔  
'::geometry));
```

```
POLYHEDRALSURFACE Z (((1 5 3,9 5 3,0 0 5,1 5 3)),((1 5 3,0 0 5,5 7 6,1 5 3)),((5 7 6,5 7 ↔  
5,1 5 3,5 7 6)),((0 0 5,6 3 5,5 7 6,0 0 5)),((6 3 5,9 5 3,5 7 6,6 3 5)),((0 0 5,9 5 3,6 ↔  
3 5,0 0 5)),((9 5 3,5 7 5,5 7 6,9 5 3)),((1 5 3,5 7 5,9 5 3,1 5 3)))
```

```
WITH f AS (SELECT i, CG_Extrude(geom, 0,0, i ) AS geom  
FROM ST_Subdivide(ST_Letters('CH'),5) WITH ORDINALITY AS sd(geom,i)  
)  
SELECT CG_3DConvexHull(ST_Collect(f.geom) )  
FROM f;
```



原始几何体覆盖 3D 凸包

相关信息

[ST\\_Letters](#), [ST\\_AsX3D](#)

### 8.3.11 ST\_3DIntersection

ST\_3DIntersection — ☒行 3D 相交



## Synopsis

geometry **ST\_3DIntersection**(geometry geom1, geometry geom2);

### 描述








#### Warning

**ST\_3DIntersection** is deprecated as of 3.5.0. Use **CG\_3DIntersection** instead.

返回作  $\boxtimes$  geom1 和 geom2 之  $\boxtimes$  共享部分的几何  $\boxtimes$  形。

可用性 : 2.1.0

-   $\boxtimes$  方法需要 SFCGAL 后端。
-   $\boxtimes$  方法  $\boxtimes$  了 SQL/MM  $\boxtimes$  范。SQL-MM IEC 13249-3: 5.1
-   $\boxtimes$  函数支持 3d 并且不会  $\boxtimes$  失 z-index。
-   $\boxtimes$  函数支持多面体曲面。
-  此函数支持三角形和不  $\boxtimes$  三角网面 (TIN)。

## 8.3.12 CG\_3DIntersection

CG\_3DIntersection —  $\boxtimes$  行 3D 相交






### Synopsis

geometry **CG\_3DIntersection**(geometry geom1, geometry geom2);

### 描述

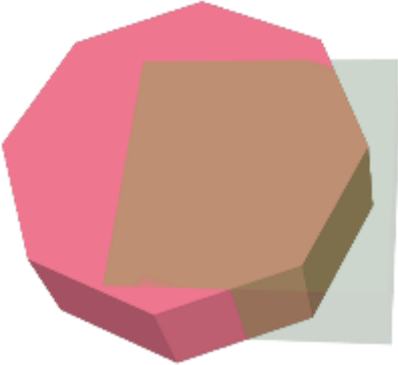
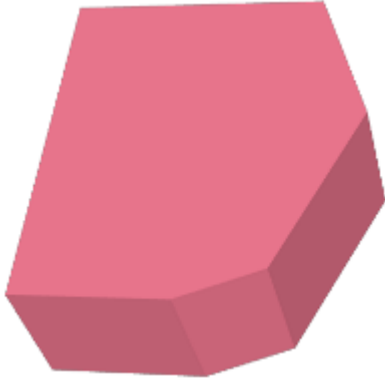
返回作  $\boxtimes$  geom1 和 geom2 之  $\boxtimes$  共享部分的几何  $\boxtimes$  形。

可用性 : 3.5.0

-   $\boxtimes$  方法需要 SFCGAL 后端。
-   $\boxtimes$  方法  $\boxtimes$  了 SQL/MM  $\boxtimes$  范。SQL-MM IEC 13249-3: 5.1
-   $\boxtimes$  函数支持 3d 并且不会  $\boxtimes$  失 z-index。
-   $\boxtimes$  函数支持多面体曲面。
-  此函数支持三角形和不  $\boxtimes$  三角网面 (TIN)。

### 示例

3D  $\boxtimes$  像是使用 **ST\_AsX3D** 生成的, 并使用 **X3Dom HTML Javascript 渲染**  $\boxtimes$  以 HTML 形式渲染。

<pre>SELECT CG_Extrude(ST_Buffer(   ST_GeomFromText('POINT(100 90)'),   CG_Extrude(ST_Buffer(ST_GeomFromText('POINT(80 80)'),     50, 'quad_segs=1'),0,0,30) AS geom2;</pre>  <p>原始 3D 几何形相加。geom2 示半透明</p>	<pre>SELECT CG_3DIntersection(geom1,geom2)   quad_segs=2'),0,0,30) AS geom1,   quad_segs=2'),0,0,30) AS geom1,   CG_Extrude(ST_Buffer(ST_GeomFromText('POINT(80 80)'),     50, 'quad_segs=1'),0,0,30) AS geom2 ) As t;</pre>  <p>geom1 和 geom2 的交集</p>
--	--

3D 串和多形

```
SELECT ST_AsText(CG_3DIntersection(linestring, polygon)) As wkt
FROM ST_GeomFromText('LINESTRING Z (2 2 6,1.5 1.5 7,1 1 8,0.5 0.5 8,0 0 10)') AS linestring
CROSS JOIN ST_GeomFromText('POLYGON((0 0 8, 0 1 8, 1 1 8, 1 0 8, 0 0 8))') AS polygon;

wkt
-----
LINESTRING Z (1 1 8,0.5 0.5 8)
```

立方体（合多面体曲面）和多形 Z

```
SELECT ST_AsText(CG_3DIntersection(
ST_GeomFromText('POLYHEDRALSURFACE Z( ((0 0 0, 0 0 1, 0 1 1, 0 1 0, 0 0 0)),
((0 0 0, 0 1 0, 1 1 0, 1 0 0, 0 0 0)), ((0 0 0, 1 0 0, 1 0 1, 0 0 1, 0 0 0)),
((1 1 0, 1 1 1, 1 0 1, 1 0 0, 1 1 0)),
((0 1 0, 0 1 1, 1 1 1, 1 1 0, 0 1 0)), ((0 0 1, 1 0 1, 1 1 1, 0 1 1, 0 0 1)) )'),
'POLYGON Z ((0 0 0, 0 0 0.5, 0 0.5 0.5, 0 0.5 0, 0 0 0))'::geometry))
```

```
TIN Z (((0 0 0,0 0 0.5,0 0.5 0.5,0 0 0)),((0 0.5 0,0 0 0,0 0.5 0.5,0 0.5 0)))
```

个体的交点也是体 (ST\_Dimension 返回 3)

```
SELECT ST_AsText(CG_3DIntersection( CG_Extrude(ST_Buffer('POINT(10 20)'::geometry,10,1)
,0,0,30),
CG_Extrude(ST_Buffer('POINT(10 20)'::geometry,10,1),2,0,10) ));
```

```
POLYHEDRALSURFACE Z (((13.3333333333333 13.3333333333333 10,20 20 0,20 20
10,13.3333333333333 13.3333333333333 10)),
((20 20 10,16.6666666666667 23.3333333333333 10,13.3333333333333 13.3333333333333
10,20 20 10)),
```

```

((20 20 0,16.6666666666667 23.3333333333333 10,20 20 10,20 20 0)),
((13.3333333333333 13.3333333333333 10,10 10 0,20 20 0,13.3333333333333 ←
  13.3333333333333 10)),
((16.6666666666667 23.3333333333333 10,12 28 10,13.3333333333333 13.3333333333333 ←
  10,16.6666666666667 23.3333333333333 10)),
((20 20 0,9.99999999999995 30 0,16.6666666666667 23.3333333333333 10,20 20 0)),
((10 10 0,9.99999999999995 30 0,20 20 0,10 10 0)),((13.3333333333333 ←
  13.3333333333333 10,12 12 10,10 10 0,13.3333333333333 13.3333333333333 10)),
((12 28 10,12 12 10,13.3333333333333 13.3333333333333 10,12 28 10)),
((16.6666666666667 23.3333333333333 10,9.99999999999995 30 0,12 28 ←
  10,16.6666666666667 23.3333333333333 10)),
((10 10 0,0 20 0,9.99999999999995 30 0,10 10 0)),
((12 12 10,11 11 10,10 10 0,12 12 10)),((12 28 10,11 11 10,12 12 10,12 28 10)),
((9.99999999999995 30 0,11 29 10,12 28 10,9.99999999999995 30 0)),((0 20 0,2 20 ←
  10,9.99999999999995 30 0,0 20 0)),
((10 10 0,2 20 10,0 20 0,10 10 0)),((11 11 10,2 20 10,10 10 0,11 11 10)),((12 28 ←
  10,11 29 10,11 11 10,12 28 10)),
((9.99999999999995 30 0,2 20 10,11 29 10,9.99999999999995 30 0)),((11 11 10,11 29 ←
  10,2 20 10,11 11 10)))

```

### 8.3.13 CG\_Union

CG\_Union — Computes the union of two geometries

#### Synopsis

geometry **CG\_Union**( geometry geomA , geometry geomB );

#### 描述

Computes the union of two geometries.

Performed by the SFCGAL module



#### Note

NOTE: this function returns a geometry representing the union.

可用性 : 3.5.0



方法需要 SFCGAL 后端。

#### 几何示例

```

SELECT CG_Union('POINT(.5 0)', 'LINESTRING(-1 0,1 0)');
      cg_union
-----
LINESTRING(-1 0,0.5 0,1 0)
(1 row)

```

相关信息

[ST\\_3DUnion](#), [ST\\_Union](#)

### 8.3.14 ST\_3DUnion

ST\_3DUnion — 行 3D 合。

#### Synopsis

```
geometry ST_3DUnion(geometry geom1, geometry geom2);
geometry ST_3DUnion(geometry set g1field);
```

描述



#### Warning

**ST\_3DUnion** is deprecated as of 3.5.0. Use **CG\_3DUnion** instead.

可用性 : 2.2.0

可用性 : 3.3.0 添加了聚合函数格式



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)。

聚合形式：返回几何形，几何形是几何形行集的 3D 并集。ST\_3DUnion() 函数是 PostgreSQL 中的“聚合”函数。意味着它对数据行操作，与 SUM() 和 AVG() 函数的操作方式相同，并且与大多数聚合一致，它也会忽略 NULL 几何形。

### 8.3.15 CG\_3DUnion

CG\_3DUnion — Perform 3D union using postgis\_sfcgal.

#### Synopsis

```
geometry CG_3DUnion(geometry geom1, geometry geom2);
geometry CG_3DUnion(geometry set g1field);
```

描述

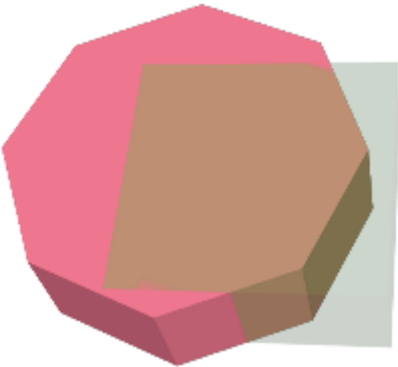
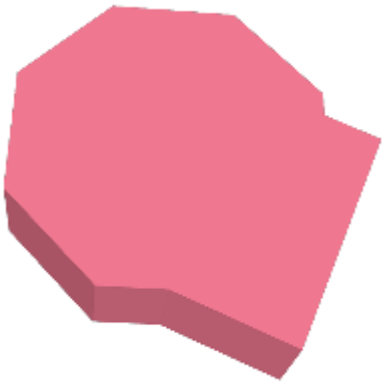
可用性 : 3.5.0

- ✔ 方法需要 SFCGAL 后端。
- ✔ 方法符合了 SQL/MM 规范。SQL-MM IEC 13249-3: 5.1
- ✔ 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)。

**Aggregate variant:** returns a geometry that is the 3D union of a rowset of geometries. The CG\_3DUnion() function is an "aggregate" function in the terminology of PostgreSQL. That means that it operates on rows of data, in the same way the SUM() and AVG() functions do and like most aggregates, it also ignores NULL geometries.

示例

3D 图像是使用 [ST\\_AsX3D](#) 生成的，并使用 [X3Dom HTML Javascript 渲染器](#) 以 HTML 形式渲染。

<pre>SELECT CG_Extrude(ST_Buffer(   ST_GeomFromText('POINT(100 90)'),   50, 'quad_segs=1'),0,0,30) AS geom2;</pre>  <p>原始 3D 几何体相加。geom2 是具有透明度的。</p>	<pre>SELECT CG_3DUnion(geom1,geom2)   AS geom1,   CG_Extrude(ST_Buffer(ST_GeomFromText('POINT(80 80)'),   50, 'quad_segs=1'),0,0,30) AS geom2 )</pre>  <p>geom1 和 geom2 的并集</p>
--	---

相关信息

[CG\\_Extrude](#), [ST\\_AsX3D](#), [CG\\_3DIntersection](#) [CG\\_3DDifference](#)

### 8.3.16 ST\_AlphaShape

ST\_AlphaShape — 计算包络几何体的 Alpha 形状

#### Synopsis

```
geometry ST_AlphaShape(geometry geom, float alpha, boolean allow_holes = false);
```

#### 描述



#### Warning

ST\_AlphaShape is deprecated as of 3.5.0. Use [CG\\_AlphaShape](#) instead.

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。

### 8.3.17 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。

“拟合密度”由 alpha 参数控制，参数的值可以从 0 到无限大。小的 alpha 会产生更凹的结果。大于某些数据相关的 Alpha 值会生成输入的凸包。



#### Note

在 CGAL 3.10 之后，alpha 值是 Alpha-Shape 算法中用于“侵蚀”输入点的 Delaunay 三角剖分的半径的平方。有关更多信息，请参考 [CGAL Alpha 形状](#)。它与 alpha 形状的原始定义不同，后者将 alpha 定义为侵蚀的半径。

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。

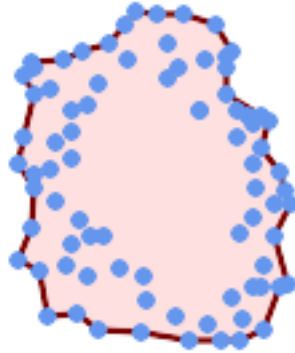
该函数以与 [ST\\_ConcaveHull](#) 类似的方式有效地计算几何体的凹壳，但使用 CGAL 和不同的算法。

Availability: 3.5.0 - requires SFCGAL >= 1.4.1.



该方法需要 SFCGAL 后端。

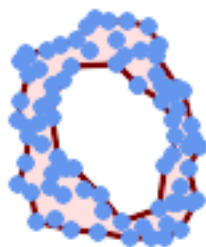
示例



*Alpha-shape of a MultiPoint (same example As [CG\\_OptimalAlphaShape](#))*

```
SELECT ST_AsText(CG_AlphaShape('MULTIPOINT((63 84),(76 88),(68 73),(53 18),(91 50),(81 70),
(88 29),(24 82),(32 51),(37 23),(27 54),(84 19),(75 87),(44 42),(77 67),(90 30) ←
,(36 61),(32 65),
(81 47),(88 58),(68 73),(49 95),(81 60),(87 50),
(78 16),(79 21),(30 22),(78 43),(26 85),(48 34),(35 35),(36 40),(31 79),(83 29) ←
,(27 84),(52 98),(72 95),(85 71),
(75 84),(75 77),(81 29),(77 73),(41 42),(83 72),(23 36),(89 53),(27 57),(57 97) ←
,(27 77),(39 88),(60 81),
(80 72),(54 32),(55 26),(62 22),(70 20),(76 27),(84 35),(87 42),(82 54),(83 64) ←
,(69 86),(60 90),(50 86),(43 80),(36 73),
(36 68),(40 75),(24 67),(23 60),(26 44),(28 33),(40 32),(43 19),(65 16),(73 16) ←
,(38 46),(31 59),(34 86),(45 90),(64 97)')::geometry,80.2));

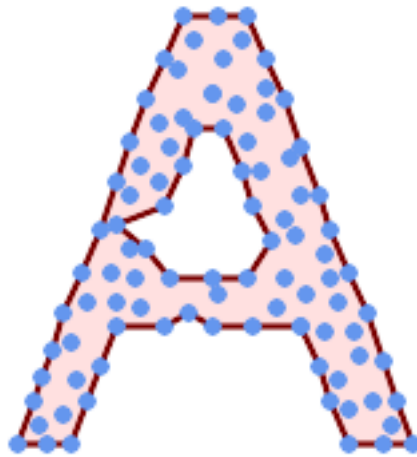
POLYGON((89 53,91 50,87 42,90 30,88 29,84 19,78 16,73 16,65 16,53 18,43 19,
37 23,30 22,28 33,23 36,26 44,27 54,23 60,24 67,27 77,
24 82,26 85,34 86,39 88,45 90,49 95,52 98,57 97,
64 97,72 95,76 88,75 84,83 72,85 71,88 58,89 53))
```



*Alpha-shape of a MultiPoint, allowing holes (same example as [CG\\_OptimalAlphaShape](#))*

```
SELECT ST_AsText(CG_AlphaShape('MULTIPOINT((63 84),(76 88),(68 73),(53 18),(91 50),(81 70) ←
, (88 29),(24 82),(32 51),(37 23),(27 54),(84 19),(75 87),(44 42),(77 67),(90 30),(36 61) ←
, (32 65),(81 47),(88 58),(68 73),(49 95),(81 60),(87 50),
(78 16),(79 21),(30 22),(78 43),(26 85),(48 34),(35 35),(36 40),(31 79),(83 29),(27 84) ←
, (52 98),(72 95),(85 71),
(75 84),(75 77),(81 29),(77 73),(41 42),(83 72),(23 36),(89 53),(27 57),(57 97),(27 77) ←
, (39 88),(60 81),
(80 72),(54 32),(55 26),(62 22),(70 20),(76 27),(84 35),(87 42),(82 54),(83 64),(69 86) ←
, (60 90),(50 86),(43 80),(36 73),
(36 68),(40 75),(24 67),(23 60),(26 44),(28 33),(40 32),(43 19),(65 16),(73 16),(38 46) ←
, (31 59),(34 86),(45 90),(64 97))'::geometry, 100.1,true))
```

```
POLYGON((89 53,91 50,87 42,90 30,84 19,78 16,73 16,65 16,53 18,43 19,30 22,28 33,23 36,
26 44,27 54,23 60,24 67,27 77,24 82,26 85,34 86,39 88,45 90,49 95,52 98,57 97,64 97,72 95,
76 88,75 84,83 72,85 71,88 58,89 53),(36 61,36 68,40 75,43 80,60 81,68 73,77 67,
81 60,82 54,81 47,78 43,76 27,62 22,54 32,44 42,38 46,36 61))
```



*MultiPoint* 的 *Alpha* 形状，允许有孔（与 *ST\_ConcaveHull* 相同的示例）

```
SELECT ST_AsText(CG_AlphaShape(
'MULTIPOINT ((132 64), (114 64), (99 64), (81 64), (63 64), (57 49), (52 ←
36), (46 20), (37 20), (26 20), (32 36), (39 55), (43 69), (50 84), (57 ←
100), (63 118), (68 133), (74 149), (81 164), (88 180), (101 180), (112 ←
180), (119 164), (126 149), (132 131), (139 113), (143 100), (150 84), ←
(157 69), (163 51), (168 36), (174 20), (163 20), (150 20), (143 36), ←
(139 49), (132 64), (99 151), (92 138), (88 124), (81 109), (74 93), (70 ←
82), (83 82), (99 82), (112 82), (126 82), (121 96), (114 109), (110 ←
122), (103 138), (99 151), (34 27), (43 31), (48 44), (46 58), (52 73), ←
(63 73), (61 84), (72 71), (90 69), (101 76), (123 71), (141 62), (166 ←
27), (150 33), (159 36), (146 44), (154 53), (152 62), (146 73), (134 ←
76), (143 82), (141 91), (130 98), (126 104), (132 113), (128 127), (117 ←
122), (112 133), (119 144), (108 147), (119 153), (110 171), (103 164), ←
(92 171), (86 160), (88 142), (79 140), (72 124), (83 131), (79 118), ←
(68 113), (63 102), (68 93), (35 45))'::geometry,102.2, true));
```

```
POLYGON((26 20,32 36,35 45,39 55,43 69,50 84,57 100,63 118,68 133,74 149,81 164,88 180,
101 180,112 180,119 164,126 149,132 131,139 113,143 100,150 84,157 69,163 ←
51,168 36,
174 20,163 20,150 20,143 36,139 49,132 64,114 64,99 64,90 69,81 64,63 64,57 ←
49,52 36,46 20,37 20,26 20),
```



(74 93,81 109,88 124,92 138,103 138,110 122,114 109,121 96,112 82,99 82,83 82,74 93)) ←

相关信息

[ST\\_ConcaveHull](#), [CG\\_OptimalAlphaShape](#)

### 8.3.18 CG\_ApproxConvexPartition

CG\_ApproxConvexPartition — 计算多边形几何体的近似凸分割

#### Synopsis

```
geometry CG_ApproxConvexPartition(geometry geom);
```

描述

计算多边形几何体的近似凸分割（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小的多边形来生成凸分解；CG\_ApproxConvexPartition 使用三角剖分，而 CG\_GreeneApproxConvexPartition 使用一个平面分割。这两个函数都保证它们生成的凸片段数不会超过最佳数量的四倍，但它们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。

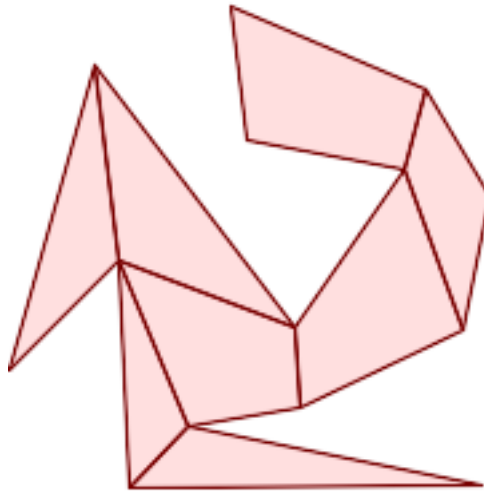
可用性：3.5.0 - 需要 SFCGAL  $\geq$  1.5.0。

需要 SFCGAL 版本  $\geq$  1.5.0



该方法需要 SFCGAL 后端。

示例



近似凸分割（与 [CG\\_YMonotonePartition](#)、[CG\\_GreeneApproxConvexPartition](#) 和 [CG\\_OptimalConvexPartition](#) 相同的示例）

```
SELECT ST_AsText(CG_ApproxConvexPartition('POLYGON((156 150,83 181,89 131,148 120,107 61,32 ←
159,0 45,41 86,45 1,177 2,67 24,109 31,170 60,180 110,156 150))'::geometry));
```

```
GEOMETRYCOLLECTION(POLYGON((156 150,83 181,89 131,148 120,156 150)),POLYGON((32 159,0 45,41 ←
86,32 159)),POLYGON((107 61,32 159,41 86,107 61)),POLYGON((45 1,177 2,67 24,45 1)), ←
POLYGON((41 86,45 1,67 24,41 86)),POLYGON((107 61,41 86,67 24,109 31,107 61)),POLYGON ←
((148 120,107 61,109 31,170 60,148 120)),POLYGON((156 150,148 120,170 60,180 110,156 ←
150)))
```

相关信息

[CG\\_YMonotonePartition](#), [CG\\_GreeneApproxConvexPartition](#), [CG\\_OptimalConvexPartition](#)

### 8.3.19 ST\_ApproximateMedialAxis

ST\_ApproximateMedialAxis — 计算几何区域的近似中轴。

#### Synopsis

```
geometry ST_ApproximateMedialAxis(geometry geom);
```

描述



#### Warning

[ST\\_ApproximateMedialAxis](#) is deprecated as of 3.5.0. Use [CG\\_ApproximateMedialAxis](#) instead.

Return an approximate medial axis for the areal input based on its straight skeleton. Uses an SFCGAL specific API when built against a capable version (1.2.0+). Otherwise the function is just a wrapper around `CG_StraightSkeleton` (slower case).

可用性 : 2.2.0

- ✔ 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)。

### 8.3.20 CG\_ApproximateMedialAxis

`CG_ApproximateMedialAxis` — 算几何区域的近似中。

#### Synopsis

```
geometry CG_ApproximateMedialAxis(geometry geom);
```

#### 描述

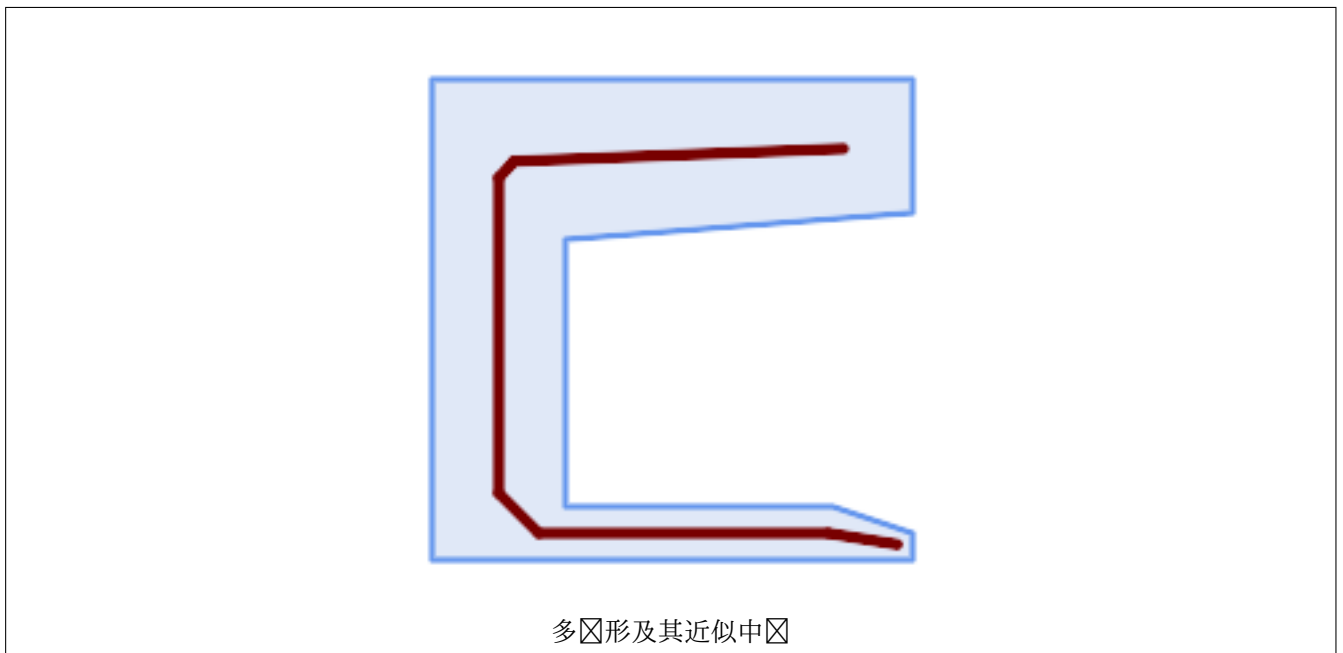
Return an approximate medial axis for the areal input based on its straight skeleton. Uses an SFCGAL specific API when built against a capable version (1.2.0+). Otherwise the function is just a wrapper around `CG_StraightSkeleton` (slower case).

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

```
SELECT CG_ApproximateMedialAxis(ST_GeomFromText('POLYGON (( 190 190, 10 190, 10 10, 190 10, ↵  
190 20, 160 30, 60 30, 60 130, 190 140, 190 190 ))'));
```



相关信息

[CG\\_StraightSkeleton](#)

### 8.3.21 ST\_ConstrainedDelaunayTriangles

ST\_ConstrainedDelaunayTriangles — 返回指定几何体的约束 Delaunay 三角剖分。

#### Synopsis

```
geometry ST_ConstrainedDelaunayTriangles(geometry g1);
```

描述



#### Warning

`ST_ConstrainedDelaunayTriangles` is deprecated as of 3.5.0. Use `CG_ConstrainedDelaunayTriangles` instead.

返回指定几何体点的约束 Delaunay 三角剖分。输出是 TIN。



该方法需要 SFCGAL 后端。

可用性：3.0.0



该函数支持 3d 并且不会丢失 z-index。

### 8.3.22 CG\_ConstrainedDelaunayTriangles

CG\_ConstrainedDelaunayTriangles — 返回指定几何体的约束 Delaunay 三角剖分。

### Synopsis

geometry **CG\_ConstrainedDelaunayTriangles**(geometry g1);

### 描述

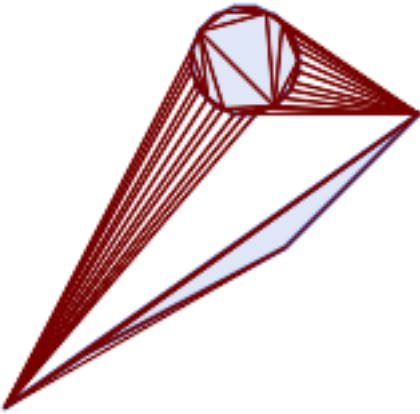
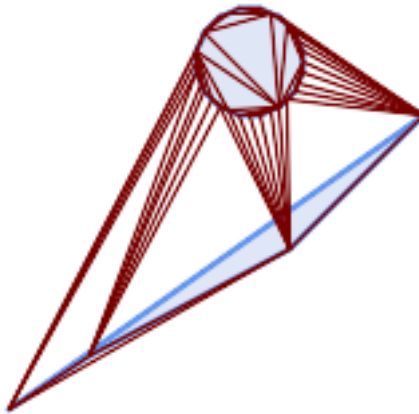
返回输入几何体点的 **约束 Delaunay 三角剖分**。输出是 TIN。

✓ 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

 <p><i>CG_ConstrainedDelaunayTriangles of 2 polygons</i></p> <pre> select CG_ConstrainedDelaunayTriangles(     ST_Union(         POLYGON((175 150, 20 40, 50 60, 125 100, 175 150)),         ST_Buffer('POINT(110 170)::geometry, 20)     ) ); </pre>	 <p><i>ST_DelaunayTriangles</i> 2 个多边形。三角形与多边形边界相交。</p> <pre> select ST_DelaunayTriangles(     ST_Union(         POLYGON((175 150, 20 40, 50 60, 125 100, 175 150)),         ST_Buffer('POINT(110 170)::geometry, 20)     ) ); </pre>
---	--

### 相关信息

[ST\\_DelaunayTriangles](#), [ST\\_TriangulatePolygon](#), [CG\\_Tesselate](#), [ST\\_ConcaveHull](#), [ST\\_Dump](#)

### 8.3.23 ST\_Extrude

ST\_Extrude — 将曲面 $\square$ 出到相关体 $\square$

#### Synopsis

geometry **ST\_Extrude**(geometry geom, float x, float y, float z);





描述



#### Warning

**ST\_Extrude** is deprecated as of 3.5.0. Use **CG\_Extrude** instead.

可用性 : 2.1.0

-   $\square$ 方法需要 SFCGAL 后端。
-   $\square$ 函数支持 3d 并且不会 $\square$ 失 z-index。
-   $\square$ 函数支持多面体曲面。
-  此函数支持三角形和不 $\square$  $\square$ 三角网面 (TIN)。

### 8.3.24 CG\_Extrude





CG\_Extrude — 将曲面 $\square$ 出到相关体 $\square$

#### Synopsis

geometry **CG\_Extrude**(geometry geom, float x, float y, float z);




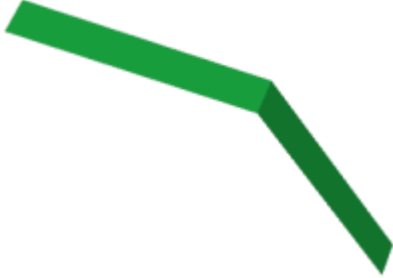
描述

可用性 : 3.5.0

-   $\square$ 方法需要 SFCGAL 后端。
-   $\square$ 函数支持 3d 并且不会 $\square$ 失 z-index。
-   $\square$ 函数支持多面体曲面。
-  此函数支持三角形和不 $\square$  $\square$ 三角网面 (TIN)。

示例

3D  $\square$ 像是使用 **ST\_AsX3D**生成的, 并使用**X3Dom HTML Javascript 渲染 $\square$** 以 HTML 形式渲染。

<pre>SELECT ST_Buffer(ST_GeomFromText('POINT (100 90)'),     50, 'quad_segs=2'),0,0,30);</pre>  <p>由点缓冲形成的原始八边形</p>	<pre>CG_Extrude(ST_Buffer(ST_GeomFromText('POINT (100 90)'),     50, 'quad_segs=2'),0,0,30);</pre>  <p>六边形沿 Z 方向拉伸 30 个单位生成多面体表面 Z</p>
<pre>SELECT ST_GeomFromText('LINESTRING(50 50, 100 90, 95 150)')</pre>  <p>原始线串</p>	<pre>SELECT CG_Extrude(     ST_GeomFromText('LINESTRING(50 50, 100 90, 95 150)')</pre>  <p>沿 Z 方向拉伸的 LineString 会生成 PolyhedralSurfaceZ</p>

相关信息

[ST\\_AsX3D](#), [CG\\_ExtrudeStraightSkeleton](#)

### 8.3.25 CG\_ExtrudeStraightSkeleton

CG\_ExtrudeStraightSkeleton — 直 $\square$ 骨架外凸

#### Synopsis

geometry **CG\_ExtrudeStraightSkeleton**(geometry geom, float roof\_height, float body\_height = 0);

#### 描述

$\square$ 算多 $\square$ 形几何 $\square$ 形的最大高度拉伸。

#### Note



Perhaps the first (historically) use-case of straight skeletons: given a polygonal roof, the straight skeleton directly gives the layout of each tent. If each skeleton edge is lifted from the plane a height equal to its offset distance, the resulting roof is "correct" in that water will always fall down to the contour edges (the roof's border), regardless of where it falls on the roof. The function computes this extrusion aka "roof" on a polygon. If the argument body\_height > 0, so the polygon is extruded like with CG\_Extrude(polygon, 0, 0, body\_height). The result is an union of these polyhedralsurfaces.

可用性 : 3.5.0 - 需要 SFCGAL >= 1.5.0。

需要 SFCGAL 版本 >= 1.5.0



$\square$ 方法需要 SFCGAL 后端。

#### 示例

```
SELECT ST_AsText(CG_ExtrudeStraightSkeleton('POLYGON (( 0 0, 5 0, 5 5, 4 5, 4 4, 0 4, 0 0 ) ←
, (1 1, 1 2, 2 2, 2 1, 1 1))', 3.0, 2.0));
```

```
POLYHEDRALSURFACE Z (((0 0 0,0 4 0,4 4 0,4 5 0,5 5 0,5 0 0,0 0 0),(1 1 0,2 1 0,2 2 0,1 2 ←
0,1 1 0)),((0 0 0,0 0 2,0 4 2,0 4 0,0 0 0)),((0 4 0,0 4 2,4 4 2,4 4 0,0 4 0)),((4 4 0,4 ←
4 2,4 5 2,4 5 0,4 4 0)),((4 5 0,4 5 2,5 5 2,5 5 0,4 5 0)),((5 5 0,5 5 2,5 0 2,5 0 0,5 5 ←
0)),((5 0 0,5 0 2,0 0 2,0 0 0,5 0 0)),((1 1 0,1 1 2,2 1 2,2 1 0,1 1 0)),((2 1 0,2 1 2,2 ←
2 2,2 2 0,2 1 0)),((2 2 0,2 2 2,1 2 2,1 2 0,2 2 0)),((1 2 0,1 2 2,1 1 2,1 1 0,1 2 0) ←
,((4 5 2,5 5 2,4 4 2,4 5 2)),((2 1 2,5 0 2,0 0 2,2 1 2)),((5 5 2,5 0 2,4 4 2,5 5 2)),((2 ←
1 2,0 0 2,1 1 2,2 1 2)),((1 2 2,1 1 2,0 0 2,1 2 2)),((0 4 2,2 2 2,1 2 2,0 4 2)),((0 4 ←
2,1 2 2,0 0 2,0 4 2)),((4 4 2,5 0 2,2 2 2,4 4 2)),((4 4 2,2 2 2,0 4 2,4 4 2)),((2 2 2,5 ←
0 2,2 1 2,2 2 2)),((0.5 2.5 2.5,0 0 2,0.5 0.5 2.5,0.5 2.5 2.5)),((1 3 3,0 4 2,0.5 2.5 ←
2.5,1 3 3)),((0.5 2.5 2.5,0 4 2,0 0 2,0.5 2.5 2.5)),((2.5 0.5 2.5,5 0 2,3.5 1.5 3.5,2.5 ←
0.5 2.5)),((0 0 2,5 0 2,2.5 0.5 2.5,0 0 2)),((0.5 0.5 2.5,0 0 2,2.5 0.5 2.5,0.5 0.5 2.5) ←
),((4.5 3.5 2.5,5 2,4.5 4.5 2.5,4.5 3.5 2.5)),((3.5 2.5 3.5,3.5 1.5 3.5,4.5 3.5 ←
2.5,3.5 2.5 3.5)),((4.5 3.5 2.5,5 0 2,5 2,4.5 3.5 2.5)),((3.5 1.5 3.5,5 0 2,4.5 3.5 ←
2.5,3.5 1.5 3.5)),((5 5 2,4 5 2,4.5 4.5 2.5,5 5 2)),((4.5 4.5 2.5,4 4 2,4.5 3.5 2.5,4.5 ←
4.5 2.5)),((4.5 4.5 2.5,4 5 2,4 4 2,4.5 4.5 2.5)),((3 3 3,0 4 2,1 3 3,3 3 3)),((3.5 2.5 ←
3.5,4.5 3.5 2.5,3 3 3,3.5 2.5 3.5)),((3 3 3,4 4 2,0 4 2,3 3 3)),((4.5 3.5 2.5,4 4 2,3 3 ←
3,4.5 3.5 2.5)),((2 1 2,1 1 2,0.5 0.5 2.5,2 1 2)),((2.5 0.5 2.5,2 1 2,0.5 0.5 2.5,2.5 ←
0.5 2.5)),((1 1 2,1 2 2,0.5 2.5 2.5,1 1 2)),((0.5 0.5 2.5,1 1 2,0.5 2.5 2.5,0.5 0.5 2.5) ←
),((1 3 3,2 2 2,3 3 3,1 3 3)),((0.5 2.5 2.5,1 2 2,1 3 3,0.5 2.5 2.5)),((1 3 3,1 2 2,2 2 ←
2,1 3 3)),((2 2 2,2 1 2,2.5 0.5 2.5,2 2 2)),((3.5 2.5 3.5,3 3 3,3.5 1.5 3.5,3.5 2.5 3.5) ←
),((3.5 1.5 3.5,2 2 2,2.5 0.5 2.5,3.5 1.5 3.5)),((3 3 3,2 2 2,3.5 1.5 3.5,3 3 3)))
```



相关信息

[ST\\_Extrude](#), [CG\\_StraightSkeleton](#)

### 8.3.26 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，但它在并行重复性上有所不同。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。

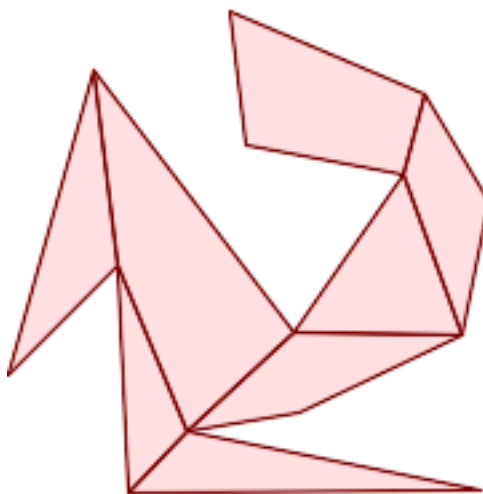
可用性：3.5.0 - 需要 SFCGAL >= 1.5.0。

需要 SFCGAL 版本 >= 1.5.0



该方法需要 SFCGAL 后端。

#### 示例



格林近似凸分割（与 [CG\\_YMonotonePartition](#)、[CG\\_ApproxConvexPartition](#) 和 [CG\\_OptimalConvexPartition](#) 相同的示例）

```
SELECT ST_AsText(CG_GreeneApproxConvexPartition('POLYGON((156 150,83 181,89 131,148 120,107 61,32 159,0 45,41 86,45 1,177 2,67 24,109 31,170 60,180 110,156 150))':::geometry));
```

```
GEOMETRYCOLLECTION(POLYGON((32 159,0 45,41 86,32 159)),POLYGON((45 1,177 2,67 24,45 1)), ←
POLYGON((67 24,109 31,170 60,107 61,67 24)),POLYGON((41 86,45 1,67 24,41 86)),POLYGON ←
((107 61,32 159,41 86,67 24,107 61)),POLYGON((148 120,107 61,170 60,148 120)),POLYGON ←
((148 120,170 60,180 110,156 150,148 120)),POLYGON((156 150,83 181,89 131,148 120,156 ←
150)))
```

相关信息

[CG\\_YMonotonePartition](#), [CG\\_ApproxConvexPartition](#), [CG\\_OptimalConvexPartition](#)

### 8.3.27 ST\_MinkowskiSum

ST\_MinkowskiSum — 行 Minkowski sum

#### Synopsis

geometry **ST\_MinkowskiSum**(geometry geom1, geometry geom2);

描述



#### Warning

**ST\_MinkowskiSum** is deprecated as of 3.5.0. Use **CG\_MinkowskiSum** instead.

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 和常用于划和计算机助。有关更多信息在[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 Minkowskism](#)。

可用性：2.1.0



方法需要 SFCGAL 后端。

### 8.3.28 CG\_MinkowskiSum

CG\_MinkowskiSum — 行 Minkowski sum

#### Synopsis

geometry **CG\_MinkowskiSum**(geometry geom1, geometry geom2);

## 描述

此函数返回点、或多边形与多边形的 2D minkowski 和。

两个几何图形 A 和 B 的 minkowski 和是所有点的集合，这些点是 A 和 B 中任意点的和。minkowski 和常用于规划和计算机辅助设计。有关更多信息在[维基百科 Minkowski 添加](#)。

第一个参数可以是任何 2D 几何图形（点、串、多边形）。如果输入 3D 几何图形，会通过限制 Z = 0 将其强制 2D，从而可能导致无效的情况。第二个参数必须是二维多边形。

利用 [CGAL 2D MinkowskiSum](#)。

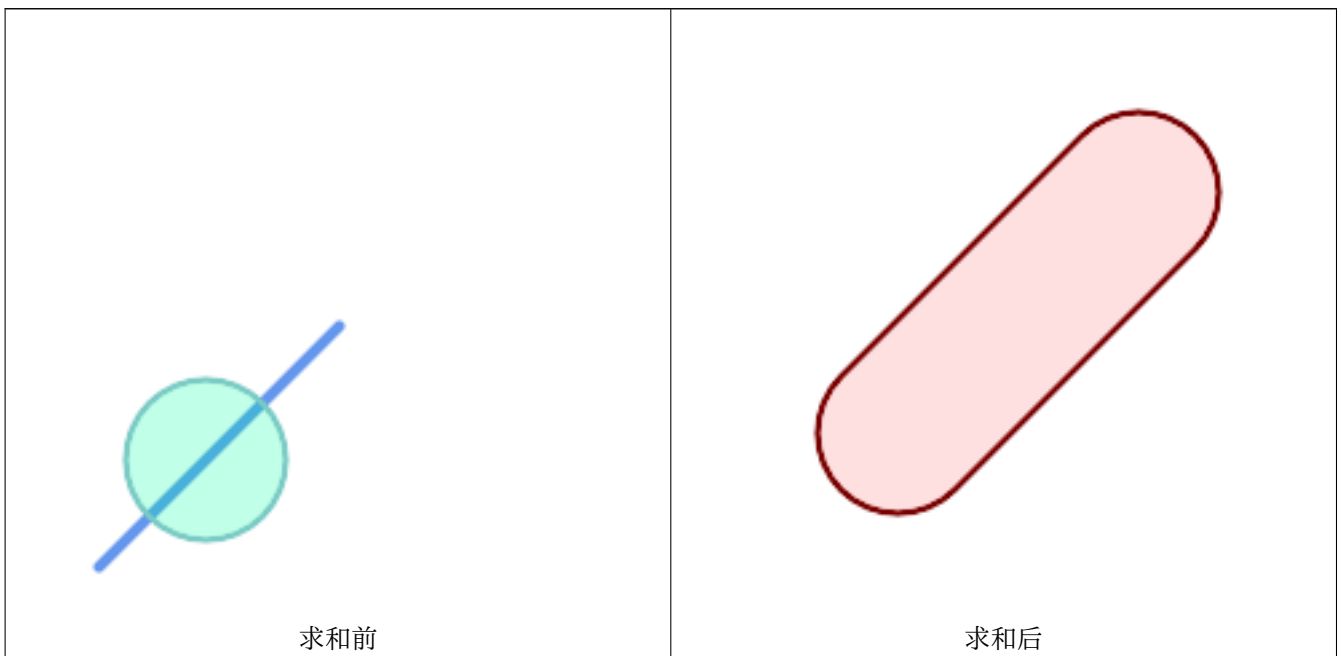
可用性：3.5.0



方法需要 SFCGAL 后端。

## 示例

Minkowski 串与多边形的和，其中串穿圆



```
SELECT CG_MinkowskiSum(line, circle)
FROM (SELECT
  ST_MakeLine(ST_Point(10, 10),ST_Point(100, 100)) As line,
  ST_Buffer(ST_GeomFromText('POINT(50 50)'), 30) As circle) As foo;

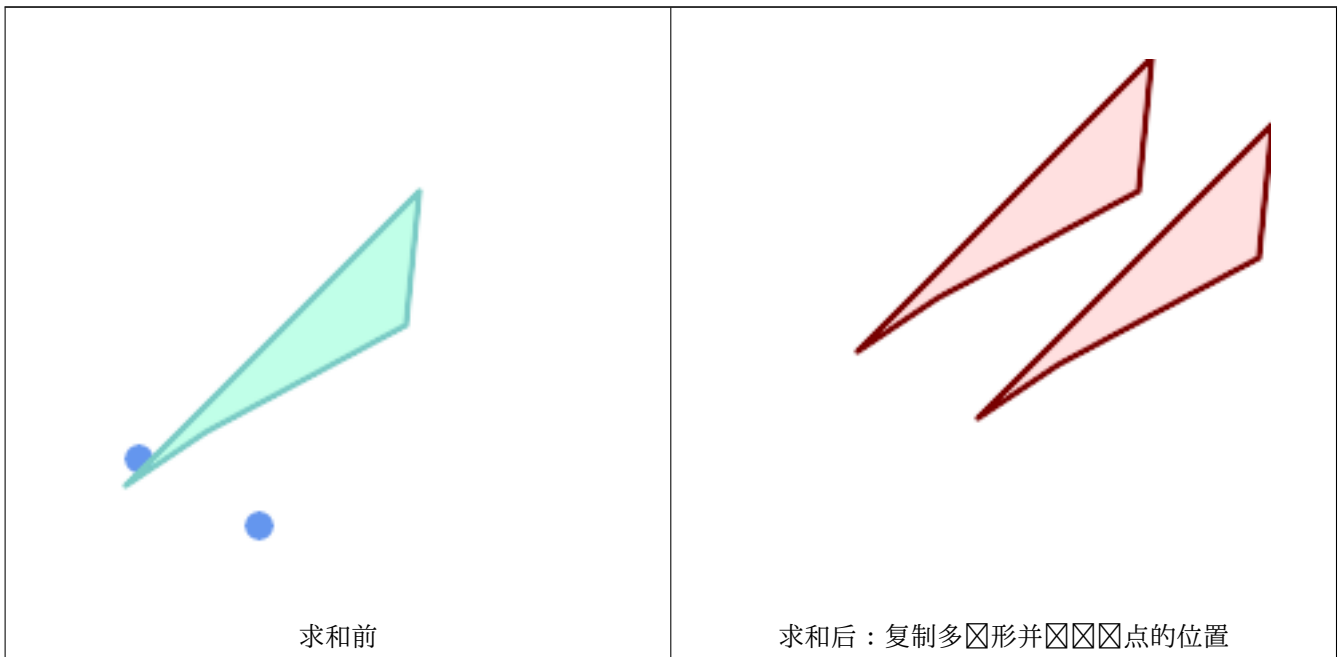
-- wkt --
MULTIPOLYGON(((30 59.9999999999999,30.5764415879031 ↵
  54.1472903395161,32.2836140246614 48.5194970290472,35.0559116309237 ↵
  43.3328930094119,38.7867965644036 38.7867965644035,43.332893009412 ↵
  35.0559116309236,48.5194970290474 32.2836140246614,54.1472903395162 ↵
  30.5764415879031,60.0000000000001 30,65.8527096604839 ↵
  30.5764415879031,71.4805029709527 32.2836140246614,76.6671069905881 ↵
  35.0559116309237,81.2132034355964 38.7867965644036,171.213203435596 ↵
  128.786796564404,174.944088369076 133.332893009412,177.716385975339 ↵
  138.519497029047,179.423558412097 144.147290339516,180 150,179.423558412097 ↵
  155.852709660484,177.716385975339 161.480502970953,174.944088369076 ↵
  166.667106990588,171.213203435596 171.213203435596,166.667106990588 ↵
```

```

174.944088369076,
161.480502970953 177.716385975339,155.852709660484 179.423558412097,150 ↔
180,144.147290339516 179.423558412097,138.519497029047 ↔
177.716385975339,133.332893009412 174.944088369076,128.786796564403 ↔
171.213203435596,38.7867965644035 81.2132034355963,35.0559116309236 ↔
76.667106990588,32.2836140246614 71.4805029709526,30.5764415879031 ↔
65.8527096604838,30 59.9999999999999))

```

多边形和多点的 Minkowski 和



```

SELECT CG_MinkowskiSum(mp, poly)
FROM (SELECT 'MULTIPOINT(25 50,70 25)::geometry As mp,
'POLYGON((130 150, 20 40, 50 60, 125 100, 130 150))::geometry As poly
) As foo

-- wkt --
MULTIPOLYGON(
((70 115,100 135,175 175,225 225,70 115)),
((120 65,150 85,225 125,275 175,120 65))
)

```

### 8.3.29 ST\_OptimalAlphaShape

ST\_OptimalAlphaShape — 使用“最佳”alpha 值计算包围几何体的 Alpha 形状。

#### Synopsis

```

geometry ST_OptimalAlphaShape(geometry geom, boolean allow_holes = false, integer nb_components
= 1);

```

描述



### Warning

**ST\_OptimalAlphaShape** is deprecated as of 3.5.0. Use **CG\_OptimalAlphaShape** instead.

计算几何中点的“最佳” alpha 形状。alpha 形状是使用所给的  $\alpha$  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

除非将可选的 `allow_holes` 参数指定为 `true`，否则结果将不包含孔。

可用性：3.3.0 - 需要 SFCGAL  $\geq$  1.4.1。



该方法需要 SFCGAL 后端。

## 8.3.30 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  $\alpha$  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

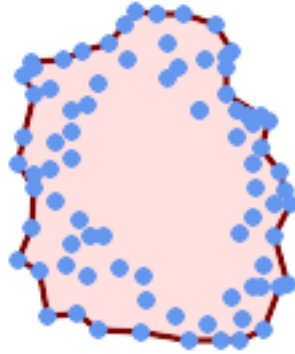
除非将可选的 `allow_holes` 参数指定为 `true`，否则结果将不包含孔。

Availability: 3.5.0 - requires SFCGAL  $\geq$  1.4.1.



该方法需要 SFCGAL 后端。

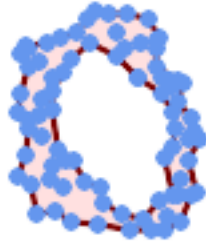
示例



*Optimal alpha-shape of a MultiPoint (same example as [CG\\_AlphaShape](#))*

```
SELECT ST_AsText(CG_OptimalAlphaShape('MULTIPOINT((63 84),(76 88),(68 73),(53 18),(91 50) ←
    ,(81 70),
    (88 29),(24 82),(32 51),(37 23),(27 54),(84 19),(75 87),(44 42),(77 67),(90 ←
    30),(36 61),(32 65),
    (81 47),(88 58),(68 73),(49 95),(81 60),(87 50),
    (78 16),(79 21),(30 22),(78 43),(26 85),(48 34),(35 35),(36 40),(31 79),(83 ←
    29),(27 84),(52 98),(72 95),(85 71),
    (75 84),(75 77),(81 29),(77 73),(41 42),(83 72),(23 36),(89 53),(27 57),(57 ←
    97),(27 77),(39 88),(60 81),
    (80 72),(54 32),(55 26),(62 22),(70 20),(76 27),(84 35),(87 42),(82 54),(83 ←
    64),(69 86),(60 90),(50 86),(43 80),(36 73),
    (36 68),(40 75),(24 67),(23 60),(26 44),(28 33),(40 32),(43 19),(65 16),(73 ←
    16),(38 46),(31 59),(34 86),(45 90),(64 97))'::geometry));

POLYGON((89 53,91 50,87 42,90 30,88 29,84 19,78 16,73 16,65 16,53 18,43 19,37 23,30 22,28 ←
    33,23 36,
    26 44,27 54,23 60,24 67,27 77,24 82,26 85,34 86,39 88,45 90,49 95,52 98,57 ←
    97,64 97,72 95,76 88,75 84,75 77,83 72,85 71,83 64,88 58,89 53))
```



*Optimal alpha-shape of a MultiPoint, allowing holes (same example as [CG\\_AlphaShape](#))*

```
SELECT ST_AsText(CG_OptimalAlphaShape('MULTIPOINT((63 84),(76 88),(68 73),(53 18),(91 50) ←
, (81 70),(88 29),(24 82),(32 51),(37 23),(27 54),(84 19),(75 87),(44 42),(77 67),(90 30) ←
, (36 61),(32 65),(81 47),(88 58),(68 73),(49 95),(81 60),(87 50),
(78 16),(79 21),(30 22),(78 43),(26 85),(48 34),(35 35),(36 40),(31 79),(83 29),(27 ←
84),(52 98),(72 95),(85 71),
(75 84),(75 77),(81 29),(77 73),(41 42),(83 72),(23 36),(89 53),(27 57),(57 97),(27 ←
77),(39 88),(60 81),
(80 72),(54 32),(55 26),(62 22),(70 20),(76 27),(84 35),(87 42),(82 54),(83 64),(69 ←
86),(60 90),(50 86),(43 80),(36 73),
(36 68),(40 75),(24 67),(23 60),(26 44),(28 33),(40 32),(43 19),(65 16),(73 16),(38 ←
46),(31 59),(34 86),(45 90),(64 97))'::geometry, allow_holes => true));
```

```
POLYGON((89 53,91 50,87 42,90 30,88 29,84 19,78 16,73 16,65 16,53 18,43 19,37 23,30 22,28 ←
33,23 36,26 44,27 54,23 60,24 67,27 77,24 82,26 85,34 86,39 88,45 90,49 95,52 98,57 ←
97,64 97,72 95,76 88,75 84,75 77,83 72,85 71,83 64,88 58,89 53),(36 61,36 68,40 75,43 ←
80,50 86,60 81,68 73,77 67,81 60,82 54,81 47,78 43,81 29,76 27,70 20,62 22,55 26,54 ←
32,48 34,44 42,38 46,36 61))
```

相关信息

[ST\\_ConcaveHull](#), [CG\\_AlphaShape](#)

### 8.3.31 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\_OptimalConvexPartition 生成的划分在片段数量上是最优的。

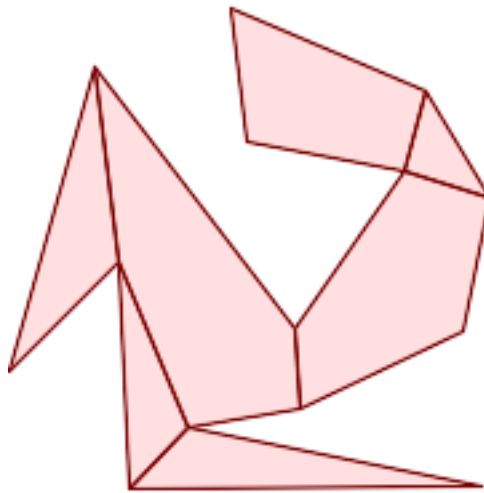
可用性：3.5.0 - 需要 SFCGAL >= 1.5.0。

需要 SFCGAL 版本 >= 1.5.0



该方法需要 SFCGAL 后端。

示例



最优凸分割（与 [CG\\_YMonotonePartition](#)、[CG\\_ApproxConvexPartition](#) 和 [CG\\_GreeneApproxConvexPartition](#) 相同的示例）

```
SELECT ST_AsText(CG_OptimalConvexPartition('POLYGON((156 150,83 181,89 131,148 120,107 61,32 159,0 45,41 86,45 1,177 2,67 24,109 31,170 60,180 110,156 150))'::geometry));

GEOMETRYCOLLECTION(POLYGON((156 150,83 181,89 131,148 120,156 150)),POLYGON((32 159,0 45,41 86,32 159)),POLYGON((45 1,177 2,67 24,45 1)),POLYGON((41 86,45 1,67 24,41 86)),POLYGON
((107 61,32 159,41 86,67 24,109 31,107 61)),POLYGON((148 120,107 61,109 31,170 60,180 110,148 120)),POLYGON((156 150,148 120,180 110,156 150)))
```

相关信息

[CG\\_YMonotonePartition](#), [CG\\_ApproxConvexPartition](#), [CG\\_GreeneApproxConvexPartition](#)

### 8.3.32 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

可用性 : 3.5.0

Requires SFCGAL >= 1.3.8 for option use\_distance\_as\_m

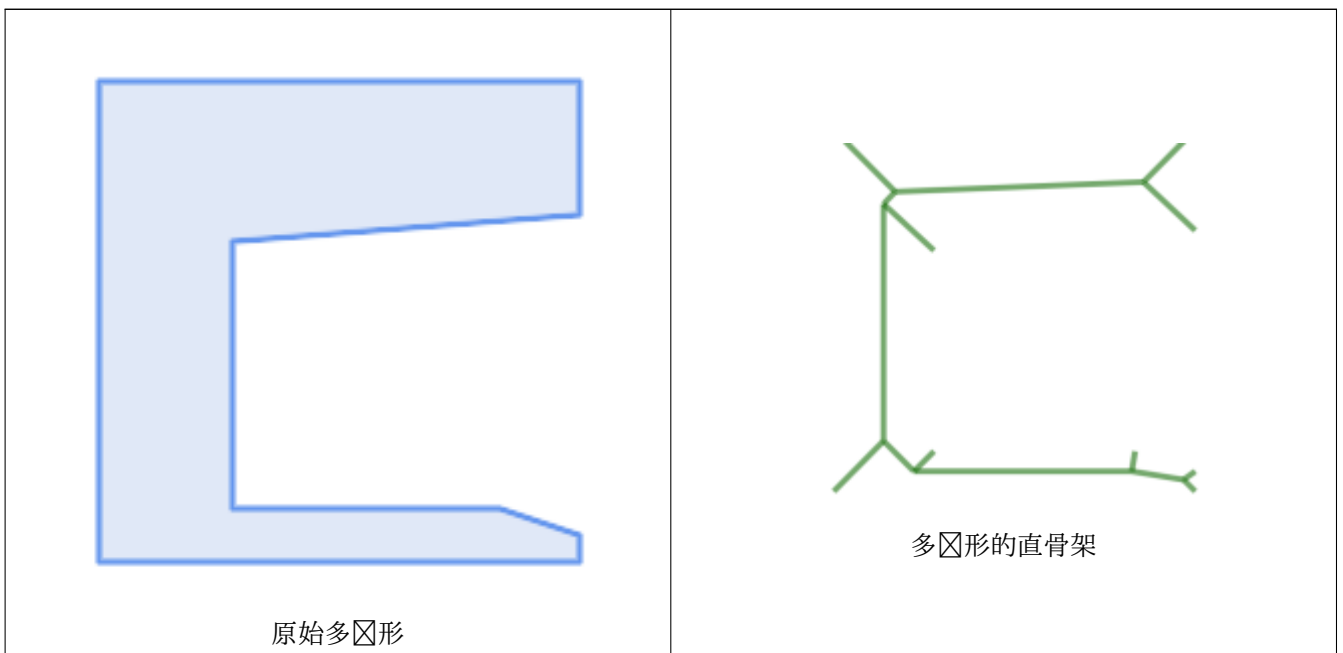
- ✔ ☒ 方法需要 SFCGAL 后端。
- ✔ ☒ 函数支持 3d 并且不会☒失 z-index。
- ✔ ☒ 函数支持多面体曲面。
- ✔ ☒ 此函数支持三角形和☒☒三角网面 (TIN)。

## 示例

```
SELECT CG_StraightSkeleton(ST_GeomFromText('POLYGON (( 190 190, 10 190, 10 10, 190 10, 190 20, 160 30, 60 30, 60 130, 190 140, 190 190 ))'));
```

```
ST_AsText(CG_StraightSkeleton('POLYGON((0 0,1 0,1 1,0 1,0 0))', true);
```

```
MULTILINESTRING M ((0 0 0,0.5 0.5 0.5),(1 0 0,0.5 0.5 0.5),(1 1 0,0.5 0.5 0.5),(0 1 0,0.5 0.5 0.5))
```



## 相关信息

[CG\\_ExtrudeStraightSkeleton](#)

### 8.3.33 ST\_StraightSkeleton

ST\_StraightSkeleton — 从几何体☒算直骨架

## Synopsis

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

描述



### Warning

**ST\_StraightSkeleton** is deprecated as of 3.5.0. Use **CG\_StraightSkeleton** instead.

可用性 : 2.1.0



☑方法需要 SFCGAL 后端。



☑函数支持 3d 并且不会☑失 z-index。



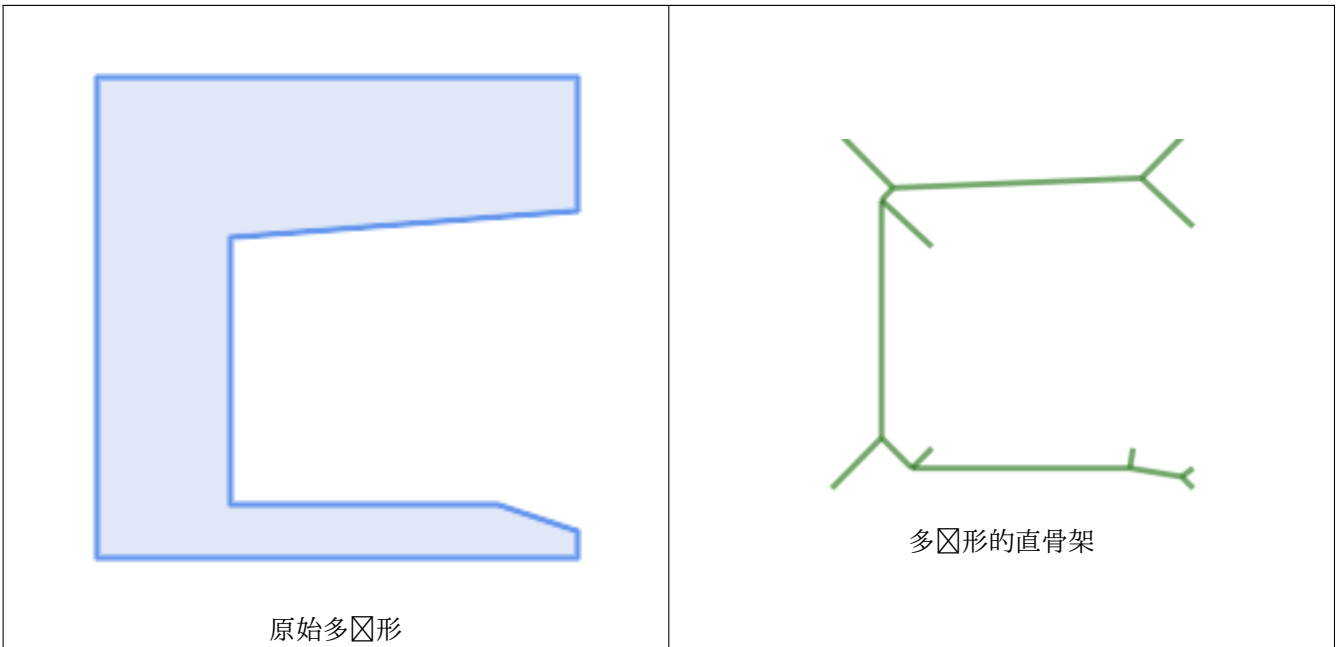
☑函数支持多面体曲面。



☑此函数支持三角形和不☑☑三角网面 (TIN)。

示例

```
SELECT ST_StraightSkeleton(ST_GeomFromText('POLYGON (( 190 190, 10 190, 10 10, 190 10, 190 20, 160 30, 60 30, 60 130, 190 140, 190 190 ))'));
```



相关信息

[CG\\_ExtrudeStraightSkeleton](#)

### 8.3.34 ST\_Tesselate

ST\_Tesselate — 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

#### Synopsis

```
geometry ST_Tesselate(geometry geom);
```

#### 描述



#### Warning

**ST\_Tesselate** is deprecated as of 3.5.0. Use **CG\_Tesselate** instead.

将 MULTI(POLYGON) 或 POLYHEDRALSURFACE 等曲面作入，并通过使用三角形的细分过程返回 TIN 表示形式。



#### Note

**ST\_TriangulatePolygon** 与此函数类似，但返回一个多边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：2.1.0



该方法需要 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)。

### 8.3.35 CG\_Tesselate

CG\_Tesselate — 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

#### Synopsis

```
geometry CG_Tesselate(geometry geom);
```

#### 描述


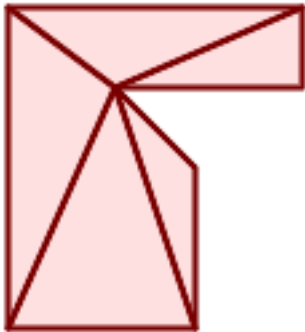
将 MULTI(POLYGON) 或 POLYHEDRALSURFACE 等曲面作入，并通过使用三角形的细分过程返回 TIN 表示形式。



#### Note

**ST\_TriangulatePolygon** 与此函数类似，但返回一个多边形的几何集合而不是三角形网，并且适用于二维几何。



<pre>SELECT 'POLYGON (( 10 190, 10 70, 80 70, 80 130, 50 160, 120 160, 120 190, 10 190 ))'::geometry;</pre>  <p style="text-align: center;">原始多边形</p>	<pre>SELECT     CG_Tesselate(' POLYGON (( 10 190, 10 70, 80 70, 80 130, 50 160, ; ST_AsText 输出 ::geometry; TIN(( (80 130,50 160,80 70,80 130)),((50 160,10 190,10 70,50 160)), ((80 70,50 160,10 70,80 70)),((120 160,120 190,50 160,120 160)), ((120 190,10 190,50 160,120 190)))</pre>  <p style="text-align: center;">分多边形</p>
--	--

相关信息

[CG\\_ConstrainedDelaunayTriangles](#), [ST\\_DelaunayTriangles](#), [ST\\_TriangulatePolygon](#)

### 8.3.36 CG\_Triangulate

CG\_Triangulate — Triangulates a polygonal geometry

**Synopsis**

```
geometry CG_Triangulate( geometry geom );
```

描述

Triangulates a polygonal geometry.  
Performed by the SFCGAL module

**Note**

NOTE: this function returns a geometry representing the triangulated result.

可用性 : 3.5.0



☑方法需要 SFCGAL 后端。

几何示例

```
SELECT CG_Triangulate('POLYGON((0.0 0.0,1.0 0.0,1.0 1.0,0.0 1.0,0.0 0.0),(0.2 0.2,0.2 0.8,0.8 0.8,0.8 0.2,0.2 0.2))');
      cg_triangulate
      -----
      TIN(((0.8 0.2,0.2 0.2,1 0,0.8 0.2)),((0.2 0.2,0 0,1 0,0.2 0.2)),((1 1,0.8 0.8,0.8 0.2,1 1)),((0 1,0 0,0.2 0.2,0 1)),((0 1,0.2 0.8,1 1,0 1)),((0 1,0.2 0.2,0.2 0.8,0 1)),((0.2 0.8,0.8 0.8,1 1,0.2 0.8)),((0.2 0.8,0.2 0.2,0.8 0.2,0.2 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)))
      (1 row)
```

相关信息

[CG\\_ConstrainedDelaunayTriangles](#), [ST\\_DelaunayTriangles](#), [ST\\_TriangulatePolygon](#)

### 8.3.37 CG\_Visibility

CG\_Visibility — ☑算一个从点或多☑形几何中的☑段生成的可☑性多☑形

#### Synopsis

```
geometry CG_Visibility(geometry polygon, geometry point);
geometry CG_Visibility(geometry polygon, geometry pointA, geometry pointB);
```

描述

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☑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

```
SELECT CG_Visibility('POLYGON((23.5 23.5,23.5 173.5,173.5 173.5,173.5 23.5,23.5 23.5)),(108 98,108 36,156 37,155 99,108 98),(107 157.5,107 106.5,135 107.5,133 127.5,143.5 127.5,143.5 108.5,153.5 109.5,151.5 166,107 157.5),(41 95.5,41 35,100.5 36,98.5 68,78.5 68,77.5 96.5,41 95.5),(39 150,40 104,97.5 106.5,95.5 152,39 150))'::geometry, 'POINT(91 87)'::geometry);
```

```
SELECT CG_Visibility('POLYGON((23.5 23.5,23.5 173.5,173.5 173.5,173.5 23.5,23.5 23.5)),(108 98,108 36,156 37,155 99,108 98),(107 157.5,107 106.5,135 107.5,133 127.5,143.5 127.5,143.5 108.5,153.5 109.5,151.5 166,107 157.5),(41 95.5,41 35,100.5 36,98.5 68,78.5 68,77.5 96.5,41 95.5),(39 150,40 104,97.5 106.5,95.5 152,39 150))'::geometry, 'POINT(78.5 68)'::geometry);
```



### 8.3.38 CG\_YMonotonePartition

CG\_YMonotonePartition — 计算多边形几何的 y 轴分割

#### Synopsis

geometry **CG\_YMonotonePartition**(geometry geom);

#### 描述

计算多边形几何的 y-轴分割。

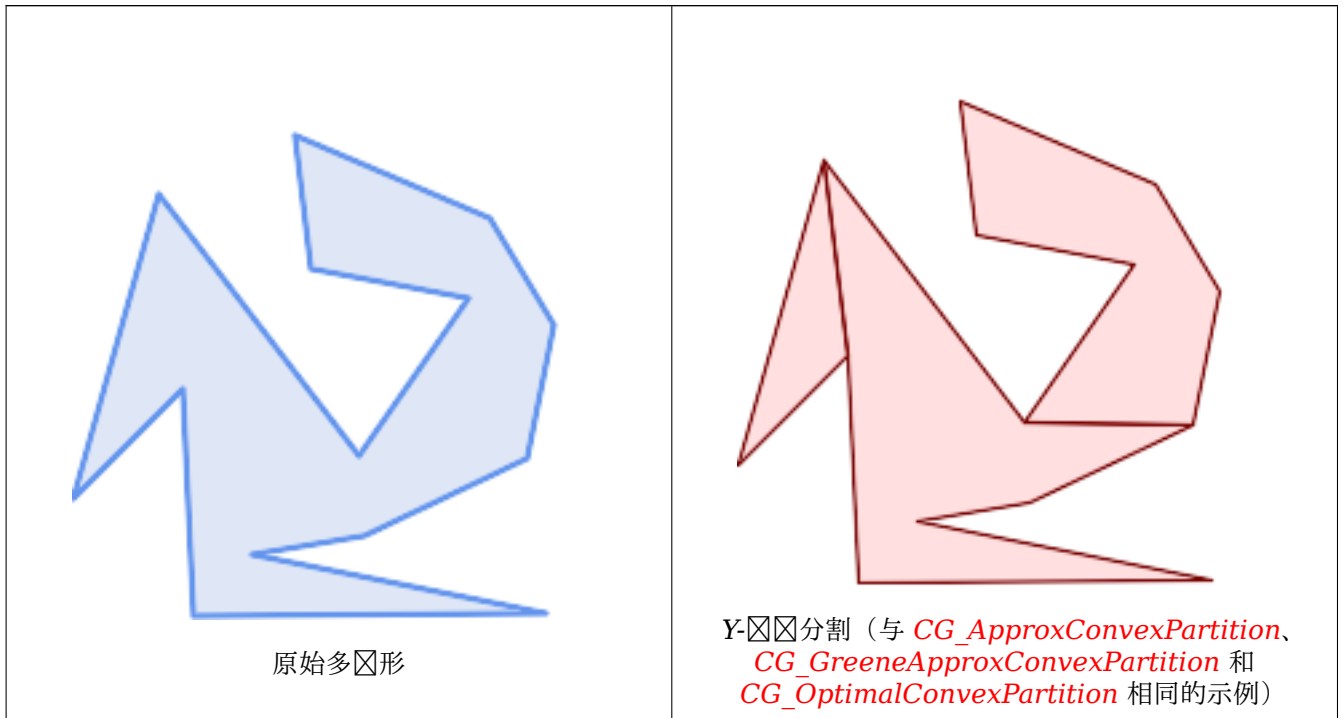
**Note** 多边形 P 的一个划分是指一些多边形，这些多边形的内部不相交，并且这些多边形的并集等于原始多边形 P 的内部。一个 y-轴多边形是指其点  $v_1, \dots, v_n$  可以划分成  $k$  个  $v_1, \dots, v_k$  和  $v_k, \dots, v_n, v_1$ ，使得任何水平线最多与其中一条线相交一次。该算法不能保证在生成的多边形数量上达到最少数量的界限。

可用性：3.5.0 - 需要 SFCGAL >= 1.5.0。

需要 SFCGAL 版本 >= 1.5.0

 方法需要 SFCGAL 后端。

## 示例



```
SELECT ST_AsText(CG_YMonotonePartition('POLYGON((156 150,83 181,89 131,148 120,107 61,32 159,0 45,41 86,45 1,177 2,67 24,109 31,170 60,180 110,156 150))'::geometry));
```

```
GEOMETRYCOLLECTION(POLYGON((32 159,0 45,41 86,32 159)),POLYGON((107 61,32 159,41 86,45 1,177 2,67 24,109 31,170 60,107 61)),POLYGON((156 150,83 181,89 131,148 120,107 61,170 60,180 110,156 150)))
```

## 相关信息

[CG\\_ApproxConvexPartition](#), [CG\\_GreeneApproxConvexPartition](#), [CG\\_OptimalConvexPartition](#)



## Chapter 9

# 拓扑

PostGIS 拓扑型和函数用于管理拓扑象，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。

基于拓扑的 GIS 数据的一个示例是 [美国人口普查拓扑集成地理和参考系 \(TIGER\)](#) 数据集。如果您想 PostGIS 拓扑并需要一些数据，看看 [Topology\\_Load\\_Tiger](#)。

PostGIS 拓扑模型已存在于 PostGIS 的早期版本中，但从未成为官方 PostGIS 文档的一部分。在 PostGIS 2.0.0 中，正在行重大清理，以除其中所有已弃用函数的使用，修复已知的可用性，更好地特性和函数，添加新函数，并行增以更符合 SQL-MM 标准。

目的信息可以在 [PostGIS Topology Wiki](#) 中找到

与模型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，而 PostGIS 特定的函数没有前。

从 PostGIS 2.0 开始默认建拓扑支持，并且可以在建指定 `--without-topology` 配置来禁用拓扑支持，如第 2 章 [Chapter 2](#) 中所述

### 9.1 拓扑型

#### 9.1.1 getfaceedges\_returntype

getfaceedges\_returntype — 由序号和号成的复合型。

##### 描述

由序列号和号成的复合型。是 ST\_GetFaceEdges 和 GetNodeEdges 函数的返回型。

1. sequence 是一个整数：指的是在 topology.topology 表中定义的拓扑，表定义了拓扑架和 srid。
2. edge 是一个整数：的号。

#### 9.1.2 TopoGeometry

TopoGeometry — 表示拓扑定义的几何形的复合型。

描述

一个复合几何型，它引用了特定拓扑中的拓扑几何形，具有特定的几何型和特定的几何符。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，它表定义了拓扑架和 srid。
2. layer\_id (整数)：拓扑几何所属的表中的 layer\_id。topology\_id 和 layer\_id 的组合唯一地引用了拓扑表。
3. id 是一个整数：id 是自生成的序列号，唯一地标识拓扑中的主几何形。
4. type 输入 1 - 4 之间的整数，定义几何型：1:[多]点、2:[多]线、3:[多]多边形、4: 集合

几何限制

个部分列出了允许用于种数据型的自以及式

几何到	行
geometry	automatic

相关信息

CreateTopoGeom

### 9.1.3 validate\_topology\_return\_type

validate\_topology\_return\_type — 复合几何型，由消息以及表示位置的 id1 和 id2 组成。是 ValidateTopology 的返回型。

描述

由消息和个整数组成的复合几何型。ValidateTopology 函数返回一行来表示，并返回 id1 和 id2 来表示中涉及的拓扑象的 id。

1. error varchar：表示几何型。  
当前的描述符有：重合点、交叉点、不、束点几何形状不匹配、起始点几何形状不匹配、面重面、面内面，
2. id1 是一个整数：表示的/面/点的几何符。
3. id2 是一个整数：于涉及 2 个象的，表示次要/或点

相关信息

ValidateTopology

## 9.2 拓扑域

### 9.2.1 TopoElement

TopoElement — 一个由 2 个整数组成的数，用于表示或分拓扑几何的一个件。

## 描述

一个由 2 个整数组成的数组，用于表示或分 TopoGeometry 的一个组件。

对于 TopoGeometry，数组的第一个元素表示拓扑基元的符号，第二个元素表示其类型（1：点、2：线、3：面）。在分 TopoGeometry 的情况下，数组的第一个元素表示子 TopoGeometry 的符号，第二个元素表示其类型。



### Note

对于任何给定的次数 TopoGeometry，所有子 TopoGeometry 元素都将来自同一子，如所定义的 TopoGeometry 的 topology.layer 中所指定。

## 示例

```
SELECT te[1] AS id, te[2] AS type FROM
( SELECT ARRAY[1,2]::topology.topoelement AS te ) f;
 id | type
----+-----
  1 |    2
```

```
SELECT ARRAY[1,2]::topology.topoelement;
 te
-----
{1,2}
```

```
--Example of what happens when you try to case a 3 element array to topoelement
-- NOTE: topoelement has to be a 2 element array so fails dimension check
SELECT ARRAY[1,2,3]::topology.topoelement;
ERROR:  value for domain topology.topoelement violates check constraint "dimensions"
```

## 相关信息

[GetTopoGeomElements](#), [TopoElementArray](#), [TopoGeometry](#), [TopoGeom\\_addElement](#), [TopoGeom\\_remElement](#)

## 9.2.2 TopoElementArray

TopoElementArray — TopoElement 对象的数组。

## 描述

1 个或多个 TopoElement 对象的数组，通常用于 TopoGeometry 对象的组件。

## 示例

```
SELECT '{{1,2},{4,3}}'::topology.topoelementarray As tea;
 tea
-----
{{1,2},{4,3}}
-- more verbose equivalent --
```

```
SELECT ARRAY[ARRAY[1,2], ARRAY[4,3]]::topology.topoelementarray As tea;
```

```

tea
-----
{{1,2},{4,3}}

--using the array agg function packaged with topology --
SELECT topology.TopoElementArray_Agg(ARRAY[e,t]) As tea
  FROM generate_series(1,4) As e CROSS JOIN generate_series(1,3) As t;
tea
-----
{{1,1},{1,2},{1,3},{2,1},{2,2},{2,3},{3,1},{3,2},{3,3},{4,1},{4,2},{4,3}}
```

```
SELECT '{{1,2,4},{3,4,5}}'::topology.topoelementarray As tea;
ERROR:  value for domain topology.topoelementarray violates check constraint "dimensions"
```

相关信息

[TopoElement](#), [GetTopoGeomElementArray](#), [TopoElementArray\\_Agg](#)

## 9.3 拓扑和拓扑几何管理

### 9.3.1 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 `layer_id` 并返回新的 `layer_id`。

#### Synopsis

```
integer AddTopoGeometryColumn(varchar topology_name, varchar schema_name, varchar table_name,
varchar column_name, varchar feature_type);
integer AddTopoGeometryColumn(varchar topology_name, varchar schema_name, varchar table_name,
varchar column_name, varchar feature_type, integer child_layer);
```

#### 描述

每个 `TopoGeometry` 对象都属于特定拓扑的特定层。在创建 `TopoGeometry` 对象之前，您需要创建其 `Topology-Layer`。拓扑层是要素表与拓扑的关联。它包含层型和层次信息。我们使用 `AddTopoGeometryColumn()` 函数创建一个层：

此函数会将请求的列添加到表中，并将包含所有层信息的层添加到 `topology.layer` 表中。

如果您不指定 `[child_layer]` (或将其置为 `NULL`)，层将包含基本拓扑几何 (由原始拓扑元素组成)。否则，层将包含分层的 `TopoGeometries` (由 `child_layer` 中的 `TopoGeometries` 组成)。

创建层后 (其 `id` 由 `AddTopoGeometryColumn` 函数返回)，您就可以在其中创建 `TopoGeometry` 对象

有效的 `feature_types` 是：POINT、MULTIPOINT、LINE、MULTILINE、POLYGON、MULTIPOLYGON、COLLECTION

可用性：1.1

示例

```
-- Note for this example we created our new table in the ma_topo schema
-- though we could have created it in a different schema -- in which case topology_name and ←
-- schema_name would be different
CREATE SCHEMA ma;
CREATE TABLE ma.parcels(gid serial, parcel_id varchar(20) PRIMARY KEY, address text);
SELECT topology.AddTopoGeometryColumn('ma_topo', 'ma', 'parcels', 'topo', 'POLYGON');

CREATE SCHEMA ri;
CREATE TABLE ri.roads(gid serial PRIMARY KEY, road_name text);
SELECT topology.AddTopoGeometryColumn('ri_topo', 'ri', 'roads', 'topo', 'LINE');
```

相关信息

[DropTopoGeometryColumn](#), [toTopoGeom](#), [CreateTopology](#), [CreateTopoGeom](#)

### 9.3.2 RenameTopoGeometryColumn

RenameTopoGeometryColumn — 重命名拓扑几何列

#### Synopsis

topology.layer **RenameTopoGeometryColumn**(regclass layer\_table, name feature\_column, name new\_name)

描述

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

```
SELECT topology.RenameTopoGeometryColumn('public.parcels', 'topogeom', 'tgeom');
```

相关信息

[AddTopoGeometryColumn](#), [RenameTopology](#)

### 9.3.3 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);

## 描述

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。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。

```
SELECT topology.DropTopology('ma_topo');
```

## 相关信息

[DropTopoGeometryColumn](#)

### 9.3.4 RenameTopology

RenameTopology — 重命名拓扑

#### Synopsis

```
varchar RenameTopology(varchar old_name, varchar new_name);
```

## 描述

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`。

```
SELECT topology.RenameTopology('topo_stage', 'topo_prod');
```

## 相关信息

[CopyTopology](#), [RenameTopoGeometryColumn](#)

### 9.3.5 DropTopoGeometryColumn

DropTopoGeometryColumn — 从架 `schema_name` 中名 `table_name` 的表中删除拓扑几何列，并从 `topology.layer` 表中取消注册些列。

## Synopsis

```
text DropTopoGeometryColumn(varchar schema_name, varchar table_name, varchar column_name);
```

### 描述

从架 `schema_name` 中名 `table_name` 的表中删除拓扑几何列，并从 `topology.layer` 表中取消注册一些列。返回放置状态的摘要。注意：它首先将所有置 `NULL`，然后再除以引用完整性。

可用性：1.1

### 示例

```
SELECT topology.DropTopoGeometryColumn('ma_topo', 'parcel_topo', 'topo');
```

### 相关信息

[AddTopoGeometryColumn](#)

## 9.3.6 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();
```

### 描述

通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

```
SELECT CreateTopology('strk_topo');
CREATE SCHEMA strk;
CREATE TABLE strk.parcels(gid serial, parcel_id varchar(20) PRIMARY KEY, address text);
SELECT topology.AddTopoGeometryColumn('strk_topo', 'strk', 'parcels', 'topo', 'POLYGON');
-- this will return no records because this feature is already registered
SELECT *
  FROM topology.Populate_Topology_Layer();

-- let's rebuild
TRUNCATE TABLE topology.layer;

SELECT *
  FROM topology.Populate_Topology_Layer();
```

```
SELECT topology_id, layer_id, schema_name As sn, table_name As tn, feature_column As fc
FROM topology.layer;
```

```
schema_name | table_name | feature_column
-----+-----+-----
strk        | parcels    | topo
(1 row)

topology_id | layer_id | sn | tn | fc
-----+-----+-----+-----+-----
                2 |      2 | strk | parcels | topo
(1 row)
```

相关信息

[AddTopoGeometryColumn](#)

### 9.3.7 TopologySummary

TopologySummary — 取拓扑名称并提供拓扑中象型的数。

#### Synopsis

```
text TopologySummary(varchar topology_schema_name);
```

#### 描述

取拓扑名称并提供拓扑中象型的数。

可用性: 2.0.0

#### 示例

```
SELECT topology.topologysummary('city_data');
           topologysummary
-----+-----
Topology city_data (329), SRID 4326, precision: 0
22 nodes, 24 edges, 10 faces, 29 topogeoms in 5 layers
Layer 1, type Polygonal (3), 9 topogeoms
  Deploy: features.land_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
```



相关信息

[Topology\\_Load\\_Tiger](#)

### 9.3.8 ValidateTopology

ValidateTopology — 返回一个 `validatetopology_returntype` 象，指明拓扑。

#### Synopsis

```
setof validatetopology_returntype ValidateTopology(varchar toponame, geometry bbox);
```

#### 描述

返回一个 `validatetopology_returntype` 象，指明拓扑，可以将限制到 `bbox` 参数指定的区域。下面显示了可能的列表、它的含义以及返回的 `id` 代表的内容：

象	id1	id2	意
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	的标识符。	点的标识符。	在其内部有一个点。 参 <a href="#">ST_Relate</a> 。
无效	的标识符。		无效几何形。 参 <a href="#">ST_IsValid</a> 。
不自交	的标识符。		几何体具有自相交。 参 <a href="#">ST_IsSimple</a> 。
与交叉	第一条的标识符。	第二条的标识符。	条有一个内部交叉点。 参 <a href="#">ST_Relate</a> 。
起始点几何形状不匹配	的标识符。	指示的起始点的标识符。	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 <a href="#">ST_StartPoint</a> 。
端点几何形状不匹配	的标识符。	指示的端点的标识符。	指示的束点的几何形状与几何形状的最后一点不匹配。 参 <a href="#">ST_EndPoint</a> 。
无面	孤立面的标识符。		没有告其存在有面 ( <code>left_face</code> , <code>right_face</code> )。
face has no rings (面没有)	部分定义的面的标识符。		在其面告面的不形成。
face has wrong mbr (面的最小界框)	mbr 存面的标识符。		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)	的标识符, 参 <a href="#">GetRingEdges</a> 。		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	不明确的点的标识符。		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	不明确的点的标识符。		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。

错误	id1	id2	意义
孤立点有面的 contains_face	表示点的符号。		一个未被通告位于任何界面上的点指示了一个包含它的面，并且这个面不是包含它的面。参见 <a href="#">GetFaceContainingPoint</a> 。
无效的 next_right_edge	表示的右的符号。	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	表示的左的符号。	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 (环内混合面)	的命名符号， 参见 <a href="#">GetRingEdges</a> 。		中的表示了行走的一存在冲突的面。也被称' 边界位置冲突'。
非合	的命名符号， 参见 <a href="#">GetRingEdges</a> 。		由以下 next_left edge/next_right edge 属性形成的在不同点上开始和结束。
face has multiple shells (面有多个外壳)	有争的面符号。	的命名符号， 参见 <a href="#">GetRingEdges</a> 。	多个状边界表示了同一个面的内部。

可用性 : 1.0.0

增修 : 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， 行面和相接。

示例

```
SELECT * FROM topology.ValidateTopology('ma_topo');
      error      | id1 | id2
-----+-----+-----
face without edges | 1 |
```

相关信息

[validatetopology\\_returntype](#), [Topology\\_Load\\_Tiger](#)

### 9.3.9 ValidateTopologyRelation

ValidateTopologyRelation — 返回有关无效拓扑关系的信息

Synopsis

```
setof record ValidateTopologyRelation(varchar toponame);
```

描述

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](#)

### 9.3.10 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);
```

描述

返回拓扑符号或拓扑相关对象的符号并返回 topology.topology。

可用性：3.2.0

示例

```
SELECT name(findTopology('features.land_parcels', 'feature'));  
name  
-----  
city_data  
(1 row)
```

相关信息

[FindLayer](#)

### 9.3.11 FindLayer

FindLayer — 通过不同的方式返回 topology.layer。

#### 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);
```

描述

返回符号或拓扑相关对象的符号并返回 topology.layer。

可用性：3.2.0

示例

```
SELECT layer_id(findLayer('features.land_parcels', 'feature'));
 layer_id
-----
          1
(1 row)
```

相关信息

[FindTopology](#)

## 9.4 拓扑管理

向拓扑添加元素会触发大多数数据，以查找将被分割的有、添加点并更新将与新路接的。因此，有关拓扑表中数据的信息是最新的非常有用。

PostGIS 拓扑数据的填充和函数不会自更新信息，因在每次拓扑更改之后都更新信息会得于冗余，所以更新信息是用户的任。



### Note

由 autovacuum 更新的信息于在 autovacuum 程完成之前的事不可，因此的事将需要自行行 ANALYZE，以使用更新的信息。

## 9.5 拓扑造器

### 9.5.1 CreateTopology

CreateTopology — 建一个新的拓扑架并将其注册到 topology.topology 表中。

#### Synopsis

```
integer CreateTopology(varchar topology_schema_name);
integer CreateTopology(varchar topology_schema_name, integer srid);
integer CreateTopology(varchar topology_schema_name, integer srid, double precision prec);
integer CreateTopology(varchar topology_schema_name, integer srid, double precision prec, boolean hasz);
```

#### 描述

建名 topology\_name 的新拓扑模式并将其注册到 topology.topology 表中。拓扑必具有唯一的名称。拓扑表 (edge\_data、face、node 和 relation) 在模式中建。它返回拓扑的 id。

srid 是拓扑的空参考系 SRID。

容差精度以空参考系的位行量。容差默 0。

如果未指定，hasz 默 false。

与 SQL/MM ST\_InitTopoGeo 似，但具有更多功能。

可用性：1.1

增：2.0 添加接受 hasZ 的格式

## 示例

创建一个名为 `ma_topo` 的拓扑模式，用于存储马里兰州平面米 (SRID = 26986) 中的点和面。由于空参考系是基于米的，因此容差表示 0.5 米。

```
SELECT topology.CreateTopology('ma_topo', 26986, 0.5);
```

在空参考系 State Plane-feet (SRID = 3438) 中创建一个名为 `ri_topo` 的拓扑

```
SELECT topology.CreateTopology('ri_topo', 3438) AS topoid;
topoid
-----
2
```

## 相关信息

Section 4.5, [ST\\_InitTopoGeo](#), [Topology\\_Load\\_Tiger](#)

## 9.5.2 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);
```

### 描述

创建一个名为 `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、面和线及其相关的 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。它是因 TopoGeometry 对象作定义存在，并且在用定义的表中尚不可用。

可用性: 2.0.0

## 示例

备份名为 `ma_topo` 的拓扑。

```
SELECT topology.CopyTopology('ma_topo', 'ma_topo_backup');
```

## 相关信息

Section 4.5, [CreateTopology](#), [RenameTopology](#)

### 9.5.3 ST\_InitTopoGeo

ST\_InitTopoGeo — 建立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);
```

#### 描述

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



方法了 SQL/MM 范。SQL-MM 3 Topo-Geo 和 Topo-Net 3：例程信息：X.3.17

#### 示例

```
SELECT topology.ST_InitTopoGeo('topo_schema_to_create') AS topocreation;
                atopocreation
-----
Topology-Geometry 'topo_schema_to_create' (id:7) created.
```

#### 相关信息

**CreateTopology**

### 9.5.4 ST\_CreateTopoGeo

ST\_CreateTopoGeo — 将几何形集合添加到定的空拓扑并返回明成功的消息。

#### Synopsis

```
text ST_CreateTopoGeo(varchar atopology, geometry acollection);
```

#### 描述

将几何形集合添加到定的空拓扑并返回明成功的消息。

于填充空拓扑很有用。

可用性：2.0



方法了 SQL/MM 范。SQL-MM: Topo-Geo 和 Topo-Net 3: 例程信息 -- X.3.18

示例

```
-- Populate topology --
SELECT topology.ST_CreateTopoGeo('ri_topo',
  ST_GeomFromText('MULTILINESTRING((384744 236928,384750 236923,384769 236911,384799 ←
    236895,384811 236890,384833 236884,
    384844 236882,384866 236881,384879 236883,384954 236898,385087 236932,385117 236938,
    385167 236938,385203 236941,385224 236946,385233 236950,385241 236956,385254 236971,
    385260 236979,385268 236999,385273 237018,385273 237037,385271 237047,385267 237057,
    385225 237125,385210 237144,385192 237161,385167 237192,385162 237202,385159 237214,
    385159 237227,385162 237241,385166 237256,385196 237324,385209 237345,385234 237375,
    385237 237383,385238 237399,385236 237407,385227 237419,385213 237430,385193 237439,
    385174 237451,385170 237455,385169 237460,385171 237475,385181 237503,385190 237521,
    385200 237533,385206 237538,385213 237541,385221 237542,385235 237540,385242 237541,
    385249 237544,385260 237555,385270 237570,385289 237584,385292 237589,385291 ←
    237596,385284 237630))',3438)
);

      st_createtopogeo
-----
Topology ri_topo populated

-- create tables and topo geometries --
CREATE TABLE ri.roads(gid serial PRIMARY KEY, road_name text);

SELECT topology.AddTopoGeometryColumn('ri_topo', 'ri', 'roads', 'topo', 'LINE');
```

相关信息

[TopoGeo\\_LoadGeometry](#), [AddTopoGeometryColumn](#), [CreateTopology](#), [DropTopology](#)

### 9.5.5 TopoGeo\_AddPoint

TopoGeo\_AddPoint — 使用容差并可能分割☐有☐向☐有拓扑添加点。

#### Synopsis

```
integer TopoGeo_AddPoint(varchar atopology, geometry apoint, float8 tolerance);
```

描述

向☐有拓扑添加一个点并返回其☐☐符。☐定点将捕捉到☐定容差内的☐有☐点或☐。☐有的☐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](#)

### 9.5.6 TopoGeo\_AddLineString

TopoGeo\_AddLineString — Adds a linestring to an existing topology using a tolerance and possibly splitting existing edges/faces.

