

PostGIS 3.7.0alpha1 マニュアル

Contents

1	導入	1
1.1	プロジェクト運営委員会	1
1.2	現在の中核貢献者	2
1.3	過去の中核貢献者	2
1.4	他の貢献者	3
2	PostGIS インストール	6
2.1	簡略版	6
2.2	ソースからのコンパイルとインストール	6
2.2.1	ソースの取得	7
2.2.2	インストール要件	7
2.2.3	コンフィギュレーション	8
2.2.4	ビルド	10
2.2.5	PostGIS エクステンションのビルドとデプロイ	10
2.2.6	テスト	12
2.2.7	インストール	15
2.3	共通の問題	15
3	PostGIS 管理	16
3.1	パフォーマンスチューニング	16
3.1.1	起動時	16
3.1.2	実行時	17
3.2	ラスタ機能の設定	17
3.3	空間データベースの作成	18
3.3.1	エクステンションを使って空間データベースを有効にする	18
3.3.2	エクステンションを使わずに空間データベースを有効にする (お勧めしません)	18
3.4	空間データベースのアップグレード	19
3.4.1	ソフトアップグレード	19
3.4.1.1	エクステンションを使ったソフトアップグレード	19
3.4.1.2	エクステンションを使わないソフトアップグレード	20
3.4.1.3	ハードアップグレード	21

4 データ管理	23
4.1 空間データモデル	23
4.1.1 OGC ジオメトリ	23
4.1.1.1 ポイント (Point)	24
4.1.1.2 ラインストリング (LineString)	24
4.1.1.3 リニアリング (LinearRing)	24
4.1.1.4 ポリゴン (Polygon)	24
4.1.1.5 マルチポイント (MultiPoint)	24
4.1.1.6 マルチラインストリング (MultiLineString)	25
4.1.1.7 マルチポリゴン (MultiPolygon)	25
4.1.1.8 ジオメトリコレクション (GeometryCollection)	25
4.1.1.9 多面体サーフェス (PolyhedralSurface)	25
4.1.1.10 三角形 (Triangle)	25
4.1.1.11 TIN	25
4.1.2 SQL/MM Part 3 - 曲線	26
4.1.2.1 曲線ストリング (CircularStringCircularString)	26
4.1.2.2 複合曲線 (CompoundCurve)	26
4.1.2.3 曲線ポリゴン (CurvePolygon)	26
4.1.2.4 マルチ曲線 (Multicurve)	26
4.1.2.5 マルチサーフェス (MultiSurface)	27
4.1.3 WKT と WKB	27
4.2 ジオメトリデータタイプ	28
4.2.1 PostGIS EWKB と EWKT	28
4.3 ジオグラフィデータタイプ	30
4.3.1 ジオグラフィテーブルの生成	30
4.3.2 ジオグラフィテーブルの使用	31
4.3.3 ジオグラフィ型を使用すべき時	33
4.3.4 ジオグラフィに関する高度なよくある質問	33
4.4 ジオメトリ検証	34
4.4.1 単純ジオメトリ	34
4.4.2 妥当なジオメトリ	36
4.4.3 妥当性の管理	38
4.5 空間参照系	38
4.5.1 SPATIAL_REF_SYS テーブル	39
4.5.2 ユーザー定義空間参照系	40
4.6 空間テーブル	41
4.6.1 空間テーブルを作る	41
4.6.2 GEOMETRY_COLUMNS ビュー	42
4.6.3 手動でジオメトリカラムを geometry_columns に登録する	42

4.7	空間データのロード	44
4.7.1	SQL を使ってロードする	45
4.7.2	シェープファイルローダを使う	45
4.8	空間データの抽出	47
4.8.1	SQL を使ってデータを抽出する	47
4.8.2	ダンパを使う	48
4.9	空間インデックス	49
4.9.1	GiST インデックス	49
4.9.2	BRIN インデックス	50
4.9.3	SP-GiST インデックス	51
4.9.4	インデックス使用のチューニング	52
5	空間クエリ	53
5.1	空間関係の決定	53
5.1.1	次元拡張 9 交差モデル	53
5.1.2	名前付き空間関係	55
5.1.3	一般的な空間関係	56
5.2	空間インデックスを使う	58
5.3	空間 SQL の例	58
6	性能向上に関する技法	62
6.1	大きなジオメトリを持つ小さなテーブル	62
6.1.1	問題の説明	62
6.1.2	応急処置	62
6.2	ジオメトリインデックスで CLUSTER を実行する	63
6.3	次元変換の回避	63
7	PostGIS リファレンス	65
7.1	PostGIS Geometry/Geography/Box データ型	65
7.1.1	box2d	65
7.1.2	box3d	66
7.1.3	geometry	66
7.1.4	geometry_dump	67
7.1.5	geography	67
7.2	テーブル管理関数	68
7.2.1	AddGeometryColumn	68
7.2.2	DropGeometryColumn	70
7.2.3	DropGeometryTable	71
7.2.4	Find_SRID	71
7.2.5	Populate_Geometry_Columns	72

7.2.6	UpdateGeometrySRID	73
7.3	ジオメトリコンストラクタ	74
7.3.1	ST_Collect	74
7.3.2	ST_LineFromMultiPoint	76
7.3.3	ST_MakeEnvelope	77
7.3.4	ST_MakeLine	77
7.3.5	ST_MakePoint	79
7.3.6	ST_MakePointM	80
7.3.7	ST_MakePolygon	81
7.3.8	ST_Point	83
7.3.9	ST_PointZ	84
7.3.10	ST_PointM	85
7.3.11	ST_PointZM	85
7.3.12	ST_Polygon	86
7.3.13	ST_TileEnvelope	87
7.3.14	ST_HexagonGrid	88
7.3.15	ST_Hexagon	90
7.3.16	ST_SquareGrid	91
7.3.17	ST_Square	92
7.3.18	ST_Letters	93
7.3.19	ST_MakeNurbsCurve	94
7.4	ジオメトリアクセサ	95
7.4.1	GeometryType	95
7.4.2	ST_Boundary	96
7.4.3	ST_BoundingDiagonal	98
7.4.4	ST_CoordDim	99
7.4.5	ST_Dimension	99
7.4.6	ST_Dump	100
7.4.7	ST_DumpPoints	102
7.4.8	ST_DumpSegments	106
7.4.9	ST_DumpRings	108
7.4.10	ST_EndPoint	109
7.4.11	ST_Envelope	110
7.4.12	ST_ExteriorRing	112
7.4.13	ST_GeometryN	113
7.4.14	ST_GeometryType	115
7.4.15	ST_HasArc	116
7.4.16	ST_InteriorRingN	117
7.4.17	ST_NumCurves	118

7.4.18ST_CurveN	118
7.4.19ST_IsClosed	119
7.4.20ST_IsCollection	121
7.4.21ST_IsEmpty	122
7.4.22ST_IsPolygonCCW	123
7.4.23ST_IsPolygonCW	123
7.4.24ST_IsRing	124
7.4.25ST_IsSimple	125
7.4.26ST_M	126
7.4.27ST_MemSize	127
7.4.28ST_NDims	128
7.4.29ST_NPoints	128
7.4.30ST_NRings	129
7.4.31ST_NumGeometries	130
7.4.32ST_NumInteriorRings	131
7.4.33ST_NumInteriorRing	131
7.4.34ST_NumPatches	132
7.4.35ST_NumPoints	132
7.4.36ST_PatchN	133
7.4.37ST_PointN	134
7.4.38ST_Points	135
7.4.39ST_StartPoint	136
7.4.40ST_StartM	137
7.4.41ST_EndM	138
7.4.42ST_SetStartM	139
7.4.43ST_SetEndM	140
7.4.44ST_Summary	141
7.4.45ST_X	142
7.4.46ST_Y	143
7.4.47ST_Z	144
7.4.48ST_ZMFlag	145
7.4.49ST_HasZ	145
7.4.50ST_HasM	146
7.4.51ST_ControlPoints	147
7.4.52ST_Degree	148
7.4.53ST_Weights	148
7.4.54ST_Knots	149
7.4.55ST_NumControlPoints	150
7.4.56ST_NurbsCurveIsRational	150

7.4.57	ST_IsNurbsCurve	151
7.4.58	ST_Evaluate	152
7.4.59	ST_NurbsToLineString	153
7.5	ジオメトリエディタ	153
7.5.1	ST_AddPoint	153
7.5.2	ST_CollectionExtract	154
7.5.3	ST_CollectionHomogenize	155
7.5.4	ST_CurveToLine	157
7.5.5	ST_Scroll	159
7.5.6	ST_FlipCoordinates	160
7.5.7	ST_Force2D	161
7.5.8	ST_Force3D	161
7.5.9	ST_Force3DZ	162
7.5.10	ST_Force3DM	163
7.5.11	ST_Force4D	164
7.5.12	ST_ForceCollection	164
7.5.13	ST_ForceCurve	165
7.5.14	ST_ForcePolygonCCW	166
7.5.15	ST_ForcePolygonCW	167
7.5.16	ST_ForceSFS	167
7.5.17	ST_ForceRHR	167
7.5.18	ST_LineExtend	168
7.5.19	ST_LineToCurve	169
7.5.20	ST_Multi	170
7.5.21	ST_Normalize	171
7.5.22	ST_Project	171
7.5.23	ST_QuantizeCoordinates	172
7.5.24	ST_RemovePoint	174
7.5.25	ST_RemoveRepeatedPoints	175
7.5.26	ST_RemoveIrrelevantPointsForView	176
7.5.27	ST_RemoveSmallParts	178
7.5.28	ST_Reverse	180
7.5.29	ST_Segmentize	180
7.5.30	ST_SetPoint	182
7.5.31	ST_ShiftLongitude	183
7.5.32	ST_WrapX	184
7.5.33	ST_SnapToGrid	185
7.5.34	ST_Snap	186
7.5.35	ST_SwapOrdinates	189

7.6 ジオメトリ検証	190
7.6.1 ST_IsValid	190
7.6.2 ST_IsValidDetail	191
7.6.3 ST_IsValidReason	193
7.6.4 ST_MakeValid	194
7.7 空間参照系関数	199
7.7.1 ST_InverseTransformPipeline	199
7.7.2 ST_SetSRID	200
7.7.3 ST_SRID	201
7.7.4 ST_Transform	202
7.7.5 ST_TransformPipeline	204
7.7.6 postgis_srs_codes	206
7.7.7 postgis_srs	206
7.7.8 postgis_srs_all	207
7.7.9 postgis_srs_search	208
7.8 ジオメトリ入力	209
7.8.1 Well-Known Text (WKT)	209
7.8.1.1 ST_BdPolyFromText	209
7.8.1.2 ST_BdMPolyFromText	210
7.8.1.3 ST_GeogFromText	210
7.8.1.4 ST_GeographyFromText	211
7.8.1.5 ST_GeomCollFromText	211
7.8.1.6 ST_GeomFromEWKT	212
7.8.1.7 ST_GeomFromMARC21	214
7.8.1.8 ST_GeometryFromText	216
7.8.1.9 ST_GeomFromText	216
7.8.1.10 ST_LineFromText	218
7.8.1.11 ST_MLineFromText	219
7.8.1.12 ST_MPointFromText	220
7.8.1.13 ST_MPolyFromText	220
7.8.1.14 ST_PointFromText	221
7.8.1.15 ST_PolygonFromText	222
7.8.1.16 ST_WKTToSQL	223
7.8.2 Well-Known Binary (WKB)	223
7.8.2.1 ST_GeogFromWKB	223
7.8.2.2 ST_GeomFromEWKB	224
7.8.2.3 ST_GeomFromWKB	225
7.8.2.4 ST_LineFromWKB	226
7.8.2.5 ST_LinestringFromWKB	227

7.8.2.6	ST_PointFromWKB	228
7.8.2.7	ST_WKBToSQL	229
7.8.3	その他の書式	229
7.8.3.1	ST_Box2dFromGeoHash	229
7.8.3.2	ST_GeomFromGeoHash	230
7.8.3.3	ST_GeomFromGML	231
7.8.3.4	ST_GeomFromGeoJSON	234
7.8.3.5	ST_GeomFromKML	235
7.8.3.6	ST_GeomFromTWKB	235
7.8.3.7	ST_GMLToSQL	236
7.8.3.8	ST_LineFromEncodedPolyline	236
7.8.3.9	ST_PointFromGeoHash	237
7.8.3.10	ST_FromFlatGeobufToTable	238
7.8.3.11	ST_FromFlatGeobuf	238
7.9	ジオメトリ出力	239
7.9.1	Well-Known Text (WKT)	239
7.9.1.1	ST_AsEWKT	239
7.9.1.2	ST_AsText	240
7.9.2	Well-Known Binary (WKB)	241
7.9.2.1	ST_AsBinary	241
7.9.2.2	ST_AsEWKB	243
7.9.2.3	ST_AsHEXEWKB	244
7.9.3	その他の書式	244
7.9.3.1	ST_AsEncodedPolyline	244
7.9.3.2	ST_AsFlatGeobuf	245
7.9.3.3	ST_AsGeobuf	246
7.9.3.4	ST_AsGeoJSON	246
7.9.3.5	ST_AsGML	249
7.9.3.6	ST_AsKML	252
7.9.3.7	ST_AsLatLonText	253
7.9.3.8	ST_AsMARC21	255
7.9.3.9	ST_AsMVTGeom	257
7.9.3.10	ST_AsMVT	259
7.9.3.11	ST_AsSVG	260
7.9.3.12	ST_AsTWKB	261
7.9.3.13	ST_AsX3D	262
7.9.3.14	ST_GeoHash	266
7.10	演算子	267
7.10.1	バウンディングボックス演算子	267

7.10.1.1	267
7.10.1.2(geometry,box2df)	268
7.10.1.3(box2df,geometry)	269
7.10.1.4(box2df,box2df)	269
7.10.1.5	270
7.10.1.6(geometry,gidx)	271
7.10.1.7(gidx,geometry)	272
7.10.1.8(gidx,gidx)	273
7.10.1.9<	274
7.10.1.10<	275
7.10.1.11>	276
7.10.1.12<	277
7.10.1.13<	277
7.10.1.14	278
7.10.1.15>	279
7.10.1.16@	280
7.10.1.17@ (geometry,box2df)	281
7.10.1.18@ (box2df,geometry)	282
7.10.1.19@ (box2df,box2df)	283
7.10.1.20>	283
7.10.1.21>	284
7.10.1.22	285
7.10.1.23(geometry,box2df)	286
7.10.1.24(box2df,geometry)	287
7.10.1.25(box2df,box2df)	287
7.10.1.26=	288
7.10.2 距離演算子	289
7.10.2.1<->	289
7.10.2.2 =	291
7.10.2.3<#>	292
7.10.2.4<->>	293
7.11 空間関係関数	294
7.11.1 トポロジ関係関数	294
7.11.1.1 ST_3DIntersects	294
7.11.1.2 ST_Contains	295
7.11.1.3 ST_ContainsProperly	299
7.11.1.4 ST_CoveredBy	300
7.11.1.5 ST_Covers	301
7.11.1.6 ST_Crosses	303

7.11.1.7ST_Disjoint	305
7.11.1.8ST_Equals	306
7.11.1.9ST_Intersects	307
7.11.1.10ST_LineCrossingDirection	309
7.11.1.11ST_OrderingEquals	312
7.11.1.12ST_Overlaps	313
7.11.1.13ST_Relate	316
7.11.1.14ST_RelateMatch	318
7.11.1.15ST_Touches	319
7.11.1.16ST_Within	321
7.11.2距離関係関数	323
7.11.2.1ST_3DDWithin	323
7.11.2.2ST_3DDFullyWithin	324
7.11.2.3ST_DFullyWithin	325
7.11.2.4ST_DWithin	326
7.11.2.5ST_PointInsideCircle	327
7.12計測関数	328
7.12.1ST_Area	328
7.12.2ST_Azimuth	330
7.12.3ST_Angle	331
7.12.4ST_ClosestPoint	332
7.12.5ST_3DClosestPoint	334
7.12.6ST_Distance	335
7.12.7ST_3DDistance	337
7.12.8ST_DistanceSphere	339
7.12.9ST_DistanceSpheroid	339
7.12.10ST_FrechetDistance	340
7.12.11ST_HausdorffDistance	341
7.12.12ST_Length	343
7.12.13ST_Length2D	344
7.12.14ST_3DLength	345
7.12.15ST_LengthSpheroid	345
7.12.16ST_LongestLine	347
7.12.17ST_3DLongestLine	349
7.12.18ST_MaxDistance	350
7.12.19ST_3DMaxDistance	351
7.12.20ST_MinimumClearance	352
7.12.21ST_MinimumClearanceLine	353
7.12.22ST_Perimeter	353

7.12.2	ST_Perimeter2D	355
7.12.2	ST_3DPerimeter	356
7.12.2	ST_ShortestLine	356
7.12.2	ST_3DShortestLine	358
7.13	重ね合わせ関数	359
7.13.1	ST_ClipByBox2D	359
7.13.2	ST_Difference	360
7.13.3	ST_Intersection	361
7.13.4	ST_MemUnion	364
7.13.5	ST_Node	364
7.13.6	ST_Split	365
7.13.7	ST_Subdivide	367
7.13.8	ST_SymDifference	371
7.13.9	ST_UnaryUnion	372
7.13.1	ST_Union	373
7.14	ジオメトリ処理関数	376
7.14.1	ST_Buffer	376
7.14.2	ST_BuildArea	380
7.14.3	ST_Centroid	382
7.14.4	ST_ChaikinSmoothing	384
7.14.5	ST_CatmullRomSmoothing	385
7.14.6	ST_ConcaveHull	388
7.14.7	ST_ConvexHull	391
7.14.8	ST_DelaunayTriangles	393
7.14.9	ST_FilterByM	398
7.14.1	ST_GeneratePoints	399
7.14.1	ST_GeometricMedian	400
7.14.1	ST_LineMerge	401
7.14.1	ST_MaximumInscribedCircle	403
7.14.1	ST_LargestEmptyCircle	405
7.14.1	ST_MinimumBoundingCircle	407
7.14.1	ST_MinimumBoundingRadius	408
7.14.1	ST_OrientedEnvelope	409
7.14.1	ST_OffsetCurve	410
7.14.1	ST_PointOnSurface	414
7.14.2	ST_Polygonize	416
7.14.2	ST_ReducePrecision	418
7.14.2	ST_SharedPaths	420
7.14.2	ST_Simplify	421

7.14.25	ST_SimplifyPreserveTopology	423
7.14.25	ST_SimplifyPolygonHull	425
7.14.26	ST_SimplifyVW	428
7.14.27	ST_SetEffectiveArea	429
7.14.28	ST_TriangulatePolygon	431
7.14.29	ST_VoronoiLines	433
7.14.30	ST_VoronoiPolygons	434
7.15	カバレッジ	436
7.15.1	ST_CoverageInvalidEdges	436
7.15.2	ST_CoverageClean	437
7.15.3	ST_CoverageSimplify	440
7.15.4	ST_CoverageEdges	441
7.15.5	ST_CoverageUnion	442
7.16	アフィン変換	444
7.16.1	ST_Affine	444
7.16.2	ST_Rotate	445
7.16.3	ST_RotateX	446
7.16.4	ST_RotateY	447
7.16.5	ST_RotateZ	448
7.16.6	ST_Scale	449
7.16.7	ST_Translate	451
7.16.8	ST_TransScale	452
7.17	クラスタリング関数	453
7.17.1	ST_ClusterDBSCAN	453
7.17.2	ST_ClusterIntersecting	455
7.17.3	ST_ClusterIntersectingWin	455
7.17.4	ST_ClusterRelateWin	456
7.17.5	ST_MinimumSpanningTree	458
7.17.6	ST_ClusterKMeans	458
7.17.7	ST_ClusterWithin	461
7.17.8	ST_ClusterWithinWin	461
7.18	バウンディングボックス関数	462
7.18.1	Box2D	462
7.18.2	Box3D	463
7.18.3	ST_EstimatedExtent	464
7.18.4	ST_Expand	465
7.18.5	ST_Extent	466
7.18.6	ST_3DExtent	468
7.18.7	ST_MakeBox2D	469

7.18.8	ST_3DMakeBox	469
7.18.9	ST_XMax	470
7.18.10	ST_XMin	471
7.18.11	ST_YMax	472
7.18.12	ST_YMin	473
7.18.13	ST_ZMax	474
7.18.14	ST_ZMin	475
7.18.15	ST_XSize	476
7.18.16	ST_YSize	477
7.18.17	ST_ZSize	478
7.18.18	ST_MMin	478
7.18.19	ST_MMax	479
7.18.20	ST_MSize	480
7.19	線型参照	480
7.19.1	ST_LineInterpolatePoint	480
7.19.2	ST_3DLineInterpolatePoint	482
7.19.3	ST_LineInterpolatePoints	483
7.19.4	ST_LineLocatePoint	484
7.19.5	ST_LineSubstring	485
7.19.6	ST_LocateAlong	487
7.19.7	ST_LocateBetween	488
7.19.8	ST_LocateBetweenElevations	490
7.19.9	ST_InterpolatePoint	490
7.19.10	ST_3DInterpolatePoint	491
7.19.11	ST_AddMeasure	492
7.20	トラジェクトリ関数	493
7.20.1	ST_IsValidTrajectory	493
7.20.2	ST_ClosestPointOfApproach	494
7.20.3	ST_DistanceCPA	495
7.20.4	ST_CPAWithin	496
7.21	バージョン関数	496
7.21.1	PostGIS_Extensions_Upgrade	496
7.21.2	PostGIS_Full_Version	497
7.21.3	PostGIS_GEOS_Version	498
7.21.4	PostGIS_GEOS_Compiled_Version	498
7.21.5	PostGIS_Liblwgeom_Version	499
7.21.6	PostGIS_LibXML_Version	499
7.21.7	PostGIS_LibJSON_Version	500
7.21.8	PostGIS_Lib_Build_Date	500

7.21.9	PostGIS_Lib_Version	501
7.21.10	PostGIS_PROJ_Version	501
7.21.11	PostGIS_PROJ_Compiled_Version	502
7.21.12	PostGIS_Wagyu_Version	502
7.21.13	PostGIS_Scripts_Build_Date	503
7.21.14	PostGIS_Scripts_Installed	503
7.21.15	PostGIS_Scripts_Released	504
7.21.16	PostGIS_Version	505
7.22	Grand Unified Custom 変数 (GUC)	505
7.22.1	postgis.gdal_datapath	505
7.22.2	postgis.gdal_enabled_drivers	506
7.22.3	postgis.enable_outdb_rasters	508
7.22.4	postgis.gdal_vsi_options	509
7.22.5	postgis.gdal_cpl_debug	509
7.23	トラブルシューティング関数	509
7.23.1	PostGIS_AddBBox	509
7.23.2	PostGIS_DropBBox	510
7.23.3	PostGIS_HasBBox	511
8	SFCGAL 関数リファレンス	512
8.1	SFCGAL 管理関数	512
8.1.1	postgis_sfcgal_version	512
8.1.2	postgis_sfcgal_full_version	512
8.2	SFCGAL アクセサとセッター	513
8.2.1	CG_ForceLHR	513
8.2.2	CG_IsPlanar	513
8.2.3	CG_IsSolid	514
8.2.4	CG_MakeSolid	514
8.2.5	CG_Orientation	515
8.2.6	CG_Area	515
8.2.7	CG_3DArea	516
8.2.8	CG_Volume	516
8.2.9	ST_ForceLHR	517
8.2.10	ST_IsPlanar	518
8.2.11	ST_IsSolid	518
8.2.12	ST_MakeSolid	519
8.2.13	ST_Orientation	519
8.2.14	ST_3DArea	520
8.2.15	ST_Volume	521

8.3	SFCGAL 処理関数および関係関数	522
8.3.1	CG_Intersection	522
8.3.2	CG_Intersects	523
8.3.3	CG_3DIntersects	523
8.3.4	CG_Difference	524
8.3.5	ST_3DDifference	525
8.3.6	CG_3DDifference	526
8.3.7	CG_Distance	527
8.3.8	CG_3DDistance	527
8.3.9	ST_3DConvexHull	528
8.3.10	CG_3DConvexHull	529
8.3.11	ST_3DIntersection	530
8.3.12	CG_3DIntersection	531
8.3.13	CG_Union	533
8.3.14	ST_3DUnion	534
8.3.15	CG_3DUnion	534
8.3.16	ST_AlphaShape	535
8.3.17	CG_AlphaShape	536
8.3.18	CG_ApproxConvexPartition	539
8.3.19	ST_ApproximateMedialAxis	541
8.3.20	CG_ApproximateMedialAxis	541
8.3.21	CG_GenerateFlatRoof	542
8.3.22	CG_GenerateHippedRoof	543
8.3.23	CG_GenerateGableRoof	544
8.3.24	CG_GenerateSkillionRoof	545
8.3.25	CG_GenerateRoof	546
8.3.26	CG_PolygonRepair	547
8.3.27	ST_ConstrainedDelaunayTriangles	548
8.3.28	CG_ConstrainedDelaunayTriangles	548
8.3.29	ST_Extrude	549
8.3.30	CG_Extrude	550
8.3.31	CG_ExtrudeStraightSkeleton	551
8.3.32	CG_GreeneApproxConvexPartition	552
8.3.33	ST_MinkowskiSum	554
8.3.34	CG_MinkowskiSum	554
8.3.35	ST_OptimalAlphaShape	556
8.3.36	CG_OptimalAlphaShape	557
8.3.37	CG_OptimalConvexPartition	559
8.3.38	CG_StraightSkeleton	560

8.3.39	ST_StraightSkeleton	561
8.3.40	ST_Tessellate	562
8.3.41	CG_Tessellate	563
8.3.42	CG_Triangulate	565
8.3.43	CG_Visibility	566
8.3.44	CG_YMonotonePartition	567
8.3.45	CG_StraightSkeletonPartition	568
8.3.46	CG_3DBuffer	569
8.3.47	CG_Rotate	571
8.3.48	CG_2DRotate	572
8.3.49	CG_3DRotate	572
8.3.50	CG_RotateX	573
8.3.51	CG_RotateY	574
8.3.52	CG_RotateZ	574
8.3.53	CG_Scale	575
8.3.54	CG_3DScale	575
8.3.55	CG_3DScaleAroundCenter	576
8.3.56	CG_Translate	577
8.3.57	CG_3DTranslate	577
8.3.58	CG_Simplify	578
8.3.59	CG_3DAlphaWrapping	581
8.3.60	CG_NurbsCurveInterpolate	583
8.3.61	CG_NurbsCurveApproximate	584
8.3.62	CG_NurbsCurveDerivative	585
9	トポロジ	587
9.1	トポロジープリミティブテーブル	587
9.2	トポロジ型	588
9.2.1	getfaceedges_returntype	588
9.2.2	TopoGeometry	588
9.2.3	validatetopology_returntype	589
9.3	トポロジドメイン	589
9.3.1	TopoElement	589
9.3.2	TopoElementArray	590
9.4	トポロジ管理と TopoGeometry 管理	591
9.4.1	AddTopoGeometryColumn	591
9.4.2	RenameTopoGeometryColumn	592
9.4.3	DropTopology	592
9.4.4	RenameTopology	593

9.4.5	DropTopoGeometryColumn	593
9.4.6	FixCorruptTopoGeometryColumn	594
9.4.7	Populate_Topology_Layer	595
9.4.8	TopologySummary	596
9.4.9	ValidateTopology	596
9.4.10	ValidateTopologyRelation	599
9.4.11	ValidateTopologyPrecision	599
9.4.12	MakeTopologyPrecise	600
9.4.13	FindTopology	600
9.4.14	FindLayer	601
9.4.15	TotalTopologySize	602
9.4.16	UpgradeTopology	602
9.5	トポロジ統計管理	603
9.6	トポロジコンストラクタ	603
9.6.1	CreateTopology	603
9.6.2	CopyTopology	604
9.6.3	ST_InitTopoGeo	605
9.6.4	ST_CreateTopoGeo	605
9.6.5	TopoGeo_AddPoint	606
9.6.6	TopoGeo_AddLineString	607
9.6.7	TopoGeo_AddPolygon	607
9.6.8	TopoGeo_LoadGeometry	608
9.7	トポロジエディタ	609
9.7.1	ST_AddIsoNode	609
9.7.2	ST_AddIsoEdge	609
9.7.3	ST_AddEdgeNewFaces	610
9.7.4	ST_AddEdgeModFace	610
9.7.5	ST_RemEdgeNewFace	611
9.7.6	ST_RemEdgeModFace	612
9.7.7	ST_ChangeEdgeGeom	613
9.7.8	ST_ModEdgeSplit	614
9.7.9	ST_ModEdgeHeal	614
9.7.10	ST_NewEdgeHeal	615
9.7.11	ST_MoveIsoNode	615
9.7.12	ST_NewEdgesSplit	616
9.7.13	ST_RemoveIsoNode	617
9.7.14	ST_RemoveIsoEdge	618
9.8	トポロジアクセサ	618
9.8.1	GetEdgeByPoint	618

9.8.2	GetFaceByPoint	619
9.8.3	GetFaceContainingPoint	620
9.8.4	GetNodeByPoint	621
9.8.5	GetTopologyID	622
9.8.6	GetTopologySRID	622
9.8.7	GetTopologyName	623
9.8.8	ST_GetFaceEdges	623
9.8.9	ST_GetFaceGeometry	624
9.8.10	GetRingEdges	625
9.8.11	GetNodeEdges	625
9.8.12	FindVertexSegmentPairsBelowDistance	626
9.9	トポロジ処理	626
9.9.1	Polygonize	626
9.9.2	AddNode	627
9.9.3	AddEdge	628
9.9.4	AddFace	629
9.9.5	ST_Simplify	631
9.9.6	RemoveUnusedPrimitives	631
9.10	TopoGeometry コンストラクタ	632
9.10.1	CreateTopoGeom	632
9.10.2	toTopoGeom	633
9.10.3	TopoElementArray_Agg	635
9.10.4	TopoElement	635
9.11	TopoGeometry エディタ	636
9.11.1	clearTopoGeom	636
9.11.2	TopoGeom_addElement	637
9.11.3	TopoGeom_remElement	637
9.11.4	TopoGeom_addTopoGeom	638
9.11.5	toTopoGeom	638
9.12	TopoGeometry アクセサ	639
9.12.1	GetTopoGeomElementArray	639
9.12.2	GetTopoGeomElements	639
9.12.3	ST_SRID	640
9.13	TopoGeometry 出力	640
9.13.1	AsGML	640
9.13.2	AsTopoJSON	643
9.14	トポロジ空間関係関数	644
9.14.1	Equals	644
9.14.2	Intersects	645
9.15	トポロジのインポートとエクスポート	646
9.15.1	トポロジエクスポートの使用	646
9.15.2	トポロジインポートの使用	647

10	ラスタデータの管理、クエリ、アプリケーション	648
10.1	ラスタのロードと生成	648
10.1.1	raster2pgsql を使ってラスタをロードする	648
10.1.1.1	使用例	648
10.1.1.2	raster2pgsql オプション	649
10.1.2	PostGIS ラスタ関数を用いたラスタの生成	651
10.1.3	「データベース外」クラウドラスタの使用	652
10.2	ラスタカタログ	652
10.2.1	ラスタカラムカタログ	653
10.2.2	ラスタオーバビュー	654
10.3	PostGIS ラスタを使ったカスタムアプリケーションの構築	654
10.3.1	ST_AsPNG を他の関数とあわせて使った PHP 出力例	655
10.3.2	ST_AsPNG を他の関数とあわせて使った ASP.NET C# 出力例	655
10.3.3	raster クエリを画像ファイルで出力する Java コンソールアプリケーション	657
10.3.4	PLPython を使って SQL を介して画像をダンプする	658
10.3.5	PSQL でラスタを出力する	659
11	ラスタリファレンス	660
11.1	ラスタサポートデータ型	661
11.1.1	geomval	661
11.1.2	addbandarg	661
11.1.3	rastbandarg	661
11.1.4	raster	662
11.1.5	reclassarg	662
11.1.6	summarystats	663
11.1.7	unionarg	663
11.2	ラスタ管理	664
11.2.1	AddRasterConstraints	664
11.2.2	DropRasterConstraints	666
11.2.3	AddOverviewConstraints	667
11.2.4	DropOverviewConstraints	668
11.2.5	PostGIS_GDAL_Version	668
11.2.6	PostGIS_Raster_Lib_Build_Date	669
11.2.7	PostGIS_Raster_Lib_Version	669
11.2.8	ST_GDALDrivers	670
11.2.9	UpdateRasterSRID	675
11.2.10	ST_CreateOverview	675
11.3	ラスタコンストラクタ	676
11.3.1	ST_AddBand	676

11.3.2	ST_AsRaster	679
11.3.3	ST_AsRasterAgg	681
11.3.4	ST_Band	682
11.3.5	ST_MakeEmptyCoverage	684
11.3.6	ST_MakeEmptyRaster	685
11.3.7	ST_Tile	686
11.3.8	ST_Retile	688
11.3.9	ST_FromGDALRaster	689
11.4	ラスタアクセサ	690
11.4.1	ST_GeoReference	690
11.4.2	ST_Height	691
11.4.3	ST_IsEmpty	691
11.4.4	ST_MemSize	692
11.4.5	ST_MetaData	693
11.4.6	ST_NumBands	693
11.4.7	ST_PixelHeight	694
11.4.8	ST_PixelWidth	695
11.4.9	ST_ScaleX	696
11.4.10	ST_ScaleY	696
11.4.11	ST_RasterToWorldCoord	697
11.4.12	ST_RasterToWorldCoordX	698
11.4.13	ST_RasterToWorldCoordY	699
11.4.14	ST_Rotation	700
11.4.15	ST_SkewX	701
11.4.16	ST_SkewY	701
11.4.17	ST_SRID	702
11.4.18	ST_Summary	703
11.4.19	ST_UpperLeftX	703
11.4.20	ST_UpperLeftY	704
11.4.21	ST_Width	705
11.4.22	ST_WorldToRasterCoord	705
11.4.23	ST_WorldToRasterCoordX	706
11.4.24	ST_WorldToRasterCoordY	707
11.5	ラスタバンドアクセサ	707
11.5.1	ST_BandMetaData	707
11.5.2	ST_BandNoDataValue	709
11.5.3	ST_BandIsNoData	710
11.5.4	ST_BandPath	711
11.5.5	ST_BandFileSize	711

11.5.6	ST_BandFileTimestamp	712
11.5.7	ST_BandPixelType	712
11.5.8	ST_MinPossibleValue	713
11.5.9	ST_HasNoBand	714
11.6	ラスタ 픽セル 액세스와 세터	715
11.6.1	ST_PixelAsPolygon	715
11.6.2	ST_PixelAsPolygons	715
11.6.3	ST_PixelAsPoint	716
11.6.4	ST_PixelAsPoints	717
11.6.5	ST_PixelAsCentroid	718
11.6.6	ST_PixelAsCentroids	719
11.6.7	ST_Value	720
11.6.8	ST_NearestValue	723
11.6.9	ST_SetZ	725
11.6.10	ST_SetM	726
11.6.11	ST_Neighborhood	727
11.6.12	ST_SetValue	729
11.6.13	ST_SetValues	730
11.6.14	ST_DumpValues	738
11.6.15	ST_PixelOfValue	739
11.7	라스타 에디터	741
11.7.1	ST_SetGeoReference	741
11.7.2	ST_SetRotation	742
11.7.3	ST_SetScale	743
11.7.4	ST_SetSkew	744
11.7.5	ST_SetSRID	745
11.7.6	ST_SetUpperLeft	745
11.7.7	ST_Resample	746
11.7.8	ST_Rescale	747
11.7.9	ST_Reskew	749
11.7.10	ST_SnapToGrid	750
11.7.11	ST_Resize	751
11.7.12	ST_Transform	753
11.8	라스타 밴드 에디터	755
11.8.1	ST_SetBandNoDataValue	755
11.8.2	ST_SetBandIsNoData	756
11.8.3	ST_SetBandPath	758
11.8.4	ST_SetBandIndex	759
11.9	라스타 밴드 통계 정보와 해석	760

11.9.1	ST_Count	760
11.9.2	ST_CountAgg	761
11.9.3	ST_Histogram	762
11.9.4	ST_Quantile	764
11.9.5	ST_SummaryStats	766
11.9.6	ST_SummaryStatsAgg	768
11.9.7	ST_ValueCount	769
11.10	スタ入力	772
11.10.1	ST_RastFromWKB	772
11.10.2	ST_RastFromHexWKB	773
11.11	出力	773
11.11.1	ST_AsBinary/ST_AsWKB	773
11.11.2	ST_AsHexWKB	774
11.11.3	ST_AsGDALRaster	775
11.11.4	ST_AsJPEG	776
11.11.5	ST_AsPNG	777
11.11.6	ST_AsTIFF	778
11.12	スタ処理: 地図代数	779
11.12.1	ST_Clip	779
11.12.2	ST_ColorMap	783
11.12.3	ST_Grayscale	786
11.12.4	ST_Intersection	788
11.12.5	ST_MapAlgebra (callback function version)	789
11.12.6	ST_MapAlgebra (expression version)	796
11.12.7	ST_MapAlgebraExpr	798
11.12.8	ST_MapAlgebraExpr	801
11.12.9	ST_MapAlgebraFct	805
11.12.10	ST_MapAlgebraFct	809
11.12.11	ST_MapAlgebraFctNgb	813
11.12.12	ST_Reclass	815
11.12.13	ST_ReclassExact	817
11.12.14	ST_Union	819
11.13	組み込み地図代数コールバック関数	820
11.13.1	ST_Distinct4ma	820
11.13.2	ST_InvDistWeight4ma	821
11.13.3	ST_Max4ma	822
11.13.4	ST_Mean4ma	824
11.13.5	ST_Min4ma	825
11.13.6	ST_MinDist4ma	826

11.13.\$T_Range4ma	827
11.13.\$T_StdDev4ma	828
11.13.\$T_Sum4ma	829
11.14 スタ処理: DEM (標高)	830
11.14.\$T_Aspect	830
11.14.\$T_HillShade	832
11.14.\$T_Roughness	834
11.14.\$T_Slope	835
11.14.\$T_TPI	836
11.14.\$T_TRI	837
11.14.\$T_InterpolateRaster	838
11.14.\$T_Contour	839
11.15 スタ処理: ラスタからジオメトリ	840
11.15.Box3D	840
11.15.\$T_ConvexHull	841
11.15.\$T_DumpAsPolygons	842
11.15.\$T_Envelope	843
11.15.\$T_MinConvexHull	844
11.15.\$T_Polygon	845
11.15.\$T_IntersectionFractions	846
11.16 スタ演算子	847
11.16.&&	847
11.16.&<	848
11.16.&>	849
11.16.#	850
11.16.@	850
11.16.=	851
11.16.?	852
11.17 スタとラスタバンドの空間関係関数	852
11.17.\$T_Contains	852
11.17.\$T_ContainsProperly	853
11.17.\$T_Covers	854
11.17.\$T_CoveredBy	855
11.17.\$T_Disjoint	856
11.17.\$T_Intersects	857
11.17.\$T_Overlaps	858
11.17.\$T_Touches	859
11.17.\$T_SameAlignment	860
11.17.\$T_NotSameAlignmentReason	861

11.17.\$T_Within	862
11.17.\$Z_DWithin	863
11.17.\$Z_DFullyWithin	864
11.18. スタに関する技法	865
11.18. データベース外ラスタ	865
11.18.1. 多数のファイルを持つディレクトリ	865
11.18.1. 開くことができるファイルの最大数	865
11.18.1.2.1. システム全体で開くことができるファイルの最大数	866
11.18.1.2.2. プロセスごとの開けるファイルの最大数	866
12 PostGIS 追加機能	868
13 PostGIS 関数索引	869
13.1 PostGIS 集約関数	869
13.2 PostGIS ウィンドウ関数	870
13.3 PostGIS SQL-MM 準拠関数	871
13.4 PostGIS ジオグラフィ対応関数	875
13.5 PostGIS ラスタ機能関数	876
13.6 PostGIS ジオメトリ/ジオグラフィ/ラスタのダンプ関数	882
13.7 PostGIS ボックス関数	882
13.83 次元対応 PostGIS 関数	884
13.9 PostGIS 曲線ジオメトリ対応関数	890
13.10 PostGIS 多面体サーフェス対応関数	894
13.11 PostGIS 関数対応マトリクス	897
13.12 新規作成/機能強化/変更された PostGIS 関数	917
13.12.1 PostGIS 3.7 で新規作成/機能強化された関数	917
13.12.2 PostGIS 3.6 で新規作成/機能強化された関数	919
13.12.3 PostGIS 3.5 で新規作成/機能強化された関数	920
13.12.4 PostGIS 3.4 で新規作成/機能強化された関数	922
13.12.5 PostGIS 3.3 で新規作成/機能強化された関数	923
13.12.6 PostGIS 3.2 で新規作成/機能強化された関数	924
13.12.7 PostGIS 3.1 で新規作成/機能強化された関数	926
13.12.8 PostGIS 3.0 で新規作成/機能強化された関数	927
13.12.9 PostGIS 2.5 で新規作成/機能強化された関数	929
13.12.10 PostGIS 2.4 で新規作成/機能強化された関数	931
13.12.11 PostGIS 2.3 で新規作成/機能強化された関数	932
13.12.12 PostGIS 2.2 で新規作成/機能強化された関数	934
13.12.13 PostGIS 2.1 で新規作成/機能強化された関数	936
13.12.14 PostGIS 2.0 で新規作成/機能強化された関数	942
13.12.15 PostGIS 1.5 で新規作成/機能強化された関数	952
13.12.16 PostGIS 1.4 で新規作成/機能強化された関数	954
13.12.17 PostGIS 1.3 で新規作成/機能強化された関数	954

14	問題を報告する	955
14.1	ソフトウェアのバグを報告する	955
14.2	文書の問題を報告する	955

A	付録	956
A.1	PostGIS 3.7.0alpha1	956
A.1.1	大幅な変更	956
A.1.2	新機能	956
A.1.3	性能強化	957
A.1.4	Bug Fixes	958

Abstract

PostGIS は、オブジェクト RDB である **PostgreSQL** の拡張で、GIS (地理情報システム) オブジェクトを格納することができます。PostGIS は、GiST ベースの R 木空間インデックスをサポートし、GIS オブジェクトの解析および処理を行う機能を持ちます。



本マニュアルは、3.7.0alpha1 版のマニュアルです。



この作品は **クリエイティブ・コモンズ表示 - 継承 3.0 非移植ライセンス** の下に提供されています。好きなようにこの材料を使うことができますが、PostGIS Project のクレジット提示を求めます。また可能な限り <http://postgis.net> へのリンクを求めます。

Chapter 1

導入

PostGIS は、PostgreSQL リレーショナルデータベースの空間拡張です。Refractions Research Inc が、空間データベース技術の研究プロジェクトとして作成しました。Refractions はカナダ・ブリティッシュコロンビア州・ビクトリアにある、データインテグレーションとカスタムソフトウェア開発に特化した、GIS とデータベースのコンサルティング会社です。

PostGIS は、現在では OSGeo 財団のプロジェクトです。多数の FOSS4G 開発者と PostGIS の機能と多彩さから大きな利益を得る世界中の企業が、PostGIS の開発と資金提供を行っています。

PostGIS プロジェクトの開発グループは、PostGIS が、OGC と SQL/MM 空間標準の領域における重要な GIS 機能、高度なトポロジ構築 (カバレッジ、サーフェス、ネットワーク)、GIS データの表示と編集を行うデスクトップユーザインタフェースツールのデータソース、およびウェブベースのアクセスツールのためのデータソースに、より良く対応するよう、サポートと機能強化を行う予定です。

1.1 プロジェクト運営委員会

PostGIS プロジェクト運営委員会 (PostGIS Project Steering Committee, PSC) は、総合的な指示、リリースサイクル、ドキュメンテーション、支援活動に関する調整を行っています。また、委員会は、全体的なユーザサポート、PostGIS コミュニティからのパッチの受け付けと適用、開発者のコミットのアクセス、新しい委員、API の重要な変更といった、PostGIS を含む雑多な問題に関する投票を行っています。

Raúl Marín Rodríguez MVT 機能、誤り修正、パフォーマンスと安定性の向上、GitHub キュレーション、PostGIS と PostgreSQL のリリースの調整

Regina Obe CI とウェブサイトメンテナンス、Windows 製品版と実験版のビルド、文書作成、PostGIS と PostgreSQL とのリリース整合、X3D 対応、管理関数。

Darafei Praliaskouski さん インデックス改善、誤り修正とジオメトリ/ジオグラフィ関数の改善、SFCGAL、ラスタ、GitHub キュレーション、継続的インテグレーション対応。

Paul Ramsey (委員長) PostGIS プロジェクトの副創始者。総合的なバグフィクス、ジオグラフィ機能、ジオグラフィとジオメトリのインデックス機能 (2 次元、3 次元、n 次元インデックスとあらゆる空間インデックス)、ジオメトリ内部構造、GEOS 機能の統合と GEOS リリースとの調整、PostgreSQL のリリースとの調整、ローダ/ダンパ、シェープファイル GUI ローダ。

Sandro Santilli 誤り修正とメンテナンス、継続的インテグレーション対応、git ミラーの管理、関数管理、GEOS の新機能の統合、GEOS リリースとの調整、トポロジ機能、ラスタフレームワークと低水準 API 関数。

1.2 現在の中核貢献者

Nicklas Avén 距離関数の強化 (3 次元距離、関係関数を含む) と追加、Tiny WKB 出力書式 (TWKB) と一般的なユーザサポート。

Loïc Bartoletti SFCGAL 機能強化とメンテナンスと継続的インテグレーション対応

Dan Baston ジオメトリクラスタリング関数の追加、他のジオメトリアルゴリズムの強化、GEOS の強化、および全体のユーザ対応

Martin Davis GEOS 機能強化と文書

Björn Harrtell MapBox ベクタタイル関数、GeoBuf 関数、Flatgeobuf 関数。Gitea の試験と GitLab の実験。

Aliaksandr Kalenik ジオメトリ処理、PostgreSQL GiST、共通部の誤り修正

Laurențiu Nicola 誤り修正、メンテナンス、安定化、効率化

1.3 過去の中核貢献者

Bborie Park 以前の PSC メンバ。ラスタ開あ h 津、GDAL との統合、ラスタローダ、ユーザ対応、共通部の誤り修正、様々な OS (Slackware、Mac、Windows 等多数) での試験

Mark Cave-Ayland 以前の PSC メンバ。誤り修正とメンテナンスの活動、空間インデックス選択性とバインデイング、ローダ/ダンパ、およびシェープファイル GUI ローダの調整、新機能の統合と強化。

Jorge Arévalo ラスター開発、GDAL ドライバー機能、ローダー。

Olivier Courtin (名誉) XML (KML, GML)/GeoJSON 入出力関数と 3 次元対応と誤り修正。

Chris Hodgson 以前の PSC メンバ。一般的な開発、サイトと Buildbot のメンテナンス、OSGeo インキューベーション管理。

Mateusz Loskot CMake の PostGIS への対応、Python 版のオリジナルのラスタローダと低級ラスタ API 関数の構築。

Kevin Neufeld 以前の PSC メンバ。文書と文書補助ツール、Buildbot のメンテナンス、PostGIS ニュースグループでの高度なユーザサポート、PostGIS メンテナンス機能の強化。

Dave Blasby PostGIS のオリジナルの開発/副創始者。サーバサイドのオブジェクト、インデックスのバインデイングや多数のサーバサイドの解析機能を記述。

Jeff Lounsbury シェープファイルのローダ/ダンパのオリジナル開発者。

Mark Leslie 中核機能の、継続的なメンテナンスと開発。曲線機能の強化。シェープファイル GUI ローダ。

Pierre Racine PostGIS ラスタ実装の設計。ラスタ全体のアーキテクチャ、プロトタイプ作成、プログラミング補助

David Zwarg ラスタ開発 (ほとんど地図代数解析関数)。

1.4 他の貢献者

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

企業 これらは、PostGIS プロジェクトに開発者の時間、ホスティング、または直接の資金提供のいずれかの貢献をした企業です。アルファベット順:

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

- [Azavea](#)
 - [Boundless](#)
 - [Cadcorp](#)
 - [Camptocamp](#)
 - [Carto](#)
 - [Crunchy Data](#)
 - [ボストン市 \(近隣地区開発局\)](#)
 - [ヘルシンキ市](#)
 - [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](#)
 - [ミシガン工科大学研究所](#)
 - [カナダ天然資源省](#)
 - [ノルウェー森林景観研究所](#)
 - [ノルウェー生物経済研究所 \(NIBIO\)](#)
 - [OSGeo 財団](#)
 - [Oslandia](#)
 - [Palantir Technologies](#)
 - [Paragon Corporation](#)
 - [Postgres Pro](#)
 - [R3 GIS](#)
 - [Refractions Research](#)
 - [トスカナ州- SITA](#)
 - [Safe Software](#)
 - [Sirius Corporation plc](#)
 - [ウスター市](#)
 - [カリフォルニア大デービス校節足動媒介感染症センター](#)
 - [ラヴァル大学](#)
 - [米国国勢調査局](#)
 - [米国国務省 \(人道情報部門\)](#)
 - [Zonar Systems](#)
-

クラウドファンディングキャンペーン クラウドファンディングキャンペーンは、PostGIS 開発チームが走らせているキャンペーンです。欲しくて仕方ない機能に資金を与えて、多数の人々にサービスを提供できるようにするためのものです。それぞれのキャンペーンでは、特定の機能または機能の集合に焦点が当てられます。それぞれのスポンサーは、必要な資金提供のうち少しだけを提供し、十分な人/組織の寄付で、たくさんの助けになる作業に支払う基金を持ちます。他の多くの人が寄付に協力してくれそうな機能に関するアイデアがありましたら、[PostGIS newsgroup](#)に、その考えを投稿して下さい。一緒に実現できます。

PostGIS は核に機能のコミュニティでの資金集めを試しています。[PledgeBank](#)を使って、2 件の成功事例がありました。

postgistopology - 10 以上のスポンサーが TopoGeometry 機能の構築と 2.0.0 でのトポロジ対応強化とのために、それぞれ 250 米ドルを寄付しました。

postgis64windows - 20 のスポンサーが、Windows 上での PostGIS 64 ビット版に必要な作業のために、それぞれ 100 米ドルを寄付しました。

重要なサポライブラリ ジオメトリ演算ライブラ **GEOS**

地理空間データ抽象化ライブラリ**GDAL**は、PostGIS 2 で導入されたラスタ機能の多くに使われています。また、GDAL の PostGIS 対応に必要な改善で GDAL プロジェクトに貢献しています。

地図投影ライブラリ**PROJ**

最後ですがおろそかにできないのが**PostgreSQL**です。PostGIS はこの巨人に立っています。PostGIS の速度と柔軟性は、PostgreSQL が提供する拡張性、優れたクエリプランナ、GiST インデックス、多数の SQL 機能があって初めて成り立ちます。

Chapter 2

PostGIS インストール

本章では、PostGIS のインストールに必要な手順について説明します。

2.1 簡略版

全ての依存がパスに入っているとする場合、次のようにコンパイルします。

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

PostGIS をインストールした後は、利用したいデータベース個々内で利用可能にする (Section 3.3) か、アップグレード (Section 3.4) する必要があります。

2.2 ソースからのコンパイルとインストール

Note

多くの OS で、ビルドされた PostgreSQL/PostGIS パッケージがあります。多くの場合、コンパイルが必要なのは、最もひどい最先端の版が欲しい場合やパッケージメンテナンスを行う人ぐらいです。

本節では、一般的なコンパイル手順を示します。Windows 用や他の OS 用等にコンパイルするなら、[PostGIS User contributed compile guides](#)や[PostGIS Dev Wiki](#)で、より詳細な助けが見つかるかも知れません。

多くの OS 用のビルド済みパッケージの一覧は[PostGIS Pre-built Packages](#)にあります。

Windows ユーザの場合は、スタックビルダか、[PostGIS Windows download site](#)から安定版を得ることができます。また、週に 1 回か 2 回のビルドと刺激的なことがあった時の随時ビルドとを行っている[very bleeding-edge windows experimental builds](#)もあります。これらは PostGIS の進行中のリリースでの試験に使用します。

PostGIS モジュールは、PostgreSQL バックエンドサーバの拡張です。PostGIS 3.7.0alpha1 では、コンパイルのために、完全な PostgreSQL サーバヘッダが必要です。PostgreSQL 14 - 18 の間でビルドできます。古い版の PostgreSQL はサポートされません。

PostgreSQL をインストールしていないなら PostgreSQL インストールガイドを参照して下さい。<http://www.postgresql.org/docs/>にあります。

Note

GEOS 機能を有効にするために、PostgreSQL をインストール時に明示的に標準 C++ ライブラリに対する明示的なリンクが必要になる場合があります。



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

これは、古い開発ツールとインチキ C++ 例外との対話のための応急処置です。怪しい問題 (望んでいないのにバックエンドが閉じたりそれに近い挙動を起こす) を経験したなら、このトリックを試してみてください。もちろん、これを行うには PostgreSQL をはじめてからコンパイルし直す必要があります。

次のステップでは、PostGIS ソースのコンフィギュレーションとコンパイルに概要を記述します。これらは、Linux ユーザ用に書いてありますので、Windows や Mac では動作しません。

2.2.1 ソースの取得

ダウンロードサイト <https://download.osgeo.org/postgis/source/postgis-3.7.0alpha1.tar.gz> からソースのアーカイブを入手します。

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

これで、カレントディレクトリの下に `postgis-3.7.0alpha1` ができます。

もしくは `git` レポジトリ <https://git.osgeo.org/gitea/postgis/postgis/> からチェックアウトします。

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

新しく作られた `postgis` ディレクトリに移動して、インストールを続けます。

```
./configure
```

2.2.2 インストール要件

PostGIS のビルドと利用のために、次のものがが必要です。

必須

- PostgreSQL 14 - 18、18。PostgreSQL の完全なインストール (サーバプログラムのヘッダファイルを含む) が必要です。PostgreSQL は <https://www.postgresql.org> にあります。
完全な PostgreSQL/PostGIS 対応表と PostGIS/GEOS 対応表については <http://trac.osgeo.org/postgis/wiki/UsersWikiPostgreSQLPostGIS> をご覧ください。
- GNU C コンパイラ (gcc)。ANSI C コンパイラの中には、PostGIS をコンパイルできるものもありますが、gcc でコンパイルするのが最も問題が少ないと見ています。
- GNU Make (gmake または make)。多くのシステムで、GNU make がデフォルトの make になっています。make -v を実行して版を確認して下さい。他版の make では、PostGIS の Makefile を完全に処理しきれないかもしれません。
- 投影変換ライブラリ Proj。Proj 6.1 以上が必要です。Proj ライブラリは、PostGIS の座標系投影変換機能に使われます。Proj は <https://proj.org/> からダウンロードできます。
- ジオメトリライブラリ GEOS、3.8.0 以上ですが、新しい関数と機能を完全に活用するには GEOS 3.14 が必要です。GEOS は <https://libgeos.org/> からダウンロードできます。

- LibXML2, 2.5.x 以上。現在は、LibXML2 はインポート関数 (ST_GeomFromGML と ST_GeomFromKML) で使われています。LibXML2 は<https://gitlab.gnome.org/GNOME/libxml2/-/releases>からダウンロードできます。
- JSON-C 0.9 以上。JSON-C は現在、ST_GeomFromGeoJson による GeoJSON の取り込みに使われます。JSON-C は<https://github.com/json-c/json-c/releases/>からダウンロード可能です。
- The GNU Multiple Precision Arithmetic Library (libgmp). Used to control precision of math operations in PostGIS Topology. GMP is available for download from <https://gmplib.org/>.
- GDAL バージョン 3 以上が推奨されます。ラスタ機能に必要です。<https://gdal.org/download.html>にあります。
- PostgreSQL+JIT でコンパイルする場合には、LLVM 6 版以上が必要です。<https://trac.osgeo.org/postgis/-/ticket/4125>を参照して下さい。

オプション

- GDAL (擬似的任意)。ラスターが必要ない場合に限り不要です。Section 3.2の説明に従って使用したいドライバーを有効にしてください。
- GTK (GTK+2.0, 2.8+ が必要)。シェープファイルのローダである shp2pgsql-gui のコンパイル用です。<http://www.gtk.org/>にあります。
- SFCGAL, 1.4.1 or higher is required, 2.3+ is needed to be able to use all functionality. SFCGAL can be used to provide additional 2D and 3D advanced analysis functions to PostGIS cf Chapter 8. <https://sfcgal.gitlab.io/sfcgal/>.
- ST_AsMVT を有効にするには、protobuf-c ライブラリ (実行時) と protoc-c コンパイラ (ビルド時) が必要です。protobuf-c の正しい最小版を確認するには、pkg-config が必要です。[protobuf-c](#)をご覧ください。デフォルトでは、PostGIS は、MVT ポリゴンを高速に評価するために Wagyu を使用していますが、C++11 コンパイラが必要です。CXXFLAGS を使って、PostgreSQL インストールに使ったのと同じコンパイラを使います。これを無効化して GEOS を代わりに使う場合には、コンフィギュレーション時に `--without-wagyu` を指定します。
- CUnit (CUnit)。レグレッションテストに必要です。<http://cunit.sourceforge.net/>にあります。
- DocBook (xsltproc)。文書のビルドに必要です。<http://www.docbook.org/>にあります。
- DBLatex (dlatex)。文書を PDF でビルドするのに必要です。<http://dlatex.sourceforge.net/>にあります。
- GraphicsMagick (gm convert) は文書で使われる画像の生成に必要です。GraphicsMagick がない場合には、ImageMagick (convert または magick convert) で同じようなコマンドライン構文が使えます。GraphicsMagick は<http://www.graphicsmagick.org/>にあり、ImageMagick は<https://imagemagick.org/>にあります。

2.2.3 コンフィギュレーション

ほとんどの Linux のインストールと同様に、最初のステップでは、ソースコードのビルドに使われる Makefile を生成します。これは、シェルスクリプトが行います。

./configure

パラメータを付けない場合には、このコマンドは自動で、PostGIS のソースコードのビルドを行うのに必要なコンポーネントやライブラリをシステム上で探します。**./configure** とするのが一般的な使い方ですが、標準的でない位置に必要なライブラリやプログラムを置いてある場合のために、いくつかのパラメータを受け付けます。

次のリストで、共通して使われるパラメータを示します。完全なリストについては、**--help** または **--help=short** パラメータを使って下さい。

- with-library-minor-version** PostGIS 3.0 以降では、デフォルトではライブラリファイルのファイル名にマイナーバージョンが入らなくなりました。PostGIS 3 のライブラリは **postgis-3** で終わります。**pg_upgrade** を簡単にするために実施された変更ですが、サーバに PostGIS 3 シリーズは一つのマイナーバージョンのものだけしかインストールできません。**postgis-3.0** といったようにマイナーバージョンをファイル名に含む古い振る舞いにしたいなら、コンフィギュレーション実行の際に次のスイッチを追加します。
- prefix=PREFIX** PostGIS ライブラリと SQL スクリプトのインストール先を指定します。デフォルトでは、検出された PostgreSQL のインストール先と同じになります。

**Caution**

このパラメータは現在のところ壊れていて、PostgreSQL のインストール先にしかインストールされません。このバグのトラックについては<http://trac.osgeo.org/postgis/ticket/635>をご覧ください。

- with-pgconfig=FILE** PostgreSQL は、PostGIS などの拡張に対して PostgreSQL のインストール先ディレクトリを伝える **pg_config** というユーティリティを持っています。PostGIS の対象とする特定の PostgreSQL のインストール先を手動で指定する場合に、このパラメータ (**--with-pgconfig=/path/to/pg_config**) を使います。
- with-gdalconfig=FILE** 必須ライブラリである GDAL は、ラスタ機能に必要な機能を提供します。GDAL には、インストール先ディレクトリをインストールスクリプトに伝える **gdal-config** があります。PostGIS のビルドに使う特定の GDAL を手動で指定する場合に、このパラメータ (**--with-gdalconfig=/path/to/gdal-config**) を使います。
- with-geosconfig=FILE** 必須のジオメトリライブラリである GEOS には、ソフトウェアのインストール時に GEOS のインストール先ディレクトリを伝える **geos-config** というユーティリティがあります。PostGIS のビルドに使う特定の GEOS を手動で指定する場合に、このパラメータ (**--with-geosconfig=/path/to/geos-config**) を使います。
- with-xml2config=FILE** LibXML は GeomFromKML/GML 処理を行うのに必須のライブラリです。通常は libxml をインストールしているなら発見されますが、発見できない場合や特定の版を使用したい場合は、**xml2-config** を指定してインストールスクリプトに LibXML のインストール先ディレクトリを伝えます。PostGIS のビルドに使う特定の LibXML を手動で指定する場合に、このパラメータ (**--with-xml2config=/path/to/xml2-config**) を使います。
- with-projdir=DIR** Proj は PostGIS に必須の投影変換ライブラリです。PostGIS のビルドに使う特定の Proj のインストールディレクトリを手動で指定する場合は、このパラメータ (**--with-projdir=/path/to/projdir**) を使います。
- with-libiconv=DIR** iconv のインストール先ディレクトリを指定します。
- with-jsondir=DIR** **JSON-C** は、MIT ライセンスの JSON ライブラリで、PostGIS の ST_GeomFromJSON に必須です。PostGIS のビルドに使う特定の JSON-C を手動で指定する場合は、このパラメータ (**--with-jsondir=/path/to/jsondir**) を使います。
- with-gui** データインポート GUI (GTK+2.0 が必要) をコンパイルします。このパラメータによって、**shp2pgsql-gui** という、**shp2pgsql** のグラフィカルユーザインタフェースが作成されます。
- without-raster** ラスタ機能なしでコンパイルします。
- without-topology** トポロジー機能なしでコンパイルします。
- with-gettext=no** デフォルトでは、**gettext** の検出とこれを用いたコンパイルを試みますが、ローダ破損を引き起こす非互換性問題のもとで実行する場合には、このコマンドで無効にできます。これを使ったコンフィギュレーションによって解決する問題の例は<http://trac.osgeo.org/postgis/ticket/748>にあります。ご注意: これを切ることで多くの機能がなくなるわけではありません。まだ文書化されていなくて試験段階である GUI ロードにおける内部のヘルプ/ラベル機能に使われています。
- with-sfcgal=PATH** デフォルトでは、このスイッチなしでは SFCGAL 対応でインストールされません。PATH は、**sfcgal-config** へのパスを指定することができる追加的な引数です。

--without-phony-revision Git レポジトリの現在の HEAD に一致するように、`postgis_revision.h` の更新を無効にします。

Note

PostGIS をコードレポジトリから得る場合には、はじめに次のスクリプトを実行します。

**./autogen.sh**

このスクリプトによって **configure** スクリプトが生成されます。これは PostGIS のインストールに関するカスタマイズに使われます。

PostGIS をアーカイブファイルで入手する場合には、**configure** が既に生成されているので **./autogen.sh** は不要です。

2.2.4 ビルド

Makefile が生成されたら、PostGIS のビルドは、次のコマンドを実行するだけです。

make

出力の最後の行に "PostGIS was built successfully. Ready to install." と出れば終わりです。

全ての関数は文書から生成されたコメントを持っています。後でコメントを手持ちの空間データベースにインストールしたい場合には、**docbook** が必要なコマンドを実行します。`postgis_comments.sql` と他のパッケージのコメントファイルは `tar.gz` ディストリビューションの `doc` フォルダ内に同梱されています。この `tar` ボールからインストールする場合にはコメントを生成する必要がありません。コメントは `CREATE EXTENSION` インストールにも含まれています。

make comments

make cheatsheets ターゲットで、HTML のチートシートが生成されます。クイックリファレンスまたは学習用資料に適しています。これを作るには `xsltproc` が必要です。`doc` フォルダに `topology_cheatsheet.html`、`raster_cheatsheet.html`、`postgis_cheatsheet.html` の 4 ファイルが生成されます。

HTML と PDF のビルド済みのものは [PostGIS / PostgreSQL Study Guides](#) にあります。

make cheatsheets

2.2.5 PostGIS エクステンションのビルドとデプロイ

PostGIS エクステンションは PostgreSQL エクステンション機能が有効なら自動でビルド、インストールされます。

ソースレポジトリからビルドしている場合は、関数の記述を最初にビルドする必要があります。これらは、**docbook** がインストールされている時にビルドされます。手動でインストールするには次のようにします。

make comments

アーカイブファイルからのビルドの場合は、ビルド済みのものがあるので、コメントのビルドは必須ではありません。

PostgreSQL 9.1 を対象にビルドしている場合は、エクステンションは自動的に **make install** 処理の一部としてビルドすべきです。必要なら `extensions` フォルダからビルドできますし、他のサーバで必要ならファイルの複製ができます。

```
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 は、テスト実行のために psql を使用し、psql 環境変数を使用します。一般的な psql 環境変数で上書きすると便利なのが PGUSER,PGPORT, and PGHOST です。環境変数を参照して下さい。

エクステンションファイルは、常に、OS に関係なく同じ版の PostGIS では同じです。PostGIS バイナリを既にインストールしている限りは、エクステンションファイルのある OS から別のものに複写して大丈夫です。

開発用と異なる別のサーバでエクステンションを手動でインストールしたい場合は、サーバにない時に必要となる通常の PostGIS のバイナリだけでなく、次のファイルを extensions フォルダから PostgreSQL インストール先の PostgreSQL / share / extension フォルダに複写します。

- 指定されていない場合のインストールするエクステンションの版等の情報を示す制御ファイル postgis.control, postgis_topology.control。
- エクステンションごとの/sql フォルダにあるファイル全て。extensions/postgis/sql/*.sql, extensions/postgis_topology/sql/*.sql は PostgreSQL share/extension フォルダの最上位に複写する必要があることに注意して下さい。

以上を実行すると、PgAdmin -> extension で postgis, postgis_topology が有効なエクステンションとして見えます。

psql を使う場合は、次のクエリを実行してエクステンションがインストールされていることを確認できます。

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

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

クエリを行ったデータベースにエクステンションがインストールされている場合は、installed_version カラムに記載が見えます。レコードが返ってこない場合は、PostGIS EXTENSION がインストールされていないことになります。PgAdmin III 1.14 以上では、データベースブラウザツリーの extensions セクションで提供されていて、右クリックでアップグレードまたアンインストールできます。

有効なエクステンションがある場合、pgAdmin エクステンションインタフェースまたは次の SQL の実行によって、選択したデータベースに PostGIS エクステンションをインストールできます。

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

psql では、どの版が、どのスキーマにインストールされているかを見ることができます。

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

List of installed extensions

```
-[ RECORD 1 ]-----
Name          | postgis
Version       | 3.7.0alpha1
Schema       | public
```

Description	PostGIS geometry, geography, and raster spat..
-[RECORD 2]	-----
Name	postgis_raster
Version	3.0.0dev
Schema	public
Description	PostGIS raster types and functions
-[RECORD 3]	-----
Name	postgis_topology
Version	3.7.0alpha1
Schema	topology
Description	PostGIS topology spatial types and functions

Warning

追加テーブルの `spatial_ref_sys`、`layer`、`topology` は明示的にはバックアップできません。`postgis` または `postgis_topology` のそれぞれがバックアップされるときに限ってバックアップが行われます。これはデータベース全体をバックアップする時だけ発生します。PostGIS に同梱されていない SRID レコードだけがバックアップに取り込まれます。なので、私たちが提供する SRID エントリーは変更しないでください。また、変更が残るとは期待しないでください。問題を発見した時にはチケットを発行してください。CREATE EXTENSION で作られ、エクステンションの指定されたバージョンと同じものと仮定されるので拡張テーブルの構造はバックアップされません。この挙動は現在の PostgreSQL エクステンションモデルに組み込まれています。

素晴らしいエクステンション機構を使わずに 3.7.0alpha1 をインストールした場合には、それぞれのエクステンションが持つ関数をパッケージするためのコマンドを実行して、エクステンションに基づくように変更できます。PostgreSQL 13 では、パッケージしない方法でのインストールは削除されましたので、PostgreSQL 13 にアップグレードする前にエクステンションをビルドするように変更する必要があります。

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

2.2.6 テスト

PostGIS のテストを行うには、次のコマンドを実行します。

make check

このコマンドで、実際の PostgreSQL データベースに対して生成したライブラリを使用した、様々なチェックとレグレッションテストを行います。

**Note**

PostgreSQL, GEOS または Proj を標準の位置にインストールしていない場合には、環境変数 `LD_LIBRARY_PATH` に、ライブラリの位置を追加する必要があるかも知れません。

**Caution**

現在のところ **make check** は、チェックを行う際に環境変数 `PATH` と `PGPORT` を見ています。コンフィギュレーション時に指定した PostgreSQL とは限りません。つまり **--with-pgconfig** を使って特定したものではありません。PATH を編集して、コンフィギュレーションの際に検出した PostgreSQL と一致するようにして下さい。もしくは、間もなく襲ってくる頭痛に対処する準備をしておいて下さい。

Note

PostgreSQL スーパーユーザーでないサンドボックス化したビルドアカウントは、`POSTGIS_REGRESS_DB_OWNER` を設定することで、レグレッションテストの間ではデータベースの所有を委任することができます。このハークネスは、より特権が少ないアカウントを使った接続を持続している間において、指名されたロールが持つ一時的なレグレッションテストのデータベースを生成します。エクステンション生成者がデータベース所有者と異なるときは `POSTGIS_REGRESS_ROLE_EXT_CREATOR` と併用します。

これらの変数によって、アカウントがスーパーユーザーになるためのプロンプトを使わずに完全なアップグレードとエクステンションのインストールを実行するための自動化環境が可能となります。ただし、対象となる PostgreSQL インスタンスがこの委任ロールによるエクステンションのインストールが許可されている場合に限りです。

成功したなら、`make check` で約 500 個のテストを生成します。結果は次のようなかんじになります (かなりの行を省略しています)。

```
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

```

出力は次のようなかんじになります。

```

===== dropping database "contrib_regression" =====
DROP DATABASE
===== creating database "contrib_regression" =====
CREATE DATABASE
ALTER DATABASE
===== installing fuzzystmatch =====
CREATE EXTENSION
===== installing postgis =====
CREATE EXTENSION

```

```
===== running regression test queries =====
test test-normalize_address    ... ok

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

2.2.7 インストール

PostGIS をインストールするには、次のコマンドを実行します。

make install

これにより、PostGIS のインストールファイルが、**--prefix** パラメータで指定した、適切なサブディレクトリに複写されます。次に特筆すべきサブディレクトリを示します。

- ローダとダンパのバイナリのインストール先は[**prefix**]/bin です。
- **postgis.sql** などの SQL ファイルのインストール先は[**prefix**]/share/contrib です。
- PostGIS ライブラリのインストール先は[**prefix**]/lib です。

先に **make comments** を実行して**postgis_comments.sql**, **raster_comments.sql** を生成していた場合は、次のコマンドを実行すると、これらの SQL ファイルがインストールされます。

make comments-install



Note

postgis_comments.sql, **raster_comments.sql**, **topology_comments.sql** は、**xsltproc** の外部依存ができたので、通常のビルドとインストールから切り離されました。

2.3 共通の問題

インストールやアップグレードが思うようにいかない時にチェックすることがいくつかあります。

1. PostgreSQL 14 以上をインストールしているか、実行中の PostgreSQL と同じ版のソースでコンパイルしているか、をチェックします。(Linux の) ディストリビューションによって既に PostgreSQL がインストールされている時や、PostgreSQL を以前にインストールして忘れた場合に、混乱が発生することがあります。PostGIS は PostgreSQL 14 以上で動作します。古い版のものを使った場合には、おかしな予想外のエラーメッセージが表示されます。実行中の PostgreSQL の版をチェックするには、**psql** を使ってデータベースを接続して、次のクエリを実行して下さい。

```
SELECT version();
```

RPM ベースのディストリビューションを実行している場合、プリインストールされたパッケージが存在するかのチェックは、**rpm** コマンドを使って **rpm -qa | grep postgresql** でチェックできます。

2. アップグレードに失敗する場合、既に PostGIS がインストールされているデータベースにリストアしているか確認して下さい。

```
SELECT PostGIS_Full_Version();
```

また、コンフィギュアが正しく PostgreSQL、Proj4 ライブラリ、GEOS ライブラリのインストール先を検出したかチェックして下さい。

1. コンフィギュアからの出力で**postgis_config.h** ファイルが作られます。POSTGIS_PGSQL_VERSION、POSTGIS_PROJ_VERSION および POSTGIS_GEOS_VERSION 変数が正しくセットされたかをチェックして下さい。

Chapter 3

PostGIS 管理

3.1 パフォーマンスチューニング

PostGIS の性能向上のための調整は PostgreSQL の作業量の調整と非常に似ています。追加して留意すべき点は、ジオメトリとラスタは重大きいのでメモリ関連の最適化は他の PostgreSQL クエリと比べて影響が大きいことだけです。

PostgreSQL の最適化に関する一般的な詳細は、[Tuning your PostgreSQL Server](#)をご覧ください。

`postgresql.conf` または `postgresql.auto.conf` を触らなくても、`ALTER SYSTEM` コマンドでサーバーレベルのコンフィギュレーションが可能です。

```
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;
```

PostgreSQL の設定に加えて、PostGIS には [Section 7.22](#)で挙げる独自設定があります。

3.1.1 起動時

次に示す設定は `postgresql.conf` にあります:

`constraint_exclusion`

- デフォルト: `partition`
- これは一般的にテーブルのパーティショニングに使われます。デフォルトの“`partition`”では、テーブルが継承階層の中にある場合にはプランナーに制約を考慮するテーブル解析だけを強制し、他の場合にはプランナーペナルティを払うのを避けるようにします。

`shared_buffers`

- デフォルト: ~128MB
- 利用可能な RAM の 25% から 40% を設定します。Windows では高く設定することができないかも知れません。

`max_worker_processes`で有効なバックグラウンドプロセス数の最大値をサーバーに対して制御できます。間接的にパラレルクエリーに参加するワーカーの上限値も設定できると言えます。

- デフォルト: 8
- システムが対応できるバックグラウンドプロセスの最大値を設定します。このパラメータはサーバ起動時のみ設定できます。

3.1.2 実行時

work_mem - 並べ替えや複雑なクエリに使われるメモリのサイズの設定

- デフォルト: 1-4MB
- 大きなデータベースの場合や、複雑なクエリの場合、RAM が多い場合は値を大きくするように調整します。
- 同時接続ユーザ数が多い場合や、RAM が少ない場合には値を小さくするように調整します。
- たくさんの RAM を持ち、少数の開発者しかいない場合は次のようにします:

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

maintenance_work_mem (訳注: 日本語訳) - VACUUM, CREATE INDEX 等で使われるメモリのサイズ

- デフォルト: 16-64MB
- 一般的には低すぎます - メモリスワップの間、入出力が拘束され、オブジェクトがロックされます
- たくさんの RAM を持つ本番サーバでは 32MB から 1GB が推奨ですが、同時接続ユーザ数に依存します。たくさんの RAM を持ち、少数の開発者しかいない場合は次のようにします:

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

max_parallel_workers_per_gather

最近の PostgreSQL と PostGIS のパラレルクエリー機能では、1 回のプランで要求する可能性のあるワーカー数を決定するためにこの設定が使われます。0 より大きい数を設定した場合には、これらの関係する **ST_Intersects** のような関係関数を含むクエリーが複数のプロセスを使うことができ、そうすると 2 倍を超える速度になります。多数の予備プロセスがあるなら、この値を持っているプロセス数まで上げるべきです。ただし、これにあわせて **max_worker_processes** を少なくともこの数と同じまで増やすのを忘れないでください。

- デフォルト: 0
- 単一の Gather ノードが開始できるワーカーの最大数を設定します。並列ワーカーは、**max_worker_processes** で確立されたプロセスのプールから取得されます。要求したワーカー数は、実際には実行可能になっていない場合があることに注意して下さい。これが発生する場合には、想定より少ないワーカーでプランが実行されることになり、場合によっては非効率になります。この値を 0 (デフォルト値) にすると、パラレルクエリー実行が無効になります。

3.2 ラスタ機能の設定

ラスタ機能を有効にしたら、下に示す確実な設定方法を読んだ方がいいです。

データベース外ラスタとすべてのラスタードライバーはデフォルトでは無効です。有効にするには、サーバー側で環境変数 **POSTGIS_GDAL_ENABLED_DRIVERS** と **POSTGIS_ENABLE_OUTDB_RASTERS** とを設定する必要があります。よりクロスプラットフォームな方法として、対応する [Section 7.22](#)を設定することができます。

データベース外ラスタを有効にするには次のようにします:

```
POSTGIS_ENABLE_OUTDB_RASTERS=1
```


他の値を入れたり、値を入れない場合には、データベース外ラスタは無効になります。

インストールした GDAL で利用可能なドライバーの全てを有効にするには、次の環境変数を設定します:

```
POSTGIS_GDAL_ENABLED_DRIVERS=ENABLE_ALL
```

一部のドライバーのみ有効にしたい場合には、環境変数を次のように設定します:

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

**Note**

Windows 環境の場合には、ドライバーリストに引用符をつけないで下さい

環境変数は OS によって変化します。Ubuntu や Debian 上で apt-postgresql を介して PostgreSQL をインストールした場合には、`/etc/postgresql/MAJOR/CLUSTER/environment` (MAJOR はメジャーバージョン、CLUSTER はクラスター名が入ります) を編集する方が良いです。

Windows でサービスとして実行している場合には、システム変数で設定します。Windows 7 では、コンピュータを右クリックしてプロパティをクリックするか、エクスプローラの検索バーにコントロールパネル\すべてのコントロールパネル項目\システムを指定します。それから、システムの詳細設定 -> 詳細設定 -> 環境変数 を順にクリックして、新しいシステム環境変数を追加します。

環境変数を設定した後は、設定を反映させるために、PostgreSQL サービスの再起動が必要です。

3.3 空間データベースの作成

3.3.1 エクステンションを使って空間データベースを有効にする

`extensions/postgis` モジュールをコンパイル、インストール済みの場合には、EXTENSION 機構を使うことでデータベースを空間データベースに切り替えることができます。

中核となる PostGIS エクステンションには、ジオメトリ、ジオグラフィ、`spatial_ref_sys` および全ての関数とコメントが含まれています。ラスタとトポロジは別のエクステンションになっています。

空間データベースにしたいデータベース上で次の SQL を実行します:

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

3.3.2 エクステンションを使わずに空間データベースを有効にする (お勧めしません)

**Note**

これは、通常は PostgreSQL のエクステンションのディレクトリ内に PostGIS をインストールできないか、したくない場合 (たとえばテスト中や開発中、または制限のある環境内) にのみ必要となります。

ビルドの際に指定した `[prefix]/share/contrib` 内にある様々な SQL ファイルをロードして PostGIS オブジェクトと関数の定義をデータベースに追加します。

中核の PostGIS オブジェクト (ジオメトリ型とジオグラフィ型、これらに対応する関数) は `postgis.sql` スクリプトにあります。ラスタオブジェクトは `rtpostgis.sql` スクリプトにあります。トポロジオブジェクトは `topology.sql` スクリプトにあります。

EPSG 座標系定義 ID の完全なテーブルを得るには、`spatial_ref_sys.sql` 定義ファイルをロードして `spatial_ref_sys` テーブルを生成して下さい。これによりジオメトリ関数 `ST_Transform()` が実行できるようになります。

PostGIS 関数にコメントを追加したい場合には、`postgis_comments.sql` スクリプト内にコメントがあるのが分かります。コメントは **psql** のターミナルウィンドウから単に `\dd [関数名]` と打ち込むだけで見ることができます。

ターミナルで次のシェルコマンドを実行します:

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

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

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

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

3.4 空間データベースのアップグレード

既存の空間データベースのアップグレードは、新しい PostGIS オブジェクト定義の置き換えや導入を必要とするとき、慎重を要することがあります。

不幸なことに、定義の全てが実行中のデータベース内で簡単に置き換えられるわけではないので、ダンプ/リロードが最善策となることがあります。

PostGIS には、マイナーバージョンアップやバグフィクスリリースの場合に使うソフトアップグレードと、メジャーアップグレードで使うハードアップグレードが用意されています。

PostGIS をアップグレードしようとする前にデータのバックアップを取ることは、常に価値のあるものです。`pg_dump` で `-Fc` フラグを使うと、ハードアップグレードでダンプを常にリストアすることができます。

3.4.1 ソフトアップグレード

エクステンションを使ってデータベースをインストールした場合には、エクステンションモデルでアップグレードしなければなりません。古い SQL スクリプトを使ってインストールした場合には、SQL スクリプトは既にサポートされていないので、エクステンションに切り替えるべきです。

3.4.1.1 エクステンションを使ったソフトアップグレード

PostGIS をエクステンションとして入れている場合には、アップグレード m のエクステンションを使う必要があります。エクステンションでのマイナーアップグレードではほとんど痛みは伴いません。

インストール済みのバージョンのうち最新バージョンにアップグレードするには **PostGIS_Extensions_Upgrade** 関数を使います。

```
SELECT postgis_extensions_upgrade();
```

PostGIS 2.5 をまだ使っているデータベースをアップグレードするときには、**ALTER EXTENSION** を `postgis_extensions_upgrade()` に先立って使います。ラスターエクステンションが独立する前の `postgis_raster` オブジェクトを確実に主エクステンションに折り込むようにするためです。

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

複数のバージョンの PostGIS がインストールされていて、最新版にはアップグレードしたくない場合には、明示的なバージョンの指定ができます。次のようにします:

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

次のようなエラー通知が表示されることがあります:

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

この場合には、データベースをバックアップして、Section 3.3.1 に記述されているように新しいデータベースを生成し、バックアップを新しいデータベースにリストアしなければなりません。

次のようなメッセージを得ることがあります:

```
Version "3.7.0alpha1" of extension "postgis" is already installed
```

この場合には、全てアップデートされていて、安全に無視できます。開発版からその次のバージョン (新しいバージョン番号を得ていないもの) にアップグレードしようとしないう限り、`"next"` をバージョン文字列に追加できますが、次回に `"next"` を削除する必要があります:

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

**Note**

PostGIS をバージョン指定なしにインストールした場合には、しばしばリストア前の PostGIS エクステンションの再インストールをとばすことができます。バックアップには `CREATE EXTENSION postgis` があって、リストアの際に最新版になります。

**Note**

PostGIS エクステンションを 3.0.0 より前からアップグレードする場合には、ラスター機能が不要なら、安全に削除できる新しい `postgis_raster` エクステンションを持つことになります。削除は次のようにします:

```
DROP EXTENSION postgis_raster;
```

3.4.1.2 エクステンションを使わないソフトアップグレード

エクステンションはこれまでの多数のリリースに対応したインストール手法です。今なおパッケージ化されていないものをインストールしているのなら、エクステンションで作られたデータベース内で **HARD UPGRADE** を実行して、保持データをそちらに移行させて下さい。

3.4.1.3 ハードアップグレード

ハードアップグレードとは、PostGIS で利用可能なデータベースの完全なダンプ/リロードを意味します。PostGIS オブジェクトの内部格納状態が変更される場合や、ソフトウェアアップグレードができない場合に、ハードアップグレードが必要です。付録の [リリースノート](#) に、バージョンごとのダンプ/リロード (ハードアップグレード) の要否を記載しています。

ダンプ/リロード作業は `postgis_restore` スクリプトが補助します。このスクリプトを使うと、PostGIS (古いものを含む) に属する定義を全て飛ばしてダンプします。また、重複シンボルエラーや非推奨オブジェクトを持越すことなく、スキーマとデータを PostGIS をインストールしたデータベースにリストアできます。

Windows 用に関する追加情報は [Windows Hard upgrade](#) にあります。

手続きは次の通りです:

1. アップグレードしたデータベース (`olddb` と呼ぶことにしましょう) の「カスタム書式」のダンプを、バイナリ BLOB データを含めたダンプを指定して (`-b`)、`verbose` モード (`-v`) で生成します。ユーザはデータベースのオーナーになることができ、PostgreSQL のスーパーユーザである必要はありません。

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

2. 新しいデータベースに PostGIS を、PostGIS が無い状態からインストールします。このデータベースを `newdb` と呼ぶことにします。この作業に関する説明については [Section 3.3.2](#) と [Section 3.3.1](#) を参照して下さい。

ダンプにある `spatial_ref_sys` のエントリはリストアされますが、既にある `spatial_ref_sys` のエントリを上書きしません。リストア対象のデータベースに公式データセットの訂正が確実に伝わるようにするためです。なんらかの理由で標準のエントリを上書きしたい場合には、`newdb` を生成する際に `spatial_ref_sys.sql` ファイルをロードしないだけです。

データベースが本当に古くて、ビューや関数の中に、非推奨になってから長い関数があるような場合には、それらの関数やビューを使えるようにする `legacy.sql` をロードする必要があるでしょう。ただし「本当に」必要な場合に限りです。可能なら、ビューや関数をダンプせずにアップグレードすることを検討して下さい。非推奨関数は、`uninstall_legacy.sql` で後から削除することができます。

3. バックアップを新しい `newdb` データベースにリストアするには、`postgis_restore` を使います。psql が予期せぬエラーを標準エラー出力に出すことがあります。このエラーのログは保存しておいて下さい。

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

エラーは次の場合に起こります:

1. ビューまたは関数の中に非推奨の PostGIS オブジェクトを使っているものがある場合。これの訂正には、リストア前に `legacy.sql` スクリプトのロードを試してみることができます。非推奨オブジェクトをまだ持っているバージョンの PostGIS に戻して、コードを作り替えた後に再移行を試みることもできます。`legacy.sql` を利用する場合は、コードを非推奨関数を使わないように訂正し、また `uninstall_legacy.sql` をロードして非推奨オブジェクトを削除するのを忘れないでください。
2. ダンプファイル内の `spatial_ref_sys` にあるカスタムレコードが不正な SRID になっている場合。妥当な SRID 値は 0 より大きく 999000 より小さくなります。999000 から 999999 の間は内部利用のための予約領域ですが、SRID 値 > 999999 の場合は、その値は一切使用できません。全ての不正な SRID を持つ独自レコードは、SRID 値 > 999999 の場合には、予約領域に入るよう SRID 値が変更されたうえで保持されます。しかし、値を保持するために設定されている `spatial_ref_sys` テーブルのチェック制約が外れます。場合によっては (複数の不正な SRID が同じ予約領域の SRID 値に変換されるとき)、主キーも外れます。これを修正するために、独自の SRS を妥当な値 (910000..910999 の範囲) の SRID に複写し、全てのテーブルを新しい SRID に変更 ([UpdateGeometrySRID](#)) して、`spatial_ref_sys` から不正なエントリを削除します。そして、次のようにチェック制約を再構築します:

```
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));
```

フランスIGN地図を含む古いデータベースをアップグレードする場合には、おそらく SRID が範囲外になり、データベースのインポート時に次のような問題に遭遇します:

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

この場合には、次のステップを試すことができます。最初に `postgis_restore` から出た IGN を SQL から完全に破棄します。そのために次のコマンドを実行します:

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

次のコマンドを実行します:

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

その後、新しいデータベースを生成し、必要な PostGIS エクステンションを有効化して、[このスクリプト](#)で確実にフランス IGN の系を挿入します。これらの処理の後、次のようにデータのインポートを行います:

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

Chapter 4

データ管理

4.1 空間データモデル

4.1.1 OGC ジオメトリ

Open Geospatial Consortium (OGC) は、地理空間データのモデルを提供するために *Simple Features Access* (SFA) 標準を開発しました。これは、ジオメトリ (**Geometry**) の基本的な空間タイプを、空間解析処理実行のための操作や変換といった演算に沿って定義します。PostGIS は空間解析タスクを実現するために OGC ジオメトリモデル値を PostgreSQL データ型の **geometry**、**geography** として実装しています。

ジオメトリは抽象的なタイプです。ジオメトリ値は複数ある具体的なサブタイプの一つに属します。サブタイプは様々な種類の様々な次元のジオメトリの形状を表現するものです。これらには原子的なタイプである **ポイント (Point)**、**ラインストリング (LineString)**、**リニアリング (LinearRing)**、**ポリゴン (Polygon)** があります。また、コレクション (**collection**) (訳注: 「マルチ系」と書いている場合があります) タイプの **マルチポイント (MultiPoint)**、**マルチラインストリング (MultiLineString)**、**マルチポリゴン (MultiPolygon)**、**ジオメトリコレクション (GeometryCollection)** があります。 *Simple Features Access - Part 1: Common architecture v1.2.1* では **多面体サーフェス (PolyhedralSurface)**、**三角形 (Triangle)**、**TIN** が追加されています。

ジオメトリは 2 次元デカルト平面上の形状をモデル化しています。多面体サーフェス、三角形、TIN は 3 次元空間内の形状を表現することもできます。形状のサイズと位置は座標によって指定されます。それぞれの座標は、平面上で位置を判定する X と Y の座標軸値を持っています。形状はポイントと線分から構築され、ポイントは単一の座標で定められ、線分は二つの座標値から定められます。

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

ジオメトリ値は、そのジオメトリが組み込まれている座標系を示す空間参照系に関連付けられます。空間参照系はジオメトリの SRID 番号で識別されます。X 軸と Y 軸の単位は空間参照系によって決まります。平面参照系では伝統的に X 座標値と Y 座標値が東、北をそれぞれ示します。地理参照系では、経度と緯度を表現しています。SRID が 0 の場合には、軸の単位が無い、無限の直交平面を表します。Section 4.5 を参照して下さい。

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

ジオメトリ値は **empty** になることがあります。空値とは、非マルチ系ジオメトリの場合は頂点を持っていないという意味で、コレクションでは要素を持っていないという意味です。

ジオメトリ値の重要なプロパティは範囲 (**extent**) またはバウンディングボックス (**bounding box**) です。OGC モデルではエンベロープ (**envelope**) と呼ばれています。これは、ジオメトリの座標を囲む 2 次元または 3 次元

のボックスです。ジオメトリの座標空間内の範囲を表現するための、また、二つのジオメトリの相互関係をチェックするための、効率の良い方法です。

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

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

4.1.1.1 ポイント (Point)

ポイントは、座標空間内の一つの位置を表現する 0 次元ジオメトリです。

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

4.1.1.2 ラインストリング (LineString)

ラインストリングは連続する一連の線分で形成される 1 次元のラインです。線分はそれぞれ 2 点で定義付けられ、ある線分の終点は次の線分の始点を形成します。OGC 妥当なラインストリングには、0 または 2 以上のポイントがあります。ただし PostGIS はラインストリングの一つのポイントを許容します。ラインストリングは、自身とクロスする場合があります (自己交差)。始端と終端とが同じ場合にはラインストリングは閉じたことになります。自己交差しない場合には、ラインストリングは単純です。

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

4.1.1.3 リニアリング (LinearRing)

リニアリングは閉じていて、かつ単純なラインスリングです。始端と終端は同じでなければなりませんし、ラインは自己交差してはなりません。

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

4.1.1.4 ポリゴン (Polygon)

ポリゴンは 2 次元平面領域です。一つの外側の境界 (殻) と 0 個以上の内の境界 (穴) とで区切られています。それぞれの境界は **リニアリング** です。

```
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 マルチポイント (MultiPoint)

マルチポイントはポイントのコレクションです。

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

4.1.1.6 マルチラインSTRING (MultiLineString)

マルチラインSTRINGはラインSTRINGのコレクションです。各要素が閉じている場合には、そのマルチラインSTRINGは閉じています。

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

4.1.1.7 マルチポリゴン (MultiPolygon)

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

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

4.1.1.8 ジオメトリコレクション (GeometryCollection)

ジオメトリコレクションは、ジオメトリの異種 (混合) のコレクションです。

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

4.1.1.9 多面体サーフェス (PolyhedralSurface)

多角形はサーフェスは、パッチまたはエッジを共有する面の隣接するコレクションです。それぞれのパッチは平面ポリゴンです。ポリゴンが Z 値を持つ場合には、サーフェスは 3 次元になります。

```
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)

三角形は三つの異なる非共線頂点で定義されるポリゴンです。三角形はポリゴンですので、四つの座標で指定され、一つ目と四つ目は同じです。

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

4.1.1.11 TIN

TIN は **Triangulated Irregular Network** を表現する、オーバーラップしない **三角形** のコレクションです。

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


4.1.2 SQL/MM Part 3 - 曲線

ISO/IEC 13249-3 SQL Multimedia - Spatial 標準 (SQL/MM) は、OGC SFA を拡張して、曲線ジオメトリを含むサブタイプを定義しています。SQL/MM タイプは XYM, XYZ, XYZM に対応します。



Note

SQL-MM 実装での全ての浮動小数点数の比較では、所定の丸め誤差があります。現在は 1E-8 です。

4.1.2.1 曲線ストリング (CircularStringCircularString)

曲線ストリングは、基本的な曲線タイプです。線形の世界のラインストリングに似ています。単一の円弧線分は、始点、終点 (1 番目と 3 番目)、弧の他の点の三つの点で定義されます。閉じた円を指定するには、開始点と終了点を同じにし、中間点を対称点 (円弧の中心) に置きます。連続する円弧では、前の円弧の終端と次の円弧の始端とが同じです。よって曲線ストリングは 1 以上の奇数個のポイントを持つことになります。

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

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

4.1.2.2 複合曲線 (CompoundCurve)

複合曲線は、曲線区間と直線区間の両方を含むことができる単一の連続した曲線です。このことは、整形された要素を持つことに加えて、全ての要素の最後のポイントは次の要素の最初のポイントでなければならないことを意味します。

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

4.1.2.3 曲線ポリゴン (CurvePolygon)

曲線ポリゴンは、外側の輪がひとつで 0 以上の内側のリングがある点はポリゴンに似ています。違いは、ポリゴンのリングはラインストリングですが曲線ポリゴンのリングは曲線ストリングまたは複合ストリングである点です。

PostGIS 1.4 から、PostGIS で曲線ポリゴンで複合曲線をサポートするようになりました。

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

例: CIRCULARSTRING と LINESTRING からなる COMPOUNDCURVE で定義される外殻を持ち、CIRCULARSTRING で定義される穴を持つ CURVEPOLYGON

```
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 マルチ曲線 (Multicurve)

マルチ曲線は曲線のコレクションで、ラインストリング、曲線ストリング、複合曲線を含むことができます。

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

4.1.2.5 マルチサーフェス (MultiSurface)

マルチサーフェスはサーフェスのコレクションです。サーフェスは (線形) ポリゴンまたは曲線ポリゴンとすることができます。

```
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 と WKB

OGC SFA 仕様では、ジオメトリ値を外部で使用するための表現として二つの標準書式が定義されています。Well-Known Text (WKT) と Well-Known Binary (WKB) です。WKT と WKB は両方ともそのオブジェクトを定義するタイプと座標に関する情報を含んでいます。

Well-Known Text (WKT) で空間データの標準的な文字表現が可能です。空間オブジェクトの WKT 表現の例を次に挙げます。

- 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 の入出力は関数 **ST_AsText** と **ST_GeomFromText** によって提供されます。

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

例えば、WKT と SRID からの空間オブジェクトの生成と挿入のステートメントは次の通りです。

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

Well-Known Binary (WKB) は、空間データのバイナリデータ (バイト列) で、移植可能かつ正確な表現です。空間オブジェクトの WKB 表現を次に挙げます。

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

WKB の入出力は関数 **ST_AsBinary** と **ST_GeomFromWKB** が提供されています。次のように使います。

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

たとえば、WKB から空間オブジェクトの生成、挿入は次のようにします。

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

4.2 ジオメトリデータタイプ

PostGIS は、**geometry** という PostgreSQL データ型を定義して、OGC Simple Features model を実装しています。これで、内部タイプコード (**GeometryType** と **ST_GeometryType** 参照) で全てのジオメトリのサブタイプを表現します。これにより、**geometry** 型のカラムを持つよう定義されたテーブルの行として、空間地物をモデリングすることが可能となります。

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



Note

PostGIS は、空間関数にプリフィクス“ST_”を付けて、SFA 標準に従っています。これは、“Spatial and Temporal (空間と時間)”を示していますが、標準の時間の部分はまだ開発していません。その代わりに“Spatial Type (空間タイプ)”と解釈できます。

SFA 標準は、空間オブジェクトは空間参照系識別子 (SRID) を含むと規程しています。SRID は、空間オブジェクトをデータベースに挿入するために生成した時に求められます (デフォルトとして 0 になるかも知れません)。**ST_SRID** と Section 4.5 をご覧下さい。

ジオメトリのクエリを効率的にするため、PostGIS では様々な種類の空間インデックスを定義しています。詳細については Section 4.9 と Section 5.2 をご覧下さい。

4.2.1 PostGIS EWKB と EWKT

OGC SFA 仕様は、まず 2 次元ジオメトリのみに対応しました。また、入出力表現にジオメトリの SRID は取り入れていません。OGC SFA 仕様 1.2.1 (ISO 19125 標準に準拠) では 3 次元 (XYZ) と M 値 (XYM と XYZM) 座標に対応するようになりましたが、SRID 値の取り込みは依然行われていません。

これらの制限のため、PostGIS では拡張書式である EWKB と EWKT を定義しました。3 次元 (XYZ, XYM) と 4 次元 (XYZN) 座標系に対応し、SRID 情報を取り込めるようにしました。すべてのジオメトリ情報を含めたので、PostGIS は EWKB を格納用書式 (DUMP ファイル等) として使えるようになりました。

PostGIS データオブジェクトの「カノニカルな形式」のために EWKB と EWKT を使います。入力では、バイナリデータのカノニカルな形式は EWKB、テキストデータについては EWKB か EWKT が受け付けられます。これにより、HEXEWKB または EWKT のテキスト値から **::geometry** を使用してキャストを行い、ジオメトリ値

が生成できるようになりました。出力では、バイナリのカノニカルな形式は EWKB で、テキストは HEXEWKB (HEX エンコードを施した EWKB) です。

たとえば、この手続きでは、EWKT テキスト値からのキャストでジオメトリを生成して、HEXWKB のカノニカルな形式を使った出力を行います。

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

PostGIS EWKT 出力は OGC WKT と次の通り相違点があります。

- XYZ ジオメトリで Z 修飾子が省略されます。
OGC: POINT Z (1 2 3)
EWKT: POINT (1 2 3)
- M 値を含む XYM ジオメトリ:
OGC: POINT M (1 2 3)
EWKT: POINTM (1 2 3)
- 4 次元ジオメトリで ZM 修飾子を省略:
OGC: POINT ZM (1 2 3 4)
EWKT: POINT (1 2 3 4)

EWKT は、次のように OGC/ISO 書式で発生しうる過剰次元と不整合を回避しています。

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



Caution

PostGIS の拡張書式は OGC 書式の上位互換であり、全ての妥当な OGC WKB/WKT は妥当な EWKB/EWKT でもあります。しかし、OGC が PostGIS の定義と衝突する方法で書式を拡張した場合には、将来的に書式を変更する可能性があります。ゆえに、この互換性に * 頼るべきではありません *!

空間オブジェクトの EWKT テキスト表現の例:

- POINT(0 0 0) -- XYZ
- SRID=32632;POINT(0 0) -- SRID 付き XY
- POINTM(0 0 0) -- XYM
- POINT(0 0 0 0) -- XYZM
- SRID=4326;MULTIPOINTM(0 0 0,1 2 1) -- SRID 付き XYM
- MULTILINESTRING((0 0 0,1 1 0,1 2 1),(2 3 1,3 2 1,5 4 1))
- POLYGON((0 0 0,4 0 0,4 4 0,0 4 0,0 0 0),(1 1 0,2 1 0,2 2 0,1 2 0,1 1 0))
- MULTIPOLYGON((((0 0 0,4 0 0,4 4 0,0 4 0,0 0 0),(1 1 0,2 1 0,2 2 0,1 2 0,1 1 0)),((-1 -1 0,-1 -2 0,-2 -2 0,-2 -1 0,-1 -1 0)))
- GEOMETRYCOLLECTIONM(POINTM(2 3 9), LINESTRINGM(2 3 4, 3 4 5))

- MULTICURVE((0 0, 5 5), CIRCULARSTRING(4 0, 4 4, 8 4))
- POLYHEDRALSURFACE(((0 0 0, 0 0 1, 0 1 1, 0 1 0, 0 0 0)), ((0 0 0, 0 1 0, 1 1 0, 1 0 0, 0 0 0)), ((0 0 0, 1 0 0, 1 0 1, 0 0 1, 0 0 0)), ((1 1 0, 1 1 1, 1 0 1, 1 0 0, 1 1 0)), ((0 1 0, 0 1 1, 1 1 1, 1 1 0, 0 1 0)), ((0 0 1, 1 0 1, 1 1 1, 0 1 1, 0 0 1)))
- TRIANGLE ((0 0, 0 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)))

これらの書式を使う入出力は次の関数を使うと有効です。

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

たとえば、EWKT を使って PostGIS の空間オブジェクトを作成し挿入するステートメントは次の通りです。

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

4.3 ジオグラフィデータタイプ

geography データタイプによって地理座標 ("geographic", "geodetic", "lat/lon", "lon/lat" など) 上の空間地物表現にネイティブに対応できます。地理座標系は角度 (度) 単位で表現される球面座標系です。

PostGIS ジオメトリ型の基礎は平面です。平面上の 2 点間の最短経路は直線です。ジオメトリに関する関数 (面積、距離、長さ、インタセクション等) は直線ベクトルとデカルト平面を使って計算しています。これで実装が簡単になり実行速度も上がりますが、地球の球面の上にあるデータについては不正確になります。

PostGIS ジオグラフィというデータ型は球面モデルに基づいています。球面上の 2 点の最短経路は大円の弧にあたります。ジオグラフィの関数 (面積、距離、長さ、インタセクション等) は球面上の弧を使います。球面上の世界の形状を考慮に入れるので、より正確な結果が得られます。

基礎となる数学はより複雑になるため、ジオグラフィ型で定義された関数はジオメトリ型で定義された関数よりも少なくなります。時間が経つにつれて新しいアルゴリズムが追加されて、ジオグラフィの機能が拡大していきます。回避策として、ジオメトリ型とジオグラフィ型との相互変換が可能です。

ジオメトリ型のように、ジオグラフィデータは空間参照系識別子 (Spatial Reference System Identifier, SRID) を持っています。spatial_ref_sys で定義されている地理座標系 (経度/緯度) の空間参照系で利用できます。独自の地理座標系の空間参照系を追加することもできます。Section 4.5.2 を参照してください。

計測関数 (例 [ST Distance](#), [ST Length](#), [ST Perimeter](#), [ST Area](#)) によって返されるものの単位と、[ST_DWithin](#) の引数で与えられる距離との、空間参照系の単位は、メートルです。

4.3.1 ジオグラフィテーブルの生成

ジオグラフィデータを格納するテーブルは、SQL ステートメント [CREATE TABLE](#) に geography 型のカラムを付けることで生成することができます。2 次元ラインストリングを WGS84 地理座標系 (SRID 4326) で保存するジオグラフィカラムを持つテーブルを生成する例を次に示します。

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

二つの任意の型修飾子に対応するジオグラフィ型:

- 空間の型修飾子は、カラム内で許される形状の種類や次元を規制します。値によって空間型は POINT、LINESTRING、POLYGON、MULTIPOINT、MULTILINESTRING、MULTIPOLYGON、GEOMETRYCOLLECTION が可能です。ジオグラフィ型は曲線や三角形、多面体サーフェスに対応していません。型修飾子に後置詞 Z、M、ZM を付けることで、座標次元の制約に対応しています。たとえば、'LINESTRINGM' は、3 次元で 3 番目の軸は M であるラインストリングのみを許します。同様に 'POINTZM' では 4 次元 (XYZM) データが求められます。
- SRID 修飾子は空間参照系 (SRID) を特定の数値になるよう制約します。省略した場合には、デフォルトは 4326 (WGS84 地理座標系) となり、全ての計算は WGS84 を使ったものになります。

ジオグラフィカラムを持つテーブルの生成の例を次に挙げます。

- SRID がデフォルトの 4326 (WGS84 経度/緯度) である 2 次元ポイントジオグラフィを持つテーブルの生成:

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

- NAD83 緯度/経度の 2 次元ポイントジオグラフィを持つテーブルの生成:

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

- SRID を 4326 で明示した 3 次元 (XYZ) ポイントジオグラフィを持つテーブルの生成:

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

- SRID がデフォルトの 4326 である 2 次元ラインストリングジオグラフィを持つテーブルの生成:

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

- SRID がデ 4326 (NAD 1927 経度/緯度) である 2 次元ポリゴンジオグラフィを持つテーブルの生成:

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

ジオグラフィカラムは `geography_columns` システムビューに登録されます。`geography_columns` ビューにクエリを出してテーブルを見るには、次の通りにします。

```
SELECT * FROM geography_columns;
```

空間インデックスはジオメトリカラムと同じように機能します。PostGIS は、カラム型がジオグラフィであると通知したうえで、ジオメトリに使う通常の平面用インデックスでなく、球面を基にした適切なインデックスを生成します。

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

4.3.2 ジオグラフィテーブルの使用

ジオメトリと同じ方法でジオグラフィテーブルにデータを挿入できます。ジオメトリデータは、SRID 4326 の場合には、ジオグラフィ型に自動キャストされます。**EWKT** と **EWKB** 書式はジオグラフィ値を指定するために使うことができます。

```
-- 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` テーブルにある地理 (経度/緯度) 参照系は、ジオグラフィの SRID として指定することができます。非地理座標系を使うとエラーが発生します。

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

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

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

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

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

クエリと計測関数はメートル単位となります。そのため距離パラメータはメートル (面積の場合は平方メートル) 単位となります。

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

シアトルからロンドンへの (LINESTRING(-122.33 47.606, 0.0 51.5)) 大円航路を行く航空機がレイキャビク (POINT(-21.96 64.15)) にどれだけ近づくかを計算することで、ジオグラフィの力を見ることができます ([航路の地図表示](#))。

ジオグラフィ型は、レイキャビクとシアトルーロンドン間の大円航路との距離について、球面上で 122.235 km という本当の最短距離を計算します。

```
-- 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
```

ジオメトリ型では、平面の世界地図上で見て、レイキャビクとシアトルーロンドン間の直線とのデカルト距離が計算され、意味がありません。計算結果の名目上の単位は「度」ですが、点間の本当の角度差に応じるものではなく、「度」と呼ぶこと自体が不正確です。

```
-- 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 ジオグラフィ型を使用すべき時

ジオグラフィ型によって、経度緯度座標でデータを格納できるようになりましたが、ジオグラフィで定義されている関数が、ジオメトリより少ないのと、実行に CPU 時間がかかる、というところが犠牲になっています。

選択した型が、期待する領域から出ないことを、ジオメトリ型にして使用する条件とすべきです。使用するデータは地球全体か、大陸か、州か、自治体か？

- データが小さいエリア内におさまるなら、適切な投影を選択してジオメトリを使うのが、効率面でも機能面でも最も良い方法です。
- データが地球全体か大陸なら、ジオグラフィで投影法の細かい問題を気にせずにシステムを構築できるでしょう。経度/緯度のデータを保存して、ジオグラフィで定義された関数使います。
- 投影法を理解していなくて、学習したくもなくて、かつ、ジオグラフィで使える関数が限られていることを受け入れるのなら、ジオグラフィを使った方が簡単です。単純にデータを経度/緯度でロードして、そこから進めて下さい。

ジオグラフィとジオメトリ間のサポート状況の比較については [Section 13.11](#) をご覧下さい。ジオグラフィ関数の簡潔なリストと説明については [Section 13.4](#) をご覧下さい。

4.3.4 ジオグラフィに関する高度なよくある質問

1. 球または回転楕円体のどちらで計算するのでしょうか？

By default, all distance and area calculations are done on the spheroid. You should find that the results of calculations in local areas match up well with local planar results in good local projections. Over larger areas, the spheroidal calculations will be more accurate than any calculation done on a projected plane. The difference is that the spheroid takes into account the curvature and non-spherical shape of the Earth, whereas planar calculations assume a flat surface. 全てのジオグラフィ関数には、最後の真偽パラメータを 'FALSE' にすると球面を使った計算を行うというオプションがあります。これは、特にジオメトリが非常に単純である場合に計算を速くするためのものです。

2. 日付変更線や極に関してはどうなっていますか？

全ての計算に日付変更線や極の概念がありません。座標は球 (経度/緯度) であるので、日付変更線とクロスする形状は、計算の観点からは、他のものと変わりありません。

3. 処理できる最も長い弧はどうなりますか？

大圏の弧を 2 点の「補完線」として使用しています。任意の 2 点は、実際には 2 方向につながっていて、どちらの方向に行くかに依存します。PostGIS の全てのコードは、大圏コースの 2 コースのうち * 短い * 方でつながっていると仮定しています。結果として、180 度以上の弧を持つ形状は正しくモデル化されません。

4. なぜヨーロッパやロシアといった大きな範囲の面積計算はとても遅いのですか？

ポリゴンがとんでもなく大きいからです。二つの理由から、大きなエリアは悪いです。一つは、バウンダリボックスが大きいと、どのようなクエリを走らせても、インデックスがフィーチャーを引っ張ってくる傾向にあるためです。もう一つは、頂点数が巨大で、テスト (距離、包含) 関数では、少なくとも 1 回、通常は N (N は、もう一方のフィーチャーの頂点数) 回、頂点を横断しなければならないためです。ジオメトリでは、大きなポリゴンを持っているけれども小さな範囲のクエリを実行する時、ジオメトリデータ情報を小片に「非正規化」します。これにより、インデックスが効果的にオブジェクトの一部を問い合わせるようになり、またクエリが常にオブジェクト全体を引っ張りこむようなことがなくなります。ST_Subdivideを参照して下さい。ヨーロッパ全体を一つのポリゴンに * 格納できる * からといって、* そうすべき * だというわけではありません。

4.4 ジオメトリ検証

PostGIS は Open Geospatial Consortium (OGC) の Simple Feature Specification に準拠しています。この標準では、単純なジオメトリと妥当なジオメトリの概念が定義されています。これらの定義によって、Simple Feature のジオメトリモデルが一貫性があり、かつ明確な方法で空間オブジェクトを表現することができ、効率的な計算を助けます (OGC Simple Feature と SQL/MM とにおいては、単純性と妥当性について同じ定義です)。

4.4.1 単純ジオメトリ

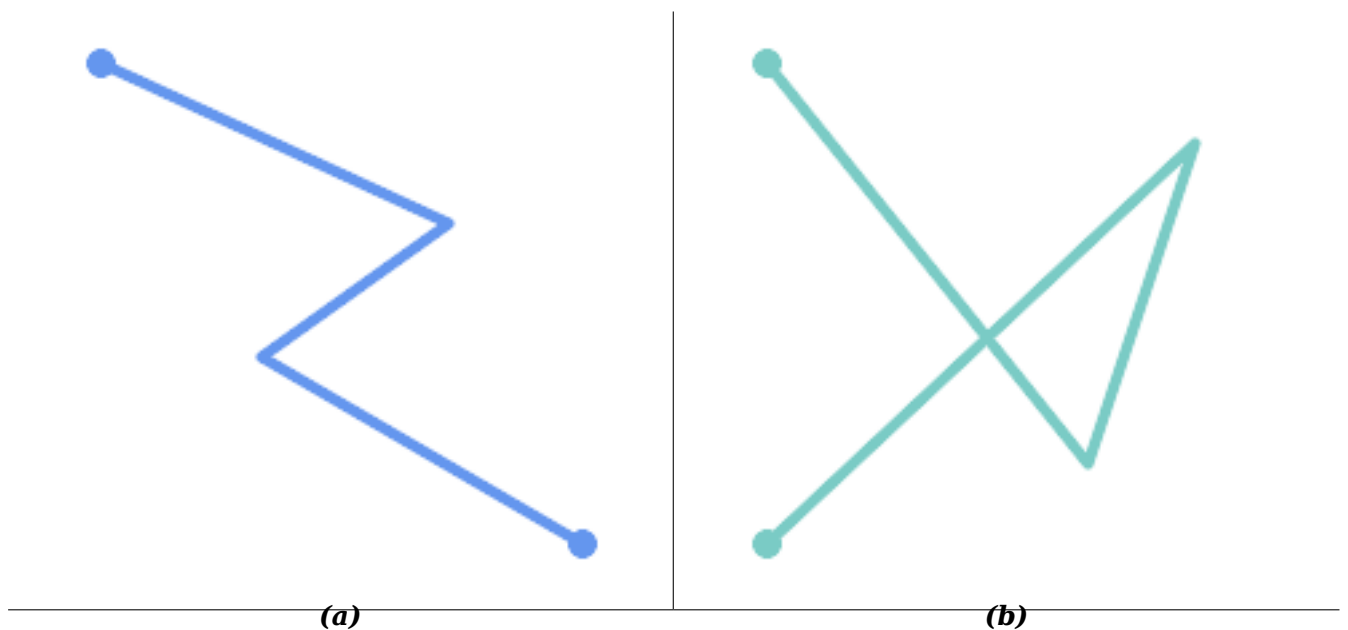
単純なジオメトリは、自己交差や自己接触といった異常な幾何学上のポイントを持たないジオメトリです。

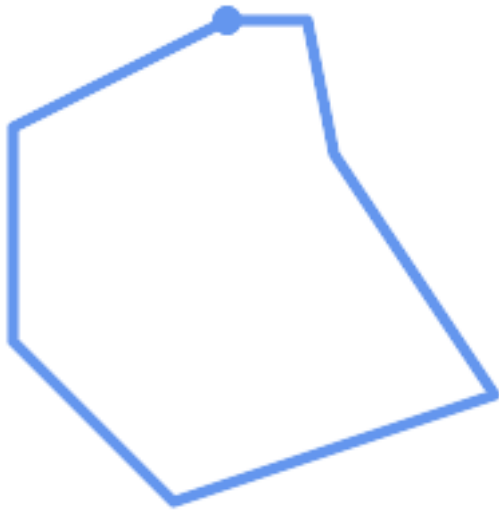
POINT は 0 次元ジオメトリオブジェクトとして常に単純です。

MULTIPOINT が単純になる条件は、任意の二つの座標 (POINT) が同じでないことです。

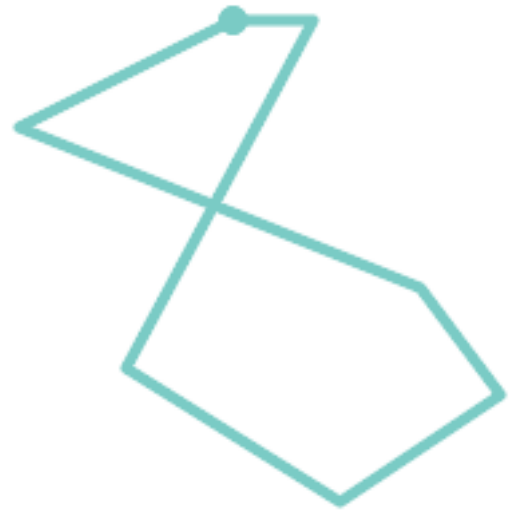
LINESTRING は、同じポイントを二回通過しないものが単純です。単純なラインストリングの端点が同一の場合には、閉じているとされ、線形リングと呼ばれます。

(a) と (c) は単純な LINESTRING です。(b) と (d) は単純ではありません。(c) は閉じた線形リングです。





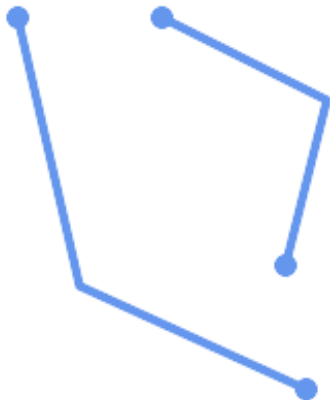
(c)



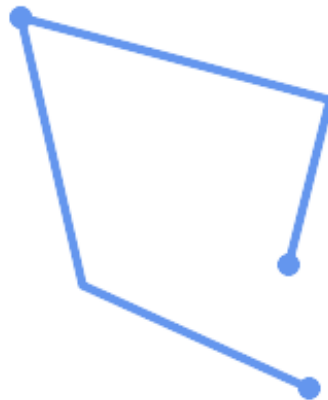
(d)

MULTILINESTRING は、要素が全て単純で、かつ、全ての要素同士のインタセクションが要素の境界上でのみ出現する場合には、単純です。

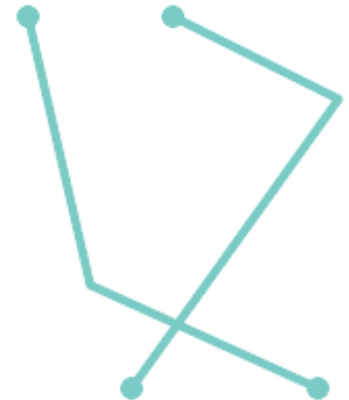
(e) と (f) は単純な **MULTILINESTRING** です。(g) は単純ではありません。



(e)



(f)



(g)

POLYGON は線形リングから形成されるので、妥当なポリゴンジオメトリは常に単純です。

ジオメトリが単純かどうかを試すには **ST_IsSimple** 関数を使います。次のようにします。

```
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
```

一般的に **PostGIS** 関数は引数ジオメトリの単純性を求めています。単純性は主にジオメトリの妥当性を定義するための基礎として用いられます。空間データモデルによっては要件としていることもあります (たとえば、線形ネットワークはしばしばクロスを認めません)。マルチポイントと線形ジオメトリは **ST_UnaryUnion** を使って単純にできます。

4.4.2 妥当なジオメトリ

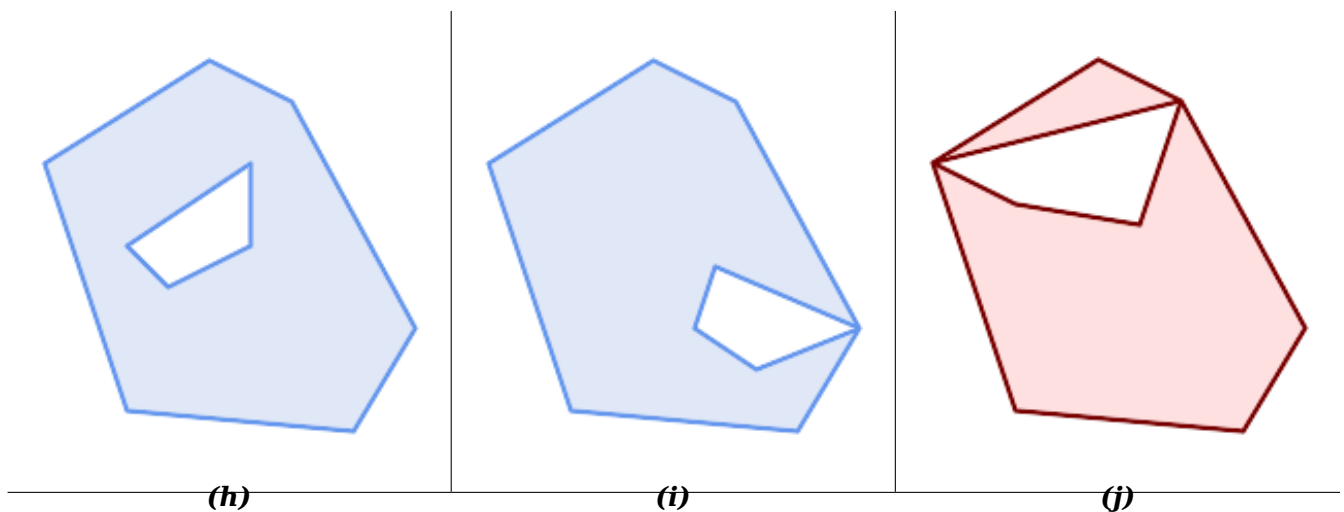
ジオメトリの妥当性は主に 2 次元ジオメトリ (**POLYGON** と **MULTIPOLYGON**) に適用されます。妥当性はポリゴンジオメトリが平面領域を明確にモデル化できる規則によって定義されます。

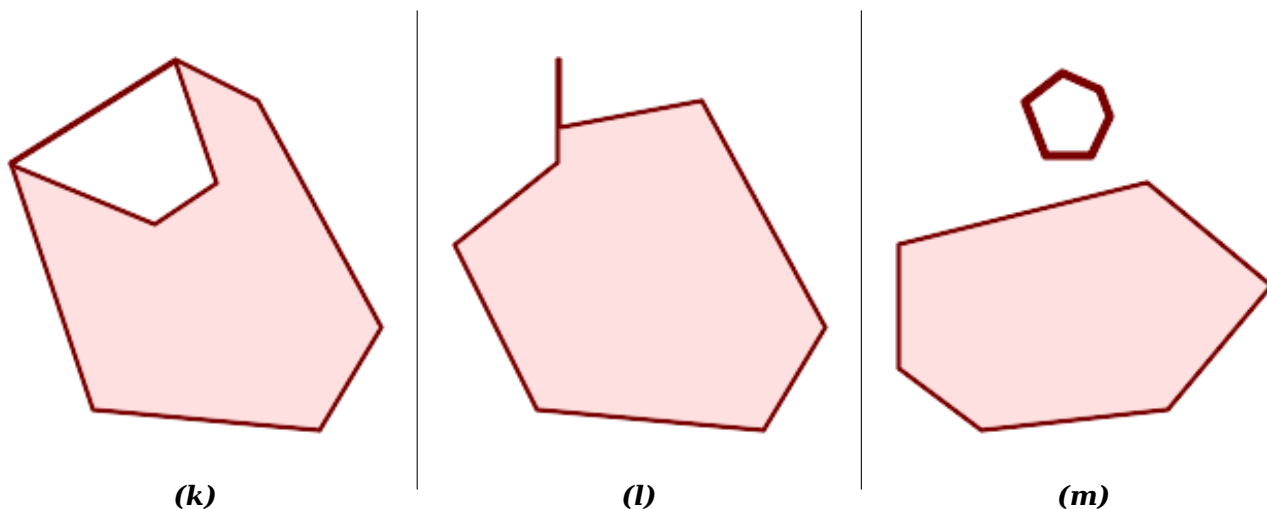
POLYGON は次の条件では妥当です。

1. ポリゴン境界リング (外側の殻リングと内側の穴リング) が単純 (交差も自己接触もしていない) であること。これによりポリゴンは切断線、トゲ、循環を持つことができなくなります。これは、ポリゴンの穴を外側のリングの自己接触 (いわゆる "inverted hole" (逆穴)) でなく、内側のリングとして表現されなければならないことを意味します。
2. 境界リングがクロスしないこと
3. 境界リングは点で接触したとしても接点として接触すること (線上にあってはなりません)
4. 内側リングは外側リング内にあること
5. ポリゴン内部は単純に接続されていること (リングはポリゴンを複数に分割するように接触してはなりません)

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

(h) と **(i)** は妥当な **POLYGON** です。**(j-m)** は不正です。**(j)** は妥当な **MULTIPOLYGON** として表すことができます。

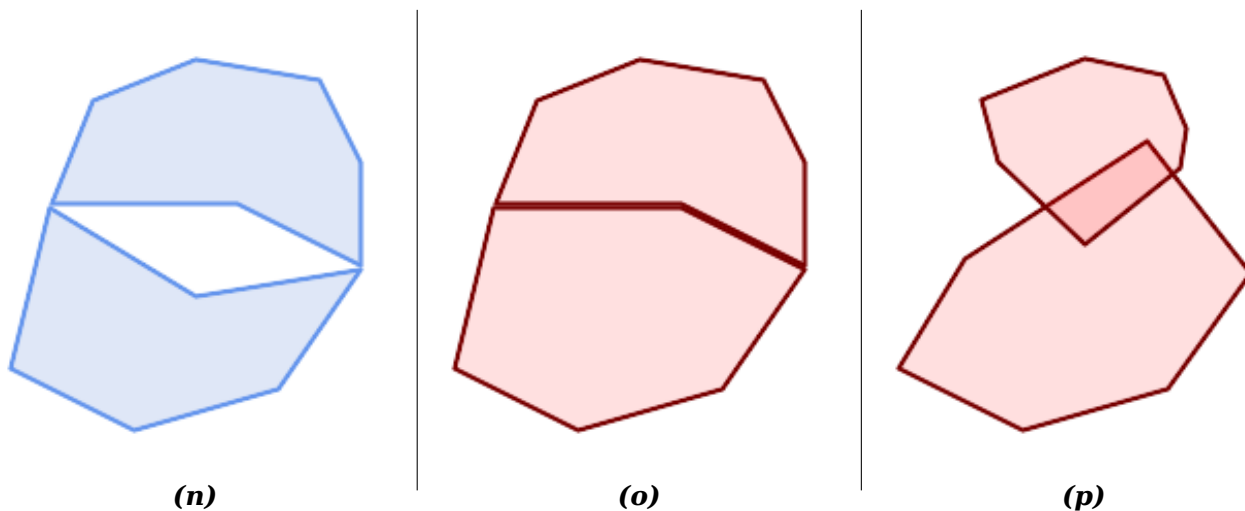




MULTIPOLYGON は次の条件では妥当です。

1. 要素となる POLYGON が妥当であること
2. 要素がオーバーラップしない (内部同士がインタセクトしない) こと
3. 要素同士の接触が点だけである (線に沿って接触しない) こと

(n) は妥当な MULTIPOLYGON です。(o) と (p) は不正です。



これらの規則は妥当なポリゴンジオメトリも単純であることも示しています。

線ジオメトリについては、**LINESTRING** が少なくとも二つのポイントを持ち、長さが 0 でない (少なくとも二つの異なるポイントを持つことと同じ)、というのが唯一の妥当性規則です。単純でない (自己交差がある) ラインは妥当です。

```
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 と MULTIPOINT は妥当性規則を持っていません。

4.4.3 妥当性の管理

PostGIS は妥当なジオメトリも不正なジオメトリも、生成も格納もできます。このため、不正なジオメトリを検出し、フラグを付け、訂正することができます。OGC 妥当性規則が求める規則 (長さが 0 のラインストリングや逆穴を持つポリゴン等) よりも厳格であることもあります。

PostGIS が提供する関数の多くは、引数ジオメトリが妥当であるとの仮定によっています。たとえば、ポリゴンの外部に穴があるポリゴンの面積を計算しても意味がありませんし、単純でない境界線からポリゴンを形成するのも意味がありません。妥当なジオメトリ入力を仮定することで、トポロジ的に正しいことを確認する必要がなくなるので、関数がより効率的に動作することができます (例外として、長さ 0 のラインと反転したポリゴンは一般的に正しく取り扱われます)。また、ほとんどの PostGIS 関数は、入力ジオメトリが妥当な場合には、妥当なジオメトリ出力を生成します。これにより、PostGIS 関数を安全に連鎖させられます。

PostGIS 関数を呼ぶときに予期しないエラーメッセージ ("GEOS Intersection() threw an error!" 等) に遭遇する場合には、まず関数の引数が妥当かどうかを確認します。妥当でないなら、次に示す方法のいずれかによる、処理中のデータの妥当性の確認を検討して下さい。



Note

関数が妥当な入力でエラーを報告する場合には、PostGIS または使用しているライブラリの一つの中にエラーがあるのを発見することがありますが、その際は PostGIS プロジェクトに報告して下さい。PostGIS 関数が妥当な入力から不正なジオメトリを返す場合も同様です。

ジオメトリが妥当かをテストするには **ST_IsValid** 関数を使います。次のようにします。

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

```
t
```

ジオメトリの不正性の性質と位置に関する情報は **ST_IsValidDetail** 関数で得られます。次のようにします。

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

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

不正なジオメトリを自動的に訂正することが望ましいような状況があります。その際は **ST_MakeValid** 関数を使います (**ST_MakeValid** は不正な入力を許す特別な関数です)。

複雑なジオメトリの不正性テストには多大な CPU 時間を取るようになるため、デフォルトでは、ジオメトリのロード時に PostGIS は妥当性の確認をしません。データソースが信用できない場合には、チェック制約を使って、テーブル上で妥当性を強制的に確認することができます。次のようにします。

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

4.5 空間参照系

空間参照系 (Spatial Reference System, SRS) (座標参照系、Coordinate Reference System, CRS と呼ばれます) は、ジオメトリが地表上の位置をどのように参照するかを定義しています。SRS には次の通り三種あります。

- 測地 (geodetic) 空間参照系は、地表に直接対応付けられる極座標系 (経度と緯度) を使います。

- 投影 (projected) 空間参照系は、回転楕円体面を「平面にする」ための数学的な投影変換を使います。距離、面積、角度といった量を直接計測することが可能な位置座標系です。この座標系はデカルト座標系ですので、原点と二つの直交軸 (通常は来北と東方向) が定義されています。個々の投影座標系は、定まった距離単位 (通常はメートルかフィート) を使います。投影座標系は、歪みを避けて定義された座標範囲に納めるために、適応範囲を制限してもいいことになっています。
- 局所 (local) 座標系は、地表への参照がないデカルト座標系です。PostGIS では SRID 値を 0 に指定します。

使用されている空間参照系には多数の相違点があります。一般的空間参照系は欧州石油調査グループ (European Petroleum Survey Group) の [EPSG database](#) で標準化されています。利便性向上のため PostGIS (と多くの空間系) は SRID と呼ぶ整数を使って空間参照系の定義を参照します。

ジオメトリは、SRID 値で空間参照系に関連付けられています。SRID 値の取得には [ST_SRID](#) を使います。ジオメトリの SRID の設定には [ST_SetSRID](#) を使います。ジオメトリ構築関数の中には、SRID を与えられるものもあります ([ST_Point](#) や [ST_MakeEnvelope](#) 等)。EWKT 書式は `SRID=n;` を前置することで SRID に対応できます。

二つのジオメトリを処理する空間関数 ([オーバレイ関数](#)、[関係関数](#) など) の入力ジオメトリは、同じ空間参照系である (同じ SRID を持つ) 必要があります。[ST_Transform](#) や [ST_TransformPipeline](#) を使うことで、ジオメトリデータを異なる空間参照系に変換できます。関数から返されるジオメトリは、入力ジオメトリと同じ空間参照系になります。

4.5.1 SPATIAL_REF_SYS テーブル

PostGIS が使用する SPATIAL_REF_SYS テーブルは利用可能な空間参照系を定義する OGC 準拠のデータベーステーブルです。このテーブルは、数値で SRID を持ち、文字列で座標系の記述を持っています。

spatial_ref_sys の定義は次の通りです。

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

カラムは次の通りです。

srid データベース内の [Spatial Reference System](#) (SRS, 空間参照系) で一意に識別される整数コードです。

auth_name この参照系に引用されている標準の名前もしくは標準そのものです。たとえば「EPSG」は妥当な auth_name です。

auth_srid 空間参照系の ID は auth_name に引用される機関によって定義されます。ここが EPSG の場合には、これは EPSG コードです。

srtext 空間参照系の Well-Known テキスト表現です。たとえば、WKT SRS の表現は、次のようになります。

```
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]
]

```

SRS WKT の詳細については、OGC 標準の [Well-known text representation of coordinate reference systems](#) をご覧ください。

proj4text PostGIS は座標変換機能を提供するために Proj4 ライブラリを用いています。proj4text カラムには、特定の SRID を示す Proj4 座標定義文字列が入ります。たとえば次のようになります。

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

詳細情報については [PROJ ウェブサイト](#) をご覧ください。spatial_ref_sys.sql ファイルには、全ての EPSG 投影について、srttext と proj4text の定義があります。

投影変換で空間参照系の定義を使用する場合には、次の戦略を取ります。

- auth_name と auth_srid がある (NULL でない) 場合には、これに基づいて PROJSRs を使います (存在する場合)。
- srttext がある場合には、可能ならそれを使用して SRS を生成します。
- proj4text がある場合には、可能ならこれを使用して SRS を生成します。

4.5.2 ユーザー定義空間参照系

PostGISspatial_ref_sys テーブルには [PROJ](#) 投影ライブラリで処理される最も一般的な空間参照系定義 3000 件以上があります。しかし、そこに無い多くの座標系があります。空間参照系に関する必要な情報がある場合は、SRS 定義をテーブルに追加できます。PROJ に詳しいなら独自の空間参照系を定義することもできます。ほとんどの空間参照系は地域的なものであり、目的の範囲外で使用する場合は意味を持たない点に注意してください。

PostGIS のコアセットに入っていない空間参照系を探すための素晴らしい資料が <http://spatialreference.org/> にあります。

一般的に使用される空間参照系には [4326 - WGS 84 経度緯度](#)、[4269 - NAD 83 経度緯度](#)、[3395 - WGS 84 メルカトル](#)、[2163 - 米国ナショナルアトラス正積図法](#)、60 個の WGS84 UTM ゾーンがあります。UTM ゾーンは計測に最適ですが、6 度 (訳注: 経度) の領域のみをカバーします (対象地域に使用する UTM ゾーンを決定するには [utmzone PostGIS plpgsql helper function](#) を参照してください)。

米国の州では、州平面空間参照系 (メートルまたはフィート単位) を使用します。この空間参照系は州ごとに一つか二つ存在します。ほとんどのメートル単位のものコアのセットに存在しますが、フィート単位の多数のものや ESRI が作成したものは [spatialreference.org](#) からロードする必要があります。

地球外の座標系でさえも定義することができます。たとえば [Mars 2000](#) です。この火星の座標系は非平面 (回転楕円体の度) ですが、geography 型で、度でなくメートル単位で長さや近接測定値を取得することができます。

割当外の SRID と PROJ 定義を使って米国中央のランベルト正角円錐図法の独自座標系をロードする例を次に示します。

```

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 空間テーブル

4.6.1 空間テーブルを作る

`geometry` 型のカラムを付けた **CREATE TABLE** SQL ステートメントでジオメトリデータを保存するテーブルを生成することができます。次の例では、BC-アルベルス座標系 (SRID 3005) の 2 次元 (XY) ラインストリングを保存するジオメトリカラムを持つテーブルを生成します。

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

`geometry` 型は、次の通り、二つの任意指定型修飾子に対応しています。

- 空間タイプ修飾子はカラムで許される形状と次元の種類を制約するものです。値は、対応している **ジオメトリタイプ** (POINT, LINESTRING, POLYGON, MULTIPOINT, MULTILINESTRING, MULTIPOLYGON, GEOMETRYCOLLECTION 等) なら全て可能です。空間タイプ修飾子は、後置詞 Z, M, ZM を付け加えることで座標次元の制約に対応します。例えば、`LINESTRINGM` 修飾子では、3 次元で 3 番目が M 軸となるラインストリングだけを許します。同様に、`POINTZM` では 4 次元 (XYZM) データが求められます。
- SRID** 修飾子は **空間参照系** の SRID を特定の数値に制約します。省略した場合には、デフォルトは 0 となります。