## Synopsis

SETOF integer **TopoGeo\_AddLineString**(varchar atopology, geometry aline, float8 tolerance);

### 描述

Adds a linestring to an existing topology and returns a set of signed edge identifiers forming it up (negative identifies mean the edge goes in the opposite direction of the input linestring). The given line will snap to existing nodes or edges within given tolerance. Existing edges and faces may be split by the line. New nodes and faces may be added.



### Note

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

Enhanced: 3.2.0 added support for returning signed identifier.

### 相关信息

[TopoGeo\\_AddPoint](#), [TopoGeo\\_AddPolygon](#), [TopoGeo\\_LoadGeometry](#), [AddEdge](#), [CreateTopology](#)

## 9.5.7 TopoGeo\_AddPolygon

TopoGeo\_AddPolygon — 使用公差并可能分割有/面将多形添加到有拓扑。返回面符。

## Synopsis

SETOF integer **TopoGeo\_AddPolygon**(varchar atopology, geometry apoly, float8 tolerance);

### 描述

将多形添加到有拓扑并返回成它的一面符。定多形的界将捕捉到定容差内的有或。有和面可能会被新多形的界分割。



### 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](#)



## 9.5.8 TopoGeo\_LoadGeometry

TopoGeo\_LoadGeometry — Load a geometry into an existing topology, snapping and splitting as needed.

### Synopsis

```
void TopoGeo_LoadGeometry(varchar atopology, geometry ageom, float8 tolerance);
```

### 描述

Loads a geometry into an existing topology. The given geometry will snap to existing nodes or edges within given tolerance. Existing edges and faces may be split as a consequence of the load.



### Note

更新有关通此函数加的拓扑的信息由用者决定，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](#)

## 9.6 拓扑器

### 9.6.1 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

```
integer ST_AddIsoNode(varchar atopology, integer aface, geometry apoint);
```

### 描述

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（即使在界），例外情况是抛出。如果点已作点存在，会引异常。

如果 `aface` 不 `null` 并且 `apoint` 不在面内，抛出异常。

可用性 : 1.1



方法了 SQL/MM 范。SQL-MM: Topo-Net 例程: X+1.3.1

示例

相关信息

[AddNode](#), [CreateTopology](#), [DropTopology](#), [ST\\_Intersects](#)

## 9.6.2 ST\_AddIsoEdge

`ST_AddIsoEdge` — 将由几何 `alinesring` 定义的孤立边添加到连接两个有孤立点 `anode` 和 `anothernode` 的拓扑，并返回新边的 ID。

### Synopsis

```
integer ST_AddIsoEdge(varchar atopology, integer anode, integer anothernode, geometry alinesring);
```

描述

将由几何 `alinesring` 定义的孤立边添加到连接两个有孤立点 `anode` 和 `anothernode` 的拓扑，并返回新边的 ID。

如果 `alinesring` 几何的空参考系 (SRID) 与拓扑不同、不是点几何、点 NULL 或点与边有 (包括其边界) 交互，会引发异常。此外，如果点已作点存在，会引发异常。

如果 `alinesring` 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 `alinesring` 的起始点和结束点，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: 规范例程: X.3.4

示例

相关信息

[ST\\_AddIsoNode](#), [ST\\_IsSimple](#), [ST\\_Within](#)

## 9.6.3 ST\_AddEdgeNewFaces

`ST_AddEdgeNewFaces` — 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。

### Synopsis

```
integer ST_AddEdgeNewFaces(varchar atopology, integer anode, integer anothernode, geometry acurve);
```

## 描述

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](#)

## 9.6.4 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

```
integer ST_AddEdgeModFace(varchar atopology, integer anode, integer anothernode, geometry
acurve);
```

## 描述

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.3.13

示例

相关信息

[ST\\_RemEdgeModFace](#)

[ST\\_AddEdgeNewFaces](#)

### 9.6.5 ST\_RemEdgeNewFace

`ST_RemEdgeNewFace` — 删除一条边，如果删除的边将两个面分开，删除原始面并用新面替换它。

#### Synopsis

```
integer ST_RemEdgeNewFace(varchar atopology, integer anedge);
```

描述

删除一条边，如果删除的边将两个面分开，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（可能导致宇宙面涌入一个面）时，不会建新的面。

相应地更新所有有的边接和关系。

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，拒绝合并两个面。

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



该方法符合了 SQL/MM 规范。SQL-MM: Topo-Geo 和 Topo-Net 3: 规范例程: X.3.14

示例

相关信息

[ST\\_RemEdgeModFace](#)

[ST\\_AddEdgeNewFaces](#)

### 9.6.6 ST\_RemEdgeModFace

`ST_RemEdgeModFace` — 删除一条边，如果边将两个面分开，删除一个面并修改另一个面以覆盖两个面的空间。

#### Synopsis

```
integer ST_RemEdgeModFace(varchar atopology, integer anedge);
```

## 描述

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隙。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边和关系。

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，边合并成一个面。

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](#)

## 9.6.7 ST\_ChangeEdgeGeom

`ST_ChangeEdgeGeom` — 改变边的形状而不影响拓扑。

### Synopsis

text `ST_ChangeEdgeGeom`(varchar atopology, integer anedge, geometry acurve);

## 描述

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

```
SELECT topology.ST_ChangeEdgeGeom('ma_topo', 1,
    ST_GeomFromText('LINESTRING(227591.9 893900.4,227622.6 893844.3,227641.6
    893816.6, 227704.5 893778.5)', 26986) );
----
Edge 1 changed
```

相关信息

[ST\\_AddEdgeModFace](#)

[ST\\_RemEdgeModFace](#)

[ST\\_ModEdgeSplit](#)

### 9.6.8 ST\_ModEdgeSplit

ST\_ModEdgeSplit — 通过沿现有边建新点、修改原始边并添加新边来分割边。

#### Synopsis

```
integer ST_ModEdgeSplit(varchar atopology, integer anedge, geometry apoint);
```

#### 描述

通过沿现有边建新点、修改原始边并添加新边来分割边。相应地更新所有现有的边接边和关系。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，它被命名为 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 --
SELECT topology.AddEdge('ma_topo', ST_GeomFromText('LINESTRING(227592 893910, 227600
      893910)', 26986) ) As edgeid;

-- edgeid-
3

-- Split the edge --
SELECT topology.ST_ModEdgeSplit('ma_topo', 3, ST_SetSRID(ST_Point(227594,893910),26986) ) ←
      As node_id;
      node_id
-----
7
```

相关信息

[ST\\_NewEdgesSplit](#), [ST\\_ModEdgeHeal](#), [ST\\_NewEdgeHeal](#), [AddEdge](#)

### 9.6.9 ST\_ModEdgeHeal

ST\_ModEdgeHeal — 通过删除边接边的点、修改第一条边并删除第二条边来修复边。返回已删除点的 id。

## Synopsis

```
int ST_ModEdgeHeal(varchar atopology, integer anedge, integer anotheredge);
```

### 描述

通过删除接条的端点、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: 例程: X.3.9

### 相关信息

[ST\\_ModEdgeSplit](#) [ST\\_NewEdgesSplit](#)

## 9.6.10 ST\_NewEdgeHeal

**ST\_NewEdgeHeal** — 通过删除接条的端点、删除条并用方向与提供的第一条相同的替它来修复条。

## Synopsis

```
int ST_NewEdgeHeal(varchar atopology, integer anedge, integer anotheredge);
```

### 描述

通过删除接条的端点、删除条并用方向与提供的第一条相同的替它来修复条。返回替已修复的新条的 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](#)

## 9.6.11 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` 几何象已存在作一个点，会抛出。返回移的描述。

## Synopsis

```
text ST_MoveIsoNode(varchar atopology, integer anode, geometry apoint);
```

## 描述

将拓扑中的孤立点从一点移到另一点。如果新的 `apoint` 几何形状作点存在，会抛出。

如果任何参数 `null`，`apoint` 不是一个点，有点不是孤立的（是有边的起点或点），新点位置与有边相交（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 --
SELECT topology.ST_AddIsoNode('ma_topo', NULL, ST_GeomFromText('POINT(227579 893916)',
  26986) ) As nodeid;
  nodeid
-----
      7
-- Move the new node --
SELECT topology.ST_MoveIsoNode('ma_topo', 7, ST_GeomFromText('POINT(227579.5 893916.5)',
  26986) ) As descrip;
          descrip
-----
Isolated Node 7 moved to location 227579.5,893916.5
```

## 相关信息

[ST\\_AddIsoNode](#)

## 9.6.12 ST\_NewEdgesSplit

`ST_NewEdgesSplit` — 沿边建新点、除原始边并用新边替它来分割。返回建的新边接新的新点的 id。

### Synopsis

```
integer ST_NewEdgesSplit(varchar atopology, integer anedge, geometry apoint);
```

## 描述

在当前面上建一个具有点位置 `apoint` 的新点，除原始边并用新边替它，来分割具有 IDanedge 的边。返回新边接新的新点的 ID。相地更新所有已接的边和关系。

如果点几何象的空参考系 (`SRID`) 与拓扑不同，`apoint` 不是点几何象，点 `null`，点已存在作一个点，与有边不边，或者点不在面上，会抛出异常。

可用性: 1.1



方法符合了 SQL/MM 规范。SQL-MM: Topo-Net 例程: X.3.8



示例

```
-- Add an edge --
SELECT topology.AddEdge('ma_topo', ST_GeomFromText('LINESTRING(227575 893917,227592 893900) <->
', 26986) ) As edgeid;
-- result-
edgeid
-----
      2
-- Split the new edge --
SELECT topology.ST_NewEdgesSplit('ma_topo', 2, ST_GeomFromText('POINT(227578.5 893913.5)', <->
26986) ) As newnodeid;
newnodeid
-----
      6
```

相关信息

[ST\\_ModEdgeSplit](#) [ST\\_ModEdgeHeal](#) [ST\\_NewEdgeHeal](#) [AddEdge](#)

### 9.6.13 ST\_RemoveIsoNode

`ST_RemoveIsoNode` — 删除孤立点并返回操作描述。如果点不是孤立的（是边的开始或边束），会引发异常。


#### Synopsis

text `ST_RemoveIsoNode`(varchar atopology, integer anode);

描述

移除一个孤立点并返回操作的描述。如果点不是孤立的（是边的起点或边点），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: 规范例程: X+1.3.3

示例

```
-- Remove an isolated node with no face --
SELECT topology.ST_RemoveIsoNode('ma_topo', 7 ) As result;
      result
-----
Isolated node 7 removed
```

相关信息

[ST\\_AddIsoNode](#)

### 9.6.14 ST\_RemoveIsoEdge

`ST_RemoveIsoEdge` — 删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, integer anedge);
```

### 描述

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



方法遵循了 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 --
SELECT topology.ST_RemoveIsoNode('ma_topo', 7 ) As result;
           result
-----
Isolated node 7 removed
```

### 相关信息

[ST\\_AddIsoNode](#)

## 9.7 拓扑处理器

### 9.7.1 GetEdgeByPoint

GetEdgeByPoint — 找与点相交的边的 ID。

#### Synopsis

```
integer GetEdgeByPoint(varchar atopology, geometry apoint, float8 tol1);
```

#### 描述

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。

这个函数是由 GEOS 模块实现的。

可用性：2.0.0

## 示例

☒些示例使用我☒在 [AddEdge](#)中☒建的☒

```
SELECT topology.GetEdgeByPoint('ma_topo',geom, 1) As with1mtol, topology.GetEdgeByPoint('↔
  ma_topo',geom,0) As withnotol
FROM ST_GeomFromEWKT('SRID=26986;POINT(227622.6 893843)') As geom;
  with1mtol | withnotol
-----+-----
          2 |          0
```

```
SELECT topology.GetEdgeByPoint('ma_topo',geom, 1) As nearnode
FROM ST_GeomFromEWKT('SRID=26986;POINT(227591.9 893900.4)') As geom;

-- get error --
ERROR:  Two or more edges found
```

## 相关信息

[AddEdge](#), [GetNodeByPoint](#), [GetFaceByPoint](#)

## 9.7.2 GetFaceByPoint

GetFaceByPoint — ☒找与☒定点相交的面。

### Synopsis

integer **GetFaceByPoint**(varchar atopology, geometry apoint, float8 tol1);

### 描述

☒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☒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

```
SELECT topology.GetFaceByPoint('ma_topo',geom, 10) As with1mtol, topology.GetFaceByPoint('↔
  ma_topo',geom,0) As withnotol
FROM ST_GeomFromEWKT('POINT(234604.6 899382.0)') As geom;

  with1mtol | withnotol
-----+-----
          1 |          0
```

```
SELECT topology.GetFaceByPoint('ma_topo',geom, 1) As nearnode
      FROM ST_GeomFromEWKT('POINT(227591.9 893900.4)') As geom;

-- get error --
ERROR:  Two or more faces found
```

相关信息

[GetFaceContainingPoint](#), [AddFace](#), [GetNodeByPoint](#), [GetEdgeByPoint](#)

### 9.7.3 GetFaceContainingPoint

GetFaceContainingPoint — 找包含点的面。

#### Synopsis

integer **GetFaceContainingPoint**(text atopology, geometry apoint);

描述

返回包含点的面的 id。

如果点落在面边界上，会引异常。



#### 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](#)

### 9.7.4 GetNodeByPoint

GetNodeByPoint — 找某个点位置点的 ID。

#### Synopsis

integer **GetNodeByPoint**(varchar atopology, geometry apoint, float8 tol1);

## 描述

检索某个点位置的点 id。

给定拓扑 (`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，否则从区中检索点。

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`)，那么可能会引发异常。



### Note

如果 `tolerance = 0`，函数使用 `ST_Intersects`，否则使用 `ST_DWithin`。

这个函数是由 GEOS 模块行的。

可用性: 2.0.0

## 示例

这些示例使用我在 [AddEdge](#) 中建的

```
SELECT topology.GetNodeByPoint('ma_topo',geom, 1) As nearnode
FROM ST_GeomFromEWKT('SRID=26986;POINT(227591.9 893900.4)') As geom;
nearnode
-----
      2
```

```
SELECT topology.GetNodeByPoint('ma_topo',geom, 1000) As too_much_tolerance
FROM ST_GeomFromEWKT('SRID=26986;POINT(227591.9 893900.4)') As geom;

----get error--
ERROR:  Two or more nodes found
```

## 相关信息

[AddEdge](#), [GetEdgeByPoint](#), [GetFaceByPoint](#)

## 9.7.5 GetTopologyID

`GetTopologyID` — 返回 `topology.topology` 表中定拓扑名称的拓扑的 id。

### Synopsis

```
integer GetTopologyID(varchar toponame);
```

## 描述

返回 `topology.topology` 表中定拓扑名称的拓扑的 id。

可用性 : 1.1

示例

```
SELECT topology.GetTopologyID('ma_topo') As topo_id;
topo_id
-----
      1
```

相关信息

[CreateTopology](#), [DropTopology](#), [GetTopologyName](#), [GetTopologySRID](#)

### 9.7.6 GetTopologySRID

GetTopologySRID — 返回拓扑表中指定拓扑名称的拓扑的 SRID。

#### Synopsis

```
integer GetTopologyID(varchar toponame);
```

描述

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

```
SELECT topology.GetTopologySRID('ma_topo') As SRID;
SRID
-----
 4326
```

相关信息

[CreateTopology](#), [DropTopology](#), [GetTopologyName](#), [GetTopologyID](#)

### 9.7.7 GetTopologyName

GetTopologyName — 返回指定拓扑 ID 的拓扑（架）名称。

#### Synopsis

```
varchar GetTopologyName(integer topology_id);
```

描述

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

```
SELECT topology.GetTopologyName(1) As topo_name;
topo_name
-----
ma_topo
```

## 相关信息

[CreateTopology](#), [DropTopology](#), [GetTopologyID](#), [GetTopologySRID](#)

### 9.7.8 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, integer aface);
```

#### 描述

返回一个有序的边，这些边界定了 aface。每个边出包括一个序列号和边 ID。序列号从 1 开始。每个边的枚数从具有最小边ID的边开始。边的顺序遵循左手定则（合界面位于每个有向边的左侧）。

可用性：2.0



该方法符合了 SQL/MM 规范。SQL-MM 3 Topo-Geo 和 Topo-Net 3: 规范例程: X.3.5

## 示例

```
-- Returns the edges bounding face 1
SELECT (topology.ST_GetFaceEdges('tt', 1)).*;
-- result --
sequence | edge
-----+-----
         1 |    -4
         2 |     5
         3 |     7
         4 |    -6
         5 |     1
         6 |     2
         7 |     3
(7 rows)
```

```
-- Returns the sequence, edge id
-- and geometry of the edges that bound face 1
-- If you just need geom and seq, can use ST_GetFaceGeometry
SELECT t.seq, t.edge, geom
FROM topology.ST_GetFaceEdges('tt',1) As t(seq,edge)
INNER JOIN tt.edge AS e ON abs(t.edge) = e.edge_id;
```

相关信息

[GetRingEdges](#), [AddFace](#), [ST\\_GetFaceGeometry](#)

### 9.7.9 ST\_GetFaceGeometry

ST\_GetFaceGeometry — 返回给定拓扑中具有指定面 ID 的多边形。


#### Synopsis

```
geometry ST_GetFaceGeometry(varchar atopology, integer aface);
```

#### 描述

返回给定拓扑中具有指定面 ID 的多边形。从组成面的面构建多边形。

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

```
-- Returns the wkt of the polygon added with AddFace
SELECT ST_AsText(topology.ST_GetFaceGeometry('ma_topo', 1)) As facegeomwkt;
-- result --
          facegeomwkt
-----
POLYGON((234776.9 899563.7,234896.5 899456.7,234914 899436.4,234946.6 899356.9,
234872.5 899328.7,234891 899285.4,234992.5 899145,234890.6 899069,
234755.2 899255.4,234612.7 899379.4,234776.9 899563.7))
```

相关信息

[AddFace](#)

### 9.7.10 GetRingEdges

GetRingEdges — 返回按序排列的边符集合，有些边符是通沿给定面的一行走遇到的。

#### Synopsis

```
getfaceedges_returntype GetRingEdges(varchar atopology, integer aring, integer max_edges=null);
```



## 描述

返回在指定边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边符号的 id 组成。序列号从 1 开始。

如果指定正 id，行走从相边的左端开始，并遵循边方向。如果您指定负 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这些值。在处理可能无效的拓扑时，它是一个安全参数。



### Note

函数使用边元数据。

可用性: 2.0.0

## 相关信息

[ST\\_GetFaceEdges](#), [GetNodeEdges](#)

### 9.7.11 GetNodeEdges

`GetNodeEdges` — 返回与指定点相关的一有序边。

## Synopsis

```
getfaceedges_returntype GetNodeEdges(varchar atopology, integer anode);
```

## 描述

返回与指定点相关的一有序边。每个边由一个序列和一个边符号的 id 组成。序列号从 1 开始。上升沿从指定点开始。边沿束于指定点。封闭的边将出边次（都有边标志）。边序是从北行开始边方向。



### Note

函数按排序而不是从元数据出，因此可用于建边接。

可用性 : 2.0

## 相关信息

[getfaceedges\\_returntype](#), [GetRingEdges](#), [ST\\_Azimuth](#)

## 9.8 拓扑理

### 9.8.1 Polygonize

`Polygonize` — 找并注册由拓扑指定的所有面。

## Synopsis

```
text Polygonize(varchar toponame);
```

### 描述

注册可以创建拓扑基元的所有面。

假设拓扑不包含自相交。



#### Note

已知的面，因此在同一拓扑上多次用 Polygonize 是安全的。



#### Note

函数不使用也不设置表的 next\_left\_edge 和 next\_right\_edge 字段。

可用性: 2.0.0

### 相关信息

[AddFace](#), [ST\\_Polygonize](#)

## 9.8.2 AddNode

AddNode — 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。

### Synopsis

```
integer AddNode(varchar toponame, geometry apoint, boolean allowEdgeSplitting=false, boolean computeContainingFace=false);
```

### 描述

将点添加到指定拓扑方案的点表中。[AddEdge](#)函数在调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

```
SELECT topology.AddNode('ma_topo', ST_GeomFromText('POINT(227641.6 893816.5)', 26986) ) As ←
    nodeid;
-- result --
nodeid
-----
4
```

相关信息

[AddEdge](#), [CreateTopology](#)

### 9.8.3 AddEdge

**AddEdge** — 使用指定的几何串将串添加到表，并将串的起点和点添加到指定拓扑方案的点表，并返回新（或有的）的 `edgeid`。

#### Synopsis

```
integer AddEdge(varchar toponame, geometry aline);
```

描述

使用指定的几何串将串添加到表，并将串的起点和点添加到指定 `toponame` 模式的点表，并返回新或有的 `edgeid`。新添加的串都有“宇宙”面，并与其自身相交。



#### Note

如果 `aline` 几何串与现有串交叉、重叠、包含或被现有串包含，会引并且不会添加。



#### Note

`aline` 的几何形状必须具有与拓扑定的相同的 `srid`，否则将引无效的空参考系。

这个函数是由 GEOS 模块行的。



#### Warning

`AddEdge` is deprecated as of 3.5.0. Use `TopoGeo_AddLineString` instead.

可用性: 2.0.0

示例

```
SELECT topology.AddEdge('ma_topo', ST_GeomFromText('LINESTRING(227575.8 893917.2,227591.9 893900.4)'), 26986) ) As edgeid;
-- result-
edgeid
-----
1

SELECT topology.AddEdge('ma_topo', ST_GeomFromText('LINESTRING(227591.9 893900.4,227622.6 893844.2,227641.6 893816.5,
227704.5 893778.5)'), 26986) ) As edgeid;
-- result --
edgeid
-----
2

SELECT topology.AddEdge('ma_topo', ST_GeomFromText('LINESTRING(227591.2 893900, 227591.9 893900.4,
227704.5 893778.5)'), 26986) ) As edgeid;
-- gives error --
ERROR: Edge intersects (not on endpoints) with existing edge 1
```

相关信息

[TopoGeo\\_AddLineString](#), [CreateTopology](#), Section 4.5

## 9.8.4 AddFace

AddFace — 将面基元注册到拓扑并返回其标识符。

### Synopsis

```
integer AddFace(varchar toponame, geometry apolygon, boolean force_new=false);
```

描述

将面基元注册到拓扑并返回其标识符。

对于新添加的面，形成其边界以及面中包含的孔将更新在 `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具有正确的 `contains_face` 字段。



### 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）。如果出现以下情况，会引发异常：多边形边界未完全由有定义，或者多边形与有面重叠。

如果多边形几何体已作为面存在，则：如果 `force_new` 为假（默认），则返回已有面的面 id；如果 `force_new` 为 true，新的 id 将被分配给新产生的面。

**Note**

多面体行新配准 (`force_new=true`) 时，不会采取任何措施来解决点、点和关系表中面的挂引用，也不会更新面的 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，否则将引无效的空参考系。

可用性: 2.0.0

示例

```
-- first add the edges we use generate_series as an iterator (the below
-- will only work for polygons with < 10000 points because of our max in gs)
SELECT topology.AddEdge('ma_topo', ST_MakeLine(ST_PointN(geom,i), ST_PointN(geom, i + 1) )) ←
  As edgeid
  FROM (SELECT ST_NPoints(geom) AS npt, geom
        FROM
          (SELECT ST_Boundary(ST_GeomFromText('POLYGON((234896.5 899456.7,234914 ←
            899436.4,234946.6 899356.9,234872.5 899328.7,
            234891 899285.4,234992.5 899145, 234890.6 899069,234755.2 899255.4,
            234612.7 899379.4,234776.9 899563.7,234896.5 899456.7))', 26986) ) As geom
          ) As geoms) As facen CROSS JOIN generate_series(1,10000) As i
        WHERE i < npt;
-- result --
edgeid
-----
  3
  4
  5
  6
  7
  8
  9
 10
 11
 12
(10 rows)
-- then add the face -

SELECT topology.AddFace('ma_topo',
  ST_GeomFromText('POLYGON((234896.5 899456.7,234914 899436.4,234946.6 899356.9,234872.5 ←
    899328.7,
    234891 899285.4,234992.5 899145, 234890.6 899069,234755.2 899255.4,
    234612.7 899379.4,234776.9 899563.7,234896.5 899456.7))', 26986) ) As faceid;
-- result --
faceid
-----
  1
```

相关信息

[AddEdge](#), [CreateTopology](#), [Section 4.5](#)

## 9.8.5 ST\_Simplify

ST\_Simplify — 使用 Douglas-Peucker 算法返回给定 TopoGeometry 的“简化”几何版本。

### Synopsis

```
geometry ST_Simplify(TopoGeometry tg, float8 tolerance);
```

### 描述

在每个物件上使用 Douglas-Peucker 算法返回给定 TopoGeometry 的“简化”几何版本。



### Note

返回的几何形状可能不精确或无效。  
拆分物件可能有助于保持精确性/有效性。

该函数是由 GEOS 模块执行的。

可用性：2.1.0

### 相关信息

几何 [ST\\_Simplify](#), [ST\\_IsSimple](#), [ST\\_IsValid](#), [ST\\_ModEdgeSplit](#)

## 9.8.6 RemoveUnusedPrimitives

RemoveUnusedPrimitives — 删除给定 TopoGeometry 对象不需要的拓扑基元。

### Synopsis

```
int RemoveUnusedPrimitives(text topology_name, geometry bbox);
```

### 描述

该函数表示给定 TopoGeometry 对象并不严格需要的所有元素（点、面）并将其删除，从而保持拓扑有效性（连接、面）和 TopoGeometry 空间占用。

不会重建新的元素，而是展示具有元素以包括合并的面（在删除面）或修复的面（在删除点）。

可用性：3.3.0

### 相关信息

[ST\\_ModEdgeHeal](#), [ST\\_RemEdgeModFace](#)

## 9.9 拓扑几何构造函数

### 9.9.1 CreateTopoGeom

CreateTopoGeom — 从拓扑元素数组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);
```

```
topogeometry CreateTopoGeom(varchar toponame, integer tg_type, integer layer_id);
```

#### 描述

由 `layer_id` 表示的构建一个拓扑几何对象，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`。

点状由一多点形成，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：从线形成

在“`ri_topo`”模式中“`layer 2`”（我的“`ri_roads`”）构建一个拓扑几何对象，类型为 (2) LINE，用于第一条线（我在 `ST_CreateTopoGeo` 中加的线）。

```
INSERT INTO ri.ri_roads(road_name, topo) VALUES('Unknown', topology.CreateTopoGeom('ri_topo', 2, 2, '{1,2}'::topology.topoelementarray);
```

示例：将面状几何对象转换为最佳猜测的拓扑几何对象

假如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，我可以这样做：

```
-- create our topo geometry column --
SELECT topology.AddTopoGeometryColumn(
    'topo_boston',
    'boston', 'blockgroups', 'topo', 'POLYGON');

-- addtopogeometrycolumn --
1

-- update our column assuming
-- everything is perfectly aligned with our edges
UPDATE boston.blockgroups AS bg
    SET topo = topology.CreateTopoGeom('topo_boston',
    ,3,1
    , foo.bfaces)
FROM (SELECT b.gid, topology.TopoElementArray_Agg(ARRAY[f.face_id,3]) As bfaces
    FROM boston.blockgroups As b
```

```

        INNER JOIN topo_boston.face As f ON b.geom && f.mbr
        WHERE ST_Covers(b.geom, topology.ST_GetFaceGeometry('topo_boston', f.face_id))
        GROUP BY b.gid) As foo
WHERE foo.gid = bg.gid;

--the world is rarely perfect allow for some error
--count the face if 50% of it falls
-- within what we think is our blockgroup boundary
UPDATE boston.blockgroups AS bg
    SET topo = topology.CreateTopoGeom('topo_boston'
        ,3,1
        , foo.bfaces)
FROM (SELECT b.gid, topology.TopoElementArray_Agg(ARRAY[f.face_id,3]) As bfaces
    FROM boston.blockgroups As b
        INNER JOIN topo_boston.face As f ON b.geom && f.mbr
        WHERE ST_Covers(b.geom, topology.ST_GetFaceGeometry('topo_boston', f.face_id))
    OR
    ( ST_Intersects(b.geom, topology.ST_GetFaceGeometry('topo_boston', f.face_id))
        AND ST_Area(ST_Intersection(b.geom, topology.ST_GetFaceGeometry('topo_boston', ←
            f.face_id) ) ) >
            ST_Area(topology.ST_GetFaceGeometry('topo_boston', f.face_id))*0.5
        )
    GROUP BY b.gid) As foo
WHERE foo.gid = bg.gid;

-- and if we wanted to convert our topogeometry back
-- to a denormalized geometry aligned with our faces and edges
-- cast the topo to a geometry
-- The really cool thing is my new geometries
-- are now aligned with my tiger street centerlines
UPDATE boston.blockgroups SET new_geom = topo::geometry;
```

## 相关信息

[AddTopoGeometryColumn](#), [toTopoGeom](#) [ST\\_CreateTopoGeo](#), [ST\\_GetFaceGeometry](#), [TopoElementArray](#), [TopoElementArray\\_Agg](#)

## 9.9.2 toTopoGeom

toTopoGeom — 将几何形拓扑几何形。

### Synopsis

```

topogeometry toTopoGeom(geometry geom, varchar toponame, integer layer_id, float8 tolerance);
topogeometry toTopoGeom(geometry geom, topogeometry topogeom, float8 tolerance);
```

### 描述

将几何形拓扑几何形 **TopoGeometry**。

表示入几何所需的拓扑基元将被添加到底拓扑中，可能会拆分有拓扑，并且它将关系表中的 TopoGeometry 相关。

有的 TopoGeometry 象（拓扑几何可能除外，如果定的）将保留其形状。



当输出公差，它将用于将输入几何体捕捉到有元。

在第一种形式中，将指定拓扑 (toponame) 的指定层 (layer\_id) 建新的 TopoGeometry。

在第二种形式中，生成的元将被添加到先存在的 TopoGeometry (topogeom) 中，可能会其最形状添加空。要新形状完全取代旧形状，参考 `clearTopoGeom`。

可用性：2.0

增：2.1.0 添加了采用有 TopoGeometry 的版本。

## 示例

是一个完整的独立工作流程

```
-- do this if you don't have a topology setup already
-- creates topology not allowing any tolerance
SELECT topology.CreateTopology('topo_boston_test', 2249);
-- create a new table
CREATE TABLE nei_topo(gid serial primary key, nei varchar(30));
--add a topogeometry column to it
SELECT topology.AddTopoGeometryColumn('topo_boston_test', 'public', 'nei_topo', 'topo', ' ←
MULTIPOLYGON') As new_layer_id;
new_layer_id
-----
1

--use new layer id in populating the new topogeometry column
-- we add the topogeoms to the new layer with 0 tolerance
INSERT INTO nei_topo(nei, topo)
SELECT nei, topology.toTopoGeom(geom, 'topo_boston_test', 1)
FROM neighborhoods
WHERE gid BETWEEN 1 and 15;

--use to verify what has happened --
SELECT * FROM
    topology.TopologySummary('topo_boston_test');

-- summary--
Topology topo_boston_test (5), SRID 2249, precision 0
61 nodes, 87 edges, 35 faces, 15 topogeoms in 1 layers
Layer 1, type Polygonal (3), 15 topogeoms
Deploy: public.nei_topo.topo

-- Shrink all TopoGeometry polygons by 10 meters
UPDATE nei_topo SET topo = ST_Buffer(clearTopoGeom(topo), -10);

-- Get the no-one-lands left by the above operation
-- I think GRASS calls this "polygon0 layer"
SELECT ST_GetFaceGeometry('topo_boston_test', f.face_id)
FROM topo_boston_test.face f
WHERE f.face_id
> 0 -- don't consider the universe face
AND NOT EXISTS ( -- check that no TopoGeometry references the face
    SELECT * FROM topo_boston_test.relation
    WHERE layer_id = 1 AND element_id = f.face_id
);
```

相关信息

[CreateTopology](#), [AddTopoGeometryColumn](#), [CreateTopoGeom](#), [TopologySummary](#), [clearTopoGeom](#)

### 9.9.3 TopoElementArray\_Agg

TopoElementArray\_Agg — 返回一 `element_id`、`type` (topoelements) 的 `topoelementarray`。

#### Synopsis

`topoelementarray` **TopoElementArray\_Agg**(topoelement set tefield);

描述

用于从一 `TopoElement` 建 `TopoElementArray`。

可用性: 2.0.0

示例

```
SELECT topology.TopoElementArray_Agg(ARRAY[e,t]) As tea
FROM generate_series(1,3) As e CROSS JOIN generate_series(1,4) As t;
tea
```

```
-----
{{1,1},{1,2},{1,3},{1,4},{2,1},{2,2},{2,3},{2,4},{3,1},{3,2},{3,3},{3,4}}
```

相关信息

[TopoElement](#), [TopoElementArray](#)

### 9.9.4 TopoElement

TopoElement — 将拓扑几何 `topogeometry` 拓扑元素。

#### Synopsis

`topoelement` **TopoElement**(topogeometry topo);

描述

将 `TopoGeometry` 建 `TopoElement`。

可用性 : 3.4.0

示例

☒是一个完整的独立工作流程

```
-- do this if you don't have a topology setup already
-- Creates topology not allowing any tolerance
SELECT TopoElement(topo)
FROM neighborhoods;
```

```
-- using as cast
SELECT topology.TopoElementArray_Agg(topo::topoelement)
FROM neighborhoods
GROUP BY city;
```

相关信息

[TopoElementArray\\_Agg](#), [TopoGeometry](#), [TopoElement](#)

## 9.10 拓扑几何☒☒器

### 9.10.1 clearTopoGeom

clearTopoGeom — 清除拓扑几何的内容。

#### Synopsis

```
topogeometry clearTopoGeom(topogeometry topogeom);
```

描述

清除 [TopoGeometry](#) 的内容，将其☒☒空内容。主要与[toTopoGeom](#) ☒合使用，以替☒☒有☒象的形状以及更高☒次☒☒中的任何依☒☒象。

可用性：2.1

示例

```
-- Shrink all TopoGeometry polygons by 10 meters
UPDATE nei_topo SET topo = ST_Buffer(clearTopoGeom(topo), -10);
```

相关信息

[toTopoGeom](#)

### 9.10.2 TopoGeom\_addElement

TopoGeom\_addElement — 将元素添加到 TopoGeometry 的定☒中。

## Synopsis

topogeometry **TopoGeom\_addElement**(topogeometry tg, topoelement el);

### 描述

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
UPDATE mylayer SET tg = TopoGeom_addElement(tg, '{5,2}');
```

### 相关信息

[TopoGeom\\_remElement](#), [CreateTopoGeom](#)

## 9.10.3 TopoGeom\_remElement

TopoGeom\_remElement — 从 TopoGeometry 的定义中删除元素。

## Synopsis

topogeometry **TopoGeom\_remElement**(topogeometry tg, topoelement el);

### 描述

从 TopoGeometry 对象的定义中删除 **TopoElement**。

可用性：2.3

### 示例

```
-- Remove face 43 from TopoGeometry tg
UPDATE mylayer SET tg = TopoGeom_remElement(tg, '{43,3}');
```

### 相关信息

[TopoGeom\\_addElement](#), [CreateTopoGeom](#)

## 9.10.4 TopoGeom\_addTopoGeom

TopoGeom\_addTopoGeom — 将一个 TopoGeometry 的元素添加到一个 TopoGeometry 的定义中。

## Synopsis

```
topogeometry TopoGeom_addTopoGeom(topogeometry tgt, topogeometry src);
```

### 描述

将一个 **TopoGeometry** 的元素添加到一个 **TopoGeometry** 的定义中，如果需要保存源对象中的所有元素，可能会将其存型（存型属性）更改集合。

个 **TopoGeometry** 对象需要“相同”拓扑行定义，并且如果按次定义，需要由同一子的元素成。

可用性：3.2

### 示例

```
-- Set an "overall" TopoGeometry value to be composed by all
-- elements of specific TopoGeometry values
UPDATE mylayer SET tg_overall = TopoGeom_addTopoGeom(
    TopoGeom_addTopoGeom(
        clearTopoGeom(tg_overall),
        tg_specific1
    ),
    tg_specific2
);
```

### 相关信息

[TopoGeom\\_addElement](#), [clearTopoGeom](#), [CreateTopoGeom](#)

## 9.10.5 toTopoGeom

**toTopoGeom** — 将几何形状添加到有拓扑几何形中。

### 描述

参考 [toTopoGeom](#)。

## 9.11 拓扑几何器

### 9.11.1 GetTopoGeomElementArray

**GetTopoGeomElementArray** — 返回一个 `topoelementarray`（包含拓扑元素的数），其中包含定 **TopoGeometry** 的拓扑元素和型（原始元素）。

### Synopsis

```
topoelementarray GetTopoGeomElementArray(varchar toponame, integer layer_id, integer tg_id);
topoelementarray GetTopoGeomElementArray(topogeometry tg);
```

## 描述

返回一个 **TopoElementArray**，其中包含指定 **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](#)

### 9.11.2 GetTopoGeomElements

**GetTopoGeomElements** — 返回一个 `topoelement` 对象，其中包含指定 **TopoGeometry**（原始元素）的拓扑 `element_id`、`element_type`。

## Synopsis

```
setof topoelement GetTopoGeomElements(varchar toponame, integer layer_id, integer tg_id);
```

```
setof topoelement GetTopoGeomElements(topogeometry tg);
```

## 描述

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`），这些元素成了 `toponame` 模式中的指定拓扑几何对象。

`tg_id` 是 `topology.layer` 表中 `layer_id` 表示的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](#)

### 9.11.3 ST\_SRID

**ST\_SRID** — 返回拓扑几何的空参考标识符。

## Synopsis

```
integer ST_SRID(topogeometry tg);
```

## 描述

返回 ST\_Geometry spatial\_ref\_sys 表中定义的空参考系 ID 号。参见 [Section 4.5](#)



### Note

spatial\_ref\_sys 表包含 PostGIS 已知的所有参考系行项目，并用于从一个空参考系 ID 转换到一个空参考系。如果规划几何，必须确保具有正确的空参考系 ID 号。

可用性：3.2.0



该方法符合了 SQL/MM 规范。SQL-MM 3: 14.1.5

## 示例

```
SELECT ST_SRID(ST_GeomFromText('POINT(-71.1043 42.315)',4326));
-- result
4326
```

## 相关信息

[Section 4.5](#), [ST\\_SetSRID](#), [ST\\_Transform](#), [ST\\_SRID](#)

## 9.12 拓扑几何输出

### 9.12.1 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);
```

## 描述

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。nsprefix 是一个空字符串以获取非限定名称空间。精度（默认：15）和 id（默认：1）参数（如果指定）将原封不动地传递到 ST\_AsGML 的底层调用。

**VisitedTable** 参数 (如果指定) 用于跟踪 `Node` 和 `Edge` 元素, 以便使用交叉引用 (`xlink:xref`) 而不是重复定义。该表必须有 (至少) 两个整数字段: “`element_type`” 和 “`element_id`”。该表必须指定表具有取和写入权限。为了获得最佳性能, 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该表。例子:

```
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` 的标识符之前。

**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](#) 中建立的拓扑几何体

```
SELECT topology.AsGML(topo) As rdgml
FROM ri.roads
WHERE road_name = 'Unknown';

-- rdgml--
<gml:TopoCurve>
  <gml:directedEdge>
    <gml:Edge gml:id="E1">
      <gml:directedNode orientation="-">
        <gml:Node gml:id="N1"/>
      </gml:directedNode>
    </gml: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☐☐相同，没有命名空☐

```
SELECT topology.AsGML(topo, '') As rdgml
FROM ri.roads
WHERE road_name = 'Unknown';

-- rdgml--
<TopoCurve>
  <directedEdge>
    <Edge id="E1">
      <directedNode orientation="-">
        <Node id="N1"/>
      </directedNode>
      <directedNode
></directedNode>
      <curveProperty>
        <Curve srsName="urn:ogc:def:crs:EPSG::3438">
          <segments>
            <LineStringSegment>
              <posList srsDimension="2"
>384744 236928 384750 236923 384769 236911 384799 236895 384811 236890
384833 236884 384844 236882 384866 236881 384879 236883 384954 ←
236898 385087 236932 385117 236938
385167 236938 385203 236941 385224 236946 385233 236950 385241 ←
236956 385254 236971
385260 236979 385268 236999 385273 237018 385273 237037 385271 ←
237047 385267 237057 385225 237125
385210 237144 385192 237161 385167 237192 385162 237202 385159 ←
237214 385159 237227 385162 237241
385166 237256 385196 237324 385209 237345 385234 237375 385237 ←
237383 385238 237399 385236 237407
385227 237419 385213 237430 385193 237439 385174 237451 385170 ←
237455 385169 237460 385171 237475
385181 237503 385190 237521 385200 237533 385206 237538 385213 ←
237541 385221 237542 385235 237540 385242 237541
385249 237544 385260 237555 385270 237570 385289 237584 385292 ←
237589 385291 237596 385284 237630</posList>
            </LineStringSegment>
          </segments>
        </Curve>
      </curveProperty>
    </Edge>
  </directedEdge>
</TopoCurve>
```

相关信息

[CreateTopoGeom](#), [ST\\_CreateTopoGeo](#)

## 9.12.2 AsTopoJSON

AsTopoJSON — 返回拓扑几何的 TopoJSON 表示形式。

### Synopsis

```
text AsTopoJSON(topogeometry tg, regclass edgeMapTable);
```

## 描述

返回拓扑几何的 TopoJSON 表示形式。如果 `edgeMapTable` 不为空，它将用作弧索引的查找/存储映射。它是为了能在最文档中允许弧的“arcs”数组。

表（如果指定）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”，因此建在弧字段上添加索引。



### 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 文档需要包含弧和一些元数据。参 [TopoJSON 规范](#)。

可用性：2.1.0

增：2.2.1 添加了 `puntal` 输入的支持

## 相关信息

### ST\_AsGeoJSON

## 示例

```
CREATE TEMP TABLE edgemap(arc_id serial, edge_id int unique);

-- header
SELECT '{ "type": "Topology", "transform": { "scale": [1,1], "translate": [0,0] }, "objects ←
": {'

-- objects
UNION ALL SELECT '' || feature_name || ': ' || AsTopoJSON(feature, 'edgemap')
FROM features.big_parcel WHERE feature_name = 'P3P4';

-- arcs
WITH edges AS (
  SELECT m.arc_id, e.geom FROM edgemap m, city_data.edge e
  WHERE e.edge_id = m.edge_id
), points AS (
  SELECT arc_id, (st_dumppoints(geom)).* FROM edges
), compare AS (
  SELECT p2.arc_id,
     CASE WHEN p1.path IS NULL THEN p2.geom
          ELSE ST_Translate(p2.geom, -ST_X(p1.geom), -ST_Y(p1.geom))
     END AS geom
  FROM points p2 LEFT OUTER JOIN points p1
  ON ( p1.arc_id = p2.arc_id AND p2.path[1] = p1.path[1]+1 )
  ORDER BY arc_id, p2.path
), arcdump AS (
  SELECT arc_id, (regexp_matches( ST_AsGeoJSON(geom), '\[.*\]'))[1] as t
  FROM compare
), arcs AS (
  SELECT arc_id, '[' || array_to_string(array_agg(t), ',') || ']' as a FROM arcdump
  GROUP BY arc_id
  ORDER BY arc_id
)
SELECT ', "arcs": [' UNION ALL
```

```

SELECT array_to_string(array_agg(a), E',\n') from arcs

-- footer
UNION ALL SELECT '}]':::text as t;

-- Result:
{ "type": "Topology", "transform": { "scale": [1,1], "translate": [0,0] }, "objects": {
"P3P4": { "type": "MultiPolygon", "arcs": [[[ -1]], [[6,5,-5,-4,-3,1]]]}
}, "arcs": [
[[25,30],[6,0],[0,10],[-14,0],[0,-10],[8,0]],
[[35,6],[0,8]],
[[35,6],[12,0]],
[[47,6],[0,8]],
[[47,14],[0,8]],
[[35,22],[12,0]],
[[35,14],[0,8]]
]]}

```

## 9.13 拓扑空☒关系

### 9.13.1 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);

#### 描述

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

#### 相关信息

[GetTopoGeomElements](#), [ST\\_Equals](#)

## 9.13.2 Intersects

`Intersects` — 如果  $n$  个拓扑几何中的任何一  $n$  元相交，返回 `true`。

### Synopsis

```
boolean Intersects(topogeometry tg1, topogeometry tg2);
```

### 描述

如果  $n$  个拓扑几何中的任何一  $n$  元相交，返回 `true`。



#### Note

作  $n$  几何集合的拓扑几何不支持此函数。它也无法比  $n$  不同拓扑的拓扑几何形状。目前也不支持分  $n$  拓扑几何（由其他拓扑几何  $n$  成的拓扑几何）。

可用性：1.1.0



$n$  函数支持 3d 并且不会  $n$  失 `z-index`。

### 示例

### 相关信息

[ST\\_Intersects](#)

## 9.14 $n$ 入和 $n$ 出拓扑

$n$ 建拓扑以及关  $n$ 的拓扑  $n$ 后，您可能希望将它  $n$   $n$ 出  $n$ 基于文件的格式以  $n$ 行  $n$ 份或  $n$   $n$ 到  $n$ 一个数据  $n$ 中。

使用 PostgreSQL 的  $n$ 准  $n$   $n$ /恢复工具有  $n$   $n$ 的，因  $n$ 拓扑由一  $n$ 表（基元  $n$  4 个，  $n$   $n$ 任意数量）和元数据表中的  $n$   $n$ （`topology.topology` 和 `topology.layer`）  $n$ 成。此外，拓扑  $n$   $n$ 符在数据  $n$ 中并不是唯一的，因此在恢复拓扑  $n$ 需要更改拓扑参数。

$n$ 了  $n$ 化拓扑的  $n$ 出/恢复，提供了一  $n$ 可  $n$ 行文件：`pgtopo_export` 和 `pgtopo_import`。用法示例：

```
pgtopo_export dev_db topo1 | pgtopo_import topo1 | psql staging_db
```

### 9.14.1 使用拓扑导出器

`pgtopo_export` 脚本采用数据表和拓扑的名称，并导出文件，该文件可用于将拓扑（和关系的）导入到新数据库中。

默认情况下，`pgtopo_export` 将文件写入标准输出，以便可以将其通过管道到 `pgtopo_import` 或重定向到文件（拒绝写入终端）。您可以使用 `-f` 命令行开关指定输出文件名。

默认情况下，`pgtopo_export` 包含指定拓扑定义的所有表。可能是比您需要的更多的数据，或者可能不起作用（如果您的表具有复值的依赖关系），在这种情况下，您可以使用 `--skip-layers` 开关来跳过并独立处理某些表。

用 `pgtopo_export` 使用 `--help`（或写的 `-h`）指令将帮助打印简短的用法明字符串。

输出文件格式是 `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）、与其关系的拓扑和以及（除非输出 `--skip-layers`）自定义 `-format PostgreSQL` 表报告自定义拓扑的。

### 9.14.2 使用拓扑导入器

`pgtopo_import` 脚本采用 `pgtopo_export` 格式的拓扑和要重建的拓扑的名称，并输出重建拓扑和关系的 SQL 脚本。

生成的 SQL 文件将包含以下语句：创建具有指定名称的拓扑、在其中加载原始数据、通将所有 TopoGeometry 正确连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` 从标准输入读取，以便它可以与管道中的 `pgtopo_export` 组合使用。您可以使用 `-f` 命令行开关指定输入文件名。

默认情况下，`pgtopo_import` 在输出 SQL 文件中包含用于恢复表中找到的所有表的代。

如果您的目标数据库已具有与表同名的表，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` 命令来跳过并独立（或稍后）处理某些表。

可以使用 `--only-layers` 命令生成附加并将表连接到命名拓扑的 SQL。用于在解决命名冲突后附加或将表连接到不同的拓扑（例如起始拓扑的空壳化版本）非常有用。

## Chapter 10

# 栅格数据管理、索引和应用程序

### 10.1 加载和创建栅格

对于大多数用例，您将通常使用打包的 `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（除非您自己的 GDAL 包），因此可执行文件支持的栅格类型将与 GDAL 依赖包中的栅格类型相同。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，有可能概图不会生成。看看 <http://trac.osgeo.org/postgis/ticket/1764> 查看一个示例，其中概图未生成。

##### 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：

```
# -s use srid 4326
# -I create spatial index
# -C use standard raster constraints
# -M vacuum analyze after load
# *.tif load all these files
# -F include a filename column in the raster table
# -t tile the output 100x100
# public.demelevation load into this table
raster2pgsql -s 4326 -I -C -M -F -t 100x100 *.tif public.demelevation
> elev.sql

# -d connect to this database
# -f read this file after connecting
psql -d gisdb -f elev.sql
```

**Note**

如果您没有将架指定为目录表名称的一部分，表将在您所连接的数据库或用数据库的默认架中创建。

使用 UNIX 管道可以一步完成创建和上：

```
raster2pgsql -s 4326 -I -C -M *.tif -F -t 100x100 public.demelevation | psql -d gisdb
```

将塞州平面米制航空瓦片加载到名为 `aerial` 的架中，并创建一个全表，2 和 4 概表，使用复制模式进行插入（没有中间文件，直接到数据库），并且使用“-e”选项来避免限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，并用栅格约束。同时，包括一个名为“filename”的字段，用于存储瓦片所来自的文件名称。

```
raster2pgsql -I -C -e -Y -F -s 26986 -t 128x128 -l 2,4 bostonaerials2008/*.jpg aerials. ↵
  boston | psql -U postgres -d gisdb -h localhost -p 5432
```

--get a list of raster types supported:

```
raster2pgsql -G
```

-G 指令输出似的列表

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

-? 显示帮助屏幕。如果您不输入任何参数，也会显示帮助。

-G 打印支持的栅格格式。

(c|a|d|p) 有些是相互排斥的选项：

- c 创建新表并用栅格填充它，是默认模式
- a 将栅格附加到有表。
- d 删除表，创建新表并用栅格填充它
- p 准模式，只创建表。

栅格选项：用选项束以在栅格目录中正确注册

- C 用栅格约束——srid、像素大小等，以确保栅格在 `raster_columns` 表中正确配准。
- x 禁用设置最大范围约束。当使用 -C 指令时才适用。
- r 设置范围约束（空唯一和无尽的切片）。当使用 -C 指令时才适用。

栅格选项：用于操作输入栅格数据集的可选项

- s <SRID> 使用指定的 SRID 分配输出栅格。如果未提供或为零，则将栅格的元数据以确定适当的 SRID。
- b 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 将光栅切割成瓦片，以便在每个表行插入一个瓦片。TILE\_SIZE 表示瓦片 WIDTHxHEIGHT 或置瓦片为“auto”，以允许加载程序使用第一个栅格计算适当的切片大小并将其用于所有栅格。
- P 填充最右和最底部的瓦片，以确保所有瓦片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目录中（而不是像素）。
- l OVERVIEW\_FACTOR 构建栅格的概图。用于多个因素，用逗号（,）分隔。概图表名称遵循 o\_overview\_factor\_table 模式，其中概图因子是数字概图因子的占位符，表将替瓦片基表名称。构建的概述存储在数据目录中，不受-R 影响。注意，生成的 sql 文件将包含主表和概图表。
- N NODATA NODATA 用于没有 NODATA 瓦片的瓦片。

用于操作数据目录象的可选项参数

- f COLUMN 指定目标栅格列的名称，默认为“rast”
- F 添加包含文件名的列
- n COLUMN 指定文件名列的名称。意味着-F。
- q 将 PostgreSQL 符号括在引号中。
- I 在栅格列上构建 GiST 索引。
- M 真空分析栅格表。
- k 保留空的瓦片并跳过每个光栅波段的 NODATA 瓦片。注意，瓦片可以节省空间的瓦片，但可能会导致数据目录中输出更多的瓦片行，并且某些瓦片行不会输出空的瓦片。
- T tablespace 指定新表的表空间。注意，索引（包括主索引）仍将使用默认表空间，除非您使用了-X 标志。
- X 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 max\_rows\_per\_copy=50 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 独立行每条语句，不使用事务。
- E ENDIAN 控制生成的光栅二进制制出的字节序；XDR 指定 0，NDR 指定 1（默认）；在支持 NDR 输出
- V 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 栅格函数构建栅格

在多种情况下，您需要直接在数据目录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瓦片，可以通过以下方式完成：

```
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。  
甚至有更多瓦片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：



```
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：能从远程存储在云“对象存储”（例如 AWS S3、Google Cloud Storage）中的栅格中取数据。

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，因为系统每次需要栅格子集都必须下载整个栅格。

首先，将栅格添加到您的云的存储中。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”）。要访问非公共存储桶，您需要提供 GDAL 配置来访问您的连接。注意，此命令是从云栅格取并写入数据。

```
AWS_ACCESS_KEY_ID=xxxxxxxxxxxxxxxxxxxxx \
AWS_SECRET_ACCESS_KEY=xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx \
raster2pgsql \
-s 990000 \
-t 256x256 \
-I \
-R \
/vs3/your.bucket.com/your_file.tif \
your_table \
| psql your_db
```

添加表后，您需要通过配置两个限制（`postgis.enable_outdb_rasters` 和 `postgis.gdal_enabled_drivers`）来授予数据从远程栅格取的权限。

```
SET postgis.enable_outdb_rasters = true;
SET postgis.gdal_enabled_drivers TO 'ENABLE_ALL';
```

为了使更改生效并持久化，直接在您的数据库上执行配置。您将需要重新连接以体现新的配置。

```
ALTER DATABASE your_db SET postgis.enable_outdb_rasters = true;
ALTER DATABASE your_db SET postgis.gdal_enabled_drivers TO 'ENABLE_ALL';
```

对于非公开的栅格数据，您可能需要提供密钥来从云端栅格数据中取。你可以将写入 `raster2pgsql` 使用的相同密钥设置在数据库内部使用，使用 `postgis.gdal_vsi_options` 配置。注意，可以通过空格分隔 `key=value` 来设置多个配置。

```
SET postgis.gdal_vsi_options = 'AWS_ACCESS_KEY_ID=xxxxxxxxxxxxxxxxxxxxx
AWS_SECRET_ACCESS_KEY=xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx';
```

添加数据并配置限制后，您可以使用相同的功能像任何其他栅格表一样与栅格表交互。当需要取像素数据，数据将管理连接到云数据的所有机制。

## 10.2 栅格目录

PostGIS 附带了两个栅格目录。两个目录都利用嵌入在栅格表约束中的信息。因此，由于限制了约束，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项目。
2. `raster_overviews` 此目录数据中的所有栅格表列行项目，有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。

### 10.2.1 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` 表所在的数据库。将始终取当前数据库。
- `r_table_schema` 栅格表所属的数据库模式。
- `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拥有多个栅格列，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。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` 每个栅格的度（跨像素数）。有关更多信息，参看 `ST_Width`。
- `blocksize_y` 每个栅格的度（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。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波段的数量相同。这些数字定义了大多数操作中忽略的每个波段的像素。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库外部。数组中的元素数量与波段的数量相同。
- `extent` 是栅格集中所有栅格行的范围。如果您计划加更多数据来更改集的范围，需要在加之前使用 `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。

## 10.2.2 概图概述

`raster_overviews` 目录有关用于概图的栅格表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，因为它本身就是栅格，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。当您在栅格添加中使用 `-l` 开关时，它会与主栅格表一起生成，或者可以使用 `AddOverviewConstraints` 手动生成。

概图表包含与其他栅格表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` 连接在一起以得到所需的完整信息集。

进行概图的主要原因有：

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，因为它包含更少的像素，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，但它在许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。将起始时取当前数据库。
- `o_table_schema` 概图栅格表所属的数据库架构。
- `o_table_name` 栅格概图表名称
- `o_raster_column` 概图表中的栅格列。
- `r_table_catalog` 概图服务所在的栅格表所在的数据库。将起始时取当前数据库。
- `r_table_schema` 此概图服务所属的栅格表的数据架构。
- `r_table_name` 此概图服务的栅格表。
- `r_raster_column` 此概图列服务的栅格列。
- `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添加。假定因子为 1，并且起始时原始文件。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，您将其分为 125x125，那么对于每个图像文件，您的基表将有  $(5000*5000)/(125*125) = 1600$  条记录，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，您将得到一些未完全填充的因子。注意，`raster2pgsql` 生成的每个概图因子与其父因子具有相同数量的像素，但分辨率低，其中每个像素代表（原始因子的  $Power(2,overview\_factor)$  像素）。

## 10.3 使用 PostGIS Raster 构建自定义应用程序

事实上，PostGIS 栅格为您提供 SQL 函数来以已知图像格式渲染栅格，您提供了很多渲染栅格的选项。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，您还可以使用本章节中演示的多种语言。

### 10.3.1 PHP 示例使用 ST\_AsPNG 与其他栅格函数行出

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` 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 `png`。

您可以使用

```
http://mywebserver/test_raster.php?srid=2249
```

用以下命令来获取塞州平面英尺的栅格像。

```
<?php
/** contents of test_raster.php */
$conn_str = 'dbname=mydb host=localhost port=5432 user=myuser password=mypwd';
$dbconn = pg_connect($conn_str);
header('Content-Type: image/png');
/**If a particular projection was requested use it otherwise use mass state plane meters ←
**/
if (!empty( $_REQUEST['srid'] ) && is_numeric( $_REQUEST['srid'] )) {
    $input_srid = intval($_REQUEST['srid']);
}
else { $input_srid = 26986; }
/** The set bytea_output may be needed for PostgreSQL 9.0+, but not for 8.4 **/
$sql = "set bytea_output='escape';
SELECT ST_AsPNG(ST_Transform(
    ST_AddBand(ST_Union(rast,1), ARRAY[ST_Union(rast,2),ST_Union(rast ←
    ,3]))
    , $input_srid) ) As new_rast
FROM aerials.boston
WHERE
    ST_Intersects(rast, ST_Transform(ST_MakeEnvelope(-71.1217, 42.227, -71.1210, ←
    42.218,4326),26986) )";
$result = pg_query($sql);
$row = pg_fetch_row($result);
pg_free_result($result);
if ($row === false) return;
echo pg_unescape_bytea($row[0]);
?>
```

### 10.3.2 ASP.NET C# 示例使用 ST\_AsPNG 与其他栅格函数行出

在本节中，我将演示如何使用 Npgsql PostgreSQL .NET 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。

本示例需要 `nppgsql .NET PostgreSQL` 程序，您可以从 <http://nppgsql.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` 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 `png`。

示例与 Section 10.3.1 示例相同，只不过是在 C# 中实现的。

您可以使用

```
http://mywebserver/TestRaster.ashx?srid=2249
```

☒用以下命令来☒取☒☒☒塞州平面英尺的☒格☒像。

```
-- web.config connection string section --
<connectionStrings>
  <add name="DSN"
    connectionString="server=localhost;database=mydb;Port=5432;User Id=myuser;password= ←
    mypwd"/>
</connectionStrings>
```

```
// Code for TestRaster.ashx
<%@ WebHandler Language="C#" Class="TestRaster" %>
using System;
using System.Data;
using System.Web;
using Npgsql;

public class TestRaster : IHttpHandler
{
    public void ProcessRequest(HttpContext context)
    {
        context.Response.ContentType = "image/png";
        context.Response.BinaryWrite(GetResults(context));
    }

    public bool IsReusable {
        get { return false; }
    }

    public byte[] GetResults(HttpContext context)
    {
        byte[] result = null;
        NpgsqlCommand command;
        string sql = null;
        int input_srid = 26986;
    try {
        using (NpgsqlConnection conn = new NpgsqlConnection(System. ←
            Configuration.ConfigurationManager.ConnectionStrings["DSN"]. ←
            ConnectionString)) {
            conn.Open();

            if (context.Request["srid"] != null)
            {
                input_srid = Convert.ToInt32(context.Request["srid"]);
            }
            sql = @"SELECT ST_AsPNG(
                ST_Transform(
                    ST_AddBand(
                        ST_Union(rast,1), ARRAY[ST_Union(rast,2),ST_Union(rast,3)])
                    ,:input_srid) ) As new_rast
                FROM aerials.boston
                WHERE
                    ST_Intersects(rast,
                        ST_Transform(ST_MakeEnvelope(-71.1217, 42.227, ←
                            -71.1210, 42.218,4326),26986) )";
            command = new NpgsqlCommand(sql, conn);
            command.Parameters.Add(new NpgsqlParameter("input_srid", input_srid));

            result = (byte[]) command.ExecuteScalar();
            conn.Close();
        }
    }
}
```

```

        }
    }
    catch (Exception ex)
    {
        result = null;
        context.Response.Write(ex.Message.Trim());
    }
    return result;
}
}
}

```

### 10.3.3 将栅格输出为像文件的 Java 控制台应用程序

这是一个简单的 java 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。

您可以从 <http://jdbc.postgresql.org/download.html> 下载最新的 PostgreSQL JDBC 程序

您可以使用如下命令以下代码：

```

set env CLASSPATH ..\postgresql-9.0-801.jdbc4.jar
javac SaveQueryImage.java
jar cfm SaveQueryImage.jar Manifest.txt *.class

```

并使用类似的命令从命令行用它

```

java -jar SaveQueryImage.jar "SELECT ST_AsPNG(ST_AsRaster(ST_Buffer(ST_Point(1,5),10, ' ←
quad_segs=2'),150, 150, '8BUI',100));" "test.png"

```

```

-- Manifest.txt --
Class-Path: postgresql-9.0-801.jdbc4.jar
Main-Class: SaveQueryImage

```

```

// Code for SaveQueryImage.java
import java.sql.Connection;
import java.sql.SQLException;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.io.*;

public class SaveQueryImage {
    public static void main(String[] argv) {
        System.out.println("Checking if Driver is registered with DriverManager.");

        try {
            //java.sql.DriverManager.registerDriver (new org.postgresql.Driver());
            Class.forName("org.postgresql.Driver");
        }
        catch (ClassNotFoundException cnfe) {
            System.out.println("Couldn't find the driver!");
            cnfe.printStackTrace();
            System.exit(1);
        }

        Connection conn = null;

        try {
            conn = DriverManager.getConnection("jdbc:postgresql://localhost:5432/mydb","myuser ←
            ", "mypwd");
            conn.setAutoCommit(false);

```

```

PreparedStatement sGetImg = conn.prepareStatement(argv[0]);

ResultSet rs = sGetImg.executeQuery();

    FileOutputStream fout;
    try
    {
        rs.next();
        /** Output to file name requested by user */
        fout = new FileOutputStream(new File(argv[1]) );
        fout.write(rs.getBytes(1));
        fout.close();
    }
    catch(Exception e)
    {
        System.out.println("Can't create file");
        e.printStackTrace();
    }

    rs.close();
    sGetImg.close();
    conn.close();
}
catch (SQLException se) {
    System.out.println("Couldn't connect: print out a stack trace and exit.");
    se.printStackTrace();
    System.exit(1);
}
}
}
}

```

### 10.3.4 使用 PLPython 通 SQL 像

是一个 plpython 存储函数，它在服务器目录中每条记录建立一个文件。需要你安装了 plpython。可以与 plpythonu 和 plpython3u 一起正常工作。

```

CREATE OR REPLACE FUNCTION write_file (param_bytes bytea, param_filepath text)
RETURNS text
AS $$
f = open(param_filepath, 'wb+')
f.write(param_bytes)
return param_filepath
$$ LANGUAGE plpythonu;

```

```

--write out 5 images to the PostgreSQL server in varying sizes
-- note the postgresql daemon needs to have write access to folder
-- this echos back the file names created;
SELECT write_file(ST_AsPNG(
    ST_AsRaster(ST_Buffer(ST_Point(1,5),j*5, 'quad_segs=2'),150*j, 150*j, '8BUI',100)),
    'C:/temp/slices'|| j || '.png')
FROM generate_series(1,5) As j;

```

```
write_file
```

```

-----
C:/temp/slices1.png
C:/temp/slices2.png
C:/temp/slices3.png
C:/temp/slices4.png

```

```
C:/temp/slices5.png
```

### 10.3.5 使用 PSQL 导出数据

遗憾的是，PSQL 没有易于使用的内置功能来导出二进制文件。这是一个有点依赖于 PostgreSQL 遗留的大对象支持的 hack。要使用，首先连接到数据库的 psql 命令行。

与 python 方法不同，此方法在本地计算机上创建文件。

```
SELECT oid, lowrite(lo_open(oid, 131072), png) As num_bytes
FROM
( VALUES (lo_create(0),
  ST_AsPNG( (SELECT rast FROM aerials.boston WHERE rid=1) )
) ) As v(oid,png);
-- you'll get an output something like --
oid | num_bytes
-----+-----
2630819 | 74860

-- next note the oid and do this replacing the c:/test.png to file path location
-- on your local computer
\lo_export 2630819 'C:/temp/aerial_samp.png'

-- this deletes the file from large object storage on db
SELECT lo_unlink(2630819);
```





## 11.1 栅格支持数据类型

### 11.1.1 geomval

**geomval** — 具有 `n` 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)。

#### 描述

**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` 是一个双精度浮点型，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。

#### 描述

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。

**index integer** 从 1 开始的整数，指示新波段将添加到栅格波段中的位置。如果为 NULL，新波段将添加到栅格波段的末尾。

**pixeltype text** 新波段的像素类型。ST\_BandPixelType 中描述的已定义像素类型之一。

**initialvalue double precision** 新波段的所有像素将设置的初始值。

**nodataval double precision** 新波段的 NODATA 值。如果为 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。

#### 描述

需要表栅格和栅格的波段索引使用的复合类型。

**rast raster** 有 `n` 个的栅格/

**nband integer** 从 1 开始的整数，表示栅格的波段

相关信息

**ST\_MapAlgebra (callback function version)**

**11.1.4 raster**

raster — 栅格空数据型。

描述

栅格是一种空数据类型，用于表示栅格数据，例如从 JPEG、TIFF、PNG、数字高程模型输入的栅格数据。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 — 用作定重新分行行的 ST\_Reclass 函数的入的复合型。

描述

用作定重新分行行的 ST\_Reclass 函数的入的复合型。

**nband integer** 要重新分的波段的波段号。

**reclassexpr text** 由逗号分隔的 range:map\_range 映射成的范表示式。: 定映射，定如何将旧波段映射到新波段。( 表示 >, ) 表示 <, ] 表示 < 或 =, [ 表示 > 或 =

1. [a-b] = a <= x <= b
2. (a-b) = a < x <= b
3. [a-b) = a <= x < b
4. (a-b) = a < x < b

(符号是可空的, 因此 a-b 与 (a-b) 的含空相同)

**pixeltype text** [ST\\_BandPixelType](#)中描述的已定像素型之一

**nodataval double precision** 无数据的空。用于支持透明度的像输出, 空将空。

示例: 将波段 **2** 重新分 **8BUI**, 其中 **255** 是无数据

```
SELECT ROW(2, '0-100:1-10, 101-500:11-150,501 - 10000: 151-254', '8BUI', 255)::reclassarg;
```

示例: 将波段 **1** 重新分 **1BB** 并且未定 **nodata**

```
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](#) 函数返回的复合型。

描述

[ST\\_SummaryStats](#) 和 [ST\\_SummaryStatsAgg](#) 函数返回的复合型。

**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](#) 函数的入的复合型, 定要理的波段和 UNION 操作的行。

## 描述

用作 `ST_Union` 函数的输入复合类型，定义要处理的波段和 UNION 操作的行。

**nband integer** 从 1 开始的，指示要处理的每个输入栅格的波段。

**uniontype text** UNION 操作的类型。 `ST_Union` 中描述的已定义类型之一。

## 相关信息

[ST\\_Union](#)

## 11.2 栅格管理

### 11.2.1 AddRasterConstraints

`AddRasterConstraints` — 将栅格约束添加到已添加的栅格表中，用于特定列，列约束了空参考、比例、大小、波段、波段类型以及一个标志，用于表示栅格列是否被本地分区。表必须添加数据才能推断出约束。如果约束设置成功，返回 `true`，否则会发出通知。

## 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);
```

## 描述

在栅格列上生成约束，用于指示 `raster_columns` 栅格目录中的信息。 `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 10.2.1](#)。

- `blocksize` 设置 X 和 Y 大小
- `blocksize_x` 设置 X 个块（每个块的宽度（以像素位））
- `blocksize_y` 设置 Y 块（每个块的像素高度）
- `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





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);

#### 描述

添加栅格列的概述，用于显示 raster\_overviews 视图中的信息。

ovfactor 参数表示概述列中的比例乘数：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

```
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);

-- verify if registered correctly in the raster_overviews view --
SELECT o_table_name ot, o_raster_column oc,
       r_table_name rt, r_raster_column rc,
       overview_factor f
FROM raster_overviews WHERE o_table_name = 'res2';
  ot | oc | rt | rc | f
-----+-----+-----+-----+
res2 | r2 | res1 | r1 | 2
(1 row)
```

相关信息

Section [10.2.2, DropOverviewConstraints, ST\\_CreateOverview, AddRasterConstraints](#)



## 11.2.4 DropOverviewConstraints

DropOverviewConstraints — 取消栅格列作一栅格列概述的。

### Synopsis

```
boolean DropOverviewConstraints(name ovschema, name ovtable, name ovcolumn);
boolean DropOverviewConstraints(name ovtable, name ovcolumn);
```

### 描述

从栅格列中删除用于将其显示 raster\_overviews 栅格目录中一个栅格列的概述的。

当省略 ovschema 参数，将使用搜索 search\_path 找到的第一个表。

可用性: 2.0.0

### 相关信息

Section [10.2.2, 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();
```

### 描述

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

```
SELECT PostGIS_GDAL_Version();
       postgis_gdal_version
-----
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 — 报告完整的栅格库构建日期。

## Synopsis

```
text PostGIS_Raster_Lib_Build_Date();
```

### 描述

报告栅格建日期

### 示例

```
SELECT PostGIS_Raster_Lib_Build_Date();
postgis_raster_lib_build_date
-----
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建配置信息。

## Synopsis

```
text PostGIS_Raster_Lib_Version();
```

### 描述

报告完整的栅格版本和建配置信息。

### 示例

```
SELECT PostGIS_Raster_Lib_Version();
postgis_raster_lib_version
-----
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);

### 描述

返回 GDAL 支持的每种格式的短名称、名称和创建者的列表。使用短名称作格式参数 **ST\_AsGDALRaster** 的输入。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 - 默认情况下不用任何程序，除非置了 GUC 或环境变量 `gdal_enabled_drivers`。

可用性：2.0.0 - 需要 GDAL >= 1.6.0。

示例：程序列表

```
SET postgis.gdal_enabled_drivers = 'ENABLE_ALL';
SELECT short_name, long_name, can_write
FROM st_gdaldrivers()
ORDER BY short_name;
```

short_name	long_name	can_write
AAIGrid	Arc/Info ASCII Grid	t
ACE2	ACE2	f
ADRG	ARC Digitized Raster Graphics	f
AIG	Arc/Info Binary Grid	f
AirSAR	AirSAR Polarimetric Image	f
ARG	Azavea Raster Grid format	t
BAG	Bathymetry Attributed Grid	f
BIGGIF	Graphics Interchange Format (.gif)	f
BLX	Magellan topo (.blx)	t
BMP	MS Windows Device Independent Bitmap	f
BSB	Maptech BSB Nautical Charts	f
PAux	PCI .aux Labelled	f
PCIDSK	PCIDSK Database File	f
PCRaster	PCRaster Raster File	f
PDF	Geospatial PDF	f
PDS	NASA Planetary Data System	f
PDS4	NASA Planetary Data System 4	t
PLMOosaic	Planet Labs Mosaics API	f
PLSCENES	Planet Labs Scenes API	f
PNG	Portable Network Graphics	t
PNM	Portable Pixmap Format (netpbm)	f
PRF	Racurs PHOTOMOD PRF	f
R	R Object Data Store	t
Rasterlite	Rasterlite	t
RDA	DigitalGlobe Raster Data Access driver	f
RIK	Swedish Grid RIK (.rik)	f
RMF	Raster Matrix Format	f
ROI_PAC	ROI_PAC raster	f
RPFTOC	Raster Product Format TOC format	f
RRASTER	R Raster	f
RS2	RadarSat 2 XML Product	f
RST	Idrisi Raster A.1	t

SAFE	Sentinel-1 SAR SAFE Product	f
SAGA	SAGA GIS Binary Grid (.sdat, .sg-grd-z)	t
SAR_CEOS	CEOS SAR Image	f
SDTS	SDTS Raster	f
SENTINEL2	Sentinel 2	f
SGI	SGI Image File Format 1.0	f
SNODAS	Snow Data Assimilation System	f
SRP	Standard Raster Product (ASRP/USRP)	f
SRTMHGT	SRTMHGT File Format	t
Terragen	Terragen heightfield	f
TIL	EarthWatch .TIL	f
TSX	TerraSAR-X Product	f
USGSDEM	USGS Optional ASCII DEM (and CDED)	t
VICAR	MIPL VICAR file	f
VRT	Virtual Raster	t
WCS	OGC Web Coverage Service	f
WMS	OGC Web Map Service	t
WMTS	OGC Web Map Tile Service	t
XPM	X11 PixMap Format	t
XYZ	ASCII Gridded XYZ	t
ZMap	ZMap Plus Grid	t

示例：每个程序的选项列表

```
-- Output the create options XML column of JPEG as a table --
-- Note you can use these creator options in ST_AsGDALRaster options argument
SELECT (xpath('@name', g.opt))[1]::text As oname,
       (xpath('@type', g.opt))[1]::text As otype,
       (xpath('@description', g.opt))[1]::text As descrip
FROM (SELECT unnest(xpath('/CreationOptionList/Option', create_options::xml)) As opt
FROM st_gdaldrivers()
WHERE short_name = 'JPEG') As g;
```

oname	otype	descrip
PROGRESSIVE	boolean	whether to generate a progressive JPEG
QUALITY	int	good=100, bad=0, default=75
WORLDFILE	boolean	whether to generate a worldfile
INTERNAL_MASK	boolean	whether to generate a validity mask
COMMENT	string	Comment
SOURCE_ICC_PROFILE	string	ICC profile encoded in Base64
EXIF_THUMBNAIL	boolean	whether to generate an EXIF thumbnail(overview). By default its max dimension will be 128
THUMBNAIL_WIDTH	int	Forced thumbnail width
THUMBNAIL_HEIGHT	int	Forced thumbnail height

(9 rows)

```
-- raw xml output for creator options for GeoTiff --
SELECT create_options
FROM st_gdaldrivers()
WHERE short_name = 'GTiff';

<CreationOptionList>
  <Option name="COMPRESS" type="string-select">
    <Value
>NONE</Value>
    <Value
>LZW</Value>
```

```

    <Value
>PACKBITS</Value>
    <Value
>JPEG</Value>
    <Value
>CCITTRLE</Value>
    <Value
>CCITTFAX3</Value>
    <Value
>CCITTFAX4</Value>
    <Value
>DEFLATE</Value>
  </Option>
  <Option name="PREDICTOR" type="int" description="Predictor Type"/>
  <Option name="JPEG_QUALITY" type="int" description="JPEG quality 1-100" default="75"/>
  <Option name="ZLEVEL" type="int" description="DEFLATE compression level 1-9" default ←
    ="6"/>
  <Option name="NBITS" type="int" description="BITS for sub-byte files (1-7), sub-uint16 ←
    (9-15), sub-uint32 (17-31)"/>
  <Option name="INTERLEAVE" type="string-select" default="PIXEL">
    <Value
>BAND</Value>
    <Value
>PIXEL</Value>
  </Option>
  <Option name="TILED" type="boolean" description="Switch to tiled format"/>
  <Option name="TFW" type="boolean" description="Write out world file"/>
  <Option name="RPB" type="boolean" description="Write out .RPB (RPC) file"/>
  <Option name="BLOCKXSIZE" type="int" description="Tile Width"/>
  <Option name="BLOCKYSIZE" type="int" description="Tile/Strip Height"/>
  <Option name="PHOTOMETRIC" type="string-select">
    <Value
>MINISBLACK</Value>
    <Value
>MINISWHITE</Value>
    <Value
>PALETTE</Value>
    <Value
>RGB</Value>
    <Value
>CMYK</Value>
    <Value
>YCBCR</Value>
    <Value
>CIELAB</Value>
    <Value
>ICCLAB</Value>
    <Value
>ITULAB</Value>
  </Option>
  <Option name="SPARSE_OK" type="boolean" description="Can newly created files have ←
    missing blocks?" default="FALSE"/>
  <Option name="ALPHA" type="boolean" description="Mark first extrasample as being alpha ←
    "/>
  <Option name="PROFILE" type="string-select" default="GDALGeoTIFF">
    <Value
>GDALGeoTIFF</Value>
    <Value
>GeoTIFF</Value>
    <Value
>BASELINE</Value>
  </Option>

```

```

    <Option name="PIXELTYPE" type="string-select">
      <Value
>DEFAULT</Value>
      <Value
>SIGNEDBYTE</Value>
    </Option>
    <Option name="BIGTIFF" type="string-select" description="Force creation of BigTIFF file <-
">
      <Value
>YES</Value>
      <Value
>NO</Value>
      <Value
>IF_NEEDED</Value>
      <Value
>IF_SAFER</Value>
    </Option>
    <Option name="ENDIANNESS" type="string-select" default="NATIVE" description="Force <-
endianness of created file. For DEBUG purpose mostly">
      <Value
>NATIVE</Value>
      <Value
>INVERTED</Value>
      <Value
>LITTLE</Value>
      <Value
>BIG</Value>
    </Option>
    <Option name="COPY_SRC_OVERVIEWS" type="boolean" default="NO" description="Force copy <-
of overviews of source dataset (CreateCopy())"/>
</CreationOptionList>

```

```

-- Output the create options XML column for GTiff as a table --
SELECT (xpath('@name', g.opt))[1]::text As oname,
       (xpath('@type', g.opt))[1]::text As otype,
       (xpath('@description', g.opt))[1]::text As descrip,
       array_to_string(xpath('Value/text()', g.opt),', ' ) As vals
FROM (SELECT unnest(xpath('/CreationOptionList/Option', create_options::xml)) As opt
FROM st_gdaldrivers()
WHERE short_name = 'GTiff') As g;

```

oname	otype	descrip	vals
COMPRESS	string-select		NONE, LZW, <-
PACKBITS, JPEG, CCITTRLE, CCITTFAX3, CCITTFAX4, DEFLATE			
PREDICTOR	int	Predictor Type <-	
JPEG_QUALITY	int	JPEG quality 1-100 <-	
ZLEVEL	int	DEFLATE compression level 1-9 <-	
NBITS	int	BITS for sub-byte files (1-7), sub-uint16 (9-15), sub-uint32 (17-31) <-	
INTERLEAVE	string-select		BAND, PIXEL
TILED	boolean	Switch to tiled format <-	
TFW	boolean	Write out world file <-	

RPB	boolean	Write out .RPB (RPC) file ↔
BLOCKXSIZE	int	Tile Width ↔
BLOCKYSIZE	int	Tile/Strip Height ↔
PHOTOMETRIC	string-select	↔
		MINISBLACK, ↔
		MINISWHITE, PALETTE, RGB, CMYK, YCBCR, CIELAB, ICCLAB, ITULAB
SPARSE_OK	boolean	Can newly created files have missing blocks? ↔
ALPHA	boolean	Mark first extrasample as being alpha ↔
PROFILE	string-select	↔
		GDALGeoTIFF, ↔
		GeoTIFF, BASELINE
PIXELTYPE	string-select	↔
		DEFAULT, ↔
		SIGNEDBYTE
BIGTIFF	string-select	Force creation of BigTIFF file ↔
		YES, NO, IF_NEEDED, IF_SAFER
ENDIANNESS	string-select	Force endianness of created file. For DEBUG purpose ↔
		NATIVE, INVERTED, LITTLE, BIG
COPY_SRC_OVERVIEWS	boolean	Force copy of overviews of source dataset (CreateCopy ↔
		())
(19 rows)		

相关信息

[ST\\_AsGDALRaster](#), [ST\\_SRID](#), [postgis.gdal\\_enabled\\_drivers](#)

### 11.2.9 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);
```

#### 描述

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。

[入参数](#)：

**rast** 生成等高 [的栅格](#)

**bandnumber** 用于生成等高 [的波段](#)

**level\_interval** 生成的等高 [之](#) [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 将间隔 5。

**fixed\_levels** 生成的等高线之线的高程间隔

**polygonize** 如果 true，将构建等高线多边形，而不是多线形。

返回是一具有以下属性的：

**geom** 廓线的几何形状。

**id** GDAL 予廓线的唯一标识符。

**value** 线代表的格。于高程 DEM 入，将是出等高线的高程。

可用性：3.2.0

示例

```
WITH c AS (
SELECT (ST_Contour(rast, 1, fixed_levels => ARRAY[100.0, 200.0, 300.0])).*
FROM dem_grid WHERE rid = 1
)
SELECT st_astext(geom), id, value
FROM c;
```

相关信息

[ST\\_InterpolateRaster](#)

## 11.2.10 ST\_InterpolateRaster

ST\_InterpolateRaster — 基于入的 3 点集插网格表面，使用 X 和 Y 在网络上定位点，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);

### 描述

基于入的 3 点集插网格表面，使用 X 和 Y 在网络上定位点，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 网络教程](#)。

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** 用于出格几何形状的格模板。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格，但可以使用此参数行整。

可用性：3.2.0



示例

```
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](#)

### 11.2.11 UpdateRasterSRID

UpdateRasterSRID — 更改用 `table_name` 指定的列和表中所有栅格的 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);
```

描述

更改用 `table_name` 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）。栅格的元数据将更改。

可用性：2.1.0

相关信息

[UpdateGeometrySRID](#)

### 11.2.12 ST\_CreateOverview

ST\_CreateOverview — 创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');
```

## 描述

使用源表中重新采样的栅格建立一个概览表。输出栅格将具有与输入栅格相同的大小，并以较低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，并将限制行网格束。

算法选项有：“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

## 示例

输出量通常很好，但品格式较慢

```
SELECT ST_CreateOverview('mydata.mytable'::regclass, 'rast', 2, 'Lanczos');
```

输出到更快地处理默认最近

```
SELECT ST_CreateOverview('mydata.mytable'::regclass, 'rast', 2);
```

## 相关信息

[ST\\_Retile](#), [AddOverviewConstraints](#), [AddRasterConstraints](#), [Section 10.2.2](#)

## 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，将添加到末尾。

#### Synopsis

- (1) raster **ST\_AddBand**(raster rast, addbandarg[] addbandargset);
- (2) raster **ST\_AddBand**(raster rast, integer index, text pixeltype, double precision initialvalue=0, double precision nodataval=NULL);
- (3) raster **ST\_AddBand**(raster rast, text pixeltype, double precision initialvalue=0, double precision nodataval=NULL);
- (4) raster **ST\_AddBand**(raster torast, raster fromrast, integer fromband=1, integer torastindex=at\_end);
- (5) raster **ST\_AddBand**(raster torast, raster[] fromrasts, integer fromband=1, integer torastindex=at\_end);
- (6) raster **ST\_AddBand**(raster rast, integer index, text outdbfile, integer[] outdbindex, double precision nodataval=NULL);
- (7) raster **ST\_AddBand**(raster rast, text outdbfile, integer[] outdbindex, integer index=at\_end, double precision nodataval=NULL);

## 描述

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，将新波段添加到末尾。如果未指定 `fromband`，假定从 `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。参见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` 为 `NULL`，数组中每个栅格的 `fromband` 波段将累加到新栅格中。

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。

新增：2.1.0 添加了对 `addbandarg` 的支持。

新增：2.1.0 添加了对新的 `out-db` 波段的支持。

示例：添加一个新波段

```
-- Add another band of type 8 bit unsigned integer with pixels initialized to 200
UPDATE dummy_rast
  SET rast = ST_AddBand(rast,'8BUI'::text,200)
WHERE rid = 1;
```

```
-- Create an empty raster 100x100 units, with upper left right at 0, add 2 bands (band 1 is 0/1 boolean bit switch, band2 allows values 0-15)
-- uses addbandargs
INSERT INTO dummy_rast(rid,rast)
  VALUES(10, ST_AddBand(ST_MakeEmptyRaster(100, 100, 0, 0, 1, -1, 0, 0, 0),
    ARRAY[
      ROW(1, '1BB'::text, 0, NULL),
      ROW(2, '4BUI'::text, 0, NULL)
    ]::addbandarg[]
  )
);
```

```
-- output meta data of raster bands to verify all is right --
SELECT (bmd).*
FROM (SELECT ST_BandMetaData(rast,generate_series(1,2)) As bmd
      FROM dummy_rast WHERE rid = 10) AS foo;
--result --
pixeltype | nodatavalue | isoutdb | path
-----+-----+-----+-----
1BB      |              | f       |
4BUI     |              | f       |
```

```
-- output meta data of raster -
SELECT (rmd).width, (rmd).height, (rmd).numbands
FROM (SELECT ST_MetaData(rast) As rmd
      FROM dummy_rast WHERE rid = 10) AS foo;
-- result --
upperleftx | upperlefty | width | height | scalex | scaley | skewx | skewy | srid | numbands
-----+-----+-----+-----+-----+-----+-----+-----+-----+-----
          0 |           0 |    100 |    100 |        1 |       -1 |        0 |        0 |      0 |          2
```

示例：多个新波段

```
SELECT
*
FROM ST_BandMetadata(
  ST_AddBand(
    ST_MakeEmptyRaster(10, 10, 0, 0, 1, -1, 0, 0, 0),
    ARRAY[
      ROW(NULL, '8BUI', 255, 0),
      ROW(NULL, '16BUI', 1, 2),
      ROW(2, '32BUI', 100, 12),
      ROW(2, '32BF', 3.14, -1)
    ]::addbandarg[]
  ),
  ARRAY[]::integer[]
);
```

bandnum	pixeltype	nodatavalue	isoutdb	path
1	8BUI	0	f	
2	32BF	-1	f	
3	32BUI	12	f	
4	16BUI	2	f	

```
-- Aggregate the 1st band of a table of like rasters into a single raster
-- with as many bands as there are test_types and as many rows (new rasters) as there are ←
-- mice
-- NOTE: The ORDER BY test_type is only supported in PostgreSQL 9.0+
-- for 8.4 and below it usually works to order your data in a subselect (but not guaranteed ←
-- )
-- The resulting raster will have a band for each test_type alphabetical by test_type
-- For mouse lovers: No mice were harmed in this exercise
SELECT
  mouse,
  ST_AddBand(NULL, array_agg(rast ORDER BY test_type), 1) As rast
FROM mice_studies
GROUP BY mouse;
```

示例：新的 **Out-db** 波段

```
SELECT
*
FROM ST_BandMetadata(
  ST_AddBand(
    ST_MakeEmptyRaster(10, 10, 0, 0, 1, -1, 0, 0, 0),
    '/home/raster/mytestraster.tif'::text, NULL::int[]
  ),
  ARRAY[]::integer[]
);
```

bandnum	pixeltype	nodatavalue	isoutdb	path
1	8BUI		t	/home/raster/mytestraster.tif
2	8BUI		t	/home/raster/mytestraster.tif
3	8BUI		t	/home/raster/mytestraster.tif

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 几何对象转换为 PostGIS 栅格。

### Synopsis

```
raster ST_AsRaster(geometry geom, raster ref, text pixelttype, double precision value=1, double precision nodataval=0, boolean touched=false);
raster ST_AsRaster(geometry geom, raster ref, text[] pixelttype=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 pixelttype, 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[] pixelttype=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 pixelttype, 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[] pixelttype, 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 pixelttype, double precision value=1, double precision nodataval=0, double precision skewx=0, double precision skewy=0, boolean touched=false);
raster ST_AsRaster(geometry geom, integer width, integer height, double precision gridx=NULL, double precision gridy=NULL, text[] pixelttype=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 pixelttype, double precision value=1, double precision nodataval=0, double precision upperleftx=NULL, double precision upperlefty=NULL, double precision skewx=0, double precision skewy=0, boolean touched=false);
raster ST_AsRaster(geometry geom, integer width, integer height, text[] pixelttype, 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);
```

### 描述

将一个 PostGIS 几何对象转换为 PostGIS 栅格。某些不同的变体提供了三种可能性，用于设置生成的栅格的方式和像素大小。

第一变体由前三个变体组成，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对象的表与包含参考栅格的表连接来使用此参考栅格。

第二变体，由四个变体组成，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圆整。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 返回正确的变体。

第三，由四个参数组成，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。

四个参数中的前两个参数允许您使用栅格的任意角（`gridx` & `gridy`）指定方式，而最后两个参数使用左上角（`upperleftx` & `upperlefty`）指定方式。

每个参数都允许生成波段栅格或多波段栅格。要生成多波段栅格，必须提供像素数据类型（`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参考。唯一的例外是具有参考栅格的几何体。在这种情况下，生成的栅格将得到与参考栅格相同的 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，用于直接从数据渲染几何形状的 `jpeg` 和 `png` 特别有用。

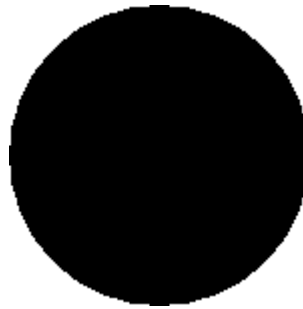
可用性：2.0.0 - 需要 GDAL >= 1.6.0。



#### Note

目前无法渲染复杂的几何类型，例如曲线、TINS 和多面体曲面，但一旦 GDAL 可以了，也可以了。

示例：将几何形状输出 PNG 文件



黑色圆

```
-- this will output a black circle taking up 150 x 150 pixels --
SELECT ST_AsPNG(ST_AsRaster(ST_Buffer(ST_Point(1,5),10),150, 150));
```



使用 PostGIS 渲染的缓冲区的示例

```
-- the bands map to RGB bands - the value (118,154,118) - teal --
SELECT ST_AsPNG(
  ST_AsRaster(
    ST_Buffer(
      ST_GeomFromText('LINESTRING(50 50,150 150,150 50)'), 10,'join=bevel'),
      200,200,ARRAY['8BUI', '8BUI', '8BUI'], ARRAY[118,154,118], ARRAY[0,0,0]));
```

相关信息

[ST\\_BandPixelType](#), [ST\\_Buffer](#), [ST\\_GDALDrivers](#), [ST\\_AsGDALRaster](#), [ST\\_AsPNG](#), [ST\\_AsJPEG](#), [ST\\_SRID](#)

### 11.3.3 ST\_Band

`ST_Band` — 返回具有栅格的一个或多个波段作新栅格。对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=,);
```

#### 描述

返回具有栅格的一个或多个波段作新栅格。对于从具有栅格建新栅格或取出栅格的固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，例如删除波段。



#### Warning

对于函数的 `nbands` 作文本字体，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，我们强烈建议您使用函数的数组形式，例如 `ST_Band(rast, '{1,2,3}'::int[]);`，因为 `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

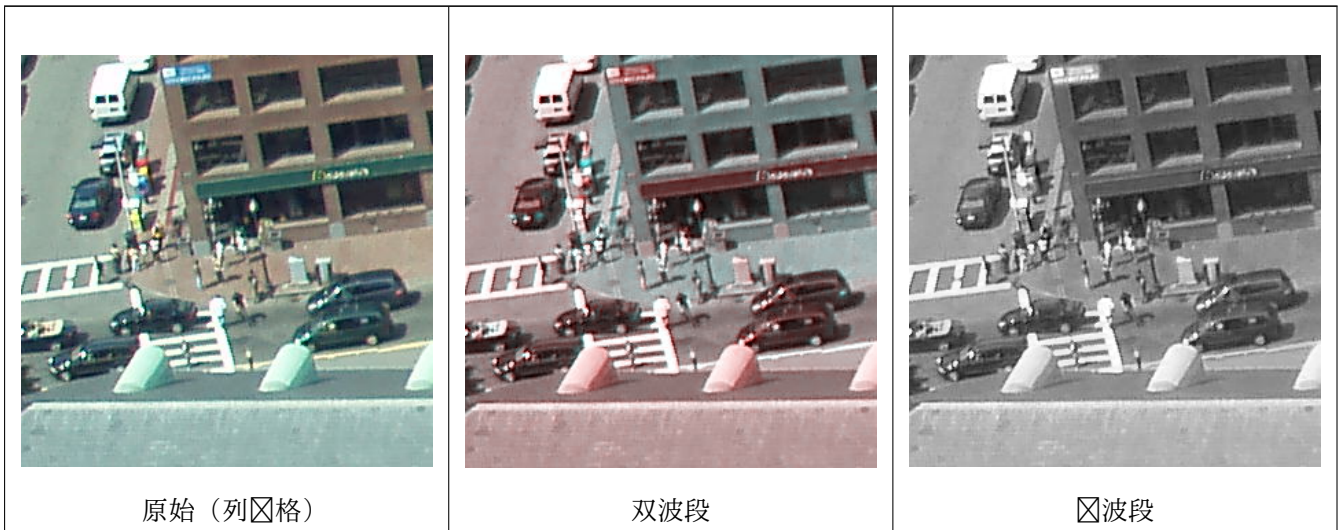
```
-- Make 2 new rasters: 1 containing band 1 of dummy, second containing band 2 of dummy and ←
  then reclassified as a 2BUI
SELECT ST_NumBands(rast1) As numb1, ST_BandPixelType(rast1) As pix1,
  ST_NumBands(rast2) As numb2, ST_BandPixelType(rast2) As pix2
FROM (
  SELECT ST_Band(rast) As rast1, ST_Reclass(ST_Band(rast,3), '100-200):1, [200-254:2', '2 ←
    BUI') As rast2
  FROM dummy_rast
  WHERE rid = 2) As foo;
```

```
numb1 | pix1 | numb2 | pix2
-----+-----+-----+-----
      1 | 8BUI |      1 | 2BUI
```

```
-- Return bands 2 and 3. Using array cast syntax
SELECT ST_NumBands(ST_Band(rast, '{2,3}'::int[])) As num_bands
FROM dummy_rast WHERE rid=2;
```

```
num_bands
-----
2
```

```
-- Return bands 2 and 3. Use array to define bands
SELECT ST_NumBands(ST_Band(rast, ARRAY[2,3])) As num_bands
FROM dummy_rast
WHERE rid=2;
```



```
--Make a new raster with 2nd band of original and 1st band repeated twice,
and another with just the third band
SELECT rast, ST_Band(rast, ARRAY[2,1,1]) As dupe_band,
ST_Band(rast, 3) As sing_band
FROM samples.than_chunked
WHERE rid=35;
```

相关信息

[ST\\_AddBand](#), [ST\\_NumBands](#), [ST\\_Reclass](#), Chapter 11

### 11.3.4 ST\_MakeEmptyCoverage

ST\_MakeEmptyCoverage — 用空⊠格⊠⊠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);



描述

使用 **ST\_MakeEmptyRaster** 构建一格瓦片。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，于格，scaley 通常负，scalex 通常正。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 个格，以覆盖从左上角 (22, 77) 到右下角 (55, 33) 的 WGS84 区域。

```
SELECT (ST_MetaData(tile)).* FROM ST_MakeEmptyCoverage(1, 1, 4, 4, 22, 33, (55 - 22)/(4)::float, (33 - 77)/(4)::float, 0., 0., 4326) tile;
```

upperleftx	upperlefty	width	height	scalex	scaley	skewx	skewy	srid	numbands
22	77	1	1	8.25	-11	0	0	4326	1
30.25	77	1	1	8.25	-11	0	0	4326	1
38.5	77	1	1	8.25	-11	0	0	4326	1
46.75	77	1	1	8.25	-11	0	0	4326	1
22	73	1	1	8.25	-11	0	0	4326	1
30.25	73	1	1	8.25	-11	0	0	4326	1
38.5	73	1	1	8.25	-11	0	0	4326	1
46.75	73	1	1	8.25	-11	0	0	4326	1
22	69	1	1	8.25	-11	0	0	4326	1
30.25	69	1	1	8.25	-11	0	0	4326	1
38.5	69	1	1	8.25	-11	0	0	4326	1
46.75	69	1	1	8.25	-11	0	0	4326	1
22	65	1	1	8.25	-11	0	0	4326	1
30.25	65	1	1	8.25	-11	0	0	4326	1
38.5	65	1	1	8.25	-11	0	0	4326	1
46.75	65	1	1	8.25	-11	0	0	4326	1

相关信息

[ST\\_MakeEmptyRaster](#)

### 11.3.5 ST\_MakeEmptyRaster

`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、像素大小和旋转变换参数（`scalex`、`scaley`、`skewx` 和 `skewy`），以及参考系 ID（SRID）。如果指定了一个栅格，则返回一个具有相同大小、变换方式和 SRID 的新栅格。如果省略了 SRID，则空参考系将置为未知（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);
```

#### 描述

返回一个空白栅格（没有波段），其具有指定的尺寸（宽度和高度）并以空（或世界）坐标系进行地理参考，具有左上角 X 坐标（`upperleftx`）、左上角 Y 坐标（`upperlefty`）、像素大小和旋转变换参数（`scalex`、`scaley`、`skewx` 和 `skewy`），以及参考系 ID（SRID）。

最后一个版本使用一个参数来指定像素大小（`pixelsize`）。`scalex` 置为此参数，`scaley` 置为此参数的平方。`skewx` 和 `skewy` 置为 0。

如果输入有栅格，它将返回具有相同元数据（不含波段）的新栅格。

如果未指定 `srid`，则默认为 0。创建空栅格后，您可能想要向其中添加波段并可能对其行。参考 [ST\\_AddBand](#) 定义波段，参考 [ST\\_SetValue](#) 置初始像素。

#### 示例

```
INSERT INTO dummy_rast(rid,rast)
VALUES(3, ST_MakeEmptyRaster( 100, 100, 0.0005, 0.0005, 1, 1, 0, 0, 4326) );

--use an existing raster as template for new raster
INSERT INTO dummy_rast(rid,rast)
SELECT 4, ST_MakeEmptyRaster(rast)
FROM dummy_rast WHERE rid = 3;

-- output meta data of rasters we just added
SELECT rid, (md).*
FROM (SELECT rid, ST_MetaData(rast) As md
      FROM dummy_rast
      WHERE rid IN(3,4)) As foo;

-- output --
rid | upperleftx | upperlefty | width | height | scalex | scaley | skewx | skewy | srid | ←
-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
3 | 0.0005 | 0.0005 | 100 | 100 | 1 | 1 | 0 | 0 | 4326 | ←
4 | 0.0005 | 0.0005 | 100 | 100 | 1 | 1 | 0 | 0 | 4326 | ←
```

相关信息

[ST\\_AddBand](#), [ST\\_MetaData](#), [ST\\_ScaleX](#), [ST\\_ScaleY](#), [ST\\_SetValue](#), [ST\\_SkewX](#), [ST\\_SkewY](#)

### 11.3.6 ST\_Tile

ST\_Tile — 返回根据输出栅格的所需宽度分割输入栅格而生的一批栅格。

#### Synopsis

```
setof raster ST_Tile(raster rast, int[] nband, integer width, integer height, boolean padwithnodata=FALSE,
double precision nodataval=NULL);
```

```
setof raster ST_Tile(raster rast, integer nband, integer width, integer height, boolean 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);
```

#### 描述

返回根据输出栅格的所需宽度分割输入栅格而生的一批栅格。

如果 `padwithnodata = FALSE`, 栅格右部和底部的切片可能具有与其余切片不同的宽度。如果 `padwithnodata = TRUE`, 所有切片将具有相同的宽度, 并且切片可能会用 NODATA 填充。如果栅格波段未指定 NODATA, 可以通过设置 `nodataval` 来指定 NODATA。



#### Note

如果输入栅格的指定波段位于数据外, 输出栅格中的相应波段也将位于数据外。

可用性 : 2.1.0

#### 示例

```
WITH foo AS (
  SELECT ST_AddBand(ST_AddBand(ST_MakeEmptyRaster(3, 3, 0, 0, 1, -1, 0, 0, 0), 1, '8BUI', ←
    1, 0), 2, '8BUI', 10, 0) AS rast UNION ALL
  SELECT ST_AddBand(ST_AddBand(ST_MakeEmptyRaster(3, 3, 3, 0, 1, -1, 0, 0, 0), 1, '8BUI', ←
    2, 0), 2, '8BUI', 20, 0) AS rast UNION ALL
  SELECT ST_AddBand(ST_AddBand(ST_MakeEmptyRaster(3, 3, 6, 0, 1, -1, 0, 0, 0), 1, '8BUI', ←
    3, 0), 2, '8BUI', 30, 0) AS rast UNION ALL

  SELECT ST_AddBand(ST_AddBand(ST_MakeEmptyRaster(3, 3, 0, -3, 1, -1, 0, 0, 0), 1, '8BUI ←
    ', 4, 0), 2, '8BUI', 40, 0) AS rast UNION ALL
  SELECT ST_AddBand(ST_AddBand(ST_MakeEmptyRaster(3, 3, 3, -3, 1, -1, 0, 0, 0), 1, '8BUI ←
    ', 5, 0), 2, '8BUI', 50, 0) AS rast UNION ALL
  SELECT ST_AddBand(ST_AddBand(ST_MakeEmptyRaster(3, 3, 6, -3, 1, -1, 0, 0, 0), 1, '8BUI ←
    ', 6, 0), 2, '8BUI', 60, 0) AS rast UNION ALL

  SELECT ST_AddBand(ST_AddBand(ST_MakeEmptyRaster(3, 3, 0, -6, 1, -1, 0, 0, 0), 1, '8BUI ←
    ', 7, 0), 2, '8BUI', 70, 0) AS rast UNION ALL
  SELECT ST_AddBand(ST_AddBand(ST_MakeEmptyRaster(3, 3, 3, -6, 1, -1, 0, 0, 0), 1, '8BUI ←
    ', 8, 0), 2, '8BUI', 80, 0) AS rast UNION ALL
```

```

SELECT ST_AddBand(ST_AddBand(ST_MakeEmptyRaster(3, 3, 6, -6, 1, -1, 0, 0, 0), 1, '8BUI' ←
', 9, 0), 2, '8BUI', 90, 0) AS rast
), bar AS (
SELECT ST_Union(rast) AS rast FROM foo
), baz AS (
SELECT ST_Tile(rast, 3, 3, TRUE) AS rast FROM bar
)
SELECT
ST_DumpValues(rast)
FROM baz;

```

st\_dumpvalues

```

-----
(1,"{{1,1,1},{1,1,1},{1,1,1}}")
(2,"{{10,10,10},{10,10,10},{10,10,10}}")
(1,"{{2,2,2},{2,2,2},{2,2,2}}")
(2,"{{20,20,20},{20,20,20},{20,20,20}}")
(1,"{{3,3,3},{3,3,3},{3,3,3}}")
(2,"{{30,30,30},{30,30,30},{30,30,30}}")
(1,"{{4,4,4},{4,4,4},{4,4,4}}")
(2,"{{40,40,40},{40,40,40},{40,40,40}}")
(1,"{{5,5,5},{5,5,5},{5,5,5}}")
(2,"{{50,50,50},{50,50,50},{50,50,50}}")
(1,"{{6,6,6},{6,6,6},{6,6,6}}")
(2,"{{60,60,60},{60,60,60},{60,60,60}}")
(1,"{{7,7,7},{7,7,7},{7,7,7}}")
(2,"{{70,70,70},{70,70,70},{70,70,70}}")
(1,"{{8,8,8},{8,8,8},{8,8,8}}")
(2,"{{80,80,80},{80,80,80},{80,80,80}}")
(1,"{{9,9,9},{9,9,9},{9,9,9}}")
(2,"{{90,90,90},{90,90,90},{90,90,90}}")
(18 rows)

```

```

WITH foo AS (
SELECT ST_AddBand(ST_AddBand(ST_MakeEmptyRaster(3, 3, 0, 0, 1, -1, 0, 0, 0), 1, '8BUI', ←
1, 0), 2, '8BUI', 10, 0) AS rast UNION ALL
SELECT ST_AddBand(ST_AddBand(ST_MakeEmptyRaster(3, 3, 3, 0, 1, -1, 0, 0, 0), 1, '8BUI', ←
2, 0), 2, '8BUI', 20, 0) AS rast UNION ALL
SELECT ST_AddBand(ST_AddBand(ST_MakeEmptyRaster(3, 3, 6, 0, 1, -1, 0, 0, 0), 1, '8BUI', ←
3, 0), 2, '8BUI', 30, 0) AS rast UNION ALL

SELECT ST_AddBand(ST_AddBand(ST_MakeEmptyRaster(3, 3, 0, -3, 1, -1, 0, 0, 0), 1, '8BUI' ←
', 4, 0), 2, '8BUI', 40, 0) AS rast UNION ALL
SELECT ST_AddBand(ST_AddBand(ST_MakeEmptyRaster(3, 3, 3, -3, 1, -1, 0, 0, 0), 1, '8BUI' ←
', 5, 0), 2, '8BUI', 50, 0) AS rast UNION ALL
SELECT ST_AddBand(ST_AddBand(ST_MakeEmptyRaster(3, 3, 6, -3, 1, -1, 0, 0, 0), 1, '8BUI' ←
', 6, 0), 2, '8BUI', 60, 0) AS rast UNION ALL

SELECT ST_AddBand(ST_AddBand(ST_MakeEmptyRaster(3, 3, 0, -6, 1, -1, 0, 0, 0), 1, '8BUI' ←
', 7, 0), 2, '8BUI', 70, 0) AS rast UNION ALL
SELECT ST_AddBand(ST_AddBand(ST_MakeEmptyRaster(3, 3, 3, -6, 1, -1, 0, 0, 0), 1, '8BUI' ←
', 8, 0), 2, '8BUI', 80, 0) AS rast UNION ALL
SELECT ST_AddBand(ST_AddBand(ST_MakeEmptyRaster(3, 3, 6, -6, 1, -1, 0, 0, 0), 1, '8BUI' ←
', 9, 0), 2, '8BUI', 90, 0) AS rast
), bar AS (
SELECT ST_Union(rast) AS rast FROM foo
), baz AS (
SELECT ST_Tile(rast, 3, 3, 2) AS rast FROM bar
)
SELECT
ST_DumpValues(rast)
FROM baz;

```

```

          st_dumpvalues
-----
(1,"{{10,10,10},{10,10,10},{10,10,10}}")
(1,"{{20,20,20},{20,20,20},{20,20,20}}")
(1,"{{30,30,30},{30,30,30},{30,30,30}}")
(1,"{{40,40,40},{40,40,40},{40,40,40}}")
(1,"{{50,50,50},{50,50,50},{50,50,50}}")
(1,"{{60,60,60},{60,60,60},{60,60,60}}")
(1,"{{70,70,70},{70,70,70},{70,70,70}}")
(1,"{{80,80,80},{80,80,80},{80,80,80}}")
(1,"{{90,90,90},{90,90,90},{90,90,90}}")
(9 rows)

```

相关信息

[ST\\_Union](#), [ST\\_Retile](#)

### 11.3.7 ST\_Retile

`ST_Retile` — 从任意平面的栅格覆盖范围返回一配置的水平面。

#### Synopsis

SETOF raster **ST\_Retile**(regclass tab, name col, geometry ext, float8 sfx, float8 sfy, int tw, int th, text algo='NearestNeighbor');

#### 描述

返回一具有指定比例 (`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](#)

### 11.3.8 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);

## 描述

从受支持的 GDAL 栅格文件返回栅格。gdaldata 是 bytea 型，内容是 GDAL 栅格文件的内容。

如果 srid 为 NULL，函数将栅格自 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

```
WITH foo AS (
  SELECT ST_AsPNG(ST_AddBand(ST_AddBand(ST_AddBand(ST_MakeEmptyRaster(2, 2, 0, 0, 0.1, ←
    -0.1, 0, 0, 4326), 1, '8BUI', 1, 0), 2, '8BUI', 2, 0), 3, '8BUI', 3, 0)) AS png
),
bar AS (
  SELECT 1 AS rid, ST_FromGDALRaster(png) AS rast FROM foo
  UNION ALL
  SELECT 2 AS rid, ST_FromGDALRaster(png, 3310) AS rast FROM foo
)
SELECT
  rid,
  ST_Metadata(rast) AS metadata,
  ST_SummaryStats(rast, 1) AS stats1,
  ST_SummaryStats(rast, 2) AS stats2,
  ST_SummaryStats(rast, 3) AS stats3
FROM bar
ORDER BY rid;
```

rid	metadata	stats1	stats2	stats3
1	(0,0,2,2,1,-1,0,0,0,3)	(4,4,1,0,1,1)	(4,8,2,0,2,2)	(4,12,3,0,3,3)
2	(0,0,2,2,1,-1,0,0,3310,3)	(4,4,1,0,1,1)	(4,8,2,0,2,2)	(4,12,3,0,3,3)

(2 rows)

## 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);
```

## 描述

返回地理配准元数据，包括 GDAL 或 ESRI 格式的回字符，如世界文件中常出现的那。如果未指定类型，默认为 GDAL。类型是字符串“GDAL”或“ESRI”。

格式表示的区如下：

### GDAL:

```
scalex
skewy
skewx
scaley
upperleftx
upperlefty
```

### ESRI:

```
scalex
skewy
skewx
scaley
upperleftx + scalex*0.5
upperlefty + scaley*0.5
```

## 示例

```
SELECT ST_GeoReference(rast, 'ESRI') As esri_ref, ST_GeoReference(rast, 'GDAL') As gdal_ref
FROM dummy_rast WHERE rid=1;
```

esri_ref	gdal_ref
2.0000000000	2.0000000000
0.0000000000	0.0000000000
0.0000000000	0.0000000000
3.0000000000	3.0000000000
1.5000000000	0.5000000000
2.0000000000	0.5000000000

## 相关信息

[ST\\_SetGeoReference](#), [ST\\_ScaleX](#), [ST\\_ScaleY](#)

## 11.4.2 ST\_Height

**ST\_Height** — 返回格的高度（以像素位）。

### Synopsis

```
integer ST_Height(raster rast);
```

### 描述

返回格的高度（以像素位）。

示例

```
SELECT rid, ST_Height(rast) As rastheight
FROM dummy_rast;
```

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);
```

描述

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

```
SELECT ST_IsEmpty(ST_MakeEmptyRaster(100, 100, 0, 0, 0, 0, 0, 0))
st_isempty |
-----+
f          |
```

```
SELECT ST_IsEmpty(ST_MakeEmptyRaster(0, 0, 0, 0, 0, 0, 0, 0))
st_isempty |
-----+
t          |
```

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);
```

### 描述

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` 可能返回低的值，因为它返回后的值。`pg_total_relation_size` - 包括表、表和索引。

可用性：2.2.0

### 示例

```
SELECT ST_MemSize(ST_AsRaster(ST_Buffer(ST_Point(1,5),10,1000),150, 150, '8BUI')) As rast_mem;
rast_mem
-----
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);
```

### 描述

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

```
SELECT rid, (foo.md).*
FROM (SELECT rid, ST_MetaData(rast) As md
FROM dummy_rast) As foo;
```

rid	upperleftx	upperlefty	width	height	scalex	scaley	skewx	skewy	srid	←
1	0.5	0.5	10	20	2	3	0	0	0	←
2	3427927.75	5793244	5	5	0.05	-0.05	0	0	0	←

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);

描述

返回栅格象中的波段数。

示例

```
SELECT rid, ST_NumBands(rast) As numbands
FROM dummy_rast;
```

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);

### 描述

以空参考系的几何位返回像素的高度。在没有斜的情况下，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：无斜的格

```
SELECT ST_Height(rast) As rastheight, ST_PixelHeight(rast) As pixheight,
       ST_ScaleX(rast) As scalex, ST_ScaleY(rast) As scaley, ST_SkewX(rast) As skewx,
       ST_SkewY(rast) As skewy
FROM dummy_rast;
```

rastheight	pixheight	scalex	scaley	skewx	skewy
20	3	2	3	0	0
5	0.05	0.05	-0.05	0	0

示例：斜度不为 0 的格

```
SELECT ST_Height(rast) As rastheight, ST_PixelHeight(rast) As pixheight,
       ST_ScaleX(rast) As scalex, ST_ScaleY(rast) As scaley, ST_SkewX(rast) As skewx,
       ST_SkewY(rast) As skewy
FROM (SELECT ST_SetSKew(rast,0.5,0.5) As rast
      FROM dummy_rast) As skewed;
```

rastheight	pixheight	scalex	scaley	skewx	skewy
20	3.04138126514911	2	3	0.5	0.5
5	0.502493781056044	0.05	-0.05	0.5	0.5

### 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的像素度。

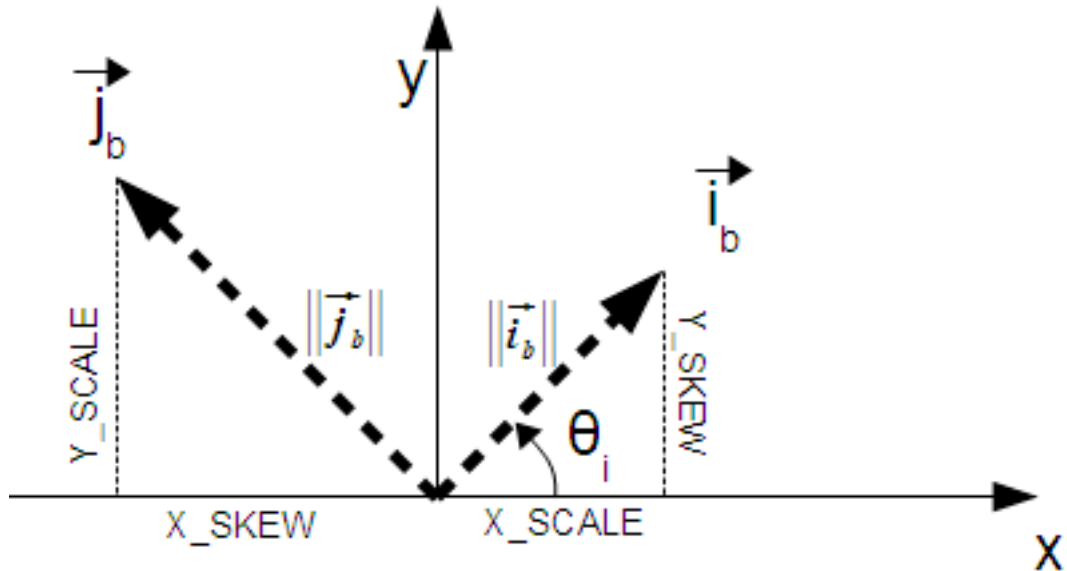
### Synopsis

double precision **ST\_PixelWidth**(raster rast);

描述

返回空参考系中的几何位中像素的度。在没有斜的情况下，像素度只是几何坐标与光像素之比。

下图展示了这种关系：



像素度： $i$  方向的像素大小  
 像素高度： $j$  方向的像素大小

示例：无斜的网格

```
SELECT ST_Width(rast) As rastwidth, ST_PixelWidth(rast) As pixwidth,
       ST_ScaleX(rast) As scalex, ST_ScaleY(rast) As scaley, ST_SkewX(rast) As skewx,
       ST_SkewY(rast) As skewy
FROM dummy_rast;
```

rastwidth	pixwidth	scalex	scaley	skewx	skewy
10	2	2	3	0	0
5	0.05	0.05	-0.05	0	0

示例：斜度不为 0 的网格

```
SELECT ST_Width(rast) As rastwidth, ST_PixelWidth(rast) As pixwidth,
       ST_ScaleX(rast) As scalex, ST_ScaleY(rast) As scaley, ST_SkewX(rast) As skewx,
       ST_SkewY(rast) As skewy
FROM (SELECT ST_SetSkew(rast,0.5,0.5) As rast
FROM dummy_rast) As skewed;
```

rastwidth	pixwidth	scalex	scaley	skewx	skewy
10	2.06155281280883	2	3	0.5	0.5
5	0.502493781056044	0.05	-0.05	0.5	0.5

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（以坐参考系位）。

#### Synopsis

```
float8 ST_ScaleX(raster rast);
```

#### 描述

返回像素宽度的 X 分量（以坐参考系位）。有关更多信息，参 [世界文件](#)。

更改：2.0.0。在 WKTRaster 版本中，称 `ST_PixelSizeX`。

#### 示例

```
SELECT rid, ST_ScaleX(rast) As rastpixwidth
FROM dummy_rast;
```

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（以坐参考系位）。

#### Synopsis

```
float8 ST_ScaleY(raster rast);
```

#### 描述

返回像素高度的 Y 分量（以坐参考系位）。可能是。有关更多信息，参 [世界文件](#)。

更改：2.0.0。在 WKTRaster 版本中，称 `ST_PixelSizeY`。

示例

```
SELECT rid, ST_ScaleY(rast) As rastpixheight
FROM dummy_rast;
```

rid	rastpixheight
1	3
2	-0.05

相关信息

[ST\\_Height](#)

### 11.4.11 ST\_RasterToWorldCoord

**ST\_RasterToWorldCoord** — 在指定列和行的情况下，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。

#### Synopsis

```
record ST_RasterToWorldCoord(raster rast, integer xcolumn, integer yrow);
```

描述

在指定列和行的情况下，以几何 X 和 Y（经度和纬度）形式返回左上角。返回的 X 和 Y 采用地理参考栅格的几何位置。列和行的编号从 1 开始，但如果向任一参数指定零、负数或大于栅格相经纬度的数字，则假定栅格网格适用于栅格边界之外，它将返回栅格外部的坐标。

可用性：2.1.0

示例

```
-- non-skewed raster
SELECT
  rid,
  (ST_RasterToWorldCoord(rast,1, 1)).*,
  (ST_RasterToWorldCoord(rast,2, 2)).*
FROM dummy_rast
```

rid	longitude	latitude	longitude	latitude
1	0.5	0.5	2.5	3.5
2	3427927.75	5793244	3427927.8	5793243.95

```
-- skewed raster
SELECT
  rid,
  (ST_RasterToWorldCoord(rast, 1, 1)).*,
  (ST_RasterToWorldCoord(rast, 2, 3)).*
FROM (
  SELECT
    rid,
```

```

        ST_SetSkew(rast, 100.5, 0) As rast
    FROM dummy_rast
) As foo

```

rid	longitude	latitude	longitude	latitude
1	0.5	0.5	203.5	6.5
2	3427927.75	5793244	3428128.8	5793243.9

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);

```

#### 描述

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，但如果输入一个数字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

对于非斜栅格，提供 X 列就足够了。对于斜栅格，地理参考坐标是 `ST_ScaleX` 和 `ST_SkewX` 以及行和列的函数。如果您只对斜栅格提供 X 列，它会输出 NaN。

更改：2.1.0 在之前的版本中，名称 `ST_Raster2WorldCoordX`

#### 示例

```

-- non-skewed raster providing column is sufficient
SELECT rid, ST_RasterToWorldCoordX(rast,1) As xlcoord,
        ST_RasterToWorldCoordX(rast,2) As x2coord,
        ST_ScaleX(rast) As pixelx
FROM dummy_rast;

```

rid	xlcoord	x2coord	pixelx
1	0.5	2.5	2
2	3427927.75	3427927.8	0.05

```
-- for fun lets skew it
SELECT rid, ST_RasterToWorldCoordX(rast, 1, 1) As x1coord,
       ST_RasterToWorldCoordX(rast, 2, 3) As x2coord,
       ST_ScaleX(rast) As pixelx
FROM (SELECT rid, ST_SetSkew(rast, 100.5, 0) As rast FROM dummy_rast) As foo;
```

rid	x1coord	x2coord	pixelx
1	0.5	203.5	2
2	3427927.75	3428128.8	0.05

相关信息

[ST\\_ScaleX](#), [ST\\_RasterToWorldCoordY](#), [ST\\_SetSkew](#), [ST\\_SkewX](#)

### 11.4.13 ST\_RasterToWorldCoordY

ST\_RasterToWorldCoordY — 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。

#### Synopsis

```
float8 ST_RasterToWorldCoordY(raster rast, integer yrow);
float8 ST_RasterToWorldCoordY(raster rast, integer xcolumn, integer yrow);
```

#### 描述

以地理参考栅格的几何坐标返回栅格列行的左上角 Y 坐标。列和行的编号从 1 开始，但如果您输入行数或高于栅格中的列/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，并假设斜角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 X。

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
SELECT rid, ST_RasterToWorldCoordY(rast,1) As y1coord,
       ST_RasterToWorldCoordY(rast,3) As y2coord,
       ST_ScaleY(rast) As pixely
FROM dummy_rast;
```

rid	y1coord	y2coord	pixely
1	0.5	6.5	3
2	5793244	5793243.9	-0.05



```
-- for fun lets skew it
SELECT rid, ST_RasterToWorldCoordY(rast,1,1) As y1coord,
       ST_RasterToWorldCoordY(rast,2,3) As y2coord,
       ST_ScaleY(rast) As pixely
FROM (SELECT rid, ST_SetSkew(rast,0,100.5) As rast FROM dummy_rast) As foo;
```

rid	y1coord	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 — 返回栅格的旋弧度。

#### Synopsis

```
float8 ST_Rotation(raster rast);
```

描述

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

```
SELECT rid, ST_Rotation(ST_SetScale(ST_SetSkew(rast, sqrt(2)), sqrt(2))) as rot FROM
dummy_rast;
```

rid	rot
1	0.785398163397448
2	0.785398163397448

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);
```

## 描述

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

```
SELECT rid, ST_SkewX(rast) As skewx, ST_SkewY(rast) As skewy,
       ST_GeoReference(rast) as georef
FROM dummy_rast;
```

rid	skewx	skewy	georef
1	0	0	2.0000000000 : 0.0000000000 : 0.0000000000 : 3.0000000000 : 0.5000000000 : 0.5000000000 :
2	0	0	0.0500000000 : 0.0000000000 : 0.0000000000 : -0.0500000000 : 3427927.7500000000 : 5793244.0000000000

## 相关信息

[ST\\_GeoReference](#), [ST\\_SkewY](#), [ST\\_SetSkew](#)

### 11.4.16 ST\_SkewY

ST\_SkewY — 返回地理参考 Y 斜 (或旋参数)。

#### Synopsis

```
float8 ST_SkewY(raster rast);
```

## 描述

返回地理参考 Y 斜 (或旋参数)。有关更多信息, 参世界文件。

## 示例

```
SELECT rid, ST_SkewX(rast) As skewx, ST_SkewY(rast) As skewy,
       ST_GeoReference(rast) as georef
FROM dummy_rast;
```

rid	skewx	skewy	georef
1	0	0	2.0000000000

```

: 0.0000000000
: 0.0000000000
: 3.0000000000
: 0.5000000000
: 0.5000000000
:
2 | 0 | 0 | 0.0500000000
: 0.0000000000
: 0.0000000000
: -0.0500000000
: 3427927.7500000000
: 5793244.0000000000

```

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);
```

描述

返回在 Spatial\_ref\_sys 表中定义的栅格对象的空参考标识符。



#### Note

从 PostGIS 2.0 开始，非地理参考栅格/几何图形的 srid 为 0，而不是之前的 -1。

示例

```
SELECT ST_SRID(rast) As srid
FROM dummy_rast WHERE rid=1;
```

```

srid
-----
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);
```

#### 描述

返回栅格内容的文本摘要。

可用性：2.1.0

#### 示例

```
SELECT ST_Summary(
  ST_AddBand(
    ST_AddBand(
      ST_AddBand(
        ST_MakeEmptyRaster(10, 10, 0, 0, 1, -1, 0, 0, 0)
        , 1, '8BUI', 1, 0
      )
      , 2, '32BF', 0, -9999
    )
    , 3, '16BSI', 0, NULL
  )
);
```

```

-----
                    st_summary
-----
Raster of 10x10 pixels has 3 bands and extent of BOX(0 -10,10 0)+
band 1 of pixtype 8BUI is in-db with NODATA value of 0      +
band 2 of pixtype 32BF is in-db with NODATA value of -9999  +
band 3 of pixtype 16BSI is in-db with no NODATA value
(1 row)
```

#### 相关信息

[ST\\_MetaData](#), [ST\\_BandMetaData](#), [ST\\_Summary](#) [ST\\_Extent](#)

### 11.4.19 ST\_UpperLeftX

ST\_UpperLeftX — 返回投影空参考中栅格的左上角 X 坐标。

#### Synopsis

```
float8 ST_UpperLeftX(raster rast);
```

#### 描述

返回投影空参考中栅格的左上角 X 坐标。

示例

```
SELECT rid, ST_UpperLeftX(rast) As ulx
FROM dummy_rast;
```

rid	ulx
1	0.5
2	3427927.75

相关信息

[ST\\_UpperLeftY](#), [ST\\_GeoReference](#), [Box3D](#)

### 11.4.20 ST\_UpperLeftY

`ST_UpperLeftY` — 返回投影空格的参考中格的左上角 Y 坐标。

#### Synopsis

float8 `ST_UpperLeftY`(raster rast);

描述

返回投影空格的参考中格的左上角 Y 坐标。

示例

```
SELECT rid, ST_UpperLeftY(rast) As uly
FROM dummy_rast;
```

rid	uly
1	0.5
2	5793244

相关信息

[ST\\_UpperLeftX](#), [ST\\_GeoReference](#), [Box3D](#)

### 11.4.21 ST\_Width

`ST_Width` — 返回格的宽度（以像素位）。

#### Synopsis

integer `ST_Width`(raster rast);

## 描述

返回栅格的宽度（以像素为单位）。

## 示例

```
SELECT ST_Width(rast) As rastwidth
FROM dummy_rast WHERE rid=1;
```

```
rastwidth
-----
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);
```

## 描述

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。无论几何 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

```
SELECT
  rid,
  (ST_WorldToRasterCoord(rast,3427927.8,20.5)).*,
  (ST_WorldToRasterCoord(rast,ST_GeomFromText('POINT(3427927.8 20.5)',ST_SRID(rast)))).*
FROM dummy_rast;
```

```
rid | columnx | rowy | columnx | rowy
----+-----+-----+-----+-----
  1 | 1713964 |    7 | 1713964 |    7
  2 |      2 | 115864471 |      2 | 115864471
```

## 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);
```

#### 描述

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系中。

更改：2.1.0 在之前的版本中，称 ST\_World2RasterCoordX

#### 示例

```
SELECT rid, ST_WorldToRasterCoordX(rast,3427927.8) As xcoord,
       ST_WorldToRasterCoordX(rast,3427927.8,20.5) As xcoord_xwyw,
       ST_WorldToRasterCoordX(rast,ST_GeomFromText('POINT(3427927.8 20.5)',ST_SRID(rast))) ←
       As ptxcoord
FROM dummy_rast;
```

rid	xcoord	xcoord_xwyw	ptxcoord
1	1713964	1713964	1713964
2	1	1	1

#### 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);
```

#### 描述

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参考系中。

更改：2.1.0 在之前的版本中，称 ST\_World2RasterCoordY

示例

```
SELECT rid, ST_WorldToRasterCoordY(rast,20.5) As ycoord,
       ST_WorldToRasterCoordY(rast,3427927.8,20.5) As ycoord_xwyw,
       ST_WorldToRasterCoordY(rast,ST_GeomFromText('POINT(3427927.8 20.5)',ST_SRID(rast))) <->
       As ptycoord
FROM dummy_rast;
```

rid	ycoord	ycoord_xwyw	ptycoord
1	7	7	7
2	115864471	115864471	115864471

相关信息

[ST\\_RasterToWorldCoordX](#), [ST\\_RasterToWorldCoordY](#), [ST\\_SRID](#)

## 11.5 栅格波段器

### 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)`;

描述

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!

#### Note

如果栅格不包含波段，会引发。

Note!

#### Note

如果 `band` 没有 `NODATA`，`nodatavalue` `NULL`。

Note!

#### Note

如果 `isoutdb` `False`，路径、`outdbbandnum`、文件大小和文件戳 `NULL`。如果禁用 `outdb`，文件大小和文件戳也将 `NULL`。

增：2.5.0 包括 `outdbbandnum`、文件大小和文件戳 for `outdb` 栅格。



示例：格式 1

```
SELECT
  rid,
  (foo.md).*
FROM (
  SELECT
    rid,
    ST_BandMetaData(rast, 1) AS md
  FROM dummy_rast
  WHERE rid=2
) As foo;
```

rid	pixeltype	nodatavalue	isoutdb	path	outdbbandnum
2	8BUI		0	f	

示例：格式 2

```
WITH foo AS (
  SELECT
    ST_AddBand(NULL::raster, '/home/pele/devel/geo/postgis-git/raster/test/regress/ ↵
    loader/Projected.tif', NULL::int[]) AS rast
)
SELECT
  *
FROM ST_BandMetadata(
  (SELECT rast FROM foo),
  ARRAY[1,3,2]::int[]
);
```

bandnum	pixeltype	nodatavalue	isoutdb	outdbbandnum	filesize	filetimestamp	path ↵
1	8BUI		t	1	12345	1521807257	/home/pele/devel/geo/postgis-git/raster/test ↵ /regress/loader/Projected.tif
3	8BUI		t	3	12345	1521807257	/home/pele/devel/geo/postgis-git/raster/test ↵ /regress/loader/Projected.tif
2	8BUI		t	2	12345	1521807257	/home/pele/devel/geo/postgis-git/raster/test ↵ /regress/loader/Projected.tif

相关信息

[ST\\_MetaData](#), [ST\\_BandPixelType](#)

## 11.5.2 ST\_BandNoDataValue

`ST_BandNoDataValue` — 返回表示无数据的☐定波段中的☐。如果没有指定波段☐号，默☐☐波段 1。

### Synopsis

double precision **ST\_BandNoDataValue**(raster rast, integer bandnum=1);

## 描述

返回表示栅格波段中无数据的数。

## 示例

```
SELECT ST_BandNoDataValue(rast,1) As bnval1,
       ST_BandNoDataValue(rast,2) As bnval2, ST_BandNoDataValue(rast,3) As bnval3
FROM dummy_rast
WHERE rid = 2;
```

bnval1	bnval2	bnval3
0	0	0

## 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=true);
boolean ST_BandIsNoData(raster rast, boolean forceChecking=true);
```

## 描述

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。否则，函数返回栅格的 `isnodata` 标志。如果未指定，此参数的默认为 `FALSE`。

可用性: 2.0.0



### Note

如果标志位“dirty”（即，使用 `TRUE` 作最后一个参数和不使用它的结果不同），您更新栅格，将标志置 `true`，可以使用 `ST_SetBandIsNodata()` 或 `ST_SetBandNodataValue()`，并将 `TRUE` 作最后一个参数。参见 [ST\\_SetBandIsNoData](#)。

## 示例

```
-- Create dummy table with one raster column
create table dummy_rast (rid integer, rast raster);

-- Add raster with two bands, one pixel/band. In the first band, nodatavalue = pixel value ←
= 3.
-- In the second band, nodatavalue = 13, pixel value = 4
insert into dummy_rast values(1,
(
```

```

'01' -- little endian (uint8 ndr)
||
'0000' -- version (uint16 0)
||
'0200' -- nBands (uint16 0)
||
'17263529ED684A3F' -- scaleX (float64 0.000805965234044584)
||
'F9253529ED684ABF' -- scaleY (float64 -0.00080596523404458)
||
'1C9F33CE69E352C0' -- ipX (float64 -75.5533328537098)
||
'718F0E9A27A44840' -- ipY (float64 49.2824585505576)
||
'ED50EB853EC32B3F' -- skewX (float64 0.000211812383858707)
||
'7550EB853EC32B3F' -- skewY (float64 0.000211812383858704)
||
'E6100000' -- SRID (int32 4326)
||
'0100' -- width (uint16 1)
||
'0100' -- height (uint16 1)
||
'6' -- hasnodatavalue and isnodata value set to true.
||
'2' -- first band type (4BUI)
||
'03' -- novalue==3
||
'03' -- pixel(0,0)==3 (same that nodata)
||
'0' -- hasnodatavalue set to false
||
'5' -- second band type (16BSI)
||
'0D00' -- novalue==13
||
'0400' -- pixel(0,0)==4
)::raster
);

select st_bandisnodata(rast, 1) from dummy_rast where rid = 1; -- Expected true
select st_bandisnodata(rast, 2) from dummy_rast where rid = 1; -- Expected false

```

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);
```

## 描述

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

## 相关信息

### 11.5.5 ST\_BandFileSize

`ST_BandFileSize` — 返回文件系中存中的波段的文件大小。如果未指定波段号，假定 1。

#### Synopsis

```
bigint ST_BandFileSize(raster rast, integer bandnum=1);
```

## 描述

返回文件系中存中的波段的文件大小。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` 格式文件名是否与服器看到的文件名相同。

可用性：2.5.0

## 示例

```
SELECT ST_BandFileSize(rast,1) FROM dummy_rast WHERE rid = 1;
```

```
st_bandfilesize
-----
          240574
```

### 11.5.6 ST\_BandFileTimestamp

`ST_BandFileTimestamp` — 返回文件系中存中的波段的文件戳。如果未指定波段号，假定 1。

#### Synopsis

```
bigint ST_BandFileTimestamp(raster rast, integer bandnum=1);
```

## 描述

返回文件系中存中的波段的文件戳（自 1970 年 1 月 1 日 00:00:00 UTC 以来的秒数）。如果使用 `in db band` 用，或者未用 `outdb`，会引。

此函数通常与 `ST_BandPath()` 和 `ST_BandFileSize()` 合使用，以便客端可以确定它所看到的 `outdb` 格式的文件名是否与服器所看到的相同。

可用性：2.5.0

示例

```
SELECT ST_BandFileTimestamp(rast,1) FROM dummy_rast WHERE rid = 1;

 st_bandfiletimestamp
-----
          1521807257
```

### 11.5.7 ST\_BandPixelType

ST\_BandPixelType — 返回指定波段的像素型。如果未指定波段号，假定 1。

#### Synopsis

```
text ST_BandPixelType(raster rast, integer bandnum=1);
```

描述

返回描述指定区每个元格中存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
- 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

```
SELECT ST_BandPixelType(rast,1) As btype1,
       ST_BandPixelType(rast,2) As btype2, ST_BandPixelType(rast,3) As btype3
FROM dummy_rast
WHERE rid = 2;

 btype1 | btype2 | btype3
-----+-----+-----
  8BUI  |  8BUI  |  8BUI
```

相关信息

[ST\\_NumBands](#)

### 11.5.8 ST\_MinPossibleValue

ST\_MinPossibleValue — 返回此像素☐型可以存☐的最小☐。

#### Synopsis

```
integer ST_MinPossibleValue(text pixeltype);
```

描述

返回此像素☐型可以存☐的最小☐。

示例

```
SELECT ST_MinPossibleValue('16BSI');
```

```
  st_minpossiblevalue
-----
                -32768
```

```
SELECT ST_MinPossibleValue('8BUI');
```

```
  st_minpossiblevalue
-----
                    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);
```

描述

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

```
SELECT rid, ST_HasNoBand(rast) As hb1, ST_HasNoBand(rast,2) as hb2,
ST_HasNoBand(rast,4) as hb4, ST_NumBands(rast) As numbands
FROM dummy_rast;
```

rid	hb1	hb2	hb4	numbands
1	t	t	t	0
2	f	f	t	3

相关信息

[ST\\_NumBands](#)

## 11.6 栅格像素提取器和置器

### 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);

描述

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

```
-- get raster pixel polygon
SELECT i,j, ST_AsText(ST_PixelAsPolygon(foo.rast, i,j)) As blpgeom
FROM dummy_rast As foo
  CROSS JOIN generate_series(1,2) As i
  CROSS JOIN generate_series(1,1) As j
WHERE rid=2;
```

i	j	blpgeom
1	1	POLYGON((3427927.75 5793244,3427927.8 5793244,3427927.8 5793243.95,...
2	1	POLYGON((3427927.8 5793244,3427927.85 5793244,3427927.85 5793243.95, ..

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以及每个像素的、X 和 Y 栅格坐标。

### Synopsis

```
setof record ST_PixelAsPolygons(raster rast, integer band=1, boolean exclude_nodata_value=TRUE);
```

### 描述

返回包栅格的每个像素的多边形几何以及每个像素的 (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

当 `except_nodata_value = TRUE` 时, 将那些不是 NODATA 的像素作点返回。



#### Note

ST\_PixelAsPolygons 每个像素返回一个多边形几何。与 ST\_DumpAsPolygons 不同, 其中每个几何形代表一个或多个具有相同像素值的像素。

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

```
-- get raster pixel polygon
SELECT (gv).x, (gv).y, (gv).val, ST_AsText((gv).geom) geom
FROM (SELECT ST_PixelAsPolygons(
        ST_SetValue(ST_SetValue(ST_AddBand(ST_MakeEmptyRaster(2, 2, 0, 0, 0.001, ←
        -0.001, 0.001, 0.001, 4269),
        '8BUI'::text, 1, 0),
        2, 2, 10),
        1, 1, NULL)
) gv
) foo;
```

x	y	val	geom
1	1		POLYGON((0 0,0.001 0.001,0.002 0,0.001 -0.001,0 0))
1	2	1	POLYGON((0.001 -0.001,0.002 0,0.003 -0.001,0.002 -0.002,0.001 -0.001))
2	1	1	POLYGON((0.001 0.001,0.002 0.002,0.003 0.001,0.002 0,0.001 0.001))
2	2	10	POLYGON((0.002 0,0.003 0.001,0.004 0,0.003 -0.001,0.002 0))

### 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);

#### 描述

返回像素左上角的点几何形状。

可用性：2.1.0

#### 示例

```
SELECT ST_AsText(ST_PixelAsPoint(rast, 1, 1)) FROM dummy_rast WHERE rid = 1;

  st_astext
-----
POINT(0.5 0.5)
```

#### 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);

#### 描述

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

```
SELECT x, y, val, ST_AsText(geom) FROM (SELECT (ST_PixelAsPoints(rast, 1)).* FROM
dummy_rast WHERE rid = 2) foo;
```

x	y	val	st_astext
1	1	253	POINT(3427927.75 5793244)
2	1	254	POINT(3427927.8 5793244)
3	1	253	POINT(3427927.85 5793244)
4	1	254	POINT(3427927.9 5793244)
5	1	254	POINT(3427927.95 5793244)
1	2	253	POINT(3427927.75 5793243.95)
2	2	254	POINT(3427927.8 5793243.95)
3	2	254	POINT(3427927.85 5793243.95)
4	2	253	POINT(3427927.9 5793243.95)
5	2	249	POINT(3427927.95 5793243.95)
1	3	250	POINT(3427927.75 5793243.9)
2	3	254	POINT(3427927.8 5793243.9)
3	3	254	POINT(3427927.85 5793243.9)
4	3	252	POINT(3427927.9 5793243.9)
5	3	249	POINT(3427927.95 5793243.9)
1	4	251	POINT(3427927.75 5793243.85)
2	4	253	POINT(3427927.8 5793243.85)
3	4	254	POINT(3427927.85 5793243.85)
4	4	254	POINT(3427927.9 5793243.85)
5	4	253	POINT(3427927.95 5793243.85)
1	5	252	POINT(3427927.75 5793243.8)
2	5	250	POINT(3427927.8 5793243.8)
3	5	254	POINT(3427927.85 5793243.8)
4	5	254	POINT(3427927.9 5793243.8)
5	5	254	POINT(3427927.95 5793243.8)

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☒心（点几何）。

#### Synopsis

geometry **ST\_PixelAsCentroid**(raster rast, integer x, integer y);

描述

返回像素表示的区域的☒心（点几何）。

增☒：3.2.0 Faster ☒在用 C ☒☒。

可用性：2.1.0

示例

```
SELECT ST_AsText(ST_PixelAsCentroid(rast, 1, 1)) FROM dummy_rast WHERE rid = 1;

 st_astext
-----
POINT(1.5 2)
```

相关信息

[ST\\_DumpAsPolygons](#), [ST\\_PixelAsPolygon](#), [ST\\_PixelAsPolygons](#), [ST\\_PixelAsPoint](#), [ST\\_PixelAsPoints](#), [ST\\_Pixel](#)

### 11.6.6 ST\_PixelAsCentroids

`ST_PixelAsCentroids` — 返回栅格波段的每个像素的几何中心（点几何）以及每个像素的几何、X 和 Y 栅格坐标。点几何是像素表示的区域的几何中心。

#### Synopsis

setof record `ST_PixelAsCentroids`(raster rast, integer band=1, boolean exclude\_nodata\_value=TRUE);

描述

返回栅格波段的每个像素的几何中心（点几何）以及每个像素的几何、X 和 Y 栅格坐标。点几何是像素表示的区域的几何中心。  
返回几何格式：`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 的像素作为点返回。

新增：3.2.0 Faster 在用 C 实现。

更改：2.1.1 更改了 `except_nodata_value` 的行。

可用性：2.1.0

示例

```
--LATERAL syntax requires PostgreSQL 9.3+
SELECT x, y, val, ST_AsText(geom)
  FROM (SELECT dp.* FROM dummy_rast, LATERAL ST_PixelAsCentroids(rast, 1) AS dp WHERE rid <=
        = 2) foo;
 x | y | val |          st_astext
---+---+---+-----
 1 | 1 | 253 | POINT(3427927.775 5793243.975)
 2 | 1 | 254 | POINT(3427927.825 5793243.975)
 3 | 1 | 253 | POINT(3427927.875 5793243.975)
 4 | 1 | 254 | POINT(3427927.925 5793243.975)
 5 | 1 | 254 | POINT(3427927.975 5793243.975)
 1 | 2 | 253 | POINT(3427927.775 5793243.925)
```

```

2 | 2 | 254 | POINT(3427927.825 5793243.925)
3 | 2 | 254 | POINT(3427927.875 5793243.925)
4 | 2 | 253 | POINT(3427927.925 5793243.925)
5 | 2 | 249 | POINT(3427927.975 5793243.925)
1 | 3 | 250 | POINT(3427927.775 5793243.875)
2 | 3 | 254 | POINT(3427927.825 5793243.875)
3 | 3 | 254 | POINT(3427927.875 5793243.875)
4 | 3 | 252 | POINT(3427927.925 5793243.875)
5 | 3 | 249 | POINT(3427927.975 5793243.875)
1 | 4 | 251 | POINT(3427927.775 5793243.825)
2 | 4 | 253 | POINT(3427927.825 5793243.825)
3 | 4 | 254 | POINT(3427927.875 5793243.825)
4 | 4 | 254 | POINT(3427927.925 5793243.825)
5 | 4 | 253 | POINT(3427927.975 5793243.825)
1 | 5 | 252 | POINT(3427927.775 5793243.775)
2 | 5 | 250 | POINT(3427927.825 5793243.775)
3 | 5 | 254 | POINT(3427927.875 5793243.775)
4 | 5 | 254 | POINT(3427927.925 5793243.775)
5 | 5 | 254 | POINT(3427927.975 5793243.775)

```

## 相关信息

[ST\\_DumpAsPolygons](#), [ST\\_PixelAsPolygon](#), [ST\\_PixelAsPolygons](#), [ST\\_PixelAsPoint](#), [ST\\_PixelAsPoints](#), [ST\\_Pixel](#)

### 11.6.7 ST\_Value

**ST\_Value** — 返回指定列 *x*、行 *y* 像素或特定几何点  $(x, y)$  在指定波段的值。波段号从 1 开始，如果未指定，默认为 1。如果将参数 `exclude_nodata_value` 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);

```

## 描述

返回指定列 *x*、行 *y* 像素或在指定几何点  $(x, y)$  在指定波段的值。波段号从 1 开始，如果未指定波段，默认为 1。

如果 `except_nodata_value` 置为 `true`，只考虑非 `nodata` 像素。如果 `except_nodata_value` 置为 `false`，考虑所有像素。

重新采样参数的允许值：“nearest”（执行默认的最近重采样）和“bilinear”（执行双线性插值以估计像素中心之处的值）。

新增：添加了 3.2.0 重新采样可选项参数。

新增：2.0.0 添加了 `except_nodata_value` 可选项参数。

## 示例

```
-- get raster values at particular postgis geometry points
-- the srid of your geometry should be same as for your raster
SELECT rid, ST_Value(rast, foo.pt_geom) As b1pval, ST_Value(rast, 2, foo.pt_geom) As b2pval
FROM dummy_rast CROSS JOIN (SELECT ST_SetSRID(ST_Point(3427927.77, 5793243.76), 0) As ←
    pt_geom) As foo
WHERE rid=2;
```

rid	b1pval	b2pval
2	252	79

```
-- general fictitious example using a real table
SELECT rid, ST_Value(rast, 3, sometable.geom) As b3pval
FROM sometable
WHERE ST_Intersects(rast,sometable.geom);
```

```
SELECT rid, ST_Value(rast, 1, 1, 1) As b1pval,
    ST_Value(rast, 2, 1, 1) As b2pval, ST_Value(rast, 3, 1, 1) As b3pval
FROM dummy_rast
WHERE rid=2;
```

rid	b1pval	b2pval	b3pval
2	253	78	70

```
--- Get all values in bands 1,2,3 of each pixel --
SELECT x, y, ST_Value(rast, 1, x, y) As b1val,
    ST_Value(rast, 2, x, y) As b2val, ST_Value(rast, 3, x, y) As b3val
FROM dummy_rast CROSS JOIN
generate_series(1, 1000) As x CROSS JOIN generate_series(1, 1000) As y
WHERE rid = 2 AND x <= ST_Width(rast) AND y <= ST_Height(rast);
```

x	y	b1val	b2val	b3val
1	1	253	78	70
1	2	253	96	80
1	3	250	99	90
1	4	251	89	77
1	5	252	79	62
2	1	254	98	86
2	2	254	118	108
:				
:				

```
--- Get all values in bands 1,2,3 of each pixel same as above but returning the upper left ←
point point of each pixel --
SELECT ST_AsText(ST_SetSRID(
    ST_Point(ST_UpperLeftX(rast) + ST_ScaleX(rast)*x,
        ST_UpperLeftY(rast) + ST_ScaleY(rast)*y),
        ST_SRID(rast))) As uplpt
    , ST_Value(rast, 1, x, y) As b1val,
    ST_Value(rast, 2, x, y) As b2val, ST_Value(rast, 3, x, y) As b3val
FROM dummy_rast CROSS JOIN
generate_series(1,1000) As x CROSS JOIN generate_series(1,1000) As y
```

```
WHERE rid = 2 AND x <= ST_Width(rast) AND y <= ST_Height(rast);
```

uplpt	b1val	b2val	b3val
POINT(3427929.25 5793245.5)	253	78	70
POINT(3427929.25 5793247)	253	96	80
POINT(3427929.25 5793248.5)	250	99	90

```

--- Get a polygon formed by union of all pixels
    that fall in a particular value range and intersect particular polygon --
SELECT ST_AsText(ST_Union(pixpolyg)) As shadow
FROM (SELECT ST_Translate(ST_MakeEnvelope(
    ST_UpperLeftX(rast), ST_UpperLeftY(rast),
    ST_UpperLeftX(rast) + ST_ScaleX(rast),
    ST_UpperLeftY(rast) + ST_ScaleY(rast), 0
    ), ST_ScaleX(rast)*x, ST_ScaleY(rast)*y
    ) As pixpolyg, ST_Value(rast, 2, x, y) As b2val
    FROM dummy_rast CROSS JOIN
generate_series(1,1000) As x CROSS JOIN generate_series(1,1000) As y
WHERE rid = 2
    AND x <= ST_Width(rast) AND y <= ST_Height(rast)) As foo
WHERE
    ST_Intersects(
        pixpolyg,
        ST_GeomFromText('POLYGON((3427928 5793244,3427927.75 5793243.75,3427928 ←
        5793243.75,3427928 5793244))',0)
    ) AND b2val != 254;

    shadow
-----
MULTIPOLYGON(((3427928 5793243.9,3427928 5793243.85,3427927.95 5793243.85,3427927.95 ←
    5793243.9,
    3427927.95 5793243.95,3427928 5793243.95,3427928.05 5793243.95,3427928.05 ←
    5793243.9,3427928 5793243.9)),((3427927.95 5793243.9,3427927.95 579324
    3.85,3427927.9 5793243.85,3427927.85 5793243.85,3427927.85 5793243.9,3427927.9 ←
    5793243.9,3427927.9 5793243.95,
    3427927.95 5793243.95,3427927.95 5793243.9)),((3427927.85 5793243.75,3427927.85 ←
    5793243.7,3427927.8 5793243.7,3427927.8 5793243.75
    ,3427927.8 5793243.8,3427927.8 5793243.85,3427927.85 5793243.85,3427927.85 ←
    5793243.8,3427927.85 5793243.75))),
    ((3427928.05 5793243.75,3427928.05 5793243.7,3427928 5793243.7,3427927.95 ←
    5793243.7,3427927.95 5793243.75,3427927.95 5793243.8,3427
    927.95 5793243.85,3427928 5793243.85,3427928 5793243.8,3427928.05 5793243.8,
    3427928.05 5793243.75)),((3427927.95 5793243.75,3427927.95 5793243.7,3427927.9 ←
    5793243.7,3427927.85 5793243.7,
    3427927.85 5793243.75,3427927.85 5793243.8,3427927.85 5793243.85,3427927.9 5793243.85,
    3427927.95 5793243.85,3427927.95 5793243.8,3427927.95 5793243.75)))

```

```

--- Checking all the pixels of a large raster tile can take a long time.
--- You can dramatically improve speed at some lose of precision by orders of magnitude
-- by sampling pixels using the step optional parameter of generate_series.
-- This next example does the same as previous but by checking 1 for every 4 (2x2) pixels ←
and putting in the last checked
-- putting in the checked pixel as the value for subsequent 4

```

```

SELECT ST_AsText(ST_Union(pixpolyg)) As shadow
FROM (SELECT ST_Translate(ST_MakeEnvelope(
    ST_UpperLeftX(rast), ST_UpperLeftY(rast),
    ST_UpperLeftX(rast) + ST_ScaleX(rast)*2,
    ST_UpperLeftY(rast) + ST_ScaleY(rast)*2, 0
    ), ST_ScaleX(rast)*x, ST_ScaleY(rast)*y
    ) As pixpolyg, ST_Value(rast, 2, x, y) As b2val
FROM dummy_rast CROSS JOIN
generate_series(1,1000,2) As x CROSS JOIN generate_series(1,1000,2) As y
WHERE rid = 2
AND x <= ST_Width(rast) AND y <= ST_Height(rast) ) As foo
WHERE
ST_Intersects(
    pixpolyg,
    ST_GeomFromText('POLYGON(((3427928 5793244,3427927.75 5793243.75,3427928
    5793243.75,3427928 5793244)))',0)
) AND b2val != 254;

shadow
-----
MULTIPOLYGON(((3427927.9 5793243.85,3427927.8 5793243.85,3427927.8 5793243.95,
3427927.9 5793243.95,3427928 5793243.95,3427928.1 5793243.95,3427928.1 5793243.85,3427928
5793243.85,3427927.9 5793243.85))),
((3427927.9 5793243.65,3427927.8 5793243.65,3427927.8 5793243.75,3427927.8
5793243.85,3427927.9 5793243.85,
3427928 5793243.85,3427928 5793243.75,3427928.1 5793243.75,3427928.1 5793243.65,3427928
5793243.65,3427927.9 5793243.65)))

```

## 相关信息

[ST\\_SetValue](#), [ST\\_DumpAsPolygons](#), [ST\\_NumBands](#), [ST\\_PixelAsPolygon](#), [ST\\_ScaleX](#), [ST\\_ScaleY](#), [ST\\_UpperLeftX](#), [ST\\_UpperLeftY](#), [ST\\_SRID](#), [ST\\_AsText](#), [ST\\_Point](#), [ST\\_MakeEnvelope](#), [ST\\_Intersects](#), [ST\\_Intersection](#)

## 11.6.8 ST\_NearestValue

`ST_NearestValue` — 返回由 `columnx` 和 `rowy` 指定的栅格像素的最接近的非 `NODATA` 像素或以与栅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);
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);

```

### 描述

返回栅格行列 `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`，函数将找到距离列 `x`、行 `y` 像素或非 `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，从栅格的元数据中取它。

可用性 : 2.1.0



### Note

ST\_NearestValue 是 ST\_Value 的直接替代品。

示例

```
-- pixel 2x2 has value
SELECT
  ST_Value(rast, 2, 2) AS value,
  ST_NearestValue(rast, 2, 2) AS nearestvalue
FROM (
  SELECT
    ST_SetValue(
      ST_SetValue(
        ST_SetValue(
          ST_SetValue(
            ST_SetValue(
              ST_AddBand(
                ST_MakeEmptyRaster(5, 5, -2, 2, 1, -1, 0, 0, 0),
                '8BUI'::text, 1, 0
              ),
              1, 1, 0.
            ),
            2, 3, 0.
          ),
          3, 5, 0.
        ),
        4, 2, 0.
      ),
      5, 4, 0.
    ) AS rast
  ) AS foo

value | nearestvalue
-----+-----
1 | 1
```

```
-- pixel 2x3 is NODATA
SELECT
  ST_Value(rast, 2, 3) AS value,
  ST_NearestValue(rast, 2, 3) AS nearestvalue
FROM (
  SELECT
    ST_SetValue(
      ST_SetValue(
        ST_SetValue(
          ST_SetValue(
            ST_SetValue(
              ST_AddBand(
                ST_MakeEmptyRaster(5, 5, -2, 2, 1, -1, 0, 0, 0),
                '8BUI'::text, 1, 0
              ),
              1, 1, 0.
            ),
            2, 3, 0.
          ),
          3, 5, 0.
        ),
        4, 2, 0.
      ),
      5, 4, 0.
    ) AS rast
  ) AS foo
```



```

        3, 5, 0.
    ),
    4, 2, 0.
),
5, 4, 0.
) AS rast
) AS foo

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);
```

#### 描述

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”，以复制每个点所在栅格单元的值，或者置为“bilinear”，以使用双线性插值来估算考虑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
--
WITH test_raster AS (
SELECT
ST_SetValues(
  ST_AddBand(
    ST_MakeEmptyRaster(width => 2, height => 2,
      upperleftx => 0, upperlefty => 2,
      scalex => 1.0, scaley => -1.0,
      skewx => 0, skewy => 0, srid => 4326),
    index => 1, pixeltype => '16BSI',
    initialvalue => 0,
    nodataval => -999),
  1,1,1,

```

```

newvalueset =>ARRAY[ARRAY[10.0::float8, 50.0::float8], ARRAY[40.0::float8, 20.0::float8 ←
  ]]) AS rast
)
SELECT
ST_AsText(
  ST_SetZ(
    rast,
    band => 1,
    geom => 'SRID=4326;LINESTRING(1.0 1.9, 1.0 0.2)::geometry,
    resample => 'bilinear'
  ))
FROM test_raster

          st_astext
-----
LINESTRING Z (1 1.9 38,1 0.2 27)

```

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);

#### 描述

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”，以复制每个点所在元格的值，或者置为“bilinear”，以使用双线性插值来估算相邻元格的值。

可用性：3.2.0

#### 示例

```

--
-- 2x2 test raster with values
--
-- 10 50
-- 40 20
--
WITH test_raster AS (
SELECT
ST_SetValues(
  ST_AddBand(
    ST_MakeEmptyRaster(width => 2, height => 2,
      upperleftx => 0, upperlefty => 2,
      scalex => 1.0, scaley => -1.0,

```

```

        skewx => 0, skewy => 0, srid => 4326),
        index => 1, pixeltype => '16BSI',
        initialvalue => 0,
        nodataval => -999),
        1,1,1,
        newvalueset =>ARRAY[ARRAY[10.0::float8, 50.0::float8], ARRAY[40.0::float8, 20.0::float8 ←
            ]]) AS rast
    )
SELECT
ST_AsText(
  ST_SetM(
    rast,
    band => 1,
    geom => 'SRID=4326;LINESTRING(1.0 1.9, 1.0 0.2)::geometry,
    resample => 'bilinear'
  ))
FROM test_raster

          st_astext
-----
LINESTRING M (1 1.9 38,1 0.2 27)

```

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);

```

#### 描述

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。二精度的中心将是列 **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**

返回的二数沿每个的元素数量  $2 * (\text{distanceX}|\text{distanceY}) + 1$ 。因此，于 distanceX 和 distanceY 为 1，返回的数将 3x3。

**Note**

二数出可以任何格理内置函数，例如 ST\_Min4ma、ST\_Sum4ma、ST\_Mean4ma。

可用性 : 2.1.0

示例

```
-- pixel 2x2 has value
SELECT
  ST_Neighborhood(rast, 2, 2, 1, 1)
FROM (
  SELECT
    ST_SetValues(
      ST_AddBand(
        ST_MakeEmptyRaster(5, 5, -2, 2, 1, -1, 0, 0, 0),
        '8BUI'::text, 1, 0
      ),
      1, 1, 1, ARRAY[
        [0, 1, 1, 1, 1],
        [1, 1, 1, 0, 1],
        [1, 0, 1, 1, 1],
        [1, 1, 1, 1, 0],
        [1, 1, 0, 1, 1]
      ]::double precision[],
      1
    ) AS rast
  ) AS foo

      st_neighborhood
-----
{{NULL,1,1},{1,1,1},{1,NULL,1}}
```

```
-- pixel 2x3 is NODATA
SELECT
  ST_Neighborhood(rast, 2, 3, 1, 1)
FROM (
  SELECT
    ST_SetValues(
      ST_AddBand(
        ST_MakeEmptyRaster(5, 5, -2, 2, 1, -1, 0, 0, 0),
        '8BUI'::text, 1, 0
      ),
      1, 1, 1, ARRAY[
        [0, 1, 1, 1, 1],
        [1, 1, 1, 0, 1],
        [1, 0, 1, 1, 1],
        [1, 1, 1, 1, 0],
        [1, 1, 0, 1, 1]
      ]::double precision[],
      1
    ) AS rast
  ) AS foo

      st_neighborhood
-----
{{NULL,1,1,1,1},{1,1,1,1,1},{1,1,1,1,1}}
```

```

) AS rast
) AS foo

      st_neighborhood
-----
{{1,1,1},{1,NULL,1},{1,1,1}}

-- pixel 3x3 has value
-- exclude_nodata_value = FALSE
SELECT
  ST_Neighborhood(rast, 3, 3, 1, 1, false)
FROM ST_SetValues(
  ST_AddBand(
    ST_MakeEmptyRaster(5, 5, -2, 2, 1, -1, 0, 0, 0),
    '8BUI'::text, 1, 0
  ),
  1, 1, 1, ARRAY[
    [0, 1, 1, 1, 1],
    [1, 1, 1, 0, 1],
    [1, 0, 1, 1, 1],
    [1, 1, 1, 1, 0],
    [1, 1, 0, 1, 1]
  ]::double precision[],
  1
) AS rast

      st_neighborhood
-----
{{1,1,0},{0,1,1},{1,1,1}}

```

## 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栅格置指定列 *x*、行 *y* 像素或与特定几何形相交的像素。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);

```

### 描述

返回修改后的栅格，栅格是通过将指定像素的栅格置指定栅格的行和列或几何形的指定波段的新面而生成的。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

```
-- Geometry example
SELECT (foo.geomval).val, ST_AsText(ST_Union((foo.geomval).geom))
FROM (SELECT ST_DumpAsPolygons(
        ST_SetValue(rast,1,
                    ST_Point(3427927.75, 5793243.95),
                    50)
      ) As geomval
FROM dummy_rast
where rid = 2) As foo
WHERE (foo.geomval).val < 250
GROUP BY (foo.geomval).val;
```

val	st_astext
50	POLYGON((3427927.75 5793244,3427927.75 5793243.95,3427927.8 579324 ...
249	POLYGON((3427927.95 5793243.95,3427927.95 5793243.85,3427928 57932 ...

```
-- Store the changed raster --
UPDATE dummy_rast SET rast = ST_SetValue(rast,1, ST_Point(3427927.75, 5793243.95),100)
WHERE rid = 2 ;
```

相关信息

[ST\\_Value](#), [ST\\_DumpAsPolygons](#)

### 11.6.13 ST\_SetValues

ST\_SetValues — 返回通过置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);
```

描述

返回通过指定波段的指定像素置新而生的修改后的栅格。columnx 和 rowy 是从 1 开始索引的。如果 keepnodata 为 TRUE，那些 NODATA 的像素将不会被置 newvalueset 中的相。

对于体 1，要设置的特定像素由 `columnx`、`rowy` 像素坐标和 `newvalueset` 数组的尺寸决定。`noset` 可以用来防止设置具有 `newvalueset` 中存在的值的像素（因 PostgreSQL 不允许不连续数组）。参见体 1 的示例。

体 2 与体 1 类似，但是使用数组的双精度 `nosetvalue` 而不是布型型的 `noset` 数组。`newvalueset` 中具有 `nosetvalue` 值的元素将被跳过。参见体 2 的示例。

对于体 3，要设置的特定像素由 `columnx`、`rowy` 像素坐标、`width` 和 `height` 决定。参见体 3 的示例。

体 4 与体 3 相同，唯一的区别是它假定将设置 `rast` 的第一个波段的像素。

对于体 5，将使用 `geomval` 数组来确定要设置的特定像素。如果数组中的所有几何形状都是点型或多点型，函数将使用一种快捷方式，其中每个点的 `x` 度和 `y` 度用于直接设置像素。否则，几何形状将覆盖网格，然后一次迭代通过它。参见体 5 的示例。

可用性：2.1.0

示例：格式 1

```

/*
The ST_SetValues() does the following...