ジオグラフィカラムを持つテーブルの生成の例を次に挙げます。

- デフォルト SRID であらゆる種類のジオメトリを保持するテーブルの生成:

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

- 2 次元ポイントで SRID がデフォルトのテーブル作成:

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

- 3 次元 (XYZ) ポイントで SRID が 3005 のテーブル作成:

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

- 4 次元 (XYZM) ラインストリングで SRID がデフォルトのテーブル作成:

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

- 2 次元ポリゴンで SRID が 4276 (NAD 1927 地理座標系) のテーブル作成:

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

一つのテーブルが一つ以上のジオメトリカラムを持つことができます。テーブル生成時に指定するか、**ALTER TABLE** SQL ステートメントを使って追加するかで実現できます。次に 3 次元ラインストリングを格納するカラムを追加する例を示します。

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


4.6.2 GEOMETRY_COLUMNS ビュー

OGC *Simple Features Specification for SQL* は、ジオメトリテーブル構造を記述するための **GEOMETRY_COLUMNS** メタデータテーブルを定義しています。PostGIS では **geometry_columns** は、データベースのシステムカタログテーブルから読み取るビューです。これによって、空間メタデータ情報が常に現在定義されているテーブルやビューと矛盾しくなくなります。

```
\d geometry_columns
```

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

カラムは次の通りです。

f_table_catalog, **f_table_schema**, **f_table_name** ジオメトリカラムを持っている地物テーブルの完全修飾名。PostgreSQL には "catalog" の類似カラムが無いので、このカラムは空白のままです。"schema" については PostgreSQL スキーマ名が使われます (デフォルトは **public** です)。

f_geometry_column フィーチャーテーブル内のジオメトリカラムの名前。

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

srid このテーブルのジオメトリの座標系として使用される座標系空間参照系の ID です。spatial_ref_sys テーブルを参照する外部キーです (Section 4.5.1 を参照して下さい)。

type 空間オブジェクトの型。空間カラムを単一型に制限するには、POINT、LINESTRING、POLYGON、MULTIPOINT、MULTILINESTRING、MULTIPOLYGON、GEOMETRYCOLLECTION のうちのいずれかを、また、XYM で使う場合には、LINESTRINGM、POLYGONM、MULTIPOINTM、MULTILINESTRINGM、MULTIPOLYGONM、GEOMETRYCOLLECTIONM のうちのいずれかを使います。複数の型が混合するコレクションの場合は "GEOMETRY" を型とすることができます。

4.6.3 手動でジオメトリカラムを **geometry_columns** に登録する

これが必要になる事例に、SQL ビューとバルクインサートの二つがあります。バルクインサートの場合には、カラムに制約を与えるか、ALTER TABLE を実行することで、**geometry_columns** テーブル内の登録を訂正することができます。ビューの場合には、CAST 演算を使用します。カラムが型修飾子に基づく場合には、生成処理によって正しく登録されるので、何も行う必要がありません。ジオメトリに適用する空間関数を持たないビューも、基礎となるテーブルのジオメトリカラムと同じように登録されます。

For example, suppose 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;
```

To register the geometry column correctly, cast the transformed 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 the geometry type is known to be a 2D Polygon, specify it in the cast.

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

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

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

Create a 2D spatial index on the new table.

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

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

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

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

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

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

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

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

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

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

```
\d pois_ny
```

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

Indexes:

```
"pois_ny_pkey" PRIMARY KEY, btree (gid)
```

Check constraints:

```
"enforce_dims_geom_2160" CHECK (st_ndims(geom_2160) = 2)
"enforce_geotype_geom_2160" CHECK (geometrytype(geom_2160) = 'POINT'::text
OR geom_2160 IS NULL)
"enforce_srid_geom_2160" CHECK (st_srid(geom_2160) = 2160)
```

`geometry_columns` では、両方とも正しく登録されています。

```
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

しかし、次のようにビューを作ろうとします。

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

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

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

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

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

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

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

4.7 空間データのロード

空間テーブルを作成したら、これで GIS データをデータベースにアップロードする準備ができたことになります。現在、PostGIS/PostgreSQL データベースにデータをロードするには、SQL ステートメントを使う、またはシェープファイルのローダ/ダンプを使う、という二つの方法があります。

4.7.1 SQL を使ってロードする

空間データを文字表現 (WKT か WKB) に変換できたら、SQL を使うのが PostGIS にデータを持たせる最も簡単です。SQL ユーティリティの `psql` を使用して、SQL の `INSERT` ステートメントのテキストファイルをロードすると、データを PostGIS/PostgreSQL に一括読み込みできます。

データアップロードファイル (たとえば `roads.sql`) は次のようになるでしょう。

```
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 ファイルの PostgreSQL へのロードは `psql` を使います。次のようにします。

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

4.7.2 シェープファイルローダを使う

`shp2pgsql` データローダは、ESRI シェープファイルを PostGIS/PostgreSQL データベースに、ジオメトリまたはジオグラフィとして挿入するための適切な SQL に変換します。ローダには、次に示すコマンドラインフラグによって区別される、いくつかの操作モードがあります。

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

グラフィカルユーザーインタフェースを持つ `shp2pgsql-gui` もあります。コマンドラインローダのオプションのほとんどが使えます。これは、スクリプト化されていない 1 回限りのロードの場合や、PostGIS 初心者がロードする場合に、簡単に使用できます。PgAdminIII のプラグインとすることもできます。

(c|a|d|p) 相互に排他的なオプションです。

- c** 新しいテーブルの作成とシェープファイルからのデータの読み込みを行います。これがデフォルトモードです。
- a** シェープファイルからデータベーステーブルにデータを追加します。複数のファイルをロードするためにこのオプションを使う場合は、これらのファイルは同じ属性と同じデータ型を持つ必要があります。
- d** シェープファイルにあるデータを持つ新しいテーブルを作成する前にデータベーステーブルを削除します。
- p** テーブル作成の SQL コードを生成するだけで、実際のデータは追加しません。このモードは、テーブル作成とデータロードとを完全に分けたい場合に使用します。

-? ヘルプ画面を表示します。

-D 出力データに PostgreSQL のダンプ書式を用います。このモードは **-a**, **-c**, **-d** と組み合わせて利用します。デフォルトの "insert" による SQL 書式よりも、大変早くロードできます。大きなデータセットではこちらを使用して下さい。

- f **<fid_column>**, --feature-id-column **<fid_column>** Specify the name of the feature id column. The default is gid. The value must be a simple PostgreSQL identifier and cannot use a PostgreSQL system column name.
- s [**<FROM_SRID>:**]**<SRID>** 指定した SRID を持つジオメトリテーブルの生成や追加を行います。FROM_SRID が与えられた場合には、入力シェープファイルに、これを使います。この場合には、ジオメトリは変更先 SRID に投影変換します。
- k 識別子 (カラム、スキーマおよび属性) の大文字小文字を保持します。シェープファイルの属性は全て大文字であることを注意して下さい。
- i 全ての整数を標準の 32 ビット整数に強制します。DBF ヘッダではそれが正当であったとしても、64 ビットの bigint を生成しません。
- I ジオメトリカラムに GiST インデックスを生成します。
- u Create the target table as UNLOGGED. This can speed up loading transient staging data, but PostgreSQL does not preserve unlogged table contents after a crash or unclean shutdown.
- m -m a_file_name で、長いカラム名を 10 文字の DBF カラム名に対応付けるファイルを指定します。ファイルは、1 以上の行を持ちます。各行は空白区切りで二つの名前を持ち、行頭行末に空白を入れません。例を次に示します。

```
COLUMNNAME DBFFIELD1
AVERYLONGCOLUMNNAME DBFFIELD2
```

- S マルチ系ジオメトリの替りに単一ジオメトリを生成します。全てのジオメトリが実際に単一である (たとえば単一の外環でなる MULTIPOLYGON や単一の頂点でなる MULTIPOINT) 場合にのみ成功します。
- t **<dimensionality>** 出力ジオメトリが特定の次元を持つよう強制します。次元は、2D, 3DZ, 3DM, 4D の文字列を使います。
入力の次元が出力より小さい場合には、出力では 0 が入ります。入力の次元が大きい場合には、外されます。
- w 出力書式を WKB でなく WKT にします。精度が低下して、座標変動が発生しうることには注意が必要です。
- e トランザクションを使わずに、ステートメントごとに実行するようにします。エラーの元となる不良なジオメトリがいくつか含んでいる時に、大半の良好なデータのロードが可能にするものです。ダンプ書式ではトランザクションを常に使うので、-D フラグを指定している場合には使えません。
- W **<encoding>** 入力データ (dbf ファイル) のエンコーディングを指定します。全ての dbf の属性は指定されたエンコーディングから UTF8 に変換されます。SQL 出力結果には SET CLIENT_ENCODING to UTF8 が含まれるようになり、バックエンドは UTF-8 からデータベースが内部利用のために設定したエンコーディングに再変換できます。
- N **<policy>** NULL ジオメトリ操作方針 (insert*= 挿入, skip= スキップ, abort= 強制終了) を選択します。
- n DBF ファイルのみインポートします。対応するシェープファイルを持っていない場合、自動的にこのモードになり、DBF ファイルのみロードします。このフラグは、完全なシェープファイル群を持っていて、属性データだけが欲しくてジオメトリが欲しくない時のみ使用します。
- G ジオメトリ型のかわりに、ジオグラフィ型で、WGS84 経度緯度 (SRID=4326) を使用します (経度緯度データが必要です)。
- T **<tablespace>** 新しいテーブルのテーブル空間を指定します。-X パラメータが使われない場合には、インデックスはデフォルトのテーブル空間を使用します。PostgreSQL 文書には、テーブル空間を用いるべき時に関する良い文書があります。
- X **<tablespace>** 新しいテーブルのインデックスで使われるテーブル空間を指定します。主キーインデックスに適用され、-I が合わせて使われている場合には GiST 空間インデックスにも適用されます。
- Z このフラグをこれを使う時、ANALYZE 手続きの生成を防ぎます。-Z フラグが無い (デフォルトの振る舞い) 場合には、ANALYZE 手続きが生成されます。

ローダを使って入力ファイルを生成してアップロードするセッション例は次の通りです。

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

変換とアップロードは UNIX のパイプを使うと一回で実行できます。

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

4.8 空間データの抽出

空間データは SQL かシェープファイルダンパを使うと抽出できます。SQL の節では空間テーブルで比較とクエリに使用できる関数を示します。

4.8.1 SQL を使ってデータを抽出する

データベース外へのデータ抽出の最も簡単な方法は、抽出するデータセットを定義し、**SELECT** 問い合わせを使って、結果カラムを解析可能なテキストファイルにダンプすることです。

```
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)

返されるレコードの数を減らすためにある種の制限が必要になる場合があります。属性ベースで制限をかける場合には、非空間テーブルで使うのと同じ SQL 文を使います。空間に制限をかけるには次の関数を使います。

ST_Intersects この関数は、二つのジオメトリが空間を共有しているかどうかをテストします。

= この関数で、二つのジオメトリが幾何的に同一であるかを見ることができます。たとえば、`'POLYGON((0 0,1 1,1 0,0 0))'` は `'POLYGON((0 0,1 1,1 0,0 0))'` と同じかを見ることができます (これは同じとなります)。

次に、これらの演算子をクエリで使うことができます。SQL コマンドラインからジオメトリとボックスの指定を行うときは、明示的に文字列表現をジオメトリに変換しなければならないことに注意して下さい。たとえば、次のようになります。ただし 312 は架空の空間参照系番号で、ここでのデータに合致しています。

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

上のクエリは "ROADS_GEOM" テーブルから、その値と等価である単一のレコードを返します。

道路がポリゴンで定義した面を通過するかどうかをチェックするには次のようにします。

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

最も一般的な空間クエリは「フレームベース」のクエリでしょう。これは、表示するためのデータの価値のある「マップフレーム」を取得するために、データブラウザやウェブマップのようなクライアントソフトウェアに使われます。

"&&" 演算子を使うとき、比較フィーチャーを BOX3D か GEOMETRY かに指定することができます。ただし、GEOMETRY を指定すると、そのバウンディングボックスが比較に使われます。

次に示すクエリのように、フレームに BOX3D オブジェクトを使います。

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

エンベロープの投影を指定するために SRID 312 を使っていることに注意して下さい。

4.8.2 ダンプを使う

pgsql2shp テーブルダンプは、データベースに直接接続して、テーブル (あるいはクエリによって定義されたもの) をシェープファイルに変換するものです。基本的な文は次の通りです。

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

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

コマンドラインオプションは次の通りです。

- f <filename>** 特定のファイル名に出力を書きこみます。
- h <host>** 接続先データベースのホスト名。
- p <port>** 接続先データベースのポート。
- P <password>** データベースに接続するためのパスワード。
- u <user>** データベースに接続する際のユーザー名。
- g <geometry column>** 複数のジオメトリカラムを持つテーブルの場合の、シェープファイルの出力に使用するジオメトリカラム。
- b** バイナリカーソルを使います。これは、実行時間を短くしますが、テーブルの非ジオメトリ属性がテキストへのキャストを持っていない場合には、動作しません。
- r** Raw モード。gid フィールドを落としたり、カラム名をエスケープしてはいけません。
- m filename** 識別名を 10 文字名に再割り当てします。ファイルの中身は、一つの空白で区切られ、前と後に空白が無い二つのシンボルの行からなります。VERYLONGSYMBOL SHORTONE ANOTHERVERYLONGSYMBOL SHORTER 等となります。

4.9 空間インデックス

インデックスによって巨大データセットの空間データベースの使用が可能となります。インデックス無しでは、地物の検索を行う際に、データベースの全てのレコードに対するシーケンシャルスキャンが必要となります。インデックスによって、レコード探索のために早く移動できる構造を構築するので、検索速度が向上します。

一般的に属性データに使われるインデックス手法であり B 木は、空間データではあまり有用ではありません。1 次元データの格納とクエリにだけしか対応していないためです。ジオメトリのような 2 次元以上の次元を持つデータでは、全ての次元の範囲を指定できるインデックス手法が求められます。PostgreSQL の空間データ処理に関する主要な利点の一つに、多次元データで上手く動作する GiST、BRIN、SP-GiST の複数のインデックス手法を提供していることです。

- **GiST (Generalized Search Tree)** インデックスは、データを「一方にあるもの」「オーバーラップするもの」「内部にあるもの」に分解するもので、GIS データを含む幅広い範囲で使えます。PostGIS は GiST インデックス空間データを R 木インデックス実装のベースにしています。GiST は最も一般的に使われ、多目的なインデックス手法で、非常に良好な問い合わせ効率を提供しています。
- **BRIN (Block Range Index)** インデックスは、空間範囲を集計することで動作します。探索は範囲のスキャンを通して行われます。BRIN は一部の種類 (空間的にソートされ、更新がほぼ無いか全く無い) のデータだけに適切です。しかし、インデックス生成時間は非所に早く、インデックスサイズは非常に小さくなります。
- **SP-GiST (Space-Partitioned Generalized Search Tree)** は 4 分木、kd 木、基数木 (トライ木) のような部分木探索に対応する一般的なインデックス手法です。

空間インデックスはジオメトリのバウンディングボックスだけを格納します。空間クエリはインデックスは初期フィルタとして使用して、クエリ条件に一致する可能性のあるジオメトリを早く求めます。ほとんどの空間クエリでは、空間述語関数を使って特定の空間条件をテストする二次フィルタが必要です。空間述語関数を使ったクエリの詳細情報については [Section 5.2](#) をご覧下さい。

また、[PostGIS Workshop section on spatial indexes](#) と [PostgreSQL manual](#) もご覧下さい。

4.9.1 GiST インデックス

GiST は「汎用検索木 (Generalized Search Tree)」の意味で、多次元データのインデックスの一般化された形式です。PostGIS は GiST 上で実装している R 木インデックスを空間データのインデックスに使用しています。GiST は最も一般的に使われ、多目的なインデックス手法で、クエリ能率を非常に良くします。他の GiST の実装は、通常の B 木インデックスに従わない全ての種類の不規則なデータ構造 (整数配列、スペクトラルデータ等) の検索速度を向上させるために使います。詳細情報については [PostgreSQL manual](#) をご覧ください。

GIS データテーブルが数千行を超えたら、空間検索の速度向上のためインデックスを構築したくなるでしょう (これは属性検索でない場合です。属性でしたら通常のインデックスを属性フィールドに追加します)。

GiST インデックスをジオメトリカラムに追加するための文は次の通りです。

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

上の文では常に 2 次元インデックスを構築します。n 次元インデックスをジオメトリ型で使うには、次の文でインデックスを生成できます。

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

空間インデックスの構築は、計算量を集中させて行われます。また、この時には、テーブルへの書き込みアクセスがブロックされます。そのため、本番システムではより遅い CONCURRENTLY を選択するかも知れません。次のようにします。

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

インデックス構築後に、時々 PostgreSQL にテーブルの統計情報を集めさせると助かります。クエリプランの最適化に使われます。

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


4.9.2 BRIN インデックス

BRIN は“Block Range Index”の意味です。PostgreSQL が提供する汎用インデックスです。BRIN は不可逆インデックスですので、レコードが検索条件に合致するか確認する二次チェックが必要です (これは提供されているすべての空間インデックスと同じです)。BRIN は、インデックス生成が非常に高速で、インデックスサイズが非常に小さく、読み込み効率がそこそこ良いものです。これの主目的は、巨大テーブルに対してテーブル内の物理位置と相関するカラムの上にインデックスを生成することです。BRIN は各種属性データ構造 (整数、配列など) の高速検索が可能です。詳細情報については [PostgreSQL manual](#) を参照してください。

空間テーブルが、ひとたび数千行を超えると、データの空間検索の速度向上にインデックスが必要と感じることになります。GiST インデックスは、サイズがデータベースで使える RAM 容量を超えず、インデックスのストレージサイズに余裕があり、書き込み時のインデックス更新コストにも余裕があるなら、非常に高いパフォーマンスを発揮します。そうでない場合には、非常に大きなテーブルにおいては、BRIN インデックスを代替に考えることができます。

BRIN インデックスは、連続するテーブルブロックの集合 (ブロック範囲と言います) の全てのジオメトリを囲むバウンディングボックスを格納します。インデックスを使用した問い合わせを実行する時に、問い合わせ範囲とインタセクトするブロック範囲を見つけるためにスキャンします。これは、データが物理的に整列していて、ブロック範囲のバウンディングボックスのオーバーラップが最小である (理想的には相互に排他的である) 場合に限って効率的です。結果インデックスは非常に小さいサイズですが、通常、読み込み効率は、同じデータにおける GiST インデックスより悪くなります。

BRIN インデックスの構築は、は GiST インデックスと比べて、CPU 集中を非常に減らします。BRIN インデックスは GiST インデックスよりも、同じデータに対して 10 倍速く構築するのが普通です。BRIN インデックスはテーブルブロックの範囲ごとに一つのバウンディングボックスしか格納しないので、GiST インデックスと比べて、ディスクスペースを 1000 倍少なくできます。

レンジ内で要約するブロック数を選択できます。この数字を減らすと、インデックスは大きくなりますが、効率向上の助けになる可能性があります。

BRIN を効果的にするには、テーブルデータをブロック範囲のオーバーラップの量を最小にするような物理的オーダーで格納します。データが既に適切に並び替えられているかも知れません (たとえば、既に空間オーダーで並び替えられているデータセットを他のデータベースからロードする場合)。そうでない場合には、一つの空間キーによるデータの並べ替えで実現できます。一つの方法として、ジオメトリ値で並べ替えた新しいテーブルを生成することです (最近の PostGIS のバージョンで効果的なヒルベルト曲線オーダーが使われています)。

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

もしくは、データは、ジオハッシュを (一時的な) インデックスに使い、そのインデックスでクラスタリングを行うことによって適切に並べ替えることができます。

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

BRIN インデックスをジオメトリカラムに追加するための文は次の通りです。

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

上の文で 2 次元インデックスを構築します。3 次元インデックスをビルドするには、この文を使います。

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

また、4 次元演算子クラスを使う 4 次元インデックスを使うこともできます。

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

上記のコマンドでは、範囲のブロック数はデフォルトの 128 を使用しています。集計で範囲のブロック数を指定するには、この文を使います。

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

また、BRIN インデックスは、多数の行で一つのインデックス値を格納することを心に留めておいて下さい。テーブルに違う次元のジオメトリを格納する場合には、インデックスの効率が悪くなります。この効率欠落を回避するには、格納したジオメトリの次元数の最小値となる演算子クラスを選択します。

geography 型も BRIN インデックスに対応しています。BRIN インデックスをジオグラフィカラムに構築するための文は次の通りです:

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

上の文では常に回転楕円体面上の地理空間オブジェクトの 2 次元インデックスを構築します。

現在のところは「包括対応」だけをここで考えています。これは、**&&**, **~**, **@** の演算子だけが 2 次元で使われることを意味します (ジオメトリとジオグラフィの両方)。**&&&** 演算子は 3 次元ジオメトリで使えます。しばらくは KNN 検索に対応しません。

BRIN と他のインデックスとの重要な違いは、データベースがインデックスを動的に保守しないことです。テーブルの空間データを変更すると、単純にインデックスの末尾に追加しています。このためインデックス探索の能率が時間とともに低下します。インデックスは **VACUUM** か空間関数 **brin_summarize_new_values(regclass)** を実行することで更新できます。このため、BRIN は読み込み専用か、書き込みがほとんど発生しないようなデータでの利用では最も適切になりえます。詳細情報については、[manual](#) をご覧下さい。

空間データに BRIN を使用して集計するには:

- インデックス構築時間は非常に速く、インデックスサイズは非常に小さいです。
- インデックスのクエリ時間は GiST より遅いですが、十分許容できます。
- テーブルデータを空間順序で並べ替える必要があります。
- 手動でインデックスの保守をする必要があります。
- 巨大なテーブルであって、オーバーラップが少ないか無く (ポイントなど)、かつ静的か頻繁には変更しないようなものに、最も適しています。
- 比較的多数のデータレコードを返すクエリでの使用が、より効果的です。

4.9.3 SP-GiST インデックス

SP-GiST は、「空間分割された一般探索木」を表します。四分木、k 次元木、基数木 (トライ木) のような分割探索木に対応するインデックスの総称的な形式です。このデータ構造の一般的な機能は、検索空間を反復して分割することですが、分割は等しいサイズである必要はありません。SP-GiST は、GIS インデックスだけでなく、電話回線のルーティングや、IP ルーティング、部分文字列検索等といった、様々な種類のデータを探索する速度の向上に使われます。詳細情報については [PostgreSQL manual](#) をご覧下さい。

GiST インデックスを利用しているので、空間オブジェクトを覆うバウンディングボックスを保存するという意味で、SP-GiST インデックスは不可逆です。SP-GiST インデックスは、GiST インデックスの代替と考えることができます。

一度 GIS データテーブルが数千行を超えると、データの空間探索の速度向上に SP-GiST インデックスを使うと良いかも知れません。「ジオメトリ」カラムに SP-GiST インデックスを構築するための文は次の通りです。

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

上の文では、2 次元インデックスを構築します。ジオメトリ型の 3 次元インデックスは、次のように、3 次元演算子クラスを使用して生成します。

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

空間インデックスの構築は、計算量を集中させて行われます。また、この時には、テーブルへの書き込みアクセスがブロックされます。そのため、本番システムでは、より遅い CONCURRENTLY を選択するかも知れません。次のようにします。

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

インデックス構築後に、時々 PostgreSQL にテーブルの統計情報を集めさせると助かります。クエリプランの最適化に使われます。

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

SP-GiST インデックスは次の演算子を含むクエリの実行速度を向上させられます。

- 2 次元インデックスについては <<, &<, &>, >>, <<|, &<|, |&>, |>>, &&, @>, <@, ~= です。
- 3 次元インデックスについては &/&, ~=, @>>, and <<@。

現時点では kNN 探索に対応していません。

4.9.4 インデックス使用のチューニング

通常、インデックスは知らないうちにデータアクセスの速度を向上します。ひとたびインデックスを構築すれば、PostgreSQL クエリプランナは自動的にクエリの能率を向上させるために使うべきかどうかを決定します。しかし、プランナが既存のインデックスを選択せず、遅いシーケンシャルスキャンを使い続ける場合があります。

空間インデックスが使われていないのが分かった場合には、少しの行えることがあります。

- クエリプランの試験とクエリの確認で、必要なものを計算できます。誤った JOIN や忘れ去られたテーブルや間違ったテーブルでは、予期しないテーブルレコード検索が複数回行われることがあります。クエリプランを得るにはクエリの先頭に **EXPLAIN** を付けて実行します。
- テーブル内の値の数量と分布に関する統計情報を収集するとともに、クエリプランナにインデックス使用にかかる意思決定のための、より良い情報を与えるようにします。**VACUUM ANALYZE** は両方を計算します。
データベースに対する定期的な vacuum は常に実行するべきです。多くの PostgreSQL データベースエージェントは、閑散時の cron ジョブとして定期的に **VACUUM** を実行します。
- **VACUUM** が役に立たない場合には、**SET ENABLE_SEQSCAN TO OFF;** コマンドを使用して、一時的にプランナにインデックス情報の使用を強制することができます。この方法で、プランナがインデックス使用を多くしたクエリプランを生成できるかどうかを確認できます。このコマンドはデバッグにのみ使用してください。一般的に言えば、プランナはインデックスを使用するタイミングをよく知っています。クエリを実行したら **SET ENABLE_SEQSCAN TO ON;** を実行して、他のクエリでは通常操作にすることを忘れないでください。
- **SET ENABLE_SEQSCAN TO OFF;** でクエリ速度が向上する場合には、PostgreSQL のハードウェア関連のチューンが行われていないのかも知れません。プランナがシーケンシャル対インデックスのコストが誤っている場合には、**RANDOM_PAGE_COST** (postgresql.conf 内にあります) の値を変更してみてください。または **SET RANDOM_PAGE_COST TO 1.1;** とします。**RANDOM_PAGE_COST** のデフォルト値は 4.0 です。1.1 (SSD の場合) または 2.0 (高速磁気ディスクの場合) を試してみてください。値を小さくするほど、プランナがインデックススキャンをしやすくなります。
- **SET ENABLE_SEQSCAN TO OFF;** がクエリの助けにならないなら、クエリは PostgreSQL プランナがまだ最適化できない SQL 構成なのかも知れません。プランナが処理できるようにクエリを再記述できるかも知れません。例えば、インライン SELECT を持つ副問い合わせがあると、効果的なプランを作らないことがあり、**LATERAL JOIN** を使うように書き換えることができます。

詳細情報については PostgreSQL マニュアルの[問い合わせ計画節](#)をご覧ください。

Chapter 5

空間クエリ

空間データベースのレゾナードトルは、通常ならデスクトップ GIS の機能が必要なクエリをデータベース内で実行することです。PostGIS を使うには、使用可能な空間関数は何かを知り、またクエリ内でどう使うかを知って、適切なインデックスで能率を向上させることが求められます。

5.1 空間関係の決定

空間関係は、二つのジオメトリについて、一方がもう一方にどのような相互関係になっているかを示すものです。ジオメトリのクエリにおける基本的な機能です。

5.1.1 次元拡張 9 交差モデル

OpenGIS Simple Features Implementation Specification for SQLによると「二つのジオメトリの比較の基本的なアプローチは、二つのジオメトリの内部、境界、外部のインタセクションの比較と、『インタセクション行列』の要素に基づく 2 ジオメトリの関係の分類です」。

点集合トポロジでは、2 次元空間に埋め込まれたジオメトリの中にあるポイントは、次に示す三つの集合に分類されます。

境界

ジオメトリの境界は、一次元低いジオメトリです。POINT では、次元が 0 になり、境界は空集合です。LINESTRING の境界は二つの端点です。POLYGON の境界は、外環と内環の線です。

内部 (Interior)

ジオメトリの内部は、ジオメトリの境界以外のポイントです。POINT では、内部はポイント自体です。LINESTRING の内部は端点の間のポイントの集合です。POLYGON の内部は、ポリゴン内部の面です。

外部 (Exterior)

ジオメトリの外部はジオメトリが組み込まれた空間の残りです。言い換えると、ジオメトリの内部にも境界にもない点の全てです。これは 2 次元の閉じていない面になります。

次元拡張 9 交差モデル (Dimensionally Extended 9-Intersection Model, DE-9IM) は、二つのジオメトリの空間関係を九つの交差の次元を指定することで記述します。交差次元は 3×3 の交差行列で正式に表現することができます。

ジオメトリ g に対する内部、境界、外部は $I(g)$ 、 $B(g)$ 、 $E(g)$ と表記します。また、 $dim(s)$ は s の集合を $\{0, 1, 2, F\}$ の値で示します。

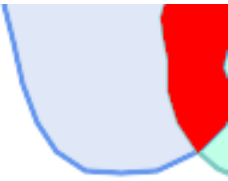
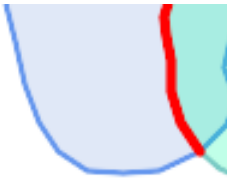
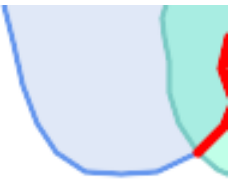
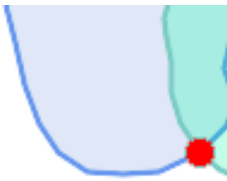
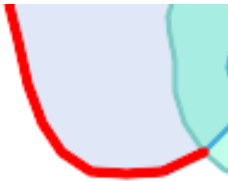
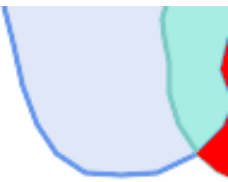
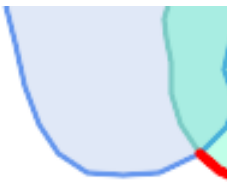
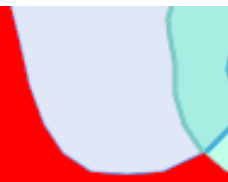
- 0 => 点
- 1 => 線
- 2 => 面
- F => 空集合

この表記法を使うと、二つのジオメトリ a と b の交差行列は次の通りです。

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

二つのオーバーラップするポリゴンについて可視化すると、次のようになります。



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

左から右に、上から下に読みます。交差行列の文字列表現は'**212101212**' です。

詳細情報については次をご覧ください。

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

5.1.2 名前付き空間関係

共通の空間関係を簡単に決定できるように、PGC SFS は名前付き空間関係述語の集合を定義しています。PostGIS では **ST_Contains**、**ST_Crosses**、**ST_Disjoint**、**ST_Equals**、**ST_Intersects**、**ST_Overlaps**、**ST_Touches**、**ST_Within** が提供されています。非標準の空間関係述語 **ST_Covers**、**ST_CoveredBy**、**ST_ContainsProperly** も定義されています。

空間述語は通常 SQL の **WHERE** 節や **JOIN** 節内で条件に使用されます。名前付き空間述語は、インデックスが有効なら自動的に空間インデックスを使うので、バウンディングボックス演算子 **&&** を使う必要はありません。例えば次のようになります。


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

詳細や図については[PostGIS Workshop](#)をご覧ください。

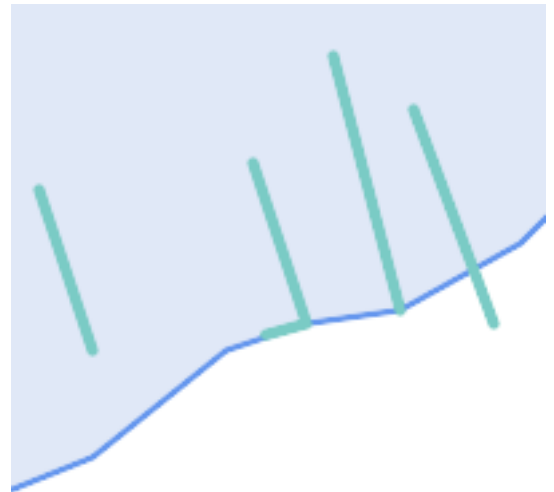
5.1.3 一般的な空間関係

名前付き空間関係が求める空間フィルタ条件を与えるのに不十分となる場合があります。



For example, consider a linear dataset representing a road network. It may be required to identify all road segments that cross each other, not at a point, but in a line (perhaps to validate a business rule such as ensuring that no two different roads share a common segment). In this case **ST_Crosses** does not provide the necessary spatial filter, since for linear features it returns true only where they cross at a point.

2 ステップ解決法を示します。まず、空間的にインタセクトしている同路線の二本を抜き出し (**ST_Intersects**)、実際にインタセクトしている部分を計算 (**ST_Intersection**) します。次いで、インタセクトしている部分の **ST_GeometryType** が **LINestring** かどうかを確認します (**[MULTI]POINT**、**[MULTI]LINestring** 等の **GEOMETRYCOLLECTION** を返す場合に適切に処理します)。
明らかに、より単純でより速い解法が望ましいです。



二つ目の例では、湖の境界とインタセクトし、かつ終端が岸に上がっている波止場を見つけます。言い換えると、波止場が湖に含まれるが完全には含まれず、湖の境界線とインタセクトして、波止場の終端が確実に湖内または境界にある場合を指します。空間述語を併用すると求める地物を見つけることができます。

- `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`

…言うまでもないですが非常に複雑ですね。

この要件は完全な DE-9IM 交差行列の計算で満たすことができます。PostGIS は、これを行う `ST_Relate` 関数を提供しています。次のようにします:

```
SELECT ST_Relate(
  'LINESTRING (1 1, 5 5)',
  'POLYGON ((3 3, 3 7, 7 7, 7 3, 3 3))' );
```

```
1010F0212
```

特定の空間関係をテストするには、交差行列パターンを使います。これは、追加シンボル `{T,*}` で拡張された行列表現です。

- `T =>` インタセクションの次元は空ではないという意味です。すなわち `{0,1,2}` のいずれかです。
- `* =>` 何でも良い

交差行列パターンを使って、特定の空間関係の評価がより簡潔な方法で可能です。交差行列パターンのテストに `ST_Relate` と `ST_RelateMatch` を使うことができます。上に挙げた一つ目の例では、二つのラインがライン内部でインタセクトする交差行列パターンは `'1*1***1**'` となります。

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


二つ目の例です。一本のラインが部分的にポリゴン内部とポリゴン外部とにある場合の交差行列パターンは'**102101FF2**'となります。

```
-- 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 空間インデックスを使う

空間条件を使用するクエリを構築する時、最良の効果を得るには、空間インデックスが存在する場合に (Section 4.9参照) これを確実に使用することが重要です。そのためには、**WHERE** 節や **ON** 節で、空間演算子またはインデックス対応関数を使用しなければなりません。

空間演算子には、バウンディングボックス演算子 (最もよく使われるのは**&&**です。Section 7.10.1参照)、および近傍クエリで使用される距離演算子 (最もよく使われるのは**<->**です。Section 7.10.2参照) が含まれます。

インデックス対応関数は、自動的にバウンディングボックス演算子を空間条件に追加します。インデックス対応関数は空間関係述語を含みます。空間関係述語には、**ST_Contains**, **ST_ContainsProperly**, **ST_CoveredBy**, **ST_Covers**, **ST_Crosses**, **ST_Intersects**, **ST_Overlaps**, **ST_Touches**, **ST_Within**, **ST_Within**, **ST_3DIntersects**があり、距離述語には**ST_DWithin**, **ST_DFullyWithin**, **ST_3DDFullyWithin**, **ST_3DDWithin** があります。

ST_Distance といった関数は、演算の最適化のためにはインデックスを使用しません。例えば、次のクエリは、大きなテーブルでは非常に遅くなります。

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

このクエリは **geom_table** テーブル内の、(100000, 200000) のポイントから 100 単位内にある全てのジオメトリを選択します。テーブル内の個々のポイントと指定したポイントとの距離を計算しているため、非常に遅くなります。すなわち、1 回の **ST_Distance()** の計算で、テーブルの全ての行について計算することになります。

インデックス対応関数**ST_DWithin**を使用すると、処理行数を実質的に減らすことができます。次のようにします。

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

このクエリは、同じジオメトリを選択しますが、より効率的な方法を取ります。**ST_DWithin()** が内部で **&&** 演算子をクエリジオメトリのバウンディングボックスを拡大したボックスで使うことによって可能となります。**geom** 上に空間インデックスが存在するなら、クエリプランナは距離計算の前に対象行数を減らすためにインデックスを使えることを認識します。空間インデックスによって、バウンディングボックスが拡張された範囲とオーバーラップするジオメトリだけを検索して、そのため、求めようとする距離内にあるかも知れないジオメトリを検索することができます。その後で、結果集合内のレコードを含めるかどうかを確認するための実際の距離計算が行われます。

詳細情報と例については[PostGIS Workshop](#)をご覧ください。

5.3 空間 SQL の例

本節の例では、線の道路のテーブルとポリゴンの市区町村境界テーブルとを使います。**bc_roads** テーブルの定義は次の通りです。

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

bc_municipality テーブルの定義は次の通りです。

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

1. 道路の総延長は *km* 表記でいくらになるでしょう？

この問題は、次のようなとても単純な SQL で答えを得ることができます。

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

km_roads
70842.1243039643

2. プリンスジョージ市の大きさは *ha* 表記でいくらになるでしょう？

このクエリでは、属性条件 (municipality name, 自治体名) に (ポリゴン面積の) 空間計算を併用しています。

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

hectares
32657.9103824927

3. 県内で最も大きな面積となる自治体はどこでしょう？

このクエリでは、順序付けの値に空間計測関数を使っています。この問題に対しては複数の方法がありますが、最も効果的な方法は次の通りです。

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

name	hectares
TUMBLER RIDGE	155020.02556131

このクエリの答えを出すためには、全てのポリゴンの面積を求める必要があることに注意して下さい。このクエリを多く実行する場合、性能向上のためにテーブルに **area** カラムを追加して、別のインデックスを追加することができるようにするのは、意義のあることです。結果を距離について降順に並べ替え、PostgreSQL の "LIMIT" コマンドを用いることで、**max()** のような集約関数を使わずに、簡単に最も大きい値を集約関数を得ることができます。

4. 各自治体内に含まれる道路の総延長はいくらでしょう？

これは、二つのテーブルからデータを持ち込んで (結合して) いるので「空間結合」の例です。しかし、結合の条件として共通キーの上で接続するという普通のリレーションのやり方でなく空間インタラクション条件 (「含む」) を使っています。

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

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

このクエリは、テーブル内の全ての道路の合計を最終結果 (この例での話ですが約 250Km の道です) にまとめられるので、少し時間がかかります。より小さいオーバーレイ (数百の道路で数千のレコード) の場合、応答はもっと早くなりえます。

5. プリンスジョージ市内の全ての道路からなるテーブルを作ります。

これは「オーバーレイ」の例です。つまり、二つのテーブルを取得して、空間的に切り取られた結果からなる新しいテーブルを出力します。上で示した「空間結合」と違い、このクエリは実際に新しいジオメトリを生成します。生成されたオーバーレイはターボのかかった空間結合みたいなもので、より確かな解析作業に便利です。

```
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. ビクトリア州の「ダグラス通り」の長さは km 表記でいくらになるでしょう？

```
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. 穴を持つ自治体ポリゴンのうち最も大きいのはどれでしょう？

```
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

性能向上に関する技法

6.1 大きなジオメトリを持つ小さなテーブル

6.1.1 問題の説明

現版の PostgreSQL では TOAST テーブルに関するクエリオプティマイザの弱さに苦しみます。TOAST テーブルは、(長いテキスト、イメージ、多数の頂点を持つ複合ジオメトリといった) 通常のデータページに適合しない、(データサイズという意味では) 巨大な値を納めるための「拡張部屋」の一種です。詳細情報は [PostgreSQL ドキュメントの TOAST](#) をご覧ください。

(高解像度で全てのヨーロッパの国の境界を含むテーブルのような) 大きなジオメトリがあるうえ、行がそう多くないテーブルを持つようになると、この問題が出てきます。テーブル自体は小さいのですが、多くの TOAST スペースを使います。例として、テーブル自体は概ね 80 行で 3 データページしか使わなくても TOAST テーブルで 8225 ページを使うとします。

ここで、ジオメトリ演算子の `&&` を使って、ほとんどマッチしないようなバウンダリボックスを検索するクエリを出してみます。クエリオプティマイザにはテーブルは 3 ページ 80 行しかないように見えます。オプティマイザは、小さなテーブルを順に走査の方がインデックスを使うよりも早いと見積もります。そして、GiST インデックスは無視すると決めます。通常なら、この見積もりは正しいです。しかし、この場合は `&&` 演算子が全てのジオメトリをディスクから呼び出してバウンディングボックスと比較しなければならなくなり、ゆえに、全ての TOAST ページもまた呼び出す必要があります。

この問題に苦しむかどうかを見るには、PostgreSQL の `"EXPLAIN ANALYZE"` コマンドを使います。詳細情報と技術情報については、PostgreSQL 性能メーリングリストのスレッド <http://archives.postgresql.org/pgsql-performance/2005-02/msg00030.php> をご覧下さい。

また、PostGIS の新しいスレッド <https://lists.osgeo.org/pipermail/postgis-devel/2017-June/026209.html> もご覧下さい。

6.1.2 応急処置

PostgreSQL コミュニティでは、TOAST を意識したクエリ見積もりを作ることで、この問題を解決しようとしています。今のところは、二つの応急処置があります。

一つは、クエリプランナにインデックスの使用を強制することです。クエリを発行する前に `"SET enable_seqscan TO off;"` をサーバに送信します。これは基本的にクエリプランナに対して可能な限り順に走査することを避けるよう強制します。そのため GiST インデックスを通常使うようになります。しかし、このフラグは接続するたびに設定しなければならず、他のケースにおいてはクエリプランナに誤った見積もりをさせることになるので、`"SET enable_seqscan TO on;"` をクエリの後に送信すべきです。

もう一つは、順に走査することをクエリプランナが考える程度に早くすることです。これは、バウンダリボックスの「キャッシュ」を行う追加カラムを作成し、このカラムにマッチさせるようにすることで達成することができます。ここでの例では次のようになります。

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

そして、次のように、`&&` 演算子を `geom_column` に対して行っていたものを `bbox` に変更します。

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

もちろん、`mytable` の行を変更または追加したら、`bbox` を「同期」するようにしなければなりません。最もすっきりした方法はトリガです。もしくは、アプリケーションを変更して `bbox` カラムの現状を保持するか、テーブル更新後にいつも `UPDATE` クエリを実行するかも対応できます。

6.2 ジオメトリインデックスで **CLUSTER** を実行する

読み込むことがほとんどで、かつほとんどのクエリでひとつのインデックスを使うようなテーブルのために、PostgreSQL は `CLUSTER` コマンドを提供しています。このコマンドは、全てのデータ行を、インデックス基準にあわせて物理的に再整理するので、二つの性能の利点を生みます。一つは、インデックスの範囲走査のために、データテーブルのシーク回数が劇的に減少することです。もう一つは、いくつかの小さなインデックス間隔に集中する場合には、データ行が分布するデータページがより少なくなることで、より効率的なキャッシュを持つことです (この点は、PostgreSQL マニュアルの `CLUSTER` コマンドのドキュメントを読むように仕向けられていると感じて下さい)。

しかし、GiST インデックスは単純に `NULL` 値を無視するため現在のところ PostGIS の GiST インデックスのクラスタリングはできず、次のようなエラーメッセージを得ます。

```
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.
```

ヒントメッセージにある通り、テーブルに `"not null"` 制限を追加することで、この欠陥にとりあえず対応できます。例を示します。

```
ALTER TABLE my_table ALTER COLUMN geom SET not null;
```

```
ALTER TABLE
```

もちろん、ジオメトリカラムで実際に `NULL` 値が必要な場合、この対応はできません。さらには、制限を追加するには上の方法を使わなければならない、`"ALTER TABLE blubb ADD CHECK (geometry is not null);"` のような `CHECK` 制限は使えません。

6.3 次元変換の回避

ときどき、テーブルで 3 次元、4 次元のデータを持つのに、常に OpenGIS 準拠の `ST_AsText()` または `ST_AsBinary()` 関数を使ってアクセスして 2 次元ジオメトリを出力させるようなことが起きます。内部で `ST_Force_2d()` 関数を呼んでいるために発生しますが、これは、大きなジオメトリでは重大なオーバヘッドを誘引することになります。このオーバヘッドを回避するには、一度追加された次元を前もって落とし、かつこれを永続化するのが適当かも知れません。

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

`AddGeometryColumn()` を使ってジオメトリカラムを追加した場合、ジオメトリの次元に関する制限があることに注意してください。この制限を迂回するには、制限の削除が必要になります。`geometry_columns` テーブル内のエントリを更新して、その後で制限を再作成することを忘れないで下さい。

大きなテーブルの場合、**WHERE** 節、およびプライマリキー若しくは他の適切な基準によってテーブルの一部への **UPDATE** を制限させて、**UPDATE** の実行の間に単に `"VACUUM;"` と実行することで、**UPDATE** をより小さい塊に分割するのが賢いやり方かもしれません。これにより、テンポラリディスクスペースが劇的に減少します。さらに、次元混合のジオメトリを持つ場合、`"WHERE dimension(the_geom)>2"` によって **UPDATE** を制限することで、2次元で書かれているジオメトリの再書き込みをスキップさせることができます。

Chapter 7

PostGIS リファレンス

ここで示す関数は PostGIS ユーザが必要とすると思われる関数です。この他に、一般的なユーザが使わない PostGIS オブジェクトに対して求められるサポート関数があります。



Note
PostGIS は、既存の名前付け方針から SQL-MM 中心の方針への切り替えを開始しています。結果として、ユーザが知っていて愛用している関数の多くが標準空間型 (ST) プレフィクスを使うように名前変更されました。以前の関数はまだ有効ですが、更新された等価な関数があるものについては、この文書の一覧から外しています (訳注: 非推奨関数は PostGIS 2.0 では基本的に外れています)。これらの関数は非推奨であり、将来のリリースでは削除されますので、* 使わないでください *。

7.1 PostGIS Geometry/Geography/Box データ型

7.1.1 box2d

box2d — 2 次元バウンディングボックスを表現する型。

説明

box2d は、ジオメトリまたはジオメトリコレクションの、2 次元の囲い込んでいるボックスを表現するために使われる空間データ型です。たとえば、集約関数 **ST_Extent** は box2d インスタンスを返します。

xmin, ymin, xmax, ymax の値を含む表現。これらは、X と Y の範囲の最小値と最大値を示しています。

box2d のテキスト表現は BOX(1 2,5 6) のようになります。

キャストの挙動

このテーブルでは、このデータ型で許容される明示的なキャストと自動キャストの一覧を挙げます。

キャスト先	ふるまい
box3d	自動
geometry	自動

関連情報

Section 13.7

7.1.2 box3d

box3d — 3 次元バウンディングボックスを表現する型。

説明

box3d は、ジオメトリまたはジオメトリのコレクションを囲む 3 次元のボックスを表現するために使われる PostGIS 空間データ型です。たとえば、集約関数の **ST_3DExtent** は box3d オブジェクトを返します。

この表現は、xmin, ymin, zmin, xmax, ymax, zmax です。これらは、X, Y, Z の範囲の最小値と最大値を取ります。

box3d のテキスト表現は BOX3D(1 2 3,5 6 5) のようになります。

キャストの挙動

このテーブルでは、このデータ型で許容される明示的なキャストと自動キャストの一覧を挙げます。

キャスト先	ふるまい
box	自動
box2d	自動
geometry	自動

関連情報

Section **13.7**

7.1.3 geometry

geometry — 平面座標系を持つ空間地物を表現する型。

説明

geometry は、平面 (ユークリッド) 座標系上の地物を表現するために使われる基本的な PostGIS の空間データ型です。

ジオメトリ上の全ての空間演算子は、ジオメトリが所属する空間参照系の単位を使います。

キャストの挙動

このテーブルでは、このデータ型で許容される明示的なキャストと自動キャストの一覧を挙げます。

キャスト先	ふるまい
box	自動
box2d	自動
box3d	自動
bytea	自動
geography	自動
text	自動

関連情報

Section 4.1, Section 13.3

7.1.4 geometry_dump

geometry_dump — 複雑なジオメトリの部品を記述するために使われる複合型です。

説明

geometry_dump は、次のフィールドを持つ複合型です。

- geom - ダンプされたジオメトリの要素を表現するジオメトリです。ジオメトリタイプは、使われた関数に依存します。
- path[] - ダンプされたジオメトリ内における geom 要素へのパスを定義する 1 次整数配列。パス配列は 1 始まりです (path[1] が最初の要素です)。

ST_Dump* 系関数で複雑なジオメトリを構成部品に分解する出力型として使います。

関連情報

Section 13.6

7.1.5 geography

geography — 地理座標系 (回転楕円体) 座標系を持つ空間地物を表現する型です。

説明

geography は、地理座標系上で地物を表現するために使われる空間型です。地理座標系は回転楕円体で地球をモデル化します。

ジオグラフィ型を用いた空間演算によって、回転楕円体モデルを考慮するので、より精度の良い結果が得られます。

キャストの挙動

このテーブルでは、このデータ型で許容される明示的なキャストと自動キャストの一覧を挙げます。

キャスト先	ふるまい
geometry	明示的なキャスト

関連情報

Section 4.3, Section 13.4

7.2 テーブル管理関数

7.2.1 AddGeometryColumn

AddGeometryColumn — ジオメトリカラムを既存のテーブルに追加します。

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);
```

説明

ジオメトリカラムを既存の属性テーブルに追加します。schema_name はスキーマ名です。srid は SPATIAL_REF_SYS テーブルのエントリを参照する整数でなければなりません。type は 'POLYGON' や 'MULTILINESTRING' といった、ジオメトリタイプを示す文字でなければなりません。指定したスキーマが存在しない (または現在の search_path からは見えない) 場合、または指定した SRID、ジオメトリタイプもしくは次元が不正である場合はエラーが投げられます。

Note



Changed: 2.0.0 geometry_columns がシステムカタログを読むビューになったため、geometry_columns を更新しないようになりました。デフォルトでは制約を生成せず、PostgreSQL の型修飾子を使います。この関数による WGS 84 の POINT カラムの構築と ALTER TABLE some_table ADD COLUMN geom geometry(Point,4326); とは等価です。
Changed: 2.0.0 制約を使う必要がある場合には、use_typmod を FALSE にします。

Note



Changed: 2.0.0 ビューについては、geometry_columns への手動登録はできなくなりました。しかし、typmod テーブルジオメトリに対して構築されていて、かつラップ関数が無いビューは、親テーブルカラムの typmod の挙動を継承するので、正しく登録されます。他のジオメトリを出力するジオメトリ関数を使うビューについては、ビューのジオメトリカラムが正しく登録されるようにするため、typmod ジオメトリへのキャストが必要です。Section 4.6.3 を参照して下さい。

- ✓ このメソッドは **OGC Simple Features Implementation Specification for SQL 1.1** の実装です。
- ✓ この関数は 3 次元に対応し、Z 値を削除しません。
- ✓ このメソッドは曲線ストリングと曲線に対応しています。

Enhanced: 2.0.0 use_typmod 引数が導入されました。デフォルトでは制約を基にしたものでなく typmod ジオメトリカラムが生成されます。

例

This example creates a schema to hold data.

```
CREATE SCHEMA my_schema;
```

Create a simple PostgreSQL table in the schema.

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

Describing the table shows a single id column.

```
\d 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 column using the old constraint-based behavior.

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

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

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

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

Describe the table again to see the new geometry columns.

```
\d 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)
```

The geometry_columns view also registers the new columns.

```
SELECT f_geometry_column As col_name, type, srid, coord_dimension As ndims
FROM geometry_columns
WHERE f_table_name = 'my_spatial_table' AND f_table_schema = 'my_schema';
```

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

関連情報

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

7.2.2 DropGeometryColumn

DropGeometryColumn — ジオメトリカラムを空間テーブルから除去します。

Synopsis

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

説明

ジオメトリカラムを空間テーブルから除去します。schema_name は geometry_columns テーブルの該当行の f_table_schema フィールドと一致しなければならないことにご注意ください。

- ✔ このメソッドは [OGC Simple Features Implementation Specification for SQL 1.1](#) の実装です。
- ✔ この関数は 3 次元に対応し、Z 値を削除しません。
- ✔ このメソッドは曲線ストリングと曲線に対応しています。



Note

Changed: 2.0.0 この関数は後方互換のためのものです。geometry_columns は現在はシステムカタログに対するビューですので、他のテーブルのカラムと同じように ALTER TABLE を使った削除が可能です。

例

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

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

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

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

関連情報

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

7.2.3 DropGeometryTable

DropGeometryTable — テーブルと geometry_columns の当該テーブルへの参照の全てを削除します。

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);
```

説明

テーブルと geometry_columns の当該テーブルへの参照の全てを削除します。スキーマ対応版 PostgreSQL ではスキーマが与えられない場合は current_schema() を使います。



Note

Changed: 2.0.0 でこの関数は後方互換のためのものです。geometry_columns は現在はシステムカタログに対するビューですので、他のテーブルのカラムと同じように DROP TABLE を使った削除が可能です。

例

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

```
my_schema.my_spatial_table dropped.
```

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

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

関連情報

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

7.2.4 Find_SRID

Find_SRID — ジオメトリカラムで定義されている SRID を返します。

Synopsis

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

説明

指定したジオメトリカラム SRID 整数値を GEOMETRY_COLUMNS テーブルの探索によって返します。ジオメトリカラムが正しく追加されていない (例: [AddGeometryColumn](#)関数) 場合には、この関数は動作しません。

例

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

```
4269
```

関連情報

[ST_SRID](#)

7.2.5 Populate_Geometry_Columns

Populate_Geometry_Columns — ジオメトリカラムが型修飾子で定義されるか、適切な空間制約を持つようにします。

Synopsis

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

説明

ジオメトリカラムが適切な型修飾子を持つか、`geometry_columns` ビュー内で正しく登録されていることを確実にするために空間制約を持つようにします。デフォルトでは、型修飾子を持たないすべてのジオメトリカラムを型修飾子を持つカラムに変換します。

後方互換のためと、それぞれの子テーブルが異なるジオメトリタイプを持つテーブル継承といった空間テーブルにとって必要があるためとの二つの理由から、古い CHECK 制約の挙動がなお有効になっています。古い挙動が必要な場合には、新しいオプション引数で `use_typmod=false` を渡す必要があります。これが実行されると、型修飾子なしのジオメトリカラムが生成され、三つの制約が定義されます。特に、これは、テーブルに属するすべてのジオメトリカラムが少なくとも三つの制約を持つことを意味します:

- `enforce_dims_the_geom` - あらゆるジオメトリが同じ次元を持つことを確実にします ([ST_NDims](#)をご覧ください)
- `enforce_geotype_the_geom` - あらゆるジオメトリが同じ型を持つことを確実にします ([GeometryType](#)をご覧ください)
- `enforce_srid_the_geom` - あらゆるジオメトリが同じ投影法になることを確実にします ([ST_SRID](#)をご覧ください)

テーブルに `oid` がある場合には、この関数はテーブルのジオメトリカラム全てについて、SRID と次元とジオメトリタイプを判定して、必要に応じて制約を追加しようとします。成功した場合には、`geometry_columns` に適切な行が追加され、その他の場合には、例外が捕まえられ、問題を記述したエラーが通知されます。

ビューの `oid` がある場合、テーブルの場合と同じで、SRID と次元とジオメトリタイプを判定して、適切なエントリを `geometry_columns` テーブルに挿入しますが、制約の追加はされません。

パラメーターの無い形式は、`geometry_columns` の行を削除したうえで、全ての空間テーブルと空間ビューについて再挿入し、適切な空間制約をテーブルに追加する、パラメーター付きの形式の単純なラッパーです。パラメーターが無い形式は、検出したジオメトリカラムの数の要約と `geometry_columns` に挿入された行の数とを返します。パラメーター付きの形式は単純に `geometry_columns` に挿入された行の数を返します。

Availability: 1.4.0

Changed: 2.0.0 デフォルトでは、ジオメトリタイプの制限について、制約を確認する代わりに型修飾子を使います。新しい `use_typmod` を `FALSE` に設定して使うことで、制約確認を使用することができます。

Enhanced: 2.0.0 `use_typmod` 任意引数が導入されました。カラムが型修飾子で生成されるか制約チェックで作られるかの制御ができます。

例

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

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

1

\d myspatial_table

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

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

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

1

\d myspatial_table_cs

Table "public.myspatial_table_cs"			
Column	Type		Modifiers
gid	integer	not null default nextval('myspatial_table_cs_gid_seq'::regclass)	
geom	geometry		

Check constraints:

"enforce_dims_geom" CHECK (st_ndims(geom) = 2)

"enforce_geotype_geom" CHECK (geometrytype(geom) = 'LINESTRING'::text OR geom IS NULL)

"enforce_srid_geom" CHECK (st_srid(geom) = 4326)

7.2.6 UpdateGeometrySRID

UpdateGeometrySRID — ジオメトリカラム内の全ての地物の SRID を更新し、テーブルのメタデータを更新します。

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);
```

説明

ジオメトリカラム内の全ての地物の SRID を更新し、制約を更新し、`geometry_columns` の参照を更新します。カラムが型定義で強制されているなら、型定義は変更されます。ご注意: スキーマ対応版 PostgreSQL では、スキーマが提供されていない場合には、`current_schema()` を使用します。



この関数は 3 次元に対応し、Z 値を削除しません。



このメソッドは曲線ストリングと曲線に対応しています。

例

ジオメトリを、**EWKT 書式**を使って、SRID を持つ道路テーブルに挿入します:

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

これにより道路テーブルが、以前がどんな SRID であっても、4326 に変更されます:

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

上述の例と、次の DDL 手続き (訳注: DDL は Data Definition Language の略で、データ構造の操作を行う言語を指し、この場合は CREATE TABLE や ALTER TABLE 等が該当します) とは同じです:

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

ロードしたデータの変換座標系が誤りである (または unknown になっている) けれども Web メルカトルに一度の処理で変換したい場合、DDL で実行可能ですが、PostGIS 管理関数では一度の処理ですむ等価なものはありません。

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

関連情報

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

7.3 ジオメトリコンストラクタ

7.3.1 ST_Collect

ST_Collect — ジオメトリの集合からジオメトリコレクションまたはマルチ系ジオメトリを生成します。

Synopsis

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

説明

ジオメトリを集めてジオメトリコレクションにします。結果はマルチ系ジオメトリかジオメトリコレクションかのいずれかで、この差は、入力ジオメトリのタイプが同じか異なるか (均質か不均質か) で決まります。入力ジオメトリはコレクション内で変更されることはありません。

- 1 番目の形式: 二つの入力ジオメトリを受け付ける。
- 2 番目の形式: ジオメトリの配列を受け付ける。
- 3 番目の形式: ジオメトリの行集合を受け付ける集約関数。

Note



入力ジオメトリのいずれかがコレクション (マルチ系ジオメトリまたはジオメトリコレクション) の場合には、**ST_Collect** はジオメトリコレクションを返します (入れ子になったコレクションを含む唯一のタイプであるため)。これを避けるには、サブクエリで**ST_Dump** を使い、入力コレクションを分解できない要素にまで分解します (下に例があります)。

Note



ST_Collect と**ST_Union**は似ているように見えますが、実際には全く異なる処理を行います。**ST_Collect** は入力ジオメトリを変更せずにコレクションにする集約関数です。**ST_Union** は、オーバーラップしている時は幾何学的に併合し、インタセクトするところでラインストリングを分割します。境界をディゾルブするときには単一のジオメトリを返す可能性があります。

Availability: 1.4.0 - **ST_Collect(geometry)** が導入されました。**ST_Collect** がより多くのジオメトリをより早く扱えるよう強化されました。



この関数は 3 次元に対応し、Z 値を削除しません。



このメソッドは曲線ストリングと曲線に対応しています。

例 - 二つ入力を引数に取る形式

2 次元ポイントを収集します。

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

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

2 次元ポイントの収集

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

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

曲線を収集します。

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

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

例 - 配列を引数に取る形式

サブクエリから配列を生成するコンストラクタの使用。

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

値から配列を生成するコンストラクタの使用。

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

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

例 - 集約関数の形式

テーブル内のジオメトリのグループ化による複数コレクションを生成します。

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

関連情報

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

7.3.2 ST_LineFromMultiPoint

ST_LineFromMultiPoint — マルチポイントジオメトリからラインストリングを生成します。

Synopsis

geometry **ST_LineFromMultiPoint**(geometry aMultiPoint);

説明

マルチポイントジオメトリからラインストリングを生成します。

ポイントまたはラインストリングの入力からラインを生成するには[ST_MakeLine](#)を使います。



この関数は 3 次元に対応し、Z 値を削除しません。

例

マルチポイントジオメトリからラインストリングを生成します。

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

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

関連情報

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

7.3.3 ST_MakeEnvelope

ST_MakeEnvelope — 座標値の最小値と最大値から矩形ポリゴンを生成します。

Synopsis

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

説明

X と Y の最小値と最大値から矩形ポリゴンを生成します。入力値は SRID で指定された空間参照系に合わせなければなりません。SRID が指定されていない場合には、不明な空間参照系 (SRID 0) が使われます。

Availability: 1.5

Enhanced: 2.0 SRID 指定なしでエンベロープを指定できるようになりました。

例: バウンディングボックスポリゴンの生成

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

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

関連情報

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

7.3.4 ST_MakeLine

ST_MakeLine — POINT、MULTIPOINT、LINESTRING から LINESTRING を生成します。

Synopsis

geometry **ST_MakeLine**(geometry geom1, geometry geom2);

geometry **ST_MakeLine**(geometry[] geoms_array);

geometry **ST_MakeLine**(geometry set geoms);

説明

ポイント、マルチポイントまたはラインストリングのジオメトリの点を含むラインストリングを生成します。他のジオメトリではエラーが発生します。

1 番目の形式: 二つの入力ジオメトリを受け付ける。

2 番目の形式: ジオメトリの配列を受け付ける。

形式 3: ジオメトリの行集合を受け付ける約関数。入力ジオメトリの順序を確実にするには、関数呼び出しで **ORDER BY** を使うか、**ORDER BY** 節を持つサブクエリを使います。

入力ラインストリングの開始位置で重複するノードはひとつのポイントに減らされます。ポイントとマルチポイントの入力での重複するポイントは減らされません。マルチラインストリングの要素は、コレクション内の出現順で処理されます。出力ラインストリングから重複ポイントを削除するには **ST_RemoveRepeatedPoints** が使えます。



この関数は 3 次元に対応し、Z 値を削除しません。

Availability: 3.7.0 - マルチラインストリング入力要素への対応が導入されました

Availability: 2.3.0 - MULTIPOINT 入力要素への対応が導入されました

Availability: 2.0.0 - LINESTRING 入力要素への対応が導入されました

Availability: 1.4.0 - **ST_MakeLine(geomarray)** が導入されました。**ST_MakeLine** 集約関数はより多くのポイントをより早く扱うための強化が施されています。

例: 二つ入力を引数に取る形式

二つのポイントで構成されるラインを生成します。

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

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

二つの 3 次元ポイントから 3 次元ラインを生成します。

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

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

二つの接続されていないラインストリングからラインを生成します。

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

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

例: 配列を引数に取る形式

並べ替えを伴うサブクエリで作られた配列からラインを生成します。

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

3 次元ポイントの配列から 3 次元ラインを生成します。

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

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

例: 集約関数の形式

この例では GPS トラックの集合からポイントの時間ベースのシーケンスを問い合わせています。結果ジオメトリは、GPS トラックの移動順ポイントで構成されるラインストリングです。

ORDER BY 節を使うことで、正しい順序の LINESTRING が生成できます。

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

集約内での整列ができない時は、サブクエリーで整列させます。ただし、クエリープランにサブクエリーの整列が反映されない場合があります。

```
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;
```

関連情報

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

7.3.5 ST_MakePoint

ST_MakePoint — 2 次元、3 次元 (XYZ)、4 次元のポイントを生成します。

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);
```

説明

XY 2 次元、XYZ 3 次元、XYZM 4 次元のポイントジオメトリを生成します。XYM 座標のポイントを生成するには [ST_MakePointM](#) を使用します。

生成したポイントの SRID を指定するには [ST_SetSRID](#) を使います。

OGC 準拠ではありませんが、ST_MakePoint は [ST_GeomFromText](#) や [ST_PointFromText](#) より高速です。また、数値による座標値を使用できます。

**Note**

地理座標系について、X は経度で、Y は緯度です。

**Note**

[ST_Point](#), [ST_PointZ](#), [ST_PointM](#), [ST_PointZM](#) 関数を使用すると、与えられた SRID を持つポイントを生成することができます。



この関数は 3 次元に対応し、Z 値を削除しません。

例

This example creates a point with an unknown SRID.

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

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

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

This example creates a 3D point with altitude.

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

This example returns the Z coordinate of a point.

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

```
1.5
```

関連情報

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

7.3.6 ST_MakePointM

ST_MakePointM — X, Y, M 値からポイントを生成します。

Synopsis

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

説明

X, Y, M 座標値を持つポイントを生成します。XY, XYZ, XYZM 座標のポイントを生成するには [ST_MakePoint](#) を使います。

生成したポイントの SRID を指定するには [ST_SetSRID](#) を使います。



Note

地理座標系について、X は経度で、Y は緯度です。



Note

[ST_PointM](#), [ST_PointZM](#)関数を使うと、与えられた SRID を持ち、M 値を持つポイントを生成することができます。

例

**Note**

ST_AsEWKT は文字列出力のために使います。**ST_AsText**が M 値に対応していないためです。

不明な SRID でポイントを生成します。

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

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

WGS 84 地理座標系の M 値を持つポイントの生成。

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

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

生成したポイントの M 値を取得します。

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

```
10
```

関連情報

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

7.3.7 ST_MakePolygon

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

Synopsis

geometry **ST_MakePolygon**(geometry linestring);

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

説明

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

Variant 1: Accepts one shell ring.

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

**Note**

この関数はマルチラインストリングを受け付けません。ラインストリングの生成には**ST_LineMerge**を使用します。また、ラインストリングを抽出するには**ST_Dump**を使用します。



この関数は 3 次元に対応し、Z 値を削除しません。

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

例: 単一入力の形式

2 次元ラインストリングからポリゴンを生成します。

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

開いたラインストリングを閉じるために **ST_StartPoint** と **ST_AddPoint** を使用したうえでポリゴンを生成します。

```
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;
```

3 次元ラインストリングからポリゴンを生成します。

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

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

M 値を持つラインストリングからポリゴンを生成します。

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

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

Create a CurvePolygon from a CircularString shell.

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

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

例: 内部の穴を持つ外殻の形式

余分な穴を持つドーナツポリゴンを生成します。

```
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;
```

湖を表現する穴を持つ県の境界の集合を生成します。入力はあるポリゴン/マルチポリゴンのテーブルと水涯線のラインストリングのテーブルです。湖を構成するラインは **ST_IsClosed** で判定します。県を示すラインは **ST_Boundary** で抽出します。**ST_MakePolygon** が必要ですので、境界を **ST_LineMerge** で単一の **LINESTRING** に強制します (県が二つ以上の領域を持つ島を持つ場合には、不正なポリゴンを生成します)。LEFT JOIN を使って、湖の無い県を含む全ての県が返ることを保証しています。



Note

NULL 配列を **ST_MakePolygon** に渡すと NULL が返るので、CASE 式を使っています。

```

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;

```

他には、関連サブクエリと行集合を配列に変換する `ARRAY()` コンストラクタとを使う手法があります。

```

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;

```

関連情報

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

7.3.8 ST_Point

`ST_Point` — X, Y と SRID の値からポイントを生成します。

Synopsis

```

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

```

説明

与えられた X と Y の座標値からポイントを返します。これは、X と Y を取る SQL-MM の [ST_MakePoint](#) と同等です。



Note

地理座標系について、X は経度で、Y は緯度です。

Enhanced: 3.2.0 SRID 任意引数が追加されました。古いバージョンでは、ジオメトリに SRID を与えるには `ST_SetSRID` を併用しなければなりませんでした。



このメソッドは SQL/MM 仕様の実装です。SQL-MM 3: 6.1.2

例: ジオメトリ

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

指定した SRID を持つポイントの生成:

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

SRID を指定する別の方法:

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

例: ジオグラフィ

ジオグラフィ のポイント、キャストの書き方 `::` を使って生成します:

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

PostGIS 3.2 より前のコードでは `CAST` を使います:

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

ポイントの座標が地理座標系 (WGS84 等) でない場合には、ジオグラフィにキャストする前に座標系変換を行う必要があります。この例では、ペンシルバニア州平面フィート (SRID 2273) 上のポイントを WGS84 (SRID 4326) に座標系変換を行っています。

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

関連情報

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

7.3.9 ST_PointZ

`ST_PointZ` — X, Y, Z と SRID の値からポイントを生成します。

Synopsis

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

説明

与えた X,Y,Z 座標値を持ち、与えた場合は SRID 値も持つポイントを生成します。

Enhanced: 3.2.0 SRID 任意引数が追加されました。古いバージョンでは、ジオメトリに SRID を与えるには `ST_SetSRID` を併用しなければなりませんでした。

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)
```

関連情報

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

7.3.10 ST_PointM

ST_PointM — X, Y, M と SRID の値からポイントを生成します。

Synopsis

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

説明

与えた X,Y,M 座標値を持ち、与えた場合は SRID 値も持つポイントを生成します。

Enhanced: 3.2.0 SRID 任意引数が追加されました。古いバージョンでは、ジオメトリに SRID を与えるには ST_SetSRID を併用しなければなりませんでした。

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)
```

関連情報

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

7.3.11 ST_PointZM

ST_PointZM — X, Y, Z, M と SRID の値からポイントを生成します。

Synopsis

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

説明

与えた X,Y,Z,M 座標値を持ち、与えた場合は SRID 値も持つポイントを生成します。

Enhanced: 3.2.0 SRID 任意引数が追加されました。古いバージョンでは、ジオメトリに SRID を与えるには ST_SetSRID を併用しなければなりませんでした。

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)
```

関連情報

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

7.3.12 ST_Polygon

ST_Polygon — ラインストリングから指定した SRID を持つポリゴンを生成します。

Synopsis

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

説明

与えられたラインストリングから構築し、srid から空間参照系を指定したポリゴンを返します。

ST_Polygon は[ST_MakePolygon](#)の形式 1 の、SRID の設定を追加したものに似ています。

穴を持つポリゴンを生成するには、[ST_MakePolygon](#)の形式 2 を使い、[ST_SetSRID](#)を使います。



Note

この関数はマルチラインストリングを受け付けません。ラインストリングの生成には[ST_LineMerge](#)を使用します。また、ラインストリングを抽出するには[ST_Dump](#)を使用します。

- ✔ このメソッドは[OGC Simple Features Implementation Specification for SQL 1.1](#)の実装です。
- ✔ このメソッドは SQL/MM 仕様の実装です。SQL-MM 3: 8.3.2
- ✔ この関数は 3 次元に対応し、Z 値を削除しません。

例

2 次元ポリゴンを生成します。

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

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

3 次元ポリゴンを生成します。

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

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

関連情報

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

7.3.13 ST_TileEnvelope

`ST_TileEnvelope` — [Webメルカトル](#) (SRID:3857) 上で [XYZ タイル](#) を使った矩形ポリゴンを生成します。

Synopsis

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

説明

[XYZ tile system](#)でのタイルの範囲を示す長方形ポリゴンを生成します。タイルはズームレベルを示す `Z` と、そのレベルでのタイルインデックスを示す `X, Y` で指定します。[ST_AsMVTGeom](#)でジオメトリを `MVT` タイルの座標空間に変換のために必要なタイル境界を定義するために使います。

デフォルトでは、タイルエンベロープの座標系は[Web Mercator](#) (SRID: 3857) で、標準的なウェブメルカトルの範囲 (-20037508.342789, 20037508.342789) を取ります。これが `MVT` タイルで使用される最も標準的な座標系です。任意パラメータ `bounds` を使うと、あらゆる座標系のタイルを生成することができます。SRID を持ち、ズームレベル 0 の時の、ジオメトリの `XYZ` タイルシステムが内側に存在する矩形を示すジオメトリを与えます。

任意パラメータ `margin` は、タイルを与えられた割合だけ拡張するために使います。たとえば `margin=0.125` と指定すると、12.5% 拡張します。これは、[ST_AsMVTGeom](#)で使われている、タイルサイズが 4096 の時に `buffer=512` と指定するのと等価です。タイル表示領域の外側にバッファを生成するために使いますが、この範囲はタイル描画に影響を与えます。たとえば、市名 (ポイント) を一つのタイルの端付近にあっても、そのラベルが二つのタイルで描画されるべきです。拡張したタイルをクエリで使うと、両方のタイルに市名のポイントが含まれることになります。負数を指定すると反対に縮小します。`-0.5` より小さい値は、タイルを完全に消すことになるので、許可されません。`ST_AsMVTGeom` と併用する時はマージンを指定しないでください。[ST_AsMVT](#)の例をご覧ください。

Enhanced: 3.1.0 `margin` パラメータが追加されました。

Availability: 3.0.0

例: タイルエンベロープの構築

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

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

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

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

関連情報

[ST_MakeEnvelope](#)

7.3.14 ST_HexagonGrid

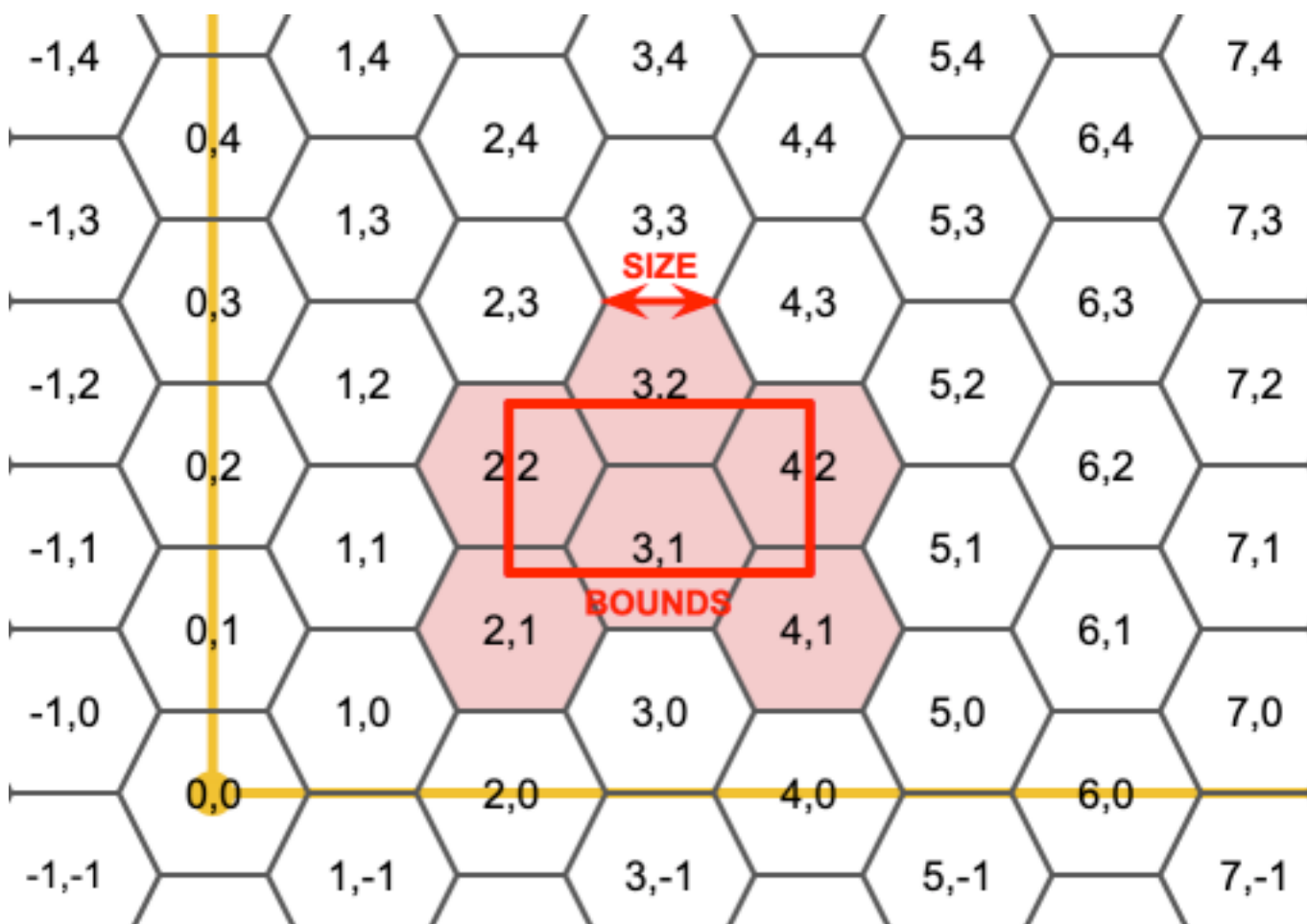
ST_HexagonGrid — 引数ジオメトリの境界を完全にカバーする六角形とセルインデックスを返します。

Synopsis

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

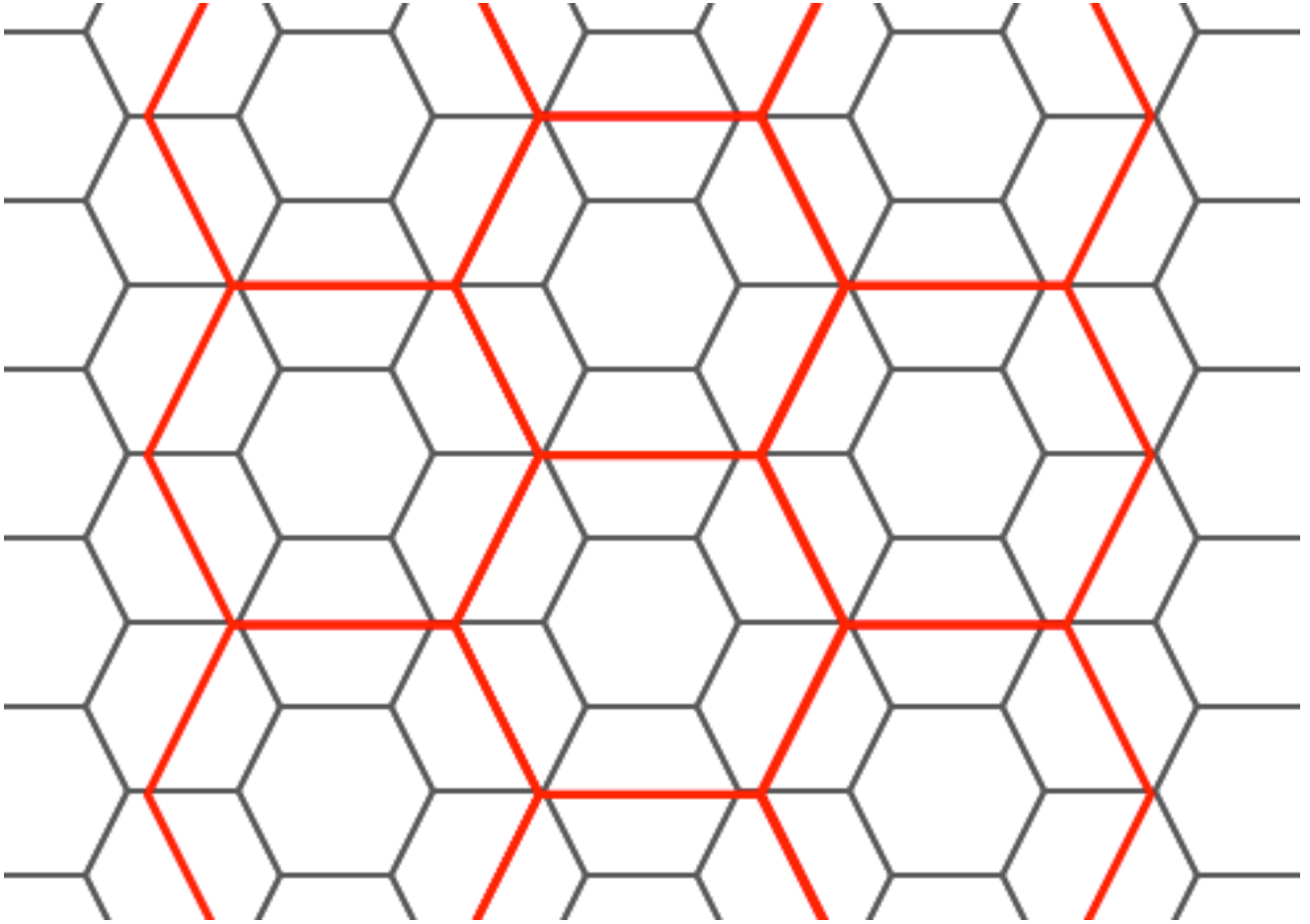
説明

平面の六角形タイルの概念から始まります。(地球の六角形タイルではなくH3タイルスキーマではありません)。平面 SRS とエッジサイズを与えると、SRS の原点から始まり、平面の一意な六角形のタイル、すなわち Tiling(SRS, Size) が一つ存在します。この関数は、指定された Tiling(SRS, Size) 内の六角形が指定された境界を覆っているかどうかという質問に答えます。



出力六角形の SRS は境界ジオメトリの SRS です。

六角形の辺の長さを 2 倍または 3 倍にすることで、元のタイルに適合する新しい親タイルが生成されます。残念ながら子タイルが完全に親タイルの中に入るような親六角形タイルを生成することはできません。



Availability: 3.1.0

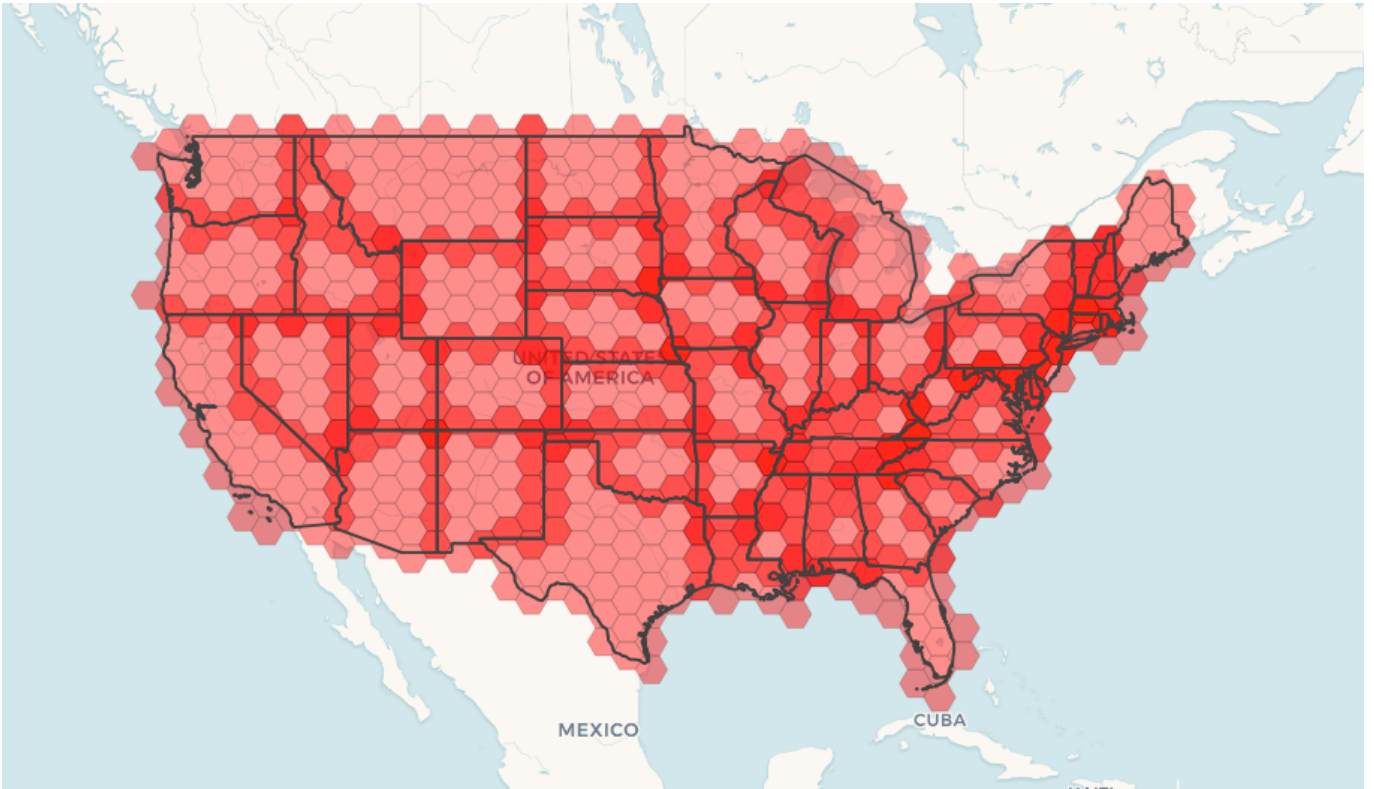
例: 六角形の中の点を数え上げる

六角形タイリングに対してポイントをまとめるには、ポイントの範囲を境界として使って六角形グリッドを生成し、グリッドに空間的に結合します。

```
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;
```

例: ポリゴンの六角形カバレッジの生成

ポリゴン境界ごとに六角形の集合を生成し、六角形とインタセクトしないものを除外すると、ポリゴンごとのタイルとなります。



州のタイルは、それぞれの州の六角形のカバレッジとなり、複数の六角形が州境界で重なります。

**Note**

LATERAL キーワードは、FROM リスト内の、対象より前のテーブルを参照する時は、集合を返す関数に暗黙的に含まれます。CROSS JOIN LATERAL、CROSS JOIN、指定なし、は、この例の同じ構成要素です。

```
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)
```

関連情報

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

7.3.15 ST_Hexagon

ST_Hexagon — 与えられたエッジサイズと六角形グリッド空間内のセル座標を使って単一の六角形を返します。

Synopsis

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

説明

ST_HexagonGridと同じ六角形タイルの概念を使いますが、求めるセルの座標に一つだけの六角形を生成します。任意でタイルの原点の座標を調整できます。デフォルトの原点座標は 0,0 です。

六角形は SRID の設定なしで生成されるので、SRID を期待する値に設定するために**ST_SetSRID**を使います。

Availability: 3.1.0

例: 原点で六角形の生成

```
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))
```

関連情報

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

7.3.16 ST_SquareGrid

ST_SquareGrid — 引数ジオメトリの境界を完全にカバーするグリッド正方形とセルインデックスを返します。

Synopsis

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

説明

平面の正方形タイルの概念から始まります。与えられた平面 SRS とエッジサイズに対して、SRS 原点から始まり、一意の平面の正方形タイル (SRS, Size) が一つ存在します。この関数は、与えられたタイル (SRS, Size) 内のグリッド内のどのタイルが与えられた境界とオーバラップするかという問題に答えを出します。

出力正方形の SRS は境界ジオメトリの SRS です。

正方形の 2 倍またはエッジサイズによって、新しい親タイルが生成されます。親タイルは完全に元のタイルに適合します。標準のウェブマップにおけるメルカトルのタイルは、メルカトル平面の 2 の累乗での正方形です。

Availability: 3.1.0

例: 国の 1 度グリッドの生成

グリッドは国の境界全体を埋めます。国に接触する四角形が欲しい場合には、**ST_Intersects** 後にフィルタリングする必要があります。

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

例: 正方形内のポイントの数え上げ (細かくした単一のグリッドを使用)

正方形タイルのポイントのサマリを行うには、境界としてポイントの拡張を使った正方形グリッドを生成し、グリッドに空間的に結合します。推定範囲は実際の範囲と異なる場合がありますのでご注意ください。慎重に取り扱うようにし、最低でも確実にテーブルを解析して下さい。

```
SELECT COUNT(*), squares.geom
FROM
  pointtable AS pts
  INNER JOIN
  ST_SquareGrid(
    1000,
    ST_SetSRID(ST_EstimatedExtent('pointtable', 'geom'), 3857)
  ) AS squares
ON ST_Intersects(pts.geom, squares.geom)
GROUP BY squares.geom
```

例: ポイントごとのグリッドの集合を使った正方形内のポイント数え上げ

これは最初の例と同じ結果になりますが、ポイント数が多くなると遅くなります。

```
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
```

関連情報

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

7.3.17 ST_Square

ST_Square — 与えられたエッジサイズと六角形グリッド空間内のセル座標を使って単一の正方形を返します。

Synopsis

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

説明

[ST_SquareGrid](#)と同じ正方形タイルの概念を使っていますが、求めるセルの座標に一つだけの正方形を生成します。任意でタイルの原点の座標を調整できます。デフォルトの原点座標は 0,0 です。

与えられた原点の SRID で正方形が生成されます。与える原点の SRID が不明 (デフォルトのまま) の場合には[ST_SetSRID](#)を使って SRID を設定します。

Availability: 3.1.0

例: 原点で四角形の生成

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

関連情報

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

7.3.18 ST_Letters

ST_Letters — デフォルトの開始位置を原点とし、デフォルトの高さを 100 とする、ジオメトリとして描画された文字を返します。

Synopsis

geometry **ST_Letters**(text letters, json font);

説明

組込みフォントを使って、出力文字列をマルチポリゴンとして描画します。ディセンダからキャピタルまでの文字高さは 100.0 です。デフォルトのベースラインの開始位置は原点に置かれます。フォントのオーバーライドは文字をキーとした JSON マッピングと、ディセンダからキャピタルまでの 1000 単位の高さを持つフォント形状の TWKB を base64 エンコードしたものを渡します。

テキストは、デフォルトでは原点に生成されるので、テキストの位置変更とサイズ変更とを行います。最初に `ST_Scale` 関数を適用し、その後 `ST_Translate` 関数を適用します。

Availability: 3.3.0

例: 単語 'Yo' の生成

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



ST_Letter で生成した文字

例: 単語の拡大と移動

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

関連情報

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

7.3.19 ST_MakeNurbsCurve

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

Synopsis

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

説明

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

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

The `control_points` must be a `LINESTRING` geometry defining the control polygon. The number of control points must be at least `degree + 1`.

The optional `weights` array assigns a weight to each control point. All weights must be positive. If `NULL` or omitted, uniform weights (1.0) are used, resulting in a non-rational B-spline.

The optional `knots` array defines the parameter space. If `NULL` or omitted, a uniform clamped knot vector is automatically generated. If provided, the knot vector must have exactly `(npoints + degree + 1)` elements and must be non-decreasing.

Availability: 3.7.0

例

This example creates a degree-2 NURBS curve with uniform weights.

```
SELECT ST_MakeNurbsCurve(2, 'LINESTRING(0 0, 5 10, 10 0)::geometry);
```

This example creates a rational NURBS curve with custom weights.

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

This example creates a NURBS curve with custom weights and knots.

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

関連情報

[ST_NurbsToLineString](#), [ST_Evaluate](#), [ST_ControlPoints](#)

7.4 ジオメトリアクセサ

7.4.1 GeometryType

GeometryType — ジオメトリのタイプを文字列で返します。

Synopsis

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

説明

ジオメトリ型を 'LINESTRING', 'POLYGON', 'MULTIPOINT' などの文字列で返します。

OGC SPEC s2.1.1.1 - このジオメトリインスタンスがメンバーになっているジオメトリのインスタンス化可能な派生タイプの名前を返します。インスタンス化可能な派生タイプの名前は、文字列として返されます。



Note

この関数は、'POINTM' 等が返るので、ジオメトリが M 値を持っているかどうかを示します。

Enhanced: 2.0.0 多面体サーフェス対応、三角対応、TIN 対応が導入されました。

- ✓ このメソッドは [OGC Simple Features Implementation Specification for SQL 1.1](#) の実装です。
- ✓ このメソッドは曲線ストリングと曲線に対応しています。
- ✓ この関数は 3 次元に対応し、Z 値を削除しません。
- ✓ この関数は多面体サーフェスに対応しています。
- ✓ この関数は三角形と不規則三角網 (TIN) に対応しています。

例

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

LINESTRING

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

```

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

```

TIN

関連情報

[ST_GeometryType](#)

7.4.2 ST_Boundary

ST_Boundary — ジオメトリの境界を返します。

Synopsis

geometry **ST_Boundary**(geometry geomA);

説明

ジオメトリの組み合わせ境界の閉包を返します (訳注: ラインストリングは端点、ポリゴンはエッジ、複合オブジェクトは境界のうち奇数番)。組み合わせ境界は OGC 仕様の 3.12.3.2 節に記述されています。結果として出てくる境界は、OGC SPEC 3.12.2 で議論されているように、ジオメトリプリミティブを使って表現できます。

GEOS モジュールで実現しています。



Note

2.0.0 より前の版では、この関数は GEOMETRYCOLLECTION. を与えると例外を投げました。2.0.0 以上では代わりに NULL が返ります (非対応入力)。

✔ このメソッドは [OGC Simple Features Implementation Specification for SQL 1.1](#) の実装です。OGC SPEC s2.1.1.1

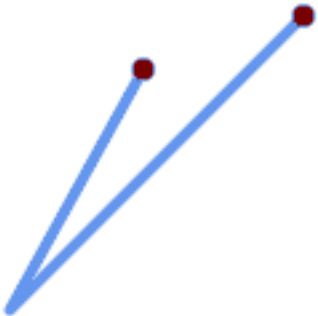
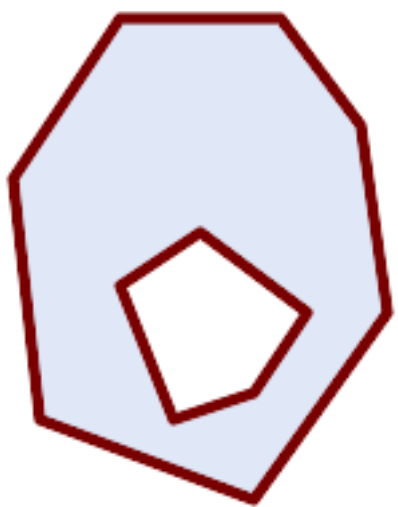
✔ このメソッドは SQL/MM 仕様の実装です。SQL-MM IEC 13249-3: 5.1.17

✔ この関数は 3 次元に対応し、Z 値を削除しません。

Enhanced: 2.1.0 三角対応が導入されました。

Changed: 3.2.0 TIN に対応しました。GEOS を使いません。曲線を線形化しません。

例

	
境界ポイントを重ねたラインストリング <pre>SELECT ST_AsText(ST_Boundary(geom)) FROM (SELECT 'LINESTRING(100 150,50 60, ← 70 80, 160 170) '::geometry As geom) As f; MULTIPOINT((100 150),(160 170))</pre>	境界マルチラインストリングを重ねたポリゴンの穴 <pre>SELECT ST_AsText(ST_Boundary(geom)) FROM (SELECT 'POLYGON ((10 130, 50 190, 110 190, 140 ← 150, 150 80, 100 10, 20 40, 10 130), ← (70 40, 100 50, 120 80, 80 110, ← 50 90, 70 40))'::geometry As geom) As f; 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)')));
```

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

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

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

This example uses a 3D polygon.

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

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

This example uses a 3D MultiLineString.

```
SELECT ST_AsEWKT(ST_Boundary(ST_GeomFromEWKT('MULTILINESTRING((1 1 1,0 0 0.5, -1 1 1),(1 1 ←
0.5,0 0 0.5, -1 1 0.5, 1 1 0.5) ))'));
```

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


関連情報

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

7.4.3 ST_BoundingDiagonal

ST_BoundingDiagonal — ジオメトリのバウンディングボックスの対角線を返します。

Synopsis

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

説明

与えられたジオメトリのバウンディングボックスの対角線をラインストリングで返します。最小値のポイントを始点とし、最大値のポイントを終点とする、二つのポイントからなるラインストリングになります。入力ジオメトリが空の場合には、対角線は **LINestring EMPTY** となります。

fits パラメータは、最良適合が必要かどうかを指定するものです。FALSE の場合には、幾分大きめなバウンディングボックスの対角線を受け付けることができます (多数の頂点からなるジオメトリの取得が早くなります)。いずれにしても返された対角線のバウンディングボックスは常に入力ジオメトリを含みます。

返されるラインストリングは常に、入力ジオメトリの **SRID** と次元 (**Z** と **M** があること) を維持します。

**Note**

縮退した (入力の頂点が一つ) 場合、返されるラインストリングは形式的に不正です (内部が無い)。トポロジ的には妥当です。

Availability: 2.2.0



この関数は 3 次元に対応し、Z 値を削除しません。



この関数は M 値に対応します。

例

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

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

```
-10
```

関連情報

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

7.4.4 ST_CoordDim

ST_CoordDim — ジオメトリの座標次元を返します。

Synopsis

```
integer ST_CoordDim(geometry geomA);
```

説明

ST_Geometry 値の座標次元を返します。

この関数は **ST_NDims** の MM 対応の別名です。

-  このメソッドは **OGC Simple Features Implementation Specification for SQL 1.1** の実装です。
-  このメソッドは SQL/MM 仕様の実装です。SQL-MM 3: 5.1.3
-  このメソッドは曲線ストリングと曲線に対応しています。
-  この関数は 3 次元に対応し、Z 値を削除しません。
-  この関数は多面体サーフェスに対応しています。
-  この関数は三角形と不規則三角網 (TIN) に対応しています。

例

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

```
3
```

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

```
2
```

関連情報

ST_NDims

7.4.5 ST_Dimension

ST_Dimension — ST_Geometry 値の座標次元を返します。

Synopsis

```
integer ST_Dimension(geometry g);
```

説明

ジオメトリの固有次元を返します。ジオメトリは座標次元以下でなければなりません。OGC SPEC s2.1.1.1 - 0 なら POINT、1 なら LINESTRING、2 なら POLYGON で、GEOMETRYCOLLECTION の場合は要素ごとの次元の最大値です。不明なジオメトリ (空の GEOMETRYCOLLECTION 等) の場合は NULL が返ります。



このメソッドは SQL/MM 仕様の実装です。SQL-MM 3: 5.1.2

Enhanced: 2.0.0 多面体サーフェス対応と TIN 対応が導入されました。空ジオメトリを与えた場合に例外を投げなくなりました。



Note

2.0.0 より前では、空ジオメトリを与えると例外を投げていました。



この関数は多面体サーフェスに対応しています。



この関数は三角形と不規則三角網 (TIN) に対応しています。

例

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

```
1
```

関連情報

[ST_NDims](#)

7.4.6 ST_Dump

ST_Dump — ジオメトリの要素となる geometry_dump 行の集合を返します。

Synopsis

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

説明

ジオメトリ要素を抽出する、集合を返す関数 (SRF=Set-Returning Function) です。[geometry_dump](#) 行の集合を返します。各要素はジオメトリ (*geom* フィールド) と整数配列 (*path* フィールド) からなります。

非マルチ系ジオメトリタイプ (POINT,LINESTRING,POLYGON) では、*path* 配列が空で *geom* が入力ジオメトリと同じになる単一の行が返ります。コレクションまたはマルチ系ジオメトリでは、個々の要素と、コレクションの要素位置を示す *path* とからなる行を返します。

ST_Dump はジオメトリを展開するのに使います。[ST_Collect](#)/GROUP BY の逆で、この関数の中で新行を作成します。たとえば、MULTIPOLYGON を POLYGON に展開するために使います。

Enhanced: 2.0.0 多面体サーフェス対応、三角対応、TIN 対応が導入されました。

Availability: PostGIS 1.0.0RC1. PostgreSQL 7.3 以上が必要です。

**Note**

1.3.4 より前では、曲線を含むジオメトリで使用すると、この関数はクラッシュします。これは 1.3.4 以上で訂正されています。

- ✔ このメソッドは曲線ストリングと曲線に対応しています。
- ✔ この関数は多面体サーフェスに対応しています。
- ✔ この関数は三角形と不規則三角網 (TIN) に対応しています。
- ✔ この関数は 3 次元に対応し、Z 値を削除しません。

標準的な例

```
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)

多面体サーフェス、**TIN**、三角形の例

This example breaks a PolyhedralSurface into its faces.

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

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

This example uses a TIN geometry.

```

SELECT (g.gdump).path, ST_AsEWKT((g.gdump).geom) as wkt
FROM
  (SELECT
    ST_Dump(ST_GeomFromEWKT('TIN (((
      0 0 0,
      0 0 1,
      0 1 0,
      0 0 0
    )), ((
      0 0 0,
      0 1 0,
      1 1 0,
      0 0 0
    ))
    )') ) AS gdump
  ) AS g;

```

path	wkt
{1}	TRIANGLE((0 0 0,0 0 1,0 1 0,0 0 0))
{2}	TRIANGLE((0 0 0,0 1 0,1 1 0,0 0 0))

関連情報

[geometry_dump](#), [Section 13.6](#), [ST_Collect](#), [ST_GeometryN](#)

7.4.7 ST_DumpPoints

ST_DumpPoints — ジオメトリ内の座標の行である `geometry_dump` 行の集合を返します。

Synopsis

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

説明

ジオメトリの座標 (頂点) を抽出する、集合を返す関数 (SRF=Set-Returning Function) です。[geometry_dump](#) 行の集合を返します。各要素はジオメトリ (*geom* フィールド) と整数配列 (*path* フィールド) からなります。

- *geom* フィールドには、与えられたジオメトリの座標を表現する `POINT` が入ります。
- *path* フィールド (`integer[]`) は、与えられたジオメトリの要素内の座標位置を列挙するインデックスです。インデックスは 1 始まりです。たとえば、`LINESTRING` に対しては、`LINESTRING` の *n* 番目の座標を *i* とすると、`{i}` となります。`POLYGON` に対しては、*i* を環番号 (1 が外環、続いて内環)、*j* を環の座標位置とすると、`{i,j}` となります。

座業を含む単一ジオメトリを取得するには [ST_Points](#) を使います。

Enhanced: 2.1.0 速度向上しました。C 言語で実装しなおしました。

Enhanced: 2.0.0 多面体サーフェス対応、三角対応、TIN 対応が導入されました。

Availability: 1.5.0

- ✔ このメソッドは曲線ストリングと曲線に対応しています。
- ✔ この関数は多面体サーフェスに対応しています。
- ✔ この関数は三角形と不規則三角網 (TIN) に対応しています。
- ✔ この関数は 3 次元に対応し、Z 値を削除しません。

ラインストリングのテーブルのノードへの古典的な分割

```
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)

標準的な例



```
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)

多面体サーフェス、**TIN**、三角形の例

This example uses a cube represented as a PolyhedralSurface.

```

SELECT (g.gdump).path, ST_AsEWKT((g.gdump).geom) as wkt
FROM
  (SELECT
    ST_DumpPoints(ST_GeomFromEWKT('POLYHEDRALSURFACE( ((0 0 0, 0 0 1, 0 1 1, 0 1 0, 0 0 ←
    0)),
    ((0 0 0, 0 1 0, 1 1 0, 1 0 0, 0 0 0)), ((0 0 0, 1 0 0, 1 0 1, 0 0 1, 0 0 0)),
    ((1 1 0, 1 1 1, 1 0 1, 1 0 0, 1 1 0)),
    ((0 1 0, 0 1 1, 1 1 1, 1 1 0, 0 1 0)), ((0 0 1, 1 0 1, 1 1 1, 0 1 1, 0 0 1)) )' ) AS gdump
  ) AS g;

```

path	wkt
{1,1,1}	POINT(0 0 0)
{1,1,2}	POINT(0 0 1)
{1,1,3}	POINT(0 1 1)
{1,1,4}	POINT(0 1 0)
{1,1,5}	POINT(0 0 0)
{2,1,1}	POINT(0 0 0)
{2,1,2}	POINT(0 1 0)
{2,1,3}	POINT(1 1 0)
{2,1,4}	POINT(1 0 0)
{2,1,5}	POINT(0 0 0)
{3,1,1}	POINT(0 0 0)
{3,1,2}	POINT(1 0 0)
{3,1,3}	POINT(1 0 1)
{3,1,4}	POINT(0 0 1)
{3,1,5}	POINT(0 0 0)
{4,1,1}	POINT(1 1 0)
{4,1,2}	POINT(1 1 1)
{4,1,3}	POINT(1 0 1)
{4,1,4}	POINT(1 0 0)
{4,1,5}	POINT(1 1 0)
{5,1,1}	POINT(0 1 0)
{5,1,2}	POINT(0 1 1)
{5,1,3}	POINT(1 1 1)
{5,1,4}	POINT(1 1 0)
{5,1,5}	POINT(0 1 0)
{6,1,1}	POINT(0 0 1)
{6,1,2}	POINT(1 0 1)
{6,1,3}	POINT(1 1 1)
{6,1,4}	POINT(0 1 1)
{6,1,5}	POINT(0 0 1)

(30 rows)

This example uses a Triangle geometry.

```
SELECT (g.gdump).path, ST_AsText((g.gdump).geom) as wkt
FROM
  (SELECT
    ST_DumpPoints(ST_GeomFromEWKT('TRIANGLE ((
      0 0,
      0 9,
      9 0,
      0 0
    ))')) AS gdump
  ) AS g;
```

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

This example uses a TIN geometry.

```
SELECT (g.gdump).path, ST_AsEWKT((g.gdump).geom) as wkt
FROM
  (SELECT
    ST_DumpPoints(ST_GeomFromEWKT('TIN (((
      0 0 0,
```



```

        0 0 1,
        0 1 0,
        0 0 0
    )), ((
        0 0 0,
        0 1 0,
        1 1 0,
        0 0 0
    ))
  )' ) AS gdump
) AS g;

```

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

(8 rows)

関連情報

[geometry_dump](#), [Section 13.6](#), [ST_Dump](#), [ST_DumpRings](#), [ST_Points](#)

7.4.8 ST_DumpSegments

`ST_DumpSegments` — ジオメトリ内の辺の行である `geometry_dump` 行の集合を返します。

Synopsis

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

説明

ジオメトリの辺を抽出する、集合を返す関数 (SRF=Set-Returning Function) です。[geometry_dump](#) 行の集合を返します。各要素はジオメトリ (*geom* フィールド) と整数配列 (*path* フィールド) からなります。

- *geom* フィールドの `LINESTRING` は与えられたジオメトリの線分を表現し、`CIRCULARSTRING` は円弧線分を表現します。
- *path* フィールド (`integer[]`) は、与えられたジオメトリの要素内の辺の始点位置を列挙するインデックスです。インデックスは 1 始まりです。たとえば、`LINESTRING` に対しては、`LINESTRING` の *n* 番目の辺を *i* とすると、`{i}` となります。`POLYGON` に対しては、*i* を環番号 (1 が外環、続いて内環)、*j* を環の辺の始点位置とすると、`{i,j}` となります。

Availability: 3.2.0



この関数は三角形と不規則三角網 (TIN) に対応しています。



この関数は 3 次元に対応し、Z 値を削除しません。

標準的な例

```
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、三角形の例

This example uses a Triangle geometry.

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

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

(3 rows)

This example uses a TIN geometry.

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

```

    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)

関連情報

[geometry_dump](#), [Section 13.6](#), [ST_Dump](#), [ST_DumpRings](#)

7.4.9 ST_DumpRings

`ST_DumpRings` — ポリゴンのリングごとの `geometry_dump` 行の集合を返します。

Synopsis

```
geometry_dump[] ST_DumpRings(geometry a_polygon);
```

説明

ジオメトリ要素を抽出する、集合を返す関数 (SRF=Set-Returning Function) です。[geometry_dump](#) 行の集合を返します。各要素はジオメトリ (*geom* フィールド) と整数配列 (*path* フィールド) からなります。

geom フィールドは、各々のリングを POLYGON として持ちます。*path* フィールドはリングのインデックス番号を持つ配列長 1 の整数配列です。外側リング (外周) はインデックス番号が 0 番になります。内側リング (穴) は 1 以上になります。



Note

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

Availability: PostGIS 1.1.3 PostgreSQL 7.3 以上が必要です。



この関数は 3 次元に対応し、Z 値を削除しません。

例

全てのリングのポリゴンとしての抽出。

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

ポリゴンから外殻と穴の抽出。

```
SELECT path, ST_AsText(geom) As geom
FROM ST_DumpRings('POLYGON ((1 9, 9 9, 9 1, 1 1, 1 9), (2 2, 2 3, 3 3, 3 2, 2 2), ←
(4 2, 4 4, 6 4, 6 2, 4 2))');
```

path	geom
{0}	POLYGON((1 9,9 9,9 1,1 1,1 9))
{1}	POLYGON((2 2,2 3,3 3,3 2,2 2))
{2}	POLYGON((4 2,4 4,6 4,6 2,4 2))

関連情報

[geometry_dump](#), Section [13.6](#), [ST_Dump](#), [ST_ExteriorRing](#), [ST_InteriorRingN](#)

7.4.10 ST_EndPoint

ST_EndPoint — Returns the last point of a LineString, CircularLineString, or NURBSCurve.

Synopsis

geometry **ST_EndPoint**(geometry g);

説明

Returns the last point of a LINESTRING, CIRCULARLINESTRING, or NURBSCURVE geometry as a POINT. Returns NULL if the input is not a LINESTRING, CIRCULARLINESTRING, or NURBSCURVE.

- ✓ このメソッドは SQL/MM 仕様の実装です。SQL-MM 3: 7.1.4
- ✓ この関数は 3 次元に対応し、Z 値を削除しません。
- ✓ このメソッドは曲線ストリングと曲線に対応しています。

Note



Changed: 2.0.0 一つのジオメトリマルチラインストリングで動作しなくなりました。PostGIS の古いバージョンでは、この関数は一つのマルチラインストリングで動作し、終端ポイントを返します。2.0.0 では、他のマルチラインストリングと同様に NULL を返します。古い動作は文書化されていない機能でしたが、データを LINESTRING として格納していると思われるユーザーは、2.0.0 で NULL が返されることを経験するかも知れません。

例

ラインストリングの終端ポイント

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

```
POINT(3 3)
```

ラインストリング以外の終端ポイントは NULL

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

```
t
```

3次元ラインストリングの終端ポイント

3D endpoint.

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

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

CIRCULARSTRING の終端ポイント

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

```
POINT(6 3)
```

End point of a NURBSCurve

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

```
POINT(2 0)
```

関連情報

[ST_PointN](#), [ST_StartPoint](#)

7.4.11 ST_Envelope

ST_Envelope — ジオメトリのバウンディングボックスを表現するジオメトリを返します。

Synopsis

geometry **ST_Envelope**(geometry g1);

説明

与えられたジオメトリの倍精度浮動小数点数 (float8) の最小バウンディングボックスをジオメトリで返します。ポリゴンはバウンディングボックスの角のポイントで定義されます ((MINX, MINY), (MINX, MAXY), (MAXX, MAXY), (MAXX, MINY), (MINX, MINY))。 (PostGIS は ZMIN/ZMAX も追加します)。

縮退する場合 (縦のライン、ポイント) は POLYGON より低い次元のジオメトリ、すなわち POINT または LINESTRING を返します。

Availability: 1.5.0 挙動が変更され出力が float4 から float8 になりました。

- ✔ このメソッドは [OGC Simple Features Implementation Specification for SQL 1.1](#) の実装です。s2.1.1.1
- ✔ このメソッドは SQL/MM 仕様の実装です。SQL-MM 3: 5.1.19

例

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

```
POINT(1 3)
```

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

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

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

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

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

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

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



ポイントとラインストリングの最小バウンディングボックス

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

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

関連情報

[Box2D](#), [Box3D](#), [ST_OrientedEnvelope](#)

7.4.12 ST_ExteriorRing

ST_ExteriorRing — ポリゴンの外環を表現するラインストリングを返します。

Synopsis

geometry **ST_ExteriorRing**(geometry a_polygon);

説明

POLYGON の外環を表現する LINESTRING を返します。ジオメトリがポリゴンでない場合は NULL を返します。



Note

この関数はマルチポリゴンでは動作しません。マルチポリゴンに対しては[ST_GeometryN](#)または[ST_Dump](#)を併用して下さい。

- ✔ このメソッドは[OGC Simple Features Implementation Specification for SQL 1.1](#)の実装です。2.1.5.1
- ✔ このメソッドは SQL/MM 仕様の実装です。SQL-MM 3: 8.2.3, 8.3.3
- ✔ この関数は 3 次元に対応し、Z 値を削除しません。

例

This example extracts exterior rings from a table of polygons.

```
SELECT gid, ST_ExteriorRing(geom) AS ering
FROM sometable;
```

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

```
GROUP BY gid;
```

```
--3d Example
```

```
SELECT ST_AsEWKT(ST_ExteriorRing(
    ST_GeomFromEWKT('POLYGON((0 0 1, 1 1 1, 1 2 1, 1 1 1, 0 0 1))')
));

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

関連情報

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

7.4.13 ST_GeometryN

ST_GeometryN — ジオメトリコレクションの要素を一つ返します。

Synopsis

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

説明

入力ジオメトリが GEOMETRYCOLLECTION, MULTIPOINT, MULTILINESTRING, MULTICURVE, MULTIPOLYGON, POLYHEDRALSURFACE の場合には、1 始まりで N 番目の要素であるジオメトリを返します。これ以外の場合には NULL を返します。



Note

OGC 仕様のため 0.8.0 版からインデックスを 1 始まりにしています。これより前の版では 0 始まりになっています。



Note

ジオメトリの全ての要素を抽出するには [ST_Dump](#) の方が効率的ですし、単一ジオメトリでも動作します。

Enhanced: 2.0.0 多面体サーフェス対応、三角対応、TIN 対応が導入されました。

Changed: 2.0.0 以前の版では非マルチのジオメトリでは NULL が返りました。ST_GeometryN(...,1) の場合にはジオメトリを返すよう変更されました。



このメソッドは [OGC Simple Features Implementation Specification for SQL 1.1](#) の実装です。



このメソッドは SQL/MM 仕様の実装です。SQL-MM 3: 9.1.5



この関数は 3 次元に対応し、Z 値を削除しません。



このメソッドは曲線ストリングと曲線に対応しています。



この関数は多面体サーフェスに対応しています。



この関数は三角形と不規則三角網 (TIN) に対応しています。

標準的な例

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);
```

多面体サーフェス、**TIN**、三角形の例

This example breaks a PolyhedralSurface into its faces.

```
SELECT ST_AsEWKT(ST_GeometryN(p_geom,3)) As geom_ewkt
FROM (SELECT ST_GeomFromEWKT('POLYHEDRALSURFACE(
((0 0 0, 0 0 1, 0 1 1, 0 1 0, 0 0 0)),
((0 0 0, 0 1 0, 1 1 0, 1 0 0, 0 0 0)),
((0 0 0, 1 0 0, 1 0 1, 0 0 1, 0 0 0)),
((1 1 0, 1 1 1, 1 0 1, 1 0 0, 1 1 0)),
((0 1 0, 0 1 1, 1 1 1, 1 1 0, 0 1 0)),
((0 0 1, 1 0 1, 1 1 1, 0 1 1, 0 0 1))
)') AS p_geom ) AS a;
```

Output:

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

This example uses a TIN geometry.

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

```

```

    ))
  ) AS geom
) AS g;

```

Output:

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

関連情報

[ST_Dump](#), [ST_NumGeometries](#)

7.4.14 ST_GeometryType

ST_GeometryType — ジオメトリの SQL-MM 型を文字列で返します。

Synopsis

```
text ST_GeometryType(geometry g1);
```

説明

ジオメトリ型を 'ST_LineString', 'ST_Polygon', 'ST_MultiPolygon' 等の文字列で返します。この関数は `GeometryType(geometry)` とは異なり、先頭に 'ST' が付き、M 値を持っているかを示しません。

Enhanced: 2.0.0 多面体サーフェス対応が導入されました。

-  このメソッドは SQL/MM 仕様の実装です。SQL-MM 3: 5.1.4
-  この関数は 3 次元に対応し、Z 値を削除しません。
-  この関数は多面体サーフェスに対応しています。

例

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

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

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

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

ST_Tin

関連情報

[GeometryType](#)

7.4.15 ST_HasArc

ST_HasArc — ジオメトリに円弧が含まれているかどうかテストします。

Synopsis

boolean **ST_HasArc**(geometry geomA);

説明

ジオメトリまたはジオメトリコレクションに曲線ラインストリングが含まれている場合に TRUE を返します。

Availability: 1.2.3?

-  この関数は 3 次元に対応し、Z 値を削除しません。
-  このメソッドは曲線ストリングと曲線に対応しています。

例

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

t

関連情報

[ST_CurveToLine](#), [ST_LineToCurve](#)

7.4.16 ST_InteriorRingN

ST_InteriorRingN — ポリゴンの N 番目の内環 (穴) を返します。

Synopsis

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

説明

ポリゴンの N 番目の内環を返します。ジオメトリがポリゴンでないか N が範囲外の場合は NULL を返します。



Note

この関数はマルチポリゴンでは動作しません。マルチポリゴンに対しては[ST_GeometryN](#)または[ST_Dump](#)を併用して下さい。

- ✔ このメソッドは[OGC Simple Features Implementation Specification for SQL 1.1](#)の実装です。
- ✔ このメソッドは SQL/MM 仕様の実装です。SQL-MM 3: 8.2.6, 8.3.5
- ✔ この関数は 3 次元に対応し、Z 値を削除しません。

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;
```

関連情報

[ST_ExteriorRing](#), [ST_BuildArea](#), [ST_Collect](#), [ST_Dump](#), [ST_NumInteriorRing](#), [ST_NumInteriorRings](#)

7.4.17 ST_NumCurves

ST_NumCurves — 複合曲線内の曲線の数进行返します。

Synopsis

integer **ST_NumCurves**(geometry a_compoundcurve);

説明

複合曲線内の曲線の数进行返します。入力が空の複合曲線の場合には 0 を返し、複合曲線でない場合には NULL を返します。



このメソッドは SQL/MM 仕様の実装です。SQL-MM 3: 8.2.6, 8.3.5



この関数は 3 次元に対応し、Z 値を削除しません。

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');
```

関連情報

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

7.4.18 ST_CurveN

ST_CurveN — 複合曲線の N 番目の曲線ジオメトリを返します。

Synopsis

geometry **ST_CurveN**(geometry a_compoundcurve, integer index);

説明

複合曲線の N 番目の曲線ジオメトリを返します。インデックスは 1 始まりです。ジオメトリが複合曲線でない場合やインデックスが範囲外の場合には NULL を返します。



このメソッドは SQL/MM 仕様の実装です。SQL-MM 3: 8.2.6, 8.3.5



この関数は 3 次元に対応し、Z 値を削除しません。

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));
```

関連情報

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

7.4.19 ST_IsClosed

ST_IsClosed — ラインストリングの始点と終点が一致しているかをテストします。多面体サーフェスについては閉じているか (立体であるか) をテストします。

Synopsis

boolean **ST_IsClosed**(geometry g);

説明

LINESTRING の始点と終点が一致する場合に **TRUE** を返します。多面体サーフェスについては、サーフェスが面 (開いている) か立体 (閉じている) かをテストします。



このメソッドは [OGC Simple Features Implementation Specification for SQL 1.1](#) の実装です。



このメソッドは SQL/MM 仕様の実装です。SQL-MM 3: 7.1.5, 9.3.3



Note

SQL-MM では **ST_IsClosed(NULL)** の結果は 0 と定義されていますが、PostGIS では **NULL** を返します。



この関数は 3 次元に対応し、Z 値を削除しません。



このメソッドは曲線ストリングと曲線に対応しています。

Enhanced: 2.0.0 多面体サーフェス対応が導入されました。



この関数は多面体サーフェスに対応しています。

ラインストリングとポイントの例

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

```
f
```

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

```
t
```

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

```
f
```

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

```
t
```

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

```
t
```

多面体サーフェスの例

This example uses a cube.

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

```
t
```

This example uses the same cube with one side missing.

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

```
f
```

関連情報

[ST_IsRing](#)

7.4.20 ST_IsCollection

ST_IsCollection — ジオメトリのタイプがジオメトリコレクションかをテストします。

Synopsis

```
boolean ST_IsCollection(geometry g);
```

説明

ジオメトリのタイプがジオメトリコレクションである場合に **TRUE** を返します。コレクションは次の通りです。

- ジオメトリコレクション
- マルチポイント、マルチポリゴン、マルチラインストリング、マルチ曲線、マルチサーフェス
- 複合曲線



Note

この関数はジオメトリのタイプを解析します。これは、空のコレクションである場合、または一つのエレメントを持つコレクションである場合には **TRUE** を返すことを意味します。



この関数は 3 次元に対応し、Z 値を削除しません。



このメソッドは曲線ストリングと曲線に対応しています。

例

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

```
f
```

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

```
t
```

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

```
t
```

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

```
t
```

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

```
t
```


関連情報

[ST_NumGeometries](#)

7.4.21 ST_IsEmpty

ST_IsEmpty — ジオメトリが空かをテストします。

Synopsis

boolean **ST_IsEmpty**(geometry geomA);

説明

ジオメトリが空ジオメトリの場合に **true** を返します。true の場合には、このジオメトリは、空のジオメトリコレクション、ポリゴン、ポイント等です。



Note

SQL-MM では、ST_IsEmpty(NULL) は 0 を返しますが、PostGIS では NULL を返します。

- ✔ このメソッドは[OGC Simple Features Implementation Specification for SQL 1.1](#)の実装です。s2.1.1.1
- ✔ このメソッドは SQL/MM 仕様の実装です。SQL-MM 3: 5.1.7
- ✔ このメソッドは曲線ストリングと曲線に対応しています。



Warning

Changed: 2.0.0 以前の版の PostGIS では ST_GeomFromText('GEOMETRYCOLLECTION(EMPTY)') を許しました。PostGIS 2.0.0 では、SQL/MM 標準により準拠させるため、これは不正となります。

例

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

```
t
```

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

```
t
```

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

```
f
```

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

```
t
(1 row)
```

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

```
t
```

7.4.22 ST_IsPolygonCCW

ST_IsPolygonCCW — ポリゴンが反時計回りの外環を持っていて、時計回りの内環を持っているかをテストします。

Synopsis

boolean **ST_IsPolygonCCW** (geometry geom);

説明

入力ジオメトリの全てのポリゴン要素の外環については反時計回りで、全ての内環については時計回りである場合には、TRUE を返します。

ジオメトリがポリゴン要素を持っていない場合には TRUE を返します。



Note

閉じたラインストリングはポリゴン要素とみなされません。単一の閉じたラインストリングを渡すと、右回り左回りにかかわらず TRUE が得られます。



Note

ポリゴン要素の内環が逆回りになっていない (すなわち外環と同じ方向で回る内環が 1 個以上ある) 場合には、ST_IsPolygonCW と ST_IsPolygonCCW の両方ともに FALSE を返します。

Availability: 2.4.0



この関数は 3 次元に対応し、Z 値を削除しません。



この関数は M 値に対応します。

関連情報

[ST_ForcePolygonCW](#) , [ST_ForcePolygonCCW](#) , [ST_IsPolygonCW](#)

7.4.23 ST_IsPolygonCW

ST_IsPolygonCW — ポリゴンが時計回りの外環を持っていて、反時計回りの内環を持っているかをテストします。

Synopsis

boolean **ST_IsPolygonCW** (geometry geom);

説明

入力ジオメトリの全てのポリゴン要素の外環については時計回りで、全ての内環については反時計回りである場合には、TRUE を返します。

ジオメトリがポリゴン要素を持っていない場合には TRUE を返します。



Note

閉じたラインストリングはポリゴン要素とみなされません。単一の閉じたラインストリングを渡すと、右回り左回りにかかわらず TRUE が得られます。



Note

ポリゴン要素の内環が逆回りになっていない (すなわち外環と同じ方向で回る内環が 1 個以上ある) 場合には、ST_IsPolygonCW と ST_IsPolygonCCW の両方ともに FALSE を返します。

Availability: 2.4.0



この関数は 3 次元に対応し、Z 値を削除しません。



この関数は M 値に対応します。

関連情報

[ST_ForcePolygonCW](#) , [ST_ForcePolygonCCW](#) , [ST_IsPolygonCW](#)

7.4.24 ST_IsRing

ST_IsRing — ラインストリングが閉じていてかつ単純であるかをテストします。

Synopsis

boolean **ST_IsRing**(geometry g);

説明

Returns TRUE if this LINESTRING is both [ST_IsClosed](#) (ST_StartPoint(g) ~= ST_EndPoint(g)) and [ST_IsSimple](#) (does not self intersect).



このメソッドは[OGC Simple Features Implementation Specification for SQL 1.1](#)の実装です。2.1.5.1



このメソッドは SQL/MM 仕様の実装です。SQL-MM 3: 7.1.6



Note

SQL-MM では ST_IsRing(NULL) の結果は 0 と定義されていますが、PostGIS では NULL を返します。

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)
```

関連情報

[ST_IsClosed](#), [ST_IsSimple](#), [ST_StartPoint](#), [ST_EndPoint](#)

7.4.25 ST_IsSimple

ST_IsSimple — ジオメトリが自己インタセクトまたは自己接触となるポイントが無いかをテストします。

Synopsis

boolean **ST_IsSimple**(geometry geomA);

説明

ジオメトリが自己インタセクションや自己接触のような異常な幾何学ポイントを持っていない場合に TRUE を返します。OGC のジオメトリ単純性と妥当性の定義に関する詳細情報については「[ジオメトリの OpenGIS 準拠を確実にする](#)」をご覧ください。



Note

SQL-MM では、**ST_IsSimple(NULL)** は 0 を返しますが、PostGIS では NULL を返します。

- ✔ このメソッドは [OGC Simple Features Implementation Specification for SQL 1.1](#) の実装です。s2.1.1.1
- ✔ このメソッドは SQL/MM 仕様の実装です。SQL-MM 3: 5.1.8
- ✔ この関数は 3 次元に対応し、Z 値を削除しません。

例

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

f

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

f

関連情報

[ST_IsValid](#)

7.4.26 ST_M

ST_M — ポイントの M 値を返します。

Synopsis

float **ST_M**(geometry a_point);

説明

ポイントの M 座標値を返し、有効でないなら NULL を返します。入力はポイントでなければなりません。



Note

これは (いまだに)OGC 仕様に入っていないが、ポイント座標抽出関数のリストを完全にするために挙げています。

- ✔ このメソッドは [OGC Simple Features Implementation Specification for SQL 1.1](#) の実装です。
- ✔ このメソッドは SQL/MM 仕様の実装です。
- ✔ この関数は 3 次元に対応し、Z 値を削除しません。

例

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

4

関連情報

[ST_GeomFromEWKT](#), [ST_X](#), [ST_Y](#), [ST_Z](#)

7.4.27 ST_MemSize

ST_MemSize — ジオメトリが取るメモリ空間の合計を返します。

Synopsis

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

説明

ジオメトリが取るメモリ空間の合計をバイト単位で返します。

この関数は、PostgreSQL ビルトイン [データベースオブジェクト管理関数](#) の `pg_column_size`, `pg_size_pretty`, `pg_relation_size`, `pg_total_relation_size` を補完します。

Note



テーブルのバイト単位のサイズを与える `pg_relation_size` は `ST_MemSize` より小さいバイト数が返ります。これは `pg_relation_size` が TOAST 化されたテーブルの寄与を追加せず、TOAST テーブルに格納された大きなジオメトリを加えないためです。

`pg_total_relation_size` - テーブル、TOAST テーブル、インデックスを含みます。

`pg_column_size` は、ジオメトリがカラム内で取る領域がどれだけかを、圧縮を考慮して返します。そのため、`ST_MemSize` より小さくなる場合があります。

- ✔ この関数は 3 次元に対応し、Z 値を削除しません。
- ✔ このメソッドは曲線ストリングと曲線に対応しています。
- ✔ この関数は多面体サーフェスに対応しています。
- ✔ この関数は三角形と不規則三角網 (TIN) に対応しています。

Changed: 2.2.0 命名規則に従うために `ST_MemSize` に変更しました。

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 geomsizesize,
sum(ST_MemSize(geom))*1.00/pg_total_relation_size('public.neighborhoods')*100 As pergeom
FROM neighborhoods;
fulltable_size geomsizesize pergeom
-----
262144          96238          36.711883544921875000000
```

7.4.28 ST_NDims

ST_NDims — ST_Geometry 値の座標次元を返します。

Synopsis

integer **ST_NDims**(geometry g1);

説明

ジオメトリの座標次元返します。PostGIS では、2 - (X,Y), 3 - (X,Y,Z), (X,Y,M), 4 - (X,Y,Z,M) に対応しています。



この関数は 3 次元に対応し、Z 値を削除しません。

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
```

関連情報

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

7.4.29 ST_NPoints

ST_NPoints — ジオメトリのポイント (頂点) の数を返します。

Synopsis

integer **ST_NPoints**(geometry g1);

説明

ジオメトリのポイントの数を返します。全てのジオメトリに対して動作します。

Enhanced: 2.0.0 多面体サーフェス対応が導入されました。



Note

1.3.4 より前では、曲線を含むジオメトリで使用すると、この関数はクラッシュします。これは 1.3.4 以上で訂正されています。

- ✓ この関数は 3 次元に対応し、Z 値を削除しません。
- ✓ このメソッドは曲線ストリングと曲線に対応しています。
- ✓ この関数は多面体サーフェスに対応しています。

例

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

--Polygon in 3D space
SELECT ST_NPoints(ST_GeomFromEWKT('LINESTRING(77.29 29.07 1,77.42 29.26 0,77.27 29.31 -1,77.29 29.07 3)'));
4
```

関連情報

[ST_NumPoints](#)

7.4.30 ST_NRings

ST_NRings — ポリゴンジオメトリのリング数を返します。

Synopsis

integer **ST_NRings**(geometry geomA);

説明

ジオメトリがポリゴンまたはマルチポリゴンの場合、リング数を返します。NumInteriorRings と違い、外環も数えます。

- ✓ この関数は 3 次元に対応し、Z 値を削除しません。
- ✓ このメソッドは曲線ストリングと曲線に対応しています。

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 |          0
(1 row)
```

関連情報

[ST_NumInteriorRings](#)

7.4.31 ST_NumGeometries

ST_NumGeometries — ジオメトリコレクションの要素数を返します。

Synopsis

integer **ST_NumGeometries**(geometry geom);

説明

ジオメトリコレクション等 (GEOMETRYCOLLECTION や MULTI 系) の要素数を返します。空ジオメトリでなく、ジオメトリコレクション等でもないジオメトリに対しては 1 を返します。空ジオメトリに対しては 0 を返します。

Enhanced: 2.0.0 多面体サーフェス対応、三角対応、TIN 対応が導入されました。

Changed: 2.0.0 前の版では、ジオメトリがコレクション/マルチ系でない場合には NULL を返しました。2.0.0 以上では、POLYGON, LINESTRING, POINT といった単一ジオメトリについては 1 を返します。

-  このメソッドは SQL/MM 仕様の実装です。SQL-MM 3: 9.1.4
-  この関数は 3 次元に対応し、Z 値を削除しません。
-  この関数は多面体サーフェスに対応しています。
-  この関数は三角形と不規則三角網 (TIN) に対応しています。

例

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

```
SELECT ST_NumGeometries(ST_GeomFromText('LINESTRING(77.29 29.07,77.42 29.26,77.27 29.31,77.29 29.07)'));
1

--Geometry Collection Example - multis count as one geom in a collection
SELECT ST_NumGeometries(ST_GeomFromEWKT('GEOMETRYCOLLECTION(MULTIPOINT((-2 3),(-2 2)),
LINESTRING(5 5 ,10 10),
POLYGON((-7 4.2,-7.1 5,-7.1 4.3,-7 4.2)))'));
3
```

関連情報

[ST_GeometryN](#), [ST_Multi](#)

7.4.32 ST_NumInteriorRings

ST_NumInteriorRings — ポリゴンの内環 (穴) の数を返します。

Synopsis

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

説明

ポリゴンジオメトリの内環の数を返します。ジオメトリがポリゴンでない場合には、NULL を返します。



このメソッドは SQL/MM 仕様の実装です。SQL-MM 3: 8.2.5

Changed: 2.0.0 - 以前の版では、MULTIPOLYGON を渡して最初の POLYGON の内環の数を返すことができませんでした。

例

This example counts the interior rings of a polygon.

```
SELECT gid, field1, field2, ST_NumInteriorRings(geom) AS numholes
FROM sometable;

--If you have multipolygons
--And you want to know the total number of interior rings in the MULTIPOLYGON
SELECT gid, field1, field2, SUM(ST_NumInteriorRings(geom)) AS numholes
FROM (SELECT gid, field1, field2, (ST_Dump(geom)).geom As geom
      FROM sometable) As foo
GROUP BY gid, field1,field2;
```

関連情報

[ST_NumInteriorRing](#), [ST_InteriorRingN](#)

7.4.33 ST_NumInteriorRing

ST_NumInteriorRing — ポリゴンの内環 (穴) の数を返します。ST_NumInteriorRings の別名です。

Synopsis

```
integer ST_NumInteriorRing(geometry a_polygon);
```

関連情報

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

7.4.34 ST_NumPatches

ST_NumPatches — 多面体サーフェスのフェイス数を返します。多面体でないジオメトリの場合には NULL を返します。

Synopsis

```
integer ST_NumPatches(geometry g1);
```

説明

多面体サーフェスのフェイス数を返します。多面体でないジオメトリの場合には NULL を返します。ST_NumGeometries の別名で、MM の名前付けに対応するためのものです。MM 規約を気にしない場合は ST_NumGeometries の方が速いです。

Availability: 2.0.0

- ✔ この関数は 3 次元に対応し、Z 値を削除しません。
- ✔ このメソッドは [OGC Simple Features Implementation Specification for SQL 1.1](#) の実装です。
- ✔ このメソッドは SQL/MM 仕様の実装です。SQL-MM ISO/IEC 13249-3: 8.5
- ✔ この関数は多面体サーフェスに対応しています。

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) ←
                ) )''));
6
```

関連情報

[ST_GeomFromEWKT](#), [ST_NumGeometries](#)

7.4.35 ST_NumPoints

ST_NumPoints — ラインストリングまたは曲線ストリングのポイント数を返します。

Synopsis

```
integer ST_NumPoints(geometry g1);
```

説明

`ST_LineString` または `ST_CircularString` のポイント数を返します。1.4 より前は仕様通りにラインストリングにのみ対応していました。1.4 以上ではラインストリングだけでなく頂点数を返す `ST_NPoints` の別名です。多目的で多数のジオメトリタイプで動作する `ST_NPoints` を使うことを考えて下さい。

- ✔ このメソッドは **OGC Simple Features Implementation Specification for SQL 1.1** の実装です。
- ✔ このメソッドは SQL/MM 仕様の実装です。SQL-MM 3: 7.2.4

例

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

関連情報

`ST_NPoints`

7.4.36 ST_PatchN

`ST_PatchN` — 多面体サーフェスの N 番目のジオメトリ (フェイス) を返します。

Synopsis

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

説明

ジオメトリが `POLYHEDRALSURFACE` か `POLYHEDRALSURFACEM` の場合には、1 始まりで N 番目のジオメトリ (フェイス) を返します。それ以外の場合には、`NULL` を返します。多面体サーフェスを引数にとる `ST_GeometryN` と同じ答えが返ります。 `ST_GeometryN` の方が速いです。



Note

インデックスは 1 始まりです。



Note

ジオメトリの全ての要素を抽出するには **`ST_Dump`** が最も効率的です。

Availability: 2.0.0

- ✔ このメソッドは SQL/MM 仕様の実装です。SQL-MM ISO/IEC 13249-3: 8.5
- ✔ この関数は 3 次元に対応し、Z 値を削除しません。
- ✔ この関数は多面体サーフェスに対応しています。

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);
```

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

関連情報

[ST_AsEWKT](#), [ST_GeomFromEWKT](#), [ST_Dump](#), [ST_GeometryN](#), [ST_NumGeometries](#)

7.4.37 ST_PointN

ST_PointN — ジオメトリの最初のラインストリングまたは曲線ストリングの N 番目のポイントを返します。

Synopsis

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

説明

ラインストリングまたは曲線ストリングの N 番目の点を返します。負数はラインストリングの終端から逆方向に遡って数えます。-1 は終端を指します。ジオメトリにラインストリングが無い場合には、NULL を返します。



Note

OGC 仕様では 0.8.0 版から要素番号は 1 始まりです。逆方向要素番号 (負の要素番号) は OGC 仕様にはありません。前の版では 0 始まりで実装していました。



Note

マルチラインストリング内のラインストリングの N 番目のポイントを得るには、ST_Dump を併用します。

- ✓ このメソッドは [OGC Simple Features Implementation Specification for SQL 1.1](#) の実装です。
- ✓ このメソッドは SQL/MM 仕様の実装です。SQL-MM 3: 7.2.5, 7.3.5
- ✓ この関数は 3 次元に対応し、Z 値を削除しません。
- ✓ このメソッドは曲線ストリングと曲線に対応しています。

**Note**

Changed: 2.0.0 単一ジオメトリの MULTILINESTRING で動作しなくなりました。単一のラインストリングからなる MULTILINESTRING については幸運にも動いていて、最初のポイントを返していました。2.0.0 では他の MULTILINESTRING と同様に NULL を返すようになりました。
Changed: 2.3.0 : 負数インデックスが有効になりました (-1 は終端を指します)

例

This example extracts all POINT geometries from a LineString.