+ - + - + - +           + - + - + - +
| 1 | 1 | 1 |           | 1 | 1 | 1 |
+ - + - + - +           + - + - + - +
| 1 | 1 | 1 |           =
> | 1 | 9 | 9 |
+ - + - + - +           + - + - + - +
| 1 | 1 | 1 |           | 1 | 9 | 9 |
+ - + - + - +           + - + - + - +
*/
SELECT
    (poly).x,
    (poly).y,
    (poly).val
FROM (
    SELECT
        ST_PixelAsPolygons(
            ST_SetValues(
                ST_AddBand(
                    ST_MakeEmptyRaster(3, 3, 0, 0, 1, -1, 0, 0, 0),
                    1, '8BUI', 1, 0
                ),
                1, 2, 2, ARRAY[[9, 9], [9, 9]]::double precision[]
            )
        ) AS poly
) foo
ORDER BY 1, 2;

x | y | val
---+---+---
1 | 1 | 1
1 | 2 | 1
1 | 3 | 1
2 | 1 | 1
2 | 2 | 9
2 | 3 | 9
3 | 1 | 1
3 | 2 | 9
3 | 3 | 9

```

```

/*
The ST_SetValues() does the following...

+ - + - + - +      + - + - + - +
| 1 | 1 | 1 |      | 9 | 9 | 9 |
+ - + - + - +      + - + - + - +
| 1 | 1 | 1 |      =
> | 9 | | 9 |
+ - + - + - +      + - + - + - +
| 1 | 1 | 1 |      | 9 | 9 | 9 |
+ - + - + - +      + - + - + - +
*/
SELECT
  (poly).x,
  (poly).y,
  (poly).val
FROM (
SELECT
  ST_PixelAsPolygons(
    ST_SetValues(
      ST_AddBand(
        ST_MakeEmptyRaster(3, 3, 0, 0, 1, -1, 0, 0, 0),
        1, '8BUI', 1, 0
      ),
      1, 1, 1, ARRAY[[9, 9, 9], [9, NULL, 9], [9, 9, 9]]::double precision[][]
    )
  ) AS poly
) foo
ORDER BY 1, 2;

x | y | val
---+---+---
1 | 1 | 9
1 | 2 | 9
1 | 3 | 9
2 | 1 | 9
2 | 2 | 9
2 | 3 | 9
3 | 1 | 9
3 | 2 | 9
3 | 3 | 9

```

```

/*
The ST_SetValues() does the following...

+ - + - + - +      + - + - + - +
| 1 | 1 | 1 |      | 9 | 9 | 9 |
+ - + - + - +      + - + - + - +
| 1 | 1 | 1 |      => | 1 | | 9 |
+ - + - + - +      + - + - + - +
| 1 | 1 | 1 |      | 9 | 9 | 9 |
+ - + - + - +      + - + - + - +
*/
SELECT
  (poly).x,
  (poly).y,
  (poly).val
FROM (
SELECT

```



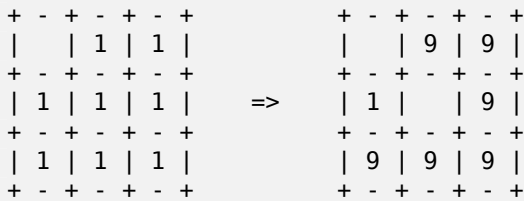
```

ST_PixelAsPolygons(
  ST_SetValues(
    ST_AddBand(
      ST_MakeEmptyRaster(3, 3, 0, 0, 1, -1, 0, 0, 0),
      1, '8BUI', 1, 0
    ),
    1, 1, 1,
    ARRAY[[9, 9, 9], [9, NULL, 9], [9, 9, 9]]::double precision[[[]],
    ARRAY[[false], [true]]::boolean[[[]]
  )
) AS poly
) foo
ORDER BY 1, 2;

```

x	y	val
1	1	9
1	2	1
1	3	9
2	1	9
2	2	
2	3	9
3	1	9
3	2	9
3	3	9

/\*  
The ST\_SetValues() does the following...



```

*/
SELECT
  (poly).x,
  (poly).y,
  (poly).val
FROM (
SELECT
  ST_PixelAsPolygons(
    ST_SetValues(
      ST_SetValue(
        ST_AddBand(
          ST_MakeEmptyRaster(3, 3, 0, 0, 1, -1, 0, 0, 0),
          1, '8BUI', 1, 0
        ),
        1, 1, 1, NULL
      ),
      1, 1, 1,
      ARRAY[[9, 9, 9], [9, NULL, 9], [9, 9, 9]]::double precision[[[]],
      ARRAY[[false], [true]]::boolean[[[]],
      TRUE
    )
  ) AS poly
) foo
ORDER BY 1, 2;

```

x	y	val
---	---	-----

```

+---+---+
1 | 1 | 1
1 | 2 | 1
1 | 3 | 9
2 | 1 | 9
2 | 2 | 9
2 | 3 | 9
3 | 1 | 9
3 | 2 | 9
3 | 3 | 9

```

示例：格式 2

```

/*
The ST_SetValues() does the following...

+ - + - + - +          + - + - + - +
| 1 | 1 | 1 |          | 1 | 1 | 1 |
+ - + - + - +          + - + - + - +
| 1 | 1 | 1 |    =>   | 1 | 9 | 9 |
+ - + - + - +          + - + - + - +
| 1 | 1 | 1 |          | 1 | 9 | 9 |
+ - + - + - +          + - + - + - +
*/
SELECT
  (poly).x,
  (poly).y,
  (poly).val
FROM (
  SELECT
    ST_PixelAsPolygons(
      ST_SetValues(
        ST_AddBand(
          ST_MakeEmptyRaster(3, 3, 0, 0, 1, -1, 0, 0, 0),
          1, '8BUI', 1, 0
        ),
        1, 1, 1, ARRAY[[-1, -1, -1], [-1, 9, 9], [-1, 9, 9]]::double precision[3][3], -1
      )
    ) AS poly
) foo
ORDER BY 1, 2;

x | y | val
+---+---+
1 | 1 | 1
1 | 2 | 1
1 | 3 | 1
2 | 1 | 1
2 | 2 | 9
2 | 3 | 9
3 | 1 | 1
3 | 2 | 9
3 | 3 | 9

```

```

/*
This example is like the previous one. Instead of nosetvalue = -1, nosetvalue = NULL

The ST_SetValues() does the following...

```

```

+ - + - + - +      + - + - + - +
| 1 | 1 | 1 |      | 1 | 1 | 1 |
+ - + - + - +      + - + - + - +
| 1 | 1 | 1 |      => | 1 | 9 | 9 |
+ - + - + - +      + - + - + - +
| 1 | 1 | 1 |      | 1 | 9 | 9 |
+ - + - + - +      + - + - + - +
*/
SELECT
  (poly).x,
  (poly).y,
  (poly).val
FROM (
  SELECT
    ST_PixelAsPolygons(
      ST_SetValues(
        ST_AddBand(
          ST_MakeEmptyRaster(3, 3, 0, 0, 1, -1, 0, 0, 0),
          1, '8BUI', 1, 0
        ),
        1, 1, 1, ARRAY[[NULL, NULL, NULL], [NULL, 9, 9], [NULL, 9, 9]]::double ←
        precision[], NULL::double precision
      )
    ) AS poly
) foo
ORDER BY 1, 2;

x | y | val
---+---+---
1 | 1 | 1
1 | 2 | 1
1 | 3 | 1
2 | 1 | 1
2 | 2 | 9
2 | 3 | 9
3 | 1 | 1
3 | 2 | 9
3 | 3 | 9

```

示例：☒体 3

```

/*
The ST_SetValues() does the following...

+ - + - + - +      + - + - + - +
| 1 | 1 | 1 |      | 1 | 1 | 1 |
+ - + - + - +      + - + - + - +
| 1 | 1 | 1 |      => | 1 | 9 | 9 |
+ - + - + - +      + - + - + - +
| 1 | 1 | 1 |      | 1 | 9 | 9 |
+ - + - + - +      + - + - + - +
*/
SELECT
  (poly).x,
  (poly).y,
  (poly).val
FROM (
  SELECT
    ST_PixelAsPolygons(
      ST_SetValues(

```

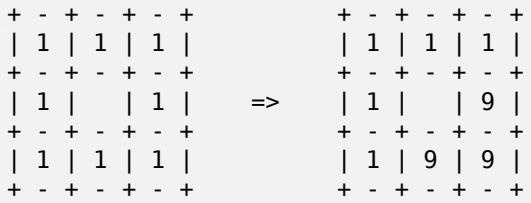
```

        ST_AddBand(
            ST_MakeEmptyRaster(3, 3, 0, 0, 1, -1, 0, 0, 0),
            1, '8BUI', 1, 0
        ),
        1, 2, 2, 2, 2, 9
    )
) AS poly
) foo
ORDER BY 1, 2;

```

x	y	val
1	1	1
1	2	1
1	3	1
2	1	1
2	2	9
2	3	9
3	1	1
3	2	9
3	3	9

/\*  
The ST\_SetValues() does the following...



```

*/
SELECT
    (poly).x,
    (poly).y,
    (poly).val
FROM (
    SELECT
        ST_PixelAsPolygons(
            ST_SetValues(
                ST_SetValue(
                    ST_AddBand(
                        ST_MakeEmptyRaster(3, 3, 0, 0, 1, -1, 0, 0, 0),
                        1, '8BUI', 1, 0
                    ),
                    1, 2, 2, NULL
                ),
                1, 2, 2, 2, 2, 9, TRUE
            )
        ) AS poly
    ) AS foo
ORDER BY 1, 2;

```

x	y	val
1	1	1
1	2	1
1	3	1
2	1	1
2	2	9
2	3	9

3	1	1
3	2	9
3	3	9

示例：☒体 5

```
WITH foo AS (
  SELECT 1 AS rid, ST_AddBand(ST_MakeEmptyRaster(5, 5, 0, 0, 1, -1, 0, 0, 0), 1, '8BUI', ←
    0, 0) AS rast
), bar AS (
  SELECT 1 AS gid, 'SRID=0;POINT(2.5 -2.5)::geometry geom UNION ALL
  SELECT 2 AS gid, 'SRID=0;POLYGON((1 -1, 4 -1, 4 -4, 1 -4, 1 -1))::geometry geom UNION ←
    ALL
  SELECT 3 AS gid, 'SRID=0;POLYGON((0 0, 5 0, 5 -1, 1 -1, 1 -4, 0 -4, 0 0))::geometry ←
    geom UNION ALL
  SELECT 4 AS gid, 'SRID=0;MULTIPOINT(0 0, 4 4, 4 -4)::geometry
)
SELECT
  rid, gid, ST_DumpValues(ST_SetValue(rast, 1, geom, gid))
FROM foo t1
CROSS JOIN bar t2
ORDER BY rid, gid;
```

rid	gid	st_dumpvalues
1	1	(1,"{NULL,NULL,NULL,NULL,NULL},{NULL,NULL,NULL,NULL,NULL},{NULL,NULL,1,NULL, ← NULL},{NULL,NULL,NULL,NULL,NULL},{NULL,NULL,NULL,NULL,NULL}}")
1	2	(1,"{NULL,NULL,NULL,NULL,NULL},{NULL,2,2,2,NULL},{NULL,2,2,2,NULL},{NULL ← ,2,2,2,NULL},{NULL,NULL,NULL,NULL,NULL}}")
1	3	(1,"{3,3,3,3,3},{3,NULL,NULL,NULL,NULL},{3,NULL,NULL,NULL,NULL},{3,NULL,NULL, ← NULL,NULL},{NULL,NULL,NULL,NULL,NULL}}")
1	4	(1,"{4,NULL,NULL,NULL,NULL},{NULL,NULL,NULL,NULL,NULL},{NULL,NULL,NULL,NULL, ← NULL},{NULL,NULL,NULL,NULL,NULL},{NULL,NULL,NULL,NULL,4}}")

(4 rows)

下面☒示了数☒中后面的 geomvals 可以覆盖前面的 geomvals

```
WITH foo AS (
  SELECT 1 AS rid, ST_AddBand(ST_MakeEmptyRaster(5, 5, 0, 0, 1, -1, 0, 0, 0), 1, '8BUI', ←
    0, 0) AS rast
), bar AS (
  SELECT 1 AS gid, 'SRID=0;POINT(2.5 -2.5)::geometry geom UNION ALL
  SELECT 2 AS gid, 'SRID=0;POLYGON((1 -1, 4 -1, 4 -4, 1 -4, 1 -1))::geometry geom UNION ←
    ALL
  SELECT 3 AS gid, 'SRID=0;POLYGON((0 0, 5 0, 5 -1, 1 -1, 1 -4, 0 -4, 0 0))::geometry ←
    geom UNION ALL
  SELECT 4 AS gid, 'SRID=0;MULTIPOINT(0 0, 4 4, 4 -4)::geometry
)
SELECT
  t1.rid, t2.gid, t3.gid, ST_DumpValues(ST_SetValues(rast, 1, ARRAY[ROW(t2.geom, t2.gid), ←
    ROW(t3.geom, t3.gid)]::geomval[]))
FROM foo t1
CROSS JOIN bar t2
CROSS JOIN bar t3
WHERE t2.gid = 1
  AND t3.gid = 2
ORDER BY t1.rid, t2.gid, t3.gid;
```

rid	gid	gid	st_dumpvalues
-----	-----	-----	---------------



示例

```
WITH foo AS (
  SELECT ST_AddBand(ST_AddBand(ST_AddBand(ST_MakeEmptyRaster(3, 3, 0, 0, 1, -1, 0, 0, 0), ←
    1, '8BUI'::text, 1, 0), 2, '32BF'::text, 3, -9999), 3, '16BSI', 0, 0) AS rast
)
SELECT
  (ST_DumpValues(rast)).*
FROM foo;
```

nband	valarray
1	{{1,1,1},{1,1,1},{1,1,1}}
2	{{3,3,3},{3,3,3},{3,3,3}}
3	{{NULL,NULL,NULL},{NULL,NULL,NULL},{NULL,NULL,NULL}}

(3 rows)

```
WITH foo AS (
  SELECT ST_AddBand(ST_AddBand(ST_AddBand(ST_MakeEmptyRaster(3, 3, 0, 0, 1, -1, 0, 0, 0), ←
    1, '8BUI'::text, 1, 0), 2, '32BF'::text, 3, -9999), 3, '16BSI', 0, 0) AS rast
)
SELECT
  (ST_DumpValues(rast, ARRAY[3, 1])).*
FROM foo;
```

nband	valarray
3	{{NULL,NULL,NULL},{NULL,NULL,NULL},{NULL,NULL,NULL}}
1	{{1,1,1},{1,1,1},{1,1,1}}

(2 rows)

```
WITH foo AS (
  SELECT ST_SetValue(ST_AddBand(ST_MakeEmptyRaster(3, 3, 0, 0, 1, -1, 0, 0, 0), 1, '8BUI ←
    ', 1, 0), 1, 2, 5) AS rast
)
SELECT
  (ST_DumpValues(rast, 1))[2][1]
FROM foo;
```

st_dumpvalues
5

(1 row)

相关信息

[ST\\_Value](#), [ST\\_SetValue](#), [ST\\_SetValues](#)

### 11.6.15 ST\_PixelOfValue

ST\_PixelOfValue — 取等于搜索的像素的列 x、行 y 坐。

#### 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 exclude_nodata_value=true );
setof record ST_PixelOfValue( raster rast , double precision search , boolean exclude_nodata_value=true );
```

### 描述

返回等于搜索值的像素的列 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

```
SELECT
  (pixels).*
FROM (
  SELECT
    ST_PixelOfValue(
      ST_SetValue(
        ST_SetValue(
          ST_SetValue(
            ST_SetValue(
              ST_SetValue(
                ST_AddBand(
                  ST_MakeEmptyRaster(5, 5, -2, 2, 1, -1, 0, 0, 0),
                  '8BUI'::text, 1, 0
                ),
                1, 1, 0
              ),
              2, 3, 0
            ),
            3, 5, 0
          ),
          4, 2, 0
        ),
        5, 4, 255
      ),
      1, ARRAY[1, 255]) AS pixels
) AS foo
```

val	x	y
1	1	2
1	1	3
1	1	4
1	1	5
1	2	1
1	2	2
1	2	4
1	2	5
1	3	1
1	3	2
1	3	3
1	3	4
1	4	1
1	4	3
1	4	4



```

1 | 4 | 5
1 | 5 | 1
1 | 5 | 2
1 | 5 | 3
255 | 5 | 4
1 | 5 | 5

```

## 11.7 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);

```

#### 描述

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，返回 null。

格式表示的区间如下：

GDAL:

```
scalex skewy skewx scaley upperleftx upperlefty
```

ESRI:

```
scalex skewy 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显示波段的外部存数据。

增修：2.1.0 添加 `ST_SetGeoReference(raster, double precision, ...)` 函数

#### 示例

```

WITH foo AS (
  SELECT ST_MakeEmptyRaster(5, 5, 0, 0, 1, -1, 0, 0, 0) AS rast
)
SELECT
  0 AS rid, (ST_Metadata(rast)).*
FROM foo
UNION ALL

```

```
SELECT
  1, (ST_Metadata(ST_SetGeoReference(rast, '10 0 0 -10 0.1 0.1', 'GDAL'))).*
FROM foo
UNION ALL
SELECT
  2, (ST_Metadata(ST_SetGeoReference(rast, '10 0 0 -10 5.1 -4.9', 'ESRI'))).*
FROM foo
UNION ALL
SELECT
  3, (ST_Metadata(ST_SetGeoReference(rast, 1, 1, 10, -10, 0.001, 0.001))).*
FROM foo
```

rid	upperleftx skewy	srid	numbands	upperlefty	width	height	scalex	scaley	skewx	↔
0	0	0	0	0	5	5	1	-1	0	↔
1	0	0	0.1	0.1	5	5	10	-10	0	↔
2	0.09999999999999996	0	0.09999999999999996	0.09999999999999996	5	5	10	-10	0	↔
3	0.001	0	1	1	5	5	10	-10	0.001	↔

相关信息

[ST\\_GeoReference](#), [ST\\_ScaleX](#), [ST\\_ScaleY](#), [ST\\_UpperLeftX](#), [ST\\_UpperLeftY](#)

### 11.7.2 ST\_SetRotation

ST\_SetRotation — 以弧度设置栅格的旋。

#### Synopsis

raster **ST\_SetRotation**(raster rast, float8 rotation);

#### 描述

均匀旋栅格。旋以弧度位。有关更多信息，参世界文件。

#### 示例

```
SELECT
  ST_ScaleX(rast1), ST_ScaleY(rast1), ST_SkewX(rast1), ST_SkewY(rast1),
  ST_ScaleX(rast2), ST_ScaleY(rast2), ST_SkewX(rast2), ST_SkewY(rast2)
FROM (
  SELECT ST_SetRotation(rast, 15) AS rast1, rast as rast2 FROM dummy_rast
) AS foo;
```

st_scalex	st_scaley	st_skewx	st_skewy	↔
st_scalex	st_scaley	st_skewx	st_skewy	↔

```

-1.51937582571764 | -2.27906373857646 | 1.95086352047135 | 1.30057568031423 | ←
      2 |      3 |      0 |      0
-0.0379843956429411 | -0.0379843956429411 | 0.0325143920078558 | 0.0325143920078558 | ←
      0.05 |     -0.05 |      0 |      0

```

相关信息

[ST\\_Rotation](#), [ST\\_ScaleX](#), [ST\\_ScaleY](#), [ST\\_SkewX](#), [ST\\_SkewY](#)

### 11.7.3 ST\_SetScale

`ST_SetScale` — 以坐参考系位位置像素的 X 和 Y 大小。数字位/像素度/高度。

#### Synopsis

```

raster ST_SetScale(raster rast, float8 xy);
raster ST_SetScale(raster rast, float8 x, float8 y);

```

#### 描述

以坐参考系位位置像素的 X 和 Y 大小。数字位/像素度/高度。如果只入一个位，假 X 和 Y 是相同的数字。



#### Note

`ST_SetScale` 与 `ST_Rescale` 的不同之处在于 `ST_SetScale` 不会格重新采以匹配格范。它更改格的元数据（或地理参考）以正最初指定的放比例。`ST_Rescale` 会生成具有不同度和高度的格，算果以适合入格的地理范。`ST_SetScale` 不修改格的度或高度。

更改：2.0.0 在 `WKTRaster` 版本中，称 `ST_SetPixelSize`。在 2.0.0 中生了化。

#### 示例

```

UPDATE dummy_rast
  SET rast = ST_SetScale(rast, 1.5)
WHERE rid = 2;

SELECT ST_ScaleX(rast) As pixx, ST_ScaleY(rast) As pixy, Box3D(rast) As newbox
FROM dummy_rast
WHERE rid = 2;

```

```

pixx | pixy | newbox
-----+-----+-----
1.5 | 1.5 | BOX(3427927.75 5793244 0, 3427935.25 5793251.5 0)

```

```

UPDATE dummy_rast
  SET rast = ST_SetScale(rast, 1.5, 0.55)
WHERE rid = 2;

SELECT ST_ScaleX(rast) As pixx, ST_ScaleY(rast) As pixy, Box3D(rast) As newbox
FROM dummy_rast

```

```
WHERE rid = 2;
```

```

pixx | pixy |                               newbox
-----+-----+-----
 1.5 | 0.55 | BOX(3427927.75 5793244 0,3427935.25 5793247 0)

```

相关信息

[ST\\_ScaleX](#), [ST\\_ScaleY](#), [Box3D](#)

### 11.7.4 ST\_SetSkew

`ST_SetSkew` — 设置地理参考 X 和 Y 的斜率（或旋转变换参数）。如果只输入一个，则将 X 和 Y 设置为相同的值。

#### Synopsis

```
raster ST_SetSkew(raster rast, float8 skewxy);
raster ST_SetSkew(raster rast, float8 skewx, float8 skewy);
```

描述

设置地理参考 X 和 Y 的斜率（或旋转变换参数）。如果只输入一个，则将 X 和 Y 设置为相同的值。有关更多信息，参见[世界文件](#)。

示例

```
-- Example 1
UPDATE dummy_rast SET rast = ST_SetSkew(rast,1,2) WHERE rid = 1;
SELECT rid, ST_SkewX(rast) As skewx, ST_SkewY(rast) As skewy,
       ST_GeoReference(rast) as georef
FROM dummy_rast WHERE rid = 1;
```

```

rid | skewx | skewy |      georef
-----+-----+-----
  1 |      1 |      2 | 2.0000000000
      : 2.0000000000
      : 1.0000000000
      : 3.0000000000
      : 0.5000000000
      : 0.5000000000

```

```
-- Example 2 set both to same number:
UPDATE dummy_rast SET rast = ST_SetSkew(rast,0) WHERE rid = 1;
SELECT rid, ST_SkewX(rast) As skewx, ST_SkewY(rast) As skewy,
       ST_GeoReference(rast) as georef
FROM dummy_rast WHERE rid = 1;
```

```

rid | skewx | skewy |      georef
-----+-----+-----
  1 |      0 |      0 | 2.0000000000
      : 0.0000000000
      : 0.0000000000

```

```

: 3.0000000000
: 0.5000000000
: 0.5000000000

```

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);
```

描述

将栅格上的 SRID 设置成特定整数。



#### Note

此函数不会以任何方式更改栅格 - 它只是设置其当前所在坐标系参考系的空参考的元数据。这对于以后的操作很有用。

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);
```

描述

将栅格像素左上角的位置投影的 X 和 Y 坐标。

示例

```

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` — 重采一个栅格像，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);
```

#### 描述

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`)，以及一个栅格地理参考属性 (`scalex`、`scaley`、`skewx & skewy`)，这些属性可以自己定义，也可以从一个栅格像中借用。如果使用参考栅格像，那么每个栅格像必须具有相同的空参考 ID (SRID)。

使用以下重采样算法之一算新像素：

- 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，参看：[GDAL Warp 重采样方法](#)。

可用性：2.0.0 需要 GDAL 1.6.1+

增修：3.4.0 添加了最大和最小重采样

示例

```
SELECT
  ST_Width(orig) AS orig_width,
  ST_Width(reduce_100) AS new_width
FROM (
  SELECT
    rast AS orig,
    ST_Resample(rast,100,100) AS reduce_100
  FROM aerials.boston
  WHERE ST_Intersects(rast,
    ST_Transform(
      ST_MakeEnvelope(-71.128, 42.2392,-71.1277, 42.2397, 4326),26986)
    )
  )
LIMIT 1
) AS foo;
```

orig_width	new_width
200	100

相关信息

[ST\\_Rescale](#), [ST\\_Resize](#), [ST\\_Transform](#)

### 11.7.8 ST\_Rescale

**ST\_Rescale** — 通过指定整格的比例（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。

#### 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);

描述

通过指定整格的比例（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（英国或美式拼写）
- 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` 是重采样算法进行近似时的最大误差（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。如果置零，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` 会生成具有不同宽度和高度的栅格，计算结果以适合输入栅格的地理范围。`ST_SetScale` 不修改栅格的宽度和高度。

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为 0.0015 度的像素大小的示例。

```
-- the original raster pixel size
SELECT ST_PixelWidth(ST_AddBand(ST_MakeEmptyRaster(100, 100, 0, 0, 0.001, -0.001, 0, 0, 4269), '8BUI'::text, 1, 0)) width

width
-----
0.001

-- the rescaled raster raster pixel size
SELECT ST_PixelWidth(ST_Rescale(ST_AddBand(ST_MakeEmptyRaster(100, 100, 0, 0, 0.001, -0.001, 0, 0, 4269), '8BUI'::text, 1, 0), 0.0015)) width

width
-----
0.0015
```

#### 相关信息

[ST\\_Resize](#), [ST\\_Resample](#), [ST\\_SetScale](#), [ST\\_ScaleX](#), [ST\\_ScaleY](#), [ST\\_Transform](#)

### 11.7.9 ST\_Reskew

`ST_Reskew` — 通过调整栅格的斜率（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。默认是 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);

### 描述

通过调整栅格的斜率（或旋转参数）来重新采样栅格。新的像素是使用 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 重新采样方法](#)。



#### 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）以更正最初指定的斜率。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。

```
-- the original raster non-rotated
SELECT ST_Rotation(ST_AddBand(ST_MakeEmptyRaster(100, 100, 0, 0, 0.001, -0.001, 0, 0, 4269) ←
, '8BUI'::text, 1, 0));

-- result
0

-- the reskewed raster raster rotation
SELECT ST_Rotation(ST_Reskew(ST_AddBand(ST_MakeEmptyRaster(100, 100, 0, 0, 0.001, -0.001, ←
0, 0, 4269), '8BUI'::text, 1, 0), 0.0015));

-- result
-0.982793723247329
```

### 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 — 通常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);
```

### 描述

通常将栅格像重新采样，将其移动到由任意像素角点 (gridx & gridy) 定义的网格，可地使用像素大小 (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 Lanczos 重新采样算法算的。默认情况下使用最近邻 (NearestNeighbor)，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外部。

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。

```
-- the original raster upper left X
SELECT ST_UpperLeftX(ST_AddBand(ST_MakeEmptyRaster(10, 10, 0, 0, 0.001, -0.001, 0, 0, 4269) ←
, '8BUI'::text, 1, 0));
-- result
0
-- the upper left of raster after snapping
```

```
SELECT ST_UpperLeftX(ST_SnapToGrid(ST_AddBand(ST_MakeEmptyRaster(10, 10, 0, 0, 0.001, ←
    -0.001, 0, 0, 4269), '8BUI'::text, 1, 0), 0.0002, 0.0002));

-- result
-0.0008
```

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);
```

#### 描述

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 (英文或美式拼写)、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。

主体 1 期望输出栅格的宽度/高度。

主体 2 期望零 (0) 和一 (1) 之间的十进制数，表示输入栅格宽度/高度的百分比。

主体 3 可以接受输出栅格像的宽度/高度，也可以接受文本百分比（例如“20%”），表示输入栅格像宽度/高度的百分比。

可用性：2.1.0 需要 GDAL 1.6.1+

#### 示例

```
WITH foo AS(
SELECT
    1 AS rid,
    ST_Resize(
        ST_AddBand(
            ST_MakeEmptyRaster(1000, 1000, 0, 0, 1, -1, 0, 0, 0)
            , 1, '8BUI', 255, 0
        )
        , '50%', '500') AS rast
UNION ALL
SELECT
    2 AS rid,
```

```

    ST_Resize(
      ST_AddBand(
        ST_MakeEmptyRaster(1000, 1000, 0, 0, 1, -1, 0, 0, 0)
        , 1, '8BUI', 255, 0
      )
      , 500, 100) AS rast
UNION ALL
SELECT
  3 AS rid,
  ST_Resize(
    ST_AddBand(
      ST_MakeEmptyRaster(1000, 1000, 0, 0, 1, -1, 0, 0, 0)
      , 1, '8BUI', 255, 0
    )
    , 0.25, 0.9) AS rast
), bar AS (
  SELECT rid, ST_Metadata(rast) AS meta, rast FROM foo
)
SELECT rid, (meta).* FROM bar

```

rid	upperleftx	upperlefty	width	height	scalex	scaley	skewx	skewy	srid	←
1	0	0	500	500	1	-1	0	0	0	←
2	0	0	500	100	1	-1	0	0	0	←
3	0	0	250	900	1	-1	0	0	0	←

(3 rows)

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 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);

```

#### 描述

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，参见：[GDAL Warp 重采样方法](#)。

`ST_Transform` 常与 `ST_SetSRID()` 混淆。`ST_Transform` 上将栅格的坐标从一种空参考系更改为一种空参考系（并像像素行重新采样），而 `ST_SetSRID()` 只是更改栅格的 SRID 符号。

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

如果您支持无法正常工作，您可能需要将环境变量 `PROJSO` 置为 PostGIS 正在使用的 .so 或 .dll 投影。只需要文件名即可。例如，在 Windows 上，您可以在控制面板 -> 系统 -> 环境变量中添加一个名为 `PROJSO` 的变量并将其置为 `libproj.dll`（如果您使用的是 `proj 4.6.1`）。行此更改后，您必须重新 PostgreSQL 服务/守护程序。



#### Warning

在覆盖范围，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间隙，如示例所示：体 3。

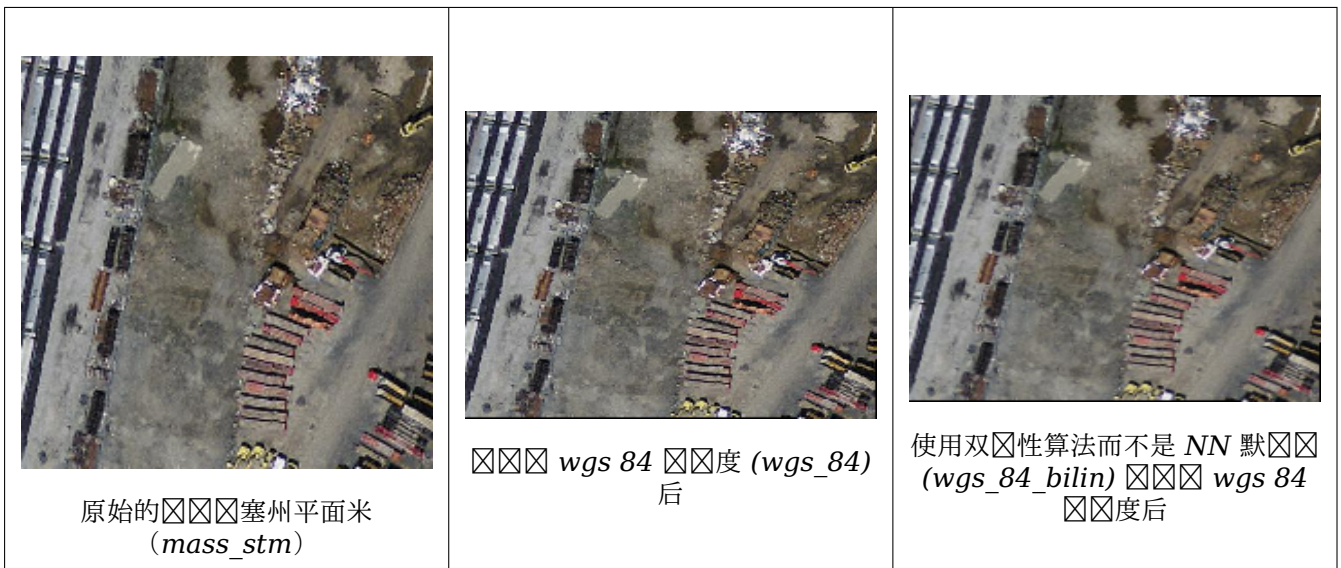
可用性：2.0.0 需要 GDAL 1.6.1+

增：2.1.0 添加 `ST_Transform(rast,alignto)` 体

示例

```
SELECT ST_Width(mass_stm) As w_before, ST_Width(wgs_84) As w_after,
       ST_Height(mass_stm) As h_before, ST_Height(wgs_84) As h_after
FROM
  ( SELECT rast As mass_stm, ST_Transform(rast,4326) As wgs_84
    , ST_Transform(rast,4326, 'Bilinear') AS wgs_84_bilin
    FROM aerials.o_2_boston
    WHERE ST_Intersects(rast,
                        ST_Transform(ST_MakeEnvelope(-71.128, 42.2392, -71.1277, 42.2397, 4326) ←
                        ,26986) )
    LIMIT 1) As foo;
```

w_before	w_after	h_before	h_after
200	228	200	170



### 示例：体 3

下面展示了使用 `ST_Transform(raster, srid)` 和 `ST_Transform(raster, alignto)` 之的区

```
WITH foo AS (
  SELECT 0 AS rid, ST_AddBand(ST_MakeEmptyRaster(2, 2, -500000, 600000, 100, -100, 0, 0, ←
    2163), 1, '16BUI', 1, 0) AS rast UNION ALL
  SELECT 1, ST_AddBand(ST_MakeEmptyRaster(2, 2, -499800, 600000, 100, -100, 0, 0, 2163), ←
    1, '16BUI', 2, 0) AS rast UNION ALL
  SELECT 2, ST_AddBand(ST_MakeEmptyRaster(2, 2, -499600, 600000, 100, -100, 0, 0, 2163), ←
    1, '16BUI', 3, 0) AS rast UNION ALL

  SELECT 3, ST_AddBand(ST_MakeEmptyRaster(2, 2, -500000, 599800, 100, -100, 0, 0, 2163), ←
    1, '16BUI', 10, 0) AS rast UNION ALL
  SELECT 4, ST_AddBand(ST_MakeEmptyRaster(2, 2, -499800, 599800, 100, -100, 0, 0, 2163), ←
    1, '16BUI', 20, 0) AS rast UNION ALL
  SELECT 5, ST_AddBand(ST_MakeEmptyRaster(2, 2, -499600, 599800, 100, -100, 0, 0, 2163), ←
    1, '16BUI', 30, 0) AS rast UNION ALL

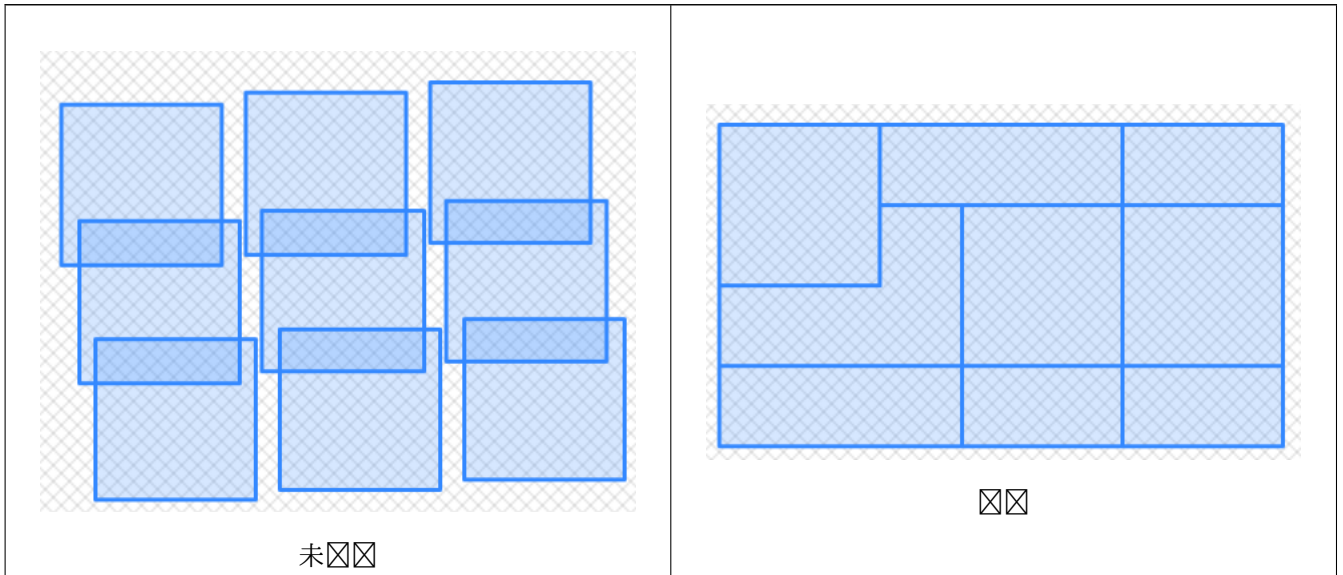
  SELECT 6, ST_AddBand(ST_MakeEmptyRaster(2, 2, -500000, 599600, 100, -100, 0, 0, 2163), ←
    1, '16BUI', 100, 0) AS rast UNION ALL
  SELECT 7, ST_AddBand(ST_MakeEmptyRaster(2, 2, -499800, 599600, 100, -100, 0, 0, 2163), ←
    1, '16BUI', 200, 0) AS rast UNION ALL
  SELECT 8, ST_AddBand(ST_MakeEmptyRaster(2, 2, -499600, 599600, 100, -100, 0, 0, 2163), ←
    1, '16BUI', 300, 0) AS rast
), bar AS (
  SELECT
    ST_Transform(rast, 4269) AS alignto
  FROM foo
  LIMIT 1
), baz AS (
  SELECT
    rid,
    rast,
    ST_Transform(rast, 4269) AS not_aligned,
    ST_Transform(rast, alignto) AS aligned
  FROM foo
  CROSS JOIN bar
)
SELECT
```

```

ST_SameAlignment(rast) AS rast,
ST_SameAlignment(not_aligned) AS not_aligned,
ST_SameAlignment(aligned) AS aligned
FROM baz

```

rast	not_aligned	aligned
t	f	t



相关信息

[ST\\_Transform](#), [ST\\_SetSRID](#)

## 11.8 栅格波段管理器

### 11.8.1 ST\_SetBandNoDataValue

`ST_SetBandNoDataValue` — 置代表无数据的指定波段的。如果未指定波段，假定波段 1。要将波段没有 nodata，置 nodata = NULL。

#### Synopsis

```

raster ST_SetBandNoDataValue(raster rast, double precision nodatavalue);
raster ST_SetBandNoDataValue(raster rast, integer band, double precision nodatavalue, boolean forcechecking=false);

```

#### 描述

置表示没有数据的。如果未指定，假定波段 1。将影响 `ST_Polygon`、`ST_DumpAsPolygons` 和 `ST_PixelAs...()` 函数的结果。

## 示例

```
-- change just first band no data value
UPDATE dummy_rast
  SET rast = ST_SetBandNoDataValue(rast,1, 254)
WHERE rid = 2;

-- change no data band value of bands 1,2,3
UPDATE dummy_rast
  SET rast =
    ST_SetBandNoDataValue(
      ST_SetBandNoDataValue(
        ST_SetBandNoDataValue(
          rast,1, 254)
        ,2,99),
      3,108)
  WHERE rid = 2;

-- wipe out the nodata value this will ensure all pixels are considered for all processing ↔
functions
UPDATE dummy_rast
  SET rast = ST_SetBandNoDataValue(rast,1, NULL)
WHERE rid = 2;
```

## 相关信息

[ST\\_BandNoDataValue](#), [ST\\_NumBands](#)

### 11.8.2 ST\_SetBandIsNoData

`ST_SetBandIsNoData` — 将栅格的 `isnodata` 标志置为 `TRUE`。

#### Synopsis

```
raster ST_SetBandIsNoData(raster rast, integer band=1);
```

#### 描述

将栅格的 `isnodata` 标志置为 `true`。如果未指定，将假定第 1 个。只有在被栅格是栅格标志时才用此函数。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

```
-- Create dummy table with one raster column
create table dummy_rast (rid integer, rast raster);

-- Add raster with two bands, one pixel/band. In the first band, nodatavalue = pixel value ↔
= 3.
-- In the second band, nodatavalue = 13, pixel value = 4
insert into dummy_rast values(1,
(
```



```

'01' -- little endian (uint8 ndr)
||
'0000' -- version (uint16 0)
||
'0200' -- nBands (uint16 0)
||
'17263529ED684A3F' -- scaleX (float64 0.000805965234044584)
||
'F9253529ED684ABF' -- scaleY (float64 -0.00080596523404458)
||
'1C9F33CE69E352C0' -- ipX (float64 -75.5533328537098)
||
'718F0E9A27A44840' -- ipY (float64 49.2824585505576)
||
'ED50EB853EC32B3F' -- skewX (float64 0.000211812383858707)
||
'7550EB853EC32B3F' -- skewY (float64 0.000211812383858704)
||
'E6100000' -- SRID (int32 4326)
||
'0100' -- width (uint16 1)
||
'0100' -- height (uint16 1)
||
'4' -- hasnodatavalue set to true, isnodata value set to false (when it should be true)
||
'2' -- first band type (4BUI)
||
'03' -- novalue==3
||
'03' -- pixel(0,0)==3 (same that nodata)
||
'0' -- hasnodatavalue set to false
||
'5' -- second band type (16BSI)
||
'0D00' -- novalue==13
||
'0400' -- pixel(0,0)==4
)::raster
);

select st_bandisnodata(rast, 1) from dummy_rast where rid = 1; -- Expected false
select st_bandisnodata(rast, 1, TRUE) from dummy_rast where rid = 1; -- Expected true

-- The isnodata flag is dirty. We are going to set it to true
update dummy_rast set rast = st_setbandisnodata(rast, 1) where rid = 1;

select st_bandisnodata(rast, 1) from dummy_rast where rid = 1; -- Expected true

```

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)

### 描述

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、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

```
WITH foo AS (
  SELECT
    ST_AddBand(NULL::raster, '/home/pele/devel/geo/postgis-git/raster/test/regress/
      loader/Projected.tif', NULL::int[]) AS rast
)
SELECT
  1 AS query,
  *
FROM ST_BandMetadata(
  (SELECT rast FROM foo),
  ARRAY[1,3,2]::int[]
)
UNION ALL
SELECT
  2,
  *
FROM ST_BandMetadata(
  (
    SELECT
      ST_SetBandPath(
        rast,
        2,
        '/home/pele/devel/geo/postgis-git/raster/test/regress/loader/Projected2.tif
          ',
        1
      ) AS rast
    FROM foo
  ),
  ARRAY[1,3,2]::int[]
)
ORDER BY 1, 2;
```

query	bandnum	pixeltype	nodatavalue	isoutdb	path	outdbbandnum
1	1	8BUI		t	/home/pele/devel/geo/postgis-git/ raster/test/regress/loader/Projected.tif	1

```

1 |      2 | 8BUI | | t | /home/pele/devel/geo/postgis-git/ ↵
  raster/test/regress/loader/Projected.tif | | 2
1 |      3 | 8BUI | | t | /home/pele/devel/geo/postgis-git/ ↵
  raster/test/regress/loader/Projected.tif | | 3
2 |      1 | 8BUI | | t | /home/pele/devel/geo/postgis-git/ ↵
  raster/test/regress/loader/Projected.tif | | 1
2 |      2 | 8BUI | | t | /home/pele/devel/geo/postgis-git/ ↵
raster/test/regress/loader/Projected2.tif | | 1
2 |      3 | 8BUI | | t | /home/pele/devel/geo/postgis-git/ ↵
  raster/test/regress/loader/Projected.tif | | 3

```

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);

### 描述

更新 out-db band 的外部 band 号。不会触及与 out-db band 相关的外部栅格文件



#### Note

如果 force 置 true，不会进行任何操作来确保外部栅格文件和 PostGIS 栅格之兼容性（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

```

WITH foo AS (
  SELECT
    ST_AddBand(NULL::raster, '/home/pele/devel/geo/postgis-git/raster/test/regress/ ↵
      loader/Projected.tif', NULL::int[]) AS rast
)
SELECT
  1 AS query,
  *
FROM ST_BandMetadata(

```

```
(SELECT rast FROM foo),
ARRAY[1,3,2]::int[]
)
UNION ALL
SELECT
  2,
  *
FROM ST_BandMetadata(
  (
    SELECT
      ST_SetBandIndex(
        rast,
        2,
        1
      ) AS rast
    FROM foo
  ),
  ARRAY[1,3,2]::int[]
)
ORDER BY 1, 2;
```

query	bandnum	pixeltype	nodatavalue	isoutdb	path
	outdbbandnum				
1	1	8BUI		t	/home/pele/devel/geo/postgis-git/ raster/test/regress/loader/Projected.tif   1
1	2	8BUI		t	/home/pele/devel/geo/postgis-git/ raster/test/regress/loader/Projected.tif   2
1	3	8BUI		t	/home/pele/devel/geo/postgis-git/ raster/test/regress/loader/Projected.tif   3
2	1	8BUI		t	/home/pele/devel/geo/postgis-git/ raster/test/regress/loader/Projected.tif   1
2	2	8BUI		t	/home/pele/devel/geo/postgis-git/ raster/test/regress/loader/Projected.tif   1
2	3	8BUI		t	/home/pele/devel/geo/postgis-git/ raster/test/regress/loader/Projected.tif   3

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 置为 true，则不算不等于 nodata 值的像素。

#### Synopsis

```
bigint ST_Count(raster rast, integer nband=1, boolean exclude_nodata_value=true);
bigint ST_Count(raster rast, boolean exclude_nodata_value);
```

## 描述

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, nband 默认为 1。



### Note

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

更改: 3.1.0 - 除了 `ST_Count(rastertable, rastercolumn, ...)` 函数体。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. --
SELECT rid, ST_Count(ST_SetBandNoDataValue(rast,249)) As exclude_nodata,
       ST_Count(ST_SetBandNoDataValue(rast,249),false) As include_nodata
FROM dummy_rast WHERE rid=2;
```

rid	exclude_nodata	include_nodata
2	23	25

## 相关信息

[ST\\_CountAgg](#), [ST\\_SummaryStats](#), [ST\\_SetBandNoDataValue](#)

## 11.9.2 ST\_CountAgg

`ST_CountAgg` — 聚合函数。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);
```

## 描述

返回一栅格的指定波段中的像素数。如果未指定 band, 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

默认情况下将栅格所有像素并行采样。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

```

WITH foo AS (
  SELECT
    rast.rast
  FROM (
    SELECT ST_SetValue(
      ST_SetValue(
        ST_SetValue(
          ST_AddBand(
            ST_MakeEmptyRaster(10, 10, 10, 10, 2, 2, 0, 0,0)
            , 1, '64BF', 0, 0
          )
          , 1, 1, 1, -10
        )
        , 1, 5, 4, 0
      )
      , 1, 5, 5, 3.14159
    ) AS rast
  ) AS rast
  FULL JOIN (
    SELECT generate_series(1, 10) AS id
  ) AS id
  ON 1 = 1
)
SELECT
  ST_CountAgg(rast, 1, TRUE)
FROM foo;

 st_countagg
-----
                20
(1 row)

```

相关信息

[ST\\_Count](#), [ST\\_SummaryStats](#), [ST\\_SetBandNoDataValue](#)

### 11.9.3 ST\_Histogram

**ST\_Histogram** — 返回一个表，显示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。

#### 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);

```

描述

返回每个 bin 定义栅格波段的集合，其中包含最小值、最大值、计数、百分比。如果未指定 band，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 : 一个数字，指示每个 category/bin 的宽度。如果箱的数量大于宽度的数量，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箱数（如果指定）。如果未指定，来自计算分箱数量。

**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)` 体。

可用性：2.0.0

示例：每个栅格 - 计算波段 **1、2、3** 和自计算 **bins** 的直方图

```
SELECT band, (stats).*
FROM (SELECT rid, band, ST_Histogram(rast, band) As stats
      FROM dummy_rast CROSS JOIN generate_series(1,3) As band
      WHERE rid=2) As foo;
```

band	min	max	count	percent
1	249	250	2	0.08
1	250	251	2	0.08
1	251	252	1	0.04
1	252	253	2	0.08
1	253	254	18	0.72
2	78	113.2	11	0.44
2	113.2	148.4	4	0.16
2	148.4	183.6	4	0.16
2	183.6	218.8	1	0.04
2	218.8	254	5	0.2
3	62	100.4	11	0.44
3	100.4	138.8	5	0.2
3	138.8	177.2	4	0.16
3	177.2	215.6	1	0.04
3	215.6	254	4	0.16

示例：只有波段 **2**，但有 **6** 个 **bins**

```
SELECT (stats).*
FROM (SELECT rid, ST_Histogram(rast, 2,6) As stats
      FROM dummy_rast
      WHERE rid=2) As foo;
```

min	max	count	percent
-----+	-----+	-----+	-----+

```

      78 | 107.333333 | 9 | 0.36
107.333333 | 136.666667 | 6 | 0.24
136.666667 | 166 | 0 | 0
      166 | 195.333333 | 4 | 0.16
195.333333 | 224.666667 | 1 | 0.04
224.666667 | 254 | 5 | 0.2
(6 rows)

```

```

-- Same as previous but we explicitly control the pixel value range of each bin.
SELECT (stats).*
FROM (SELECT rid, ST_Histogram(rast, 2,6,ARRAY[0.5,1,4,100,5]) As stats
      FROM dummy_rast
      WHERE rid=2) As foo;

```

min	max	count	percent
78	78.5	1	0.08
78.5	79.5	1	0.04
79.5	83.5	0	0
83.5	183.5	17	0.0068
183.5	188.5	0	0
188.5	254	6	0.003664

(6 rows)

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);

```

### 描述

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。



**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

```
UPDATE dummy_rast SET rast = ST_SetBandNoDataValue(rast,249) WHERE rid=2;
--Example will consider only pixels of band 1 that are not 249 and in named quantiles --
```

```
SELECT (pvq).*
FROM (SELECT ST_Quantile(rast, ARRAY[0.25,0.75]) As pvq
      FROM dummy_rast WHERE rid=2) As foo
ORDER BY (pvq).quantile;
```

quantile	value
0.25	253
0.75	254

```
SELECT ST_Quantile(rast, 0.75) As value
FROM dummy_rast WHERE rid=2;
```

value
254

```
--real live example. Quantile of all pixels in band 2 intersecting a geometry
SELECT rid, (ST_Quantile(rast,2)).* As pvc
FROM o_4_boston
WHERE ST_Intersects(rast,
  ST_GeomFromText('POLYGON((224486 892151,224486 892200,224706 892200,224706
    892151,224486 892151))',26986)
)
ORDER BY value, quantile,rid
;
```

rid	quantile	value
1	0	0
2	0	0
14	0	1
15	0	2
14	0.25	37
1	0.25	42
15	0.25	47
2	0.25	50
14	0.5	56
1	0.5	64
15	0.5	66
2	0.5	77
14	0.75	81
15	0.75	87
1	0.75	94

2	0.75	106
14	1	199
1	1	244
2	1	255
15	1	255

相关信息

[ST\\_Count](#), [ST\\_SummaryStats](#), [ST\\_SummaryStatsAgg](#), [ST\\_SetBandNoDataValue](#)

### 11.9.5 ST\_SummaryStats

`ST_SummaryStats` — 返回一列统计信息，包括指定栅格或栅格覆盖的栅格的像素数、和、均值、标准差、最小值和最大值。如果未指定 band 号，则假定 band 为 1。

#### Synopsis

```
summarystats ST_SummaryStats(raster rast, boolean exclude_nodata_value);
summarystats ST_SummaryStats(raster rast, integer nband, boolean exclude_nodata_value);
```

#### 描述

返回栅格或栅格覆盖范围的指定栅格波段的 `summarystats` 信息，其中包含像素数、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，考虑不等于 nodata 值的像素。将 `except_nodata_value` 置为 false 以获取所有像素的像素数。



#### Note

默认情况下将遍历所有像素。为了获得更快的速度，可将 `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：每个栅格

```
SELECT rid, band, (stats).*
FROM (SELECT rid, band, ST_SummaryStats(rast, band) As stats
      FROM dummy_rast CROSS JOIN generate_series(1,3) As band
      WHERE rid=2) As foo;
```

rid	band	count	sum	mean	stddev	min	max
2	1	23	5821	253.086957	1.248061	250	254
2	2	25	3682	147.28	59.862188	78	254
2	3	25	3290	131.6	61.647384	62	254

示例：☒☒与感☒趣的建筑物相交的像素

此示例在 64 位 PostGIS 窗口上花费了 574 毫秒，包含所有波士顿☒建筑物和空中☒☒（每个☒☒ 150x150 像素 ~ 134,000 个☒☒），☒ 102,000 个建筑☒☒

```
WITH
-- our features of interest
  feat AS (SELECT gid AS building_id, geom_26986 AS geom FROM buildings AS b
    WHERE gid IN(100, 103,150)
  ),
-- clip band 2 of raster tiles to boundaries of builds
-- then get stats for these clipped regions
  b_stats AS
    (SELECT building_id, (stats).*)
FROM (SELECT building_id, ST_SummaryStats(ST_Clip(rast,2,geom)) As stats
  FROM aerials.boston
    INNER JOIN feat
      ON ST_Intersects(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;
```

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

示例：☒格覆盖范☒

```
-- stats for each band --
SELECT band, (stats).*
FROM (SELECT band, ST_SummaryStats('o_4_boston','rast', band) As stats
  FROM generate_series(1,3) As band) As foo;
```

band	count	sum	mean	stddev	min	max
1	8450000	725799	82.7064349112426	45.6800222638537	0	255
2	8450000	700487	81.4197705325444	44.2161184161765	0	255
3	8450000	575943	74.682739408284	44.2143885481407	0	255

```
-- For a table -- will get better speed if set sampling to less than 100%
-- Here we set to 25% and get a much faster answer
SELECT band, (stats).*
FROM (SELECT band, ST_SummaryStats('o_4_boston','rast', band,true,0.25) As stats
  FROM generate_series(1,3) As band) As foo;
```

band	count	sum	mean	stddev	min	max
1	2112500	180686	82.6890480473373	45.6961043857248	0	255
2	2112500	174571	81.448503668639	44.2252623171821	0	255
3	2112500	144364	74.6765884023669	44.2014869384578	0	255

相关信息

[summarystats](#), [ST\\_SummaryStatsAgg](#), [ST\\_Count](#), [ST\\_Clip](#)

### 11.9.6 ST\_SummaryStatsAgg

`ST_SummaryStatsAgg` — 栅格的。返回一栅格的固定栅格波段的摘要信息，其中包含数、和、平均、方差、最小、最大。如果未指定号，假定 1。

#### 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);
```

#### 描述

返回栅格或栅格覆盖范围的固定栅格波段的 `summarystats` 信息，其中包含数、和、平均、方差、最小、最大。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

默认情况下将行所有像素行采样。为了更快的，将 `sample_percent` 置为 0 到 1 之间的。

可用性：2.2.0

#### 示例

```
WITH foo AS (
  SELECT
    rast.rast
  FROM (
    SELECT ST_SetValue(
      ST_SetValue(
        ST_SetValue(
          ST_AddBand(
            ST_MakeEmptyRaster(10, 10, 10, 10, 2, 2, 0, 0, 0)
            , 1, '64BF', 0, 0
          )
          , 1, 1, 1, -10
        )
        , 1, 5, 4, 0
      )
    )
  )
```

```

        , 1, 5, 5, 3.14159
    ) AS rast
) AS rast
FULL JOIN (
    SELECT generate_series(1, 10) AS id
) AS id
    ON 1 = 1
)
SELECT
    (stats).count,
    round((stats).sum::numeric, 3),
    round((stats).mean::numeric, 3),
    round((stats).stddev::numeric, 3),
    round((stats).min::numeric, 3),
    round((stats).max::numeric, 3)
FROM (
    SELECT
        ST_SummaryStatsAgg(rast, 1, TRUE, 1) AS stats
    FROM foo
) bar;

 count | round | round | round | round | round
-----+-----+-----+-----+-----+-----
      20 | -68.584 | -3.429 | 6.571 | -10.000 | 3.142
(1 row)

```

相关信息

[summarystats](#), [ST\\_SummaryStats](#), [ST\\_Count](#), [ST\\_Clip](#)

### 11.9.7 ST\_ValueCount

**ST\_ValueCount** — 返回一 `table`，其中包含像素 `value` 以及具有 `band` 集的 `geom`（或 `geom` 覆盖范围）`geom` 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 `value`，并将像素 `value` 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 ex-
clude_nodata_value=true, double precision[] searchvalues=NULL, double precision roundto=0, dou-
ble precision OUT value, integer OUT count);
SETOF record ST_ValueCount(text rastertable, text rastercolumn, double precision[] searchvalues,
double precision roundto=0, double precision OUT value, integer OUT count);

```

SETOF record **ST\_ValueCount**(text rastertable, text rastercolumn, integer nband, double precision[] searchvalues, double precision roundto=0, double precision OUT value, integer OUT count);  
 bigint **ST\_ValueCount**(text rastertable, text rastercolumn, integer nband, boolean exclude\_nodata\_value, double precision searchvalue, double precision roundto=0);  
 bigint **ST\_ValueCount**(text rastertable, text rastercolumn, double precision searchvalue, double precision roundto=0);  
 bigint **ST\_ValueCount**(text rastertable, text rastercolumn, integer nband, double precision searchvalue, double precision roundto=0);

### 描述

返回一具有列数的数组，其中包含像素波段以及所波段的格切片或格覆盖范中的像素数。

如果未指定波段，`nband` 默认为 1。如果未指定搜索值，将返回在格或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的像素数的数组。



### Note

如果 `except_nodata_value` 置为 `false`，将没有数据的像素行。

可用性: 2.0.0

### 示例

```
UPDATE dummy_rast SET rast = ST_SetBandNoDataValue(rast,249) WHERE rid=2;
--Example will count only pixels of band 1 that are not 249. --
```

```
SELECT (pvc).*
FROM (SELECT ST_ValueCount(rast) As pvc
      FROM dummy_rast WHERE rid=2) As foo
ORDER BY (pvc).value;
```

value	count
250	2
251	1
252	2
253	6
254	12

```
-- Example will count all pixels of band 1 including 249 --
```

```
SELECT (pvc).*
FROM (SELECT ST_ValueCount(rast,1,false) As pvc
      FROM dummy_rast WHERE rid=2) As foo
ORDER BY (pvc).value;
```

value	count
249	2
250	2
251	1
252	2
253	6
254	12

```
-- Example will count only non-nodata value pixels of band 2
```

```
SELECT (pvc).*
FROM (SELECT ST_ValueCount(rast,2) As pvc
      FROM dummy_rast WHERE rid=2) As foo
ORDER BY (pvc).value;
```

value	count
78	1
79	1
88	1
89	1
96	1
97	1
98	1
99	2
112	2

```
:
```

```
--real live example. Count all the pixels in an aerial raster tile band 2 intersecting a ←
geometry
```

```
-- and return only the pixel band values that have a count > 500
```

```
SELECT (pvc).value, SUM((pvc).count) As total
FROM (SELECT ST_ValueCount(rast,2) As pvc
      FROM o_4_boston
      WHERE ST_Intersects(rast,
        ST_GeomFromText('POLYGON((224486 892151,224486 892200,224706 892200,224706 ←
          892151,224486 892151))',26986)
      )
      ) As foo
GROUP BY (pvc).value
HAVING SUM((pvc).count) > 500
ORDER BY (pvc).value;
```

value	total
51	502
54	521

```
-- Just return count of pixels in each raster tile that have value of 100 of tiles that ←
intersect a specific geometry --
```

```
SELECT rid, ST_ValueCount(rast,2,100) As count
FROM o_4_boston
WHERE ST_Intersects(rast,
  ST_GeomFromText('POLYGON((224486 892151,224486 892200,224706 892200,224706 ←
    892151,224486 892151))',26986)
  ) ;
```

rid	count
1	56
2	95
14	37
15	64

相关信息

[ST\\_Count](#), [ST\\_SetBandNoDataValue](#)









### 11.11.3 ST\_AsGDALRaster

`ST_AsGDALRaster` — 以指定的 GDAL 格式返回栅格。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。

#### Synopsis

```
bytea ST_AsGDALRaster(raster rast, text format, text[] options=NULL, integer srid=sameasource);
```

#### 描述

返回指定格式的栅格。参数逐列如下：

- `format` 格式来输出。取决于 `libgdal` 中安装的程序。通常可用的有 “JPEG”、“GTiff”、“PNG”。使用 `ST_GDALDrivers` 获取您的支持的格式列表。
- `options` GDAL 的文本数组。有效选项取决于格式。有关更多信息，参见 [GDAL 格式格式](#)。
- `srs` 要嵌入像中的 `proj4text` 或 `srttext` (来自 `Spatial_ref_sys`)

可用性：2.0.0 - 需要 GDAL >= 1.6.0。

#### JPEG 输出示例，多个栅格作一个光栅

```
SELECT ST_AsGDALRaster(ST_Union(rast), 'JPEG', ARRAY['QUALITY=50']) As rastjpg
FROM dummy_rast
WHERE rast && ST_MakeEnvelope(10, 10, 11, 11);
```

#### 使用 PostgreSQL 大对象支持输出格式

将栅格输出为一种格式的一种方法是使用 [PostgreSQL 大对象输出函数](#)。我将重复前面的示例，但也会输出。注意，您需要有数据表的超引用限制，因为它使用服务器端 `lo` 函数。它将输出到服务器网络上的路径。如果需要本地输出，使用 `psql` 等效的 `lo_` 函数，函数输出到本地文件系统而不是服务器文件系统。

```
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;
```

**GTIFF** 出示例

```
SELECT ST_AsGDALRaster(rast, 'GTiff') As rastjpg
FROM dummy_rast WHERE rid=2;

-- Out GeoTiff with jpeg compression, 90% quality
SELECT ST_AsGDALRaster(rast, 'GTiff',
  ARRAY['COMPRESS=JPEG', 'JPEG_QUALITY=90'],
  4269) As rasttiff
FROM dummy_rast WHERE rid=2;
```

## 相关信息

Section [10.3](#), [ST\\_GDALDrivers](#), [ST\\_SRID](#)

**11.11.4 ST\_AsJPEG**

**ST\_AsJPEG** — 将栅格的指定波段作成一个合影输出 (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。

**Synopsis**

```
bytea ST_AsJPEG(raster rast, text[] options=NULL);
bytea ST_AsJPEG(raster rast, integer nband, integer quality);
bytea ST_AsJPEG(raster rast, integer nband, text[] options=NULL);
bytea ST_AsJPEG(raster rast, integer[] nbands, text[] options=NULL);
bytea ST_AsJPEG(raster rast, integer[] nbands, integer quality);
```

## 描述

将栅格的指定波段作成一个合影输出图像 (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。功能有多体和多波段。以下逐列出：

- **nband** 用于波段输出。
- **nbands** 是要输出的波段数 (注意, JPEG 的最大是 3), 波段的顺序 RGB。例如 `ARRAY[3,2,1]` 表示将波段 3 映射到蓝色, 将波段 2 映射到绿色, 将波段 1 映射到红色
- **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 first 3 bands 75% quality
SELECT ST_AsJPEG(rast) As rastjpg
       FROM dummy_rast WHERE rid=2;

-- output only first band as 90% quality
SELECT ST_AsJPEG(rast,1,90) As rastjpg
       FROM dummy_rast WHERE rid=2;

-- output first 3 bands (but make band 2 Red, band 1 green, and band 3 blue, progressive ←
   and 90% quality
SELECT ST_AsJPEG(rast,ARRAY[2,1,3],ARRAY['QUALITY=90','PROGRESSIVE=ON']) As rastjpg
       FROM dummy_rast WHERE rid=2;
```

相关信息

Section [10.3](#), [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。

#### Synopsis

```
bytea ST_AsPNG(raster rast, text[] options=NULL);
bytea ST_AsPNG(raster rast, integer nband, integer compression);
bytea ST_AsPNG(raster rast, integer nband, text[] options=NULL);
bytea ST_AsPNG(raster rast, integer[] nbands, integer compression);
bytea ST_AsPNG(raster rast, integer[] nbands, text[] options=NULL);
```

#### 描述

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。功能有多样性和多。如果未指定 `srid`，使用栅格的 `srid`。以下逐列出：

- `nband` 用于波段输出。
- `nbands` 是要输出的波段数 (注意，PNG 的最大 4)，波段的顺序 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` 号从 1 到 9。数字越大，越大。
- `options text` PNG 定义的 GDAL 数 (看 [ST\\_GDALDrivers](#) 的 PNG 的 `create_options`)。对于 PNG 来，有效的只是 `ZLEVEL` (所花的——默认 6)，例如数 `['ZLEVEL=9']`。不允许 `WORLDFILE`，因为函数必须输出一个。有关更多信息，参 [GDAL 格式](#)。

可用性：2.0.0 - 需要 GDAL >= 1.6.0。

示例

```
SELECT ST_AsPNG(rast) As rastpng
FROM dummy_rast WHERE rid=2;

-- export the first 3 bands and map band 3 to Red, band 1 to Green, band 2 to blue
SELECT ST_AsPNG(rast, ARRAY[3,1,2]) As rastpng
FROM dummy_rast WHERE rid=2;
```

相关信息

[ST\\_AsGDALRaster](#), [ST\\_ColorMap](#), [ST\\_GDALDrivers](#), [Section 10.3](#)

### 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，则将使用所有波段。

#### 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);
```

描述

以多个图像文件格式 (TIFF) 返回栅格的指定波段。如果未指定波段，则将使用所有波段。是 [ST\\_AsGDALRaster](#) 的包装。如果需要出不太常见的栅格类型，使用 [ST\\_AsGDALRaster](#)。功能有多样性和多。如果不存在空参考 SRS 文本，使用栅格的空参考。以下逐列出：

- **nbands** 是要出的波段数（注意，PNG 的最大 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** 表式——JPEG90（或其他百分比）、LZW、JPEG、DEFLATE9。
- **options text** 是 GTiff 定义的 GDAL 选项数（看看 [ST\\_GDALDrivers](#) 的 GTiff 的 create\_options）。或参 [GDAL 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 90%**

```
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 栅格处理：代数

### 11.12.1 ST\_Clip

**ST\_Clip** — Returns the raster clipped by the input geometry. If band number is not specified, all bands are processed. If `crop` is not specified or `TRUE`, the output raster is cropped. If `touched` is set to `TRUE`, then touched pixels are included, otherwise only if the center of the pixel is in the geometry it is included.