```
SELECT ST_AsText(ST_PointN(
    column1,
    generate_series(1, ST_NPoints(column1))
))
FROM ( VALUES ('LINESTRING(0 0, 1 1, 2 2)::geometry') ) AS foo;
```

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

This example uses a CircularString.

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

```
POINT(3 2)
```

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

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

関連情報

[ST_NPoints](#)

7.4.38 ST_Points

ST_Points — ジオメトリの全ての座標を含むマルチポイントを返します。

Synopsis

geometry **ST_Points**(geometry geom);

説明

ジオメトリの全ての座標を含むマルチポイントを返します。リングジオメトリの始点と終点を含む重複ポイントは保存されます (重複ポイントの削除が必要なら **ST_RemoveRepeatedPoints** を呼ぶと削除した結果が得られます)。親ジオメトリの個々の座標のポイントに関する情報を取得するには **ST_DumpPoints** を使います。

M 値と Z 値が存在する場合には保持されます。



このメソッドは曲線ストリングと曲線に対応しています。



この関数は 3 次元に対応し、Z 値を削除しません。

Availability: 2.3.0

例

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

関連情報

ST_RemoveRepeatedPoints, **ST_DumpPoints**

7.4.39 ST_StartPoint

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

Synopsis

geometry **ST_StartPoint**(geometry geomA);

説明

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



このメソッドは SQL/MM 仕様の実装です。SQL-MM 3: 7.1.3



この関数は 3 次元に対応し、Z 値を削除しません。



このメソッドは曲線ストリングと曲線に対応しています。

Note

Enhanced: 3.2.0 全てのジオメトリのポイントを返すようになりました。以前のバージョンではラインストリング以外では NULL を返していました。

Changed: 2.0.0 一つの MULTILINESTRING で動作しなくなりました。PostGIS の古いバージョンでは、この関数は、一つのラインストリングからなる MULTILINESTRING については幸運にも動いていて、始端ポイントを返していました。2.0.0 では他の MULTILINESTRING と同様に NULL を返すようになりました。古い動作は文書化されていない機能でしたが、データを LINESTRING として格納していると思われるユーザーは、2.0.0 で NULL が返されることを経験するかも知れません。



例

ラインストリングの始端ポイント

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

```
POINT(0 1)
```

Start point of a Point is the Point itself

```
SELECT ST_AsText(ST_StartPoint('POINT(0 1)::geometry));
```

```
POINT(0 1)
```

3次元ラインストリングの始端ポイント

```
SELECT ST_AsEWKT(ST_StartPoint('LINESTRING(0 1 1, 0 2 2)::geometry));
```

```
POINT(0 1 1)
```

CIRCULARSTRING の始端ポイント

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

```
POINT(5 2)
```

Start point of a NURBS Curve

```
SELECT ST_AsText(ST_StartPoint('NURBS CURVE(2, (0 0, 1 1, 2 0))::geometry));
```

```
POINT(0 0)
```

関連情報

[ST_EndPoint](#), [ST_PointN](#)

7.4.40 ST_StartM

ST_StartM — Returns the M coordinate of the first point of a LineString, CircularLineString, or NURBS Curve.

Synopsis

```
float8 ST_StartM(geometry geom);
```

説明

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



このメソッドは曲線ストリングと曲線に対応しています。

例

Start M value of a LineString

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

10

Start M value of a CircularString

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

5

Start M value of a NURBSCurve

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

100

Geometry without M dimension returns NULL

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

t

関連情報

[ST_EndM](#), [ST_SetStartM](#), [ST_StartPoint](#)

7.4.41 ST_EndM

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

Synopsis

float8 **ST_EndM**(geometry geom);

説明

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



このメソッドは曲線ストリングと曲線に対応しています。

例

End M value of a LineString

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

30

End M value of a CircularString

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

15

End M value of a NURBSCurve

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

300

Geometry without M dimension returns NULL

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

t

関連情報

[ST_StartM](#), [ST_SetEndM](#), [ST_EndPoint](#)

7.4.42 ST_SetStartM

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

Synopsis

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

説明

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



このメソッドは曲線ストリングと曲線に対応しています。

例

Set start M value of a LineString

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

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

Add M dimension and set start M value

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

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

Set start M value of a CircularString

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

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

Set start M value of a NURBSCurve

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

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

関連情報

[ST_SetEndM](#), [ST_StartM](#), [ST_StartPoint](#)

7.4.43 ST_SetEndM

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

Synopsis

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

説明

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



このメソッドは曲線ストリングと曲線に対応しています。

例

Set end M value of a LineString

```
SELECT ST_AsText(ST_SetEndM('LINESTRING M(0 0 10, 1 1 20, 2 0 30)::geometry, 999));
```

```
LINESTRING M (0 0 10,1 1 20,2 0 999)
```

Add M dimension and set end M value

```
SELECT ST_AsText(ST_SetEndM('LINESTRING(0 0, 1 1, 2 0)::geometry, 75));
```

```
LINESTRING M (0 0 0,1 1 0,2 0 75)
```

Set end M value of a CircularString

```
SELECT ST_AsText(ST_SetEndM('CIRCULARSTRING M(0 0 5, 1 1 10, 2 0 15)::geometry, 88));
```

```
CIRCULARSTRING M (0 0 5,1 1 10,2 0 88)
```

Set end M value of a NURBSCurve

```
SELECT ST_AsText(ST_SetEndM('NURBSCURVE M(2, (0 0 100, 1 1 200, 2 0 300))::geometry, 999)) ←  
;
```

```
NURBSCURVE M (DEGREE 2,CONTROLPOINTS M (NURBSPOINT(WEIGHTEDPOINT M (0 0 100),WEIGHT 1), ←  
NURBSPOINT(WEIGHTEDPOINT M (1 1 200),WEIGHT 1),NURBSPOINT(WEIGHTEDPOINT M (2 0 999), ←  
WEIGHT 1)),KNOTS (KNOT(0,3),KNOT(1,3)))
```

関連情報

[ST_SetStartM](#), [ST_EndM](#), [ST_EndPoint](#)

7.4.44 ST_Summary

ST_Summary — ジオメトリについての要約文を返します。

Synopsis

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

説明

ジオメトリについての要約文を返します。

ジオメトリ型の後の角括弧で示されたフラグには次の意味があります。

- M: M 軸を持ちます
- Z: Z 軸を持ちます
- B: バウンディングボックスを持ちます

- G: 測地座標系 (ジオグラフィ) です
- S: 空間参照系を持ちます

✔ このメソッドは曲線ストリングと曲線に対応しています。

✔ この関数は多面体サーフェスに対応しています。

✔ この関数は三角形と不規則三角網 (TIN) に対応しています。

Availability: 1.2.2

Enhanced: 2.0.0 でジオグラフィ対応が追加されました。

Enhanced: 2.1.0 空間参照系を持つかを示す S フラグが追加されました。

Enhanced: 2.2.0 TIN と曲線の対応が追加されました。

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)

関連情報

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

7.4.45 ST_X

ST_X — ポイントの X 値を返します。

Synopsis

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

説明

ポイントの X 座標値を返し、有効でないなら NULL を返します。入力ポイントは点でなければなりません。



Note

ジオメトリの X 値の最小値と最大値を得るには **ST_XMin** と **ST_XMax** を使います。

✔ このメソッドは SQL/MM 仕様の実装です。SQL-MM 3: 6.1.3

✔ この関数は 3 次元に対応し、Z 値を削除しません。

例

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

```
1
```

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

```
1.5
```

関連情報

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

7.4.46 ST_Y

ST_Y — ポイントの Y 値を返します。

Synopsis

```
float ST_Y(geometry a_point);
```

説明

ポイントの Y 座標値を返し、有効でないなら NULL を返します。入力ポイントは点でなければなりません。



Note

ジオメトリの Y 値の最小値と最大値を得るには **ST_YMin** と **ST_YMax** を使います。

✔ このメソッドは **OGC Simple Features Implementation Specification for SQL 1.1** の実装です。

✔ このメソッドは SQL/MM 仕様の実装です。SQL-MM 3: 6.1.4

✔ この関数は 3 次元に対応し、Z 値を削除しません。

例

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

2

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

1.5

関連情報

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

7.4.47 ST_Z

ST_Z — ポイントの Z 値を返します。

Synopsis

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

説明

ポイントの Z 座標値を返し、有効でないなら NULL を返します。入力はポイントでなければなりません。



Note

ジオメトリの Z 値の最小値と最大値を得るには [ST_ZMin](#) と [ST_ZMax](#) を使います。



このメソッドは SQL/MM 仕様の実装です。



この関数は 3 次元に対応し、Z 値を削除しません。

例

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

3

関連情報

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

7.4.48 ST_ZMFlag

ST_ZMFlag — ジオメトリの ZM 座標次元を示す符号を返します。

Synopsis

```
smallint ST_ZMFlag(geometry geomA);
```

説明

ジオメトリの ZM 座標次元を示す符号を返します。

値は 0=XY, 1=XYM, 2=XYZ, 3=XYZM となります。

-  この関数は 3 次元に対応し、Z 値を削除しません。
-  このメソッドは曲線ストリングと曲線に対応しています。

例

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

```
0
```

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

```
1
```

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

```
2
```

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

```
3
```

関連情報

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

7.4.49 ST_HasZ

ST_HasZ — ジオメトリが Z 値を持っているかどうかを確認します。

Synopsis

```
boolean ST_HasZ(geometry geom);
```


説明

入力ジオメトリが **Z** 値を持っているかどうかを確認して、真偽値を返します。ジオメトリが **Z** 値を持っている場合には **TRUE** を返し、そうでない場合には **FALSE** を返します。

通常は、**Z** 値を持つジオメトリオブジェクトは 3 次元 (3D) ジオメトリを表現し、**Z** 値を持たないものは 2 次元 (2D) ジオメトリとなります。

この関数はジオメトリが標高や高さに関する情報を持つかどうかを判断するのに使います。

Availability: 3.5.0



この関数は 3 次元に対応し、**Z** 値を削除しません。



この関数は **M** 値に対応します。

例

```
SELECT ST_HasZ(ST_GeomFromText('POINT(1 2 3)'));
true
```

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

関連情報

[ST_ZMFlag](#)

[ST_HasM](#)

7.4.50 ST_HasM

ST_HasM — ジオメトリが **M** 値をもっているかどうかを確認します。

Synopsis

boolean **ST_HasM**(geometry geom);

説明

入力ジオメトリが **M** 値を持っているかどうかを確認して、真偽値を返します。ジオメトリが **M** 値を持っている場合には **TRUE** を返し、そうでない場合には **FALSE** を返します。

M 次元を持つジオメトリオブジェクトは通常、空間地物に関連付けられた計測値または追加データを表現します。

この関数はジオメトリが **M** 値に関する情報を持つかどうかを判断するのに使います。

Availability: 3.5.0



この関数は 3 次元に対応し、**Z** 値を削除しません。



この関数は **M** 値に対応します。

例

```
SELECT ST_HasM(ST_GeomFromText('POINTM(1 2 3)'));
true
```

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

関連情報

[ST_ZMFlag](#)

[ST_HasZ](#)

7.4.51 ST_ControlPoints

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

Synopsis

geometry **ST_ControlPoints**(geometry nurbscurve);

説明

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

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

Availability: 3.7.0

例

This example extracts control points from a NURBS curve.

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

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

This example counts the control points.

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

```
3
```

関連情報

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

7.4.52 ST_Degree

ST_Degree — Returns the polynomial degree of a NURBS curve.

Synopsis

```
integer ST_Degree(geometry nurbscurve);
```

説明

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

Availability: 3.7.0

例

This example returns the degree of a NURBS curve.

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

```
2
```

This example checks whether a NURBS curve is cubic.

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

```
t
```

関連情報

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

7.4.53 ST_Weights

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

Synopsis

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

説明

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

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

Availability: 3.7.0

例

This example returns weights from a rational NURBS curve.

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

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

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

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

```
null
```

関連情報

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

7.4.54 ST_Knots

ST_Knots — Returns the knot vector of a NURBS curve.

Synopsis

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

説明

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

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

Availability: 3.7.0

例

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

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

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

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

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

```
null
```

関連情報

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

7.4.55 ST_NumControlPoints

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

Synopsis

integer **ST_NumControlPoints**(geometry nurbscurve);

説明

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

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

Availability: 3.7.0

例

This example returns the number of control points.

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

```
3
```

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

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

```
t
```

関連情報

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

7.4.56 ST_NurbsCurveIsRational

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

Synopsis

boolean **ST_NurbsCurveIsRational**(geometry nurbscurve);

説明

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

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

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

Availability: 3.7.0

例

This example checks whether a curve is rational.

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

t

This example checks a non-rational curve.

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

f

関連情報

[ST_MakeNurbsCurve](#), [ST_Weights](#), [ST_IsNurbsCurve](#)

7.4.57 ST_IsNurbsCurve

ST_IsNurbsCurve — Checks if a geometry is a NURBS curve.

Synopsis

boolean **ST_IsNurbsCurve**(geometry geom);

説明

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

Returns NULL if the input is NULL.

Availability: 3.7.0

例

This example checks whether a geometry is a NURBS curve.

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

t

This example tests a regular LineString.

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

f

関連情報

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

7.4.58 ST_Evaluate

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

Synopsis

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

説明

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

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

Availability: 3.7.0

例

This example returns the midpoint of a NURBS curve.

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

This example samples multiple points along a curve.

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

関連情報

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

7.4.59 ST_NurbsToLineString

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

Synopsis

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

説明

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

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

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

Availability: 3.7.0

例

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

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

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

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

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

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

関連情報

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

7.5 ジオメトリエディタ

7.5.1 ST_AddPoint

ST_AddPoint — ラインストリングにポイントを追加します。

Synopsis

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

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

説明

LINESTRING の *position* の位置 (0 はじまり) の前にポイントを追加します。 *position* パラメータが省略されるか-1 の場合には、LINESTRING の末尾に追加されます。

Availability: 1.1.0



この関数は 3 次元に対応し、Z 値を削除しません。

例

3 次元ラインの末尾へのポイントの追加

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

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

テーブル内の全てのラインについて、閉じていないラインにだけ始端を末尾に追加することで、閉じていることを保証します。

```
UPDATE sometable
SET geom = ST_AddPoint(geom, ST_StartPoint(geom))
FROM sometable
WHERE ST_IsClosed(geom) = false;
```

関連情報

[ST_RemovePoint](#), [ST_SetPoint](#)

7.5.2 ST_CollectionExtract

ST_CollectionExtract — ジオメトリコレクションを与えると、指定されたタイプの要素だけからなるマルチジオメトリを返します。

Synopsis

```
geometry ST_CollectionExtract(geometry collection);
geometry ST_CollectionExtract(geometry collection, integer type);
```

説明

ジオメトリコレクションを指定すると、要素のタイプが統一されたマルチジオメトリを返します。

type が指定されていない場合には、最大次元のジオメトリだけを含むマルチジオメトリを返します。このため、ポリゴンはラインに優先され、ラインはポイントに優先されます。

type が指定されている場合には、指定されたタイプだけを含むマルチジオメトリを返します。指定されたタイプの要素が無い場合には、EMPTY ジオメトリを返します。ポイント、ライン、ポリゴンだけに対応しています。タイプの番号は次の通りです。

- 1 == POINT
- 2 == LINESTRING

- 3 == POLYGON

非マルチジオメトリの入力に対しては、ジオメトリのタイプと指定したタイプが合致している場合には変更せず返します。合致しない場合には、指定したタイプの EMPTY ジオメトリを返します。ST_Multi"/> を使ってマルチ系ジオメトリに変換する必要があるならST_Multiを使います。



Warning

マルチポリゴンの結果は妥当性チェックを行いません。ポリゴン要素が隣接やオーバーラップしている場合には、結果ジオメトリは不正となります (たとえば、この関数をST_Splitの結果に適用すると発生します)。この状況に陥っているかはST_IsValidで確認でき、ST_MakeValidで修復できます。

Availability: 1.5.0



Note

1.5.3 より前のこの関数は非マルチジオメトリの入力に対して、指定タイプに関係なく変更せずに戻しました。1.5.3 で指定タイプに合致しない単一ジオメトリ入力に対して NULL を返すようになりました。2.0.0 で、合致しないジオメトリに対して、指定タイプの EMPTY ジオメトリを返すようになりました。

例

最大次元となるタイプの抽出:

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

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

ポイントの抽出 (type 1 == POINT):

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

```
MULTIPOINT((0 0))
```

ラインの抽出 (type 2 == LINESTRING):

```
SELECT ST_AsText(ST_CollectionExtract('GEOMETRYCOLLECTION(GEOMETRYCOLLECTION(LINESTRING(0 ←
  0, 1 1)),LINESTRING(2 2, 3 3))',
  2));
```

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

関連情報

ST_CollectionHomogenize, ST_Multi, ST_IsValid, ST_MakeValid

7.5.3 ST_CollectionHomogenize

ST_CollectionHomogenize — ジオメトリコレクションを与えると、最も単純な表現を返します。

Synopsis

geometry **ST_CollectionHomogenize**(geometry collection);

説明

ジオメトリコレクションを与えると、「最も単純な」表現を返します。

- 同種の要素からなるコレクションが適切なマルチ系ジオメトリとして返されます。
- タイプ混合のコレクションはフラットな単一の GEOMETRYCOLLECTION に変換されます。
- 単一の非マルチジオメトリ要素からなるコレクションはその要素が返されます。
- 非マルチジオメトリは変更されずに返ります。マルチジオメトリへの変換が必要なら **ST_Multi** を使います。



Warning

この関数は結果ジオメトリの妥当性を保証されず、隣接やオーバーラップする複数ポリゴンからは不正な MULTIPOLYGON が生成されます。この状況に陥っているかは **ST_IsValid** で確認でき、**ST_MakeValid** で修復できます。

Availability: 2.0.0

例

単一要素のコレクションから非マルチジオメトリへの変換

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

```
POINT(0 0)
```

ネスト下単一要素のコレクションから非マルチジオメトリへの変換:

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

```
POINT(0 0)
```

コレクションからマルチ系ジオメトリへの変換:

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

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

ネストしたタイプ混合のコレクションからフラットなジオメトリコレクションへの変換:

```
SELECT ST_AsText(ST_CollectionHomogenize('GEOMETRYCOLLECTION(POINT(0 0), GEOMETRYCOLLECTION ←  
( LINESTRING(1 1, 2 2))')));
```

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

ポリゴンのコレクションから (不正な) マルチポリゴンへの変換:

```
SELECT ST_AsText(ST_CollectionHomogenize('GEOMETRYCOLLECTION (POLYGON ((10 50, 50 50, 50 ←  
10, 10 10, 10 50)), POLYGON ((90 50, 90 10, 50 10, 50 50, 90 50)))'));
```

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

関連情報

[ST_CollectionExtract](#), [ST_Multi](#), [ST_IsValid](#), [ST_MakeValid](#)

7.5.4 ST_CurveToLine

ST_CurveToLine — 曲線を含むジオメトリを線ジオメトリに変換します。

Synopsis

geometry **ST_CurveToLine**(geometry curveGeom, float tolerance, integer tolerance_type, integer flags);

説明

CIRCULARSTRING を LINESTRING に、CURVEPOLYGON を POLYGON に、MULTISURFACE を MULTIPOLYGON に、それぞれ変換します。CIRCULARSTRING ジオメトリタイプに対応していないデバイスへの出力に使用します。

与えられたジオメトリを線型ジオメトリに変換します。それぞれの曲線ジオメトリまたは辺は、`tolerance` とオプションを使用して線形近似に変換します (デフォルトでは 4 分の 1 円ごとに 32 辺でオプションなしです)。

`tolerance\_type` 引数によって `tolerance` 引数の解釈が決定されます。

- 0 (デフォルト): tolerance は 4 分の 1 円の最大辺数です。
- 1: tolerance は曲線からラインまでの最大差です。単位は入力ジオメトリの単位です。
- 2: tolerance は生成される半径がなす角度のラジアン単位の最大値です。

`flags` 引数はビットフィールドです。デフォルトでは 0 です。次のビットに対応します。

- 1: 対称となる (方向独立) 出力。
- 2: 角度維持。対称出力を生成する時に角度 (辺長) 減少を避けます。対称フラグが OFF の時は何の効果もありません。

Availability: 1.3.0

Enhanced: 2.4.0 最大距離差による許容範囲と最大角度による許容範囲に対応し、対称出力に対応しました。

Enhanced: 3.0.0 線形化した弧ごとの最小線分数を実装しました。トポロジ的な崩壊を防ぐためです。



このメソッドは [OGC Simple Features Implementation Specification for SQL 1.1](#) の実装です。



このメソッドは SQL/MM 仕様の実装です。SQL-MM 3: 7.1.7



この関数は 3 次元に対応し、Z 値を削除しません。



このメソッドは曲線ストリングと曲線に対応しています。

例

```
SELECT ST_AsText(ST_CurveToLine(ST_GeomFromText('CIRCULARSTRING(220268 150415,220227 150505,220227 150406)')));

LINESTRING(220268 150415,220269.95064912 150416.539364228,220271.823415575 150418.17258804,220273.613787707 150419.895736857,
220275.317452352 150421.704659462,220276.930305234 150423.594998003,220278.448460847 150425.562198489,
220279.868261823 150427.60152176,220281.186287736 150429.708054909,220282.399363347 150431.876723113,
220283.50456625 150434.10230186,220284.499233914 150436.379429536,220285.380970099 150438.702620341,220286.147650624 150441.066277505,
220286.797428488 150443.464706771,220287.328738321 150445.892130112,220287.740300149 150448.342699654,
220288.031122486 150450.810511759,220288.200504713 150453.289621251,220288.248038775 150455.77405574,
220288.173610157 150458.257830005,220287.977398166 150460.734960415,220287.659875492 150463.199479347,
220287.221807076 150465.64544956,220286.664248262 150468.066978495,220285.988542259 150470.458232479,220285.196316903 150472.81345077,
220284.289480732 150475.126959442,220283.270218395 150477.39318505,220282.140985384 150479.606668057,
220280.90450212 150481.762075989,220279.5637474 150483.85421628,220278.12195122 150485.87804878,
220276.582586992 150487.828697901,220274.949363179 150489.701464356,220273.226214362 150491.491836488,
220271.417291757 150493.195501133,220269.526953216 150494.808354014,220267.559752731 150496.326509628,
220265.520429459 150497.746310603,220263.41389631 150499.064336517,220261.245228106 150500.277412127,
220259.019649359 150501.38261503,220256.742521683 150502.377282695,220254.419330878 150503.259018879,
220252.055673714 150504.025699404,220249.657244448 150504.675477269,220247.229821107 150505.206787101,
220244.779251566 150505.61834893,220242.311439461 150505.909171266,220239.832329968 150506.078553494,
220237.347895479 150506.126087555,220234.864121215 150506.051658938,220232.386990804 150505.855446946,
220229.922471872 150505.537924272,220227.47650166 150505.099855856,220225.054972724 150504.542297043,
220222.663718741 150503.86659104,220220.308500449 150503.074365683,
220217.994991777 150502.167529512,220215.72876617 150501.148267175,
220213.515283163 150500.019034164,220211.35987523 150498.7825509,
220209.267734939 150497.441796181,220207.243902439 150496,
220205.293253319 150494.460635772,220203.420486864 150492.82741196,220201.630114732 150491.104263143,
220199.926450087 150489.295340538,220198.313597205 150487.405001997,220196.795441592 150485.437801511,
220195.375640616 150483.39847824,220194.057614703 150481.291945091,220192.844539092 150479.123276887,220191.739336189 150476.89769814,
220190.744668525 150474.620570464,220189.86293234 150472.297379659,220189.096251815 150469.933722495,
220188.446473951 150467.535293229,220187.915164118 150465.107869888,220187.50360229 150462.657300346,
220187.212779953 150460.189488241,220187.043397726 150457.710378749,220186.995863664 150455.22594426,
220187.070292282 150452.742169995,220187.266504273 150450.265039585,220187.584026947 150447.800520653,
220188.022095363 150445.35455044,220188.579654177 150442.933021505,220189.25536018 150440.541767521,
```

```
220190.047585536 150438.18654923,220190.954421707 150435.873040558,220191.973684044 ↵
150433.60681495,
220193.102917055 150431.393331943,220194.339400319 150429.237924011,220195.680155039 ↵
150427.14578372,220197.12195122 150425.12195122,
220198.661315447 150423.171302099,220200.29453926 150421.298535644,220202.017688077 ↵
150419.508163512,220203.826610682 150417.804498867,
220205.716949223 150416.191645986,220207.684149708 150414.673490372,220209.72347298 ↵
150413.253689397,220211.830006129 150411.935663483,
220213.998674333 150410.722587873,220216.22425308 150409.61738497,220218.501380756 ↵
150408.622717305,220220.824571561 150407.740981121,
220223.188228725 150406.974300596,220225.586657991 150406.324522731,220227 150406)
```

This example uses 3D input.

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

```
LINESTRING(220268 150415 1,220269.95064912 150416.539364228 1.0181172856673,
220271.823415575 150418.17258804 1.03623457133459,220273.613787707 150419.895736857 ↵
1.05435185700189,...AD INFINITUM ...
220225.586657991 150406.324522731 1.32611114201132,220227 150406 3)
```

Use only 2 segments to approximate quarter circle.

```
SELECT ST_AsText(ST_CurveToLine(ST_GeomFromText('CIRCULARSTRING(220268 150415,220227 ↵
150505,220227 150406)'),2));
```

```
LINESTRING(220268 150415,220287.740300149 150448.342699654,220278.12195122 150485.87804878,
220244.779251566 150505.61834893,220207.243902439 150496,220187.50360229 150462.657300346,
220197.12195122 150425.12195122,220227 150406)
```

Ensure approximated line is no further than 20 units away from. Original curve, and make the result direction-neutral.

```
SELECT ST_AsText(ST_CurveToLine('CIRCULARSTRING(0 0,100 -100,200 0)')::geometry,
20, -- Tolerance
1, -- Above is max distance between curve and line
1 -- Symmetric flag
));
```

```
LINESTRING(0 0,49.99999999999996 -86.60254037844383,149.9999999999994 ↵
-86.6025403784439,200 0)
```

関連情報

[ST_LineToCurve](#)

7.5.5 ST_Scroll

ST_Scroll — 閉じた LINESTRING の開始点を変更する。

Synopsis

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

説明

閉じた LINESTRING の開始/終了点を *point* で指定した頂点に変更します。

Availability: 3.2.0

- ✔ この関数は 3 次元に対応し、Z 値を削除しません。
- ✔ この関数は M 値に対応します。

例

閉じたラインの 3 番目の頂点を開始点にする

```
SELECT ST_AsEWKT(ST_Scroll('SRID=4326;LINESTRING(0 0 0 1, 10 0 2 0, 5 5 4 2,0 0 0 1)', ' ←  
POINT(5 5 4 2)'));
```

```
SRID=4326;LINESTRING(5 5 4 2,0 0 0 1,10 0 2 0,5 5 4 2)
```

関連情報

[ST_Normalize](#)

7.5.6 ST_FlipCoordinates

ST_FlipCoordinates — X 値と Y 値を入れ替えたジオメトリを返します。

Synopsis

geometry **ST_FlipCoordinates**(geometry geom);

説明

与えられたジオメトリの X 値と Y 値を入れ替えたものを返します。緯度経度 (Y,X) で表現される座標値を持つジオメトリを修正するのに使います。

Availability: 2.0.0

- ✔ このメソッドは曲線ストリングと曲線に対応しています。
- ✔ この関数は 3 次元に対応し、Z 値を削除しません。
- ✔ この関数は M 値に対応します。
- ✔ この関数は多面体サーフェスに対応しています。
- ✔ この関数は三角形と不規則三角網 (TIN) に対応しています。

例

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

```
POINT(2 1)
```

関連情報

[ST_SwapOrdinates](#)

7.5.7 ST_Force2D

ST_Force2D — ジオメトリを 2 次元モードに強制します。

Synopsis

geometry **ST_Force2D**(geometry geomA);

説明

ジオメトリを「2 次元モード」に強制させます。全ての出力表現は XY 座標値のみを持つことになります。OGC 準拠の出力 (OGC は 2 次元ジオメトリのみ策定しています) に強制するために使われます。

Enhanced: 2.0.0 多面体サーフェス対応が導入されました。

Changed: 2.1.0 2.0.x の間は ST_Force_2D と呼ばれていました。

- ✓ このメソッドは曲線ストリングと曲線に対応しています。
- ✓ この関数は多面体サーフェスに対応しています。
- ✓ この関数は 3 次元に対応し、Z 値を削除しません。

例

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

```
CIRCULARSTRING(1 1,2 3,4 5,6 7,5 6)
```

```
SELECT ST_AsEWKT(ST_Force2D('POLYGON((0 0 2,0 5 2,5 0 2,0 0 2),(1 1 2,3 1 2,1 3 2,1 1 2))' ));
```

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

関連情報

[ST_Force3D](#)

7.5.8 ST_Force3D

ST_Force3D — ジオメトリを XYZ モードに強制します。これは ST_Force3DZ の別名です。

Synopsis

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

説明

ジオメトリを XYZ モードに強制します。これは ST_Force_3DZ の別名です。ジオメトリが Z 値を持っていない場合は *Zvalue* の Z 値を追加します。

Enhanced: 2.0.0 多面体サーフェス対応が導入されました。

Changed: 2.1.0 2.0.x の間は ST_Force_3D と呼ばれていました。

Changed: 3.1.0. 0 でない Z 値を指定できるようになりました。

- ✔ この関数は多面体サーフェスに対応しています。
- ✔ このメソッドは曲線ストリングと曲線に対応しています。
- ✔ この関数は 3 次元に対応し、Z 値を削除しません。

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)')));
```

```
CIRCULARSTRING(1 1 2,2 3 2,4 5 2,6 7 2,5 6 2)
```

```
SELECT ST_AsEWKT(ST_Force3D('POLYGON((0 0,0 5,5 0,0 0),(1 1,3 1,1 3,1 1))')); 
```

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

関連情報

[ST_AsEWKT](#), [ST_Force2D](#), [ST_Force3DM](#), [ST_Force3DZ](#)

7.5.9 ST_Force3DZ

ST_Force3DZ — ジオメトリを XYZ モードに強制します。

Synopsis

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

説明

ジオメトリを XYZ モードに強制します。これは ST_Force_3DZ の別名です。ジオメトリが Z 値を持っていない場合は *Zvalue* の Z 値を追加します。

Enhanced: 2.0.0 多面体サーフェス対応が導入されました。

Changed: 2.1.0 2.0.x の間は ST_Force_3DZ と呼ばれていました。

Changed: 3.1.0. 0 でない Z 値を指定できるようになりました。

- ✔ この関数は多面体サーフェスに対応しています。
- ✔ この関数は 3 次元に対応し、Z 値を削除しません。
- ✔ このメソッドは曲線ストリングと曲線に対応しています。

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)')));
```

```
CIRCULARSTRING(1 1 2,2 3 2,4 5 2,6 7 2,5 6 2)
```

```
SELECT ST_AsEWKT(ST_Force3DZ('POLYGON((0 0,0 5,5 0,0 0),(1 1,3 1,1 3,1 1))'));
```

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

関連情報

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

7.5.10 ST_Force3DM

ST_Force3DM — ジオメトリを XYM モードに強制します。

Synopsis

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

説明

ジオメトリを XYM モードに強制します。ジオメトリが M 値を持っていない場合は *Mvalue* の M 値を追加します。Z 値を持っている場合は Z 値は除去されます。

Changed: 2.1.0 2.0.x の間は ST_Force_3DM と呼ばれていました。

Changed: 3.1.0. 0 でない M 値を指定できるようになりました。

 このメソッドは曲線ストリングと曲線に対応しています。

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)')));
```

```
CIRCULARSTRINGM(1 1 0,2 3 0,4 5 0,6 7 0,5 6 0)
```

```
SELECT ST_AsEWKT(ST_Force3DM('POLYGON((0 0 1,0 5 1,5 0 1,0 0 1),(1 1 1,3 1 1,1 3 1,1 1 1))'));
```

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

関連情報

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

7.5.11 ST_Force4D

ST_Force4D — ジオメトリを XYZM モードに強制します。

Synopsis

geometry **ST_Force4D**(geometry geomA, float Zvalue = 0.0, float Mvalue = 0.0);

説明

ジオメトリを XYZM モードに強制します。Z 値や M 値が無い場合は *Zvalue* と *Mvalue* を追加します。

Changed: 2.1.0 2.0.x の間は ST_Force_4D と呼ばれていました。

Changed: 3.1.0. 0 でない Z 値と M 値を指定できるようになりました。

- ✔ この関数は 3 次元に対応し、Z 値を削除しません。
- ✔ このメソッドは曲線ストリングと曲線に対応しています。

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)')));
```

```
CIRCULARSTRING(1 1 2 0,2 3 2 0,4 5 2 0,6 7 2 0,5 6 2 0)
```

```
SELECT ST_AsEWKT(ST_Force4D('MULTILINESTRINGM((0 0 1,0 5 2,5 0 3,0 0 4),(1 1 1,3 1 1,1 3 1,1 1 1))'));
```

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

関連情報

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

7.5.12 ST_ForceCollection

ST_ForceCollection — ジオメトリをジオメトリコレクションに変換します。

Synopsis

geometry **ST_ForceCollection**(geometry geomA);

説明

ジオメトリをジオメトリコレクションに変換します。これは WKB 表現を単純化するのに便利です。

Enhanced: 2.0.0 多面体サーフェス対応が導入されました。

Availability: 1.2.2 1.3.4 より前は、曲線を含むジオメトリで使うとクラッシュしました。これは 1.3.4 以上では訂正されています。

Changed: 2.1.0 2.0.x の間は ST_Force_Collection と呼ばれていました。

- ✔ この関数は多面体サーフェスに対応しています。
- ✔ この関数は 3 次元に対応し、Z 値を削除しません。
- ✔ このメソッドは曲線ストリングと曲線に対応しています。

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))'));
```

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

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

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

This example uses a PolyhedralSurface.

```
SELECT ST_AsEWKT(ST_ForceCollection('POLYHEDRALSURFACE(((0 0 0,0 0 1,0 1 1,0 1 0,0 0 0)),
((0 0 0,0 1 0,1 1 0,1 0 0,0 0 0)),
((0 0 0,1 0 0,1 0 1,0 0 1,0 0 0)),
((1 1 0,1 1 1,1 0 1,1 0 0,1 1 0)),
((0 1 0,0 1 1,1 1 1,1 1 0,0 1 0)),
((0 0 1,1 0 1,1 1 1,0 1 1,0 0 1)))'));
```

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

関連情報

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

7.5.13 ST_ForceCurve

ST_ForceCurve — 該当する場合は、ジオメトリを曲線タイプに変換します。

Synopsis

geometry **ST_ForceCurve**(geometry g);

説明

可能ならジオメトリを曲線表現に変更します。ラインは複合曲線になり、マルチラインはマルチ曲線になり、ポリゴンは曲線ポリゴンになり、マルチポリゴンはマルチサーフェスになります。入力ジオメトリが既に曲線表現であるなら、入力と同じ値が返されます。

Availability: 2.2.0



この関数は 3 次元に対応し、Z 値を削除しません。



このメソッドは曲線ストリングと曲線に対応しています。

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))
```

```
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))
```

関連情報

[ST_LineToCurve](#)

7.5.14 ST_ForcePolygonCCW

ST_ForcePolygonCCW — 全ての外環を反時計回りに、全ての内環を時計回りに、それぞれ強制します。

Synopsis

geometry **ST_ForcePolygonCCW** (geometry geom);

説明

(マルチ) ポリゴンに対して、外環を反時計回りに、内環を時計回りに強制します。ポリゴンでないジオメトリは、変更されずに返されます。

Availability: 2.4.0



この関数は 3 次元に対応し、Z 値を削除しません。



この関数は M 値に対応します。

関連情報

[ST_ForcePolygonCW](#) , [ST_IsPolygonCCW](#) , [ST_IsPolygonCW](#)

7.5.15 ST_ForcePolygonCW

ST_ForcePolygonCW — 全ての外環を時計回りに、全ての内環を反時計回りに、それぞれ強制します。

Synopsis

```
geometry ST_ForcePolygonCW ( geometry geom );
```

説明

(マルチ) ポリゴンに対して、外環を時計回りに、内環を反時計回りに強制します。ポリゴンでないジオメトリは、変更されずに返されます。

Availability: 2.4.0

- ✔ この関数は 3 次元に対応し、Z 値を削除しません。
- ✔ この関数は M 値に対応します。

関連情報

[ST_ForcePolygonCCW](#) , [ST_IsPolygonCCW](#) , [ST_IsPolygonCW](#)

7.5.16 ST_ForceSFS

ST_ForceSFS — SFS 1.1 ジオメトリタイプのみ使うようジオメトリに強制します。

Synopsis

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

説明

- ✔ この関数は多面体サーフェスに対応しています。
- ✔ この関数は三角形と不規則三角網 (TIN) に対応しています。
- ✔ このメソッドは曲線ストリングと曲線に対応しています。
- ✔ この関数は 3 次元に対応し、Z 値を削除しません。

7.5.17 ST_ForceRHR

ST_ForceRHR — ポリゴンの頂点の方向を右回りに強制します。

Synopsis

```
geometry ST_ForceRHR(geometry g);
```

説明

ポリゴンの頂点の方向は **Right-Hand-Rule** に従います。この際、ポリゴンの領域は、境界線の右側になります。特に、外環は時計回りに強制され、内環は反時計回りに強制されます。この関数は **ST_ForcePolygonCW** の別名です。

**Note**

上の RHR の定義は、他の文脈で使われる場合の定義と矛盾します。これを解消するには、**ST_ForcePolygonCW** を使うことを推奨します。

Enhanced: 2.0.0 多面体サーフェス対応が導入されました。



この関数は 3 次元に対応し、Z 値を削除しません。



この関数は多面体サーフェスに対応しています。

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))'))
```

```
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))
```

関連情報

ST_ForcePolygonCCW , **ST_ForcePolygonCW** , **ST_IsPolygonCCW** , **ST_IsPolygonCW** , **ST_BuildArea**, **ST_Polygonize**, **ST_Reverse**

7.5.18 ST_LineExtend

ST_LineExtend — 指定距離ぶん前後に延長されたラインを返します。

Synopsis

geometry **ST_LineExtend**(geometry line, float distance_forward, float distance_backward=0.0);

説明

新しい始点 (と終点) を与えられた距離を取って追加することで、指定距離ぶん前後に延長されたラインを返します。距離を 0 にした場合には、ポイントは追加されません。非負の距離値のみ受け付けます。追加ポイントの方向は始点 (と終点の二つの異なるポイントによって決定されます。重複ポイントは無視されます。

Availability: 3.4.0

例: ラインを前方に **5** 単位延長し、後方に **6** 単位延長する

```
SELECT ST_AsText(ST_LineExtend('LINESTRING(0 0, 0 10)::geometry, 5, 6));
```

```
LINESTRING(0 -6,0 0,0 10,0 15)
```

関連情報

[ST_LineSubstring](#), [ST_LocateAlong](#), [ST_Project](#)

7.5.19 ST_LineToCurve

`ST_LineToCurve` — 曲線を含むジオメトリを線ジオメトリに変換します。

Synopsis

geometry **ST_LineToCurve**(geometry geomANoncircular);

説明

LINESTRING/POLYGON を CIRCULARSTRING と曲線ポリゴンに変換します。等価の曲線を記述するのに必要なポイントが少なくなります。



Note

入力ラインSTRING/ポリゴンが、曲線をはっきりと表現するのには不十分な場合には、関数は入力ジオメトリと同じものを返します。

Availability: 1.3.0



この関数は 3 次元に対応し、Z 値を削除しません。



このメソッドは曲線STRINGと曲線に対応しています。

例

This example uses 2D input.

```
SELECT ST_AsText(ST_LineToCurve(foo.geom)) As curvedastext, ST_AsText(foo.geom) As
  non_curvedastext
FROM (SELECT ST_Buffer('POINT(1 3)::geometry, 3) As geom) As foo;
```

curvedastext	non_curvedastext
-- ↩	

CURVEPOLYGON(CIRCULARSTRING(4 3,3.12132034355964 0.878679656440359, POLYGON((4 ↩	
3,3.94235584120969 2.41472903395162,3.77163859753386 1.85194970290473,	
1 0,-1.12132034355965 5.12132034355963,4 3)) 3.49440883690764 ↩	
1.33328930094119,3.12132034355964 0.878679656440359,	


```
| 2.66671069905881 ↵
| 0.505591163092366,2.14805029
| 0.228361402466141,
| 1.58527096604839 ↵
| 0.0576441587903094,1 ↵
| 0,
| 0.414729033951621 ↵
| 0.0576441587903077,-0.1480502
| 0.228361402466137,
| -0.666710699058802 ↵
| 0.505591163092361,-1.1213203
| 0.878679656440353,
| -1.49440883690763 ↵
| 1.33328930094119,-1.77163859
| 1.85194970290472
| --ETC-- ↵
| ,3.94235584120969 ↵
| 3.58527096604839,4 ↵
| 3))
```

This example uses 3D input.

```
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 3 3,3 3 3)	LINESTRING Z (3 3 3,2.414213562373095 ↵
1.585786437626905 3,1 1 3,-0.414213562373095 1.585786437626905 3,-1 3 ↵	
3,-0.414213562373095 4.414213562373095 3,1 5 3,2.414213562373095 4.414213562373096 3,3 3 ↵	
3)	
(1 row)	

関連情報

[ST_CurveToLine](#)

7.5.20 ST_Multi

ST_Multi — マルチ系ジオメトリを返します。

Synopsis

geometry **ST_Multi**(geometry geom);

説明

マルチ系ジオメトリを返します。ジオメトリが既にマルチ系なら変更せずに返します。

例

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

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

関連情報

[ST_AsText](#)

7.5.21 ST_Normalize

ST_Normalize — 標準的な形式に変えたジオメトリを返します。

Synopsis

geometry **ST_Normalize**(geometry geom);

説明

正規化/標準化された形式のジオメトリを返します。ポリゴンの環における頂点の順序、ポリゴンにおける環の順序、複合ジオメトリにおける要素の順序が変更されることがあります。

ほとんどの場合、試験目的 (期待した結果と実際に得た結果との比較) でのみ使用します。

Availability: 2.3.0

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)
  )
)'));

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))
```

関連情報

[ST_Equals](#),

7.5.22 ST_Project

ST_Project — 始点から距離と方位で算出されたポイントを返します。

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);
```

説明

始点から測地線に沿って与えられた距離と方位で算出されたポイントを返します。測地問題と言われるものです。

2 ポイント版は、方位を黙示的に定義するために一つ目のポイントから二つ目のポイントに向かう線を使い、距離は以前と同様に使います。

距離はメートルで与えます。負数に対応しています。

方位はラジアンで与えます。真北 (方位 0) から時計回りに増えます。真北から時計回りに数えます。

- 北は方位ゼロ (0 度) です
- 東は方位 $\pi/2$ (90 度) です
- 南は方位 π (180 度) です
- 西は方位 $3\pi/2$ (270 度) です

負の方位値と 2π (360 度) を超える値に対応しています。

Availability: 2.0.0

Enhanced: 2.4.0 負の距離と非正規化方位を許容するようになりました。

Enhanced: 3.4.0 ジオメトリ引数と、`azimuth` を省略した 2 ポイント形式を許します。

例: **100,000** メートル、方位 **45** 度で計算されるポイント

```
SELECT ST_AsText(ST_Project('POINT(0 0)::geography', 100000, radians(45.0)));  
  
POINT(0.635231029125537 0.639472334729198)
```

関連情報

[ST_Azimuth](#), [ST_Distance](#), [PostgreSQL function radians\(\)](#)

7.5.23 ST_QuantizeCoordinates

`ST_QuantizeCoordinates` — 座標値の最下位ビットを 0 にします。

Synopsis

```
geometry ST_QuantizeCoordinates ( geometry g , int prec_x , int prec_y , int prec_z , int prec_m );
```

説明

`ST_QuantizeCoordinates` は、指定した小数点以下の桁数での座標値表現に必要なビット数 (N) を決定し、最大の有効ビット数 N 以外を 0 にします。結果の座標値は元の値を丸めますが、圧縮性が改善されます。これにより、ジオメトリカラムが **compressible storage type** を使って、ディスク使用を劇的に減少させることができます。この関数によって、小数点以下の桁数の異なる指定が可能になります。指定されていない次元は x 次元の精度を持つものとし、負の桁数は、小数点以上の桁数の参照と解釈されます (例: `prec_x=-2` は 100 付近の座標値を保存します)。

`ST_QuantizeCoordinates` が生成する座標は、これらの座標を含むジオメトリや、ジオメトリ内のこれらの相対的な位置から独立しています。結果として、ジオメトリ間に存在するトポロジ関係は、この関数の使用によって影響を受けることはありません。この関数は、ジオメトリの内在的な精度より低い桁数では不正なジオメトリを生成する可能性があります。

Availability: 2.5.0

技術背景

PostGIS はすべての座標値を倍精度浮動小数点数として格納し、15 桁の有効桁数を確実に表すことができます。ただし、PostGIS では、本質的に 15 桁未満のデータの管理ができます。例としては、小数点以下 6 桁の精度の地理座標として提供される TIGER データがあります (故に、必要な有効桁数は、経度は 9 桁、緯度は 8 桁です)。

有効桁数が 15 の時、多数のありえる 9 桁の表現があります。倍精度浮動小数点数は 52 の明示的なビット数を座標の仮数部に使っています。有効桁数 9 桁では仮数部は 30 ビットだけで必要で、22 ビットは有効ではありません。これらの値を好きなものにすることができ、結局は入力値を丸める数字となります。例えば、100.123456 という値は 100.123456000000, 100.123456000001 および 100.123456432199 に近い数として表現されます。全ては等しく妥当で、これらの入力では `ST_AsText(geom, 6)` は同じ結果を返します。これらのビット数をあらゆる値にセットすることができるので、`ST_QuantizeCoordinates` は無効ビットとなる 22 ビットに 0 をセットします。長い座標値のビット列では、連続的な 0 のブロックから、PostgreSQL によって効率的に圧縮されたパターンを生成します。



Note

ジオメトリのディスク上のサイズだけが `ST_QuantizeCoordinates` の影響を潜在的に受けます。ジオメトリのメモリ利用を報告する `ST_MemSize` は、ジオメトリに使われるディスク上のサイズにかかわらず同じ値を返します。

例

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

POINT(100.12344360351562 0)

WITH test AS (SELECT 'POINT (123.456789123456 123.456789123456)::geometry' AS geom)
SELECT
  digits,
  encode(ST_QuantizeCoordinates(geom, digits), 'hex'),
  ST_AsText(ST_QuantizeCoordinates(geom, digits))
FROM test, generate_series(15, -15, -1) AS digits;
```

digits	encode	st_astext
15	010100000005f9a72083cdd5e405f9a72083cdd5e40	POINT(123.456789123456 123.456789123456)

```

14 | 010100000005f9a72083cdd5e405f9a72083cdd5e40 | POINT(123.456789123456 ↵
    123.456789123456)
13 | 010100000005e9a72083cdd5e405e9a72083cdd5e40 | POINT(123.45678912345599 ↵
    123.45678912345599)
12 | 01010000000409a72083cdd5e40409a72083cdd5e40 | POINT(123.45678912345556 ↵
    123.45678912345556)
11 | 01010000000009a72083cdd5e40009a72083cdd5e40 | POINT(123.45678912345466 ↵
    123.45678912345466)
10 | 0101000000009872083cdd5e40009872083cdd5e40 | POINT(123.45678912344738 ↵
    123.45678912344738)
9 | 0101000000008072083cdd5e40008072083cdd5e40 | POINT(123.45678912336007 ↵
    123.45678912336007)
8 | 0101000000000070083cdd5e40000070083cdd5e40 | POINT(123.45678912103176 ↵
    123.45678912103176)
7 | 0101000000000060083cdd5e40000060083cdd5e40 | POINT(123.4567891061306 ↵
    123.4567891061306)
6 | 0101000000000000083cdd5e40000000083cdd5e40 | POINT(123.45678901672363 ↵
    123.45678901672363)
5 | 0101000000000000003cdd5e40000000003cdd5e40 | POINT(123.456787109375 ↵
    123.456787109375)
4 | 0101000000000000003cdd5e40000000003cdd5e40 | POINT(123.456787109375 ↵
    123.456787109375)
3 | 01010000000000000038dd5e400000000038dd5e40 | POINT(123.45654296875 ↵
    123.45654296875)
2 | 0101000000000000000dd5e40000000000dd5e40 | POINT(123.453125 123.453125)
1 | 0101000000000000000dc5e40000000000dc5e40 | POINT(123.4375 123.4375)
0 | 0101000000000000000c05e4000000000c05e40 | POINT(123 123)
-1 | 010100000000000000005e4000000000005e40 | POINT(120 120)
-2 | 01010000000000000000584000000000005840 | POINT(96 96)
-3 | 01010000000000000000584000000000005840 | POINT(96 96)
-4 | 01010000000000000000584000000000005840 | POINT(96 96)
-5 | 01010000000000000000584000000000005840 | POINT(96 96)
-6 | 01010000000000000000584000000000005840 | POINT(96 96)
-7 | 01010000000000000000584000000000005840 | POINT(96 96)
-8 | 01010000000000000000584000000000005840 | POINT(96 96)
-9 | 01010000000000000000584000000000005840 | POINT(96 96)
-10 | 01010000000000000000584000000000005840 | POINT(96 96)
-11 | 01010000000000000000584000000000005840 | POINT(96 96)
-12 | 01010000000000000000584000000000005840 | POINT(96 96)
-13 | 01010000000000000000584000000000005840 | POINT(96 96)
-14 | 01010000000000000000584000000000005840 | POINT(96 96)
-15 | 01010000000000000000584000000000005840 | POINT(96 96)
(31 rows)

```

関連情報

[ST_SnapToGrid](#)

7.5.24 ST_RemovePoint

ST_RemovePoint — ラインストリングからポイントを削除します。

Synopsis

geometry **ST_RemovePoint**(geometry linestring, integer offset);

説明

ラインストリングからポイントを削除します。インデックスは 0 始まりです。閉じたリングを開いたラインストリングに変えるのに使います。

Enhanced: 3.2.0

Availability: 1.1.0



この関数は 3 次元に対応し、Z 値を削除しません。

例

閉じたライン (リング) の終点を削除するとラインが閉じなくなることが保証されています。geom が LINESTRING であると仮定するします。

```
UPDATE sometable
SET geom = ST_RemovePoint(geom, ST_NPoints(geom) - 1)
FROM sometable
WHERE ST_IsClosed(geom);
```

関連情報

[ST_AddPoint](#), [ST_NPoints](#), [ST_NumPoints](#)

7.5.25 ST_RemoveRepeatedPoints

ST_RemoveRepeatedPoints — 重複ポイントを除いたジオメトリを返します。

Synopsis

geometry **ST_RemoveRepeatedPoints**(geometry geom, float8 tolerance = 0.0);

説明

重複ポイントを除いたジオメトリを返します。この関数は (MULTI)LINESTRING、(MULTI)POLYGON、MULTIPOINT のみを処理しますが、全ての種類のジオメトリを与えることができます。GEOMETRYCOLLECTION の要素は個別に処理されます。LINESTRING の端点は保持されます。

ゼロでない *tolerance* パラメータを与えた場合には、相互に許容範囲距離内の頂点は重複頂点とみなされます。距離は 2 次元 (XY 平面) で計算されます。

Enhanced: 3.2.0

Availability: 2.2.0



この関数は多面体サーフェスに対応しています。



この関数は 3 次元に対応し、Z 値を削除しません。

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)
```

例: ジオメトリコレクションの要素は個別に処理されます。

```
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))
```

例: 許容距離内にある繰り返しポイントの削除。

```
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)
```

関連情報

[ST_Simplify](#)

7.5.26 ST_RemoveIrrelevantPointsForView

ST_RemoveIrrelevantPointsForView — 指定した矩形表示範囲での描画に関係のないジオメトリのポイントを削除します。

Synopsis

geometry **ST_RemoveIrrelevantPointsForView**(geometry geom, box2d bounds, boolean cartesian_hint = false);

説明

指定した矩形表示範囲での描画に関係のないポイントを外した **geometry** を返します。

この関数を使用すると、特定の範囲内のみで描画する必要があるジオメトリに対して素早い前処理を実行できます。**(MULTI)POLYGON** と **(MULTI)LINESTRING** のタイプのジオメトリだけが評価されます。他のジオメトリは変更されません。

この関数は次の点において **ST_ClipByBox2D()** と対照的です

- この関数は、新しいインタセクトするポイントを計算することなくポイントの並べ替えを行うので、丸め誤差が回避され、通常は効率が向上し、
- ポイント数が同じ又は近いジオメトリを返し、
- 指定した描画範囲内では同じ描画結果になり、

- 自己交差を発生させ、結果ジオメトリを不正にする可能性があります (下の例をご覧ください)。

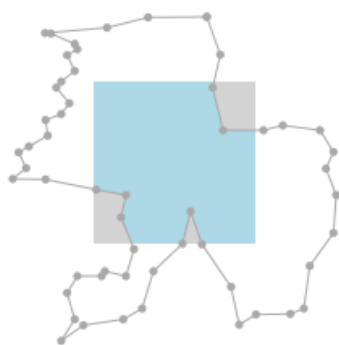
`cartesian_hint` を `TRUE` に設定した場合には、アルゴリズムは、結果のポイント数をさらに減らすために、デカルト数学を含む、追加の最適化を適用します。結果の座標が他の (非デカルト) 座標系に投影変換される場合には、この任意引数を使用することで、描画の不自然な諧調やノイズが現れる可能性があることに注意して下さい。



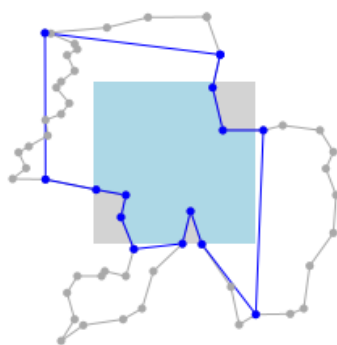
Warning

ポリゴンに対しては、この関数は現在は結果の妥当性を保証しません。この状況は `ST_IsValid` で確認でき、`ST_MakeValid` で修復することができます。

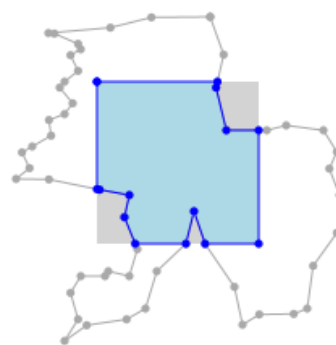
original
55 points



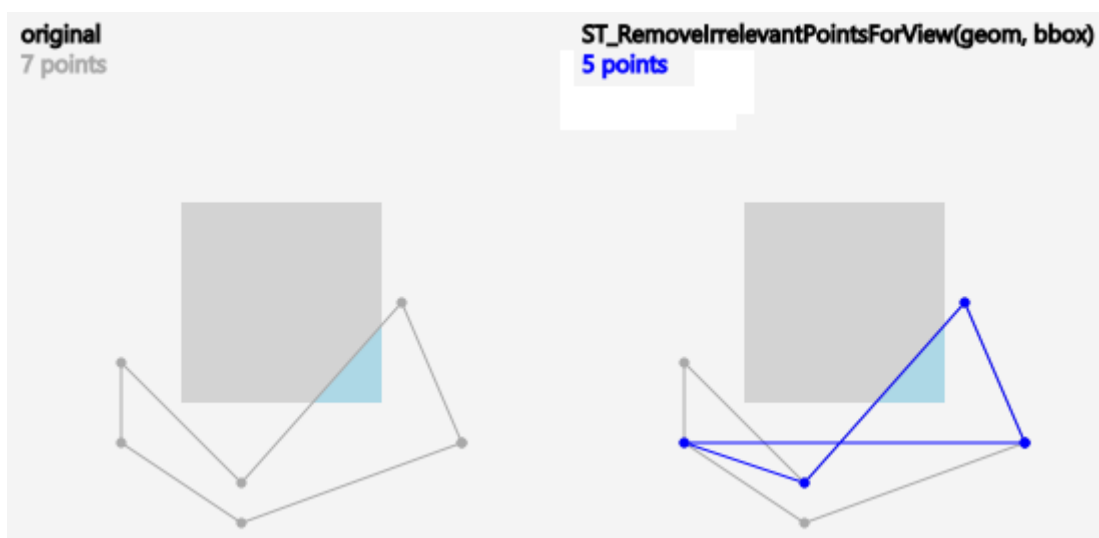
`ST_RemoveIrrelevantPointsForView(geom, bbox)`
15 points



`ST_ClipByBox2D(geom, bbox)`
15 points



例: `ST_RemoveIrrelevantPointsForView()` をポリゴンに適用しています。青いポイントが残り、灰色の描画範囲も残したうえで、結果の描画を行います (薄青の領域)。



例: ポイントが並べ替えられただけで、新しいポイントが計算されなかったという事実のため、`ST_RemoveIrrelevantPointsForView()` の結果に自己交差が含まれることがあります。

Availability: 3.5.0

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));
```

```
MULTIPOLYGON(((10 10,40 10,20 20,10 20,10 10))),((10 10,20 10,20 20,10 20,10 10)))
```

```
SELECT ST_AsText(ST_RemoveIrrelevantPointsForView(ST_GeomFromText('MULTILINESTRING((0 0, 10 ↵
    0,20 0,30 0), (0 15, 5 15, 10 15, 15 15, 20 15, 25 15, 30 15, 40 15), (13 13,15 15,17 ↵
    17)))'),
    ST_MakeEnvelope(12,12,18,18), true));
```

```
MULTILINESTRING((10 15,15 15,20 15),(13 13,15 15,17 17))
```

```
SELECT ST_AsText(ST_RemoveIrrelevantPointsForView(ST_GeomFromText('LINESTRING(0 0, 10 0,20 ↵
    0,30 0)'),
    ST_MakeEnvelope(12,12,18,18), true));
```

```
LINESTRING 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));
```

```
POLYGON((15 30,30 0,0 0,15 30))
```

```
SELECT ST_AsText(ST_RemoveIrrelevantPointsForView(ST_GeomFromText('POLYGON((0 30, 15 30, 30 ↵
    30, 30 0, 0 0, 0 30)))'),
    ST_MakeEnvelope(12,12,18,18));
```

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

関連情報

[ST_ClipByBox2D](#), [ST_Intersection](#)

7.5.27 ST_RemoveSmallParts

ST_RemoveSmallParts — ジオメトリの全ての小さな部位 (ポリゴンのリングまたはラインストリング) を削除します。

Synopsis

geometry **ST_RemoveSmallParts**(geometry geom, double precision minSizeX, double precision minSizeY);

説明

小さい部位 (外側リングや内側リングやラインストリング) を持たない **geometry** を返します。

この関数は、地図の簡略化における、小さい島や穴を削除するといった前処理段階で使います。

(MULTI)POLYGON と (MULTI)LINESTRING のタイプのジオメトリだけが評価されます。他のジオメトリは変更されません。

minSizeX が 0 より大きい場合には、*minSizeX* より小さい幅の部位が分けられます。

minSizeY が 0 より大きい場合には、*minSizeY* より小さい高さの部位が分けられます。

minSizeX と *minSizeY* は、ともにジオメトリの座標系の単位で計測されます。

ポリゴンタイプの場合には、評価はリングごとに個別に行われ、次の結果のいずれかが導かれます：

- 元のジオメトリ
- 全てのリングで頂点が減った POLYGON、
- 内側リングの数が減った POLYGON (さらに頂点が減っている場合あり)、
- POLYGON EMPTY、
- 要素ポリゴン数が減った MULTIPOLYGON (さらに内側リング数や頂点数が減っている場合あり)、
- MULTIPOLYGON EMPTY。

ラインストリングタイプの場合には、ラインストリングごとに評価が行われるので、次のような結果が導かれます：

- 元のジオメトリ
- 頂点数の減った LINESTRING、
- LINESTRING EMPTY、
- 要素ラインストリング数の減った MULTILINESTRING (さらに頂点が減っている場合あり)、
- MULTILINESTRING EMPTY。



例：マルチポリゴンを適用した *ST_RemoveSmallParts()*。青い領域が残った領域です。

Availability: 3.5.0

例

```
SELECT ST_AsText(ST_RemoveSmallParts(ST_GeomFromText('MULTIPOLYGON(
    ((60 160, 120 160, 120 220, 60 220, 60 160), (70 170, 70 210, 110 210, 110 170, 70 170)),
    ((85 75, 155 75, 155 145, 85 145, 85 75)),
    ((50 110, 70 110, 70 130, 50 130, 50 110)))'),
    50, 50));
```

```
MULTIPOLYGON(((60 160,120 160,120 220,60 220,60 160)),((85 75,155 75,155 145,85 145,85 75))
```

```
SELECT ST_AsText(ST_RemoveSmallParts(ST_GeomFromText('LINESTRING(10 10, 20 20)'),
    50, 50));
```

```
LINESTRING EMPTY
```

7.5.28 ST_Reverse

ST_Reverse — 頂点の順序を逆にしたジオメトリを返します。

Synopsis

geometry **ST_Reverse**(geometry g1);

説明

どのジオメトリでも使用可能です。頂点の順序を逆にします。

Enhanced: 2.4.0 曲線対応が導入されました。

-  この関数は 3 次元に対応し、Z 値を削除しません。
-  この関数は多面体サーフェスに対応しています。

例

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

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

7.5.29 ST_Segmentize

ST_Segmentize — 与えた長さを超える線分を持たないよう変更したジオメトリ/ジオグラフィを返します。

Synopsis

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

説明

`max_segment_length` を超える長さの線分を持たないように変更したジオメトリ/ジオグラフィを返します。長さは 2 次元で計算されます。線分は常に等長の線分に分割されます。

- ジオメトリについては、最大長の単位は空間参照系の単位です。
- ジオグラフィについては、最大長の単位はメートルです。距離は球面で計算します。追加される頂点は、線分の端点間で定まる球面の大円弧に沿うように作られます。



Note

この関数は長い線分を短くするだけです。最大長より短い線分を長くすることはしません。



Warning

長い線分を含む入力では、比較的短い `max_segment_length` を指定すると、非常に多くの頂点が追加される可能性があります。これは、引数が最大長ではなく線分数として誤って指定された場合に、意図せずに発生する可能性があります。

Availability: 1.2.2

Enhanced: 3.0.0 ジオメトリの分割において、現在は、同じ長さに分割しています

Enhanced: 2.3.0 ジオグラフィの分割において、現在は、同じ長さに分割しています

Enhanced: 2.1.0 ジオグラフィ対応が導入されました。

Changed: 2.1.0 ジオグラフィ対応の導入の結果、`ST_Segmentize('LINESTRING(1 2, 3 4)', 0.5)` とすると、あいまい関数エラーが発生します。入力ではジオメトリかジオグラフィかを確実に指定する必要があります。`ST_GeomFromText`、`ST_GeogFromText`、使いたい型へのキャスト (例: `ST_Segmentize('LINESTRING(1 2, 3 4)::geometry, 0.5)`) を行います

例

ラインの分割。長い線分は均等に分割され、短い線分は分割されません。

```
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))
```

ポリゴンの分割:

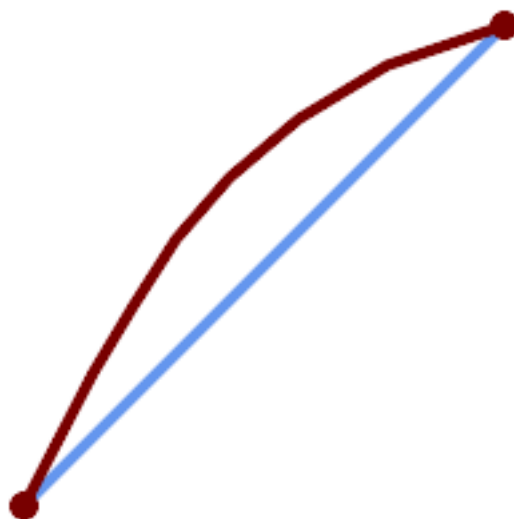
```
SELECT ST_AsText(ST_Segmentize('POLYGON((0 0, 0 8, 30 0, 0 0))'::geometry,
10));
```

```
POLYGON((0 0,0 8,7.5 6,15 4,22.5 2,30 0,20 0,10 0,0 0))
```

2000 キロメートルの最大線分長を使ったジオグラフィのラインの分割。頂点は、端点間をつなぐ大円弧に沿って追加されます。

```
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)
```



大円弧に沿って細分化されたジオグラフィのライン

関連情報

[ST_LineSubstring](#)

7.5.30 ST_SetPoint

ST_SetPoint — ラインストリングのポイントを与えられたポイントに置き換えます。

Synopsis

```
geometry ST_SetPoint(geometry linestring, integer zerobasedposition, geometry point);
```

説明

ラインストリングの N 番目を与えられたポイントに置き換えます。インデックスは 0 はじまりです。負のインデックス値を与えると末尾から数えます。-1 は末尾のポイントを指します。これは、頂点が一つ動いた時に接続のリレーションシップを維持しようとする場合のトリガに特に便利です。

Availability: 1.1.0

Updated 2.3.0 : 添え字の負数

✔ この関数は 3 次元に対応し、Z 値を削除しません。

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)'));
```

```
LINESTRING(-1 1,-1 3)
```

Change last point in a line string (let's 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;
```

```
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
CROSS JOIN ST_PointN(g,1) as p;
```

```
LINESTRING(0 0,1 1,0 0,3 3,4 4)
```

関連情報

[ST_AddPoint](#), [ST_NPoints](#), [ST_NumPoints](#), [ST_PointN](#), [ST_RemovePoint](#)

7.5.31 ST_ShiftLongitude

ST_ShiftLongitude — 経度座標値を-180 度から 180 度の範囲と 0 度から 360 度の範囲との二つの範囲を行き来するようシフトします。

Synopsis

geometry **ST_ShiftLongitude**(geometry geom);

説明

ジオメトリの全てのポイント/頂点を読み、経度値が-180 度から 0 度の範囲にあるのを 180 度から 360 度の範囲にシフトして、180 度から 360 度の範囲にあるのを-180 度から 0 度の範囲にシフトします。この関数は対称となりますので、-180 度から 180 度の範囲のデータを 0 度から 360 度の範囲の表現にし、0 度から 360 度の範囲のデータを-180 度から 180 度の範囲の表現にします。



Note

この関数は、SRID 436 (WGS84 地理座標系) のような緯度経度の座標値を持つデータの場合に限って使います。



Warning

1.3.4 より前では MULTIPOINT では動作しないバグがありました。1.3.4 以上では MULTIPOINT でも動作します。

✔ この関数は 3 次元に対応し、Z 値を削除しません。

Enhanced: 2.0.0 多面体サーフェス対応と TIN 対応が導入されました。

ご注意: この関数は 2.0.0 で "ST_Shift_Longitude" から名称変更しました。

✔ この関数は多面体サーフェスに対応しています。

✔ この関数は三角形と不規則三角網 (TIN) に対応しています。

例

This example performs a forward longitude shift on a single point.

```
SELECT ST_AsText(ST_ShiftLongitude('SRID=4326;POINT(270 0)')::geometry)
```

```
POINT(-90 0)
```

This example performs a reverse longitude shift on a single point.

```
SELECT ST_AsText(ST_ShiftLongitude('SRID=4326;POINT(-90 0)')::geometry)
```

```
POINT(270 0)
```

This example shows that LineStrings are shifted only where coordinates require it.

```
SELECT ST_AsText(ST_ShiftLongitude('SRID=4326;LINESTRING(174 12, 182 13)')::geometry)
```

```
LINESTRING(174 12, -178 13)
```

関連情報

[ST_WrapX](#)

7.5.32 ST_WrapX

ST_WrapX — ジオメトリを X 値で回り込ませます。

Synopsis

geometry **ST_WrapX**(geometry geom, float8 wrap, float8 move);

説明

入力ジオメトリを分割して、全ての結果要素が「回り込み ('wrap')」線から 'move' パラメータで決められた方向、すなわち、右側 ('move' が負数) または左側 ('move' が正数) に全ての要素が落ちるように移動させ、最後に再結合します。



Note

経度緯度入力を「再センタリング」して、対象地物が一方からもう一方に飛ばないようにするのに使えます。

Availability: 2.3.0 GEOS が必要です。

✔ この関数は 3 次元に対応し、Z 値を削除しません。

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);
```

関連情報

[ST_ShiftLongitude](#)

7.5.33 ST_SnapToGrid

ST_SnapToGrid — 入力ジオメトリの全ての点を規則的なグリッドにスナップします。

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);
```

説明

1, 2, 3 番目の形式では、入力ジオメトリの全てのポイントを原点とセルサイズを定めたグリッドにスナップします。同じセルに落ちた、連続するポイントを削除します。引数ジオメトリのジオメトリタイプを定義できないポイントしか残らなかった場合は、NULL を返します。コレクション内で崩壊したジオメトリはそこから削除されます。精度を落とすのに使います。

4 番目の形式は、1.1.0 で導入されました。入力ジオメトリの全てのポイントを原点 (第 2 引数で指定するもので、ポイントでなければなりません) とセルサイズを定めたグリッドにスナップします。グリッドにスナップしたくない次元についてはサイズに 0 を指定します。



Note

返されるジオメトリは単純性が失われているかも知れません ([ST_IsSimple](#)を参照してください)。



Note

1.1.0 版より前では、この関数は常に 2 次元ジオメトリを返しました。1.1.0 版からは、返されるジオメトリの次元数は、入力値のうちで手のつけられていない最大の次元と同じになります。全てのグリッドの次元を定義するには、第 2 引数にジオメトリを取る形式を使って下さい。

Availability: 1.0.0RC1

Availability: 1.1.0 - Z 値と M 値に対応しました



この関数は 3 次元に対応し、Z 値を削除しません。

例

This example snaps geometries to a precision grid of 10^{-3} .

```
UPDATE mytable
SET geom = ST_SnapToGrid(geom, 0.001);

SELECT ST_AsText(ST_SnapToGrid(ST_GeomFromText('LINESTRING(1.1115678 2.123, 4.111111 3.2374897, 4.11112 3.23748667)'),
                                0.001));
```

LINESTRING(1.112 2.123,4.111 3.237)

This example snaps a 4D geometry.

```
SELECT ST_AsEWKT(ST_SnapToGrid(ST_GeomFromEWKT('LINESTRING(-1.1115678 2.123 2.3456 1.11111,
4.111111 3.2374897 3.1234 1.1111, -1.1111112 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));
```

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

With a 4D geometry - the ST_SnapToGrid(geom,size) only touches x and y coords but keeps m and z the same.

```
SELECT ST_AsEWKT(ST_SnapToGrid(ST_GeomFromEWKT('LINESTRING(-1.1115678 2.123 3 2.3456,
4.111111 3.2374897 3.1234 1.1111)'),
0.01)
);
```

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

関連情報

[ST_Snap](#), [ST_AsEWKT](#), [ST_AsText](#), [ST_GeomFromText](#), [ST_GeomFromEWKT](#), [ST_Simplify](#)

7.5.34 ST_Snap

ST_Snap — 入力ジオメトリの辺と頂点を参照ジオメトリの頂点にスナップします。

Synopsis

geometry **ST_Snap**(geometry input, geometry reference, float tolerance);

説明

ジオメトリの頂点と辺を、もう一つのジオメトリの頂点にスナップします。スナップが実行される位置を制御するにはスナップ距離許容値を使います。結果ジオメトリはスナップされた頂点を持つ入力ジオメトリです。スナップが発生しなかった場合には、入力ジオメトリが変更されずに返されます。

一つのジオメトリからもう一つへの変換によって、近傍エッジ (ノード生成とインタセクション計算で問題を引き起こします) を除くことになり、オーバーレイ演算のロバスト性が改善されます。

あまりに多数のスナップを行った場合には、生成されるトポロジが不正になる可能性があります。いつスナップが安全かを判定するために、ヒューリスティックにスナップされた頂点の数と位置が決めるしかありません。しかし、省略された潜在的なスナップになりえます。

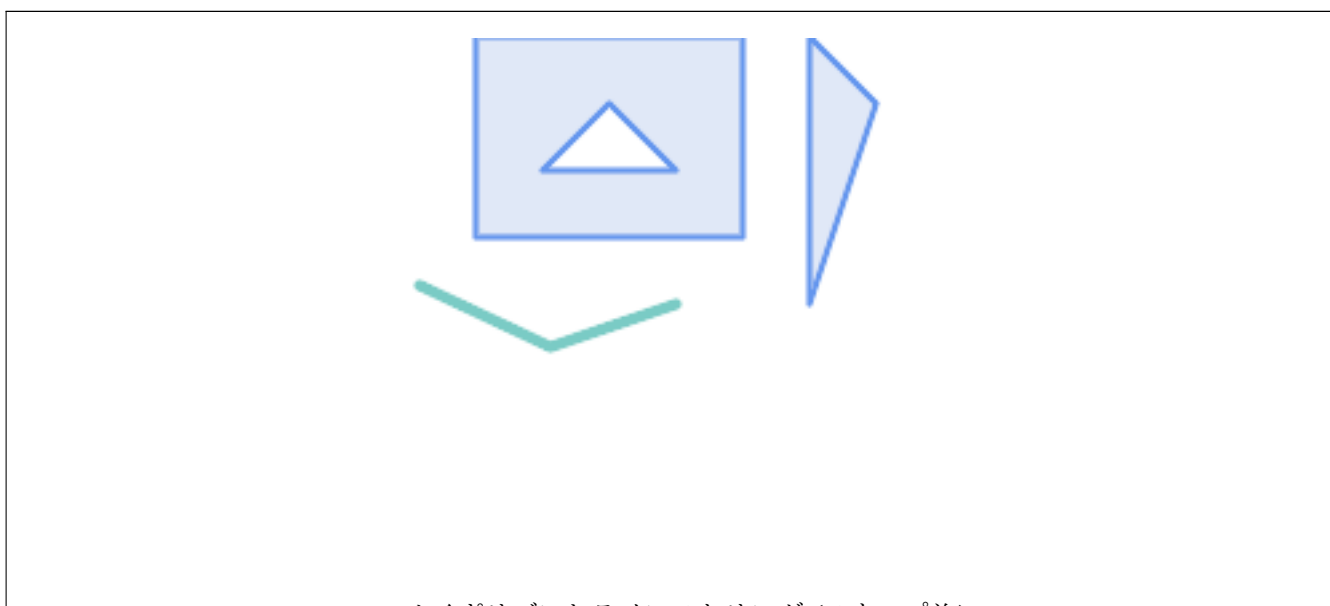
**Note**

返されるジオメトリは単純性が失われているかも知れません ([ST_IsSimple](#)を参照してください) し、妥当性が失われているかも知れません ([ST_IsValid](#)を参照して下さい)。

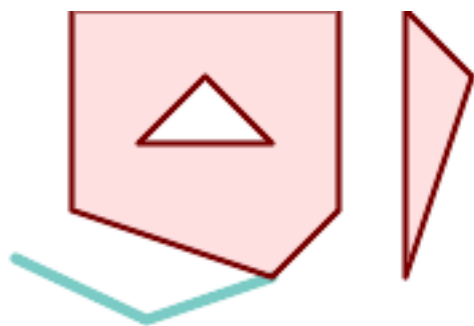
GEOS モジュールで実現しています。

Availability: 2.0.0

例



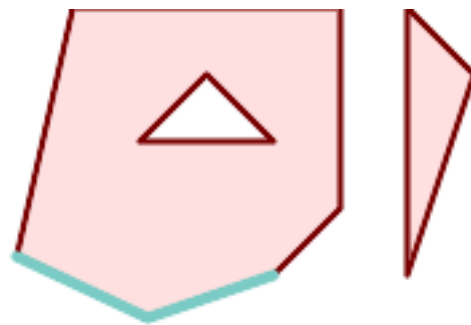
マルチポリゴンとラインストリング (スナップ前)



ラインストリングに 1.01 の許容距離でマルチポリゴンをスナップします。新しいマルチポリゴンはラインストリングにつながります。

```
SELECT ST_AsText(ST_Snap(poly,line, ←
    ST_Distance(poly,line)*1.01)) AS polysnapped
FROM (SELECT
    ST_GeomFromText('MULTIPOLYGON(
        ((26 125, 26 200, 126 200, 126 125, ←
        26 125 ),
        ( 51 150, 101 150, 76 175, 51 150 ) ←
        ),
        (( 151 100, 151 200, 176 175, 151 ←
        100 )))') As poly,
    ST_GeomFromText('LINESTRING (5 ←
    107, 54 84, 101 100)') As line
    ) As foo;

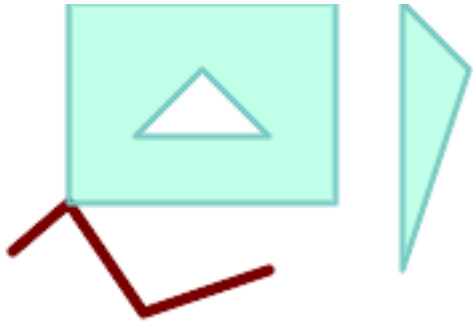
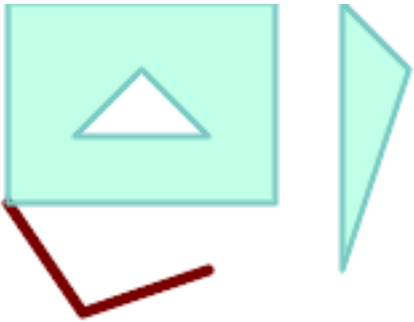
MULTIPOLYGON(((26 125,26 200,126 200,126 ←
    125,101 100,26 125),
    (51 150,101 150,76 175,51 150)),((151 ←
    100,151 200,176 175,151 100)))
```



マルチラインストリングに 1.25 の許容距離でマルチポリゴンをスナップします。新しいマルチポリゴンはラインストリングにつながります。

```
SELECT ST_AsText(ST_Snap(poly,line, ←
    ST_Distance(poly,line)*1.25)
    ) AS polysnapped
FROM (SELECT
    ST_GeomFromText('MULTIPOLYGON(
        (( 26 125, 26 200, 126 200, 126 125, ←
        26 125 ),
        ( 51 150, 101 150, 76 175, 51 150 ) ←
        ),
        (( 151 100, 151 200, 176 175, 151 ←
        100 )))') As poly,
    ST_GeomFromText('LINESTRING (5 ←
    107, 54 84, 101 100)') As line
    ) As foo;

MULTIPOLYGON(((5 107,26 200,126 200,126 ←
    125,101 100,54 84,5 107),
    (51 150,101 150,76 175,51 150)),((151 ←
    100,151 200,176 175,151 100)))
```

	
元のマルチポリゴンに 1.01 の許容距離でラインストリングをスナップします。新しいラインストリングはマルチポリゴンにつながります。 <pre>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;</pre> LINESTRING(5 107,26 125,54 84,101 100)	元のマルチポリゴンに 1.25 の許容距離でラインストリングをスナップします。新しいラインストリングはマルチポリゴンにつながります。 <pre>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;</pre> LINESTRING(26 125,54 84,101 100)

関連情報

[ST_SnapToGrid](#)

7.5.35 ST_SwapOrdinates

ST_SwapOrdinates — 与えられたジオメトリにおいて与えられた座標の値を入れ替えたジオメトリを返します。

Synopsis

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

説明

与えられたジオメトリにおいて与えられた座標の値を入れ替えたジオメトリを返します。

ords 引数は 2 文字の文字列で、入れ替える座標名を示します。座標名は **x**, **y**, **z**, **m** が有効です。

Availability: 2.2.0

- ✔ このメソッドは曲線ストリングと曲線に対応しています。
- ✔ この関数は 3 次元に対応し、Z 値を削除しません。
- ✔ この関数は M 値に対応します。
- ✔ この関数は多面体サーフェスに対応しています。
- ✔ この関数は三角形と不規則三角網 (TIN) に対応しています。

例

This example scales the M value by 2.

```
SELECT ST_AsText(ST_SwapOrdinates(ST_Scale(ST_SwapOrdinates(g, 'xm'),
    2, 1
),
    'xm')
) FROM ( SELECT 'POINT ZM (0 0 0 2)::geometry g ) foo;
```

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

関連情報

[ST_FlipCoordinates](#)

7.6 ジオメトリ検証

7.6.1 ST_IsValid

ST_IsValid — ジオメトリが 2 次元で整形されているかのテスト。

Synopsis

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

説明

ST_Geometry 値が整形式であり、2 次元で妥当かどうかを OGC 規則に沿ってテストします。3 次元と 4 次元のジオメトリでは、評価は 2 次元で行います。ジオメトリが不正なら、PostgreSQL NOTICE が出力され、不正である理由の詳細が示されます。

flags パラメータを持つ版が対応する値については[ST_IsValidDetail](#)で説明しています。

ジオメトリ評価に関する詳細情報については [Section 4.4](#)を参照してください。

**Note**

SQL-MM では、ST_IsValid(NULL) は 0 を返しますが、PostGIS では NULL を返します。

GEOS モジュールで実現しています。

フラグを受け付ける形式は、2.0.0 から有効になりました。



このメソッドはOGC Simple Features Implementation Specification for SQL 1.1の実装です。



このメソッドは SQL/MM 仕様の実装です。SQL-MM 3: 5.1.9

**Note**

OGC-SFS も SQL-MM も ST_IsValid でフラグ引数を含む仕様になっていません。フラグは PostGIS 独自拡張です。

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;
```

```
NOTICE: Self-intersection at or near point 0 0
```

```
good_line | bad_poly
-----+-----
t         | f
```

関連情報

[ST_IsSimple](#), [ST_IsValidReason](#), [ST_IsValidDetail](#),

7.6.2 ST_IsValidDetail

ST_IsValidDetail — ジオメトリが妥当か、妥当でないなら理由と位置をそれぞれ示す `valid_detail` 行を返します。

Synopsis

```
valid_detail ST_IsValidDetail(geometry geom, integer flags);
```

説明

`valid_detail` 行を返します。これには、ジオメトリが妥当かどうかを示す真偽値 (`valid`)、不正である理由を示す文字列 (`reason`)、不正である位置を指摘するジオメトリ (`location`) からなります。

不正ジオメトリの詳細報告の生成をする [ST_IsValid](#) と [ST_IsValidReason](#) の組み合わせを改善するために使います。

任意パラメータ `flags` はビットフィールドです。次の値を持つことができます。

- 0: 通常の OGC SFS 評価モデルを使用します。
- 1: ある種の自己接触リング (逆の外リングと逆の穴リング) を妥当とします。この評価モデルはこれらのツールで使われるため「ESRI フラグ」とも言われます。OGC モデルでは不正とされることに注意してください。

GEOS モジュールで実現しています。

Availability: 2.0.0

例

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

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

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

```
--simple example
SELECT * FROM ST_IsValidDetail('LINESTRING(220227 150406,2220227 150407,222020 150410)');
```

valid	reason	location
t		

関連情報

[ST_IsValid](#), [ST_IsValidReason](#)

7.6.3 ST_IsValidReason

ST_IsValidReason — ジオメトリが妥当か否かを示す文字列を返し、不正な場合は理由を返します。

Synopsis

```
text ST_IsValidReason(geometry geomA);
text ST_IsValidReason(geometry geomA, integer flags);
```

説明

ジオメトリが妥当かどうか、不正な場合はその理由を示す文字列を返します。

不正なジオメトリと理由の詳細報告を生成するのに、**ST_IsValid**と併用します。

許される flags は、**ST_IsValidDetail**にあります。

GEOS モジュールで実現しています。

Availability: 1.4

Availability: 2.0 フラグを取る形式。

例

This example uses an invalid bow-tie polygon.

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

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

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

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



```

gid |      validity_info
-----+-----
5330 | Self-intersection [32 5]
5340 | Self-intersection [42 5]
5350 | Self-intersection [52 5]

--simple example
SELECT ST_IsValidReason('LINESTRING(220227 150406,220227 150407,22020 150410)');

st_isvalidreason
-----
Valid Geometry

```

関連情報

[ST_IsValid](#), [ST_Summary](#)

7.6.4 ST_MakeValid

ST_MakeValid — 頂点を失うことなしに不正なジオメトリを妥当なジオメトリにしようと試みます。

Synopsis

```

geometry ST_MakeValid(geometry input);
geometry ST_MakeValid(geometry input, text params);

```

説明

与えられた不正なジオメトリを、入力ジオメトリの頂点を捨てずに、妥当な表現で生成しようとします。妥当なジオメトリは変更せずに返します。

対応する入力は POINTS, MULTIPOINTS, LINESTRING, MULTILINESTRING, POLYGON, MULTIPOLYGON, GEOMETRYCOLLECTION およびそれらの混交したものです。

完全または部分的な次元崩壊の場合には、出力ジオメトリは、同次元または低い次元のジオメトリのコレクションか、低い次元のジオメトリコレクションになります。

単一ポリゴンは、自己インタセクトがある場合には、マルチポリゴンになります。

params 引数は、妥当なジオメトリを構築するために使うメソッドを選択するための任意文字列を与えるのに使えます。任意文字列は `"method=linework|structure keepcollapsed=true|false"` という書式に従います。**"params"** 引数がない場合には、**"linework"** アルゴリズムがデフォルトとして使われます。

"method" キーに対して取り得る値は二つあります。

- **"linework"** はデフォルトのアルゴリズムで、最初に全てのラインを抽出して、そのラインを繋ぎ合わせ、最後に繋いだラインから値を出力します。頂点の欠落が無いことを求めているので、複雑なコレクションを出力として生成できます。
- **"structure"** は内側リングと外側リングを区別します。外側リングを結合してジオメトリを新規生成してから、全ての内側リングの領域を差し引きます。次元崩壊したラインは破棄されて結果が出力されるので、より直感的と言えるでしょう。

"keepcollapsed" キーは **"structure"** アルゴリズムの時だけ見られて、**"true"** または **"false"** を取ります。

- "keepcollapsed" のデフォルト値は"false" です。頂点が 1 個のラインストリングなどの低い空間次元に崩壊するジオメトリーは削除されます。
- "keepcollapsed" が"true" の場合には、低い空間次元に崩壊するジオメトリー要素も保持されます。

GEOS モジュールで実現しています。

Availability: 2.0.0

Enhanced: 2.0.1 速度の改善

Enhanced: 2.1.0 GEOMETRYCOLLECTION と MULTIPOINT の対応の追加

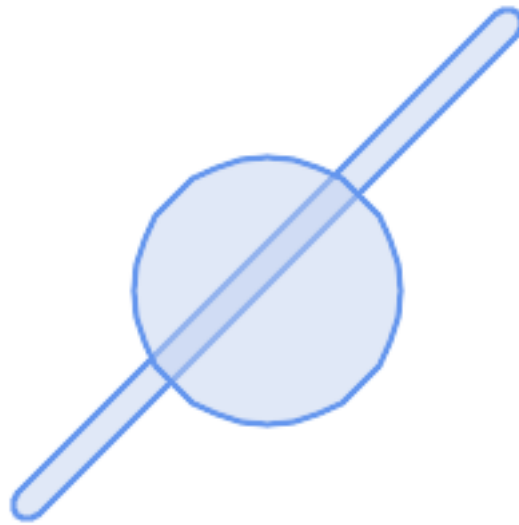
Enhanced: 3.1.0 NaN 値を持つ座標の削除が追加されました。

Enhanced: 3.2.0, アルゴリズムに関する任意パラメータ'linework' と'structure' が追加されました。GEOS 3.10.0 以上が必要です。

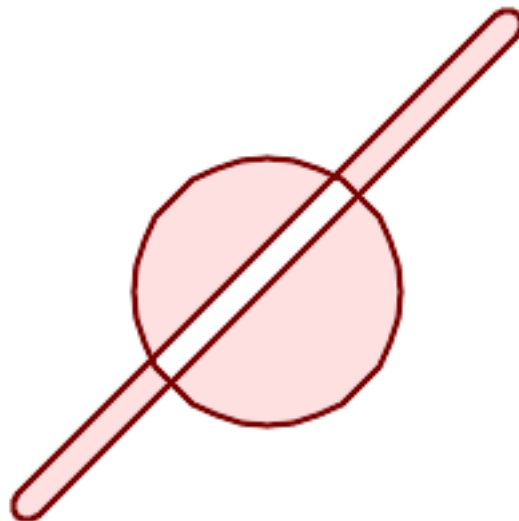


この関数は 3 次元に対応し、Z 値を削除しません。

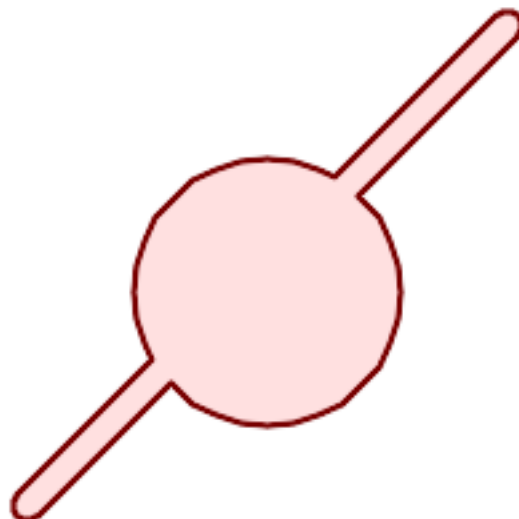
例



before_geom: 二つのオーバーラップしているポリゴンからなるマルチポリゴン

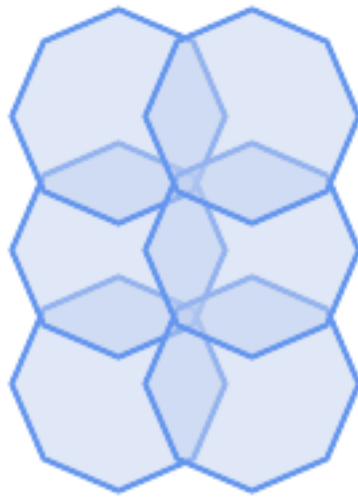


after_geom: 4つのオーバーラップしないポリゴンのマルチポリゴン

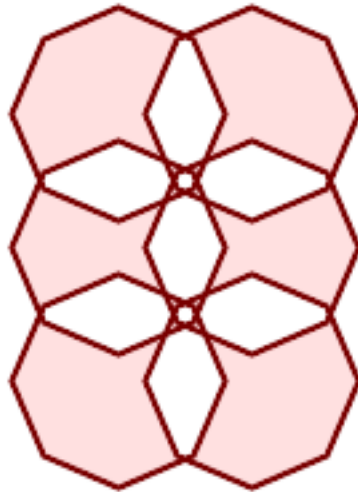


after_geom_structure: 一つのオーバーラップしないポリゴンからなるマルチポリゴン

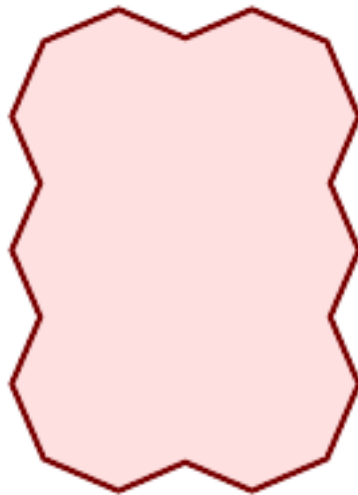
```
SELECT f.geom AS before_geom, ST_MakeValid(f.geom) AS after_geom, ST_MakeValid(f.geom, ←
  'method=structure') AS after_geom_structure
FROM (SELECT 'MULTIPOLYGON(((186 194,187 194,188 195,189 195,190 195,
191 195,192 195,193 194,194 194 194,194 193,195 192,195 191,
```

before_geom: 六つのオーバーラップしているポリゴンからなるマルチポリゴン



after_geom: 14 個のオーバーラップしていないポリゴンからなるマルチポリゴン



after_geom_structure: 一つのオーバーラップしないポリゴンからなるマルチポリゴン

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

例

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

```
POINT(0 0)
```

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

```
LINESTRING EMPTY
```

関連情報

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

7.7 空間参照系関数

7.7.1 ST_InverseTransformPipeline

ST_InverseTransformPipeline — 定義した座標変換パイプラインの逆変換を使って、異なる空間参照系に座標値を変換した新しいジオメトリを返します。

Synopsis

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

説明

定義した座標変換パイプラインの逆方向への変換を使って、異なる空間参照系に座標値を変換した新しいジオメトリを返します。

変換パイプラインの記述に関する詳細については[ST_TransformPipeline](#)を参照して下さい。

Availability: 3.4.0

入力ジオメトリの SRID は無視され、任意パラメータ `to_srid` から値が提供されていない場合には出力ジオメトリの SRID は 0 に設定されます。[ST_TransformPipeline](#)を使うときには、パイプラインが順方向に実行されます。`ST_InverseTransformPipeline()`を使うと、パイプラインは逆方向に実行されます。

パイプラインを用いた変換は[ST_Transform](#)の特別版です。ほとんどの場合、`ST_Transform`は、座標系間の変換において正しい演算子を選択します。こちらの方が推奨されます。

例

EPSG:16031 変換を使った WGS 84 経度緯度から UTM31N への変換

This example transforms in the inverse direction.

```
SELECT ST_AsText(  
  ST_InverseTransformPipeline(  
    'POINT(426857.9877165967 5427937.523342293)::geometry,  
    'urn:ogc:def:coordinateOperation:EPSG::16031'  
  )  
) AS wgs_geom;
```

```
POINT(2 48.99999999999999)
```

GDA2020 の例。

This example uses ST_Transform with automatic selection of a conversion pipeline.

```
SELECT ST_AsText(ST_Transform('SRID=4939;POINT(143.0 -37.0)::geometry, 7844)) AS ↔  
  gda2020_auto;
```

```
POINT(143.00000635638918 -36.999986706128176)
```

関連情報

[ST_Transform](#), [ST_TransformPipeline](#)

7.7.2 ST_SetSRID

ST_SetSRID — ジオメトリに SRID を設定します。

Synopsis

geometry **ST_SetSRID**(geometry geom, integer srid);

説明

ジオメトリの SRID を特定の整数値に設定します。クエリのためのバウンディングボックスを生成する際に使います。



Note

この関数はジオメトリを変換せず、ジオメトリが仮定する空間参照系を定義するメタデータを設定するだけです。ジオメトリを新しい投影法に変換したい場合は[ST_Transform](#)を使います。



このメソッドは[OGC Simple Features Implementation Specification for SQL 1.1](#)の実装です。



このメソッドは曲線ストリングと曲線に対応しています。

例

Mark a point as WGS 84 longitude/latitude.

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

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

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

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

```
SRID=3785;POINT(-13732990.8753491 6178458.96425423)
```

関連情報

Section 4.5, [ST_SRID](#), [ST_Transform](#), [UpdateGeometrySRID](#)

7.7.3 ST_SRID

ST_SRID — ジオメトリの空間参照系識別子を返します。

Synopsis

integer **ST_SRID**(geometry g1);

説明

ST_Geometry の spatial_ref_sys テーブルで定義されている空間参照系の識別番号を返します。Section 4.5を参照して下さい。



Note

spatial_ref_sys テーブルは PostGIS が知る参照系の全てのカタログを作っていて、ある空間参照系から他の空間参照系に変換するために使われます。ジオメトリの変換を予定している場合は正しい空間参照系の識別番号を持っているか確認することは重要です。



このメソッドは [OGC Simple Features Implementation Specification for SQL 1.1](#) の実装です。s2.1.1.1



このメソッドは SQL/MM 仕様の実装です。SQL-MM 3: 5.1.5



このメソッドは曲線ストリングと曲線に対応しています。

例

```
SELECT ST_SRID(ST_GeomFromText('POINT(-71.1043 42.315)',4326));  
4326
```


関連情報

Section 4.5, [ST_SetSRID](#), [ST_Transform](#), [ST_SRID](#), [ST_SRID](#)

7.7.4 ST_Transform

ST_Transform — 異なる空間参照系に投影変換された新しいジオメトリを返します。

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);
```

説明

異なる空間参照系に投影変換された新しいジオメトリを返します。変換先空間参照系である `to_srid` は、妥当な SRID 整数パラメータ (`spatial_ref_sys` にあるということ) です。他にも、`to_proj` と `from_proj` に PROJ.4 文字列で定義された空間参照系を指定することができますが、最適化されません。変換先空間参照系が SRID の代わりに PROJ.4 文字列で表現されている場合には、出力ジオメトリの SRID は 0 になります。`from_proj` を使う場合には、入力ジオメトリは定義された SRID を持っていなければなりません。

ST_Transform はしばしば [ST_SetSRID](#) と混同されます。**ST_Transform** は実際にジオメトリの座標を、ある空間参照系から他のものに変換します。**ST_SetSRID** は単にジオメトリの SRID を変更するだけです。

ST_Transform は変換元空間参照系と変換先空間参照系に与える適切な変換パイプラインを自動的に選択します。特定の変換法を使用するには [ST_TransformPipeline](#) を使います。

**Note**

PostGIS は PROJ 対応でコンパイルする必要があります。PROJ 対応でコンパイルしたかを確認するには [PostGIS_Full_Version](#) を使います。

**Note**

一つ以上の変換を行う場合は、インデックスの利点を得るために、使用する変換に関する関数インデックスを持つと便利です。

**Note**

1.3.4 より前では、曲線を含むジオメトリで使用すると、この関数はクラッシュします。これは 1.3.4 以上で訂正されています。

Enhanced: 2.0.0 多面体サーフェス対応が導入されました。

Enhanced: 2.3.0 直接の PROJ.4 文字列への対応が導入されました。



このメソッドは SQL/MM 仕様の実装です。SQL-MM 3: 5.1.6



このメソッドは曲線ストリングと曲線に対応しています。



この関数は多面体サーフェスに対応しています。

例

マサチューセッツ州平面座標系 (アメリカ測量フィート) を WGS84 経度緯度に変更します。

```
SELECT ST_AsText(ST_Transform(ST_GeomFromText('POLYGON((743238 2967416,743238 2967450,
743265 2967450,743265.625 2967416,743238 2967416))',2249),
4326)) As wgs_geom;
```

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

3D Circular String example.

```
SELECT ST_AsEWKT(ST_Transform(ST_GeomFromEWKT('SRID=2249;CIRCULARSTRING(743238 2967416 ↵
1,743238 2967450 2,743265 2967450 3,743265.625 2967416 3,743238 2967416 4)'),
4326));
```

```
SRID=4326;CIRCULARSTRING(-71.1776848522251 42.3902896512902 1,-71.1776843766326 ↵
42.3903829478009 2,
-71.1775844305465 42.3903826677917 3,
-71.1775825927231 42.3902893647987 3,-71.1776848522251 42.3902896512902 4)
```

部分関数インデックスを作る例です。全てのジオメトリが入っているとは確信できないテーブルのためには、スペースの節約とインデックスを小さく効率的にするために、NULL ジオメトリを無視する部分インデックスを使うのが最善です。

```
CREATE INDEX idx_geom_26986_parcel
ON parcels
USING gist
(ST_Transform(geom, 26986))
WHERE geom IS NOT NULL;
```

PROJ.4 テキストを使って、独自の空間参照系に投影変換する例です。

Find intersection of two polygons near the North pole, using a custom Gnomonic projection. See <http://boundlethe-peel/>.

```
WITH data AS (
SELECT
ST_GeomFromText('POLYGON((170 50,170 72,-130 72,-130 50,170 50))', 4326) AS p1,
ST_GeomFromText('POLYGON((-170 68,-170 90,-141 90,-141 68,-170 68))', 4326) AS p2,
'+proj=gnom +ellps=WGS84 +lat_0=70 +lon_0=-160 +no_defs'::text AS gnom
)
SELECT ST_AsText(ST_Transform(ST_Intersection(ST_Transform(p1, gnom), ST_Transform(p2, ↵
gnom)),
gnom, 4326))
FROM data;
```

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

変換の挙動の設定

グリッドシフトを含む座標変換は、ときどき失敗します。たとえば、PROJ.4 にグリッドシフトファイルを付けてビルドされていなかった場合や、座標がグリッドシフト定義の範囲内に無い、といった場合です。デフォルトでは、PostGIS はグリッドシフトファイルが無い場合はエラーを投げますが、この挙動は、PROJ.4 テキストの

`to_proj` 値の変更を試みたり、`spatial_ref_sys` テーブルの `proj4text` 値を変更したりすることで、SRID 毎の原則を設定することができます。

たとえば、`proj4text` パラメータ `+datum=NAD87` は次に示す `+nadgrids` パラメータの短縮形です。

```
+nadgrids=@conus,@alaska,@ntv2_0.gsb,@ntv1_can.dat
```

接頭辞 `@` は、ファイルが無くてもエラー報告をしないという意味ですが、適切だった (発見されてオーバーラップした) ファイルがないままりストの終わりに達した場合はエラーが出ます。

逆に、少なくとも標準的なファイルが確実にあって欲しいけれども、該当が無いまま全てのファイルが走査された場合に、`NULL` 変換としたいなら、次が使えます。

```
+nadgrids=@conus,@alaska,@ntv2_0.gsb,@ntv1_can.dat,null
```

`NULL` グリッドシフトファイルは、世界全体をカバーして、シフトを行わない、妥当なグリッドシフトファイルです。完全な例のために、正しい範囲にない SRID 4267 への変換でエラーが投げられないよう PostGIS を変えたのなら、次のようにします。

```
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;
```

関連情報

Section 4.5, [ST_SetSRID](#), [ST_SRID](#), [UpdateGeometrySRID](#), [ST_TransformPipeline](#)

7.7.5 ST_TransformPipeline

`ST_TransformPipeline` — 定義されている座標変換パイプラインを使用して異なる空間参照系に変換された新しいジオメトリを返します。

Synopsis

geometry **ST_TransformPipeline**(geometry g1, text pipeline, integer to_srid);

説明

定義されている座標変換パイプラインを使用して異なる空間参照系に変換された新しいジオメトリを返します。変換パイプラインは次の文字列書式のいずれかをつかって定義されます。

- `urn:ogc:def:coordinateOperation:AUTHORITY::CODE`。単純な `EPSG:CODE` 文字列は座標操作を一意に識別できません。CRS 定義に同じ EPSG コードが使用できるためです。
- `PROJ` パイプライン文字列: `+proj=pipeline ...`。自動の軸正規化が適用されません。必要なら呼び出し元が追加のパイプライン段階を必要とします。もしくは `axiswap` 段階を削除する必要があります。
- 操作の連結: `urn:ogc:def:coordinateOperation,coordinateOperation:EPSG::3895,coordinateOperat`

Availability: 3.4.0

入力ジオメトリの SRID は無視され、任意パラメータ `to_srid` から値が提供されていない場合には出力ジオメトリの SRID は 0 に設定されます。`ST_TransformPipeline()` を使うときには、パイプラインが順方向に実行されます。[ST_InverseTransformPipeline](#) を使うと、パイプラインは逆方向に実行されます。

パイプラインを用いた変換は [ST_Transform](#) の特別版です。ほとんどの場合、`ST_Transform` は、座標系間の変換において正しい演算子を選択します。こちらの方が推奨されます。

例

EPSG:16031 変換を使った WGS 84 経度緯度から UTM31N への変換

This example transforms in the forward direction.

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

```
POINT(426857.9877165967 5427937.523342293)
```

This example transforms in the inverse direction.

```
SELECT ST_AsText(
  ST_InverseTransformPipeline(
    'POINT(426857.9877165967 5427937.523342293)::geometry',
    'urn:ogc:def:coordinateOperation:EPSG::16031'
  )
) AS wgs_geom;
```

```
POINT(2 48.99999999999999)
```

GDA2020 の例。

This example uses ST_Transform with automatic selection of a conversion pipeline.

```
SELECT ST_AsText(ST_Transform('SRID=4939;POINT(143.0 -37.0)::geometry', 7844)) AS ↵
  gda2020_auto;
```

```
POINT(143.00000635638918 -36.999986706128176)
```

Using a defined conversion (EPSG:8447)

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

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

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

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

関連情報

[ST_Transform](#), [ST_InverseTransformPipeline](#)

7.7.6 postgis_srs_codes

postgis_srs_codes — 指定した機関に関連付けられた SRS コードの一覧を返します。

Synopsis

```
setof text postgis_srs_codes(text auth_name);
```

説明

与えられた auth_name に関する全ての auth_srid の集合を返します。

Availability: 3.4.0

Proj 6 以上

例

EPSG 機関に関連付けられているコードの最初の 10 件の一覧を得ます。

```
SELECT * FROM postgis_srs_codes('EPSG') ORDER BY 1 LIMIT 10;
```

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

関連情報

[postgis_srs](#), [postgis_srs_all](#), [postgis_srs_search](#)

7.7.7 postgis_srs

postgis_srs — 求める機関と空間参照識別子に関するメタデータレコードを返します。

Synopsis

```
setof record postgis_srs(text auth_name, text auth_srid);
```

説明

指定した `auth_name` に関して求める `auth_srid` のメタデータレコードを貸します。レコードには `auth_name`, `auth_srid`, `sname`, `srttext`, `proj4text` と適用範囲の隅を示す `point_sw` と `point_ne` が含まれます。

Availability: 3.4.0

Proj 6 以上

例

EPSG:3005 のメタデータを取得します。

```
SELECT * FROM postgis_srs('EPSG', '3005');
```

```
auth_name | EPSG
auth_srid | 3005
sname     | NAD83 / BC Albers
srttext   | PROJCS["NAD83 / BC Albers", ... ]
proj4text | +proj=aea +lat_0=45 +lon_0=-126 +lat_1=50 +lat_2=58.5 +x_0=1000000 +y_0=0 +
      datum=NAD83 +units=m +no_defs +type=crs
point_sw  | 0101000020E6100000E17A14AE476161C000000000000204840
point_ne  | 0101000020E610000085EB51B81E855CC0E17A14AE47014E40
```

関連情報

[postgis_srs_codes](#), [postgis_srs_all](#), [postgis_srs_search](#)

7.7.8 postgis_srs_all

`postgis_srs_all` — Proj データベース内のあらゆる空間参照系のメタデータレコードを返します。

Synopsis

`postgis_srs_all` レコードの集合 (void);

説明

Proj データベース内にある全てのメタデータレコードを返します。レコードには `auth_name`, `auth_srid`, `sname`, `srttext`, `proj4text` と適用範囲の隅を示す `point_sw` と `point_ne` が含まれます。

Availability: 3.4.0

Proj 6 以上

例

Proj データベースから最初の 10 件のメタデータレコードを得ます。

```
SELECT auth_name, auth_srid, sname FROM postgis_srs_all() ORDER BY auth_name, auth_srid
LIMIT 10;
```

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

関連情報

[postgis_srs_codes](#), [postgis_srs](#), [postgis_srs_search](#)

7.7.9 postgis_srs_search

postgis_srs_search — bounds パラメータを完全に含む適用範囲を持つ投影座標系のメタデータレコードを返します。

Synopsis

setof record **postgis_srs_search**(geometry bounds, text auth_name=EPSG);

説明

bounds パラメータを完全に含む適用範囲を持つ投影座標系のメタデータレコードを返します。レコードには auth_name, auth_srid, sname, srtext, proj4text と適用範囲の隅を示す point_sw と point_ne が含まれます。

この検索では、投影座標系のみが対象となります。お持ちのデータの範囲で動作しうる座標系を探索することを目的としています。

Availability: 3.4.0

Proj 6 以上

例

ルイジアナ州の投影座標系を探索します。

```
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)')
WHERE sname LIKE '%Louisiana%'
ORDER BY auth_name, auth_srid::integer
LIMIT 3;
```

auth_name point_ne	auth_srid	sname	point_sw	↔
-----+-----+-----+-----+-----				
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)				

最大範囲を得るためにテーブルをスキャンして、適していると言えそうな投影座標系を見つけます。

```
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;
```

関連情報

[postgis_srs_codes](#), [postgis_srs_all](#), [postgis_srs](#)

7.8 ジオメトリ入力

7.8.1 Well-Known Text (WKT)

7.8.1.1 ST_BdPolyFromText

ST_BdPolyFromText — マルチラインストリングの Well-Known Text 表現による、閉じたラインストリングの任意のコレクションからポリゴンを生成します。

Synopsis

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

説明

マルチラインストリングの Well-Known Text 表現による、閉じたラインストリングの任意のコレクションからポリゴンを構築します。



Note
WKT が MULTILINESTRING でない場合には、エラーが投げられます。出力が MULTIPOLYGON になる場合には、エラーが投げられますが、この場合は ST_BdMPolyFromText を使うか PostGIS 独特のアプローチとして **ST_BuildArea()** をご覧ください。



このメソッドはOGC Simple Features Implementation Specification for SQL 1.1の実装です。s3.2.6.2 GEOS モジュールで実現しています。

Availability: 1.1.0

関連情報

[ST_BuildArea](#), [ST_BdMPolyFromText](#)

7.8.1.2 ST_BdMPolyFromText

ST_BdMPolyFromText — マルチラインストリングの Well-Known Text 表現による、閉じたラインストリングの任意のコレクションからマルチポリゴンを構築します。

Synopsis

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

説明

マルチラインストリングの Well-Known Text 表現による、閉じたラインストリングの任意のコレクションからマルチポリゴンを構築します。



Note

WKT が MULTILINESTRING でない場合には、エラーが投げられます。出力が単一のポリゴンであってもマルチポリゴンに強制されます。単一のポリゴンが返って欲しい場合は[ST_BdPolyFromText](#)を使うか PostGIS 独特のアプローチとして[ST_BuildArea\(\)](#)をご覧ください。



このメソッドはOGC Simple Features Implementation Specification for SQL 1.1の実装です。s3.2.6.2 GEOS モジュールで実現しています。

Availability: 1.1.0

関連情報

[ST_BuildArea](#), [ST_BdPolyFromText](#)

7.8.1.3 ST_GeogFromText

ST_GeogFromText — Well-Known Text 表現または拡張 WKT から指定したジオグラフィ値を返します。

Synopsis

geography **ST_GeogFromText**(text EWKT);

説明

Well-Known Text 表現または拡張 WKT から指定したジオグラフィ値を返します。SRID 4326 を仮定します。この関数は ST_GeographyFromText の別名です。ポイントは常に経度緯度形式で表現されます。

例

This example converts longitude/latitude coordinates to geography.

```
ALTER TABLE sometable ADD COLUMN geog geography(POINT,4326);
UPDATE sometable SET geog = ST_GeogFromText('SRID=4326;POINT(' || lon || ' ' || lat || ')') ←
;

--- specify a geography point using EPSG:4267, NAD27
SELECT ST_AsEWKT(ST_GeogFromText('SRID=4267;POINT(-77.0092 38.889588)'));
```

関連情報

[ST_AsText](#), [ST_GeographyFromText](#)

7.8.1.4 ST_GeographyFromText

ST_GeographyFromText — Well-Known Text 表現または拡張 WKT から指定したジオグラフィ値を返します。

Synopsis

geography **ST_GeographyFromText**(text EWKT);

説明

Well-Known Text 表現から指定したジオグラフィ値を返します。SRID 4326 を仮定します。

関連情報

[ST_GeogFromText](#), [ST_AsText](#)

7.8.1.5 ST_GeomCollFromText

ST_GeomCollFromText — WKT 表現と与えられた SRID からジオメトリのコレクションを生成します。SRID が与えられていない場合は 0 とします。

Synopsis

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

説明

Well-Kown-Text (WKT) 表現のコレクションと与えられた SRID からジオメトリのコレクションを生成します。SRID が与えられていない場合は 0 とします。

OGC SPEC 3.2.6.2 - 任意引数 SRID は仕様適合のためです。

WKT が GEOMETRYCOLLECTION でない場合には、NULL を返します。

**Note**

全ての WKT ジオメトリがジオメトリコレクションであると絶対的な確信を持てる場合は、この関数は使わないでください。付加的な評価ステップが追加されるので ST_GeomFromText より遅いです。



このメソッドは **OGC Simple Features Implementation Specification for SQL 1.1** の実装です。s3.2.6.2



このメソッドは SQL/MM 仕様の実装です。

例

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

関連情報

ST_GeomFromText, **ST_SRID**

7.8.1.6 ST_GeomFromEWKT

ST_GeomFromEWKT — 拡張 Well-Known Text 表現 (EWKT) から指定された ST_Geometry 値を返します。

Synopsis

geometry **ST_GeomFromEWKT**(text EWKT);

説明

拡張 Well-Known Text 表現 (EWKT) から PostGIS ST_Geometry オブジェクトを生成します。

**Note**

EWKT 書式は OGC 標準ではなく PostGIS 独特の書式で、空間参照系 ID (SRID) を含みます。

Enhanced: 2.0.0 多面体サーフェス対応と TIN 対応が導入されました。



この関数は 3 次元に対応し、Z 値を削除しません。



このメソッドは曲線ストリングと曲線に対応しています。



この関数は多面体サーフェスに対応しています。



この関数は三角形と不規則三角網 (TIN) に対応しています。

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,-71.1038446938243 42.3151006300338,
-71.1038315271889 42.315094347535,-71.1037393329282 42.315054824985,
-71.1035447555574 42.3152608696313,-71.1033436658644 42.3151648370544,
-71.1032580383161 42.3152269126061,-71.103223066939 42.3152517403219,
-71.1031880899493 42.3152774590236)),
((-71.1043632495873 42.315113108546,-71.1043583974082 42.3151211109857,
-71.1043443253471 42.3150676015829,-71.1043850704575 42.3150793250568,-71.1043632495873 42.315113108546)))');
```

3D circular string.

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

This example uses a PolyhedralSurface.

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

関連情報

[ST_AsEWKT](#), [ST_GeomFromText](#)

7.8.1.7 ST_GeomFromMARC21

ST_GeomFromMARC21 — MARC21/XML 地理データを入力に取り、PostGIS ジオメトリオブジェクトを返します。

Synopsis

```
geometry ST_GeomFromMARC21 ( text marcxml );
```

説明

この関数は MARC21/XML データから PostGIS ジオメトリを生成するものです。POINT または POLYGON を含みます。同じ MARC21/XML データで複数の地理データが存在する場合には、MULTIPOINT または MULTIPOLYGON を返します。データにジオメトリタイプが混じっている場合には、GEOMETRYCOLLECTION を返します。MARC21/XML データに地理データ (データフィールド:034) を含まない場合には NULL を返します。対応する LOC MARC21/XML のバージョンは次の通りです。

- [MARC21/XML 1.1](#)

Availability: 3.3.0 libxml2 2.6 以上が必要です。



Note

MARC21/XML コード化地理数値データでは、今のところコード化した座標の空間参照系を記述する手段がないため、この関数は常に SRID 0 のジオメトリを返すことになります。



Note

返される POLYGON ジオメトリは常に時計回り方向です。

例

POINT を一つ含み hddd.dddddd でエンコードした MARC21/XML 地理データの変換

```
SELECT
  ST_AsText(ST_GeomFromMARC21('
    <record xmlns="http://www.loc.gov/MARC21/slim">
      <leader
>000000nz a2200000nc 4500</leader>
      <controlfield tag="001"
>040277569</controlfield>
      <datafield tag="034" ind1=" " ind2=" ">
        <subfield code="d"
>W004.500000</subfield>
        <subfield code="e"
>W004.500000</subfield>
        <subfield code="f"
>N054.250000</subfield>
        <subfield code="g"
>N054.250000</subfield>
      </datafield>
    </record>
  >'));
```

POINT(-4.5 54.25)

POLYGON を一つ含み hddmmss でエンコードした MARC21/XML 地理データの変換

```
SELECT
  ST_AsText(ST_GeomFromMARC21('
    <record xmlns="http://www.loc.gov/MARC21/slim">
      <leader
>01062cem a2200241 a 4500</leader>
      <controlfield tag="001"
> 84696781 </controlfield>
      <datafield tag="034" ind1="1" ind2=" ">
        <subfield code="a"
>a</subfield>
        <subfield code="b"
>50000</subfield>
        <subfield code="d"
>E0130600</subfield>
        <subfield code="e"
>E0133100</subfield>
        <subfield code="f"
>N0523900</subfield>
        <subfield code="g"
>N0522300</subfield>
      </datafield>
    </record>
  >'));
```

POLYGON((13.1 52.65,13.516666666666667 52.65,13.516666666666667 52.38333333333333,13.1 52.38333333333333,13.1 52.65)) ←

POLYGON と POINT を含む MARC21/XML 地理データの変換:

```
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
>' ));

```

```

GEOMETRYCOLLECTION(POLYGON((13.1 52.65,13.516666666666667 52.65,13.516666666666667 52.383333333333333,13.1 52.383333333333333,13.1 52.65)),POINT(-4.5 54.25))

```

関連情報

[ST_AsMARC21](#)

7.8.1.8 ST_GeometryFromText

ST_GeometryFromText — Well-Known Text 表現 (WKT) から指定した ST_Geometry 値を返します。これは ST_GeomFromText の別名です。

Synopsis

```

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

```

説明



このメソッドは [OGC Simple Features Implementation Specification for SQL 1.1](#) の実装です。



このメソッドは SQL/MM 仕様の実装です。SQL-MM 3: 5.1.40

関連情報

[ST_GeomFromText](#)

7.8.1.9 ST_GeomFromText

ST_GeomFromText — Well-Known Text 表現 (WKT) から指定した ST_Geometry を返します。

Synopsis

```

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

```

説明

OGC Well-Known Text 表現から PostGIS ST_Geometry オブジェクトを生成します。

**Note**

ST_GeomFromText 関数には二つの形式があります。一つ目は、SRID を取らず、空間参照系を持たない (SRID=0) ジオメトリを返すものです。二つ目は、SRID を第 2 引数に取り、メタデータの一部として SRID を含むジオメトリを返すものです。



このメソッドは **OGC Simple Features Implementation Specification for SQL 1.1** の実装です。s3.2.6.2 - 任意引数 SRID は仕様適合のためです。



このメソッドは SQL/MM 仕様の実装です。SQL-MM 3: 5.1.40



このメソッドは曲線ストリングと曲線に対応しています。

**Note**

OGC 非準拠ですが、**ST_MakePoint** は ST_GeomFromText や ST_PointFromText より早いです。座標値に数値を使っている場合には簡単になるという利点もあります。他の選択肢として **ST_Point** は、速度面では **ST_MakePoint** と似ていて、かつ OGC 準拠ですが、2 次元ポイントしか対応していません。

**Warning**

Changed: 2.0.0 前の版では ST_GeomFromText('GEOMETRYCOLLECTION(EMPTY)') が許されていました。SQL/MM 標準への適合のため PostGIS 2.0.0 では不正とされます。今は ST_GeomFromText('GEOMETRYCOLLECTION EMPTY') となります。

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)');
```

関連情報

[ST_GeomFromEWKT](#), [ST_GeomFromWKB](#), [ST_SRID](#)

7.8.1.10 ST_LineFromText

ST_LineFromText — WKT 表現と与えられた SRID からジオメトリを生成します。SRID が与えられていない場合は 0 (不明) となります。

Synopsis

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

説明

WKT 表現と与えられた SRID からジオメトリを生成します。SRID が与えられていない場合は 0 (不明) となります。渡された WKT が LINESTRING でない場合には NULL が返ります。



Note

OGC SPEC 3.2.6.2 - 任意引数 SRID は仕様適合のためです。



Note

全てのジオメトリが LINESTRING であると知っている場合は、ST_GeomFromText を使う方が効率的です。この関数は ST_GeomFromText の呼び出しと、LINESTRING を返すかどうかの評価とを行います。



このメソッドは [OGC Simple Features Implementation Specification for SQL 1.1](#) の実装です。s3.2.6.2



このメソッドは SQL/MM 仕様の実装です。SQL-MM 3: 7.2.8

例

```
SELECT ST_AsText(ST_LineFromText('LINESTRING(1 2, 3 4)')) AS aline,
       ST_LineFromText('POINT(1 2)') IS NULL AS null_return;
```

aline	null_return
LINESTRING(1 2,3 4)	t

関連情報

[ST_GeomFromText](#)

7.8.1.11 ST_MLineFromText

ST_MLineFromText — WKT 表現から指定した ST_MultiLineString 値を返します。

Synopsis

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

説明

Well-Kown-Text (WKT) 表現のコレクションと与えられた SRID からジオメトリを生成します。SRID が与えられていない場合は 0 (不明) とします。

OGC SPEC 3.2.6.2 - 任意引数 SRID は仕様適合のためです。

WKT が MULTILINESTRING でない場合は NULL を返します。



Note

全ての WKT ジオメトリがマルチラインストリングであると絶対的な確信を持てる場合は、この関数は使わないでください。付加的な評価ステップが追加されるので ST_GeomFromText より遅いです。

- ✓ このメソッドは [OGC Simple Features Implementation Specification for SQL 1.1](#) の実装です。s3.2.6.2
- ✓ このメソッドは SQL/MM 仕様の実装です。SQL-MM 3: 9.4.4

例

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

関連情報

[ST_GeomFromText](#)

7.8.1.12 ST_MPointFromText

ST_MPointFromText — Well-Known Text (WKT) 表現と与えられた SRID からジオメトリを生成します。SRID を与えない場合は 0 (不明) となります。

Synopsis

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

説明

Well-Known Text (WKT) 表現と与えられた SRID からジオメトリを生成します。SRID が与えられていない場合は 0 (不明) となります。

OGC SPEC 3.2.6.2 - 任意引数 SRID は仕様適合のためです。

WKT が MULTIPOINT でない場合は NULL を返します。



Note

全ての WKT ジオメトリがマルチラインストリングであると絶対的な確信を持てる場合は、この関数は使わないでください。付加的な評価ステップが追加されるので ST_GeomFromText より遅いです。

- ✔ このメソッドは [OGC Simple Features Implementation Specification for SQL 1.1](#) の実装です。3.2.6.2
- ✔ このメソッドは SQL/MM 仕様の実装です。SQL-MM 3: 9.2.4

例

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

関連情報

[ST_GeomFromText](#)

7.8.1.13 ST_MPolyFromText

ST_MPolyFromText — Well-Known Text (WKT) 表現と与えられた SRID からマルチポリゴンを生成します。SRID が与えられていない場合は 0 (不明) となります。

Synopsis

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

説明

Well-Known Text (WKT) 表現と与えられた SRID からマルチポリゴンを生成します。SRID が与えられていない場合は 0 (不明) となります。

OGC SPEC 3.2.6.2 - 任意引数 SRID は仕様適合のためです。

WKT が MULTIPOLYGON でない場合はエラーを投げます。

**Note**

全ての WKT ジオメトリがマルチポリゴンであると絶対的な確信を持てる場合は、この関数は使わないでください。付加的な評価ステップが追加されるので ST_GeomFromText より遅いです。

- ✔ このメソッドは **OGC Simple Features Implementation Specification for SQL 1.1** の実装です。s3.2.6.2
- ✔ このメソッドは SQL/MM 仕様の実装です。SQL-MM 3: 9.6.4

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);
```

関連情報

ST_GeomFromText, ST_SRID

7.8.1.14 ST_PointFromText

ST_PointFromText — WKT と与えられた SRID からポイントジオメトリを生成します。SRID が与えられていない場合は 0 (不明) とします。

Synopsis

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

説明

OGC Well-Known Text 表現から PostGIS の ST_Geometry ポイントオブジェクトを生成します。SRID が与えられていない場合は不明 (現在は 0) とします。ジオメトリが WKT ポイント表現でない場合は NULL を返します。完全に不正な WKT ならエラーが投げられます。



Note

ST_PointFromText には二つの形式があります。一つ目は、SRID を取らずに空間参照系を定義していないジオメトリを返すものです。二つ目は、空間参照系識別番号を第 2 引数に取り、SRID をメタデータの一部として含む ST_Geometry を返すものです。SRID は spatial_ref_sys テーブルで定義されていなければなりません。



Note

全ての WKT ジオメトリがジオメトリコレクションであると絶対的な確信を持てる場合は、この関数は使わないでください。付加的な評価ステップが追加されるので ST_GeomFromText より遅いです。経度緯度座標からポイントを生成していて、OGC 対応よりもパフォーマンスと精度を重視する場合は、**ST_MakePoint**か、OGC 対応の別名である**ST_Point**を使用して下さい。



このメソッドはOGC Simple Features Implementation Specification for SQL 1.1の実装です。s3.2.6.2 - 任意引数 SRID は仕様適合のためです。



このメソッドは SQL/MM 仕様の実装です。SQL-MM 3: 6.1.8

例

```
SELECT ST_PointFromText('POINT(-71.064544 42.28787)');
SELECT ST_PointFromText('POINT(-71.064544 42.28787)', 4326);
```

関連情報

ST_GeomFromText, **ST_MakePoint**, **ST_Point**, **ST_SRID**

7.8.1.15 ST_PolygonFromText

ST_PolygonFromText — Well-Known Text (WKT) 表現と与えられた SRID からジオメトリを生成します。SRID を与えない場合は 0 (不明) となります。

Synopsis

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

説明

WKT 表現と与えられた SRID からジオメトリを生成します。SRID が与えられていない場合は 0 (不明) とします。WKT がポリゴンでない場合は NULL を返します。

OGC SPEC 3.2.6.2 - 任意引数 SRID は仕様適合のためです。

**Note**

全ての WKT ジオメトリがポリゴンであると絶対的な確信を持てる場合は、この関数は使わないでください。付加的な評価ステップが追加されるので ST_GeomFromText より遅いです。



このメソッドは **OGC Simple Features Implementation Specification for SQL 1.1** の実装です。s3.2.6.2



このメソッドは SQL/MM 仕様の実装です。SQL-MM 3: 8.3.6

例

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

```
010300000000100000000500000006...
```

```
SELECT ST_PolygonFromText('POINT(1 2)') IS NULL AS point_is_notpoly;
```

```
t
```

関連情報

[ST_GeomFromText](#)

7.8.1.16 ST_WKTToSQL

ST_WKTToSQL — Well-Known Text 表現 (WKT) から指定した ST_Geometry 値を返します。これは ST_GeomFromText の別名です。

Synopsis

geometry **ST_WKTToSQL**(text WKT);

説明



このメソッドは SQL/MM 仕様の実装です。SQL-MM 3: 5.1.34

関連情報

[ST_GeomFromText](#)

7.8.2 Well-Known Binary (WKB)**7.8.2.1 ST_GeogFromWKB**

ST_GeogFromWKB — Well-Known Binary ジオメトリ表現 (WKB) または拡張 WKB(EWKB) からジオグラフィックインスタンスを生成します。

Synopsis

geography **ST_GeogFromWKB**(bytea wkb);

説明

ST_GeogFromWKB は、ジオメトリの Well-Known Binary 表現 (WKB) または PostGIS 拡張 WKB を得て、適切なジオグラフィ型のインスタンスを生成します。この関数は SQL のジオメトリファクトリの役割を果たします。SRID が指定されていない場合には、デフォルトは 4326 (WGS 84 経度緯度) となります。



このメソッドは曲線ストリングと曲線に対応しています。

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\000\0037\205\353Q\270~\300\323Mb\020
  X\231C@\020X9\264\310~\300\217\302\365\230C@' )
);
```

```
LINESTRING(-113.98 39.198,-113.981 39.195)
```

関連情報

[ST_GeogFromText](#), [ST_AsBinary](#)

7.8.2.2 ST_GeomFromEWKB

ST_GeomFromEWKB — 拡張 Well-Known Binary 表現 (EWKB) から指定した ST_Geometry 値を返します。

Synopsis

geometry **ST_GeomFromEWKB**(bytea EWKB);

説明

拡張 Well-Known Binary 表現 (EWKB) から PostGIS の ST_Geometry オブジェクトを構築します。



Note

EWKB 書式は OGC 標準ではなく PostGIS 独特の書式で、空間参照系識別番号 (SRID) を含みます。

Enhanced: 2.0.0 多面体サーフェス対応と TIN 対応が導入されました。



この関数は 3 次元に対応し、Z 値を削除しません。



このメソッドは曲線ストリングと曲線に対応しています。



この関数は多面体サーフェスに対応しています。



この関数は三角形と不規則三角網 (TIN) に対応しています。

例

NAD83 経度緯度 (4269) の LINESTRING(-71.160281 42.258729,-71.160837 42.259113,-71.161144 42.25932) のバイナリ表現です。



Note

ご注意: byte 配列は、\で区切られ、' を持ちますが、standard_conforming_strings が切られている場合には、\と" でエスケープします。正確には AsEWKB 表現とはあいません。

```
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

PostgreSQL ではデフォルトで standard_conforming_strings が設定されています。単体クエリーでの必要に応じて、またはデータベースやサーバーレベルで、デフォルトを変更することができます。次に standard_conforming_strings = on でどう動作するかを示します。この場合には標準 ANSI の' がエスケープされますがバックスラッシュはエスケープされません

```
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')
```

関連情報

[ST_AsBinary](#), [ST_AsEWKB](#), [ST_GeomFromWKB](#)

7.8.2.3 ST_GeomFromWKB

ST_GeomFromWKB — Well-Known Binary ジオメトリ表現 (WKB) と任意パラメーターの SRID からジオメトリインスタンスを生成します。

Synopsis

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

説明

ST_GeomFromWKB は、ジオメトリの Well-Known Binary 表現と空間参照系識別番号 (SRID) を取り、適切なジオメトリタイプのインスタンスを生成します。この関数は、SQL のジオメトリファクトリの役割を果たします。これは、ST_WKBToSQL の代替名です。

SRID が指定されていない場合、0 (不明) となります。

**Note**

全てのジオメトリが **LINESTRING** であると知っている場合は、**ST_GeomFromWKB**を使う方が効率的です。この関数は**ST_GeomFromWKB**の呼び出しと、**LINESTRING** を返すかどうかの評価とを行います。



このメソッドは **OGC Simple Features Implementation Specification for SQL 1.1**の実装です。s3.2.6.2



このメソッドは SQL/MM 仕様の実装です。SQL-MM 3: 7.2.9

例

```
SELECT ST_AsText(ST_LineFromWKB(ST_AsBinary(ST_GeomFromText('LINESTRING(1 2, 3 4)')))) AS ←
  aline,
  ST_LineFromWKB(ST_AsBinary(ST_GeomFromText('POINT(1 2)')) IS NULL AS null_return;
```

aline	null_return
LINESTRING(1 2,3 4)	t

関連情報

ST_GeomFromWKB, **ST_LinestringFromWKB**

7.8.2.5 ST_LinestringFromWKB

ST_LinestringFromWKB — WKB 表現と与えられた SRID からジオメトリを生成します。

Synopsis

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

説明

ST_LinestringFromWKB は、ジオメトリの Well-Known Binary 表現と空間参照系識別番号 (SRID) を取り、適切なジオメトリタイプのインスタンスを生成します。この場合、**LINESTRING** ジオメトリです。この関数は SQL のジオメトリファクトリの役割を果たします。

SRID が指定されていない場合は 0 (不明) となります。入力 **bytea** が **LINESTRING** を表現していない場合は **NULL** を返します。これは**ST_LineFromWKB**の別名です。

**Note**

OGC SPEC 3.2.6.2 - 任意引数 SRID は仕様適合のためです。

**Note**

全てのジオメトリが **LINESTRING** であると知っている場合は、**ST_GeomFromWKB**を使う方が効率的です。この関数は**ST_GeomFromWKB**の呼び出しと、**LINESTRING** を返すかどうかの評価とを行います。

- ✔ このメソッドはOGC Simple Features Implementation Specification for SQL 1.1の実装です。s3.2.6.2
- ✔ このメソッドは SQL/MM 仕様の実装です。SQL-MM 3: 7.2.9

例

```
SELECT
  ST_AsText(ST_LinestringFromWKB(ST_AsBinary(ST_GeomFromText('LINESTRING(1 2, 3 4)'))
)) AS aline,
  ST_LinestringFromWKB(ST_AsBinary(ST_GeomFromText('POINT(1 2)'))
) IS NULL AS null_return;
```

aline	null_return
LINESTRING(1 2,3 4)	t

関連情報

[ST_GeomFromWKB](#), [ST_LineFromWKB](#)

7.8.2.6 ST_PointFromWKB

ST_PointFromWKB — WKB と与えられた SRID からジオメトリを生成します。

Synopsis

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

説明

ST_PointFromWKB は、ジオメトリの Well-Known Binary 表現と空間参照系識別番号 (SRID) を取り、適切なジオメトリタイプのインスタンスを生成します。この場合、POINT ジオメトリです。この関数は SQL のジオメトリファクトリの役割を果たします。

SRID が指定されていない場合は 0 (不明) となります。入力 bytea が POINT ジオメトリを表現しないなら NULL が返されます。

- ✔ このメソッドはOGC Simple Features Implementation Specification for SQL 1.1の実装です。s3.2.7.2
- ✔ このメソッドは SQL/MM 仕様の実装です。SQL-MM 3: 6.1.9
- ✔ この関数は 3 次元に対応し、Z 値を削除しません。
- ✔ このメソッドは曲線ストリングと曲線に対応しています。

例

```
SELECT
  ST_AsText(ST_PointFromWKB(ST_AsEWKB('POINT(2 5)')::geometry)
);
```

```
POINT(2 5)
```

```
SELECT
  ST_AsText(ST_PointFromWKB(ST_AsEWKB('LINESTRING(2 5, 2 6)')::geometry)
);
```

```
null
```

関連情報

[ST_GeomFromWKB](#), [ST_LineFromWKB](#)

7.8.2.7 ST_WKBToSQL

ST_WKBToSQL — Well-Known Binary 表現 (WKB) から ST_Geometry 値を生成します。これは SRID を取らない ST_GeomFromWKB の別名です。

Synopsis

geometry **ST_WKBToSQL**(bytea WKB);

説明



このメソッドは SQL/MM 仕様の実装です。SQL-MM 3: 5.1.36

関連情報

[ST_GeomFromWKB](#)

7.8.3 その他の書式

7.8.3.1 ST_Box2dFromGeoHash

ST_Box2dFromGeoHash — GeoHash 文字列から BOX2D を返します。

Synopsis

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

説明

GeoHash 文字列から BOX2D を返します。

`precision` が指定されない場合には、`ST_Box2dFromGeoHash` は、入力ジオハッシュ文字列の完全な精度に基づいた BOX2D を返します。

`precision` が指定されている場合には、`ST_Box2dFromGeoHash` は、BOX2D を生成するために、GeoHash からの多数の文字を使用します。低い精度の値では大きな BOX2D を返し、値が大きいほど精度が増します。

Availability: 2.1.0

例

```
SELECT ST_Box2dFromGeoHash('9qqj7nmxcggy4d0dbxqz0');
```

```
BOX(-115.172816 36.114646,-115.172816 36.114646)
```

```
SELECT ST_Box2dFromGeoHash('9qqj7nmxcggy4d0dbxqz0', 0);
```

```
BOX(-180 -90,180 90)
```

```
SELECT ST_Box2dFromGeoHash('9qqj7nmxcggy4d0dbxqz0', 10);
```

```
BOX(-115.17282128334 36.1146408319473,-115.172810554504 36.1146461963654)
```

関連情報

[ST_GeoHash](#), [ST_GeomFromGeoHash](#), [ST_PointFromGeoHash](#)

7.8.3.2 ST_GeomFromGeoHash

`ST_GeomFromGeoHash` — GeoHash 文字列からジオメトリを返します。

Synopsis

geometry **ST_GeomFromGeoHash**(text geohash, integer precision=full_precision_of_geohash);

説明

GeoHash 文字列からジオメトリを返します。ジオメトリは GeoHash バウンディングボックスのポリゴン表現となります。

`precision` を指定しない場合には、`ST_GeomFromGeoHash` は、入力 GeoHash 文字列の最大精度に基づくポリゴンを返します。

`precision` が指定されると、`ST_GeomFromGeoHash` は、ポリゴンを生成するため GeoHash からの多数の文字を使います。

Availability: 2.1.0

例

```
SELECT ST_AsText(ST_GeomFromGeoHash('9qqj7nmxcggy4d0dbxqz0'));
```

```
POLYGON((-115.172816 36.114646,-115.172816 36.114646,-115.172816 36.114646,↔  
36.114646,-115.172816 36.114646))
```

```
SELECT ST_AsText(ST_GeomFromGeoHash('9qqj7nmxcggy4d0dbxqz0', 4));
```

```
POLYGON((-115.3125 36.03515625,-115.3125 36.2109375,-114.9609375 36.2109375,↔  
36.03515625,-115.3125 36.03515625))
```

```
SELECT ST_AsText(ST_GeomFromGeoHash('9qqj7nmxcggy4d0dbxqz0', 10));
```

```
POLYGON((-115.17282128334 36.1146408319473,-115.17282128334 ↔  
36.1146461963654,-115.172810554504 36.1146461963654,-115.172810554504 ↔  
36.1146408319473,-115.17282128334 36.1146408319473))
```

関連情報

[ST_GeoHash](#), [ST_Box2dFromGeoHash](#), [ST_PointFromGeoHash](#)

7.8.3.3 ST_GeomFromGML

ST_GeomFromGML — GML 表現から PostGIS ジオメトリオブジェクトを出力します。

Synopsis

```
geometry ST_GeomFromGML(text geomgml);  
geometry ST_GeomFromGML(text geomgml, integer srid);
```

説明

OGC GML 表現から PostGIS ST_Geometry オブジェクトを生成します。

ST_GeomFromGML は、GML のうちジオメトリ部分でのみ動作します。GML 文書全体に使用しようとするとエラーが投げられます。

サポートされている OGC GML の版は次のとおりです。

- GML 3.2.1 Namespace
- GML 3.1.1 Simple Features profile SF-2 (GML 3.1.0 と 3.0.0 の後方互換)
- GML 2.1.2

OGC GML 標準については、<http://www.opengeospatial.org/standards/gml>をご覧ください。

Availability: 1.5 libxml2 1.6+ が必要です。

Enhanced: 2.0.0 多面体サーフェス対応と TIN 対応が導入されました。

Enhanced: 2.0.0 SRID 任意引数が追加されました。



この関数は 3 次元に対応し、Z 値を削除しません。



この関数は多面体サーフェスに対応しています。



この関数は三角形と不規則三角網 (TIN) に対応しています。

GML は、複合次元 (たとえば、2 次元と 3 次元が同じ **MultiGeometry** 内にある) を許します。PostGIS ジオメトリは許さないで、**ST_GeomFromGML** は、Z 次元が無いジオメトリを一つでも発見すると、ジオメトリ全体を 2 次元に変換します。

GML は同じ **MultiGeometry** 内での複合 SRS をサポートします。PostGIS ではサポートしないので、**ST_GeomFromGML** は、この場合には、全てのサブジオメトリをルートノードの SRS に投影変換します。GML のルートノードに **srsName** 属性が無い場合、関数はエラーを投げます。

ST_GeomFromGML 関数は、明示的な GML 名前空間について杓子定規ではありません。共通使用で名前空間の明示を避けることができます。ただし、GML 内で **XLink** 機能を使いたい場合は必要です。



Note

ST_GeomFromGML は SQL/MM 曲線ジオメトリに対応していません。

例 - **srsName** 属性を持つ単一のジオメトリ

```
SELECT ST_GeomFromGML($$
  <gml:LineString xmlns:gml="http://www.opengis.net/gml"
                    srsName="EPSG:4269">
    <gml:coordinates>
      -71.16028,42.258729 -71.160837,42.259112 -71.161143,42.25932
    </gml:coordinates>
  </gml:LineString>
$$);
```

例 - **XLink** 使用法

```
SELECT ST_GeomFromGML($$
  <gml:LineString xmlns:gml="http://www.opengis.net/gml"
                    xmlns:xlink="http://www.w3.org/1999/xlink"
                    srsName="urn:ogc:def:crs:EPSG::4269">
    <gml:pointProperty>
      <gml:Point gml:id="p1"
        ><gml:pos
        >42.258729 -71.16028</gml:pos
        ></gml:Point>
      </gml:pointProperty>
      <gml:pos
        >42.259112 -71.160837</gml:pos>
      <gml:pointProperty>
        <gml:Point xlink:type="simple" xlink:href="#p1"/>
      </gml:pointProperty>
    </gml:LineString>
$$);
```

例 - 多面体サーフェス

```

SELECT ST_AsEWKT(ST_GeomFromGML('
<gml:PolyhedralSurface xmlns:gml="http://www.opengis.net/gml">
<gml:polygonPatches>
  <gml:PolygonPatch>
    <gml:exterior>
      <gml:LinearRing
><gml:posList srsDimension="3"
>0 0 0 0 0 1 0 1 1 0 1 0 0 0 0</gml:posList
></gml:LinearRing>
      </gml:exterior>
    </gml:PolygonPatch>
    <gml:PolygonPatch>
      <gml:exterior>
        <gml:LinearRing
><gml:posList srsDimension="3"
>0 0 0 0 1 0 1 1 0 1 0 0 0 0 0</gml:posList
></gml:LinearRing>
        </gml:exterior>
      </gml:PolygonPatch>
    <gml:PolygonPatch>
      <gml:exterior>
        <gml:LinearRing
><gml:posList srsDimension="3"
>0 0 0 1 0 0 1 0 1 0 0 1 0 0 0</gml:posList
></gml:LinearRing>
        </gml:exterior>
      </gml:PolygonPatch>
    <gml:PolygonPatch>
      <gml:exterior>
        <gml:LinearRing
><gml:posList srsDimension="3"
>1 1 0 1 1 1 1 0 1 1 0 0 1 1 0</gml:posList
></gml:LinearRing>
        </gml:exterior>
      </gml:PolygonPatch>
    <gml:PolygonPatch>
      <gml:exterior>
        <gml:LinearRing
><gml:posList srsDimension="3"
>0 1 0 0 1 1 1 1 1 1 0 0 1 0 0</gml:posList
></gml:LinearRing>
        </gml:exterior>
      </gml:PolygonPatch>
    <gml:PolygonPatch>
      <gml:exterior>
        <gml:LinearRing
><gml:posList srsDimension="3"
>0 0 1 1 0 1 1 1 1 0 1 1 0 0 1</gml:posList
></gml:LinearRing>
        </gml:exterior>
      </gml:PolygonPatch>
</gml:polygonPatches>
</gml:PolyhedralSurface
>')));

```

```

POLYHEDRALSURFACE(((0 0 0,0 0 1,0 1 1,0 1 0,0 0 0)),
((0 0 0,0 1 0,1 1 0,1 0 0,0 0 0)),
((0 0 0,1 0 0,1 0 1,0 0 1,0 0 0)),
((1 1 0,1 1 1,1 0 1,1 0 0,1 1 0)),
((0 1 0,0 1 1,1 1 1,1 1 0,0 1 0)),

```



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

関連情報

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

7.8.3.4 ST_GeomFromGeoJSON

ST_GeomFromGeoJSON — ジオメトリの GeoJSON 表現を入力として、PostGIS ジオメトリオブジェクトを出力します。

Synopsis

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

説明

GeoJSON 表現から PostGIS ジオメトリオブジェクトを生成します。

ST_GeomFromGeoJSON は、JSON のうちジオメトリ部分でのみ動作します。JSON 文書全体を使おうとするとエラーが投げられます。

Enhanced: 3.0.0 パースされたジオメトリのデフォルトの SRID は、他に指定していない場合には 4326 となります。

Enhanced: 2.5.0 JSON と JSONB の入力を受け付けるようになりました。

Availability: 2.0.0 JSON-C 0.9 以上が必要です。



Note

有効な JSON-C が無い場合には、出力の替りに、エラー通知を得ます。JSON-C を有効にするには `--with-jsondir=/path/to/json-c` をコンフィギュアで指定します。詳細については [Section 2.2.3](#) をご覧ください。



この関数は 3 次元に対応し、Z 値を削除しません。

例

```
SELECT ST_AsText(ST_GeomFromGeoJSON('{"type":"Point","coordinates":[-48.23456,20.12345]}')) AS wkt; ↵
```

```
POINT(-48.23456 20.12345)
```

This example creates a 3D LineString.

```
SELECT ST_AsText(ST_GeomFromGeoJSON('{"type":"LineString","coordinates": [[1,2,3],[4,5,6],[7,8,9]]}')) AS wkt; ↵
```

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

関連情報

[ST_AsText](#), [ST_AsGeoJSON](#), [Section 2.2.3](#)

7.8.3.5 ST_GeomFromKML

ST_GeomFromKML — ジオメトリの KML 表現の入力を取り、PostGIS ジオメトリオブジェクトを出力します。

Synopsis

geometry **ST_GeomFromKML**(text geomkml);

説明

OGC KML 表現から PostGIS ST_Geometry オブジェクトを生成します。

ST_GeomFromKML は、KML のうちジオメトリ部分でのみ動作します。KML 文書全体に使用しようとするエラーが投げられます。

対応する OGC KML の版は次の通りです。

- KML 2.2.0 Namespace

OGC KML 標準については<http://www.opengeospatial.org/standards/kml>をご覧ください。

Availability: 1.5 libxml2 2.6 以上が必要です。



この関数は 3 次元に対応し、Z 値を削除しません。



Note

ST_GeomFromKML 関数は SQL/MM 曲線ジオメトリに対応していません。

例 - **srsName** 属性を持つ単一のジオメトリ

```
SELECT ST_GeomFromKML($$
  <LineString>
    <coordinates>
  >-71.1663,42.2614
    -71.1667,42.2616</coordinates>
  </LineString>
$$);
```

関連情報

[Section 2.2.3](#), [ST_AsKML](#)

7.8.3.6 ST_GeomFromTWKB

ST_GeomFromTWKB — TWKB ("[Tiny Well-Known Binary](#)") ジオメトリ表現からジオメトリインスタンスを生成します。

Synopsis

geometry **ST_GeomFromTWKB**(bytea twkb);

説明

ST_GeomFromTWKB は、TWKB ("**Tiny Well-Known Binary**") ジオメトリ表現を取り、適切なジオメトリタイプとなるインスタンスを生成します。

例

```
SELECT ST_AsText(ST_GeomFromTWKB(ST_AsTWKB('LINESTRING(126 34, 127 35)::geometry')));
```

```
LINESTRING(126 34, 127 35)
```

```
SELECT ST_AsEWKT(ST_GeomFromTWKB(E'\\x620002f7f40dbce4040105'));
```

```
LINESTRING(-113.98 39.198,-113.981 39.195)
```

関連情報

[ST_AsTWKB](#)

7.8.3.7 ST_GMLToSQL

ST_GMLToSQL — GML 表現から指定した **ST_Geometry** 値を返します。これは **ST_GeomFromGML** の別名です。

Synopsis

geometry **ST_GMLToSQL**(text geomgml);
geometry **ST_GMLToSQL**(text geomgml, integer srid);

説明

 このメソッドは SQL/MM 仕様の実装です。SQL-MM 3: 5.1.50 (曲線対応を除く)

Availability: 1.5 libxml2 1.6+ が必要です。

Enhanced: 2.0.0 多面体サーフェス対応と TIN 対応が導入されました。

Enhanced: 2.0.0 SRID 任意引数が追加されました。

関連情報

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

7.8.3.8 ST_LineFromEncodedPolyline

ST_LineFromEncodedPolyline — エンコード化ポリラインからラインストリングを生成します。

Synopsis

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

説明

エンコード化ポリラインからラインストリングを生成します。

任意パラメータ **precision** は、ポリライン符号化の際の桁数を決定するものです。符号化と復号とで同じ値であるべきで、異なる場合には座標が正しくなりません。

<http://developers.google.com/maps/documentation/utilities/polylinealgorithm> を参照して下さい。

Availability: 2.2.0

例

This example creates a LineString from a polyline.

```
SELECT ST_AsEWKT(ST_LineFromEncodedPolyline('_p~iF~ps|U_uLNnqC_mqNvxq`@'));
```

```
SRID=4326;LINESTRING(-120.2 38.5,-120.95 40.7,-126.453 43.252)
```

This example selects a different precision than the one used for polyline encoding.

```
SELECT ST_AsEWKT(ST_LineFromEncodedPolyline('_p~iF~ps|U_uLNnqC_mqNvxq`@',6));
```

```
SRID=4326;LINESTRING(-12.02 3.85,-12.095 4.07,-12.6453 4.3252)
```

関連情報

[ST_AsEncodedPolyline](#)

7.8.3.9 ST_PointFromGeoHash

ST_PointFromGeoHash — GeoHash 文字列からポイントを返します。

Synopsis

point **ST_PointFromGeoHash**(text geohash, integer precision=full_precision_of_geohash);

説明

GeoHash 文字列からポイントを返します。ポイントは GeoHash の中心点を表します。

precision を指定しない場合には、**ST_PointFromGeoHash** は、入力 GeoHash 文字列の最大精度に基づくポイントを返します。

precision を指定した場合には、**ST_PointFromGeoHash** は、ポイント生成のために、GeoHash から多数の文字を使用します。

Availability: 2.1.0

例

```
SELECT ST_AsText(ST_PointFromGeoHash('9qqj7nmxcggy4d0dbxqz0'));
```

```
POINT(-115.172816 36.114646)
```

```
SELECT ST_AsText(ST_PointFromGeoHash('9qqj7nmxcggy4d0dbxqz0', 4));
```

```
POINT(-115.13671875 36.123046875)
```

```
SELECT ST_AsText(ST_PointFromGeoHash('9qqj7nmxcggy4d0dbxqz0', 10));
```

```
POINT(-115.172815918922 36.1146435141563)
```

関連情報

[ST_GeoHash](#), [ST_Box2dFromGeoHash](#), [ST_GeomFromGeoHash](#)

7.8.3.10 ST_FromFlatGeobufToTable

ST_FromFlatGeobufToTable — FlatGeobuf データの構造に基づいてテーブルを生成します。

Synopsis

void **ST_FromFlatGeobufToTable**(text schemaname, text tablename, bytea FlatGeobuf input data);

説明

FlatGeobuf データの構造に基づいてテーブルを生成します (<http://flatgeobuf.org>)。

schema スキーマ名。

table テーブル名。

data 入力 FlatGeobuf データ。

Availability: 3.2.0

7.8.3.11 ST_FromFlatGeobuf

ST_FromFlatGeobuf — FlatGeobuf データを読みます。

Synopsis

setof anyelement **ST_FromFlatGeobuf**(anyelement Table reference, bytea FlatGeobuf input data);

説明

FlatGeobuf データを読みます (<http://flatgeobuf.org>)。ご注意: PostgreSQL の bytea 型は 1GB を超えられません。

tabletype テーブルタイプへの参照。

data 入力 FlatGeobuf データ。

Availability: 3.2.0

7.9 ジオメトリ出力

7.9.1 Well-Known Text (WKT)

7.9.1.1 ST_AsEWKT

ST_AsEWKT — ジオメトリの SRID メタデータが付いた Well-Known Text (WKT) 表現を返します。

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);
```

説明

SRID を前に置いたジオメトリの Well-Known Text 表現を返します。任意引数 *maxdecimaldigits* で、出力で使われる小数点以下の最大桁数を減らすことができます (デフォルトは 15 です)。

EWKT 表現を PostGIS ジオメトリに逆変換するには **ST_GeomFromEWKT** を使用します。



Warning

任意引数 *maxdecimaldigits* を使用することで、出力ジオメトリが不正になる可能性があります。これを回避するには、前もって **ST_ReducePrecision** に適切なグリッドサイズを与えてこれを使用します。



Note

WKT 仕様は SRID を含みません。OGC WKT 書式を得るには **ST_AsText** を使用します。



Warning

WKT 書式は精度を維持しないので、切り捨てを防ぐには、データ転送に **ST_AsBinary** または **ST_AsEWKB** を使用します。

Enhanced: 3.1.0 精度パラメータ (任意) に対応しました。

Enhanced: 2.0.0 ジオグラフィ対応、多面体サーフェス対応、三角形対応、TIN 対応が導入されました。



この関数は 3 次元に対応し、Z 値を削除しません。



このメソッドは曲線ストリングと曲線に対応しています。



この関数は多面体サーフェスに対応しています。



この関数は三角形と不規則三角網 (TIN) に対応しています。

関連情報

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

7.9.2.3 ST_AsHEXEWKB

ST_AsHEXEWKB — ジオメトリの HEXEWKB 表現を (文字列として) 返します。リトルエンディアン (NDR) またはビッグエンディアン (XDR) のどちらかのエンコーディングを使います。

Synopsis

```
text ST_AsHEXEWKB(geometry g1, text NDRorXDR);
text ST_AsHEXEWKB(geometry g1);
```

説明

ジオメトリの HEXEWKB 表現を (文字列として) 返します。リトルエンディアン (NDR) またはビッグエンディアン (XDR) のどちらかのエンコーディングを使います。エンコーディングを指定しない場合は NDR を使います。



Note

Availability: 1.2.2

- ✔ この関数は 3 次元に対応し、Z 値を削除しません。
- ✔ このメソッドは曲線ストリングと曲線に対応しています。

例

```
SELECT ST_AsHEXEWKB(ST_GeomFromText('POLYGON((0 0,0 1,1 1,1 0,0 0))',4326));
```

```
0103000020E6100000010000000500
000000000000000000000000000000
0000000000000000000000000000F03F
000000000000F03F000000000000F03F000000000000F03
F000000000000000000000000000000000000000000000000
```

Which gives the same answer as:

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

7.9.3 その他の書式

7.9.3.1 ST_AsEncodedPolyline

ST_AsEncodedPolyline — ラインストリングジオメトリから符号化したポリラインを返します。

Synopsis

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

説明

符号化したポリラインを返します。この書式は Google Maps では `precision=5` で、Open Source Routing Machine では `precision=5` と `6` を使います。

任意パラメータ `precision` は、ポリライン符号化の際の桁数を決定するものです。符号化と復号とで同じ値であるべきで、異なる場合には座標が正しくなりません。

Availability: 2.2.0

例

基本

```
SELECT ST_AsEncodedPolyline(GeomFromEWKT('SRID=4326;LINESTRING(-120.2 38.5,-120.95 40.7,-126.453 43.252)'));
```

```
_p~iF~ps|U_uLLnnqC_mqNvxq`@
```

ジオグラフィ型のラインストリングとジオグラフィ型のセグメント化したラインストリングを結合して Google Maps に置きます。

This example builds the SQL for a Boston-to-San-Francisco path with segments every 100 kilometers.

```
SELECT ST_AsEncodedPolyline(ST_Segmentize(
    ST_GeogFromText('LINESTRING(-71.0519 42.4935,-122.4483 37.64)'),
    100000)::geometry) As encodedFlightPath;
```

JavaScript コードは、クエリの結果を `$` 変数と置き換えると次のようになります。

```
<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>
```

関連情報

[ST_LineFromEncodedPolyline](#), [ST_Segmentize](#)

7.9.3.2 ST_AsFlatGeobuf

`ST_AsFlatGeobuf` — 行の集合の FlatGeobuf 表現を返します。

Synopsis

```
bytea ST_AsFlatGeobuf(anyelement set row);
bytea ST_AsFlatGeobuf(anyelement row, bool index);
bytea ST_AsFlatGeobuf(anyelement row, bool index, text geom_name);
```

説明

`FeatureCollection` に対応する行の集合の `FlatGeobuf` 表現 (<http://flatgeobuf.org>) を返します。ご注意: PostgreSQL の `bytea` 型は 1GB を超えられません。

`row` 少なくとも一つのジオメトリカラムを持つ行データ。

`index` 空間インデックスの生成の切り替え。デフォルトは `FALSE` です。

`geom_name` 行データにおけるジオメトリカラムのカラム名。NULL の場合には、最初に見つけたジオメトリカラムとします。

Availability: 3.2.0

7.9.3.3 ST_AsGeobuf

`ST_AsGeobuf` — 行集合の `Geobuf` 表現を返します。

Synopsis

```
bytea ST_AsGeobuf(anyelement set row);
bytea ST_AsGeobuf(anyelement row, text geom_name);
```

説明

`FeatureCollection` に対応する行集合の `Geobuf` 表現 (<https://github.com/mapbox/geobuf>) を返します。最適な格納のために最大精度を決定しますが、そのために全ての入力ジオメトリは解析されます。現在の形式での `Geobuf` はストリーム化できないので、完全な出力はメモリ内で組み立てられることに注意して下さい。

`row` 少なくとも一つのジオメトリカラムを持つ行データ。

`geom_name` 行データにおけるジオメトリカラムのカラム名。NULL の場合には、最初に見つけたジオメトリカラムとします。

Availability: 2.4.0

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;
```

```
GAAiEAo0CgwIBBoIAAAAAGIAAAE=
```

7.9.3.4 ST_AsGeoJSON

`ST_AsGeoJSON` — `GeoJSON` 形式のジオメトリまたは地物を返します。

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);
```

説明

ジオメトリについては GeoJSON の "geometry" オブジェクトとして返し、行について GeoJSON の "feature" オブジェクトとして返します。

GeoJSON のジオメトリと地物の表現は、[GeoJSON specifications RFC 7946](#)に準拠しますが、入力ジオメトリの CRS が WGS84 経度緯度 ([EPSG:4326](#), [urn:ogc:def:crs:OGC::CRS84](#)) 以外の時は準拠しません。GeoJSON の geometry オブジェクトに短い CRS SRID 識別子がデフォルトで追加されます。2 次元と 3 次元のジオメトリの両方に対応します。GeoJSON は SFS 1.1 ジオメトリタイプにのみ対応します (たとえば曲線には対応していません)。

`geom_column` パラメータは複数のジオメトリカラムを区別するために使われます。省略すると、レコードの最初のジオメトリカラムに決定されます。逆にパラメータを渡すとカラムの型の調査が省かれます。

`maxdecimaldigits` 引数は、出力で使われる小数部の桁数の最大値を減らすために使われます (デフォルトでは 9)。EPSG:4326 を使っていて、表示専用でジオメトリを出力する場合には、`maxdecimaldigits=6` が、多くの地図で良い選択となります。



Warning

任意引数 `maxdecimaldigits` を使用することで、出力ジオメトリが不正になる可能性があります。これを回避するには、前もって [ST_ReducePrecision](#) に適切なグリッドサイズを与えてこれを使用します。

`options` 引数は、GeoJSON 出力で BBOX または CRS を追加するために使われます。次のようにします。

- 0: オプションなし
- 1: GeoJSON BBOX
- 2: GeoJSON Short CRS (たとえば EPSG:4326)
- 4: GeoJSON Long CRS (たとえば [urn:ogc:def:crs:EPSG:4326](#))
- 8: EPSG:4326 でない場合に GeoJSON Short CRS (デフォルト)

`id_column` パラメータは、返される GeoJSON の地物の "id" メンバの値設定に使われます。GeoJSON RFC によると、これは、主キーのように地物が共通して使用する識別子を持つときは常に * 使うべき * とされています。指定しない場合には、生成される地物に "id" メンバが入りません。ジオメトリ以外のカラムは、潜在的なキーも含めて、地物の "properties" メンバ内に入ります。

入力レコードのカラム名に重複があるとき、"properties" オブジェクトは重複キーを持つことになります。この場合には `ST_AsGeoJSON` は警告を出します。PostgreSQL の `jsonb` は重複キーがあると最後の値だけをキープするためです。

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

GeoJSON は [geojson.io](#) と [geojsonlint.com](#) で、オンラインでのテストと表示が可能です。また、Web マッピングフレームワークで広く対応されています。

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

Availability: 1.3.4

Availability: 1.5.0 ジオグラフィが導入されました。

Changed: 2.0.0 デフォルト引数と名前付き引数に対応しました。

Changed: 3.0.0 レコードの入力に対応しました

Changed: 3.0.0 EPSG:4326 以外の場合の SRID 出力。

Changed: 3.5.0 地物の id を含むカラムを指定できるようになりました

Changed: 3.7.0 キー重複に関する警告の追加



この関数は 3 次元に対応し、Z 値を削除しません。

例

FeatureCollection の生成:

```
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"}}]}
```

Feature の生成:

```
SELECT ST_AsGeoJSON(t.*, id_column =
> 'id')
FROM (VALUES (1, 'one', 'POINT(1 1)::geometry)) AS t(id, name, geom);
```

```
{"type": "Feature", "geometry": {"type": "Point", "coordinates": [1,1]}, "id": 1, "properties": {"name": "one"}}
```

データを GeoJSON 使用に準拠した WGS84 経度緯度に変換するのを忘れないで下さい。

```
SELECT ST_AsGeoJSON(ST_Transform(geom,4326)) FROM fe_edges LIMIT 1;
```

```
{"type": "MultiLineString", "coordinates": [[[ -89.734634999999997, 31.492072000000000],
[ -89.734955999999997, 31.492237999999997]]]}
```

3 次元ジオメトリへの対応:

```
SELECT ST_AsGeoJSON('LINESTRING(1 2 3, 4 5 6)');
```

```
{"type": "LineString", "coordinates": [[[1,2,3],[4,5,6]]]}
```

Options 引数は、GeoJSON 出力で BBOX または CRS を追加するために使われます:

```
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]}
```


関連情報

[ST_GeomFromGeoJSON](#), [ST_ForcePolygonCCW](#), [ST_Transform](#)

7.9.3.5 ST_AsGML

ST_AsGML — GML 第 2 版または第 3 版としてジオメトリを返します。

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);
```

説明

Geography Markup Language (GML) 要素としてジオメトリを返します。version パラメータは、指定する場合には 2 か 3 になります。version パラメータが無い場合には、デフォルトは 2 です。maxdecimaldigits 引数は出力に使用する小数点以下桁数 (デフォルトは 15 です) の最大値を減らすために使えます。



Warning

任意引数 *maxdecimaldigits* を使用することで、出力ジオメトリが不正になる可能性があります。これを回避するには、前もって [ST_ReducePrecision](#) に適切なグリッドサイズを与えてこれを使用します。

GML2 では 2.1.2 版を参照し、GML3 では 3.1.1 を参照します。

最後の 'options' 引数はビットフィールドです。GML 出力の CRS 出力型を定義するために、また緯度/経度でデータを宣言するために使います。

- 0: GML Short CRS (たとえば EPSG:4326)、デフォルト値
- 1: GML Long CRS (たとえば urn:ogc:def:crs:EPSG:4326)
- 2: GML 3 のみ対応。srsDimension 属性を出力から削除します。
- 4: GML 3 のみ対応。線について <Curve> でなく <LineString> 要素を使います。
- 16: データは緯度/経度 (すなわち SRID=4326) です。デフォルトではデータは平面上にあると仮定します。このオプションは GML 3.1.1 による出力でのみ使われ、軸のオーダに関連します。これを設定すると、座標の順序を入れ替えるので、データベースの経度/緯度の順でなく緯度/経度の順になります。
- 32: ジオメトリのボックス (エンベロープ) を出力します。

'namespace prefix' 引数は、カスタム名前空間のプリフィクスを指定したり、名前空間プリフィクスを指定しない (空にした場合) ために使用します。NULL を指定するか省略した場合には、'gml' プレフィクスを使用します。

Availability: 1.3.2

Availability: 1.5.0 ジオグラフィが導入されました。

Enhanced: 2.0.0 プレフィクスが導入されました。GML 3 用である `options` の 4 は、曲線のかわりにラインストリングを使えるようにするためのものです。GML 3 の多面体サーフェスと TIN が導入されました。`options` の 32 はボックスを出力するために導入されました。

Changed: 2.0.0 デフォルトの名前付き引数を使います。

Enhanced: 2.1.0 GML 3 用に `id` が導入されました。



Note

ST_AsGML の GML 3 版以上では多面体サーフェスと TIN に対応しています。

- ✔ このメソッドは SQL/MM 仕様の実装です。SQL-MM IEC 13249-3: 17.2
- ✔ この関数は 3 次元に対応し、Z 値を削除しません。
- ✔ この関数は多面体サーフェスに対応しています。
- ✔ この関数は三角形と不規則三角網 (TIN) に対応しています。

例: 2 版

```
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>
```

例: 3 版

Flip coordinates and output extended EPSG (16 | 1)

```
SELECT ST_AsGML(3, ST_GeomFromText('POINT(5.234234233242 6.34534534534)',4326), 5, 17);
      st_asgml
      -----
      <gml:Point srsName="urn:ogc:def:crs:EPSG::4326"
><gml:pos srsDimension="2"
>6.34535 5.23423</gml:pos
></gml:Point>
```

-- Output the envelope (32) --

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

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

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

This example uses a PolyhedralSurface.

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

```

    <gml:exterior>
      <gml:LinearRing>
        <gml:posList srsDimension="3"
>0 1 0 0 1 1 1 1 1 1 1 1 0 0 1 0</gml:posList>
      </gml:LinearRing>
    </gml:exterior>
  </gml:PolygonPatch>
  <gml:PolygonPatch>
    <gml:exterior>
      <gml:LinearRing>
        <gml:posList srsDimension="3"
>0 0 1 1 0 1 1 1 1 0 1 1 0 0 1</gml:posList>
      </gml:LinearRing>
    </gml:exterior>
  </gml:PolygonPatch>
</gml:polygonPatches>
</gml:PolyhedralSurface>

```

関連情報

[ST_GeomFromGML](#)

7.9.3.6 ST_AsKML

ST_AsKML — ジオメトリを KML 要素として返します。

Synopsis

```

text ST_AsKML(geometry geom, integer maxdecimaldigits=15, text nprefix=NULL);
text ST_AsKML(geography geog, integer maxdecimaldigits=15, text nprefix=NULL);

```

説明

ジオメトリを Keyhole Markup Language (KML) 要素として返します。デフォルトの最大の小数点以下の桁数は 15 で、デフォルトの名前空間はプリフィクス無しです。



Warning

任意引数 *maxdecimaldigits* を使用することで、出力ジオメトリが不正になる可能性があります。これを回避するには、前もって [ST_ReducePrecision](#) に適切なグリッドサイズを与えてこれを使用します。



Note

PostGIS が Proj 対応でコンパイルされている必要があります。 [PostGIS_Full_Version](#) を使って Proj 対応でコンパイルされているか確認して下さい。



Note

Availability: 1.2.2 - version パラメータが付く形式は 1.3.2 からです。

**Note**

Enhanced: 2.0.0 - プレフィックスの名前空間の追加、デフォルト値と名前付き引数の追加

**Note**

Changed: 3.0.0 - "version" の付いた形式の削除

**Note**

AsKML 出力は SRID を持たないジオメトリでは動作しません。



この関数は 3 次元に対応し、Z 値を削除しません。

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>
```

関連情報

[ST_AsSVG](#), [ST_AsGML](#)

7.9.3.7 ST_AsLatLonText

ST_AsLatLonText — 与えられたポイントの度・分・秒表現を返します。

Synopsis

text **ST_AsLatLonText**(geometry pt, text format=“);

説明

ポイントの度・分・秒表現を返します。



Note

緯度/経度座標系のポイントを前提としています。X(経度) と Y(緯度) 座標系は「正常な」範囲 (経度は-180 から 180、緯度は-90 から 90) に正常化されます。

`text` 引数は結果文字列のための書式を含む書式文字列です。日付書式文字列に近いものです。妥当なトークンは "D" が度、"M" が分、"S" が秒、"C" (cardinal direction) が 4 方位 (NSEW) です。DMS トークンは、求める幅と精度で示すために、繰り返せます ("SSS.SSSS" では "1.0023" になります)。

"M" と "S" と "C" は必須ではありません。"C" が省略された場合には、南または西の場合には "-" 符号がついたうえで、指定した精度で、度が表示されます。"M" も省略された場合には、指定した精度の桁数で十進の度が表示されます。

書式文字列が省略された (または長さが 0 の) 場合には、デフォルトの書式が使われます。

Availability: 2.0

例

デフォルト書式。

```
SELECT (ST_AsLatLonText('POINT (-3.2342342 -2.32498)'));
```

```
2°19'29.928"S 3°14'3.243"W
```

書式を指定 (デフォルトと同じ)。

```
SELECT (ST_AsLatLonText('POINT (-3.2342342 -2.32498)', 'D°M'S.SSS"C'));
```

```
2°19'29.928"S 3°14'3.243"W
```

D, M, S, C 以外の文字は通過するだけです。

```
SELECT (ST_AsLatLonText('POINT (-3.2342342 -2.32498)', 'D degrees, M minutes, S seconds to the C'));
```

```
2 degrees, 19 minutes, 30 seconds to the S 3 degrees, 14 minutes, 3 seconds to the W
```

4 方位文字でなく符号で示された度。

```
SELECT (ST_AsLatLonText('POINT (-3.2342342 -2.32498)', 'D°M'S.SSS'));
```

```
-2°19'29.928" -3°14'3.243"
```

十進の度。

```
SELECT (ST_AsLatLonText('POINT (-3.2342342 -2.32498)', 'D.DDDD degrees C'));
```

```
2.3250 degrees S 3.2342 degrees W
```

過大な値が正常化されます。

```
SELECT (ST_AsLatLonText('POINT (-302.2342342 -792.32498)'));
```

```
72°19'29.928"S 57°45'56.757"E
```

7.9.3.8 ST_AsMARC21

ST_AsMARC21 — ジオメトリを、地理データフィールド (034) を持つ MARC21/XML データとして返します。

Synopsis

```
text ST_AsMARC21 ( geometry geom , text format='hdddmss' );
```

説明

この関数は与えられたジオメトリのバウンディングボックスを表現する **コード化地図数値データ** を持つ MARC21/XML のデータを返します。**format** パラメータを使うと、MARC21/XML が対応する全てのフォーマットにおける **\$d**、**\$e**、**\$f**、**\$g** サブフィールド内の座標を符号化できます。妥当なフォーマットは次の通りです。

- 経緯度識別値、度、分、秒 (デフォルト): **hdddmss**
- 経緯度識別値付き十進の度: **hddd.dddddd**
- 経緯度識別値なし十進の度: **ddd.dddddd**
- 経緯度識別値付き十進の分: **hdddm.mmm**
- 経緯度識別値なし十進の分: **dddm.mmm**
- 経緯度識別値付き十進の秒: **hdddmss.sss**

小数点は **hdddm,mmm** のようにカンマでも構いません。

10 進数形式の精度は小数点以下の数字の数によって制限されます。十進の分で **hddmm.mm** の場合には 2 桁の精度となります。

この関数は Z 値と M 値を無視します。

対応する LOC MARC21/XML のバージョンは次の通りです。

- **MARC21/XML 1.1**

Availability: 3.3.0



Note

この関数は経度/緯度でないジオメトリには対応していません。MARC21/XML 標準 (コード化地図数値データ) で対応していないためです。



Note

MARC21/XML 標準では、コード化地図数値データに空間参照系情報を与えることはできません。これは MARC21/XML への変換で空間参照系情報が失われることを意味します。

例

POINT の hdddmss (デフォルト) を使った MARC21/XML への変換

```
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>
```

POLYGON の、十進の度を使った MARC21/XML への変換

```
SELECT ST_AsMARC21('SRID=4326;POLYGON((-4.5792388916015625 ↔
54.18172660239091,-4.56756591796875 ↔
54.196993557130355,-4.546623229980469 ↔
54.18313300502024,-4.5792388916015625 54.18172660239091))::geometry,' ↔
hddd.dddd');

          st_asmarc21
-----
<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>
```

GEOMETRYCOLLECTION の、十進の分を使った MARC21/XML への変換。MARC21/XML 出力のジオメトリの順序はコレクション内の順序に対応しています。

```
SELECT ST_AsMARC21('SRID=4326;GEOMETRYCOLLECTION(POLYGON((13.1 52.65,13.516666666666667 52.65,13.516666666666667 52.383333333333333,13.1 52.383333333333333,13.1 52.65)),POINT(-4.5 54.25))'::geometry,'hddmm.mmmm');
```

st_asmarc21

```
-----
<record xmlns="http://www.loc.gov/MARC21/slim">
  <datafield tag="034" ind1="1" ind2=" ">
    <subfield code="a">
      <subfield code="d">
        <subfield code="e">
          <subfield code="f">
            <subfield code="g">
              </datafield>
            <datafield tag="034" ind1="1" ind2=" ">
              <subfield code="a">
                <subfield code="d">
                  <subfield code="e">
                    <subfield code="f">
                      <subfield code="g">
                        </datafield>
                      </record>
```

関連情報

[ST_GeomFromMARC21](#)

7.9.3.9 ST_AsMVTGeom

ST_AsMVTGeom — ジオメトリを MVT タイルの座標空間に変換します。

Synopsis

geometry **ST_AsMVTGeom**(geometry geom, box2d bounds, integer extent=4096, integer buffer=256, boolean clip_geom=true);

説明

ジオメトリを MVT ([Mapbox Vector Tile](#)) タイルの座標空間に変換し、必要ならタイルの境界で切り抜きます。ジオメトリは対象地図の座標系でなければなりません (必要なら[ST_Transform](#)を使います)。一般にこれは[Web Mercator](#) (SRID:3857) です。

この関数は、ジオメトリに妥当性を保持しようとし、必要なら修正します。これによって、結果ジオメトリが低い次元に落ちる可能性があります。

The function performs geometry simplification appropriate to the integer tile coordinate space. After transforming to tile coordinates it snaps coordinates to the integer grid, removes repeated points, and removes points that lie on straight lines. It does not perform general-purpose, scale-dependent simplification such as **ST_Simplify**. To reduce detail before tile conversion, simplify the geometry in map coordinates before calling **ST_AsMVTGeom**.

対象地図の座標空間内のタイルを示す四角形の境界を与えなければなりません。ジオメトリが変換され、必要なら切り抜きます。境界は**ST_TileEnvelope**で生成できます。

この関数はジオメトリを**ST_AsMVT**が求める座標空間に変換するために使います。

geom は変換するジオメトリで、対象地図の座標系です。

bounds は地図の座標空間内のタイルの四角形境界で、バッファはありません。

extent は**MVT specification**で定義されるタイル座標空間でのタイルのサイズです。デフォルトは 4096 です。

buffer は、ジオメトリの切り抜きを行うためのタイル座標空間におけるバッファサイズです。デフォルトは 256 です。

clip_geom はジオメトリを切り抜くかそのまま符号化するかを制御する真偽値です。デフォルトは TRUE です。

Availability: 2.4.0



Note

3.0 からは、MVT ポリゴンのクリップと検証にコンフィギュア時に Wagyu を選択することができるようになりました。このライブラリは、デフォルトの GEOS と比べて、速度が速く、より正確な結果が得られますが、小さいポリゴンが削除されることがあります。

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));
```

```
MULTIPOLYGON(((5 4096,10 4091,10 4096,5 4096)),((5 4096,0 4101,0 4096,5 4096)))
```

問い合わせとジオメトリのクリッピングを行うために計算されたタイル境界を使う Web メルカトルのタイルの標準的な例です。data.geom カラムの SRID は 4326 です。

```
SELECT ST_AsMVTGeom(ST_Transform(geom, 3857 ),
    ST_TileEnvelope(12, 513, 412),
    extent =
> 4096,
    buffer =
> 64) AS geom
FROM data
WHERE geom && ST_Transform(ST_TileEnvelope(12, 513, 412, margin =
> (64.0 / 4096)),4326)
```

関連情報

ST_AsMVT, **ST_TileEnvelope**, **PostGIS_Wagyu_Version**

7.9.3.10 ST_AsMVT

ST_AsMVT — 行集合の MVT 表現を返す集約関数です。

Synopsis

```
bytea ST_AsMVT(anyelement set row);  
bytea ST_AsMVT(anyelement row, text name);  
bytea ST_AsMVT(anyelement row, text name, integer extent);  
bytea ST_AsMVT(anyelement row, text name, integer extent, text geom_name);  
bytea ST_AsMVT(anyelement row, text name, integer extent, text geom_name, text feature_id_name);
```

説明

タイルレイヤに対応する行集合を **Mapbox Vector Tile** 表現で返す集約関数です。行には地物ジオメトリとして符号化される **geometry** が必ずあります。ジオメトリはタイルの座標空間に存在しなければならず、**MVT specification** として妥当でなければなりません。 **ST_AsMVTGeom** はジオメトリをタイルの座標空間に変換するのに使えます。他のカラムは地物の属性として符号化されます。

Mapbox Vector Tile 書式は、様々な属性就業を持つ地物を格納することができます。この能力を使うには、JSON オブジェクトを 1 レベルの深さで持っている行データ内の JSONB カラムを提供します。JSONB 値のキーと値は地物の属性として符号化されます。

この関数への複数の呼び出しを `||` や **STRING_AGG** で繋げることで、複数レイヤのタイルを作ることができます。



Important

GEOMETRYCOLLECTION を行の要素として、この関数を呼ばないでください。しかしながら、ジオメトリコレクションを含ませるための準備として、**ST_AsMVTGeom** を使うことができます。

row 少なくとも一つのジオメトリカラムを持つ行データ。

name はレイヤ名です。デフォルトは "default" という文字列です。

extent は、仕様で定義されている画面空間内のタイル範囲です。NULL の場合には、4096 をデフォルト値とします。

geom_name は、行データ内のジオメトリカラム名です。デフォルトはジオメトリカラムのうちの先頭です。PostgreSQL はデフォルトでは自動的に **引用符で括らない識別子は小文字になります**。これは、ジオメトリカラムがカラム名が引用符で括られる ("MyMVTGeom" 等) ようになっていない場合には、このパラメータは小文字で渡さなければなりません。

feature_id_name は行データの地物 ID カラムの名前です。NULL または負数の場合には地物 ID は設定されません。名前が合致し、妥当な型 (smallint, integer, bigint) である最初のカラムが地物 ID に使われます。後続のカラムは全てプロパティとして追加されます。JSON プロパティには対応していません。

Enhanced: 3.0 - 地物 ID への対応を追加。

Enhanced: 2.5.0 - パラレルクエリへの対応の追加。

Availability: 2.4.0

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;

```

関連情報

[ST_AsMVTGeom](#), [ST_TileEnvelope](#)

7.9.3.11 ST_AsSVG

ST_AsSVG — ジオメトリから SVG パスデータを返します。

Synopsis

text **ST_AsSVG**(geometry geom, integer rel=0, integer maxdecimaldigits=15);
text **ST_AsSVG**(geography geog, integer rel=0, integer maxdecimaldigits=15);

説明

Scalar Vector Graphics (SVG) としてジオメトリを返します。第 2 引数に 1 を指定すると、相対移動によるパスデータ実装を返し、絶対移動の場合はデフォルト (または 0) とします。第 3 引数は、出力の十進数の最大桁数を減らすために使います (デフォルトは 15 です)。ポイントジオメトリは、'rel' が 0 のときはポイントは cx/cy に、'rel' が 1 のときは x/y に、それぞれ出力されます。マルチポイントはコンマ (",") で区切られ、ジオメトリコレクションはセミコロン (";") で区切られます。

PostGIS SVG グラフィックを使うには、ST_AsSVG からの出力を操作するための PL/pgSQL 関数を提供する [pg_svg](#) ライブラリをチェックアウトして下さい。

Enhanced: 3.4.0 全ての曲線タイプに対応しました

Changed: 2.0.0 - デフォルト引数と名前付き引数に対応しました。



Note

Availability: 1.2.2. Availability: 1.4.0 PostGIS 1.4.0 で<http://www.w3.org/TR/SVG/paths.html#PathDataBNF>に従うため、絶対パスに L コマンドが入りました。



このメソッドは曲線ストリングと曲線に対応しています。

例

```
SELECT ST_AsSVG('POLYGON((0 0,0 1,1 1,1 0,0 0))'::geometry);
```

```
M 0 0 L 0 -1 1 -1 1 0 Z
```

曲線ストリング

```
SELECT ST_AsSVG(ST_GeomFromText('CIRCULARSTRING(-2 0,0 2,2 0,0 2,2 4)'));;
```

```
M -2 0 A 2 2 0 0 1 2 0 A 2 2 0 0 1 2 -4
```

マルチ曲線

```
SELECT ST_AsSVG('MULTICURVE((5 5,3 5,3 3,0 3),  
CIRCULARSTRING(0 0,2 1,2 2))'::geometry, 0, 0);
```

```
M 5 -5 L 3 -5 3 -3 0 -3 M 0 0 A 2 2 0 0 0 2 -2
```

マルチサーフェス

```
SELECT ST_AsSVG('MULTISURFACE(  
CURVEPOLYGON(CIRCULARSTRING(-2 0,-1 -1,0 0,1 -1,2 0,0 2,-2 0),  
(-1 0,0 0.5,1 0,0 1,-1 0)),  
((7 8,10 10,6 14,4 11,7 8)))'::geometry, 0, 2);
```

```
M -2 0 A 1 1 0 0 0 0 0 A 1 1 0 0 0 2 0 A 2 2 0 0 0 -2 0 Z  
M -1 0 L 0 -0.5 1 0 0 -1 -1 0 Z  
M 7 -8 L 10 -10 6 -14 4 -11 Z
```

7.9.3.12 ST_AsTWKB

ST_AsTWKB — TWKB (Tiny Well-Known Binary) としてジオメトリを出力します。

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);

説明

TWKB (Tiny Well-Known Binary) 書式としてジオメトリを返します。TWKB は、出力サイズを最小化することに焦点を当てた **compressed binary format** (圧縮バイナリ書式) です。

decimaldigits パラメータによって、出力に持たせる精度を制御できます。デフォルトでは、符号化前に値は単位上の最近値に丸められます。精度が高いものを転送したいなら、これらの数字を大きくします。たとえば、1 の値は小数点の右に 1 桁の数字が保存されます。

include sizes と **include bounding boxes** パラメータによって、符号化オブジェクトの長さに関する任意情報の有無やオブジェクトの境界の有無を制御できます。デフォルトでは、無しです。クライアントソフトウェアが、これらの情報を使用しないなら、このパラメータを有効にしないで下さい。空白が増える (しかも空白の抑制が TWKB のポイントである) ためです。

配列入力の形式は、ジオメトリのコレクションを変換して、一意の識別子を TWKB コレクションに持たせるためのものです。これは、コレクションを展開して、オブジェクト内部に関する情報にさらにアクセスするのに使えます。 **array_agg** 関数を使用して配列を生成できます。他のパラメータは、単純な形式のものと同じです。

**Note**

書式仕様は<https://github.com/TWKB/Specification>にあります。JavaScript クライアントを構築するプログラムは<https://github.com/TWKB/twkb.js>にあります。

Enhanced: 2.4.0 メモリと速度の改善。

Availability: 2.2.0

例

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

```
\x02000202020808
```

識別子を含む TWKB オブジェクトの集計を生成するには、まず、`"array_agg()"` を使って求めるジオメトリとオブジェクトを集計して、その後に適切な TWKB 関数を呼んでいます。

```
SELECT ST_AsTWKB(array_agg(geom), array_agg(gid)) FROM mytable;
```

```
\x040402020400000202
```

関連情報

[ST_GeomFromTWKB](#), [ST_AsBinary](#), [ST_AsEWKB](#), [ST_AsEWKT](#), [ST_GeomFromText](#)

7.9.3.13 ST_AsX3D

`ST_AsX3D` — ジオメトリを X3D ノード要素書式 (ISO-IEC-19776-1.2-X3DEncodings-XML) で返します。

Synopsis

text **ST_AsX3D**(geometry g1, integer maxdecimaldigits=15, integer options=0);

説明

X3D XML で表されたノード要素 <http://www.web3d.org/standards/number/19776-1> としてのジオメトリを返します。maxdecimaldigits (精度) を指定しない場合には、デフォルトは 15 です。

Note

PostGIS ジオメトリを X3D に変換するための任意引数が様々あります。X3D ジオメトリ型は、PostGIS ジオメトリタイプに対応付けされていないためです。また、より良い対応付けになると思われるものの、ほとんどのレンダリングツールが今のところに対応していないため、開発者が避けてきた新しい X3D タイプに対応付けをしていないためでもあります。これらは開発者が決定した対応付けです。開発者が皆さんにより好まれる対応付けを示せるようなアイデアや方法に関する考えを持っているなら、お気軽にバグチケットを出して下さい。

次に現時点の PostGIS 2 次元/3 次元型から X3D 型への対応付けを示します。

'options' 引数はビットフィールドです。X3D GeoCoordinates Geospatial ノードを表現するかどうか、また、X/Y 軸を反対にするかどうかで使います。`ST_AsX3D` は、デフォルトではデータベースの形式 (経度, 緯度または X, Y) で出力しますが、X3D の lat/lon, y/x のデフォルトが好まれることがあります。

- 0: データベース内の X/Y 順 (経度/緯度 =X, Y が標準です) とします。デフォルト値です。非空間座標 (一般的な古い Coordinate 要素です) です。
- 1: X と Y を反対にします。GeoCoordinate 任意スイッチと併せて使用されると、出力は"latitude_first" (緯度が先) となり、座標が同じように反対になります。
- 2: GeoSpatial GeoCoordinates 内への座標出力。WGS 84 経度緯度 (SRID: 4326) でない場合にエラーが投げられます。現在は GeoCoordinate 型のみ対応します。[X3D specs specifying a spatial reference system](#). を参照して下さい。デフォルト出力は GeoCoordinate geoSystem='''GD'' ''WE'' ''longitude_first''' となります。X3D のデフォルトである GeoCoordinate geoSystem='''GD'' ''WE'' ''latitude_first''' とするには、(2 + 1) = 3 とします。

PostGIS タイプ	2 次元 X3D タイプ	3 次元 X3D タイプ
LINESTRING	未実装 - PolyLine2D の予定	LineSet
MULTILINESTRING	未実装 - PolyLine2D の予定	IndexedLineSet
MULTIPOINT	Polypoint2D	PointSet
POINT	空白区切り座標値を出力	空白区切り座標値を出力
(MULTI) POLYGON, POLYHEDRALSURFACE	不正な X3D マークアップ	IndexedFaceSet (内環は現在は他の faceset として出力)
TIN	TriangleSet2D (未実装)	IndexedTriangleSet

**Note**

2 次元ジオメトリ対応はまだ不完全です。内環は現在は分けられたポリゴンとして描画されるだけです。作業中です。

3 次元空間については、特に[X3D Integration with HTML5](#)で大きな進歩があります。

また、描画されたジオメトリを閲覧するための素晴らしいオープンソースの X3D ビューアがあります。Free Wrl <http://freewrl.sourceforge.net/>のバイナリが Mac, Linux, Windows 用であります。ジオメトリを見るためのパッケージである FreeWRL_Launcher を使います。

この関数と[x3dDom html/js open source toolkit](#)を用いた[PostGIS minimalist X3D viewer](#)を開いてみて下さい。

Availability: 2.0.0: ISO-IEC-19776-1.2-X3DEncodings-XML

Enhanced: 2.2.0: GeoCoordinates と軸 (x/y, 経度/緯度) の反転に対応しました。詳細は options を見て下さい。

- ✓ この関数は 3 次元に対応し、Z 値を削除しません。
- ✓ この関数は多面体サーフェスに対応しています。
- ✓ この関数は三角形と不規則三角網 (TIN) に対応しています。

例: 完全に機能する **X3D** 文書の作成 - **FreeWrl** や他の **X3D** ビューアで見ることができる立方体を生成しています。

```
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>
> ' ||
    ST_AsX3D(ST_GeomFromEWKT('POLYHEDRALSURFACE( ((0 0 0, 0 0 1, 0 1 1, 0 1 0, 0 0 0)),
((0 0 0, 0 1 0, 1 1 0, 1 0 0, 0 0 0)), ((0 0 0, 1 0 0, 1 0 1, 0 0 1, 0 0 0)),
((1 1 0, 1 1 1, 1 0 1, 1 0 0, 1 1 0)),
((0 1 0, 0 1 1, 1 1 1, 1 1 0, 0 1 0)), ((0 0 1, 1 0 1, 1 1 1, 0 1 1, 0 0 1)) )' )) ||
    </Shape>
</Transform>
</Scene>
</X3D>
>' As x3ddoc;

```

```

x3ddoc
-----
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE X3D PUBLIC "ISO//Web3D//DTD X3D 3.0//EN" "http://www.web3d.org/specifications/x3d ←
-3.0.dtd">
<X3D>
  <Scene>
    <Transform>
      <Shape>
        <Appearance>
          <Material emissiveColor='0 0 1' />
        </Appearance>
        <IndexedFaceSet coordIndex='0 1 2 3 -1 4 5 6 7 -1 8 9 10 11 -1 12 13 14 15 -1 16 17 ←
18 19 -1 20 21 22 23'>
          <Coordinate point='0 0 0 0 0 1 0 1 1 0 1 0 0 0 0 0 1 0 1 1 0 1 0 0 0 0 1 0 0 ←
1 0 1 0 0 1 1 1 0 1 1 1 1 0 1 1 0 0 0 1 0 0 1 1 1 1 1 1 1 0 0 0 1 1 0 1 1 1 ←
1 0 1 1' />
        </IndexedFaceSet>
      </Shape>
    </Transform>
  </Scene>
</X3D>

```

PostGIS の建物

このクエリの出力を **x3d scene viewer** にコピーして貼り付けて、**Show** をクリックします。

```

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 のデータ細分化と押し出しによって形成された建物

例: 高さ **3** 単位で精度が **6** 桁の八角柱

```
SELECT ST_AsX3D(ST_Translate(ST_Force3D(ST_Buffer(ST_Point(10,10),5, 'quad_segs=2')), 0,0,
3),
6) As x3dfrag;
```

x3dfrag

```
-----
<IndexedFaceSet convex='false' coordIndex='0 1 2 3 4 5 6 7'
><Coordinate point='15 10 3 13.535534 6.464466 3 10 5 3 6.464466 6.464466 3 5 10 3 6.464466 ←
13.535534 3 10 15 3 13.535534 13.535534 3 ' /></IndexedFaceSet>
```

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 0 1 0 1 0 0 0 0 0 1 0 1 1 0' /></IndexedTriangleSet>
```

例: 閉じたラインストリング (穴のあるポリゴンの境界)

```
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 — ジオメトリの GeoHash 表現を返します。

Synopsis

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

説明

ジオメトリの **ジオハッシュ** 表現を計算します。ジオハッシュは地理座標系のポイントを、前置に基づいた文字列として符号化したもので、ソート可能かつ検索可能です。短いジオハッシュではポイントは低精度に表現されています。これは、ポイントを含むボックスと考えることができます。

0 でない範囲を持つ、ポイントでないジオメトリ値はジオハッシュコードに対応付けできます。コードの精度はジオメトリの地理範囲に依存します。

maxchars が指定されていない場合には、返されるジオハッシュコードは、入力ジオメトリを含む最も小さいセルです。ポイントでは、20 文字の精度 (入力の完全な倍精度浮動小数点数の保持十分です) ジオハッシュを返します。他のジオメトリタイプでは、より低い精度のジオハッシュを返しますが、精度はジオメトリの範囲に依存します。大きいジオメトリは低い精度となり、小さいジオメトリは高い精度になります。ジオハッシュコードで決まるボックスは常に入力入力地物を含んでいます。

maxchars が指定された場合には、この値は、返されるジオハッシュコードの最大文字数になります。入力ジオメトリは (おそらく) より低い精度で表現されます。ポイントでない場合には、計算の最初のポイントはジオメトリのバウンダリボックスの中心となります。

Availability: 1.4.0



Note

ST_GeoHash の入力ジオメトリは、地理座標系 (経度/緯度) でなければなりません。



このメソッドは曲線ストリングと曲線に対応しています。

例

```
SELECT ST_GeoHash(ST_Point(-126,48));
```

```
c0w3hf1s70w3hf1s70w3
```

```
SELECT ST_GeoHash(ST_Point(-126,48), 5);
```

```
c0w3h
```

This example shows that the GeoHash for a line containing a point is a prefix of the point code.

```
SELECT ST_GeoHash('LINESTRING(-126 48, -126.1 48.1)::geometry');
```

```
c0w3
```

関連情報

[ST_GeomFromGeoHash](#), [ST_PointFromGeoHash](#), [ST_Box2dFromGeoHash](#)

7.10 演算子

7.10.1 バウンディングボックス演算子

7.10.1.1 &&

&& — A の 2 次元バウンディングボックスが B の 2 次元バウンディングボックスとインタセクトする場合に TRUE を返します。

Synopsis

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

説明

&& 演算子は、A の 2 次元バウンディングボックスが B の 2 次元バウンディングボックスとインタセクトする場合に TRUE を返します。



Note

このオペランドは、ジオメトリで使用できるインデックスを使用します。

Enhanced: 2.0.0 多面体サーフェス対応が導入されました。

Availability: 1.5.0 ジオグラフィ対応が導入されました。



このメソッドは曲線ストリングと曲線に対応しています。



この関数は多面体サーフェスに対応しています。

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)

関連情報

[ST_Intersects](#), [ST_Extent](#), [|&>](#), [&>](#), [&<|](#), [&<](#), [~](#), [@](#)

7.10.1.2 &&(geometry,box2df)

&&(geometry,box2df) — ジオメトリの (キャッシュされている)2 次元バウンディングボックスが単精度浮動小数点数による 2 次元バウンディングボックスとインタセクトする場合に **TRUE** を返します。

Synopsis

boolean **&&**(geometry A , box2df B);

説明

&& 演算子は、ジオメトリ A のキャッシュされた 2 次元バウンディングボックスが、2 次元バウンディングボックス B とインタセクトする場合に、**TRUE** を返します。単精度浮動小数点数によるバウンディングボックスを使います。B が倍精度浮動小数点数を使う **box2d** である場合には、単精度浮動小数点数によるバウンディングボックス (**BOX2DF**) に変換されることになります。



Note

この被演算子は、ユーザが使うためというよりも内部で BRIN インデックスに使うために導入されました。

Availability: 2.3.0 BRIN (Block Range INDEXes) が導入されました。



このメソッドは曲線ストリングと曲線に対応しています。



この関数は多面体サーフェスに対応しています。

例

```
SELECT ST_Point(1,1) && ST_MakeBox2D(ST_Point(0,0), ST_Point(2,2)) AS overlaps;
```

overlaps
t

(1 row)

関連情報

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

7.10.1.3 &&(box2df,geometry)

&&(box2df,geometry) — 単精度浮動小数点数による 2 次元バウンディングボックスがジオメトリの (キャッシュされている)2 次元バウンディングボックスとインタセクトする場合に **TRUE** を返します。

Synopsis

```
boolean &&( box2df A , geometry B );
```

説明

&& 演算子は、2 次元バウンディングボックス A が、ジオメトリ B のキャッシュされた 2 次元バウンディングボックスとインタセクトする場合に、**TRUE** を返します。単精度浮動小数点数によるバウンディングボックスを使います。A が倍精度浮動小数点数を使う **box2d** である場合には、単精度浮動小数点数によるバウンディングボックス (**BOX2DF**) に変換されることになります。



Note

この被演算子は、ユーザが使うためというよりも内部で BRIN インデックスに使うために導入されました。

Availability: 2.3.0 BRIN (Block Range INdexes) が導入されました。



このメソッドは曲線ストリングと曲線に対応しています。



この関数は多面体サーフェスに対応しています。

例

```
SELECT ST_MakeBox2D(ST_Point(0,0), ST_Point(2,2)) && ST_Point(1,1) AS overlaps;

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

関連情報

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

7.10.1.4 &&(box2df,box2df)

&&(box2df,box2df) — 二つの単精度浮動小数点数による 2 次元バウンディングボックス (**BOX2DF**) が相互にインタセクトする場合に **TRUE** を返します。

Synopsis

```
boolean &&( box2df A , box2df B );
```

説明

&& 演算子は、2 次元バウンディングボックス A と B が相互にインタセクトする場合に、TRUE を返します。単精度浮動小数点数によるバウンディングボックスを使います。A または B が倍精度浮動小数点数を使う **box2d** である場合には、バウンディングボックス (**BOX2DF**) に変換されることになります。

**Note**

この演算子は、ユーザが使うためというよりも内部で BRIN インデックスに使うために導入されました。

Availability: 2.3.0 BRIN (Block Range INdexes) が導入されました。



このメソッドは曲線ストリングと曲線に対応しています。



この関数は多面体サーフェスに対応しています。

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)
```

関連情報

&&(geometry,box2df), &&(box2df,geometry), ~(geometry,box2df), ~(box2df,geometry), ~(box2df,box2df), @ (geometry,box2df), @ (box2df,geometry), @ (box2df,box2df)

7.10.1.5 &&&

&&& — A の n 次元バウンディングボックスが B の n 次元バウンディングボックスとインタセクトする場合に TRUE を返します。

Synopsis

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

説明

&&& 演算子は、A の n 次元バウンディングボックスが B の n 次元バウンディングボックスとインタセクトする場合に TRUE を返します。

**Note**

このオペランドは、ジオメトリで使用できるインデックスを使用します。

Availability: 2.0.0

- ✔ このメソッドは曲線ストリングと曲線に対応しています。
- ✔ この関数は多面体サーフェスに対応しています。
- ✔ この関数は三角形と不規則三角網 (TIN) に対応しています。
- ✔ この関数は 3 次元に対応し、Z 値を削除しません。

例: **3** 次元ラインストリング

```
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

例: **XYM** ラインストリング

```
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

関連情報

&&

7.10.1.6 &&&(geometry,gidx)

&&&(geometry,gidx) — ジオメトリの (キャッシュされている)**n** 次元バウンディングボックスが単精度浮動小数点数による **n** 次元バウンディングボックス (GIDX) とインタセクトする場合に **TRUE** を返します。

Synopsis

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

説明

&&& 演算子は、ジオメトリ **A** のキャッシュされた **n** 次元バウンディングボックスが、**n** 次元バウンディングボックス **B** とインタセクトする場合に、**TRUE** を返します。単精度浮動小数点数によるバウンディングボックスを使います。B が倍精度浮動小数点数を使う **box3d** である場合には、単精度浮動小数点数による 3 次元バウンディングボックス (**GIDX**) に変換されることになります。

**Note**

この演算子は、ユーザが使うためというよりも内部で **BRIN** インデックスに使うために導入されました。

Availability: 2.3.0 **BRIN** (**B**lock **R**ange **I**ndexes) が導入されました。

- ✓ このメソッドは曲線ストリングと曲線に対応しています。
- ✓ この関数は多面体サーフェスに対応しています。
- ✓ この関数は三角形と不規則三角網 (**TIN**) に対応しています。
- ✓ この関数は 3 次元に対応し、**Z** 値を削除しません。

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)
```

関連情報

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

7.10.1.7 &&&(gidx,geometry)

&&&(gidx,geometry) — 単精度浮動小数点数による **n** 次元バウンディングボックス (**GIDX**) がジオメトリの (キャッシュされている)**n** 次元バウンディングボックスとインタセクトする場合に **TRUE** を返します。

Synopsis

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

説明

&&& 演算子は、**n** 次元バウンディングボックス **A** が、ジオメトリ **B** のキャッシュされた **n** 次元バウンディングボックスとインタセクトする場合に、**TRUE** を返します。単精度浮動小数点数によるバウンディングボックスを使います。A が倍精度浮動小数点数を使う **box3d** である場合には、単精度浮動小数点数による 3 次元バウンディングボックス (**GIDX**) に変換されることになります。

**Note**

この演算子は、ユーザが使うためというよりも内部で BRIN インデックスに使うために導入されました。

Availability: 2.3.0 BRIN (Block Range INdexes) が導入されました。

- ✓ このメソッドは曲線ストリングと曲線に対応しています。
- ✓ この関数は多面体サーフェスに対応しています。
- ✓ この関数は三角形と不規則三角網 (TIN) に対応しています。
- ✓ この関数は 3 次元に対応し、Z 値を削除しません。

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)
```

関連情報

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

7.10.1.8 &&&(gidx,gidx)

&&&(gidx,gidx) — 二つの単精度浮動小数点数による n 次元バウンディングボックス (GIDX) が相互にインタセクトする場合に TRUE を返します。

Synopsis

boolean **&&&**(gidx A , gidx B);

説明

&&& 演算子は、n 次元バウンディングボックス A と B が相互にインタセクトする場合に、TRUE を返します。単精度浮動小数点数によるバウンディングボックスを使います。A または B が倍精度浮動小数点数を使う **box3d** である場合には、単精度浮動小数点数による 3 次元バウンディングボックス (GIDX) に変換されることになります。

**Note**

この演算子は、ユーザが使うためというよりも内部で BRIN インデックスに使うために導入されました。

Availability: 2.3.0 BRIN (Block Range INdexes) が導入されました。

- ✔ このメソッドは曲線ストリングと曲線に対応しています。
- ✔ この関数は多面体サーフェスに対応しています。
- ✔ この関数は三角形と不規則三角網 (TIN) に対応しています。
- ✔ この関数は 3 次元に対応し、Z 値を削除しません。

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)
```

関連情報

[&&&\(geometry,gidx\), &&&\(gidx,geometry\)](#)

7.10.1.9 &<

&< — A のバウンディングボックスが B のバウンディングボックスをオーバラップするか、B のバウンディングボックスの左にある場合に **TRUE** を返します。

Synopsis

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

説明

&< 演算子は、A のバウンディングボックスが B のバウンディングボックスをオーバラップするか、B のバウンディングボックスの左にある場合に **TRUE** を返します。条件についてより詳細に言うと、B のバウンディングボックスをオーバラップするか B のバウンディングボックスの右に * ない * 場合です。



Note

これらのオペランドは、ジオメトリで使用できるインデックスを使用します。

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)
```

関連情報

[&&](#), [|&>](#), [&>](#), [&<|](#)

7.10.1.10 &<|

&<| — A のバウンディングボックスが B のバウンディングボックスをオーバラップするか、B のバウンディングボックスの下にある場合に **TRUE** を返します。

Synopsis

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

説明

&<| 演算子は、A のバウンディングボックスが B のバウンディングボックスをオーバラップするか、B のバウンディングボックスの下にある場合に **TRUE** を返します。条件についてより詳細に言うと、B のバウンディングボックスをオーバラップするか B のバウンディングボックスの上に * ない * 場合です。



このメソッドは曲線ストリングと曲線に対応しています。



この関数は多面体サーフェスに対応しています。



Note

これらのオペランドは、ジオメトリで使用できるインデックスを使用します。

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)

関連情報

[&&](#), [|&>](#), [&>](#), [&<](#)

7.10.1.11 &>

&> — A のバウンディングボックスが B のバウンディングボックスをオーバラップするか、B のバウンディングボックスの右にある場合に **TRUE** を返します。

Synopsis

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

説明

&> 演算子は、A のバウンディングボックスが B のバウンディングボックスをオーバラップするか、B のバウンディングボックスの右にある場合に **TRUE** を返します。条件についてより詳細に言うと、B のバウンディングボックスをオーバラップするか B のバウンディングボックスの左に * ない * 場合です。



Note

このオペランドは、ジオメトリで利用できるインデックスを使用します。

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)

関連情報

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

7.10.1.12 <<

<< — A のバウンダリボックスが、厳密に B のバウンダリボックスの左にある場合に TRUE を返します。

Synopsis

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

説明

<< 演算子は A のバウンダリボックスが、厳密に B のバウンダリボックスの左にある場合に TRUE を返します。

**Note**

このオペランドは、ジオメトリで使用できるインデックスを使用します。

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)

関連情報

>>, |>>, <<|

7.10.1.13 <<|

<<| — A のバウンダリボックスが、厳密に B のバウンダリボックスの下にある場合に TRUE を返します。

Synopsis

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

説明

`<<|` 演算子は、A のバウンダリボックスが、厳密に B のバウンダリボックスの下にある場合に **TRUE** を返します。

**Note**

このオペランドは、ジオメトリで利用できるインデックスを使用します。

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)

関連情報

`<<`, `>>`, `|>>`

7.10.1.14 =

`=` — ジオメトリ/ジオグラフィ A の座標と座標の並び順がジオメトリ/ジオグラフィ B と同じ場合に **TRUE** を返します。

Synopsis

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

説明

`=` 演算子は、ジオメトリ/ジオグラフィ A の座標と座標の並び順がジオメトリ/ジオグラフィ B と同じ場合に **TRUE** を返します。PostgreSQL は、ジオメトリの内部の順位付けと比較 (`GROUP BY` や `ORDER BY` 節などで使います) のために、ジオメトリ用に定義されている `=`, `<` および `>` 演算子を使います。

**Note**

この演算子は、同じ座標値と並び順である、全ての点で確実に同じジオメトリ/ジオグラフィだけを同じと考えます。「空間的に等価」、つまり、並び順は無視し、異なる表現でも同じ空間領域を占めるかどうかをテストするには、**`ST_OrderingEquals`**または**`ST_Equals`**を使います。

**Caution**

この演算子は、ジオメトリで有効なインデックスを一切 * 使いません *。インデックスを使った確実な等価性試験を行うには、`=` と `&&` を併用します。

Changed: 2.4.0, 以前の版では、ジオメトリ自体の等価性でなくバウンディングボックスが等価かどうかを見ていました。バウンディングボックスが等価かどうかを知る必要がある場合には、代わりに `~=` を使います。



このメソッドは曲線ストリングと曲線に対応しています。



この関数は多面体サーフェスに対応しています。

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
```

関連情報

[ST_Equals](#), [ST_OrderingEquals](#), [~=](#)

7.10.1.15 >>

>> — A のバウンダリボックスが、厳密に B のバウンダリボックスの右にある場合に TRUE を返します。

Synopsis

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

説明

>> 演算子は、A のバウンダリボックスが、厳密に B のバウンダリボックスの右にある場合に TRUE を返します。



Note

このオペランドは、ジオメトリで使用できるインデックスを使用します。

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)

関連情報

<<, |>>, <<|

7.10.1.16 @

@ — A のバウンダリボックスが B のバウンダリボックスに含まれている場合に TRUE を返します。

Synopsis

```
boolean @( geometry A , geometry B );
```

説明

@ 演算子は、A のバウンダリボックスが B のバウンダリボックスに、完全に含まれている場合に TRUE を返します。



Note

このオペランドは、ジオメトリで使用できるインデックスを使用します。

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)
```

関連情報

~, &&

7.10.1.17 @(geometry,box2df)

@(geometry,box2df) — ジオメトリの 2 次元バウンディングボックスが単精度浮動小数点数による 2 次元バウンディングボックス (BOX2DF) に包含される場合に TRUE を返します。

Synopsis

```
boolean @( geometry A , box2df B );
```

説明

@ 演算子は、ジオメトリ A の 2 次元バウンディングボックスが、2 次元バウンディングボックス B に包含される場合に、TRUE を返します。単精度浮動小数点数によるバウンディングボックスを使います。B が倍精度浮動小数点数を使う box2d である場合には、単精度浮動小数点数によるバウンディングボックス (BOX2DF) に変換されることになります。



Note

この被演算子は、ユーザが使うためというよりも内部で BRIN インデックスに使うために導入されました。

Availability: 2.3.0 BRIN (Block Range INdexes) が導入されました。



このメソッドは曲線ストリングと曲線に対応しています。



この関数は多面体サーフェスに対応しています。

例

```
SELECT ST_Buffer(ST_GeomFromText('POINT(2 2)'), 1) @ ST_MakeBox2D(ST_Point(0,0), ST_Point(5,5)) AS is_contained;
```

t

関連情報

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

7.10.1.18 @(box2df,geometry)

[@\(box2df,geometry\)](#) — 単精度浮動小数点数による 2 次元バウンディングボックス (BOX2DF) がジオメトリの 2 次元バウンディングボックスに包含される場合に TRUE を返します。

Synopsis

boolean @(box2df A , geometry B);

説明

@ 演算子は、2 次元バウンディングボックス A が、ジオメトリ B の 2 次元バウンディングボックスに包含される場合に、TRUE を返します。単精度浮動小数点数によるバウンディングボックスを使います。A が倍精度浮動小数点数を使う `box2d` である場合には、単精度浮動小数点数によるバウンディングボックス (BOX2DF) に変換されることとなります。



Note

この被演算子は、ユーザが使うためというよりも内部で BRIN インデックスに使うために導入されました。

Availability: 2.3.0 BRIN (Block Range INdexes) が導入されました。



このメソッドは曲線ストリングと曲線に対応しています。



この関数は多面体サーフェスに対応しています。

例

```
SELECT ST_MakeBox2D(ST_Point(2,2), ST_Point(3,3)) @ ST_Buffer(ST_GeomFromText('POINT(1 1)'), 10) AS is_contained;
```

t

関連情報

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

7.10.1.19 @(box2df,box2df)

@(box2df,box2df) — 二つの単精度浮動小数点数による n 次元バウンディングボックス (GIDX) の一方がもう一方を包含する場合に TRUE を返します。

Synopsis

```
boolean @( box2df A , box2df B );
```

説明

@ 演算子は、2 次元バウンディングボックス A が 2 次元バウンディングボックス B に包含される場合には、TRUE を返します。単精度浮動小数点数によるバウンディングボックスを使います。A または B が倍精度浮動小数点数を使う box2d である場合には、単精度浮動小数点数によるバウンディングボックス (BOX2DF) に変換されることになります。



Note

この被演算子は、ユーザが使うためというよりも内部で BRIN インデックスに使うために導入されました。

Availability: 2.3.0 BRIN (Block Range INdexes) が導入されました。



このメソッドは曲線ストリングと曲線に対応しています。



この関数は多面体サーフェスに対応しています。

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;
```

t

関連情報

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

7.10.1.20 |&>

|&> — A のバウンディングボックスが B のバウンディングボックスをオーバーラップするか、B のバウンディングボックスの上にある場合に TRUE を返します。

Synopsis

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

説明

|&> 演算子は、A のバウンディングボックスが B のバウンディングボックスをオーバラップするか、B のバウンディングボックスの上にある場合に **TRUE** を返します。条件についてより詳細に言うと、B のバウンディングボックスをオーバラップするか B のバウンディングボックスの下に * ない * 場合です。

**Note**

このオペランドは、ジオメトリで使えるインデックスを使用します。

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)

関連情報

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

7.10.1.21 |>>

|>> — A のバウンダリボックスが、厳密に B のバウンダリボックスの上にある場合に **TRUE** を返します。

Synopsis

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

説明

|>> 演算子は、A のバウンダリボックスが、厳密に B のバウンダリボックスの上にある場合に **TRUE** を返します。

**Note**

このオペランドは、ジオメトリで使えるインデックスを使用します。

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)

関連情報

<<, >>, <<|

7.10.1.22 ~

~ — A のバウンディングボックスが B のバウンディングボックスを含む場合に TRUE を返します。

Synopsis

boolean ~(geometry A , geometry B);

説明

~ 演算子は、A のバウンディングボックスが B のバウンディングボックスを、完全に含む場合に TRUE を返します。



Note

これらのオペランドは、ジオメトリで使用するインデックスを使用します。

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)
```

関連情報

[@, &&](#)

7.10.1.23 ~(geometry,box2df)

~(geometry,box2df) — ジオメトリの (キャッシュされている)2 次元バウンディングボックスが単精度浮動小数点数による n 次元バウンディングボックス (GIDX) を包含する場合に TRUE を返します。

Synopsis

```
boolean ~( geometry A , box2df B );
```

説明

~ 演算子は、ジオメトリ A の 2 次元バウンディングボックスが、2 次元バウンディングボックス B を包含する場合に、TRUE を返します。単精度浮動小数点数によるバウンディングボックスを使います。B が倍精度浮動小数点数を使う box2d である場合には、単精度浮動小数点数によるバウンディングボックス (BOX2DF) に変換されることになります。



Note

この被演算子は、ユーザが使うためというよりも内部で BRIN インデックスに使うために導入されました。

Availability: 2.3.0 BRIN (Block Range INdexes) が導入されました。

✓ このメソッドは曲線ストリングと曲線に対応しています。

✓ この関数は多面体サーフェスに対応しています。

例

```
SELECT ST_Buffer(ST_GeomFromText('POINT(1 1)'), 10) ~ ST_MakeBox2D(ST_Point(0,0), ST_Point(↵
(2,2)) AS contains;
```

```
t
```

関連情報

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

7.10.1.24 ~(box2df,geometry)

~(box2df,geometry) — 単精度浮動小数点数による 2 次元バウンディングボックス (BOX2DF) をジオメトリの (キャッシュされている)2 次元バウンディングボックスが包含する場合に TRUE を返します。

Synopsis

```
boolean ~( box2df A , geometry B );
```

説明

~ 演算子は、2 次元バウンディングボックス A が、ジオメトリ B の 2 次元バウンディングボックスを包含する場合に、TRUE を返します。単精度浮動小数点数によるバウンディングボックスを使います。A が倍精度浮動小数点数を使う box2d である場合には、単精度浮動小数点数によるバウンディングボックス (BOX2DF) に変換されることになります。



Note

この被演算子は、ユーザが使うためというよりも内部で BRIN インデックスに使うために導入されました。

Availability: 2.3.0 BRIN (Block Range INdexes) が導入されました。



このメソッドは曲線ストリングと曲線に対応しています。



この関数は多面体サーフェスに対応しています。

例

```
SELECT ST_MakeBox2D(ST_Point(0,0), ST_Point(5,5)) ~ ST_Buffer(ST_GeomFromText('POINT(2 2)') ←  
    , 1) AS contains;
```

```
t
```

関連情報

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

7.10.1.25 ~(box2df,box2df)

~(box2df,box2df) — 二つの単精度浮動小数点数による 2 次元バウンディングボックス (BOX2DF) の一方がもう一方を包含する場合に TRUE を返します。

Synopsis

```
boolean ~( box2df A , box2df B );
```

説明

～ 演算子は、2次元バウンディングボックス A が、2次元バウンディングボックス B を包含する場合に、TRUE を返します。単精度浮動小数点数によるバウンディングボックスを使います。A または B が倍精度浮動小数点数を使う box2d である場合には、単精度浮動小数点数によるバウンディングボックス (BOX2DF) に変換されることになります。

**Note**

この被演算子は、ユーザが使うためというよりも内部で BRIN インデックスに使うために導入されました。

Availability: 2.3.0 BRIN (Block Range INDEXes) が導入されました。



このメソッドは曲線ストリングと曲線に対応しています。



この関数は多面体サーフェスに対応しています。

例

```
SELECT ST_MakeBox2D(ST_Point(0,0), ST_Point(5,5)) ~ ST_MakeBox2D(ST_Point(2,2), ST_Point(3,3)) AS contains;
```

t

関連情報

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

7.10.1.26 ~ =

~ = — A のバウンディングボックスが B のバウンディングボックスと同じ場合に TRUE を返します。

Synopsis

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

説明

~= 演算子はジオメトリ/ジオグラフィ A のバウンディングボックスがジオメトリ/ジオグラフィ B のバウンディングボックスと同じ場合に TRUE を返します。

**Note**

このオペランドは、ジオメトリで利用できるインデックスを使用します。

Availability: 1.5.0 挙動が変更されました



この関数は多面体サーフェスに対応しています。

**Warning**

この演算子の挙動は PostGIS 1.5 で、実際のジオメトリとしての等価性のテストから、バウンディングボックスの等価性のテストに変更されました。ハードアップグレードまたはソフトアップグレードを実行している場合は、データベースがどの挙動を持つかに動作が依存して、ややこしくなります。データベースがどの挙動を持つか判断するために、下のクエリを実行することができます。本当の等価性をチェックするには **ST_OrderingEquals** または **ST_Equals** を使用します。

例

```
SELECT 'LINESTRING(0 0, 1 1)::geometry ~= 'LINESTRING(0 1, 1 0)::geometry AS equality;
```

t

関連情報

ST_Equals, **ST_OrderingEquals**, =

7.10.2 距離演算子

7.10.2.1 <->

<-> — A と B の 2 次元距離を返します。

Synopsis

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

説明

<-> 演算子は二つのジオメトリの 2 次元距離を返します。“ORDER BY” 句内で使われ、インデックスの援助を受けた近傍結果集合の結果を返します。ジオメトリではデカルト距離を使い、ジオグラフィでは球面上の距離を計算します。

**Note**

このオペランドはジオメトリで利用できるインデックスを使用します。他の演算子との相違点は、ORDER BY 句でのみインデックスが使用される点です。

**Note**

ジオメトリのひとつが定数となる (副問い合わせ/共通テーブル式にない) 場合 (a.geom でなく 'SRID=3005;POINT(1011102 450541)::geometry' 等となる場合) には、インデックスが有効になるだけです。

詳細な例については[OpenGeo workshop: Nearest-Neighbour Searching](#)を参照して下さい。

Enhanced: 2.2.0 -- 真の KNN (k 近傍法) のジオメトリーとジオグラフィーのふるまい。ジオグラフィーの KNN は回転楕円体でなく球面に基づきます。

Changed: 2.2.0 -- 真の KNN が有効になると、古いハイブリッド法による回避策は遅くなることがあります。下の例を見てください。

Availability: 2.0.0 -- 弱い KNN によって、実際の距離の代わりにジオメトリーの重心に基づく近傍が得られます。ポイントは確実な結果を得て、他のタイプは全て不確実な結果を得ます。

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 の生の答は次のとおりです。

```
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)

二つのクエリで“EXPLAIN ANALYZE”を実行すると、二つ目で速度が改善したことが分かります。

プランで直接確実な距離を生成できない時には、真の近傍を見つけるには複合クエリーを追加居ます。まずインデックスを使った KNN による CTE (Common Table Expression, WITH 句) クエリーを実行し、その後正しい順序を得るための確実なクエリーを実施します。次のようにします:

```
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)

関連情報

[ST_DWithin](#), [ST_Distance](#), [<#>](#)

7.10.2.2 `|=|`

`|=|` — A トラジェクトリと B トラジェクトリとの最接近する時の距離を返します。

Synopsis

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

説明

`|=|` 演算子は二つのトラジェクトリー ([ST_IsValidTrajectory](#)参照) の間の 3 次元距離を返します。[ST_DistanceCPA](#)と同じですが、この演算子は N 次元インデックスを使った最近傍探索を行います。



Note
この演算子は、ジオメトリで使用可能な ND GiST (n 次元 GiST) インデックスを使用します。他の空間インデックスを使う演算子と違い、ORDER BY 句でのみ空間インデックスを使います。



Note
ジオメトリのひとつが定数となる (副問い合わせ/共通テーブル式にない) 場合 (a.geom でなく 'SRID=3005;LINESTRINGM(0 0 0,0 0 1) '::geometry 等となる場合) には、インデックスが有効になるだけです。

Availability: 2.2.0.

例

This example orders stored trajectories by their distance from a literal query trajectory.

```
\set qt 'ST_AddMeasure(ST_MakeLine(ST_MakePointM(-350, 300, 0), ST_MakePointM(-410, 490, 0) ↵
), 10, 20)'
```

```
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)

関連情報

[ST_DistanceCPA](#), [ST_ClosestPointOfApproach](#), [ST_IsValidTrajectory](#)

7.10.2.3 <#>

<#> — A のバウンディングボックスと B のバウンディングボックスの 2 次元距離を返します。

Synopsis

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

説明

<#> 演算子は、二つの浮動小数点数型ボックスの間の距離を返します。場合によっては空間インデックスから読みます。最近傍の概算距離での並び替えに使えます。



Note

このオペランドはジオメトリで利用できるインデックスを使用します。他の演算子との相違点は、ORDER BY 句でのみインデックスが使用される点です。



Note

ジオメトリのひとつが `g1.geom <#>` と違って `ORDER BY (ST_GeomFromText('POINT(1 2)') <#> geom)` というように定数である場合には、インデックスが有効になるだけです。

Availability: 2.0.0.

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)

関連情報

[ST_DWithin](#), [ST_Distance](#), [<->](#)

7.10.2.4 <<->>

<<->> — A と B の間またはそれらのバウンディングボックスの間の n 次元距離を返します

Synopsis

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

説明

<<->> 演算子は、二つのジオメトリのバウンディングボックスの重心の n 次元 (ユークリッド) 距離を返します。近傍の概ねの距離による並び替えに使用します。



Note

このオペランドはジオメトリで利用できるインデックスを使用します。他の演算子との相違点は、ORDER BY 句でのみインデックスが使用される点です。

**Note**

ジオメトリのひとつが定数となる (副問い合わせ/共通テーブル式にない) 場合 (a.geom でなく 'SRID=3005;POINT(1011102 450541)::geometry' 等となる場合) には、インデックスが有効になるだけです。

Availability: 2.2.0.

関連情報

<->

7.11 空間関係関数

7.11.1 トポロジ関係関数

7.11.1.1 ST_3DIntersects

ST_3DIntersects — 二つのジオメトリが 3 次元空間において空間的にインタセクトするかどうかをテストします。ポイント、ラインストリング、ポリゴン、多面体サーフェス (面) についてのみ動作します。

Synopsis

```
boolean ST_3DIntersects( geometry geomA , geometry geomB );
```

説明

オーバラップ、接触、包含は全て、ジオメトリがインタセクトしていることを意味しています。これらが TRUE を返す場合は、空間的にインタセクトしています。非接続は、空間インタセクトについて FALSE となります。

**Note**

この関数を実行すると、対象ジオメトリにおいて使用できるインデックスを使用したバウンディングボックスによる比較が自動的に行われます。

**Note**

浮動小数点数のロバスト性の障害のため、ジオメトリ処理後に期待するようなインタセクトが常にあるわけではないこととなります。たとえば、ジオメトリに対するラインストリング上の最接近点がラインストリング上に無いことがあります。インタセクトと考えたいのにセンチメートル程度の距離があると、この問題は発生しますが、この種の問題への対応には、**ST_3DDWithin**を使います。

Changed: 3.0.0 SFCGAL バックエンドが削除され、GEOS バックエンドでは TIN に対応しました。

Availability: 2.0.0



この関数は 3 次元に対応し、Z 値を削除しません。



この関数は多面体サーフェスに対応しています。



この関数は三角形と不規則三角網 (TIN) に対応しています。



このメソッドは SQL/MM 仕様の実装です。SQL-MM IEC 13249-3: 5.1

ジオメトリの例

```
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 の例

```
SELECT ST_3DIntersects('TIN(((0 0 0,1 0 0,0 1 0,0 0 0)))'::geometry, 'POINT(.1 .1 0)'::geometry);
```

```
t
```

関連情報

[ST_3DDWithin](#), [ST_Intersects](#)

7.11.1.2 ST_Contains

ST_Contains — B の全てのポイントが A 内にあり、かつ、双方の内部に共有点が存在するかどうかをテストします。

Synopsis

boolean **ST_Contains**(geometry geomA, geometry geomB);

説明

ジオメトリ A がジオメトリ B を含む場合には TRUE を返します。A が B を含むというのは、B のポイントの全てが A の内側 (つまり A の内部または境界) にあり (A の外部に B のポイントが全くないとも言えます)、かつ B の内部の少なくとも一つのポイントが A の内部にある場合に限ります。

数学用語では: $ST\_Contains(A, B) \Leftrightarrow (A \sqsupset B = B) \wedge (Int(A) \sqsupset Int(B) \neq \square)$

包含関係は反射関係です。あらゆるジオメトリは自己を包含します (対照的に **ST_ContainsProperly** 述語内ではジオメトリは確実な包含にはなりません)。また、包含関係は非対称です。ST_Contains(A,B) = true かつ ST_Contains(B,A) = true の場合には、二つのジオメトリはトポロジ的に等価です (ST_Equals(A,B) = true)。

ST_Contains は **ST_Within** の反対です。ST_Contains(A,B) = ST_Within(B,A) となります。



Note

内部が必ず共通の点を持たなければならないので、定義の微妙な点は、ポリゴンとラインが境界であるラインとポイントを含まないことです。詳細については、[Subtleties of OGC Covers, Contains, Within](#) をご覧下さい。ST_Covers 述語は、より包括的な関係を提供します。

**Note**

この関数を実行すると、対象ジオメトリにおいて使用できるインデックスを使用したバウンディングボックスによる比較が自動的行われます。インデックスの使用を避けるには `_ST_Contains` 関数を使います。

GEOS モジュールで実現しています。

Enhanced: 2.3.0 PIP short-circuit (ポリゴンとポイントに限定した高速判定) を少ないポイントからなるマルチポイントに対応することができるよう拡張しました。以前の版ではポリゴンとポイントの組み合わせにだけ対応していました。

**Important**

Enhanced: 3.0.0 GEOMETRYCOLLECTION への対応が可能となりました

**Important**

この関数を不正なジオメトリで呼ばないでください。予期しない結果が返されます。

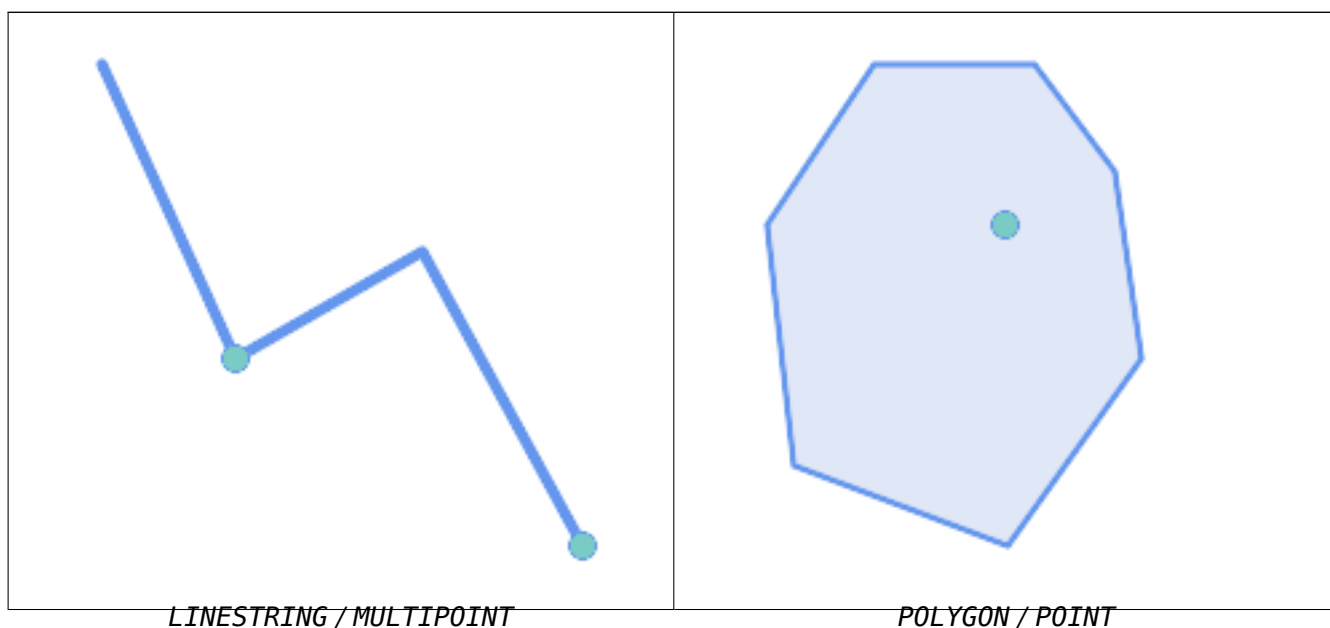
ご注意: これは論理値を返して整数を返さないのが「許される」版です。

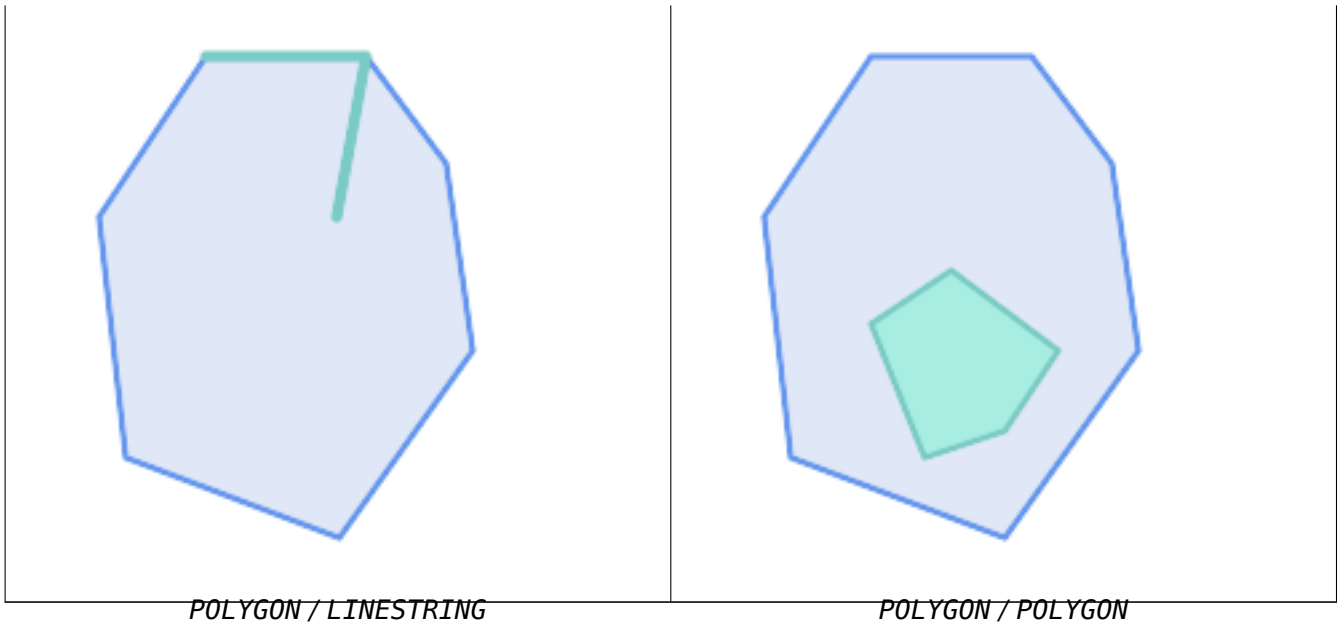
✓ このメソッドは **OGC Simple Features Implementation Specification for SQL 1.1** の実装です。s2.1.1.2 // s2.1.13.3 - `within(geometry B, geometry A)` と同じ

✓ このメソッドは SQL/MM 仕様の実装です。SQL-MM 3: 5.1.31

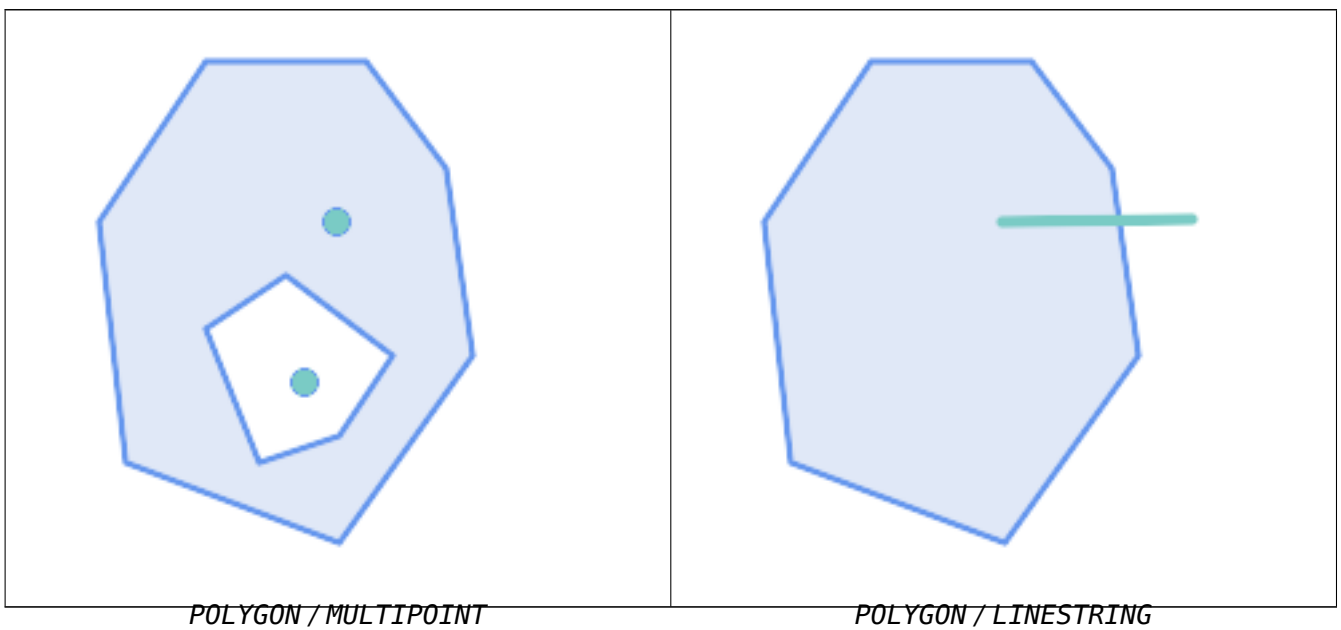
例

次に示す図全てで、`ST_Contains` は `TRUE` を返します。





次に示す状況では `ST_Contains` は `FALSE` を返します。



次に示す状況にあつては、内部のインタセクトの状態のために (`ST_Covers` だと `TRUE` を返すのに) `ST_Contains` は `FALSE` を返します。



This example tests a circle within another circle.

```
SELECT ST_Contains(smallc, bigc) As smallcontainsbig,
       ST_Contains(bigc, smallc) As bigcontainssmall,
       ST_Contains(bigc, ST_Union(smallc, bigc)) as bigcontainsunion,
       ST_Equals(bigc, ST_Union(smallc, bigc)) as bigisunion,
       ST_Covers(bigc, ST_ExteriorRing(bigc)) As bigcoversexterior,
       ST_Contains(bigc, ST_ExteriorRing(bigc)) As bigcontainsexterior
FROM (SELECT ST_Buffer(ST_GeomFromText('POINT(1 2)'), 10) As smallc,
       ST_Buffer(ST_GeomFromText('POINT(1 2)'), 20) As bigc) As foo;
```

smallcontainsbig	bigcontainssmall	bigcontainsunion	bigisunion	bigcoversexterior	bigcontainsexterior	
-----+	-----+	-----+	-----+	-----+	-----+	↔
f	t	t	t	t	f	

This example demonstrates the difference between contains and contains-properly predicates.

```
SELECT ST_GeometryType(geomA) As geomtype, ST_Contains(geomA, geomA) AS acontainsa, ↔
       ST_ContainsProperly(geomA, geomA) AS acontainspropa,
       ST_Contains(geomA, ST_Boundary(geomA)) As acontainsba, ST_ContainsProperly(geomA, ↔
       ST_Boundary(geomA)) As acontainspropba
FROM (VALUES ( ST_Buffer(ST_Point(1,1), 5,1) ),
            ( ST_MakeLine(ST_Point(1,1), ST_Point(-1,-1) ) ),
            ( ST_Point(1,1) )
      ) As foo(geomA);
```

geomtype	acontainsa	acontainspropa	acontainsba	acontainspropba
-----+	-----+	-----+	-----+	-----+
ST_Polygon	t	f	f	f
ST_LineString	t	f	f	f
ST_Point	t	t	f	f

関連情報

[ST_Boundary](#), [ST_ContainsProperly](#), [ST_Covers](#), [ST_CoveredBy](#), [ST_Equals](#), [ST_Within](#)

7.11.1.3 ST_ContainsProperly

ST_ContainsProperly — B の全てのポイントが A の内部にあるかをテストします。

Synopsis

```
boolean ST_ContainsProperly(geometry geomA, geometry geomB);
```

説明

B の全てのポイントが A の内部にある (もしくは B のポイントで A の外部にあるポイントが無い) 場合に TRUE を返します。

数学用語では: $ST\_ContainsProperly(A, B) \Leftrightarrow Int(A) \cap B = B$

二つのジオメトリの DE-9IM 交差行列が [T**FF*FF*] に合致する場合には、A が B を完全に含みます。

A は自身を正しく含むことはありませんが、自身を含みます。

大きいポリゴンジオメトリでのジオメトリの集合のインタセクションを計算する際にこの述語を使います。インタセクションはかなり遅いので、ContainsProperly を使って、対象ジオメトリのうち全体が領域内にあるものを抜き出すことができ、効率的になります。これらの場面では、インタセクションは確実に元の対象ジオメトリであることが直感的に分かります。



Note

この関数を実行すると、対象ジオメトリにおいて使用できるインデックスを使用したバウンディングボックスによる比較が自動的に行われます。インデックスの使用を避けるには `_ST_ContainsProperly` 関数を使います。



Note

この関数は、個々のポイントでトポロジを計算する必要が無く、より効率的に計算できる点で、`ST_Contains` と `ST_Intersects` より優れています。

GEOS モジュールで実現しています。

Availability: 1.4.0



Important

Enhanced: 3.0.0 GEOMETRYCOLLECTION への対応が可能となりました



Important

この関数を不正なジオメトリで呼ばないでください。予期しない結果が返されます。

例

This example tests a circle within another circle.

```
SELECT ST_ContainsProperly(smallc, bigc) As smallcontainspropbig,
       ST_ContainsProperly(bigc,smallc) As bigcontainspropsmall,
       ST_ContainsProperly(bigc, ST_Union(smallc, bigc)) as bigcontainspropunion,
       ST_Equals(bigc, ST_Union(smallc, bigc)) as bigisunion,
       ST_Covers(bigc, ST_ExteriorRing(bigc)) As bigcoversexterior,
       ST_ContainsProperly(bigc, ST_ExteriorRing(bigc)) As bigcontainsexterior
FROM (SELECT ST_Buffer(ST_GeomFromText('POINT(1 2)'), 10) As smallc,
       ST_Buffer(ST_GeomFromText('POINT(1 2)'), 20) As bigc) As foo;
```

smallcontainspropbig	bigcontainspropsmall	bigcontainspropunion	bigisunion	bigcoversexterior	bigcontainsexterior
f	t	f	t	t	f

This example demonstrates the difference between contains and contains-properly predicates.

```
SELECT ST_GeometryType(geomA) As geomtype, ST_Contains(geomA,geomA) AS acontainsa,
       ST_ContainsProperly(geomA, geomA) AS acontainspropa,
       ST_Contains(geomA, ST_Boundary(geomA)) As acontainsba, ST_ContainsProperly(geomA,
       ST_Boundary(geomA)) As acontainspropba
FROM (VALUES ( ST_Buffer(ST_Point(1,1), 5,1) ),
            ( ST_MakeLine(ST_Point(1,1), ST_Point(-1,-1) ) ),
            ( ST_Point(1,1) )
      ) As foo(geomA);
```

geomtype	acontainsa	acontainspropa	acontainsba	acontainspropba
ST_Polygon	t	f	f	f
ST_LineString	t	f	f	f
ST_Point	t	t	f	f

関連情報

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

7.11.1.4 ST_CoveredBy

ST_CoveredBy — A の全てのポイントが B 内にあるかをテストします。

Synopsis

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

説明

ジオメトリ/ジオグラフィ A の全ての点がジオメトリ/ジオグラフィ B の内側にある (つまり B の境界か内部とインタセクトする) 場合に TRUE を返します。A に B の外側にある (B の外部にある) 点が無いかどうかをテストする、と言い換えられます。

数学用語では: $ST\_CoveredBy(A, B) \Leftrightarrow A \sqsubset B = A$

`ST_CoveredBy` は `ST_Covers` の反対です。 $ST\_CoveredBy(A, B) = ST\_Covers(B, A)$ となります。

一般に、この関数は `ST_Within` の代わりに使います。「境界がジオメトリの内側に無い」という奇妙な言葉が定義に含まれないためです。



Note

この関数を実行すると、対象ジオメトリにおいて使用できるインデックスを使用したバウンディングボックスによる比較が自動的に行われます。インデックスの使用を避けるには `_ST_CoveredBy` 関数を使います。



Important

Enhanced: 3.0.0 GEOMETRYCOLLECTION への対応が可能となりました



Important

この関数を不正なジオメトリで呼ばないでください。予期しない結果が返されます。

GEOS モジュールで実現しています。

Availability: 1.2.2

ご注意: これは論理値を返して整数を返さないのが「許される」版です。

これは OGC 標準と違いますが Oracle は持っています。

例

This example tests a circle covered by another circle.

```
SELECT ST_CoveredBy(smallc,smallc) As smallinsmall,
       ST_CoveredBy(smallc, bigc) As smallcoveredbybig,
       ST_CoveredBy(ST_ExteriorRing(bigc), bigc) As exteriorcoveredbybig,
       ST_Within(ST_ExteriorRing(bigc),bigc) As exeriorwithinbig
FROM (SELECT ST_Buffer(ST_GeomFromText('POINT(1 2)'), 10) As smallc,
       ST_Buffer(ST_GeomFromText('POINT(1 2)'), 20) As bigc) As foo;
```

smallinsmall	smallcoveredbybig	exteriorcoveredbybig	exeriorwithinbig
t	t	t	f

(1 row)

関連情報

[ST_Contains](#), [ST_Covers](#), [ST_ExteriorRing](#), [ST_Within](#)

7.11.1.5 ST_Covers

`ST_Covers` — B の全ての点が A 内にあるかをテストします。

Synopsis

```
boolean ST_Covers(geometry geomA, geometry geomB);  
boolean ST_Covers(geography geogpolyA, geography geogpointB);
```

説明

ジオメトリ/ジオグラフィ B の全てのポイントがジオメトリ/ジオグラフィ A の内側にある (内部または境界とインタセクトする) 場合に TRUE を返します。A の外部に B の点が無いことと等価です。

数学用語では: $ST\_Covers(A, B) \Leftrightarrow A \sqcup B = B$

ST_Covers は **ST_CoveredBy** の逆です。ST_Covers(A,B) = ST_CoveredBy(B,A) となります。

一般に、この関数は **ST_Contains** の代わりに使われるべきものです。定義が「ジオメトリがその境界を含まない」という奇妙なものになっていないからです。



Note

この関数を実行すると、対象ジオメトリにおいて使用できるインデックスを使用したバウンディングボックスによる比較が自動的行われます。インデックスの使用を避けるには `_ST_Covers` 関数を使います。



Important

Enhanced: 3.0.0 GEOMETRYCOLLECTION への対応が可能となりました



Important

この関数を不正なジオメトリで呼ばないでください。予期しない結果が返されます。

GEOS モジュールで実現しています。

Enhanced: 2.4.0 ジオグラフィ型を使う形式においてポリゴンの中のポリゴンとポリゴンの中のラインストリングへの対応を追加

Enhanced: 2.3.0 ジオメトリについて、PIP short-circuit (ポリゴンとポイントに限定した高速判定) を少ないポイントからなるマルチポイントに対応することができるよう拡張しました。以前の版ではポリゴンとポイントの組み合わせにだけ対応していました。

Availability: 1.5 - ジオグラフィ対応が導入されました。

Availability: 1.2.2

ご注意: これは論理値を返して整数を返さないのが「許される」版です。

これは OGC 標準と違いますが Oracle は持っています。

例

ジオメトリの例

This example tests a circle covering another circle.

```
SELECT ST_Covers(smallc,smallc) As smallinsmall,
       ST_Covers(smallc, bigc) As smallcoversbig,
       ST_Covers(bigc, ST_ExteriorRing(bigc)) As bigcoversexterior,
       ST_Contains(bigc, ST_ExteriorRing(bigc)) As bigcontainsexterior
FROM (SELECT ST_Buffer(ST_GeomFromText('POINT(1 2)'), 10) As smallc,
       ST_Buffer(ST_GeomFromText('POINT(1 2)'), 20) As bigc) As foo;
```

```
smallinsmall | smallcoversbig | bigcoversexterior | bigcontainsexterior
-----+-----+-----+-----
t           | f           | t           | f
(1 row)
```

ジオグラフィの例

This example compares a point with a 300-meter buffer against a point and against a 10-meter buffer around that point.

```
SELECT ST_Covers(geog_poly, geog_pt) As poly_covers_pt,
       ST_Covers(ST_Buffer(geog_pt,10), geog_pt) As buff_10m_covers_cent
FROM (SELECT ST_Buffer(ST_GeogFromText('SRID=4326;POINT(-99.327 31.4821)'), 300) As
       geog_poly,
       ST_GeogFromText('SRID=4326;POINT(-99.33 31.483)') As geog_pt ) As foo;
```

```
poly_covers_pt | buff_10m_covers_cent
-----+-----
f           | t
```

関連情報

[ST_Contains](#), [ST_CoveredBy](#), [ST_Within](#)

7.11.1.6 ST_Crosses

ST_Crosses — 二つのジオメトリが内部に共有ポイントを持ち、かつそれだけにならないようになっているかテストします。

Synopsis

boolean **ST_Crosses**(geometry g1, geometry g2);

説明

二つのジオメトリを比較して、「空間的にクロスしている」場合に **TRUE** を返します。ジオメトリは内部の共通点をいくつか持っていますが、全て持っているわけではありません。ジオメトリの内部の共通部分は空ではなく、二つのジオメトリの最大図形次元より小さくなければなりません。二つのジオメトリの共通部分は、入力ジオメトリのいずれとも同じであってはなりません。クロスでないなら **FALSE** を返します。クロスの関係は対称ですが無反射です。

数学用語では: $ST\_Crosses(A, B) \Leftrightarrow (dim(Int(A) \cap Int(B)) < \max(dim(Int(A)), dim(Int(B))) \wedge (A \not\sqsubset B \wedge A \not\sqsupset B)$

DE-9IM 交差行列が次の通り合致するとジオメトリはクロスしています:

- Point/Line, Point/Area, Line/Area の場合には T*T*****

- Line/Point, Area/Point, Area/Line の場合には **T*****T****
- Line/Line の場合には **0*******
- 点/点の状況や面/面の状況では、結果は **FALSE** です

**Note**

OpenGIS Simple Feature Specification では、この述語は Point/Line, Point/Area, Line/Line, Line/Area の場合についてのみ定義されています。JTS/GEOS では、Line/Point, Area/Point, Area/Line について拡張しています。これによって関係が対称になっています。

**Note**

この関数を実行すると、対象ジオメトリにおいて使用できるインデックスを使用したバウンディングボックスによる比較が自動的に行われます。

**Important**

Enhanced: 3.0.0 GEOMETRYCOLLECTION への対応が可能となりました



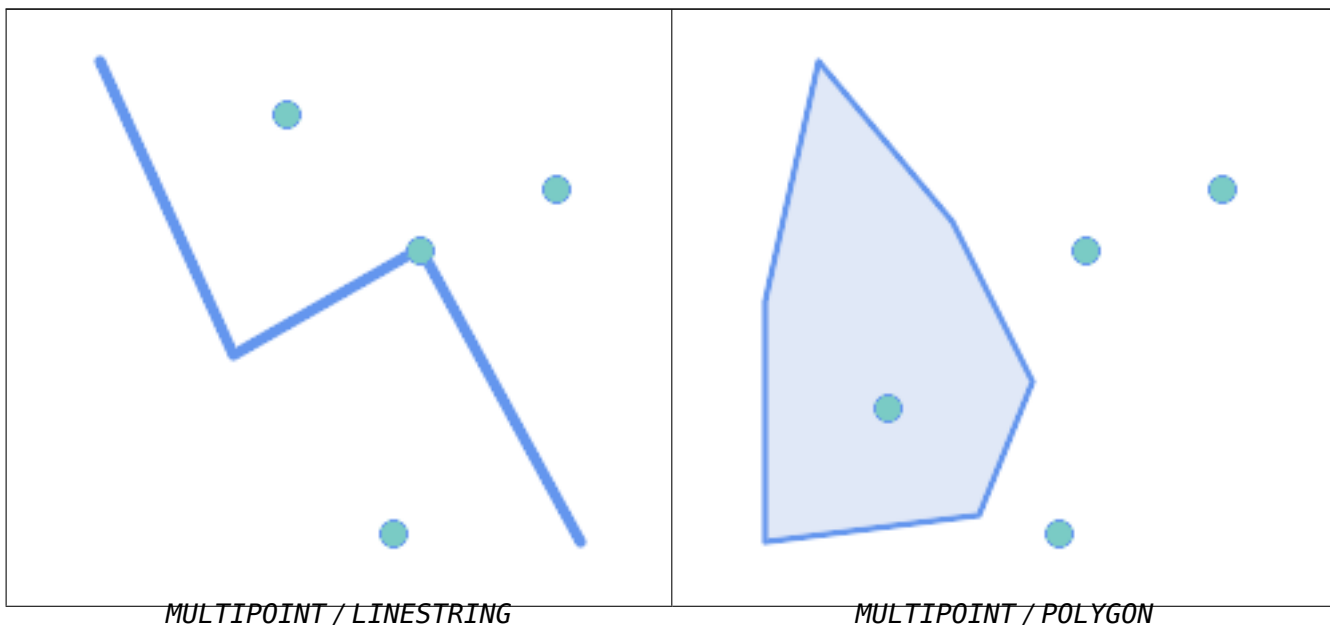
このメソッドは **OGC Simple Features Implementation Specification for SQL 1.1** の実装です。s2.1.13.3

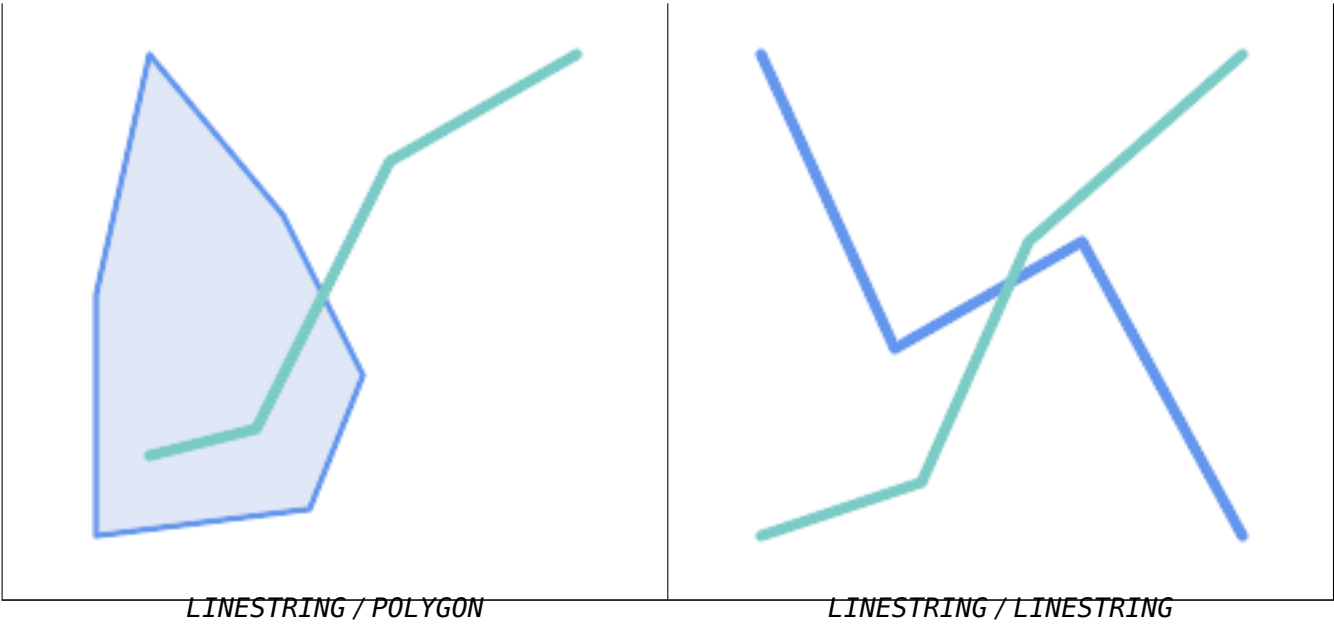


このメソッドは SQL/MM 仕様の実装です。SQL-MM 3: 5.1.29

例

次の場合には全て **TRUE** を返します。





roads (道路) と highways (高速道路) の 2 つのテーブルを持っている場面を考えます。

<pre>CREATE TABLE roads (id serial NOT NULL, geom geometry, CONSTRAINT roads_pkey PRIMARY KEY (↵ road_id));</pre>	<pre>CREATE TABLE highways (id serial NOT NULL, the_gem geometry, CONSTRAINT roads_pkey PRIMARY KEY (↵ road_id));</pre>

次のようなクエリを使って、highway とクロスする roads のリストを決定します。

```
SELECT roads.id  
FROM roads, highways  
WHERE ST_Crosses(roads.geom, highways.geom);
```

関連情報

[ST_Contains](#), [ST_Overlaps](#)

7.11.1.7 ST_Disjoint

ST_Disjoint — 二つのジオメトリが共有点を持たないようになっているかテストします。

Synopsis

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

説明

二つのジオメトリが接続されていない場合に TRUE を返します。ジオメトリに共有点がない場合を指します。二つのジオメトリが他の空間関係が TRUE である場合には接続されていない状況にはありません。接続されていない場合に [ST_Intersects](#) は FALSE を返します。
数学用語では: $ST\_Disjoint(A, B) \Leftrightarrow A \cap B = \emptyset$

**Important**

Enhanced: 3.0.0 GEOMETRYCOLLECTION への対応が可能となりました

GEOS モジュールで実現しています。

**Note**

この関数呼び出しはインデックスを使いません。**ST_Intersects**の否定には、インデックスを使用する、より効率の良い選択肢を使うことができます: `ST_Disjoint(A,B) = NOT ST_Intersects(A,B)`

**Note**

ご注意: これは論理値を返して整数を返さないのが「許される」版です。



このメソッドは**OGC Simple Features Implementation Specification for SQL 1.1**の実装です。s2.1.1.2 //s2.1.13.3 - a.Relate(b, 'FF*FF****')



このメソッドは SQL/MM 仕様の実装です。SQL-MM 3: 5.1.26

例

```
SELECT ST_Disjoint('POINT(0 0)::geometry, 'LINESTRING ( 2 0, 0 2 ) '::geometry);
```

t

```
SELECT ST_Disjoint('POINT(0 0)::geometry, 'LINESTRING ( 0 0, 0 2 ) '::geometry);
```

f

関連情報

ST_Intersects

7.11.1.8 ST_Equals

ST_Equals — 二つのジオメトリが同じ点集合になっているかテストします。

Synopsis

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

説明

ジオメトリが「トポロジ的に等価である」場合に **TRUE** を返します。'=' より良い答を得るのに使います。トポロジ的に等価であるとは、ジオメトリが同じ図形次元で、点集合が同じ空間を占めていることを指します。頂点の順位が同じでなくても構いません。ポイントの順序の確認には一貫して**ST_OrderingEquals**を使います(**ST_OrderingEquals** は、単純なポイントの順序が同じであるかを確認するより多少厳しいです)。

数学用語では: $ST\_Equals(A, B) \Leftrightarrow A = B$

次の関係が成り立ちます: $ST\_Equals(A, B) \Leftrightarrow ST\_Within(A,B) \wedge ST\_Within(B,A)$

**Important**

Enhanced: 3.0.0 GEOMETRYCOLLECTION への対応が可能となりました



このメソッドは**OGC Simple Features Implementation Specification for SQL 1.1**の実装です。s2.1.1.2



このメソッドは SQL/MM 仕様の実装です。SQL-MM 3: 5.1.24

Changed: 2.2.0 この関数は、どちらのジオメトリも不正であっても、バイナリで同じ場合なら **TRUE** を返します。

例

```
SELECT ST_Equals(ST_GeomFromText('LINESTRING(0 0, 10 10)'),
  ST_GeomFromText('LINESTRING(0 0, 5 5, 10 10)'));
```

t

```
SELECT ST_Equals(ST_Reverse(ST_GeomFromText('LINESTRING(0 0, 10 10)'),
  ST_GeomFromText('LINESTRING(0 0, 5 5, 10 10)')));
```

t

関連情報

ST_IsValid, **ST_OrderingEquals**, **ST_Reverse**, **ST_Within**

7.11.1.9 ST_Intersects

ST_Intersects — 二つのジオメトリがインタセクトしている (少なくとも一つの共有点がある) かどうかテストします。

Synopsis

```
boolean ST_Intersects( geometry geomA , geometry geomB );
boolean ST_Intersects( geography geogA , geography geogB );
```

説明

二つのジオメトリがインタセクトする場合に **TRUE** を返します。任意の共有点を持つ場合を指します。

ジオグラフィに対しては、0.00001 メートルの距離許容値が使われます (このため非常に近いポイントはインタセクトしているとみなされます)。

数学用語では: $ST\_Intersects(A, B) \Leftrightarrow A \cap B \neq \emptyset$

DE-9IM 交差行列が次の通り合致するとジオメトリはインタセクトしています:

- T*****
- *T*****
- ***T*****
- ****T*****

空間的なインタセクションは全ての他の空間関係テスト関数に含まれます。例外は **ST_Disjoint** で、全てのジオメトリがインタセクトしていないかどうかをテストすることになります。

**Note**

この関数を実行すると、対象ジオメトリにおいて使用できるインデックスを使用したバウンディングボックスによる比較が自動的に行われます。

Changed: 3.0.0 SFCGAL 版を削除し、2 次元 TIN のネイティブ対応を追加しました。

Enhanced: 2.5.0 ジオメトリコレクションに対応しました。

Enhanced: 2.3.0 PIP short-circuit (ポリゴンとポイントに限定した高速判定) を少ないポイントからなるマルチポイントに対応することができるよう拡張しました。以前の版ではポリゴンとポイントの組み合わせにだけ対応していました。

ジオメトリについては、GEOS モジュールで実現しています。ジオグラフィについてはネイティブです。

Availability: 1.5 ジオグラフィ対応が導入されました。

**Note**

ジオグラフィでは、この関数は 0.00001 メートルの距離許容を持ち、回転楕円体計算でなく球面を使います。

**Note**

ご注意: これは論理値を返して整数を返さないのが「許される」版です。

✔ このメソッドは **OGC Simple Features Implementation Specification for SQL 1.1** の実装です。s2.1.1.2 //s2.1.13.3 - $ST\_Intersects(g1, g2) \rightarrow \text{Not}(ST\_Disjoint(g1, g2))$

✔ このメソッドは SQL/MM 仕様の実装です。SQL-MM 3: 5.1.27

✔ このメソッドは曲線ストリングと曲線に対応しています。

✔ この関数は三角形と不規則三角網 (TIN) に対応しています。

ジオメトリの例

```
SELECT ST_Intersects('POINT(0 0)::geometry, 'LINESTRING ( 2 0, 0 2 ) '::geometry);
```

f

```
SELECT ST_Intersects('POINT(0 0)::geometry, 'LINESTRING ( 0 0, 0 2 ) '::geometry);
```

t

This example looks up rows in a table; for faster lookup, the geometry column should have a GiST index.