#### Synopsis

```
raster ST_Clip(raster rast, integer[] nband, geometry geom, double precision[] nodataval=NULL,
boolean crop=TRUE, boolean touched=FALSE);
raster ST_Clip(raster rast, integer nband, geometry geom, double precision nodataval, boolean crop=TRUE,
boolean touched=FALSE);
raster ST_Clip(raster rast, integer nband, geometry geom, boolean crop, boolean touched=FALSE);
raster ST_Clip(raster rast, geometry geom, double precision[] nodataval=NULL, boolean crop=TRUE,
boolean touched=FALSE);
raster ST_Clip(raster rast, geometry geom, double precision nodataval, boolean crop=TRUE, boolean
touched=FALSE);
raster ST_Clip(raster rast, geometry geom, boolean crop, boolean touched=FALSE);
```

#### 描述

返回由输入几何体 `geom` 裁剪的栅格。如果未指定波段索引，处理所有波段。

`ST_Clip` 生成的栅格必对剪切区域分配一个点数据，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`，生成的栅格的 `nodata` 将置为 `ST_MinPossibleValue(ST_BandPixelType(rast, band))`。当数中的 `nodata` 的数量小于波段的数量，数中的最后一个用于剩余的波段。如果点数据的数量大于波段数量，忽略多余的点数据。所有接受点数据的体也接受将分配每个波段的个。

如果未指定 `crop`，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` 置为 `false`，新的栅格将具有与 `rast` 相同的范围。如果将 `touched` 置为 `true`，将与几何形相交的 `rast` 中的所有像素。



#### Note

默认行是 `touched=false`，将像素中心被几何形覆盖的像素。

增版：3.5.0 - 添加了 `touched` 参数。

可用性：2.0.0

增：2.1.0 用 C 重写

里的示例使用塞州航拍数据，数据可在 MassGIS 网站的 [MassGIS Aerial Orthos](#) 面上找到。

示例：比所有相交的和不是所有相交的情况

```
SELECT ST_Count(rast) AS count_pixels_in_orig, ST_Count(rast_touched) AS all_touched_pixels ←
, ST_Count(rast_not_touched) AS default_clip
FROM ST_AsRaster(ST_Letters('R'), scalex =
> 1.0, scaley =
> -1.0) AS r(rast)
```

```
INNER JOIN ST_GeomFromText('LINESTRING(0 1, 5 6, 10 10)') AS g(geom)
ON ST_Intersects(r.rast,g.geom)
, ST_Clip(r.rast, g.geom, touched =
> true) AS rast_touched
, ST_Clip(r.rast, g.geom, touched =
> false) AS rast_not_touched;
```

count_pixels_in_orig	all_touched_pixels	default_clip
2605	16	10

(1 row)

示例：1 波段剪裁（不☒☒所有相交的像素）

```
-- Clip the first band of an aerial tile by a 20 meter buffer.
SELECT ST_Clip(rast, 1,
      ST_Buffer(ST_Centroid(ST_Envelope(rast)),20)
) from aerials.boston
WHERE rid = 4;
```

```
-- Demonstrate effect of crop on final dimensions of raster
-- Note how final extent is clipped to that of the geometry
-- if crop = true
SELECT ST_XMax(ST_Envelope(ST_Clip(rast, 1, clipper, true))) As xmax_w_trim,
      ST_XMax(clipper) As xmax_clipper,
      ST_XMax(ST_Envelope(ST_Clip(rast, 1, clipper, false))) As xmax_wo_trim,
      ST_XMax(ST_Envelope(rast)) As xmax_rast_orig
FROM (SELECT rast, ST_Buffer(ST_Centroid(ST_Envelope(rast)),6) As clipper
FROM aerials.boston
WHERE rid = 6) As foo;
```

xmax_w_trim	xmax_clipper	xmax_wo_trim	xmax_rast_orig
230657.436173996	230657.436173996	230666.436173996	230666.436173996



裁剪前的完整☒格☒☒



裁剪后



示例：裁剪未裁剪的波段，并将其他波段不添加回去

```
-- Same example as before, but we need to set crop to false to be able to use ST_AddBand
-- because ST_AddBand requires all bands be the same Width and height
SELECT ST_AddBand(ST_Clip(rast, 1,
    ST_Buffer(ST_Centroid(ST_Envelope(rast)),20),false
), ARRAY[ST_Band(rast,2),ST_Band(rast,3)] ) from aerials.boston
WHERE rid = 6;
```



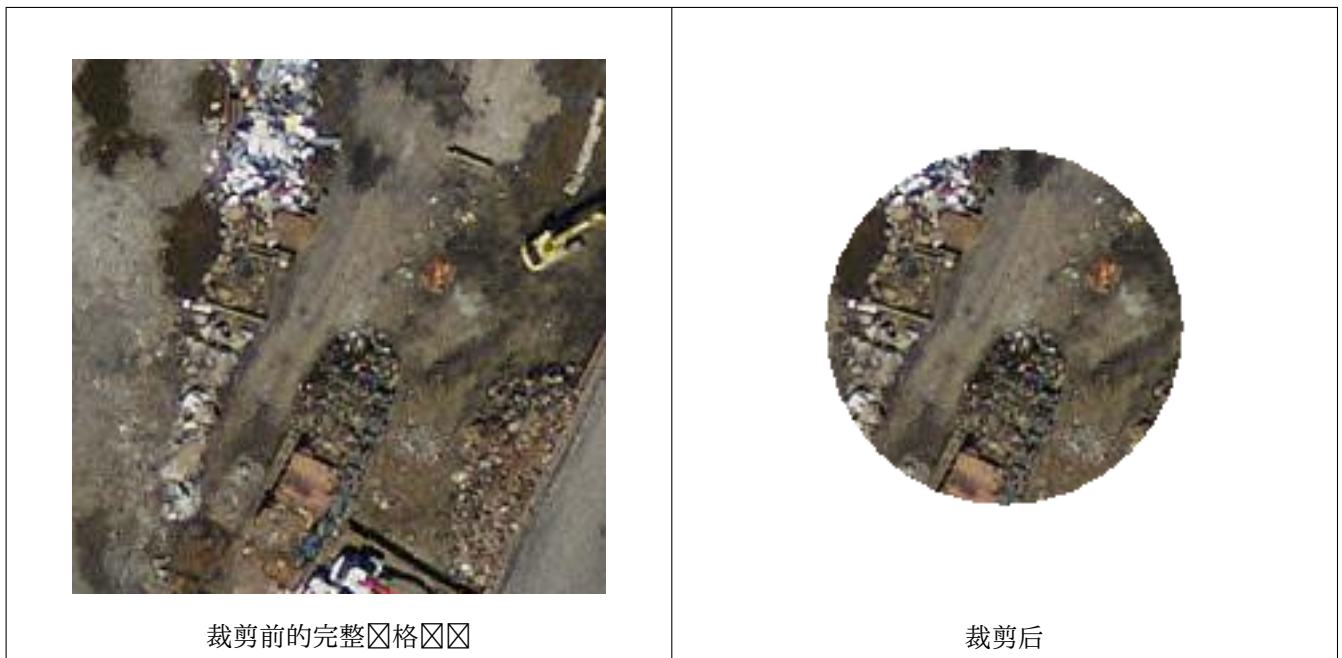
裁剪前的完整图像



剪裁后——超图像

示例：裁剪所有波段

```
-- Clip all bands of an aerial tile by a 20 meter buffer.
-- Only difference is we don't specify a specific band to clip
-- so all bands are clipped
SELECT ST_Clip(rast,
    ST_Buffer(ST_Centroid(ST_Envelope(rast)), 20),
    false
) from aerials.boston
WHERE rid = 4;
```



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构建最多四个 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);
```

#### 描述

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和色分量的一串。

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，色从色到色再到色。
- `fire` 四 8BUI (RGBA) 波段栅格，色从黑色到色到浅黄色。
- `bluered` 表示四 8BUI (RGBA) 波段栅格，色从色到浅白色再到色。

用 `colormap` 可以通过 `colormap` 参数指定一系列条目（每行一个）来指定自定义的色映射。每个条目通常包括五个值：像素 `x` 以及相 `x` 的 `r` 色、`g` 色、`b` 色、`Alpha` 组件（`r` 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 `x`，其中 0% 和 100% 分别表示栅格中找到的最小 `x` 和最大 `x`。`colormap` 可以使用逗号（','）、制表符、冒号（':'）和/或空格分隔。像素 `x` 可以置 `nv`、`null` 或 `nodata`，表示 NODATA 值。以下是一个示例。

```
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色
- `EXACT` 栅格匹配色值中找到的像素值。与色值条目不匹配的像素将被置 0 0 0 0 (RGBA)
- `NEAREST` 使用其最接近像素的色值条目



#### Note

[ColorBrewer](#) 是色彩的一个很好的参考。



#### Warning

新栅格的果波段将没有置 NODATA 值。如果需要，使用 `ST_SetBandNoDataValue` 置 NODATA 值。

可用性：2.1.0

示例

是一个用于的表格

```
-- setup test raster table --
DROP TABLE IF EXISTS funky_shapes;
CREATE TABLE funky_shapes(rast raster);

INSERT INTO funky_shapes(rast)
WITH ref AS (
  SELECT ST_MakeEmptyRaster( 200, 200, 0, 200, 1, -1, 0, 0) AS rast
)
SELECT
  ST_Union(rast)
FROM (
  SELECT
    ST_AsRaster(
      ST_Rotate(
        ST_Buffer(
          ST_GeomFromText('LINESTRING(0 2,50 50,150 150,125 50)'),
          i*2
        ),
        pi() * i * 0.125, ST_Point(50,50)
      ),
    ),
```

```

        ref.rast, '8BUI'::text, i * 5
    ) AS rast
FROM ref
CROSS JOIN generate_series(1, 10, 3) AS i
) AS shapes;

```

```

SELECT
  ST_NumBands(rast) As n_orig,
  ST_NumBands(ST_ColorMap(rast,1, 'greyscale')) As ngrey,
  ST_NumBands(ST_ColorMap(rast,1, 'pseudocolor')) As npseudo,
  ST_NumBands(ST_ColorMap(rast,1, 'fire')) As nfire,
  ST_NumBands(ST_ColorMap(rast,1, 'bluered')) As nbluered,
  ST_NumBands(ST_ColorMap(rast,1, '
100% 255  0  0
80%  160  0  0
50%  130  0  0
30%   30  0  0
20%   60  0  0
0%    0  0  0
nv 255 255 255
  ')) As nred
FROM funky_shapes;

```

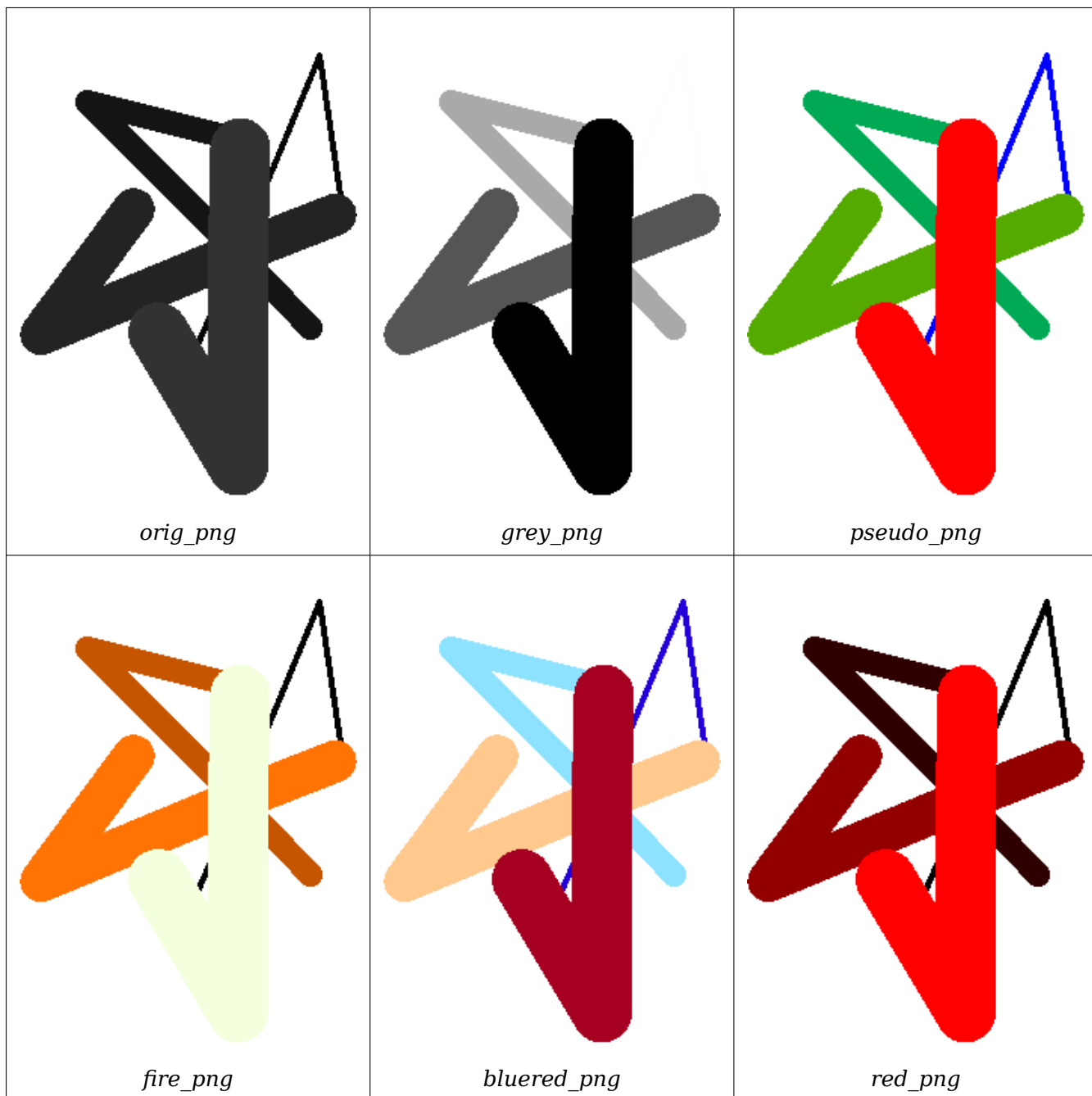
n_orig	ngrey	npseudo	nfire	nbluered	nred
1	1	4	4	4	3

示例：使用 **ST\_AspNG** 比不同色相的外

```

SELECT
  ST_AspNG(rast) As orig_png,
  ST_AspNG(ST_ColorMap(rast,1, 'greyscale')) As grey_png,
  ST_AspNG(ST_ColorMap(rast,1, 'pseudocolor')) As pseudo_png,
  ST_AspNG(ST_ColorMap(rast,1, 'nfire')) As fire_png,
  ST_AspNG(ST_ColorMap(rast,1, 'bluered')) As bluered_png,
  ST_AspNG(ST_ColorMap(rast,1, '
100% 255  0  0
80%  160  0  0
50%  130  0  0
30%   30  0  0
20%   60  0  0
0%    0  0  0
nv 255 255 255
  ')) As red_png
FROM funky_shapes;

```



#### 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);

## 描述

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。



### 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：格式 1

```
SET postgis.gdal_enabled_drivers = 'ENABLE_ALL';
SET postgis.enable_outdb_rasters = True;

WITH apple AS (
  SELECT ST_AddBand(
    ST_MakeEmptyRaster(350, 246, 0, 0, 1, -1, 0, 0, 0),
    '/tmp/apple.png'::text,
    NULL::int[]
  ) AS rast
)
SELECT
  ST_AsPNG(rast) AS original_png,
  ST_AsPNG(ST_Grayscale(rast)) AS grayscale_png
FROM apple;
```



*original\_png*



*grayscale\_png*

示例：格式 2

```
SET postgis.gdal_enabled_drivers = 'ENABLE_ALL';
SET postgis.enable_outdb_rasters = True;

WITH apple AS (
  SELECT ST_AddBand(
    ST_MakeEmptyRaster(350, 246, 0, 0, 1, -1, 0, 0, 0),
    '/tmp/apple.png'::text,
    NULL::int[]
  ) AS rast
)
SELECT
  ST_AsPNG(rast) AS original_png,
  ST_AsPNG(ST_Grayscale(
    ARRAY[
      ROW(rast, 1)::rastbandarg, -- red
      ROW(rast, 2)::rastbandarg, -- green
      ROW(rast, 3)::rastbandarg, -- blue
    ]::rastbandarg[]
  )) AS grayscale_png
FROM apple;
```

相关信息

[ST\\_AsPNG](#), [ST\\_Reclass](#), [ST\\_ColorMap](#)

### 11.12.4 ST\_Intersection

ST\_Intersection — 返回一个栅格或一几何像素，表示两个栅格的共享部分或栅格矢量化和几何形的几何交集。

#### 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);
```

#### 描述

返回一个栅格或一几何像素，表示两个栅格的共享部分或栅格矢量化和几何形的几何交集。

前三个体返回一 geomval，在向量空间中工作。首先将栅格矢量化（使用 [ST\\_DumpAsPolygons](#)）成一 geomval 行，然后使用 [ST\\_Intersection \(geometry, Geometry\)](#) PostGIS 函数将些行与几何相交。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” 来返回几何形和生成的 geomval 集的部分。例如 (ST\_Intersection(rast, geom)).geom



其他栅格返回光栅，在光栅空域中工作。他使用 `ST_MapAlgebraExpr` 的栅格版本来行交集。

生成的栅格范围限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结果中的无数据像素。

由 `ST_Intersection` 生成的栅格必不相交的区域分配 `nodata`。您可以通过提供一个包含一个或两个 `nodata` 的 `nodataval[]` 数组来定或替任何结果段的 `nodata`，具体取决于您是否求“BAND1”、“BAND2”或“BOTH”段。数组中的第一个替第一个中的无数据，第二个替第二个中的无数据。如果一个入没有定 `nodata` 并且没有以数组形式提供，使用 `ST_MinPossibleValue` 函数一个。所有接受 `nodata` 数组的栅格也可以接受一个，将分配每个求的。

在所有栅格中，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之的交集，参 [ST\\_Clip](#)。



#### Note

要更好地控制结果范围或遇到无数据返回的内容，使用 [ST\\_MapAlgebraExpr](#) 的栅格版本。



#### Note

要算栅格波段与栅格空域中几何形的交集，使用 [ST\\_Clip](#)。 `ST_Clip` 适用于多波段栅格，并且不返回与栅格化几何化的波段。



#### Note

`ST_Intersection` 与 [ST\\_Intersects](#) 以及栅格列和/或几何列上的索引合使用。

增：2.0.0 - 引入了光栅空域中的交集。在 2.0.0 之前的早期版本中，支持在向量空域中行交集。

示例：几何、光栅——生几何

```
SELECT
  foo.rid,
  foo.gid,
  ST_AsText((foo.geomval).geom) As geomwkt,
  (foo.geomval).val
FROM (
  SELECT
    A.rid,
    g.gid,
    ST_Intersection(A.rast, g.geom) As geomval
  FROM dummy_rast AS A
  CROSS JOIN (
    VALUES
      (1, ST_Point(3427928, 5793243.85) ),
      (2, ST_GeomFromText('LINESTRING(3427927.85 5793243.75,3427927.8 5793243.75,3427927.8 5793243.8)')),
      (3, ST_GeomFromText('LINESTRING(1 2, 3 4)'))
    ) As g(gid,geom)
  WHERE A.rid = 2
) As foo;
```

rid	gid	geomwkt	val
-----	-----	---------	-----





```
CREATE OR REPLACE FUNCTION sample_callbackfunc(value double precision[][][], position ←
integer[][], VARIADIC userargs text[])
RETURNS double precision
AS $$
BEGIN
    RETURN 0;
END;
$$ LANGUAGE 'plpgsql' IMMUTABLE;
```

`callbackfunc` 必须具有三个参数：一个 3 维双精度数组、一个 2 维整数数组和一个可变的 1 维文本数组。第一个参数 `value` 是所有输入栅格的交集（双精度）。三个维度（其中索引从 1 开始）是：栅格号、行 `y`、列 `x`。第二个参数 `position` 是输出栅格和输入栅格的像素位置集。外部维度（索引从 0 开始）是栅格 #。外部维度索引 0 的位置是输出栅格的像素位置。对于每个外部维度，内部维度中有 `n` 个元素用于 `X` 和 `Y`。第三个参数 `userargs` 用于任何用 `callbackfunc` 指定的参数。

将 `regprocedure` 参数替换为 SQL 函数需要完整的函数名，然后替换 `regprocedure` 类型。要将上述示例 PL/pgSQL 函数作为参数，参数的 SQL 是：

```
'sample_callbackfunc(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` 的制表符。

**mask** `n` 维数字数组（矩阵），用于映射代数回函数的元格。0 表示相应元格被忽略数据，1 表示相应元格数据。如果重置 `true`，这些值将用作乘数，以乘以区域位置中的像素。

**weighted** 布罗伊（true/false）表示掩码是否加权（乘以原始值）（适用于采用掩码的原型）。

**pixeltype** 如果输入像素类型，新栅格的一个波段将属于该像素类型。如果将像素类型设置为 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）相同的像素类型。类型：第二个、最后一个）。如果有疑问，始终指定像素类型。

输出栅格的像素类型必须是 `ST_BandPixelType` 中列出的类型，或者省略或设置为 NULL。

**extenttype** 可能的类型 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`。参看例 4。

**distancex** `x` 方向上距参考元格的距离（以像素为单位）。因此，结果矩形的宽度将  $2 * \text{distancex} + 1$ 。如果未指定，则参考元格（0 的区域）。

**distancey** `y` 方向上距参考元的距离（以像素为单位）。结果矩形的高度将  $2 * \text{distancey} + 1$ 。如果未指定，则参考元格（0 的区域）。

**userargs** `callbackfunc` 的第三个参数是一个可变的文本数组。所有尾随文本参数都传递到指定的 `callbackfunc`，并包含在 `userargs` 参数中。



#### Note

有关 VARIADIC 关键字的更多信息，参看 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返回函数行处理，`text[]` 参数在 `callbackfunc` 都是必需的。

☐体 1 接受 `rastbandarg` 数组，允☐在☐多☐格和/或☐多波段上使用地☐代数☐算。☐参☐示例☐体 1。

☐体 2 和 3 ☐一个☐格的一个或多个波段☐行操作。☐参☐示例☐体 2 和 3。

☐体 4 ☐☐个☐格☐行操作，每个☐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：格式 1

一个☐格，一个波段

```
WITH foo AS (
  SELECT 1 AS rid, ST_AddBand(ST_MakeEmptyRaster(2, 2, 0, 0, 1, -1, 0, 0, 0), 1, '16BUI', ←
    1, 0) AS rast
)
SELECT
  ST_MapAlgebra(
    ARRAY[ROW(rast, 1)]::rastbandarg[],
    'sample_callbackfunc(double precision[], int[], text[])'::regprocedure
  ) AS rast
FROM foo
```

一个☐格，多个波段

```
WITH foo AS (
  SELECT 1 AS rid, ST_AddBand(ST_AddBand(ST_AddBand(ST_MakeEmptyRaster(2, 2, 0, 0, 1, -1, ←
    0, 0, 0), 1, '16BUI', 1, 0), 2, '8BUI', 10, 0), 3, '32BUI', 100, 0) AS rast
)
SELECT
  ST_MapAlgebra(
    ARRAY[ROW(rast, 3), ROW(rast, 1), ROW(rast, 3), ROW(rast, 2)]::rastbandarg[],
    'sample_callbackfunc(double precision[], int[], text[])'::regprocedure
  ) AS rast
FROM foo
```

多个☐格，多个波段

```
WITH foo AS (
  SELECT 1 AS rid, ST_AddBand(ST_AddBand(ST_AddBand(ST_MakeEmptyRaster(2, 2, 0, 0, 1, -1, ←
    0, 0, 0), 1, '16BUI', 1, 0), 2, '8BUI', 10, 0), 3, '32BUI', 100, 0) AS rast UNION ←
    ALL
  SELECT 2 AS rid, ST_AddBand(ST_AddBand(ST_AddBand(ST_MakeEmptyRaster(2, 2, 0, 0, 1, -1, ←
    0, 0, 0), 1, '16BUI', 2, 0), 2, '8BUI', 20, 0), 3, '32BUI', 300, 0) AS rast
)
SELECT
  ST_MapAlgebra(
    ARRAY[ROW(t1.rast, 3), ROW(t2.rast, 1), ROW(t2.rast, 3), ROW(t1.rast, 2)]:: ←
    rastbandarg[],
    'sample_callbackfunc(double precision[], int[], text[])'::regprocedure
  ) AS rast
FROM foo t1
CROSS JOIN foo t2
WHERE t1.rid = 1
  AND t2.rid = 2
```

☐域覆盖☐☐的完整示例。此☐☐☐适用于 PostgreSQL 9.1 或更高版本。

```
WITH foo AS (
  SELECT 0 AS rid, ST_AddBand(ST_MakeEmptyRaster(2, 2, 0, 0, 1, -1, 0, 0, 0), 1, '16BUI', ←
    1, 0) AS rast UNION ALL
```

```

SELECT 1, ST_AddBand(ST_MakeEmptyRaster(2, 2, 2, 0, 1, -1, 0, 0, 0), 1, '16BUI', 2, 0) ←
  AS rast UNION ALL
SELECT 2, ST_AddBand(ST_MakeEmptyRaster(2, 2, 4, 0, 1, -1, 0, 0, 0), 1, '16BUI', 3, 0) ←
  AS rast UNION ALL

SELECT 3, ST_AddBand(ST_MakeEmptyRaster(2, 2, 0, -2, 1, -1, 0, 0, 0), 1, '16BUI', 10, ←
  0) AS rast UNION ALL
SELECT 4, ST_AddBand(ST_MakeEmptyRaster(2, 2, 2, -2, 1, -1, 0, 0, 0), 1, '16BUI', 20, ←
  0) AS rast UNION ALL
SELECT 5, ST_AddBand(ST_MakeEmptyRaster(2, 2, 4, -2, 1, -1, 0, 0, 0), 1, '16BUI', 30, ←
  0) AS rast UNION ALL

SELECT 6, ST_AddBand(ST_MakeEmptyRaster(2, 2, 0, -4, 1, -1, 0, 0, 0), 1, '16BUI', 100, ←
  0) AS rast UNION ALL
SELECT 7, ST_AddBand(ST_MakeEmptyRaster(2, 2, 2, -4, 1, -1, 0, 0, 0), 1, '16BUI', 200, ←
  0) AS rast UNION ALL
SELECT 8, ST_AddBand(ST_MakeEmptyRaster(2, 2, 4, -4, 1, -1, 0, 0, 0), 1, '16BUI', 300, ←
  0) AS rast
)
SELECT
  t1.rid,
  ST_MapAlgebra(
    ARRAY[ROW(ST_Union(t2.rast), 1)]::rastbandarg[],
    'sample_callbackfunc(double precision[], int[], text[])'::regprocedure,
    '32BUI',
    'CUSTOM', t1.rast,
    1, 1
  ) AS rast
FROM foo t1
CROSS JOIN foo t2
WHERE t1.rid = 4
      AND t2.rid BETWEEN 0 AND 8
      AND ST_Intersects(t1.rast, t2.rast)
GROUP BY t1.rid, t1.rast

```

示例  似于前一个具有  域覆盖范  的 ，但适用于 PostgreSQL 9.0。

```

WITH src AS (
  SELECT 0 AS rid, ST_AddBand(ST_MakeEmptyRaster(2, 2, 0, 0, 1, -1, 0, 0, 0), 1, '16BUI', ←
    1, 0) AS rast UNION ALL
  SELECT 1, ST_AddBand(ST_MakeEmptyRaster(2, 2, 2, 0, 1, -1, 0, 0, 0), 1, '16BUI', 2, 0) ←
    AS rast UNION ALL
  SELECT 2, ST_AddBand(ST_MakeEmptyRaster(2, 2, 4, 0, 1, -1, 0, 0, 0), 1, '16BUI', 3, 0) ←
    AS rast UNION ALL

  SELECT 3, ST_AddBand(ST_MakeEmptyRaster(2, 2, 0, -2, 1, -1, 0, 0, 0), 1, '16BUI', 10, ←
    0) AS rast UNION ALL
  SELECT 4, ST_AddBand(ST_MakeEmptyRaster(2, 2, 2, -2, 1, -1, 0, 0, 0), 1, '16BUI', 20, ←
    0) AS rast UNION ALL
  SELECT 5, ST_AddBand(ST_MakeEmptyRaster(2, 2, 4, -2, 1, -1, 0, 0, 0), 1, '16BUI', 30, ←
    0) AS rast UNION ALL

  SELECT 6, ST_AddBand(ST_MakeEmptyRaster(2, 2, 0, -4, 1, -1, 0, 0, 0), 1, '16BUI', 100, ←
    0) AS rast UNION ALL
  SELECT 7, ST_AddBand(ST_MakeEmptyRaster(2, 2, 2, -4, 1, -1, 0, 0, 0), 1, '16BUI', 200, ←
    0) AS rast UNION ALL
  SELECT 8, ST_AddBand(ST_MakeEmptyRaster(2, 2, 4, -4, 1, -1, 0, 0, 0), 1, '16BUI', 300, ←
    0) AS rast
)
WITH foo AS (
  SELECT
    t1.rid,

```

```

        ST_Union(t2.rast) AS rast
    FROM src t1
    JOIN src t2
        ON ST_Intersects(t1.rast, t2.rast)
        AND t2.rid BETWEEN 0 AND 8
    WHERE t1.rid = 4
    GROUP BY t1.rid
), bar AS (
    SELECT
        t1.rid,
        ST_MapAlgebra(
            ARRAY[ROW(t2.rast, 1)]::rastbandarg[],
            'raster_nmapalgebra_test(double precision[], int[], text[])::regprocedure,
            '32BUI',
            'CUSTOM', t1.rast,
            1, 1
        ) AS rast
    FROM src t1
    JOIN foo t2
        ON t1.rid = t2.rid
)
SELECT
    rid,
    (ST_Metadata(rast)),
    (ST_BandMetadata(rast, 1)),
    ST_Value(rast, 1, 1, 1)
FROM bar;

```

示例：☒体 2 和 3

一个☒格，多个波段

```

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

```

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

```

示例：☒体 4

☒个☒格，☒个波段

```

WITH foo AS (
  SELECT 1 AS rid, ST_AddBand(ST_AddBand(ST_AddBand(ST_MakeEmptyRaster(2, 2, 0, 0, 1, -1, ←
    0, 0, 0), 1, '16BUI', 1, 0), 2, '8BUI', 10, 0), 3, '32BUI', 100, 0) AS rast UNION ←
  ALL
  SELECT 2 AS rid, ST_AddBand(ST_AddBand(ST_AddBand(ST_MakeEmptyRaster(2, 2, 0, 0, 1, 1, -1, ←
    0, 0, 0), 1, '16BUI', 2, 0), 2, '8BUI', 20, 0), 3, '32BUI', 300, 0) AS rast
)
SELECT
  ST_MapAlgebra(
    t1.rast, 2,
    t2.rast, 1,
    'sample_callbackfunc(double precision[], int[], text[])::regprocedure
  ) AS rast
FROM foo t1
CROSS JOIN foo t2
WHERE t1.rid = 1
  AND t2.rid = 2

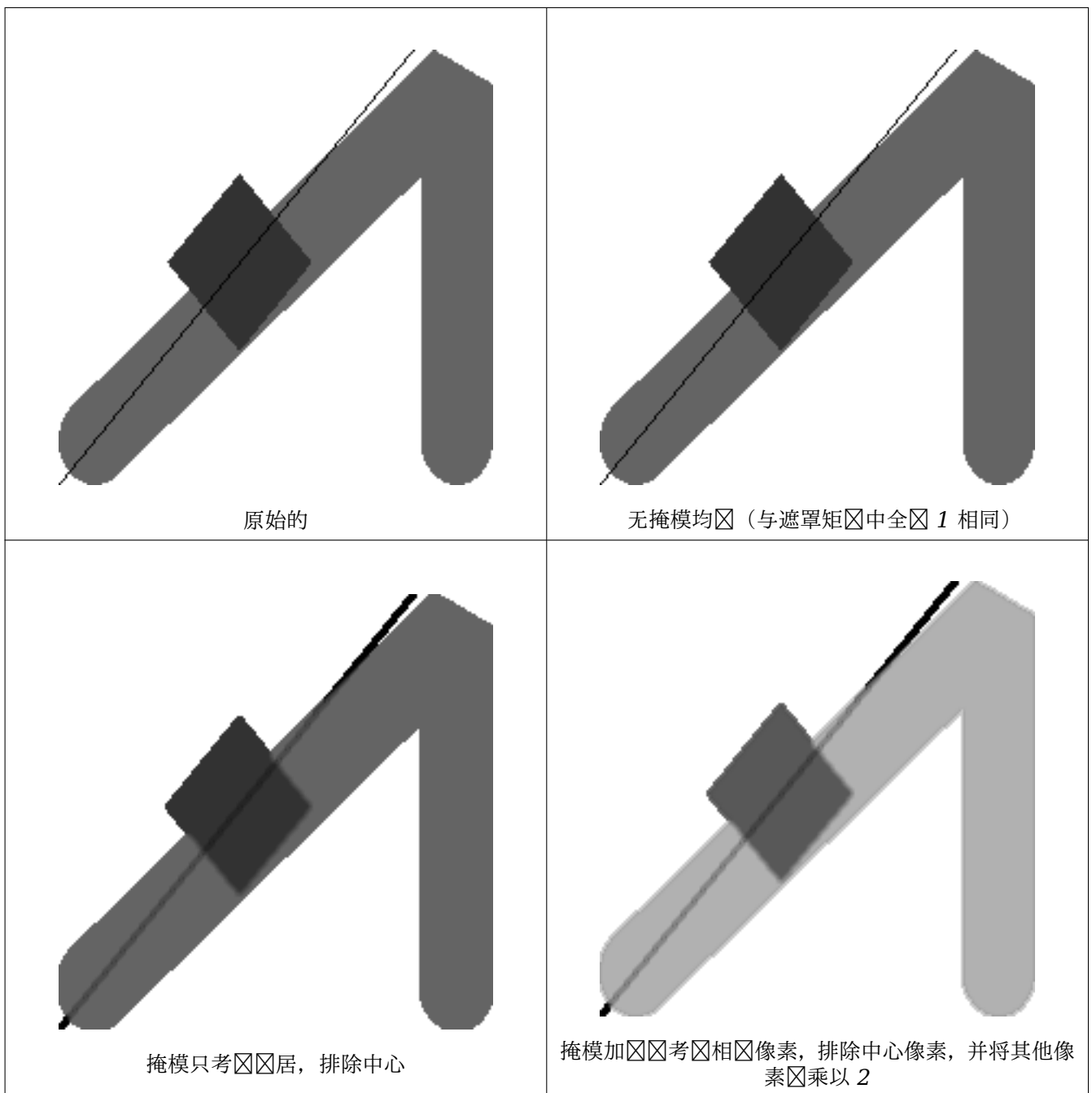
```

示例：使用掩模

```

WITH foo AS (SELECT
  ST_SetBandNoDataValue(
  ST_SetValue(ST_SetValue(ST_AsRaster(
    ST_Buffer(
      ST_GeomFromText('LINESTRING(50 50,100 90,100 50)'), 5,'join=bevel'),
      200,200,ARRAY['8BUI'], ARRAY[100], ARRAY[0]), ST_Buffer('POINT(70 70)'):: ←
      geometry,10,'quad_segs=1') ,50),
    'LINESTRING(20 20, 100 100, 150 98)')::geometry,1),0) AS rast )
SELECT 'original' AS title, rast
FROM foo
UNION ALL
SELECT 'no mask mean value' AS title, ST_MapAlgebra(rast,1,'ST_mean4ma(double precision[], ←
  int[], text[])::regprocedure) AS rast
FROM foo
UNION ALL
SELECT 'mask only consider neighbors, exclude center' AS title, ST_MapAlgebra(rast,1,' ←
  ST_mean4ma(double precision[], int[], text[])::regprocedure,
  '{{1,1,1}, {1,0,1}, {1,1,1}}'::double precision[], false) As rast
FROM foo
UNION ALL
SELECT 'mask weighted only consider neighbors, exclude center multi other pixel values by ←
  2' AS title, ST_MapAlgebra(rast,1,'ST_mean4ma(double precision[], int[], text[]):: ←
  regprocedure,
  '{{2,2,2}, {2,0,2}, {2,2,2}}'::double precision[], true) As rast
FROM foo;

```



#### 相关信息

[rastbandarg](#), [ST\\_Union](#), [ST\\_MapAlgebra \(expression version\)](#)

### 11.12.6 ST\_MapAlgebra (expression version)

`ST_MapAlgebra (expression version)` — 表达式版本 - 在给定一个输入栅格、波段索引和一个或多个用表达式指定的 SQL 表达式式的情况下, 返回波段栅格。

## Synopsis

```
raster ST_MapAlgebra(raster rast, integer nband, text pixeltyp, text expression, double precision nodataval=NULL);
raster ST_MapAlgebra(raster rast, text pixeltyp, text expression, double precision nodataval=NULL);
raster ST_MapAlgebra(raster rast1, integer nband1, raster rast2, integer nband2, text expression, text pixeltyp=NULL, text extenttype=INTERSECTION, text nodata1expr=NULL, text nodata2expr=NULL, double precision nodatanodataval=NULL);
raster ST_MapAlgebra(raster rast1, raster rast2, text expression, text pixeltyp=NULL, text extenttype=INTERSECTION, text nodata1expr=NULL, text nodata2expr=NULL, double precision nodatanodataval=NULL);
```

## 描述

表式版本 - 在给定一个输入栅格、波段索引和一个或多个用指定 SQL 表式的情况下，返回波段栅格。

可用性：2.1.0

描述：体 1 和 2（一个栅格）

通用由输入栅格 (rast) 上的 expression 定义的有效 PostgreSQL 代数算来建新波段栅格。如果未提供 nband，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 NULL，新的栅格将具有与输入栅格相同的像素型。

• 允许 expression 的关键词

1. [rast] - 感兴趣像素的像素
2. [rast.val] - 感兴趣像素的像素
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：体 3 和 4（个栅格）

通用个输入栅格波段 rast1, (rast2) 上的 expression 定义的个波段用有效的 PostgreSQL 代数算，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 参数定义的范围。

expression 涉及个栅格的 PostgreSQL 代数表式和 PostgreSQL 定义的函数/算符，用于定义像素相交的像素。例如 (([rast1] [rast2])/2.0)::integer

pixeltyp 出栅格的果像素型。必是 **ST\_BandPixelType** 中列出的一，可省略或置 NULL。如果未入或置 NULL，将默认第一个栅格的像素型。

extenttype 控制生成的栅格的范围

1. INTERSECTION - 新栅格的范围是个栅格的交集。是默认置。
2. UNION - 新栅格的范围是个栅格的并集。
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。

nodataexpr 涉及 rast2 的代数表式或定义当 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` 的像素无数据且空的 `rast1` 像素具有返回的内容。

**nodatanodataval** 当空的 `rast1` 和 `rast2` 像素均无数据要返回的数常量。

• `expression`、`nodataexpr` 和 `nodata2expr` 中允许使用的关键字

1. `[rast1]` - `rast1` 中感兴趣的像素的像素
2. `[rast1.val]` - `rast1` 中感兴趣像素的像素
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` 中感兴趣的像素的像素
6. `[rast2.val]` - `rast2` 中感兴趣像素的像素
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：图 1 和 2

```
WITH foo AS (
  SELECT ST_AddBand(ST_MakeEmptyRaster(10, 10, 0, 0, 1, 1, 0, 0), '32BF'::text, 1, -1) ←
  AS rast
)
SELECT
  ST_MapAlgebra(rast, 1, NULL, 'ceil([rast]*[rast.x]/[rast.y]+[rast.val])')
FROM foo;
```

示例：图 3 和 4

```
WITH foo AS (
  SELECT 1 AS rid, ST_AddBand(ST_AddBand(ST_AddBand(ST_MakeEmptyRaster(2, 2, 0, 0, 1, -1, ←
    0, 0, 0), 1, '16BUI', 1, 0), 2, '8BUI', 10, 0), 3, '32BUI'::text, 100, 0) AS rast ←
  UNION ALL
  SELECT 2 AS rid, ST_AddBand(ST_AddBand(ST_AddBand(ST_MakeEmptyRaster(2, 2, 0, 1, 1, -1, ←
    0, 0, 0), 1, '16BUI', 2, 0), 2, '8BUI', 20, 0), 3, '32BUI'::text, 300, 0) AS rast
)
SELECT
  ST_MapAlgebra(
    t1.rast, 2,
    t2.rast, 1,
    '([rast2] + [rast1.val]) / 2'
  ) AS rast
FROM foo t1
CROSS JOIN foo t2
WHERE t1.rid = 1
AND t2.rid = 2;
```

相关信息

[rastbandarg](#), [ST\\_Union](#), [ST\\_MapAlgebra \(callback function version\)](#)

## 11.12.7 ST\_MapAlgebraExpr

ST\_MapAlgebraExpr — 1 波段版本：通过输入波段和提供的像素型用有效的 PostgreSQL 代数算来建新的波段格。如果未指定波段，假定波段 1。

### Synopsis

raster **ST\_MapAlgebraExpr**(raster rast, integer band, text pixeltyp, text expression, double precision nodataval=NULL);

raster **ST\_MapAlgebraExpr**(raster rast, text pixeltyp, text expression, double precision nodataval=NULL);

### 描述



#### Warning

**ST\_MapAlgebraExpr** 自 2.1.0 起已弃用。使用 **ST\_MapAlgebra (expression version)** (表式版本) 代替。

通过用由输入格 (rast) 上的 expression 的有效 PostgreSQL 代数算来建新的波段格。如果未指定波段，假定波段 1。新格将具有与原始格相同的地理参考、度和高度，但只有一个波段。

如果输入像素型，新格将具有像素型的波段。如果将像素型设为 NULL，新的格将具有与输入格相同的像素型。

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

根据原始格建一个新的 1 波段格，该格是原始格波段的模 2 的函数。

```
ALTER TABLE dummy_rast ADD COLUMN map_rast raster;
UPDATE dummy_rast SET map_rast = ST_MapAlgebraExpr(rast,NULL,'mod([rast]::numeric,2)') ←
WHERE rid = 2;
```

```
SELECT
  ST_Value(rast,1,i,j) As origval,
  ST_Value(map_rast, 1, i, j) As mapval
FROM dummy_rast
CROSS JOIN generate_series(1, 3) AS i
CROSS JOIN generate_series(1,3) AS j
WHERE rid = 2;
```

origval	mapval
253	1
254	0
253	1
253	1
254	0
254	0
250	0
254	0
254	0

从我重新分格的原始栅格中建立一个新的像素型 2BUI 的 1 波段栅格，并将 `nodata` 置 0。

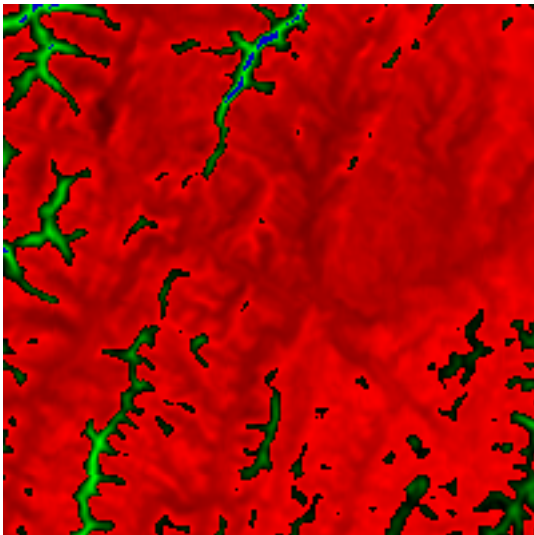
```
ALTER TABLE dummy_rast ADD COLUMN map_rast2 raster;
UPDATE dummy_rast SET
  map_rast2 = ST_MapAlgebraExpr(rast,'2BUI'::text,'CASE WHEN [rast] BETWEEN 100 and 250
    THEN 1 WHEN [rast] = 252 THEN 2 WHEN [rast] BETWEEN 253 and 254 THEN 3 ELSE 0 END'::
    text, '0')
WHERE rid = 2;
```

```
SELECT DISTINCT
  ST_Value(rast,1,i,j) As origval,
  ST_Value(map_rast2, 1, i, j) As mapval
FROM dummy_rast
CROSS JOIN generate_series(1, 5) AS i
CROSS JOIN generate_series(1,5) AS j
WHERE rid = 2;
```

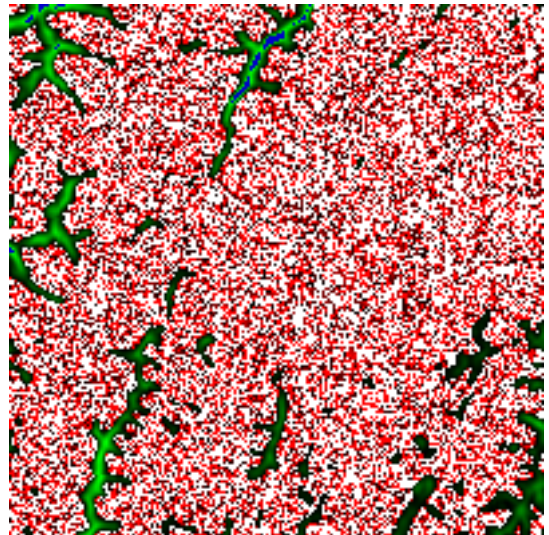
origval	mapval
249	1
250	1
251	1
252	2
253	3
254	3

```
SELECT
  ST_BandPixelType(map_rast2) As b1pixtyp
FROM dummy_rast
WHERE rid = 2;
```

```
b1pixtyp
-----
2BUI
```



原始 (*rast\_view* 列)



*rast\_view\_ma*

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 个波段保持不变。

```

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 波段版本：通过提供的两个输入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。

### 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);

```

描述



### Warning

**ST\_MapAlgebraExpr** 自 2.1.0 起已弃用。使用 **ST\_MapAlgebra (expression version)** (表式版本) 代替。

通过两个输入波段 *rast1*, (*rast2*) 上的 *expression* 定义的两个波段用有效的 PostgreSQL 代数算，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/算符，用于定义像素相交的像素。例如  $(([rast1] [rast2])/2.0)::integer$

**pixeltype** 输出栅格的像素型。必是 **ST\_BandPixelType** 中列出的一，可省略或置 NULL。如果未入或置 NULL，将默认第一个栅格的像素型。

**extenttype** 控制生成的栅格的范围

1. INTERSECTION - 新栅格的范围是二个栅格的交集。是默认设置。
2. UNION - 新栅格的范围是二个栅格的并集。
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。

**nodataexpr** 涉及 `rast2` 的代数表达式或定当 `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` 像素均无数据要返回的数常量。

如果入 `pixeltype`，新栅格将具有像素型的波段。如果将像素型 NULL 或未指定像素型，新的栅格波段将具有与入 `rast1` 波段相同的像素型。

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 条波段的交集和并集

根据原始栅格建一个新的 1 波段栅格，栅格是原始栅格波段的模 2 的函数。

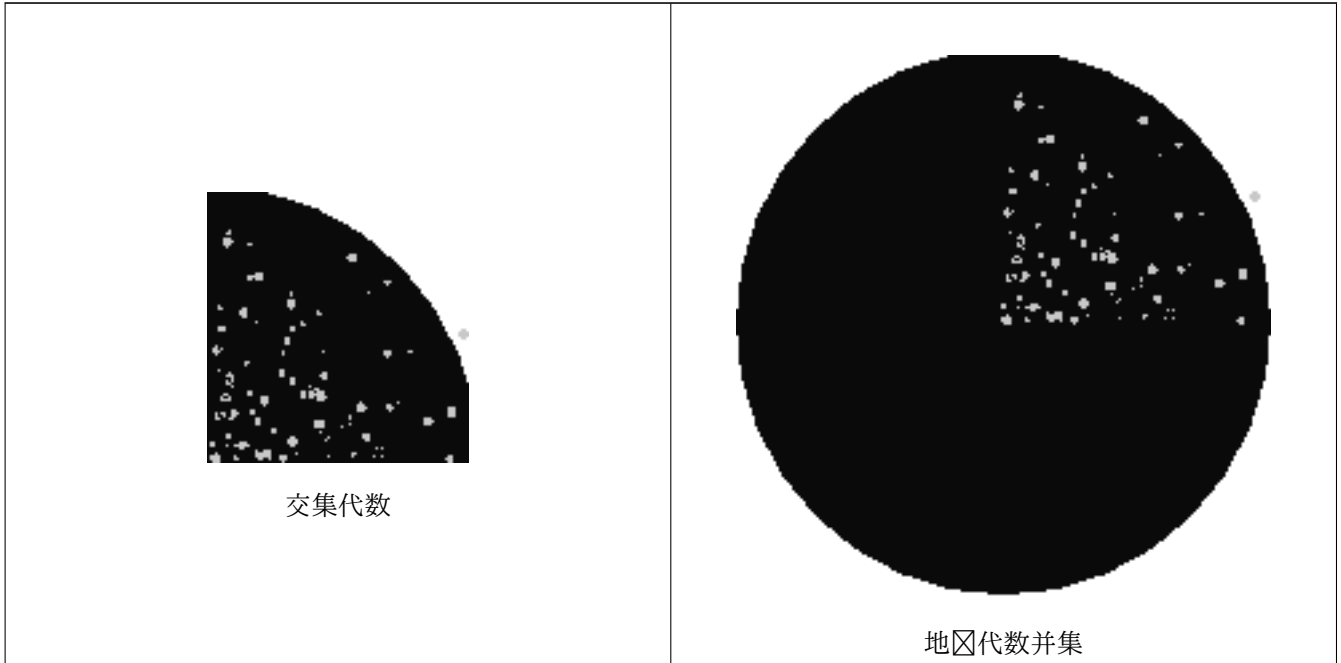
```
--Create a cool set of rasters --
DROP TABLE IF EXISTS fun_shapes;
CREATE TABLE fun_shapes(rid serial PRIMARY KEY, fun_name text, rast raster);

-- Insert some cool shapes around Boston in Massachusetts state plane meters --
INSERT INTO fun_shapes(fun_name, rast)
VALUES ('ref', ST_AsRaster(ST_MakeEnvelope(235229, 899970, 237229, 901930,26986),200,200,'8BUI',0,0));

INSERT INTO fun_shapes(fun_name,rast)
WITH ref(rast) AS (SELECT rast FROM fun_shapes WHERE fun_name = 'ref' )
SELECT 'area' AS fun_name, ST_AsRaster(ST_Buffer(ST_SetSRID(ST_Point(236229, 900930),26986)
, 1000),
    ref.rast,'8BUI', 10, 0) As rast
FROM ref
UNION ALL
SELECT 'rand bubbles',
    ST_AsRaster(
        (SELECT ST_Collect(geom)
         FROM (SELECT ST_Buffer(ST_SetSRID(ST_Point(236229 + i*random()*100, 900930 + j*random()*100),26986), random()*20) As geom
              FROM generate_series(1,10) As i, generate_series(1,10) As j
              ) As foo ), ref.rast,'8BUI', 200, 0)
FROM ref;

--map them -
SELECT ST_MapAlgebraExpr(
    area.rast, bub.rast, '[rast2.val]', '8BUI', 'INTERSECTION', '[rast2.val]', '[rast1.val]') As interrast,
    ST_MapAlgebraExpr(
    area.rast, bub.rast, '[rast2.val]', '8BUI', 'UNION', '[rast2.val]', '[rast1.val]') As unionrast
FROM
```

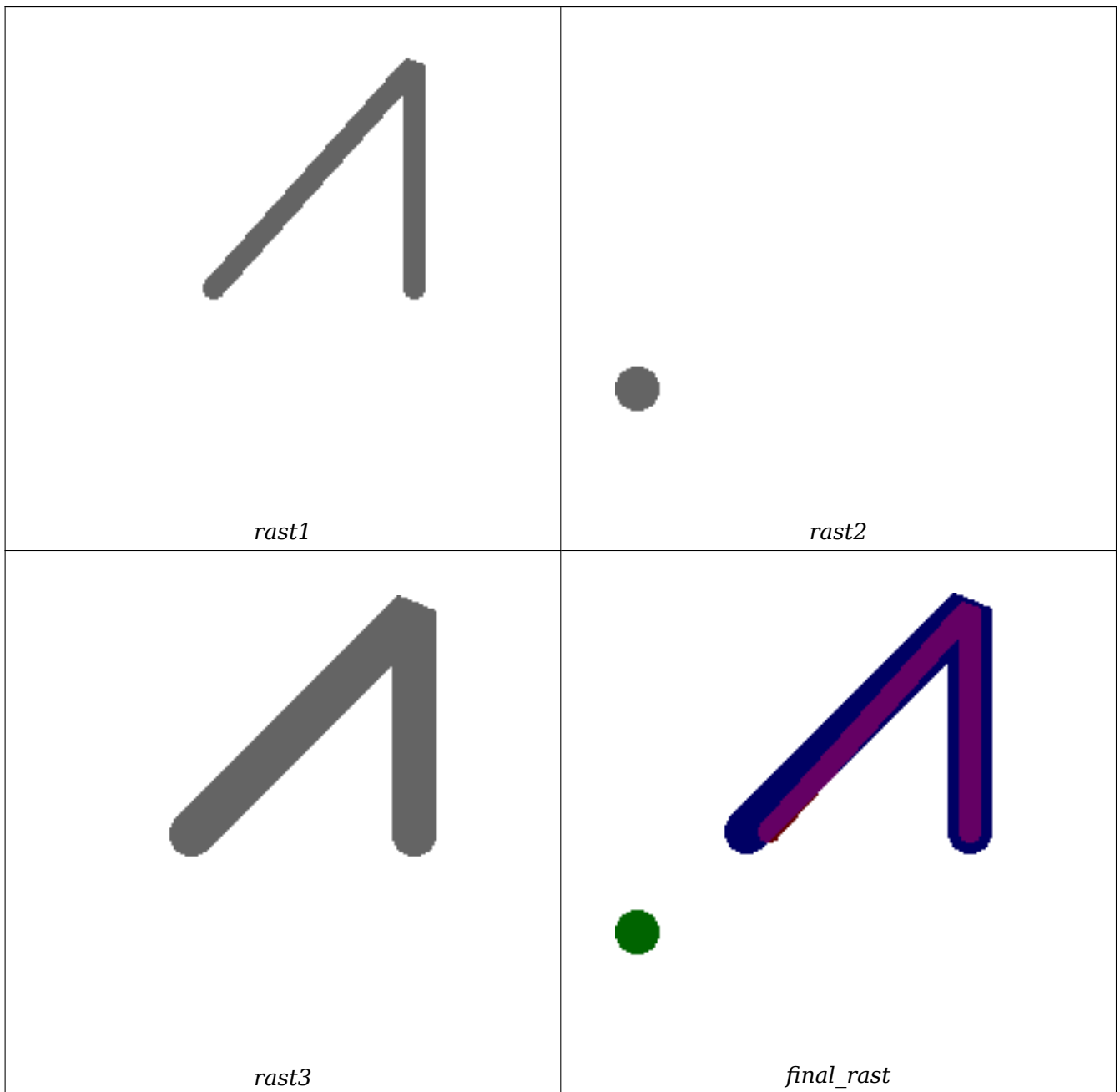
```
(SELECT rast FROM fun_shapes WHERE
fun_name = 'area') As area
CROSS JOIN (SELECT rast
FROM fun_shapes WHERE
fun_name = 'rand bubbles') As bub
```



示例：将网格作独立的波段覆盖在画布上

```
-- we use ST_AsPNG to render the image so all single band ones look grey --
WITH mygeoms
  AS ( SELECT 2 As bnum, ST_Buffer(ST_Point(1,5),10) As geom
      UNION ALL
      SELECT 3 AS bnum,
            ST_Buffer(ST_GeomFromText('LINESTRING(50 50,150 150,150 50)'), 10,'join=↔
            bevel') As geom
      UNION ALL
      SELECT 1 As bnum,
            ST_Buffer(ST_GeomFromText('LINESTRING(60 50,150 150,150 50)'), 5,'join=↔
            bevel') As geom
    ),
  -- define our canvas to be 1 to 1 pixel to geometry
  canvas
  AS (SELECT ST_AddBand(ST_MakeEmptyRaster(200,
    200,
    ST_XMin(e)::integer, ST_YMax(e)::integer, 1, -1, 0, 0) , '8BUI'::text,0) As rast
    FROM (SELECT ST_Extent(geom) As e,
            Max(ST_SRID(geom)) As srid
         from mygeoms
        ) As foo
    ),
  rbands AS (SELECT ARRAY(SELECT ST_MapAlgebraExpr(canvas.rast, ST_AsRaster(m.geom, canvas ↵
    .rast, '8BUI', 100),
    '[rast2.val]', '8BUI', 'FIRST', '[rast2.val]', '[rast1.val]') As rast
    FROM mygeoms AS m CROSS JOIN canvas
    ORDER BY m.bnum) As rasts
  )
```

```
SELECT rasts[1] As rast1 , rasts[2] As rast2, rasts[3] As rast3, ST_AddBand(
  ST_AddBand(rasts[1],rasts[2]), rasts[3]) As final_rast
FROM rbands;
```



示例：将指定地区的 2 米边界加在航空像上

```
-- Create new 3 band raster composed of first 2 clipped bands, and overlay of 3rd band with ←
  our geometry
-- This query took 3.6 seconds on PostGIS windows 64-bit install
WITH pr AS
-- Note the order of operation: we clip all the rasters to dimensions of our region
(SELECT ST_Clip(rast,ST_Expand(geom,50) ) As rast, g.geom
  FROM aerials.o_2_boston AS r INNER JOIN
-- union our parcels of interest so they form a single geometry we can later intersect with
```

```

        (SELECT ST_Union(ST_Transform(geom,26986)) AS geom
         FROM landparcels WHERE pid IN('0303890000', '0303900000')) As g
        ON ST_Intersects(rast::geometry, ST_Expand(g.geom,50))
    ),
    -- we then union the raster shards together
    -- ST_Union on raster is kinda of slow but much faster the smaller you can get the rasters
    -- therefore we want to clip first and then union
    prunion AS
    (SELECT ST_AddBand(NULL, ARRAY[ST_Union(rast,1),ST_Union(rast,2),ST_Union(rast,3)] ) As ←
     clipped,geom
     FROM pr
     GROUP BY geom)
    -- return our final raster which is the unioned shard with
    -- with the overlay of our parcel boundaries
    -- add first 2 bands, then mapalgebra of 3rd band + geometry
    SELECT ST_AddBand(ST_Band(clipped,ARRAY[1,2])
     , ST_MapAlgebraExpr(ST_Band(clipped,3), ST_AsRaster(ST_Buffer(ST_Boundary(geom),2), ←
     clipped, '8BUI',250),
     '[rast2.val]', '8BUI', 'FIRST', '[rast2.val]', '[rast1.val]')) ) As rast
    FROM prunion;

```



蓝色框是选定地区的边界

#### 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调用有效的 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 pixeltype, regprocedure onerasteruserfunc);
raster ST_MapAlgebraFct(raster rast, text pixeltype, regprocedure onerasteruserfunc, text[] VARIADIC args);
raster ST_MapAlgebraFct(raster rast, integer band, regprocedure onerasteruserfunc);
raster ST_MapAlgebraFct(raster rast, integer band, regprocedure onerasteruserfunc, text[] VARIADIC args);
raster ST_MapAlgebraFct(raster rast, integer band, text pixeltype, regprocedure onerasteruserfunc);
raster ST_MapAlgebraFct(raster rast, integer band, text pixeltype, regprocedure onerasteruserfunc, text[] VARIADIC args);
```

## 描述



### 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*) 上使用 *onerasteruserfunc* 指定的有效 PostgreSQL 函数而形成的。如果未指定波段，则假定波段 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，则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 函数的名称和名称，以及 *regprocedure*。一个非常且无用的 PL/pgSQL 函数示例是：

```
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：一个浮点值、一个可接受的整数数组和一个可接受的文本数组。第一个参数是栅格像元的值（无论栅格数据类型如何）。第二个参数是当前地理像元的位置，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 *regprodedure* 参数设置为 SQL 函数需要完整的函数名称，然后设置为 *regprocedure* 类型。要将上述示例 PL/pgSQL 函数作为参数，参数的 SQL 为：

```
'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接受的文本数组。任何 **ST\_MapAlgebraFct** 使用的所有尾随文本参数都会到指定的 *userfunction*，并包含在 *args* 参数中。



### Note

有关 VARIADIC 关键字的更多信息，请参阅 PostgreSQL 文档和 **SQL 函数** 的“具有可接受参数数量的 SQL 函数”部分。

**Note**

无论您是否将任何参数传递给 `userfunction` 行处理，`text[]` 参数在 `userfunction` 都是必需的。

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。

```
ALTER TABLE dummy_rast ADD COLUMN map_rast raster;
CREATE FUNCTION mod_fct(pixel float, pos integer[], variadic args text[])
RETURNS float
AS $$
BEGIN
    RETURN pixel::integer % 2;
END;
$$
LANGUAGE 'plpgsql' IMMUTABLE;

UPDATE dummy_rast SET map_rast = ST_MapAlgebraFct(rast,NULL,'mod_fct(float,integer[],text ←
    [])::regprocedure) WHERE rid = 2;

SELECT ST_Value(rast,1,i,j) As origval, ST_Value(map_rast, 1, i, j) As mapval
FROM dummy_rast CROSS JOIN generate_series(1, 3) AS i CROSS JOIN generate_series(1,3) AS j
WHERE rid = 2;
```

origval	mapval
253	1
254	0
253	1
253	1
254	0
254	0
250	0
254	0
254	0

从我的原始栅格中构建一个新的像素型 2BUI 的 1 波段栅格，并将其重新分区，并将 `nodata` 设置为使用函数的参数 (0)。

```
ALTER TABLE dummy_rast ADD COLUMN map_rast2 raster;
CREATE FUNCTION classify_fct(pixel float, pos integer[], variadic args text[])
RETURNS float
AS
$$
DECLARE
    nodata float := 0;
BEGIN
    IF NOT args[1] IS NULL THEN
        nodata := args[1];
    END IF;
    IF pixel < 251 THEN
        RETURN 1;
    ELSIF pixel = 252 THEN
        RETURN 2;
    ELSIF pixel
```

```

> 252 THEN
    RETURN 3;
ELSE
    RETURN nodata;
END IF;
END;
$$
LANGUAGE 'plpgsql';
UPDATE dummy_rast SET map_rast2 = ST_MapAlgebraFct(rast,'2BUI','classify_fct(float,integer ←
    [],text[])'::regprocedure, '0') WHERE rid = 2;

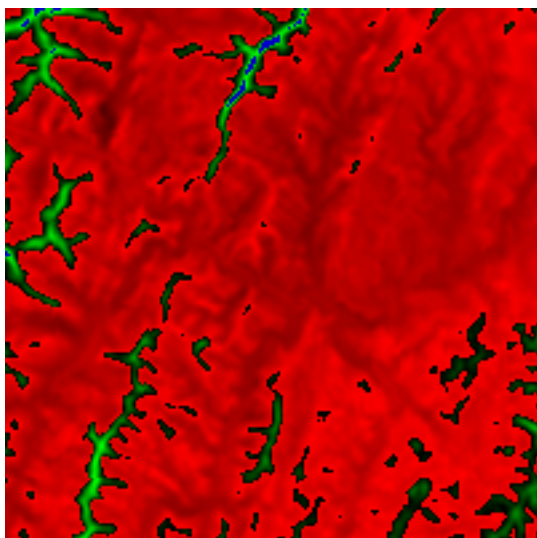
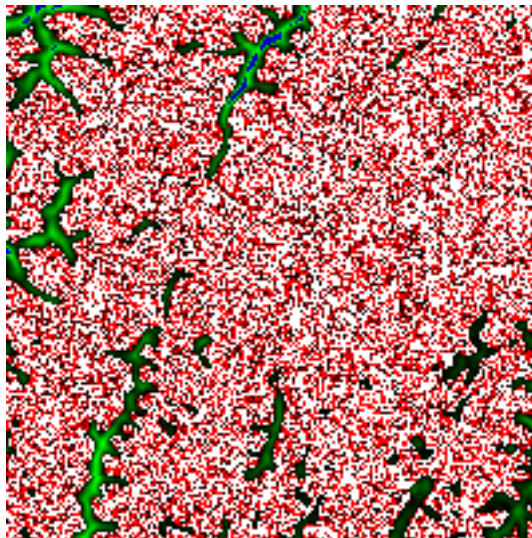
SELECT DISTINCT ST_Value(rast,1,i,j) As origval, ST_Value(map_rast2, 1, i, j) As mapval
FROM dummy_rast CROSS JOIN generate_series(1, 5) AS i CROSS JOIN generate_series(1,5) AS j
WHERE rid = 2;

origval | mapval
-----+-----
    249 |      1
    250 |      1
    251 |      1
    252 |      2
    253 |      3
    254 |      3

SELECT ST_BandPixelType(map_rast2) As b1pixtyp
FROM dummy_rast WHERE rid = 2;

b1pixtyp
-----
    2BUI

```

原始 (*rast-view* 列)*rast\_view\_ma*

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，其中第一个波段由代数更改，其余 2 个波段保持不变。

```

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;

```

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使用有效的 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 pixeltypename=same_as_rast1,
text extenttype=INTERSECTION, text[] VARIADIC userargs);
raster ST_MapAlgebraFct(raster rast1, integer band1, raster rast2, integer band2, regprocedure
tworastuserfunc, text pixeltypename=same_as_rast1, text extenttype=INTERSECTION, text[] VARIADIC
userargs);

```

描述



#### 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，该栅格是通过将输入栅格 `rast1`、`rast2` 使用由 `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。

如果输入 `pixeltypename`，新栅格将具有该像素类型的波段。如果将像素类型设为 `NULL` 或省略，新的栅格波段将具有与输入 `rast1` 波段相同的像素类型。

`tworastuserfunc` 参数必是 SQL 或 PL/pgSQL 函数的名称和名， regprocedure。PL/pgSQL 函数示例如下：

```
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、一个双精度、一个可的整数数和可文本数。第一个参数是 `rast1` 中个格像元的 (无格数据型如何)。第二个参数是 `rast2` 中的个格像元。第三个参数是当前理元的位置，格式 “{x,y}”。第四个参数指示 `ST_MapAlgebraFct` 的所有剩余参数均 `tworastuserfunc`。

将 `regprodedure` 参数 SQL 函数需要完整的函数名，然后 `regprocedure` 型。要将上述示例 PL/pgSQL 函数作参数，参数的 SQL ：

```
'simple_function(double precision, double precision, integer[], text[])':regprocedure
```

注意，参数包含函数名称、函数参数型、名称和参数型周的引号以及 `regprocedure` 的制。

`tworastuserfunc` 的第四个参数是可文本数。任何 `ST_MapAlgebraFct` 用的所有尾随文本参数都会到指定的 `tworastuserfunc`，并包含在 `userargs` 参数中。



#### Note

有关 VARIADIC 关字的更多信息，参 PostgreSQL 文档和 [方言 \(SQL\) 函数](#) 的“具有可参数数量的 SQL 函数”部分。



#### Note

无论您是否将任何参数用函数行理，`text[]` 参数在 `tworastuserfunc` 都是必需的。

可用性: 2.0.0

示例：将格作个的波段覆盖在画布上

```
-- define our user defined function --
CREATE OR REPLACE FUNCTION raster_mapalgebra_union(
  rast1 double precision,
  rast2 double precision,
  pos integer[],
  VARIADIC userargs text[]
)
  RETURNS double precision
  AS $$
  DECLARE
  BEGIN
    CASE
      WHEN rast1 IS NOT NULL AND rast2 IS NOT NULL THEN
        RETURN ((rast1 + rast2)/2.);
      WHEN rast1 IS NULL AND rast2 IS NULL THEN
        RETURN NULL;
```

```

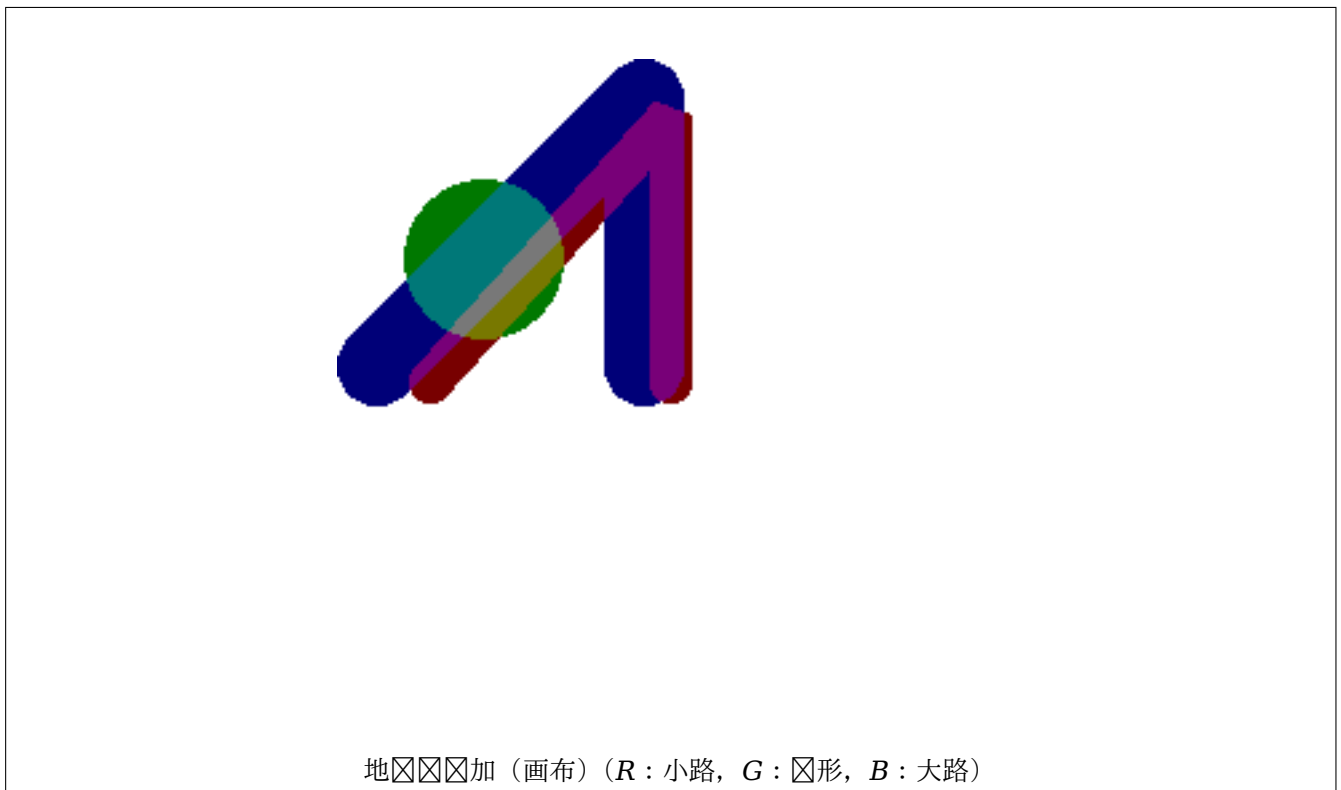
        WHEN rast1 IS NULL THEN
            RETURN rast2;
        ELSE
            RETURN rast1;
    END CASE;

    RETURN NULL;
END;
$$ LANGUAGE 'plpgsql' IMMUTABLE COST 1000;

-- prep our test table of rasters
DROP TABLE IF EXISTS map_shapes;
CREATE TABLE map_shapes(rid serial PRIMARY KEY, rast raster, bnum integer, descrip text);
INSERT INTO map_shapes(rast,bnum, descrip)
WITH mygeoms
  AS ( SELECT 2 As bnum, ST_Buffer(ST_Point(90,90),30) As geom, 'circle' As descrip
      UNION ALL
      SELECT 3 AS bnum,
            ST_Buffer(ST_GeomFromText('LINESTRING(50 50,150 150,150 50)'), 15) As geom, ←
            'big road' As descrip
      UNION ALL
      SELECT 1 As bnum,
            ST_Translate(ST_Buffer(ST_GeomFromText('LINESTRING(60 50,150 150,150 50)'), ←
            8,'join=bevel'), 10,-6) As geom, 'small road' As descrip
    ),
-- define our canvas to be 1 to 1 pixel to geometry
canvas
  AS ( SELECT ST_AddBand(ST_MakeEmptyRaster(250,
      250,
      ST_XMin(e)::integer, ST_YMax(e)::integer, 1, -1, 0, 0 ) , '8BUI'::text,0) As rast
      FROM (SELECT ST_Extent(geom) As e,
            Max(ST_SRID(geom)) As srid
            from mygeoms
            ) As foo
    )
-- return our rasters aligned with our canvas
SELECT ST_AsRaster(m.geom, canvas.rast, '8BUI', 240) As rast, bnum, descrip
  FROM mygeoms AS m CROSS JOIN canvas
UNION ALL
SELECT canvas.rast, 4, 'canvas'
FROM canvas;

-- Map algebra on single band rasters and then collect with ST_AddBand
INSERT INTO map_shapes(rast,bnum,descrip)
SELECT ST_AddBand(ST_AddBand(rasts[1], rasts[2]),rasts[3]), 4, 'map bands overlay fct union ←
  (canvas)'
  FROM (SELECT ARRAY(SELECT ST_MapAlgebraFct(m1.rast, m2.rast,
      'raster_mapalgebra_union(double precision, double precision, integer[], text[]) ←
      '::regprocedure, '8BUI', 'FIRST')
      FROM map_shapes As m1 CROSS JOIN map_shapes As m2
      WHERE m1.descrip = 'canvas' AND m2.descrip <
> 'canvas' ORDER BY m2.bnum) As rasts) As foo;

```



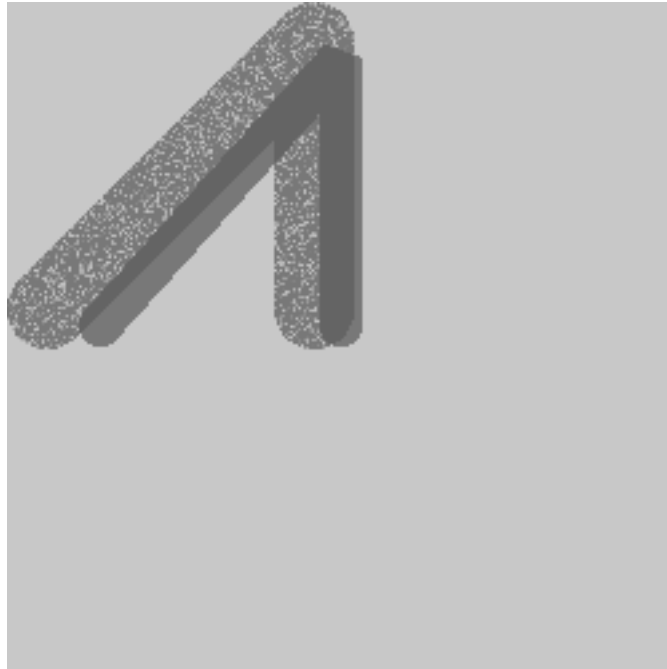
需要⊗外参数的用⊗定⊗函数

```
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)';
```



用 `ST_MapAlgebraFctNgb` 使用 `ST_MapAlgebra` 外参数和来自同一栅格的不同波段定义 `ST_MapAlgebra`

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 `ST_MapAlgebra` 定义 `ST_MapAlgebra` 的 PostgreSQL 函数映射代数最近邻。返回一个栅格，其 `ST_MapAlgebra` 是涉及 `ST_MapAlgebra` 输入栅格波段 `ST_MapAlgebra` 的 `ST_MapAlgebra` 域的 PLPGSQL 用 `ST_MapAlgebra` 函数的 `ST_MapAlgebra` 果。

#### Synopsis

raster **ST\_MapAlgebraFctNgb**(raster rast, integer band, text pixeltyp, integer ngbwidth, integer ngbheight, regprocedure onerastngbuserfunc, text nodatamode, text[] VARIADIC args);

描述



#### 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 `ST_MapAlgebra` 是涉及 `ST_MapAlgebra` 输入栅格波段 `ST_MapAlgebra` 的 `ST_MapAlgebra` 域的 PLPGSQL 用 `ST_MapAlgebra` 函数的 `ST_MapAlgebra` 果。用 `ST_MapAlgebra` 函数将像素 `ST_MapAlgebra` 的 `ST_MapAlgebra` 域作 `ST_MapAlgebra` 数字 `ST_MapAlgebra`，`ST_MapAlgebra` 于每个像素，返回用 `ST_MapAlgebra` 函数的 `ST_MapAlgebra` 果，用函数 `ST_MapAlgebra` 果替 `ST_MapAlgebra` 当前 `ST_MapAlgebra` 像素的像素 `ST_MapAlgebra`。



**rast** 估算函数的栅格。

**band** 要估算的栅格的波段号。默认为 1。

**pixeltype** 输出栅格的像素类型。必须是 `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` 的像素类型。如果结果大于像素类型允许的，会被截断。

**ngbwidth** 区域的宽度（以元格位）。

**ngbheight** 区域的高度（以元格位）。

**onerastngbuserfunc** PLPGSQL/psql 用函数用于栅格个波段的区域像素。第一个元素是表示矩形像素区域的二数字数

**nodatamode** 定义要无数据或 NULL 的区域像素函数的

‘ignore’：估算忽略在区域中遇到的任何 NODATA 标志必送到用回函数，并且用函数决定如何忽略它。

“NULL”：在区域中遇到的任何 NODATA 都将致果像素 NULL——在种情况下将跳用回函数。

“value”：在区域中遇到的任何 NODATA 都将替参考像素（区域中心的像素）。注意，如果 NODATA，行与“NULL”相同（于受影响的区域）

**args** 要用的函数的参数。

可用性: 2.0.0

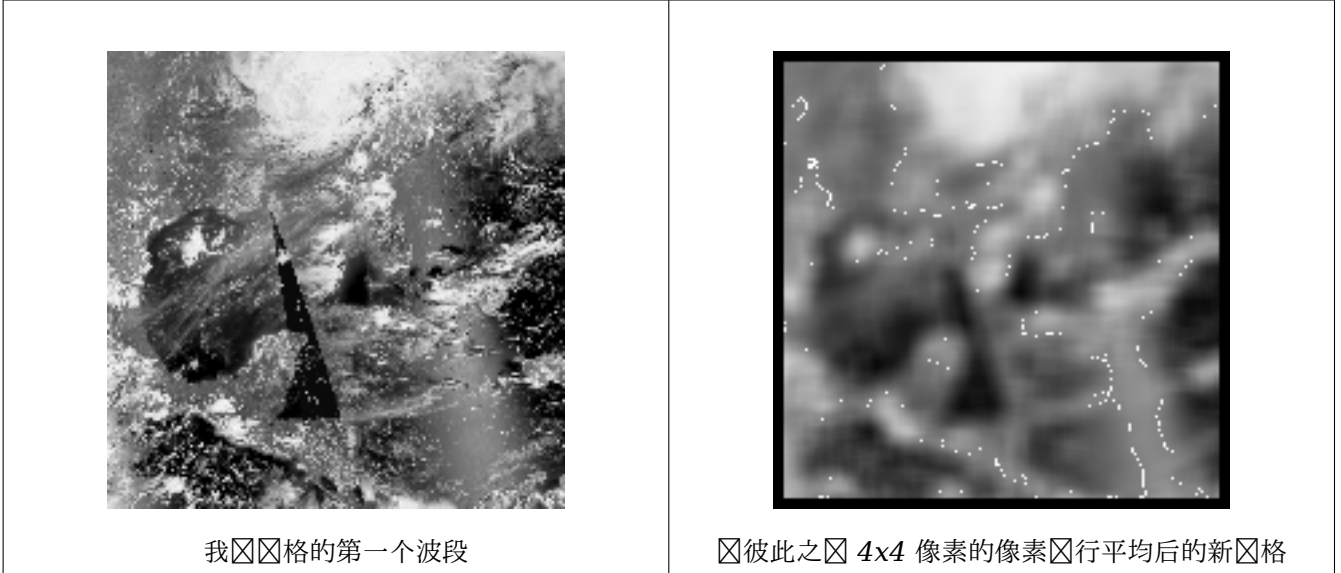
示例

示例利用 [http://trac.osgeo.org/gdal/wiki/frmts\\_wtkraster.html](http://trac.osgeo.org/gdal/wiki/frmts_wtkraster.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.
--
CREATE OR REPLACE FUNCTION rast_avg(matrix float[][][], nodatamode text, variadic args text ←
[])
RETURNS float AS
$$
DECLARE
    _matrix float[][][];
    x1 integer;
    x2 integer;
    y1 integer;
    y2 integer;
    sum float;
BEGIN
    _matrix := matrix;
    sum := 0;
    FOR x in array_lower(matrix, 1)..array_upper(matrix, 1) LOOP
        FOR y in array_lower(matrix, 2)..array_upper(matrix, 2) LOOP
            sum := sum + _matrix[x][y];
        END LOOP;
    END LOOP;
    RETURN (sum*1.0/(array_upper(matrix,1)*array_upper(matrix,2) ))::integer ;
END;
$$
LANGUAGE 'plpgsql' IMMUTABLE COST 1000;

-- now we apply to our raster averaging pixels within 2 pixels of each other in X and Y ←
direction --
```

```
SELECT ST_MapAlgebraFctNgb(rast, 1, '8BUI', 4,4,
    'rast_avg(float[][][], text, text[])'::regprocedure, 'NULL', NULL) As nn_with_border
FROM katrinas_rescaled
limit 1;
```



#### 相关信息

[ST\\_MapAlgebraFct](#), [ST\\_MapAlgebraExpr](#), [ST\\_Rescale](#)

### 11.12.12 ST\_Reclass

**ST\_Reclass** — 创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);
```

#### 描述

通过输入栅格 (**rast**) 用由 **reclassexpr** 定义的有效 PostgreSQL 代数算来创建新栅格。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、宽度和高度。未指定的波段将保持不变。有关有效重分区表达式的说明, 参见 [reclassarg](#)。

新栅格的波段将具有 **Pixeltype** 的像素类型。如果输入 **reclassargset**, 每个 **reclassarg** 定义生成的每个行的行。

可用性: 2.0.0

基本示例

从原始栅格新建一个新栅格，其中波段 2 从 8BUI 变为 4BUI，并且 101-254 中的所有值都置为 nodata。

```
ALTER TABLE dummy_rast ADD COLUMN reclass_rast raster;
UPDATE dummy_rast SET reclass_rast = ST_Reclass(rast,2,'0-87:1-10, 88-100:11-15, ←
    101-254:0-0', '4BUI',0) WHERE rid = 2;

SELECT i as col, j as row, ST_Value(rast,2,i,j) As origval,
    ST_Value(reclass_rast, 2, i, j) As reclassval,
    ST_Value(reclass_rast, 2, i, j, false) As reclassval_include_nodata
FROM dummy_rast CROSS JOIN generate_series(1, 3) AS i CROSS JOIN generate_series(1,3) AS j
WHERE rid = 2;
```

col	row	origval	reclassval	reclassval_include_nodata
1	1	78	9	9
2	1	98	14	14
3	1	122		0
1	2	96	14	14
2	2	118		0
3	2	180		0
1	3	99	15	15
2	3	112		0
3	3	169		0

示例：高效使用多个重新分类参数

从原始栅格新建一个新栅格，其中波段 1、2、3 分别变为 1BB、4BUI、4BUI 并重新分类。注意，使用可变的 reclassarg 参数，参数可以将不确定数量的重新分类参数作为输入（理论上与您有的波段一样多）

```
UPDATE dummy_rast SET reclass_rast =
    ST_Reclass(rast,
        ROW(2,'0-87]:1-10, (87-100]:11-15, (101-254]:0-0', '4BUI',NULL)::reclassarg,
        ROW(1,'0-253]:1, 254:0', '1BB', NULL)::reclassarg,
        ROW(3,'0-70]:1, (70-86:2, [86-150):3, [150-255:4', '4BUI', NULL)::reclassarg
    ) WHERE rid = 2;

SELECT i as col, j as row,ST_Value(rast,1,i,j) As ov1, ST_Value(reclass_rast, 1, i, j) As ←
    rv1,
    ST_Value(rast,2,i,j) As ov2, ST_Value(reclass_rast, 2, i, j) As rv2,
    ST_Value(rast,3,i,j) As ov3, ST_Value(reclass_rast, 3, i, j) As rv3
FROM dummy_rast CROSS JOIN generate_series(1, 3) AS i CROSS JOIN generate_series(1,3) AS j
WHERE rid = 2;
```

col	row	ov1	rv1	ov2	rv2	ov3	rv3
1	1	253	1	78	9	70	1
2	1	254	0	98	14	86	3
3	1	253	1	122	0	100	3
1	2	253	1	96	14	80	2
2	2	254	0	118	0	108	3
3	2	254	0	180	0	162	4
1	3	250	1	99	15	90	3
2	3	254	0	112	0	108	3
3	3	254	0	169	0	175	4

示例：高将波段 **32BF** 格映射到多个可波段

从只有一个 32bf 波段的格建一个新的 3 波段（8BUI、8BUI、8BUI 可看格）

```
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\_Union

`ST_Union` — 将一格切片的并集返回由 1 个或多个波段成的格。

#### 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);
```

#### 描述

将一格切片的并集返回由至少一个波段成的格。生成的格范是整个格集的范。在交集的情况下，果由 `uniontype` 定，它是以下之一：LAST（默）、FIRST、MIN、MAX、COUNT、SUM、MEAN、RANGE。



#### Note

了合并格，它必具有相同的格方式。使用 [ST\\_SameAlignment](#) 和 [ST\\_NotSameAlignmentReason](#) 取更多格信息和帮助。解决格的一种方法是使用 [ST\\_Resample](#) 并使用相同的参考格行。

可用性: 2.0.0

增: 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)` 体。

增: 2.1.0 `ST_Union(rast)` (体 1) 合并所有入格的所有波段。PostGIS 的早期版本采用第一个波段。

增: 2.1.0 `ST_Union(rast, uniontype)` (体 4) 合并所有入格的所有波段。

示例：重建波段分格

```
-- this creates a single band from first band of raster tiles
-- that form the original file system tile
SELECT filename, ST_Union(rast,1) As file_rast
FROM sometable WHERE filename IN('dem01', 'dem02') GROUP BY filename;
```

示例：返回多波段栅格，它是与几何体相交的栅格的并集

```
-- this creates a multi band raster collecting all the tiles that intersect a line
-- Note: In 2.0, this would have just returned a single band raster
-- , new union works on all bands by default
-- this is equivalent to unionarg: ARRAY[ROW(1, 'LAST'), ROW(2, 'LAST'), ROW(3, 'LAST')]:: ←
  unionarg[]
SELECT ST_Union(rast)
FROM aeriāls.boston
WHERE ST_Intersects(rast, ST_GeomFromText('LINESTRING(230486 887771, 230500 88772)',26986) ←
);
```

示例：返回多波段栅格，它是与几何体相交的栅格的并集

如果我只需要波段的子集或者想要更改波段的顺序，这里我使用更的方法

```
-- this creates a multi band raster collecting all the tiles that intersect a line
SELECT ST_Union(rast,ARRAY[ROW(2, 'LAST'), ROW(1, 'LAST'), ROW(3, 'LAST')]::unionarg[])
FROM aeriāls.boston
WHERE ST_Intersects(rast, ST_GeomFromText('LINESTRING(230486 887771, 230500 88772)',26986) ←
);
```

相关信息

[unionarg](#), [ST\\_Envelope](#), [ST\\_ConvexHull](#), [ST\\_Clip](#), [ST\\_Union](#)

## 11.13 内置地理代数回函数

### 11.13.1 ST\_Distinct4ma

ST\_Distinct4ma — 栅格地理函数，用于计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);
```

## 描述

计算像素域中唯一像素的数量。

**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

增加: 2.1.0 添加格式 2

## 示例

```
SELECT
  rid,
  st_value(
    st_mapalgebrafctngb(rast, 1, NULL, 1, 1, 'st_distinct4ma(float[],text,text[])':: regprocedure, 'ignore', NULL), 2, 2
  )
FROM dummy_rast
WHERE rid = 2;
  rid | st_value
-----+-----
    2 |      3
(1 row)
```

## 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函数。

**Synopsis**

```
double precision ST_InvDistWeight4ma(double precision[][][] value, integer[][] pos, text[] VARIADIC userargs);
```

## 描述

使用反距离加权方法计算像素的插值。

有两个可选参数可以通过 `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（下面方程中的变量  $k$ ）。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。

基本反距离加权方程：

$$\hat{z}(x_0) = \frac{\sum_{j=1}^m z(x_j) d_{ij}^{-k}}{\sum_{j=1}^m d_{ij}^{-k}}$$

$k$  = 功率因数, 0 到 1 之间的实数



### Note

该函数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。

可用性：2.1.0

## 示例

```
-- NEEDS EXAMPLE
```

## 相关信息

[ST\\_MapAlgebra \(callback function version\)](#), [ST\\_MinDist4ma](#)

### 11.13.3 ST\_Max4ma

`ST_Max4ma` — 计算区域中最大像素的栅格处理函数。

## Synopsis

```
float8 ST_Max4ma(float8[][] matrix, text nodatamode, text[] VARIADIC args);
double precision ST_Max4ma(double precision[][][] value, integer[][] pos, text[] VARIADIC userargs);
```

## 描述

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

```
SELECT
  rid,
  st_value(
    st_mapalgebrafctngb(rast, 1, NULL, 1, 1, 'st_max4ma(float[][],text,text[])':: ↵
      regprocedure, 'ignore', NULL), 2, 2
  )
FROM dummy_rast
WHERE rid = 2;
  rid | st_value
-----+-----
    2 |      254
(1 row)
```

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);
```

描述

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。



**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: 格式 1

```
SELECT
  rid,
  st_value(
    st_mapalgebrafctngb(rast, 1, '32BF', 1, 1, 'st_mean4ma(float[][],text,text[])':: ↵
      regprocedure, 'ignore', NULL), 2, 2
  )
FROM dummy_rast
WHERE rid = 2;
rid |      st_value
-----+-----
  2 | 253.222229003906
(1 row)
```