```
SELECT id, name FROM cities WHERE ST_Intersects(geom, 'SRID=4326;POLYGON((28 53,27.707 52.293,27 52,26.293 52.293,26 53,26.293 53.707,27 54,27.707 53.707,28 53))'::geometry);
```

```
id | name
----+-----
  2 | Minsk
(1 row)
```

ジオグラフィの例

```
SELECT ST_Intersects('SRID=4326;LINESTRING(-43.23456 72.4567,-43.23456 72.4568)::geography',
'SRID=4326;POINT(-43.23456 72.4567772)::geography');

```

t

関連情報

[&&](#), [ST_3DIntersects](#), [ST_Disjoint](#)

7.11.1.10 ST_LineCrossingDirection

ST_LineCrossingDirection — 二つのラインストリングがどのように交差しているかを示す数字を返します。

Synopsis

integer **ST_LineCrossingDirection**(geometry linestringA, geometry linestringB);

説明

二つのラインストリングを与えると、-3 から 3 までの整数が返ります。この整数は、どのようにクロスしているかを示すもので、0 は交差無しを意味します。この関数は **LINESTRING** にのみ対応しています。

交差の番号は次の通りです。

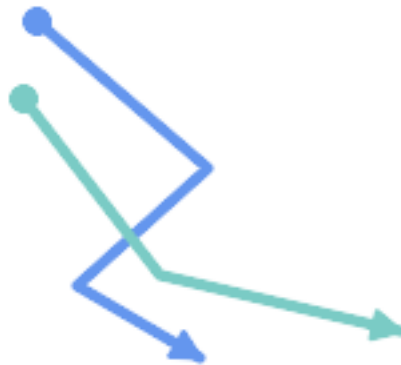
- 0: クロスが無い
- -1: 左クロス

- 1: 右クロス
- -2: 複数クロスで最後が左
- 2: 複数クロスで最後が右
- -3: 複数クロスで開始終了ともに左
- 3: 複数クロスで開始終了ともに右

Availability: 1.4

例

例: 左クロスと右クロス

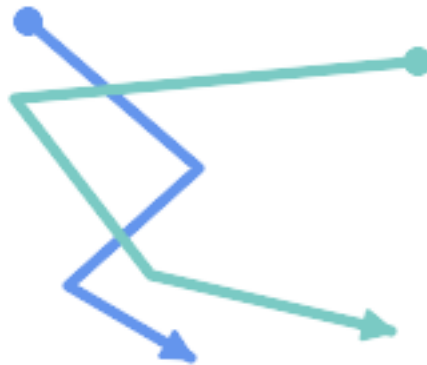


青: ライン A、緑: ライン B

```
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

例: 複数回クロスで開始終了ともに左と、複数回クロスで開始終了ともに右

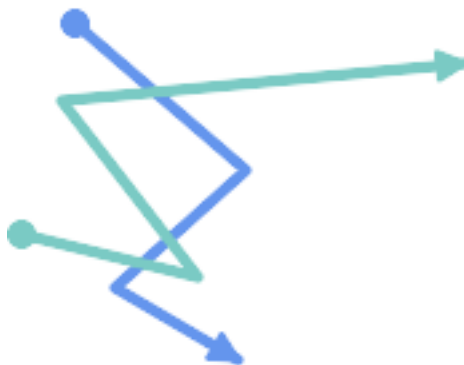


青: ライン A、緑: ライン B

```
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

例: 複数回クロスで最後が左と複数回クロスで最後が右



青: ライン A、緑: ライン 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
```

例: 全てのクロスするストリートを見つける

```
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;
```

関連情報

ST_Crosses

7.11.1.11 ST_OrderingEquals

ST_OrderingEquals — 二つのジオメトリが同じジオメトリを表現し、かつ点の並び順が同じかどうかをテストします。

Synopsis

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

説明

ST_OrderingEquals は、二つのジオメトリを比較して、ジオメトリが同じで、座標値が同じ順序である場合には、t (TRUE) を返し、それ以外の場合には、f (FALSE) を返します。



Note

この関数は、SQL-MM 仕様ではなく ArcSDE SQL 仕様に従って実装しています。
http://edndoc.esri.com/arcsws/9.1/sql_api/sqlapi3.htm#ST_OrderingEquals をご覧ください。



このメソッドは SQL/MM 仕様の実装です。SQL-MM 3: 5.1.43

例

```
SELECT ST_OrderingEquals('LINESTRING(0 0, 10 10)',
  'LINESTRING(0 0, 5 5, 10 10)');
```

f

```
SELECT ST_OrderingEquals('LINESTRING(0 0, 10 10)',
  'LINESTRING(0 0, 10 10)');
```

t

```
SELECT ST_OrderingEquals('POLYGON((0 0, 0 1, 1 1, 1 0, 0 0))',
  'POLYGON((0 0, 1 0, 1 1, 0 1, 0 0))');
```

f

関連情報

&&, [ST_Equals](#), [ST_Reverse](#)

7.11.1.12 ST_Overlaps

ST_Overlaps — 二つのジオメトリが同じ次元を持ち、インタセクトして、かつ相手と重ならない点少なくとも一つあるかをテストします。

Synopsis

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

説明

ジオメトリ A と B が「空間的にオーバーラップする」場合に TRUE を返します。ジオメトリが同じ次元で、内部のインタセクションも同じ次元で、少なくとも一つの点がもう一方の外側にある (一方がもう一方を覆っている状態でないのと等価です) と、二つのジオメトリがオーバーラップしていると言います。オーバーラップの関係は、対称性があり、無反射性があります。

数学用語では: $ST\_Overlaps(A, B) \Leftrightarrow (dim(A) = dim(B) = dim(Int(A) \cap Int(B))) \wedge (A \cap B \neq A) \wedge (A \cap B \neq B)$



Note

この関数を実行すると、対象ジオメトリにおいて使用できるインデックスを使用したバウンディングボックスによる比較が自動的に行われます。インデックスの使用を避けるには `_ST_Overlaps` 関数を使います。

GEOS モジュールで実現しています。



Important

Enhanced: 3.0.0 GEOMETRYCOLLECTION への対応が可能となりました

ご注意: これは論理値を返して整数を返さないのが「許される」版です。



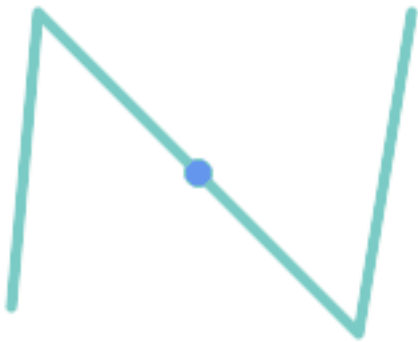
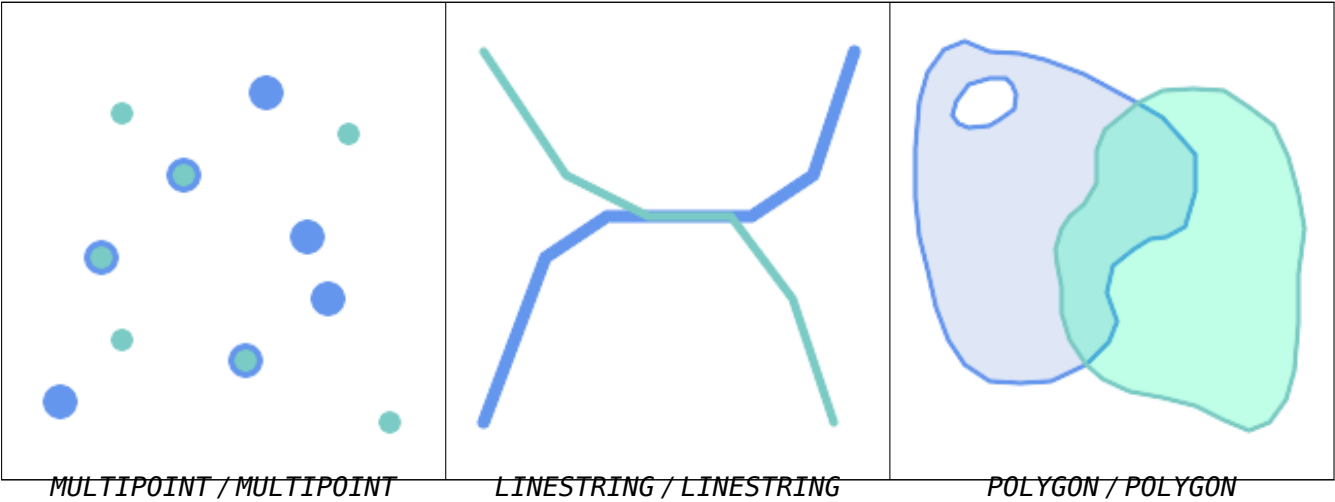
このメソッドは [OGC Simple Features Implementation Specification for SQL 1.1](#) の実装です。s2.1.1.2 // s2.1.13.3



このメソッドは SQL/MM 仕様の実装です。SQL-MM 3: 5.1.32

例

次に示す図全てで、ST_Overlaps は TRUE を返します。



ラインストリング上のポイントが含まますが、次元が低いのでオーバラップにもクロスにもなりません。

```
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
```

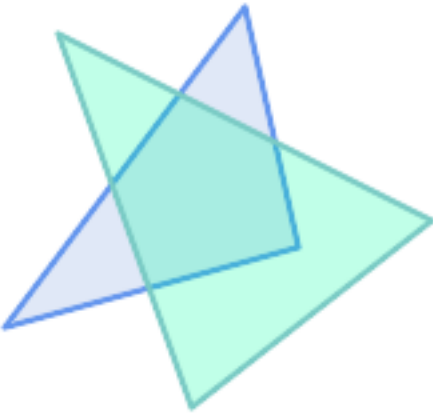
overlaps	crosses	intersects	b_contains_a
f	f	t	t



部分的にポリゴンを覆うラインストリングはインタセクトもクロスもしますが、異なり次元なのでオーバラップしません。

```
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



二つのポリゴンがインタセクトするものの一方が他方のオーバラップを含んでいませんが、インタセクトが同じ次元なのでクロスしません。

```
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

関連情報

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

7.11.1.13 ST_Relate

ST_Relate — 二つのジオメトリが与えられた交差行列パターンに合致するトポロジ関係があるかどうかを見るか、交差行列を計算するかします。

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);
```

説明

これらの関数で、二つのジオメトリに対する[次元拡張 9 交差モデル](#) (Dimensionally Extended 9-Intersection Model, DE-9IM) で定義される空間 (トポロジ) 関係のテストと評価が可能になります。

DE-9IM は 9 要素の行列で、二つのジオメトリの内部、境界、外部のインタセクションの次元を示しています。'F', '0', '1', '2' の文字による 9 文字の文字列で表現します (例えば 'FF1FF0102')。

intersection matrix pattern で、特定の種類の空間関係のテストができます。パターンでは追加の文字'T' (「インタセクションが空でない」という意味) と'*' (「値はなんでもいい」という意味) を使うことができます。一般的な空間関係については、名前の付いた関数[ST_Contains](#), [ST_ContainsProperly](#), [ST_Covers](#), [ST_CoveredBy](#), [ST_Crosses](#), [ST_Disjoint](#), [ST_Equals](#), [ST_Intersects](#), [ST_Overlaps](#), [ST_Touches](#), [ST_Within](#)として提供されます。明示的なパターンを使うことで、インタセクト、クロス等の複数のテストを一回でできるようになります。また、名前のある空間関係関数を持たない空間関係のテストも可能になります。たとえば、「内部インタセクション」という空間関係は、名前の付いた空間関係関数では評価できませんが、T***** の DE-9IM で評価できます。

詳細情報については [Section 5.1](#)をご覧ください。

一つ目の形式: 二つのジオメトリが、与えられた `intersectionMatrixPattern` による空間関係に合うかどうかを見ます。



Note

他の名前の付いた空間関係述語の多くと異なり、この関数は自動ではインデックスの呼び出しを * 行いません *。インタセクト * しない * ジオメトリで TRUE になる関係があるためです。インタセクションが求められる関係パターンを使用している場合には、関数呼び出しに && を取り入れてください。



Note

存在するなら、名前の追加空間関係関数を使う方が良いです。空間インデックスが存在するなら自動で使ってくれるからです。また、完全な関係評価では有効にならない能率最適化が実装されていることがあります。

二つ目の形式: 二つのジオメトリの空間関係の DE-9IM 行列文字列を返します。行列文字列は、**ST_RelateMatch** を使った DE-9IM パターンと合致するかテストさせることができます。

三つ目の形式: 二つ目の形式と同じですが、境界ノード規則の指定ができる点が異なります。この規則によって、マルチラインストリングの端点が DE-9IM の内部または境界上にあると判定されるかどうかを細かく制御できます。boundaryNodeRule の値は次の通りです:

- **1: OGC-Mod2** - 線の端点が奇数回出現する場合に境界内にあるとします。これは OGC SFS 標準で定義された規則で、ST_Relate のデフォルトです。
- **2: Endpoint** - 全ての端点は境界上にあります。
- **3: MultivalentEndpoint** - 端点が 2 回以上出現する場合に境界内にあるとします。言い換えると、境界は全ての「接続された」または「内部の」端点です (「接続していない」や「外部の」端点ではない)。
- **4: MonovalentEndpoint** - 端点が 1 回だけ出現する場合に限って、境界内にあるとします。言い換えると境界は全ての「接続していない」または「外部の」端点です。

OGC 仕様にはありませんが実装しました。s2.1.13.2 をご覧下さい。

✔ このメソッドは **OGC Simple Features Implementation Specification for SQL 1.1** の実装です。s2.1.1.2 // s2.1.13.3

✔ このメソッドは SQL/MM 仕様の実装です。SQL-MM 3: 5.1.25

GEOS モジュールで実現しています。

Enhanced: 2.0.0 - 境界ノード規則が追加されました。



Important

Enhanced: 3.0.0 GEOMETRYCOLLECTION への対応が可能となりました

例

真偽値関数を使って空間関係を見ます。

```
SELECT ST_Relate('POINT(1 2)', ST_Buffer('POINT(1 2)', 2), '0FFFFFF212');
```

t

```
SELECT ST_Relate('POINT(1 2)', ST_Buffer('POINT(1 2)', 2), '*FF*FF212');
```

t

独自の空間関係パターンを問い合わせ条件としてテストします。空間インデックスの使用を有効にするために && を使っています。

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*****');
```

空間関係交差行列を計算します。

```
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
```

異なる境界ノード規則を使って、端点が重複する LINESTRING と MULTILINESTRING 間の空間関係を計算します (3 3):

- **OGC-Mod2** 規則 (1) を使うと、重複端点は MULTILINESTRING の内部になり、DE-9IM 行列の [aB:bl] は 0 で、[aB:bB] は F です。
- **Endpoint** 規則 (2) を使うと、重複端点は MULTILINESTRING の境界となり、DE-9IM 行列の [aB:bl] は F で、[aB:bB] は 0 です。

```
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
```

関連情報

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` — DE-9IM 交差行列が交差行列パターンに合致するかどうかを見ます。

Synopsis

boolean **ST_RelateMatch**(text intersectionMatrix, text intersectionMatrixPattern);

説明

次元拡張 9 交差モデル (Dimensionally Extended 9-Intersection Model, DE-9IM) `intersectionMatrix` の値が `intersectionMatrixPattern` を満たすかどうかを見ます。交差行列値は [ST_Relate](#) で計算します。

詳細情報については Section 5.1 をご覧下さい。

GEOS モジュールで実現しています。

Availability: 2.0.0

例

```
SELECT ST_RelateMatch('101202FFF', 'TTTTTTFFF') ;
```

t

あるポリゴンとの様々な相対的な位置にあるラインを想定した交差行列値に合致する一般的な空間関係のパターン

```
SELECT pat.name AS relationship, pat.val AS pattern,
       mat.name AS position, mat.val AS matrix,
       ST_RelateMatch(mat.val, pat.val) AS match
FROM (VALUES ( 'Equality', 'T1FF1FFF1' ),
      ( 'Overlaps', 'T*T***T**' ),
      ( 'Within', 'T*F**F***' ),
      ( 'Disjoint', 'FF*FF*****' )) AS pat(name,val)
CROSS JOIN
  (VALUES ('non-intersecting', 'FF1FF0212'),
    ('overlapping', '1010F0212'),
    ('inside', '1FF0FF212')) AS mat(name,val);
```

relationship	pattern	position	matrix	match
Equality	T1FF1FFF1	non-intersecting	FF1FF0212	f
Equality	T1FF1FFF1	overlapping	1010F0212	f
Equality	T1FF1FFF1	inside	1FF0FF212	f
Overlaps	T*T***T**	non-intersecting	FF1FF0212	f
Overlaps	T*T***T**	overlapping	1010F0212	t
Overlaps	T*T***T**	inside	1FF0FF212	f
Within	T*F**F***	non-intersecting	FF1FF0212	f
Within	T*F**F***	overlapping	1010F0212	f
Within	T*F**F***	inside	1FF0FF212	t
Disjoint	FF*FF*****	non-intersecting	FF1FF0212	t
Disjoint	FF*FF*****	overlapping	1010F0212	f
Disjoint	FF*FF*****	inside	1FF0FF212	f

関連情報

Section 5.1, [ST_Relate](#)

7.11.1.15 ST_Touches

ST_Touches — 二つのジオメトリが少なくとも一つの共有点を持ち、かつ内部でインタセクトしていないようになっているかテストします。

Synopsis

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

説明

A と B がインタセクトするが A の内部と B の内部がインタセクトしない場合には **TRUE** を返します。A と B が少なくとも一つの共有点があり、共有点が少なくとも一つの境界の上にあることと同じです。ポイント/ポイント入力では、ポイントは境界を持たないため、常に **FALSE** を返します。

数学用語では: $ST\_Touches(A, B) \Leftrightarrow (Int(A) \cap Int(B) = \emptyset) \wedge (A \cap B \neq \emptyset)$

この関係は、二つのジオメトリの DE-9IM 交差行列がどれか一つに合致すると、関係が保持されていることとなります:

- FT*****
- F**T*****
- F***T****

**Note**

この関数を実行すると、対象ジオメトリにおいて使用できるインデックスを使用したバウンディングボックスによる比較が自動的行われます。インデックスの使用を避けるには `_ST_Touches` を代わりに使います。

**Important**

Enhanced: 3.0.0 GEOMETRYCOLLECTION への対応が可能となりました



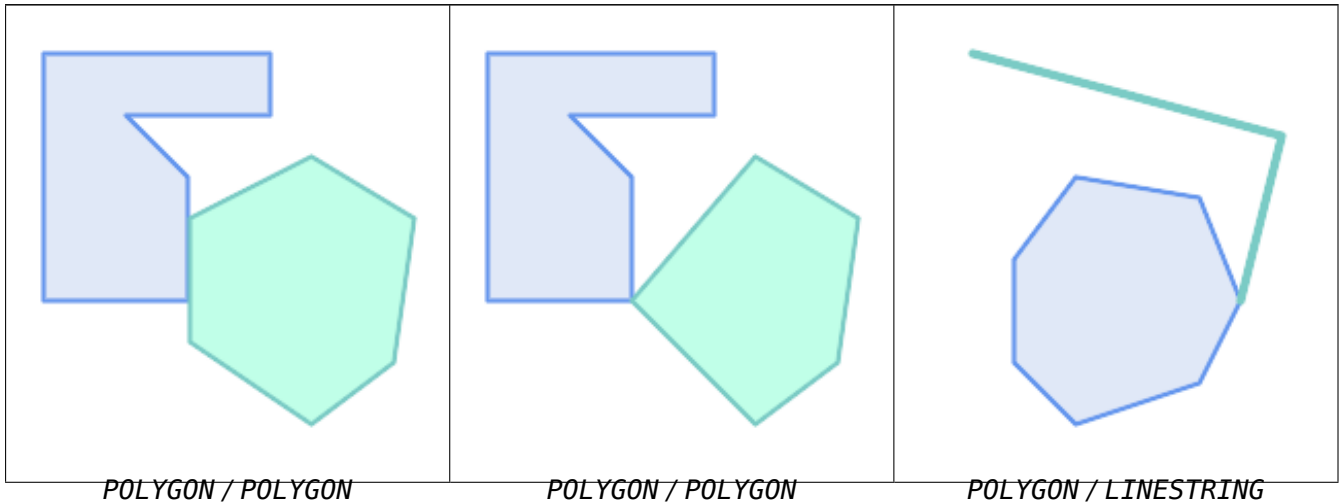
このメソッドは **OGC Simple Features Implementation Specification for SQL 1.1** の実装です。s2.1.1.2 // s2.1.13.3

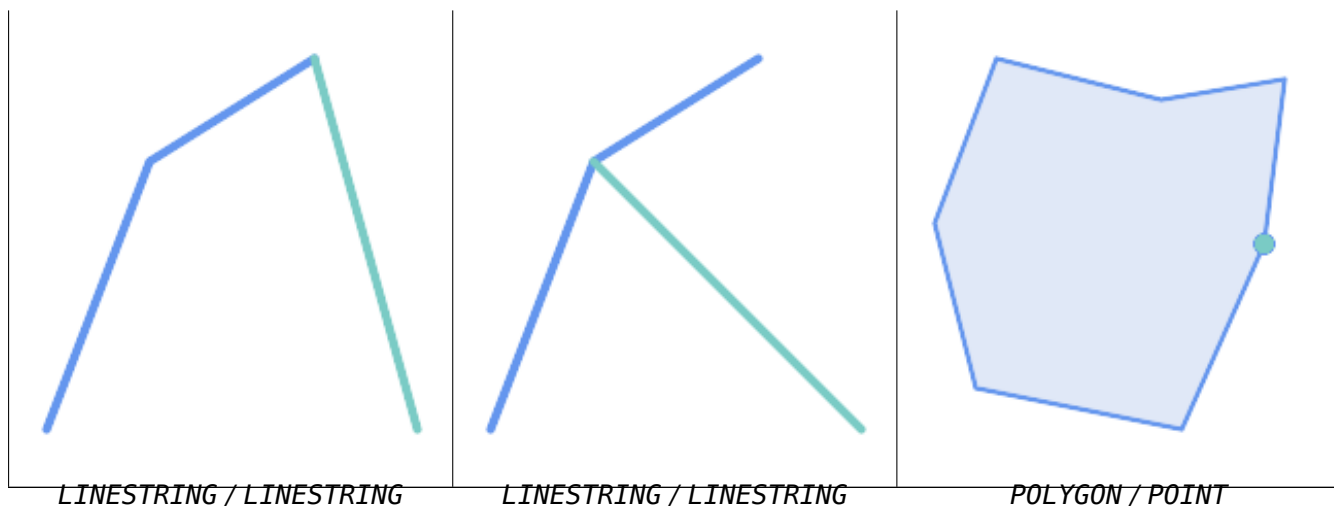


このメソッドは SQL/MM 仕様の実装です。SQL-MM 3: 5.1.28

例

次の例では `ST_Touches` 述語は `TRUE` を返します。





```
SELECT ST_Touches('LINESTRING(0 0, 1 1, 0 2)::geometry, 'POINT(1 1)::geometry');
```

```
f
```

```
SELECT ST_Touches('LINESTRING(0 0, 1 1, 0 2)::geometry, 'POINT(0 2)::geometry');
```

```
t
```

7.11.1.16 ST_Within

ST_Within — A の全てのポイントが B 内にあり、かつ両方の内部が共有点を持つかどうかをテストします。

Synopsis

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

説明

ジオメトリ A がジオメトリ B 内にある場合に TRUE を返します。A の全ての点が B の内側 (内部または境界) にあり、A の内部と B の内部に少なくとも一つの共有点がある場合に限って、A は B 内にあると言えます。

渡すジオメトリが両方とも同じ SRID でなければ、この関数は意味がありません。

数学用語では: $ST\_Within(A, B) \Leftrightarrow (A \sqcap B = A) \wedge (Int(A) \sqcap Int(B) \neq \square)$

内にある、という関係性には反射性があります。全てのジオメトリは、自分の内にあると言えます。また、非対称性を持ちます。ST_Within(A,B) = true かつ ST_Within(B,A) = true の場合には、二つのジオメトリは必ずトポロジ的に等価 (ST_Equals(A,B) = true) になります。

ST_Within は **ST_Contains** の反対です。ST_Within(A,B) = ST_Contains(B,A) となります。



Note

内部が必ず共有点を持つので、定義は、ポリゴンまたはラインの境界上にあるラインまたはポイントはジオメトリの内にないという微妙なものになります。詳細については、[Subtleties of OGC Covers, Contains, Within](#) をご覧下さい。ST_CoveredBy 述語はより包括的な関係を提供します。

**Note**

この関数を実行すると、対象ジオメトリにおいて使用できるインデックスを使用したバウンディングボックスによる比較が自動的行われます。インデックスの使用を避けるには `_ST_Within` 関数を使います。

GEOS モジュールで実現しています。

Enhanced: 2.3.0 ジオメトリについて、PIP short-circuit (ポリゴンとポイントに限定した高速判定) を少ないポイントからなるマルチポイントに対応することができるよう拡張しました。以前の版ではポリゴンとポイントの組み合わせにだけ対応していました。

**Important**

Enhanced: 3.0.0 GEOMETRYCOLLECTION への対応が可能となりました

**Important**

この関数を不正なジオメトリで呼ばないでください。予期しない結果が返されます。

ご注意: これは論理値を返して整数を返さないのが「許される」版です。



このメソッドは **OGC Simple Features Implementation Specification for SQL 1.1** の実装です。s2.1.1.2 // s2.1.13.3 - a.Relate(b, 'T**F***')



このメソッドは SQL/MM 仕様の実装です。SQL-MM 3: 5.1.30

例

This example tests a circle within another circle.

```
SELECT ST_Within(smallc,smallc) As smallinsmall,
       ST_Within(smallc, bigc) As smallinbig,
       ST_Within(bigc,smallc) As biginsmall,
       ST_Within(ST_Union(smallc, bigc), bigc) as unioninbig,
       ST_Within(bigc, ST_Union(smallc, bigc)) as beginunion,
       ST_Equals(bigc, ST_Union(smallc, bigc)) as bigisunion
FROM
(
SELECT ST_Buffer(ST_GeomFromText('POINT(50 50)'), 20) As smallc,
       ST_Buffer(ST_GeomFromText('POINT(50 50)'), 40) As bigc) As foo;
```

smallinsmall	smallinbig	biginsmall	unioninbig	beginunion	bigisunion
t	t	f	t	t	t

(1 row)



関連情報

[ST_Contains](#), [ST_CoveredBy](#), [ST_Equals](#), [ST_IsValid](#)

7.11.2 距離関係関数

7.11.2.1 ST_3DDWithin

ST_3DDWithin — 二つの 3 次元ジオメトリが与えられた 3 次元距離内にあるかどうかをテストします。

Synopsis

```
boolean ST_3DDWithin(geometry g1, geometry g2, double precision distance_of_srid);
```

説明

二つのジオメトリ値の 3 次元距離が **distance_of_srid** 以下なら **TRUE** を返します。距離の単位はジオメトリの空間参照系の単位です。この関数から意味のある結果を得るには、引数に使うジオメトリを同じ座標系 (同じ SRID を持つ) にしなければなりません。



Note

この関数を実行すると、対象ジオメトリにおいて使用できるインデックスを使用したバウンディングボックスによる比較が自動的に行われます。

- ✔ この関数は 3 次元に対応し、Z 値を削除しません。
- ✔ この関数は多面体サーフェスに対応しています。
- ✔ このメソッドは SQL/MM 仕様の実装です。SQL-MM ?

Availability: 2.0.0

例

Geometry example - units in meters (SRID: 2163 US National Atlas Equal area) (3D point and line compared 2D point and line) Note that currently no vertical datum support so Z is not transformed and assumed to be same units as final.

```
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
```

関連情報

[ST_3DDFullyWithin](#), [ST_DWithin](#), [ST_DFullyWithin](#), [ST_3DDistance](#), [ST_Distance](#), [ST_3DMaxDistance](#), [ST_Transform](#)

7.11.2.2 ST_3DDFullyWithin

ST_3DDFullyWithin — 二つの 3 次元ジオメトリが完全に与えられた 3 次元距離内にあるかどうかをテストします。

Synopsis

boolean **ST_3DDFullyWithin**(geometry g1, geometry g2, double precision distance);

説明

3 次元ジオメトリが他のジオメトリとの距離が、完全に指定した範囲内なら **true** を返します。距離の単位はジオメトリの空間参照系で定義されているものとされます。この関数が意味を持つためには、与えられるジオメトリは両方とも同じ座標系で同じ **SRID** を持つ必要があります。



Note

この関数を実行すると、対象ジオメトリにおいて使用できるインデックスを使用したバウンディングボックスによる比較が自動的行われます。

Availability: 2.0.0



この関数は 3 次元に対応し、Z 値を削除しません。



この関数は多面体サーフェスに対応しています。

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
```

関連情報

[ST_3DDWithin](#), [ST_DWithin](#), [ST_DFullyWithin](#), [ST_3DMaxDistance](#)

7.11.2.3 ST_DFullyWithin

ST_DFullyWithin — ジオメトリが完全に他のジオメトリの指定距離内にあるかどうかをテストします

Synopsis

boolean **ST_DFullyWithin**(geometry g1, geometry g2, double precision distance);

説明

g2 が完全に g1 の distance で指定した距離内にある場合に TRUE を返します。見た目で言うと、g2 が g1 の distance ぶん膨らませたバッファに含まれる場合に TRUE を返します。

**Note**

この関数を実行すると、対象ジオメトリにおいて使用できるインデックスを使用したバウンディングボックスによる比較が自動的に行われます。

Availability: 1.5.0

Changed: 3.5.0 : この関数のロジックとしては、今のところ、バッファ内に包含するかどうかのテストを使っています。ST_MaxDistance アルゴリズムではありません。前の版と結果が異なる可能性があります、ユーザの期待に近づくはずです。

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

```

関連情報

[ST_MaxDistance](#), [ST_DWithin](#), [ST_3DDWithin](#), [ST_3DDFullyWithin](#)

7.11.2.4 ST_DWithin

ST_DWithin — 二つのジオメトリが与えられた距離内にあるかどうかをテストします。

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);

```

説明

ジオメトリが与えられた距離内にある場合には **TRUE** を返します。

geometry: 距離の単位は空間参照系で定義される単位です。この関数が意味のあるものにするためには、与えられるジオメトリは両方とも同じ座標系である (同じ **SRID** を持つ) 必要があります。

geography: 単位はメートルで、距離の測定の既定値は **use_spheroid = true** です。より高速な評価のために、**use_spheroid = false** として球面で測定します。



Note

3次元ジオメトリでは[ST_3DDWithin](#)を使います。



Note

この関数の呼び出しによって、ジオメトリで使用可能なインデックスを使用したバウンディングボックスの比較が自動的行われます。



このメソッドは[OGC Simple Features Implementation Specification for SQL 1.1](#)の実装です。

Availability: 1.5.0 ジオグラフィが導入されました。

Enhanced: 2.1.0 ジオグラフィ版の速度が改善しました。詳細については[Making Geography faster](#)をご覧ください。

Enhanced: 2.1.0 曲線ジオメトリ対応が導入されました。

1.3 以前の[ST_Expand](#)は、距離をテストするために、**&&** と、**ST_Distance** とを一般的に併用していました。1.3.4 より前では、この関数はそのロジックを使っていました。1.3.4 から **ST_DWithin** は、より速いショートサーキットを使った距離関数を使います。

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 a receiver with limited range can receive.
-- Data is geometry in Spherical Mercator (SRID=3857), ranges are approximate.

-- Create geometry index that will check proximity limit of user to tower
CREATE INDEX ON broadcasting_towers using gist (geom);

-- Create geometry index that will check proximity limit of tower to user
CREATE INDEX ON broadcasting_towers using gist (ST_Expand(geom, sending_range));

-- Query towers that 4-kilometer receiver in Minsk Hackerspace can get
-- Note: two conditions, because shorter LEAST(b.sending_range, 4000) will not use index.
SELECT b.tower_id, b.geom
  FROM broadcasting_towers b
 WHERE ST_DWithin(b.geom, 'SRID=3857;POINT(3072163.4 7159374.1)', 4000)
    AND ST_DWithin(b.geom, 'SRID=3857;POINT(3072163.4 7159374.1)', b.sending_range);
```

関連情報

[ST_Distance](#), [ST_3DDWithin](#)

7.11.2.5 ST_PointInsideCircle

ST_PointInsideCircle — ポイントジオメトリが中心と半径で定められた円の内側にあるかをテストします。

Synopsis

boolean **ST_PointInsideCircle**(geometry a_point, float center_x, float center_y, float radius);

説明

ジオメトリが点であり、center_x,center_y の中心点と radius の半径を持つ円の内側にある場合は TRUE を返します。

**Warning**

この関数は空間インデックスを使用しません。代わりに**ST_DWithin**を使って下さい。

Availability: 1.2

Changed: 2.2.0 前のバージョンでは ST_Point_Inside_Circle と呼ばれていました。

例

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

t

関連情報

ST_DWithin

7.12 計測関数

7.12.1 ST_Area

ST_Area — ポリゴンジオメトリの面積を返します。

Synopsis

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

説明

ポリゴンジオメトリの面積を返します。ジオメトリ型では 2 次元のデカルト (平面) 面積が、SRID で指定した単位で計算されます。ジオグラフィ型では、デフォルトでは、面積は回転楕円体上にあるものとし、平方メートル単位で計算されます。より速いものの精度が落ちる計算のために、**ST_Area(geog, false)** で球面モデルを使った計算ができます。

Enhanced: 2.0.0 - 2 次元多面体サーフェス対応が導入されました。

Enhanced: 2.2.0 - 精度とロバスト性の向上のために **GeographicLib** を使って回転楕円体面上での計測を行うようにしています。この新機能を使うには、**Proj 4.9.0** 以上が必要です。

Changed: 3.0.0 - SFCGAL に依存しなくなりました。

-  このメソッドは**OGC Simple Features Implementation Specification for SQL 1.1**の実装です。
-  このメソッドは SQL/MM 仕様の実装です。SQL-MM 3: 8.1.2, 9.5.3
-  この関数は多面体サーフェスに対応しています。



Note

多面体サーフェスについて、2 次元多面体サーフェスのみ対応します (2.5 次元は対応しません)。2.5 次元について、0 でない答が与えられるかも知れませんが、完全に XY 平面に付いているサーフェスのみです。

例

マサチューセッツのプロットから平方フィートの面積を返すものと、乗算で平方フィートから平方メートルに変換するものです。EPSG:2249 はマサチューセッツ州フィート平面であるので、平方フィートで面積が出ることに注意して下さい。

```

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;

```

平方フィートの面積を返し、平方メートルを得るためにメートル単位マサチューセッツ州平面 (EPSG:26986) に変換します。一つ目は EPSG:2249 でフィート単位マサチューセッツ州平面なので平方フィートとなり、二つ目は変換後は EPSG:26986 でメートル単位マサチューセッツ州平面なので平方メートルになるので、注意して下さい。

```

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;

```

ジオグラフィ型を使って平方フィートと平方メートルを返すものです。ジオメトリをジオグラフィに変換することに注意して下さい (変換に先立ってジオメトリが WGS 84 緯度経度の EPSG:4326 であることを確認します)。ジオグラフィは常にメートルで計測されます。これは比較のためのデモンストレーションです。通常は、ジオグラフィ型が既に格納されているものです。

```
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;
b''|b''-b''b''-b''b''-b''b''-b''b''-b''b''-b''b''-b''b''-b''b''-b''b''-b''b''-b''b''-b''b''- ←
b''-b''b''-b''b''-b''b''-b''b''-b''b''-b''b''-b''b''-b''b''-b''b''-b''b''-b''b''-b''b''- ←
'-b''b''-b''b''-b''b''-b''b''-b''b''-b''b''-b''b''-b''b''-b''b''-b''b''-b''b''-b''b''- ←
'-b''b''-b''b''-b''b''-b''b''-b''b''-b''b''-b''b''-b''b''-b''b''-b''b''-b''b''-b''b''- ←
'-b''b''-b''b''-b''b''-b''b''-b''b''-b''b''-b''b''-b''b''-b''b''-b''b''-b''b''-b''b''- ←
b''|b'' sqft_spheroid b''|b'' sqft_sphere b''|b'' sqm_spheroid b''|b''
b''|b''b''-b''b''-b''b''-b''b''-b''b''-b''b''-b''b''-b''b''-b''b''-b''b''-b''b''-b''b''-b''b''- ←
b''-b''b''-b''b''-b''b''-b''b''-b''b''-b''b''-b''b''-b''b''-b''b''-b''b''-b''b''-b''b''- ←
'-b''b''-b''b''-b''b''-b''b''-b''b''-b''b''-b''b''-b''b''-b''b''-b''b''-b''b''-b''b''- ←
'-b''b''-b''b''-b''b''-b''b''-b''b''-b''b''-b''b''-b''b''-b''b''-b''b''-b''b''-b''b''- ←
'-b''b''-b''b''-b''b''-b''b''-b''b''-b''b''-b''b''-b''b''-b''b''-b''b''-b''b''-b''b''- ←
b''|b'' 928.684405784452 b''|b'' 927.049336105925 b''|b'' 86.2776044979692 b''|b''
b''|b''b''-b''b''-b''b''-b''b''-b''b''-b''b''-b''b''-b''b''-b''b''-b''b''-b''b''-b''b''-b''b''- ←
b''-b''b''-b''b''-b''b''-b''b''-b''b''-b''b''-b''b''-b''b''-b''b''-b''b''-b''b''-b''b''- ←
'-b''b''-b''b''-b''b''-b''b''-b''b''-b''b''-b''b''-b''b''-b''b''-b''b''-b''b''-b''b''- ←
'-b''b''-b''b''-b''b''-b''b''-b''b''-b''b''-b''b''-b''b''-b''b''-b''b''-b''b''-b''b''- ←
'-b''b''-b''b''-b''b''-b''b''-b''b''-b''b''-b''b''-b''b''-b''b''-b''b''-b''b''-b''b''- ←

```

ジオグラフィのデータが既にある場合:

```

select ST_Area(geog) / 0.3048 ^ 2 sqft,
       ST_Area(the_geog) sqm
from somegeogtable;

```

関連情報

[ST_3DArea](#), [ST_GeomFromText](#), [ST_GeographyFromText](#), [ST_SetSRID](#), [ST_Transform](#)

7.12.2 ST_Azimuth

ST_Azimuth — 北を基準とした 2 点間の線の方角角を返します。

Synopsis

```

float ST_Azimuth(geometry origin, geometry target);
float ST_Azimuth(geography origin, geography target);

```

説明

原点から目標点に向いたラジアン単位の方角を返しますが、2 点が一致する場合には NULL を返します。方角角は、Y 軸 (ジオメトリ) または子午線北向き (ジオグラフィ) から時計回りに増えていきます。すなわち、北は 0、北東は $\pi/4$ 、東は $\pi/2$ 、南東は $3\pi/4$ 、南は π 、南西は $5\pi/4$ 、西は $3\pi/2$ 、北西は $7\pi/4$ 、となります。

ジオグラフィでは、方角角計算問題は [inverse geodesic problem](#) として知られます。

方角角は参照ベクトルと一つのポイントとの間の角度と定義される数学的概念で、角度の単位はラジアンです。ラジアン単位の結果は PostgreSQL 関数 `degrees()` で度に変換できます。

方位角は、垂直軸に沿ってシフトさせるには`ST_Translate`と併用します。実装についてはPostGIS wiki内の関数 `upgis_lineshift()` をご覧下さい。

Availability: 1.1.0

Enhanced: 2.0.0 ジオグラフィ対応が導入されました。

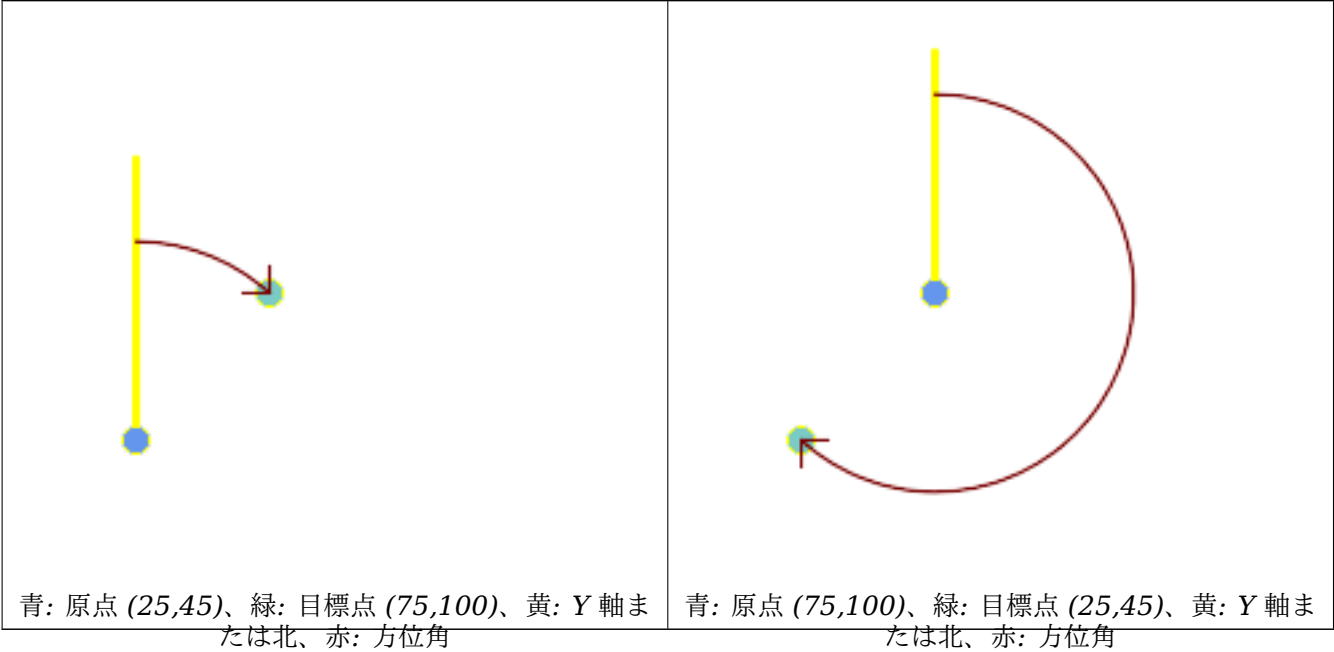
Enhanced: 2.2.0 - 精度とロバスト性の向上のために `GeographicLib` を使って回転楕円体面上での計測を行うようになっています。この新機能を使うには、`Proj 4.9.0` 以上が必要です。

例

度単位のジオメトリの方位

```
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
-----+-----	
42.27368900609371	222.27368900609372



関連情報

`ST_Angle`, `ST_Point`, `ST_Translate`, `ST_Project`, [PostgreSQL Math Functions](#)

7.12.3 ST_Angle

`ST_Angle` — 3 点もしくは 4 点、または 2 線で定義される二つのベクタ間の角度を返します。

Synopsis

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

説明

二つのベクタの時計回りの角度を計算します。

形式 **1**: P1-P2-P3 がなす角度を計算します。4 番目のポイントが充てられた場合には、P1-P2 と P3-P4 がなす角度を計算します。

形式 **2**: 入力ラインで定義される始端と終端ベクトル S1-E1 と S2-E2 の間の角度を計算します。

結果は正の角度で 0 と 2π ラジアンの間です。ラジアンは PostgreSQL 関数 `degrees()` で度に変換できます。

`ST_Angle(P1,P2,P3) = ST_Angle(P2,P1,P2,P3)` となることに注意して下さい。

Availability: 2.5.0

例

3 点間の角度

```
SELECT degrees(ST_Angle('POINT(0 0)', 'POINT(10 10)', 'POINT(20 0)'));
```

270

4 点で定義されたベクタ間の角度

```
SELECT degrees(ST_Angle('POINT (10 10)', 'POINT (0 0)', 'POINT(90 90)', 'POINT (100 80)'));
```

269.99999999999999

ラインの始端と終端で定義されるベクタ間の角度

```
SELECT degrees(ST_Angle('LINESTRING(0 0, 0.3 0.7, 1 1)', 'LINESTRING(0 0, 0.2 0.5, 1 0)'));
```

45

関連情報

[ST_Azimuth](#)

7.12.4 ST_ClosestPoint

`ST_ClosestPoint` — g1 上にある、g2 と最近傍となる 2 次元ポイントを返します。これは、あるジオメトリから他のジオメトリへの最短ラインの一つ目のポイントです。

Synopsis

geometry **ST_ClosestPoint**(geometry geom1, geometry geom2);

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

説明

geom2 と最近傍となる geom1 上の 2 次元ポイントを返します。これは、ジオメトリ間の最短ライン ([ST_ShortestLine](#) で生成されるものと同じです) の一つ目のポイントです。



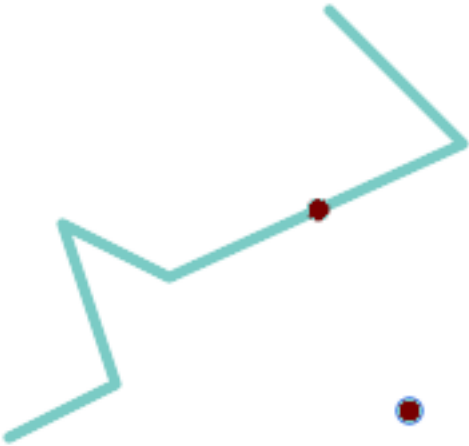
Note

3 次元ジオメトリの場合には[ST_3DClosestPoint](#)の方が良いでしょう。

Enhanced: 3.4.0 - ジオグラフィに対応しました。

Availability: 1.5.0

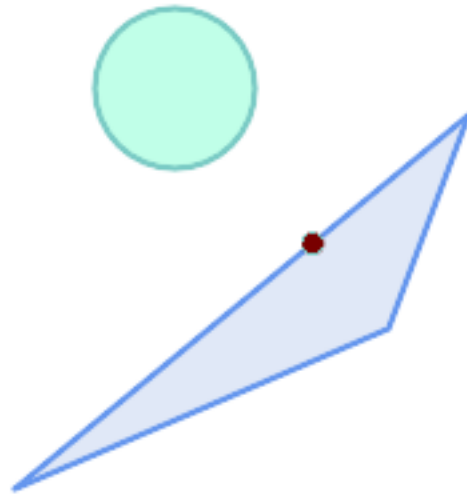
例



ポイントとラインストリングの最近傍ポイントは引数で与えたポイントそのものです。ラインストリングとポイントの最近傍ポイントはラインストリング上のポイントです。

```
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)



ポリゴン A 上のポリゴン B への最近傍点

```
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 101.89887534906195)
```

関連情報

[ST_3DClosestPoint](#), [ST_Distance](#), [ST_LongestLine](#), [ST_ShortestLine](#), [ST_MaxDistance](#)

7.12.5 ST_3DClosestPoint

ST_3DClosestPoint — g1 上の、g2 に最も近い 3 次元ポイントを返します。これは 3 次元の最短ラインの始点です。

Synopsis

geometry **ST_3DClosestPoint**(geometry g1, geometry g2);

説明

g1 上の、g2 に最も近い 3 次元ポイントを返します。3 次元最短線の一つ目のポイントです。3 次元最短線の長さは、3 次元距離と同じです。



この関数は 3 次元に対応し、Z 値を削除しません。



この関数は多面体サーフェスに対応しています。

Availability: 2.0.0

Changed: 2.2.0 - 二つの 2 次元ジオメトリが入力である場合には、2 次元ポイントが返ります (古い挙動では、存在しない Z の値について 0 を仮定していました)。2 次元と 3 次元の場合には、もはや、存在しない Z の値について 0 を仮定しません。

例

ラインストリングとポイント -- 3次元と2次元の両方の最も近いポイント	
<pre>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;</pre>	
cp3d_line_pt	cp2d_line_pt
-----+-----	
POINT(54.69937988676193 128.93502291722837 11.547586950660556) POINT(100 100 30)	
ラインストリングとマルチポイント -- 3次元と2次元の両方の最も近いポイント	
<pre>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;</pre>	
cp3d_line_pt	cp2d_line_pt
-----+-----	
POINT(54.69937988676193 128.93502291722837 11.547586950660556) POINT(50 75)	
マルチラインストリングとポリゴン -- 3次元と2次元の両方の最も近いポイント	
<pre>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;</pre>	
cp3d	cp2d
-----+-----	
POINT(40.45454545454545 53.63636363636364 5) POINT(20 40)	

関連情報

[ST_AsEWKT](#), [ST_ClosestPoint](#), [ST_3DDistance](#), [ST_3DShortestLine](#)

7.12.6 ST_Distance

ST_Distance — 二つのジオメトリ値またはジオグラフィ値間の距離を返します。

Synopsis

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

説明

geometry型では、二つのジオメトリ間の 2 次元のデカルト (平面) 距離の最小値を返します。単位は投影の単位 (空間参照系の単位) です。

geography型では、二つのジオグラフィ間の最小の測地距離を、メートル単位で返します。SRID で決定される回転楕円体で計算します。**use_spheroid** が **FALSE** の場合には、球面が計算に使われます。

✔ このメソッドは **OGC Simple Features Implementation Specification for SQL 1.1** の実装です。

✔ このメソッドは SQL/MM 仕様の実装です。SQL-MM 3: 5.1.23

✔ このメソッドは曲線ストリングと曲線に対応しています。

Availability: 1.5.0 1.5 でジオグラフィ対応が導入されました。大きいジオメトリや頂点の多いジオメトリについての速度が改善しました

Enhanced: 2.1.0 ジオグラフィ版の速度が改善しました。詳細については **Making Geography faster** をご覧ください。

Enhanced: 2.1.0 - 曲線ジオメトリ対応が導入されました。

Enhanced: 2.2.0 - 精度とロバスト性の向上のために **GeographicLib** を使って回転楕円体面上での計測を行うようにしています。この新機能を使うには、**Proj 4.9.0** 以上が必要です。

Changed: 3.0.0 - SFCGAL に依存しなくなりました。

ジオメトリの例

ジオメトリの例 - 平面の単位で 4326 は WGS84 経度緯度なので、単位は度。

```
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

ジオメトリの例 - メートル単位 (SRID: 3857, 一般的な Web マップのピクセルに比例)。値は正しくありませんが、近傍の比較は正しくでき、KNN や KMeans などのアルゴリズムに適しています。

```
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

ジオメトリの例 - メートル単位 (SRID:3857, 上と同じですが、歪みを考慮して cos(lat) で修正しています)

```
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

ジオメトリの例 - メートル単位 (SRID: 26986, マサチューセッツ州平面メートル) (マサチューセッツで最も精度が高い)

```
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
```

ジオメトリの例 - メートル単位 (SRID: 2163, 米国ナショナルアトラス正積図法) (最も精度が低い)

```
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
```

ジオグラフィの例

ジオメトリの例と同じですが、メートル単位である点に注意して下さい - 球面の使用で、わずかな速度向上と低精度の計算を行います。

```
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
```

関連情報

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

7.12.7 ST_3DDistance

ST_3DDistance — 投影座標系の単位で、二つのジオメトリ間の 3 次元デカルト距離の最小値を返します (空間参照系に基づきます)。

Synopsis

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


説明

投影座標系の単位で、二つのジオメトリ間の 3 次元デカルト距離の最小値を返します (空間参照系に基づきます)。

- ✔ この関数は 3 次元に対応し、Z 値を削除しません。
- ✔ この関数は多面体サーフェスに対応しています。
- ✔ このメソッドは SQL/MM 仕様の実装です。SQL-MM ISO/IEC 13249-3

Availability: 2.0.0

Changed: 2.2.0 - 2 次元と 3 次元の場合には、もはや、存在しない Z の値について 0 を仮定しません。

Changed: 3.0.0 - SFCGAL 版は削除されました

例

Geometry example - units in meters (SRID: 2163 US National Atlas Equal area) (3D point and line compared 2D point and line) Note that currently no vertical datum support so Z is not transformed and assumed to be same units as final.

```
SELECT ST_3DDistance(
    ST_Transform('SRID=4326;POINT(-72.1235 42.3521 4) '::geometry,2163),
    ST_Transform('SRID=4326;LINESTRING(-72.1260 42.45 15, ←
    -72.123 42.1546 20) '::geometry,2163)
) As dist_3d,
ST_Distance(
    ST_Transform('SRID=4326;POINT(-72.1235 42.3521) '::geometry ←
    ,2163),
    ST_Transform('SRID=4326;LINESTRING(-72.1260 42.45, -72.123 ←
    42.1546) '::geometry,2163)
) As dist_2d;
```

dist_3d		dist_2d
127.295059324629		126.66425605671

Multilinestring and polygon both 3D and 2D distance. Same example as 3D closest point example.

```
SELECT ST_3DDistance(poly, mline) As dist3d,
    ST_Distance(poly, mline) As dist2d
FROM (SELECT 'POLYGON((175 150 5, 20 40 5, 35 45 5, 50 60 5, 100 100 5, 175 150 5) ←
)::geometry as poly,
    'MULTILINESTRING((175 155 2, 20 40 20, 50 60 -2, 125 100 1, 175 155 1), (1 ←
    10 2, 5 20 1)) '::geometry as mline) as foo;
```

dist3d		dist2d
0		0

関連情報

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

7.12.8 ST_DistanceSphere

ST_DistanceSphere — 球面の地球モデルを使って、二つの経度/緯度ジオメトリの最小距離をメートル単位で返します。

Synopsis

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

説明

二つの経度緯度ジオメトリの間の最短距離をメートル単位で返します。SRID で定義された回転楕円体に由来する半径となる球面を使います。**ST_DistanceSpheroid**より高速ですが、精度が悪くなります。PostGIS の 1.5 より前の版ではポイント間の距離の計測だけを実装していました。

Availability: 1.5 - ポイント以外のジオメトリが導入されました。以前の版ではポイントでのみ動作しました。

Changed: 2.2.0 前の版ではこの関数は ST_Distance_Sphere と呼ばれていました

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
70424.71	70438.00	0.72900	65871.18

関連情報

ST_Distance, **ST_DistanceSpheroid**

7.12.9 ST_DistanceSpheroid

ST_DistanceSpheroid — 回転楕円体面の地球モデルを使って、二つの経度/緯度ジオメトリの最小距離を返します。

Synopsis

float **ST_DistanceSpheroid**(geometry geomlonlatA, geometry geomlonlatB, spheroid measurement_spheroid)

説明

与えられた回転楕円体面の、二つの経度/緯度ジオメトリの最小距離をメートル単位で返します。回転楕円体面の詳細については[ST_LengthSpheroid](#)をご覧ください。



Note

この関数はジオメトリの SRID を見ません。ジオメトリの座標は与えられた回転楕円体に基づくものと仮定します。

Availability: 1.5 - ポイント以外のジオメトリが導入されました。以前の版ではポイントでのみ動作しました。

Changed: 2.2.0 前の版ではこの関数は `ST_Distance_Sphere` と呼ばれていました

例

```
SELECT round(
    CAST(
        ST_DistanceSpheroid(ST_Centroid(geom), ST_GeomFromText('POINT(-118 38)', 4326), 'SPHEROID["WGS 84",6378137,298.257223563]')
        As numeric),2) As dist_meters_spheroid,
    round(CAST(ST_DistanceSphere(ST_Centroid(geom), ST_GeomFromText('POINT(-118 38)', 4326)) As numeric),2) As dist_meters_sphere,
    round(
        CAST(ST_Distance(
            ST_Transform(ST_Centroid(geom), 32611),
            ST_Transform(ST_GeomFromText('POINT(-118 38)', 4326), 32611)
        ) As numeric),2) As dist_utm11_meters
FROM
    (SELECT ST_GeomFromText('LINESTRING(-118.584 38.374,-118.583 38.5)', 4326) As geom)
as foo;
```

dist_meters_spheroid	dist_meters_sphere	dist_utm11_meters
70454.92	70424.71	70438.00

関連情報

[ST_Distance](#), [ST_DistanceSphere](#)

7.12.10 ST_FrechetDistance

`ST_FrechetDistance` — 二つのジオメトリのフレシェ距離を返します。

Synopsis

float **ST_FrechetDistance**(geometry g1, geometry g2, float densifyFrac = -1);

説明

両方のジオメトリの離散点への制限を受けたフレシェ距離の計算アルゴリズムの実装は **Computing Discrete Fréchet Distance** を基にしています。フレシェ距離は曲線の位置と点の並び順とを考慮に入れた曲線間の類似度を計測するものです。ハウスドルフ距離よりも良いことがしばしばあります。

任意引数 `densifyFrac` が指定されると、この関数は、離散フレシェ距離の計算の前に、辺密度を増加させます。`densifyFrac` パラメータは辺の高密度化に使う比率を設定します。それぞれの辺は多数の等長の辺に分割され、分割辺長の合計長に対する比は与えた比率に近くなります。

単位はジオメトリの空間参照系の単位です。



Note

現在の実装では、離散位置は、頂点のみに対応しています。任意の密度でポイントを使用することができるよう拡張されています。



Note

`densityFrac` に小さい値を指定すると、フレシェ距離の精度が増します。しかし、分割辺数の 2 乗に比例して計算時間とメモリ利用量が增大します。

GEOS モジュールで実現しています。

Availability: 2.4.0 - GEOS $\geq$ 3.7.0 が必要です

例

```
SELECT ST_FrechetDistance('LINESTRING (0 0, 100 0)::geometry, 'LINESTRING (0 0, 50 50, 100 50)::geometry');
```

```
70.7106781186548
```

```
SELECT ST_FrechetDistance('LINESTRING (0 0, 100 0)::geometry, 'LINESTRING (0 0, 50 50, 100 50)::geometry, 0.5);
```

```
50
```

関連情報

[ST_HausdorffDistance](#)

7.12.11 ST_HausdorffDistance

`ST_HausdorffDistance` — 二つのジオメトリ間のハウスドルフ距離を返します。

Synopsis

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

説明

二つのジオメトリ間の **ハウスドルフ距離** を返します。ハウスドルフ距離は二つのジオメトリのどれくらい似ているかを示す尺度です。

この関数は実際に「離散ハウスドルフ距離」を計算します。これはジオメトリの離散ポイントにおけるハウスドルフ距離の計算を行ったものです。*densifyFrac* パラメータが指定された場合には、離散ハウスドルフ距離の計算の前に辺の密度を高くして、精度の高い結果が得られるようにします。個々の辺は、与えた割合に近い辺長で等分されます。

単位はジオメトリの空間参照系の単位です。

**Note**

このアルゴリズムは標準的なハウスドルフ距離と等価では * ありません *。しかし、使用可能な場面の大部分で正しくなる近似計算がなされています。重要なものに、それぞれが概ね平行で概ね等しい長さのラインストリングがあります。これはラインのマッチングに使える基準です。

Availability: 1.5.0

例



2 線間のハウスドルフ距離 (赤) と距離 (黄)

```
SELECT ST_HausdorffDistance(geomA, geomB),
       ST_Distance(geomA, geomB)
FROM (SELECT 'LINESTRING (20 70, 70 60, 110 70, 170 70)'::geometry AS geomA,
            'LINESTRING (20 90, 130 90, 60 100, 190 100)'::geometry AS geomB) AS t;
```

st_hausdorffdistance	st_distance
37.26206567625497	20

Example: 高密度化したうえで求めたハウスドルフ距離。

```
SELECT ST_HausdorffDistance(
  'LINESTRING (130 0, 0 0, 0 150)'::geometry,
  'LINESTRING (10 10, 10 150, 130 10)'::geometry,
  0.5);
```

70

例: 個々の建物について、最もよく表現する区画を見つけます。最初に建物ジオメトリとインタセクトする区画を求めます。DISTINCT ON で、個々の建物が一回ずつしかリストに挙がらないことが保障されます。ORDER BY .. ST_HausdorffDistance で、最も建物に近い区画を選択します。

```
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);
```

関連情報

[ST_FrechetDistance](#)

7.12.12 ST_Length

ST_Length — 線系ジオメトリの 2 次元長を返します。

Synopsis

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

説明

ジオメトリ型: LINESTRING, MULTILINESTRING, ST_Curve, ST_MultiCurve の場合には 2 次元デカルト距離を返します。面ジオメトリでは 0 を返すので [ST_Perimeter](#) を代わりに使います。長さの単位はジオメトリの空間参照系で決まります。

ジオグラフィー型: 測地線の逆解法を用います。周囲長の単位はメートルです。回転楕円体は SRID で指定されますが、SRID が指定されていない場合には WGS84 が使われます。use_spheroid = false の場合には回転楕円体面の代わりに球面で計算します。

現在は、ジオメトリに対しては ST_Length2D の別名ですが、高次元対応に変更されるかも知れません。



Warning

Changed: 2.0.0 大幅な変更 -- 以前の版ではジオグラフィの POLYGON や MULTIPOLYGON への適用によって POLYGON や MULTIPOLYGON の周囲長を返しました。2.0.0 版ではジオメトリの挙動に従うため 0 を返すように変更しました。ポリゴンの周囲長を求める場合は、ST_Perimeter を使います



Note

ジオグラフィでは計算は回転楕円体面モデルを使用します。計算が速い反面精度が低い球面計算を使うには、ST_Length(gg,false) とします。



このメソッドは [OGC Simple Features Implementation Specification for SQL 1.1](#) の実装です。s2.1.5.1



このメソッドは SQL/MM 仕様の実装です。SQL-MM 3: 7.1.2, 9.3.4

Availability: 1.5.0 ジオグラフィ t 対応が導入されました。

ジオメトリの例

ラインストリングのフィート単位の長さを返します。EPSG:2249 はフィート単位のマサチューセッツ州平面なので、フィート単位になることに注意して下さい。

```
SELECT ST_Length(ST_GeomFromText(
'LINESTRING(743238 2967416,743238 2967450,743265 2967450,
743265.625 2967416,743238 2967416)',
2249));
```

```
122.63074400009504
```

This example transforms a WGS 84 LineString to Massachusetts State Plane meters.

```
SELECT ST_Length(ST_Transform(
    ST_GeomFromEWKT('SRID=4326;LINESTRING(-72.1260 42.45, -72.1240 42.45666, ←
    -72.123 42.1546)'),
    26986
));
```

```
34309.4563576191
```

ジオグラフィの例

WGS84 ジオグラフィのラインの長さを返します。

This example uses the default spheroid calculation.

```
SELECT ST_Length(the_geog) As length_spheroid, ST_Length(the_geog,false) As length_sphere
FROM (SELECT ST_GeographyFromText('SRID=4326;LINESTRING(-72.1260 42.45, -72.1240 42.45666, ←
-72.123 42.1546)') As the_geog)
As foo;
```

```
length_spheroid | length_sphere
-----+-----
34310.5703627284 | 34346.20609607416
```

関連情報

[ST_GeographyFromText](#), [ST_GeomFromEWKT](#), [ST_LengthSpheroid](#), [ST_Perimeter](#), [ST_Transform](#)

7.12.13 ST_Length2D

ST_Length2D — LINESTRING または MULTILINESTRING に対して、ジオメトリの 2 次元長を返します。これは **ST_Length** の別名です。

Synopsis

```
float ST_Length2D(geometry a_2dlinestring);
```

説明

LINESTRING または MULTILINESTRING に対して、ジオメトリの 2 次元長を返します。これは ST_Length の別名です。

関連情報

[ST_Length](#), [ST_3DLength](#)

7.12.14 ST_3DLength

ST_3DLength — 線ジオメトリの 3 次元長を返します。

Synopsis

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

説明

LINESTRING または MULTILINESTRING の場合には、そのジオメトリの 3 次元長または 2 次元長を返します。2 次元ラインでは 2 次元長を返します (ST_Length と ST_Length2D と同じです)。



この関数は 3 次元に対応し、Z 値を削除しません。



このメソッドは SQL/MM 仕様の実装です。SQL-MM IEC 13249-3: 7.1, 10.3

Changed: 2.0.0 以前の版では ST_Length3D と呼ばれていました

例

3 次元ケーブルの長さをフィート単位で返します。EPSG:2249 はフィート単位のマサチューセッツ州平面なので、フィート単位になることに注意して下さい。

```
SELECT ST_3DLength(ST_GeomFromText(
'LINESTRING(743238 2967416 1,743238 2967450 1,743265 2967450 3,
743265.625 2967416 3,743238 2967416 3)',
2249));
```

```
122.70471674145682
```

関連情報

[ST_Length](#), [ST_Length2D](#)

7.12.15 ST_LengthSpheroid

ST_LengthSpheroid — 回転楕円体面上の経度緯度のジオメトリの 2 次元または 3 次元の長さ/周長を返します。

Synopsis

```
float ST_LengthSpheroid(geometry a_geometry, spheroid a_spheroid);
```

説明

回転楕円体面のジオメトリの周囲長を計算します。ジオメトリの座標が経度/緯度で、投影せずに周囲長を求めたい場合に使います。回転楕円体面は次のようなテキスト値で指定します:

```
SPHEROID[<NAME
>,<SEMI-MAJOR AXIS
>,<INVERSE FLATTENING
>]
```

例:

```
SPHEROID["GRS_1980",6378137,298.257222101]
```

Availability: 1.2.2

Changed: 2.2.0 これより前の版では、これは `ST_Length_Spheroid` と呼ばれ、`ST_3DLength_Spheroid` という別名を持っていました。



この関数は 3 次元に対応し、Z 値を削除しません。

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
85204.52077117894	13986.87252824332	71217.64824293563

This example uses 3D input.

```
SELECT ST_LengthSpheroid(geom, sph_m ) As tot_len,
ST_LengthSpheroid(ST_GeometryN(geom,1), sph_m) As len_line1,
ST_LengthSpheroid(ST_GeometryN(geom,2), sph_m) As len_line2
FROM (SELECT ST_GeomFromEWKT('MULTILINESTRING((-118.584 38.374 20, -118.583 38.5 30),
(-71.05957 42.3589 75, -71.061 43 90))') As geom,
CAST('SPHEROID["GRS_1980",6378137,298.257222101]' As spheroid) As sph_m) as foo;
```

tot_len	len_line1	len_line2
85204.52592562366	13986.876103023424	71217.64982260024

関連情報

[ST_GeometryN](#), [ST_Length](#)

7.12.16 ST_LongestLine

`ST_LongestLine` — 二つのジオメトリ間の 2 次元最長ラインを返します。

Synopsis

```
geometry ST_LongestLine(geometry g);  
geometry ST_LongestLine(geometry g1, geometry g2);
```

説明

二つのジオメトリのポイントで作られる 2 次元の最長ラインを返します。ラインは `g1` 上から始まり、`g2` 上で終わります。

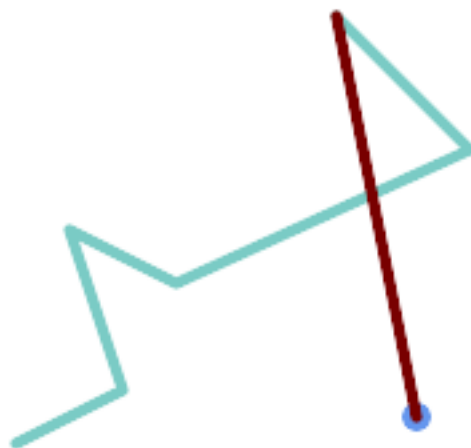
最長ラインは常に二つの頂点となります。一つ以上の最長ラインが発見された場合には、この関数は最初のを返します。ラインの長さは [ST_MaxDistance](#) が返す距離と同じです。

If only one geometry is provided, or `g1` and `g2` are the same geometry, returns the line between the two vertices farthest apart in the geometry. The endpoints of the line lie on the circle computed by [ST_MinimumBoundingCircle](#).

Availability: 1.5.0

Enhanced: 3.7.0 - support for a single geometry input.

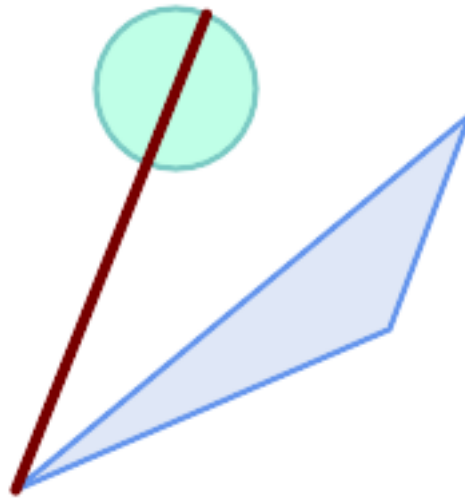
例



ポイントとラインストリングの間の最長となるライン

```
SELECT ST_AsText(ST_LongestLine(  
    'POINT (160 40)',  
    'LINESTRING (10 30, 50 50, 30 110, 70 90, 180 140, 130 190)' )  
    ) AS lline;
```

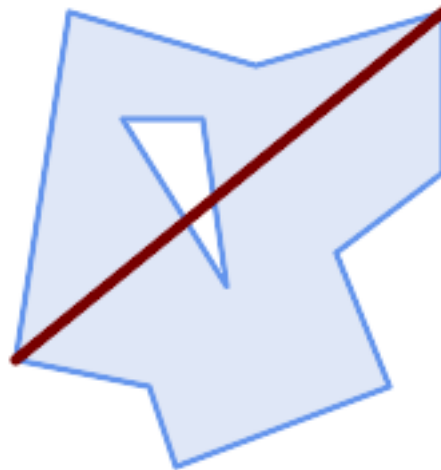
```
LINESTRING(160 40,130 190)
```



二つのポリゴンの間の最長となるライン

```
SELECT ST_AsText(ST_LongestLine(
  'POLYGON ((190 150, 20 10, 160 70, 190 150))',
  ST_Buffer('POINT(80 160)', 30)
)) AS llinewkt;
```

```
LINESTRING(20 10,91.4805029709527 187.7163859753386)
```



一つのジオメトリを横切る最長のラインです。ラインの長さは最大距離と同じです。ラインの端点は最小境界円上に存在します。

```
SELECT ST_AsText(ST_LongestLine(geom)) AS llinewkt,
       ST_MaxDistance(geom) AS max_dist,
       ST_Length(ST_LongestLine(geom)) AS lenll
FROM (SELECT 'POLYGON ((40 180, 110 160, 180 180, 180 120, 140 90, 160 40, 80 10, 70 40, 20 50, 40 180),
       (60 140, 99 77.5, 90 140, 60 140))'::geometry AS geom) AS t;
```

llinewkt		max_dist		lenll
-----+-----				
LINESTRING(20 50,180 180)		206.15528128088303		206.15528128088303

関連情報

[ST_MaxDistance](#), [ST_ShortestLine](#), [ST_3DLongestLine](#), [ST_MinimumBoundingCircle](#)

7.12.17 ST_3DLongestLine

ST_3DLongestLine — 二つのジオメトリ間の 3 次元最長ラインを返します。

Synopsis

geometry **ST_3DLongestLine**(geometry g1, geometry g2);

説明

二つのジオメトリ間の 3 次元最長ラインを返します。該当が複数ある場合には最初の最長ラインを返します。返されるラインは、g1 から始まり g2 で終わります。ラインの 3 次元長は[ST_3DMaxDistance](#)が返す距離と同じです。

Availability: 2.0.0

Changed: 2.2.0 - 二つの 2 次元ジオメトリが入力である場合には、2 次元ポイントが返ります (古い挙動では、存在しない Z の値について 0 を仮定していました)。2 次元と 3 次元の場合には、もはや、存在しない Z の値について 0 を仮定しません。

-  この関数は 3 次元に対応し、Z 値を削除しません。
-  この関数は多面体サーフェスに対応しています。

例

ラインストリングとポイント -- 3 次元と 2 次元の最長ライン

```
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)::' AS
       geometry As line
      ) As foo;
```

lol3d_line_pt		lol2d_line_pt
-----+-----		
LINESTRING(50 75 1000,100 100 30)		LINESTRING(98 190,100 100)

ラインストリングとマルチポイント -- 3次元と2次元の最長ライン

```
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)
```

マルチラインストリングとポリゴンの3次元と2次元の最長ライン

```
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)
```

関連情報

[ST_3DClosestPoint](#), [ST_3DDistance](#), [ST_LongestLine](#), [ST_3DShortestLine](#), [ST_3DMaxDistance](#)

7.12.18 ST_MaxDistance

ST_MaxDistance — 二つのジオメトリ間の2次元最長距離を空間参照系の単位で返します。

Synopsis

```
float ST_MaxDistance(geometry g);
float ST_MaxDistance(geometry g1, geometry g2);
```

説明

二つのジオメトリの2次元最長距離を空間参照系の単位で返します。最長距離は常に二つの頂点間で発生します。これは[ST_LongestLine](#)が返すラインの長さと同じです。

If only one geometry is provided, or g1 and g2 are the same geometry, returns the distance between the two vertices farthest apart in that geometry.

Availability: 1.5.0

Enhanced: 3.7.0 - support for a single geometry input.

例

ポイントとラインの最長距離。

```
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
```

一つのジオメトリ内の頂点間の最長距離。

```
SELECT ST_MaxDistance('POLYGON ((10 10, 10 0, 0 0, 10 10)) '::geometry);
```

```
14.142135623730951
```

関連情報

[ST_Distance](#), [ST_LongestLine](#), [ST_DFullyWithin](#)

7.12.19 ST_3DMaxDistance

ST_3DMaxDistance — 二つのジオメトリ間の 3 次元最大デカルト距離 (空間参照系に基づく) を空間参照系の単位で返します。

Synopsis

```
float ST_3DMaxDistance(geometry g1, geometry g2);
```

説明

二つのジオメトリ間の 3 次元の最大デカルト距離を空間参照系の単位で返します。



この関数は 3 次元に対応し、Z 値を削除しません。



この関数は多面体サーフェスに対応しています。

Availability: 2.0.0

Changed: 2.2.0 - 2 次元と 3 次元の場合には、もはや、存在しない Z の値について 0 を仮定しません。

例

Geometry example - units in meters (SRID: 2163 US National Atlas Equal area) (3D point and line compared 2D point and line) Note that currently no vertical datum support so Z is not transformed and assumed to be same units as final.

```
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

関連情報

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

7.12.20 ST_MinimumClearance

ST_MinimumClearance — ジオメトリのクリアランスの最小値を返します。この値はジオメトリのロバスト性を示すものです。

Synopsis

float **ST_MinimumClearance**(geometry g);

説明

ST_IsValid (ポリゴン) または **ST_IsSimple** (ライン) に従って、ジオメトリは妥当性基準を満たしますが、そのうちの一つの頂点が短い距離だけ移動すると不正になります。これは、テキスト書式 (WKT, KML, GML, GeoJSON 等) に変換する際の精度の損失や、倍精度浮動小数点座標値を使わない書式 (MapInfo TAB 等) で発生する可能性があります。

クリアランスの最小値は、座標精度を変更するためのジオメトリのロバスト性を定量的に計測したものです。ジオメトリの頂点が不正なジオメトリとなることなしに移動できる最大距離です。

ジオメトリがクリアランスの最小値 **e** を持つ場合:

- 距離 **e** より近くなるジオメトリ内の二つの個別の頂点はありません。
- 終端となる辺を除いて、辺との距離が **e** より近い頂点は存在しません。

ジオメトリのクリアランスの最小値が存在しない (単一 POINT、要素が同じ MULTIPOINT 等) 場合には、返り値は **Infinity** になります。

精度の損失による有効性の問題を回避するには、**ST_ReducePrecision** を使うと、ポリゴンジオメトリの妥当性を確実に維持しつつ座標値の精度を減らされます。

Availability: 2.3.0

例

```
SELECT ST_MinimumClearance('POLYGON ((0 0, 1 0, 1 1, 0.5 3.2e-4, 0 0))');
```

```
0.00032
```

関連情報

[ST_MinimumClearanceLine](#), [ST_IsSimple](#), [ST_IsValid](#), [ST_ReducePrecision](#)

7.12.21 ST_MinimumClearanceLine

`ST_MinimumClearanceLine` — ジオメトリの最小クリアランスを示す、2 点のラインストリングを返します。

Synopsis

Geometry **ST_MinimumClearanceLine**(geometry g);

説明

ジオメトリの最小クリアランスを示す、2 点のラインストリングを返します。ジオメトリが最小クリアランスを持たない場合には、`LINESTRING EMPTY` が返ります。

GEOS モジュールで実現しています。

Availability: 2.3.0 - GEOS 3.6.0 以上が必要です。

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)
```

関連情報

[ST_MinimumClearance](#)

7.12.22 ST_Perimeter

`ST_Perimeter` — ポリゴンジオメトリまたはジオグラフィの境界の長さを返します。

Synopsis

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

説明

ジオメトリ/ジオグラフィが `ST_Srrface` または `ST_MultiSurface` (POLYGON または MULTIPOLYGON) の場合に、2 次元周囲長を返します。面ジオメトリでない場合には 0 を返します。ラインストリングについては `ST_Length` を使います。ジオメトリに対しては、周囲長の計測単位は空間参照系によります。

ジオグラフィ型では、測地線の逆解法を用います。この際、周囲長の単位はメートルです。回転楕円体は SRID で指定されますが、SRID が指定されていない場合には WGS84 が使われます。 `use_spheroid = false` の場合には回転楕円体面の代わりに球面で近似します。

現在は、この関数は `ST_Perimeter2D` の別名ですが、高次元対応に変更されるかも知れません。

✔ このメソッドは **OGC Simple Features Implementation Specification for SQL 1.1** の実装です。s2.1.5.1

✔ このメソッドは SQL/MM 仕様の実装です。SQL-MM 3: 8.1.3, 9.5.4

Availability: 2.0.0 ジオグラフィ対応が導入されました。

例: ジオメトリ

POLYGON と MULTIPOLYGON に対するフィート単位の周囲長を返します。EPSG:2249 はフィート単位のマサチューセッツ平面なので、フィート単位です。

```
SELECT ST_Perimeter(ST_GeomFromText(
'POLYGON((743238 2967416,743238 2967450,743265 2967450,
743265.625 2967416,743238 2967416))',
2249));
```

```
122.63074400009504
```

```
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));
```

```
845.227713366825
```

例: ジオグラフィ

ポリゴンとマルチポリゴンのメートル単位とフィート単位の周囲長を返します。ジオグラフィ (WGS 84 経度緯度) であることに注意して下さい。

```
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
-----+-----		
37.37904625414049		122.63466618812497

This example uses a MultiPolygon.

```
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

関連情報

[ST_GeogFromText](#), [ST_GeomFromText](#), [ST_Length](#)

7.12.23 ST_Perimeter2D

ST_Perimeter2D — ポリゴンジオメトリの 2 次元周長を返します。ST_Perimeter の別名です。

Synopsis

```
float ST_Perimeter2D(geometry geomA);
```

説明

ポリゴンジオメトリの 2 次元周長を返します。



Note

これは、現在は ST_Perimeter の別名です。将来的に、ST_Perimeter がジオメトリの最大次元の周囲長を返すようになるかも知れません。これは、まだ考慮中です。

関連情報

[ST_Perimeter](#)

7.12.24 ST_3DPerimeter

ST_3DPerimeter — ポリゴンジオメトリの 3 次元周長を返します。

Synopsis

float **ST_3DPerimeter**(geometry geomA);

説明

POLYGON または MULTIPOLYGON ジオメトリの場合には、3 次元周囲長を返します。ジオメトリが 2 次元の場合には、2 次元周囲長を返します。



この関数は 3 次元に対応し、Z 値を削除しません。



このメソッドは SQL/MM 仕様の実装です。SQL-MM ISO/IEC 13249-3: 8.1, 10.5

Changed: 2.0.0 以前の版では ST_Perimeter3D と呼ばれていました。

例

フィート単位のマサチューセッツ州平面での、大気中にある、わずかに持ち上げられたポリゴンの周囲長です。

```
SELECT ST_3DPerimeter(geom), ST_Perimeter2D(geom), ST_Perimeter(geom) FROM
      (SELECT ST_GeomFromEWKT('SRID=2249;POLYGON((743238 2967416 2,743238 ↵
                2967450 1,
743265.625 2967416 1,743238 2967416 2))') As geom) As foo;
```

ST_3DPerimeter	st_perimeter2d	st_perimeter
105.46579359767418	105.43299727218763	105.43299727218763

関連情報

[ST_GeomFromEWKT](#), [ST_Perimeter](#), [ST_Perimeter2D](#)

7.12.25 ST_ShortestLine

ST_ShortestLine — 二つのジオメトリの 3 次元の最短ラインを返します。

Synopsis

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

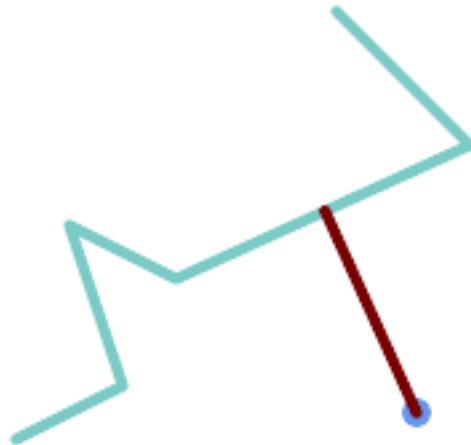
説明

二つのジオメトリの間の 2 次元の最短ラインを返します。返されるラインは geom1 内から始まり、geom2 内で終わります。geom1 と geom2 がインタセクトしている場合には、結果はインタセクションするポイントを始点と終点にしたラインになります。ラインの長さは [ST_Distance](#) で g1 と g2 を引数に取った時と同じになります。

Enhanced: 3.4.0 - ジオグラフィに対応しました。

Availability: 1.5.0

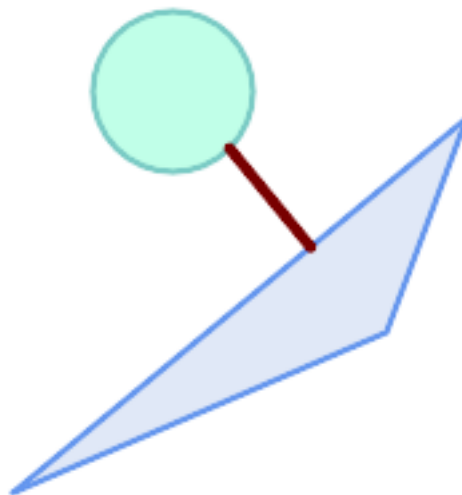
例



ポイントとラインストリングの最短ライン

```
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)
```



ポリゴン間の最短ライン

```
SELECT ST_AsText(ST_ShortestLine(
  'POLYGON ((190 150, 20 10, 160 70, 190 150))',
  ST_Buffer('POINT(80 160)', 30)
) ) AS llinewkt;
```

```
LINESTRING(131.59149149528952 101.89887534906195,101.21320343559643 138.78679656440357)
```

関連情報

[ST_ClosestPoint](#), [ST_Distance](#), [ST_LongestLine](#), [ST_MaxDistance](#)

7.12.26 ST_3DShortestLine

ST_3DShortestLine — 二つのジオメトリの 3 次元の最短ラインを返します。

Synopsis

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

説明

二つのジオメトリの 3 次元長が最短となるラインを返します。見つかった最短線が複数ある場合は、最初のみ返します。**g1** と **g2** が 1 点のみでインタセクトする場合は、インタセクション点を開始点と終了点とします。**g1** と **g2** が 1 点より多くインタセクトする場合は、同じ点を開始点と終了点としますが、その点は不定です。返されるラインは常に **g1** 側を始点、**g2** 側を終点とします。この関数が返すラインの長さは、常に **g1** と **g2** を指定した場合の [ST_3DDistance](#) が返す値と同じになります。

Availability: 2.0.0

Changed: 2.2.0 - 二つの 2 次元ジオメトリが入力である場合には、2 次元ポイントが返ります (古い挙動では、存在しない Z の値について 0 を仮定していました)。2 次元と 3 次元の場合には、もはや、存在しない Z の値について 0 を仮定しません。

-  この関数は 3 次元に対応し、Z 値を削除しません。
-  この関数は多面体サーフェスに対応しています。

例

ラインストリングとポイント -- 3 次元と 2 次元の最短ライン

```
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) ' '::
       geometry As line
       ) As foo;
```

shl3d_line_pt	shl2d_line_pt
LINESTRING(54.69937988676193 128.93502291722837 11.547586950660556,100 100 30)	LINESTRING(73.07692307692307 115.38461538461539,100 100)

ラインストリングとマルチポイント -- 3次元と2次元の最短ライン

```
SELECT ST_AsEWKT(ST_3DShortestLine(line,pt)) AS shl3d_line_pt,
       ST_AsEWKT(ST_ShortestLine(line,pt)) As shl2d_line_pt
FROM (SELECT 'MULTIPOINT(100 100 30, 50 74 1000) '::geometry As pt,
            'LINESTRING (20 80 20, 98 190 1, 110 180 3, 50 75 900) '::
            geometry As line
      ) As foo;
```

shl3d_line_pt		shl2d_line_pt
-----+-----		
LINESTRING(54.69937988676193 128.93502291722837 11.547586950660556,100 100 30)		←
LINESTRING(50 75 50 74)		←

マルチラインストリングとポリゴンの3次元と2次元の最短ライン

```
SELECT ST_AsEWKT(ST_3DShortestLine(poly, mline)) As shl3d,
       ST_AsEWKT(ST_ShortestLine(poly, mline)) As shl2d
FROM (SELECT ST_GeomFromEWKT('POLYGON((175 150 5, 20 40 5, 35 45 5, 50 60 5,
100 100 5, 175 150 5))') As poly,
       ST_GeomFromEWKT('MULTILINESTRING((175 155 2, 20 40 20, 50 60 -2, 125
100 1, 175 155 1),
       (1 10 2, 5 20 1))') As mline ) As foo;
```

shl3d		←
shl2d		←

LINESTRING(40.45454545454545 53.63636363636364 5,40.45454545454545 53.63636363636364		←
5) LINESTRING(20 40,20 40)		←

関連情報

[ST_3DClosestPoint](#), [ST_3DDistance](#), [ST_LongestLine](#), [ST_ShortestLine](#), [ST_3DMaxDistance](#)

7.13 重ね合わせ関数

7.13.1 ST_ClipByBox2D

ST_ClipByBox2D — 長方形内に落ちるジオメトリの一部を返します。

Synopsis

geometry **ST_ClipByBox2D**(geometry geom, box2d box);

説明

2次元ボックスでジオメトリを切り抜きます。高速ですが不正な方法になることもあります。トポロジ的に不正な入力ジオメトリでは例外が投げられ結果を返しません。出力ジオメトリの妥当性は保証されません (ポリゴンで自己インタセクションができるかも知れません)。

GEOS モジュールで実現しています。

Availability: 2.2.0

例

This example relies on the implicit cast from geometry to box2d for the second parameter.

```
SELECT ST_ClipByBox2D(geom, ST_MakeEnvelope(0,0,10,10)) FROM mytab;
```

関連情報

[ST_Intersection](#), [ST_MakeBox2D](#), [ST_MakeEnvelope](#)

7.13.2 ST_Difference

ST_Difference — ジオメトリ B とインタセクトしていないジオメトリ A の一部を表現するジオメトリを計算します。

Synopsis

geometry **ST_Difference**(geometry geomA, geometry geomB, float8 gridSize = -1);

説明

ジオメトリ B とインタセクトしていないジオメトリ A の一部を表現するジオメトリを計算します。これは **A - ST_Intersection(A,B)** と同じです。A が完全に B に包含されている場合には、適切なタイプの空のジオメトリが返されます。



Note

これは、重ね合わせ関数では唯一の入力順序を気にしなければならない関数です。ST_Difference(A, B) は常に A の一部を返します。

If the optional `gridSize` parameter is given (GEOS-3.9.0 or higher required), all result vertices are guaranteed to fall on a snap-rounded grid of the specified size. Note that operations performed on a grid may contain small artifacts produced during grid alignment, see [ST_ReducePrecision](#).

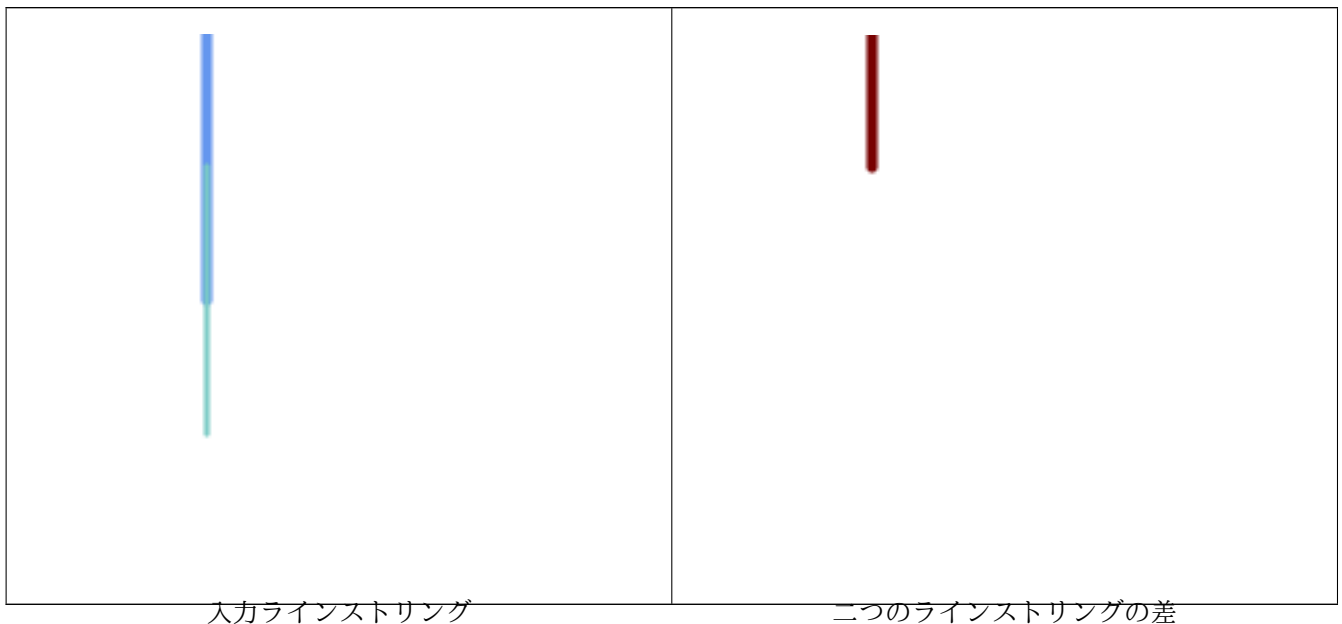
GEOS モジュールで実現しています。

Enhanced: 3.1.0 `gridSize` パラメータを受け付けるようになりました。

`gridSize` パラメータを使うには GEOS 3.9.0 以上が必要です。

- ✔ このメソッドは [OGC Simple Features Implementation Specification for SQL 1.1](#) の実装です。s2.1.1.3
- ✔ このメソッドは SQL/MM 仕様の実装です。SQL-MM 3: 5.1.20
- ✔ この関数は 3 次元に対応し、Z 値を削除しません。ただし、結果は XY のみを使用して計算されます。結果の Z 値は複写するか、平均値になるか、補間されます。

例



次では、2次元ラインストリングの差を求めます。

```
SELECT ST_AsText(ST_Difference('LINESTRING(50 100, 50 200)::geometry,
                                'LINESTRING(50 50, 50 150)::geometry'
                                ));
```

```
LINESTRING(50 150,50 200)
```

次では、3次元ラインストリングの差を求めます。

```
SELECT ST_AsEWKT(ST_Difference('MULTIPOINT(-118.58 38.38 5,-118.60 38.329 6,-118.614 38.281 ←
                                7)' :: geometry,
                                'POINT(-118.614 38.281 5)' :: geometry
                                ));
```

```
MULTIPOINT(-118.6 38.329 6,-118.58 38.38 5)
```

関連情報

[ST_SymDifference](#), [ST_Intersection](#), [ST_Union](#), [ST_ReducePrecision](#)

7.13.3 ST_Intersection

ST_Intersection — ジオメトリ A とジオメトリ B の共通部分を表現するジオメトリを返します。

Synopsis

```
geometry ST_Intersection( geometry geomA , geometry geomB , float8 gridSize = -1 );
geography ST_Intersection( geography geogA , geography geogB );
```


説明

二つのジオメトリのポイント集合の交差を表現するジオメトリを返します。言い換えると、ジオメトリ A とジオメトリ B の一部であって、二つのジオメトリで共有される部分を返します。

ジオメトリが共通するポイントを持っていない (つまり接続されていない) 場合には、適切なタイプの空のジオメトリが返されます。

If the optional `gridSize` parameter is given (GEOS-3.9.0 or higher required), all result vertices are guaranteed to fall on a snap-rounded grid of the specified size. Note that operations performed on a grid may contain small artifacts produced during grid alignment, see [ST_ReducePrecision](#).

ST_Intersects と併用する **ST_Intersection** は、バウンディングボックス、バッファや、対象の国や領域の内部にあるジオメトリの部分のみが必要な場合に使う領域クエリといったようなジオメトリの切り抜きに使えます。

Note



この関数のジオグラフィ版はジオメトリ実装にかぶせた薄いラッパーです。最初に二つのジオグラフィ値のバウンディングボックスに適合する最適な SRID を決定します (二つのジオグラフィ値が UTM ゾーンの半分以上だけ同じ UTM ゾーンでない場合はどちらか一つを選びます) (UTM またはランベルト正積方位図法の北極又は南極を優先し、最悪の場合にメルカトルに後退します)。次に、最適な平面空間参照系でインタセクトする領域を計算して、WGS84 ジオグラフィに再変換します。



Warning

この関数は M 値が存在している場合には削除します。



Warning

3 次元ジオメトリで動作しますが、SFCGAL ベースの **ST_3DIntersection** は、3 次元ジオメトリの確実な 3 次元インタセクトした領域を返すので、これを使った方がいいかも知れません。この関数は Z 値があっても動作しますが、Z 値は平均化されます。

GEOS モジュールで実現しています。

Enhanced: 3.1.0 `gridSize` パラメータを受け付けるようになりました

`gridSize` パラメータを使うには GEOS 3.9.0 以上が必要です

Changed: 3.0.0 SFCGAL 非依存になりました。

Availability: 1.5 ジオグラフィ型が導入されました。



このメソッドは **OGC Simple Features Implementation Specification for SQL 1.1** の実装です。s2.1.1.3



このメソッドは SQL/MM 仕様の実装です。SQL-MM 3: 5.1.18



この関数は 3 次元に対応し、Z 値を削除しません。ただし、結果は XY のみを使用して計算されます。結果の Z 値は複写するか、平均値になるか、補間されます。

例

```
SELECT ST_AsText(ST_Intersection('POINT(0 0)::geometry', 'LINESTRING ( 2 0, 0 2 ) '::
  geometry));
```

POINT EMPTY

```
SELECT ST_AsText(ST_Intersection('POINT(0 0)::geometry, 'LINESTRING ( 0 0, 0 2 ) ':: geometry));
```

POINT(0 0)

国別に全てのライン (trains) を切り抜きます。国のジオメトリは POLYGON または MULTIPOLYGON であると仮定します。ご注意: ポイントだけを共有する trails は気にしないので、LINESTRING または MULTILINESTRING の結果となるインタセクションだけを保持しています。ジオメトリコレクションを個々のマルチ系ジオメトリの要素に分解するためにダンプが必要です。下の例は非常に汎用的で、WHERE 節を変更するだけでポリゴンでも動作します。

```
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;
```

ランドマーク等のポリゴンに対しては、ポリゴンを除いたジオメトリを 0.0 でバッファを実行すると空ジオメトリコレクションが得られる、という速度向上のための技を使うことができます (それで、ポリゴン、ラインストリング、ポイントを含むジオメトリコレクションを 0.0 でバッファを実行すると、ポリゴンのみが残る、ジオメトリコレクションでなくなります)。

```
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));
```

例: 2.5 次元ジオメトリー

本当の 3 次元で見てインターセクトした箇所ではないことに注意して下さい。2.5 次元ジオメトリーを使っています (Z 値を持つジオメトリですが計算は 2 次元で行います)。ST_3DIntersection を使った同じ例と比較して下さい。

```
select ST_AsText(ST_Intersection(linestring, polygon)) As wkt
from ST_GeomFromText('LINESTRING Z (2 2 6,1.5 1.5 7,1 1 8,0.5 0.5 8,0 0 10)') AS linestring
CROSS JOIN ST_GeomFromText('POLYGON((0 0 8, 0 1 8, 1 1 8, 1 0 8, 0 0 8))') AS polygon;
```

```
LINESTRING Z (1 1 8,0.5 0.5 8,0 0 10)
```

関連情報

ST_3DIntersection, ST_Difference, ST_Union, ST_ClipByBox2D, ST_Dimension, ST_Dump, ST_Force2D, ST_SymDifference, ST_Intersects, ST_Multi, ST_ReducePrecision

7.13.4 ST_MemUnion

ST_MemUnion — ジオメトリを結合する集約関数で、メモリを効率的に使用しますが処理時間のかかるものです。

Synopsis

geometry **ST_MemUnion**(geometry set geomfield);

説明

入力ジオメトリを結合してオーバーラップをしない結果ジオメトリを生成する集約関数です。出力は単一ジオメトリ、マルチ系ジオメトリ、ジオメトリコレクションのいずれにかなる可能性があります。



Note

ST_Unionと同じ結果を生成しますが、メモリ使用が少なく、処理時間が長くなります。この集約関数はジオメトリの逐次加算的結合で動作しています。これは、最初に配列を蓄積して高速アルゴリズムで結合する ST_Union 集約関数と反対です。



この関数は 3 次元に対応し、Z 値を削除しません。ただし、結果は XY のみを使用して計算されます。結果の Z 値は複写するか、平均値になるか、補間されます。

例

```
SELECT id,  
       ST_MemUnion(geom) as singlegeom  
FROM sometable f  
GROUP BY id;
```

関連情報

[ST_Union](#)

7.13.5 ST_Node

ST_Node — ラインストリングの集合にノードを作成します。

Synopsis

geometry **ST_Node**(geometry geom);

説明

ラインストリングのコレクションに完全にノードが加えられたものを表現する (マルチ) ラインストリングを返します。全ての入力ノードを保存したうえで、可能な最低の数だけノードを加えています。結果の線はディゾルブされています (重複線が削除されます)。

これは **ST_Polygonize** への入力に適した、完全にノードを加えた線を生成する良い方法です。

ST_UnaryUnion も、ラインのノード追加とディゾルブに使うことができます。 **gridSize** を指定する任意引数を与えると、より単純かつよりロバストな出力が得られます。集約関数版については **ST_Union** をご覧ください。



この関数は 3 次元に対応し、Z 値を削除しません。

GEOS モジュールで実現しています。

Availability: 2.0.0

Changed: 2.4.0 この関数は内部で **GEOSUnaryUnion** の代わりに **GEOSNode** を使用しています。ラインストリングの並び順と方向が **PostGIS 2.4** より前のものと違うことになるかも知れません。

例

自己交差する 3 次元ラインストリングにノードを加えます。

```
SELECT ST_AsText(ST_Node('LINESTRINGZ(0 0 0, 10 10 10, 0 10 5, 10 0 3)')::geometry)
      ) As  output;
```

```
MULTILINESTRING Z ((0 0 0,5 5 4.5),(5 5 4.5,10 10 10,0 10 5,5 5 4.5),(5 5 4.5,10 0 3))
```

線を共有する二つのラインストリングにノードを加えます。結果の線はディゾルブされます。

```
SELECT ST_AsText(ST_Node('MULTILINESTRING ((2 5, 2 1, 7 1), (6 1, 4 1, 2 3, 2 5))')::
      geometry)
      ) As  output;
```

```
MULTILINESTRING((2 5,2 3),(2 3,2 1,4 1),(4 1,2 3),(4 1,6 1),(6 1,7 1))
```

関連情報

ST_UnaryUnion, **ST_Union**

7.13.6 ST_Split

ST_Split — ジオメトリを他のジオメトリで分割してできたジオメトリのコレクションを返します。

Synopsis

geometry **ST_Split**(geometry input, geometry blade);

説明

この関数は (MULTI)POINT、(MULTI)LINESTRING、(MULTI)POLYGON の境界による LINESTRING の分割と、LINESTRING による (MULTI)POLYGON の分割に対応します。(MULTI)POLYGON を刃として使うとき、線要素 (境界) は入力に分割に使われます。結果ジオメトリは常にコレクションです。

この関数は **ST_Union** の逆のようなものです。返されるコレクションに **ST_Union** を与えると、理論的には元のジオメトリが得られます (丸め誤差のために確実に同じになるものではありません)。



Note

入力と刃が数値の精度の問題でインタセクトしない場合には、入力が思ったように分割されない可能性があります。この状況を避けるには、先に、**ST_Snap** に小さい許容値を与えて、入力を刃にスナップする必要がありますかも知れません。

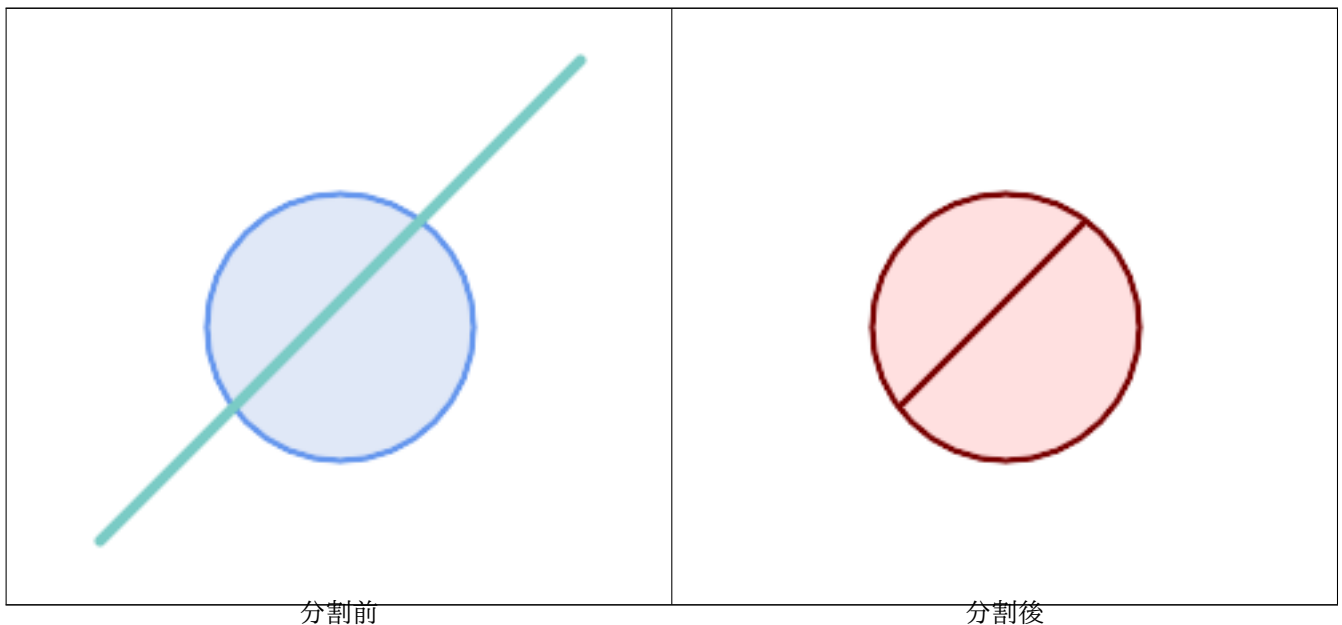
Availability: 2.0.0 GEOS が必要です

Enhanced: 2.2.0 ライン分割をマルチライン、マルチポイントまたはポリゴンもしくはマルチポリゴンの境界で行えるようにしました。

Enhanced: 2.5.0 マルチラインによるポリゴンの分割に対応するようになりました。

例

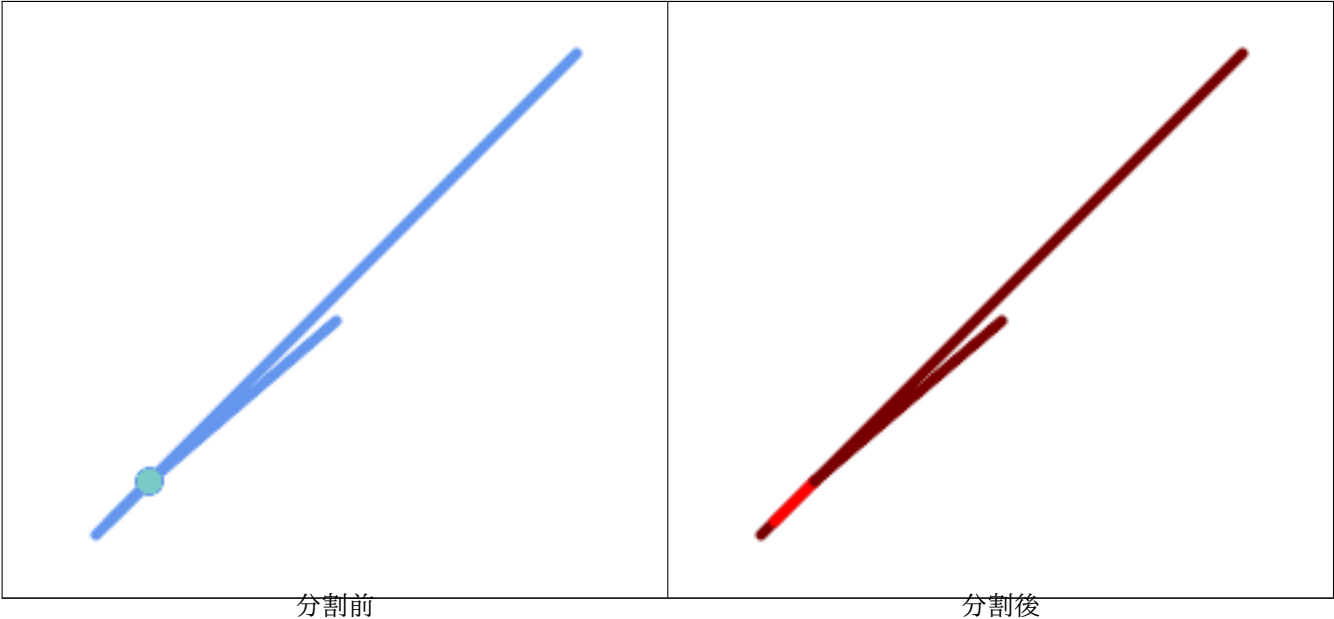
ラインでポリゴンを分割します。



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

```
GEOMETRYCOLLECTION(
  POLYGON((150 90,149.039264020162 80.2454838991936,146.193976625564 70.8658283817455,...),
  POLYGON(...))
)
```

マルチラインストリングをポイントで分割します。ポイントは確実にラインストリングの要素の上に存在するようにします。



```
SELECT ST_AsText(ST_Split('MULTILINESTRING((10 10, 190 190), (15 15, 30 30, 100 90))',
    ST_Point(30,30)));
```

```
GEOMETRYCOLLECTION(
  LINESTRING(10 10,30 30),
  LINESTRING(30 30,190 190),
  LINESTRING(15 15,30 30),
  LINESTRING(30 30,100 90)
)
```

ラインストリングをライン上にないポイントで分割します。ST_Snapを使ってラインをポイントにスナップして分割できるようにしています。

```
WITH data AS (SELECT
  'LINESTRING(0 0, 100 100)>:::geometry AS line,
  'POINT(51 50)>::: geometry AS point
)
SELECT ST_AsText(ST_Split(ST_Snap(line, point, 1), point)) AS snapped_split,
  ST_AsText(ST_Split(line, point)) AS not_snapped_not_split
FROM data;
```

snapped_split		not_snapped_not_split
-----+-----		
GEOMETRYCOLLECTION(LINESTRING(0 0,51 50),LINESTRING(51 50,100 100)) GEOMETRYCOLLECTION(↵ LINESTRING(0 0,100 100))		

関連情報

ST_Snap, ST_Union

7.13.7 ST_Subdivide

ST_Subdivide — ジオメトリの線の分割を計算します。

Synopsis

```
setof geometry ST_Subdivide(geometry geom, integer max_vertices=256, float8 gridSize = -1);
```

説明

geom を直線で分割したジオメトリの集合を返します。個々の部分は **max_vertices** 以下の頂点になります。

閉じたボックスを表現するために必要なポイント数は 5 個ですから、**max_vertices** は 5 以上です。

If the optional **gridSize** parameter is given (GEOS-3.9.0 or higher required), all result vertices are guaranteed to fall on a snap-rounded grid of the specified size. Note that operations performed on a grid may contain small artifacts produced during grid alignment, see [ST_ReducePrecision](#).

ポリゴン内のポイントと他の空間演算は、通常ではインデックスが付いた細分化されたデータセットの方が速くなります。通常は分割したジオメトリのバウンディングボックスは元ジオメトリのものよりも小さい領域になるので、インデックスのクエリで「当たり」となる場合が少なくなります。インデックスの再チェックによって実行される空間演算で処理されるポイント数が少なくなるので、「当たり」の場合は速度が速くなります。

Note



分割したジオメトリをジオグラフィーにキャストすると、得られるジオグラフィーは元のものとは異なることになる場合があります。分割によって平面空間 (ジオメトリ) に頂点が追加されます。頂点が境界に沿って挿入される場合には、ジオグラフィー表現のエッジは測地線上の辺として補間されます。歪みを最小化するには、最初に [ST_Segmentize](#) を使って測地線上に頂点を追加してジオグラフィーを高密度化して、それから分割までの間にジオメトリにキャストします。



Note

これは[集合を返す関数](#) (SRF) です。単一ジオメトリ値を含む行の集合を返します。SELECT リストまたは FROM 節で、結果ジオメトリごとに一つのレコードとなる結果集合を生成できます。

GEOS モジュールで実現しています。

Availability: 2.2.0

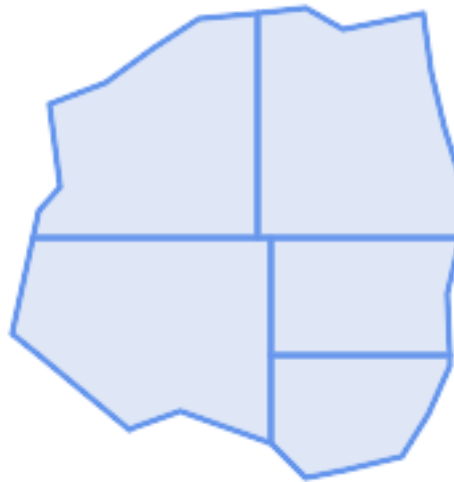
Enhanced: 2.5.0 ポリゴン分割で存在するポイントを再利用して頂点数の最小値を 8 から 5 に変更。

Enhanced: 3.1.0 **gridSize** パラメータを受け付けるようになりました。

gridSize パラメータを使うには GEOS 3.9.0 以上が必要です

例

例: 個々のポリゴンが 10 頂点とならないようなポリゴンの分割と、個々に一意の ID の割り当て。

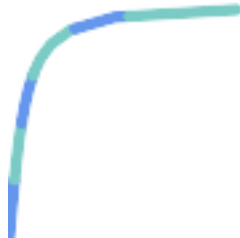


最大 10 頂点での分割

```
SELECT row_number() OVER() As rn, ST_AsText(geom) As wkt
FROM (SELECT ST_Subdivide('POLYGON((132 10,119 23,85 35,68 29,66 28,49 42,32 56,22
64,32 110,40 119,36 150,
57 158,75 171,92 182,114 184,132 186,146 178,176 184,179 162,184 141,190 122,
190 100,185 79,186 56,186 52,178 34,168 18,147 13,132 10))'::geometry,10)) AS f(
geom);
```

rn	b'' b''	wkt
1	b'' b''	POLYGON((85 35,68 29,66 28,32 56,22 64,29.82608695652174 100,119 100,119 23,85 35))
2	b'' b''	POLYGON((186 52,178 34,168 18,147 13,132 10,119 23,119 56,186 56,186 52))
3	b'' b''	POLYGON((185 79,186 56,119 56,119 100,190 100,185 79))
4	b'' b''	POLYGON((40 119,36 150,57 158,75 171,92 182,114 184,114 100,29.82608695652174 100,32 110,40 119))
5	b'' b''	POLYGON((132 186,146 178,176 184,179 162,184 141,190 122,190 100,114 100,114 184,132 186))

例: ST_Segmentize(geography, distance) を使って長いジオグラフィ型ラインの高密度化を行い、ST_Subdivide を使って結果ラインを 8 頂点のラインに分割します。高密度化することで、分割時にジオメトリのジオグラフィ表現への変更によって与えられる影響が最小化されます。



ラインの緻密化と分割。

```
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)
```

例: テーブルの適切な位置での複雑なジオメトリの分割。元のジオメトリのレコードはソーステーブルから削除され、分割された結果ジオメトリが代わりの新しいレコードになります。

```
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;
```

例: 分割されたジオメトリを持つ新しいテーブルを作ります。元のジオメトリのキーを保持して、新しいテーブルは元のテーブルと結合できます。ST_Subdivide は集合を返す関数で、行は単一の値を持つものですので、この構文で、結果の部分ごとに一つの行となるテーブルが自動的に生成されます。

```
CREATE TABLE subdivided_geoms AS
  SELECT pkey, ST_Subdivide(geom) AS geom
  FROM original_geoms;
```

関連情報

[ST_ClipByBox2D](#), [ST_Segmentize](#), [ST_Split](#), [ST_NPoints](#), [ST_ReducePrecision](#)

7.13.8 ST_SymDifference

`ST_SymDifference` — ジオメトリ A とジオメトリ B がインタセクトしていない部分を表現するジオメトリを返します。

Synopsis

```
geometry ST_SymDifference(geometry geomA, geometry geomB, float8 gridSize = -1);
```

説明

ジオメトリ A とジオメトリ B がインタセクトしていない部分を表現するジオメトリを返します。この関数は、`ST_Union(A,B) - ST_Intersection(A,B)` と同じです。SymDifference (対象差) と呼ばれるのは `ST_Union(A,B) - ST_Intersection(A,B)` だからです。

If the optional `gridSize` parameter is given (GEOS-3.9.0 or higher required), all result vertices are guaranteed to fall on a snap-rounded grid of the specified size. Note that operations performed on a grid may contain small artifacts produced during grid alignment, see [ST_ReducePrecision](#).

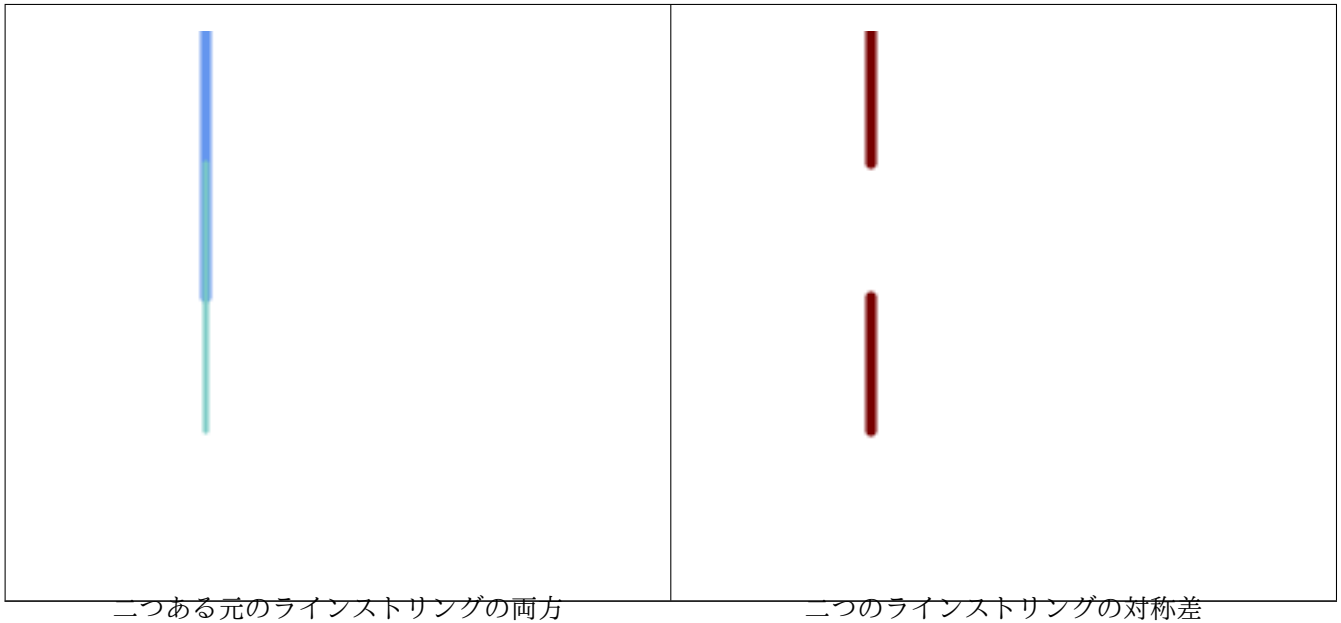
GEOS モジュールで実現しています。

Enhanced: 3.1.0 `gridSize` パラメータを受け付けるようになりました。

`gridSize` パラメータを使うには GEOS 3.9.0 以上が必要です

-  このメソッドは [OGC Simple Features Implementation Specification for SQL 1.1](#) の実装です。s2.1.1.3
-  このメソッドは SQL/MM 仕様の実装です。SQL-MM 3: 5.1.21
-  この関数は 3 次元に対応し、Z 値を削除しません。ただし、結果は XY のみを使用して計算されます。結果の Z 値は複写するか、平均値になるか、補間されます。

例



This example shows a 2D-safe symmetric difference of two LineStrings.

```
SELECT ST_AsText(ST_SymDifference(ST_GeomFromText('LINESTRING(50 100, 50 200)'),
    ST_GeomFromText('LINESTRING(50 50, 50 150)')
  )
);
```

```
MULTILINESTRING((50 150,50 200),(50 50,50 100))
```

```
--When used in 3d doesn't quite do the right thing
SELECT ST_AsEWKT(ST_SymDifference(ST_GeomFromEWKT('LINESTRING(1 2 1, 1 4 2)'),
    ST_GeomFromEWKT('LINESTRING(1 1 3, 1 3 4)')))
```

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

関連情報

[ST_Difference](#), [ST_Intersection](#), [ST_Union](#), [ST_ReducePrecision](#)

7.13.9 ST_UnaryUnion

ST_UnaryUnion — 単一のジオメトリの要素の結合を計算します。

Synopsis

geometry **ST_UnaryUnion**(geometry geom, float8 gridSize = -1);

説明

単一入力の形式の **ST_Union** です。入力は単一ジオメトリ、マルチ系ジオメトリ、ジオメトリコレクションが可能です。結合は、入力の個々のエレメントに適用されます。

この関数は要素間のオーバーラップが原因で不正とされるマルチポリゴンを修正するのに使えます。しかしながら、入力の要素は妥当でなければなりません。蝶ネクタイポリゴンのような入力の個々の要素はエラーを引き起こすかも知れません。このため **ST_MakeValid** を使う方が良いかも知れません。

この関数は、他に、クロスやオーバーラップするラインストリングのコレクションを **単純** にするためのノード追加とディソルブに使われます (**ST_Node** も同様のことを行いますが、任意引数 `gridSize` を与えることができません。)

一度に結合するジオメトリの数量を調整するために `ST_UnaryUnion` と **ST_Collect** とを混ぜることができます。メモリ使用と処理時間のトレードオフが可能になり、`ST_Union` と **ST_MemUnion** とのバランスが得られます。

If the optional `gridSize` parameter is given (GEOS-3.9.0 or higher required), all result vertices are guaranteed to fall on a snap-rounded grid of the specified size. Note that operations performed on a grid may contain small artifacts produced during grid alignment, see **ST_ReducePrecision**.

 この関数は 3 次元に対応し、Z 値を削除しません。ただし、結果は XY のみを使用して計算されます。結果の Z 値は複写するか、平均値になるか、補間されます。

Enhanced: 3.1.0 `gridSize` パラメータを受け付けるようになりました。

`gridSize` パラメータを使うには GEOS 3.9.0 以上が必要です

Availability: 2.0.0

関連情報

ST_Union, **ST_MemUnion**, **ST_MakeValid**, **ST_Collect**, **ST_Node**, **ST_ReducePrecision**

7.13.10 ST_Union

ST_Union — 入力ジオメトリのポイント集合の結合を表現するジオメトリを返します。

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);
```

説明

入力ジオメトリを結合してオーバーラップをしない結果ジオメトリを生成します。出力は単一ジオメトリ、マルチ系ジオメトリ、ジオメトリコレクションのいずれにかなる可能性があります。次の通り複数の形式があります。

二入力形式: 二つの入力ジオメトリの結合のジオメトリを返します。いずれかの入力が `NULL` の場合には `NULL` を返します。

配列形式: ジオメトリ配列を結合したジオメトリを返します。

集約関数形式: ジオメトリの集合を結合したジオメトリを返します。`ST_Union()` 関数は、PostgreSQL 用語で言うところの「集約関数」です。つまり、`SUM()` や `MEAN()` と同じ方法で複数のデータ行の操作を行い、他の集約関数と同じように `NULL` ジオメトリを無視します。

非集約関数で入力の一つの形式である **ST_UnaryUnion** も参照して下さい。

ST_Union の配列形式と集合形式では、<http://blog.cleverelephant.ca/2009/01/must-faster-unions-in-postgis-14.html> で紹介されている早いカスケード結合アルゴリズムを使っています。

If the optional `gridSize` parameter is given (GEOS-3.9.0 or higher required), all result vertices are guaranteed to fall on a snap-rounded grid of the specified size. Note that operations performed on a grid may contain small artifacts produced during grid alignment, see **ST_ReducePrecision**.

**Note**

ST_Collect は、結果に重ね合わせが無いことを求めている場合には、時々 **ST_Union** の代わりに用いられます。**ST_Collect** は、集められたジオメトリに処理を実行しないため、通常は **ST_Union** より早い動作速度になります。

GEOS モジュールで実現しています。

ST_Union は MULTILINESTRING を生成し、新しい LINESTRING を単一の LINESTRING に縫い付けません。LINESTRING に縫い付けるには **ST_LineMerge** を使用します。

ご注意: この関数は以前は、“Union” から名称変更して `GeomUnion()` と呼ばれていました。UNION は SQL の予約語であるためです。

Enhanced: 3.1.0 `gridSize` パラメータを受け付けるようになりました。

`gridSize` パラメータを使うには GEOS 3.9.0 以上が必要です

Changed: 3.0.0 SFCGAL 非依存になりました。

Availability: 1.4.0 - **ST_Union** が機能強化されました。**ST_Union(geomarray)** が導入され、PostgreSQL の高速なコレクションの集約が導入されました。



このメソッドは **OGC Simple Features Implementation Specification for SQL 1.1** の実装です。s2.1.1.3

**Note**

集約関数版は、OGC 仕様に明示的に定義されていません。



このメソッドは SQL/MM 仕様の実装です。SQL-MM 3: 5.1.19 ポリゴンが含まれる時、Z 値 (標高) を持ちます。



この関数は 3 次元に対応し、Z 値を削除しません。ただし、結果は XY のみを使用して計算されます。結果の Z 値は複写するか、平均値になるか、補間されます。

例

集約関数版の例

```
SELECT id,
       ST_Union(geom) as singlegeom
FROM sometable f
GROUP BY id;
```

非集約関数の例

```
SELECT ST_AsText(ST_Union('POINT(1 2)::geometry', 'POINT(-2 3)::geometry'));
```

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

```
SELECT ST_AsText(ST_Union('POINT(1 2)::geometry, 'POINT(1 2)::geometry));
```

```
POINT(1 2)
```

3D example - sort of supports 3D (and with mixed geometry types!)

```
SELECT ST_AsEWKT(ST_Union(geom))
FROM (
  SELECT 'POLYGON Z((-7 4.2 5, -7.1 4.2 5, -7.1 4.3 5, -7 4.2 5))'::geometry geom
  UNION ALL
  SELECT 'POINT Z(5 5 5)'::geometry geom
  UNION ALL
  SELECT 'POINT Z(-2 3 1)'::geometry geom
  UNION ALL
  SELECT 'LINESTRING Z(5 5 5, 10 10 10)'::geometry geom
) AS foo;
```

```
GEOMETRYCOLLECTION(POINT(-2 3 1),LINESTRING(5 5 5,10 10 10),POLYGON((-7 4.2 5,-7.1 4.2 5, ←
  5,-7.1 4.3 5,-7 4.2 5)))
```

This 3D example does not mix dimensions.

```
SELECT ST_AsEWKT(ST_Union(geom))
FROM (
  SELECT 'POLYGON((-7 4.2 2, -7.1 4.2 3, -7.1 4.3 2, -7 4.2 2))'::geometry geom
  UNION ALL
  SELECT 'POINT(5 5 5)'::geometry geom
  UNION ALL
  SELECT 'POINT(-2 3 1)'::geometry geom
  UNION ALL
  SELECT 'LINESTRING(5 5 5, 10 10 10)'::geometry geom
) AS foo;
```

```
GEOMETRYCOLLECTION(POINT(-2 3 1),LINESTRING(5 5 5,10 10 10),POLYGON((-7 4.2 2,-7.1 4.2 3, ←
  3,-7.1 4.3 2,-7 4.2 2)))
```

This example unions an array built from a query.

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

This example unions an array of linestrings.

```
SELECT ST_AsText(ST_Union(ARRAY[
  ST_GeomFromText('LINESTRING(1 2, 3 4)'),
  ST_GeomFromText('LINESTRING(3 4, 4 5)')
])) AS wktunion;
```

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

関連情報

[ST_CoverageUnion](#), [ST_UnaryUnion](#), [ST_MemUnion](#), [ST_Collect](#), [ST_Intersection](#), [ST_Difference](#), [ST_SymD](#),
[ST_LineMerge](#)

7.14 ジオメトリ処理関数

7.14.1 ST_Buffer

ST_Buffer — あるジオメトリからの距離が指定された距離以下となる点全ての集合となるジオメトリを返します。

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);
```

説明

ジオメトリ/ジオグラフィからの距離が与えられた距離より短くなる全ての点を表現する POLYGON または MULTIPOLYGON を生成します。距離に負数を指定すると、ジオメトリは拡大されずに縮小されます。負の距離ではポリゴンを中心に縮小し切る可能性があり、その時は POLYGON EMPTY を返します。ポイントとラインで負の距離を指定すると、常に空を返します。

ジオメトリの場合、指定される距離の単位は、ジオメトリの空間参照系の単位です。ジオグラフィの場合、指定される距離の単位はメートルです。

3 番目のパラメータは任意で、バッファの精度とスタイルを扱います。バッファの円弧の精度は四分の一円の近似に使用される辺の数です (デフォルトは 8)。バッファのスタイルはキー = 値のペアを空白区切りでリストにして指定します。キーは次の通りです。

- 'quad_segs=#' : 四分の一円近似に使う辺の数 (デフォルトは 8)。
- 'endcap=round|flat|square' : 終端スタイル (デフォルトは"round")。'butt' は'flat' の同義語として受け付けます。
- 'join=round|mitre|bevel' : 接続スタイル (デフォルトは"round")。'miter' も'mitre' の同義語として受け付けます。
- 'mitre_limit=#.#' : マイター比 (訳注: 継ぎ目の内側と外側の距離と線幅との比) の最大値 (継ぎ目スタイルが miter である場合のみ有効)。'miter_limit' は'mitre_limit' の同義語として受け付けます。
- 'side=both|left|right' : デフォルトは'both' です。'left' または'right' でジオメトリの片側へのバッファが行われますが、左右はラインの方向で決まります。これは LINESTRING ジオメトリにのみ適用可能です。POINT または POLYGON ジオメトリに影響を与えません。'left' または'right' を指定した場合にはデフォルトの終端スタイルは正方形になります。

Note



この関数のジオグラフィー版はジオメトリ実装にかぶせた薄いラッパーです。ジオメトリのバウンディングボックスに最適な平面空間参照系を決定します (UTM、ランベルト正積方位図法の北極/南極、最後はメルカトルです)。バッファは平面空間で計算され、WGS84 に戻されます。このため、入力がある UTM ゾーンと比べて非常に大きくなったり、日付変更線とクロスしたりする場合に、求める動作をしないことがあります。



Note

不正な入力を操作することができ、出力は常に妥当なポリゴンジオメトリです。距離 0 によるバッファ生成は時々不正なポリゴンの修復方法として使われます。 **ST_MakeValid**の方がマルチポリゴンも扱えるので、この目的に使うのにより適切です。



Note
バッファ作成は時々、距離内にある地物を検索する際に使われます。この使い方については、[ST_DWithin](#)の方が効率的です。



Note
この関数は Z 値を無視します。この関数を 3 次元ジオメトリ上で使用したとしても、常に 2 次元の結果となります。

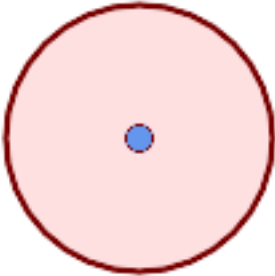
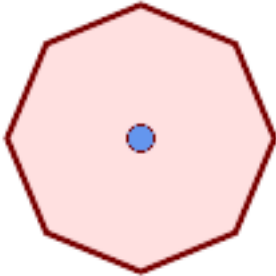
Enhanced: 2.5.0 - `ST_Buffer` のジオメトリ対応版が強化され、バッファを施す側を `side=both|left|right` で指定できるようになりました。

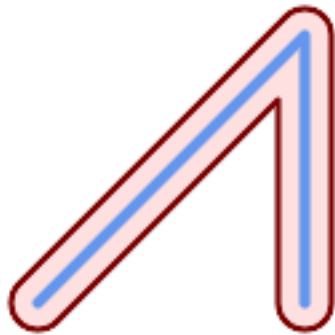
Availability: 1.5 - `ST_Buffer` が強化され、様々な終端と継ぎ目に対応するようになりました。たとえば、道路ラインストリングを道路ポリゴンに変換する際に終端を丸でなく平面や四角で処理したい場合などに使えます。ジオグラフィ用の薄いラップが追加されました。

GEOS モジュールで実現しています。

- ✔ このメソッドは [OGC Simple Features Implementation Specification for SQL 1.1](#) の実装です。s2.1.1.3
- ✔ このメソッドは SQL/MM 仕様の実装です。SQL-MM IEC 13249-3: 5.1.30

例

 <i>quad_segs=8 (デフォルト)</i> <pre>SELECT ST_Buffer(ST_GeomFromText('POINT(100 90)'), 50, 'quad_segs=8');</pre>	 <i>quad_segs=2 (不十分)</i> <pre>SELECT ST_Buffer(ST_GeomFromText('POINT(100 90)'), 50, 'quad_segs=2');</pre>



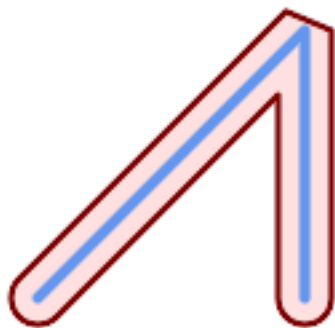
endcap=round join=round (デフォルト)

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



endcap=square

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



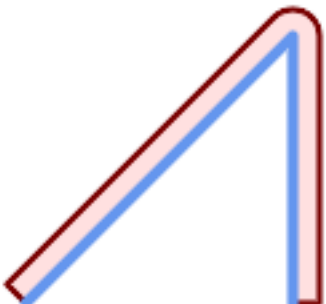
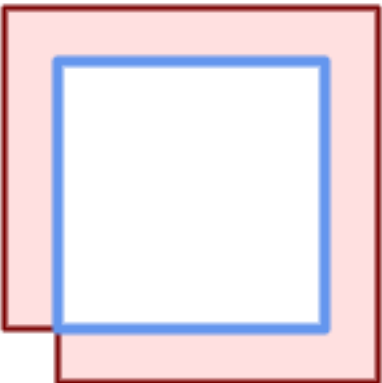
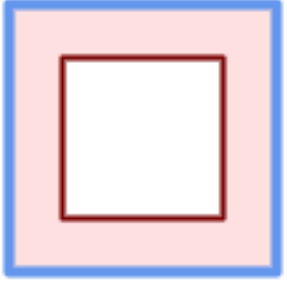
join=bevel

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



join=mitre mitre_limit=5.0 (デフォルトの最大マ
イター比)

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

 <i>side=left</i> <pre>SELECT ST_Buffer(ST_GeomFromText('LINESTRING(50 50,150 150,150 50)') , 10, 'side=left');</pre>	 <i>side=right</i> <pre>SELECT ST_Buffer(ST_GeomFromText('LINESTRING(50 50,150 150,150 50)') , 10, 'side=right');</pre>
 右回り、ポリゴン境界は左 <pre>SELECT ST_Buffer(ST_ForceRHR(ST_Boundary(ST_GeomFromText('POLYGON ((50 50, 50 150, 150 150, 150 50, 50 50))') , 20, 'side=left');</pre>	 右回り、ポリゴン境界は右 <pre>SELECT ST_Buffer(ST_ForceRHR(ST_Boundary(ST_GeomFromText('POLYGON ((50 50, 50 150, 150 150, 150 50, 50 50))') , 20, 'side=right');</pre>

A buffered point approximates a circle. A buffered point forcing approximation of (see diagram) 2 points per quarter circle is poly with 8 sides (see diagram)

```
SELECT ST_NPoints(ST_Buffer(ST_GeomFromText('POINT(100 90)'), 50)) As promisingcircle_pcount,  
       ST_NPoints(ST_Buffer(ST_GeomFromText('POINT(100 90)'), 50, 2)) As lamecircle_pcount;
```

promisingcircle_pcount lamecircle_pcount

```
-----+-----
          33 |          9
```

A lighter, less smooth circle using only 2 points per quarter circle is an octagon.

The following example creates a 100 meter octagon in NAD83 longitude/latitude by transforming to the Massachusetts state plane meter projection and then buffering.

```
SELECT ST_AsText(ST_Buffer(
ST_Transform(ST_SetSRID(ST_Point(-71.063526, 42.35785), 4269), 26986),
100,2)) As octagon;
```

```
POLYGON((236057.5905746494 900908.7599186979,236028.30125276805  ←
900838.0492405792,235957.5905746494 900808.7599186979,235886.87989653074  ←
900838.0492405792,235857.5905746494 900908.7599186979,235886.87989653074  ←
900979.4705968165,235957.5905746494 901008.7599186979,236028.30125276805  ←
900979.4705968165,236057.5905746494 900908.7599186979))
```

関連情報

[ST_Collect](#), [ST_DWithin](#), [ST_SetSRID](#), [ST_Transform](#), [ST_Union](#), [ST_MakeValid](#)

7.14.2 ST_BuildArea

ST_BuildArea — 与えられたジオメトリの構成ラインワークから面ジオメトリを生成します。

Synopsis

geometry **ST_BuildArea**(geometry geom);

説明

入力ジオメトリの構成するラインワークから面ジオメトリを生成します。入力は **LINESTRING**、**MULTILINESTRING**、**POLYGON**、**MULTIPOLYGON**、**GEOMETRYCOLLECTION** が可能です。入力ラインワークがポリゴン形成しない場合には、**NULL** が返ります。

ST_MakePolygonと違い、この関数は複数の線で形成されたリングを受け付け、任意の数量のポリゴン形成できます。

この関数は内側リングを穴に変換します。内側リングもポリゴンに変換したい場合には**ST_Polygonize**を使います。

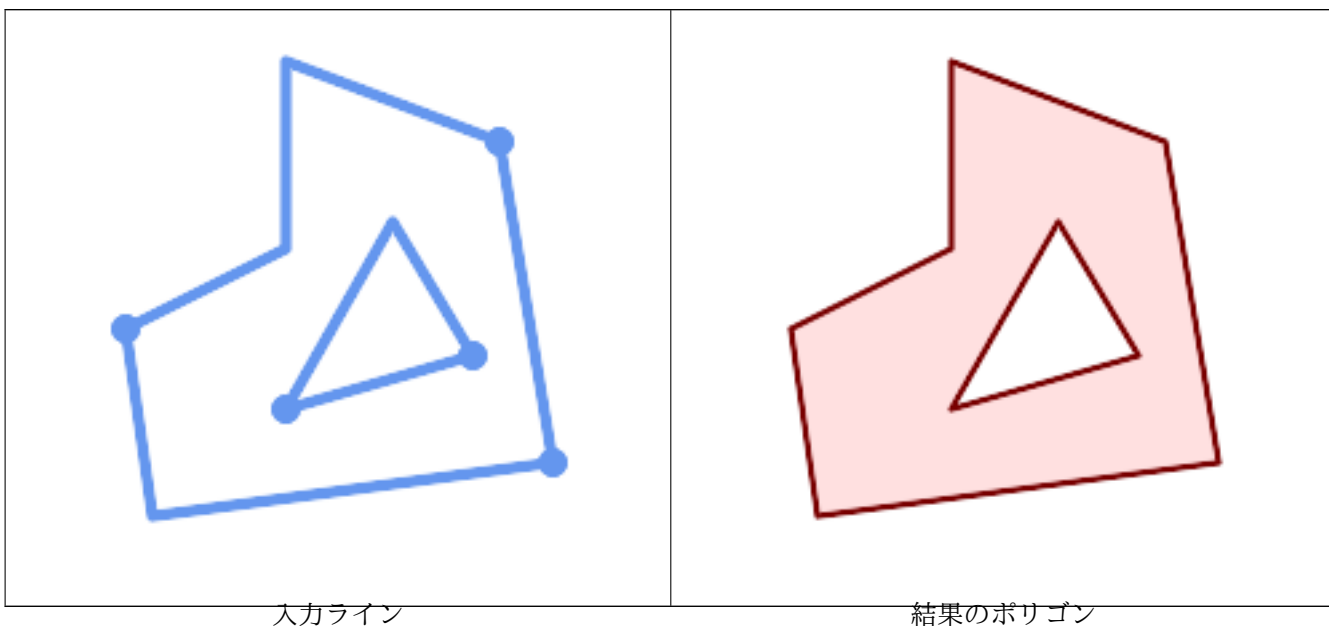


Note

この関数が確実に動作するためには、ラインワークに正しくノードが加えられていなければなりません。ラインにノードを加えるには**ST_Node**を使用します。入力ラインワークがクロスする場合には、この関数は不正なポリゴン生成します。**ST_MakeValid**で、出力を確実に妥当なものにすることができます。

Availability: 1.1.0

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))) AS wkt
FROM data;
```

```
POLYGON((180 40,30 20,20 90,80 120,80 190,160 160,180 40),(150 80,120 130,80 60,150 80))
```



二つの円ポリゴンからのドーナツの生成

```
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;
```

関連情報

[ST_Collect](#), [ST_MakePolygon](#), [ST_MakeValid](#), [ST_Node](#), [ST_Polygonize](#), [ST_BdPolyFromText](#), [ST_BdMPolyFr](#)
(標準 OGC インタフェースを持つこの関数へのラップ)

7.14.3 ST_Centroid

ST_Centroid — ジオメトリの幾何学的重心を返します。

Synopsis

```
geometry ST_Centroid(geometry g1);  
geography ST_Centroid(geography g1, boolean use_spheroid = true);
```

説明

ジオメトリの幾何学的重心を計算します。**[MULTI]POINT** に対しては、入力座標の算術平均として計算されます。**[MULTI]LINESTRING** に対しては、各辺の重み付き長さとして計算されます。**[MULTI]POLYGON** に対しては、面積から計算されます。空ジオメトリが与えられた場合には、空の **GEOMETRYCOLLECTION** が返されます。**NULL** が与えられた場合には、**NULL** が返されます。**CIRCULARSTRING** または **COMPOUNDCURVE** が与えられた場合には、まず **CurveToLine** で直線に変換されてから、**LINESTRING** と同じ計算を行います。

入力が混合次元の場合には、結果は最大次元のジオメトリの重心と同じになります (低次元ジオメトリは重心に 0 の「重み」を与えるためです)。

ポリゴンジオメトリに対しては、重心は必ずしもポリゴンの内部にあるわけではないことに注意して下さい。たとえば、下図の C の形のポリゴンの重心をご覧ください。ポリゴン内部にポイントがあることを保障するには **ST_PointOnSurface** を使います。

New in 2.3.0 : **CIRCULARSTRING** と **COMPOUNDCURVE** に対応するようになりました (**CurveToLine** を使います)。

Availability: 2.4.0 ジオグラフィが導入されました。



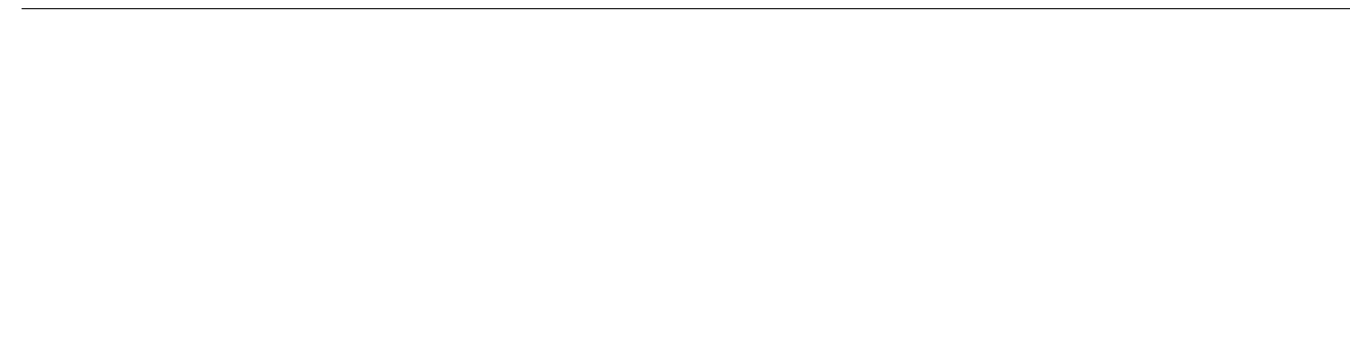
このメソッドは **OGC Simple Features Implementation Specification for SQL 1.1** の実装です。

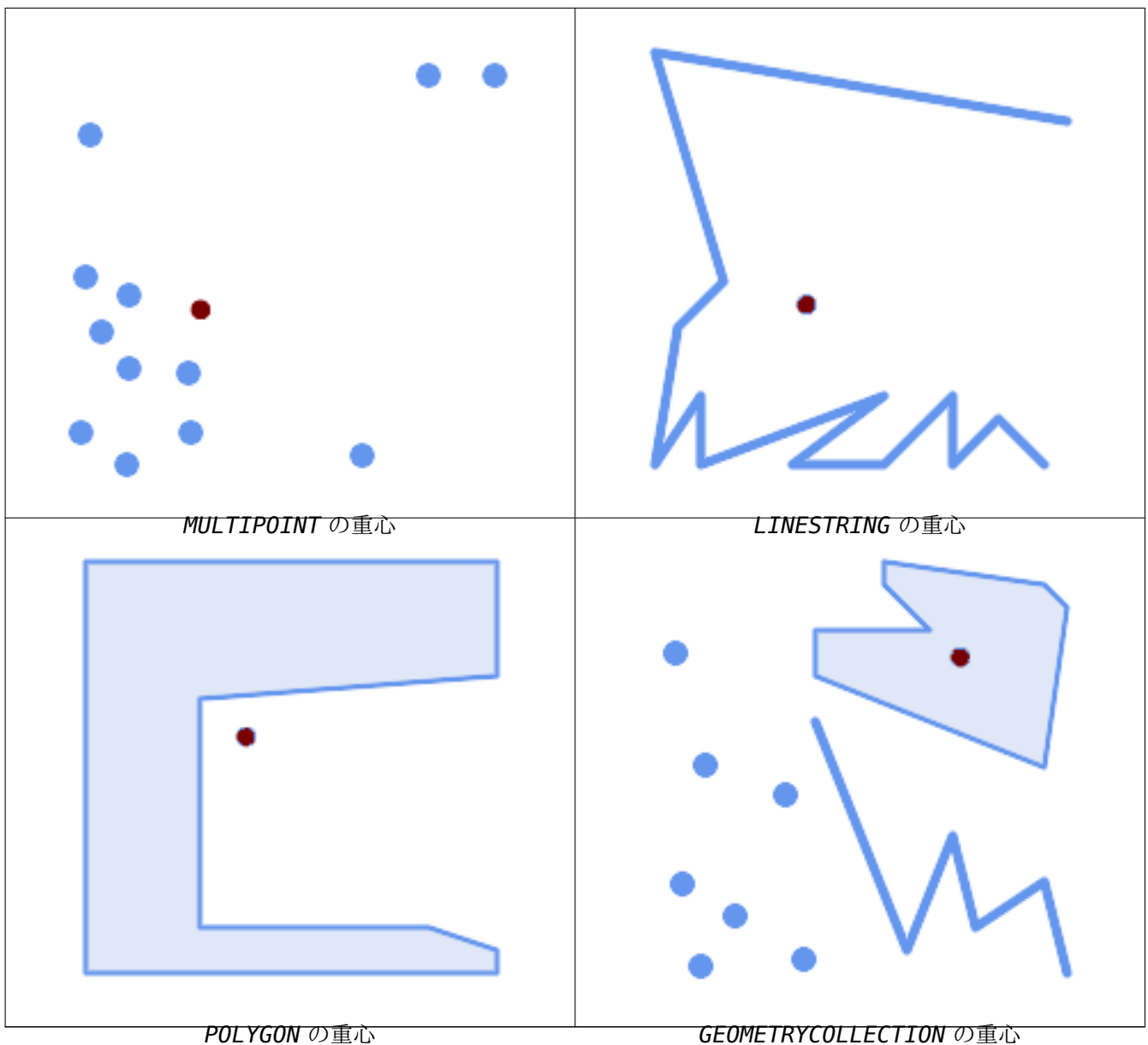


このメソッドは SQL/MM 仕様の実装です。SQL-MM 3: 8.1.4, 9.5.5

例

次に示す図では、赤点が入力ジオメトリの重心です。





```
SELECT ST_AsText(ST_Centroid('MULTIPOINT ( -1 0, -1 2, -1 3, -1 4, -1 7, 0 1, 0 3, 1 1, 2 0, 6 0, 7 8, 9 8, 10 6 )'));
```

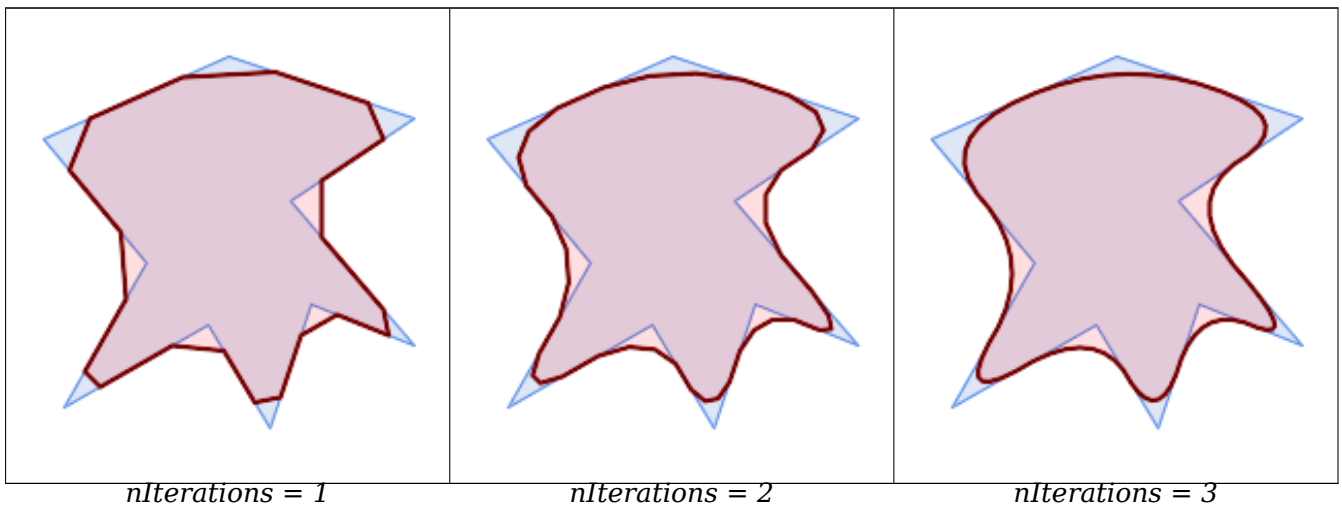
```
st_astext
```

```
-----
POINT(2.307692307692308 3.307692307692308)
(1 row)
```

```
SELECT ST_AsText(ST_Centroid(g))
FROM ST_GeomFromText('CIRCULARSTRING(0 2, -1 1,0 0, 0.5 0, 1 0, 2 1, 1 2, 0.5 2, 0 2)') AS g ;
-----
POINT(0.5 1)
```

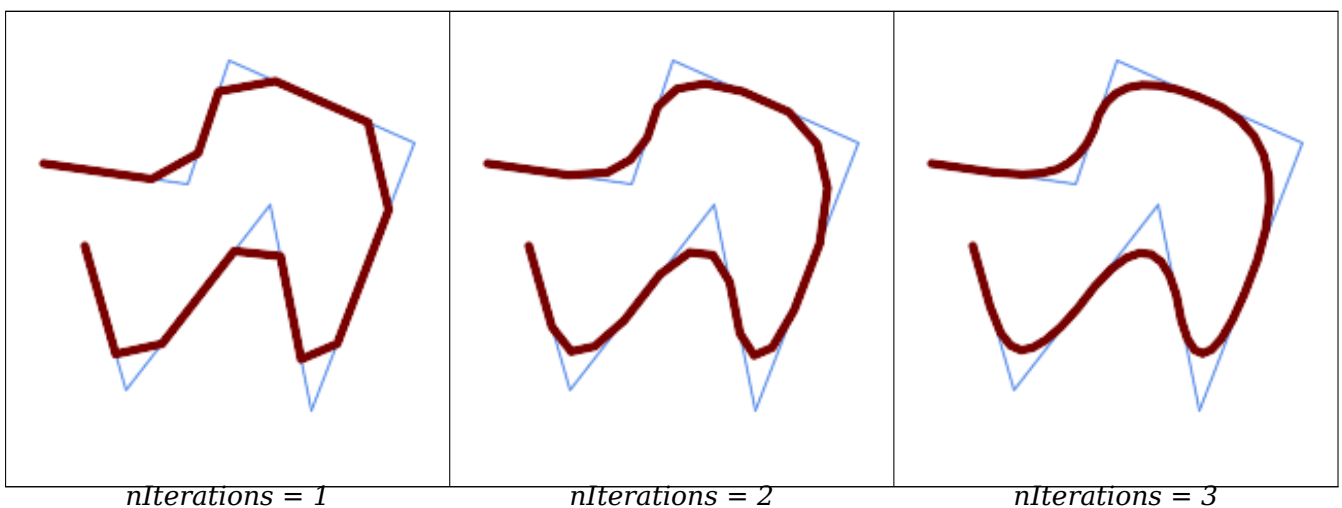
```
SELECT ST_AsText(ST_Centroid(g))
FROM ST_GeomFromText('COMPOUNDCURVE(CIRCULARSTRING(0 2, -1 1,0 0),(0 0, 0.5 0, 1 0), CIRCULARSTRING( 1 0, 2 1, 1 2),(1 2, 0.5 2, 0 2))' ) AS g;
```


1, 2, 3 反復を使ったポリゴンの平滑化:



```
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));
```

1, 2, 3 反復を使ったラインストリングの平滑化:



```
SELECT ST_ChaikinSmoothing('LINESTRING (10 140, 80 130, 100 190, 190 150, 140 20, 120 120, 50 30, 30 100)',
    generate_series(1, 3));
```

関連情報

[ST_Simplify](#), [ST_SimplifyPreserveTopology](#), [ST_SimplifyVW](#)

7.14.5 ST_CatmullRomSmoothing

ST_CatmullRomSmoothing — Catmull-Rom スプライン曲線を用いて平滑化されたジオメトリを返します。

Synopsis

geometry **ST_CatmullRomSmoothing**(geometry geom, integer nSegments = 5);

説明

Catmull-Rom スプライン アルゴリズムを用いた、ラインまたはポリゴンジオメトリの平滑化を行います。**ST_ChaikinSmoothing** と違い、これは補間スプラインで、出力曲線は元の頂点の全てを通ります。連続する頂点の間には `nSegments - 1` 個の補間ポイントが挿入されます。

`nSegments` パラメーターで出力の密度を制御します。`nSegments = 5` (デフォルト) では、スパンごとに 5 個の辺に分かれます (スパンごとに 4 個のポイントが新設されます)。最小値は 2 です。

平滑化には少なくとも 4 個の頂点が必要です。頂点数が少ないジオメトリが与えられると変更せずに返します。ポイントとマルチポイントが与えられると、常に変更せずに返します。

Note



出力曲線の頂点数は、閉じていないラインストリングの場合は $1 + (N-1) * nSegments$ で、閉じてリングになっている場合には $1 + N * nSegments$ です。ここで `N` は入力頂点数です。大きなジオメトリの場合には、ポイント数を減らすために結果に対して単純化を施します (**ST_Simplify**, **ST_SimplifyPreserveTopology**, **ST_SimplifyVW** を参照して下さい)。

結果には、`Z` 値と `M` 値が存在する場合には、これらの補間値が入ります。



この関数は 3 次元に対応し、`Z` 値を削除しません。

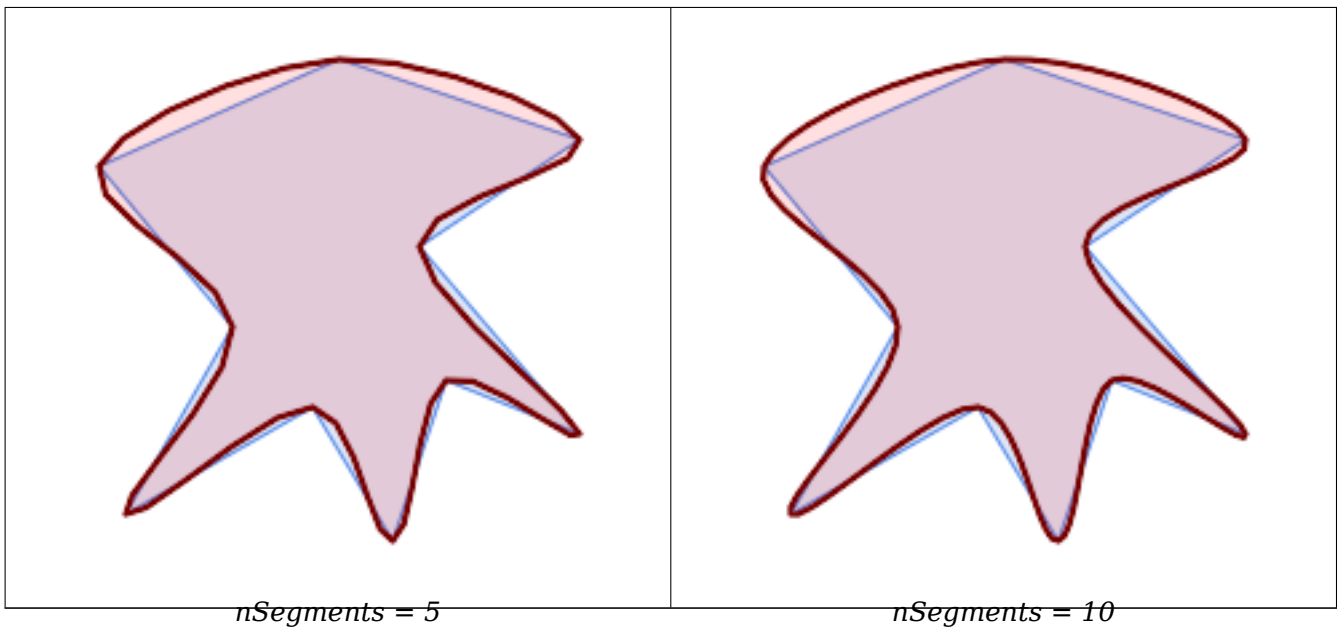
Availability: 3.6.0

例

4 点が共線であるラインストリングの平滑化 (`nSegments=5`):

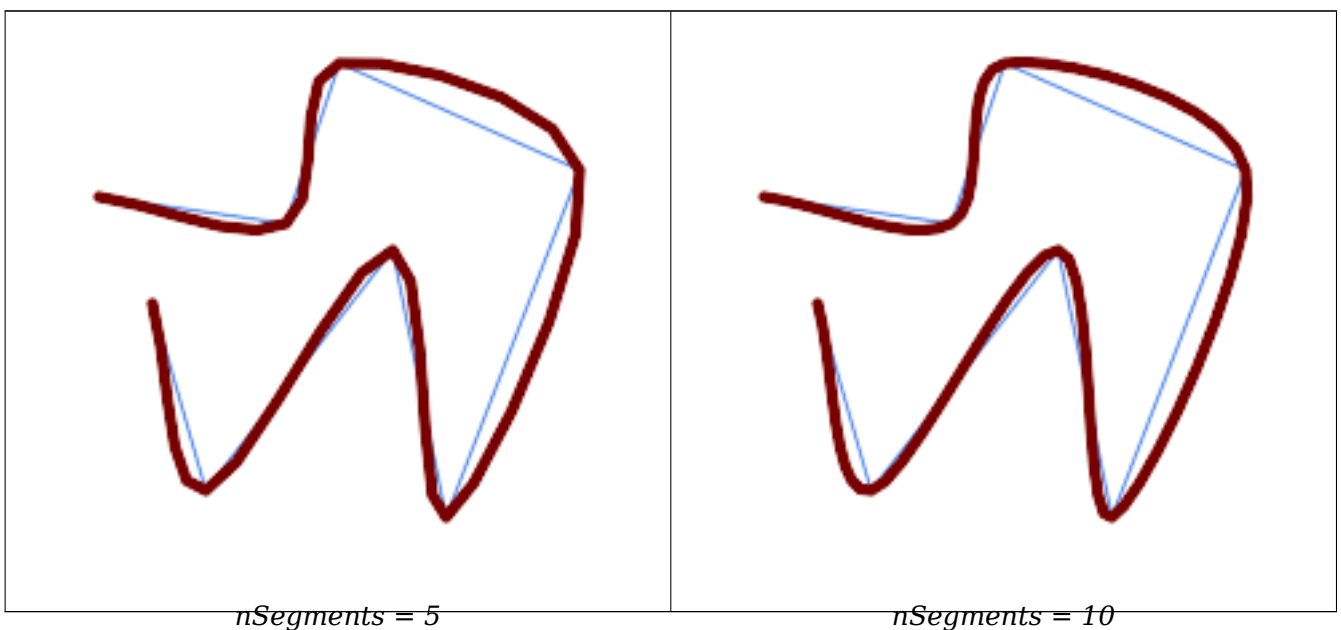
```
SELECT ST_AsText(ST_CatmullRomSmoothing('LINESTRING(0 0, 5 0, 10 0, 15 0)'));  
  
LINESTRING(0 0,1 0,2 0,3 0,4 0,5 0,6 0,7 0,8 0,9 0,10 0,11 0,12 0,13 0,14 0,15 0)
```

ポリゴンの平滑化 (`nSegments = 5` と `10`):



```
SELECT ST_CatmullRomSmoothing('POLYGON ((20 20, 60 90, 100 150, 100 190, 190 160, 130 120, 190 50, 140 70, 120 10, 90 60, 20 20))',
generate_series(5, 10, 5) );
```

ラインストリングの平滑化 (nSegments = 5 と 10):



```
SELECT ST_CatmullRomSmoothing('LINESTRING (10 140, 80 130, 100 190, 190 150, 140 20, 120 120, 50 30, 30 100)',
generate_series(5, 10, 5) );
```

関連情報

[ST_ChaikinSmoothing](#), [ST_Simplify](#), [ST_SimplifyPreserveTopology](#), [ST_SimplifyVW](#)

7.14.6 ST_ConcaveHull

ST_ConcaveHull — 全ての入力ジオメトリの頂点を含む凹ジオメトリを計算します。

Synopsis

```
geometry ST_ConcaveHull(geometry param_geom, float param_pctconvex, boolean param_allow_holes = false);
```

説明

凹包は (通常は) 入力ジオメトリを包含する凹ジオメトリで、凹ジオメトリの頂点は入力頂点の部分集合です。一般的に凹包はポリゴンです。二つ以上の同一線上の点の凹包は 2 点のラインストリングです。一つ以上の同一点の凹包はポイントです。param_allow_holes 引数に TRUE を与えない限りポリゴンは穴を持ちません。

凹包はジオメトリの集合を「真空パック」したジオメトリと考えることができます。輪ゴムで囲うのに似ている **凸包** と違います。凹包は一般的に面積が小さく、また入力ポイントからなる境界が自然になります。

param_pctconvex は計算される凹包の凹度を制御するためのものです。1 では凸包になります。1 から 0 に変化させると凹度が増加していきます。0 では最大の凹度の凹包になります (ただし単一ポリゴンのままです)。適切な値は入力データ固有のものですが、多くの場合 0.3 と 0.1 の間が適正な結果となります。



Note

技術的には param_pctconvex は、入力ポイントのドロネー三角分割における最長エッジと最短エッジとの差の割合で示される長さを決定します。この長さより長いエッジは、三角分割で「浸食」されます。残りの三角形は凹包を形成します。

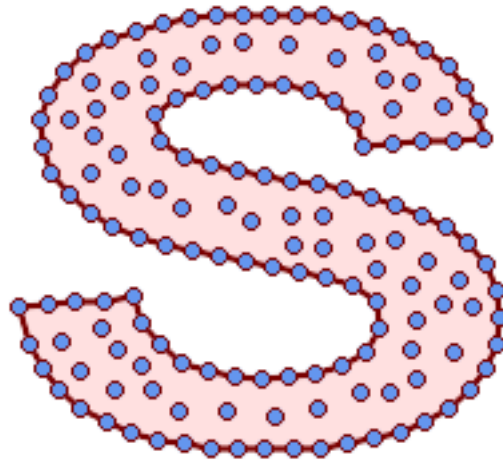
ポイントとラインの入力では、凹包は入力ポイントをポイントとラインの入力では、凹包は全ての入力ポイントを包みます。ポリゴン入力では、凹包は全ての入力ポイントを含み、なおかつ入力ポリゴンが及ぶ全ての面を含みます。ポリゴン入力でもポイントとして計算した凹包が欲しい場合には、**ST_Points**を使って、まずポイントに変換して下さい。

This is not an aggregate function. To compute the concave hull of a set of geometries use **ST_Collect** (e.g. **ST_ConcaveHull(ST_Collect(geom), 0.80)**).

Availability: 2.0.0

Enhanced: 3.3.0, GEOS 3.11 から GEOS ネイティブ実装が有効になりました

例

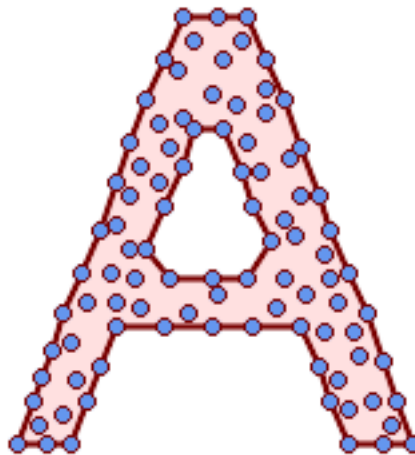


マルチポイントの凹包

```

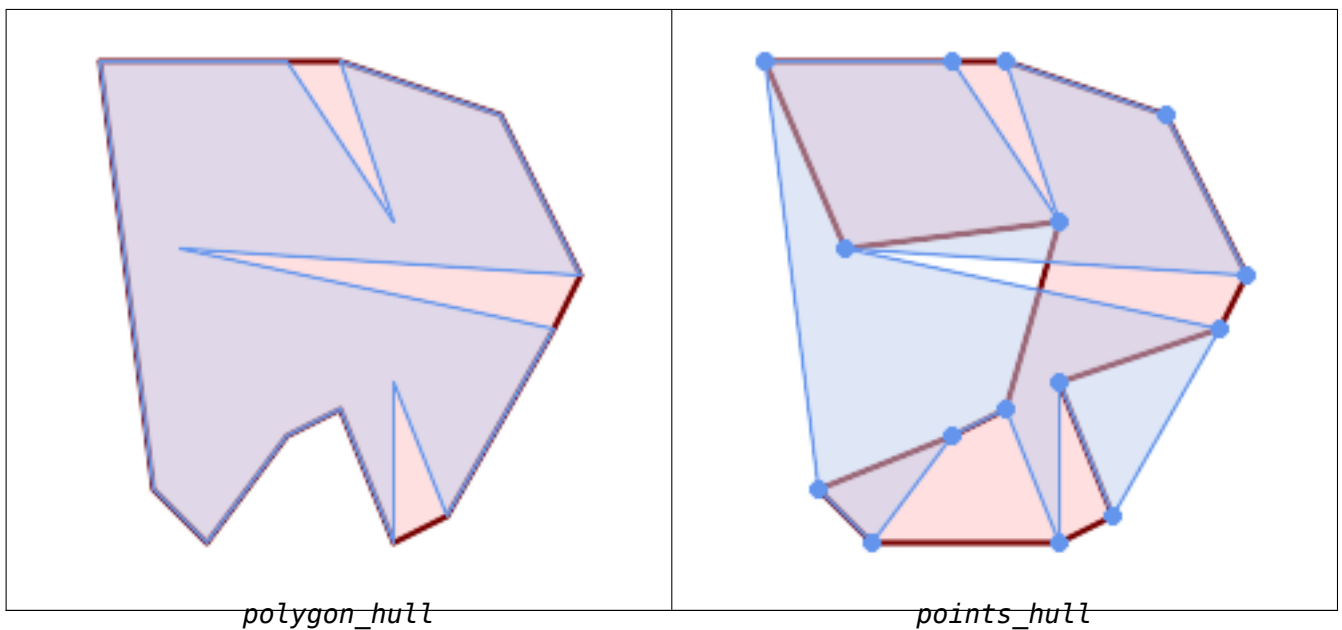
SELECT ST_AsText(ST_ConcaveHull('MULTIPOINT ((10 72), (5376), (56 66), (63 58), (71 51), ←
(81 48), (91 46), (101 45), (111 46), (121 47), (131 50), (140 55), (145 64), (144 74), ←
(135 80), (125 83), (115 85), (105 87), (95 89), (85 91), (75 93), (65 95), (55 98), (45 ←
102), (37 107), (29 114), (22 122), (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))

```



穴を作るマルチポイントの凹包

```
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))
```



ポリゴンの凹包と構成ポイントの凹包との比較。凹包はポリゴンの境界を考慮しますが、ポイントを基にした凹包では考慮されません。

```
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;
```

ST_Collect の併用でジオメトリ集合の凹包の計算。

This example estimates an infected area from point observations.

```
SELECT disease_type,
       ST_ConcaveHull(ST_Collect(obs_pnt), 0.3 ) AS geom
FROM disease_obs
GROUP BY disease_type;
```

関連情報

[ST_ConvexHull](#), [ST_Collect](#), [ST_AlphaShape](#), [ST_OptimalAlphaShape](#)

7.14.7 ST_ConvexHull

ST_ConvexHull — ジオメトリの凸包を計算します。

Synopsis

geometry **ST_ConvexHull**(geometry geomA);

説明

ジオメトリの凸包を計算します。凸包は、入力ジオメトリのすべてを囲む最小の凸ジオメトリです。

凸包は、ジオメトリの集合に輪ゴムをかけて得られるジオメトリと見ることができます。これは「真空パック」に似ている **凹包** と異なります。凸包は観察ポイントの集合に基づいて、響を受けた領域を決定するために、しばしば使われます。

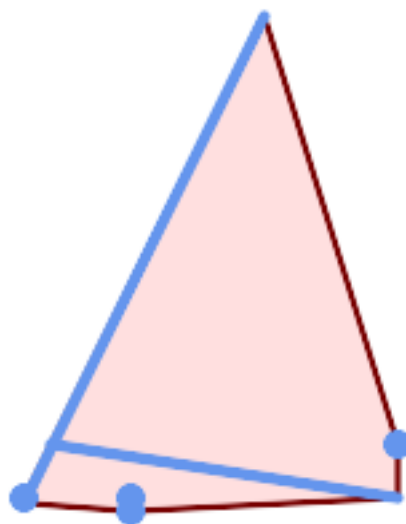
凸包は一般的にポリゴンです。二つ以上の同一線上のポイントの凸包は、2 点のラインストリングになります。一つ以上の同一ポイントの凸包はポイントです。

集約関数ではありません。ジオメトリ集合の凸包を計算するには、ジオメトリ集合をジオメトリコレクションに集約する **ST_Collect** を使います (ST_ConvexHull(ST_Collect(geom)) 等)。

GEOS モジュールで実現しています。

- ✔ このメソッドは **OGC Simple Features Implementation Specification for SQL 1.1** の実装です。s2.1.1.3
- ✔ このメソッドは SQL/MM 仕様の実装です。SQL-MM IEC 13249-3: 5.1.16
- ✔ この関数は 3 次元に対応し、Z 値を削除しません。

例



マルチラインストリングとマルチポイントの凸包

```
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))
```

ST_Collect を使ってジオメトリ集合の凸包を計算します。

This example returns an estimated infected area from point observations.

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

関連情報

[ST_Collect](#), [ST_ConcaveHull](#), [ST_MinimumBoundingCircle](#)

7.14.8 ST_DelaunayTriangles

ST_DelaunayTriangles — ジオメトリの頂点のドロネー三角形を返します。

Synopsis

```
geometry ST_DelaunayTriangles(geometry g1, float tolerance = 0.0, int4 flags = 0);
```

説明

入力ジオメトリの頂点の**ドロネー三角形**を計算します。任意引数 **tolerance** は、入力頂点が近いとスナップする際に使います。ロバスト性が増すことがあります。結果ジオメトリの境界は入力頂点の凸包と同じになります。結果ジオメトリの表現は、次に示す **flags** コードによって変わります。

- 0 - 三角形の POLYGON からなる GEOMETRYCOLLECTION (デフォルト)
- 1 - 三角形の辺の MULTILINESTRING
- 2 - 三角形の TIN

GEOS モジュールで実現しています。

Availability: 2.1.0

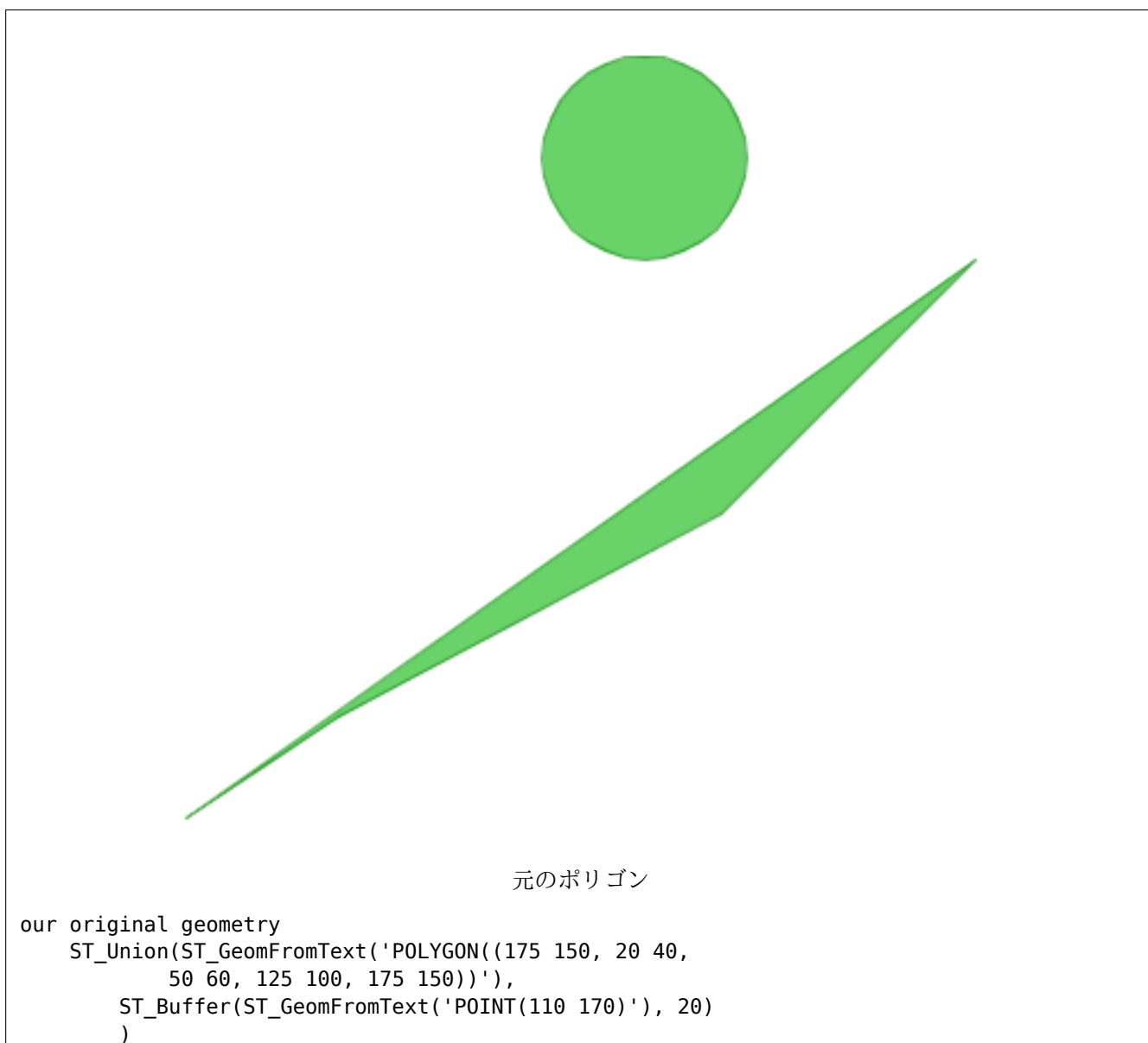


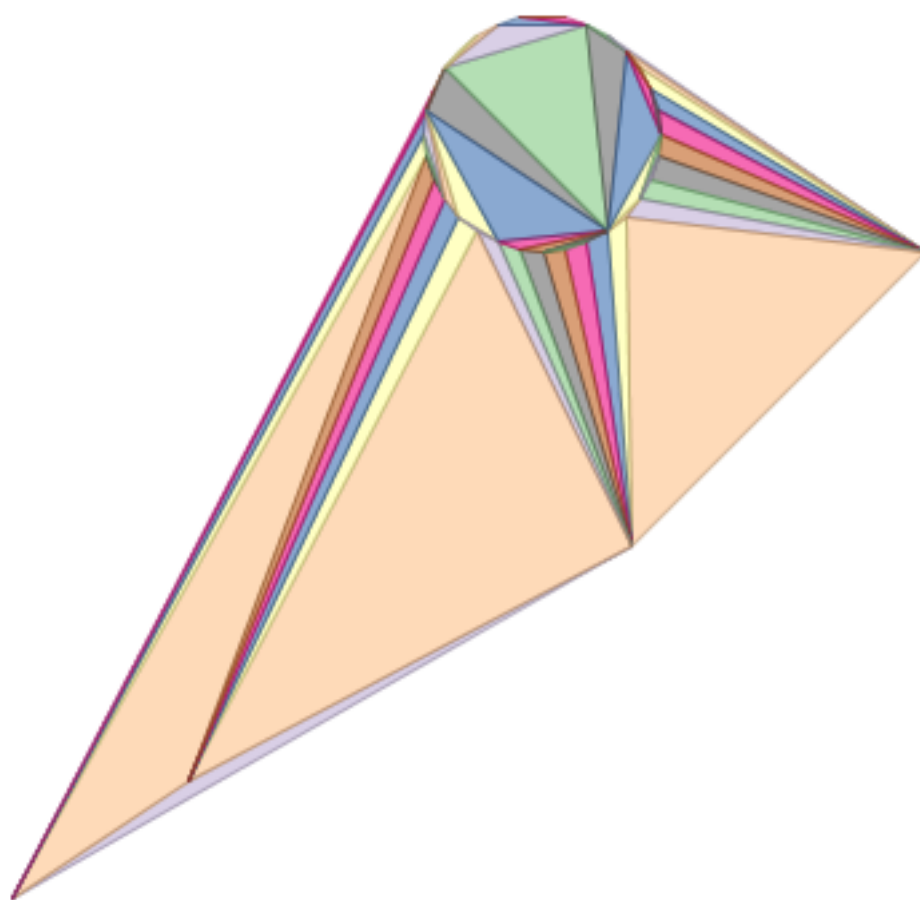
この関数は 3 次元に対応し、Z 値を削除しません。



この関数は三角形と不規則三角網 (TIN) に対応しています。

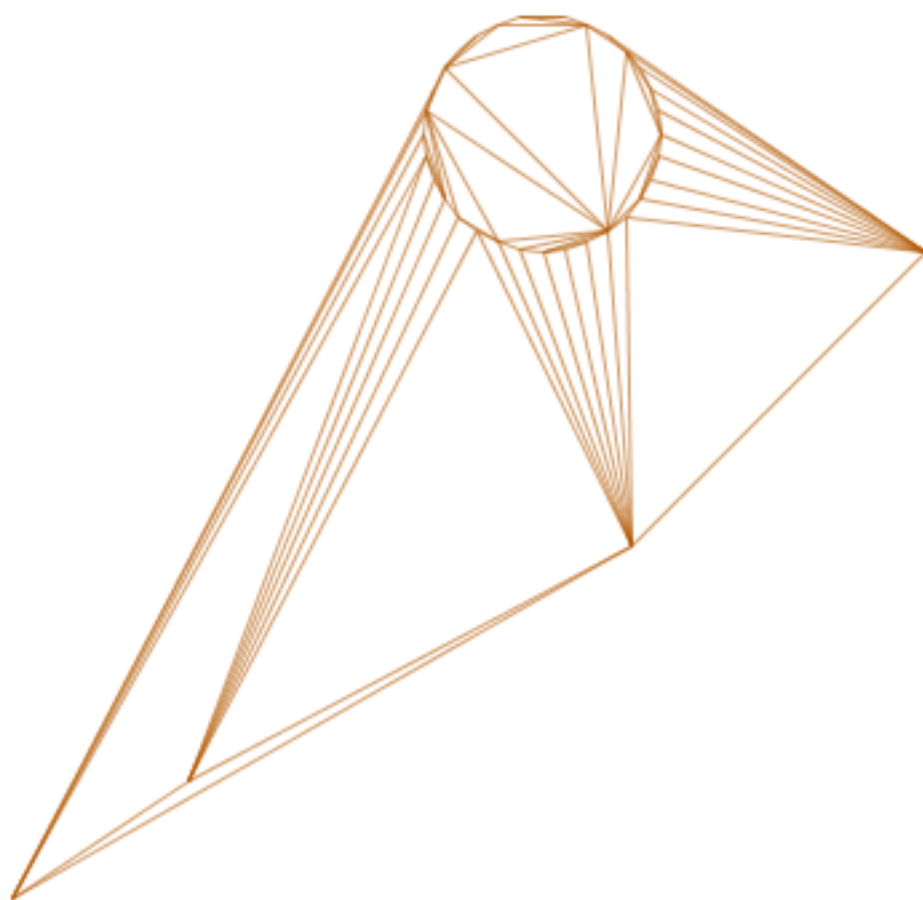
例





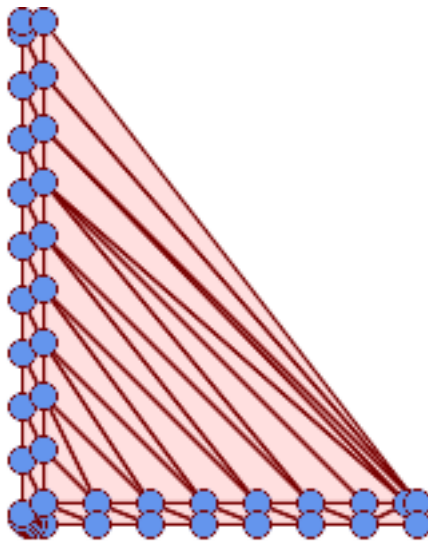
二つのポリゴンの *ST_DelaunayTriangles*: ポリゴンごとに異なる色をつけたドロネー三角形ポリゴン geometries overlaid multilinestring triangles

```
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 dttriag;
```



Delaunay triangles as a MultiLineString

```
SELECT
  ST_DelaunayTriangles(ST_Union(ST_GeomFromText('POLYGON((175 150, 20 40,
    50 60, 125 100, 175 150))'),
    ST_Buffer(ST_GeomFromText('POINT(110 170)'), 20)
  ),0.001,1)
As  dtriag;
```



Delaunay triangles of 45 points as 55 triangle polygons

this produces a table of 42 points that form an L shape

```
SELECT (ST_DumpPoints(ST_GeomFromText('MULTIPOINT(14 14,34 14,54 14,74 14,94 14,114 14,134 14,150 14,154 14,154 6,134 6,114 6,94 6,74 6,54 6,34 6,14 6,10 6,8 6,7 7,6 8,6 10,6 30,6 50,6 70,6 90,6 110,6 130,6 150,6 170,6 190,6 194,14 194,14 174,14 154,14 134,14 114,14 94,14 74,14 54,14 34,14 14)'))).geom
      INTO TABLE l_shape;
```

output as individual polygon triangles

```
SELECT ST_AsText((ST_Dump(geom)).geom) As wkt
FROM ( SELECT ST_DelaunayTriangles(ST_Collect(geom)) As geom
FROM l_shape) As foo;
```

wkt

```
POLYGON((6 194,6 190,14 194,6 194))
POLYGON((14 194,6 190,14 174,14 194))
POLYGON((14 194,14 174,154 14,14 194))
POLYGON((154 14,14 174,14 154,154 14))
POLYGON((154 14,14 154,150 14,154 14))
POLYGON((154 14,150 14,154 6,154 14))
```

Z 値を持つ頂点を使う例。

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)))
```

関連情報

[ST_VoronoiPolygons](#), [ST_TriangulatePolygon](#), [ST_ConstrainedDelaunayTriangles](#), [ST_VoronoiLines](#), [ST_Con](#)

7.14.9 ST_FilterByM

ST_FilterByM — M 値に基づいて頂点を削除します。

Synopsis

geometry **ST_FilterByM**(geometry geom, double precision min, double precision max = null, boolean returnM = false);

説明

M 値に基づいた頂点ポイントのフィルタリングを行います。返されるジオメトリは最小値 (**min** 値) 以上かつ最大値 (**max** 値) 以下の M 値を持つポイントだけでできています。最大値の引数が指定されていない場合には、最小値のみ考慮されます。第 4 引数が指定されていない場合には、M 値は結果ジオメトリに存在しません。結果ジオメトリの頂点ポイントの数が、ジオメトリを構成するに必要な数に達しない場合には、空ジオメトリが返されます。ジオメトリコレクション内の、十分なポイントを持たないジオメトリ要素は消えます。

この関数は主に ST_SetEffectiveArea との併用を意図しています。ST_SetEffectiveArea は頂点の有効面積を M 値に設定します。ST_FilterByM によるフィルタリングだけで、他の計算なしに簡略化されたジオメトリを得られます。



Note

ポイント数が基準を満たすのに十分でない時の ST_SimplifyVW の返り値と ST_FilterByM の返り値とで違いがあります。ST_FilterByM は空ジオメトリを返し、ST_SimplifyVW は十分なポイントを持つジオメトリを返します。



Note

返されるジオメトリは不正である場合があることに注意して下さい。



Note

この関数は全ての次元を返し、Z 値も M 値も残ります。

Availability: 2.5.0

例

フィルタリングされたラインストリング

```
SELECT ST_AsText(ST_FilterByM(geom,30)) simplified
FROM (SELECT ST_SetEffectiveArea('LINESTRING(5 2, 3 8, 6 20, 7 25, 10 10)::geometry) geom ↵
) As foo;
```

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

関連情報

[ST_SetEffectiveArea](#), [ST_SimplifyVW](#)

7.14.10 ST_GeneratePoints

ST_GeneratePoints — ポリゴン内やマルチポリゴン内にランダムなマルチポイントを生成します。

Synopsis

geometry **ST_GeneratePoints**(geometry g, integer npoints, integer seed = 0);

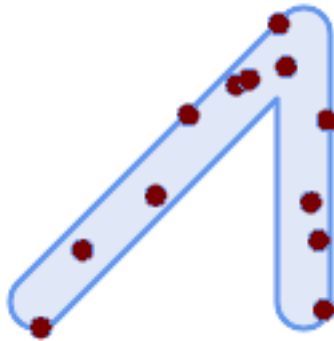
説明

ST_GeneratePoints は、入力の面の内側に、与えられた数の疑似乱数によってマルチポイントを生成します。決定的な点列を生成する場合には任意パラメータ **seed** を使います。この際、0 より大きい数を指定します。

Availability: 2.3.0

Enhanced: 3.0.0 seed パラメータの追加

例



乱数シード値に 1996 を使った、元のポリゴンの上に重なった 12 個のポイントの生成

```
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;
```

ポリゴンテーブル s があって、ポリゴンごとに 12 の個別のポイントが返されます。結果は実行するたびに異なります。

```
SELECT s.id, dp.path[1] AS pt_id, dp.geom
FROM s, ST_DumpPoints(ST_GeneratePoints(s.geom,12)) AS dp;
```

関連情報

[ST_DumpPoints](#)

7.14.11 ST_GeometricMedian

ST_GeometricMedian — マルチポイントの幾何学的中央値を返します。

Synopsis

```
geometry ST_GeometricMedian ( geometry geom, float8 tolerance = NULL, int max_iter = 10000,
boolean fail_if_not_converged = false);
```

説明

マルチポイントジオメトリの幾何中央値の近似値を、Weiszfeld アルゴリズムを使って計算します。幾何中央値は、入力ポイントとの距離の合計を最小にするポイントです。幾何中央値によって、重心よりもはみ出しにくい中心測定ができます。

このアルゴリズムでは、成功した回次の間の距離の変化が、`tolerance` パラメータよりも小さくなるまで繰り返します。`max_iterations` 回を超えた場合には、関数は `fail_if_not_converged` を `FALSE` に指定している (デフォルト) 場合を除いて、エラーを生成して終了します。

`tolerance` 値が渡されていない場合には、デフォルトの許容値は、入力ジオメトリの範囲を基に計算されます。

存在する場合には、入力ポイントの `M` 値は相対的な重みに解釈されます。

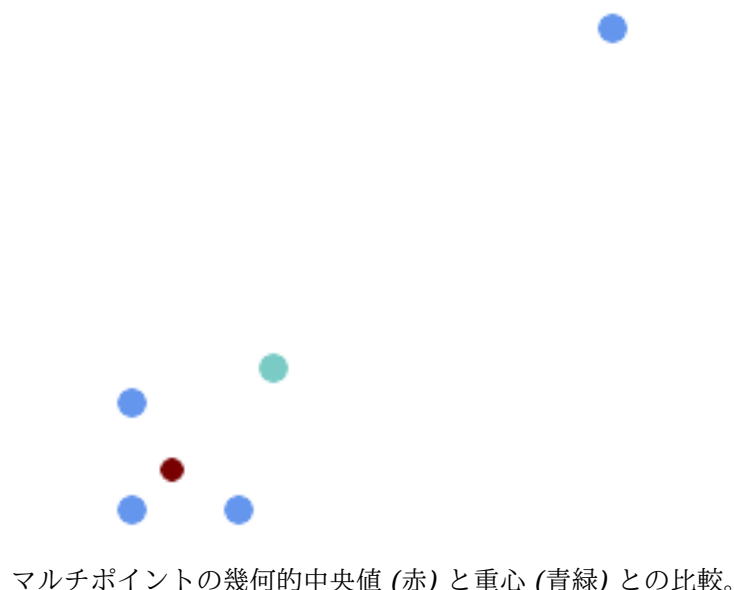
Availability: 2.3.0

Enhanced: 2.5.0 ポイントの重みとしての `M` 値の対応が追加されました。

✔ この関数は 3 次元に対応し、`Z` 値を削除しません。

✔ この関数は `M` 値に対応します。

例



```
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)
```

関連情報

[ST_Centroid](#)

7.14.12 ST_LineMerge

ST_LineMerge — MULTILINESTRING を縫い合わせて形成したラインを返します。

Synopsis

```
geometry ST_LineMerge(geometry amultilinestring);
geometry ST_LineMerge(geometry amultilinestring, boolean directed);
```

説明

MULTILINESTRING の要素を結合して形成された LINESTRING または MULTILINESTRING を返します。ラインは 2 方向交点の終端で結合します。ラインは 3 方向以上の交点では結合しません。

directed が TRUE の場合には、ST_LineMerge はラインストリグ内のポイントの順序を変更しないため、反対方向のラインはマージされません。



Note

MULTILINESTRING/LINESTRING でのみ使用します。他のジオメトリタイプでは空の GEMETRYCOLLECTION を返します。

GEOS モジュールで実現しています。

Enhanced: 3.3.0 directed パラメータを付け付けるようになりました。

directed パラメータを使うには GEOS 3.11.0 以上が必要です。

Availability: 1.1.0

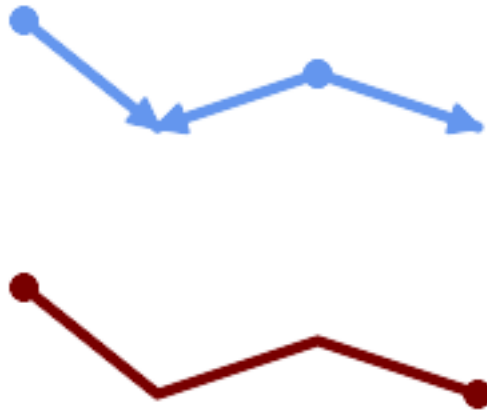


Warning

この関数は M 値を取り除きます。

例

次数 **2**

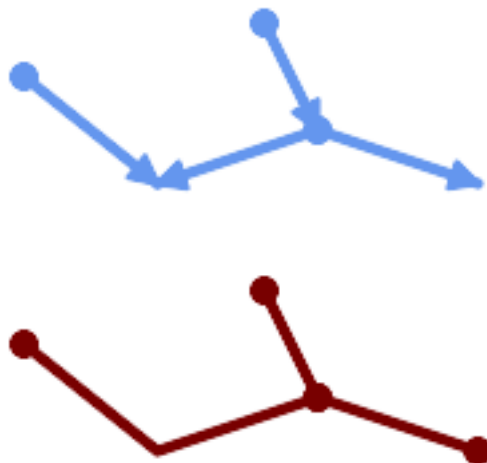


次数 2 のノードはマージされます。

```
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)
```

次数 **3** 以上



インタセクト位置で方向数 > 2 となるラインはマージされません。

```
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))
```

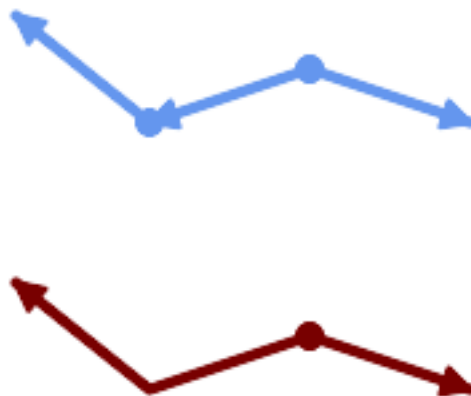
接触していないライン

接触するラインが無くマージができない場合には、元の MULTILINESTRING が返されます。

```
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))
```

directed パラメーター



directed = TRUE の場合には、対応方向のラインはマージされません。

```
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))
```

Z 値を取り扱う

```
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)
```

関連情報

[ST_Segmentize](#), [ST_LineSubstring](#)

7.14.13 ST_MaximumInscribedCircle

ST_MaximumInscribedCircle — ジオメトリに含まれる最大の円を計算します。

Synopsis

(geometry, geometry, double precision) **ST_MaximumInscribedCircle**(geometry geom);

説明

(MULTI)POLYGON に含まれるか、全てのラインとポイントをオーバーラップしない最大の円を探します。返されるレコードには次のフィールドがあります。

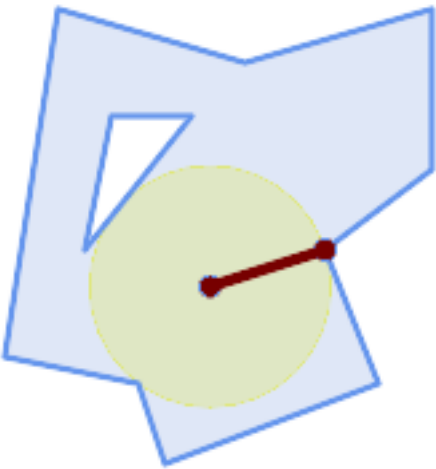
- center - 円の中心点
- nearest - 中心に最も近い、ジオメトリ上のポイント
- radius - 円の半径

ポリゴン系の入力の際は、内側のリングを境界として使用して、円は境界リングに内接します。ラインやポイントの入力の際は、入力ラインとポイントを境界に使用して、円は入力の凸包に内接します。

Availability: 3.1.0.

GEOS 3.9.0 以上が必要です。

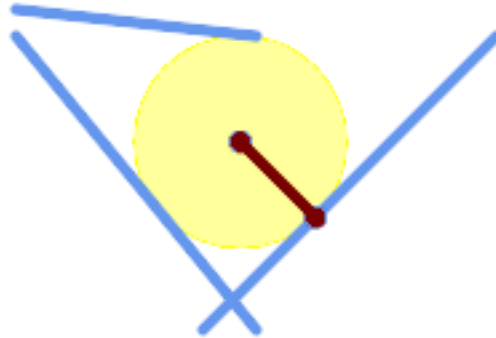
例



ポリゴンの最大の内接円。中心点、最近傍点、半径が返ります。

```
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))');
```

radius	center	nearest
45.18896951053177	POINT(96.9287109375 76.3232421875)	POINT(61.64205411585366 104.55256764481707)



マルチラインストリングの最大の内接円。中心点、最近点と半径が返ります。

関連情報

[ST_MinimumBoundingRadius](#), [ST_LargestEmptyCircle](#)

7.14.14 ST_LargestEmptyCircle

`ST_LargestEmptyCircle` — ジオメトリとオーバラップ市内最大の円を計算します。

Synopsis

(geometry, geometry, double precision) **ST_LargestEmptyCircle**(geometry geom, double precision tolerance=0.0, geometry boundary=POINT EMPTY);

説明

障害物となるポイント集合またはライン集合に重ならない最大の円を探します (ポリゴンジオメトリは障害物として取り込まれますが、境界線しか使われません)。円の中心はポリゴン境界の内側に存在するという制約があります。ポリゴン境界は、デフォルトでは入力ジオメトリの凸包です。円の中心は、障害物から最も遠い距離を持つ境界の内部の点です。円自体は中心点と、円半径を決める障害物上にある点の最近傍点から決まります。

円の中心は、反復的なアルゴリズムを使って、距離許容値で指定された精度に決定されます。精度距離が指定されていない場合には、妥当なデフォルトが使われます。

次のフィールドを含むレコードを返します:

- **center** - 円の中心点
- **nearest** - 中心に最も近い、ジオメトリ上のポイント
- **radius** - 円の半径

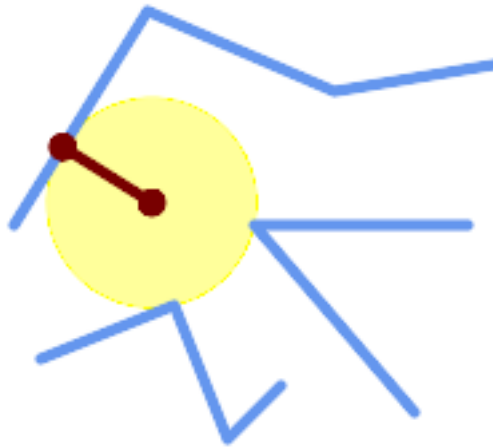
ポリゴン内部において最大の空の円を見つけるには、[ST_MaximumInscribedCircle](#)をご覧ください。

Availability: 3.4.0.

GEOS 3.9.0 以上が必要です。

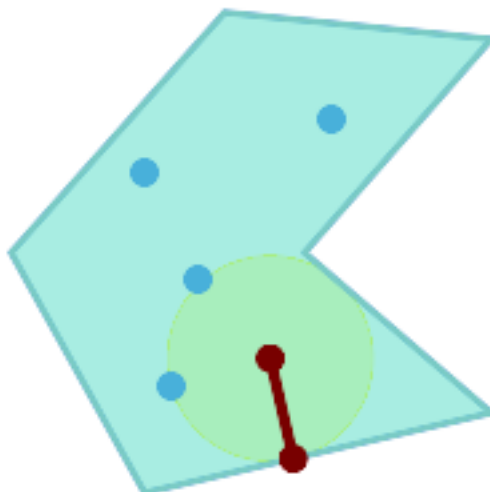
例

```
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))');
```



ラインの集合の中で最大の空の円。

```
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
);
```



ポリゴン内に存在する制約を課されたポイント集合内の最大の空の円。制約ポリゴンの境界は障害物として含めるとともに、円の中心の制約としても指定しなければなりません。

関連情報

[ST_MinimumBandingRadius](#)

7.14.15 ST_MinimumBandingCircle

ST_MinimumBandingCircle — 入力ジオメトリを含む最小の円を返します。

Synopsis

geometry **ST_MinimumBandingCircle**(geometry geomA, integer num_segs_per_circle=48);

説明

入力ジオメトリを含む最小の円を返します。

Note



円はポリゴンで近似されます。デフォルトでは、1 周の 4 分の 1 で 48 辺です。このポリゴンは円の最小バウンディングボックスの近似であるので、入力ジオメトリのいくつかの点はポリゴンに入らない可能性があります。この近似は辺の数を増やすことによって改善され、辺の数を増やすことで得られる不利益は小さいです。ポリゴン近似が適切でない場合のアプリケーションにおいては、[ST_MinimumBandingRadius](#)を使います。

ジオメトリの集合の最小境界円を得るには[ST_Collect](#)と併用します。

最小円 (最大対角線) 上の二つのポイントを計算するには[ST_MinimumBandingCircle](#)を使います。

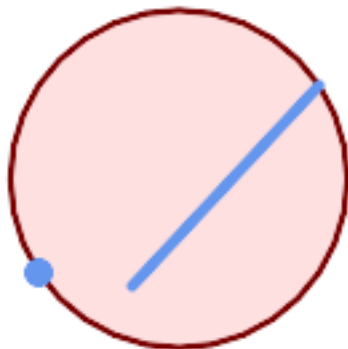
ポリゴンの面積を最小包含円の面積で割った比率は、*Reock* コンパクト度スコアと呼ばれます。

GEOS モジュールで実現しています。

Availability: 1.4.0

例

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



ポイントとラインストリングの最小包含円です。4 分の 1 円の近似に 8 区分使用しています。

```
SELECT ST_AsText(ST_MinimumBoundingCircle(ST_Collect(ST_GeomFromText('LINESTRING(55 75,125 150)'),
                                     ST_Point(20, 80)), 8
                                )) As wktmlbc;
```

```
POLYGON((72.5 178.4024474346694,84.86920388664029 177.18418718546332,96.76306620464646 173.57622349601382,107.72451249529237 167.71720838753708,117.33230052487836 159.83230052487838,125.21720838753708 150.22451249529237,131.07622349601382 139.26306620464646,134.68418718546332 127.3692038866403,135.9024474346694 115,134.68418718546332 102.63079611335971,131.07622349601382 90.73693379535354,125.21720838753708 79.77548750470766,117.33230052487838 70.16769947512164,107.72451249529237 62.282791612462916,96.76306620464648 56.42377650398619,84.86920388664032 52.815812814536685,72.50000000000001 51.59755256533059,60.13079611335969 52.815812814536685,48.236933795353536 56.42377650398618,37.27548750470765 62.28279161246291,27.667699475121637 70.16769947512162,19.782791612462923 79.77548750470763,13.923776503986204 90.73693379535355,10.315812814536692 102.63079611335968,9.097552565330588 114.99999999999999,10.315812814536685 127.3692038866403,13.923776503986197 139.2630662046465,19.78279161246291 150.22451249529234,27.667699475121623 159.83230052487835,37.275487504707634 167.71720838753708,48.236933795353494 173.5762234960138,60.13079611335967 177.18418718546332,72.49999999999999 178.4024474346694))
```

関連情報

[ST_Collect](#), [ST_MinimumBoundingRadius](#), [ST_LargestEmptyCircle](#), [ST_LongestLine](#)

7.14.16 ST_MinimumBoundingRadius

`ST_MinimumBoundingRadius` — ジオメトリを完全に包含する最小円の中心ポイントと半径を返します。

Synopsis

(geometry, double precision) **ST_MinimumBoundingRadius**(geometry geom);

説明

ジオメトリを含む最小円の中心点と半径を計算します。返されるレコードには次のフィールドがあります。

- **center** - 円の中心点
- **radius** - 円の半径

ジオメトリの集合の最小境界円を得るには**ST_Collect**と併用します。

最小円 (最大対角線) 上の二つのポイントを計算するには**ST_MinimumBoundingCircle**を使います。

Availability: 2.3.0

例

```
SELECT ST_AsText(center), radius FROM ST_MinimumBoundingRadius('POLYGON((26426 65078,26531 65242,26075 65136,26096 65427,26426 65078))');
```

st_astext	radius
POINT(26284.84180271327 65267.114509082545)	247.43604559140655

関連情報

ST_Collect, **ST_MinimumBoundingCircle**, **ST_LongestLine**

7.14.17 ST_OrientedEnvelope

ST_OrientedEnvelope — ジオメトリを囲む最小の回転四角形を返します。

Synopsis

geometry **ST_OrientedEnvelope**(geometry geom);

説明

ジオメトリを囲む最小の回転四角形を返します。最小回転四角形が複数個存在することがあるので注意して下さい。入力が退化している場合には、ポイントまたはラインストリングを返すことがあります。

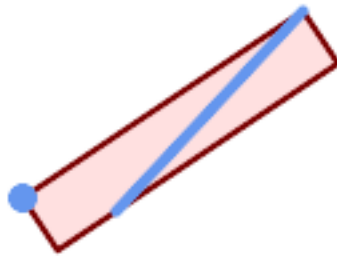
Availability: 2.5.0.

GEOS 3.6.0 以上が必要です。

例

```
SELECT ST_AsText(ST_OrientedEnvelope('MULTIPOINT ((0 0), (-1 -1), (3 2))'));
```

```
POLYGON((3 2,-1 -1,-1.12 -0.84,2.88 2.16,3 2))
```



ポイントとラインストリングの回転したエンベロープ。

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

```
POLYGON((20 80,125 150,138.0769230769241 130.38461538461573,33.07692307692291 ←  
60.384615384615195,20 80))
```

関連情報

[ST_Envelope](#) [ST_MinimumBoundingCircle](#)

7.14.18 ST_OffsetCurve

ST_OffsetCurve — 与えられた距離と方面に入力ラインをずらしたラインを返します。

Synopsis

```
geometry ST_OffsetCurve(geometry line, float signed_distance, text style_parameters=');
```

説明

与えられた距離と方面に入力ラインをずらしたラインを返します。返されるジオメトリの全てのポイントは、入力ジオメトリより与えられた距離以上には離れません。中心線に対する平行線を計算するのに使用します。

距離が正の場合には、入力ラインの左側にずらして、方向が保持されます。負の場合には、右側にずらし、逆方向のラインになります。

距離の単位は空間参照系の単位です。

入力ジオメトリがジグソーパズルのような形状の場合には、出力が **MULTILINESTRING** または **EMPTY** になることがあるので、注意して下さい。

任意指定の第 3 引数では、空白区切りの **key=value** ペアの一覧を指定して、次のような操作をすることができます。

- **'quad_segs=#'** : 4 分の 1 円区分数に近づけるために使われる区分の数 (デフォルトは 8)。
- **'join=round|mitre|bevel'** : 接続スタイル (デフォルトは "round")。'miter' も 'mitre' の同義語として受け付けます。
- **'mitre_limit=#.#'** : マイターの割合制限 (接続スタイルがマイターである場合のみ影響が出ます)।'miter_limit' も 'mitre_limit' の同義語として受け付けます。

GEOS モジュールで実現しています。

GEOS 3.11 では動作が変更され、正と負の両方のオフセットで、オフセット曲線が入力ラインと同じ方向になりました。

Availability: 2.0

Enhanced: 2.5 - GEOMETRYCOLLECTION と MULTILINESTRING への対応追加

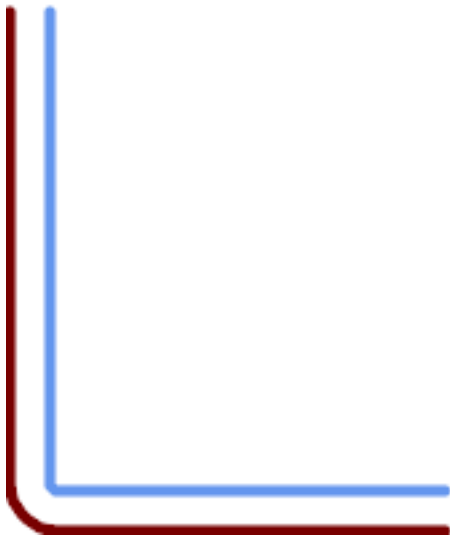
**Note**

この関数は Z 値を無視します。この関数を 3 次元ジオメトリ上で使用したとしても、常に 2 次元の結果となります。

例

道路の周りの開いたバッファの算出

```
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' 元のラインと 15
単位ずらしたライン。

```
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)
```

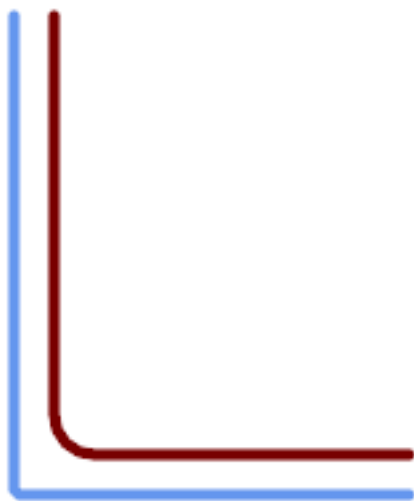


-15, 'quad_segs=4 join=round' 元のラインと-15
単位ずらしたライン

```
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)
```

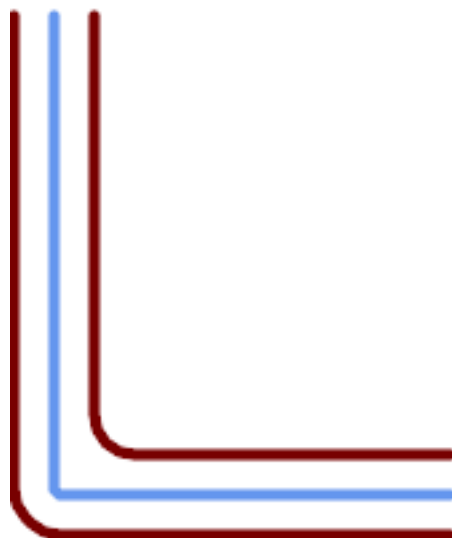


二重にずらして曲線を得ます。1 回目で逆方向にしている、 $-30 + 15 = -15$ としています。

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

morecurvy

```
LINESTRING(164 31,46 31,40.2597485145236
32.1418070123307,
35.3933982822018 35.3933982822018,
32.1418070123307 40.2597485145237,31
46,31 195)
```

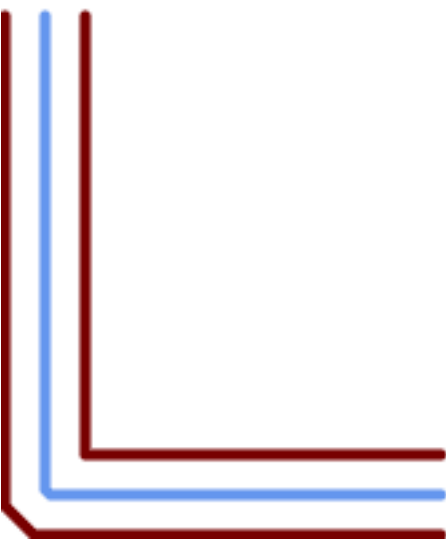


二重にずらして曲線を得て、順方向に 15 ずらして平行線を得ます。元のラインを覆います。

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

parallel curves

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

 15, 'quad_segs=4 join=bevel' と元のライン <pre>SELECT ST_AsText(ST_OffsetCurve(ST_GeomFromText('LINESTRING(164 1,18 1,7.39339828220179 5.39339828220179, 5.39339828220179 7.39339828220179,1 18,1 195)'), 15, 'quad_segs=4 join=bevel'));</pre> output <pre>LINESTRING(164 1,18 1,7.39339828220179 5.39339828220179, 5.39339828220179 7.39339828220179,1 18,1 195)</pre>	 <i>join=mitre mitre_limit=2.1</i> で、15 ずらしたものと-15 ずらしたものを集めたもの。 <pre>SELECT ST_AsText(ST_Collect(ST_OffsetCurve(geom, 15, 'quad_segs=4 join=mitre mitre_limit=2.2'), ST_OffsetCurve(geom, -15, 'quad_segs=4 join=mitre mitre_limit=2.2'))) FROM ST_GeomFromText('LINESTRING(164 1,18 1,7.39339828220179 5.39339828220179, 5.39339828220179 7.39339828220179,1 18,1 195)'), As geom;</pre> output <pre>MULTILINESTRING((164 1,11.7867965644036 1,1 11.7867965644036,1 195), (31 195,31 31,164 31))</pre>

関連情報

[ST_Buffer](#)

7.14.19 ST_PointOnSurface

ST_PointOnSurface — ポリゴン内またはジオメトリ上にあるのが保証されたポイントを返します。

Synopsis

geometry **ST_PointOnSurface**(geometry g1);

説明

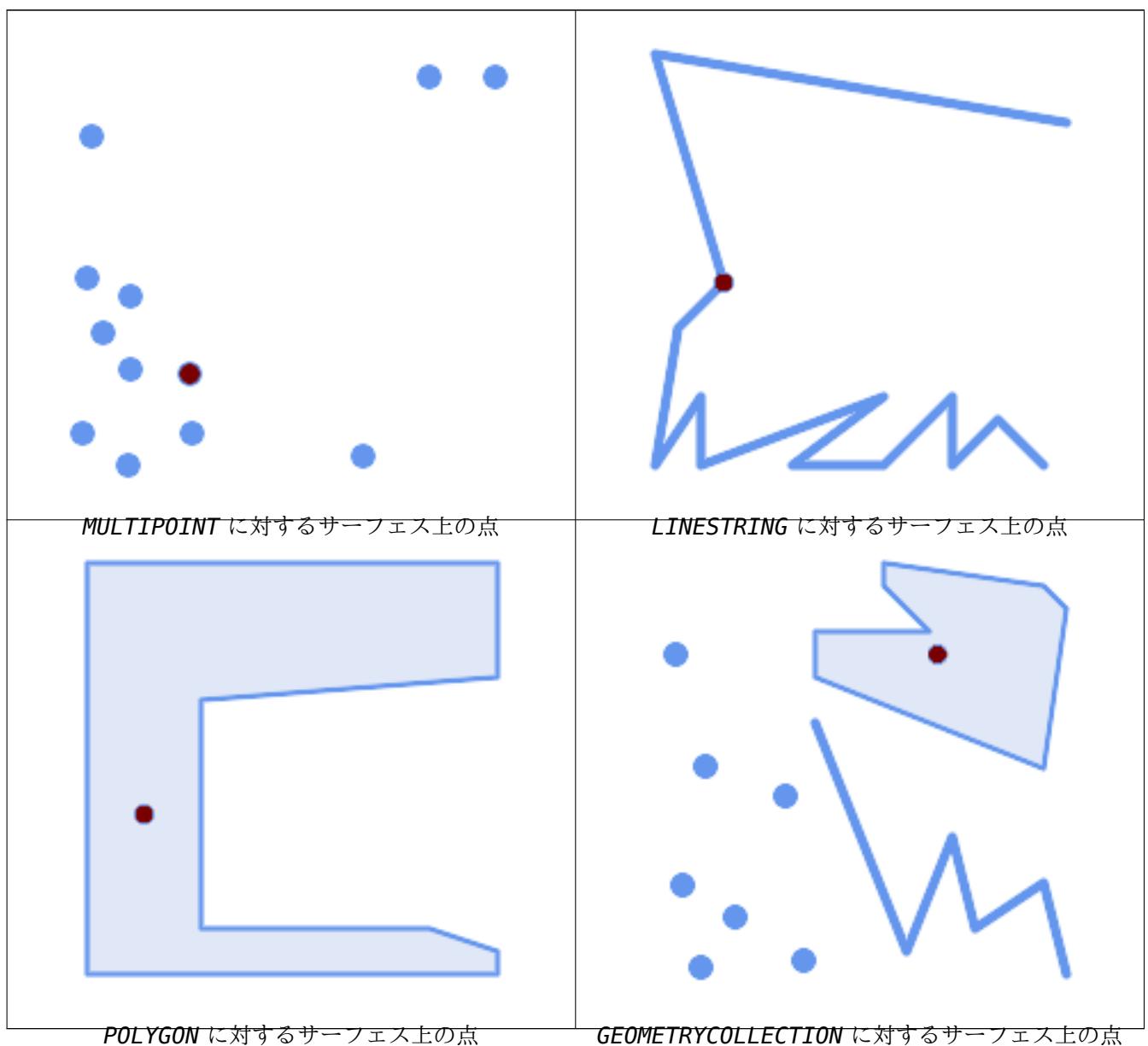
サーフェス (POLYGON、MULTIPOLYGON、CURVEPOLYGON) の内部にあることを保証された POINT を返します。PostGIS では、この関数はラインジオメトリやポイントジオメトリでも動作します。

✓ このメソッドは **OGC Simple Features Implementation Specification for SQL 1.1** の実装です。s3.2.14.2 // s3.2.18.2

✓ このメソッドは SQL/MM 仕様の実装です。SQL-MM 3: 8.1.5, 9.5.6. 仕様では、サーフェスジオメトリのみの ST_PointOnSurface が定義されています。PostGIS は、すべての一般的なジオメトリタイプに対応するよう拡張しています。他のデータベース (Oracle、DB2、ArcSDE) は、サーフェスに対してのみこの機能をサポートしているようです。SQL Server 2008 では、すべての一般的なジオメトリタイプに対応しています。

✓ この関数は 3 次元に対応し、Z 値を削除しません。

例



```

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

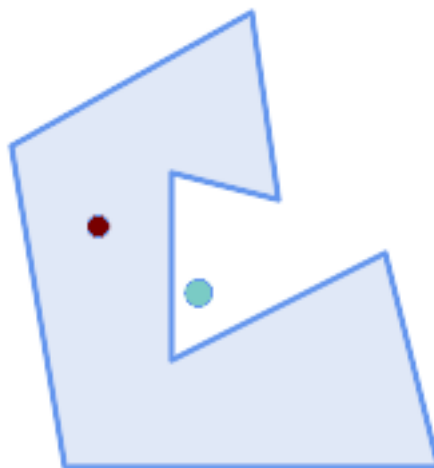
SELECT ST_AsText(ST_PointOnSurface('LINESTRING(0 5, 0 10)')::geometry);
-----
POINT(0 5)

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)

```

例: **ST_Centroid**で計算されたポイントはポリゴンの外になる場合があるのに対して、**ST_PointOnSurface**の結果はポリゴン内に存在することが保証されます。



赤: サーフェス上のポイント、緑: 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)

関連情報

ST_Centroid, **ST_MaximumInscribedCircle**

7.14.20 ST_Polygonize

ST_Polygonize — ジオメトリ集合のラインワークから形成されるポリゴンのコレクションを計算します。

Synopsis

```
geometry ST_Polygonize(geometry set geomfield);  
geometry ST_Polygonize(geometry[] geom_array);
```

説明

ジオメトリの集合のラインワークで形成されるポリゴンを含むジオメトリコレクションを生成します。入力ラインワークがポリゴンを形成しない場合には、空のジオメトリコレクションを返します。

この関数は、区切られた領域の全てを覆うポリゴンを生成します。結果が妥当なポリゴンジオメトリになるようにするには、**ST_BuildArea**を使用して、穴が埋められないようにします。



Note

この関数が正しく動作するには、入力ラインワークは正しくノードが追加されている必要があります。入力が正しくノードが追加されるようにするにはポリゴン化の前に入力ジオメトリに対して**ST_Node**を使います。



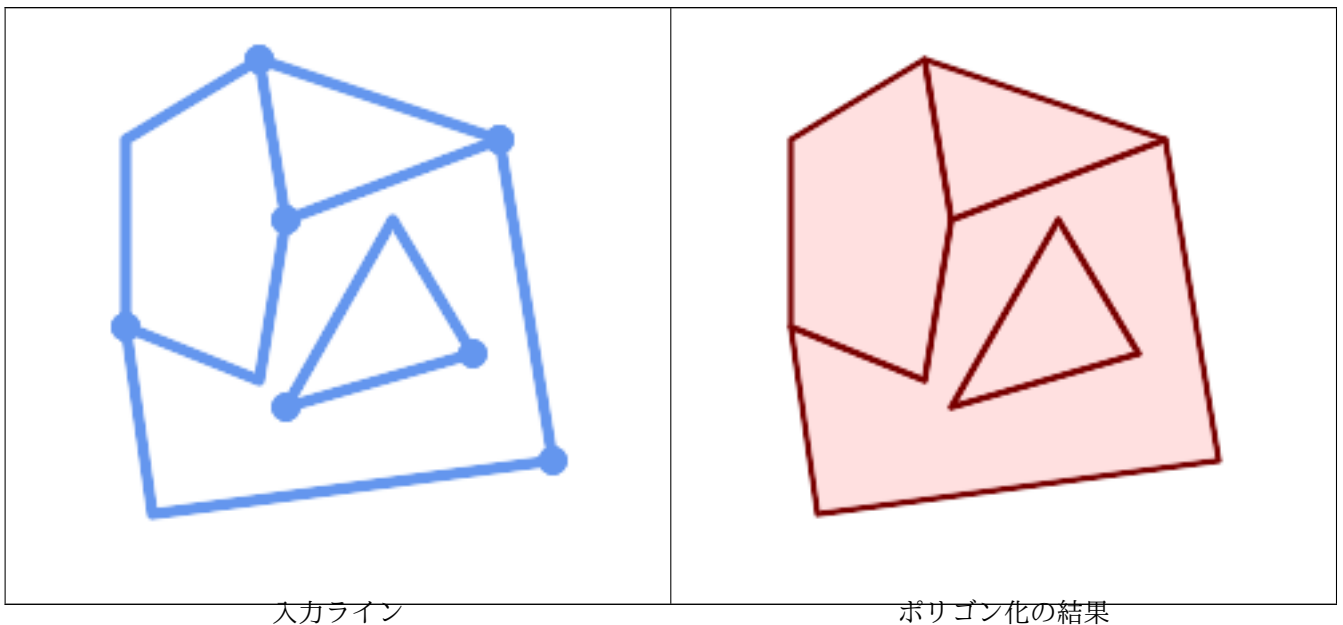
Note

ジオメトリコレクションは外部ツールで扱うのに難しいことがあります。**ST_Dump**でポリゴン化された結果を分かれたポリゴンに変換します。

GEOS モジュールで実現しています。

Availability: 1.0.0RC1

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((80 60,120 130,150 80,80 60)),POLYGON((80 130,70 70,20 90,20 160,70 190,80 130)),POLYGON((160 160,80 130,70 190,160 160)))

```

ラインストリングのテーブルのポリゴン化:

```

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))

```

関連情報

[ST_BuildArea](#), [ST_Dump](#), [ST_Node](#)

7.14.21 ST_ReducePrecision

ST_ReducePrecision — 全ての与えられたグリッド許容値に丸められたポイントからなる妥当なジオメトリを返します。

Synopsis

geometry **ST_ReducePrecision**(geometry g, float8 gridsizes);

説明

全ての与えられたグリッド許容値に丸められたポイントからなる妥当なジオメトリを返します。許容値より下の地物は削除されます。

ST_SnapToGridと違い、返されるジオメトリは妥当で、自己交差の環や崩壊した要素を伴いません。

精度提言は次の場合に使います。

- 座標精度をデータ精度に合わせる
- ジオメトリを表現するのに必要な座標の数を減らす
- 低精度書式 (出力桁数が制限される WKT, GeoJSON, KML 等のテキスト書式) への出力ジオメトリの妥当性の確保。
- 妥当なジオメトリの低精度や制度制限のあるシステム (SDE, Oracle 許容値等) へのエクスポート

Availability: 3.1.0.

GEOS 3.9.0 以上が必要です。

例

```
SELECT ST_AsText(ST_ReducePrecision('POINT(1.412 19.323)', 0.1));
```

```
POINT(1.4 19.3)
```

```
SELECT ST_AsText(ST_ReducePrecision('POINT(1.412 19.323)', 1.0));
```

```
POINT(1 19)
```

```
SELECT ST_AsText(ST_ReducePrecision('POINT(1.412 19.323)', 10));
```

```
POINT(0 20)
```

精度を落とすことで頂点数を減らすことができます。

```
SELECT ST_AsText(ST_ReducePrecision('LINESTRING (10 10, 19.6 30.1, 20 30, 20.3 30, 40 40)', 1));
```

```
LINESTRING (10 10, 20 30, 40 40)
```

妥当性を確保するために必要な場合には、精度を落とす際にポリゴンを分割します。

```
SELECT ST_AsText(ST_ReducePrecision('POLYGON ((10 10, 60 60.1, 70 30, 40 40, 50 10, 10 10))', 10));
```

```
MULTIPOLYGON(((40 40,50 10,10 10,40 40)),((60 60,70 30,40 40,60 60)))
```

関連情報

ST_SnapToGrid, **ST_Simplify**, **ST_SimplifyVW**

7.14.22 ST_SharedPaths

ST_SharedPaths — 二つの LINESTRING/MULTILINESTRING の入力共有パスのコレクションを返します。

Synopsis

```
geometry ST_SharedPaths(geometry lineal1, geometry lineal2);
```

説明

二つの入力ジオメトリが共有するパスのコレクションを返します。順方向に行くものはコレクションの一つ目の要素にあり、逆方向は二つ目の要素にあります。これらのパス自体は一つ目のジオメトリの方向をもとにします。

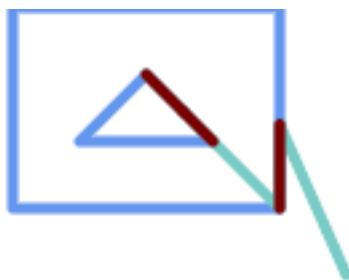
GEOS モジュールで実現しています。

Availability: 2.0.0

例: 共有パスの探索



マルチラインストリングとラインストリング



マルチラインストリングとラインストリングとの共有パスと元のジオメトリ。

```
SELECT ST_AsText(ST_SharedPaths(ST_GeomFromText('MULTILINESTRING((26 125,26 200,126 200,126 125,26 125),
(51 150,101 150,76 175,51 150))'),
ST_GeomFromText('LINESTRING(151 100,126 156.25,126 125,90 161, 76 175)')
)
)

GEOMETRYCOLLECTION(MULTILINESTRING((126 156.25,126 125),
(101 150,90 161),(90 161,76 175)),MULTILINESTRING EMPTY)
```

same example but linestring orientation flipped

```
SELECT ST_AsText(ST_SharedPaths(ST_GeomFromText('LINESTRING(76 175,90 161,126 125,126 156.25,151 100)'),
ST_GeomFromText('MULTILINESTRING((26 125,26 200,126 200,126 125,26 125),
(51 150,101 150,76 175,51 150))')
)
)

GEOMETRYCOLLECTION(MULTILINESTRING EMPTY,
MULTILINESTRING((76 175,90 161),(90 161,101 150),(126 125,126 156.25)))
```

関連情報

[ST_Dump](#), [ST_GeometryN](#), [ST_NumGeometries](#)

7.14.23 ST_Simplify

ST_Simplify — Douglas-Peucker アルゴリズムを使用して、簡略化したジオメトリを返します。

Synopsis

```
geometry ST_Simplify(geometry geom, float tolerance);
geometry ST_Simplify(geometry geom, float tolerance, boolean preserveCollapsed);
```

説明

単純化したジオメトリの表現を計算します。Douglas-Peucker algorithm を使います。単純化の tolerance (許容値) は、距離の値で、単位は入力 SRS (空間参照系) の単位です。単純化によって、単純化された線の許容値距離内にある頂点は削除されます。入力が有効であっても結果が有効にならないことがあります。

この関数は、どの種類のジオメトリでも (GEOMETRYCOLLECTION であっても) 呼ぶことができますが、ライン要素とポリゴン要素だけが単純化されます。線ジオメトリの端点は保持されます。

preserveCollapsed フラグは、与えられた許容値では削除される小さいジオメトリを保持します。たとえば、10m 許容値で 1m 長の線をたぬんかする場合に preserveCollapsed が TRUE の場合には、このラインが消えることはありません。このフラグは、非常に小さい地物がマップから消えるのを防ぐために、地図描画目的の際に使います。



Note
返されるジオメトリは単純性を喪失している場合があります (ST_IsSimple参照)、トポロジが保持されていないばあ外アリ、ポリゴンの結果が不正になる可能性があります (ST_IsValid参照)。トポロジを保全し、妥当性を確保するには、ST_SimplifyPreserveTopologyを使います。



Note
この関数は、ポリゴン間で共有される境界は保持されません。保持するにはST_CoverageSimplifyを使います。

Availability: 1.2.2

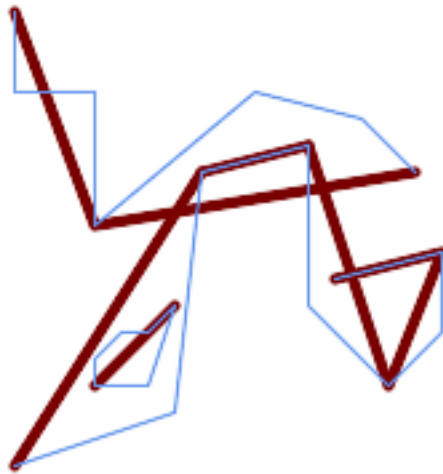
例

A circle simplified with a medium tolerance becomes an octagon, and with a large tolerance disappears.

```
SELECT ST_NPoints(geom) AS np_before,
       ST_NPoints(ST_Simplify(geom, 0.1)) AS np01_notbadcircle,
       ST_NPoints(ST_Simplify(geom, 0.5)) AS np05_notquitecircle,
       ST_NPoints(ST_Simplify(geom, 1)) AS np1_octagon,
       ST_NPoints(ST_Simplify(geom, 10)) AS np10_gone,
       (ST_Simplify(geom, 100) is null) AS np100_geometrygoesaway
FROM (SELECT ST_Buffer('POINT(1 3)', 10,12) As geom) AS t;
```

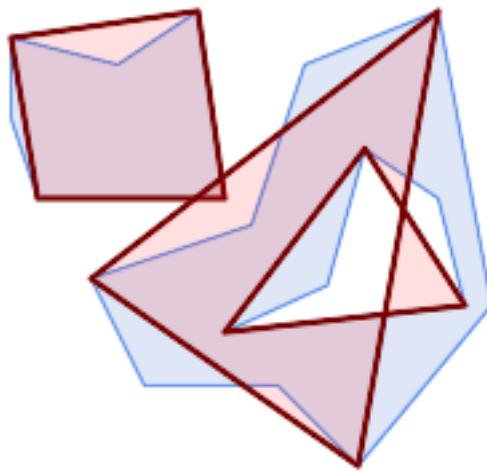
np_before	np01_notbadcircle	np05_notquitecircle	np1_octagon	np10_gone	np100_geometrygoesaway	
49	33	17	9	null	t	

ラインの集合の簡略化。ラインは簡略化後のインタセクトする場合があります。



```
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);
```

MULTIPOLYGON の簡略化。ポリゴンの結果は不正になることがあります。



```
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);
```

関連情報

[ST_IsSimple](#), [ST_SimplifyPreserveTopology](#), [ST_SimplifyVW](#), [ST_CoverageSimplify](#), [Topology](#) [ST_Simplify](#)

7.14.24 ST_SimplifyPreserveTopology

`ST_SimplifyPreserveTopology` — Douglas-Peucker アルゴリズムを使用して、単純化した妥当なジオメトリを返します。

Synopsis

geometry **ST_SimplifyPreserveTopology**(geometry geom, float tolerance);

説明

Douglas-Peucker algorithmの一種を用いて単純化したジオメトリを計算します。結果のトポロジが入力と同じになるように単純化を制限します。簡略化の **tolerance** (許容値) は距離値で、単位は入力 **SRS** の単位です。簡略化によって、トポロジが保持される限り、簡略化された線の許容距離値内の頂点が削除されます。入力が妥当かつ単純な場合には、結果は妥当かつ単純です。

この関数は、どの種類のジオメトリ (**GEOMETRYCOLLECTION** も含む) を引数にとっても呼ぶことができますが、ラインとポリゴン要素だけを単純化します。結果はリングの数は同じで、入力が相互にインタセクトしていないなら結果も相互にインタセクトしません。線ジオメトリの端点は保持されます。



Note

この関数は、ポリゴン間で共有される境界は保持されません。保持するには**ST_CoverageSimplify**を使います。

GEOS モジュールで実現しています。

Availability: 1.3.3

例

For the same example as **ST_Simplify**, **ST_SimplifyPreserveTopology** prevents the geometry from disappearing.

```
SELECT  ST_NPoints(geom) AS np_before,
        ST_NPoints(ST_SimplifyPreserveTopology(geom, 0.1)) AS np01_notbadcircle,
        ST_NPoints(ST_SimplifyPreserveTopology(geom, 0.5)) AS np05_notquitecircle,
        ST_NPoints(ST_SimplifyPreserveTopology(geom, 1))   AS np1_octagon,
        ST_NPoints(ST_SimplifyPreserveTopology(geom, 10))  AS np10_minimal,
        ST_NPoints(ST_SimplifyPreserveTopology(geom, 100)) AS np100_minimal
FROM (SELECT ST_Buffer('POINT(1 3)', 10,12) AS geom) AS t;
```

np_before	np01_notbadcircle	np05_notquitecircle	np1_octagon	np10_minimal	np100_minimal
49	33	17	9	4	

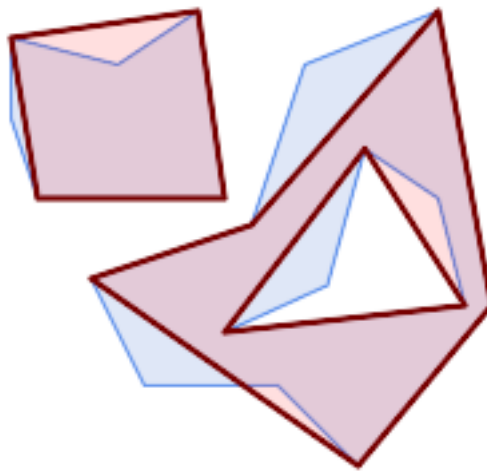
49	33	17	9	4
	4			

非交差ラインのトポロジを保全した、ラインの集合の単純化を行います。



```
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);
```

外側リングと内側リングのトポロジを保全した MULTIPOLYGON を単純化します。



```
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);
```

関連情報

[ST_Simplify](#), [ST_SimplifyVW](#), [ST_CoverageSimplify](#)

7.14.25 ST_SimplifyPolygonHull

ST_SimplifyPolygonHull — ポリゴンジオメトリに対してトポロジを保存した状態で簡略化した外側または内側の凹包を計算します。

Synopsis

```
geometry ST_SimplifyPolygonHull(geometry param_geom, float vertex_fraction, boolean is_outer = true);
```

説明

ポリゴンジオメトリに対してトポロジを保存した状態で簡略化した外側または内側の凹包を計算します。外側の凹包は入力ジオメトリを完全に覆います。内側の凹包は完全に入力ジオメトリに覆われます。返される結果は、入力ポリゴンの頂点の部分集合から形成されるポリゴンジオメトリです。MULTIPOLYGON と穴は処理され、結果は入力ジオメトリと同じ構造になります。

頂点数の減少は `vertex_fraction` パラメータで制御できます。0 から 1 の範囲を取ります。小さい値では簡略化が進み、頂点数が減り、凹性が下がります。外側の凹包も内側の凹包も、`vertex_fraction` 値が 1.0 では、元のジオメトリが生成されます。0.0 では、外側の凹包については凸包 (単一ポリゴンの場合) が生成され、内側の凹包については三角形が生成されます。

単純化処理では、頂点数が目標に達するまで、最も少ない面積を持つ凹の角を徐々に削除しています。これによりエッジがクロスするのを防ぎ、常に妥当なポリゴンジオメトリを返します。

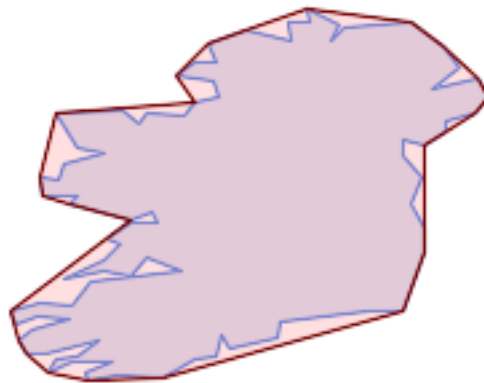
相対的に辺が長いジオメトリでより良い結果を得るには、次のように、辺の分割を行います。

GEOS モジュールで実現しています。

Availability: 3.3.0.

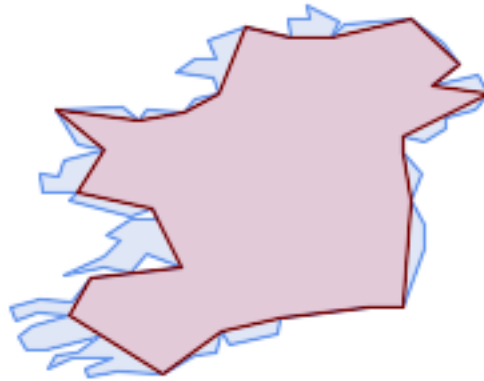
GEOS 3.11.0 以上が必要です。

例



ポリゴンの外側の凹包

```
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);
```



ポリゴンの内側の凹包

```
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);
```



中間点を挿入したマルチポリゴンの外側の凹包による簡略化

```
SELECT ST_SimplifyPolygonHull(ST_Segmentize(ST_Letters('xt'), 2.0),
  0.1);
```

関連情報

[ST_ConvexHull](#), [ST_SimplifyVW](#), [ST_ConcaveHull](#), [ST_Segmentize](#)

7.14.26 ST_SimplifyVW

ST_SimplifyVW — Visvalingam-Whyatt アルゴリズムを使用して、入力ジオメトリを簡略化したジオメトリを返します。

Synopsis

geometry **ST_SimplifyVW**(geometry geom, float tolerance);

説明

Visvalingam-Whyatt アルゴリズムを使用して、入力ジオメトリを簡略化したジオメトリを返します。**tolerance** は面積値で、単位は入力 **SRS** の単位です。単純化によって、許容値より小さい面積の「角」となる頂点が削除されます。入力が妥当な場合にでも、結果が不正になる場合があります。

この関数は、どの種類のジオメトリでも (**GEOMETRYCOLLECTION** であっても) 呼ぶことができますが、ライン要素とポリゴン要素だけが単純化されます。線ジオメトリの端点は保持されます。



Note

返されるジオメトリは単純性を喪失している場合があります (**ST_IsSimple**を参照)、トポロジが保持されていない可能性があり、ポリゴンの結果が不正になる場合があります (**ST_IsValid**参照)。トポロジを保全し、妥当性を確保するには、**ST_SimplifyPreserveTopology**を使います。**ST_CoverageSimplify**でもトポロジ保全と妥当性確保ができます。



Note

この関数は、ポリゴン間で共有される境界は保持されません。保持するには**ST_CoverageSimplify**を使います。



Note

この関数は 3 次元を扱います。第 3 次元は結果に影響を与えます。

Availability: 2.2.0

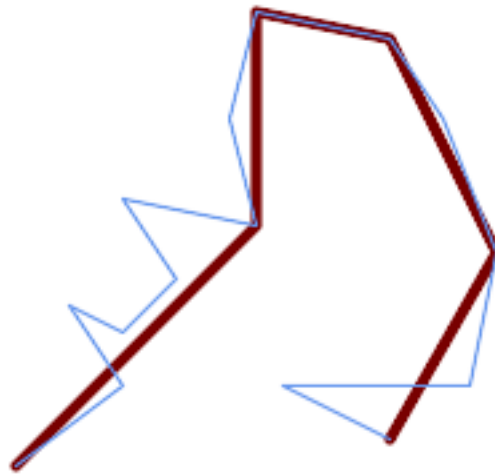
例

LINESTRING は 30 の最小面積の許容値を 30 にして簡略化しています。

```
SELECT ST_AsText(ST_SimplifyVW(geom,30)) simplified
FROM (SELECT 'LINESTRING(5 2, 3 8, 6 20, 7 25, 10 10)>:::geometry AS geom) AS t;
```

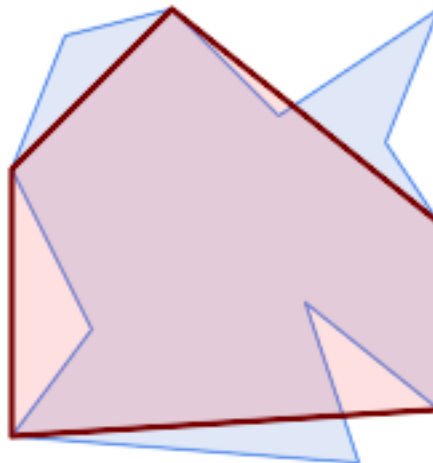
```
LINESTRING(5 2,7 25,10 10)
```

線の単純化。



```
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);
```

ポリゴンの単純化。



```
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);
```

関連情報

[ST_SetEffectiveArea](#), [ST_Simplify](#), [ST_SimplifyPreserveTopology](#), [ST_CoverageSimplify](#), [Topology ST_Simplify](#)

7.14.27 ST_SetEffectiveArea

ST_SetEffectiveArea — Visvalingam-Whyatt アルゴリズムを使って有効範囲となる個々の頂点を置きます。

Synopsis

geometry **ST_SetEffectiveArea**(geometry geom, float threshold = 0, integer set_area = 1);

説明

Visvalingam-Whyatt アルゴリズムから有効範囲となる個々の頂点を置きます。有効範囲はジオメトリの **M** 値として格納されます。任意引数である第 2 引数を使うと、しきい値以上の有効範囲となる頂点だけで構築されるジオメトリを返します。

この関数は、しきい値を使うことでサーバサイド簡略化に使えます。もう一つの任意引数はしきい値を 0 にする際に使用します。この場合、完全なジオメトリを得ますが、クライアントが非常に高速に簡略化するために使う **M** 値として格納している有効範囲を持っています。

(MULTI)LINE と (MULTI)POLYGON とで実際に動作をしますが、どのような種類のジオメトリでも安全に呼ぶことができます。簡略化はオブジェクトごとに行われるので、ジオメトリコレクションでこの関数を呼ぶことができます。



Note

返されるジオメトリは単純性 (**ST_IsSimple**参照) を失うことがあります。



Note

トポロジは保存されているとは限らず、不正なジオメトリを返すことがあります。トポロジを保存するには **ST_SimplifyPreserveTopology** を使います。



Note

出力ジオメトリは、入力時に持っていた **M** 値の情報の全てを失います。



Note

この関数は 3 次元を扱います。第 3 次元は結果に影響を与えます。

Availability: 2.2.0

例

ラインストリングの有効範囲の計算。しきい値を 0 にしているので、入力ジオメトリの全ての頂点が返ります。

```
select ST_AsText(ST_SetEffectiveArea(geom)) all_pts, ST_AsText(ST_SetEffectiveArea(geom,30) ←
) thrshld_30
FROM (SELECT 'LINESTRING(5 2, 3 8, 6 20, 7 25, 10 10)'::geometry geom) As foo;
```

all_pts	thrshld_30
LINESTRING M (5 2 3.402823466385289e+38,3 8 29,6 20 1.5,7 25 49.5,10 10 3.402823466385289e ← +38) LINESTRING M (5 2 3.402823466385289e+38,7 25 49.5,10 10 3.402823466385289e+38)	

(1 row)

関連情報

[ST_SimplifyVW](#)

7.14.28 ST_TriangulatePolygon

ST_TriangulatePolygon — ポリゴンの制約付きドロネー三角分割を計算します。

Synopsis

geometry **ST_TriangulatePolygon**(geometry geom);

説明

ポリゴンの制約付きドロネー三角分割を計算します。穴や MULTIPOLYGON に対応しています。

ポリゴンの「制約付きドロネー三角分割」は、ポリゴンの頂点から形成された三角形の集合で、ポリゴンを実際に覆い、内角和が考えられる三角分割の中で最大になります。この関数によって、ポリゴンの「最高品質」の三角分割が得られます。

Availability: 3.3.0.

GEOS 3.11.0 以上が必要です。

例

正方形の三角分割。

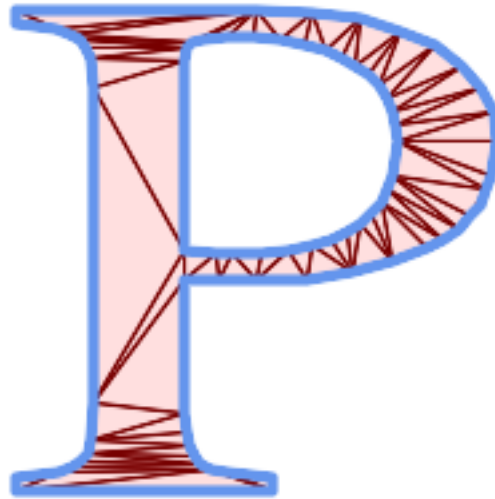
```
SELECT ST_AsText(ST_TriangulatePolygon('POLYGON((0 0, 0 1, 1 1, 1 0, 0 0))'));
```

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

例

文字 P の三角分割。

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

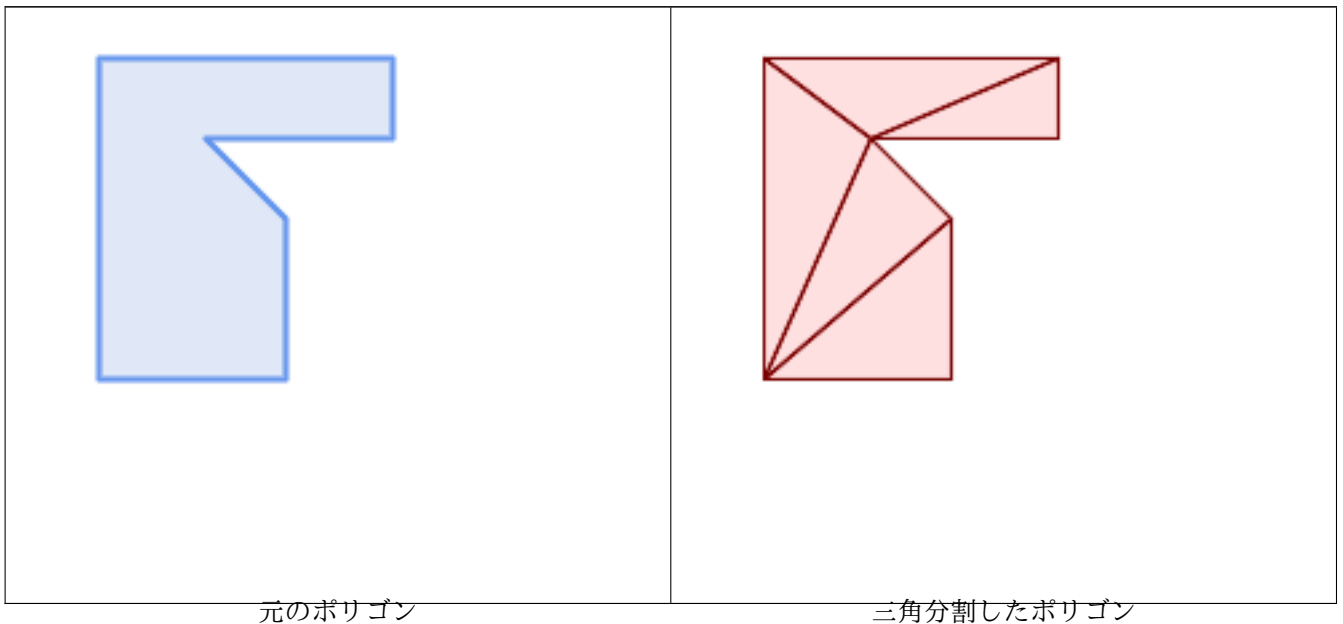


ポリゴンの三角分割

ST_Tessellate と同じ例

```
SELECT ST_AsText(ST_TriangulatePolygon('POLYGON (( 10 190, 10 70, 80 70, 80 130, 50 160, ←
    120 160, 120 190, 10 190 ))'::geometry
));
```

```
GEOMETRYCOLLECTION(POLYGON((50 160,120 190,120 160,50 160)),
    POLYGON((10 70,80 130,80 70,10 70)),
    POLYGON((50 160,10 70,10 190,50 160)),
    POLYGON((120 190,50 160,10 190,120 190)),
    POLYGON((80 130,10 70,50 160,80 130)))
```



元のポリゴン

三角分割したポリゴン

関連情報

[ST_ConstrainedDelaunayTriangles](#), [ST_DelaunayTriangles](#), [ST_Tessellate](#)

7.14.29 ST_VoronoiLines

ST_VoronoiLines — ジオメトリの頂点からボロノイ図のセルを返します。

Synopsis

```
geometry ST_VoronoiLines( geometry geom , float8 tolerance = 0.0 , geometry extend_to = NULL );
```

説明

与えられたジオメトリの頂点から 2 次元 **ボロノイ図** を生成し、ボロノイ図のサイト間の境界を MULTILINESTRING で返します。入力ジオメトリが NULL の場合には NULL を返します。入力ジオメトリの頂点の一つの場合には空ジオメトリを返します。extend_to で作られるエンベロープの面積が 0 の場合には空ジオメトリを返します。

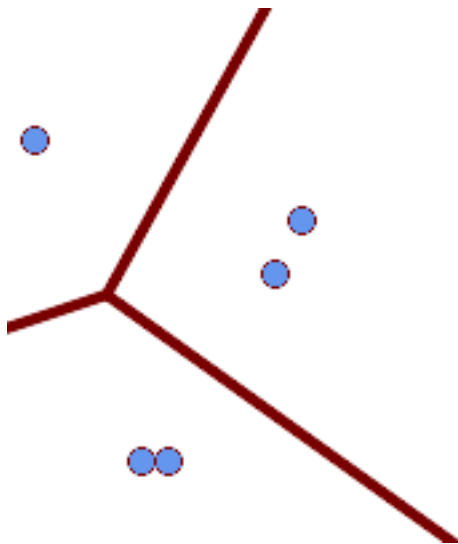
任意パラメータ:

- **tolerance**: 複数の頂点を同一のものと捉える距離。0 でない値を許容距離として与えるとアルゴリズムのロバスト性が改善します (デフォルトは 0.0)。
- **extend_to**: これが与えられている場合には、与えられたジオメトリのエンベロープを覆うように図が拡張されます。デフォルト値以上でないと有効ではありません (デフォルトは NULL で、デフォルトのエンベロープは入力ジオメトリのバウンディングボックスから 50% 拡張したものです)。

GEOS モジュールで実現しています。

Availability: 2.3.0

例



30 単位の許容距離を持つボロノイ図のライン

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

```
MULTILINESTRING((135.555555555556 270,36.8181818181818 92.2727272727273),(36.8181818181818 ←
    92.2727272727273,-110 43.3333333333333),(230 -45.7142857142858,36.8181818181818 ←
    92.2727272727273))
```


関連情報

[ST_DelaunayTriangles](#), [ST_VoronoiPolygons](#)

7.14.30 ST_VoronoiPolygons

ST_VoronoiPolygons — ジオメトリの頂点からボロノイ図のセルを返します。

Synopsis

```
geometry ST_VoronoiPolygons( geometry geom , float8 tolerance = 0.0 , geometry extend_to = NULL );
```

説明

与えられたジオメトリの頂点から 2 次元 [ボロノイ図](#) を生成します。結果は POLYGON の GEOMETRYCOLLECTION で、入力の頂点の範囲以上のエンベロープを覆います。入力ジオメトリが NULL の場合には NULL を返します。入力ジオメトリの頂点が一つの場合には空ジオメトリを返します。extend_to で作られるエンベロープの面積が 0 の場合には空ジオメトリを返します。

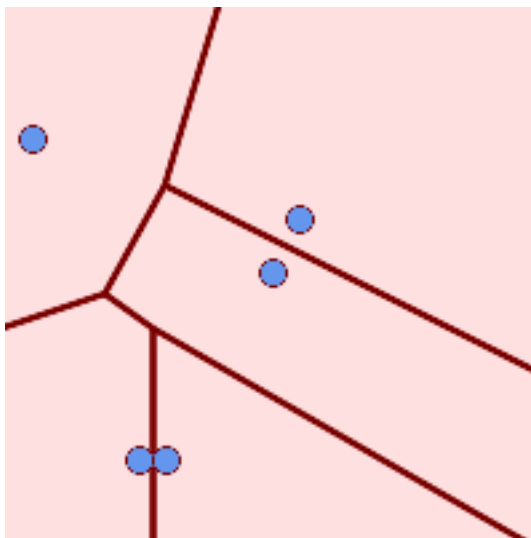
任意パラメータ:

- **tolerance**: 複数の頂点を同一のものと捉える距離。0 でない値を許容距離として与えるとアルゴリズムのロバスト性が改善します (デフォルトは 0.0)。
- **extend_to**: これが与えられている場合には、与えられたジオメトリのエンベロープを覆うように図が拡張されます。デフォルト値以上でないと有効ではありません (デフォルトは NULL で、デフォルトのエンベロープは入力ジオメトリのバウンディングボックスから 50% 拡張したものです)。

GEOS モジュールで実現しています。

Availability: 2.3.0

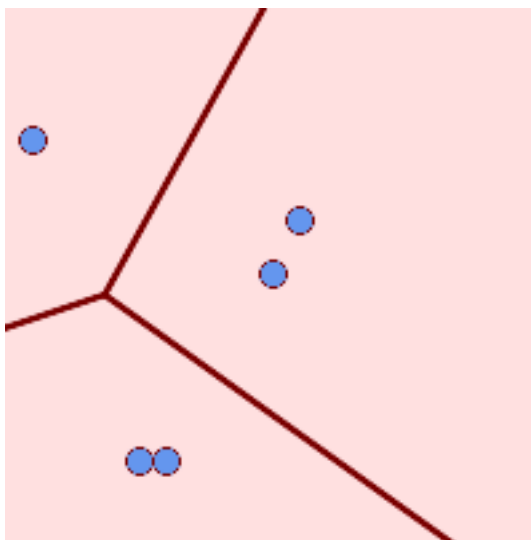
例



ボロノイ図の上にポイントを置いた図

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

GEOMETRYCOLLECTION(POLYGON((-110 270,100.5 270,59.34782608695652
  132.82608695652175,36.81818181818182 92.27272727272727,-110 43.33333333333332,-110 270))
  ,
POLYGON((-110 -90,-110 43.33333333333332,36.81818181818182 92.27272727272727,55
  79.28571428571429,55 -90,-110 -90)),
POLYGON((230 -20.71428571428572,55 79.28571428571429,36.81818181818182
  92.27272727272727,59.34782608695652 132.82608695652175,230 47.50000000000001,230
  -20.71428571428572)),POLYGON((230 -90,55 -90,55 79.28571428571429,230
  -20.71428571428572,230 -90)),
POLYGON((230 270,230 47.50000000000001,59.34782608695652 132.82608695652175,100.5 270,230
  270)))
```



許容距離 30 単位のボロノイ図

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

GEOMETRYCOLLECTION(POLYGON((-110 270,135.55555555555557 270,36.81818181818182
  92.27272727272727,-110 43.33333333333332,-110 270)),
POLYGON((230 270,230 -45.71428571428574,36.81818181818182
  92.27272727272727,135.55555555555557 270,230 270)),
POLYGON((230 -90,-110 -90,-110 43.33333333333332,36.81818181818182 92.27272727272727,230
  -45.71428571428574,230 -90)))
```

関連情報

[ST_DelaunayTriangles](#), [ST_VoronoiLines](#)

7.15 カバレッジ

7.15.1 ST_CoverageInvalidEdges

ST_CoverageInvalidEdges — ポリゴンが妥当なカバレッジの形成に失敗する位置を検索するウィンドウ関数。

Synopsis

```
geometry ST_CoverageInvalidEdges(geometry winset geom, float8 tolerance = 0);
```

説明

ウィンドウパーティション内のポリゴンが妥当なポリゴンカバレッジを形成するかどうかチェックするウィンドウ関数です。個々のポリゴン内の不正なエッジの位置 (あるなら) を示す線形インジケータを返します。

次の条件を満たすなら妥当なポリゴンの集合は妥当なカバレッジです:

- オーバラップなし - ポリゴン同士がオーバラップしない (ポリゴンの内部でインタセクトしない)
- エッジ一致 - 共有エッジに沿った頂点が同一

ウィンドウ関数として、入力ポリゴンごとに値が返されます。妥当性条件の一つ以上に違反するポリゴンについては返り値は、問題のあるエッジを含む MULTILINESTRING です。カバレッジ妥当なポリゴンに対しては NULL を返します。非ポリゴンや空ジオメトリに対しても NULL 値が返ります。

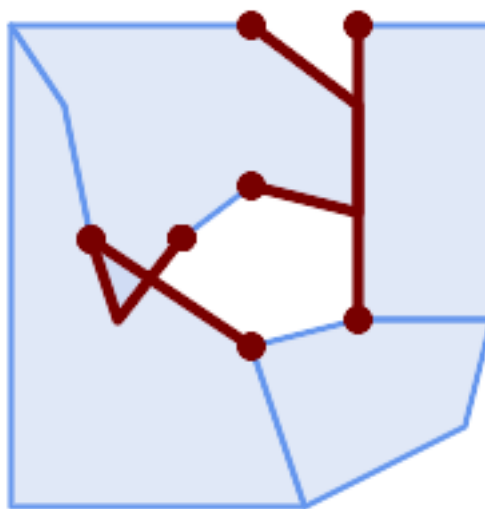
この条件では、周囲のポリゴンがエッジ一致になっている限りは、妥当なカバレッジは穴 (ポリゴン間の隙間) を含むことができます。しかしながら、非常に小さい隙間は、望まれないものとされることがしばしばあります。*tolerance* パラメーターが 0 でない距離で指定されている場合には、小さい隙間を形成するエッジは不正なものとして返されます。

カバレッジ妥当性のチェックがなされたポリゴンも妥当なジオメトリです。ST_IsValidでチェックできます。

Availability: 3.4.0

GEOS 3.12.0 以上が必要です

例



オーバラップと頂点不一致により発生した不正エッジ

```
WITH coverage(id, geom) AS (VALUES
  (1, 'POLYGON ((10 190, 30 160, 40 110, 100 70, 120 10, 10 10, 10 190))'::geometry),
  (2, 'POLYGON ((100 190, 10 190, 30 160, 40 110, 50 80, 74 110.5, 100 130, 140 120, 140 160, 100 190))'::geometry), ←
  (3, 'POLYGON ((140 190, 190 190, 190 80, 140 80, 140 190))'::geometry),
  (4, 'POLYGON ((180 40, 120 10, 100 70, 140 80, 190 80, 180 40))'::geometry)
)
SELECT id, ST_AsText(ST_CoverageInvalidEdges(geom) OVER ())
FROM coverage;
```

id	st_astext
1	LINESTRING (40 110, 100 70)
2	MULTILINESTRING ((100 130, 140 120, 140 160, 100 190), (40 110, 50 80, 74 110.5))
3	LINESTRING (140 80, 140 190)
4	null

This example tests the entire table for coverage validity.

```
SELECT true = ALL (
  SELECT ST_CoverageInvalidEdges(geom) OVER () IS NULL
  FROM coverage
);
```

関連情報

[ST_IsValid](#), [ST_CoverageUnion](#), [ST_CoverageClean](#), [ST_CoverageSimplify](#)

7.15.2 ST_CoverageClean

ST_CoverageClean — きれいな (エッジが合致し、オーバーラップがなく、隙間がない) ポリゴンカバレッジを、きれいでない入力から計算します。

Synopsis

geometry **ST_CoverageClean**(geometry winset geom, float8 gapMaximumWidth = 0, float8 snappingDistance = -1, text overlapMergeStrategy = 'MERGE_LONGEST_BORDER');

説明

妥当なポリゴン系ジオメトリの集合のエッジを調整して、きれいなカバレッジを生成するウィンドウ関数です。きれいにする処理には次を含みます:

- 小さな不一致を取り除き、共有されるエッジに同じノードが打たれるようにするために、頂点とエッジをスナップさせる
- オーバーラップするポリゴンをひとつの親ポリゴンとしてマージする
- 小さい隙間を隣接ポリゴンとしてマージする

`gapMaximumWidth` でポリゴンをマージする隙間の条件を制御します。隙間の幅がこの値以下なら隣接ポリゴンとしてマージします。

`snappingDistance` で頂点とエッジのスナップを制御します。デフォルト (-1) では自動的にスナップ距離が入力の範囲に応じて決定されます。0.0 が指定されるとスナップを行いません。

`overlapMergeStrategy` で、どれぐらいオーバーラップしているポリゴンを親ポリゴンにマージするかが指定されます:

- `MERGE_LONGEST_BORDER` - 共有境界が最長のポリゴンにマージします
- `MERGE_MAX_AREA` - 最大面積のポリゴンにマージします
- `MERGE_MIN_AREA` - 最小面積のポリゴンにマージします
- `MERGE_MIN_INDEX` - 入力インデックス (入力ポリゴンの順番で決まります) の最小値を持つポリゴンにマージします

結果はきれいなポリゴンカバレッジで、`ST_CoverageInvalidEdges`が通り、カバレッジ処理関数の入力にすることができます。

**Note**

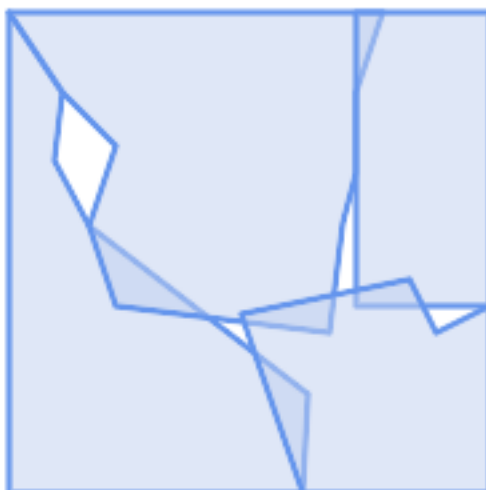
最大隙間幅の判定を助けるために、`gapMaximumWidth = > 0` できれいにし、`ST_CoverageUnion` を使って結果カバレッジを結合することで隙間が計算できます。結合した際にできる穴は、元のデータセットにある隙間に対応しています。隙間幅は、穴をポリゴンとして抽出して、`ST_MaximumInscribedCircle` を実行して得られた半径の 2 倍として計測できます。

Availability: 3.6.0 - GEOS $\geq$ 3.14.0 が必要です

例

This example populates the input table.

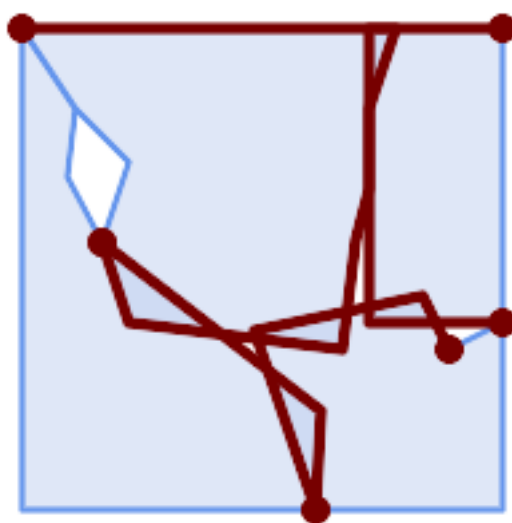
```
CREATE TABLE example AS SELECT * FROM (VALUES
  (1, 'POLYGON ((10 190, 30 160, 27 134.5, 40 110, 122 47, 120 10, 10 10, 10 190))':: geometry),
  (2, 'POLYGON ((150 190, 10 190, 30 160, 50 140, 40 110, 50 80, 130 70, 135 111, 140 130, 140 160, 150 190))'::geometry),
  (3, 'POLYGON ((140 190, 190 190, 190 80, 140 80, 140 190))'::geometry),
  (4, 'POLYGON ((190 10, 120 10, 97 77, 160 90, 170 70, 190 80, 190 10))'::geometry)
) AS v(id, geom);
```



重複部と隙間があるポリゴン

This example shows that the coverage is invalid.

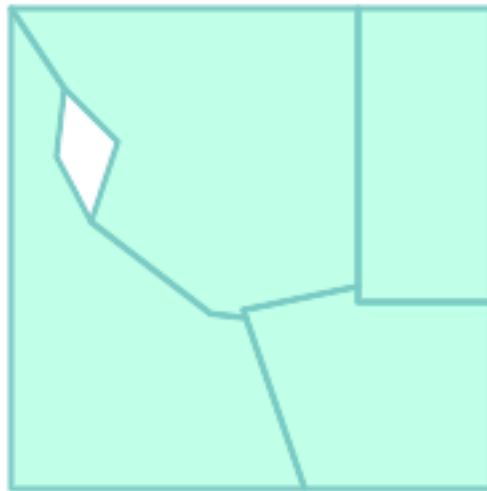
```
SELECT ST_AsText(ST_CoverageInvalidEdges(geom) OVER ())  
FROM example;
```



不正なカバレッジのエッジ

This example cleans the coverage by merging gaps with width less than or equal to 1.

```
CREATE TABLE example_clean AS  
SELECT id, ST_CoverageClean(geom, 1) OVER () AS GEOM  
FROM example;
```



重複部と狭い隙間の消去でポリゴンカバレッジをきれいにします

関連情報

[ST_CoverageInvalidEdges](#), [ST_CoverageUnion](#), [ST_CoverageSimplify](#), [ST_MaximumInscribedCircle](#)

7.15.3 ST_CoverageSimplify

`ST_CoverageSimplify` — ポリゴンカバレッジのエッジを単純化するウィンドウ関数。

Synopsis

```
geometry ST_CoverageSimplify(geometry winset geom, float8 tolerance, boolean simplifyBoundary = true);
```

説明

ポリゴンカバレッジ内でポリゴンのエッジを単純化するウィンドウ関数です。この単純化ではカバレッジのトポロジが保持されます。単純化した出力ポリゴンは共有するエッジに沿って構成され、妥当なカバレッジを形成します。単純化では [Visvalingam-Whyatt algorithm](#) 版を使います。 `tolerance` パラメータは距離の単位を持ち、単純化する三角形面積の平方根とだいたい同じです。

カバレッジの「内部の」エッジ (二つのポリゴンに共有されている) だけを単純化するには、 `simplifyBoundary` パラメーターを `FALSE` に設定します。



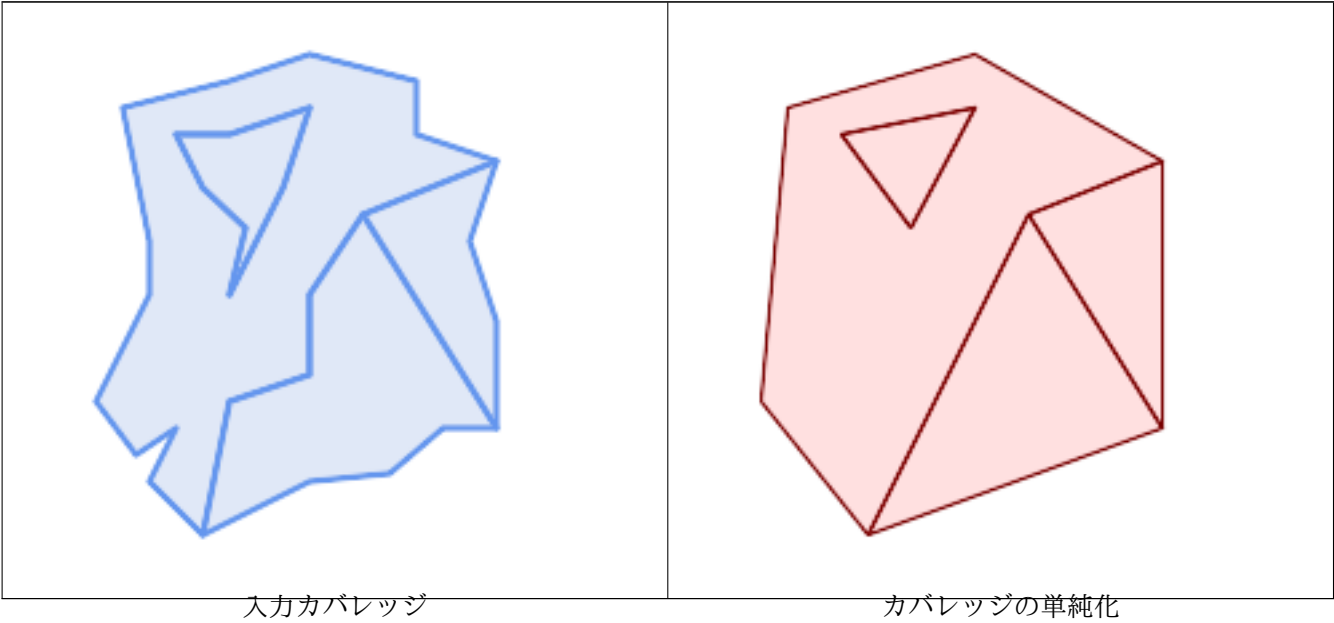
Note

入力が妥当なカバレッジでない場合には、出力に予期しない何か (境界のインタセクト、共有されているように見えるのに分かれている境界) が出現する可能性があります。カバレッジが妥当かどうかを判断するには [ST_CoverageInvalidEdges](#) を使います。

Availability: 3.4.0

GEOS 3.12.0 以上が必要です

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))

関連情報

[ST_CoverageInvalidEdges](#), [ST_CoverageUnion](#), [ST_CoverageClean](#)

7.15.4 ST_CoverageEdges

ST_CoverageEdges — ポリゴンカバレッジの重複の無いエッジを計算します。

Synopsis

```
geometry ST_CoverageEdges(geometry geom, integer edgetype = 0);
geometry ST_CoverageEdges(geometry[] geoms, integer edgetype = 0);
```


説明

ポリゴンカバレッジの重複の無いエッジを **MULTILINESTRING** 表現で返します。ポリゴンカバレッジはオーバーラップが無いポリゴンの集合で、隣接するポリゴンは共有するエッジに沿って一致する頂点を持っています。

edgetype パラメーターはどのエッジを返すかを選択するのに使われます。取りえる値は次の通りです:

- **0 (ALL)** - 全ての重複の無いエッジ (デフォルト)
- **1 (EXTERIOR)** - ポリゴンの外側のエッジのみ (非共有エッジ)
- **2 (INTERIOR)** - ポリゴンの内側のエッジのみ (共有エッジ)

Availability: 3.7.0

GEOS >= 3.15.0 が必要です

例

This example returns all unique edges of a coverage.

```
SELECT ST_AsText(ST_CoverageEdges(ST_GeomFromText('GEOMETRYCOLLECTION(POLYGON((0 0, 10 0, 10 10, 0 10, 0 0)), POLYGON((10 0, 20 0, 20 10, 10 10, 10 0)))')));

MULTILINESTRING((0 0, 10 0), (10 0, 20 0), (20 0, 20 10), (20 10, 10 10), (10 10, 0 10), (0 10, 0 0), (10 0, 10 10))
```

This example returns only the interior shared edges of a coverage.

```
SELECT ST_AsText(ST_CoverageEdges(ST_GeomFromText('GEOMETRYCOLLECTION(POLYGON((0 0, 10 0, 10 10, 0 10, 0 0)), POLYGON((10 0, 20 0, 20 10, 10 10, 10 0)))'), 2));

MULTILINESTRING((10 0, 10 10))
```

関連情報

[ST_CoverageInvalidEdges](#), [ST_CoverageSimplify](#), [ST_CoverageUnion](#)

7.15.5 ST_CoverageUnion

ST_CoverageUnion — 共有しているエッジを除去することでカバレッジを形成するポリゴンの集合の結合を計算します。

Synopsis

geometry **ST_CoverageUnion**(geometry set geom);

説明

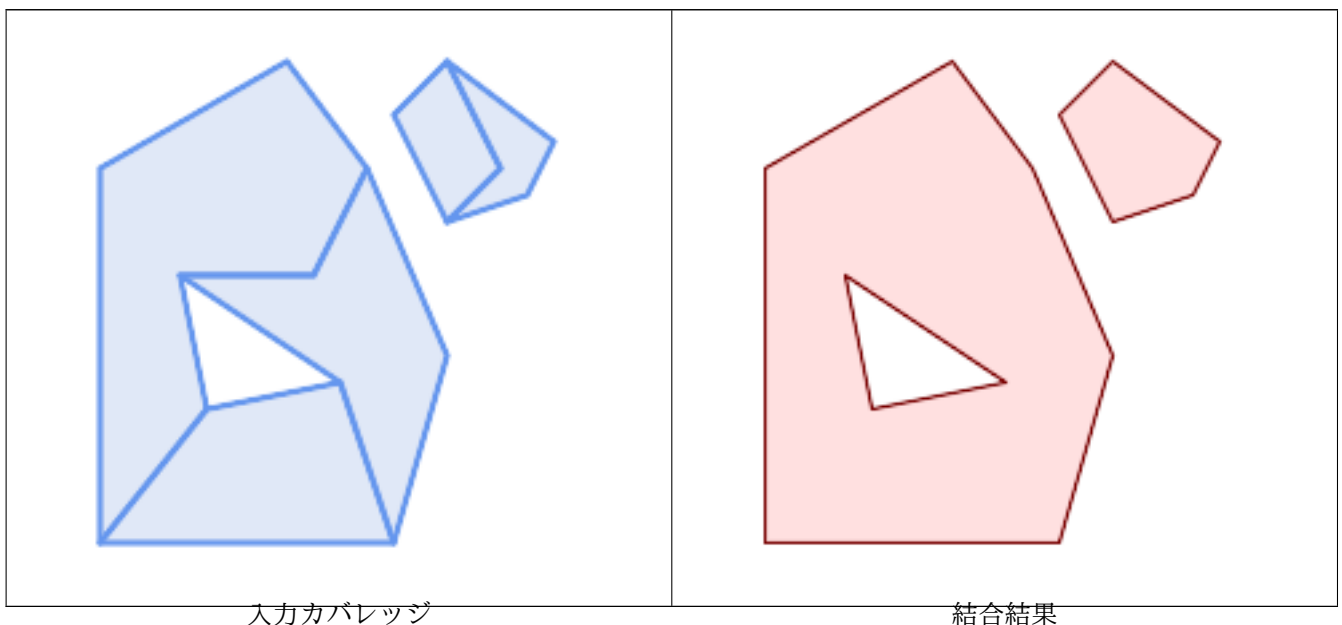
ポリゴンカバレッジを形成するポリゴンの集合の結合を行う集約関数です。結果はカバレッジと同じ領域を覆うポリゴンジオメトリです。この関数は`ST_Union`と同じ結果を返しますが、高速に計算するためにカバレッジ構造を使用します。

**Note**

入力が妥当なカバレッジでない場合には、出力に予期しない何か (マージされていないポリゴンやオーバーラップするポリゴン) が出現する可能性があります。カバレッジが妥当かどうかを判断するには`ST_CoverageInvalidEdges`を使います。

Availability: 3.4.0 - GEOS 3.8.0 以上が必要です

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)) AS wkt
FROM coverage;
```

```
MULTIPOLYGON(((10 150,80 190,110 150,140 80,120 10,10 10,10 150),(50 60,100 70,40 110,50 ←
  60)),((120 170,140 190,180 160,170 140,140 130,120 170)))
```

関連情報

`ST_CoverageInvalidEdges`, `ST_CoverageSimplify`, `ST_CoverageClean`, `ST_Union`

7.16 アフィン変換

7.16.1 ST_Affine

ST_Affine — ジオメトリに 3 次元アフィン変換を適用します。

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);
```

説明

3 次元アフィン変換をジオメトリに適用して移動、回転、拡大縮小を一度に行います。

一つ目の形式では、次のように関数を呼んでいます。

```
ST_Affine(geom, a, b, c, d, e, f, g, h, i, xoff, yoff, zoff)
```

これは次のような変換行列を表現しています。

```
/ a b c xoff \
| d e f yoff |
| g h i zoff |
\ 0 0 0 1 /
```

次のようにも表現できます。

```
x' = a*x + b*y + c*z + xoff
y' = d*x + e*y + f*z + yoff
z' = g*x + h*y + i*z + zoff
```

全ての移動/拡大縮小関数はこのようなアフィン変換を経由しています。

二つ目の形式では、2 次元アフィン変換をジオメトリに適用します。次のように関数を呼んでいます。

```
ST_Affine(geom, a, b, d, e, xoff, yoff)
```

これは次のような変換行列を表現しています。

```
/ 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 /
```

頂点は次のように変換されます。

```
x' = a*x + b*y + xoff
y' = d*x + e*y + yoff
z' = z
```

このメソッドは上述の 3 次元メソッドの特異ケースです。

Enhanced: 2.0.0 多面体サーフェス対応、三角対応、TIN 対応が導入されました。

Availability: 1.1.2 Affine から ST_Affine に名称変更しました。

**Note**

1.3.4 より前では、曲線を含むジオメトリで使用すると、この関数はクラッシュします。これは 1.3.4 以上で訂正されています。

- ✓ この関数は多面体サーフェスに対応しています。
- ✓ この関数は三角形と不規則三角網 (TIN) に対応しています。
- ✓ この関数は 3 次元に対応し、Z 値を削除しません。
- ✓ このメソッドは曲線ストリングと曲線に対応しています。

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, 0, sin(pi()), cos(pi()), 0, 0, 0, 0, 0))
FROM (SELECT ST_GeomFromEWKT('LINESTRING(1 2 3, 1 4 3)') As geom) As foo;
```

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

関連情報

[ST_Rotate](#), [ST_Scale](#), [ST_Translate](#), [ST_TransScale](#)

7.16.2 ST_Rotate

ST_Rotate — ジオメトリを原点について回転させます。

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);
```

説明

ジオメトリを原点について反時計回りに `rotRadians` ぶん回転させます。原点は `POINT` ジオメトリか、`x` と `y` の座標値を指定します。原点を指定しない場合には `POINT(0,0)` について回転させます。

Enhanced: 2.0.0 多面体サーフェス対応、三角対応、TIN 対応が導入されました。

Enhanced: 2.0.0 回転の原点を指定するパラメーターを追加しました。

Availability: 1.1.2 `Rotate` から `ST_Rotate` に名称変更しました。

- ✔ この関数は 3 次元に対応し、Z 値を削除しません。
- ✔ このメソッドは曲線ストリングと曲線に対応しています。
- ✔ この関数は多面体サーフェスに対応しています。
- ✔ この関数は三角形と不規則三角網 (TIN) に対応しています。

例

This example rotates a geometry by 180 degrees.

```
SELECT ST_AsText(ST_Rotate('LINESTRING (50 160, 50 50, 100 50)::geometry, pi()), 6);
```

```
LINESTRING(-50 -160,-50 -50,-100 -50)
```

This example rotates a geometry 30 degrees counter-clockwise around x=50, y=160.

```
SELECT ST_AsText(ST_Rotate('LINESTRING (50 160, 50 50, 100 50)::geometry, pi()/6, 50, 160) ←
```

```
, 6);
```

```
LINESTRING(50 160,105 64.737206,148.30127 89.737206)
```

This example rotates a geometry 60 degrees clockwise around its centroid.

```
SELECT ST_AsText(ST_Rotate(geom, -pi()/3, ST_Centroid(geom)), 4)
```

```
FROM (SELECT 'LINESTRING (50 160, 50 50, 100 50)::geometry AS geom) AS foo;
```

```
LINESTRING(116.4225 130.6721,21.1597 75.6721,46.1597 32.3708)
```

関連情報

[ST_Affine](#), [ST_RotateX](#), [ST_RotateY](#), [ST_RotateZ](#)

7.16.3 ST_RotateX

`ST_RotateX` — ジオメトリを X 軸について回転させます。

Synopsis

geometry **ST_RotateX**(geometry geomA, float rotRadians);

説明

ジオメトリ `geomA` を X 軸について `rotRadians` ぶん回転させます。



Note

`ST_RotateX(geomA, rotRadians)` は `ST_Affine(geomA, 1, 0, 0, 0, cos(rotRadians), -sin(rotRadians), 0, sin(rotRadians), cos(rotRadians), 0, 0, 0)` の短縮版です。

Enhanced: 2.0.0 多面体サーフェス対応、三角対応、TIN 対応が導入されました。

Availability: 1.1.2 1.2.2 で `RotateX` から `ST_RotateX` に名称変更しました。

- ✔ この関数は多面体サーフェスに対応しています。
- ✔ この関数は 3 次元に対応し、Z 値を削除しません。
- ✔ この関数は三角形と不規則三角網 (TIN) に対応しています。

例

This example rotates a line 90 degrees around the X axis.

```
SELECT ST_AsEWKT(ST_RotateX(ST_GeomFromEWKT('LINESTRING(1 2 3, 1 1 1)'), pi()/2));
```

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

関連情報

[ST_Affine](#), [ST_RotateY](#), [ST_RotateZ](#)

7.16.4 ST_RotateY

`ST_RotateY` — ジオメトリを Y 軸について回転させます。

Synopsis

geometry **ST_RotateY**(geometry geomA, float rotRadians);

説明

ジオメトリ `geomA` を Y 軸について `rotRadians` ぶん回転させます。



Note

`ST_RotateY(geomA, rotRadians)` は `ST_Affine(geomA, cos(rotRadians), 0, sin(rotRadians), 0, 1, 0, -sin(rotRadians), 0, cos(rotRadians), 0, 0, 0)` の省略形です。

Availability: 1.1.2 1.2.2 で RotateY から ST_RotateY に名称変更しました。

Enhanced: 2.0.0 多面体サーフェス対応、三角対応、TIN 対応が導入されました。

- ✔ この関数は多面体サーフェスに対応しています。
- ✔ この関数は 3 次元に対応し、Z 値を削除しません。
- ✔ この関数は三角形と不規則三角網 (TIN) に対応しています。

例

This example rotates a line 90 degrees around the Y axis.