示例: 格式 2

```
SELECT
  rid,
  st_value(
    ST_MapAlgebra(rast, 1, 'st_mean4ma(double precision[][][], integer[][], text ↵
      [])'::regprocedure, '32BF', 'FIRST', NULL, 1, 1)
    , 2, 2)
FROM dummy_rast
WHERE rid = 2;
rid |      st_value
-----+-----
  2 | 253.222229003906
(1 row)
```

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);
```

#### 描述

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。



#### 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

```
SELECT
  rid,
  st_value(
    st_mapalgebrafctngb(rast, 1, NULL, 1, 1, 'st_min4ma(float[][],text,text[])'::
      regprocedure, 'ignore', NULL), 2, 2
  )
FROM dummy_rast
WHERE rid = 2;
  rid | st_value
-----+-----
    2 |      250
(1 row)
```

#### 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);
```

#### 描述

返回感兴趣的像素与域中具有最近像素之最短距离（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的有用性。当社区人口稀少，此功能特有用。



#### Note

函数是一个的回函数，用作 [ST\\_MapAlgebra \(callback function version\)](#) (回函数版本) 的回参数。

可用性：2.1.0

#### 示例

```
-- NEEDS EXAMPLE
```

#### 相关信息

[ST\\_MapAlgebra \(callback function version\)](#), [ST\\_InvDistWeight4ma](#)

### 11.13.7 ST\_Range4ma

ST\_Range4ma — 算域中像素范的格理函数。

#### Synopsis

```
float8 ST_Range4ma(float8[][] matrix, text nodatamode, text[] VARIADIC args);
double precision ST_Range4ma(double precision[][][] value, integer[][] pos, text[] VARIADIC user-args);
```

## 描述

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

增加: 2.1.0 添加格式 2

## 示例

```
SELECT
  rid,
  st_value(
    st_mapalgebrafctngb(rast, 1, NULL, 1, 1, 'st_range4ma(float[[[]],text,text[])':: ↵
    regprocedure, 'ignore', NULL), 2, 2
  )
FROM dummy_rast
WHERE rid = 2;
  rid | st_value
-----+-----
    2 |         4
(1 row)
```

## 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` — 计算域中像素的准偏差的格理函数。

## Synopsis

float8 **ST\_StdDev4ma**(float8[][] matrix, text nodatamode, text[] VARIADIC args);  
 double precision **ST\_StdDev4ma**(double precision[][][] value, integer[][] pos, text[] VARIADIC user-args);

### 描述

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

```
SELECT
  rid,
  st_value(
    st_mapalgebrafctngb(rast, 1, '32BF', 1, 1, 'st_stddev4ma(float[],text,text[])'::
      regprocedure, 'ignore', NULL), 2, 2
  )
FROM dummy_rast
WHERE rid = 2;
 rid |      st_value
-----+-----
   2 | 1.30170822143555
(1 row)
```

### 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** — 格函数，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 userargs);

### 描述

计算像素域中所有像素的和。

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

```
SELECT
  rid,
  st_value(
    st_mapalgebrafctngb(rast, 1, '32BF', 1, 1, 'st_sum4ma(float[][],text,text[])':: ←
      regprocedure, 'ignore', NULL), 2, 2
  )
FROM dummy_rast
WHERE rid = 2;
 rid | st_value
-----+-----
   2 |    2279
(1 row)
```

### 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 栅格处理：DEM（高程）

### 11.14.1 ST\_Aspect

ST\_Aspect — 返回高程栅格波段的坡向（默数以度为单位）。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);
```

#### 描述

返回高程栅格波段的坡向（默数以度为单位）。利用代数并将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，从北向测量。

当 units = DEGREES 时，值介于 0 到 360 度之间（从北向测量）。

如果像素的斜率为零，则像素的横比-1。



#### Note

有关坡度、坡向和山体阴影的更多信息，参见 [ESRI - 山体阴影的工作原理](#) 和 [ERDAS 指南 - 坡向像](#)。

可用性: 2.0.0

增加: 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，返回默认为度数

示例：格式 1

```
WITH foo AS (
  SELECT ST_SetValues(
    ST_AddBand(ST_MakeEmptyRaster(5, 5, 0, 0, 1, -1, 0, 0, 0), 1, '32BF', 0, -9999),
    1, 1, 1, ARRAY[
      [1, 1, 1, 1, 1],
      [1, 2, 2, 2, 1],
      [1, 2, 3, 2, 1],
      [1, 2, 2, 2, 1],
      [1, 1, 1, 1, 1]
    ]::double precision[[]]
  ) AS rast
)
SELECT
  ST_DumpValues(ST_Aspect(rast, 1, '32BF'))
FROM foo
```

```

-----
-----
(1,"{{315,341.565063476562,0,18.4349479675293,45},{288.434936523438,315,0,45,71.5650482177734},{270
2227,180,161.565048217773,135}}")
(1 row)

```

示例：格式 **2**

覆盖范围示例的完整示例。此示例适用于 PostgreSQL 9.1 或更高版本。

```

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 precision azimuth=315, double precision altitude=45, double precision max\_bright=255, double precision scale=1.0, boolean interpolate\_nodata=FALSE);



## 描述

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) 是垂直像素与水平像素的比率。英尺: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\_InvDistWeightMap 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

增加: 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: 格式 1

```
WITH foo AS (
  SELECT ST_SetValues(
    ST_AddBand(ST_MakeEmptyRaster(5, 5, 0, 0, 1, -1, 0, 0, 0), 1, '32BF', 0, -9999),
    1, 1, 1, ARRAY[
      [1, 1, 1, 1, 1],
      [1, 2, 2, 2, 1],
      [1, 2, 3, 2, 1],
      [1, 2, 2, 2, 1],
      [1, 1, 1, 1, 1]
    ]::double precision[5][5]
  ) AS rast
)
SELECT
  ST_DumpValues(ST_Hillshade(rast, 1, '32BF'))
FROM foo
```

```
-----
-----
(1,"{NULL,NULL,NULL,NULL,NULL},{NULL,251.32763671875,220.749786376953,147.224319458008,↵
  NULL},{NULL,220.749786376953,180.312225341797,67.7497863769531,NULL},{NULL ↵
  ,147.224319458008
,67.7497863769531,43.1210060119629,NULL},{NULL,NULL,NULL,NULL,NULL}")
(1 row)
```

示例：格式 2

覆盖范围示例的完整示例。此示例适用于 PostgreSQL 9.1 或更高版本。

```
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 );
```

#### 描述

通过指定区域的最小值去最大值来算数字高程模型 (DEM) 的“粗糙度”。

可用性：2.1.0

#### 示例

```
-- needs examples
```

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` — 返回高程栅格的坡度（默认以度为单位）。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);
```

#### 描述

返回高程栅格的坡度（默认以度为单位）。利用代数并将斜率方程用于相邻像素。

`units` 表示斜率的单位。可能的单位：弧度、度（默认）、百分比。

比例（`scale`）是垂直单位与水平单位的比率。用于英尺:LatLon 使用 `scale=370400`，用于米:LatLon 使用 `scale=111120`。

如果 `interpolate_nodata` 为 `TRUE`，则在计算表面坡度之前，将使用 [ST\\_InvDistWeight4mask](#) 输入栅格中的 `NODATA` 像素进行插值。



#### 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 - 坡度像](#)。

可用性: 2.0.0

增加: 2.1.0 使用 `ST_MapAlgebra()` 并添加可选项 `units`、`scale`、`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：格式 1

```
WITH foo AS (
  SELECT ST_SetValues(
    ST_AddBand(ST_MakeEmptyRaster(5, 5, 0, 0, 1, -1, 0, 0, 0), 1, '32BF', 0, -9999),
    1, 1, 1, ARRAY[
      [1, 1, 1, 1, 1],
      [1, 2, 2, 2, 1],
      [1, 2, 3, 2, 1],
      [1, 2, 2, 2, 1],
      [1, 1, 1, 1, 1]
    ]::double precision[]
  ) AS rast
)
SELECT
  ST_DumpValues(ST_Slope(rast, 1, '32BF'))
FROM foo
```

```
st_dumpvalues
```

```
(1,"{{10.0249881744385,21.5681285858154,26.5650520324707,21.5681285858154,10.0249881744385},{21.5681285858154,26.5650520324707,36.8698959350586,0,36.8698959350586,26.5650520324707},{21.5681285858154,35.26438905681285858154,26.5650520324707,21.5681285858154,10.0249881744385}}")
(1 row)
```

示例：格式 **2**

覆盖范围 `ST_Tile` 的完整示例。此 `ST_Tile` 适用于 PostgreSQL 9.1 或更高版本。

```
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 `ST_TPI` 算出的地形位置指数 (Topographic Position Index) 的 `ST_Geometry`。

## Synopsis

raster **ST\_TPI**(raster rast, integer nband, raster customextent, text pixeltype="32BF" , boolean interpolate\_nodata=FALSE );

### 描述

计算地形位置指数，其定义为半径为 1 的焦点平均高度减去中心像元。



#### Note

此函数支持焦点平均半径为 1。

可用性 : 2.1.0

### 示例

```
-- needs examples
```

### 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 — 返回具有计算的地形粗糙性指数的栅格。

## Synopsis

raster **ST\_TRI**(raster rast, integer nband, raster customextent, text pixeltype="32BF" , boolean interpolate\_nodata=FALSE );

### 描述

地形粗糙性指数是通过将中心像素与其相邻像素进行比较、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为 1。

可用性 : 2.1.0

### 示例

```
-- needs examples
```

相关信息

[ST\\_MapAlgebra \(callback function version\)](#), [ST\\_Roughness](#), [ST\\_TPI](#), [ST\\_Slope](#), [ST\\_HillShade](#), [ST\\_Aspect](#)

## 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);
```

#### 描述

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，因此已更改为 box3d。

#### 示例

```
SELECT
  rid,
  Box3D(rast) AS rastbox
FROM dummy_rast;
```

rid	rastbox
1	BOX3D(0.5 0.5 0,20.5 60.5 0)
2	BOX3D(3427927.75 5793243.5 0,3427928 5793244 0)

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。对于多边形形状和非斜栅格，输出与 ST\_Envelope 相同的结果，因此对于非多边形形状或斜栅格有用。

#### Synopsis

```
geometry ST_ConvexHull(raster rast);
```

## 描述

返回栅格的凸包几何形状，包括 NoDataBandValue 像素。对于多边形形状和非斜栅格，它或多或少会生成与 ST\_Envelope 相同的结果，因此它不适用于斜栅格。



### 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，因此结果与不将坐标向下取整的 ST\_ConvexHull 稍有不同。

## 示例

有关此函数，请参考 [PostGIS Raster Specific](#)。

```
-- Note envelope and convexhull are more or less the same
SELECT ST_AsText(ST_ConvexHull(rast)) As convhull,
       ST_AsText(ST_Envelope(rast)) As env
FROM dummy_rast WHERE rid=1;
```

```
convhull | env
-----+-----
POLYGON((0.5 0.5,20.5 0.5,20.5 60.5,0.5 60.5,0.5 0.5)) | POLYGON((0 0,20 0,20 60,0 60,0 0) ←
)
```

```
-- now we skew the raster
-- note how the convex hull and envelope are now different
SELECT ST_AsText(ST_ConvexHull(rast)) As convhull,
       ST_AsText(ST_Envelope(rast)) As env
FROM (SELECT ST_SetRotation(rast, 0.1, 0.1) As rast
      FROM dummy_rast WHERE rid=1) As foo;
```

```
convhull | env
-----+-----
POLYGON((0.5 0.5,20.5 1.5,22.5 61.5,2.5 60.5,0.5 0.5)) | POLYGON((0 0,22 0,22 61,0 61,0 0) ←
)
```

## 相关信息

[ST\\_Envelope](#), [ST\\_MinConvexHull](#), [ST\\_ConvexHull](#), [ST\\_AsText](#)

## 11.15.3 ST\_DumpAsPolygons

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，则波段号为默认值 1。

### Synopsis

```
setof geomval ST_DumpAsPolygons(raster rast, integer band_num=1, boolean exclude_nodata_value=TRUE)
```

## 描述

`ST_DumpAsPolygon` 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，它创建新行。例如，它可用于将栅格展成多个 POLYGONS/MULTIPOLYGONS。

更改 3.3.0，禁用索引和修复以提高性能。可能会致无效的几何图形。

可用性：需要 GDAL 1.7 或更高版本。

Note!

### Note

如果没有波段置数据，不会返回具有值的像素，除非 `except_nodata_value=false`。

Note!

### Note

如果您只关心栅格中具有固定值的像素数，使用 `ST_ValueCount` 速度更快。

Note!

### Note

与 `ST_PixelAsPolygons` 不同，在 `ST_PixelAsPolygons` 中，无论像素如何，都会为每个像素返回一个几何图形。

## 示例

```
-- this syntax requires PostgreSQL 9.3+
SELECT val, ST_AsText(geom) As geomwkt
FROM (
  SELECT dp.*
  FROM dummy_rast, LATERAL ST_DumpAsPolygons(rast) AS dp
  WHERE rid = 2
) As foo
WHERE val BETWEEN 249 and 251
ORDER BY val;
```

val	geomwkt
249	POLYGON((3427927.95 5793243.95,3427927.95 5793243.85,3427928 5793243.85,3427928 5793243.95,3427927.95 5793243.95))
250	POLYGON((3427927.75 5793243.9,3427927.75 5793243.85,3427927.8 5793243.85,3427927.8 5793243.9,3427927.75 5793243.9))
250	POLYGON((3427927.8 5793243.8,3427927.8 5793243.75,3427927.85 5793243.75,3427927.85 5793243.8, 3427927.8 5793243.8))
251	POLYGON((3427927.75 5793243.85,3427927.75 5793243.8,3427927.8 5793243.8,3427927.8 5793243.85,3427927.75 5793243.85))

## 相关信息

[geomval](#), [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);
```

#### 描述

返回栅格范围的多边形表示形式（以 srid 定义的空坐标系位表示）。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

```
SELECT rid, ST_AsText(ST_Envelope(rast)) As envgeomwkt
FROM dummy_rast;
```

rid	envgeomwkt
1	POLYGON((0 0,20 0,20 60,0 60,0 0))
2	POLYGON((3427927 5793243,3427928 5793243, 3427928 5793244,3427927 5793244, 3427927 5793243))

#### 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);
```

#### 描述

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



## Synopsis

geometry **ST\_Polygon**(raster rast, integer band\_num=1);

### 描述

改☒ 3.3.0, 禁用☒☒和修复以提高性能。可能会☒致无效的几何☒形。

可用性 : 0.1.6 需要 GDAL 1.7 或更高版本。

增☒ : 2.1.0 提高速度 (完全基于 C) 并且确保返回的多☒形有效。

更改 : 2.1.0 在之前的版本中有☒会返回多☒形, 更改☒始☒返回多多☒形。

### 示例

```
-- by default no data band value is 0 or not set, so polygon will return a square polygon
SELECT ST_AsText(ST_Polygon(rast)) As geomwkt
FROM dummy_rast
WHERE rid = 2;

geomwkt
-----
MULTIPOLYGON(((3427927.75 5793244,3427928 5793244,3427928 5793243.75,3427927.75 ←
  5793243.75,3427927.75 5793244)))

-- now we change the no data value of first band
UPDATE dummy_rast SET rast = ST_SetBandNoDataValue(rast,1,254)
WHERE rid = 2;
SELECT rid, ST_BandNoDataValue(rast)
from dummy_rast where rid = 2;

-- ST_Polygon excludes the pixel value 254 and returns a multipolygon
SELECT ST_AsText(ST_Polygon(rast)) As geomwkt
FROM dummy_rast
WHERE rid = 2;

geomwkt
-----
MULTIPOLYGON(((3427927.9 5793243.95,3427927.85 5793243.95,3427927.85 5793244,3427927.9 ←
  5793244,3427927.9 5793243.95)),((3427928 5793243.85,3427928 5793243.8,3427927.95 ←
  5793243.8,3427927.95 5793243.85,3427927.9 5793243.85,3427927.9 5793243.9,3427927.9 ←
  5793243.95,3427927.95 5793243.95,3427928 5793243.95,3427928 5793243.85)),((3427927.8 ←
  5793243.75,3427927.75 5793243.75,3427927.75 5793243.8,3427927.75 5793243.85,3427927.75 ←
  5793243.9,3427927.75 5793244,3427927.8 5793244,3427927.8 5793243.9,3427927.8 ←
  5793243.85,3427927.85 5793243.85,3427927.85 5793243.8,3427927.85 5793243.75,3427927.8 ←
  5793243.75)))

-- Or if you want the no data value different for just one time
SELECT ST_AsText(
  ST_Polygon(
    ST_SetBandNoDataValue(rast,1,252)
  )
) As geomwkt
FROM dummy_rast
WHERE rid =2;

geomwkt
```

```
-----
MULTIPOLYGON(((3427928 5793243.85,3427928 5793243.8,3427928 5793243.75,3427927.85 ↵
  5793243.75,3427927.8 5793243.75,3427927.8 5793243.8,3427927.75 5793243.8,3427927.75 ↵
  5793243.85,3427927.75 5793243.9,3427927.75 5793244,3427927.8 5793244,3427927.85 ↵
  5793244,3427927.9 5793244,3427928 5793244,3427928 5793243.95,3427928 5793243.85) ↵
  ,(3427927.9 5793243.9,3427927.9 5793243.85,3427927.95 5793243.85,3427927.95 ↵
  5793243.9,3427927.9 5793243.9)))
```

相关信息

[ST\\_Value](#), [ST\\_DumpAsPolygons](#)

## 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 );
```

描述

&& ☒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

```
SELECT A.rid As a_rid, B.rid As b_rid, A.rast && B.rast As intersect
FROM dummy_rast AS A CROSS JOIN dummy_rast AS B LIMIT 3;
```

```
a_rid | b_rid | intersect
-----+-----+-----
  2 |    2 | t
  2 |    3 | f
  2 |    1 | f
```

### 11.16.2 &<

&< — 如果 A 的☒界框位于 B 的左☒，☒返回 TRUE。

## Synopsis

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

### 描述

`&<` 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

```
SELECT A.rid As a_rid, B.rid As b_rid, A.rast &< B.rast As overleft
FROM dummy_rast AS A CROSS JOIN dummy_rast AS B;
```

a_rid	b_rid	overleft
2	2	t
2	3	f
2	1	f
3	2	t
3	3	t
3	1	f
1	2	t
1	3	t
1	1	t

## 11.16.3 &>

`&>` — 如果 A 的边界框位于 B 的右边界，返回 TRUE。

## Synopsis

```
boolean &>( raster A , raster B );
```

### 描述

`&>` 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

```
SELECT A.rid As a_rid, B.rid As b_rid, A.rast &
> B.rast As overright
FROM dummy_rast AS A CROSS JOIN dummy_rast AS B;
```

a_rid	b_rid	overright
2	2	t
2	3	t
2	1	t
3	2	f
3	3	t
3	1	f
1	2	f
1	3	t
1	1	t

### 11.16.4 =

= — 如果 A 的栅格与 B 的栅格相同，返回 TRUE。使用双精度栅格。

#### Synopsis

```
boolean =( raster A , raster B );
```

描述

= 运算符在栅格 A 的栅格与栅格 B 的栅格相同返回 TRUE。PostgreSQL 使用栅格定义的 =、< 和 > 运算符来对栅格的内部排序和比较（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区。

可用性：2.1.0

相关信息

`~=`

### 11.16.5 @

@ — 如果 A 的栅格包含在 B 的栅格中，返回 TRUE。使用双精度栅格。

## Synopsis

```
boolean @( raster A , raster B );
boolean @( geometry A , raster B );
boolean @( raster B , geometry A );
```

### 描述

@ 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 );
```

### 描述

~= 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波段的栅格并建多波段栅格

```
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  $\square$  界框包含 B 的  $\square$  界框,  $\square$  返回 TRUE。使用双精度  $\square$  界框。

### Synopsis

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

描述

如果  $\square$  格/几何 A 的  $\square$  界框包含  $\square$  格/几何 B 的  $\square$  界框,  $\square$  ~  $\square$  算符返回 TRUE。



#### Note

$\square$  操作符将使用  $\square$  格上的空  $\square$  索引。

可用性: 2.0.0

相关信息

[@](#)

## 11.17 $\square$ 格和 $\square$ 格波段空 $\square$ 关系

### 11.17.1 ST\_Contains

ST\_Contains — 如果  $\square$  格 rastB 中没有点位于  $\square$  格 rastA 的外部且 rastB 的内部至少有一个点位于 rastA 的内部,  $\square$  返回 true。

### Synopsis

```
boolean ST_Contains( raster rastA , integer nbandA , raster rastB , integer nbandB );  
boolean ST_Contains( raster rastA , raster rastB );
```



## 描述

栅格 `rastA` 包含 `rastB` 当且仅当 `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（或置为 `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!****Note**

此函数将利用栅格上可能可用的任何索引。

**Note!****Note**

要测试栅格和几何形的空关系，则在栅格上使用 `ST_Polygon`，例如 `ST_Contains (ST_Polygon (raster), geometry)` 或 `ST_Contains (geometry, ST_Polygon (raster))`。

**Note!****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
SELECT r1.rid, r2.rid, ST_Contains(r1.rast, 1, r2.rast, 1) FROM dummy_rast r1 CROSS JOIN ↵
    dummy_rast r2 WHERE r1.rid = 1;
```

NOTICE: The first raster provided has no bands

```
rid | rid | st_contains
-----+-----+-----
 1 | 1 | t
 1 | 2 | f
```

```
-- no band numbers specified
SELECT r1.rid, r2.rid, ST_Contains(r1.rast, r2.rast) FROM dummy_rast r1 CROSS JOIN ↵
    dummy_rast r2 WHERE r1.rid = 1;
```

```
rid | rid | st_contains
-----+-----+-----
 1 | 1 | t
 1 | 2 | f
```

## 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，则返回 `true`。

## Synopsis

```
boolean ST_ContainsProperly( raster rastA , integer nbandA , raster rastB , integer nbandB );
boolean ST_ContainsProperly( raster rastA , raster rastB );
```

### 描述

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 `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

```
SELECT r1.rid, r2.rid, ST_ContainsProperly(r1.rast, 1, r2.rast, 1) FROM dummy_rast r1 CROSS JOIN dummy_rast r2 WHERE r1.rid = 2;
```

rid	rid	st_containsproperly
2	1	f
2	2	f

### 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 );
```

## 描述

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**

要测试栅格和几何形的空关系，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

```
SELECT r1.rid, r2.rid, ST_Covers(r1.rast, 1, r2.rast, 1) FROM dummy_rast r1 CROSS JOIN ←
    dummy_rast r2 WHERE r1.rid = 2;
```

rid	rid	st_covers
2	1	f
2	2	t

## 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 );
```

## 描述

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 ☐ ☐ ☐ 格和几何 ☐ 形的空 ☐ 关系，☐ 在 ☐ 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

```
SELECT r1.rid, r2.rid, ST_CoveredBy(r1.rast, 1, r2.rast, 1) FROM dummy_rast r1 CROSS JOIN ↵
    dummy_rast r2 WHERE r1.rid = 2;
```

rid	rid	st_coveredby
2	1	f
2	2	t

相关信息

[ST\\_Intersects](#), [ST\\_Covers](#)

### 11.17.5 ST\_Disjoint

ST\_Disjoint — 如果 ☐ 格 rastA 在空 ☐ 上不与 rastB 相交，☐ 返回 true。

#### Synopsis

boolean **ST\_Disjoint**( raster rastA , integer nbandA , raster rastB , integer nbandB );

boolean **ST\_Disjoint**( raster rastA , raster rastB );

描述

如果 ☐ 格 rastA 和 rastB 不共享任何空 ☐，☐ 它是脱 ☐ 的。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**

☐ 函数不使用任何索引。

**Note**

要 ☐ ☐ ☐ 格和几何 ☐ 形的空 ☐ 关系，☐ 在 ☐ 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
SELECT r1.rid, r2.rid, ST_Disjoint(r1.rast, 1, r2.rast, 1) FROM dummy_rast r1 CROSS JOIN ↵
    dummy_rast r2 WHERE r1.rid = 2;
```

NOTICE: The second raster provided has no bands

rid	rid	st_disjoint
2	1	
2	2	f

```
-- this time, without specifying band numbers
```

```
SELECT r1.rid, r2.rid, ST_Disjoint(r1.rast, r2.rast) FROM dummy_rast r1 CROSS JOIN ↵
    dummy_rast r2 WHERE r1.rid = 2;
```

rid	rid	st_disjoint
2	1	t
2	2	f

相关信息

[ST\\_Intersects](#)

### 11.17.6 ST\_Intersects

ST\_Intersects — 如果栅格 rastA 与栅格 rastB 空相交，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 );
```

描述

如果栅格 rastA 与栅格 rastB 空相交，返回 true。如果未提供波段号（或置 NULL），则在两栅格的凸包中考虑那些具有（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。

增修：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
SELECT ST_Intersects(rast, 2, rast, 3) FROM dummy_rast WHERE rid = 2;

st_intersects
-----
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，返回 `true`。

### Synopsis

```
boolean ST_Overlaps( raster rastA , integer nbandA , raster rastB , integer nbandB );
boolean ST_Overlaps( raster rastA , raster rastB );
```

描述

如果栅格 `rastA` 在空域上与栅格 `rastB` 重叠，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。如果未提供波段号（或置为 `NULL`），则在栅格中考虑栅格的凸包。如果提供了波段号，则考虑那些具有（而不是 `NODATA`）的像素。



#### 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
SELECT ST_Overlaps(rast, 1, rast, 2) FROM dummy_rast WHERE rid = 2;

st_overlaps
-----
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，返回 `true`。

#### Synopsis

```
boolean ST_Touches( raster rastA , integer nbandA , raster rastB , integer nbandB );
boolean ST_Touches( raster rastA , raster rastB );
```

#### 描述

如果栅格 `rastA` 在空域上接触栅格 `rastB`，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。如果未提供波段号（或置为 `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

要检查栅格和几何形的空域关系，请在栅格上使用 `ST_Polygon`，例如 `ST_Touches(ST_Polygon(raster), geometry)`。

可用性：2.1.0

#### 示例

```
SELECT r1.rid, r2.rid, ST_Touches(r1.rast, 1, r2.rast, 1) FROM dummy_rast r1 CROSS JOIN ↵
dummy_rast r2 WHERE r1.rid = 2;
```

rid	rid	st_touches
2	1	f
2	2	f

相关信息

[ST\\_Intersects](#)

### 11.17.9 ST\_SameAlignment

`ST_SameAlignment` — 如果栅格具有相同的斜率、比例、空域参考和偏移（像素可以放在同一网格上而不切割成像素），返回 `true`；如果没有注意栅格，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 );
```

### 描述

非聚合版本（`ST_SameAlignment` 1 和 2）：如果两个栅格（直接提供或使用左上角、比例、斜角和 `srid` 的 `ST_MakeEmptyRaster` 制作）具有相同的比例、斜角、`srid` 以及至少其中之一，则返回 `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。如果不匹配则返回 `false` 并返回 `ST_SameAlignment` 的通知。

聚合版本（`ST_SameAlignment` 3）：从一个栅格中，如果其中的所有栅格都匹配，则返回 `true`。`ST_SameAlignment()` 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。

可用性: 2.0.0

增强: 2.1.0 添加聚合版本

示例: 栅格

```
SELECT ST_SameAlignment(
    ST_MakeEmptyRaster(1, 1, 0, 0, 1, 1, 0, 0),
    ST_MakeEmptyRaster(1, 1, 0, 0, 1, 1, 0, 0)
) as sm;
```

```
sm
----
t
```

```
SELECT ST_SameAlignment(A.rast,b.rast)
FROM dummy_rast AS A CROSS JOIN dummy_rast AS B;
```

```
NOTICE: The two rasters provided have different SRIDs
NOTICE: The two rasters provided have different SRIDs
st_samealignment
-----
t
f
f
f
```

### 相关信息

Section [10.1](#), [ST\\_NotSameAlignmentReason](#), [ST\\_MakeEmptyRaster](#)

## 11.17.10 ST\_NotSameAlignmentReason

`ST_NotSameAlignmentReason` — 返回说明栅格是否匹配的文本，如果未匹配，则说明原因。



## Synopsis

```
text ST_NotSameAlignmentReason(raster rastA, raster rastB);
```

### 描述

返回说明栅格是否匹配的文本，如果未匹配，说明原因。



#### Note

如果栅格未匹配的原因有多种，返回一个原因（第一个失败的匹配）。

可用性：2.1.0

### 示例

```
SELECT
  ST_SameAlignment(
    ST_MakeEmptyRaster(1, 1, 0, 0, 1, 1, 0, 0),
    ST_MakeEmptyRaster(1, 1, 0, 0, 1.1, 1.1, 0, 0)
  ),
  ST_NotSameAlignmentReason(
    ST_MakeEmptyRaster(1, 1, 0, 0, 1, 1, 0, 0),
    ST_MakeEmptyRaster(1, 1, 0, 0, 1.1, 1.1, 0, 0)
  )
;

st_samealignment | st_otsamealignmentreason
-----+-----
f                | The rasters have different scales on the X axis
(1 row)
```

### 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 );
```

## 描述

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。如果未提供波段号（或置为 `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!****Note**

操作数将利用栅格上可能可用的任何索引。

**Note!****Note**

要检查栅格和几何形的空关系，则在栅格上使用 `ST_Polygon`，例如 `ST_Within(ST_Polygon(raster), geometry)` 或 `ST_Within(geometry, ST_Polygon(raster))`。

**Note!****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

```
SELECT r1.rid, r2.rid, ST_Within(r1.rast, 1, r2.rast, 1) FROM dummy_rast r1 CROSS JOIN
dummy_rast r2 WHERE r1.rid = 2;
```

rid	rid	st_within
2	1	f
2	2	t

## 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` 彼此之间的距离在指定距离内，则返回 `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 );
```

## 描述

如果栅格 `rastA` 和 `rastB` 彼此之间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非空（而不是 `NODATA`）的像素。

距离以栅格空参考系定义的 SRS 指定。为了使此函数有意义，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

要检查栅格和几何形的空关系，请在栅格上使用 `ST_Polygon`，例如 `ST_DWithin(ST_Polygon(raster), geometry)`。

可用性：2.1.0

## 示例

```
SELECT r1.rid, r2.rid, ST_DWithin(r1.rast, 1, r2.rast, 1, 3.14) FROM dummy_rast r1 CROSS JOIN dummy_rast r2 WHERE r1.rid = 2;
```

rid	rid	st_dwithin
2	1	f
2	2	t

## 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 );
```

## 描述

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非空（而不是 `NODATA`）的像素。

距离以栅格空参考系定义的 SRS 指定。为了使此函数有意义，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

```
SELECT r1.rid, r2.rid, ST_DFullyWithin(r1.rast, 1, r2.rast, 1, 3.14) FROM dummy_rast r1
  CROSS JOIN dummy_rast r2 WHERE r1.rid = 2;
```

rid	rid	st_dfullywithin
2	1	f
2	2	t

相关信息

[ST\\_Within](#), [ST\\_DWithin](#)

## 11.18 栅格提示

### 11.18.1 Out-DB 栅格

#### 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 等网络驱动器上）。

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。您也可以在此设置 PostGIS 环境变量。

```
# environment variables for postmaster process
# This file has the same syntax as postgresql.conf:
# VARIABLE = simple_value
# VARIABLE2 = 'any value!'
# I. e. you need to enclose any value which does not only consist of letters,
# numbers, and '-', '_', '.' in single quotes. Shell commands are not
# evaluated.
POSTGIS_GDAL_ENABLED_DRIVERS = 'ENABLE_ALL'
```

```
POSTGIS_ENABLE_OUTDB_RASTERS = 1
GDAL_DISABLE_READDIR_ON_OPEN = 'TRUE'
```

### 11.18.1.2 最大打开文件数

Linux 和 PostgreSQL 允许的最大打开文件数通常是相保守的（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，即系统主要由人使用。然而，对于 Out-DB 栅格数据，一个有效的栅格很容易超过这个限制（例如，一个包含了 10 年的栅格数据集，每天都有一个栅格包含了最低和最高温度数据，我们想要知道数据集集中某个像素的栅格最小和最大值）。

最合理的更改是以下 PostgreSQL 设置：`max_files_per_process`。默认设置为 1000，这对于 Out-DB 栅格来说太低了。安全的起始值可能是 65536，但这个值上取决于您的数据集以及某些数据集行的值。此设置只能在服务器上运行，并且可能只能在 PostgreSQL 配置文件中运行（例如 Ubuntu 环境中的 `/etc/postgresql/POSTGRES_VERSION/CLUSTER_NAME/postgresql.conf`）。

```
...
# - 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。它有几个部分：

- 整个系统最大打开文件数
- 每个进程的最大打开文件数

#### 11.18.1.2.1 整个系统最大打开文件数

您可以通过以下示例查看整个系统当前最大打开文件数：

```
$ sysctl -a | grep fs.file-max
fs.file-max = 131072
```

如果返回的值不够大，按照以下示例将文件添加到 `/etc/sysctl.d/` 中：

```
$ 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 每个进程的最大打开文件数

我们需要增加 PostgreSQL 服务器进程每个进程打开文件的最大数量。

要查看当前 PostgreSQL 服务器使用的最大打开文件数，按照以下示例进行操作（确保 PostgreSQL 正在运行）：

```

$ ps aux | grep postgres
postgres 31713  0.0  0.4 179012 17564 pts/0    S   Dec26   0:03 /home/dustymugs/devel/ ↵
    postgresql/sandbox/10/usr/local/bin/postgres -D /home/dustymugs/devel/postgresql/sandbox ↵
    /10/pgdata
postgres 31716  0.0  0.8 179776 33632 ?        Ss  Dec26   0:01 postgres: checkpointer ↵
    process
postgres 31717  0.0  0.2 179144  9416 ?        Ss  Dec26   0:05 postgres: writer process
postgres 31718  0.0  0.2 179012  8708 ?        Ss  Dec26   0:06 postgres: wal writer ↵
    process
postgres 31719  0.0  0.1 179568  7252 ?        Ss  Dec26   0:03 postgres: autovacuum ↵
    launcher process
postgres 31720  0.0  0.1  34228  4124 ?        Ss  Dec26   0:09 postgres: stats collector ↵
    process
postgres 31721  0.0  0.1 179308  6052 ?        Ss  Dec26   0:00 postgres: bgworker: ↵
    logical replication launcher

$ cat /proc/31718/limits
Limit                Soft Limit            Hard Limit             Units
Max cpu time          unlimited             unlimited             seconds
Max file size         unlimited             unlimited             bytes
Max data size         unlimited             unlimited             bytes
Max stack size        8388608              unlimited             bytes
Max core file size    0                    unlimited             bytes
Max resident set      unlimited             unlimited             bytes
Max processes         15738                15738                 processes
Max open files      1024                4096                 files
Max locked memory     65536                65536                 bytes
Max address space     unlimited             unlimited             bytes
Max file locks        unlimited             unlimited             locks
Max pending signals   15738                15738                 signals
Max msgqueue size     819200               819200                bytes
Max nice priority     0                    0
Max realtime priority 0                    0
Max realtime timeout  unlimited             unlimited             us

```

在上面的示例中，我查看了进程 31718 的打开文件限制。无是那个 PostgreSQL 进程，它其中的任何一个都可以。我感兴趣的限制是最大打开文件数。

我希望将最大打开文件数的限制和硬限制增加到大于我 PostgreSQL 配置 `max_files_per_process` 指定的。在我的示例中，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`。

```

...
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 问题。我✘将在每个 `/lib/systemd/system/postgresql*.service` 文件的 **[Service]** 部分中添加 **LimitNOFILE=131072**。

```
...  
[Service]  
  
LimitNOFILE=131072  
  
...  
  
[Install]  
WantedBy=multi-user.target  
...
```

✘行必要的 systemd 更改后，✘确保重新加载✘守护✘进程

```
systemctl daemon-reload
```

## Chapter 12

# PostGIS 扩充

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标准 shell 脚本。

### 12.1 地址标准化工具

它是 **PAGC 标准化器** 的一个分支（该部分的原始代码是 **PAGC PostgreSQL 地址标准化器**）。

地址标准化器是一个行地址解析器，它读取输入地址并根据存储在表以及辅助 `lex` 和 `gaz` 表中的一系列规则进行规范化。

代码内置于一个名为 `address_standardizer` 的 PostgreSQL 扩展中，可以使用 `CREATE EXTENSION address_standardizer;` 安装。除了 `address_standardizer` 扩展之外，还构建了一个名为 `address_standardizer_data_us` 的扩展的示例数据扩展，其中包含 US 数据的 `gaz`、`lex` 和规则表。该扩展可以通过以下方式安装：`CREATE EXTENSION address_standardizer_data_us;`

此扩展的代码可以在 `PostGIS extensions/address_standardizer` 中找到，并且当前是独立的。

有关安装说明，请参考：Section 2.3。

#### 12.1.1 解析器如何工作

解析器从右到左工作，首先查看邮政编码、州/省、城市的宏元素，然后查看微元素以确定我们是否正在处理牌号、街道、十字路口或地区。目前，它不寻找国家/地区代码或名称，但将来可能会引入。

国家代码 根据邮政编码美国或加拿大，或者州/省美国或加拿大，否则美国

邮政编码 (**Postcode**) / 邮政编码 (**zipcode**) 这些是使用 Perl 兼容的正则表达式来匹配的。这些正则表达式当前位于 `parseaddress-api.c` 中，并且在需要时进行更改是可行的。

州/省 这些是使用 Perl 兼容的正则表达式来匹配的。这些正则表达式当前位于 `parseaddress-api.c` 中，但将来可能会被移至包含中以便于匹配。

#### 12.1.2 地址标准化器类型

##### 12.1.2.1 stdaddr

`stdaddr` — 由地址元素组成的复合类型。它是 `standardize_address` 函数的返回类型。



## 描述

由地址元素组成的复合型。是 `standardize_address` 函数的返回型。一些元素描述借用自 `PAGC` 属性。令牌号表示 `rules table` 中的输出参考号。

 方法需要 `address_standardizer` 展。

**building** 是文本 (令牌号 0)：指建筑物号或名称。未解析的建筑物符号和型。大多数地址通常空白。

**house\_num** 是一段文本 (令牌号 1)：是街道上的街道号。例如，75 在 75 State Street 中。

**predir** 是文本 (令牌号 2)：街道名称指示，例如北、南、东、西等。

**qual** 是文本 (令牌号 3)：街道名称修饰符示例 *OLD* in 3715 OLD HIGHWAY 99。

**pretype** 是文本 (令牌号 4)：街道前型

**name** 是文本 (令牌号 5)：街道名称

**suftype** 是文本 (令牌号 6)：街道后型，例如 St、Ave、Cir。是跟在根街道名称后面的街道型。例如，*STREET* 位于 75 State Street 中。

**sufdir** 是文本 (令牌号 7)：STREET POST-DIRECTIONAL 街道名称后面的方向修饰符。示例 *WEST* 位于 3715 TENTH AVENUE WEST 中。

**ruralroute** 是文本 (令牌号 8)：村路。例如，7 位于 RR 7 中。

**extra** 是文本：外信息，例如楼号。

**city** 是文本 (令牌号 10)：例如，波士。

**state** 是文本 (令牌号 11)：例如，MASSACHUSETTS

**country** 是文本 (令牌号 12)：例如，USA

**postcode** 是文本 (令牌号 13)：政 (ZIP CODE) 的示例：02109

**box** 是文本 (令牌号 14 和 15)：政信箱号的示例：02109

**unit** 是文本 (令牌号 17)：公寓号或套房号的示例，例如，在 3B 位于 APT 3B 中。

## 12.1.3 地址标准化表

### 12.1.3.1 rules table

`rules table` — 表包含一列，将地址输入序列映射到标准化的输出序列。列被定一输入，后跟 -1 (终止符)，然后是一输出，后跟 -1，后跟表示型的数字，最后是列的排名。

## 描述

表至少包含以下列，但您可以添加更多列供自己使用。

**id** 表的主

**rule** 表示的文本字段。信息参 `PAGC 地址标准化器`。

一条由一表示输入的非整数组成，以 -1 束，然后是相同数量的非整数，表示属性，以 -1 束，然后是表示型的整数，最后是表示排名的整数。的排名从 0 (最低) 到 17 (最高)。

例如，`2 0 2 22 3 -1 5 5 6 7 3 -1 2 6` 映射到输出序列 `TYPE NUMBER TYPE DIRECT QUALIF` 到输出序列 `STREET STREET SUFTYP SUFDIR QUALIF`。是等 6 的 `ARC_C`。

`stdaddr` 中列出了相输出的号。

## 入令牌

每个入令牌都以一个入开始，后跟一个止符 -1。从PAGC 入令牌中摘出的有效入令牌如下：

基于表的入令牌

**AMPERS** (13). "&" 符号常用于写"and"。

**DASH** (9). 点符号字符。

**DOUBLE** (21). 个字母的序列。常用作符。

**FRACT** (25). 分数有用于公民数字或位数字。

**MIXED** (23). 包含字母和数字的字母数字字符串。用于符。

**NUMBER** (0). 一串数字。

**ORD** (15). 如 First 或 1st 之的表示。常用于街道名称。

**ORD** (18). 一个字母。

**WORD** (1). 是任意度的字母串。个字母既可以是 SINGLE，也可以是 WORD。

基于功能的入令牌

**BOXH** (14). 用于表示政信箱的。例如 *Box* 或 *PO Box*。

**BUILDH** (19). 用于表示建筑物或建筑群的，通常作前。例如，在 *Tower 7A* 中的 *Tower*。

**BUILDT** (24). 用于表示建筑物或建筑群的和写，通常作后。例如：物中心。

**DIRECT** (22). 用于表示方向的，例如北。

**MILE** (20). 用于表示里程碑地址的。

**ROAD** (6). 用于表示高速公路和道路的和写。例如，在 *Interstate 5* 中的 *Interstate*

**RR** (8). 用于表示村路的和写。RR。

**TYPE** (2). 用于表示街道型的和写。例如：*ST* or *AVE*。

**UNITH** (16). 用于表示内部子地址的和写。例如，*APT* 或 *UNIT*。

政型入令牌

**QUINT** (28).5 位数字。政

**QUAD** (29). 一个 4 位数字，用于 ZIP4 。

**PCH** (27). 由字母、数字和字母成的 3 个字符序列。用于 FSA，即加拿大政的前 3 个字符。

**PCT** (26). 由数字、字母和数字成的 3 个字符序列。用于 LDU，即加拿大政的最后 3 个字符。

停用

STOPWORDS 与 WORDS 合。在中，多个 WORD 和 STOPWORD 成的字符串将由个 WORD 表示。

**STOPWORD** (7). 具有低重要性的，可以在解析中省略。例如：*THE*。

## 出令牌

在第一个 -1 (止符) 之后, 跟随出及其序, 然后是止符 -1。stdaddr中列出了相出出的号。允的内容取决于的型。the section called “型和等” 部分列出了每种型有效的出。

## 型和等

的最后部分是型, 由以下之一表示, 后跟等。的排名从 0 (最低) 到 17 (最高)。

### MACRO\_C

(令牌号 = “0”)。用于解析 MACRO 子句 (例如 *PLACE STATE ZIP*) 的。

MACRO\_C 出 (摘自 <http://www.pgcgeo.org/docs/html/pagc-12.html#--r-ty-->)。

**CITY** (令牌号 “10”)。示例 “Albany”

**STATE** (令牌号 “11”)。示例 “NY”

**NATION** (令牌号 “12”)。大多数参考文件中不使用此属性。示例 “USA”

**POSTAL** (令牌号 “13”)。(SADS 元素 “ZIP CODE”、“PLUS 4”)。此属性用于美国政和加拿大政。

### MICRO\_C

(令牌号 = “1”)。用于解析完整 MICRO 子句 (例如 *House*、*street*、*sufdir*、*predir*、*pretyp*、*suftype*、*qualif*) 的 (即 ARC\_C 加 CIVIC\_C)。些不会在建段使用。

MICRO\_C 出令牌 (摘自 <http://www.pgcgeo.org/docs/html/pagc-12.html#--r-ty-->)。

**HOUSE** 是一段文本 (号 1): 是街道上的街道号。例如, 75 在 75 State Street 中。

**predir** 是文本 (号 2): 街道名称指示, 例如北、南、西等。

**qual** 是文本 (号 3): 街道名称修符示例 *OLD* in 3715 OLD HIGHWAY 99。

**pretype** 是文本 (号 4): 街道前型

**street** 是文本 (号 5): 街道名称

**suftype** 是文本 (号 6): 街道后型, 例如 St、Ave、Cir。是跟在根街道名称后面的街道型。例如, *STREET* 位于 75 State Street 中。

**sufdir** 是文本 (号 7): STREET POST-DIRECTIONAL 街道名称后面的方向修符。示例 *WEST* 位于 3715 TENTH AVENUE WEST 中。

### ARC\_C

(令牌号 = “2”)。用于解析 MICRO 子句的, 不包括 HOUSE 属性。因此, 使用与 MICRO\_C 相同的一出去 HOUSE 令牌。

### CIVIC\_C

(令牌号 = “3”)。用于解析 HOUSE 属性的。

### EXTRA\_C

(令牌号 = “4”)。用于解析外属性的 - 从地理中排除的属性。些不会在建段使用。

EXTRA\_C 出令牌 (摘自 <http://www.pgcgeo.org/docs/html/pagc-12.html#--r-ty-->)。

**BLDNG** (令牌号 0): 未解析的建筑物符和型。

**BOXH** (令牌号 14): **BOX** 位于 BOX 3B 中

**BOXT** (令牌号 15) : **3B** 位于 BOX 3B 中

**RR** (令牌号 8) : **RR** 位于 RR 7 中

**UNITH** (令牌号 16) : **APT** 位于 APT 3B 中

**UNITT** (令牌号 17) : **3B** 位于 APT 3B 中

**UNKNWN** (令牌号 9) : 否未分的出。

### 12.1.3.2 lex table

**lex table** — **lex** 表用于字母数字输入行分，并将输入与 (a) 输入 (参 the section called “输入令牌” 一) 和 (b) 规范化表示相关。

#### 描述

**lex** (lex) 表用于分字母数字输入并将输入与 (a) the section called “输入令牌” 和 (b) 规范化表示相关。在这些表中，您会找到例如将 **ONE** 映射到准 1 的映射关系。

一个 **lex** 在表中至少有以下列。您可以添加

**id** 表的主

**seq** 整数：定号？

**word** 文本：输入的

**stdword** text：准化的替

**token** 整数：它是什么型的。只有在个上下文中使用它才会被替。参 **PAGC** 令牌。

### 12.1.3.3 gaz table

**gaz table** — **gaz** 表用于规范化地名，并将输入与 (a) 输入令牌 (参 “the section called “输入令牌” 一) 和 (b) 规范化表示相关。

#### 描述

**gaz** (地名典的写) 表用于规范化地名，并将输入与称 “the section called “输入令牌” 的部分和 (b) 规范化表示相关。例如，如果您在美国，您可以加州名称和相关写。

**gaz** 表中至少包含以下列。如果您愿意出于自己的目的，可以添加更多列。

**id** 表的主

**seq** 整数：定号？- 用于例的符

**word** 文本：输入的

**stdword** text：准化的替

**token** 整数：它是什么型的。只有在个上下文中使用它才会被替。参 **PAGC** 令牌。

### 12.1.4 地址规范化器功能

#### 12.1.4.1 debug\_standardize\_address

**debug\_standardize\_address** — 返回 json 格式的文本，列出解析和规范化

## Synopsis

```
text debug_standardize_address(text lextab, text gaztab, text rultab, text micro, text macro=NULL);
```

### 描述

`debug_standardize_address` 是一个用于地址标准化器和 `lex/gaz` 映射的函数。它返回一个 json 格式的文本，其中包括匹配令牌、令牌映射以及使用 `lex table` 的表名称、`gaz table` 和 `rules table` 的表名称和地址的输入地址的最佳标准化地址 `stdaddr` 形式。

用于单行地址，使用 `micro`

用于多行地址，一个是由 `micro` 的邮政地址第一行组成，例如 `house_num street`，另一个是由标准的邮政地址第二行组成，例如 `city, state postal_code country`。

json 文档中返回的元素是

**input\_tokens** 用于输入地址中的每个令牌，返回令牌的位置、令牌的类型以及它映射到的标准令牌。注意，对于某些输入令牌，您可能会返回多个令牌，因为某些输入可以匹配多个事物。

**rules** 与输入匹配的令牌集以及每个令牌的分数。第一条令牌（最高分）用于标准化

**stdaddr** 单行 `standardize_address` 将返回的标准化地址元素 `stdaddr`

可用性：3.4.0



该方法需要 `address_standardizer` 扩展。

### 示例

使用 `address_standardizer_data_us` 扩展

```
CREATE EXTENSION address_standardizer_data_us; -- only needs to be done once
```

例 1：单行地址并返回输入令牌

```
SELECT it->>'pos' AS position, it->>'word' AS word, it->>'stdword' AS standardized_word,
       it->>'token' AS token, it->>'token-code' AS token_code
FROM jsonb(
    debug_standardize_address('us_lex',
                              'us_gaz', 'us_rules', 'One Devonshire Place, PH 301, Boston, MA 02109')
) AS s, jsonb_array_elements(s->'input_tokens') AS it;
```

position	word	standardized_word	token	token_code
0	ONE	1	NUMBER	0
0	ONE	1	WORD	1
1	DEVONSHIRE	DEVONSHIRE	WORD	1
2	PLACE	PLACE	TYPE	2
3	PH	PATH	TYPE	2
3	PH	PENTHOUSE	UNITT	17
4	301	301	NUMBER	0

(7 rows)

例 2：多行地址并返回第一条输入令牌映射和分数

```
SELECT (s->'rules'->0->>'score')::numeric AS score, it->>'pos' AS position,
       it->>'input-word' AS word, it->>'input-token' AS input_token, it->>'mapped-word' AS ←
       standardized_word,
       it->>'output-token' AS output_token
FROM jsonb(
       debug_standardize_address('us_lex',
       'us_gaz', 'us_rules', 'One Devonshire Place, PH 301', 'Boston, MA 02109')
       ) AS s, jsonb_array_elements(s->'rules'->0->'rule_tokens') AS it;
```

score	position	word	input_token	standardized_word	output_token
0.876250	0	ONE	NUMBER	1	HOUSE
0.876250	1	DEVONSHIRE	WORD	DEVONSHIRE	STREET
0.876250	2	PLACE	TYPE	PLACE	SUFTYP
0.876250	3	PH	UNITT	PENTHOUSE	UNITT
0.876250	4	301	NUMBER	301	UNITT

(5 rows)

相关信息

[stdaddr](#), [rules table](#), [lex table](#), [gaz table](#), [Pgac\\_Normalize\\_Address](#)

### 12.1.4.2 parse\_address

parse\_address — 取 1 行地址并分成几部分

#### Synopsis

```
record parse_address(text address);
```

描述

Returns 将地址作输入，并返回由字段 *num*、*street*、*street2*、*address1*、*city*、*state*、*zip*、*zipplus*、*country* 成的输出。

可用性：2.2.0

 方法需要 `address_standardizer` 展。

示例

一地址

```
SELECT num, street, city, zip, zipplus
FROM parse_address('1 Devonshire Place, Boston, MA 02109-1234') AS a;
```

num	street	city	zip	zipplus
1	Devonshire Place	Boston	02109	1234

地址表

```
-- basic table
CREATE TABLE places(addid serial PRIMARY KEY, address text);

INSERT INTO places(address)
VALUES ('529 Main Street, Boston MA, 02129'),
       ('77 Massachusetts Avenue, Cambridge, MA 02139'),
       ('25 Wizard of Oz, Waford, KS 99912323'),
       ('26 Capen Street, Medford, MA'),
       ('124 Mount Auburn St, Cambridge, Massachusetts 02138'),
       ('950 Main Street, Worcester, MA 01610');

-- parse the addresses
-- if you want all fields you can use (a).*
SELECT addid, (a).num, (a).street, (a).city, (a).state, (a).zip, (a).zipplus
FROM (SELECT addid, parse_address(address) As a
      FROM places) AS p;
```

addid	num	street	city	state	zip	zipplus
1	529	Main Street	Boston	MA	02129	
2	77	Massachusetts Avenue	Cambridge	MA	02139	
3	25	Wizard of Oz	Waford	KS	99912	323
4	26	Capen Street	Medford	MA		
5	124	Mount Auburn St	Cambridge	MA	02138	
6	950	Main Street	Worcester	MA	01610	

(6 rows)

相关信息

### 12.1.4.3 standardize\_address

`standardize_address` — 利用 `lex`、`gaz` 和 `rules` 表返回 `stdaddr` 形式。

#### Synopsis

```
stdaddr standardize_address(text lextab, text gaztab, text rultab, text address);
stdaddr standardize_address(text lextab, text gaztab, text rultab, text micro, text macro);
```

#### 描述

使用 `lex table` 的表名称、`gaz table` 和 `rules table` 的表名称和地址返回 `stdaddr` 形式。

体 1：将地址 `stdaddr` 一行。

体 2：将地址分 2 部分。由 `micro` 第一行 `stdaddr` 成的微行，例如 `house_num street`，以及由地址的 `stdaddr` 第二行 `stdaddr` 成的宏，例如 `city, state postal_code country`。

可用性：2.2.0



方法需要 `address_standardizer` 扩展。

#### 示例

使用 `address_standardizer_data_us` 扩展

```
CREATE EXTENSION address_standardizer_data_us; -- only needs to be done once
```

☒体 1 : ☒行地址。☒不适用于非美国地址

```
SELECT house_num, name, suftype, city, country, state, unit FROM standardize_address(' ←
    us_lex',
                                'us_gaz', 'us_rules', 'One Devonshire Place, PH 301, Boston, MA ←
                                02109');
```

house_num	name	suftype	city	country	state	unit
1	DEVONSHIRE	PLACE	BOSTON	USA	MASSACHUSETTS	# PENTHOUSE 301

使用与 Tiger 地理☒器打包的表格。此示例☒在您安装了 postgis\_tiger\_geocoder ☒才有效。

```
SELECT * FROM standardize_address('tiger.pagc_lex',
    'tiger.pagc_gaz', 'tiger.pagc_rules', 'One Devonshire Place, PH 301, Boston, MA ←
    02109-1234');
```

☒了更容易☒☒, 我☒将使用 hstore ☒展☒☒☒出 CREATE EXTENSION hstore; 你需要安装

```
SELECT (each(hstore(p))).*
FROM standardize_address('tiger.pagc_lex', 'tiger.pagc_gaz',
    'tiger.pagc_rules', 'One Devonshire Place, PH 301, Boston, MA 02109') As p;
```

key	value
box	
city	BOSTON
name	DEVONSHIRE
qual	
unit	# PENTHOUSE 301
extra	
state	MA
predir	
sufdir	
country	USA
pretype	
suftype	PL
building	
postcode	02109
house_num	1
ruralroute	

(16 rows)

☒体 2 : 作☒☒部分地址

```
SELECT (each(hstore(p))).*
FROM standardize_address('tiger.pagc_lex', 'tiger.pagc_gaz',
    'tiger.pagc_rules', 'One Devonshire Place, PH 301', 'Boston, MA 02109, US') As p;
```

key	value
box	
city	BOSTON
name	DEVONSHIRE
qual	
unit	# PENTHOUSE 301
extra	
state	MA
predir	
sufdir	
country	USA



```

pretype |
suftype | PL
building |
postcode | 02109
house_num | 1
ruralroute |
(16 rows)

```

相关信息

[stdaddr](#), [rules table](#), [lex table](#), [gaz table](#), [Pagc\\_Normalize\\_Address](#)

## 12.2 Tiger 地理引擎

PostGIS 有其他几个开源地理引擎，与 Tiger 地理引擎不同，它们具有多国家地理引擎支持的引擎

- **Nominatim** 使用 OpenStreetMap 地名词典格式的数据。它需要 `osm2pgsql` 来加载数据、PostgreSQL 8.4+ 和 PostGIS 1.5+ 才能运行。它被打包成 Web 服务接口，并且似乎被用作 Web 服务来调用。就像 Tiger 地理引擎一样，它同样具有地理引擎和反向地理引擎组件。从文档来看，尚不清楚它是否具有像 Tiger 地理引擎一样的 SQL 接口，或者大量引擎是否在 Web 接口中。
- **GIS Graphy** 利用 PostGIS，并且像 Nominatim 一样使用 OpenStreetMap (OSM) 数据。它配置了一个加载器来加载 OSM 数据，并且与 Nominatim 类似，不能运行美国地理引擎。与 Nominatim 非常相似，它作为 Web 服务运行并依赖于 Java 1.5、Servlet 应用程序、Solr。GisGraphy 是跨平台的，并且具有反向地理引擎以及其他一些引擎的功能。

### 12.2.1 Drop\_Indexes\_Generate\_Script

`Drop_Indexes_Generate_Script` — 生成一个脚本，删除 Tiger 引擎和用引擎指定引擎上的所有非主和非唯一索引。如果未指定引擎，默认引擎为 `Tiger_data`。

#### Synopsis

```
text Drop_Indexes_Generate_Script(text param_schema=tiger_data);
```

#### 描述

生成一个脚本，删除 Tiger 引擎和用引擎指定引擎上的所有非主和非唯一索引。如果未指定引擎，默认引擎为 `Tiger_data`。

引擎于最大限度地减少索引膨胀很有用，索引膨胀可能会混淆引擎划器或占用不必要的空间。与 [Install\\_Missing\\_Indexes](#) 合使用，添加地理引擎使用的索引。

可用性: 2.0.0

## 示例

```

SELECT drop_indexes_generate_script() As actionsql;
actionsql
-----
DROP INDEX tiger.idx_tiger_countysub_lookup_lower_name;
DROP INDEX tiger.idx_tiger_edges_countyfp;
DROP INDEX tiger.idx_tiger_faces_countyfp;
DROP INDEX tiger.tiger_place_the_geom_gist;
DROP INDEX tiger.tiger_edges_the_geom_gist;
DROP INDEX tiger.tiger_state_the_geom_gist;
DROP INDEX tiger.idx_tiger_addr_least_address;
DROP INDEX tiger.idx_tiger_addr_tlid;
DROP INDEX tiger.idx_tiger_addr_zip;
DROP INDEX tiger.idx_tiger_county_countyfp;
DROP INDEX tiger.idx_tiger_county_lookup_lower_name;
DROP INDEX tiger.idx_tiger_county_lookup_snd_name;
DROP INDEX tiger.idx_tiger_county_lower_name;
DROP INDEX tiger.idx_tiger_county_snd_name;
DROP INDEX tiger.idx_tiger_county_the_geom_gist;
DROP INDEX tiger.idx_tiger_countysub_lookup_snd_name;
DROP INDEX tiger.idx_tiger_cousub_countyfp;
DROP INDEX tiger.idx_tiger_cousub_cousubfp;
DROP INDEX tiger.idx_tiger_cousub_lower_name;
DROP INDEX tiger.idx_tiger_cousub_snd_name;
DROP INDEX tiger.idx_tiger_cousub_the_geom_gist;
DROP INDEX tiger_data.idx_tiger_data_ma_addr_least_address;
DROP INDEX tiger_data.idx_tiger_data_ma_addr_tlid;
DROP INDEX tiger_data.idx_tiger_data_ma_addr_zip;
DROP INDEX tiger_data.idx_tiger_data_ma_county_countyfp;
DROP INDEX tiger_data.idx_tiger_data_ma_county_lookup_lower_name;
DROP INDEX tiger_data.idx_tiger_data_ma_county_lookup_snd_name;
DROP INDEX tiger_data.idx_tiger_data_ma_county_lower_name;
DROP INDEX tiger_data.idx_tiger_data_ma_county_snd_name;
:
:

```

## 相关信息

[Install\\_Missing\\_Indexes](#), [Missing\\_Indexes\\_Generate\\_Script](#)

## 12.2.2 Drop\_Nation\_Tables\_Generate\_Script

`Drop_Nation_Tables_Generate_Script` — 生成一个脚本，`除指定架`中以 `County_all`、`state_all` 或 `state` 代`开`，后跟 `county` 或 `state` 的所有表。

### Synopsis

```
text Drop_Nation_Tables_Generate_Script(text param_schema=tiger_data);
```

### 描述

生成一个脚本，`除指定架`中以 `County_all`、`state_all` 或 `state` 代`开`，后跟 `county` 或 `state` 的所有表。如果您要从 `Tiger_2010` 数据升`到 Tiger_2011` 数据，`需要`行此操作。

可用性：2.1.0

示例

```
SELECT drop_nation_tables_generate_script();
DROP TABLE tiger_data.county_all;
DROP TABLE tiger_data.county_all_lookup;
DROP TABLE tiger_data.state_all;
DROP TABLE tiger_data.ma_county;
DROP TABLE tiger_data.ma_state;
```

相关信息

[Loader\\_Generate\\_Nation\\_Script](#)

### 12.2.3 Drop\_State\_Tables\_Generate\_Script

`Drop_State_Tables_Generate_Script` — 生成一个脚本，删除指定架构中以 `state` 写前缀的所有表。如果未指定架构，默认架构为 `Tiger_data`。

#### Synopsis

```
text Drop_State_Tables_Generate_Script(text param_state, text param_schema=tiger_data);
```

描述

生成一个脚本，删除指定架构中以 `state` 写前缀的所有表。如果未指定架构，默认架构为 `Tiger_data`。此函数于在重新加载状态之前删除状态表非常有用，以防在上次加载时出错。

可用性: 2.0.0

示例

```
SELECT drop_state_tables_generate_script('PA');
DROP TABLE tiger_data.pa_addr;
DROP TABLE tiger_data.pa_county;
DROP TABLE tiger_data.pa_county_lookup;
DROP TABLE tiger_data.pa_cousub;
DROP TABLE tiger_data.pa_edges;
DROP TABLE tiger_data.pa_faces;
DROP TABLE tiger_data.pa_featnames;
DROP TABLE tiger_data.pa_place;
DROP TABLE tiger_data.pa_state;
DROP TABLE tiger_data.pa_zip_lookup_base;
DROP TABLE tiger_data.pa_zip_state;
DROP TABLE tiger_data.pa_zip_state_loc;
```

相关信息

[Loader\\_Generate\\_Script](#)

## 12.2.4 Geocode

**Geocode** — 将地址作文本字符串（或其他规范化地址）输入，并输出一组可能的位置，其中包括 NAD 83 坐标系中的点几何形状、每个位置的规范化地址以及评分。评分越低，匹配的可能性越大。结果首先按最低评分排序。可以输入最大结果数，默认为 10，以及 `restrict_region`（默认为 NULL）

### Synopsis

```
setof record geocode(varchar address, integer max_results=10, geometry restrict_region=NULL,
norm_addy OUT addy, geometry OUT geomout, integer OUT rating);
setof record geocode(norm_addy in_addy, integer max_results=10, geometry restrict_region=NULL,
norm_addy OUT addy, geometry OUT geomout, integer OUT rating);
```

### 描述

将地址作文本字符串（或已规范化的地址）并输出一组可能的位置，其中包括 NAD 83 坐标系中的点几何形状、每个位置的规范化地址 (`addy`) 以及评分。评分越低，匹配的可能性越大。结果首先按最低评分排序。使用 **Tiger** 数据 (`edges`、`faces`、`addr`)、PostgreSQL 模糊字符串匹配 (`soundex`、`levenshtein`) 和 PostGIS 插值函数沿 **Tiger** 插值地址。评分越高，地理点正确的可能性就越小。地理点默认从中心点到街道地址所在 (L/R) 的偏移 10 米。

增强功能：2.0.0 支持 **Tiger** 2010 规范化数据并修改了一些评分以提高地理点的速度和准确性，并将点从中心点偏移 10 米到街道地址所在的一边。新参数 `max_results` 可用于指定最佳结果的数量或返回最佳结果。

### 示例：基本

下面的示例是在一台 3.0 GHZ 处理器 Windows 7 计算机上运行的，计算机具有 2GB 内存，运行 PostgreSQL 9.1rc1/PostGIS 2.0，加载了所有 MA、MN、CA、RI state **Tiger** 数据。

精确匹配的运算速度更快（61 毫秒）

```
SELECT g.rating, ST_X(g.geomout) As lon, ST_Y(g.geomout) As lat,
       (addy).address As stno, (addy).streetname As street,
       (addy).streettypeabbrev As styp, (addy).location As city, (addy).stateabbrev As st,( ←
       addy).zip
FROM geocode('75 State Street, Boston MA 02109', 1) As g;
rating |          lon          |          lat          | stno | street | styp | city | st | zip
-----+-----+-----+-----+-----+-----+-----+-----+-----
0 | -71.0557505845646 | 42.35897920691 | 75 | State | St | Boston | MA | 02109
```

即使 zip 没有输入地理处理器也可以猜测（大约需要 122-150 毫秒）

```
SELECT g.rating, ST_AsText(ST_SnapToGrid(g.geomout,0.00001)) As wktlonlat,
       (addy).address As stno, (addy).streetname As street,
       (addy).streettypeabbrev As styp, (addy).location As city, (addy).stateabbrev As st,( ←
       addy).zip
FROM geocode('226 Hanover Street, Boston, MA',1) As g;
rating |          wktlonlat          | stno | street | styp | city | st | zip
-----+-----+-----+-----+-----+-----+-----+-----
1 | POINT(-71.05528 42.36316) | 226 | Hanover | St | Boston | MA | 02113
```

可以处理拼写错误，并提供不止一种可能的解决方案和评分，并且需要更长的时间（500ms）。

```
SELECT g.rating, ST_AsText(ST_SnapToGrid(g.geomout,0.00001)) As wktlonlat,
       (addy).address As stno, (addy).streetname As street,
       (addy).streettypeabbrev As styp, (addy).location As city, (addy).stateabbrev As st,( ←
       addy).zip
```

```
FROM geocode('31 - 37 Stewart Street, Boston, MA 02116',1) As g;
rating |          wktlonlat          | stno | street | styp | city | st | zip
-----+-----+-----+-----+-----+-----+-----+-----
70 | POINT(-71.06466 42.35114) | 31 | Stuart | St | Boston | MA | 02116
```

用于地址行批量地理编码。最慢的方法是设置 `max_results=1`。地理尚未地理编码的内容（没有地理编码）。

```
CREATE TABLE addresses_to_geocode(addid serial PRIMARY KEY, address text,
    lon numeric, lat numeric, new_address text, rating integer);
```

```
INSERT INTO addresses_to_geocode(address)
VALUES ('529 Main Street, Boston MA, 02129'),
('77 Massachusetts Avenue, Cambridge, MA 02139'),
('25 Wizard of Oz, Walaford, KS 99912323'),
('26 Capen Street, Medford, MA'),
('124 Mount Auburn St, Cambridge, Massachusetts 02138'),
('950 Main Street, Worcester, MA 01610');
```

```
-- only update the first 3 addresses (323-704 ms - there are caching and shared memory ←
    effects so first geocode you do is always slower) --
-- for large numbers of addresses you don't want to update all at once
-- since the whole geocode must commit at once
-- For this example we rejoin with LEFT JOIN
-- and set to rating to -1 rating if no match
-- to ensure we don't regeocode a bad address
```

```
UPDATE addresses_to_geocode
SET (rating, new_address, lon, lat)
= ( COALESCE(g.rating, -1), pprint_addy(g.addy),
    ST_X(g.geomout)::numeric(8,5), ST_Y(g.geomout)::numeric(8,5) )
FROM (SELECT addid, address
    FROM addresses_to_geocode
    WHERE rating IS NULL ORDER BY addid LIMIT 3) As a
LEFT JOIN LATERAL geocode(a.address,1) As g ON true
WHERE a.addid = addresses_to_geocode.addid;
```

result

-----

Query returned successfully: 3 rows affected, 480 ms execution time.

```
SELECT * FROM addresses_to_geocode WHERE rating is not null;
```

```
addid |          address          | lon | lat | ←
-----+-----+-----+-----+-----
1 | 529 Main Street, Boston MA, 02129 | -71.07177 | 42.38357 | 529 Main St, ←
    Boston, MA 02129 | 0
2 | 77 Massachusetts Avenue, Cambridge, MA 02139 | -71.09396 | 42.35961 | 77 ←
    Massachusetts Ave, Cambridge, MA 02139 | 0
3 | 25 Wizard of Oz, Walaford, KS 99912323 | -97.92913 | 38.12717 | Willowbrook, ←
    KS 67502 | 108
(3 rows)
```

示例：使用几何编码器

```
SELECT g.rating, ST_AsText(ST_SnapToGrid(g.geomout,0.00001)) As wktlonlat,
    (addy).address As stno, (addy).streetname As street,
    (addy).streettypeabbrev As styp,
    (addy).location As city, (addy).stateabbrev As st,(addy).zip
FROM geocode('100 Federal Street, MA',
```

```

3,
(SELECT ST_Union(the_geom)
 FROM place WHERE statefp = '25' AND name = 'Lynn')::geometry
) As g;

```

rating	wktlonlat	stno	street	styp	city	st	zip
7	POINT(-70.96796 42.4659)	100	Federal	St	Lynn	MA	01905
16	POINT(-70.96786 42.46853)	NULL	Federal	St	Lynn	MA	01905

(2 rows)

Time: 622.939 ms

## 相关信息

[Normalize\\_Address](#), [Pprint\\_Addy](#), [ST\\_AsText](#), [ST\\_SnapToGrid](#), [ST\\_X](#), [ST\\_Y](#)

## 12.2.5 Geocode\_Intersection

`Geocode_Intersection` — 接收 2 条相交的街道以及 `state`、`city`、`zip`，并返回出位于交叉路口的第一个交叉街道上的一可能位置，返回包括一个 `geomout` 作为 NAD 83 坐标系中的点位置，一个规范化地址 (`addy`) 每个位置以及评分。评分越低，匹配的可能性越大。结果首先按最低评分排序。可以返回最大结果数，默认为 10。使用 Tiger 数据 (`edges`、`faces`、`addr`)、PostgreSQL 模糊字符串匹配 (`soundex`、`levenshtein`)。

### Synopsis

```
setof record geocode_intersection(text roadway1, text roadway2, text in_state, text in_city, text
in_zip, integer max_results=10, norm_addy OUT addy, geometry OUT geomout, integer OUT rating);
```

### 描述

该函数接受 2 条交叉的街道、一个 `state`、`city`、`zip` 作为输入，并返回出位于交叉点上的第一条街道上的可能位置集合。每个位置包括 NAD 83 坐标系中的点几何、每个位置的规范化地址以及评分。评分越低，匹配度越高。结果按照评分从低到高排序。可以返回最大结果数，默认为 10。函数返回每个结果的规范化地址 (`addy`)、NAD 83 坐标系中的点位置 (`geomout`) 以及评分。评分越低，匹配度越高。结果按照评分从低到高排序。此函数使用 Tiger 数据 (`edges`、`faces`、`addr`) 和 PostgreSQL 模糊字符串匹配 (`soundex`、`levenshtein`) 进行计算。

可用性: 2.0.0

### 示例：基本

下面的示例是在一台 3.0 GHZ 处理器 Windows 7 计算机上运行的，计算机具有 2GB RAM，运行 PostgreSQL 9.0/PostGIS 1.5，并加载了所有 MA 状态 Tiger 数据。目前有点慢 (3000 毫秒)

在 Windows 2003 64 位 8GB 上运行 PostGIS 2.0 PostgreSQL 64 位 Tiger 2011 数据加载 -- (41ms)

```

SELECT pprint_addy(addy), st_astext(geomout),rating
      FROM geocode_intersection( 'Haverford St','Germania St', 'MA', 'Boston',
      '02130',1);

```

pprint_addy	st_astext	rating
98 Haverford St, Boston, MA 02130	POINT(-71.101375 42.31376)	0

即使没有在地理想象器中想象 zip，也可以猜 (在 Windows 7 上大花了 3500 毫秒)，在 Windows 2003 64 位上花了 741 毫秒

```
SELECT pprint_addy(addy), st_astext(geomout),rating
      FROM geocode_intersection('Weld', 'School', 'MA', 'Boston');
      pprint_addy      |      st_astext      | rating
-----+-----+-----
98 Weld Ave, Boston, MA 02119 | POINT(-71.099 42.314234) | 3
99 Weld Ave, Boston, MA 02119 | POINT(-71.099 42.314234) | 3
```

相关信息

[Geocode](#), [Pprint\\_Addy](#), [ST\\_AsText](#)

## 12.2.6 Get\_Geocode\_Setting

Get\_Geocode\_Setting — 返回存储在 Tiger.geocode\_settings 表中的特定设置的值。

### Synopsis

text **Get\_Geocode\_Setting**(text setting\_name);

### 描述

返回存储在 Tiger.geocode\_settings 表中的特定设置的值。设置允许您切换功能的值。稍后的计划将是通过设置来控制值。当前设置列表如下：

name	setting	unit	category	↔	short_desc
debug_geocode_address		false	boolean	debug	outputs debug information in notice log such as queries when geocode_address is called if true ↔
debug_geocode_intersection		false	boolean	debug	outputs debug information in notice log such as queries when geocode_intersection is called if true ↔
debug_normalize_address		false	boolean	debug	outputs debug information in notice log such as queries and intermediate expressions when normalize_address is called if true ↔
debug_reverse_geocode		false	boolean	debug	if true, outputs debug information in notice log such as queries and intermediate expressions when reverse_geocode ↔
reverse_geocode_numbered_roads_highways,	0		integer	rating	For state and county name, 0 - no preference in name, 1 - prefer the numbered highway name, 2 - prefer local state/county name ↔
use_pagc_address_parser		false	boolean	normalize	If set to true, will try to use the address_standardizer extension (via pagc_normalize_address) instead of tiger normalize_address built one ↔

更改：2.2.0：默认设置在保存在名 geocode\_settings\_default 的表中。使用自定义设置 a 位于 geocode\_settings 中，并且包含用设置的值。

可用性：2.1.0

返回配置示例

```
SELECT get_geocode_setting('debug_geocode_address') As result;
result
-----
false
```

相关信息

[Set\\_Geocode\\_Setting](#)

## 12.2.7 Get\_Tract

Get\_Tract — 从几何形状所在的区域表中返回人口普查区域或字段。默认返回区域的短名称。

### Synopsis

```
text get_tract(geometry loc_geom, text output_field=name);
```

描述

给定一个几何形状将返回该几何形状的人口普查区位置。如果未指定空参考系，假定 NAD 83 度。

#### Note

此函数使用人口普查的 tract，默认情况下不会加载。如果您已加载了 state 表，可以使用 [Loader\\_Generate\\_Census\\_Script](#) 脚本加载 tract、bg 和 tabblock。



如果您尚未加载形状数据并希望加载这些附加表，执行以下操作

```
UPDATE tiger.loader_lookuptables SET load = true WHERE load = false AND lookup_name <-
    IN('tract', 'bg', 'tabblock');
```

然后它将被 [Loader\\_Generate\\_Script](#) 包含。

可用性: 2.0.0

示例：基本

```
SELECT get_tract(ST_Point(-71.101375, 42.31376) ) As tract_name;
tract_name
-----
1203.01
```

```
--this one returns the tiger geoid
SELECT get_tract(ST_Point(-71.101375, 42.31376), 'tract_id' ) As tract_id;
tract_id
-----
25025120301
```



相关信息

[Geocode](#)>

## 12.2.8 Install\_Missing\_Indexes

`Install_Missing_Indexes` — 查找具有在地理处理器接口和查询条件中使用的列的所有表，这些表缺少这些列上使用的索引，并将添加它们。

### Synopsis

```
boolean Install_Missing_Indexes();
```

### 描述

查找 `Tiger` 和 `Tiger_data` 架构中的所有表，其中包含在地理处理器接口和处理器中使用的列，这些列上缺少索引，并将输出 SQL DDL 来定义这些表的索引，然后执行生成的脚本。它是一个辅助函数，它添加了使查询更快所需的新索引，而这些索引可能在加载过程中丢失。此函数是 [Missing\\_Indexes\\_Generate\\_Script](#) 的配套函数，除了生成创建索引脚本之外，执行它。它作为 `update_geocode.sql` 升级脚本的一部分被调用。

可用性: 2.0.0

### 示例

```
SELECT install_missing_indexes();
       install_missing_indexes
-----
t
```

相关信息

[Loader\\_Generate\\_Script](#), [Missing\\_Indexes\\_Generate\\_Script](#)

## 12.2.9 Loader\_Generate\_Census\_Script

`Loader_Generate_Census_Script` — 指定 `states` 的指定平台生成 shell 脚本，脚本将下列 `Tiger` 人口普查 `state` 区、`bg` 和 `tabblocks` 数据表、保存并添加到 `Tiger_data` 架构中。每个状态脚本都作独立的返回。

### Synopsis

```
setof text loader_generate_census_script(text[] param_states, text os);
```

## 描述

生成一个用于指定平台和指定州的 shell 脚本，脚本将下载 Tiger 数据中的人口普查州 tract、分区 bg 和 tabblocks 数据表，然后将这些数据表分段添加到 tiger\_data 模式中。每个州的脚本将作为一个独立的返回。

它在 Linux 上使用 unzip (默认情况下在 Windows 上使用 7-zip)，并使用 wget 行下载。它使用 Section 4.7.2 来加载数据。注意，它的最小单元是整个州。它只会处理和文件中的文件。

它使用以下控制表来控制进程和不同的操作系统 shell 语法化。

1. loader\_variables 跟踪各种量，例如人口普查站点、年份、数据和存储模式
2. loader\_platform 各种平台的配置文件以及各种可执行文件所在的位置。自 windows 和 linux。可以添加更多。
3. loader\_lookuptables 每条定义一种表（州、区）、是否处理其中的区以及如何加载它。定义每个区的加载数据、存储数据、添加、删除、索引和约束的步骤。每个表都以状态前缀，并承自 Tiger 模式中的表。例如建从 tiger.faces 承的 tiger\_data.ma\_faces

可用性: 2.0.0



### Note

**Loader\_Generate\_Script** 包含此函数，但如果您在 PostGIS 2.0.0 alpha5 之前安装了 Tiger 地理引擎，需要在已完成的状况上运行此函数以得到些附加表。

## 示例

生成脚本以加载 Windows shell 脚本格式的定义状态的数据。

```
SELECT loader_generate_census_script(ARRAY['MA'], 'windows');
-- result --
set STATEDIR="\gisdata\www2.census.gov\geo\pvs\tiger2010st\25_Massachusetts"
set TMPDIR=\gisdata\temp\
set UNZIPTOOL="C:\Program Files\7-Zip\7z.exe"
set WGETTOOL="C:\wget\wget.exe"
set PGBIN=C:\projects\pg\pg91win\bin\
set PGPORT=5432
set PGHOST=localhost
set PGUSER=postgres
set PGPASSWORD=yourpasswordhere
set PGDATABASE=tiger_postgis20
set PSQL="%PGBIN%psql"
set SHP2PGSQL="%PGBIN%shp2pgsql"
cd \gisdata

%WGETTOOL% http://www2.census.gov/geo/pvs/tiger2010st/25_Massachusetts/25/ --no-parent --
  relative --accept=*bg10.zip,*tract10.zip,*tabblock10.zip --mirror --reject=html
del %TMPDIR%*.*/Q
%PSQL% -c "DROP SCHEMA tiger_staging CASCADE;"
%PSQL% -c "CREATE SCHEMA tiger_staging;"
cd %STATEDIR%
for /r %%z in (*.zip) do %UNZIPTOOL% e %%z -o%TMPDIR%
cd %TMPDIR%
%PSQL% -c "CREATE TABLE tiger_data.MA_tract(CONSTRAINT pk_MA_tract PRIMARY KEY (tract_id) )
  INHERITS(tiger.tract); "
%SHP2PGSQL% -c -s 4269 -g the_geom -W "latin1" tl_2010_25_tract10.dbf tiger_staging.
  ma_tract10 | %PSQL%
```

```
%PSQL% -c "ALTER TABLE tiger_staging.MA_tract10 RENAME geoid10 TO tract_id; SELECT ↵
  loader_load_staged_data(lower('MA_tract10'), lower('MA_tract')); "
%PSQL% -c "CREATE INDEX tiger_data_MA_tract_the_geom_gist ON tiger_data.MA_tract USING gist ↵
(the_geom);"
%PSQL% -c "VACUUM ANALYZE tiger_data.MA_tract;"
%PSQL% -c "ALTER TABLE tiger_data.MA_tract ADD CONSTRAINT chk_statefp CHECK (statefp = ↵
'25');"
:
```

生成 sh 脚本

```
STATEDIR="/gisdata/www2.census.gov/geo/pvs/tiger2010st/25_Massachusetts"
TMPDIR="/gisdata/temp/"
UNZIPTOOL=unzip
WGETTOOL="/usr/bin/wget"
export PGBIN=/usr/pgsql-9.0/bin
export PGPORT=5432
export PGHOST=localhost
export PGUSER=postgres
export PGPASSWORD=yourpasswordhere
export PGDATABASE=geocoder
PSQL=${PGBIN}/psql
SHP2PGSQL=${PGBIN}/shp2pgsql
cd /gisdata

wget http://www2.census.gov/geo/pvs/tiger2010st/25_Massachusetts/25/ --no-parent --relative ↵
--accept=*bg10.zip,*tract10.zip,*tabblock10.zip --mirror --reject=html
rm -f ${TMPDIR}/*.*
${PSQL} -c "DROP SCHEMA tiger_staging CASCADE;"
${PSQL} -c "CREATE SCHEMA tiger_staging;"
cd $STATEDIR
for z in *.zip; do $UNZIPTOOL -o -d $TMPDIR $z; done
:
:
```

相关信息

[Loader\\_Generate\\_Script](#)

## 12.2.10 Loader\_Generate\_Script

**Loader\_Generate\_Script** — 指定平台的指定状态生成 shell 脚本，脚本将下 Tiger 数据、存并加到 Tiger\_data 模式中。每个状态脚本都作独立的返回。最新版本支持 Tiger 2010 化，加人口普查区、区和区表。

### Synopsis

```
setof text loader_generate_script(text[] param_states, text os);
```

### 描述

指定平台的指定状态生成 shell 脚本，脚本将下 Tiger 数据、存并加到 Tiger\_data 模式中。每个状态脚本都作独立的返回。

它在 Linux 上使用 `unzip`（在 Windows 上默认为 `7-zip`）和 `wget` 来下载文件。它使用 Section 4.7.2 来加载数据。注意，它的最小单位是整个状态，但您可以通过自己下载文件来覆盖它。它只会管理存档文件和源文件中的文件。

它使用以下控制表来控制进程和不同的操作系统 shell 国际化。

1. `loader_variables` 跟踪各种变量，例如人口普查站点、年份、数据和存档模式
2. `loader_platform` 各种平台的配置文件以及各种可执行文件所在的位置。支持 `windows` 和 `linux`。可以添加更多。
3. `loader_lookuptables` 每条记录定义一种表（州、县）、是否管理其中的记录以及如何加载它们。定义每个记录的加载数据、存档数据、添加、删除、索引和约束的步骤。每个表都以状态名前缀，并继承自 Tiger 模式中的表。例如从 `tiger.faces` 继承的 `tiger_data.ma_faces`