```
SELECT ST_AsEWKT(ST_RotateY(ST_GeomFromEWKT('LINESTRING(1 2 3, 1 1 1)'), pi()/2));  
  
LINESTRING(3 2 -1,1 1 -1)
```

関連情報

[ST_Affine](#), [ST_RotateX](#), [ST_RotateZ](#)

7.16.5 ST_RotateZ

ST_RotateZ — ジオメトリを Z 軸について回転させます。

Synopsis

geometry **ST_RotateZ**(geometry geomA, float rotRadians);

説明

ジオメトリ geomA を Z 軸について rotRadians ぶん回転させます。



Note

この関数は ST_Rotate と同じです。



Note

ST_RotateZ(geomA, rotRadians) は SELECT ST_Affine(geomA, cos(rotRadians), -sin(rotRadians), 0, sin(rotRadians), cos(rotRadians), 0, 0, 0, 1, 0, 0, 0) の短縮版です。

Enhanced: 2.0.0 多面体サーフェス対応、三角対応、TIN 対応が導入されました。

Availability: 1.1.2 1.2.2 で RotateZ から ST_RotateZ に名称変更しました。

**Note**

1.3.4 より前では、曲線を含むジオメトリで使用すると、この関数はクラッシュします。これは 1.3.4 以上で訂正されています。

- ✔ この関数は 3 次元に対応し、Z 値を削除しません。
- ✔ このメソッドは曲線ストリングと曲線に対応しています。
- ✔ この関数は多面体サーフェスに対応しています。
- ✔ この関数は三角形と不規則三角網 (TIN) に対応しています。

例

This example rotates a line 90 degrees around the Z axis.

```
SELECT ST_AsEWKT(ST_RotateZ(ST_GeomFromEWKT('LINESTRING(1 2 3, 1 1 1)'), pi()/2));
```

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

This example rotates a curved circle around the Z axis.

```
SELECT ST_AsEWKT(ST_RotateZ(geom, pi()/2))
FROM (SELECT ST_LineToCurve(ST_Buffer(ST_GeomFromText('POINT(234 567)'), 3)) As geom) As ↵
foo;
```

```
CURVEPOLYGON(CIRCULARSTRING(-567 237.00000000000003,-567 231.00000000000003,-567 ↵
237.00000000000003))
```

関連情報

[ST_Affine](#), [ST_RotateX](#), [ST_RotateY](#)

7.16.6 ST_Scale

ST_Scale — 与えた係数でジオメトリを拡大縮小します。

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);
```


説明

対応するパラメータで軸を乗算してジオメトリを新しいサイズに拡大縮小します。

factor パラメータでジオメトリを取る形式では、2 次元、3 次元 (XYZ, XYM)、4 次元のポイントで、全ての対応する次元のスケージングの乗数を設定することができます。**factor** ポイントの欠けた次元については、対応する次元は拡大縮小をしないのと等価になります。

三つのジオメトリを与える形式では、拡大縮小に「仮原点」を渡すことができます。これにより、たとえば、仮原点としてジオメトリの重心を使うといった、「適切な位置での拡大縮小」が可能となります。仮原点を使わない場合には、拡大縮小は実際の原点からの位置で行われるので、全ての座標は拡大縮小係数との積になります。



Note

1.3.4 より前では、曲線を含むジオメトリで使用すると、この関数はクラッシュします。これは 1.3.4 以上で訂正されています。

Availability: 1.1.0

Enhanced: 2.0.0 多面体サーフェス対応、三角対応、TIN 対応が導入されました。

Enhanced: 2.2.0 全ての次元の拡大縮小 (**factor** パラメータ) への対応が導入されました。

Enhanced: 2.5.0 局所原点 (**origin** パラメータ) を使った拡大縮小への対応を導入しました。

- ✔ この関数は多面体サーフェスに対応しています。
- ✔ この関数は 3 次元に対応し、Z 値を削除しません。
- ✔ このメソッドは曲線ストリングと曲線に対応しています。
- ✔ この関数は三角形と不規則三角網 (TIN) に対応しています。
- ✔ この関数は M 値に対応します。

例

This example scales X, Y, and Z values.

```
SELECT ST_AsEWKT(ST_Scale(ST_GeomFromEWKT('LINESTRING(1 2 3, 1 1 1)'), 0.5, 0.75, 0.8));
```

```
LINESTRING(0.5 1.5 2.4,0.5 0.75 0.8)
```

This example scales X and Y values.

```
SELECT ST_AsEWKT(ST_Scale(ST_GeomFromEWKT('LINESTRING(1 2 3, 1 1 1)'), 0.5, 0.75));
```

```
LINESTRING(0.5 1.5 3,0.5 0.75 1)
```

This example scales X, Y, Z, and M values.

```
SELECT ST_AsEWKT(ST_Scale(ST_GeomFromEWKT('LINESTRING(1 2 3 4, 1 1 1 1)'),
```

```
ST_MakePoint(0.5, 0.75, 2, -1)));
```

```
LINESTRING(0.5 1.5 6 -4,0.5 0.75 2 -1)
```

This example scales X and Y values using a false origin.

```
SELECT ST_AsText(ST_Scale('LINESTRING(1 1, 2 2)', 'POINT(2 2)', 'POINT(1 1)::geometry));
```

```
LINESTRING(1 1,3 3)
```

関連情報

[ST_Affine](#), [ST_TransScale](#)

7.16.7 ST_Translate

ST_Translate — 与えられたオフセットでジオメトリを変換します。

Synopsis

```
geometry ST_Translate(geometry g1, float deltax, float deltax);  
geometry ST_Translate(geometry g1, float deltax, float deltax, float deltax);
```

説明

Returns a new geometry whose coordinates are translated delta x,delta y,delta z units. Units are based on the units defined in spatial reference (SRID) for this geometry. M coordinates are preserved.



Note

1.3.4 より前では、曲線を含むジオメトリで使用すると、この関数はクラッシュします。これは 1.3.4 以上で訂正されています。

Availability: 1.2.2

- ✔ この関数は 3 次元に対応し、Z 値を削除しません。
- ✔ このメソッドは曲線ストリングと曲線に対応しています。
- ✔ この関数は M 値に対応します。

例

ポイントを経度 1 度ぶん移動させます。

```
SELECT ST_AsText(ST_Translate(ST_GeomFromText('POINT(-71.01 42.37)',4326),1,0)) AS  
wgs_transgeomtxt;
```

```
POINT(-70.01 42.37)
```

ラインストリングを緯度 1 度ぶん、経度 1/2 度ぶん移動させます。

```
SELECT ST_AsText(ST_Translate(ST_GeomFromText('LINESTRING(-71.01 42.37,-71.11 42.38)',4326),  
1,0.5)) AS wgs_transgeomtxt;
```

```
LINESTRING(-70.01 42.87,-70.11 42.88)
```

3 次元ポイントを移動させます。

```
SELECT ST_AsEWKT(ST_Translate(CAST('POINT(0 0 0)' AS geometry), 5, 12,3));
```

```
POINT(5 12 3)
```

Move points with a measure coordinate

```
SELECT ST_AsEWKT(ST_Translate(ST_GeomFromEWKT('POINTM(0 0 11)'), 5, 12));
```

```
POINTM(5 12 11)
```

```
SELECT ST_AsEWKT(ST_Translate(ST_GeomFromEWKT('POINT(0 0 7 11)'), 5, 12, 3));
```

```
POINT(5 12 10 11)
```

曲線とポイントを移動させます。

```
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))
```

関連情報

[ST_Affine](#), [ST_AsText](#), [ST_GeomFromText](#)

7.16.8 ST_TransScale

ST_TransScale — 与えられた係数とオフセットでジオメトリを変換します。

Synopsis

geometry **ST_TransScale**(geometry geomA, float deltaX, float deltaY, float XFactor, float YFactor);

説明

deltaX と deltaY 引数を使ってジオメトリを移動させ、XFactor,YFactor 引数で拡大縮小させます。2 次元でのみ動作します。



Note

ST_TransScale(geomA, deltaX, deltaY, XFactor, YFactor) は **ST_Affine**(geomA, XFactor, 0, 0, 0, YFactor, 0, 0, 0, 1, deltaX*XFactor, deltaY*YFactor, 0) の短縮版です。



Note

1.3.4 より前では、曲線を含むジオメトリで使用すると、この関数はクラッシュします。これは 1.3.4 以上で訂正されています。

Availability: 1.1.0



この関数は 3 次元に対応し、Z 値を削除しません。



このメソッドは曲線ストリングと曲線に対応しています。

例

```
SELECT ST_AsEWKT(ST_TransScale(ST_GeomFromEWKT('LINESTRING(1 2 3, 1 1 1)'), 0.5, 1, 1, 2));  
  
LINESTRING(1.5 6 3,1.5 4 1)
```

This example buffers a point to approximate a circle, converts it to a curve, translates it by 1,2, and scales it by 3,4.

```
SELECT ST_AsText(ST_TransScale(ST_LineToCurve(ST_Buffer('POINT(234 567)', 3)),1,2,3,4));  
  
CURVEPOLYGON(CIRCULARSTRING(714 2276,696 2276,714 2276))
```

関連情報

[ST_Affine](#), [ST_Translate](#)

7.17 クラスティング関数

7.17.1 ST_ClusterDBSCAN

ST_ClusterDBSCAN — 入力ジオメトリごとに DBSCAN アルゴリズムを使ってクラスタ番号を返すウィンドウ関数です。

Synopsis

integer **ST_ClusterDBSCAN**(geometry winset geom, float8 eps, integer minpoints);

説明

2 次元 [Density-based spatial clustering of applications with noise \(DBSCAN\)](#) アルゴリズムを使って、入力ジオメトリ毎にクラスタ番号を返すウィンドウ関数です。[ST_ClusterKMeans](#) と違い、クラスタ数の指定は不要ですが、代わりに、クラスタを決定するために、期待する [距離\(eps\)](#) と密度 ([minpoints](#)) のパラメータを使います。

入力ジオメトリは、次のいずれかの場合にはクラスタに追加されます:

- 「核」ジオメトリという、入力ジオメトリ (自分を含む) のうち少なくとも [minpoints](#) 個が [eps](#) で指定した [距離](#) の範囲内にあるもの。または
- 「境界」ジオメトリという、核ジオメトリから [eps](#) で指定した [距離](#) の範囲内にあるものです。

境界ジオメトリは、複数のクラスタの核ジオメトリの [eps](#) 距離内に存在する場合があることに注意して下さい。どちらに割り当てても正しいので、境界ジオメトリは有効なクラスタの一つに任意に割り当てられます。この状況では、正しいクラスタが [minpoints](#) ジオメトリより少ないジオメトリで生成される可能性があります。境界ジオメトリの割当の決定性を確実にする (その結果、[ST_ClusterDBSCAN](#) の呼び出しで同じ結果が生成される) には、ウィンドウ定義内で [ORDER BY](#) 節を使います。あいまいなクラスタ割当は他の DBSCAN 実装とは異なるかも知れません。



Note

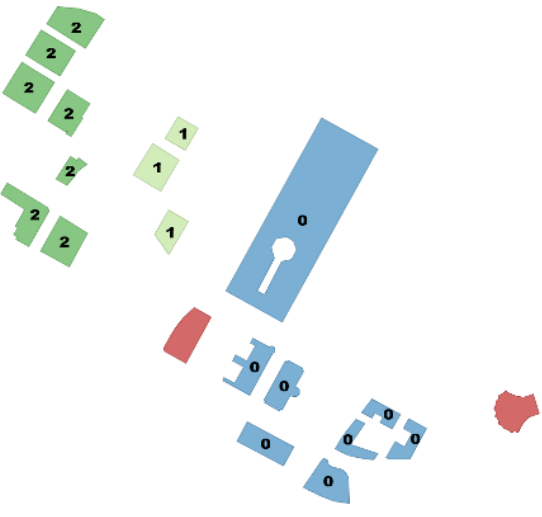
どのクラスタとの結合の基準に合わないジオメトリは、NULL のクラスタ番号が割り当てられます。

Availability: 2.3.0

 このメソッドは曲線ストリングと曲線に対応しています。

例

少なくともクラスタごとに二つ以上のポリゴンが必要な 50 メートル以内のクラスタ。



少なくともクラスタごとに二つ以上のポリゴンがある 50 メートル以内のクラスタ。一つだけなら *cid* は *NULL*。

```
SELECT name, ST_ClusterDBSCAN(geom, eps = > 50, minpoints = > 2) over () AS cid
FROM boston_polys
WHERE name
> '' AND building
> ''
      AND ST_DWithin(
        geom,
        ST_Transform(
          ST_GeomFromText('POINT ←
(-71.04054 42.35141)', 4326),
          26986),
        500
      );
```

bucket	name	
0	Manulife Tower	↔
0	Park Lane Seaport I	↔
0	Park Lane Seaport II	↔
0	Renaissance Boston Waterfront Hotel	↔
0	Seaport Boston Hotel	↔
0	Seaport Hotel & World Trade Center	↔
0	Waterside Place	↔
0	World Trade Center East	↔
1	100 Northern Avenue	↔
1	100 Pier 4	↔
1	The Institute of Contemporary Art	↔
2	101 Seaport	↔
2	District Hall	↔
2	One Marina Park Drive	↔
2	Twenty Two Liberty	↔
2	Vertex	↔
2	Vertex	↔
2	Watermark Seaport	↔
NULL	Blue Hills Bank Pavilion	↔
NULL	World Trade Center West	↔
(20 rows)		

同じクラスタ番号の区画をジオメトリコレクションに合併する例。

```
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;
```

関連情報

[ST_DWithin](#), [ST_ClusterKMeans](#), [ST_ClusterIntersecting](#), [ST_ClusterIntersectingWin](#), [ST_ClusterWithin](#), [ST_ClusterWithinWin](#)

7.17.2 ST_ClusterIntersecting

`ST_ClusterIntersecting` — 入力ジオメトリを接続関係にある集合にクラスタリングする集約関数です。

Synopsis

`geometry[] ST_ClusterIntersecting(geometry set g);`

説明

入力ジオメトリから接続されたものをクラスタとし、クラスタ間は接続されていないように分割して `GEOMETRYCOLLECTION` の配列を返す集約関数です。クラスタ内の個々のジオメトリは同一クラスタ内の他のジオメトリの少なくとも一つとインタセクトし、他のクラスタ内のいずれのジオメトリともインタセクトしません。

Availability: 2.2.0

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;
```

```
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))
```

関連情報

[ST_ClusterIntersectingWin](#), [ST_ClusterWithin](#), [ST_ClusterWithinWin](#)

7.17.3 ST_ClusterIntersectingWin

`ST_ClusterIntersectingWin` — 入力ジオメトリごとに接続された集合にクラスタリングを行い、クラスタ ID を返すウィンドウ関数です。

Synopsis

`integer ST_ClusterIntersectingWin(geometry winset geom);`

説明

インタセクトするジオメトリの接続されたクラスタを構築するウィンドウ関数です。クラスタを離れることなく、クラスタ内の全てのジオメトリを走査することができます。返り値はジオメトリ引数が入っているクラスタの番号です。入力が NULL の場合には NULL を返します。

Availability: 3.4.0

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

関連情報

[ST_ClusterIntersecting](#), [ST_ClusterWithin](#), [ST_ClusterWithinWin](#)

7.17.4 ST_ClusterRelateWin

ST_ClusterRelateWin — ジオメトリが接続するかどうかを判定する空間関係パターンを使って、入力ジオメトリを接続されたジオメトリ集合にクラスタリングして、クラスター識別子を返すウィンドウ関数です。

Synopsis

integer **ST_ClusterRelateWin**(geometry winset geom, text relate_matrix);

説明

交差するジオメトリの接続されたクラスターを構築するウィンドウ関数です。ジオメトリは、クラスターの他のメンバーとの DE9IM 空間関係行列が対で合致する場合には、クラスターに追加されます。この関数によって、境界同士が接触するすべてのオブジェクトのクラスターを構築できますが、単にオーバーラップするものは除外されます。

Availability: 3.7.0

例

このラインストリングのコレクションは **ST_ClusterIntersectingWin** を使って単一のクラスターを形成しますが、**ST_ClusterRelateWin** を使うと、端点でのみ接続する三つのグループにクラスタリングできます。



接続されオーバーラップしているラインストリング

```
CREATE TABLE clusterrelate (
  id serial,
  geom geometry);

INSERT INTO clusterrelate (geom)
VALUES
('LINESTRING(0 0,1 1)'),
('LINESTRING(2 2,1 1)'),
('LINESTRING(0 1,1 0)'),
('LINESTRING(0 1,0 4)'),
('LINESTRING(2 2,2 4)'),
('LINESTRING(1.5 2.5,2.5 3.5)');

SELECT id,
  ST_AsText(geom),
  ST_ClusterRelateWin(geom, '****0****') OVER () AS cluster
FROM clusterrelate;
```

id	st_astext	cluster
1	LINESTRING(0 0,1 1)	0
2	LINESTRING(2 2,1 1)	0
3	LINESTRING(0 1,1 0)	1
4	LINESTRING(0 1,0 4)	1
5	LINESTRING(2 2,2 4)	0
6	LINESTRING(1.5 2.5,2.5 3.5)	2

関連情報

ST_Relate, **ST_ClusterIntersectingWin**, **ST_ClusterWithinWin**

7.17.5 ST_MinimumSpanningTree

ST_MinimumSpanningTree — 最小全域木アルゴリズムを使って入力ジオメトリを接続された木にクラスタリングを行い、入力ジオメトリごとに木 ID を返すウィンドウ関数です。

Synopsis

```
integer ST_MinimumSpanningTree(geometry winset geom);
```

説明

入力ジオメトリの最小接続木 (**Minimum Spanning Tree, MST**) に基づいてラインストリングの接続グラフを構築するウィンドウ関数です。返り値は、引数のジオメトリが参加しているクラスターの番号で、最小木を構成していない場合には 0 となります。

Minimum Spanning Tree (最小全域木) はウィンドウ区画にある全てのジオメトリと接続していて、線長が最小となる木です。グラフが完全には接続されていない (例: ジオメトリ間が無限距離になっている) 場合には、最小全域フォレストを生成します。これはそれぞれの接続要素に一意の木 ID が付きます。

Availability: 3.7.0

GEOS >= 3.15.0 が必要です

例

```
SELECT id, ST_AsText(geom), ST_MinimumSpanningTree(geom) OVER () AS msp
FROM (VALUES
  (1, 'LINESTRING(0 0,1 0)::geometry),
  (2, 'LINESTRING(0 0,0 1)::geometry),
  (3, 'LINESTRING(1 1,0 1)::geometry),
  (4, 'LINESTRING(1 1,1 0)::geometry),
  (5, 'LINESTRING(0 0,1 1)::geometry),
  (6, 'LINESTRING(1 0,0 1)::geometry)
) AS t(id, geom);
```

id	st_astext	msp
1	LINESTRING(0 0,1 0)	1
2	LINESTRING(0 0,0 1)	1
3	LINESTRING(1 1,0 1)	1
4	LINESTRING(1 1,1 0)	0
5	LINESTRING(0 0,1 1)	0
6	LINESTRING(1 0,0 1)	0

関連情報

[ST_ClusterDBSCAN](#), [ST_ClusterKMeans](#), [ST_ClusterIntersectingWin](#)

7.17.6 ST_ClusterKMeans

ST_ClusterKMeans — 入力ジオメトリごとに k 平均法アルゴリズムを使ってクラスタ番号を返すウィンドウ関数です。

Synopsis

```
integer ST_ClusterKMeans( geometry winset geom , integer k , float8 max_radius );
```

説明

入力ジオメトリごとの **K 平均法** のクラスタ番号を返します。クラスタリングに使われる距離は、2 次元ジオメトリでは重心間の距離、3 次元ジオメトリではバウンディングボックスの中心点間の距離です。POINT 入力では、M 値は入力の重みとして扱われ、0 より大きくなければなりません。

`max_radius` が設定されている場合には、`ST_ClusterKMeans` は、出力クラスタが `max_radius` より半径が大きいクラスタを作らなくなるので、`k` より多いクラスタを生成します。到達可能性分析に使用します。

Enhanced: 3.2.0 `max_radius` パラメータに対応しました

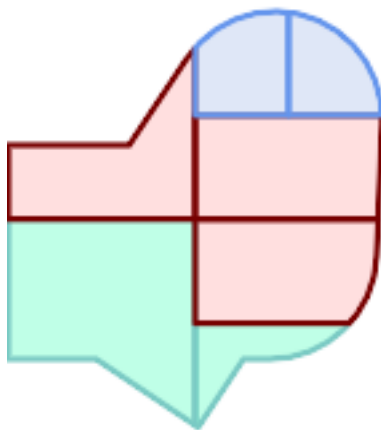
Enhanced: 3.1.0 3 次元ジオメトリと重みに対応するようになりました

Availability: 2.3.0

例

例としてダミーの区画の集合を生成します:

```
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;
```



クラスタ番号 (*cid*) による色付けを施した区画

```
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...

タイプ別による区画の分割:

```
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

例: 3次元クラスタリングと重み付けを使った、事前集計した地球規模の人口データセットのクラスタリング。
[Kontur Population Data](#)に基づいて、中心から 3000km 以下となる、少なくとも 20 の地域が識別されます:

```
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;
```



世界人口を上記仕様でクラスタリングして、**46** のクラスタを得ました。人口集中地域 (ニューヨーク、モスクワ) にクラスタの中心が来ます。グリーンランドは一つのクラスタです。日付変更線をまたぐクラスタがあります。クラスタのエッジは地球の曲線に従います。

関連情報

[ST_ClusterDBSCAN](#), [ST_ClusterIntersectingWin](#), [ST_ClusterWithinWin](#), [ST_ClusterIntersecting](#), [ST_Cluster](#), [ST_Subdivide](#), [ST_Force3D](#), [ST_Force4D](#),

7.17.7 ST_ClusterWithin

ST_ClusterWithin — 分離距離でジオメトリのクラスタリングを行う集約関数です。

Synopsis

```
geometry[] ST_ClusterWithin(geometry set g, float8 distance);
```

説明

GEOMETRYCOLLECTION の配列を返す集約関数で、個々の要素は入力ジオメトリを含むクラスタです。クラスタリングは入力ジオメトリを、それぞれのジオメトリが、同じクラスタ内の他のジオメトリのうち少なくとも一つとの距離が *distance* 以内となる集合に分割します。距離は SRID の単位を取るデカルト距離です。

ST_ClusterWithin は、[ST_ClusterDBSCAN](#) を `minpoints => 0` で実行するのと同じです。

Availability: 2.2.0



このメソッドは曲線ストリングと曲線に対応しています。

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;
```

```
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))
```

関連情報

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

7.17.8 ST_ClusterWithinWin

ST_ClusterWithinWin — 入力ジオメトリごとに分離距離を使ったクラスタリングを行い、クラスタ ID を返すウィンドウ関数です。

Synopsis

```
integer ST_ClusterWithinWin(geometry winset geom, float8 distance);
```

説明

入力ジオメトリ毎のクラスタ番号を返すウィンドウ関数です。クラスタリングはジオメトリを、それぞれのジオメトリが、同じクラスタ内の他のジオメトリのうち少なくとも一つとの距離が **distance** 以内となる集合に分割します。距離は SRID の単位を取るデカルト距離です。

ST_ClusterWithinWin は、**ST_ClusterDBSCAN** を `minpoints => 0` で実行するのと同じです。

Availability: 3.4.0

 このメソッドは曲線ストリングと曲線に対応しています。

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

関連情報

ST_ClusterWithin, **ST_ClusterDBSCAN**, **ST_ClusterIntersecting**, **ST_ClusterIntersectingWin**,

7.18 バウンディングボックス関数

7.18.1 Box2D

Box2D — ジオメトリの 2 次元範囲を表現する BOX2D を返します。

Synopsis

box2d **Box2D**(geometry geom);

説明

ジオメトリの 2 次元範囲を表現する **box2d** を返します。

Enhanced: 2.0.0 多面体サーフェス対応、三角対応、TIN 対応が導入されました。



このメソッドは曲線ストリングと曲線に対応しています。



この関数は多面体サーフェスに対応しています。



この関数は三角形と不規則三角網 (TIN) に対応しています。

例

```
SELECT Box2D(ST_GeomFromText('LINESTRING(1 2, 3 4, 5 6)'));
```

```
BOX(1 2,5 6)
```

```
SELECT Box2D(ST_GeomFromText('CIRCULARSTRING(220268 150415,220227 150505,220227 150406)'));
```

```
BOX(220186.99512189245 150406,220288.24878054656 150506.12682932706)
```

関連情報

Box3D, **ST_GeomFromText**

7.18.2 Box3D

Box3D — ジオメトリの 3 次元範囲を表現する BOX3D を返します。

Synopsis

box3d **Box3D**(geometry geom);

説明

ジオメトリの 3 次元範囲を表現する **box3d** を返します。

Enhanced: 2.0.0 多面体サーフェス対応、三角対応、TIN 対応が導入されました。



このメソッドは曲線ストリングと曲線に対応しています。



この関数は多面体サーフェスに対応しています。



この関数は三角形と不規則三角網 (TIN) に対応しています。



この関数は 3 次元に対応し、Z 値を削除しません。

例

```
SELECT Box3D(ST_GeomFromEWKT('LINESTRING(1 2 3, 3 4 5, 5 6 5)'));
```

```
BOX3D(1 2 3,5 6 5)
```

```
SELECT Box3D(ST_GeomFromEWKT('CIRCULARSTRING(220268 150415 1,220227 150505 1,220227 150406 1)'));
```

```
BOX3D(220186.99512189245 150406 1,220288.24878054656 150506.12682932706 1)
```

関連情報

[Box2D](#), [ST_GeomFromEWKT](#)

7.18.3 ST_EstimatedExtent

ST_EstimatedExtent — 空間テーブルの推定範囲を返します。

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);

説明

空間テーブルの推定範囲を **box2d** で返します。スキーマが指定されていない場合には現在のスキーマが使われます。推定範囲はジオメトリカラムの統計情報から取得します。これは通常は、**ST_Extent** または **ST_3DExtent** を使ってテーブルの確定範囲を計算するより、ずっと早く計算できます。

指定が無い場合には、子テーブル (INHERITS を使って作られたテーブル) が存在するなら、そこから得られた統計情報も使用します。parent_only が TRUE の場合には、パラメータで与えられたテーブルの統計情報だけが使用され、子テーブルの統計情報は無視されます。

PostgreSQL 8.0.0 以上では VACUUM ANALYZE で統計情報を集め、結果として得られる範囲は実際の 95% 程度です。PostgreSQL 8.0.0 より前の場合には、update_geometry_stats() で統計情報が集められ、結果の範囲は確定値です。



Note

統計情報が無い (空のテーブルまたは ANALYZE を実行していない) 場合には、この関数は NULL を返します。1.5.4 より前では、代わりに例外が投げられていました。



Note

テーブルや名前空間の名前のうち特殊文字や引用符をエスケープすると、特別な処理が必要になる可能性があります。あるユーザは「スキーマとテーブルでは、識別子エスケープ規則を使って、二重引用符を付けた文字列を生成して、その後最初と最後の二重引用符を削ります。ジオメトリカラムについてはそのまま渡します」

Availability: 1.0.0

Changed: 2.1.0 2.0.x までは ST_Estimated_Extent と呼ばれていました。



このメソッドは曲線ストリングと曲線に対応しています。

例

```
SELECT ST_EstimatedExtent('ny', 'edges', 'geom');
BOX(-8877653 4912316,-8010225.5 5589284)

SELECT ST_EstimatedExtent('feature_poly', 'geom');
BOX(-124.659652709961 24.6830825805664,-67.7798080444336 49.0012092590332)
```

関連情報

[ST_Extent](#), [ST_3DExtent](#)

7.18.4 ST_Expand

ST_Expand — 他のバウンディングボックスまたはジオメトリから拡張されたバウンディングボックスを返します。

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);
```

説明

入力のパウンディングボックスから拡張したバウンディングボックスを返します。一つの距離を指定して全ての方向に拡張するものと、方向ごとに距離を指定して拡張するものがあります。倍精度浮動小数点を使います。距離に関するクエリで使いますし、空間インデックスの利点を得るために行うクエリへのフィルタを追加するために使います。

ST_Expand は、ジオメトリを受け付けジオメトリを返す版だけでなく、[box2d](#)や[box3d](#)のデータ型を受け付けて返す形式もあります。

距離の単位は入力の空間参照系の単位です。

ST_Expand は[ST_Buffer](#)と似ていますが、ST_Buffer ではジオメトリを全ての方向に拡張するのに対して、ST_Expand はバウンディングボックスを各軸に沿って拡張する点異なります。

Note



1.3 より前の版では、インデックス化可能な距離クエリでは ST_Expand と[ST_Distance](#)とが組み合わせて使われていました。たとえば geom && ST_Expand('POINT(10 20)', 10) AND ST_Distance(geom, 'POINT(10 20)') < 10 といったようになります。これはより単純でより効率的な[ST_DWithin](#)に置き換わりしました。

Availability: 1.5.0 出力を float4 座標値から倍精度に変更しました。

Enhanced: 2.0.0 多面体サーフェス対応、三角対応、TIN 対応が導入されました。

Enhanced: 2.3.0 異なる次元の異なる量によるボックスの拡張に対応するようになりました。



この関数は多面体サーフェスに対応しています。



この関数は三角形と不規則三角網 (TIN) に対応しています。

例



Note

次に示す例では、メートル単位の投影法である米国ナショナルアトラス正積図法 (SRID=2163) を使っています。

```
--10 meter expanded box around bbox of a linestring
SELECT CAST(ST_Expand(ST_GeomFromText('LINESTRING(2312980 110676,2312923 110701,2312892 110714)', 2163),10) AS box2d);
```

```
BOX(2312882 110666,2312990 110724)
```

This example expands a 3D box by 10 meters.

```
SELECT ST_Expand(CAST('BOX3D(778783 2951741 1,794875 2970042.61545891 10)' AS box3d),10)
```

```
BOX3D(778773 2951731 -9,794885 2970052.61545891 20)
```

This example renders the text representation of a 10-meter expanded box around a point geometry.

```
SELECT ST_AsEWKT(ST_Expand(ST_GeomFromEWKT('SRID=2163;POINT(2312980 110676)'),10));
```

```
SRID=2163;POLYGON((2312970 110666,2312970 110686,2312990 110686,2312990 110666,2312970 110666))
```

関連情報

[ST_Buffer](#), [ST_DWithin](#), [ST_SRID](#)

7.18.5 ST_Extent

ST_Extent — ジオメトリのバウンディングボックスを返す集約関数です。

Synopsis

```
box2d ST_Extent(geometry set geomfield);
```

説明

ジオメトリの集合の**box2d**バウンディングボックスを返す集約関数です。

バウンディングボックスの座標は、入力ジオメトリの空間参照系に従います。

ST_Extent は Oracle Spatial/Locator の SDO_AGGR_MBR と似た発想のものです。

**Note**

ST_Extent は、3 次元ジオメトリであっても X 値と Y 値だけを持つボックスを返します。Z 値も得たいなら**ST_3DExtent**を使います。

**Note**

返された box3d 値は SRID を含みません。**ST_SetSRID**で SRID メタデータを持つジオメトリに変換して下さい。SRID は入力ジオメトリと同じです。

Enhanced: 2.0.0 多面体サーフェス対応、三角対応、TIN 対応が導入されました。



この関数は多面体サーフェスに対応しています。



この関数は三角形と不規則三角網 (TIN) に対応しています。

例

**Note**

次に示す例では、フィート単位のマサチューセッツ州平面 (SRID=2249) を使っています。

```
SELECT ST_Extent(geom) AS bextent FROM sometable;
```

```
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

This example converts the extent back into a geometry and renders its extended text representation.

```
SELECT ST_SetSRID(ST_Extent(geom),2249) AS bextent FROM sometable;
```

```
SRID=2249;POLYGON((739651.875 2908247.25,739651.875 2970042.75,794875.8125 2970042.75,
794875.8125 2908247.25,739651.875 2908247.25))
```

関連情報

[ST_EstimatedExtent](#), [ST_3DExtent](#), [ST_SetSRID](#)

7.18.6 ST_3DExtent

ST_3DExtent — ジオメトリの 3 次元バウンディングボックスを返す集約関数です。

Synopsis

```
box3d ST_3DExtent(geometry set geomfield);
```

説明

ジオメトリ集合の **box3d** (Z 値を持つ) を返す集約関数です。

バウンディングボックスの座標は、入力ジオメトリの空間参照系に従います。



Note

返された box3d 値は SRID を含みません。ST_SetSRID で SRID メタデータを持つジオメトリに変換して下さい。SRID は入力ジオメトリと同じです。

Enhanced: 2.0.0 多面体サーフェス対応、三角対応、TIN 対応が導入されました。

Changed: 2.0.0 以前の版では ST_Extent3D と呼ばれていました。

- ✔ この関数は 3 次元に対応し、Z 値を削除しません。
- ✔ このメソッドは曲線ストリングと曲線に対応しています。
- ✔ この関数は多面体サーフェスに対応しています。
- ✔ この関数は三角形と不規則三角網 (TIN) に対応しています。

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;
```

```
BOX3D(1 1 0,3 2 2)
```

This example returns the extent of several elevated CircularStrings.

```
SELECT ST_3DExtent(foo.geom) As b3extent
FROM (SELECT ST_Translate(ST_Force3DZ(ST_LineToCurve(ST_Buffer(ST_Point(x,y),1))),0,0,z) As geom
      FROM generate_series(1,3) As x
      CROSS JOIN generate_series(1,2) As y
      CROSS JOIN generate_series(0,2) As Z) As foo;
```

```
BOX3D(0 0 0,4 3 2)
```

関連情報

[ST_Extent](#), [ST_Force3DZ](#), [ST_SetSRID](#)

7.18.7 ST_MakeBox2D

ST_MakeBox2D — 二つの 2 次元のポイントジオメトリで定義される BOX2D を生成します。

Synopsis

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

説明

二つのポイントジオメトリで定義される **box2d** を生成します。これは範囲のクエリを実行する時に便利です。

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)
```

関連情報

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

7.18.8 ST_3DMakeBox

ST_3DMakeBox — 二つの 3 次元のポイントジオメトリで定義される BOX3D を生成します。

Synopsis

box3d **ST_3DMakeBox**(geometry point3DLowLeftBottom, geometry point3DUpRightTop);

説明

二つの 3 次元ポイントジオメトリで定義される **box3d** が生成されます。



この関数は 3 次元に対応し、Z 値を削除しません。

Changed: 2.0.0 以前の版では ST_MakeBox3D と呼ばれていました。

例

```
SELECT ST_3DMakeBox(ST_MakePoint(-989502.1875, 528439.5625, 10),  
    ST_MakePoint(-987121.375, 529933.1875, 10))
```

```
BOX3D(-989502.1875 528439.5625 10,-987121.375 529933.1875 10)
```

関連情報

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

7.18.9 ST_XMax

ST_XMax — 2 次元または 3 次元のバウンディングボックスまたはジオメトリの X の最大値を返します。

Synopsis

```
float ST_XMax(box3d aGeomorBox2DorBox3D);
```

説明

2 次元または 3 次元のバウンディングボックスまたはジオメトリの X の最大値を返します。



Note

この関数は box3d だけをパラメータに取るよう定義されていますが、自動キャストを介して box2d やジオメトリ値でも動作します。しかしながら、ジオメトリや box2d の文字列表現は自動キャストを行わないため受け付けません。

- ✔ この関数は 3 次元に対応し、Z 値を削除しません。
- ✔ このメソッドは曲線ストリングと曲線に対応しています。

例

```
SELECT ST_XMax('BOX3D(1 2 3, 4 5 6)');
```

```
4
```

```
SELECT ST_XMax(ST_GeomFromText('LINESTRING(1 3 4, 5 6 7)'));
```

```
5
```

```
SELECT ST_XMax(CAST('BOX(-3 2, 3 4)' AS box2d));
```

```
3
```

Observe THIS DOES NOT WORK because it will try to auto-cast the string representation to a BOX3D.

```
SELECT ST_XMax('LINESTRING(1 3, 5 6)');
```

```
ERROR: BOX3D parser - doesn't start with BOX3D(
```

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

```
220288.248780547
```

関連情報

[ST_XMin](#), [ST_YMin](#), [ST_YMax](#), [ST_ZMin](#), [ST_ZMax](#), [ST_MMin](#), [ST_MMax](#)

7.18.10 ST_XMin

ST_XMin — 2 次元または 3 次元のバウンディングボックスまたはジオメトリの X の最小値を返します。

Synopsis

```
float ST_XMin(box3d aGeomorBox2DorBox3D);
```

説明

2 次元または 3 次元のバウンディングボックスまたはジオメトリの X の最小値を返します。



Note

この関数は BOX3D に対してのみ定義されていますが、自動キャストによって BOX2D やジオメトリ値でも動作します。しかしながら、ジオメトリまたは BOX2D の文字列表現は、自動キャストを行わないため、受け付けません。



この関数は 3 次元に対応し、Z 値を削除しません。



このメソッドは曲線ストリングと曲線に対応しています。

例

```
SELECT ST_XMin('BOX3D(1 2 3, 4 5 6)');
```

```
1
```

```
SELECT ST_XMin(ST_GeomFromText('LINESTRING(1 3 4, 5 6 7)'));
```

```
1
```

```
SELECT ST_XMin(CAST('BOX(-3 2, 3 4)' AS box2d));
```

-3

Observe THIS DOES NOT WORK because it will try to auto-cast the string representation to a BOX3D.

```
SELECT ST_XMin('LINESTRING(1 3, 5 6)');
```

```
ERROR: BOX3D parser - doesn't start with BOX3D(
```

```
SELECT ST_XMin(ST_GeomFromEWKT('CIRCULARSTRING(220268 150415 1,220227 150505 2,220227 150406 3)'));
```

```
220186.995121892
```

関連情報

[ST_XMax](#), [ST_YMin](#), [ST_YMax](#), [ST_ZMin](#), [ST_ZMax](#), [ST_MMin](#), [ST_MMax](#)

7.18.11 ST_YMax

ST_YMax — 2 次元または 3 次元のバウンディングボックスまたはジオメトリの Y の最大値を返します。

Synopsis

```
float ST_YMax(box3d aGeomorBox2DorBox3D);
```

説明

2 次元または 3 次元のバウンディングボックスまたはジオメトリの Y の最大値を返します。



Note

この関数は BOX3D に対してのみ定義されていますが、自動キャストによって BOX2D やジオメトリ値でも動作します。しかしながら、ジオメトリまたは BOX2D の文字列表現は、自動キャストを行わないため、受け付けません。



この関数は 3 次元に対応し、Z 値を削除しません。



このメソッドは曲線ストリングと曲線に対応しています。

例

```
SELECT ST_YMax('BOX3D(1 2 3, 4 5 6)');
```

```
5
```

```
SELECT ST_YMax(ST_GeomFromText('LINESTRING(1 3 4, 5 6 7)'));
```

```
6
```

```
SELECT ST_YMax(CAST('BOX(-3 2, 3 4)' AS box2d));
```

```
4
```

Observe THIS DOES NOT WORK because it will try to auto-cast the string representation to a BOX3D.

```
SELECT ST_YMax('LINESTRING(1 3, 5 6)');
```

```
ERROR: BOX3D parser - doesn't start with BOX3D(
```

```
SELECT ST_YMax(ST_GeomFromEWKT('CIRCULARSTRING(220268 150415 1,220227 150505 2,220227 150406 3)'));
```

```
150506.126829327
```

関連情報

[ST_XMin](#), [ST_XMax](#), [ST_YMin](#), [ST_ZMin](#), [ST_ZMax](#), [ST_MMin](#), [ST_MMax](#)

7.18.12 ST_YMin

ST_YMin — 2 次元または 3 次元のバウンディングボックスまたはジオメトリの Y の最小値を返します。

Synopsis

```
float ST_YMin(box3d aGeomorBox2DorBox3D);
```

説明

2 次元または 3 次元のバウンディングボックスまたはジオメトリの Y の最小値を返します。



Note

この関数は BOX3D に対してのみ定義されていますが、自動キャストによって BOX2D やジオメトリ値でも動作します。しかしながら、ジオメトリまたは BOX2D の文字列表現は、自動キャストを行わないため、受け付けません。



この関数は 3 次元に対応し、Z 値を削除しません。



このメソッドは曲線ストリングと曲線に対応しています。

例

```
SELECT ST_YMin('BOX3D(1 2 3, 4 5 6)');
```

```
2
```

```
SELECT ST_YMin(ST_GeomFromText('LINESTRING(1 3 4, 5 6 7)'));
```


3

```
SELECT ST_YMin(CAST('BOX(-3 2, 3 4)' AS box2d));
```

2

Observe THIS DOES NOT WORK because it will try to auto-cast the string representation to a BOX3D.

```
SELECT ST_YMin('LINESTRING(1 3, 5 6)');
```

```
ERROR: BOX3D parser - doesn't start with BOX3D(
```

```
SELECT ST_YMin(ST_GeomFromEWKT('CIRCULARSTRING(220268 150415 1,220227 150505 2,220227 150406 3)'));
```

150406

関連情報

[ST_GeomFromEWKT](#), [ST_XMin](#), [ST_XMax](#), [ST_YMax](#), [ST_ZMin](#), [ST_ZMax](#), [ST_MMin](#), [ST_MMax](#)

7.18.13 ST_ZMax

ST_ZMax — 2 次元または 3 次元のバウンディングボックスまたはジオメトリの Z の最大値を返します。

Synopsis

```
float ST_ZMax(box3d aGeomorBox2DorBox3D);
```

説明

2 次元または 3 次元のバウンディングボックスまたはジオメトリの Z の最大値を返します。



Note

この関数は BOX3D に対してのみ定義されていますが、自動キャストによって BOX2D やジオメトリ値でも動作します。しかしながら、ジオメトリまたは BOX2D の文字列表現は、自動キャストを行わないため、受け付けません。



この関数は 3 次元に対応し、Z 値を削除しません。



このメソッドは曲線ストリングと曲線に対応しています。

例

```
SELECT ST_ZMax('BOX3D(1 2 3, 4 5 6)');
```

6

```
SELECT ST_ZMax(ST_GeomFromEWKT('LINESTRING(1 3 4, 5 6 7)'));
```

7

```
SELECT ST_ZMax('BOX3D(-3 2 1, 3 4 1)');
```

1

Observe THIS DOES NOT WORK because it will try to auto-cast the string representation to a BOX3D.

```
SELECT ST_ZMax('LINESTRING(1 3 4, 5 6 7)');
```

```
ERROR: BOX3D parser - doesn't start with BOX3D(
```

```
SELECT ST_ZMax(ST_GeomFromEWKT('CIRCULARSTRING(220268 150415 1,220227 150505 2,220227 150406 3)'));
```

3

関連情報

[ST_GeomFromEWKT](#), [ST_XMin](#), [ST_XMax](#), [ST_YMin](#), [ST_YMax](#), [ST_ZMin](#), [ST_MMin](#), [ST_MMax](#)

7.18.14 ST_ZMin

ST_ZMin — 2 次元または 3 次元のバウンディングボックスまたはジオメトリの Z の最小値を返します。

Synopsis

```
float ST_ZMin(box3d aGeomorBox2DorBox3D);
```

説明

2 次元または 3 次元のバウンディングボックスまたはジオメトリの Z の最小値を返します。



Note

この関数は BOX3D に対してのみ定義されていますが、自動キャストによって BOX2D やジオメトリ値でも動作します。しかしながら、ジオメトリまたは BOX2D の文字列表現は、自動キャストを行わないため、受け付けません。



この関数は 3 次元に対応し、Z 値を削除しません。



このメソッドは曲線ストリングと曲線に対応しています。

例

```
SELECT ST_ZMin('BOX3D(1 2 3, 4 5 6)');
```

3

```
SELECT ST_ZMin(ST_GeomFromEWKT('LINESTRING(1 3 4, 5 6 7)'));
```

4

```
SELECT ST_ZMin('BOX3D(-3 2 1, 3 4 1)');
```

1

Observe THIS DOES NOT WORK because it will try to auto-cast the string representation to a BOX3D.

```
SELECT ST_ZMin('LINESTRING(1 3 4, 5 6 7)');
```

```
ERROR: BOX3D parser - doesn't start with BOX3D(
```

```
SELECT ST_ZMin(ST_GeomFromEWKT('CIRCULARSTRING(220268 150415 1,220227 150505 2,220227 150406 3)'));
```

1

関連情報

[ST_GeomFromEWKT](#), [ST_GeomFromText](#), [ST_XMin](#), [ST_XMax](#), [ST_YMin](#), [ST_YMax](#), [ST_ZMax](#), [ST_MMin](#), [ST_MMax](#)

7.18.15 ST_XSize

ST_XSize — Returns the X size of a 2D or 3D bounding box or a geometry.

Synopsis

```
float ST_XSize(box3d aGeomorBox2DorBox3D);
```

説明

Returns the X size of a 2D or 3D bounding box or a geometry, computed as `ST_XMax(aGeomorBox2DorBox3D) - ST_XMin(aGeomorBox2DorBox3D)`.



Note

この関数は BOX3D に対してのみ定義されていますが、自動キャストによって BOX2D やジオメトリ値でも動作します。しかしながら、ジオメトリまたは BOX2D の文字列表現は、自動キャストを行わないため、受け付けません。



この関数は 3 次元に対応し、Z 値を削除しません。



このメソッドは曲線ストリングと曲線に対応しています。

Availability: 3.7.0

例

```
SELECT ST_XSize('BOX3D(1 2 3, 4 5 6)');
```

3

```
SELECT ST_XSize(ST_GeomFromText('LINESTRING(1 3, 5 6)'));
```

4

関連情報

[ST_XMin](#), [ST_XMax](#), [ST_YSize](#), [ST_ZSize](#), [ST_MSize](#)

7.18.16 ST_YSize

ST_YSize — Returns the Y size of a 2D or 3D bounding box or a geometry.

Synopsis

float **ST_YSize**(box3d aGeomorBox2DorBox3D);

説明

Returns the Y size of a 2D or 3D bounding box or a geometry, computed as ST_YMax(aGeomorBox2DorBox3D) - ST_YMin(aGeomorBox2DorBox3D).



Note

この関数は BOX3D に対してのみ定義されていますが、自動キャストによって BOX2D やジオメトリ値でも動作します。しかしながら、ジオメトリまたは BOX2D の文字列表現は、自動キャストを行わないため、受け付けません。

- ✔ この関数は 3 次元に対応し、Z 値を削除しません。
- ✔ このメソッドは曲線ストリングと曲線に対応しています。

Availability: 3.7.0

例

```
SELECT ST_YSize('BOX3D(1 2 3, 4 5 6)');
```

3

```
SELECT ST_YSize(ST_GeomFromText('LINESTRING(1 3, 5 6)'));
```

3

関連情報

[ST_YMin](#), [ST_YMax](#), [ST_XSize](#), [ST_ZSize](#), [ST_MSize](#)

7.18.17 ST_ZSize

ST_ZSize — Returns the Z size of a 2D or 3D bounding box or a geometry.

Synopsis

```
float ST_ZSize(box3d aGeomorBox2DorBox3D);
```

説明

Returns the Z size of a 2D or 3D bounding box or a geometry, computed as `ST_ZMax(aGeomorBox2DorBox3D) - ST_ZMin(aGeomorBox2DorBox3D)`.



Note

この関数は BOX3D に対してのみ定義されていますが、自動キャストによって BOX2D やジオメトリ値でも動作します。しかしながら、ジオメトリまたは BOX2D の文字列表現は、自動キャストを行わないため、受け付けません。



この関数は 3 次元に対応し、Z 値を削除しません。



このメソッドは曲線ストリングと曲線に対応しています。

Availability: 3.7.0

例

```
SELECT ST_ZSize('BOX3D(1 2 3, 4 5 6)');
```

```
3
```

```
SELECT ST_ZSize(ST_GeomFromEWKT('LINESTRING(1 3 4, 5 6 7)'));
```

```
3
```

関連情報

[ST_ZMin](#), [ST_ZMax](#), [ST_XSize](#), [ST_YSize](#), [ST_MSize](#)

7.18.18 ST_MMin

ST_MMin — ジオメトリの M 値の最小値を返します。

Synopsis

float **ST_MMin**(geometry geom);

説明

Returns the M minima of a geometry, or null if the geometry lacks M values.



この関数は 3 次元に対応し、Z 値を削除しません。



このメソッドは曲線ストリングと曲線に対応しています。

例

```
SELECT ST_MMin('POINT M (1 2 3)');
```

```
3
```

関連情報

[ST_XMin](#), [ST_XMax](#), [ST_YMin](#), [ST_YMax](#), [ST_ZMin](#), [ST_ZMax](#), [ST_MMax](#), [ST_MSize](#)

7.18.19 ST_MMax

ST_MMax — Returns the M maxima of a geometry.

Synopsis

float **ST_MMax**(geometry geom);

説明

Returns the M maxima of a geometry, or null if the geometry lacks M values.



この関数は 3 次元に対応し、Z 値を削除しません。



このメソッドは曲線ストリングと曲線に対応しています。

例

```
SELECT ST_MMax('POINT M (1 2 3)');
```

```
3
```

関連情報

[ST_XMin](#), [ST_XMax](#), [ST_YMin](#), [ST_YMax](#), [ST_ZMin](#), [ST_ZMax](#), [ST_MMin](#), [ST_MSize](#)

7.18.20 ST_MSize

ST_MSize — Returns the M size of a geometry.

Synopsis

```
float ST_MSize(geometry geom);
```

説明

Returns the M size of a geometry, computed as `ST_MMax(geom) - ST_MMin(geom)`, or null if the geometry lacks M values.



この関数は 3 次元に対応し、Z 値を削除しません。



このメソッドは曲線ストリングと曲線に対応しています。

Availability: 3.7.0

例

```
SELECT ST_MSize('LINESTRING M (1 2 3, 4 6 8)');
```

```
5
```

関連情報

[ST_MMin](#), [ST_MMax](#), [ST_XSize](#), [ST_YSize](#), [ST_ZSize](#)

7.19 線型参照

7.19.1 ST_LineInterpolatePoint

ST_LineInterpolatePoint — ラインに沿って、割合で示された位置の補間ポイントを返します。

Synopsis

```
geometry ST_LineInterpolatePoint(geometry a_linestring, float8 a_fraction);  
geography ST_LineInterpolatePoint(geography a_linestring, float8 a_fraction, boolean use_spheroid  
= true);
```

説明

ラインに沿って、割合で示された位置の補間ポイントを返します。一つ目の引数は **LINESTRING** です。二つ目の引数は 0 から 1 の間の浮動小数点数で、ライン長に対するポイントの位置の割合を表現します。Z 値と M 値が存在する場合には補間計算を行います。

ポイントに最も近いライン位置の計算については **ST_LineLocatePoint** を参照して下さい。



Note

この関数は 2 次元のポイントとして計算して、Z 値と M 値は補間値を計算します。**ST_3DLineInterpolatePoint** は 3 次元のポイントとして計算して、M 値のみ補間値を計算します。



Note

1.1.1 から、この関数は M 軸や Z 軸の内挿点も (存在するなら) 計算するようになりました。それより前の版では 0.0 となります。

Availability: 0.8.2 Z 軸と M 軸のサポートが 1.1.1 で追加されました

Changed: 2.1.0 2.0.x までは **ST_Line_Interpolate_Point** と呼んでいました。



この関数は 3 次元に対応し、Z 値を削除しません。

例



ラインストリングの 20% (0.20) 位置の補完ポイント

This example returns the point 20 percent along a line.

```
SELECT ST_AsEWKT(ST_LineInterpolatePoint(  
  'LINESTRING(25 50, 100 125, 150 190)',  
  0.2));
```

```
POINT(51.5974135047432 76.5974135047432)
```

3 次元ラインの中間点:


```
SELECT ST_AsEWKT(ST_LineInterpolatePoint(
  'LINESTRING(1 2 3, 4 5 6, 6 7 8)',
  0.5));
```

```
POINT(3.5 4.5 5.5)
```

ライン上のポイントに最も近いポイント:

```
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)
```

関連情報

[ST_LineInterpolatePoints](#), [ST_3DLineInterpolatePoint](#), [ST_LineLocatePoint](#)

7.19.2 ST_3DLineInterpolatePoint

ST_3DLineInterpolatePoint — 3 次元ラインに沿って、割合で示された位置の補間ポイントを返します。

Synopsis

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

説明

3 次元ラインに沿って、割合で示された位置の補間ポイントを返します。一つ目の引数は **LINESTRING** です。二つ目の引数は 0 から 1 の間の浮動小数点数で、ライン長に対するポイントの位置の割合を表現します。M 値が存在する場合には補間計算を行います。



Note

ST_LineInterpolatePoint 2 次元ポイントの計算と Z 値と M 値の補間値を計算しますが、この関数は 3 次元ポイントを計算し、M 値だけ補間計算を行います。

Availability: 3.0.0



この関数は 3 次元に対応し、Z 値を削除しません。

例

3 次元ラインに沿って 20% のポイントを返します

```
SELECT ST_AsText(ST_3DLineInterpolatePoint(
  'LINESTRING(25 50 70, 100 125 90, 150 190 200)',
  0.20));
```

```
POINT Z (59.06758929108215 84.06758929108216 79.0846904776219)
```

関連情報

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

7.19.3 ST_LineInterpolatePoints

`ST_LineInterpolatePoints` — ラインに沿って、割合で示された複数の位置の補間ポイントを返します。

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);
```

説明

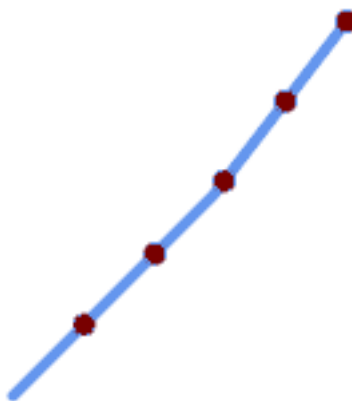
ラインに沿った割合による間隔にある一つ以上の補間ポイントを返します。一つ目の引数は `LINSTRING` でなければなりません。二つ目の引数は `float8` 型の 0 から 1 の間の値でなければなりません。この値はライン長に対するポイント間隔の割合を示します。三つ目の引数が `FALSE` の場合には、一つ以下のポイントが構築されます ([ST_LineInterpolatePoint](#)と同じです)。

結果が 0 以上のポイントを持つ場合には `POINT` が返ります。二つ以上のポイントを持つ場合には `MULTIPOINT` が返ります。

Availability: 2.5.0

- ✔ この関数は 3 次元に対応し、Z 値を削除しません。
- ✔ この関数は M 値に対応します。

例



20% 刻みで補間されたポイントを持つラインストリング

This example returns points every 20 percent along a two-dimensional line.

```
SELECT ST_AsText(ST_LineInterpolatePoints(  
  'LINESTRING(25 50, 100 125, 150 190)',  
  0.20));
```

```
MULTIPOINT((51.5974135047432 76.5974135047432), (78.1948270094864 103.194827009486) ↔  
  , (104.132163186446 130.37181214238), (127.066081593223 160.18590607119), (150 190))
```

関連情報

[ST_LineInterpolatePoint](#), [ST_LineLocatePoint](#)

7.19.4 ST_LineLocatePoint

`ST_LineLocatePoint` — ポイントに最も近いライン上のポイントの位置を割合で返します。

Synopsis

```
float8 ST_LineLocatePoint(geometry a_linestring, geometry a_point);  
float8 ST_LineLocatePoint(geography a_linestring, geography a_point, boolean use_spheroid = true);
```

説明

ラインストリング上の、与えたポイントへの最短点を、[2次元ラインストリング](#)の総延長に対する割合として 0 から 1 の区間で返します。

返された位置は、ポイント ([ST_LineInterpolatePoint](#)) または、部分ラインストリング ([ST_LineSubstring](#)) の抽出に使用することができます。

この関数は、住所番号に近づくのに使えます (訳注: 道路方式の住居表示の場合)。

Availability: 1.1.0

Changed: 2.1.0 2.0.x までは `ST_Line_Locate_Point` と呼んでいました。

例

This example approximates the street number of a house by locating the closest point on a street line. The subquery generates sample house and street data, and `ST_DWithin` excludes houses that are too far from the street.

```
SELECT ST_AsText(house_loc) As as_text_house_loc,  
       startstreet_num +  
       CAST( (endstreet_num - startstreet_num)  
            * ST_LineLocatePoint(street_line, house_loc) As integer) As ↔  
       street_num  
FROM  
(SELECT ST_GeomFromText('LINESTRING(1 2, 3 4)') As street_line,  
       ST_Point(x*1.01,y*1.03) As house_loc, 10 As startstreet_num,  
       20 As endstreet_num  
FROM generate_series(1,3) x CROSS JOIN generate_series(2,4) As y)  
As foo  
WHERE ST_DWithin(street_line, house_loc, 0.2);
```

as_text_house_loc	street_num
POINT(1.01 2.06)	10
POINT(2.02 3.09)	15
POINT(3.03 4.12)	20

This example finds the closest point on a line to a point or other geometry.

```
SELECT ST_AsText(ST_LineInterpolatePoint(
    foo.the_line,
    ST_LineLocatePoint(foo.the_line, ST_GeomFromText('POINT(4 3)'))))
FROM (SELECT ST_GeomFromText('LINESTRING(1 2, 4 5, 6 7)') As the_line) As foo;

POINT(3 4)
```

関連情報

[ST_DWithin](#), [ST_Length2D](#), [ST_LineInterpolatePoint](#), [ST_LineSubstring](#)

7.19.5 ST_LineSubstring

ST_LineSubstring — 二つの割合位置からラインの一部を返します。

Synopsis

geometry **ST_LineSubstring**(geometry a_linestring, float8 startfraction, float8 endfraction);
 geography **ST_LineSubstring**(geography a_linestring, float8 startfraction, float8 endfraction);

説明

Computes the line which is the section of the input line starting and ending at the given fractional locations. The first argument must be a LINESTRING. The second and third arguments are values in the range [0, 1] representing the start and end locations. For geometry inputs, the fractions are measured in 2D line length. The Z and M values are interpolated for added endpoints if present.

`startfraction` と `endfraction` が同じ値である場合には、[ST_LineInterpolatePoint](#)と同じになります。



Note

この関数は LINESTRING に対してのみ動作します。連続する MULTILINESTRING で使うには、先に [ST_LineMerge](#) で結合させます。



Note

1.1.1 リリースからは、この関数は M 値と Z 値を補間します。それより前のリリースでは、Z 値と M 値は不定値になります。

Enhanced: 3.4.0 ジオグラフィ対応が導入されました。

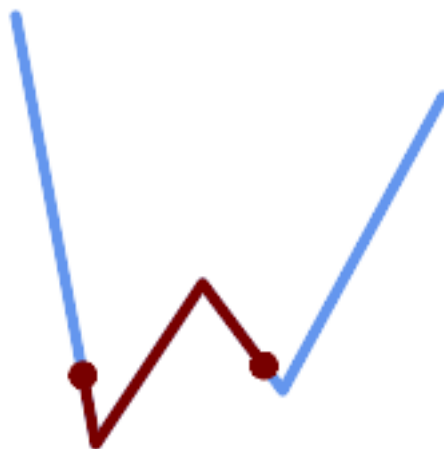
Changed: 2.1.0 2.0.x では `ST_Line_Substring` と呼ばれていました。

Availability: 1.1.0 Z 軸と M 軸のサポートが 1.1.1 で追加されました



この関数は 3 次元に対応し、Z 値を削除しません。

例



中心の 1/3 区間 (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)
```

開始点と終了点とが同じ場合には、結果は POINT になります。

```
SELECT ST_AsText(ST_LineSubstring(
  'LINESTRING(25 50, 100 125, 150 190)',
  0.333,
  0.333));
```

```
POINT(69.28469348539744 94.28469348539744)
```

LINESTRING を長さ 100 以下の断片に分解するクエリ。FOR ループと同等のものを生成するために generate_series と CROSS JOIN LATERAL とを併用しています。

```
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)

ジオグラフィ実装では回転楕円体面に沿って計測し、ジオメトリではラインに沿って計測します。

```
SELECT ST_AsText(ST_LineSubstring(
  'LINESTRING(-118.2436 34.0522, -71.0570 42.3611)::geography',
  0.333,
  0.666),
  6) AS geog_sub,
  ST_AsText(ST_LineSubstring(
    'LINESTRING(-118.2436 34.0522, -71.0570 42.3611)::geometry',
    0.333,
    0.666),
    6) AS geom_sub;
```

geog_sub	geom_sub
LINESTRING(-103.911641 38.931128,-87.941787 41.831072) LINESTRING(-102.530462 36.819064,-86.817324 39.585927) ←	

For geometry inputs, the fractional locations are based on the 2D length of the line, even when the input has Z values. The resulting Z values are interpolated along the selected 2D location.

```
WITH data AS (
  SELECT 'LINESTRING Z (0 0 0, 0 2 5, 0 10 10)::geometry' AS geom
)
SELECT ST_Length(geom) AS length_2d,
       ST_3DLength(geom) AS length_3d,
       ST_AsText(ST_LineSubstring(geom, 0, 0.5)) AS substring
FROM data;
```

length_2d	length_3d	substring
10	14.819145939191106	LINESTRING Z (0 0 0,0 2 5,0 5 6.875)

関連情報

[ST_Length](#), [ST_LineExtend](#), [ST_LineInterpolatePoint](#), [ST_LineMerge](#)

7.19.6 ST_LocateAlong

`ST_LocateAlong` — M 値に一致するジオメトリ上のポイントを返します。

Synopsis

geometry **ST_LocateAlong**(geometry geom_with_measure, float8 measure, float8 offset = 0);

説明

M 値を持つジオメトリに沿った位置を返します。結果はポイントまたはマルチポイントです。ポリゴン系入力には対応していません。

`offset` が与えられた場合には、結果は、入力ラインの左または右に、指定された距離だけ移動します。正のオフセット値で左に、負の値で右にそれぞれ移動します。



Note

この関数は、M 要素を持つライン系ジオメトリでのみ使います。

ISO/IEC 13249-3 SQL/MM 空間標準で規定されているため、このような意味になります。

Availability: 1.1.0 それまでは `ST_LocateAlong_Measure` でした。

Changed: 2.0.0 以前の版では `ST_LocateAlong_Measure` と呼ばれていました。



この関数は M 値に対応します。



このメソッドは SQL/MM 仕様の実装です。SQL-MM IEC 13249-3: 5.1.13

例

```
SELECT ST_AsText(ST_LocateAlong(  
  'MULTILINESTRINGM((1 2 3, 3 4 2, 9 4 3),(1 2 3, 5 4 5))'::geometry,  
  3));
```

```
MULTIPOINT M ((1 2 3),(9 4 3),(1 2 3))
```

関連情報

[ST_LocateBetween](#), [ST_LocateBetweenElevations](#), [ST_InterpolatePoint](#)

7.19.7 ST_LocateBetween

`ST_LocateBetween` — M 値の範囲に合致する部分ジオメトリを返します。

Synopsis

geometry **ST_LocateBetween**(geometry geom, float8 measure_start, float8 measure_end, float8 offset = 0);

説明

指定された範囲の M 値を持つ入力ジオメトリの部分ジオメトリとなるジオメトリ (コレクション) を全て返します。

`offset` が与えられた場合には、結果は、入力ラインの左または右に、指定された距離だけ移動します。正のオフセット値で左に、負の値で右にそれぞれ移動します。

凸でない POLYGON を抜き出すと不正なジオメトリを返すことがあります。

ISO/IEC 13249-3 SQL/MM 空間標準で規定されているため、このような意味になります。

Availability: 1.1.0 それより前は ST_Locate_Between_Measures でした。

Changed: 2.0.0 以前の版では ST_Locate_Between_Measures と呼ばれていました。

Enhanced: 3.0.0 - POLYGON, TIN, TRIANGLE への対応が追加されました。

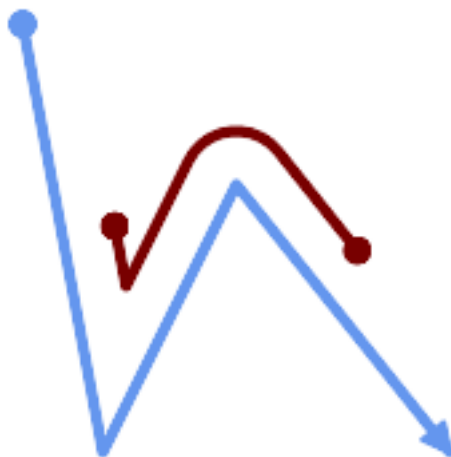
✔ この関数は M 値に対応します。

✔ このメソッドは SQL/MM 仕様の実装です。SQL-MM IEC 13249-3: 5.1

例

```
SELECT ST_AsText(ST_LocateBetween(
  'MULTILINESTRING M ((1 2 3, 3 4 2, 9 4 3),(1 2 3, 5 4 5))'::geometry,
  1.5,
  3));
```

```
GEOMETRYCOLLECTION M (LINESTRING M (1 2 3,3 4 2,9 4 3),POINT M (1 2 3))
```



M 値が 2 と 8 の間になる辺を持ち左にずれたラインストリング

```
SELECT ST_AsText(ST_LocateBetween(
  ST_AddMeasure(
    'LINESTRING (20 180, 50 20, 100 120, 180 20)',
    0,
    10),
  2,
  8,
  20));
```

```
MULTILINESTRING((54.49835019899045 104.53426957938231,58.70056060327303 ↵
82.12248075654186,82.11145618000168 128.94427190999915,84.24893681714357 ↵
132.32493442618113,87.01636951231555 135.21267035596549,90.30307285299679 ↵
137.49198684843182,93.97759758337769 139.07172433557758,97.89298381958797 ↵
139.8887023914453,101.89263860095893 139.9102465862721,105.81659870902816 ↵
139.13549527600819,109.50792827749828 137.5954340631298,112.81899532549731 ↵
135.351656550512,115.6173761888606 132.49390095108848,145.31017306064817 ↵
95.37790486135405))
```


関連情報

[ST_LocateAlong](#), [ST_LocateBetweenElevations](#)

7.19.8 ST_LocateBetweenElevations

`ST_LocateBetweenElevations` — 標高 (Z 値) 範囲にある部分ジオメトリを返します。

Synopsis

geometry **ST_LocateBetweenElevations**(geometry geom, float8 elevation_start, float8 elevation_end);

説明

標高 (Z 値) 範囲にある部分ジオメトリとなるジオメトリ (コレクション) を返します。

凸でない POLYGON を抜き出すと不正なジオメトリを返すことがあります。

Availability: 1.4.0

Enhanced: 3.0.0 - POLYGON, TIN, TRIANGLE への対応が追加されました。



この関数は 3 次元に対応し、Z 値を削除しません。

例

```
SELECT ST_AsText(ST_LocateBetweenElevations(
  'LINESTRING(1 2 3, 4 5 6)::geometry',
  2,
  4));
```

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

```
SELECT ST_AsText(ST_LocateBetweenElevations(
  'LINESTRING(1 2 6, 4 5 -1, 7 8 9)',
  6,
  9)) As ewelev;
```

```
GEOMETRYCOLLECTION Z (POINT Z (1 2 6),LINESTRING Z (6.1 7.1 6,7 8 9))
```

関連情報

[ST_Dump](#), [ST_LocateBetween](#)

7.19.9 ST_InterpolatePoint

`ST_InterpolatePoint` — ジオメトリのポイントに最も近いポイント上の補間 M 値を返します。

Synopsis

float8 **ST_InterpolatePoint**(geometry linear_geom_with_measure, geometry point);

説明

ライン系の M 値を持つジオメトリについて、与えられたポイントに最も近い位置における M 値の補間値を返します。



Note

この関数は、M 要素を持つライン系ジオメトリでのみ使います。

Availability: 2.0.0



この関数は 3 次元に対応し、Z 値を削除しません。

例

```
SELECT ST_InterpolatePoint(  
  'LINESTRING M (0 0 0, 10 0 20)',  
  'POINT(5 5)');
```

10

関連情報

[ST_AddMeasure](#), [ST_LocateAlong](#), [ST_LocateBetween](#)

7.19.10 ST_3DInterpolatePoint

ST_3DInterpolatePoint — 3 次元で最も近いジオメトリの補間 M 値を返します。

Synopsis

float8 **ST_3DInterpolatePoint**(geometry linear_geom_with_measure, geometry point);

説明

ライン系 **ZM** ジオメトリの、与えた点から最も近い位置の補間 M 値を返します。使用投影法の 3 次元 (XYZ) 距離を使います。航空機の航跡のように Z 値の変化が重要となり、Z 値を無視する [ST_InterpolatePoint](#) では間違った結果が得られるようなジオメトリに対してこの関数を使います。



Note

ラインは Z 値と M 値を持っていなければなりません。ポイントは Z 値をもっていなければなりません。

Availability: 3.7.0



この関数は 3 次元に対応し、Z 値を削除しません。

例

This example interpolates on a line rising diagonally in Z, using a point at the 3D midpoint.

```
SELECT ST_3DInterpolatePoint(  
  'LINESTRING ZM (0 0 0 0, 10 0 10 20)',  
  'POINT Z (5 0 5)');
```

10

This example uses a vertical line with zero XY extent, where 3D projection works but 2D projection does not.

```
SELECT ST_3DInterpolatePoint(  
  'LINESTRING ZM (0 0 0 0, 0 0 10 100)',  
  'POINT Z (5 5 5)');
```

50

関連情報

[ST_InterpolatePoint](#), [ST_AddMeasure](#), [ST_LocateAlong](#), [ST_LocateBetween](#)

7.19.11 ST_AddMeasure

ST_AddMeasure — ラインに沿った M 値を補間します。

Synopsis

geometry **ST_AddMeasure**(geometry geom_mline, float8 measure_start, float8 measure_end);

説明

開始点と終了点の間で線形補間した M 値を持つ派生ジオメトリを返します。ジオメトリが M 値を持たない場合には M 値が追加されます。ジオメトリが M 値を持つ場合には新値に置き換えられます。LINESTRING と MULTILINESTRING だけに対応しています。

Availability: 1.5.0



この関数は 3 次元に対応し、Z 値を削除しません。

例

```
SELECT ST_AsText(ST_AddMeasure(  
  ST_GeomFromEWKT('LINESTRING(1 0, 2 0, 4 0)'),  
  1,  
  4)) As ewelev;
```

LINESTRING M (1 0 1,2 0 2,4 0 4)

```
SELECT ST_AsText(ST_AddMeasure(  
  ST_GeomFromEWKT('LINESTRING(1 0 4, 2 0 4, 4 0 4)'),  
  10,  
  40)) As ewelev;
```

```
LINESTRING ZM (1 0 4 10,2 0 4 20,4 0 4 40)
```

```
SELECT ST_AsText(ST_AddMeasure(  
  ST_GeomFromEWKT('LINESTRINGM(1 0 4, 2 0 4, 4 0 4)'),  
  10,  
  40)) As ewelev;
```

```
LINESTRING M (1 0 10,2 0 20,4 0 40)
```

```
SELECT ST_AsText(ST_AddMeasure(  
  ST_GeomFromEWKT('MULTILINESTRINGM((1 0 4, 2 0 4, 4 0 4),(1 0 4, 2 0 4, 4 0 4)'),  
  10,  
  70)) As ewelev;
```

```
MULTILINESTRING M ((1 0 10,2 0 20,4 0 40),(1 0 40,2 0 50,4 0 70))
```

7.20 トラジェクトリ関数

7.20.1 ST_IsValidTrajectory

ST_IsValidTrajectory — ジオメトリが妥当なトラジェクトリであるかどうかをテストします。

Synopsis

boolean **ST_IsValidTrajectory**(geometry line);

説明

ジオメトリが妥当なトラジェクトリの符号化したものとなっているかをテストします。妥当なトラジェクトリは、M 値 (Measure) を持つ **LINESTRING** で表現されます。M 値は始点から順次増加していかなければなりません。

ST_ClosestPointOfApproach等の時空間関数は、妥当なトラジェクトリを入力値として期待します。

Availability: 2.2.0

 この関数は 3 次元に対応し、Z 値を削除しません。

例

This example constructs a valid trajectory.

```
SELECT ST_IsValidTrajectory(ST_MakeLine(ST_MakePointM(0, 0, 1),  
  ST_MakePointM(0, 1, 2)));
```

t

This example constructs an invalid trajectory.

```
SELECT ST_IsValidTrajectory(ST_MakeLine(ST_MakePointM(0, 0, 1), ST_MakePointM(0, 1, 0)));
```

```
NOTICE: Measure of vertex 1 (0) not bigger than measure of vertex 0 (1)
f
```

関連情報

[ST_ClosestPointOfApproach](#)

7.20.2 ST_ClosestPointOfApproach

`ST_ClosestPointOfApproach` — 二つのトラジェクトリの最接近時の距離を返します。

Synopsis

```
float8 ST_ClosestPointOfApproach(geometry track1, geometry track2);
```

説明

与えられたトラジェクトリに沿って補間された最小距離となる点の `M` 値の最小値を返します。

妥当なトラジェクトリは [ST_IsValidTrajectory](#) で確認できます。トラジェクトリの `M` 値の範囲が重ならない場合には `NULL` が返ります。

計算された `M` 値で実際のポイントを得るには [ST_LocateAlong](#) を使います。

Availability: 2.2.0



この関数は 3 次元に対応し、`Z` 値を削除しません。

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 timestamp '2015-05-26 10:00' AT TIME ZONE 'UTC'),
    extract(epoch from timestamp '2015-05-26 11:00' AT TIME ZONE 'UTC')
  ) a,
  ST_AddMeasure('LINESTRING Z (0 2 10, 12 1 2)::geometry',
    extract(epoch from timestamp '2015-05-26 10:00' AT TIME ZONE 'UTC'),
    extract(epoch from timestamp '2015-05-26 11:00' AT TIME ZONE 'UTC')
  ) b
), cpa AS (
  SELECT ST_ClosestPointOfApproach(a,b) m FROM inp
), points AS (
  SELECT ST_GeometryN(ST_LocateAlong(a,m),1) pa,
    ST_GeometryN(ST_LocateAlong(b,m),1) pb
  FROM inp, cpa
)
SELECT to_timestamp(m) AT TIME ZONE 'UTC' t,
  ST_3DDistance(pa,pb) distance,
  ST_AsText(pa, 2) AS pa, ST_AsText(pb, 2) AS pb
FROM points, cpa;
```

t	distance pb	pa	↔
2015-05-26 10:45:31.034483	1.9652147377620688	POINT ZM (7.59 0 3.79 1432637131.03)	↔
		POINT ZM (9.1 1.24 3.93 1432637131.03)	

関連情報

[ST_IsValidTrajectory](#), [ST_DistanceCPA](#), [ST_LocateAlong](#), [ST_AddMeasure](#)

7.20.3 ST_DistanceCPA

ST_DistanceCPA — 二つのトラジェクトリの最接近する時の距離を返します。

Synopsis

float8 **ST_DistanceCPA**(geometry track1, geometry track2);

説明

二つのトラジェクトリの最接近する時の (2 次元) 距離を返します。

妥当なトラジェクトリは[ST_IsValidTrajectory](#)で確認できます。トラジェクトリの M 値の範囲が重ならない場合には NULL が返ります。

Availability: 2.2.0

 この関数は 3 次元に対応し、Z 値を削除しません。

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 timestamp '2015-05-26 10:00' AT TIME ZONE 'UTC'),
  extract(epoch from timestamp '2015-05-26 11:00' AT TIME ZONE 'UTC')
) a,
  ST_AddMeasure('LINESTRING Z (0 2 10, 12 1 2)')::geometry,
  extract(epoch from timestamp '2015-05-26 10:00' AT TIME ZONE 'UTC'),
  extract(epoch from timestamp '2015-05-26 11:00' AT TIME ZONE 'UTC')
) b
)
SELECT ST_DistanceCPA(a,b) distance FROM inp;
```

1.9652147377620688

関連情報

[ST_IsValidTrajectory](#), [ST_ClosestPointOfApproach](#), [ST_AddMeasure](#), [|=|](#)

7.20.4 ST_CPAWithin

ST_CPAWithin — 二つのトラジェクトリの最接近時の距離が指定距離内であるかどうかをテストします。

Synopsis

boolean **ST_CPAWithin**(geometry track1, geometry track2, float8 dist);

説明

二つの移動オブジェクトが指定距離より近づいているかどうかをテストします。

入力は **ST_IsValidTrajectory** で確認された妥当なトラジェクトリでなければなりません。トラジェクトリの M 値の範囲が重ならない場合には FALSE を返します。

Availability: 2.2.0

 この関数は 3 次元に対応し、Z 値を削除しません。

例

```
WITH inp AS ( SELECT
  ST_AddMeasure('LINESTRING Z (0 0 0, 10 0 5)::geometry',
    extract(epoch from timestamp '2015-05-26 10:00' AT TIME ZONE 'UTC'),
    extract(epoch from timestamp '2015-05-26 11:00' AT TIME ZONE 'UTC')
  ) a,
  ST_AddMeasure('LINESTRING Z (0 2 10, 12 1 2)::geometry',
    extract(epoch from timestamp '2015-05-26 10:00' AT TIME ZONE 'UTC'),
    extract(epoch from timestamp '2015-05-26 11:00' AT TIME ZONE 'UTC')
  ) b
)
SELECT ST_CPAWithin(a,b,2), ST_DistanceCPA(a,b) distance FROM inp;
```

st_cpawithin	distance
t	1.9652147377620688

関連情報

ST_IsValidTrajectory, **ST_ClosestPointOfApproach**, **ST_DistanceCPA**, **|=|**

7.21 バージョン関数

7.21.1 PostGIS_Extensions_Upgrade

PostGIS_Extensions_Upgrade — PostGIS エクステンション (例: **postgis_raster**, **postgis_topology**, **postgis_sfcgal**) について、指定したバージョンまたは最新版にパッケージ化し、アップグレードします。

Synopsis

text **PostGIS_Extensions_Upgrade**(text target_version=null);

説明

指定されたバージョンまたは最新バージョンへの PostGIS エクステンションのパッケージ化とアップグレードを行います。そのデータベースにインストールしたエクステンションだけが必要に応じてパッケージ化とアップグレードとを実行されます。完全な PostGIS バージョンを報告し、その後、ビルドコンフィギュレーション情報を報告します。PostGIS エクステンションごとに行う複数の CREATE EXTENSION .. FROM unpackaged および ALTER EXTENSION .. UPDATE の短縮形です。現在は、postgis, postgis_raster, postgis_sfcgal, postgis_topology の四つのエクステンションのアップグレードのみを試みます。

Availability: 2.5.0



Note

Changed: 3.4.0 target_version 引数が追加されました。

Changed: 3.3.0 どの PostGIS のバージョンからでもアップグレードできるようになりました。ただし全てのシステムで動作するわけではありません。

Changed: 3.0.0 緩いエクステンションを再パッケージし、また、postgis_raster に対応しました。

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
```

関連情報

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 — 完全な PostGIS のバージョン情報とコンフィギュレーション情報を報告します。

Synopsis

```
text PostGIS_Full_Version();
```

説明

完全な PostGIS のバージョン情報とコンフィギュレーション情報を報告します。ライブラリとスクリプトとの間の同期について情報を提供して、必要に応じてアップグレードの提案に関する情報を提供します。

Enhanced: 3.4.0 現在、外部 PROJ 設定の NETWORK_ENABLED、URL_ENDPOINT、proj.db 位置の DATABASE_PATH があります

例

```
SELECT PostGIS_Full_Version();
```

```
POSTGIS="3.4.0dev 3.3.0rc2-993-g61bdf43a7" [EXTENSION] PGSQL="160" GEOS="3.12.0dev-CAPI ↵  
-1.18.0" SFCGAL="1.3.8" PROJ="7.2.1 NETWORK_ENABLED=OFF URL_ENDPOINT=https://cdn.proj. ↵  
org USER_WRITABLE_DIRECTORY=/tmp/proj DATABASE_PATH=/usr/share/proj/proj.db" GDAL="GDAL ↵  
3.2.2, released 2021/03/05" LIBXML="2.9.10" LIBJSON="0.15" LIBPROTOBUF="1.3.3" WAGYU ↵  
="0.5.0 (Internal)" TOPOLOGY RASTER
```

関連情報

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 — GEOS ライブラリのバージョン番号を返します。

Synopsis

```
text PostGIS_GEOS_Version();
```

説明

GEOS ライブラリのバージョン番号を返します。GEOS 対応が有効でない場合は `NULL` を返します。

例

```
SELECT PostGIS_GEOS_Version();
```

```
3.12.0dev-CAPI-1.18.0
```

関連情報

[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 — PostGIS のビルドに使われた GEOS ライブラリのバージョン番号を返します。

Synopsis

```
text PostGIS_GEOS_Compiled_Version();
```

説明

GEOS ライブラリのバージョン番号、または PostGIS のビルドに使ったバージョン番号を返します。

Availability: 3.4.0

例

```
SELECT PostGIS_GEOS_Compiled_Version();
```

```
3.12.0
```

関連情報

[PostGIS_GEOS_Version](#), [PostGIS_Full_Version](#)

7.21.5 PostGIS_Liblwgeom_Version

`PostGIS_Liblwgeom_Version` — liblwgeom ライブラリのバージョン番号を返します。PostGIS のバージョンと同じになるべきものです。

Synopsis

```
text PostGIS_Liblwgeom_Version();
```

説明

liblwgeom ライブラリのバージョン番号を返します。

例

```
SELECT PostGIS_Liblwgeom_Version();
```

```
3.4.0dev 3.3.0rc2-993-g61bdf43a7
```

関連情報

[PostGIS_Full_Version](#), [PostGIS_Lib_Version](#), [PostGIS_LibXML_Version](#), [PostGIS_PROJ_Version](#), [PostGIS_Version](#)

7.21.6 PostGIS_LibXML_Version

`PostGIS_LibXML_Version` — LibXML2 ライブラリのバージョン番号を返します。

Synopsis

```
text PostGIS_LibXML_Version();
```

説明

LibXML2 ライブラリのバージョン番号を返します。

Availability: 1.5

例

```
SELECT PostGIS_LibXML_Version();
```

```
2.9.10
```

関連情報

[PostGIS_Full_Version](#), [PostGIS_Lib_Version](#), [PostGIS_PROJ_Version](#), [PostGIS_GEOS_Version](#), [PostGIS_Version](#)

7.21.7 PostGIS_LibJSON_Version

PostGIS_LibJSON_Version — libjson-c ライブラリのバージョン番号を返します。

Synopsis

text **PostGIS_LibJSON_Version**();

説明

LibJSON-C ライブラリのバージョン番号を返します。

例

```
SELECT PostGIS_LibJSON_Version();
```

```
0.17
```

関連情報

[PostGIS_Full_Version](#), [PostGIS_Lib_Version](#), [PostGIS_PROJ_Version](#), [PostGIS_GEOS_Version](#), [PostGIS_Version](#)

7.21.8 PostGIS_Lib_Build_Date

PostGIS_Lib_Build_Date — PostGIS ライブラリのビルド日付を返します。

Synopsis

text **PostGIS_Lib_Build_Date**();

説明

PostGIS ライブラリのビルド日付を返します。

例

```
SELECT PostGIS_Lib_Build_Date();
```

```
2023-06-22 03:56:11
```

7.21.9 PostGIS_Lib_Version

PostGIS_Lib_Version — PostGIS のバージョン番号を返します。

Synopsis

text **PostGIS_Lib_Version()**;

説明

PostGIS のバージョン番号を返します。

例

```
SELECT PostGIS_Lib_Version();
```

```
3.4.0dev
```

関連情報

[PostGIS_Full_Version](#), [PostGIS_GEOS_Version](#), [PostGIS_LibXML_Version](#), [PostGIS_PROJ_Version](#), [PostGIS_Version](#)

7.21.10 PostGIS_PROJ_Version

PostGIS_PROJ_Version — PROJ4 のバージョン番号を返します。

Synopsis

text **PostGIS_PROJ_Version()**;

説明

PROJ ライブラリのバージョン番号と Proj の設定オプションを返します。

Enhanced: 3.4.0 現在、NETWORK_ENABLED、URL_ENDPOINT、proj.db 位置の DATABASE_PATH があります

例

```
SELECT PostGIS_PROJ_Version();
```

```
7.2.1 NETWORK_ENABLED=OFF URL_ENDPOINT=https://cdn.proj.org USER_WRITABLE_DIRECTORY=/tmp/ ↵  
proj DATABASE_PATH=/usr/share/proj/proj.db
```

関連情報

[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` — PostGIS のビルドに使われた PROJ ライブラリのバージョン番号を返します。

Synopsis

text **PostGIS_PROJ_Compiled_Version()**;

説明

PROJ ライブラリのバージョンを返しますが、PostGIS のビルドに使われたもののバージョン番号です。

Availability: 3.5.0

例

```
SELECT PostGIS_PROJ_Compiled_Version();
```

9.1.1

関連情報

[PostGIS_PROJ_Version](#), [PostGIS_Full_Version](#)

7.21.12 PostGIS_Wagyu_Version

`PostGIS_Wagyu_Version` — 内部の Wagyu ライブラリのバージョン番号を返します。

Synopsis

text **PostGIS_Wagyu_Version()**;

説明

内部の Wagyu ライブラリのバージョン番号を返します。Wagyu 対応が有効でない場合には `NULL` を返します。

例

```
SELECT PostGIS_Wagyu_Version();
```

```
0.5.0 (Internal)
```

関連情報

[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` — PostGIS スクリプトのビルド日付を返します。

Synopsis

text **PostGIS_Scripts_Build_Date**();

説明

PostGIS スクリプトのビルド日付を返します。

Availability: 1.0.0RC1

例

```
SELECT PostGIS_Scripts_Build_Date();
```

```
2023-06-22 03:56:11
```

関連情報

[PostGIS_Full_Version](#), [PostGIS_GEOS_Version](#), [PostGIS_Lib_Version](#), [PostGIS_LibXML_Version](#), [PostGIS_Version](#)

7.21.14 PostGIS_Scripts_Installed

`PostGIS_Scripts_Installed` — このデータベースにインストールした PostGIS スクリプトのバージョンを返します。

Synopsis

```
text PostGIS_Scripts_Installed();
```

説明

このデータベースにインストールした PostGIS スクリプトのバージョンを返します。



Note

この関数の出力と **PostGIS_Scripts_Released** とが合わない場合、既存のデータベースの確実なアップグレードに失敗しているかも知れません。詳細情報については **Upgrading** をご覧ください。

Availability: 0.9.0

例

```
SELECT PostGIS_Scripts_Installed();
```

```
3.4.0dev 3.3.0rc2-993-g61bdf43a7
```

関連情報

PostGIS_Full_Version, **PostGIS_Scripts_Released**, **PostGIS_Version**

7.21.15 PostGIS_Scripts_Released

PostGIS_Scripts_Released — インストールした PostGIS ライブラリとともにリリースされた **postgis.sql** スクリプトのバージョン番号を返します。

Synopsis

```
text PostGIS_Scripts_Released();
```

説明

インストールした PostGIS ライブラリとともにリリースされた **postgis.sql** スクリプトのバージョン番号を返します。



Note

1.1.0 からこの関数は **PostGIS_Lib_Version** と同じ値を返すようになりました。後方互換のためです。

Availability: 0.9.0

例

```
SELECT PostGIS_Scripts_Released();
```

```
3.4.0dev 3.3.0rc2-993-g61bdf43a7
```

関連情報

[PostGIS_Full_Version](#), [PostGIS_Scripts_Installed](#), [PostGIS_Lib_Version](#)

7.21.16 PostGIS_Version

`PostGIS_Version` — PostGIS バージョン番号とコンパイルオプションを返します。

Synopsis

```
text PostGIS_Version();
```

説明

`PostGIS_Version` バージョン番号とコンパイルオプションを返します。

例

```
SELECT PostGIS_Version();
```

```
3.4 USE_GEOS=1 USE_PROJ=1 USE_STATS=1
```

関連情報

[PostGIS_Full_Version](#), [PostGIS_GEOS_Version](#), [PostGIS_Lib_Version](#), [PostGIS_LibXML_Version](#), [PostGIS_PROJ_Version](#)

7.22 Grand Unified Custom 変数 (GUC)

7.22.1 postgis.gdal_datapath

`postgis.gdal_datapath` — GDAL の `GDAL_DATA` オプションの値を設定するためのコンフィギュレーションオプションです。設定しない場合には、`GDAL_DATA` 環境変数が使われます。

説明

GDAL の GDAL_DATA オプションの値の設定に使う PostgreSQL GUC 変数です。postgis.gdal_datapath 値は完全に GDAL のデータファイルへの物理的なパスになるべきものです。

コンフィギュレーションオプションは、GDAL のデータファイルパスがハードコーディングされていない Windows プラットフォームのためにほとんど使われます。このオプションは、GDAL のデータファイルが GDAL の期待されているパスに無いときに設定します。



Note

このオプションは、PostgreSQL のコンフィギュレーションファイル postgresql.conf で設定できます。コネクションまたはトランザクションでも設定できます。

Availability: 2.2.0



Note

GDAL_DATA に関する追加情報は、GDAL の [Configuration Options](#) にあります。

例

postgis.gdal_datapath の設定とリセット。

```
SET postgis.gdal_datapath TO '/usr/local/share/gdal.hidden';  
SET postgis.gdal_datapath TO default;
```

Windows 上における特定のデータベース上で設定する場合は次の通りです。

```
ALTER DATABASE gisdb  
SET postgis.gdal_datapath = 'C:/Program Files/PostgreSQL/9.3/gdal-data';
```

関連情報

[PostGIS_GDAL_Version](#), [ST_Transform](#)

7.22.2 postgis.gdal_enabled_drivers

postgis.gdal_enabled_drivers — PostGIS 環境で GDAL ドライバーを有効にするコンフィギュレーションオプションです。GDAL コンフィギュレーション変数 GDAL_SKIP に影響を与えます。

説明

PostGIS 環境で GDAL ドライバーを有効にするコンフィギュレーションオプションです。GDAL コンフィギュレーション変数 GDAL_SKIP に影響を与えます。このオプションは、PostgreSQL のコンフィギュレーションファイル postgresql.conf で設定できます。コネクションまたはトランザクションでも設定できます。

postgis.gdal_enabled_drivers の初期値は、PostgreSQL 開始プロセスに渡される、有効とするドライバーの一覧からなる環境変数 POSTGIS_GDAL_ENABLED_DRIVERS によって設定されます。

有効にする GDAL ドライバーは、ドライバーの短縮名またはコードで指定します。ドライバーの短縮名またはコードは [GDAL Raster Formats](#) にあります。複数のドライバーを指定するには、ドライバーの間に一つの空白を置きます。

Note

postgis.gdal_enabled_drivers には三つの特別なコードがあります。大文字小文字を区別します。



- **DISABLE_ALL** 全ての GDAL ドライバーを無効にします。DISABLE_ALL は postgis.gdal_enabled_drivers にある他の全ての値を上書きします。
- **ENABLE_ALL** 全ての GDAL ドライバーを有効にします。
- **VSICURL** GDAL の仮想ファイルシステム/vsicurl/を有効にします。

postgis.gdal_enabled_drivers が **DISABLE_ALL** についていされると、データベース外ラスタ, ST_FromGDALRaster(), ST_AsGDALRaster(), ST_AsTIFF(), ST_AsJPEG(), ST_AsPNG() を使おうとすると、エラーメッセージが返されます。

**Note**

標準的な PostGIS のインストールでは、postgis.gdal_enabled_drivers は **DISABLE_ALL** に設定されます。

**Note**

GDAL_SKIP に関する追加情報は、GDAL の [Configuration Options](#)にあります。

Availability: 2.2.0

例

Set and reset postgis.gdal_enabled_drivers for the current session.

```
SET postgis.gdal_enabled_drivers = 'ENABLE_ALL';  
SET postgis.gdal_enabled_drivers = default;
```

Set specific drivers for all new connections to a database.

```
ALTER DATABASE mygisdb SET postgis.gdal_enabled_drivers TO 'GTiff PNG JPEG';
```

Set the default for the whole database cluster to enable all drivers. This requires superuser access. Also note that database, session, and user settings override this.

This example writes the setting to postgres.auto.conf.

```
ALTER SYSTEM SET postgis.gdal_enabled_drivers TO 'ENABLE_ALL';
```

Reload the PostgreSQL configuration so the changed setting is applied.

```
SELECT pg_reload_conf();
```

関連情報

[ST_FromGDALRaster](#), [ST_AsGDALRaster](#), [ST_AsTIFF](#), [ST_AsPNG](#), [ST_AsJPEG](#), [postgis.enable_outdb_raster](#)

7.22.3 postgis.enable_outdb_rasters

`postgis.enable_outdb_rasters` — データベース外ラスタのバンドにアクセスできるようにする、真偽型のコンフィギュレーションオプション。

説明

データベース外ラスタのバンドにアクセスできるようにする、真偽型のコンフィギュレーションオプションです。このオプションは、PostgreSQL のコンフィギュレーションファイル `postgresql.conf` で設定できます。コネクションまたはトランザクションでも設定できます。

`postgis.enable_outdb_rasters` の初期値は、環境変数 `POSTGIS_ENABLE_OUTDB_RASTERS` が 0 でない値で、PostgreSQL 開始プロセスに渡されることで設定されます。



Note

`postgis.enable_outdb_rasters` が TRUE であっても、GUC `postgis.gdal_enabled_drivers` は、アクセス可能なラスタ書式を判定します。



Note

標準的な PostGIS のインストールでは、`postgis.enable_outdb_rasters` は FALSE に設定されています。

Availability: 2.2.0

例

Set and reset `postgis.enable_outdb_rasters` for the current session.

```
SET postgis.enable_outdb_rasters TO True;
SET postgis.enable_outdb_rasters = default;
SET postgis.enable_outdb_rasters = True;
SET postgis.enable_outdb_rasters = False;
```

Set the default for all new connections to a database.

```
ALTER DATABASE gisdb SET postgis.enable_outdb_rasters = true;
```

Set the default for the whole database cluster. This requires superuser access. Also note that database, session, and user settings override this.

This example writes the setting to `postgres.auto.conf`.

```
ALTER SYSTEM SET postgis.enable_outdb_rasters = true;
```

Reload the PostgreSQL configuration so the changed setting is applied.

```
SELECT pg_reload_conf();
```

関連情報

[postgis.gdal_enabled_drivers](#) [postgis.gdal_vsi_options](#)

7.22.4 postgis.gdal_vsi_options

`postgis.gdal_vsi_options` — データベース外ラスタを操作する時に使用するオプションを設定するためのコンフィギュレーション。

説明

データベース外ラスタを操作する時に使用するオプションを設定するためのコンフィギュレーション。**Configuration options**によって、どれだけ GDAL がローカルデータキャッシュを確保するか、オーバビューを読むかどうか、リモートのデータベース外データソースを使う際のアクセスキー等を制御します。

Availability: 3.2.0

例

現在のセッションでの `postgis.gdal_vsi_options` の設定:

```
SET postgis.gdal_vsi_options = 'AWS_ACCESS_KEY_ID=xxxxxxxxxxxxxxxx AWS_SECRET_ACCESS_KEY= ↵
yyyyyyyyyyyyyyyyyyyyyyyyyyyyyy';
```

LOCAL キーワードを使う現在のトランザクションのための `postgis.gdal_vsi_options` の設定:

```
SET LOCAL postgis.gdal_vsi_options = 'AWS_ACCESS_KEY_ID=xxxxxxxxxxxxxxxx ↵
AWS_SECRET_ACCESS_KEY=yyyyyyyyyyyyyyyyyyyyyyyyyy';
```

関連情報

[postgis.enable_outdb_rasters](#) [postgis.gdal_enabled_drivers](#)

7.22.5 postgis.gdal_cpl_debug

`postgis.gdal_cpl_debug` — GDAL デバッグメッセージのログを取るか取らないかを設定する真偽値。

説明

デフォルトでは、ログは標準エラー出力に印字され、低レベルのデバッグメッセージは一切出力されません。この GUC を TRUE に変更すると、GDAL ログが PostgreSQL ログストリームに出力され、PostgreSQL GUC の `client_min_message` を変更することでメッセージ量を多くも少なくもできます。

Availability: 3.6.0

関連情報

[postgis.enable_outdb_rasters](#) [postgis.gdal_enabled_drivers](#)

7.23 トラブルシューティング関数

7.23.1 PostGIS_AddBBox

`PostGIS_AddBBox` — ジオメトリにバウンディングボックスを追加します。

Synopsis

geometry **PostGIS_AddBBox**(geometry geomA);

説明

ジオメトリにバウンディングボックスを追加します。これにより、バウンディングボックスに基づく検索が早くなりますが、ジオメトリのサイズが大きくなります。



Note

バウンディングボックスは自動的にジオメトリに追加されるので、通常はこの関数は不要ですが、生成されたバウンディングボックスが何らかの理由で破損するか、バウンディングボックスを欠く古い版をインストールしている場合に使われます。古いものを削除し、再追加する必要があります。



このメソッドは曲線ストリングと曲線に対応しています。

例

```
UPDATE sometable
SET geom = PostGIS_AddBBox(geom)
WHERE PostGIS_HasBBox(geom) = false;
```

関連情報

[PostGIS_DropBBox](#), [PostGIS_HasBBox](#)

7.23.2 PostGIS_DropBBox

PostGIS_DropBBox — ジオメトリからバウンディングボックスのキャッシュを削除します。

Synopsis

geometry **PostGIS_DropBBox**(geometry geomA);

説明

ジオメトリからバウンディングボックスのキャッシュを削除します。これによりジオメトリのサイズは小さくなりますが、バウンディングボックスを基にした検索が遅くなります。破損したバウンディングボックスを削除する際にも使われます。ST_Intersects や他の関係関数がジオメトリを正しく true を返すべきジオメトリを無視すると、それが、バウンディングボックスのキャッシュが破損したことを示す合図です。

Note



バウンディングボックスは自動的にジオメトリに追加されるので、通常はこの関数は不要ですが、生成されたバウンディングボックスが何らかの理由で破損するか、バウンディングボックスを欠く古い版をインストールしている場合に使われます。古いものを削除し、再追加する必要があります。この種類の破損は 8.3-8.3.6 で観察されました。ジオメトリが変更された際に常にキャッシュされたバウンディングボックスが再計算されておらず、ダンプと再読み込みを行わずに新しい版へのアップグレードを行うと、既に破損したバウンディングボックスが訂正されないためです。次に示すように手動で収集してバウンディングボックスを再追加するか、ダンプとリロードを使います。



このメソッドは曲線ストリングと曲線に対応しています。

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);
```

関連情報

[PostGIS_AddBBox](#), [PostGIS_HasBBox](#), [Box2D](#)

7.23.3 PostGIS_HasBBox

`PostGIS_HasBBox` — ジオメトリのバウンディングボックスがキャッシュされている場合には `TRUE` を返し、他の場合には `FALSE` を返します。

Synopsis

boolean **PostGIS_HasBBox**(geometry geomA);

説明

ジオメトリのバウンディングボックスがキャッシュされている場合には `TRUE` を返し、他の場合には `FALSE` を返します。[PostGIS_AddBBox](#)と[PostGIS_DropBBox](#)でバウンディングボックスのキャッシュを制御します。



このメソッドは曲線ストリングと曲線に対応しています。

例

```
SELECT geom
FROM sometable WHERE PostGIS_HasBBox(geom) = false;
```

関連情報

[PostGIS_AddBBox](#), [PostGIS_DropBBox](#)

Chapter 8

SFCGAL 関数リファレンス

SFCGAL は CGAL の C++ ラッパライブラリです。CGAL は高度な 2 次元と 3 次元の空間関数を提供します。堅牢性のためにジオメトリ座標は正確な有理数表現を持ちます。

このライブラリのインストール手順は SFCGAL サイト (<http://www.sfcgal.org>) にあります。機能を有効にするには `create extension postgis_sfcgal` とします。

8.1 SFCGAL 管理関数

8.1.1 `postgis_sfcgal_version`

`postgis_sfcgal_version` — 使用している SFCGAL のバージョンを返します

Synopsis

```
text postgis_sfcgal_version(void);
```

説明

使用している SFCGAL のバージョンを返します

Availability: 2.1.0



このメソッドには SFCGAL バックエンドが必要です。

関連情報

[`postgis\_sfcgal\_full\_version`](#)

8.1.2 `postgis_sfcgal_full_version`

`postgis_sfcgal_full_version` — CGAL と Boost のバージョンを含む、使用している SFCGAL の完全なバージョンを返します

Synopsis

```
text postgis_sfcgal_full_version(void);
```

説明

CGAL と Boost のバージョンを含む、使用している SFCGAL の完全なバージョンを返します

Availability: 3.3.0



このメソッドには SFCGAL バックエンドが必要です。

関連情報

[postgis_sfcgal_version](#)

8.2 SFCGAL アクセサとセッター

8.2.1 CG_ForceLHR

CG_ForceLHR — LHR (Left Hand Rule) 方向に強制します。

Synopsis

```
geometry CG_ForceLHR(geometry geom);
```

説明

Availability: 3.5.0



このメソッドには SFCGAL バックエンドが必要です。



この関数は 3 次元に対応し、Z 値を削除しません。



この関数は多面体サーフェスに対応しています。



この関数は三角形と不規則三角網 (TIN) に対応しています。

8.2.2 CG_IsPlanar

CG_IsPlanar — サーフェスが平面であるかないかをチェックします。

Synopsis

```
boolean CG_IsPlanar(geometry geom);
```

説明

Availability: 3.5.0

- ✔ このメソッドには SFCGAL バックエンドが必要です。
- ✔ この関数は 3 次元に対応し、Z 値を削除しません。
- ✔ この関数は多面体サーフェスに対応しています。
- ✔ この関数は三角形と不規則三角網 (TIN) に対応しています。

8.2.3 CG_IsSolid

CG_IsSolid — ジオメトリが立体であるかどうかをテストします。妥当性チェックは行いません。

Synopsis

```
boolean CG_IsSolid(geometry geom1);
```

説明

Availability: 3.5.0

- ✔ このメソッドには SFCGAL バックエンドが必要です。
- ✔ この関数は 3 次元に対応し、Z 値を削除しません。
- ✔ この関数は多面体サーフェスに対応しています。
- ✔ この関数は三角形と不規則三角網 (TIN) に対応しています。

8.2.4 CG_MakeSolid

CG_MakeSolid — ジオメトリを立体にキャストします。チェックはしません。妥当な立体を得るには、入力ジオメトリは閉じた多面体サーフェスか閉じた TIN でなければなりません。

Synopsis

```
geometry CG_MakeSolid(geometry geom1);
```

説明

Availability: 3.5.0

- ✔ このメソッドには SFCGAL バックエンドが必要です。
 - ✔ この関数は 3 次元に対応し、Z 値を削除しません。
 - ✔ この関数は多面体サーフェスに対応しています。
 - ✔ この関数は三角形と不規則三角網 (TIN) に対応しています。
-

8.2.5 CG_Orientation

CG_Orientation — サーフェスの方向を判定します。

Synopsis

```
integer CG_Orientation(geometry geom);
```

説明

この関数はポリゴンのみ受け付けます。ポリゴンが反時計回りなら-1 を返し、時計回りなら 1 を返します。

Availability: 3.5.0

- ✔ このメソッドには SFCGAL バックエンドが必要です。
- ✔ この関数は 3 次元に対応し、Z 値を削除しません。

8.2.6 CG_Area

CG_Area — ジオメトリの面積を計算します

Synopsis

```
double precision CG_Area( geometry geom );
```

説明

ジオメトリの面積を計算します。

SFCGAL モジュールによって実行されます



Note

ご注意: この関数は、面積を表現する倍精度浮動小数点数を返します。

Availability: 3.5.0

- ✔ このメソッドには SFCGAL バックエンドが必要です。

ジオメトリの例

```
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))');
```

関連情報

[ST_3DArea](#), [ST_Area](#)

8.2.7 CG_3DArea

CG_3DArea — 3 次元の面ジオメトリの面積を計算します。立体の場合は 0 を返します。

Synopsis

```
float CG_3DArea(geometry geom1);
```

説明

Availability: 3.5.0

- ✔ このメソッドには SFCGAL バックエンドが必要です。
- ✔ このメソッドは SQL/MM 仕様の実装です。SQL-MM IEC 13249-3: 8.1, 10.5
- ✔ この関数は 3 次元に対応し、Z 値を削除しません。
- ✔ この関数は多面体サーフェスに対応しています。
- ✔ この関数は三角形と不規則三角網 (TIN) に対応しています。

例

ご注意: デフォルトでは、WKT から生成された **PolyhedralSurface** は面ジオメトリで、立体ではありません。サーフェス面を持ちます。立体に変換すると、面を持ちません。

```
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
```

関連情報

[CG_Area](#), [CG_MakeSolid](#), [CG_IsSolid](#), [CG_Area](#)

8.2.8 CG_Volume

CG_Volume — Computes the volume of a 3D geometry. Closed surfaces can have non-zero volume.

Synopsis

float **CG_Volume**(geometry geom1);

説明

Availability: 3.5.0

- ✔ このメソッドには SFCGAL バックエンドが必要です。
- ✔ この関数は 3 次元に対応し、Z 値を削除しません。
- ✔ この関数は多面体サーフェスに対応しています。
- ✔ この関数は三角形と不規則三角網 (TIN) に対応しています。
- ✔ このメソッドは SQL/MM 仕様の実装です。SQL-MM IEC 13249-3: 9.1 (CG_3DVolume と同じ)

例

A closed polyhedral surface has volume. **CG_MakeSolid** makes the solid representation explicit.

```
SELECT CG_Volume(geom) As cube_surface_vol,
       CG_Volume(CG_MakeSolid(geom)) As solid_surface_vol
FROM (SELECT 'POLYHEDRALSURFACE( ((0 0 0, 0 0 1, 0 1 1, 0 1 0, 0 0 0)),
  ((0 0 0, 0 1 0, 1 1 0, 1 0 0, 0 0 0)),
  ((0 0 0, 1 0 0, 1 0 1, 0 0 1, 0 0 0)),
  ((1 1 0, 1 1 1, 1 0 1, 1 0 0, 1 1 0)),
  ((0 1 0, 0 1 1, 1 1 1, 1 1 0, 0 1 0)),
  ((0 0 1, 1 0 1, 1 1 1, 0 1 1, 0 0 1)) )'::geometry) As f(geom);
```

```
cube_surface_vol | solid_surface_vol
-----+-----
1 | 1
```

関連情報

CG_3DArea, **CG_MakeSolid**, **CG_IsSolid**

8.2.9 ST_ForceLHR

ST_ForceLHR — LHR (Left Hand Rule) 方向に強制します。

Synopsis

geometry **ST_ForceLHR**(geometry geom);

説明

**Warning**

ST_ForceLHR は 3.5.0 では非推奨です。 **CG_ForceLHR**を代わりに使います。

Availability: 2.1.0



このメソッドには SFCGAL バックエンドが必要です。



この関数は 3 次元に対応し、Z 値を削除しません。



この関数は多面体サーフェスに対応しています。



この関数は三角形と不規則三角網 (TIN) に対応しています。

8.2.10 ST_IsPlanar

ST_IsPlanar — サーフェスが平面であるかないかをチェックします。

Synopsis

```
boolean ST_IsPlanar(geometry geom);
```

説明

**Warning**

ST_IsPlanar は 3.5.0 では非推奨です。 **CG_IsPlanar**を代わりに使います。

Availability: 2.2.0: これは 2.1.0 のマニュアルに記述されていましたが、2.1 版では偶然に外れてしまいました。



このメソッドには SFCGAL バックエンドが必要です。



この関数は 3 次元に対応し、Z 値を削除しません。



この関数は多面体サーフェスに対応しています。



この関数は三角形と不規則三角網 (TIN) に対応しています。

8.2.11 ST_IsSolid

ST_IsSolid — ジオメトリが立体であるかどうかをテストします。妥当性チェックは行いません。

Synopsis

```
boolean ST_IsSolid(geometry geom1);
```

説明

**Warning**

ST_IsSolid は 3.5.0 では非推奨です。 **CG_IsSolid** を代わりに使います。

Availability: 2.2.0



このメソッドには SFCGAL バックエンドが必要です。



この関数は 3 次元に対応し、Z 値を削除しません。



この関数は多面体サーフェスに対応しています。



この関数は三角形と不規則三角網 (TIN) に対応しています。

8.2.12 ST_MakeSolid

ST_MakeSolid — ジオメトリを立体にキャストします。チェックはしません。妥当な立体を得るには、入力ジオメトリは閉じた多面体サーフェスか閉じた TIN でなければなりません。

Synopsis

```
geometry ST_MakeSolid(geometry geom1);
```

説明

**Warning**

ST_MakeSolid は 3.5.0 では非推奨です。 **CG_MakeSolid** を代わりに使います。

Availability: 2.2.0



このメソッドには SFCGAL バックエンドが必要です。



この関数は 3 次元に対応し、Z 値を削除しません。



この関数は多面体サーフェスに対応しています。



この関数は三角形と不規則三角網 (TIN) に対応しています。

8.2.13 ST_Orientation

ST_Orientation — サーフェスの方向を判定します。

Synopsis

```
integer ST_Orientation(geometry geom);
```

説明

**Warning**

ST_Orientation は 3.5.0 では非推奨です。 **CG_Orientation** を代わりに使います。

この関数はポリゴンのみ受け付けます。ポリゴンが反時計回りなら -1 を返し、時計回りなら 1 を返します。

Availability: 2.1.0



このメソッドには SFCGAL バックエンドが必要です。



この関数は 3 次元に対応し、Z 値を削除しません。

8.2.14 ST_3DArea

ST_3DArea — 3 次元の面ジオメトリの面積を計算します。立体の場合は 0 を返します。

Synopsis

```
floatST_3DArea(geometry geom1);
```

説明

**Warning**

ST_3DArea は 3.5.0 では非推奨です。 **CG_3DArea** を代わりに使います。

Availability: 2.1.0



このメソッドには SFCGAL バックエンドが必要です。



このメソッドは SQL/MM 仕様の実装です。SQL-MM IEC 13249-3: 8.1, 10.5



この関数は 3 次元に対応し、Z 値を削除しません。



この関数は多面体サーフェスに対応しています。



この関数は三角形と不規則三角網 (TIN) に対応しています。

例

ご注意: デフォルトでは、WKT から生成された **PolyhedralSurface** は面ジオメトリで、立体ではありません。サーフェス面を持ちます。立体に変換すると、面を持ちません。

```
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
```

関連情報

[ST_Area](#), [ST_MakeSolid](#), [ST_IsSolid](#), [ST_Area](#)

8.2.15 ST_Volume

ST_Volume — Computes the volume of a 3D geometry. Closed surfaces can have non-zero volume.

Synopsis

```
float ST_Volume(geometry geom1);
```

説明



Warning

ST_Volume は 3.5.0 では非推奨です。 **CG_Volume** を代わりに使います。

Availability: 2.2.0

- ✔ このメソッドには SFCGAL バックエンドが必要です。
- ✔ この関数は 3 次元に対応し、Z 値を削除しません。
- ✔ この関数は多面体サーフェスに対応しています。
- ✔ この関数は三角形と不規則三角網 (TIN) に対応しています。
- ✔ このメソッドは SQL/MM 仕様の実装です。SQL-MM IEC 13249-3: 9.1 (ST_3DVolume と同じ)

例

A closed polyhedral surface has volume. **ST_MakeSolid** makes the solid representation explicit.


```
SELECT ST_Volume(geom) As cube_surface_vol,
       ST_Volume(ST_MakeSolid(geom)) As solid_surface_vol
FROM (SELECT 'POLYHEDRALSURFACE( ((0 0 0, 0 0 1, 0 1 1, 0 1 0, 0 0 0)),
  ((0 0 0, 0 1 0, 1 1 0, 1 0 0, 0 0 0)),
  ((0 0 0, 1 0 0, 1 0 1, 0 0 1, 0 0 0)),
  ((1 1 0, 1 1 1, 1 0 1, 1 0 0, 1 1 0)),
  ((0 1 0, 0 1 1, 1 1 1, 1 1 0, 0 1 0)),
  ((0 0 1, 1 0 1, 1 1 1, 0 1 1, 0 0 1)) )'::geometry) As f(geom);
```

```
cube_surface_vol | solid_surface_vol
-----+-----
1 | 1
```

関連情報

[ST_3DArea](#), [ST_MakeSolid](#), [ST_IsSolid](#)

8.3 SFCGAL 処理関数および関係関数

8.3.1 CG_Intersection

CG_Intersection — 二つのジオメトリのインタセクトする (共有する) 部分を計算します

Synopsis

```
geometry CG_Intersection( geometry geomA , geometry geomB );
```

説明

二つのジオメトリのインタセクトする (共有する) 部分を計算します。

SFCGAL モジュールによって実行されます



Note

ご注意: この関数はインタセクトした部分を表現するジオメトリを返します。

Availability: 3.5.0



このメソッドには SFCGAL バックエンドが必要です。

ジオメトリの例

```
SELECT ST_AsText(CG_Intersection('LINESTRING(0 0, 5 5)', 'LINESTRING(5 0, 0 5)'));
```

```
POINT(2.5 2.5)
```

関連情報

[ST_3DIntersection](#), [ST_Intersection](#)

8.3.2 CG_Intersects

CG_Intersects — 二つのジオメトリがインタセクトしている (少なくとも一つの共有点がある) かどうかテストします。

Synopsis

```
boolean CG_Intersects( geometry geomA , geometry geomB );
```

説明

二つのジオメトリがインタセクトする場合に **TRUE** を返します。任意の共有点を持つ場合を指します。
SFCGAL モジュールによって実行されます



Note

ご注意: これは論理値を返して整数を返さないのが「許される」版です。

Availability: 3.5.0



このメソッドには SFCGAL バックエンドが必要です。



この関数は三角形と不規則三角網 (TIN) に対応しています。

ジオメトリの例

```
SELECT CG_Intersects('POINT(0 0)::geometry, 'LINESTRING ( 2 0, 0 2 ) '::geometry);
```

```
f
```

```
SELECT CG_Intersects('POINT(0 0)::geometry, 'LINESTRING ( 0 0, 0 2 ) '::geometry);
```

```
t
```

関連情報

[CG_3DIntersects](#), [ST_3DIntersects](#), [ST_Intersects](#), [ST_Disjoint](#)

8.3.3 CG_3DIntersects

CG_3DIntersects — 二つの 3 次元ジオメトリがインタセクトするかどうかをテストします

Synopsis

boolean **CG_3DIntersects**(geometry geomA , geometry geomB);

説明

二つの 3 次元ジオメトリがインタセクトするかどうかをテストします。3 次元空間で共通する点が存在するなら、3 次元ジオメトリはインタセクトしています。

SFCGAL モジュールによって実行されます



Note

ご注意: これは論理値を返して整数を返さないのが「許される」版です。

Availability: 3.5.0



このメソッドには SFCGAL バックエンドが必要です。



この関数は三角形と不規則三角網 (TIN) に対応しています。

ジオメトリの例

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

t

関連情報

[CG_Intersects](#), [ST_3DIntersects](#), [ST_Intersects](#), [ST_Disjoint](#)

8.3.4 CG_Difference

CG_Difference — 二つのジオメトリの幾何学的な差を計算します

Synopsis

geometry **CG_Difference**(geometry geomA , geometry geomB);

説明

二つのジオメトリの幾何学的な差を計算します。返されるジオメトリは、geomA に存在するが geomB に存在しない点の集合です。

SFCGAL モジュールによって実行されます

**Note**

ご注意: この関数はジオメトリを返します。

Availability: 3.5.0



このメソッドには SFCGAL バックエンドが必要です。



この関数は三角形と不規則三角網 (TIN) に対応しています。

ジオメトリの例

```
SELECT ST_AsText(CG_Difference('POLYGON((0 0, 0 1, 1 1, 1 0, 0 0))'::geometry, 'LINESTRING ←  
(0 0, 2 2)'::geometry));
```

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

関連情報

[ST_3DDifference](#), [ST_Difference](#)

8.3.5 ST_3DDifference

ST_3DDifference — 3 次元の差分を計算します。

Synopsis

geometry **ST_3DDifference**(geometry geom1, geometry geom2);

説明

**Warning**

ST_3DDifference は 3.5.0 では非推奨です。 **CG_3DDifference** を代わりに使います。

geom2 に含まれない geom1 の一部を返します。

Availability: 2.2.0



このメソッドには SFCGAL バックエンドが必要です。



このメソッドは SQL/MM 仕様の実装です。SQL-MM IEC 13249-3: 5.1



この関数は 3 次元に対応し、Z 値を削除しません。



この関数は多面体サーフェスに対応しています。



この関数は三角形と不規則三角網 (TIN) に対応しています。

8.3.6 CG_3DDifference

CG_3DDifference — 3 次元の差分を計算します。

Synopsis

geometry **CG_3DDifference**(geometry geom1, geometry geom2);

説明

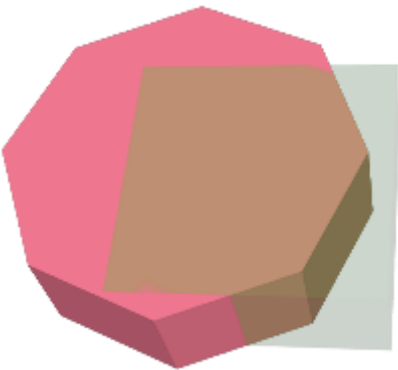
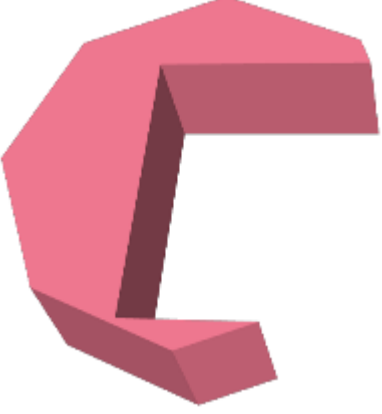
geom2 に含まれない geom1 の一部を返します。

Availability: 3.5.0

- ✓ このメソッドには SFCGAL バックエンドが必要です。
- ✓ このメソッドは SQL/MM 仕様の実装です。SQL-MM IEC 13249-3: 5.1
- ✓ この関数は 3 次元に対応し、Z 値を削除しません。
- ✓ この関数は多面体サーフェスに対応しています。
- ✓ この関数は三角形と不規則三角網 (TIN) に対応しています。

例

PostGIS 関数 **ST_AsX3D** を使って 3 次元イメージを生成し、 **X3Dom HTML Javascript redering library** を使って HTML での描画を行います。

<pre>SELECT CG_Extrude(ST_Buffer(↵ ST_GeomFromText('POINT(100 90)'), 50, 'quad_segs=2'), 0, 0, 30) AS geom1,↵ CG_Extrude(ST_Buffer(ST_GeomFromText('POINT(80 80)'), 50, 'quad_segs=2'), 0, 0, 30) AS geom2;</pre>  元の 3 次元ジオメトリを重ねたもの。geom2 は削除部分にあたります。	<pre>SELECT CG_3DDifference(geom1,geom2)</pre>  geom2 を削除した後に残るもの	<pre>CG_ geo</pre>

関連情報

[CG_Extrude](#), [ST_AsX3D](#), [CG_3DIntersection](#) [CG_3DUnion](#)

8.3.7 CG_Distance

CG_Distance — 二つのジオメトリの最小距離を返します

Synopsis

```
double precision CG_Distance( geometry geomA , geometry geomB );
```

説明

二つのジオメトリの最小距離を返します。

SFCGAL モジュールによって実行されます



Note

ご注意: この関数は、距離を表現する倍精度浮動小数点数を返します。

Availability: 3.5.0



このメソッドには SFCGAL バックエンドが必要です。



この関数は三角形と不規則三角網 (TIN) に対応しています。

ジオメトリの例

```
SELECT CG_Distance('LINESTRING(0.0 0.0,-1.0 -1.0)', 'LINESTRING(3.0 4.0,4.0 5.0)');
```

```
5
```

関連情報

[CG_3DDistance](#), [CG_Distance](#)

8.3.8 CG_3DDistance

CG_3DDistance — 二つのジオメトリの最小 3 次元距離を返します

Synopsis

```
double precision CG_3DDistance( geometry geomA , geometry geomB );
```

説明

二つのジオメトリの最小 3 次元距離を返します。
SFCGAL モジュールによって実行されます



Note

ご注意: この関数は、3 次元距離を表現する倍精度浮動小数点数を返します。

Availability: 3.5.0



このメソッドには SFCGAL バックエンドが必要です。



この関数は三角形と不規則三角網 (TIN) に対応しています。

ジオメトリの例

```
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))'); ↵
```

1

関連情報

[CG_Distance](#), [ST_3DDistance](#)

8.3.9 ST_3DConvexHull

ST_3DConvexHull — ジオメトリの 3 次元の凸包を計算します。

Synopsis

geometry **ST_3DConvexHull**(geometry geom1);

説明



Warning

ST_3DConvexHull は 3.5.0 では非推奨です。 [CG_3DConvexHull](#) を代わりに使います。

Availability: 3.3.0



このメソッドには SFCGAL バックエンドが必要です。



この関数は 3 次元に対応し、Z 値を削除しません。



この関数は多面体サーフェスに対応しています。



この関数は三角形と不規則三角網 (TIN) に対応しています。

8.3.10 CG_3DConvexHull

CG_3DConvexHull — ジオメトリの 3 次元の凸包を計算します。

Synopsis

```
geometry CG_3DConvexHull(geometry geom1);
```

説明

Availability: 3.5.0

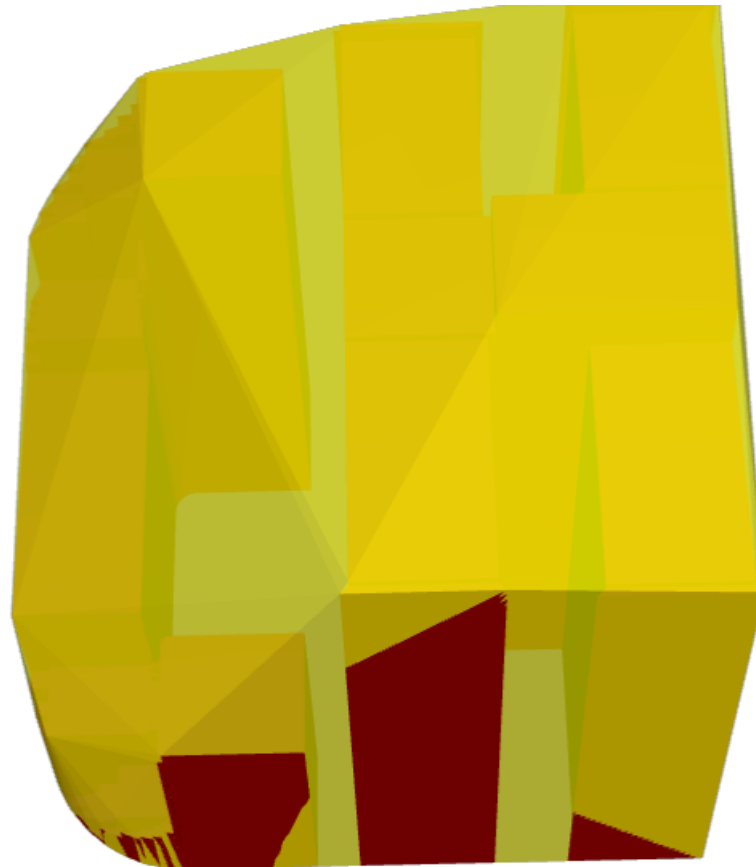
- ✔ このメソッドには SFCGAL バックエンドが必要です。
- ✔ この関数は 3 次元に対応し、Z 値を削除しません。
- ✔ この関数は多面体サーフェスに対応しています。
- ✔ この関数は三角形と不規則三角網 (TIN) に対応しています。

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;
```

3次元凸包をオーバーレイした元のジオメトリ

関連情報

[ST_Letters](#), [ST_AsX3D](#)

8.3.11 ST_3DIntersection

ST_3DIntersection — 3次元のインタセクトした (共有する) 部分を計算します。

Synopsis

```
geometry ST_3DIntersection(geometry geom1, geometry geom2);
```

説明



Warning

ST_3DIntersection は 3.5.0 では非推奨です。 **CG_3DIntersection** を代わりに使います。

geom1 と geom2 の間で共有される部分のジオメトリを返します。

Availability: 2.1.0

- ✔ このメソッドには SFCGAL バックエンドが必要です。
- ✔ このメソッドは SQL/MM 仕様の実装です。SQL-MM IEC 13249-3: 5.1
- ✔ この関数は 3 次元に対応し、Z 値を削除しません。
- ✔ この関数は多面体サーフェスに対応しています。
- ✔ この関数は三角形と不規則三角網 (TIN) に対応しています。

8.3.12 CG_3DIntersection

CG_3DIntersection — 3 次元のインタセクトした (共有する) 部分を計算します。

Synopsis

```
geometry CG_3DIntersection(geometry geom1, geometry geom2);
```

説明

geom1 と geom2 の間で共有される部分のジオメトリを返します。

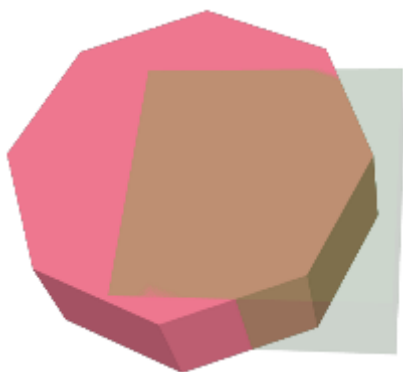
Availability: 3.5.0

- ✔ このメソッドには SFCGAL バックエンドが必要です。
- ✔ このメソッドは SQL/MM 仕様の実装です。SQL-MM IEC 13249-3: 5.1
- ✔ この関数は 3 次元に対応し、Z 値を削除しません。
- ✔ この関数は多面体サーフェスに対応しています。
- ✔ この関数は三角形と不規則三角網 (TIN) に対応しています。

例

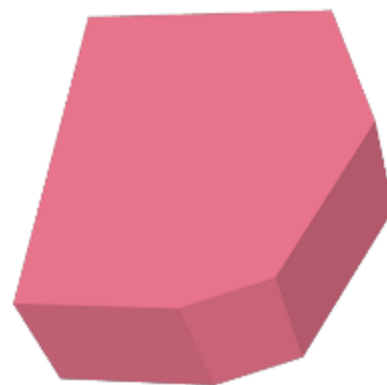
PostGIS 関数 **ST_AsX3D** を使って 3 次元イメージを生成し、**X3Dom HTML Javascript redering library** を使って HTML での描画を行います。

```
SELECT CG_Extrude(ST_Buffer(
    ST_GeomFromText('POINT(100 90)'), 50, 'quad_segs=2'), 0, 0, 30) AS geom1,
    CG_Extrude(ST_Buffer(ST_GeomFromText('POINT(80 80)'), 50, 'quad_segs=1'), 0, 0, 30) AS geom2;
```



元の 3 次元ジオメトリを重ねたもので *geom2* は半透明で表示

```
SELECT CG_3DIntersection(geom1,geom2)
    CG_Extrude(ST_Buffer(ST_GeomFromText('POINT(80 80)'), 50, 'quad_segs=1'), 0, 0, 30) AS geom2 ) As t;
```



geom1 と *geom2* の共有部分

3 次元ラインストリングとポリゴン

```
SELECT ST_AsText(CG_3DIntersection(linestring, polygon)) As wkt
FROM ST_GeomFromText('LINESTRING Z (2 2 6,1.5 1.5 7,1 1 8,0.5 0.5 8,0 0 10)') AS
linestring
CROSS JOIN ST_GeomFromText('POLYGON((0 0 8, 0 1 8, 1 1 8, 1 0 8, 0 0 8))') AS polygon;
```

```
LINESTRING Z (1 1 8,0.5 0.5 8)
```

立方体 (閉じた多面体サーフェス) と 3 次元ポリゴン

```
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)))
```

二つの立体の共通部分もまた立体です (ST_Dimension で 3 を返します)

```
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 0,9.9999999999995 30 0,16.6666666666667 23.3333333333333 10,20 20 0)),
((10 10 0,9.9999999999995 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 10,9.9999999999995 30 0,12 28 ←
  10,16.6666666666667 23.3333333333333 10)),
((10 10 0,0 20 0,9.9999999999995 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)),
((9.9999999999995 30 0,11 29 10,12 28 10,9.9999999999995 30 0)),((0 20 0,2 20 ←
  10,9.9999999999995 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)),
((9.9999999999995 30 0,2 20 10,11 29 10,9.9999999999995 30 0)),((11 11 10,11 29 ←
  10,2 20 10,11 11 10)))
```

8.3.13 CG_Union

CG_Union — 二つのジオメトリの結合を計算します

Synopsis

geometry **CG_Union**(geometry geomA , geometry geomB);

説明

二つのジオメトリの結合を計算します。

SFCGAL モジュールによって実行されます



Note

ご注意: この関数は結合を表現するジオメトリを返します。

Availability: 3.5.0



このメソッドには SFCGAL バックエンドが必要です。

ジオメトリの例

```
SELECT ST_AsText(CG_Union('POINT(.5 0)', 'LINESTRING(-1 0,1 0)'));
```

```
LINESTRING(-1 0,0.5 0,1 0)
```

関連情報

[ST_3DUnion](#), [ST_Union](#)

8.3.14 ST_3DUnion

ST_3DUnion — 3 次元の結合を計算します。

Synopsis

```
geometry ST_3DUnion(geometry geom1, geometry geom2);  
geometry ST_3DUnion(geometry set g1field);
```

説明



Warning

ST_3DUnion は 3.5.0 では非推奨です。 **CG_3DUnion** を代わりに使います。

Availability: 2.2.0

Availability: 3.3.0 集約関数の形式が追加されました

- ✔ このメソッドには SFCGAL バックエンドが必要です。
- ✔ このメソッドは SQL/MM 仕様の実装です。SQL-MM IEC 13249-3: 5.1
- ✔ この関数は 3 次元に対応し、Z 値を削除しません。
- ✔ この関数は多面体サーフェスに対応しています。
- ✔ この関数は三角形と不規則三角網 (TIN) に対応しています。

集約関数形式: ジオメトリの集合を 3 次元結合したジオメトリを返します。ST_3DUnion() 関数は、PostgreSQL 用語で言うところの「集約関数」です。つまり、SUM() や MEAN() と同じ方法で複数のデータ行の操作を行い、他の集約関数と同じように NULL ジオメトリを無視します。

8.3.15 CG_3DUnion

CG_3DUnion — postgis_sfcgal を使用して 3 次元結合を実行します。

Synopsis

```
geometry CG_3DUnion(geometry geom1, geometry geom2);  
geometry CG_3DUnion(geometry set g1field);
```

説明

Availability: 3.5.0

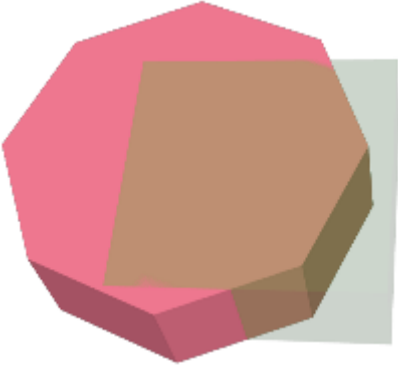
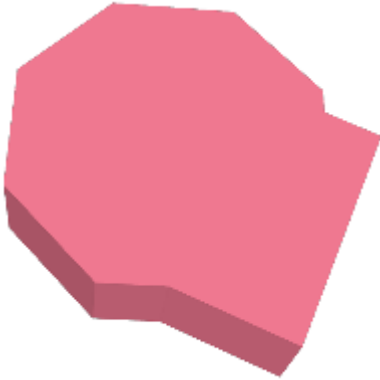
- ✔ このメソッドには SFCGAL バックエンドが必要です。
- ✔ このメソッドは SQL/MM 仕様の実装です。SQL-MM IEC 13249-3: 5.1

- ✔ この関数は 3 次元に対応し、Z 値を削除しません。
- ✔ この関数は多面体サーフェスに対応しています。
- ✔ この関数は三角形と不規則三角網 (TIN) に対応しています。

集約関数形式: ジオメトリの行集合を結合した 3 次元ジオメトリを返します。`CG_3DUnion()` 関数は、PostgreSQL 用語で言うところの「集約関数」です。つまり、`SUM()` や `MEAN()` と同じ方法で複数のデータ行の操作を行い、他のほとんどの集約関数と同じように `NULL` ジオメトリを無視します。

例

PostGIS 関数 `ST_AsX3D` を使って 3 次元イメージを生成し、`X3Dom HTML Javascript redering library` を使って HTML で の描画を行います。

<pre>SELECT CG_Extrude(ST_Buffer(↔ ST_GeomFromText('POINT(100 90)'), 50, 'quad_segs=2'), 0, 0, 30) AS geom1, ↖ CG_Extrude(ST_Buffer(ST_GeomFromText('POINT(80 80)'), 50, 'quad_segs=2'), 0, 0, 30) AS geom2; ↗</pre>  元の 3 次元ジオメトリを重ねたもの。 <i>geom2</i> は半透明で示しています。	<pre>SELECT CG_3DUnion(geom1,geom2) ↖ ↗</pre>  <i>geom1</i> と <i>geom2</i> の結合	<pre>SELECT CG_3DIntersection(geom1,geom2) ↖ ↗</pre> 3D visualization of the intersection of the two polygons.