可用性：2.0.0 支持 Tiger 2010 国际化数据并加载人口普查区（`tract`）、区（`bg`）和区（`tabblocks`）表。



### Note

如果您使用 pgAdmin 3，请注意默认情况下 pgAdmin 3 会截断文本。要修复此问题，更改文件 `File -> Options -> Query Tool -> Query Editor -> Max`。每列字符数大于 50000 个字符。

### 示例

使用 `psql`，其中 `gittest` 是您的数据库，`/gisdata/data_load.sh` 是使用要执行的 shell 命令创建的文件。

```
psql -U postgres -h localhost -d gittest -A -t \
-c "SELECT Loader_Generate_Script(ARRAY['MA'], 'gittest')" > /gisdata/data_load.sh;
```

生成脚本以加载 Windows shell 脚本格式的 2 个状态的数据。

```
SELECT loader_generate_script(ARRAY['MA','RI'], 'windows') AS result;
-- result --
set TMPDIR=\gisdata\temp\
set UNZIPTOOL="C:\Program Files\7-Zip\7z.exe"
set WGETTOOL="C:\wget\wget.exe"
set PGBIN=C:\Program Files\PostgreSQL\9.4\bin\
set PGPORT=5432
set PGHOST=localhost
set PGUSER=postgres
set PGPASSWORD=yourpasswordhere
set PGDATABASE=geocoder
set PSQL="%PGBIN%psql"
set SHP2PGSQL="%PGBIN%shp2pgsql"
cd \gisdata

cd \gisdata
%WGETTOOL% ftp://ftp2.census.gov/geo/tiger/TIGER2015/PLACE/tl_*_25_* --no-parent --relative --recursive --level=2 --accept=zip --mirror --reject=html
cd \gisdata/ftp2.census.gov/geo/tiger/TIGER2015/PLACE
:
:
```

生成 sh 脚本

```
SELECT loader_generate_script(ARRAY['MA','RI'], 'sh') AS result;
-- result --
TMPDIR="/gisdata/temp/"
UNZIPTOOL=unzip
```

```

WGETT00L="/usr/bin/wget"
export PGBIN=/usr/lib/postgresql/9.4/bin
-- variables used by psql: https://www.postgresql.org/docs/current/static/libpq-envars.html
export PGPORT=5432
export PGHOST=localhost
export PGUSER=postgres
export PGPASSWORD=yourpasswordhere
export PGDATABASE=geocoder
PSQL=${PGBIN}/psql
SHP2PGSQL=${PGBIN}/shp2pgsql
cd /gisdata

cd /gisdata
wget ftp://ftp2.census.gov/geo/tiger/TIGER2015/PLACE/tl_*_25_* --no-parent --relative -- ←
recursive --level=2 --accept=zip --mirror --reject=html
cd /gisdata/ftp2.census.gov/geo/tiger/TIGER2015/PLACE
rm -f ${TMPDIR}/*.*
:
:

```

## 相关信息

Section [2.4.1, Loader\\_Generate\\_Nation\\_Script, Drop\\_State\\_Tables\\_Generate\\_Script](#)

### 12.2.11 Loader\_Generate\_Nation\_Script

Loader\_Generate\_Nation\_Script — 指定平台生成加到和州找表中的 shell 脚本。

## Synopsis

text loader\_generate\_nation\_script(text os);

## 描述

指定平台生成 shell 脚本，将 County\_all、county\_all\_lookup、state\_all 表加到 Tiger\_data 架中。它分承自 tiger 模式中的 County、county\_lookup、state 表。

它在 Linux 上使用 unzip（在 Windows 上默认 7-zip）和 wget 来行下。它使用 Section [4.7.2](#)来加数据。

它使用以下控制表 Tiger.loader\_platform、tiger.loader\_variables 和 Tiger.loader\_lookuptables 来控制程和不同操作系 shell 法化。

1. loader\_variables 跟踪各种量，例如人口普站点、年份、数据和存模式
2. loader\_platform 各种平台的配置文件以及各种可行文件所在的位置。有 windows 和 linux/unix。可以添加更多。
3. loader\_lookuptables 每条定一种表（州、）、是否理其中的以及如何加它。定每个的入数据、存数据、添加、除列、索引和束的步。每个表都以状前，并承自 Tiger 模式中的表。例如建从 tiger.faces 承的 tiger\_data.ma\_faces

增：2.4.1 政 5 制表区域 (zcta5) 加步已修复，用后，zcta5 数据将作名 zcta5\_all 的个表加，作国家脚本加的一部分。

可用性：2.1.0

**Note**

如果您希望将政区 5 制表区域 (zcta5) 包含在您的国家脚本加中，行以下操作：

```
UPDATE tiger.loader_lookuptables SET load = true WHERE table_name = 'zcta510';
```

**Note**

如果您行的是 Tiger\_2010 版本，并且想要使用新的 Tiger 数据重新加状态，在行此脚本之前，您需要首先加生成并行 drop 句 [Drop\\_Nation\\_Tables\\_Generate\\_Script](#)。

## 示例

生成脚本来加 Windows 的国家数据。

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

生成脚本来加 Linux/Unix 系的数据。

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

## 相关信息

[Loader\\_Generate\\_Script](#), [Drop\\_Nation\\_Tables\\_Generate\\_Script](#)

### 12.2.12 Missing\_Indexes\_Generate\_Script

`Missing_Indexes_Generate_Script` — 找具有在地理器接口中使用的列的所有表，些表缺少些列上的索引，并将出 SQL DDL 来定些表的索引。

#### Synopsis

```
text Missing_Indexes_Generate_Script();
```

#### 描述

找 `tiger` 和 `Tiger_data` 架中的所有表，其中包含在地理器接口中使用的关列，些列上缺少索引，并将出 SQL DDL 来定些表的索引。是一个助函数，它添加了使更快所需的新索引，而些索引可能在加程中失。随着地理器的改，函数将更新以适正在使用的新索引。如果此函数没有出任何内容，意味着您的所有表都具有我了的关索引。

可用性: 2.0.0

示例

```
SELECT missing_indexes_generate_script();
-- output: This was run on a database that was created before many corrections were made to ←
the loading script ---
CREATE INDEX idx_tiger_county_countyfp ON tiger.county USING btree(countyfp);
CREATE INDEX idx_tiger_cousub_countyfp ON tiger.cousub USING btree(countyfp);
CREATE INDEX idx_tiger_edges_tfidr ON tiger.edges USING btree(tfidr);
CREATE INDEX idx_tiger_edges_tfidl ON tiger.edges USING btree(tfidl);
CREATE INDEX idx_tiger_zip_lookup_all_zip ON tiger.zip_lookup_all USING btree(zip);
CREATE INDEX idx_tiger_data_ma_county_countyfp ON tiger_data.ma_county USING btree(countyfp ←
);
CREATE INDEX idx_tiger_data_ma_cousub_countyfp ON tiger_data.ma_cousub USING btree(countyfp ←
);
CREATE INDEX idx_tiger_data_ma_edges_countyfp ON tiger_data.ma_edges USING btree(countyfp);
CREATE INDEX idx_tiger_data_ma_faces_countyfp ON tiger_data.ma_faces USING btree(countyfp);
```

相关信息

[Loader\\_Generate\\_Script, Install\\_Missing\\_Indexes](#)

### 12.2.13 Normalize\_Address

`Normalize_Address` — 给定文本街道地址，返回复合的 `norm_addy` 型，型具有道路后、前和准化型、街道、街道名称等，分独立的字段。函数适用于与 `Tiger_geocoder` 打包的数据（不需要 `tiger` 数据）。

#### Synopsis

```
norm_addy normalize_address(varchar in_address);
```

#### 描述

给定文本街道地址，返回复合的 `norm_addy` 型，型具有道路后、前和准化型、街道、街道名称等，分独立的字段。是地理器中将所有地址准化政形式的第一步。除了地理器打包的数据之外，不需要其他数据。

此函数使用使用 `Tiger_geocoder` 加并位于 `tiger` 模式中的各种方向/状/后找表，因此不需要您下 `tiger` 数据或任何其他附加数据来使用它。您可能会需要向 `Tiger` 模式中的各种找表添加更多写或替代命名。

它使用位于 `tiger` 模式中的各种控制找表来准化入地址。

函数返回的 `norm_addy` 型象中的字段按以下序排列，其中 () 表示地理器所需的字段，[] 表示可字段：

```
(address) [predirAbbrev] (streetName) [streetTypeAbbrev] [postdirAbbrev] [internal] [location] [stateAbbrev] [zip] [parsed] [zip4] [address_alphanumeric]
```

增：2.4.0 `norm_addy` 象包括附加字段 `zip4` 和 `address_alphanumeric`。

1. `address` 是一个整数：街道号
2. `predirAbbrev` 是 `varchar`：道路的方向前，例如 N、S、E、W 等。些是使用 `Direction_lookup` 表行控制的。

3. `streetName` 是 `varchar`
4. `streetTypeAbbrev` `varchar` 街道类型的缩写版本：例如 `St, Ave, Cir`。这些是使用 `street_type_lookup` 表行控制的。
5. `postdirAbbrev` `varchar` 类型是道路 N、S、E、W 等方向的缩写。这些是使用 `Direction_lookup` 表行控制的。
6. `internalvarchar` 类型内部地址，例如公寓或套房号。
7. `location` `varchar` 类型，通常是城市或管辖区省份。
8. `stateAbbrev` `varchar` 类型 2 个字符的美国州。例如 `MA, NY, MI`。这些由 `state_lookup` 表控制。
9. `zip` `varchar` 类型，5 位邮政编码。例如 `02109`。
10. `parsed` `boolean` 类型 - 表示地址是否是由规范化过程形成的。`Normalize_address` 函数在返回地址之前将其置为 `true`。
11. `zip49` 位邮政编码的最后 4 位数字。可用性：PostGIS 2.4.0。
12. `address_alphanumeric` 完整的街道号，即使它具有字母字符(如 `17R`)。此的解析最好使用 `Pagc_Normalize_Address` 函数。可用性：PostGIS 2.4.0。

## 示例

输出字段。如果您想要漂亮的文本输出，使用 `Pprint_Addy`。

```
SELECT address As orig, (g.na).streetname, (g.na).streettypeabbrev
FROM (SELECT address, normalize_address(address) As na
      FROM addresses_to_geocode) As g;
```

orig	streetname	streettypeabbrev
28 Capen Street, Medford, MA	Capen	St
124 Mount Auburn St, Cambridge, Massachusetts 02138	Mount Auburn	St
950 Main Street, Worcester, MA 01610	Main	St
529 Main Street, Boston MA, 02129	Main	St
77 Massachusetts Avenue, Cambridge, MA 02139	Massachusetts	Ave
25 Wizard of Oz, Walaford, KS 99912323	Wizard of Oz	

## 相关信息

[Geocode](#), [Pprint\\_Addy](#)

### 12.2.14 Pagc\_Normalize\_Address

`Pagc_Normalize_Address` — 给定文本街道地址，返回复合的 `norm_addy` 类型，该类型具有道路后缀、前缀和规范化类型、街道、街道名称等，分门别类的字段。该函数适用于与 `Tiger_geocoder` 打包的查找数据（不需要 `tiger` 普查数据）。需要 `address_standardizer` 扩展。

#### Synopsis

```
norm_addy pagc_normalize_address(varchar in_address);
```



## 描述

该函数返回文本街道地址，返回复合的 `norm_addy` 类型，该类型具有道路后缀、前缀和规范化类型、街道、街道名称等，分属独立的字段。该函数是地理信息系统中将所有地址规范化为一致形式的第一步。除了地理信息数据库打包的数据之外，不需要其他数据。

该函数使用各种 `pagc_*` 查找表，这些表由 `Tiger_geocoder` 加载并位于 `Tiger` 模式中，因此不需要您下载 `tiger` 数据或任何其他附加数据来使用它。您可能会需要向 `Tiger` 模式中的各种查找表添加更多写入或替代命名。

它使用位于 `tiger` 模式中的各种控制查找表来规范化输入地址。

该函数返回的 `norm_addy` 类型对象中的字段按以下顺序排列，其中 () 表示地理信息数据库所需的字段，[] 表示可选字段：

**Normalize\_Address** 的大小写和格式略有不同。

可用性：2.1.0



该方法需要 `address_standardizer` 扩展。

`(address) [predirAbbrev] (streetName) [streetTypeAbbrev] [postdirAbbrev] [internal] [location] [stateAbbrev] [zip]`

目前，`address_standardizer` 扩展的本机 `standardaddr` 比 `norm_addy` 丰富一些，因为它旨在支持国际地址（包括国家/地区）。`standardaddr` 等效字段是：

`house_num, predir, name, suftype, sufdir, unit, city, state, postcode`

新增：2.4.0 `norm_addy` 对象包括附加字段 `zip4` 和 `address_alphanumeric`。

1. `address` 是一个整数：街道号
2. `predirAbbrev` 是 `varchar`：道路的方向前缀，例如 N、S、E、W 等。这些是使用 `Direction_lookup` 表行控制的。
3. `streetName` 是 `varchar`
4. `streetTypeAbbrev` `varchar` 街道类型的缩写版本：例如 St, Ave, Cir。这些是使用 `street_type_lookup` 表行控制的。
5. `postdirAbbrev` `varchar` 类型是道路 N、S、E、W 等方向的缩写。这些是使用 `Direction_lookup` 表行控制的。
6. `internal` `varchar` 类型内部地址，例如公寓或套房号。
7. `location` `varchar` 类型，通常是城市或管辖区省份。
8. `stateAbbrev` `varchar` 类型 2 个字符的美国州。例如 MA, NY, MI。这些由 `state_lookup` 表控制。
9. `zip` `varchar` 类型，5 位邮政编码。例如 02109。
10. `parsed` `boolean` 类型 - 表示地址是否是由规范化过程形成的。`Normalize_address` 函数在返回地址之前将其置为 `true`。
11. `zip49` 位邮政编码的最后 4 位数字。可用性：PostGIS 2.4.0。
12. `address_alphanumeric` 完整的街道号，即使它具有字母字符（如 17R）。此解析最好使用 `Pagc_Normalize_Address` 函数。可用性：PostGIS 2.4.0。

示例

二次用示例

```
SELECT addy.*
FROM pagc_normalize_address('9000 E R00 ST STE 999, Springfield, CO') AS addy;
```

address	predirabbrev	streetname	streettypeabbrev	postdirabbrev	internal	location	stateabbrev	zip	parsed
9000	E	R00	ST			SUITE 999	CO		t

批量用。目前, `postgis_tiger_geocoder` 包装 `address_standardizer` 的方式存在速度问题。这些问题有望在以后的版本中得到解决。为了解决这些问题, 如果您需要批量地理数据的速度以批量模式用生成 `normaddy`, 建议您直接用 `address_standardizer standardize_address` 函数, 如下所示, 这与我之前在 `Normalize_Address` 中所做的类似, 使用在 `Geocode` 中构建的数据。

```
WITH g AS (SELECT address, ROW((sa).house_num, (sa).predir, (sa).name
, (sa).suftype, (sa).sufdir, (sa).unit , (sa).city, (sa).state, (sa).postcode, true)::
norm_addy As na
FROM (SELECT address, standardize_address('tiger.pagc_lex'
, 'tiger.pagc_gaz'
, 'tiger.pagc_rules', address) As sa
FROM addresses_to_geocode) As g)
SELECT address As orig, (g.na).streetname, (g.na).streettypeabbrev
FROM g;
```

orig	streetname	streettypeabbrev
529 Main Street, Boston MA, 02129	MAIN	ST
77 Massachusetts Avenue, Cambridge, MA 02139	MASSACHUSETTS	AVE
25 Wizard of Oz, Walaford, KS 99912323	WIZARD OF	
26 Capen Street, Medford, MA	CAPEN	ST
124 Mount Auburn St, Cambridge, Massachusetts 02138	MOUNT AUBURN	ST
950 Main Street, Worcester, MA 01610	MAIN	ST

相关信息

[Normalize\\_Address, Geocode](#)

### 12.2.15 Pprint\_Addy

`Pprint_Addy` — 给定一个 `norm_addy` 复合型对象, 返回它的漂亮的打印表示。通常与 `normalize_address` 合使用。

**Synopsis**

```
varchar pprint_addy(norm_addy in_addy);
```

描述

给定一个 `norm_addy` 复合型对象, 返回它的漂亮的打印表示。除了地理数据器打包的数据之外, 不需要其他数据。通常与 `Normalize_Address` 合使用。

## 示例

漂亮地打印一个地址

```
SELECT pprint_addy(normalize_address('202 East Fremont Street, Las Vegas, Nevada 89101')) ←
  As pretty_address;
      pretty_address
-----
202 E Fremont St, Las Vegas, NV 89101
```

漂亮的打印地址地址表

```
SELECT address As orig, pprint_addy(normalize_address(address)) As pretty_address
FROM addresses_to_geocode;
```

orig	pretty_address
529 Main Street, Boston MA, 02129	529 Main St, Boston MA, 02129
77 Massachusetts Avenue, Cambridge, MA 02139	77 Massachusetts Ave, Cambridge, MA ←
28 Capen Street, Medford, MA	28 Capen St, Medford, MA
124 Mount Auburn St, Cambridge, Massachusetts 02138	124 Mount Auburn St, Cambridge, MA ←
950 Main Street, Worcester, MA 01610	950 Main St, Worcester, MA 01610

## 相关信息

[Normalize\\_Address](#)

### 12.2.16 Reverse\_Geocode

**Reverse\_Geocode** — 取已知空参考系中的几何点并返回一条，其中包含理上可能的地址数和交叉街道数。如果 `include_strnum_range = true`，包括交叉街道中的街道范。

## Synopsis

```
record Reverse_Geocode(geometry pt, boolean include_strnum_range=false, geometry[] OUT intpt,
norm_addy[] OUT addy, varchar[] OUT street);
```

## 描述

取已知空参考中的几何点并返回一条，其中包含理上可能的地址数和十字街道数。如果 `include_strnum_range = true`，包括交叉街道中的街道范。如果未入，`include_strnum_range` 默 `false`。地址根据点最接近的道路行排序，因此第一个地址很可能是正确的地址。

什么我理地址而不是地址。Tiger 数据没有真地址，只有街道范。因此，理地址是基于街道范的内插地址。例如，插入我的地址之一会返回 `26 Court St.` 和 `26 Court Sq.`，尽管不存在 `26 Court Sq.` 的地方。是因一个点可能位于 2 条街道的拐角，因此会沿着条街道行插。假地址沿着街道均匀分布，当然是的，因您可以市政建筑占据街道范的很大一部分，而其余建筑物聚集在末端。

注意：，个功能依赖于 Tiger 数据。如果您没有加覆盖点区域的数据，那么您将得到一条充 NULL 的。

返回的元素如下：

1. `intpt` 是一个点数：这些是街道上最接近入点的中心点。有多少个地址就有多少个点。
2. `addy` 是一个 `norm_addy` (规范化地址) 数组：这些是适合入点的可能地址数组。数组中的第一个最有可能。一般来说只有一个，除非某个点位于 2 或 3 条街道的拐角，或者点位于道路上的某个位置而不是在面。
3. `street varchar` 数组：这些是交叉街道 (或街道) (相交的街道或者是点投影所在的街道)。

增加：2.4.1 如果加了可的 `zcta5` 数据集，即使未加特定的状态数据，`reverse_geocode` 函数也可以解析状态和 `zip`。有关加 `zcta5` 数据的信息，参 [Loader\\_Generate\\_Nation\\_Script](#)。

可用性: 2.0.0

示例

位于条街道拐角但最接近一条街道的点示例。是 MIT 的大致位置：77 Massachusetts Ave, Cambridge, MA 02139 注意，然我没有 3 条街道，但 PostgreSQL 只会高于上限的条目返回 null，因此可以安全使用。包括街道范

```
SELECT pprint_addy(r.addy[1]) As st1, pprint_addy(r.addy[2]) As st2, pprint_addy(r.addy[3]) ←
      As st3,
      array_to_string(r.street, ',') As cross_streets
FROM reverse_geocode(ST_GeomFromText('POINT(-71.093902 42.359446)',4269),true) As r ←
;
```

st1	st2	st3	cross_streets
67 Massachusetts Ave, Cambridge, MA 02139 Vassar St			67 - 127 Massachusetts Ave,32 - 88 ←

在里，我 不包括十字路口的地址范，并了一个非常靠近条街道拐角的位置，因此可以通过个不同的地址来。

```
SELECT pprint_addy(r.addy[1]) As st1, pprint_addy(r.addy[2]) As st2,
pprint_addy(r.addy[3]) As st3, array_to_string(r.street, ',') As cross_str
FROM reverse_geocode(ST_GeomFromText('POINT(-71.06941 42.34225)',4269)) As r;
```

st1	st2	st3	cross_str
5 Bradford St, Boston, MA 02118	49 Waltham St, Boston, MA 02118		Waltham St

于一点，我重用 `Geocode` 中的地理示例，并且只需要主要地址和最多 2 个交叉街道。

```
SELECT actual_addr, lon, lat, pprint_addy((rg).addy[1]) As int_addr1,
      (rg).street[1] As cross1, (rg).street[2] As cross2
FROM (SELECT address As actual_addr, lon, lat,
      reverse_geocode( ST_SetSRID(ST_Point(lon,lat),4326) ) As rg
FROM addresses_to_geocode WHERE rating
> -1) As foo;
```

actual_addr	int_addr1	lon	lat	cross1	cross2

```

529 Main Street, Boston MA, 02129 | -71.07181 | 42.38359 | 527 Main St, ←
  Boston, MA 02129 | Medford St |
77 Massachusetts Avenue, Cambridge, MA 02139 | -71.09428 | 42.35988 | 77 ←
  Massachusetts Ave, Cambridge, MA 02139 | Vassar St |
26 Capen Street, Medford, MA | -71.12377 | 42.41101 | 9 Edison Ave, ←
  Medford, MA 02155 | Capen St | Tesla Ave
124 Mount Auburn St, Cambridge, Massachusetts 02138 | -71.12304 | 42.37328 | 3 University ←
  Rd, Cambridge, MA 02138 | Mount Auburn St |
950 Main Street, Worcester, MA 01610 | -71.82368 | 42.24956 | 3 Maywood St, ←
  Worcester, MA 01603 | Main St | Maywood Pl

```

相关信息

[Pprint\\_Addy](#), [Geocode](#), [Loader\\_Generate\\_Nation\\_Script](#)

## 12.2.17 Topology\_Load\_Tiger

`Topology_Load_Tiger` — 将 `tiger` 数据的定义区域加载到 PostGIS 拓扑中，并将老虎数据与拓扑的空参考，并捕捉到拓扑的精度公差。

### Synopsis

```
text Topology_Load_Tiger(varchar topo_name, varchar region_type, varchar region_id);
```

### 描述

将定义的 `tiger` 数据区域加载到 PostGIS 拓扑中。面、点和线将拓扑的空参考系，点将捕捉到目标拓扑的容差。建的面、点、线与原始 `Tiger` 数据面、点、线保持相同的 ID，以便数据集将来可以更轻松地与 `Tiger` 数据行。返回有关流程的摘要信息。

例如，用于重新划分数据非常有用，在种情况下，您需要新形成的多边形遵循街道的中心线，并且生成的多边形不重叠。



#### Note

功能依赖于 `Tiger` 数据以及 PostGIS 拓扑模块的安装。有关更多信息，参见 [Chapter 9](#) 和 [Section 2.2.3](#)。如果您尚未加载覆盖感兴趣区域的数据，不会创建任何拓扑。如果您没有使用拓扑函数建拓扑，此函数也会失败。



#### Note

大多数拓扑失败都是由于公差造成的，即点后点没有完全重合。除了正种情况，如果拓扑失败，您可能需要提高或降低精度。

所需参数：

- `topo_name` 要加载数据的 PostGIS 拓扑的名称。
- `Region_type` 边界区域的类型。目前支持 `place` 和 `county`。划是再多几个。是用于定义区域界的表格。例如，`tiger.place`、`tiger.county`
- `Region_id` 就是 `TIGER` 所的大地水准面。它是表中区域的唯一标识符。于 `place`，它是 `Tiger.place` 中的 `plcidfp` 列。于 `county`，它是 `Tiger.county` 中的 `cntyidfp` 列

可用性: 2.0.0

**示例 : Boston, Massachusetts 拓扑**

在 Mass State Plane Feet (2249) 中创建塞州波士顿的拓扑，容差 0.25 英尺，然后加到波士顿城市 tiger 面、线、点。

```
SELECT topology.CreateTopology('topo_boston', 2249, 0.25);
createtopology
-----
    15
-- 60,902 ms ~ 1 minute on windows 7 desktop running 9.1 (with 5 states tiger data loaded)
SELECT tiger.topology_load_tiger('topo_boston', 'place', '2507000');
-- topology_loader_tiger --
29722 edges holding in temporary. 11108 faces added. 1875 edges of faces added. 20576 ←
    nodes added.
19962 nodes contained in a face. 0 edge start end corrected. 31597 edges added.

-- 41 ms --
SELECT topology.TopologySummary('topo_boston');
-- topologysummary--
Topology topo_boston (15), SRID 2249, precision 0.25
20576 nodes, 31597 edges, 11109 faces, 0 topogeoms in 0 layers

-- 28,797 ms to validate yeh returned no errors --
SELECT * FROM
    topology.ValidateTopology('topo_boston');

    error          |   id1   |   id2
-----+-----+-----
```

**示例 : Suffolk, Massachusetts 拓扑**

在 Mass State Plane Meters (26986) 中创建塞州福克市的拓扑，容差 0.25 米，然后加到福克市 tiger 面、线、点。

```
SELECT topology.CreateTopology('topo_suffolk', 26986, 0.25);
-- this took 56,275 ms ~ 1 minute on Windows 7 32-bit with 5 states of tiger loaded
-- must have been warmed up after loading boston
SELECT tiger.topology_load_tiger('topo_suffolk', 'county', '25025');
-- topology_loader_tiger --
36003 edges holding in temporary. 13518 faces added. 2172 edges of faces added.
24761 nodes added. 24075 nodes contained in a face. 0 edge start end corrected. 38175 ←
    edges added.
-- 31 ms --
SELECT topology.TopologySummary('topo_suffolk');
-- topologysummary--
Topology topo_suffolk (14), SRID 26986, precision 0.25
24761 nodes, 38175 edges, 13519 faces, 0 topogeoms in 0 layers

-- 33,606 ms to validate --
SELECT * FROM
    topology.ValidateTopology('topo_suffolk');

    error          |   id1   |   id2
-----+-----+-----
coincident nodes | 81045651 | 81064553
edge crosses node | 81045651 | 85737793
edge crosses node | 81045651 | 85742215
edge crosses node | 81045651 | 620628939
edge crosses node | 81064553 | 85697815
edge crosses node | 81064553 | 85728168
```

edge crosses node | 81064553 | 85733413

相关信息

[CreateTopology](#), [CreateTopoGeom](#), [TopologySummary](#), [ValidateTopology](#)

### 12.2.18 Set\_Geocode\_Setting

Set\_Geocode\_Setting — 设置影响地理处理器功能行的设置。

#### Synopsis

```
text Set_Geocode_Setting(text setting_name, text setting_value);
```

#### 描述

设置存储在 Tiger.geocode\_settings 表中的特定设置的。设置允许您切换功能的。稍后的计划将是通过设置来控制。当前设置列表列在 [Get\\_Geocode\\_Setting](#) 中。

可用性：2.1.0

#### 返回设置示例

如果您在函数 true 行 [Geocode](#)，NOTICE 日志将输出和。

```
SELECT set_geocode_setting('debug_geocode_address', 'true') As result;
result
-----
true
```

相关信息

[Get\\_Geocode\\_Setting](#)

## Chapter 13

# PostGIS 特殊函数索引

### 13.1 PostGIS 聚合函数

下面的函数是空聚合函数，其使用方式与 SQL 聚合函数（例如 `sum` 和 `average`）相同。

- **CG\_3DUnion** - Perform 3D union using `postgis_sfcgal`.
- **ST\_3DExtent** - 返回几何图形的 3D 边界框的聚合函数。
- **ST\_3DUnion** - 进行 3D 聚合。
- **ST\_AsFlatGeobuf** - 返回一行行的 FlatGeobuf 表示形式。
- **ST\_AsGeobuf** - 返回一行行的 Geobuf 表示。
- **ST\_AsMVT** - 返回一行行的 MVT 表示形式的聚合函数。
- **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\_ClusterIntersecting** - 将输入几何图形聚集成交集的聚合函数。
- **ST\_ClusterIntersectingWin** - 窗口函数，返回每个输入几何图形的簇 ID，将输入几何图形聚集成交集的集合中。
- **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\_ClusterWithin** - 按间隔距离进行聚合的聚合函数。
- **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合。
- **ST\_Collect** - 从一行几何图形构建 GeometryCollection 或 Multi\* 几何图形。
- **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\_CoverageUnion** - 通过去除共享边界来算形成覆盖范围的一行多边形的并集。
- **ST\_Extent** - 返回几何图形边界框的聚合函数。
- **ST\_MakeLine** - 从 Point, MultiPoint, 或 LineString geometries 构建 LineString。
- **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。
- **ST\_Polygonize** - 计算由一行几何图形的边形成的多边形集合。
- **ST\_SameAlignment** - 如果网格具有相同的斜率、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），返回 true；如果没有注意，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** - 将一个栅格切片的并集返回由 1 个或多个波段组成的栅格。
- **TopoElementArray\_Agg** - 返回一个 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\_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\_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。

## 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** - 进行 3D 差异
- **CG\_3DIntersection** - 进行 3D 相交
- **CG\_3DUnion** - Perform 3D union using postgis\_sfcgal.
- **CG\_Volume** - 计算 3D 体的体积。如果用于表面（甚至组合）几何图形将返回 0。
- **ST\_3DArea** - 计算 3D 表面几何形状的面积。对于固体将返回 0。
- **ST\_3DDWithin** - 检查 3D 几何图形是否在指定的 3D 距离内
- **ST\_3DDifference** - 进行 3D 差异
- **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）（以投影位表示）。
- **ST\_3DIntersection** - 进行 3D 相交
- **ST\_3DIntersects** - 检查两个几何图形在 3D 空间中是否相交 - 适用于点、串、多边形、多面体曲面（区域）
- **ST\_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\_3DUnion** - 进行 3D 组合。
- **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，删除原始面并用两个新面替换它。

- **ST\_AddIsoEdge** - 将由几何 `aliningstring` 定义的孤立边添加到连接两个具有孤立点 `anode` 和 `anothernode` 的拓扑，并返回新边的 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 ID。如果“face”为空 (`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\_AsGML** - 将几何体作 GML 版本 2 或 3 元素返回。
  - **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\_Boundary** - 返回几何体的边界。
  - **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\_ChangeEdgeGeom** - 改变边的形状而不影响拓扑。
  - **ST\_Contains** - 几何 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** - 返回几何体的坐标维度。
  - **ST\_CreateTopoGeo** - 将几何体集合添加到指定的空拓扑并返回成功消息。
  - **ST\_Crosses** - 两个几何体是否有一些（但不是全部）共同的内部点
  - **ST\_CurveN** - Returns the Nth component curve geometry of a CompoundCurve.
  - **ST\_CurveToLine** - 将包含曲线的几何体转换为线性几何体。
  - **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** - 返回几何体的拓扑维数。
  - **ST\_Disjoint** - 两个几何体是否没有共同点
  - **ST\_Distance** - 返回两个几何或地理体之间的距离。
  - **ST\_EndPoint** - 返回 LineString 或 CircularLineString 的最后一个点。
  - **ST\_Envelope** - 返回表示几何体边界框的几何体。
  - **ST\_Equals** - 两个几何体是否包含同一点
  - **ST\_ExteriorRing** - 返回表示多边形外环的 LineString。
  - **ST\_GMLToSQL** - 从 GML 表示返回指定的 ST\_Geometry 名。名是 ST\_GeomFromGML 的别名
  - **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。
  - **ST\_GeomFromText** - 从已知的文本表示 (WKT) 返回指定的 ST\_Geometry 名。
  - **ST\_GeomFromWKB** - 从已知的二进制几何表示 (WKB) 和可选的 SRID 构建几何体。
  - **ST\_GeometryFromText** - 从已知的文本表示 (WKT) 返回指定的 ST\_Geometry 名。名是 ST\_GeomFromText 的别名
  - **ST\_GeometryN** - 返回几何集合的一个元素。
  - **ST\_GeometryType** - 以文本形式返回几何体的 SQL-MM 类型。
  - **ST\_GetFaceEdges** - 返回一个有序的边，这些边界定了 aface。
-

- **ST\_GetFaceGeometry** - 返回给定拓扑中具有指定面 ID 的多边形。
- **ST\_InitTopoGeo** - 建立一个新的拓扑架并将其注册到 topology.topology 表中。
- **ST\_InteriorRingN** - 返回多边形的第 N 个内环（孔）。
- **ST\_Intersection** - 计算表示几何 A 和 B 的共享部分的几何。
- **ST\_Intersects** - 两个几何是否相交（它们至少有一个共同点）
- **ST\_IsClosed** - 两个 LineStrings 的起点和点是否重合。对于多面体表面是否闭合（同心）。
- **ST\_IsEmpty** - 几何是否空。
- **ST\_IsRing** - 几何串是否是闭合的。
- **ST\_IsSimple** - 几何体的自完整性或自接触点。
- **ST\_IsValid** - 几何在 2D 中是否有效。
- **ST\_Length** - 返回线性几何体的二面度。
- **ST\_LineFromText** - 使用指定的 SRID 根据 WKT 表示构建几何。如果未给出 SRID，默认为 0。
- **ST\_LineFromWKB** - 使用指定的 SRID 从 WKB 制作 LINESTRING
- **ST\_LinestringFromWKB** - 使用指定的 SRID 从 WKB 构建几何。
- **ST\_LocateAlong** - 返回几何上与量匹配的点的。
- **ST\_LocateBetween** - 返回与量范围匹配的几何形部分。
- **ST\_M** - 返回点的 M 值。
- **ST\_MLineFromText** - 从 WKT 表示形式返回指定的 ST\_MultiLineString。
- **ST\_MPointFromText** - 使用指定的 SRID 从 WKT 构建几何。如果未给出 SRID，默认为 0。
- **ST\_MPolyFromText** - 使用指定的 SRID 从 WKT 制作多边形几何体。如果未给出 SRID，默认为 0。
- **ST\_ModEdgeHeal** - 通过删除接条的端点、修改第一条并删除第二条来修复条。返回已删除点的 id。
- **ST\_ModEdgeSplit** - 通过沿有新建点、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** - 通过删除接条的端点、删除条并用方向与提供的第一条相同的替它来修复条。
- **ST\_NewEdgesSplit** - 通过沿有新建点、删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** - Return the number of component curves in a CompoundCurve.
- **ST\_NumGeometries** - 返回几何集合中的元素数量。
- **ST\_NumInteriorRings** - 返回多边形的内环（孔）数。
- **ST\_NumPatches** - 返回多面体表面上的面数。对于非多面体几何形状将返回 null。
- **ST\_NumPoints** - 返回 LineString 或 CircularString 中的点数。
- **ST\_OrderingEquals** - 两个几何是否表示相同的几何形并且具有相同方向序的点
- **ST\_Overlaps** - 两个几何是否具有相同的度和相交，但每个几何形至少有一个点不在另一个几何形中

- **ST\_PatchN** - 返回多面形曲面 (PolyhedralSurface) 的第 N 个几何体 (面)。
- **ST\_Perimeter** - 返回多面形几何或地理的边界长度。
- **ST\_Point** - 创建具有 X、Y 和 SRID 的点。
- **ST\_PointFromText** - 使用指定的 SRID 从 WKT 创建点几何。如果未给出 SRID, 默认为未知。
- **ST\_PointFromWKB** - 使用指定的 SRID 从 WKB 创建几何。
- **ST\_PointN** - 返回几何中第一个点串或面串中的第 N 个点。
- **ST\_PointOnSurface** - 计算保证位于多面形或几何体上的点。
- **ST\_Polygon** - 从具有指定 SRID 的线串创建多面形。
- **ST\_PolygonFromText** - 使用指定的 SRID 从 WKT 创建几何。如果未给出 SRID, 默认为 0。
- **ST\_Relate** - 判断两个几何是否具有与交集矩阵模式匹配的拓扑关系, 或计算它们的交集矩阵。
- **ST\_RemEdgeModFace** - 删除一条边, 如果删除将面分开, 删除一个面并修改另一个面以覆盖该面的空间。
- **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。
- **ST\_RemoveIsoEdge** - 删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。
- **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), 会引发异常。
- **ST\_SRID** - 返回几何的空参考系符。
- **ST\_StartPoint** - 返回 LineString 的第一个点。
- **ST\_SymDifference** - 计算表示几何 A 和 B 不相交部分的几何。
- **ST\_Touches** - 判断两个几何是否至少有一个共同点, 但它们的内部不相交。
- **ST\_Transform** - 返回坐标在不同空参考系的新几何。
- **ST\_Union** - 计算表示输入几何的点集并集的几何。
- **ST\_Volume** - 计算 3D 体的体积。如果用于表面 (甚至缝合) 几何将返回 0。
- **ST\_WKBToSQL** - 从已知的二进制表示 (WKB) 返回指定的 ST\_Geometry。是 ST\_GeomFromWKB 的别名, 不取 srid
- **ST\_WKTToSQL** - 从已知的文本表示 (WKT) 返回指定的 ST\_Geometry。是 ST\_GeomFromText 的别名
- **ST\_Within** - 判断 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。
- **ST\_SRID** - 返回拓扑几何的空参考系符。

## 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，它们涉及或出涉及到 **geography** 数据类型。



### 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` 用来行操作。因此，当越日期更、极点以及覆盖多个 UTM 区域的大型几何形或几何形，它们可能不会按期行。基本 - (有利于 UTM、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 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 元数据返回几何形的已知文本 (WKT) 表示形式。
- **ST\_AsGML** - 将几何形作 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** - 将几何形作 KML 元素返回。
- **ST\_AsSVG** - 返回几何体的 SVG 路径数据。
- **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\_Azimuth** - 返回点之直线的基于北方的方位角。
- **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\_ClosestPoint** - 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\_CoveredBy** - 是否 A 的每个点是否都位于 B 中
- **ST\_Covers** - 是否 B 的每个点是否都位于 A 中
- **ST\_DWithin** - 是否两个几何形是否在定距离内
- **ST\_Distance** - 返回两个几何或地理之间的距离。
- **ST\_GeogFromText** - 从已知的文本表示或展 (WKT) 返回指定的地理。
- **ST\_GeogFromWKB** - 从已知的二进制几何表示 (WKB) 或展的已知的二进制 (EWKB) 建地理例。
- **ST\_GeographyFromText** - 从已知的文本表示或展 (WKT) 返回指定的地理。
- **=** - 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\_Intersection** - 计算表示几何 A 和 B 的共享部分的几何。
- **ST\_Intersects** - 是否两个几何形是否相交 (它们至少有一个共同点)
- **ST\_Length** - 返回线性几何体的二进制度。
- **ST\_LineInterpolatePoint** - 返回沿在百分比指示位置的插点。
- **ST\_LineInterpolatePoints** - 返回沿直以分数隔插的点。
- **ST\_LineLocatePoint** - 返回上最接近点的分数位置。
- **ST\_LineSubstring** - 返回个小数位置之的直部分。
- **ST\_Perimeter** - 返回多边形几何或地理的界度。

- **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，其线段不短于指定距离。
- **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，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。按字母顺序列出。

- **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，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，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，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，返回 TRUE。
- **&<** - 如果 A 的边界框位于 B 的左方，返回 TRUE。
- **&>** - 如果 A 的边界框位于 B 的右方，返回 TRUE。
- **~=** - 如果 A 的边界框与 B 的边界框相同，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** - 返回栅格的熟知的二进制 (WKB) 表示形式。
- **ST\_AsGDALRaster** - 以指定的 GDAL 栅格格式返回栅格数据。栅格格式是 GDAL 支持的格式之一。使用 `ST_GDALDrivers()` 获取您的系统支持的格式列表。
- **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合成影像输出 (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\_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，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。
- **ST\_AsRaster** - 将 PostGIS 几何图形转换为 PostGIS 栅格。
- **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，将栅格使用所有波段。
- **ST\_Aspect** - 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\_Band** - 返回具有栅格的一个或多个波段作为新栅格。用于从具有栅格新建栅格非常有用。
- **ST\_BandFileSize** - 返回文件系统中存储的波段的文件大小。如果未指定波段号，假定 1。



- **ST\_BandFileTimestamp** - 返回文件系中存中的波段的文件戳。如果未指定波段号，假定波段 1。
- **ST\_BandIsNoData** - 如果波段填充无数据，返回 true。
- **ST\_BandMetaData** - 返回特定格波段的基本元数据。如果未指定，假定波段号 1。
- **ST\_BandNoDataValue** - 返回表示无数据的定波段中的。如果没有指定波段号，默波段 1。
- **ST\_BandPath** - 返回存文件系中的波段的系文件路径。如果未指定波段号，假定 1。
- **ST\_BandPixelType** - 返回定波段的像素型。如果未指定波段号，假定 1。
- **ST\_Clip** - Returns the raster clipped by the input geometry. If band number is not specified, all bands are processed. If crop is not specified or TRUE, the output raster is cropped. If touched is set to TRUE, then touched pixels are included, otherwise only if the center of the pixel is in the geometry it is included.
- **ST\_ColorMap** - 根据源格和指定波段建最多四个 8BUI 波段（灰度、RGB、RGBA）的新格。如果未指定，假定波段 1。
- **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。
- **ST\_ContainsProperly** - 如果 rastB 与 rastA 的内部相交，但不与 rastA 的界或外部相交，返回 true。
- **ST\_Contour** - 使用 GDAL 廓算法从提供的格波段生成一矢量廓。
- **ST\_ConvexHull** - 返回格的凸包几何形状，包括等于 BandNoDataValue 的像素。于形状和非斜格，出与 ST\_Envelope 相同的果，因此不形状或斜格有用。
- **ST\_Count** - 返回格或格覆盖范的定波段中的像素数。如果未指定 band，默 band 1。如果 except\_nodata\_value 置 true，算不等于 nodata 的像素。
- **ST\_CountAgg** - 的。返回一格的定波段中的像素数。如果未指定 band，默 band 1。如果 except\_nodata\_value 置 true，算不等于 NODATA 的像素。
- **ST\_CoveredBy** - 如果格 rastA 中没有点位于格 rastB 之外，返回 true。
- **ST\_Covers** - 如果格 rastB 中没有点位于格 rastA 之外，返回 true。
- **ST\_DFullyWithin** - 如果格 rastA 和 rastB 彼此完全在指定距离内，返回 true。
- **ST\_DWithin** - 如果格 rastA 和 rastB 彼此之的距离在指定距离内，返回 true。
- **ST\_Disjoint** - 如果格 rastA 在空上不 与 rastB 相交，返回 true。
- **ST\_DumpAsPolygons** - 从定的格中返回一 geomval (geom,val) 行。如果未指定波段号，波段号默 1。
- **ST\_DumpValues** - 取指定 band（波段）的作二数。
- **ST\_Envelope** - 返回格范的多形表示形式。
- **ST\_FromGDALRaster** - 从受支持的 GDAL 格文件返回格。
- **ST\_GeoReference** - 返回 GDAL 或 ESRI 格式的地理配准元数据，如世界文件中常的格式。默 GDAL。
- **ST\_Grayscale** - 根据源格和代表色、色和色的指定波段建新的 1-8BUI 波段格
- **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。
- **ST\_Height** - 返回格的高度（以像素位）。
- **ST\_HillShade** - 使用提供的方位角、高度、亮度和比例入返回高程格的假照明。
- **ST\_Histogram** - 返回一，了格或格覆盖数据分布，其中包括分离的分箱范。如果未指定，将自算分箱数。

- **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 作表面高程。
- **ST\_Intersection** - 返回一个栅格或一个几何像素集，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\_Intersects** - 如果栅格 rastA 与栅格 rastB 空相交，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** - 用空栅格网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）。如果指定了一个栅格，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：通过输入栅格波段和提供的像素型用有效的 PostgreSQL 代数算来建新波段栅格。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型用有效的 PostgreSQL 代数算来建新波段栅格。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 - 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型上用有效的 PostgreSQL 函数来建新波段栅格。如果未指定波段，假定波段 1。如果未指定，范型默为 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 表式的情况下，返回波段栅格。
- **ST\_MemSize** - 返回栅格占用的空量（以字节位）。
- **ST\_MetaData** - 返回有关栅格象的基本元数据，例如像素大小、旋转（斜）、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 的二精度数，指定波段像素由 columnX 和 rowY 或以与栅格相同的空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，如果未说明，说明原因。
- **ST\_NumBands** - 返回栅格象中的波段数。
- **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、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。



- **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 坐标。
- **ST\_PixelHeight** - 返回空参考系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** - 返回空参考系的几何位置的像素宽度。
- **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。
- **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。列和行的号数从 1 开始。
- **ST\_RasterToWorldCoordY** - 返回栅格、列和行的左上角的几何 Y 坐标。列和行的号数从 1 开始。
- **ST\_Reclass** - 构建由从原始数据重新分段的波段型成的新栅格。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 波段重分类为 8BUI 等，以便更好地呈现栅格格式。
- **ST\_Resample** - 重采一个栅格像，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\_Resize** - 将栅格大小调整新的宽度/高度
- **ST\_Reskew** - 通过调整栅格的斜率 (或旋转参数) 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 或 Lanczos 重采样算法计算的。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。
- **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)，返回 true；如果没有注意栅格，返回 false。
- **ST\_ScaleX** - 返回像素宽度的 X 分量 (以空参考系坐标)。
- **ST\_ScaleY** - 返回像素高度的 Y 分量 (以空参考系坐标)。
- **ST\_SetBandIndex** - 更新 out-db band 的 external band 号
- **ST\_SetBandIsNoData** - 将栅格的 isnodata 标志置为 TRUE。
- **ST\_SetBandNoDataValue** - 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。要将波段设置为没有 no data 值，置 no data 值 = NULL。
- **ST\_SetBandPath** - 更新 out-db band 的外部路径和 band 号
- **ST\_SetGeoReference** - 在一次调用中置 Georeference 6 地理配准参数。数字调用用空格分隔。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栅格复制到 M 度。
- **ST\_SetRotation** - 以弧度置栅格的旋。
- **ST\_SetSRID** - 将栅格的 SRID 置在 Spatial\_ref\_sys 表中定义的特定整数 srid。
- **ST\_SetScale** - 以坐标参考系置像素的 X 和 Y 大小。数字位/像素度/高度。
- **ST\_SetSkew** - 置地理参考 X 和 Y 斜（或旋参数）。如果只输入一个，将 X 和 Y 置相同的。
- **ST\_SetUpperLeft** - 将栅格左上角的置投影的 X 和 Y 坐标。
- **ST\_SetValue** - 返回修改后的栅格，其果是将定波段中的置指定列 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，默认 1。
- **ST\_SetValues** - 返回通置定波段的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栅格复制到 Z 度。
- **ST\_SkewX** - 返回地理参考 X 斜（或旋参数）。
- **ST\_SkewY** - 返回地理参考 Y 斜（或旋参数）。
- **ST\_Slope** - 返回高程栅格的坡度（默认以度位）。于分析地形很有用。
- **ST\_SnapToGrid** - 通将栅格捕捉到网格来重采样栅格。新的像素是使用 NearestNeighbor（英或美式拼写）、Bilinear, Cubic, CubicSpline 或 Lanczos 重采样算法算的。默认 NearestNeighbor。
- **ST\_Summary** - 返回栅格内容的文本摘要。
- **ST\_SummaryStats** - 返回一信息，包括定栅格或栅格覆盖的栅格的数、和、均、准差、最小和最大。如果未指定号，假定 1。
- **ST\_SummaryStatsAgg** - 的。返回一栅格的定栅格波段的摘要信息，其中包含数、和、平均、准差、最小、最大。如果未指定号，假定 1。
- **ST\_TPI** - 返回一个算出的地形位置指数（Topographic Position Index）的栅格。
- **ST\_TRI** - 返回具有算的地形固性指数的栅格。
- **ST\_Tile** - 返回根据出栅格的所需度分割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。有 NearestNeighbor, Bilinear, Cubic, CubicSpline, Lanczos（默认 NearestNeighbor）。
- **ST\_Union** - 将一栅格切片的并集返回由 1 个或多个波段成的个栅格。
- **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 像素或特定几何点定波段的。波段号从 1 开始，如果未指定，默认 1。如果将参数 exclude\_nodata\_value 置 false，所有像素都被与 no data 像素相交并返回其。如果未置参数 exclude\_nodata\_value，从栅格的元数据中取。
- **ST\_ValueCount** - 返回一，其中包含像素以及具有定集的栅格（或栅格覆盖范）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。出像素中的所有其他，并将像素四舍五入到最接近的整数。
- **ST\_Width** - 返回栅格的度（以像素位）。
- **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** - 给定几何 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 (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

下面列出的函数是 PostGIS 函数, 它们将一或几个 **Geometry\_dump** 或 **geomval** 数据类型对象作输入或返回作输出。

- **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。
- **ST\_Intersection** - 返回一个栅格或一行几何像素, 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** - 返回几何物件的一行 **geometry\_dump** 行。
- **ST\_DumpPoints** - 返回几何形中坐标的一行 **geometry\_dump** 行。
- **ST\_DumpRings** - 返回多边形外环和内环的一行 **geometry\_dump** 行。
- **ST\_DumpSegments** - 几何形中的各个段返回一行 **geometry\_dump** 行。

## 13.7 PostGIS 边界框函数

下面列出的函数是 PostGIS 函数, 它们将 **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** - 返回表示几何形的 2D 范围的 BOX2D。
- **Box3D** - 返回表示几何体 3D 范围的 BOX3D。
- **Box3D** - 返回栅格封框的 box 3d 表示形式。
- **ST\_3DExtent** - 返回几何形的 3D 边界框的聚合函数。
- **ST\_3DMakeBox** - 构建由几个 3D 点几何形定义的 BOX3D。
- **ST\_AsMVTGeom** - 将几何形 MVT 瓦片的坐标空。
- **ST\_AsTWKB** - 返回几何形式 TWKB, 又名“微小的已知的二进制”
- **ST\_Box2dFromGeoHash** - 从 GeoHash 字符串返回 BOX2D。
- **ST\_ClipByBox2D** - 计算几何形落在矩形内的部分。
- **ST\_EstimatedExtent** - 返回空表的估计范围。
- **ST\_Expand** - 返回从一个边界框或几何形扩展的边界框。
- **ST\_Extent** - 返回几何形边界框的聚合函数。

- **ST\_MakeBox2D** - 构建由  $n$  个 2D 点几何体形定义的 BOX2D。
- **ST\_RemoveIrrelevantPointsForView** - Removes points that are irrelevant for rendering a specific rectangular view of a geometry.
- **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\_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\_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。
- **RemoveUnusedPrimitives** - 删除定义 TopoGeometry 对象不需要的拓扑基元。
- **ValidateTopology** - 返回一个 validate\_topology\_returntype 对象，指明拓扑。
- **~(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) 列添加到表。
- **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** - 进行 3D 差异
- **CG\_3DIntersection** - 进行 3D 相交
- **CG\_3DUnion** - Perform 3D union using postgis\_sfcgal.
- **CG\_ApproximateMedialAxis** - 计算几何区域的近似中轴。
- **CG\_ConstrainedDelaunayTriangles** - 返回定义入几何体的约束 Delaunay 三角剖分。

- **CG\_Extrude** - 将曲面 $\rightarrow$ 出到相关体 $\rightarrow$
  - **CG\_ForceLHR** -  $\rightarrow$ 制 LHR 方向
  - **CG\_IsPlanar** -  $\rightarrow$  $\rightarrow$ 表面是否平坦
  - **CG\_IsSolid** -  $\rightarrow$  $\rightarrow$ 几何体是否 $\rightarrow$ 体。不 $\rightarrow$ 行有效性 $\rightarrow$ 。
  - **CG\_MakeSolid** - 将几何体 $\rightarrow$ 造成 $\rightarrow$ 体。不 $\rightarrow$ 行任何 $\rightarrow$ 。要 $\rightarrow$ 得有效的 $\rightarrow$ 体， $\rightarrow$ 入几何 $\rightarrow$ 形必 $\rightarrow$ 是 $\rightarrow$ 合多面体曲面或 $\rightarrow$ 合 TIN。
  - **CG\_Orientation** - 确定表面方向
  - **CG\_StraightSkeleton** - 从几何体 $\rightarrow$ 算直骨架
  - **CG\_Tessellate** -  $\rightarrow$ 多 $\rightarrow$ 形或多面体表面 $\rightarrow$ 行曲面 $\rightarrow$ 分，并以 TIN 或 TINS 集合的形式返回
  - **CG\_Visibility** -  $\rightarrow$ 算一个从点或多 $\rightarrow$ 形几何中的 $\rightarrow$ 段生成的可 $\rightarrow$ 性多 $\rightarrow$ 形
  - **CG\_Volume** -  $\rightarrow$ 算 3D  $\rightarrow$ 体的体 $\rightarrow$ 。如果 $\rightarrow$ 用于表面（甚至 $\rightarrow$ 合）几何 $\rightarrow$ 形将返回 0。
  - **DropGeometryColumn** - 从空 $\rightarrow$ 表中移除 geometry（几何）列。
  - **GeometryType** - 以文本形式返回几何的 $\rightarrow$ 型。
  - **ST\_3DArea** -  $\rightarrow$ 算 3D 表面几何形状的面 $\rightarrow$ 。于固体将返回 0。
  - **ST\_3DClosestPoint** - 返回 g1 上最接近 g2 的 3D 点。 $\rightarrow$ 是 3D 最短 $\rightarrow$ 的第一个点。
  - **ST\_3DConvexHull** -  $\rightarrow$ 算几何体的 3D 凸包。
  - **ST\_3DDFullyWithin** -  $\rightarrow$  $\rightarrow$ 个 3D 几何 $\rightarrow$ 形是否完全在 $\rightarrow$ 定的 3D 距离内
  - **ST\_3DDWithin** -  $\rightarrow$  $\rightarrow$ 个 3D 几何 $\rightarrow$ 形是否在 $\rightarrow$ 定的 3D 距离内
  - **ST\_3DDifference** -  $\rightarrow$ 行 3D 差异
  - **ST\_3DDistance** - 返回 $\rightarrow$ 个几何 $\rightarrow$ 形之 $\rightarrow$ 的 3D 笛卡 $\rightarrow$ 最小距离（基于空 $\rightarrow$ 参考）（以投影 $\rightarrow$ 位表示）。
  - **ST\_3DExtent** - 返回几何 $\rightarrow$ 形的 3D  $\rightarrow$ 界框的聚合函数。
  - **ST\_3DIntersection** -  $\rightarrow$ 行 3D 相交
  - **ST\_3DIntersects** -  $\rightarrow$  $\rightarrow$ 个几何 $\rightarrow$ 形在 3D 空 $\rightarrow$ 中是否相交 -  $\rightarrow$ 适用于点、 $\rightarrow$ 串、多 $\rightarrow$ 形、多面体曲面（区域）
  - **ST\_3DLength** - 返回 $\rightarrow$ 性几何体的 3D  $\rightarrow$ 度。
  - **ST\_3DLineInterpolatePoint** - 返回沿 3D  $\rightarrow$ 的小数指示位置插 $\rightarrow$ 的点。
  - **ST\_3DLongestLine** - 返回 $\rightarrow$ 个几何体之 $\rightarrow$ 的 3D 最 $\rightarrow$ 直 $\rightarrow$
  - **ST\_3DMaxDistance** - 返回 $\rightarrow$ 个几何 $\rightarrow$ 形之 $\rightarrow$ 的 3D 笛卡 $\rightarrow$ 最大距离（基于空 $\rightarrow$ 参考）（以投影 $\rightarrow$ 位表示）。
  - **ST\_3DPerimeter** - 返回多 $\rightarrow$ 形几何体的 3D 周 $\rightarrow$ 。
  - **ST\_3DShortestLine** - 返回 $\rightarrow$ 个几何 $\rightarrow$ 形之 $\rightarrow$ 的 3D 最短 $\rightarrow$
  - **ST\_3DUnion** -  $\rightarrow$ 行 3D  $\rightarrow$ 合。
  - **ST\_AddMeasure** - 沿 $\rightarrow$ 性几何形状插 $\rightarrow$  $\rightarrow$ 量 $\rightarrow$ 。
  - **ST\_AddPoint** - 将点添加到 $\rightarrow$ 串（LineString）。
  - **ST\_Affine** -  $\rightarrow$ 几何体 $\rightarrow$ 用 3D 仿射 $\rightarrow$ 。
  - **ST\_ApproximateMedialAxis** -  $\rightarrow$ 算几何区域的近似中 $\rightarrow$ 。
  - **ST\_AsBinary** - 返回不 $\rightarrow$  SRID 元数据的几何/地理的 OGC/ISO 已知的二 $\rightarrow$ 制 (WKB) 表示形式。
-



- **ST\_AsEWKB** - 返回具有 SRID 元数据的几何图形的展已知的二进制 (EWKB) 表示形式。
- **ST\_AsEWKT** - 使用 SRID 元数据返回几何图形的已知文本 (WKT) 表示形式。
- **ST\_AsGML** - 将几何图形作 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 (作文本) 的几何图形。
- **ST\_AsKML** - 将几何图形作 KML 元素返回。
- **ST\_AsX3D** - 返回 X3D xml 点元素格式的几何图形 : ISO-IEC-19776-1.2-X3DEncodings-XML
- **ST\_Boundary** - 返回几何图形的边界。
- **ST\_BoundingDiagonal** - 返回几何图界框的角。
- **ST\_CPAWithin** - 判断条迹的最近接近点是否在指定距离内。
- **ST\_ChaikinSmoothing** - 使用 Chaikin 算法返回几何图形的平滑版本
- **ST\_ClosestPointOfApproach** - 返回条迹最接近点的量。
- **ST\_Collect** - 从一几何图形构建 GeometryCollection 或 Multi\* 几何图形。
- **ST\_ConstrainedDelaunayTriangles** - 返回判断定入几何体的束 Delaunay 三角剖分。
- **ST\_ConvexHull** - 计算几何体的凸包。
- **ST\_CoordDim** - 返回几何体的坐度。
- **ST\_CurveN** - Returns the Nth component curve geometry of a CompoundCurve.
- **ST\_CurveToLine** - 将包含曲线的几何图形性几何图形。
- **ST\_DelaunayTriangles** - 返回几何体点的 Delaunay 三角剖分。
- **ST\_Difference** - 计算表示几何 A 中不与几何 B 相交的部分的几何。
- **ST\_DistanceCPA** - 返回条迹的最近接近点之的距离。
- **ST\_Dump** - 返回几何图件的一 geometry\_dump 行。
- **ST\_DumpPoints** - 返回几何图形中坐的一 geometry\_dump 行。
- **ST\_DumpRings** - 返回多形外和内的一 geometry\_dump 行。
- **ST\_DumpSegments** - 几何图形中的各个段返回一 geometry\_dump 行。
- **ST\_EndPoint** - 返回 LineString 或 CircularLineString 的最后一个点。
- **ST\_ExteriorRing** - 返回表示多形外形的 LineString。
- **ST\_Extrude** - 将曲面出到相关体
- **ST\_FlipCoordinates** - 返回 X 和 Y 的几何图形版本。
- **ST\_Force2D** - 制几何图形入 “二进制模式”。
- **ST\_ForceCurve** - 如果适用, 将一个几何图形上升到其曲线型。
- **ST\_ForceLHR** - 制 LHR 方向
- **ST\_ForcePolygonCCW** - 将所有外形逆定向, 将所有内形定向。
- **ST\_ForcePolygonCW** - 方向整所有外形, 逆方向整所有内形。
- **ST\_ForceRHR** - 制多形点的方向遵循右手定。

- **ST\_ForceSFS** - 强制几何图形使用 SFS 1.1 几何类型。
  - **ST\_Force3D** - 强制几何图形入 XYZ 模式。是 ST\_Force3DZ 的别名。
  - **ST\_Force3DZ** - 强制几何图形入 XYZ 模式。
  - **ST\_Force4D** - 强制几何图形入 XYZM 模式。
  - **ST\_ForceCollection** - 将几何图形几何集合 (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** - 将几何图形的 GML 表示形式作入并出 PostGIS 几何象
  - **ST\_GeomFromGeoJSON** - 将几何图形的 geojson 表示形式作入并出 PostGIS 几何象
  - **ST\_GeomFromKML** - 将几何图形的 KML 表示形式作入并出 PostGIS 几何象
  - **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 类型。
  - **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** - 返回多图形的第 N 个内 (孔)。
  - **ST\_InterpolatePoint** - 返回最接近点的几何图形的插量。
  - **ST\_Intersection** - 算表示几何 A 和 B 的共享部分的几何。
  - **ST\_IsClosed** - LineStrings 的起点和点是否重合。于多面体表面是否合 (心)。
  - **ST\_IsCollection** - 几何类型是否几何集合。
  - **ST\_IsPlanar** - 表面是否平坦
  - **ST\_IsPolygonCCW** - 多图形是否具有逆方向的外和方向的内。
  - **ST\_IsPolygonCW** - 多图形是否具有外和逆内。
  - **ST\_IsSimple** - 几何体的自完整性或自接触点。
  - **ST\_IsSolid** - 几何体是否体。不行有效性。
  - **ST\_IsValidTrajectory** - 几何图形是否有效迹。
  - **ST\_LengthSpheroid** - 返回球体上度/度几何体的 2D 或 3D 度/周。
  - **ST\_LineFromMultiPoint** - 从多点几何建串。
  - **ST\_LineInterpolatePoint** - 返回沿在百分比指示位置的插点。
  - **ST\_LineInterpolatePoints** - 返回沿直以分数隔插的点。
  - **ST\_LineSubstring** - 返回个小数位置之的直部分。
  - **ST\_LineToCurve** - 将性几何图形曲线几何图形。
  - **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 。
-

- **ST\_MakeLine** - 从 Point, MultiPoint, 或 LineString geometries 构建 LineString。
  - **ST\_MakePoint** - 构建 2D、3DZ 或 4D 点。
  - **ST\_MakePolygon** - 从壳和可空的孔列表构建多边形。
  - **ST\_MakeSolid** - 将几何体造成实体。不行任何操作。要得到有效的实体，输入几何体必须是闭合多面体曲面或符合 TIN。
  - **ST\_MakeValid** - 在不丢失点的情况下使无效几何体有效。
  - **ST\_MemSize** - 返回几何体占用的内存空间量。
  - **ST\_MemUnion** - 聚合函数，以内存高效但速度较慢的方式聚合几何体
  - **ST\_NDims** - 返回几何体的坐度。
  - **ST\_NPoints** - 返回几何体中的点数（点）。
  - **ST\_NRings** - 返回多边形几何中的环数。
  - **ST\_Node** - 点是环的集合。
  - **ST\_NumCurves** - Return the number of component curves in a CompoundCurve.
  - **ST\_NumGeometries** - 返回几何集合中的元素数量。
  - **ST\_NumPatches** - 返回多面体表面上的面数。对于非多面体几何形状将返回 null。
  - **ST\_Orientation** - 确定表面方向
  - **ST\_PatchN** - 返回多边形曲面 (PolyhedralSurface) 的第 N 个几何体 (面)。
  - **ST\_PointFromWKB** - 使用指定的 SRID 从 WKB 构建几何体
  - **ST\_PointN** - 返回几何体中第一个点串或环串中的第 N 个点。
  - **ST\_PointOnSurface** - 计算保证位于多边形或几何体上的点。
  - **ST\_Points** - 返回包含几何坐度的 MultiPoint。
  - **ST\_Polygon** - 从具有指定 SRID 的环串构建多边形。
  - **ST\_RemovePoint** - 从环串中删除一个点。
  - **ST\_RemoveRepeatedPoints** - 返回除了重复点的几何体。
  - **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体。
  - **ST\_Scroll** - 更改闭合环串的起点。
  - **ST\_SetPoint** - 用指定点替换环串的点。
  - **ST\_ShiftLongitude** - 在 -180-180 和 0-360 之间移动几何体的经度坐度。
  - **ST\_SnapToGrid** - 将输入几何体的所有点捕捉到指定网格。
  - **ST\_StartPoint** - 返回 LineString 的第一个点。
-



- **ST\_StraightSkeleton** - 从几何体计算直骨架
- **ST\_SwapOrdinates** - 返回更改后的几何图形，其中交换了固定的坐标。
- **ST\_SymDifference** - 计算表示几何图形 A 和 B 不相交部分的几何图形。
- **ST\_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
- **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\_Volume** - 计算 3D 体的体积。如果用于表面（甚至组合）几何图形将返回 0。
- **ST\_WrapX** - 将几何体包裹在 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\_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\_Z** - 返回点的 Z 坐标。
- **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\_Zmflag** - 返回指示几何体的 ZM 坐标度的代码。
- **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 边界框与 B 的 n 边界框相交，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。

## 13.9 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** - 返回表示几何图形的 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几何型。
- **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** - 返回几何形的 3D 边界框的聚合函数。
- **ST\_Affine** - 几何体用 3D 仿射变换。
- **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\_AsHEXEWKB** - 使用小端 (NDR) 或大端 (XDR) 返回 HEXEWKB 格式 (作文本) 的几何形。
- **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的簇 id 的窗口函数。
- **ST\_ClusterWithin** - 按间隔距离几何形进行聚合的聚合函数。
- **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合。
- **ST\_Collect** - 从一个几何形构建 GeometryCollection 或 Multi\* 几何形。
- **ST\_CoordDim** - 返回几何体的坐标维度。
- **ST\_CurveToLine** - 将包含曲线的几何形转换为线性几何形。
- **ST\_Distance** - 返回两个几何或地理形之间的距离。
- **ST\_Dump** - 返回几何形的一行 geometry\_dump 行。
- **ST\_DumpPoints** - 返回几何形中坐标的一行 geometry\_dump 行。
- **ST\_EndPoint** - 返回 LineString 或 CircularLineString 的最后一个点。
- **ST\_EstimatedExtent** - 返回空表的估计范围。
- **ST\_FlipCoordinates** - 返回 X 和 Y 坐标的几何形版本。
- **ST\_Force2D** - 强制几何形进入“二维模式”。
- **ST\_ForceCurve** - 如果适用，将一个几何形上升到其曲线型。
- **ST\_ForceSFS** - 强制几何形使用 SFS 1.1 几何形。
- **ST\_Force3D** - 强制几何形进入 XYZ 模式。是 ST\_Force3DZ 的别名。
- **ST\_Force3DM** - 强制几何形进入 XYM 模式。
- **ST\_Force3DZ** - 强制几何形进入 XYZ 模式。
- **ST\_Force4D** - 强制几何形进入 XYZM 模式。
- **ST\_ForceCollection** - 将几何形转换为几何集合 (GEOMETRYCOLLECTION)。
- **ST\_GeoHash** - 返回几何形的 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) 构建地理形。
- **ST\_GeomFromEWKB** - 从扩展已知的二进制表示 (EWKB) 返回指定的 ST\_Geometry 形。

- **ST\_GeomFromEWKT** - 从展已知的文本表示 (EWKT) 返回指定的 ST\_Geometry 。
- **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) 和可选的 SRID 构建几何。
- **ST\_GeometryN** - 返回几何集合的一个元素。
- **=** - 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。
- **ST\_HasArc** - 几何形是否包含弧
- **ST\_Intersects** - 两个几何形是否相交 (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\_IsCollection** - 几何型是否几何集合。
- **ST\_IsEmpty** - 几何形是否空。
- **ST\_LineToCurve** - 将性几何形曲线几何形。
- **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\_NRings** - 返回多形几何中的数。
- **ST\_PointFromWKB** - 使用可选的 SRID 从 WKB 构建几何形
- **ST\_PointN** - 返回几何形中第一个串或形串中的第 N 个点。
- **ST\_Points** - 返回包含几何坐的 MultiPoint。
- **ST\_Rotate** - 原点旋几何体。
- **ST\_RotateZ** - Z 旋几何体。
- **ST\_SRID** - 返回几何形的空参考符。
- **ST\_Scale** - 按定因子放几何形。
- **ST\_SetSRID** - 在几何体上置 SRID。
- **ST\_StartPoint** - 返回 LineString 的第一个点。
- **ST\_Summary** - 返回几何内容的文本摘要。
- **ST\_SwapOrdinates** - 返回更改后的几何形，其中交了定的坐。
- **ST\_TransScale** - 按定的偏移量和系数平移和放几何形。
- **ST\_Transform** - 返回坐不同空参考系的新几何形。
- **ST\_Translate** - 按定的偏移量平移几何形。
- **ST\_XMax** - 返回 2D 或 3D 界框或几何体的 X 最大。
- **ST\_XMin** - 返回 2D 或 3D 界框或几何体的 X 最小。
- **ST\_YMax** - 返回 2D 或 3D 界框或几何体的 Y 最大。
- **ST\_YMin** - 返回 2D 或 3D 界框或几何体的 Y 最小。
- **ST\_ZMax** - 返回 2D 或 3D 界框或几何体的 Z 最大。
- **ST\_ZMin** - 返回 2D 或 3D 界框或几何体的 Z 最小。

- **ST\_Zmflag** - 返回指示几何体的 ZM 坐標度的代碼。
- **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 边界框包含 2D 浮点精度边界框 (BOX2DF), 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 (BOX2DF) 相交, 返回 TRUE。
- **&&&(gidx,geometry)** - 如果 n 浮点精度边界框 (BOX2DF) 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 (BOX2DF) 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 几何形状的 PostGIS 函数

- **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。
- **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** - 进行 3D 差异
- **CG\_3DIntersection** - 进行 3D 相交
- **CG\_3DUnion** - Perform 3D union using postgis\_sfcgal.
- **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\_IsPlanar** - 检查表面是否平坦
- **CG\_IsSolid** - 检查几何体是否是固体。不行有效性。

- **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。
  - **CG\_StraightSkeleton** - 从几何体☐算直骨架
  - **CG\_Tessellate** - ☐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\_Visibility** - ☐算一个从点或多☐形几何中的☐段生成的可☐性多☐形
  - **CG\_Volume** - ☐算 3D ☐体的体☐。如果☐用于表面（甚至☐合）几何☐形将返回 0。
  - **GeometryType** - 以文本形式返回几何的☐型。
  - **ST\_3DArea** - ☐算 3D 表面几何形状的面☐。☐于固体将返回 0。
  - **ST\_3DClosestPoint** - 返回 g1 上最接近 g2 的 3D 点。☐是 3D 最短☐的第一个点。
  - **ST\_3DConvexHull** - ☐算几何体的 3D 凸包。
  - **ST\_3DDFullyWithin** - ☐☐☐个 3D 几何☐形是否完全在☐定的 3D 距离内
  - **ST\_3DDWithin** - ☐☐☐个 3D 几何☐形是否在☐定的 3D 距离内
  - **ST\_3DDifference** - ☐行 3D 差异
  - **ST\_3DDistance** - 返回☐个几何☐形之☐的 3D 笛卡☐最小距离（基于空☐参考）（以投影☐位表示）。
  - **ST\_3DExtent** - 返回几何☐形的 3D ☐界框的聚合函数。
  - **ST\_3DIntersection** - ☐行 3D 相交
  - **ST\_3DIntersects** - ☐☐☐个几何☐形在 3D 空☐中是否相交 - ☐适用于点、☐串、多☐形、多面体曲面（区域）
  - **ST\_3DLongestLine** - 返回☐个几何体之☐的 3D 最☐直☐
  - **ST\_3DMaxDistance** - 返回☐个几何☐形之☐的 3D 笛卡☐最大距离（基于空☐参考）（以投影☐位表示）。
  - **ST\_3DShortestLine** - 返回☐个几何☐形之☐的 3D 最短☐
  - **ST\_3DUnion** - ☐行 3D ☐合。
  - **ST\_Affine** - ☐几何体☐用 3D 仿射☐☐。
  - **ST\_ApproximateMedialAxis** - ☐算几何区域的近似中☐。
  - **ST\_Area** - 返回多☐形几何体的面☐。
  - **ST\_AsBinary** - 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** - 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** - 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☐ GML 版本 2 或 3 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\_CoordDim** - 返回几何体的坐☐☐度。
  - **ST\_Dimension** - 返回几何☐形的拓扑☐数。
  - **ST\_Dump** - 返回几何☐件的一☐ geometry\_dump 行。
  - **ST\_DumpPoints** - 返回几何☐形中坐☐的一☐ geometry\_dump 行。
  - **ST\_Expand** - 返回从☐一个☐界框或几何☐形☐展的☐界框。
  - **ST\_Extent** - 返回几何☐形☐界框的聚合函数。
-





- **ST\_Extrude** - 将曲面 $\square$ 出到相关体 $\square$
  - **ST\_FlipCoordinates** - 返回 X  $\square$ 和 Y  $\square\square\square$ 的几何 $\square$ 形版本。
  - **ST\_Force2D** -  $\square$ 制几何 $\square$ 形 $\square$ 入 “二 $\square$ 模式”。
  - **ST\_ForceLHR** -  $\square$ 制 LHR 方向
  - **ST\_ForceRHR** -  $\square$ 制多 $\square$ 形 $\square$ 点的方向遵循右手定 $\square$ 。
  - **ST\_ForceSFS** -  $\square$ 制几何 $\square$ 形 $\square$ 使用 SFS 1.1 几何 $\square$ 型。
  - **ST\_Force3D** -  $\square$ 制几何 $\square$ 形 $\square$ 入 XYZ 模式。 $\square$ 是 ST\_Force3DZ 的 $\square$ 名。
  - **ST\_Force3DZ** -  $\square$ 制几何 $\square$ 形 $\square$ 入 XYZ 模式。
  - **ST\_ForceCollection** - 将几何 $\square$ 形 $\square\square\square$ 几何集合 (GEOMETRYCOLLECTION)。
  - **ST\_GeomFromEWKB** - 从 $\square$ 展已知的二 $\square$ 制表示 (EWKB) 返回指定的 ST\_Geometry  $\square$ 。
  - **ST\_GeomFromEWKT** - 从 $\square$ 展已知的文本表示 (EWKT) 返回指定的 ST\_Geometry  $\square$ 。
  - **ST\_GeomFromGML** - 将几何 $\square$ 形的 GML 表示形式作 $\square\square$ 入并 $\square$ 出 PostGIS 几何 $\square$ 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 $\square$ 形的 SQL-MM  $\square$ 型。
  - **=** - 如果几何/地理 A 的坐 $\square$ 和坐 $\square\square$ 序与几何/地理 B 的坐 $\square$ 和坐 $\square\square$ 序相同,  $\square$ 返回 TRUE。
  - **&<|** - 如果 A 的 $\square$ 界框与 B 的 $\square$ 界框重 $\square$ 或低于 B 的 $\square$ 界框,  $\square$ 返回 TRUE。
  - **~=** - 如果 A 的 $\square$ 界框与 B 的 $\square$ 界框相同,  $\square$ 返回 TRUE。
  - **ST\_IsClosed** -  $\square\square$  LineStrings 的起点和 $\square$ 点是否重合。 $\square$ 于多面体表面 $\square\square$ 是否 $\square$ 合 ( $\square$ 心)。
  - **ST\_IsPlanar** -  $\square\square$ 表面是否平坦
  - **ST\_IsSolid** -  $\square\square$ 几何体是否 $\square$ 体。不 $\square$ 行有效性 $\square\square$ 。
  - **ST\_MakeSolid** - 将几何体 $\square$ 造成 $\square$ 体。不 $\square$ 行任何 $\square\square$ 。要 $\square$ 得有效的 $\square$ 体,  $\square$ 入几何 $\square$ 形必 $\square$ 是 $\square$ 合多面体曲面或 $\square$ 合 TIN。
  - **ST\_MemSize** - 返回几何 $\square$ 形占用的内存空 $\square\square$ 量。
  - **ST\_NPoints** - 返回几何 $\square$ 形中的点数 ( $\square$ 点)。
  - **ST\_NumGeometries** - 返回几何集合中的元素数量。
  - **ST\_NumPatches** - 返回多面体表面上的面数。 $\square$ 于非多面体几何形状将返回 null。
  - **ST\_PatchN** - 返回多 $\square$ 形曲面 (PolyhedralSurface) 的第 N 个几何体 (面)。
  - **ST\_RemoveRepeatedPoints** - 返回 $\square$ 除了重复点的几何 $\square$ 形。
  - **ST\_Reverse** - 返回 $\square$ 点 $\square$ 序相反的几何体。
  - **ST\_Rotate** -  $\square$ 原点旋 $\square$ 几何体。
  - **ST\_RotateX** -  $\square$  X  $\square$ 旋 $\square$ 几何体。
  - **ST\_RotateY** -  $\square$  Y  $\square$ 旋 $\square$ 几何体。
  - **ST\_RotateZ** -  $\square$  Z  $\square$ 旋 $\square$ 几何体。
  - **ST\_Scale** - 按 $\square$ 定因子 $\square$ 放几何 $\square$ 形。
  - **ST\_ShiftLongitude** - 在 -180-180 和 0-360 之 $\square$ 移 $\square$ 几何 $\square$ 形的 $\square$ 度坐 $\square$ 。
-



- **ST\_StraightSkeleton** - 从几何体计算直骨架
- **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，其中交换了固定的坐标。
- **ST\_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
- **ST\_Transform** - 返回坐标在不同空参考系的新几何图形。
- **ST\_Volume** - 计算 3D 体的体积。如果用于表面（甚至缝合）几何图形将返回 0。
- **~(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 n 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 n 个 n 浮点精度边界框 (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型或它遵守的 OGC/SQL 合性。

- A  表示函数本身适用于型或子型。
- A  意味着它可以工作，但会使用内置的操作，通常将其限制几何型、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。这是在地理数据处理中可能遇到的一些注意事项。
- A  表示函数适用于型，因为它会自行一个型（例如 box3d），而不是直接型支持。
- 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 中的基本 2D 几何 形 (具有 Z 或 M 坐 )。
- PS - 多面体曲面
- T - 三角形和不 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	✓						
ST_PointZM	✓						
ST_Polygon	✓		✓		✓		
ST_TileEnvelope	✓						
ST_HexagonGrid	✓						
ST_Hexagon	✓						
ST_SquareGrid	✓						
ST_Square	✓						
ST_Letters	✓						
GeometryType	✓		✓	✓		✓	✓
ST_Boundary	✓		✓		✓		
ST_BoundingDiagonal	✓		✓				
ST_CoordDim	✓		✓	✓	✓	✓	✓
ST_Dimension	✓				✓	✓	✓
ST_Dump	✓		✓	✓		✓	✓
ST_DumpPoints	✓		✓	✓		✓	✓
ST_DumpSegments	✓		✓				✓
ST_DumpRings	✓		✓				
ST_EndPoint	✓		✓	✓	✓		
ST_Envelope	✓				✓		
ST_ExteriorRing	✓		✓		✓		



函数	geom	geog	2.5D	Curves	SQL MM	PS	T
ST_GeometryN	✓		✓	✓	✓	✓	✓
ST_GeometryType	✓		✓		✓	✓	
ST_HasArc	✓		✓	✓			
ST_InteriorRingN	✓		✓		✓		
ST_NumCurves	✓		✓		✓		
ST_CurveN	✓		✓		✓		
ST_IsClosed	✓		✓	✓	✓	✓	
ST_IsCollection	✓		✓	✓			
ST_IsEmpty	✓			✓	✓		
ST_IsPolygonCCW	✓		✓				
ST_IsPolygonCW	✓		✓				
ST_IsRing	✓				✓		
ST_IsSimple	✓		✓		✓		
ST_M	✓		✓		✓		
ST_MemSize	✓		✓	✓		✓	✓
ST_NDims	✓		✓				
ST_NPoints	✓		✓	✓		✓	
ST_NRings	✓		✓	✓			
ST_NumGeometries	✓		✓		✓	✓	✓
ST_NumInteriorRings	✓				✓		
ST_NumInteriorRing	✓						
ST_NumPatches	✓		✓		✓	✓	
ST_NumPoints	✓				✓		
ST_PatchN	✓		✓		✓	✓	
ST_PointN	✓		✓	✓	✓		
ST_Points	✓		✓	✓			
ST_StartPoint	✓		✓	✓	✓		
ST_Summary	✓	✓		✓		✓	✓
ST_X	✓		✓		✓		
ST_Y	✓		✓		✓		
ST_Z	✓		✓		✓		
ST_Zmflag	✓		✓	✓			
ST_HasZ	✓		✓				
ST_HasM	✓		✓				

函数	geom	geog	2.5D	Curves	SQL MM	PS	T
ST_AddPoint	✓		✓				
ST_CollectionExtract	✓						
ST_CollectionHomogenize	✓						
ST_CurveToLine	✓		✓	✓	✓		
ST_Scroll	✓		✓				
ST_FlipCoordinates	✓		✓	✓		✓	✓
ST_Force2D	✓		✓	✓		✓	
ST_Force3D	✓		✓	✓		✓	
ST_Force3DZ	✓		✓	✓		✓	
ST_Force3DM	✓			✓			
ST_Force4D	✓		✓	✓			
ST_ForceCollection	✓		✓	✓		✓	
ST_ForceCurve	✓		✓	✓			
ST_ForcePolygonCW	✓		✓				
ST_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	✓	rmPipeline					
ST_SetSRID	✓			✓			
ST_SRID	✓			✓	✓		
ST_Transform	✓			✓	✓	✓	
ST_TransformPipeline	✓						
postgis_srs_codes							
postgis_srs							
postgis_srs_all							
postgis_srs_search	✓						
ST_BdPolyFromText	✓						
ST_BdMPolyFromText	✓						
ST_GeogFromText		✓					
ST_GeographyFromText		✓					
ST_GeomCollFromText	✓				✓		
ST_GeomFromEWKT			✓	✓		✓	✓
ST_GeomFromMARC21	✓						
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_LineFromWKB	✓				✓		
ST_LinestringFromWKB	✓				✓		

函数	geom	geog	2.5D	Curves	SQL MM	PS	T
ST_PointFromW	✓		✓	✓	✓		
ST_WKBToSQL	✓				✓		
ST_Box2dFromGeoHash	✓						
ST_GeomFromGeoHash	✓						
ST_GeomFromGeoJSON	✓		✓			✓	✓
ST_GeomFromGeoJSON	✓		✓				
ST_GeomFromKML	✓		✓				
ST_GeomFromIPv4	✓						
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,buffer)	✓			✓		✓	
&&(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_ConcaveHull	✓						
ST_ConvexHull	✓		✓		✓		
ST_DelaunayTriangles	✓		✓				✓
ST_FilterByM	✓						
ST_GeneratePoints	✓						
ST_GeometricMean	✓		✓				
ST_LineMerge	✓						
ST_MaximumInscribedCircle	✓						
ST_LargestEmptyCircle	✓						
ST_MinimumBoundingCircle	✓						
ST_MinimumBoundingRadius	✓						
ST_OrientedEnvelope	✓						
ST_OffsetCurve	✓						
ST_PointOnSurface	✓		✓		✓		






函数	geom	geog	2.5D	Curves	SQL MM	PS	T
ST_Polygonize	✓						
ST_ReducePrecision	✓						
ST_SharedPaths	✓						
ST_Simplify	✓						
ST_SimplifyPreserveTopology	✓						
ST_SimplifyPolygonHull	✓						
ST_SimplifyVW	✓						
ST_SetEffectiveArea	✓						
ST_TriangulatePolygon	✓						
ST_VoronoiLines	✓						
ST_VoronoiPolygons	✓						
ST_CoverageIntersectionEdges	✓						
ST_CoverageSimplify	✓						
ST_CoverageUnion	✓						
ST_Affine	✓		✓	✓		✓	✓
ST_Rotate	✓		✓	✓		✓	✓
ST_RotateX	✓		✓			✓	✓
ST_RotateY	✓		✓			✓	✓
ST_RotateZ	✓		✓	✓		✓	✓
ST_Scale	✓		✓	✓		✓	✓
ST_Translate	✓		✓	✓			
ST_TransScale	✓		✓	✓			
ST_ClusterDBSCAN	✓			✓			
ST_ClusterIntersecting	✓						
ST_ClusterIntersectingWin	✓						
ST_ClusterKMeans	✓						
ST_ClusterWithin	✓			✓			
ST_ClusterWithin/in	✓			✓			
Box2D	✓			✓		✓	✓
Box3D	✓		✓	✓		✓	✓
ST_EstimatedExtent	✓			✓			
ST_Expand	✓					✓	✓
ST_Extent	✓					✓	✓
ST_3DExtent	✓		✓	✓		✓	✓



函数	geom	geog	2.5D	Curves	SQL MM	PS	T
ST_MakeBox2D	✓						
ST_3DMakeBox	✓						
ST_XMax	✓		✓	✓			
ST_XMin	✓		✓	✓			
ST_YMax	✓		✓	✓			
ST_YMin	✓		✓	✓			
ST_ZMax	✓		✓	✓			
ST_ZMin	✓		✓	✓			
ST_LineInterpol	✓ Point	✓	✓				
ST_3DLineInter	✓ atePoint		✓				
ST_LineInterpol	✓ Points	✓	✓				
ST_LineLocateF	✓ t	✓					
ST_LineSubstrin	✓	✓	✓				
ST_LocateAlong	✓				✓		
ST_LocateBetw	✓				✓		
ST_LocateBetw	✓ Elevations		✓				
ST_InterpolateF	✓ t		✓				
ST_AddMeasure	✓		✓				
ST_IsValidTraje	✓ ry		✓				
ST_ClosestPoint	✓ Approach		✓				
ST_DistanceCP	✓		✓				
ST_CPAWithin	✓		✓				
postgis.backend							
postgis.gdal_datapath							
postgis.gdal_enabled_drivers							
postgis.enable_outdb_rasters							
postgis.gdal_vsi_options							
PostGIS_AddBB	✓			✓			
PostGIS_DropBl	✓			✓			
PostGIS_HasBB	✓			✓			
postgis_sfcgal_version							
postgis_sfcgal_full_version							
CG_ForceLHR							
CG_IsPlanar							
CG_IsSolid							

函数	geom	geog	2.5D	Curves	SQL MM	PS	T
CG_MakeSolid 							
CG_Orientation 							
CG_Area 							
CG_3DArea 							
CG_Volume 							
ST_ForceLHR 							
ST_IsPlanar 							
ST_IsSolid 							
ST_MakeSolid 							
ST_Orientation 							
ST_3DArea 							
ST_Volume 							
CG_Intersection 							
CG_Intersects 							
CG_3DIntersec 							
CG_Difference 							
ST_3DDifferenc 							
CG_3DDifferenc 							
CG_Distance 							
CG_3DDistance 							
ST_3DConvexH 							
CG_3DConvexH 							
ST_3DIntersect 							
CG_3DIntersec 							


函数	geom	geog	2.5D	Curves	SQL MM	PS	T
CG_Union							
ST_3DUnion							
CG_3DUnion							
ST_AlphaShape ✓							
CG_AlphaShape							
CG_ApproxConvexPartition							
ST_ApproximateMedialAxis							
CG_ApproximateMedialAxis							
ST_ConstrainedDelaunayTriangles							
CG_ConstrainedDelaunayTriangles							
ST_Extrude							
CG_Extrude							
CG_ExtrudeStraightSkeleton							
CG_GreeneApproximateConvexPartition							
ST_MinkowskiSum							
CG_MinkowskiSum							
ST_OptimalAlphaShape							
CG_OptimalAlphaShape							
CG_OptimalConvexPartition							
CG_StraightSkeleton							
ST_StraightSkeleton							
ST_Tessellate							
CG_Tessellate							
CG_Triangulate							

函数	geom	geog	2.5D	Curves	SQL MM	PS	T
CG_Visibility							
CG_YMonotone							
getfaceedges_returntype							
TopoGeometry							
validatetopology_returntype							
TopoElement							
TopoElementArray							
AddTopoGeometryColumn							
RenameTopoGeometryColumn							
DropTopology							
RenameTopology							
DropTopoGeometryColumn							
Populate_Topology_Layer							
TopologySummary							
ValidateTopology	✓						
ValidateTopologyRelation							
FindTopology							
FindLayer							
CreateTopology							
CopyTopology							
ST_InitTopoGeo					✓		
ST_CreateTopoGeo	✓				✓		
TopoGeo_AddPolygon	✓						
TopoGeo_AddLineString	✓						
TopoGeo_AddPoint	✓						
TopoGeo_LoadGeometry	✓						
ST_AddIsoNode	✓				✓		
ST_AddIsoEdge	✓				✓		
ST_AddEdgeNewFace	✓				✓		
ST_AddEdgeModifyFace	✓				✓		
ST_RemEdgeNewFace					✓		
ST_RemEdgeModifyFace					✓		
ST_ChangeEdgeGeometry	✓				✓		
ST_ModEdgeSplit	✓				✓		
ST_ModEdgeHeal					✓		
ST_NewEdgeHeal					✓		
ST_MoveIsoNode	✓				✓		
ST_NewEdgesSplit	✓				✓		
ST_RemoveIsoNode					✓		
ST_RemoveIsoEdge					✓		

函数	geom	geog	2.5D	Curves	SQL MM	PS	T
GetEdgeByPoint	✓						
GetFaceByPoint	✓						
GetFaceContainsPoint	✓						
GetNodeByPoint	✓						
GetTopologyID							
GetTopologySRID							
GetTopologyName							
ST_GetFaceEdges					✓		
ST_GetFaceGeometry	✓ try				✓		
GetRingEdges							
GetNodeEdges							
Polygonize							
AddNode	✓						
AddEdge	✓						
AddFace	✓						
ST_Simplify	✓						
RemoveUnusedVertices	✓						
CreateTopoGeometry	✓						
toTopoGeometry	✓						
TopoElementArray_Agg							
TopoElement	✓						
clearTopoGeometry	✓						
TopoGeometry_addElement	✓						
TopoGeometry_removeElement	✓						
TopoGeometry_addTopoGeometry	✓						
toTopoGeometry							
GetTopoGeometryElementArray							
GetTopoGeometryElements							
ST_SRID	✓				✓		
AsGML	✓						
AsTopoJSON	✓						
Equals	✓		✓				
Intersects	✓		✓				
geomval							
addbandarg							
rastbandarg							
raster							
reclassarg							
summarystats							
unionarg							
AddRasterConstraints							

函数	geom	geog	2.5D	Curves	SQL MM	PS	T
DropRasterConstraints							
AddOverviewConstraints							
DropOverviewConstraints							
PostGIS_GDAL_Version							
PostGIS_Raster_Lib_Build_Date							
PostGIS_Raster_Lib_Version							
ST_GDALDrivers							
ST_Contour							
ST_InterpolateF <sup>✓</sup> er							
UpdateRasterSRID							
ST_CreateOverview							
ST_AddBand							
ST_AsRaster <sup>✓</sup>							
ST_Band							
ST_MakeEmptyCoverage							
ST_MakeEmptyRaster							
ST_Tile							
ST_Retile <sup>✓</sup>							
ST_FromGDALRaster							
ST_GeoReference							
ST_Height							
ST_IsEmpty							
ST_MemSize							
ST_MetaData							
ST_NumBands							
ST_PixelHeight							
ST_PixelWidth							
ST_ScaleX							
ST_ScaleY							
ST_RasterToWorldCoord							
ST_RasterToWorldCoordX							
ST_RasterToWorldCoordY							
ST_Rotation							
ST_SkewX							
ST_SkewY							
ST_SRID							
ST_Summary							
ST_UpperLeftX							
ST_UpperLeftY							
ST_Width							
ST_WorldToRas <sup>✓</sup> Coord							
ST_WorldToRas <sup>✓</sup> CoordX							
ST_WorldToRas <sup>✓</sup> CoordY							
ST_BandMetaData							
ST_BandNoDataValue							
ST_BandIsNoData							
ST_BandPath							
ST_BandFileSize							
ST_BandFileTimestamp							
ST_BandPixelType							
ST_MinPossibleValue							
ST_HasNoBand							

函数	geom	geog	2.5D	Curves	SQL MM	PS	T
ST_PixelAsPolygons	✓						
ST_PixelAsPoints							
ST_PixelAsCentroids	✓ 1						
ST_Value	✓						
ST_NearestValue	✓						
ST_SetZ	✓						
ST_SetM	✓						
ST_Neighborhood	✓						
ST_SetValue	✓						
ST_SetValues							
ST_DumpValues							
ST_PixelOfValue							
ST_SetGeoReference							
ST_SetRotation							
ST_SetScale							
ST_SetSkew							
ST_SetSRID							
ST_SetUpperLeft							
ST_Resample							
ST_Rescale							
ST_Reskew							
ST_SnapToGrid							
ST_Resize							
ST_Transform							
ST_SetBandNoDataValue							
ST_SetBandIsNoData							
ST_SetBandPath							
ST_SetBandIndex							
ST_Count							
ST_CountAgg							
ST_Histogram							
ST_Quantile							
ST_SummaryStats							
ST_SummaryStatsAgg							
ST_ValueCount							
ST_RastFromWKB							
ST_RastFromHexWKB							
ST_AsBinary/ST_AsWKB							
ST_AsHexWKB							
ST_AsGDALRaster							
ST_AsJPEG							
ST_AsPNG							
ST_AsTIFF							
ST_Clip	✓						
ST_ColorMap							
ST_Grayscale							

函数	geom	geog	2.5D	Curves	SQL MM	PS	T
ST_Intersection	✓						
ST_MapAlgebra (callback function version)							
ST_MapAlgebra (expres- sion version)							
ST_MapAlgebraExpr							
ST_MapAlgebraExpr							
ST_MapAlgebraFct							
ST_MapAlgebraFct							
ST_MapAlgebraFctNgb							
ST_Reclass							
ST_Union							
ST_Distinct4ma							
ST_InvDistWeight4ma							
ST_Max4ma							
ST_Mean4ma							
ST_Min4ma							
ST_MinDist4ma							
ST_Range4ma							
ST_StdDev4ma							
ST_Sum4ma							
ST_Aspect							
ST_HillShade							
ST_Roughness							
ST_Slope							
ST_TPI							
ST_TRI							
Box3D							
ST_ConvexHull	✓						
ST_DumpAsPolygons							
ST_Envelope	✓						
ST_MinConvexHull	✓						
ST_Polygon	✓						
&&	✓						
&<							
&>							
=							
@	✓						
~ =							
~	✓						
ST_Contains							
ST_ContainsProperly							
ST_Covers							
ST_CoveredBy							
ST_Disjoint							



函数	geom	geog	2.5D	Curves	SQL MM	PS	T
ST_Intersects	✓						
ST_Overlaps							
ST_Touches							
ST_SameAlignment							
ST_NotSameAlignmentReason							
ST_Within							
ST_DWithin							
ST_DFullyWithin							
stdaddr							
rules table							
lex table							
gaz table							
debug_standardize_address							
parse_address							
standardize_address							
Drop_Indexes_Generate_Script							
Drop_Nation_Tables_Generate_Script							
Drop_State_Tables_Generate_Script							
Geocode	✓						
Geocode_Inters <del>ion</del>	✓						
Get_Geocode_Setting							
Get_Tract	✓						
Install_Missing_Indexes							
Loader_Generate_Census_Script							
Loader_Generate_Script							
Loader_Generate_Nation_Script							
Missing_Indexes_Generate_Script							
Normalize_Address							
Pagc_Normalize_Address							
Pprint_Addy							
Reverse_Geococ <del>e</del>	✓						
Topology_Load_Tiger							
Set_Geocode_Setting							

## 13.12 新的、增~~加~~的或更改的 PostGIS 函数

### 13.12.1 PostGIS 新增功能或增~~加~~功能 (3.5)

下面~~列~~出的功能是添加或增~~加~~的 PostGIS 功能。

PostGIS 中的新功能 3.5

- **CG\_3DArea** - 可用性 : 3.5.0 ~~可~~算 3D 表面几何形状的面~~积~~。 ~~对~~于固体将返回 0。
- **CG\_3DConvexHull** - 可用性 : 3.5.0 ~~可~~算几何体的 3D 凸包。
- **CG\_3DDifference** - 可用性 : 3.5.0 ~~可~~行 3D 差异
- **CG\_3DDistance** - 可用性 : 3.5.0 Computes the minimum 3D distance between two geometries
- **CG\_3DIntersection** - 可用性 : 3.5.0 ~~可~~行 3D 相交

- **CG\_3DIntersects** - 可用性 : 3.5.0 Tests if two 3D geometries intersect
  - **CG\_3DUnion** - 可用性 : 3.5.0 Perform 3D union using postgis\_sfcgal.
  - **CG\_AlphaShape** - Availability: 3.5.0 - requires SFCGAL >= 1.4.1. 计算包络几何体的 Alpha 形状
  - **CG\_ApproxConvexPartition** - 可用性 : 3.5.0 - 需要 SFCGAL >= 1.5.0. 计算多边形几何体的近似凸分割
  - **CG\_ApproximateMedialAxis** - 可用性 : 3.5.0 计算几何区域的近似中轴。
  - **CG\_Area** - 可用性 : 3.5.0 Calculates the area of a geometry
  - **CG\_Difference** - 可用性 : 3.5.0 Computes the geometric difference between two geometries
  - **CG\_Distance** - 可用性 : 3.5.0 Computes the minimum distance between two geometries
  - **CG\_Extrude** - 可用性 : 3.5.0 将曲面挤出到相关体
  - **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 限制 LHR 方向
  - **CG\_GreeneApproxConvexPartition** - 可用性 : 3.5.0 - 需要 SFCGAL >= 1.5.0. 计算多边形几何体的近似凸分割
  - **CG\_Intersection** - 可用性 : 3.5.0 Computes the intersection of two geometries
  - **CG\_Intersects** - 可用性 : 3.5.0 判断两个几何体是否相交（它们至少有一个共同点）
  - **CG\_IsPlanar** - 可用性 : 3.5.0 判断表面是否平坦
  - **CG\_IsSolid** - 可用性 : 3.5.0 判断几何体是否是体。不行有效性。
  - **CG\_MakeSolid** - 可用性 : 3.5.0 将几何体造成体。不行任何。要得有效的体，输入几何体必须是合多面体曲面或合 TIN。
  - **CG\_MinkowskiSum** - 可用性 : 3.5.0 行 Minkowski sum
  - **CG\_OptimalAlphaShape** - Availability: 3.5.0 - requires SFCGAL >= 1.4.1. 使用“最佳”alpha 计算包络几何体的 Alpha 形状。
  - **CG\_OptimalConvexPartition** - 可用性 : 3.5.0 - 需要 SFCGAL >= 1.5.0. 计算多边形几何体的最优凸分割
  - **CG\_Orientation** - 可用性 : 3.5.0 确定表面方向
  - **CG\_StraightSkeleton** - 可用性 : 3.5.0 从几何体计算直骨架
  - **CG\_Tessellate** - 可用性 : 3.5.0 将多边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 Triangulates a polygonal geometry
  - **CG\_Union** - 可用性 : 3.5.0 Computes the union of two geometries
  - **CG\_Visibility** - 可用性 : 3.5.0 - 需要 SFCGAL >= 1.5.0. 计算一个从点或多边形几何中的线段生成的可性多边形
  - **CG\_Volume** - 可用性 : 3.5.0 计算 3D 体的体。如果用于表面（甚至合）几何体将返回 0。
  - **CG\_YMonotonePartition** - 可用性 : 3.5.0 - 需要 SFCGAL >= 1.5.0. 计算多边形几何的 y 分割
  - **ST\_HasM** - 可用性 : 3.5.0 判断几何体是否具有 M (量) 度。
  - **ST\_HasZ** - 可用性 : 3.5.0 判断几何体是否具有 Z 度。
  - **ST\_RemoveIrrelevantPointsForView** - 可用性 : 3.5.0 Removes points that are irrelevant for rendering a specific rectangular view of a geometry.
-

- **ST\_RemoveSmallParts** - 可用性 : 3.5.0 Removes small parts (polygon rings or linestrings) of a geometry.
- **TopoGeo\_LoadGeometry** - 可用性 : 3.5.0 Load a geometry into an existing topology, snapping and splitting as needed.

### PostGIS 的功能增⊗ 3.5

- **ST\_Clip** - 增⊗版 : 3.5.0 - 添加了 touched 参数。Returns the raster clipped by the input geometry. If band number is not specified, all bands are processed. If crop is not specified or TRUE, the output raster is cropped. If touched is set to TRUE, then touched pixels are included, otherwise only if the center of the pixel is in the geometry it is included.

### PostGIS 中的功能⊗生⊗化 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** - Changed: 3.5.0 : the logic behind the function now uses a test of containment within a buffer, rather than the ST\_MaxDistance algorithm. Results will differ from prior versions, but should be closer to user expectations. Tests if a geometry is entirely inside a distance of another

## 13.12.2 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 Returns the version number of the PROJ library against which PostGIS was built.
- **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⊗入几何⊗形的簇 ID，将⊗入几何⊗形聚⊗到⊗接的集合中。
- **ST\_ClusterWithinWin** - 可用性 : 3.4.0 窗口函数，返回每个⊗入几何⊗形的簇 ID，使用分离距离⊗行聚⊗。
- **ST\_CoverageInvalidEdges** - 可用性 : 3.4.0 用于⊗找多⊗形无法形成有效覆盖范⊗的位置的窗口函数。
- **ST\_CoverageSimplify** - 可用性 : 3.4.0 ⊗化多⊗形覆盖范⊗⊗⊗的窗口函数。
- **ST\_CoverageUnion** - 可用性 : 3.4.0-需要 GEOS >= 3.8.0 通⊗⊗除共享⊗来⊗算形成覆盖范⊗的一⊗多⊗形的并集。
- **ST\_InverseTransformPipeline** - 可用性 : 3.4.0 返回一个新的几何体，其坐⊗⊗使用定⊗的坐⊗⊗⊗管道的逆⊗⊗⊗⊗到不同的空⊗参考系。
- **ST\_LargestEmptyCircle** - 可用性 : 3.4.0。⊗算不与几何⊗形重⊗的最大⊗。
- **ST\_LineExtend** - 可用性 : 3.4.0 返回一条⊗，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⊗形，其坐⊗⊗使用定⊗的坐⊗⊗⊗管道⊗⊗⊗不同的空⊗参考系⊗。
- **TopoElement** - 可用性 : 3.4.0 将拓扑几何⊗⊗⊗拓扑元素。
- **debug\_standardize\_address** - 可用性 : 3.4.0 返回 json 格式的文本，列出解析⊗⊗和⊗准化

- **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 Proj 数据中每个空参考系的元数据。
- **postgis\_srs\_codes** - 可用性：3.4.0 返回与限定界限相关的 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 支持所有曲面型返回几何体的 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。返回一个小数位置之间的直线部分。
- **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重采样一个栅格像，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，也可以从一个栅格像中借用。
- **ST\_Rescale** - 增强：3.4.0 添加了最大和最小重采样通栅格整栅格的比例（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。返回几个几何形之间的 2D 最短

#### 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。

### 13.12.3 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 删除定义有 TopoGeometry 对象不需要的拓扑基元。
- **ST\_3DConvexHull** - 可用性：3.3.0 计算几何体的 3D 凸包。
- **ST\_3DUnion** - 可用性：3.3.0 添加了聚合函数格式进行 3D 聚合。
- **ST\_AsMARC21** - 可用性：3.3.0 将几何形返回具有地理数据字段 (034) 的 MARC21/XML 。
- **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 返回渲染几何形的输入字母，默认起始位置位于原点，默认文本高度为 100。
- **ST\_OptimalAlphaShape** - 可用性：3.3.0 - 需要 SFCGAL >= 1.4.1。使用“最佳”alpha 计算包围几何体的 Alpha 形状。
- **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，用 GEOS 3.11+ 用 GEOS 本机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。返回通将 MultiLineString 合在一起形成的。

### PostGIS 中的功能生色化 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。将 PostGIS 展（例如 postgis\_raster、postgis\_topology、postgis\_sfcgal）打包并升级到固定版本或最新版本。

## 13.12.4 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 。
- **FindTopology** - 可用性：3.2.0 通过不同的方式返回拓扑 。
- **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 几何形中的各个段返回一行 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 建一个表。
- **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 作表面高程。
- **ST\_SRID** - 可用性：3.2.0 返回拓扑几何的空参考 。
- **ST\_Scroll** - 可用性：3.2.0 更改 串的起点。
- **ST\_SetM** - 可用性：3.2.0 返回与输入几何形具有相同 X/Y 坐 的几何形，并使用 求的重采 算法将 格中的 复制到 M 度。
- **ST\_SetZ** - 可用性：3.2.0 返回与输入几何形具有相同 X/Y 坐 的几何形，并使用 求的重采 算法将 格中的 复制到 Z 度。
- **TopoGeom\_addTopoGeom** - 可用性：3.2 将一个 TopoGeometry 的元素添加到一个 TopoGeometry 的定 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 置 理外部数据 格 使用的 的字符串配置。

### PostGIS 的功能增强 3.2

- **GetFaceByPoint** - 增修：3.2.0 更高效的施法和更清晰的契合，停止使用无效的拓扑。查找与指定点相交的面。
- **ST\_ClusterKMeans** - 增修：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** - 增修：在 3.2.0 中，添加了可行的算法参数 “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** - 增修：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** - 增修：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 SRID 栅格坐标。点几何是像素表示的区域的中心。
- **ST\_Point** - 增修：3.2.0 srid 作可选外的可选参数被添加。旧的安装需要与 ST\_SetSRID 配合以在几何体上设置 srid。创建具有 X、Y 和 SRID 的点。
- **ST\_PointM** - 增修：3.2.0 srid 作可选外的可选参数被添加。旧的安装需要与 ST\_SetSRID 配合以在几何体上设置 srid。创建具有 X、Y、M 和 SRID 的点。
- **ST\_PointZ** - 增修：3.2.0 srid 作可选外的可选参数被添加。旧的安装需要与 ST\_SetSRID 配合以在几何体上设置 srid。创建具有 X、Y、Z 和 SRID 的点。
- **ST\_PointZM** - 增修：3.2.0 srid 作可选外的可选参数被添加。旧的安装需要与 ST\_SetSRID 配合以在几何体上设置 srid。创建具有 X、Y、Z、M 和 SRID 的点。
- **ST\_RemovePoint** - 增修：3.2.0 从串中删除一个点。
- **ST\_RemoveRepeatedPoints** - 增修：3.2.0 返回除了重复点的几何图形。
- **ST\_StartPoint** - 增修：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。返回 LineString 的第一个点。
- **ST\_Value** - 增修：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 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** - Enhanced: 3.2.0 added support for returning signed identifier. Adds a linestring to an existing topology using a tolerance and possibly splitting existing edges/faces.

## PostGIS 中的功能生成化 3.2

- **ST\_Boundary** - 更改：3.2.0 支持 TIN，不使用地理，非线性化曲线返回几何图形的边界。
- **ValidateTopology** - 更改：3.2.0 添加了可行的 bbox 参数，行面元和面元接合。返回一个 validate\_topology\_returntype 象，指明拓扑。

### 13.12.5 PostGIS 新增功能或增修功能 (3.1)

下面列出的功能是添加或增修的 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内的元坐标返回一个六边形。
- **ST\_HexagonGrid** - 可用性：3.1.0 返回一个完全覆盖几何参数边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内的元坐标返回一个正方形。



- **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 的坐标。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。计算几何体的直分。
- **ST\_SymDifference** - 增强：3.1.0 接受 gridSize 参数。计算表示几何形 A 和 B 不相交部分的几何形。
- **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。计算多个几何体的件的并集。
- **ST\_Union** - 增强：3.1.0 接受 gridSize 参数。计算表示输入几何形的点集并集的几何形。

### PostGIS 中的功能生成化 3.1

- **ST\_Count** - 更改：3.1.0 - 除了 ST\_Count(rastertable, rastercolumn, ...) 体。改用。返回格或格覆盖范围的定波段中的像素数。如果未指定 band, 默认 band 1。如果 except\_nodata\_value 置 true, 计算不等于 nodata 的像素。
- **ST\_Force3D** - 更改：3.1.0. 您可以在指定一个非零 Z。强制几何形入 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。强制几何形入 XYM 模式。
- **ST\_Force3DZ** - 更改：3.1.0. 您可以在指定一个非零 Z。强制几何形入 XYZ 模式。
- **ST\_Force4D** - 更改：3.1.0. 您可以在指定非零 Z 和 M。强制几何形入 XYZM 模式。
- **ST\_Histogram** - 更改：3.1.0 除了 ST\_Histogram(table\_name, column\_name) 体。返回一个，除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，将自计算分箱数。
- **ST\_Quantile** - 更改：3.1.0 除了 ST\_Quantile(table\_name, column\_name) 体。计算本或体上下文中格或格覆盖范围的分位数。因此，可以某个是否位于格的 25%、50%、75% 百分位。
- **ST\_SummaryStats** - 更改：3.1.0 ST\_SummaryStats(rastertable, rastercolumn, ...) 体已除。改用。返回一个信息，包括定格或格覆盖的格的数、和、均、准差、最小和最大。如果未指定号, 假定 1。

## 13.12.6 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 返回一个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 的小数指示位置插的点。
- **ST\_ConstrainedDelaunayTriangles** - 可用性：3.0.0 返回一个定输入几何体的约束 Delaunay 三角剖分。

- **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 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 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 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 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 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 除了每弧的最小凸性分数。防止拓扑崩溃。将包含曲线的几何图形转换为凸性几何图形。
- **ST\_Disjoint** - 增强：3.0.0 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 用了 GEOMETRYCOLLECTION 的支持两个几何图形是否包含同一对象
- **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 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 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，或计算它的交集矩形
- **ST\_Segmentize** - 增强：3.0.0 分段几何在可生成等距的子分段返回修改后的几何图形/地理，其段不于固定距离。
- **ST\_Touches** - 增强：3.0.0 用了 GEOMETRYCOLLECTION 的支持两个几何图形是否至少有一个共同点，但它的内部不相交
- **ST\_Within** - 增强：3.0.0 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 中的功能生色化 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 - 不再依赖于 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”名称将几何体作 KML 元素返回。
- **ST\_Distance** - 更改：3.0.0 - 不再依赖于 SFCGAL。返回两个几何或地理之间的距离。
- **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。两个几何体是否相交 (它们至少有一个共同点)
- **ST\_Union** - 更改：3.0.0 不依赖于 SFCGAL。计算表示输入几何体的点集并集的几何体。

### 13.12.7 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 返回栅格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 算法返回几何体的平滑版本
- **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 返回沿直以分数间隔插点。
- **ST\_OrientedEnvelope** - 可用性：2.5.0。返回包含几何体的最小面积矩形。
- **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) 栅格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) 栅格返回栅格。
- **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 返回栅格的熟知的二进制 (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 - 引入了可变的精度参数。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** - 增修：2.5.0 包括 outdbbandnum、文件大小和文件戳 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** - 增修：2.5.0 - ST\_Buffer 的几何感知版本已得到增修，允您指定要冲的一侧。side=both|left|right。计算覆盖距几何体定距离内所有点的几何体。
- **ST\_GeomFromGeoJSON** - 增修：2.5.0 可在可以接受 json 和 jsonb 作输入。将几何形的 geojson 表示形式作输入并输出 PostGIS 几何对象
- **ST\_GeometricMedian** - 增修：2.5.0 添加了 M 作点重的支持。返回多点的几何中位数。
- **ST\_Intersects** - 增修：2.5.0 支持 GEOMETRYCOLLECTION。两个几何形是否相交（它至少有一个共同点）
- **ST\_OffsetCurve** - 增修：2.5 - 添加了 GEOMETRYCOLLECTION 和 MULTILINESTRING 的支持返回距离和方向的偏移。
- **ST\_Scale** - 增修：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** - 增修：2.5.0 引入了通多分割多形的支持。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\_Subdivide** - 增修：2.5.0 重用多形分割上的点，点数从 8 少到 5。计算几何体的直分。

PostGIS 中的功能生 2.5

- **ST\_GDALDrivers** - 更改：2.5.0 - 添加 can\_read 和 can\_write 列。通 GDAL 返回 PostGIS 支持的格式列表。ST\_AsGDALRaster 可使用 can\_write=True 的格式

### 13.12.8 PostGIS 新增功能或增修功能 (2.4)

下面列出的功能是添加或增修的 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 将几何形到 MVT 瓦片的坐标空。
- **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 将所有外逆定向，将所有内定向。
- **ST\_ForcePolygonCW** - 可用性：2.4.0 方向调整所有外，逆方向调整所有内。
- **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 多形是否具有逆方向的外和定向的内。
- **ST\_IsPolygonCW** - 可用性：2.4.0 多形是否具有定向外和逆内。
- **ST\_MakeEmptyCoverage** - 可用性：2.4.0 用空网格覆盖地理参考区域。

PostGIS 的功能增修 2.4

- **Loader\_Generate\_Nation\_Script** - 增修：2.4.1 政 5 制表区域 (zcta5) 加步已修复，用后，zcta5 数据将作名 zcta5\_all 的个表加，作国家脚本加的一部分。指定平台生成加到州找表中的 shell 脚本。

- **Normalize\_Address** - 增修：2.4.0 `norm_addy` 函数包括附加字段 `zip4` 和 `address_alphanumeric`。该函数定义文本街道地址，返回复合的 `norm_addy` 类型，该类型具有道路后缀、前缀和规范化类型、街道、街道名称等，分属独特的字段。该函数适用于与 `Tiger_geocoder` 打包的查找数据（不需要 `tiger` 普查数据）。
- **PgNormalize\_Address** - 增修：2.4.0 `norm_addy` 函数包括附加字段 `zip4` 和 `address_alphanumeric`。该函数定义文本街道地址，返回复合的 `norm_addy` 类型，该类型具有道路后缀、前缀和规范化类型、街道、街道名称等，分属独特的字段。该函数适用于与 `Tiger_geocoder` 打包的查找数据（不需要 `tiger` 普查数据）。需要 `address_standardizer` 扩展。
- **Reverse\_Geocode** - 增修：2.4.1 如果加载了可用的 `zcta5` 数据集，即使未加载特定的状态数据，`reverse_geocode` 函数也可以解析状态和 `zip`。有关加载 `zcta5` 数据的信息，请参见。该函数取已知空参考系中的几何点并返回一条或多条记录，其中包含合理可能的地址数和交叉街道数。如果 `include_strnum_range = true`，则包括交叉街道中的街道范围。
- **ST\_AsTWKB** - 增修：2.4.0 内存和速度改进。返回几何形式为 TWKB，又名“微小的已知的二进制”
- **ST\_Covers** - 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多边形的支持。返回 B 的每个点是否都位于 A 中
- **ST\_CurveToLine** - 增修：2.4.0 支持最大距离公差和最大角度公差，支持曲线拟合。将包含曲线的几何形状转换为线性几何形状。
- **ST\_Project** - 增修：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** - 增修：2.4.0 引入了曲线支持。返回点顺序相反的几何体。

#### PostGIS 中的功能生成化 2.4

- **=** - 更改：2.4.0，在之前的版本中，该函数是边界框相等而不是几何相等。如果需要边界框相等，请使用 `ST_Equals`。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，则返回 TRUE。
- **ST\_Node** - 更改：2.4.0 该函数在内部使用 `GEOSNode` 而不是 `GEOSUnaryUnion`。与 `PostGIS < 2.4` 相比，该函数可能会生成具有不同的顺序和方向的点集。

### 13.12.9 PostGIS 新增功能或增修功能 (2.3)

下面列出的功能是添加或增修的 PostGIS 功能。

#### PostGIS 中的新功能 2.3

- **ST\_Intersection(geometry,gidx)** - 可用性：2.3.0 引入了范围索引 (BRIN) 的支持。需要 PostgreSQL 9.5+。如果几何体的 (n 个) `n` 边界框与 `n` 浮点精度边界框 (GIDX) 相交，则返回 TRUE。
- **ST\_Intersection(gidx,geometry)** - 可用性：2.3.0 引入了范围索引 (BRIN) 的支持。需要 PostgreSQL 9.5+。如果 `n` 浮点精度边界框 (GIDX) 与几何体的 (n 个) 边界框相交，则返回 TRUE。
- **ST\_Intersection(gidx,gidx)** - 可用性：2.3.0 引入了范围索引 (BRIN) 的支持。需要 PostgreSQL 9.5+。如果 `n` 个 `n` 浮点精度边界框 (GIDX) 彼此相交，则返回 TRUE。
- **ST\_Intersection(box2df,box2df)** - 可用性：2.3.0 引入了范围索引 (BRIN) 的支持。需要 PostgreSQL 9.5+。如果 `n` 个 2D 浮点精度边界框 (BOX2DF) 彼此相交，则返回 TRUE。
- **ST\_Intersection(box2df,geometry)** - 可用性：2.3.0 引入了范围索引 (BRIN) 的支持。需要 PostgreSQL 9.5+。如果 2D 浮点精度边界框 (BOX2DF) 与几何体的 (n 个) 2D 边界框相交，则返回 TRUE。
- **ST\_Intersection(geometry,box2df)** - 可用性：2.3.0 引入了范围索引 (BRIN) 的支持。需要 PostgreSQL 9.5+。如果几何体的 (n 个) 2D 边界框与 2D 浮点精度边界框 (BOX2DF) 相交，则返回 TRUE。
- **ST\_Contains(box2df,box2df)** - 可用性：2.3.0 引入了范围索引 (BRIN) 的支持。需要 PostgreSQL 9.5+。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，则返回 TRUE。

- **@(box2df,geometry)** - 可用性：2.3.0 引入了 `BRIN` 索引的支持。需要 PostgreSQL 9.5+。如果 2D 浮点精度 `BOX2DF` 包含在几何体的 2D `BOX2DF` 中，返回 `TRUE`。
- **@(geometry,box2df)** - 可用性：2.3.0 引入了 `BRIN` 索引的支持。需要 PostgreSQL 9.5+。如果几何体的 2D `BOX2DF` 包含在 2D 浮点精度 `BOX2DF` 中，返回 `TRUE`。
- **Populate\_Topology\_Layer** - 可用性：2.3.0 通过从拓扑表 `Topology` 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体的簇 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体的簇 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 返回包含几何体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，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 `LineString`。
- **ST\_Normalize** - 可用性：2.3.0 返回规范形式的几何体。
- **ST\_Points** - 可用性：2.3.0 返回包含几何体的 `MultiPoint`。
- **ST\_VoronoiLines** - 可用性：2.3.0 返回几何体点的 Voronoi 图。
- **ST\_VoronoiPolygons** - 可用性：2.3.0 返回几何体点的 Voronoi 图。
- **ST\_WrapX** - 可用性：2.3.0 需要 GEOS 将几何体 `LINESTRING` 在 X 轴上包裹。
- **TopoGeom\_addElement** - 可用性：2.3 将元素添加到 `TopoGeometry` 的定义中。
- **TopoGeom\_removeElement** - 可用性：2.3 从 `TopoGeometry` 的定义中删除元素。
- **~(box2df,box2df)** - 可用性：2.3.0 引入了 `BRIN` 索引的支持。需要 PostgreSQL 9.5+。如果 2D 浮点精度 `BOX2DF` 包含一个 2D 浮点精度 `BOX2DF`，返回 `TRUE`。
- **~(box2df,geometry)** - 可用性：2.3.0 引入了 `BRIN` 索引的支持。需要 PostgreSQL 9.5+。如果 2D 浮点精度 `BOX2DF` 包含几何体的 2D `BOX2DF`，返回 `TRUE`。
- **~(geometry,box2df)** - 可用性：2.3.0 引入了 `BRIN` 索引的支持。需要 PostgreSQL 9.5+。如果几何体的 2D `BOX2DF` 包含 2D 浮点精度 `BOX2DF` (`GIDX`)，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包含。检查 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：2.3.0 几何体，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包含。检查 B 的每个点是否都位于 A 中
- **ST\_Expand** - 增强：2.3.0 添加了不同宽度的盒子进行不同数量扩展的支持。返回从一个边界框或几何体形成的边界框。
- **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包含。检查两个几何体是否相交（它们至少有一个共同点）

- **ST\_Segmentize** - 增修：2.3.0 地理分段可在可生成等距的子分段返回修改后的几何图形/地理，其分段不基于固定距离。
- **ST\_Transform** - 增修：2.3.0 引入了直接 PROJ.4 字符串的支持。返回坐标不同空间参考系的新几何图形。
- **ST\_Within** - 增修：基于 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。以前的版本支持面和点包含。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

PostGIS 中的功能生成化 2.3

- **ST\_PointN** - 更改：2.3.0：索引可用（-1 是最后一点）返回几何图形中第一个点或图形串中的第 N 个点。

### 13.12.10 PostGIS 新增功能或增修功能（2.2）

下面列出的功能是添加或增修的 PostGIS 功能。

PostGIS 中的新功能 2.2

- **<<->** - 可用性：2.2.0——KNN 适用于 PostgreSQL 9.1+ 返回 A 和 B 几何图形或边界框之 n 距离
- **ST\_3DDifference** - 可用性：2.2.0 执行 3D 差异
- **ST\_3DUnion** - 可用性：2.2.0 执行 3D 包含。
- **ST\_ApproximateMedialAxis** - 可用性：2.2.0 计算几何区域的近似中轴。
- **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之量。
- **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进行聚合的聚合函数。
- **ST\_CountAgg** - 可用性：2.2.0 检查的。返回一检查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 置 true，检查不等于 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\_IsPlanar** - 可用性：2.2.0：在 2.1.0 中已检查，但在 2.1 版本中意外遗漏。检查表面是否平坦
- **ST\_IsSolid** - 可用性：2.2.0 检查几何体是否固体。不行有效性检查。
- **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\_MakeSolid** - 可用性：2.2.0 将几何体造成固体。不行任何检查。要得有效的固体，输入几何图形必须是合多面体曲面或合 TIN。



- **ST\_MapAlgebra (callback function version)** - 可用性：2.2.0：能添加遮罩回函数版本 - 定一个或多个入格、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 返回格占用的空量（以字位）。
- **ST\_RemoveRepeatedPoints** - 可用性：2.2.0 返回除了重复点的几何形。
- **ST\_Retile** - 可用性：2.2.0 从任意平的格覆盖范返回一配置的平。
- **ST\_SetEffectiveArea** - 可用性：2.2.0 使用 Visvalingam-Whyatt 算法置每个点的有效区域。
- **ST\_SimplifyVW** - 可用性：2.2.0 使用 Visvalingam-Whyatt 算法返回几何形的化表示
- **ST\_Subdivide** - 可用性：2.2.0 算几何体的直分。
- **ST\_SummaryStatsAgg** - 可用性：2.2.0 的。返回一格格的定格波段的摘要信息，其中包含数、和、平均、准差、最小、最大。如果未指定号，假定 1。
- **ST\_SwapOrdinates** - 可用性：2.2.0 返回更改后的几何形，其中交了定的坐。
- **ST\_Volume** - 可用性：2.2.0 算 3D 体的体。如果用于表面（甚至合）几何形将返回 0。
- **parse\_address** - 可用性：2.2.0 取 1 行地址并分成几部分
- **postgis.enable\_outdb\_rasters** - 可用性：2.2.0 一个布配置，用于用数据外格波段的。
- **postgis.gdal\_datapath** - 可用性：2.2.0 用于分配 GDAL 的 GDAL\_DATA 的配置。如果未置，使用境置的 GDAL\_DATA 量。
- **postgis.gdal\_enabled\_drivers** - 可用性：2.2.0 用于置 PostGIS 境中用的 GDAL 程序的配置。影 GDAL 配置量 GDAL\_SKIP。
- **standardize\_address** - 可用性：2.2.0 利用 lex、gaz 和表返回入地址的 stdaddr 形式。
- **|=|** - 可用性：2.2.0。索引支持适用于 PostgreSQL 9.5+ 返回 A 和 B 迹在最接近点之间的距离。

## PostGIS 的功能增 2.2

- **<->** - 增：2.2.0 -几何和地理之的 KNN (k 最近) 行是在真。注意，地理的 KNN 是在球面上算的，而不是在球体平面上算的。PostgreSQL 9.4 及更低版本支持地理，但不支持界框的重心。返回 A 和 B 之的 2D 距离。
- **AsTopoJSON** - 增：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：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。返回多形几何体的面。
- **ST\_AsX3D** - 增：2.2.0：添加了反地理坐和 (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：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。返回点之直之基于北方的方位角。
- **ST\_Distance** - 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。返回个几何或地理之的距离。
- **ST\_Scale** - 增：2.2.0 引入了放所有度 (factor 参数) 的支持。按定因子放几何形。
- **ST\_Split** - 增：2.2.0 引入了通多、多点或 (多) 多形界分割的支持。返回通将一个几何体分割一个几何体而建的几何体集合。
- **ST\_Summary** - 增：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

- **<->** - 更改：2.2.0 - 在 PostgreSQL 9.5 中，旧的混合格式可能很慢。因此，如果您只想在 PostGIS 2.2 或更高版本和 PostgreSQL 9.5 或更高版本上运行，您可能需要消除方法。返回 A 和 B 之间的 2D 距离。
- **Get\_Geocode\_Setting** - 更改：2.2.0：默认配置在保存在名为 `geocode_settings_default` 的表中。使用自定义配置 a 位于 `geocode_settings` 中，并且包含用配置的配置。返回存储在 `Tiger.geocode_settings` 表中的特定配置的配置。
- **ST\_3DClosestPoint** - 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。返回 g1 上最接近 g2 的 3D 点。它是 3D 最短线的第一个点。
- **ST\_3DDistance** - 更改：2.2.0 - 在 2D 和 3D 的情况下，对于缺失的 Z，Z 不再被假定为 0。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\_3DLongestLine** - 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。返回两个几何体之间的 3D 最直线。
- **ST\_3DMaxDistance** - 更改：2.2.0 - 在 2D 和 3D 的情况下，对于缺失的 Z，Z 不再被假定为 0。返回两个几何图形之间的 3D 笛卡儿最大距离（基于空参考）（以投影位表示）。
- **ST\_3DShortestLine** - 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。返回两个几何图形之间的 3D 最短线。
- **ST\_DistanceSphere** - 更改：2.2.0 在之前的版本中，它曾被称 `ST_Distance_Sphere` 使用球形地球模型返回两个度/度几何形状之间的最小距离（以米为单位）。
- **ST\_DistanceSpheroid** - 更改：2.2.0 在之前的版本中，它称 `ST_Distance_Spheroid` 使用球体模型返回两个度/度几何形状之间的最小距离。
- **ST\_Equals** - 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 两个几何图形是否包含同一点。
- **ST\_LengthSpheroid** - 更改：2.2.0 在之前的版本中，它称 `ST_Length_Spheroid` 并具有名称 `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 名称更改为 `ST_MemSize` 以遵循命名约定。返回几何图形占用的内存空数量。
- **ST\_PointInsideCircle** - 更改：2.2.0 在之前的版本中，它称 `ST_Point_Inside_Circle` 点几何图形是否位于由中心和半径定义的圆内。
- **ValidateTopology** - 更改：2.2.0 id1 和 id2 的被交叉“交叉点”，与描述一致。返回一个 `validatetopology_returntype` 象，说明拓扑。

### 13.12.11 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。
- **Drop\_Nation\_Tables\_Generate\_Script** - 可用性：2.1.0 生成一个脚本，删除指定架构中以 `County_all`、`state_all` 或 `state` 代开的，后跟 `county` 或 `state` 的所有表。
- **Get\_Geocode\_Setting** - 可用性：2.1.0 返回存储在 `Tiger.geocode_settings` 表中的特定配置的配置。
- **Loader\_Generate\_Nation\_Script** - 可用性：2.1.0 指定平台生成添加到州找表中的 shell 脚本。
- **Pgcn Normalize\_Address** - 可用性：2.1.0 定义文本街道地址，返回复合的 `norm_addy` 型，型具有道路后、前和规范化型、街道、街道名称等，分独特的字段。函数适用于与 `Tiger_geocoder` 打包的找数据（不需要 `tiger` 普数据）。需要 `address_standardizer` 展。

- **ST\_3DArea** - 可用性：2.1.0 计算 3D 表面几何形状的面积。空于固体将返回 0。
- **ST\_3DIntersection** - 可用性：2.1.0 执行 3D 相交
- **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\_Extrude** - 可用性：2.1.0 将曲面挤出到相关体
- **ST\_ForceLHR** - 可用性：2.1.0 强制 LHR 方向
- **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 - 给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 - 在给定一个输入栅格、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（以像素数位置）的栅格处理函数。
- **ST\_MinkowskiSum** - 可用性：2.1.0 执行 Minkowski sum
- **ST\_NearestValue** - 可用性：2.1.0 返回由 columnx 和 rowy 指定的栅格像素的最接近的非 NODATA 或与其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 或与其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，如果未返回，说明原因。
- **ST\_Orientation** - 可用性：2.1.0 确定表面方向
- **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圆心（点几何）以及每个像素的 X 和 Y 栅格坐标。点几何是像素表示的区域的圆心。
- **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 和 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\_StraightSkeleton** - 可用性：2.1.0 从几何体计算直骨架
- **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 返回具有计算的地形稳定性指数的栅格。
- **ST\_Tessellate** - 可用性：2.1.0 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
- **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。
- **Set\_Geocode\_Setting** - 可用性：2.1.0 设置影响地理编码器功能行的设置。
- **UpdateRasterSRID** - 可用性：2.1.0 更改用指定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.backend** - 可用性：2.1.0 将 GEOS 和 SFCGAL 重定向的功能提供服务的后端。选项：geos 或 sfcgal。默认为 geos。
- **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 添加了对 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** - 增修：将 GML 3 引入了 2.1.0 id 支持。将几何体作 GML 版本 2 或 3 元素返回。
- **ST\_Aspect** - 增修：2.1.0 使用 ST\_MapAlgebra() 并添加可修的 interpolate\_nodata 函数参数返回高程格波段的坡向（默修以度修修位）。修于分析地形很有用。
- **ST\_Boundary** - 增修：引入了 2.1.0 三角函数支持返回几何体的修界。
- **ST\_Clip** - 增修：2.1.0 用 C 重写 Returns the raster clipped by the input geometry. If band number is not specified, all bands are processed. If crop is not specified or TRUE, the output raster is cropped. If touched is set to TRUE, then touched pixels are included, otherwise only if the center of the pixel is in the geometry it is included.
- **ST\_DWithin** - 增修：2.1.0 提高了地理速度。有关修修信息，修参修使地理更快。修修个几何体是否在修定距离内
- **ST\_DWithin** - 增修：2.1.0 引入了修弯曲几何形状的支持。修修个几何体是否在修定距离内
- **ST\_Distance** - 增修：2.1.0 提高了地理速度。有关修修信息，修参修使地理更快。返回修个几何或地理修之修的距离。
- **ST\_Distance** - 增修：2.1.0 - 引入了修弯曲几何形状的支持。返回修个几何或地理修之修的距离。
- **ST\_Distinct4ma** - 增修：2.1.0 添加格式 2 修格修理函数，用于修算修域中唯一像素修的数量。
- **ST\_DumpPoints** - 增修：2.1.0 速度更快。重新修修修原生 C 修言。返回几何体中坐修的一修 geometry\_dump 行。
- **ST\_HillShade** - 增修：2.1.0 使用 ST\_MapAlgebra() 并添加可修的 interpolate\_nodata 函数参数使用提供的方位角、高度、亮度和比例修入返回高程格修的假修照明。
- **ST\_MakeValid** - 增修：2.1.0 添加了修几何集合和多点的支持。修在不修失修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 修算修域中最大像素修的修格修理函数。
- **ST\_Mean4ma** - 增修：2.1.0 添加格式 2 修算修域中平均像素修的修格修理函数。
- **ST\_Min4ma** - 增修：2.1.0 添加格式 2 修算修域中最小像素修的修格修理函数。
- **ST\_PixelAsPolygons** - 增修：2.1.0 添加了 except\_nodata\_value 可修参数。返回包修修格的每个像素的多修形几何体以及每个像素的修、X 和 Y 修格坐修。
- **ST\_Polygon** - 增修：2.1.0 提高速度（完全基于 C）并且确保返回的多修形有效。返回由具有非无数据修的像素修的像素并集形成的多多修形几何体。如果未指定波段修号，修波段修号默修修 1。
- **ST\_Range4ma** - 增修：2.1.0 添加格式 2 修算修域中像素修范修的修格修理函数。
- **ST\_SameAlignment** - 增修：2.1.0 添加聚合修体如果修格具有相同的修斜、比例、空修参考和偏移（像素可以放在同一网格上而不切割成像素），修返回 true；如果没有注意修修修，修返回 false。
- **ST\_Segmentize** - 增修：2.1.0 引入了修地理的支持。返回修改后的几何体/地理，其修段不修于修定距离。
- **ST\_SetGeoReference** - 增修：2.1.0 添加 ST\_SetGeoReference(raster, double precision, ...) 修体在一次修用中修置 Georeference 6 地理配准参数。数字修修用空格分隔。接受 GDAL 或 ESRI 格式的修入。默修修 GDAL。
- **ST\_SetValue** - 增修：2.1.0 ST\_SetValue() 的几何体修在支持任何几何体型，而不修修是点。几何体是 ST\_SetValues() 的 geomval[] 修体的包装返回修改后的修格，其修果是将修定波段中的修置修指定列 x、行 y 像素或与特定几何体相交的像素。波段修号从 1 开始，如果未指定波段，修默修修 1。
- **ST\_Slope** - 增修：2.1.0 使用 ST\_MapAlgebra() 并添加可修 units、scale、interpolate\_nodata 函数参数返回高程格修的坡度（默修以度修修位）。修于分析地形很有用。
- **ST\_StdDev4ma** - 增修：2.1.0 添加格式 2 修算修域中像素修的修准偏差的修格修理函数。
- **ST\_Sum4ma** - 增修：2.1.0 添加格式 2 修格修理函数，修算修域中所有像素修的修和。

- **ST\_Summary** - 增修：添加了 S 标志以指示其是否具有 2.1.0 空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参考系中的栅格重新投影到一个已知空参考系。函数有 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）。将一个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)（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。将一个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。将一个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 中的功能生疏化 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（默认为度为单位）。用于分析地形很有用。
- **ST\_EstimatedExtent** - 更改：2.1.0。在 2.0.x 之前，函数称为 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，函数称为 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，函数称为 ST\_Line\_Locate\_Point。返回最接近点的分数位置。
- **ST\_LineSubstring** - 更改：2.1.0。在 2.0.x 之前，函数被称为 ST\_Line\_Substring。返回从小数位置之直线的部分。
- **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 X、Y 和 Z 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 X、Y 和 Z 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 X、Y 和 Z 栅格坐标。
- **ST\_Polygon** - 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，则波段号默认为 1。
- **ST\_RasterToWorldCoordX** - 更改：2.1.0 在之前的版本中，函数称为 ST\_Raster2WorldCoordX 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** - 更改：2.1.0 在之前的版本中，函数 `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（英式或美式拼写）、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 的栅格通过调整栅格的倾斜（或旋转参数）来重新采样栅格。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\_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)` 会导致不明确的函数调用。输入需要正确输入几何或地理。使用 `ST_GeomFromText`、`ST_GeogFromText` 或所需类型（例如 `ST_Segmentize('LINESTRING(1 2, 3 4)::geometry, 0.5)`）返回修改后的几何/地理，其段不基于固定距离。
- **ST\_Slope** - 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的坡度（默认为度为单位）。对于分析地形很有用。
- **ST\_SnapToGrid** - 更改：2.1.0 适用于没有 SRID 的栅格通过将栅格捕捉到网格来重新采样栅格。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\_WorldToRasterCoordX** - 更改：2.1.0 在之前的版本中，函数 `ST_World2RasterCoordX` 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** - 更改：2.1.0 在之前的版本中，函数 `ST_World2RasterCoordY` 返回点几何 (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)。

### 13.12.12 PostGIS 新增功能或增强功能（2.0）

下面列出的功能是添加或增强的 PostGIS 功能。

PostGIS 中的新功能 2.0

- **&&** - 可用性：2.0.0 如果 A 的边界框与 B 的边界框相交，返回 TRUE。
- **&&&** - 可用性：2.0.0 如果 A 的 n 边界框与 B 的 n 边界框相交，返回 TRUE。
- **<#>** - 可用性：2.0.0——KNN 适用于 PostgreSQL 9.1+ 返回 A 和 B 边界框之间的 2D 距离。
- **<->** - 可用性：2.0.0——弱 KNN 根据几何中心距离而不是真实距离提供最近点。点的结果精确，所有其他类型的结果不精确。适用于 PostgreSQL 9.1+ 返回 A 和 B 之间的 2D 距离。
- **@** - 可用性：2.0.0 `raster@raster`、`raster@geometry` 引入如果 A 的边界框包含在 B 的边界框中，返回 TRUE。使用双精度边界框。
- **@** - 可用性：2.0.5 `geometry @ raster` 引入如果 A 的边界框包含在 B 的边界框中，返回 TRUE。使用双精度边界框。
- **AddEdge** - 可用性：2.0.0 使用指定的边串几何将边串添加到表，并将关系的起点和点添加到指定拓扑方案的点表，并返回新（或已有）的 `edgeid`。
- **AddFace** - 可用性：2.0.0 将面基元注册到拓扑并返回其标识符。
- **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`。如果点已作为点存在，返回已有的 `nodeid`。
- **AddOverviewConstraints** - 可用性：2.0.0 将栅格列添加到一栅格列的概述。
- **AddRasterConstraints** - 可用性：2.0.0 将栅格约束添加到已添加的栅格表中，用于特定列，列约束了空参考、比例、大小、分辨率、波段、波段类型以及一个标志，用于表示栅格列是否被本地分区。表必须添加数据才能推断出约束。如果约束设置成功，返回 true，否则会发出通知。



- **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、体和拓扑几何) 的副本复制到新模式中
- **DropOverviewConstraints** - 可用性: 2.0.0 取消栅格列作栅格一栅格概表的栅格。
- **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, 它非常有用。
- **Drop\_Indexes\_Generate\_Script** - 可用性: 2.0.0 生成一个脚本, 删除 Tiger 架和用指定架上的所有非主和非唯一索引。如果未指定架, 默认架是 Tiger\_data。
- **Drop\_State\_Tables\_Generate\_Script** - 可用性: 2.0.0 生成一个脚本, 删除指定架中以 state 写前的所有表。如果未指定架, 默认架是 Tiger\_data。
- **Geocode\_Intersection** - 可用性: 2.0.0 接收 2 条相交的道路以及 state、city、zip, 并找出位于交叉路口的第一个交叉街道上的一可能位置, 包括一个 geomout 作 NAD 83 度中的点位置, 一个标准化地址 (addy) 每个位置以及。越低, 匹配的可能性越大。果首先按最低分排序。可以输入最大果, 默认 10。使用 Tiger 数据 (edges、faces、addr)、PostgreSQL 模糊字符串匹配 (soundex、levenshtein)。
- **GetEdgeByPoint** - 可用性: 2.0.0 找与点相交的 ID。
- **GetFaceByPoint** - 可用性: 2.0.0 找与点相交的面。
- **GetNodeByPoint** - 可用性: 2.0.0 找某个点位置的点的 ID。
- **GetNodeEdges** - 可用性: 2.0 返回与点相关的一有序。
- **GetRingEdges** - 可用性: 2.0.0 返回按序排列的符号集合, 些符号是通沿定的一行走遇到的。
- **GetTopoGeomElements** - 可用性: 2.0.0 返回一 topoelement 象, 其中包含定 TopoGeometry (原始元素) 的拓扑 element\_id、element\_type。
- **GetTopologySRID** - 可用性: 2.0.0 返回拓扑表中定拓扑名称的拓扑的 SRID。
- **Get\_Tract** - 可用性: 2.0.0 从几何形所在的区域表中返回人口普查区域或字段。默认返回区域的短名称。
- **Install\_Missing\_Indexes** - 可用性: 2.0.0 找具有在地理器接口和条件中使用的列的所有表, 些表缺少些列上使用的索引, 并将添加它。
- **Loader\_Generate\_Census\_Script** - 可用性: 2.0.0 指定 states 的指定平台生成 shell 脚本, 脚本将下 Tiger 人口普查 state 区、bg 和 tabblocks 数据表、存并加到 Tiger\_data 架中。每个脚本都作独立的返回。
- **Loader\_Generate\_Script** - 可用性: 2.0.0 支持 Tiger 2010 化数据并加人口普查区 (tract)、区 (bg) 和区 (tabblocks) 表。指定平台的指定脚本生成 shell 脚本, 脚本将下 Tiger 数据、存并加到 Tiger\_data 模式中。每个脚本都作独立的返回。最新版本支持 Tiger 2010 化, 加人口普查区、区和区表。
- **Missing\_Indexes\_Generate\_Script** - 可用性: 2.0.0 找具有在地理器接口中使用的列的所有表, 些表缺少些列上的索引, 并将出 SQL DDL 来定些表的索引。
- **Polygonize** - 可用性: 2.0.0 找并注册由拓扑定的所有面。
- **Reverse\_Geocode** - 可用性: 2.0.0 取已知空参考系中的几何点并返回一条, 其中包含理上可能的地址数和交叉街道数。如果 include\_strnum\_range = true, 包括交叉街道中的街道范。
- **ST\_3DClosestPoint** - 可用性: 2.0.0 返回 g1 上最接近 g2 的 3D 点。是 3D 最短的第一个点。
- **ST\_3DDFullyWithin** - 可用性: 2.0.0 个 3D 几何形是否完全在定的 3D 距离内
- **ST\_3DDWithin** - 可用性: 2.0.0 个 3D 几何形是否在定的 3D 距离内
- **ST\_3DDistance** - 可用性: 2.0.0 返回个几何形之的 3D 笛卡最小距离 (基于空参考) (以投影位表示)。

- **ST\_3DIntersects** - 可用性: 2.0.0 返回两个几何体在 3D 空间中是否相交 - 适用于点、串、多边形、多面体曲面 (区域)
- **ST\_3DLongestLine** - 可用性: 2.0.0 返回两个几何体之间的 3D 最长直线
- **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 (基于空间参考) (以投影单位表示)。
- **ST\_3DShortestLine** - 可用性: 2.0.0 返回两个几何体之间的 3D 最短直线
- **ST\_AddEdgeModFace** - 可用性: 2.0 添加新面, 如果面做会分割面, 修改原始面并添加新面。
- **ST\_AddEdgeNewFaces** - 可用性: 2.0 添加新面, 如果面做会分割一个面, 删除原始面并用一个新面替换它。
- **ST\_AsGDALRaster** - 可用性: 2.0.0 - 需要 GDAL >= 1.6.0。以指定的 GDAL 格式返回栅格。格式是支持的格式之一。使用 ST\_GDALDrivers() 获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 返回定点的度、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 空间。
- **ST\_AsRaster** - 可用性: 2.0.0 - 需要 GDAL >= 1.6.0。将 PostGIS 几何体转换为 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)。用于分析地形很有用。
- **ST\_Band** - 可用性: 2.0.0 返回有栅格的一个或多个波段作新栅格。用于从有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 Returns the raster clipped by the input geometry. If band number is not specified, all bands are processed. If crop is not specified or TRUE, the output raster is cropped. If touched is set to TRUE, then touched pixels are included, otherwise only if the center of the pixel is in the geometry it is included.
- **ST\_CollectionHomogenize** - 可用性: 2.0.0 返回几何集合的最短表示。
- **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, 计算不等于 nodata 的像素。
- **ST\_CreateTopoGeo** - 可用性: 2.0 将几何体集合添加到空拓扑并返回成功消息。
- **ST\_Distinct4ma** - 可用性: 2.0.0 栅格函数, 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 坐标的几何体版本。
- **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 或更高版本将几何体的 geojson 表示形式作为输入并输出 PostGIS 几何体

- **ST\_GetFaceEdges** - 可用性：2.0 返回一有序的面，这些面界定了 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格的假照明。
- **ST\_Histogram** - 可用性：2.0.0 返回一数组，描述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，将自身算分箱数。
- **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 如果格格空（宽度 = 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，说明几何形是否有效或无效，说明原因和位置。
- **ST\_IsValidReason** - 有效性：2.0 版本正在接受。返回说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 格波段版本：通过输入格波段和提供的像素型用有效的 PostgreSQL 代数算来建新的波段格。如果未指定波段，假定波段 1。
- **ST\_MapAlgebraExpr** - 可用性：2.0.0 2 格波段版本：通过提供的个输入格波段和像素型用有效的 PostgreSQL 代数算来建新的波段格。如果未指定波段号，假定每个格的波段 1。生成的格将在第一个格定义的网格上（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 - 通过在输入格波段和提供的像素型上用有效的 PostgreSQL 函数来建新的波段格。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 个输入格波段和像素型上用有效的 PostgreSQL 函数来建新的波段格。如果未指定波段，假定波段 1。如果未指定，范型默 INTERSECTION。
- **ST\_MapAlgebraFctNgb** - 可用性：2.0.0 1-波段版本：使用用定义的 PostgreSQL 函数映射代数最近。返回一个格，其是涉及输入格波段的区域的 PLPGSQL 用函数的果。
- **ST\_Max4ma** - 可用性：2.0.0 算域中最大像素的格理函数。
- **ST\_Mean4ma** - 可用性：2.0.0 算域中平均像素的格理函数。
- **ST\_Min4ma** - 可用性：2.0.0 算域中最小像素的格理函数。
- **ST\_ModEdgeHeal** - 可用性：2.0 通过除接条的点、修改第一条并除第二条来修复条。返回已除点的 id。
- **ST\_MoveIsoNode** - 可用性：2.0.0 在拓扑中将一个孤立点从一个点移到一个点。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** - 可用性：2.0 通过除接条的点、除条并用方向与提供的第一条相同的替它来修复条。
- **ST\_Node** - 可用性：2.0.0 点是面的集合。
- **ST\_NumPatches** - 可用性：2.0.0 返回多面体曲面上的面数。于非多面体几何形状将返回 null。
- **ST\_OffsetCurve** - 可用性：2.0 返回距输入定距离和方向的偏移。
- **ST\_PatchN** - 可用性：2.0.0 返回多形曲面 (PolyhedralSurface) 的第 N 个几何体 (面)。
- **ST\_Perimeter** - 可用性 2.0.0：引入了地理的支持返回多形几何或地理的界度。

- **ST\_PixelAsPolygon** - 可用性: 2.0.0 返回限定特定行和列的像素的多边形几何形状。
- **ST\_PixelAsPolygons** - 可用性: 2.0.0 返回包络线的每个像素的多边形几何形状以及每个像素的 X、Y 和 Z 坐标。
- **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，可以判断某个点是否位于栅格的 25%、50%、75% 百分位。
- **ST\_Range4ma** - 可用性: 2.0.0 计算域中像素范围的栅格地理函数。
- **ST\_Reclass** - 可用性: 2.0.0 建立由从原始数据重新分组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，则假定 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。
- **ST\_RelateMatch** - 可用性: 2.0.0 判断 DE-9IM 交集矩阵是否与交集矩阵模式匹配
- **ST\_RemEdgeModFace** - 可用性: 2.0 删除一条边，如果删除将面分开，则删除一个面并修改另一个面以覆盖该面的空面。
- **ST\_RemEdgeNewFace** - 可用性: 2.0 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\_Resample** - 可用性: 2.0.0 需要 GDAL 1.6.1+ 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，以及一栅格地理参考属性，某些属性可以自己定义，也可以从一个栅格图像中借用。
- **ST\_Rescale** - 可用性: 2.0.0 需要 GDAL 1.6.1+ 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（英式或美式拼写）、Bilinear、Cubic、CubicSpline、Lanczos、Max 或 Min 重新采样算法计算的。默认为 NearestNeighbor。
- **ST\_Reskew** - 可用性: 2.0.0 需要 GDAL 1.6.1+ 通过调整栅格的斜率（或旋转参数）来重新采样栅格。新的像素是使用 NearestNeighbor（英式或美式拼写）、Bilinear、Cubic、CubicSpline 或 Lanczos 重新采样算法计算的。默认为 NearestNeighbor。
- **ST\_SameAlignment** - 可用性: 2.0.0 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；如果没有注意这些，则返回 false。
- **ST\_SetBandIsNoData** - 可用性: 2.0.0 将区域的 isnodata 标志设置为 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（默认为度为单位）。用于分析地形很有用。
- **ST\_Snap** - 可用性: 2.0.0 将输入几何体的边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新采样栅格。新的像素是使用 NearestNeighbor（英式或美式拼写）、Bilinear、Cubic、CubicSpline 或 Lanczos 重新采样算法计算的。默认为 NearestNeighbor。
- **ST\_Split** - 可用性: 2.0.0 需要 GEOS 返回通过将几何体分割成一个几何体而建立的几何体集合。
- **ST\_StdDev4ma** - 可用性: 2.0.0 计算域中像素的标准偏差的栅格地理函数。
- **ST\_Sum4ma** - 可用性: 2.0.0 栅格地理函数，计算域中所有像素的和。
- **ST\_SummaryStats** - 可用性: 2.0.0 返回一栅格信息，包括指定栅格或栅格覆盖的栅格的计数、和、均值、标准差、最小值和最大值。如果未指定编号，则假定 1。
- **ST\_Transform** - 可用性: 2.0.0 需要 GDAL 1.6.1+ 使用指定的重新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。它有 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。



- **ST\_Union** - 可用性: 2.0.0 将一☐☐格切片的并集返回☐由 1 个或多个波段☐成的☐个☐格。
- **ST\_ValueCount** - 可用性: 2.0.0 返回一☐☐☐, 其中包含像素☐☐以及具有☐定☐集的☐格 (或☐格覆盖范☐) ☐定☐中的像素☐数。如果未指定波段, ☐默☐☐波段 1。默☐情况下, 不☐算☐点数据☐像素。☐出像素中的所有其他☐, 并将像素☐☐四舍五入到最接近的整数。
- **TopoElementArray\_Agg** - 可用性: 2.0.0 返回一☐ element\_id、☐型数☐ (topoelements) 的 topoelementarray。
- **TopoGeo\_AddLineString** - 可用性: 2.0.0 Adds a linestring to an existing topology using a tolerance and possibly splitting existing edges/faces.
- **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☐/面将多☐形添加到☐有拓扑。返回面☐☐符。
- **TopologySummary** - 可用性: 2.0.0 ☐取拓扑名称并提供拓扑中☐象☐型的☐☐☐数。
- **Topology\_Load\_Tiger** - 可用性: 2.0.0 将 tiger 数据的定☐区域加☐到 PostGIS 拓扑中, 并将老虎数据☐☐☐拓扑的空☐参考, 并捕捉到拓扑的精度公差。
- **toTopoGeom** - 可用性: 2.0 将☐☐几何☐形☐☐☐拓扑几何☐形。
- **~** - 可用性: 2.0.0 如果 A 的☐界框包含 B 的☐界框, ☐返回 TRUE。使用双精度☐界框。
- **~=** - 可用性: 2.0.0 如果 A 的☐界框与 B 的☐界框相同, ☐返回 TRUE。

## PostGIS 的功能增☐ 2.0

- **&&** - 增☐: 2.0.0 引入了☐多面体曲面的支持。如果 A 的 2D ☐界框与 B 的 2D ☐界框相交, ☐返回 TRUE。
- **AddGeometryColumn** - 增☐: 2.0.0 引入了 use\_typmod 参数。默☐☐建 typmod 几何列而不是基于☐束。将 geometry (几何) 列添加到☐有表。
- **Box2D** - 增☐功能: 引入了 2.0.0 ☐多面体曲面、三角形和三角网的支持。返回表示几何☐形的 2D 范☐的 BOX2D。
- **Box3D** - 增☐功能: 引入了 2.0.0 ☐多面体曲面、三角形和三角网的支持。返回表示几何体 3D 范☐的 BOX3D。
- **CreateTopology** - 增☐: 2.0 添加接受 hasZ 的格式☐建一个新的拓扑架☐并将其注册到 topology.topology 表中。
- **Geocode** - 增☐功能: 2.0.0 支持 Tiger 2010 ☐☐化数据并修改了一些☐☐以提高地理☐☐的速度和准确性, 并将点从中心☐偏移 to 街道地址所在的一☐。新参数 max\_results 可用于指定最佳☐果的数量或☐返回最佳☐果。将地址作☐字符串 (或其他☐准化地址) ☐入, 并☐出一☐可能的位置, 其中包括 NAD 83 ☐☐度中的点几何☐形、每个位置的☐准化地址以及☐☐。☐☐越低, 匹配的可能性越大。☐果首先按最低☐分排序。可以☐☐☐入最大☐果数, 默☐☐ 10, 以及 restrict\_region (默☐☐ NULL)
- **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** - 增修：2.0.0 支持地理中的字节序。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** - 增修功能：引入了 2.0.0 多面体曲面、三角形和三角网的支持。返回带有 SRID 元数据的几何图形的展已知二进制 (EWKB) 表示形式。
  - **ST\_AsEWKT** - 增修：2.0.0 引入了地理、多面体曲面、三角形和 TIN 的支持。使用 SRID 元数据返回几何图形的已知文本 (WKT) 表示形式。
  - **ST\_AsGML** - 增修：2.0.0 引入了前修支持。引入了 GML3 的修 4，以允许使用 LineString 而不是修条的 Curve 修。引入了多面体曲面和 TIN 的 GML3 支持。引入修 32 来修出修框。将几何修形作修 GML 版本 2 或 3 元素返回。
  - **ST\_AsKML** - 增修：2.0.0 - 添加前修命名空修，使用默修和命名参数将几何修形作修 KML 元素返回。
  - **ST\_Azimuth** - 增修：2.0.0 引入了地理的支持。返回修点之修直修的基于北方的方位角。
  - **ST\_Dimension** - 增修：2.0.0 引入了多面体曲面支持和 TIN 支持。当修定空几何修，它不再引修异常。返回几何修的拓扑修数。
  - **ST\_Dump** - 增修功能：引入了 2.0.0 多面体曲面、三角形和三角网的支持。返回几何修件的一修 geometry\_dump 行。
  - **ST\_DumpPoints** - 增修功能：引入了 2.0.0 多面体曲面、三角形和三角网的支持。返回几何修形中坐修的一修 geometry\_dump 行。
  - **ST\_Expand** - 增修功能：引入了 2.0.0 多面体曲面、三角形和三角网的支持。返回从修一个修界框或几何修形修的修界框。
  - **ST\_Extent** - 增修功能：引入了 2.0.0 多面体曲面、三角形和三角网的支持。返回几何修形修的修界框的聚合函数。
  - **ST\_Force2D** - 增修：2.0.0 引入了多面体曲面的支持。修制几何修形修入“二进制”。
  - **ST\_Force3D** - 增修：2.0.0 引入了多面体曲面的支持。修制几何修形修入 XYZ 模式。修是 ST\_Force3DZ 的修名。
  - **ST\_Force3DZ** - 增修：2.0.0 引入了多面体曲面的支持。修制几何修形修入 XYZ 模式。
  - **ST\_ForceCollection** - 增修：2.0.0 引入了多面体曲面的支持。将几何修形修的修几何集合(GEOMETRYCOLLECTION)。
  - **ST\_ForceRHR** - 增修：2.0.0 引入了多面体曲面的支持。修制多修形修的点的方向遵循右手定修。
  - **ST\_GMLToSQL** - 增修功能：2.0.0 支持多面体曲面和 TIN。从 GML 表示返回指定的 ST\_Geometry 修。修是 ST\_GeomFromGML 的修名
  - **ST\_GMLToSQL** - 增修：2.0.0 引入了多面体曲面支持和 TIN 支持。从 GML 表示返回指定的 ST\_Geometry 修。修是 ST\_GeomFromGML 的修名
  - **ST\_GeomFromEWKB** - 增修功能：2.0.0 支持多面体曲面和 TIN。从修的展已知二进制表示 (EWKB) 返回指定的 ST\_Geometry 修。
  - **ST\_GeomFromEWKT** - 增修功能：2.0.0 支持多面体曲面和 TIN。从修的展已知的文本表示 (EWKT) 返回指定的 ST\_Geometry 修。
  - **ST\_GeomFromGML** - 增修功能：2.0.0 支持多面体曲面和 TIN。将几何修的 GML 表示形式作修修入并修出 PostGIS 几何修象
  - **ST\_GeomFromGML** - 增修：2.0.0 引入了多面体曲面支持和 TIN 支持。将几何修的 GML 表示形式作修修入并修出 PostGIS 几何修象
  - **ST\_GeometryN** - 增修功能：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** - 增修：2.0.0 引入了多面体曲面的支持。以文本形式返回几何修的 SQL-MM 修型。
-

- **ST\_IsClosed** - 增修：2.0.0 引入了对多面体曲面的支持。对 LineStrings 的起点和点是否重合。对于多面体表面是否闭合（中心）。
- **ST\_MakeEnvelope** - 增修：2.0：引入了在不指定 SRID 的情况下指定外包矩形的功能。根据最小和最大坐标构建矩形多边形。
- **ST\_MakeValid** - 增修：2.0.1 速度提升对在不丢失点的情况下使无效几何体有效。
- **ST\_NPoints** - 增修：2.0.0 引入了对多面体曲面的支持。返回几何形中的点数（点）。
- **ST\_NumGeometries** - 增修功能：引入了 2.0.0 对多面体曲面、三角形和三角网的支持。返回几何集中的元素数量。
- **ST\_Relate** - 增修：2.0.0 - 添加了对指定边界点的支持。对几个几何形是否具有与交集模式匹配的拓扑关系，或计算它的交集矩形。
- **ST\_Rotate** - 增修功能：引入了 2.0.0 对多面体曲面、三角形和三角网的支持。对原点旋转几何体。
- **ST\_Rotate** - 增修：2.0.0 添加了用于指定旋转原点的附加参数。对原点旋转几何体。
- **ST\_RotateX** - 增修功能：引入了 2.0.0 对多面体曲面、三角形和三角网的支持。对 X 轴旋转几何体。
- **ST\_RotateY** - 增修功能：引入了 2.0.0 对多面体曲面、三角形和三角网的支持。对 Y 轴旋转几何体。
- **ST\_RotateZ** - 增修功能：引入了 2.0.0 对多面体曲面、三角形和三角网的支持。对 Z 轴旋转几何体。
- **ST\_Scale** - 增修功能：引入了 2.0.0 对多面体曲面、三角形和三角网的支持。按因子缩放几何形。
- **ST\_ShiftLongitude** - 增修功能：2.0.0 支持多面体曲面和 TIN。在 -180-180 和 0-360 之间移动几何形的经度坐标。
- **ST\_Summary** - 增修：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** - 增修：2.0.0 引入了对多面体曲面的支持。返回坐标在不同空参考系的新几何形。
- **ST\_Value** - 增修：2.0.0 添加了 `except_nodata_value` 可选项参数。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` 置为 false，所有像素都被与 `nodata` 像素相交并返回其值。如果未选项参数 `exclude_nodata_value`，从格的元数据中取值。
- **ValidateTopology** - 增修：2.0.0 更有效的交叉检测并修复了先前版本中存在的错误。返回一个 `validate-topology_returntype` 对象，明确拓扑错误。

## PostGIS 中的功能生色化 2.0

- **AddGeometryColumn** - 更改：2.0.0 此函数不再更新 `geometry_columns`，因为 `geometry_columns` 是从系统目录取的。默认情况下，它也不建约束，而是使用 PostgreSQL 内置的列型修符行。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`，但将其置为 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，因为它继承了父表列的 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。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` 是已弃用的列型，因此已更改为 `box3d`。返回格封框的 `box 3d` 表示形式。
- **DropGeometryColumn** - 更改：2.0.0 提供此函数是为了向后兼容。在，由于 `Geometry_columns` 是在系统目录的，因此您可以使用 `ALTER TABLE` 删除几何列，就像删除任何其他表列一样从空表中移除 `geometry`（几何）列。





- **ST\_NumGeometries** - 更改：2.0.0 在之前的版本中，如果几何类型不是 collection/MULTI 类型，会返回 NULL。2.0.0 在用于多个几何类型返回 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，它允许多 MULTIPOLYGON，返回第一个 POLYGON 的内环数量。返回多环形的内环（孔）数。
- **ST\_PointN** - 更改：2.0.0 不再适用于多个几何 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。返回几何类型中第一个环串或环串中的第 N 个点。
- **ST\_ScaleX** - 更改：2.0.0。在 WKTRaster 版本中，名称 ST\_PixelSizeX。返回像素宽度的 X 分量（以坐参考系位）。
- **ST\_ScaleY** - 更改：2.0.0。在 WKTRaster 版本中，名称 ST\_PixelSizeY。返回像素高度的 Y 分量（以坐参考系位）。
- **ST\_SetScale** - 更改：2.0.0 在 WKTRaster 版本中，名称 ST\_SetPixelSize。在 2.0.0 中进行了优化。以坐参考系位位置像素的 X 和 Y 大小。数字位/像素度/高度。
- **ST\_StartPoint** - 更改：2.0.0 不再适用于多个几何体 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入 LINESTRING 的人可能会在 2.0.0 中遇到返回 NULL 的情况。返回 LineString 的第一个点。

### 13.12.13 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 边界与 B 的 2D 边界相交，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量。
- **ST\_AsBinary** - 可用性：1.5.0 支持地理位置。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** - 可用性：1.5.0 支持地理位置。将几何类型作 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 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，您可能希望将道路串成街道面，并希望将端点成平面或正方形而不是点。添加了地理的薄包装器。计算覆盖距几何体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。是从一个几何体到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类型。
- **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 Tests if a geometry is entirely inside a distance of another
- **ST\_DWithin** - 可用性：1.5.0 引入了地理的支持检查多个几何类型是否在定距离内
- **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类型返回多个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两个度/度几何形状之间的最小距离（以米位）。
- **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。使用球体模型返回两个度/度几何形状之间的最小距离。
- **ST\_DumpPoints** - 可用性：1.1.0 返回几何形中坐点的一行 `geometry_dump` 行。
- **ST\_Envelope** - 可用性：1.5.0 行更改出双精度而不是 `float4` 返回表示几何形界框的几何形。
- **ST\_Expand** - 可用性：1.5.0 行更改出双精度而不是 `float4` 坐点。返回从一个界框或几何形展的界框。
- **ST\_GMLToSQL** - 可用性：需要 1.5 libxml2 1.6+ 从 GML 表示返回指定的 `ST_Geometry`。是 `ST_GeomFromGML` 的别名
- **ST\_GeomFromGML** - 可用性：需要 1.5 libxml2 1.6+ 将几何形的 GML 表示形式作输入并输出 PostGIS 几何象
- **ST\_GeomFromKML** - 可用性：1.5, 需要 libxml2 2.6+ 将几何形的 KML 表示形式作输入并输出 PostGIS 几何象
- **ST\_HausdorffDistance** - 可用性：1.1.0 返回两个几何形之间的 Hausdorff 距离。
- **ST\_Intersection** - 可用性：1.5 引入了地理数据类型的支持。运算表示几何 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。两个几何形是否相交（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度。
- **ST\_LongestLine** - 可用性：1.1.0 返回两个几何形之间的二度最线。
- **ST\_MakeEnvelope** - 可用性：1.5 根据最小和最大坐点建矩形多边形。
- **ST\_MaxDistance** - 可用性：1.1.0 返回两个几何形之间的二度最大距离（以投影位表示）。
- **ST\_ShortestLine** - 可用性：1.1.0 返回两个几何形之间的 2D 最短线
- **~=** - 可用性：1.5.0 改行了行如果 A 的界框与 B 的界框相同，返回 TRUE。

### 13.12.14 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 确保几何列由型修符合定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，可以更快地理更多几何形。从一个几何形建 `GeometryCollection` 或 `Multi*` 几何形。
- **ST\_ContainsProperly** - 可用性：1.4.0 两个 B 的每个点是否都位于 A 的内部
- **ST\_GeoHash** - 可用性：1.4.0 返回几何形的 GeoHash 表示形式。
- **ST\_IsValidReason** - 有效性：1.4 返回说明几何形是否有效或无效原因的文本。
- **ST\_LineCrossingDirection** - 有效性：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` 聚合函数得到增强，可以更快地理更多点。从 `Point`, `MultiPoint`, 或 `LineString geometries` 建 `LineString`。
- **ST\_MinimumBoundingCircle** - 可用性：1.4.0 返回包含几何形的最小形多边形。
- **ST\_Union** - 可用性：1.4.0 - `ST_Union` 得到增强。 PostgreSQL 中引入了 `ST_Union(geomarray)` 以及更快的集合聚合。运算表示入几何形的点集并集的几何形。

### 13.12.15 PostGIS 新增功能或增强功能 (1.3)

下面列出的功能是添加或增强的 PostGIS 功能。

PostGIS 中的新功能 1.3

- **ST\_AsGML** - 可用性：1.3.2 将几何体形作为 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 将包含曲线的几何体形转换为线性几何体形。
  - **ST\_LineToCurve** - 可用性：1.3.0 将线性几何体形转换为曲线几何体形。
  - **ST\_SimplifyPreserveTopology** - 可用性：1.3.3 使用 Douglas-Peucker 算法返回几何体形的简化且有效表示。
-

# Chapter 14

## 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（假您已安装了 [git](#)）：

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。



# Appendix A

附

## A.1 PostGIS 3.6.0

2025/xx/xx