関連情報

`CG_Extrude`, `ST_AsX3D`, `CG_3DIntersection` `CG_3DDifference`

8.3.16 ST_AlphaShape

`ST_AlphaShape` — ジオメトリを囲むアルファシェイプを計算します。

Synopsis

geometry **ST_AlphaShape**(geometry geom, float alpha, boolean allow_holes = false);

説明

**Warning**

ST_AlphaShape は 3.5.0 では非推奨です。代わりに **CG_AlphaShape** を使います。

ジオメトリのポイントの **アルファシェイプ** を計算します。アルファシェイプは (通常は) 入力 of 頂点を全て含む凹ポリゴンジオメトリで、このジオメトリの頂点は入力頂点の部分集合です。アルファシェイプは入力ジオメトリのシェイプを **凸包** で生成されるシェイプよりも適合したものにします。

8.3.17 CG_AlphaShape

CG_AlphaShape — ジオメトリを囲むアルファシェイプを計算します。

Synopsis

```
geometry CG_AlphaShape(geometry geom, float alpha, boolean allow_holes = false);
```

説明

ジオメトリのポイントの **アルファシェイプ** を計算します。アルファシェイプは (通常は) 入力 of 頂点を全て含む凹ポリゴンジオメトリで、このジオメトリの頂点は入力頂点の部分集合です。アルファシェイプは入力ジオメトリのシェイプを **凸包** で生成されるシェイプよりも適合したものにします。

「適合度」は **alpha** パラメータで制御します。このパラメータは 0 から無限大の数を取ります。小さい α 値では凹が増える結果になります α 値がデータ依存の値より大きいと入力の凸包が生成されます。

**Note**

CGAL 実装に従うと、 α 値は、アルファシェイプアルゴリズムで、入力ジオメトリのドロネー三角形を侵食するために使われる円の半径の 2 乗です。詳細情報については **CGAL Alpha-Shapes** をご覧ください。元のアルファシェイプの定義と違います。元は α 値を侵食円の半径と定義しています。

計算されたシェイプは **allow_holes** を TRUE に設定しない限りは穴を持ちません。

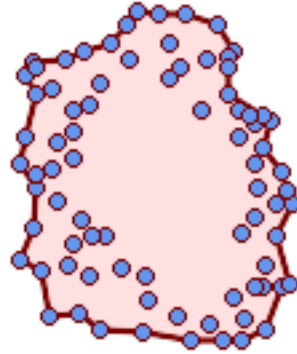
この関数はジオメトリの凹包を **ST_ConcaveHull** と似た方法で効率的に計算しますが、CGAL や異なるアルゴリズムを使います。

Availability: 3.5.0 - SFCGAL $\geq$ 1.4.1 が必要です。



このメソッドには SFCGAL バックエンドが必要です。

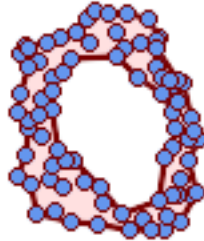
例



MULTIPOINTO のアルファシェイプ (*CG_OptimalAlphaShape*と同じ)

```
SELECT ST_AsText(
  CG_AlphaShape(
    'MULTIPOINT((63 84),(76 88),(68 73),(53 18),(91 50),(81 70),
      (88 29),(24 82),(32 51),(37 23),(27 54),(84 19),(75 87),(44 42),(77 67),(90 30),(36 ←
        61),(32 65),
      (81 47),(88 58),(68 73),(49 95),(81 60),(87 50),
      (78 16),(79 21),(30 22),(78 43),(26 85),(48 34),(35 35),(36 40),(31 79),(83 29),(27 ←
        84),(52 98),(72 95),(85 71),
      (75 84),(75 77),(81 29),(77 73),(41 42),(83 72),(23 36),(89 53),(27 57),(57 97),(27 ←
        77),(39 88),(60 81),
      (80 72),(54 32),(55 26),(62 22),(70 20),(76 27),(84 35),(87 42),(82 54),(83 64),(69 ←
        86),(60 90),(50 86),(43 80),(36 73),
      (36 68),(40 75),(24 67),(23 60),(26 44),(28 33),(40 32),(43 19),(65 16),(73 16),(38 ←
        46),(31 59),(34 86),(45 90),(64 97))'::geometry,
    80.2));
```

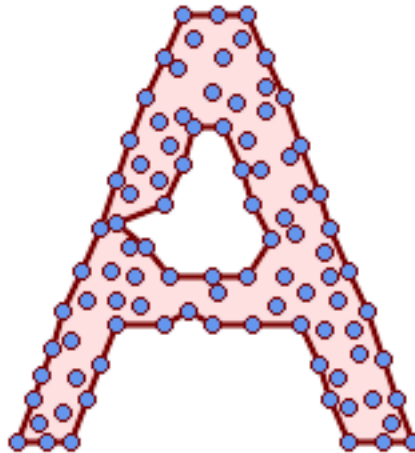
```
MULTIPOLYGON(((89 53,88 58,85 71,83 72,75 84,76 88,72 95,64 97,57 97,52 98,49 95,45 90,39 ←
  88,34 86,26 85,24 82,27 77,24 67,23 60,27 54,26 44,23 36,28 33,30 22,37 23,43 19,53 ←
  18,65 16,73 16,78 16,84 19,88 29,90 30,87 42,91 50,89 53)))
```

穴を許容した *MULTIPOINTO* のアルファシェイプ (*CG_OptimalAlphaShape*と同じ)

```
SELECT ST_AsText(
  CG_AlphaShape(
    'MULTIPOINT((63 84),(76 88),(68 73),(53 18),(91 50),(81 70),(88 29),(24 82),(32 51),(37 ←
      23),(27 54),(84 19),(75 87),(44 42),(77 67),(90 30),(36 61),(32 65),(81 47),(88 58) ←
      ,(68 73),(49 95),(81 60),(87 50),
      (78 16),(79 21),(30 22),(78 43),(26 85),(48 34),(35 35),(36 40),(31 79),(83 29),(27 84) ←
      ,(52 98),(72 95),(85 71),
      (75 84),(75 77),(81 29),(77 73),(41 42),(83 72),(23 36),(89 53),(27 57),(57 97),(27 77) ←
      ,(39 88),(60 81),
      (80 72),(54 32),(55 26),(62 22),(70 20),(76 27),(84 35),(87 42),(82 54),(83 64),(69 86) ←
      ,(60 90),(50 86),(43 80),(36 73),
      (36 68),(40 75),(24 67),(23 60),(26 44),(28 33),(40 32),(43 19),(65 16),(73 16),(38 46) ←
      ,(31 59),(34 86),(45 90),(64 97))'::geometry,
    100.1,
    true))

MULTIPOLYGON(((89 53,88 58,85 71,83 72,75 84,76 88,72 95,64 97,57 97,52 98,49 95,45 90,39 ←
  88,34 86,26 85,24 82,27 77,24 67,23 60,27 54,26 44,23 36,28 33,30 22,43 19,53 18,65 ←
  16,73 16,78 16,84 19,90 30,87 42,91 50,89 53),(36 68,40 75,43 80,60 81,68 73,77 67,81 ←
  60,82 54,81 47,78 43,76 27,62 22,54 32,44 42,38 46,36 61,36 68)))
```



マルチポイントのアルファシェイプで穴を許す (*ST_ConcaveHull*と同じ)

```
SELECT ST_AsText(
  CG_AlphaShape(
    'MULTIPOINT ((132 64), (114 64), (99 64), (81 64), (63 64), (57 49), (52 36), (46 20), (37 20), (26 20), (32 36), (39 55), (43 69), (50 84), (57 100), (63 118), (68 133), (74 149), (81 164), (88 180), (101 180), (112 180), (119 164), (126 149), (132 131), (139 113), (143 100), (150 84), (157 69), (163 51), (168 36), (174 20), (163 20), (150 20), (143 36), (139 49), (132 64), (99 151), (92 138), (88 124), (81 109), (74 93), (70 82), (83 82), (99 82), (112 82), (126 82), (121 96), (114 109), (110 122), (103 138), (99 151), (34 27), (43 31), (48 44), (46 58), (52 73), (63 73), (61 84), (72 71), (90 69), (101 76), (123 71), (141 62), (166 27), (150 33), (159 36), (146 44), (154 53), (152 62), (146 73), (134 76), (143 82), (141 91), (130 98), (126 104), (132 113), (128 127), (117 122), (112 133), (119 144), (108 147), (119 153), (110 171), (103 164), (92 171), (86 160), (88 142), (79 140), (72 124), (83 131), (79 118), (68 113), (63 102), (68 93), (35 45))'::geometry,
    102.2,
    true));

MULTIPOLYGON(((168 36,163 51,157 69,150 84,143 100,139 113,132 131,126 149,119 164,112 180,101 180,88 180,81 164,74 149,68 133,63 118,57 100,50 84,43 69,39 55,35 45,32 36,26 20,37 20,46 20,52 36,57 49,63 64,81 64,90 69,99 64,114 64,132 64,139 49,143 36,150 20,163 20,174 20,168 36),(81 109,88 124,92 138,103 138,110 122,114 109,121 96,112 82,99 82,83 82,74 93,81 109)))
```

関連情報

[ST_ConcaveHull](#), [CG_OptimalAlphaShape](#)

8.3.18 CG_ApproxConvexPartition

CG_ApproxConvexPartition — ポリゴンジオメトリの近似凸分割を計算します

Synopsis

geometry **CG_ApproxConvexPartition**(geometry geom);

説明

ポリゴンジオメトリの近似凸分割を計算します (三角形分割を使います)。

Note

ポリゴン P の分割は、内部同士がインタセクトせず、結合すると元のポリゴン P の内部と同じになるようなポリゴンの集合です。CG_ApproxConvexPartition 関数と CG_GreeneApproxConvexPartition 関数は、概ね最適な凸分割を生成します。両方の関数は、最初にポリゴンをより単純なポリゴンに分解したうえで凸分解を行います。CG_ApproxConvexPartition は三角形を使い、CG_GreeneApproxConvexPartition は単調分割を使います。両方の関数は、凸分割の要素数について最適数の 4 倍を超えないことを保証しますが、実行時の複雑さが異なります。三角形ベースの近似アルゴリズムでは、しばしば、凸要素が少なくなります。常にそうなるとは限りません。

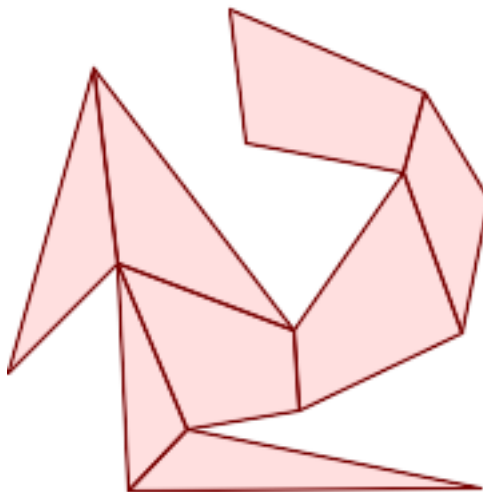
Availability: 3.5.0 - SFCGAL >= 1.5.0 が必要です。

SFCGAL >= 1.5.0 が必要



このメソッドには SFCGAL バックエンドが必要です。

例



近似凸分割 ([CG_YMonotonePartition](#), [CG_GreeneApproxConvexPartition](#) および [CG_OptimalConvexPartition](#) と同じ例です)

```
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)))
```

関連情報

[CG_YMonotonePartition](#), [CG_GreeneApproxConvexPartition](#), [CG_OptimalConvexPartition](#)

8.3.19 ST_ApproximateMedialAxis

ST_ApproximateMedialAxis — 面ジオメトリの近似的な中心軸を計算します。

Synopsis

```
geometry ST_ApproximateMedialAxis(geometry geom);
```

説明



Warning

ST_ApproximateMedialAxis は 3.5.0 では非推奨です。 **CG_ApproximateMedialAxis** を代わりに使います。

ストレートスケルトンを基に、実際の入力の近似的な中心軸を返します。可能な版 (1.2.0 以上) でビルドすると SFCGAL 独自 API を使います。そうでない場合には、CG_StraightSkeleton (遅い) 関数をラッピングした関数となります。

Availability: 2.2.0



このメソッドには SFCGAL バックエンドが必要です。



この関数は 3 次元に対応し、Z 値を削除しません。



この関数は多面体サーフェスに対応しています。



この関数は三角形と不規則三角網 (TIN) に対応しています。

8.3.20 CG_ApproximateMedialAxis

CG_ApproximateMedialAxis — 面ジオメトリの近似的な中心軸を計算します。

Synopsis

```
geometry CG_ApproximateMedialAxis(geometry geom);  
geometry CG_ApproximateMedialAxis(geometry geom, boolean projected);
```

説明

ストレートスケルトンを基に、実際の入力の近似的な中心軸を返します。可能な版 (1.2.0 以上) でビルドすると SFCGAL 独自 API を使います。そうでない場合には、CG_StraightSkeleton (遅い) 関数をラッピングした関数となります。

projected が **true** の場合には、中心軸の自由端点がポリゴン境界に届くよう延長されます (投影中心軸)。SFCGAL 2.3.0 以上が必要です。古いバージョンの SFCGAL でビルドされている場合には、通知が出され、非投影の結果が返されます。

Availability: 3.5.0

Availability: 3.7.0 - **projected** パラメーター。投影された結果を得るには SFCGAL $\geq$ 2.3.0 が必要。古いバージョンでは通知が出力され、非投影の結果になります。

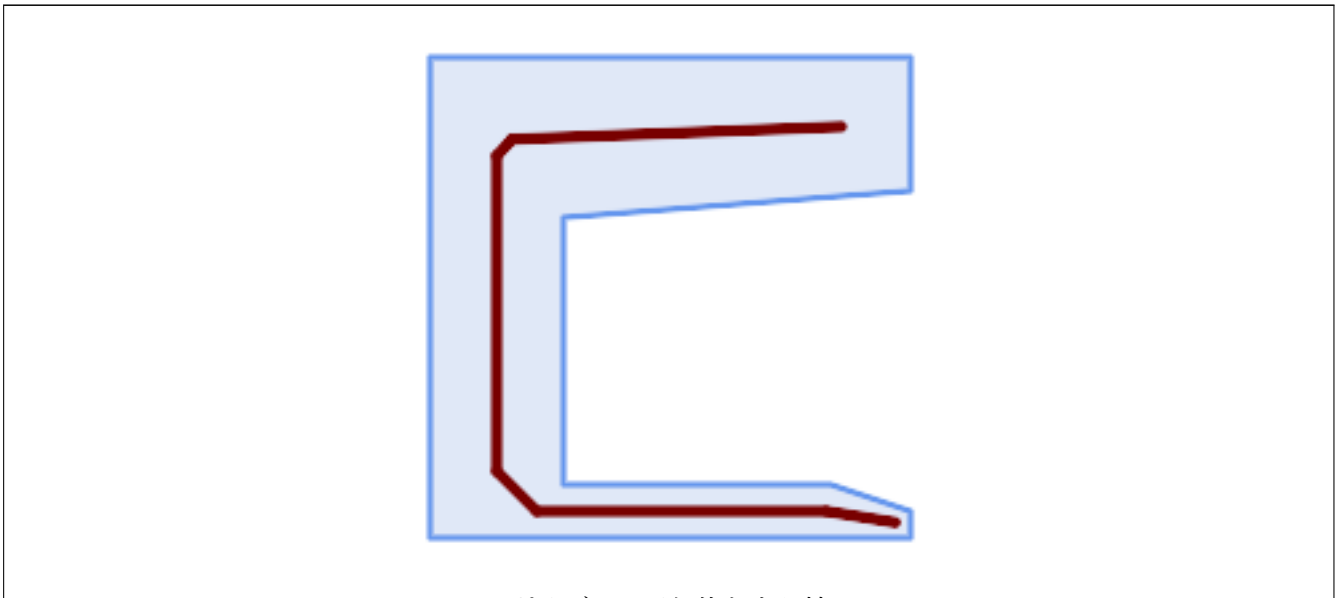
**Note**

この関数は Z 値を無視します。この関数を 3 次元ジオメトリ上で使用したとしても、常に 2 次元の結果となります。

- ✓ このメソッドには SFCGAL バックエンドが必要です。
- ✓ この関数は多面体サーフェスに対応しています。
- ✓ この関数は三角形と不規則三角網 (TIN) に対応しています。

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 ))'));
```



ポリゴンと近似的な中心軸

This example computes the projected medial axis with free endpoints extended to the polygon boundary.

```
SELECT ST_AsText(CG_ApproximateMedialAxis('POLYGON((0 0,2 0,2 1,0 1,0 0))', true));
```

```
MULTILINESTRING((0 0.5,0.5 0.5,1.5 0.5,2 0.5))
```

関連情報

[CG_StraightSkeleton](#), [CG_StraightSkeletonPartition](#)

8.3.21 CG_GenerateFlatRoof

CG_GenerateFlatRoof — 接地部ポリゴンから平らな (箱の) 屋根を生成します。

Synopsis

geometry **CG_GenerateFlatRoof**(geometry geom, float8 height);

説明

2次元接地部ポリゴンから平らな (箱状の) 屋根を 3次元 **PolyhedralSurface Z** で生成します。屋根の高さは地平面から **height** で指定した値です (デフォルト 3.0)。

Availability: 3.7.0 - SFCGAL >= 2.3.0 が必要。



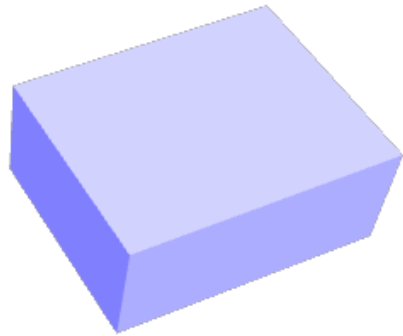
このメソッドには SFCGAL バックエンドが必要です。



この関数は多面体サーフェスに対応しています。

例

```
SELECT ST_AsText(  
  CG_GenerateFlatRoof(  
    'POLYGON((0 0,5 0,5 4,0 4,0 0))',  
    2.0));  
-- POLYHEDRALSURFACE Z (...)
```



関連情報

[CG_GenerateHippedRoof](#), [CG_GenerateGableRoof](#), [CG_GenerateSkillionRoof](#), [CG_GenerateRoof](#)

8.3.22 CG_GenerateHippedRoof

CG_GenerateHippedRoof — 接地部ポリゴンから寄棟屋根を生成する。

Synopsis

geometry **CG_GenerateHippedRoof**(geometry geom, float8 height);

説明

2次元接地部ポリゴンから寄棟屋根を 3次元 **PolyhedralSurface Z** で生成します。全側面の傾斜は頂部が合うように上がります。屋根の高さは地平面から **height** で指定した値です (デフォルト 3.0)。

Availability: 3.7.0 - SFCGAL >= 2.3.0 が必要。



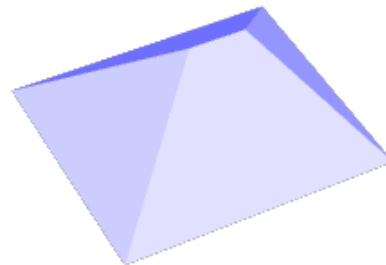
このメソッドには SFCGAL バックエンドが必要です。



この関数は多面体サーフェスに対応しています。

例

```
SELECT ST_AsText(  
  CG_GenerateHippedRoof(  
    'POLYGON((0 0,5 0,5 4,0 4,0 0))',  
    2.0));  
-- POLYHEDRALSURFACE Z (...)
```



関連情報

[CG_GenerateFlatRoof](#), [CG_GenerateGableRoof](#), [CG_GenerateSkillionRoof](#), [CG_GenerateRoof](#)

8.3.23 CG_GenerateGableRoof

CG_GenerateGableRoof — 接地部ポリゴンから切妻屋根を生成します。

Synopsis

geometry **CG_GenerateGableRoof**(geometry geom, float8 height, float8 slope_angle);

説明

2次元接地部ポリゴンから切妻屋根を3次元 **PolyhedralSurface Z** で生成します。一方の対辺は棟に対して垂直になっている終端です。もう一方の対辺は棟に平行に上がっていきます。屋根の高さは地平面から **height** で指定した値です (デフォルト 3.0)。**slope_angle** で仰角を度単位で指定します (デフォルト 30.0)。入力ジオメトリの **SRID** は保存されます。

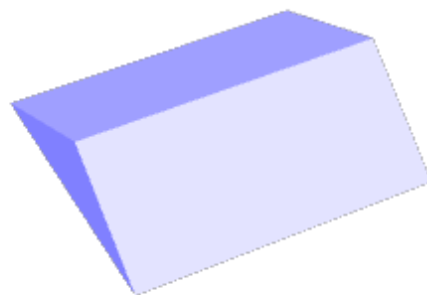
Availability: 3.7.0 - SFCGAL >= 2.3.0 が必要。

✔ このメソッドには SFCGAL バックエンドが必要です。

✔ この関数は多面体サーフェスに対応しています。

例

```
SELECT ST_CoordDim(  
  CG_GenerateGableRoof(  
    'POLYGON((0 0,5 0,5 4,0 4,0 0))',  
    2.0, 30.0));  
-- 3
```



関連情報

[CG_GenerateFlatRoof](#), [CG_GenerateHippedRoof](#), [CG_GenerateSkillionRoof](#), [CG_GenerateRoof](#)

8.3.24 CG_GenerateSkillionRoof

`CG_GenerateSkillionRoof` — Generate a skillion (single-slope) roof from a footprint polygon.

Synopsis

geometry **CG_GenerateSkillionRoof**(geometry geom, float8 height, float8 slope_angle, integer primary_edge_index);

説明

Generate a skillion (single-slope, shed-style) roof as a 3D `PolyhedralSurface Z` from a 2D footprint polygon. The entire roof surface slopes in one direction. `height` sets the maximum ridge height (default 3.0), `slope_angle` sets the pitch in degrees (default 30.0), and `primary_edge_index` selects which polygon edge defines the high side (0-based, default 0).

When the combination of `height` and `slope_angle` would cause a side wall to vanish (the slope reaches the far edge before the full height is achieved), that face is omitted from the result. This means the number of output patches can vary depending on the parameters —a lower height relative to the footprint depth produces more faces than a taller, steeper configuration.

Availability: 3.7.0 - SFCGAL >= 2.3.0 が必要。



このメソッドには SFCGAL バックエンドが必要です。



この関数は多面体サーフェスに対応しています。

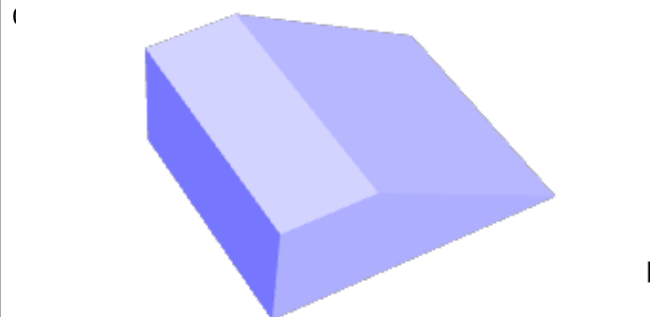
例

Height 2.0 produces 6 faces (all walls present); height 4.0 produces 5 faces (one side wall is filtered out because the slope reaches the far edge before the full height).

This example uses height 2.0 and returns 6 patches.

```
SELECT ST_NumPatches(
  'POLYGON((0 0,5 0,5 4,0 4,0 0))',
  2.0));
-- 6

-- height 4.0 -> 5 patches (one face ←
-- filtered)
SELECT ST_NumPatches(
  CG_GenerateSkillionRoof(
    'POLYGON((0 0,5 0,5 4,0 4,0 0))',
    4.0));
-- 5
```



関連情報

[CG_GenerateFlatRoof](#), [CG_GenerateHippedRoof](#), [CG_GenerateGableRoof](#), [CG_GenerateRoof](#)

8.3.25 CG_GenerateRoof

CG_GenerateRoof — Generate a roof of the requested type from a footprint polygon.

Synopsis

geometry **CG_GenerateRoof**(geometry geom, text roof_type, float8 height, float8 slope_angle, integer primary_edge_index);

説明

Generate a roof as a 3D PolyhedralSurface Z from a 2D footprint polygon. The roof_type parameter selects the roof style (default 'HIPPED'):

- FLAT —flat box roof.
- HIPPED —hipped roof (all sides slope to apex).
- GABLE —gable roof (two sloping sides, two vertical ends).
- SKILLION —single-slope (shed) roof.

height sets the roof height (default 3.0), slope_angle sets the pitch in degrees (default 30.0), and primary_edge_index selects the reference edge for skillion roofs (default 0).

Availability: 3.7.0 - SFCGAL >= 2.3.0 が必要。



このメソッドには SFCGAL バックエンドが必要です。



この関数は多面体サーフェスに対応しています。

例

```
SELECT ST_AsText(CG_GenerateRoof('POLYGON((0 0,4 0,4 4,0 4,0 0))', 'HIPPED', 2.0, 30.0, 0)) ←
;
-- POLYHEDRALSURFACE Z (((0 4 0,4 4 0,4 0 0,0 0 0,0 4 0)), ...)

SELECT ST_AsText(CG_GenerateRoof('POLYGON((0 0,4 0,4 4,0 4,0 0))', 'FLAT', 2.0));
-- POLYHEDRALSURFACE Z (((0 0 0,0 4 0,4 4 0,4 0 0,0 0 0)), ...)
```

関連情報

[CG_GenerateFlatRoof](#), [CG_GenerateHippedRoof](#), [CG_GenerateGableRoof](#), [CG_GenerateSkillionRoof](#)

8.3.26 CG_PolygonRepair

CG_PolygonRepair — Repair an invalid polygon or multipolygon.

Synopsis

geometry **CG_PolygonRepair**(geometry geom, text rule);

説明

Repair an invalid polygon or multipolygon using CGAL's 2D Polygon Repair algorithm. Returns a MultiPolygon.

The rule parameter controls the repair strategy:

- **EVEN_ODD** (default) —areas covered an odd number of times are kept. Available with CGAL 6.0+.
- **NON_ZERO** —areas with non-zero winding number are kept. Requires CGAL 6.1+.
- **UNION** —union of all input polygons. Requires CGAL 6.1+.
- **INTERSECTION** —intersection of all input polygons. Requires CGAL 6.1+.

Availability: 3.7.0 - requires SFCGAL >= 2.3.0 and CGAL >= 6.0.



このメソッドには SFCGAL バックエンドが必要です。



この関数は多面体サーフェスに対応しています。

例

This example repairs a bowtie self-intersecting polygon.

```
SELECT ST_AsText(CG_PolygonRepair('POLYGON((0 0,2 2,2 0,0 2,0 0))'));
-- MULTIPOLYGON(((0 0,1 1,0 2,0 0)),((1 1,2 0,2 2,1 1)))

-- Keep all winding area with UNION rule (requires CGAL 6.1 or later)
SELECT ST_AsText(CG_PolygonRepair('POLYGON((0 0,2 2,2 0,0 2,0 0))', 'UNION'));
-- MULTIPOLYGON(((0 0,1 1,0 2,0 0)))
```

関連情報

[ST_IsValid](#), [ST_MakeValid](#)

8.3.27 ST_ConstrainedDelaunayTriangles

ST_ConstrainedDelaunayTriangles — 入力ジオメトリの周りの制約付きドロネー三角形を返します。

Synopsis

```
geometry ST_ConstrainedDelaunayTriangles(geometry g1);
```

説明



Warning

[ST_ConstrainedDelaunayTriangles](#) は 3.5.0 では非推奨です。[CG_ConstrainedDelaunayTriangles](#) を代わりに使います。

入力ジオメトリの周りの制約付きドロネー三角形 ([Constrained Delaunay triangulation](#)) を返します。出力は TIN です。



このメソッドには SFCGAL バックエンドが必要です。

Availability: 3.0.0



この関数は 3 次元に対応し、Z 値を削除しません。

8.3.28 CG_ConstrainedDelaunayTriangles

CG_ConstrainedDelaunayTriangles — 入力ジオメトリの周りの制約付きドロネー三角形を返します。

Synopsis

```
geometry CG_ConstrainedDelaunayTriangles(geometry g1);
```

説明

入力ジオメトリの周りの制約付きドロネー三角形 ([Constrained Delaunay triangulation](#)) を返します。出力は TIN です。



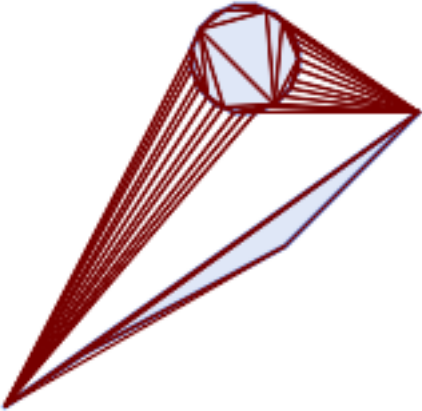
このメソッドには SFCGAL バックエンドが必要です。

Availability: 3.0.0



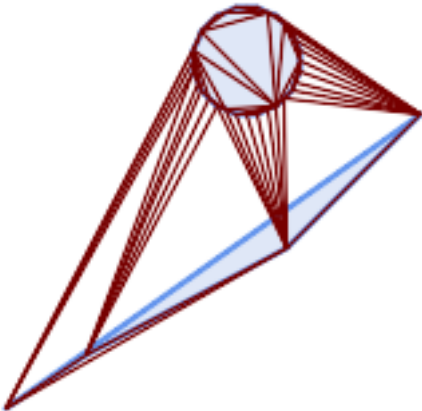
この関数は 3 次元に対応し、Z 値を削除しません。

例



二つのポリゴンの
CG_ConstrainedDelaunayTriangles

```
select CG_ConstrainedDelaunayTriangles(  
  ST_Union(  
    POLYGON((175 150, 20 40, 50 60, 125 100, 175 150)),  
    ST_Buffer('POINT(110 170)::geometry', 20)  
  )  
);
```



ST_DelaunayTriangles 二つのポリゴン。三角形の
辺はポリゴンの境界にクロスしています。

```
select ST_DelaunayTriangles(  
  ST_Union(  
    POLYGON((175 150, 20 40, 50 60, 125 100, 175 150)),  
    ST_Buffer('POINT(110 170)::geometry', 20)  
  )  
);
```

関連情報

ST_DelaunayTriangles, *ST_TriangulatePolygon*, *CG_Tessellate*, *ST_ConcaveHull*, *ST_Dump*

8.3.29 ST_Extrude

ST_Extrude — 関連するポリュームにサーフェスを押し出します。

Synopsis

geometry **ST_Extrude**(geometry geom, float x, float y, float z);

説明



Warning
ST_Extrude は 3.5.0 では非推奨です。 *CG_Extrude*を代わりに使います。

Availability: 2.1.0

- ✔ このメソッドには SFCGAL バックエンドが必要です。
- ✔ この関数は 3 次元に対応し、Z 値を削除しません。
- ✔ この関数は多面体サーフェスに対応しています。
- ✔ この関数は三角形と不規則三角網 (TIN) に対応しています。

8.3.30 CG_Extrude

CG_Extrude — 関連するボリュームにサーフェスを押し出します。

Synopsis

```
geometry CG_Extrude(geometry geom, float x, float y, float z);
```

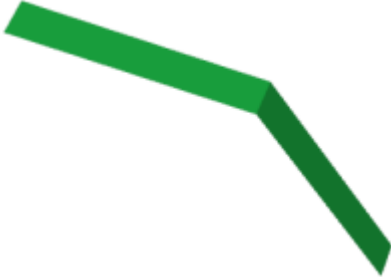
説明

Availability: 3.5.0

- ✔ このメソッドには SFCGAL バックエンドが必要です。
- ✔ この関数は 3 次元に対応し、Z 値を削除しません。
- ✔ この関数は多面体サーフェスに対応しています。
- ✔ この関数は三角形と不規則三角網 (TIN) に対応しています。

例

PostGIS 関数 **ST_AsX3D** を使って 3 次元イメージを生成し、**X3Dom HTML Javascript rednering library** を使って HTML での描画を行います。

<pre>SELECT ST_Buffer(quad_segs=2'),0,0,30);</pre> 50, ' ← 	<pre>CG_Extrude(ST_GeomFromText('POINT(100 90)'), 0, 0, 30) ←</pre> ↙  Z 方向に 30 単位押し出して得た <i>PolyhedralSurfaceZ</i>
ポイントのバッファから形成された元の八角形 <pre>SELECT ST_GeomFromText('LINESTRING(50 50, ← 100 90, 95 150)')</pre> 	<pre>SELECT CG_Extrude(ST_GeomFromText(' ← LINESTRING(50 50, 100 90, 95 150)'),0,0,10));</pre>  ラインストリングを Z 方向に押し出した <i>PolyhedralSurfaceZ</i>

元のラインストリング

関連情報

[ST_AsX3D](#), [CG_ExtrudeStraightSkeleton](#)

8.3.31 CG_ExtrudeStraightSkeleton

CG_ExtrudeStraightSkeleton — ストレートスケルトンの押し出し

Synopsis

```
geometry CG_ExtrudeStraightSkeleton(geometry geom, float roof_height, float body_height = 0);
```

説明

ポリゴンジオメトリの最大高での押出しを計算します。

Note



ストレートスケルトンの (歴史上の) 最初の使用例は、おそらく、ポリゴンの屋根を与えて、ストレートスケルトンでテントの配置を得るというものだったでしょう。スケルトンの個々のエッジは平面からオフセット距離分引き上げられ、水が、屋根のどこに落ちるか関係なく、常に輪郭のエッジ (屋根の境界線) に落ちるという点で、結果の屋根は「正しい」ものになります。この関数はこのポリゴンの押出し、別名「屋根」を計算します。引数の `body_height` が `body_height > 0` となる場合には、ポリゴンは `CG_Extrude(polygon, 0, 0, body_height)` で押し出されるようになります。結果はこれらの多面体サーフェスの結合となります。

Availability: 3.5.0 - SFCGAL $\geq$ 1.5.0 が必要です。

SFCGAL $\geq$ 1.5.0 が必要



このメソッドには SFCGAL バックエンドが必要です。

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)) ←  
,((0 4 2,0 0 2,0.5 0.5 2.5,0.5 2.5 2.5,1 3 3,0 4 2)),((0 0 2,5 0 2,3.5 1.5 3.5,2.5 0.5 ←  
2.5,0.5 0.5 2.5,0 0 2)),((5 0 2,5 5 2,4.5 4.5 2.5,4.5 3.5 2.5,3.5 2.5 3.5,3.5 1.5 3.5,5 ←  
0 2)),((5 5 2,4 5 2,4.5 4.5 2.5,5 5 2)),((4 5 2,4 4 2,4.5 3.5 2.5,4.5 4.5 2.5,4 5 2)) ←  
,((4 4 2,0 4 2,1 3 3,3 3,3.5 2.5 3.5,4.5 3.5 2.5,4 4 2)),((2 1 2,1 1 2,0.5 0.5 2.5,2.5 ←  
0.5 2.5,2 1 2)),((1 1 2,1 2 2,0.5 2.5 2.5,0.5 0.5 2.5,1 1 2)),((1 2 2,2 2 2,3 3 3,1 3 ←  
3,0.5 2.5 2.5,1 2 2)),((2 2 2,2 1 2,2.5 0.5 2.5,3.5 1.5 3.5,3.5 2.5 3.5,3 3 3,2 2 2)))
```

関連情報

[ST_Extrude](#), [CG_ExtrudeStraightSkeleton](#), [CG_StraightSkeleton](#), [CG_StraightSkeletonPartition](#)

8.3.32 CG_GreeneApproxConvexPartition

`CG_GreeneApproxConvexPartition` — ポリゴンジオメトリの近似凸分割を計算します

Synopsis

```
geometry CG_GreeneApproxConvexPartition(geometry geom);
```

説明

ポリゴンジオメトリの近似的な単調凸分割を計算します。

Note

ポリゴン P の分割は、内部同士がインタセクトせず、結合すると元のポリゴン P の内部と同じになるようなポリゴンの集合です。CG_ApproxConvexPartition 関数と CG_GreeneApproxConvexPartition 関数は、概ね最適な凸分割を生成します。両方の関数は、最初にポリゴンをより単純なポリゴンに分解したうえで凸分解を行います。CG_ApproxConvexPartition は三角形を使い、CG_GreeneApproxConvexPartition は単調分割を使います。両方の関数は、凸分割の要素数について最適数の 4 倍を超えないことを保証しますが、実行時の複雑さが異なります。三角形ベースの近似アルゴリズムでは、しばしば、凸要素が少なくなります、常にそうなるとは限りません。

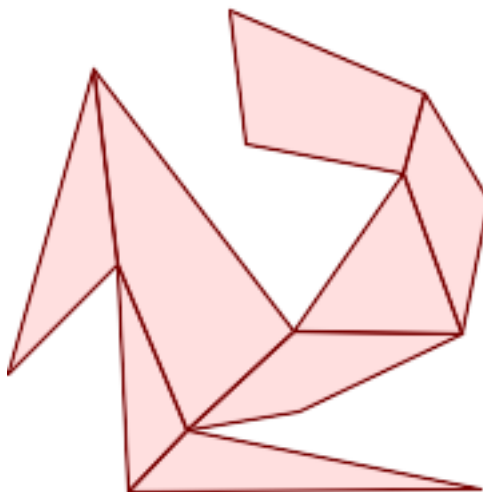
Availability: 3.5.0 - SFCGAL >= 1.5.0 が必要です。

SFCGAL >= 1.5.0 が必要



このメソッドには SFCGAL バックエンドが必要です。

例



グリーン近似凸分割 (*CG_YMonotonePartition*, *CG_ApproxConvexPartition*, *CG_OptimalConvexPartition* と同じ例です)

```
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)))
```

関連情報

CG_YMonotonePartition, *CG_ApproxConvexPartition*, *CG_OptimalConvexPartition*

8.3.33 ST_MinkowskiSum

ST_MinkowskiSum — ミンコフスキー和を求めます。

Synopsis

```
geometry ST_MinkowskiSum(geometry geom1, geometry geom2);
```

説明



Warning

ST_MinkowskiSum は 3.5.0 では非推奨です。CG_MinkowskiSumを代わりに使います。

ポリゴンと、ポイント、ライン、ポリゴンのいずれかとの 2 次元のミンコフスキー和を計算します。

二つのジオメトリ A と B のミンコフスキー和は A と B のあらゆるポイントの和の集合です。ミンコフスキー和は、しばしば動作計画と CAD で使われます。より詳細な情報について[Wikipedia Minkowski addition](#)をご覧ください。

一つ目の引数は 2 次元ジオメトリ (ポイント、ラインストリング、ポリゴン) とすることができます。3 次元ジオメトリを渡すと、Z を 0 とした 2 次元に強制され、この場合は無効と考えられます。二つ目の引数は 2 次元ポリゴンでなければなりません。

CGAL 2D Minkowskisumを利用して実装しています。

Availability: 2.1.0



このメソッドには SFCGAL バックエンドが必要です。

8.3.34 CG_MinkowskiSum

CG_MinkowskiSum — ミンコフスキー和を求めます。

Synopsis

```
geometry CG_MinkowskiSum(geometry geom1, geometry geom2);
```

説明

ポリゴンと、ポイント、ライン、ポリゴンのいずれかとの 2 次元のミンコフスキー和を計算します。

二つのジオメトリ A と B のミンコフスキー和は A と B のあらゆるポイントの和の集合です。ミンコフスキー和は、しばしば動作計画と CAD で使われます。より詳細な情報について[Wikipedia Minkowski addition](#)をご覧ください。

一つ目の引数は 2 次元ジオメトリ (ポイント、ラインストリング、ポリゴン) とすることができます。3 次元ジオメトリを渡すと、Z を 0 とした 2 次元に強制され、この場合は無効と考えられます。二つ目の引数は 2 次元ポリゴンでなければなりません。

CGAL 2D Minkowskisumを利用して実装しています。

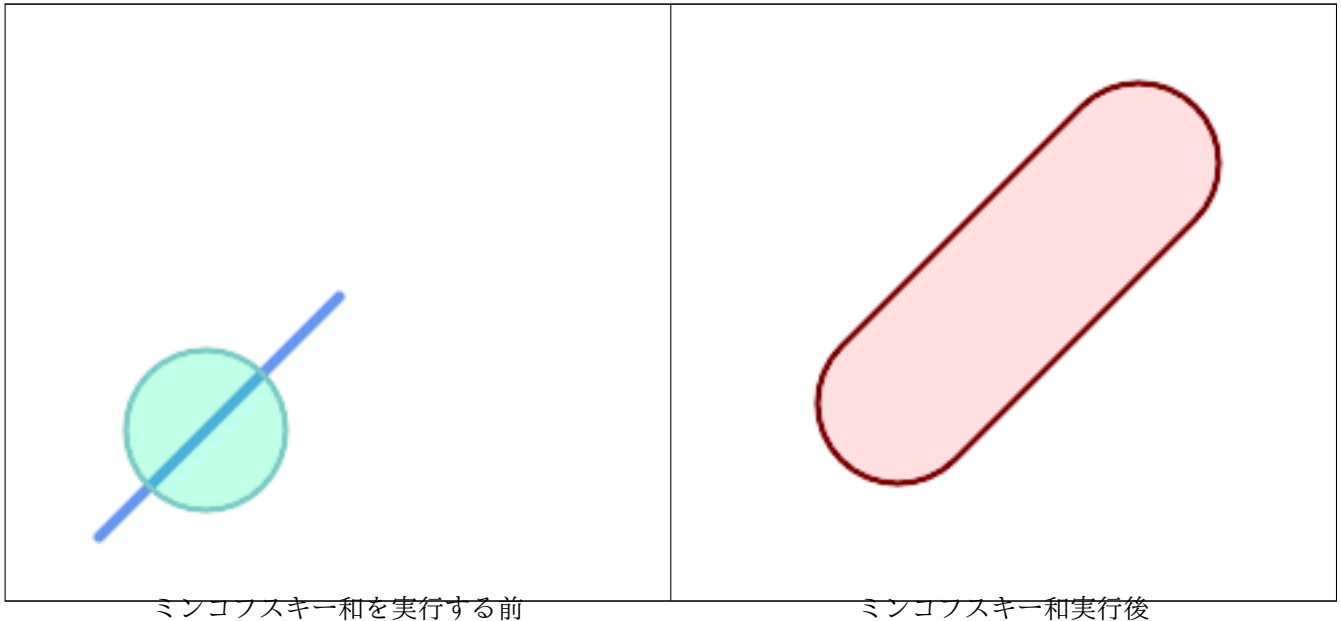
Availability: 3.5.0



このメソッドには SFCGAL バックエンドが必要です。

例

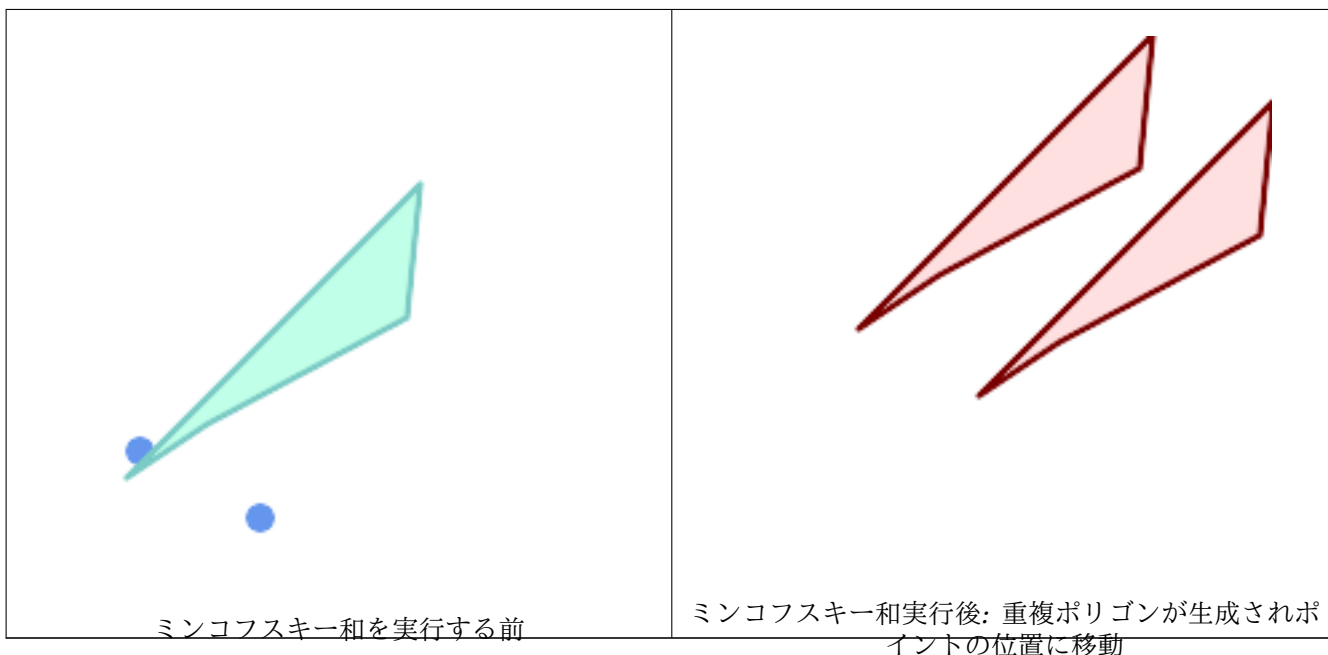
ラインストリングと円ポリゴンであって、ラインストリングがポリゴンを横切るミンコフスキー和



```
SELECT CG_MinkowskiSum(line, circle))
FROM (SELECT
  ST_MakeLine(ST_Point(10, 10),ST_Point(100, 100)) As line,
  ST_Buffer(ST_GeomFromText('POINT(50 50)'), 30) As circle) As foo;

-- wkt --
MULTIPOLYGON(((30 59.9999999999999,30.5764415879031 ↵
  54.1472903395161,32.2836140246614 48.5194970290472,35.0559116309237 ↵
  43.3328930094119,38.7867965644036 38.7867965644035,43.332893009412 ↵
  35.0559116309236,48.5194970290474 32.2836140246614,54.1472903395162 ↵
  30.5764415879031,60.0000000000000 30,65.8527096604839 ↵
  30.5764415879031,71.4805029709527 32.2836140246614,76.6671069905881 ↵
  35.0559116309237,81.2132034355964 38.7867965644036,171.213203435596 ↵
  128.786796564404,174.944088369076 133.332893009412,177.716385975339 ↵
  138.519497029047,179.423558412097 144.147290339516,180 150,179.423558412097 ↵
  155.852709660484,177.716385975339 161.480502970953,174.944088369076 ↵
  166.667106990588,171.213203435596 171.213203435596,166.667106990588 ↵
  174.944088369076,
  161.480502970953 177.716385975339,155.852709660484 179.423558412097,150 ↵
  180,144.147290339516 179.423558412097,138.519497029047 ↵
  177.716385975339,133.332893009412 174.944088369076,128.786796564403 ↵
  171.213203435596,38.7867965644035 81.2132034355963,35.0559116309236 ↵
  76.667106990588,32.2836140246614 71.4805029709526,30.5764415879031 ↵
  65.8527096604838,30 59.9999999999999)))
```

ポリゴンとマルチポイントとのミンコフスキー和



```
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))
)
```

8.3.35 ST_OptimalAlphaShape

ST_OptimalAlphaShape — 「最適」 アルファ値を使ってジオメトリを囲むアルファシェイプを計算します。

Synopsis

geometry **ST_OptimalAlphaShape**(geometry geom, boolean allow_holes = false, integer nb_components = 1);

説明



Warning

ST_OptimalAlphaShape は 3.5.0 では非推奨です。 **CG_OptimalAlphaShape** を代わりに使います。

ジオメトリのポイントの「最適」のアルファシェイプを計算します。アルファシェイプは次に示す項目のようになるよう選択された α 値を使います:

1. ポリゴン要素数は `nb_components` 以下になります (デフォルトは 1 です)
2. 全ての入力ポイントはシェイプの中に含まれます

任意引数 `allow_holes` に `TRUE` が設定されていない限りは、結果には穴が含まれません。

Availability: 3.3.0 - SFCGAL $\geq$ 1.4.1 が必要です。



このメソッドには SFCGAL バックエンドが必要です。

8.3.36 CG_OptimalAlphaShape

`CG_OptimalAlphaShape` — 「最適」アルファ値を使ってジオメトリを囲むアルファシェイプを計算します。

Synopsis

```
geometry CG_OptimalAlphaShape(geometry geom, boolean allow_holes = false, integer nb_components = 1);
```

説明

ジオメトリのポイントの「最適」のアルファシェイプを計算します。アルファシェイプは次に示す項目のようになるよう選択された α 値を使います:

1. ポリゴン要素数は `nb_components` 以下になります (デフォルトは 1 です)
2. 全ての入力ポイントはシェイプの中に含まれます

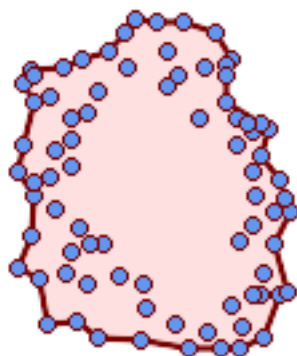
任意引数 `allow_holes` に `TRUE` が設定されていない限りは、結果には穴が含まれません。

Availability: 3.5.0 - SFCGAL $\geq$ 1.4.1 が必要です。



このメソッドには SFCGAL バックエンドが必要です。

例



MULTIPOINT の最適アルファシェイプ (*CG_AlphaShape*と同じ)

```

SELECT ST_AsText(
  CG_OptimalAlphaShape(
    'MULTIPOINT((63 84),(76 88),(68 73),(53 18),(91 50),(81 70),
      (88 29),(24 82),(32 51),(37 23),(27 54),(84 19),(75 87),(44 42),(77 67),(90 30) ←
      ,(36 61),(32 65),
      (81 47),(88 58),(68 73),(49 95),(81 60),(87 50),
      (78 16),(79 21),(30 22),(78 43),(26 85),(48 34),(35 35),(36 40),(31 79),(83 29) ←
      ,(27 84),(52 98),(72 95),(85 71),
      (75 84),(75 77),(81 29),(77 73),(41 42),(83 72),(23 36),(89 53),(27 57),(57 97) ←
      ,(27 77),(39 88),(60 81),
      (80 72),(54 32),(55 26),(62 22),(70 20),(76 27),(84 35),(87 42),(82 54),(83 64) ←
      ,(69 86),(60 90),(50 86),(43 80),(36 73),
      (36 68),(40 75),(24 67),(23 60),(26 44),(28 33),(40 32),(43 19),(65 16),(73 16) ←
      ,(38 46),(31 59),(34 86),(45 90),(64 97))':geometry));

MULTIPOLYGON(((89 53,88 58,83 64,85 71,83 72,75 77,75 84,76 88,72 95,64 97,57 97,52 98,49 ←
  95,45 90,39 88,34 86,26 85,24 82,27 77,24 67,23 60,27 54,26 44,23 36,28 33,30 22,37 ←
  23,43 19,53 18,65 16,73 16,78 16,84 19,88 29,90 30,87 42,91 50,89 53)))

```



穴を許した *MULTIPOINT* の最適アルファシェイプ (*CG_AlphaShape* と同じ)

```

SELECT ST_AsText(
  CG_OptimalAlphaShape(
    'MULTIPOINT((63 84),(76 88),(68 73),(53 18),(91 50),(81 70),(88 29),(24 82),(32 51),(37 ←
      23),(27 54),(84 19),(75 87),(44 42),(77 67),(90 30),(36 61),(32 65),(81 47),(88 58) ←
      ,(68 73),(49 95),(81 60),(87 50),
      (78 16),(79 21),(30 22),(78 43),(26 85),(48 34),(35 35),(36 40),(31 79),(83 29),(27 84) ←
      ,(52 98),(72 95),(85 71),
      (75 84),(75 77),(81 29),(77 73),(41 42),(83 72),(23 36),(89 53),(27 57),(57 97),(27 77) ←
      ,(39 88),(60 81),
      (80 72),(54 32),(55 26),(62 22),(70 20),(76 27),(84 35),(87 42),(82 54),(83 64),(69 86) ←
      ,(60 90),(50 86),(43 80),(36 73),
      (36 68),(40 75),(24 67),(23 60),(26 44),(28 33),(40 32),(43 19),(65 16),(73 16),(38 46) ←
      ,(31 59),(34 86),(45 90),(64 97))':geometry,
    allow_holes => true));

MULTIPOLYGON(((89 53,88 58,83 64,85 71,83 72,75 77,75 84,76 88,72 95,64 97,57 97,52 98,49 ←
  95,45 90,39 88,34 86,26 85,24 82,27 77,24 67,23 60,27 54,26 44,23 36,28 33,30 22,37 ←
  23,43 19,53 18,65 16,73 16,78 16,84 19,88 29,90 30,87 42,91 50,89 53),(36 68,40 75,43 ←
  80,50 86,60 81,68 73,77 67,81 60,82 54,81 47,78 43,81 29,76 27,70 20,62 22,55 26,54 ←
  32,48 34,44 42,38 46,36 61,36 68)))

```

関連情報

[ST_ConcaveHull](#), [CG_AlphaShape](#)

8.3.37 CG_OptimalConvexPartition

CG_OptimalConvexPartition — ポリゴンジオメトリの最適凸分割を計算します

Synopsis

```
geometry CG_OptimalConvexPartition(geometry geom);
```

説明

ポリゴンジオメトリの最適凸分割を計算します。



Note

ポリゴン P の分割は、内部同士がインタセクトせず、結合すると元のポリゴン P の内部と同じになるようなポリゴンの集合です。CG_OptimalConvexPartition は、分割要素数について最適な分割を生成します。

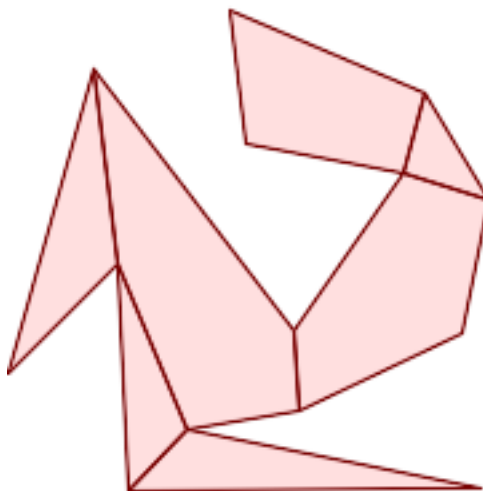
Availability: 3.5.0 - SFCGAL $\geq$ 1.5.0 が必要です。

SFCGAL $\geq$ 1.5.0 が必要



このメソッドには SFCGAL バックエンドが必要です。

例



最適凸分割 ([CG_YMonotonePartition](#), [CG_ApproxConvexPartition](#), [CG_GreeneApproxConvexPartition](#)と同じ例)

```
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)))
```

関連情報

[CG_YMonotonePartition](#), [CG_ApproxConvexPartition](#), [CG_GreeneApproxConvexPartition](#)

8.3.38 CG_StraightSkeleton

CG_StraightSkeleton — ジオメトリからストレートスケルトンを計算します。

Synopsis

geometry **CG_StraightSkeleton**(geometry geom, boolean use_distance_as_m = false);

説明

Availability: 3.5.0

任意引数 use_distance_as_m を使用するには SFCGAL >= 1.3.8 が必要です

- ✔ このメソッドには SFCGAL バックエンドが必要です。
- ✔ この関数は 3 次元に対応し、Z 値を削除しません。
- ✔ この関数は多面体サーフェスに対応しています。
- ✔ この関数は三角形と不規則三角網 (TIN) に対応しています。

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 ))'));
```

```
SELECT ST_AsText(CG_StraightSkeleton('POLYGON((0 0,1 0,1 1,0 1,0 0))', true));
```

```
MULTILINESTRING M ((0 0 0,0.5 0.5 0.5),(1 0 0,0.5 0.5 0.5),(1 1 0,0.5 0.5 0.5),(0 1 0,0.5 ←
0.5 0.5))
```

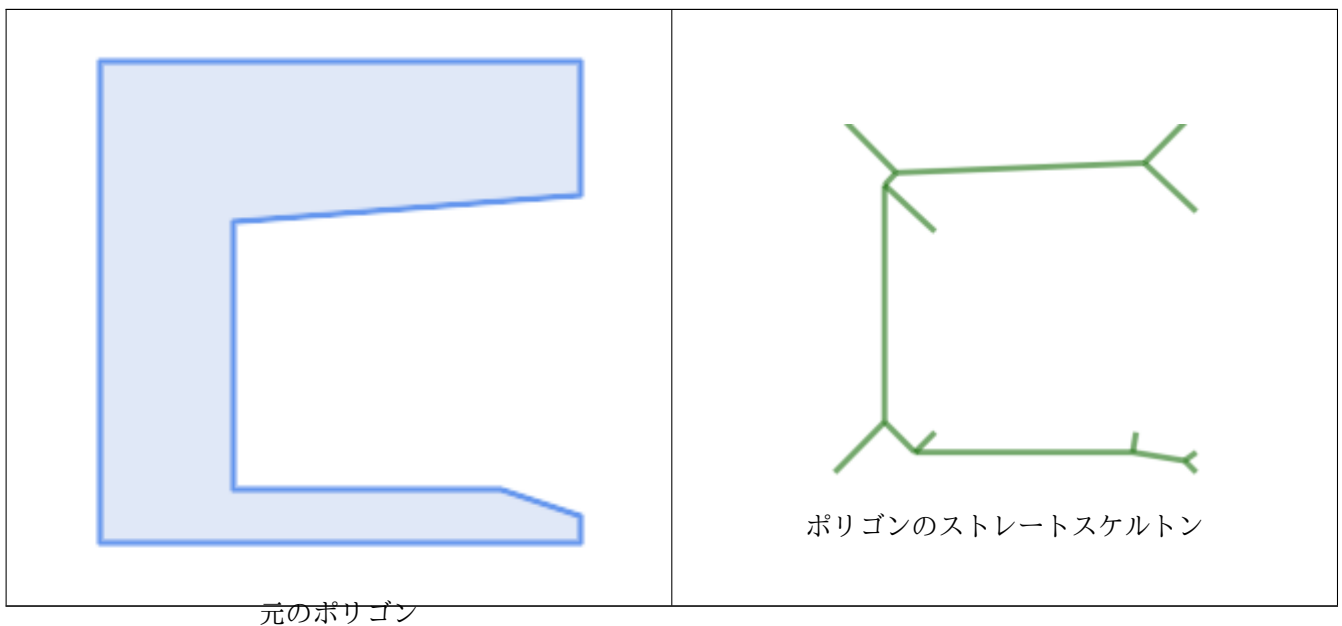
リング同士が 1 点で接触する妥当な入力ではエラーが発生することに注意してください。

```
SELECT CG_StraightSkeleton('POLYGON((0 0, 3 0, 3 3, 0 3, 0 0), (0 0, 1 2, 2 1, 0 0)))');
```

NOTICE: During straight_skeleton(A) :

```
NOTICE: with A: POLYGON((0/1 0/1,3/1 0/1,3/1 3/1,0/1 3/1,0/1 0/1),(0/1 0/1,1/1 2/1,2/1 ←
1/1,0/1 0/1))
```

ERROR: straight skeleton of Polygon with point touching rings is not implemented.



関連情報

[CG_StraightSkeletonPartition](#), [CG_ExtrudeStraightSkeleton](#)

8.3.39 ST_StraightSkeleton

ST_StraightSkeleton — ジオメトリからストレートスケルトンを計算します。

Synopsis

geometry **ST_StraightSkeleton**(geometry geom);

説明



Warning

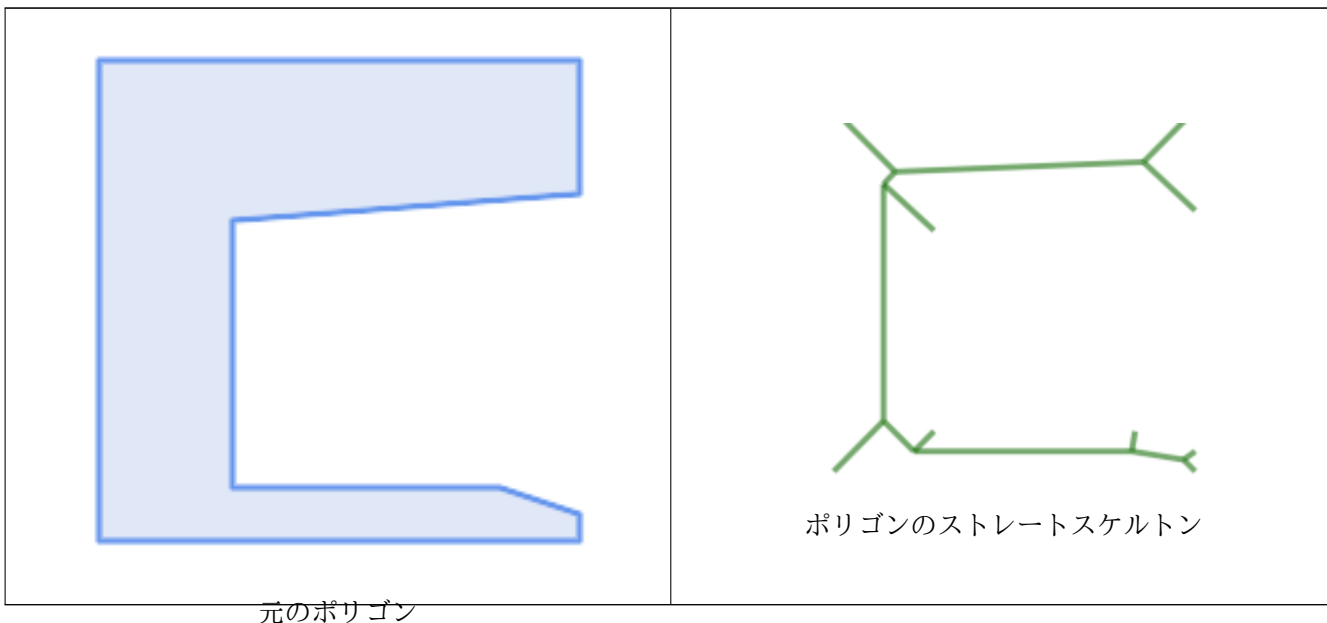
ST_StraightSkeleton は 3.5.0 では非推奨です。 **CG_StraightSkeleton** を代わりに使います。

Availability: 2.1.0

- ✔ このメソッドには SFCGAL バックエンドが必要です。
- ✔ この関数は 3 次元に対応し、Z 値を削除しません。
- ✔ この関数は多面体サーフェスに対応しています。
- ✔ この関数は三角形と不規則三角網 (TIN) に対応しています。

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 ))'));
```



関連情報

[CG_ExtrudeStraightSkeleton](#)

8.3.40 ST_Tessellate

ST_Tessellate — ポリゴンまたは多面体サーフェスのテッセレーションを計算し、TIN または TIN コレクションを返します。

Synopsis

geometry **ST_Tessellate**(geometry geom);

説明



Warning

ST_Tessellate は 3.5.0 では非推奨です。 [CG_Tessellate](#) を代わりに使います。

(MULTI)POLYGON または POLYHEDRALSURFACE のような面を入力に取り、三角形を使ったテッセレーション処理を通して TIN 表現を返します。



Note

ST_TriangulatePolygon はこの関数に似ていますが、TIN の代わりにジオメトリコレクションを返す点と 2 次元ジオメトリだけで機能する点とは除きます。

Availability: 2.1.0

- ✔ このメソッドには SFCGAL バックエンドが必要です。
- ✔ この関数は 3 次元に対応し、Z 値を削除しません。
- ✔ この関数は多面体サーフェスに対応しています。
- ✔ この関数は三角形と不規則三角網 (TIN) に対応しています。

8.3.41 CG_Tessellate

CG_Tessellate — ポリゴンまたは多面体サーフェスのテッセレーションを計算し、TIN または TIN コレクションを返します。

Synopsis

```
geometry CG_Tessellate(geometry geom);
```

説明

(MULTI)POLYGON または POLYHEDRALSURFACE のような面を入力に取り、三角形を使ったテッセレーション処理を通して TIN 表現を返します。



Note

ST_TriangulatePolygon はこの関数に似ていますが、TIN の代わりにジオメトリコレクションを返す点と 2 次元ジオメトリだけで機能する点とは除きます。

Availability: 3.5.0

- ✔ このメソッドには SFCGAL バックエンドが必要です。
- ✔ この関数は 3 次元に対応し、Z 値を削除しません。
- ✔ この関数は多面体サーフェスに対応しています。
- ✔ この関数は三角形と不規則三角網 (TIN) に対応しています。

例

```
SELECT ST_GeomFromText('POLYHEDRALSURFACE ↵
  Z( ((0 0 0, 0 0 1, 0 1 1, 0 1 0, 0 0 0)), ↵
      ((0 0 ↵
0, 0 1 0, 1 1 0, 1 0 0, 0 0 0)), ((0 0 0, ↵
      ((1 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
```



元の立方体

```
SELECT CG_Tessellate(ST_GeomFromText(' ↵
  POLYHEDRALSURFACE Z( ((0 0 0, 0 0 1, 0 1 1, 0 1 ↵
      ((0 0 0, ↵
0 1 0, 1 1 0, 1 0 0, 0 0 0)), ((0 0 0, 1 0 0, 1 ↵
      ((1 1 0, ↵
1 1 1, 1 0 1, 1 0 0, 1 1 0)), ↵
      ((0 1 0, ↵
0 1 1, 1 1 1, 1 1 0, 0 1 0)), ((0 0 1, 1 0 1, 1 ↵
  TIN Z ((0 0 0,0 0 1,0 1 1,0 0 0)),((0 1 ↵
0,0 0 0,0 1 1,0 1 0)), ↵
1 0 0, 1 0 1, 0 0 1, 0 0 0)),((0 0 0,0 1 0,1 1 ↵
0,0 0 0)), ↵
      ((1 0 0,0 0 0,1 1 ↵
0,1 0 0)),((0 0 1,1 0 0,1 0 1,0 0 1)), ↵
1, 1 1 1, 0 1 1, 0 0 0)),((0 0 0,1 0 1,0 0 0,1 0 ↵
0,0 0 1)), ↵
      ((1 1 0,1 1 1,1 0 ↵
1,1 1 0)),((1 0 0,1 1 0,1 0 1,1 0 0)), ↵
      ((0 1 0,0 1 1,1 1 ↵
1,0 1 0)),((1 1 0,0 1 0,1 1 1,1 1 0)), ↵
      ((0 1 1,1 0 1,1 1 ↵
1,0 1 1)),((0 1 1,0 0 1,1 0 1,0 1 1)))
```



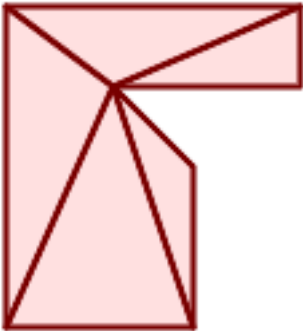
彩色した三角形によるテッセレーションを施した立方体

```
SELECT 'POLYGON (( 10 190, 10 70, 80 70, 80 130, 50 160, 120 160, 120 190, 10 190 ))'::geometry
```



元のポリゴン

```
SELECT CG_Tessellate('POLYGON (( 10 190, 10 70, 80 70, 80 130, 50 160, 120 160, 120 190, 10 190 ))'::geometry)
```



デッセレーションを施したポリゴン

関連情報

[CG_ConstrainedDelaunayTriangles](#), [ST_DelaunayTriangles](#), [ST_TriangulatePolygon](#)

8.3.42 CG_Triangulate

CG_Triangulate — ポリゴンジオメトリを三角形にします

Synopsis

geometry **CG_Triangulate**(geometry geom);

説明

ポリゴンジオメトリを三角形にします。
SFCGAL モジュールによって実行されます



Note
ご注意: この関数は三角形化した結果を表現するジオメトリを返します。

Availability: 3.5.0

 このメソッドには SFCGAL バックエンドが必要です。

ジオメトリの例

```
SELECT ST_AsText(CG_Triangulate('POLYGON((0.0 0.0,1.0 0.0,1.0 1.0,0.0 1.0,0.0 0.0),(0.2 0.2,0.2 0.8,0.8 0.8 0.2,0.2 0.2))'));
```

```
st_astext
-----
TIN(((0.8 0.2,0.2 0.2,1 0,0.8 0.2)),((0.2 0.2,0 0,1 0,0.2 0.2)),((1 1,0.8 0.8,0.8 0.2,1 1)),((0 1,0 0,0.2 0.2,0 1)),((0 1,0.2 0.8,1 1,0 1)),((0 1,0.2 0.2,0.2 0.8,0 1)),((0.2 0.8,0.8 0.8,1 1,0.2 0.8)),((0.2 0.8,0.2 0.2,0.8 0.2,0.8 0.2,0.2 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)
```

関連情報

[CG_ConstrainedDelaunayTriangles](#), [ST_DelaunayTriangles](#), [ST_TriangulatePolygon](#)

8.3.43 CG_Visibility

CG_Visibility — ポリゴンジオメトリ内のポイント又は辺から可視領域ポリゴンを計算する

Synopsis

```
geometry CG_Visibility(geometry polygon, geometry point);
geometry CG_Visibility(geometry polygon, geometry pointA, geometry pointB);
```

説明

Availability: 3.5.0 - SFCGAL >= 1.5.0 が必要です。

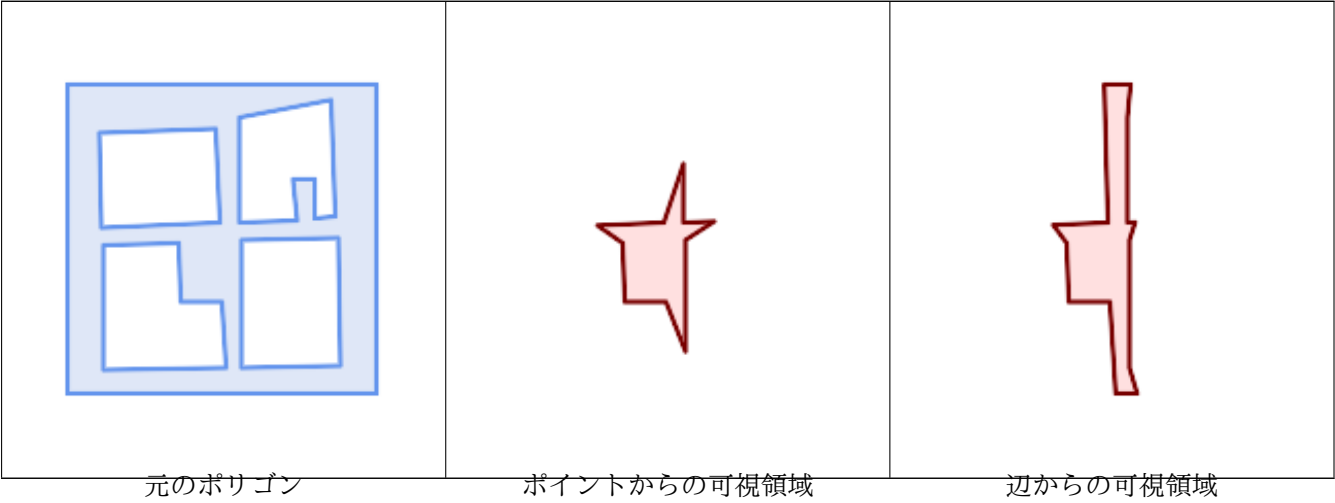
SFCGAL >= 1.5.0 が必要

-  このメソッドには SFCGAL バックエンドが必要です。
-  この関数は 3 次元に対応し、Z 値を削除しません。
-  この関数は多面体サーフェスに対応しています。
-  この関数は三角形と不規則三角網 (TIN) に対応しています。

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, 'POINT(98.5 68)'::geometry);
```



8.3.44 CG_YMonotonePartition

CG_YMonotonePartition — ポリゴンジオメトリの Y 単調分割を計算します

Synopsis

```
geometry CG_YMonotonePartition(geometry geom);
```

説明

ポリゴンジオメトリの Y 単調分割を計算します。



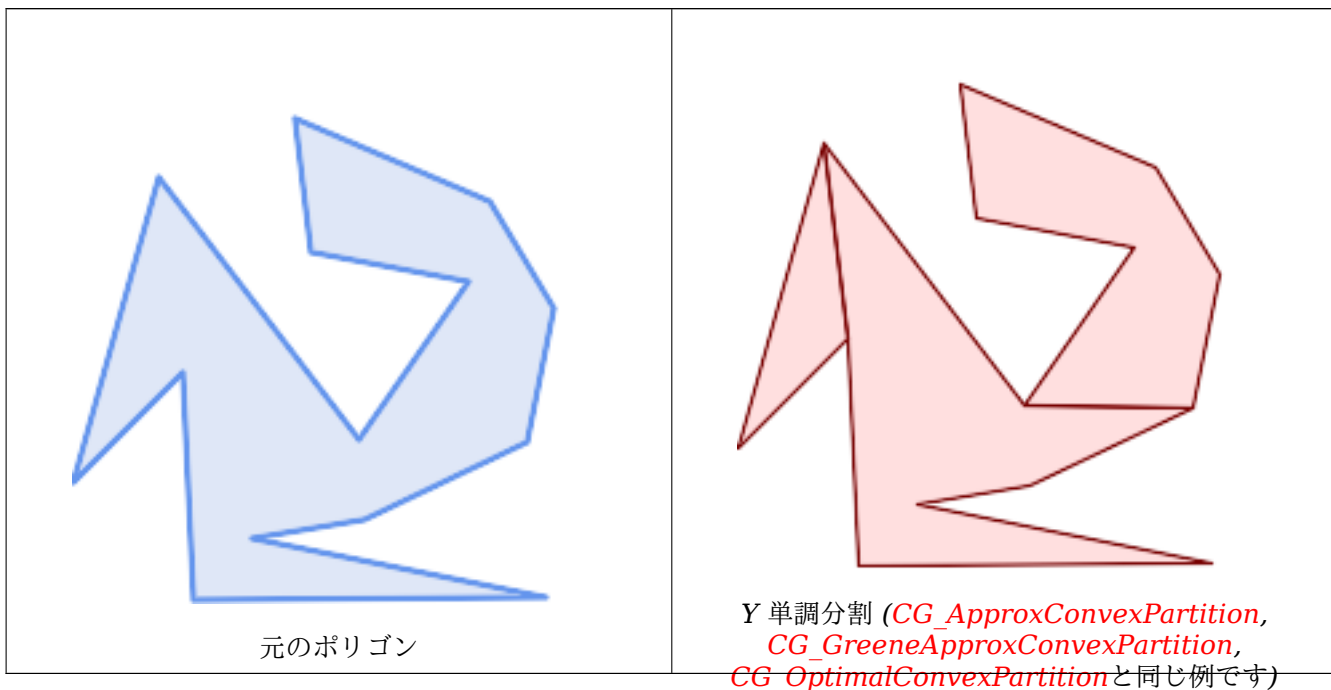
Note
ポリゴン P の分割は、内部同士がインタセクトせず、結合すると元のポリゴン P の内部と同じになるようなポリゴンの集合です。Y 単調のポリゴンとは、頂点 $v_1, \dots, v_n$ が、 $v_1, \dots, v_k$ と $v_k, \dots, v_n, v_1$ の二つのチェーンに、水平線がどちらかのチェーンに、ただだか 1 回インタセクトするようになるように分割することができるポリゴンです。このアルゴリズムは、最適なポリゴン数に対して生成されるポリゴン数の限度を保証しません。

Availability: 3.5.0 - SFCGAL >= 1.5.0 が必要です。

SFCGAL >= 1.5.0 が必要

 このメソッドには SFCGAL バックエンドが必要です。

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)))
```

関連情報

CG_ApproxConvexPartition, *CG_GreeneApproxConvexPartition*, *CG_OptimalConvexPartition*

8.3.45 CG_StraightSkeletonPartition

CG_StraightSkeletonPartition — ポリゴンのストレートスケルトン分割を計算します。

Synopsis

geometry **CG_StraightSkeletonPartition**(geometry geom, boolean auto_orientation);

説明

ポリゴンジオメトリ *geom* のストレートスケルトン分割を計算します。ストレートスケルトンはポリゴンをエッジの間引きで形成されるフェースに分割します。*auto_orientation* を TRUE に指定した場合には、この関数は正しい結果が得られるように入力ポリゴンの向きを自動的に調整します。

Availability: 3.6.0 - SFCGAL >= 2.0.0 が必要です。



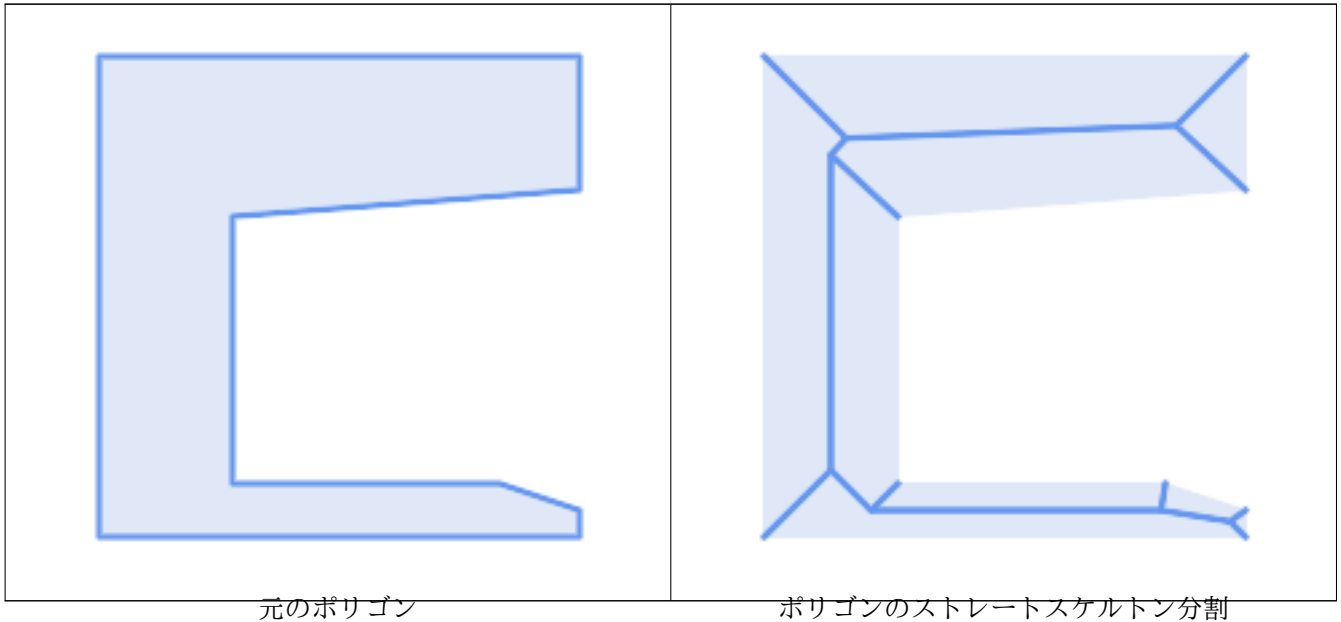
このメソッドには SFCGAL バックエンドが必要です。

例

```
SELECT ST_AsText(CG_StraightSkeletonPartition('POLYGON((0 0, 4 0, 2 2, 0 0))', true));
```

```
POLYHEDRALSURFACE(((0 0,2 0.82842712474619,2 2)),((4 0,2 0.82842712474619,0 0)),((2 2,2 0.82842712474619,4 0)))
```

```
SELECT CG_StraightSkeletonPartition(
  ST_GeomFromText('POLYGON (( 190 190, 10 190, 10 10, 190 10, 190 20,
    160 30, 60 30, 60 130, 190 140, 190 190 ))'),
  true );
```



元のポリゴン

ポリゴンのストレートスケルトン分割

関連情報

[CG_StraightSkeleton](#), [ST_Polygonize](#)

8.3.46 CG_3DBuffer

CG_3DBuffer — ジオメトリの周りの 3 次元バッファを計算します。

Synopsis

```
geometry CG_3DBuffer(geometry geom, float8 radius, integer segments, integer buffer_type);
```

説明

入力ジオメトリ *geom* まわりに *radius* で指定した 3 次元バッファを生成します。バッファは 3 次元空間に構築され、ジオメトリの周囲の立体表現を生成します。*segments* パラメータでバッファの曲線の近似に使われる辺の数を決めます。4 辺分割が最低値として必要です。*buffer_type* で生成バッファのタイプを指定します。0: 円バッファ (デフォルト) 1: 平面バッファ 2: 正方形バッファ

入力ジオメトリは POINT または LINESTRING でなければなりません。

Availability: 3.6.0 - SFCGAL >= 2.0.0 が必要です



このメソッドには SFCGAL バックエンドが必要です。

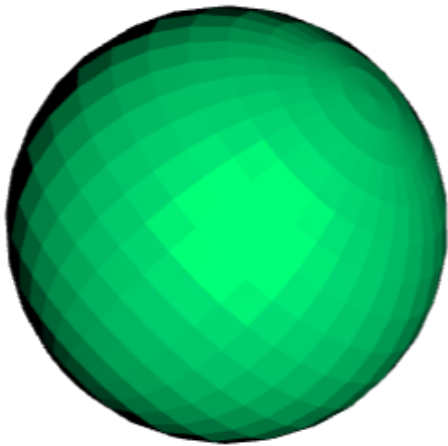
例

```
SELECT ST_AsText(CG_3DBuffer('POINT(0 0 0)', 1, 8, 0));
```

```
POLYHEDRALSURFACE Z (((0 0 1, 0.5 -0.5 0.71, 0 -0.71 0.71, 0 0 1)), ... )
```

次の画像は ST_AsX3D クエリの出力を **X3D Viewer** に貼り付けて描画したものです。

```
SELECT string_agg('<Shape
>' || ST_AsX3D(cgbuffer3d_output) || '<Appearance>
    <Material diffuseColor="0 0.8 0.2" specularColor="0 1 0"/>
  </Appearance>
</Shape
>', '');
```



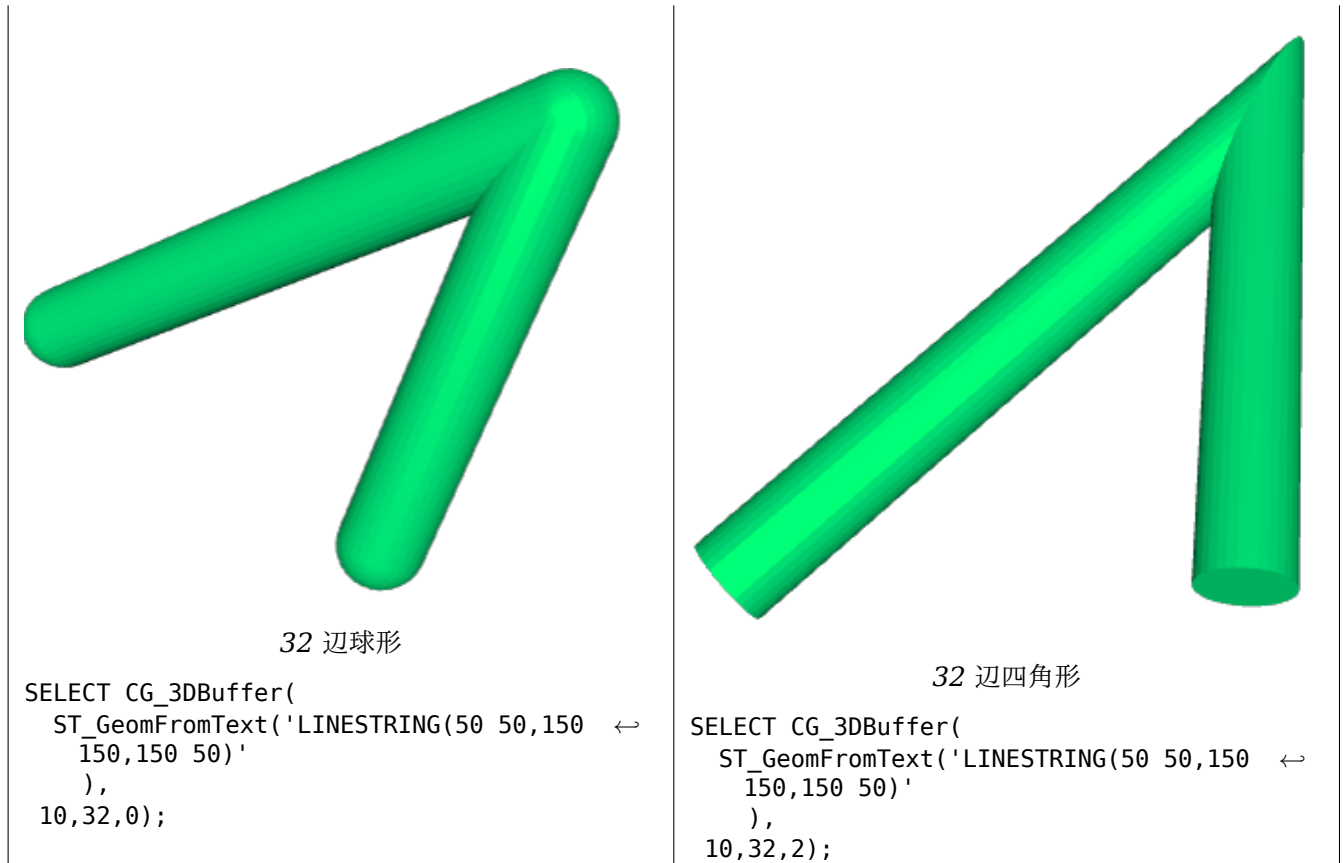
segments=32 (球形バッファ)

```
SELECT CG_3DBuffer(ST_GeomFromText('POINT (100 90)'), 50,32,0);
```



5 辺球形

```
SELECT CG_3DBuffer(
  ST_GeomFromText('POINT(100 90)'),
  50,5,0);
```



関連情報

[ST_Buffer](#), [ST_3DConvexHull](#), [ST_AsX3D](#)

8.3.47 CG_Rotate

CG_Rotate — ジオメトリを (0,0) を中心に指定した角度で回転させます。

Synopsis

geometry **CG_Rotate**(geometry geom, float8 angle);

説明

入力ジオメトリ *geom* を (0,0) を中心に *angle* ラジアンで回転させます。回転は 2 次元空間で行います。Z 値は変更されません。正の角度では反時計回りに回転させます。

Availability: 3.6.0 - SFCGAL >= 2.0.0 が必要です



このメソッドには SFCGAL バックエンドが必要です。

例

```
SELECT ST_AsText(CG_Rotate('LINESTRING(1 0, 0 1)', pi()/2));  
  
LINESTRING(6.123233995736766e-17 1, -1 6.123233995736766e-17)
```

関連情報

[CG_2DRotate](#), [ST_Rotate](#)

8.3.48 CG_2DRotate

CG_2DRotate — ジオメトリを指定した 2 次元ポイントを中心に指定した角度で回転させます。

Synopsis

geometry **CG_2DRotate**(geometry geom, float8 angle, float8 cx, float8 cy);

説明

入力ジオメトリ *geom* をポイント (*cx*, *cy*) を中心に *angle* ラジアンで回転させます。回転は 2 次元空間で行います。Z 値は削除されます。正の角度では反時計回りに回転させます。

Availability: 3.6.0 - SFCGAL >= 2.0.0 が必要です



このメソッドには SFCGAL バックエンドが必要です。

例

```
SELECT ST_AsText(CG_2DRotate('POINT(1 0)', pi()/2, 1, 1));  
  
POINT(2 1)
```

関連情報

[CG_Rotate](#), [CG_3DRotate](#)

8.3.49 CG_3DRotate

CG_3DRotate — ジオメトリを指定したベクトルを軸に 3 次元空間で回転させます。

Synopsis

geometry **CG_3DRotate**(geometry geom, float8 angle, float8 ax, float8 ay, float8 az);

説明

入力ジオメトリ *geom* を (0,0) と (*ax*, *ay*, *az*) とを結ぶベクトルを軸に *angle* ラジアンで 3 次元空間で回転させます。

Availability: 3.6.0 - SFCGAL >= 2.0.0 が必要です



このメソッドには SFCGAL バックエンドが必要です。



この関数は 3 次元に対応し、Z 値を削除しません。

例

```
SELECT ST_AsText(CG_3DRotate('POINT(1 0 0)', pi()/2, 0, 0, 1));
```

```
POINT Z (6.123233995736766e-17 1 0)
```

関連情報

[CG_RotateX](#), [CG_RotateY](#), [CG_RotateZ](#)

8.3.50 CG_RotateX

CG_RotateX — ジオメトリを X 軸を軸に指定した角度で回転させます。

Synopsis

geometry **CG_RotateX**(geometry geom, float8 angle);

説明

入力ジオメトリ *geom* を X 軸を軸に *angle* ラジアンで回転させます。

Availability: 3.6.0 - SFCGAL >= 2.0.0 が必要です



このメソッドには SFCGAL バックエンドが必要です。



この関数は 3 次元に対応し、Z 値を削除しません。

例

```
SELECT ST_AsText(CG_RotateX('POINT(0 1 0)', pi()/2));
```

```
POINT Z (0 6.123233995736766e-17 1)
```

関連情報

[CG_RotateY](#), [CG_RotateZ](#), [CG_3DRotate](#)

8.3.51 CG_RotateY

CG_RotateY — ジオメトリを Y 軸を軸に指定した角度で回転させます。

Synopsis

geometry **CG_RotateY**(geometry geom, float8 angle);

説明

入力ジオメトリ *geom* を Y 軸を軸に *angle* ラジアンで回転させます。

Availability: 3.6.0 - SFCGAL >= 2.0.0 が必要です

- ✔ このメソッドには SFCGAL バックエンドが必要です。
- ✔ この関数は 3 次元に対応し、Z 値を削除しません。

例

```
SELECT ST_AsText(CG_RotateY('POINT(1 0 0)', pi()/2));
```

```
POINT Z (6.123233995736766e-17 0 -1)
```

関連情報

[CG_RotateX](#), [CG_RotateZ](#), [CG_3DRotate](#)

8.3.52 CG_RotateZ

CG_RotateZ — ジオメトリを Z 軸を軸に指定した角度で回転させます。

Synopsis

geometry **CG_RotateZ**(geometry geom, float8 angle);

説明

入力ジオメトリ *geom* を Z 軸を軸に *angle* ラジアンで回転させます。

Availability: 3.6.0 - SFCGAL >= 2.0.0 が必要です

- ✔ このメソッドには SFCGAL バックエンドが必要です。
- ✔ この関数は 3 次元に対応し、Z 値を削除しません。

例

```
SELECT ST_AsText(CG_RotateZ('POINT(1 0 0)', pi()/2));
```

```
POINT Z (6.123233995736766e-17 1 0)
```

関連情報

[CG_RotateX](#), [CG_RotateY](#), [CG_3DRotate](#)

8.3.53 CG_Scale

CG_Scale — ジオメトリを全ての次元で与えられた係数で均一に拡大縮小します。

Synopsis

geometry **CG_Scale**(geometry geom, float8 factor);

説明

入力ジオメトリ *geom* を全ての次元 (X,Y,Z) で与えられた係数 *factor* で均一に拡大縮小します。拡大縮小は (0,0,0) を基準に実行します。

Availability: 3.6.0 - SFCGAL >= 2.0.0 が必要です



このメソッドには SFCGAL バックエンドが必要です。

例

```
SELECT ST_AsText(CG_Scale('LINESTRING(1 1, 2 2)', 2));
```

```
LINESTRING(2 2,4 4)
```

関連情報

[CG_3DScale](#), [CG_3DScaleAroundCenter](#), [ST_Scale](#)

8.3.54 CG_3DScale

CG_3DScale — ジオメトリを X,Y,Z 軸に沿って個別の係数で拡大縮小します。

Synopsis

geometry **CG_3DScale**(geometry geom, float8 factorX, float8 factorY, float8 factorZ);

説明

入力ジオメトリ *geom* を X,Y,Z 軸に沿って個別の係数で拡大縮小します。拡大縮小は (0,0,0) を基準に実行します。

Availability: 3.6.0 - SFCGAL >= 2.0.0 が必要です



このメソッドには SFCGAL バックエンドが必要です。



この関数は 3 次元に対応し、Z 値を削除しません。

例

```
SELECT ST_AsText(CG_3DScale('POINT(1 1 1)', 2, 3, 4));
```

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

関連情報

[CG_Scale](#), [CG_3DScaleAroundCenter](#)

8.3.55 CG_3DScaleAroundCenter

CG_3DScaleAroundCenter — ジオメトリを 3 次元空間で指定した点を中心として拡大縮小します。

Synopsis

geometry **CG_3DScaleAroundCenter**(geometry geom, float8 factorX, float8 factorY, float8 factorZ, float8 centerX, float8 centerY, float8 centerZ);

説明

入力ジオメトリ *geom* を X,Y,Z 軸に沿って個別の係数で、点 (*centerX*, *centerY*, *centerZ*) を中心に拡大縮小します。

Availability: 3.6.0 - SFCGAL >= 2.0.0 が必要です



このメソッドには SFCGAL バックエンドが必要です。



この関数は 3 次元に対応し、Z 値を削除しません。

例

```
SELECT ST_AsText(CG_3DScaleAroundCenter('POINT(2 2 2)', 0.5, 0.5, 0.5, 1, 1, 1));
```

```
POINT Z (1.5 1.5 1.5)
```

関連情報

[CG_Scale](#), [CG_3DScale](#)

8.3.56 CG_Translate

CG_Translate — ジオメトリを与えたオフセットで 2 次元で移動させます。

Synopsis

geometry **CG_Translate**(geometry geom, float8 deltaX, float8 deltaY);

説明

入力ジオメトリ *geom* を、X 値に *deltaX* を、Y 値に *deltaY* を、それぞれ足します。Z 値は削除されます。

Availability: 3.6.0 - SFCGAL >= 2.0.0 が必要です



このメソッドには SFCGAL バックエンドが必要です。

例

```
SELECT ST_AsText(CG_Translate('LINESTRING(1 1, 2 2)', 1, -1));
```

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

関連情報

[CG_3DTranslate](#), [ST_Translate](#)

8.3.57 CG_3DTranslate

CG_3DTranslate — ジオメトリを与えたオフセットで 3 次元で移動させます。

Synopsis

geometry **CG_3DTranslate**(geometry geom, float8 deltaX, float8 deltaY, float8 deltaZ);

説明

入力ジオメトリ *geom* を、X 値に *deltaX* を、Y 値に *deltaY* を、Z 値に *deltaZ* を、それぞれ足します。

Availability: 3.6.0 - SFCGAL >= 2.0.0 が必要です



このメソッドには SFCGAL バックエンドが必要です。



この関数は 3 次元に対応し、Z 値を削除しません。

例

```
SELECT ST_AsText(CG_3DTranslate('POINT(1 1 1)', 1, -1, 2));
```

```
POINT Z (2 0 3)
```


関連情報

[CG_Translate](#), [ST_Translate](#)

8.3.58 CG_Simplify

CG_Simplify — 主要な地物と Z/M 値を維持しつつジオメトリの複雑さを減らします。

Synopsis

```
geometry CG_Simplify(geometry geom, double precision threshold, boolean preserveTopology = false);
```

説明

SFCGAL の単純化アルゴリズムを使ってジオメトリを単純化します。ジオメトリの主要な特徴を保存しつつポイントや頂点の数を減らします。この関数は Z 値や M 値の保存もします。

アルゴリズムは制約付き三角分割を基にしている、Z 値と M 値の保存のための追加操作を持つ [CGAL Polyline Simplification 2](#) ライブラリを使っています。トポロジが保存され、ジオメトリどうしがインタセクトしているとき、Z 値と M 値はインタセクトしているポイントで補間されます。

この関数は 3 次元地形のようなジオメトリ (2.5 次元) で動作しますが、壁のような垂直なサーフェス向けには作られていません。

Availability: 3.6.0 - SFCGAL >= 2.1.0 が必要



このメソッドには SFCGAL バックエンドが必要です。



この関数は 3 次元に対応し、Z 値を削除しません。



この関数は M 値に対応します。

パラメータ

geom 入力ジオメトリ

threshold 単純化に関する最大距離のしきい値 (ジオメトリの単位)。この値が大きいほど結果ジオメトリが単純になります。

preserveTopology TRUE に設定すると、この関数はジオメトリのトポロジを確実に保存します。このモードでジオメトリがインタセクトするとき、インタセクトとするポイントにおける Z 値と M 値は補完値になります。デフォルト値は FALSE です。

返り値

Z 値と M 値を保存した単純化されたジオメトリが返ります。

例

This example simplifies a polygon with a threshold of 0.5.

```
SELECT ST_AsText(CG_Simplify(ST_GeomFromText('POLYGON((0 0, 0 1, 0.1 1, 0.2 1, 0.3 1, 0.4 1, 0.5 1, 1 1, 1 0, 0 0))'), 0.5));
```

Simplify a 3D terrain geometry while preserving topology and Z values.

```
SELECT ST_AsText(CG_Simplify(ST_GeomFromText('LINESTRING Z(0 0 0, 0 1 1, 0.1 1 1, 0.2 1 1, 0.3 1 1, 1 1 2)'), 0.2, true));
```

Simplify a geometry with both Z and M values.

```
SELECT ST_AsText(CG_Simplify(ST_GeomFromText('LINESTRING ZM(0 0 0 1, 0 1 1 2, 0.1 1 1 3, 0.2 1 1 4, 0.3 1 1 5, 1 1 2 6)'), 0.2));
```

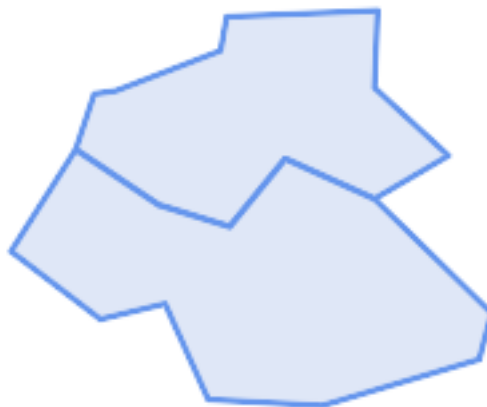
Simplify two geometries together, preserving Z and M values without preserving topology.

```
SELECT ST_AsText(CG_Simplify('GEOMETRYCOLLECTION ZM(LINESTRING ZM(-1 -1 3 4, 0 0 10 100, 1 1 20 200, 0 2 15 150, 0 5 30 300, 2 19 25 250, -4 20 15 150), POLYGON ZM((0 0 10 100, 1 1 20 200, 0 2 15 150, 0 5 30 300, 2 19 25 250, -4 20 15 150, 0 0 10 100)))', 2, false));
```

Simplify two geometries together, preserving Z and M values and topology.

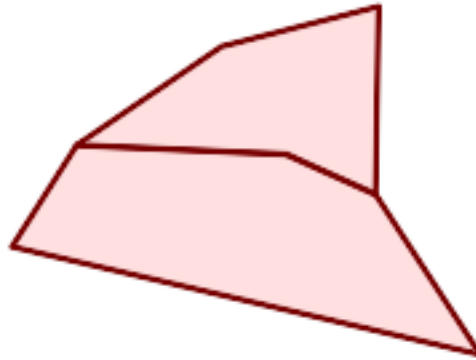
```
SELECT ST_AsText(CG_Simplify('GEOMETRYCOLLECTION ZM(LINESTRING ZM(-1 -1 3 4, 0 0 10 100, 1 1 20 200, 0 2 15 150, 0 5 30 300, 2 19 25 250, -4 20 15 150), POLYGON ZM((0 0 10 100, 1 1 20 200, 0 2 15 150, 0 5 30 300, 2 19 25 250, -4 20 15 150, 0 0 10 100)))', 2, true));
```

```
WITH depts_pds as (SELECT ST_GeomFromText('GEOMETRYCOLLECTION(
POLYGON((88.46 158.85,90.77 171.54,147.31 173.85,146.15 145,173.85 119.62,146.15 103.46,112.69 118.46,91.92 93.08,65.38 101.15,34.23 121.92,41.15 142.69,49.23 143.85,88.46 158.85)),
POLYGON((112.69 118.46,146.15 103.46,190 60.77,185.38 43.46,126.54 26.15,83.85 28.46,67.69 64.23,43.46 58.46,10 83.85,34.23 121.92,65.38 101.15,91.92 93.08,112.69 118.46)))
') as geom)
SELECT geom FROM depts_pds;
```



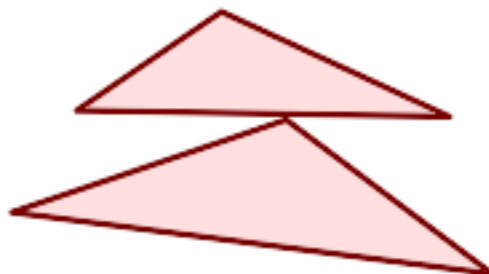
元のジオメトリ

```
WITH depts_pds as (SELECT ST_GeomFromText('GEOMETRYCOLLECTION(
POLYGON((88.46 158.85,90.77 171.54,147.31 173.85,146.15 145,173.85 119.62,146.15  ←
103.46,112.69 118.46,91.92 93.08,65.38 101.15,34.23 121.92,41.15 142.69,49.23  ←
143.85,88.46 158.85)),
POLYGON((112.69 118.46,146.15 103.46,190 60.77,185.38 43.46,126.54 26.15,83.85 28.46,67.69  ←
64.23,43.46 58.46,10 83.85,34.23 121.92,65.38 101.15,91.92 93.08,112.69 118.46)))
') as geom)
SELECT (ST_Dump(CG_Simplify(geom, 0.5, true))).geom FROM depts_pds;
```



0.5 かつトポロジ保存を指定して単純化したもの

```
WITH depts_pds as (SELECT ST_GeomFromText('GEOMETRYCOLLECTION(
POLYGON((88.46 158.85,90.77 171.54,147.31 173.85,146.15 145,173.85 119.62,146.15  ←
103.46,112.69 118.46,91.92 93.08,65.38 101.15,34.23 121.92,41.15 142.69,49.23  ←
143.85,88.46 158.85)),
POLYGON((112.69 118.46,146.15 103.46,190 60.77,185.38 43.46,126.54 26.15,83.85 28.46,67.69  ←
64.23,43.46 58.46,10 83.85,34.23 121.92,65.38 101.15,91.92 93.08,112.69 118.46)))
') as geom)
SELECT (ST_Dump(CG_Simplify(geom, 0.5, false))).geom FROM depts_pds;
```



0.5 だがトポロジ保存なしで単純化したもの

関連情報

[ST_Simplify](#), [ST_SimplifyPreserveTopology](#)

8.3.59 CG_3DAlphaWrapping

CG_3DAlphaWrapping — 厳密にジオメトリを囲む 3 次元アルファラッピングを計算します。

Synopsis

geometry **CG_3DAlphaWrapping**(geometry geom, integer relative_alpha, integer relative_offset);

説明

ジオメトリのポイントから **3D アルファラッピング** を計算します。アルファラッピングは、入力を厳密に囲む水漏れの無い方向つきサーフェスメッシュです。 **アルファシェイプ** の拡張または改良とみることができます。

relative_alpha パラメータで、出力に現れる地物を制御します。0 から無限大を取ります。relative_alpha が小さいほど、より単純な出力になりますが、元の入力より精度の低い表現になります。

relative_offset パラメータで、結果の余裕を制御します。0 から無限大まで値を取ることができます。このパラメータが 0 に設定されると、値は自動的に relative_alpha パラメータを基に決定されます。

Availability: 3.6.0 - SFCGAL >= 2.1.0 が必要



このメソッドには SFCGAL バックエンドが必要です。



この関数は 3 次元に対応し、Z 値を削除しません。

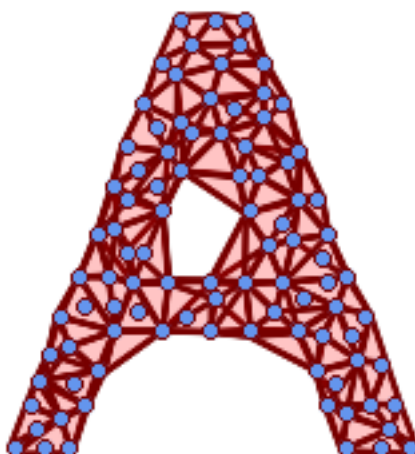
例

```
SELECT CG_3DAlphaWrapping(
  'MULTIPOINT((63 84),(76 88),(68 73),(53 18),(91 50),(81 70),
    (88 29),(24 82),(32 51),(37 23),(27 54),(84 19),(75 87),(44 42),(77 67),(90 30) ←
    ,(36 61),(32 65),
    (81 47),(88 58),(68 73),(49 95),(81 60),(87 50),(78 16),(79 21),(30 22),(78 43) ←
    ,(26 85),(48 34),
    (35 35),(36 40),(31 79),(83 29),(27 84),(52 98),(72 95),(85 71),(75 84),(75 77) ←
    ,(81 29),(77 73),
    (41 42),(83 72),(23 36),(89 53),(27 57),(57 97),(27 77),(39 88),(60 81),(80 72) ←
    ,(54 32),(55 26),
    (62 22),(70 20),(76 27),(84 35),(87 42),(82 54),(83 64),(69 86),(60 90),(50 86) ←
    ,(43 80),(36 73),
    (36 68),(40 75),(24 67),(23 60),(26 44),(28 33),(40 32),(43 19),(65 16),(73 16) ←
    ,(38 46),(31 59),
    (34 86),(45 90),(64 97))'::geometry,
  10);
```

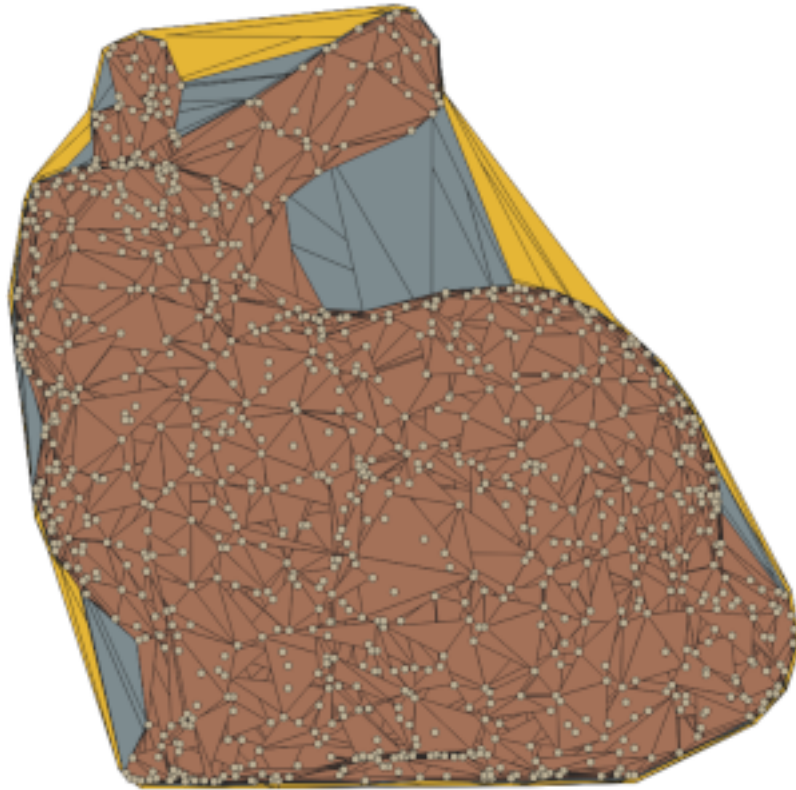


MULTIPOINT のアルファラッピング (*CG_OptimalAlphaShape*と同じ例です)

```
SELECT CG_3DAlphaWrapping(
  'MULTIPOINT((132 64),(114 64),(99 64),(81 64),(63 64),(57 49),
    (52 36),(46 20),(37 20),(26 20),(32 36),(39 55),(43 69),(50 84),(57 100),(63  ←
      118),(68 133),(74 149),
    (81 164),(88 180),(101 180),(112 180),(119 164),(126 149),(132 131),(139 113)  ←
      ,(143 100),(150 84),(157 69),(163 51),
    (168 36),(174 20),(163 20),(150 20),(143 36),(139 49),(132 64),(99 151),(92 138)  ←
      ,(88 124),(81 109),(74 93),(70 82),
    (83 82),(99 82),(112 82),(126 82),(121 96),(114 109),(110 122),(103 138),(99  ←
      151),(34 27),(43 31),(48 44),(46 58),
    (52 73),(63 73),(61 84),(72 71),(90 69),(101 76),(123 71),(141 62),(166 27),(150  ←
      33),(159 36),(146 44),(154 53),
    (152 62),(146 73),(134 76),(143 82),(141 91),(130 98),(126 104),(132 113),(128  ←
      127),(117 122),(112 133),(119 144),
    (108 147),(119 153),(110 171),(103 164),(92 171),(86 160),(88 142),(79 140),(72  ←
      124),(83 131),(79 118),(68 113),
    (63 102),(68 93),(35 45))'::geometry,
  14);
```



MULTIPOINT のアルファラッピング (*ST_ConcaveHull*と同じ例です)



relative_alpha パラメータ値の 5, 10, 20 の効果。5 の時は荒い出力です。20 まで上がっていくと、結果の精度と粒度が大幅に改善されます。

関連情報

[CG_AlphaShape](#)

8.3.60 CG_NurbsCurveInterpolate

`CG_NurbsCurveInterpolate` — Creates an interpolating NURBS curve passing through all given data points

Synopsis

`geometry` **`CG_NurbsCurveInterpolate`**(`geometry data_points`, integer degree);

説明

Creates a NURBS curve that interpolates through all provided data points. The resulting curve passes exactly through each point in the input geometry.

Parameters:

- `data_points` - A `LINESTRING` geometry containing the points to interpolate through.
- `degree` - Polynomial degree of the NURBS curve (typically 2 or 3).

Availability: 3.7.0 - SFCGAL >= 2.3.0 が必要。



このメソッドには SFCGAL バックエンドが必要です。



この関数は 3 次元に対応し、Z 値を削除しません。

例

This example creates a smooth curve interpolating through five points.

```
SELECT CG_NurbsCurveInterpolate(  
  'LINESTRING(0 0, 1 2, 3 3, 5 1, 6 0)::geometry,  
  3  
);  
  
-- 3D interpolation  
SELECT CG_NurbsCurveInterpolate(  
  'LINESTRING Z(0 0 0, 1 2 1, 3 3 0, 5 1 -1, 6 0 0)::geometry,  
  2  
);
```

関連情報

[CG_NurbsCurveApproximate](#), [ST_MakeNurbsCurve](#)

8.3.61 CG_NurbsCurveApproximate

CG_NurbsCurveApproximate — Creates an approximating NURBS curve fitting data points within a tolerance

Synopsis

geometry **CG_NurbsCurveApproximate**(geometry data_points, integer degree, float8 tolerance, integer max_control_points=100);

説明

Creates a NURBS curve that approximates the given data points within the specified tolerance. Unlike interpolation, the curve does not necessarily pass through all points but provides a smooth fit with fewer control points.

Parameters:

- **data_points** - A LINESTRING geometry containing the points to approximate.
- **degree** - Polynomial degree of the NURBS curve.
- **tolerance** - Maximum allowed distance between the curve and data points.
- **max_control_points** - Optional maximum number of control points (default: 100). Limits curve complexity.

Availability: 3.7.0 - SFCGAL >= 2.3.0 が必要。



このメソッドには SFCGAL バックエンドが必要です。



この関数は 3 次元に対応し、Z 値を削除しません。

例

This example approximates noisy data using a tolerance.

```
SELECT CG_NurbsCurveApproximate(  
  'LINESTRING(0 0, 1 2.1, 2 1.9, 3 3.2, 4 2.8, 5 1)'::geometry,  
  2,  
  0.5  
);  
  
-- Control maximum complexity  
SELECT CG_NurbsCurveApproximate(  
  'LINESTRING(0 0, 1 2, 2 2, 3 3, 4 3, 5 1, 6 0, 7 -1, 8 0)'::geometry,  
  3,  
  0.2,  
  20 -- max 20 control points  
);
```

関連情報

[CG_NurbsCurveInterpolate](#), [ST_MakeNurbsCurve](#)

8.3.62 CG_NurbsCurveDerivative

CG_NurbsCurveDerivative — Computes the derivative (tangent, curvature) of a NURBS curve at a given parameter

Synopsis

geometry **CG_NurbsCurveDerivative**(geometry nurbscurve, float8 parameter, integer derivative_order);

説明

Computes the derivative of a NURBS curve at a specific parameter value. The derivative represents the rate of change at that point:

- Order 1: Tangent vector (direction and speed)
- Order 2: Curvature vector (how the tangent changes)
- Order 3: Third-order derivative (supported maximum)

Parameters:

- nurbscurve - A NURBS curve geometry.
- parameter - Parameter value for evaluation (typically 0.0 to 1.0).
- derivative_order - Order of derivative to compute (1, 2, or 3).

Returns a POINT geometry representing the derivative vector at the specified parameter.

Availability: 3.7.0 - SFCGAL >= 2.3.0 が必要。



このメソッドには SFCGAL バックエンドが必要です。



この関数は 3 次元に対応し、Z 値を削除しません。

例

This example returns the tangent vector at the curve midpoint.

```
WITH nurbs AS (  
    SELECT ST_MakeNurbsCurve(2, 'LINESTRING(0 0, 5 10, 10 0)::geometry) AS geom  
)  
SELECT CG_NurbsCurveDerivative(geom, 0.5, 1) FROM nurbs;  
  
-- Get curvature at start of curve  
WITH nurbs AS (  
    SELECT ST_MakeNurbsCurve(3, 'LINESTRING(0 0, 2 5, 5 5, 8 2, 10 0)::geometry) AS geom  
)  
SELECT CG_NurbsCurveDerivative(geom, 0.0, 2) FROM nurbs;  
  
-- Analyze tangent at multiple points  
WITH nurbs AS (  
    SELECT ST_MakeNurbsCurve(2, 'LINESTRING(0 0, 5 10, 10 0)::geometry) AS geom  
)  
SELECT  
    t.param,  
    CG_NurbsCurveDerivative(n.geom, t.param, 1) AS tangent  
FROM nurbs n  
CROSS JOIN (VALUES (0.0), (0.25), (0.5), (0.75), (1.0)) AS t(param);
```

関連情報

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

Chapter 9

トポロジ

PostGIS トポロジ型と関数は、フェイス、エッジ、ノード等のトポロジオブジェクトを管理するために使います。

PostGIS Day Paris 2011 における Sandro Santilli さんの講演が、PostGIS トポロジの概略説明として良いです。 [Topology with PostGIS 2.0 slide deck](#) にあります。

Vincent Picavet さんはトポロジとは何か、どのように使われるか、および、対応する FOSS4G ツールに関する良い概略説明を [PostGIS Topology PGConf EU 2012](#) で出しています。

トポロジーベースの GIS データベースの例に [US Census Topologically Integrated Geographic Encoding and Referencing System \(TIGER\)](#) データベースがあります。PostGIS トポロジーの実験でデータが必要となる場合に使ってください。

PostGIS トポロジーモジュールは長い間存在していましたが、公式ドキュメントには常に存在しているわけではありませんでした。広範なクリーンアップで非推奨関数を削除し、既知の使い勝手に関する問題の修正、機能と関数の文書化、新機能追加、SQL-MM 適合性の改善がなされました。

このプロジェクトの詳細情報は [PostGIS Topology Wiki](#) にあります。

このモジュールに関する全ての関数とテーブルは、`topology` スキーマにインストールされます。

SQL/MM 標準で定義される関数は `ST_` プリフィックスを持ち、PostGIS 特有の関数はこのプリフィックスを持ちません。

トポロジー機能はデフォルトでビルドされます。Chapter 2 で説明されている通り、ビルド時のコンフィギュアオプション `--without-topology` を指定することで、無効にできます。

9.1 トポロジープリミティブテーブル

あらゆるトポロジーの中核的なプリミティブは `edge_data`、`node`、`face` テーブルに格納されます。これらのテーブルは [CreateTopology](#) で生成されたスキーマ内にあります。`edge_data` の各タプルは有向エッジを表現しています。`start_node` から `end_node` への有向曲線と、その曲線の進行方向左側にあるフェイスの識別番号 (`left_face`) と右側のもの (`right_face`) とを併せて記録しています。このため、ジオメトリ線分が 2 個のフェイスに属するとき、同じ線分が 2 度 (方向ごとに 1 度ずつ) 出現することがあります。

`next_left_edge` カラムと `next_right_edge` カラムはフェイスを辿る方法をエンコードしてこの方向情報を完全にしています。正負符号付整数を格納していて、絶対値は進行方向を向いた時の次のエッジを示し、正負符号で次のエッジが順方向化逆方向かを決定しています。正式には、次の規則があります (エッジを `e` とします):

- `abs(next_left_edge)` は `e` の左にあるフェイスを辿る場合に次に到達するエッジの識別番号です。値が正数の場合には `e` の終端ノードから次のエッジの格納されている通りの方向に進みます。負数の場合には次のエッジは逆向きに辿ることになるので、共有するフェイスは辿っている者から見たら左側にあります。

- `abs(next_right_edge)` は `e` の右にあるフェイスを辿る場合に次に到達するエッジの識別番号です。値が正数の場合には次のエッジが `e` の終端ノードから始まり次のエッジの格納されている通りの方向に進みます。負数の場合には次のエッジは逆向きにたどることになるので、共有するフェイスは辿っている者から見たら左側にあります。負数の場合には次のエッジを逆方向で辿る、すなわちエッジの終端から始めることになり、右側のフェイスが保存されます。
- ゼロは対応する側でエッジがぶら下がっていることを示します (たとえば接続するフェイスが `0` のユニバーサルフェイスになっている孤立エッジ)。edge ビューにある `abs_next_left_edge` カラムと `abs_next_right_edge` カラムは絶対値にして扱いです。

この表現は二重連結辺リストの異種で、多数のトポロジルーチンで利用されます。`GetRingEdges`や`ValidateTopology`といった関数によるフェイス境界の再構築や矛盾の診断に使われます。したがって評価時に「不正な `next_left_edge`」と「不正な `next_right_edge`」診断が報告されます。`AddEdge`のようなコンストラクターは `next_*` 属性を普通の自己参照で初期化します。`ST_AddEdgeModFace`と`ST_RemEdgeModFace`を含む編集ルーチンは、エッジが挿入されるか削除されると同時にリンクを更新します。他の大量処理 (たとえば`Polygonize`) は意図的にカラムに値を設定しないことがあり、文書ではふるまいに関する注意を明示的に出しています。

9.2 トポロジ型

9.2.1 getfaceedges_returntype

`getfaceedges_returntype` — 順序番号とエッジ番号で構成される複合型。

説明

順序番号とエッジ番号で構成される複合型。これは `ST_GetFaceEdges` と `GetNodeEdges` の返り値の型です。

1. `sequence` (整数): トポロジスキーマと SRID を定義する `topology.topology` テーブルで定義されるトポロジへの参照です。
2. `edge` (整数): エッジの識別番号です。

9.2.2 TopoGeometry

`TopoGeometry` — トポロジとして定義されたジオメトリを表現する型です。

説明

特定のトポロジレイヤ内のトポロジジオメトリを参照する複合型で、特定のタイプと特定の ID を持ちます。`TopoGeometry` の要素は `topology_id`, `layer_id`, `id integer`, `type integer` の各属性です。

1. `topology_id` (整数): トポロジスキーマと SRID を定義する `topology.topology` テーブルで定義されているトポロジへの参照です。
2. `layer_id` (整数): `TopoGeometry` が属する `layers` テーブルにおける `layer_id` です。`topology_id` と `layer_id` との組み合わせで、`topology.layers` テーブルを一意に参照できます。
3. `id` (整数): それぞれのトポロジで `TopoGeometry` を一意にするための順序整数で、自動生成されます。
4. `type` (1 から 4 の整数でジオメトリタイプを定義): 1:[multi]point, 2:[multi]line, 3:[multi]poly, 4:collection となります。

キャストの挙動

本節では、このデータ型で許容される明示的なキャストと自動キャストの一覧を挙げます。

キャスト先	ふるまい
geometry	自動

関連情報

[CreateTopoGeom](#)

9.2.3 validate_topology_returntype

`validate_topology_returntype` — エラーメッセージとエラーの場所を示す `id1` と `id2` からなる複合型です。これは `ValidateTopology` が返す型です。

説明

エラーメッセージと二つの整数からなる複合型です。[ValidateTopology](#)関数は、この集合を返します。評価エラーを示し、`id1` と `id2` でエラーを含むトポロジオブジェクトの識別番号を示します。

- error (varchar):** エラーのタイプを示します。
現在のエラー記述: `coincident nodes` (訳注: ノード重複), `edge crosses node` (訳注: エッジとノードのクロス), `edge not simple` (訳注: 単純でないエッジ), `edge end node geometry mismatch` (訳注: 終了ノードジオメトリの不整合), `edge start node geometry mismatch` (訳注: 開始ノードジオメトリの不整合), `face overlaps face` (訳注: フェイス同士のオーバーラップ), `face within face` (訳注: フェイス間の包含)
- id1 (整数):** エラーを持つエッジ/フェイス/ノードの識別番号を示します。
- id2 (整数):** 2 オブジェクトでのエラーにおける二つ目のエッジ/ノードの識別番号を示します。

関連情報

[ValidateTopology](#)

9.3 トポロジドメイン

9.3.1 TopoElement

`TopoElement` — 二つの整数の配列で、通常 `TopoGeometry` 要素を識別するために使われます。

説明

二つの整数の配列で、単純または階層を持つ[TopoGeometry](#)の一つのコンポーネントを表現するために使われます。

単純な `TopoGeometry` の場合は、配列の最初の要素がトポロジのプリミティブの識別を表現し、二つ目の要素がタイプ (1: ノード 2: エッジ, 3: フェイス) を表現します。階層的な `TopoGeometry` の場合は、一つ目の要素が子の `TopoGeometry` の識別子を表現し、二つ目の要素はレイヤ識別子を表現します。

**Note**

階層的な TopoGeometry については、全ての子の TopoGeometry 要素は同じ子レイヤから来ます。子レイヤは、定義された TopoGeometry のレイヤの topology.layer レコード内で指定されます。

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}
```

This example shows what happens when casting a three-element array to topology.topoelement. The domain requires a two-element array, so the dimension check fails.

```
SELECT ARRAY[1,2,3]::topology.topoelement;
```

```
ERROR:  value for domain topology.topoelement violates check constraint "dimensions"
```

関連情報

[GetTopoGeomElements](#), [TopoElementArray](#), [TopoGeometry](#), [TopoGeom_addElement](#), [TopoGeom_remElement](#)

9.3.2 TopoElementArray

TopoElementArray — TopoElement オブジェクトの配列。

説明

1 以上の TopoElement オブジェクトの配列で、通常は TopoGeometry オブジェクトのコンポーネントを分配するために使われます。

例

```
SELECT '{{1,2},{4,3}}'::topology.topoelementarray AS tea;
```

```
{{1,2},{4,3}}
```

This is the more verbose equivalent.

```
SELECT ARRAY[ARRAY[1, 2], ARRAY[4, 3]]::topology.topoelementarray AS tea;
```

```
{{1,2},{4,3}}
```

This example uses the array aggregate function packaged with topology.

```
SELECT topology.TopoElementArray_Agg(ARRAY[e, t] ORDER BY e, t) As tea
FROM generate_series(1, 4) As e CROSS JOIN generate_series(1, 3) As t;
```

```
{{1,1},{1,2},{1,3},{2,1},{2,2},{2,3},{3,1},{3,2},{3,3},{4,1},{4,2},{4,3}}
```

```
SELECT '{{1,2,4},{3,4,5}}'::topology.topoelementarray AS tea;
```

```
ERROR:  value for domain topology.topoelementarray violates check constraint "dimensions"
```

関連情報

[TopoElement](#), [GetTopoGeomElementArray](#), [TopoElementArray_Agg](#)

9.4 トポロジ管理と TopoGeometry 管理

9.4.1 AddTopoGeometryColumn

AddTopoGeometryColumn — 既存のテーブルに TopoGeometry カラムを追加し、`topology.layer` 内に新しいレイヤとして新しいカラムを登録して、新しい `layer_id` を返します。

Synopsis

```
integer AddTopoGeometryColumn(name topology_name, name schema_name, name table_name,
name column_name, varchar feature_type, integer child_layer);
integer AddTopoGeometryColumn(name topology_name, regclass tab, name column_name, integer
layer_id, varchar feature_type, integer child_layer);
```

説明

それぞれの TopoGeometry オブジェクトは、特定のトポロジの特定のレイヤに属します。TopoGeometry オブジェクト生成の前に、トポロジレイヤの生成が必要です。トポロジレイヤは地物テーブルとトポロジとで組織されます。また、タイプと階層の情報を持ちます。レイヤの生成には **AddTopoGeometryColumn()** を使います。

この関数は、リクエストされたカラムをテーブルに追加し、`topology.layer` テーブルに、与えられた全ての情報のレコードを追加します。

[`child_layer`] を指定しない (または NULL を指定する) 場合、このレイヤは、基本的な TopoGeometry (プリミティブなトポロジ要素で構成) を含みます。指定する場合、このレイヤは階層的な TopoGeometry (`child_layer` からの TopoGeometry で構成) を持ちます。

レイヤが生成される (この識別番号は、**AddTopoGeometryColumn** 関数が返します) と、TopoGeometry オブジェクトをこの中に構築する準備ができます。

妥当な `feature_type`: POINT, MULTIPOINT, LINE, MULTILINE, POLYGON, MULTIPOLYGON, COLLECTION

Availability: 1.1

例

This example creates the new table in the `ma_topo` schema. The table could be created in a different schema, in which case `topology_name` and `schema_name` would be different.

```
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');
```

関連情報

[DropTopoGeometryColumn](#), [toTopoGeom](#), [CreateTopology](#), [CreateTopoGeom](#)

9.4.2 RenameTopoGeometryColumn

`RenameTopoGeometryColumn` — `TopoGeometry` カラムの名前を変更します

Synopsis

`topology.layer` **`RenameTopoGeometryColumn`**(`regclass layer_table`, `name feature_column`, `name new_name`)

説明

この関数は存在する `TopoGeometry` カラムの名前を変更します。この際、メタデータ情報が適切に更新されるようにします。

Availability: 3.4.0

例

```
SELECT topology.RenameTopoGeometryColumn('public.parcels', 'topogeom', 'tgeom');
```

関連情報

[AddTopoGeometryColumn](#), [RenameTopology](#)

9.4.3 DropTopology

`DropTopology` — 使用上の注意: この関数によって、トポロジスキーマが削除され、`topology.topology` テーブルからの参照が削除され、`geometry_columns` テーブルから削除対象スキーマ内のテーブルへの参照が削除されます。

Synopsis

`integer` **`DropTopology`**(`varchar topology_schema_name`);

説明

トポロジスキーマを削除し、`topology.topology` テーブルからの参照を削除し、`geometry_columns` テーブルから削除対象スキーマ内のテーブルへの参照を削除します。この関数は * 十分に注意してご使用下さい *。残しておきたかったデータが破壊される可能性があります。スキーマが存在しない場合、スキーマへの参照に関するエントリの削除だけを行います。

Availability: 1.1

例

`ma_topo` スキーマをカスケード削除し、`topology.topology` にある参照のうち関係するものを全て削除します。

```
SELECT topology.DropTopology('ma_topo');
```

関連情報

[DropTopoGeometryColumn](#)

9.4.4 RenameTopology

RenameTopology — トポロジ名を変更します

Synopsis

```
varchar RenameTopology(varchar old_name, varchar new_name);
```

説明

トポロジスキーマの名前を変更し、`topology.topology` テーブルに格納されているメタデータを更新します。

Availability: 3.4.0

例

`topo_stage` から `topo_prod` にトポロジ名を変更します。

```
SELECT topology.RenameTopology('topo_stage', 'topo_prod');
```

関連情報

[CopyTopology](#), [RenameTopoGeometryColumn](#)

9.4.5 DropTopoGeometryColumn

DropTopoGeometryColumn — `schema_name` で指定されたスキーマ内にある `table_name` で指定されたテーブルから TopoGeometry カラムを削除し、`topology.layer` テーブルにある登録を解除します。

Synopsis

text **DropTopoGeometryColumn**(varchar schema_name, varchar table_name, varchar column_name);

説明

schema_name で指定されたスキーマ内にある table_name で指定されたテーブルから TopoGeometry カラムを削除し、topology.layer テーブルにある登録を解除します。削除の概要報告を返します。ご注意: この関数は、参照整合性チェックをすり抜けるために、まず NULL 値に上書きしてから削除します。

Availability: 1.1

例

```
SELECT topology.DropTopoGeometryColumn('ma_topo', 'parcel_topo', 'topo');
```

関連情報

[AddTopoGeometryColumn](#)

9.4.6 FixCorruptTopoGeometryColumn

FixCorruptTopoGeometryColumn — postgis_topology を 3.6.0 以上にアップグレードする際に発生するジオメトリの間違いを訂正します。

Synopsis

text **FixCorruptTopoGeometryColumn**(name layerSchema, name layerTable, name layerColumn);

説明

PostGIS トポロジー 3.6.0 未満から 3.6.0 以上にアップグレードする時、トポジオメトリカラムの定義が変更されました。このためアップグレード前に作られたトポジオメトリで間違いが発生します。この関数は影響を受けるテーブルの該当する間違いを 1 修正します。

Availability: 3.6.1

例

全てのトポロジーカラムの修正

```
SELECT topology.FixCorruptTopoGeometryColumn(schema_name, table_name, feature_column)
FROM topology.layer;
```

関連情報

[UpgradeTopology](#)

9.4.7 Populate_Topology_Layer

`Populate_Topology_Layer` — テーブルからメタデータを読み、`topology.layer` テーブルに不足しているものを追加します。

Synopsis

```
setof record Populate_Topology_Layer();
```

説明

テーブルの制約を読み、`topology.layer` テーブルに不足しているものを追加します。トポロジデータをスキーマに格納した後に、トポロジカタログで抜けているものを訂正するのに使います。

生成されたもののリストを返します。返されるカラムは `schema_name`, `table_name`, `feature_column` です。

Availability: 2.3.0

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 query returns no records because the feature is already registered.

```
SELECT *
FROM topology.Populate_Topology_Layer();
```

Rebuild the topology layer catalog.

```
TRUNCATE TABLE topology.layer;

SELECT *
FROM topology.Populate_Topology_Layer();

SELECT topology_id, layer_id, schema_name As sn, table_name As tn, feature_column As fc
FROM topology.layer;
```

```
schema_name | table_name | feature_column
-----+-----+-----
strk        | parcels    | topo
(1 row)

topology_id | layer_id | sn | tn    | fc
-----+-----+-----+-----+-----
          2 |         2 | strk | parcels | topo
(1 row)
```

関連情報

[AddTopoGeometryColumn](#)

9.4.8 TopologySummary

TopologySummary — トポロジ名を取り、トポロジ内のオブジェクトの型に関する概要の全体を提供します。

Synopsis

```
text TopologySummary(varchar topology_schema_name);
```

説明

トポロジ名を取り、トポロジ内のオブジェクトの型に関する概要の全体を提供します。

Availability: 2.0.0

例

```
SELECT topology.TopologySummary('city_data');
```

```
Topology city_data (329), SRID 4326, precision: 0
22 nodes, 24 edges, 10 faces, 29 topogeoms in 5 layers
Layer 1, type Polygonal (3), 9 topogeoms
Deploy: features.land_parcels.feature
Layer 2, type Puntal (1), 8 topogeoms
Deploy: features.traffic_signs.feature
Layer 3, type Lineal (2), 8 topogeoms
Deploy: features.city_streets.feature
Layer 4, type Polygonal (3), 3 topogeoms
Hierarchy level 1, child layer 1
Deploy: features.big_parcels.feature
Layer 5, type Puntal (1), 1 topogeoms
Hierarchy level 1, child layer 2
Deploy: features.big_signs.feature
```

9.4.9 ValidateTopology

ValidateTopology — トポロジの問題についての詳細を示す `validatetopology_returntype` の集合を返します。

Synopsis

```
setof validatetopology_returntype ValidateTopology(varchar toponame, geometry bbox);
```

説明

トポロジの問題の詳細説明に関する `validatetopology_returntype` オブジェクトを返します。任意に `bbox` パラメータで指定された範囲に確認を制限します。

エラーの一覧を示します。可能性のある、意味することろ、返される `id` 表現は次の通りです:

Error	id1	id2	意味
coincident nodes (訳注: 重複ノード)	最初のノードの識別子。	2 番目のノードの識別子。	二つのノードは同じジオメトリを持ちます。
edge crosses node (訳注: エッジとノードのクロス)	エッジの識別子。	ノードの識別子。	エッジが内部にノードを持っています。 ST_Relate を参照して下さい。
invalid edge (訳注: 不正なエッジ)	エッジの識別子。		エッジジオメトリが不正です。 ST_IsValid をご覧ください。
edge not simple (訳注: 単純でないエッジ)	エッジの識別子。		エッジジオメトリが自己交差しています。 ST_IsSimple を参照して下さい。
edge crosses edge (訳注: エッジとエッジのクロス)	最初のエッジの識別子。	2 番目のエッジの識別子。	二つのエッジが内部でインタセクトしています。 ST_Relate をご覧ください。
edge start node geometry mismatch (訳注: 開始ノードジオメトリの不整合)	エッジの識別子。	示された開始ノードの識別子。	エッジの開始ノードとして示されたノードのジオメトリが、エッジジオメトリの最初のポイントと合致しません。 ST_StartPoint を参照して下さい。
edge end node geometry mismatch (訳注: 終了ノードジオメトリの不整合)	エッジの識別子。	示された終了ノードの識別子。	エッジの終了ノードとして示されたノードのジオメトリがエッジジオメトリの最後のポイントと合致しません。 ST_EndPoint を参照して下さい。
face without edges (訳注: エッジのないフェイス)	取り残されたフェイスの識別子。		既存のフェイスを左右いずれかの側に (left face , right face として) 持つエッジが存在しません。
face has no rings (訳注: 環のないフェイス)	部分的に定義されたフェイスの識別子。		環を形成しない側のフェイスを示すエッジ。
face has wrong mbr (訳注: 誤ったバウンディングボックスを持ったフェイス)	不正な MBR キャッシュを持つフェイスの識別子。		フェイスの最小バウンディング四角形が、サイド上のフェイスを示すエッジのコレクションの最小バウンディングボックスと合致しません。
hole not in advertised face (訳注: 穴がフェイス内に存在しない)	リングを識別する符号付きのエッジ識別子。 GetRingEdges を参照して下さい。		エッジの環が、外環のフェイスが異なるフェイスに含まれているフェイスを示します。
not-isolated node has not-containing face (訳注: 非孤立ノードが非孤立フェイスを持っています)	はっきりしないノードの識別子。		一つ以上のエッジの境界上にあると示された一つのノードが、そのノードを含んでいるフェイスを示しています。

Error	id1	id2	意味
isolated node has containing_face (訳注: 孤立ノードが containing_face を持っています)	はっきりしないノードの識別子。		どのエッジの境界上にもないと報告されるノードが、それを含むフェイスが無いことを示しています。
isolated node has wrong containing_face (訳注: この孤立ノードを含んでいるフェイスが不正)	不正確なノードの識別子。		どのエッジの境界上にもないと報告されるノードが、そのノードを実際に含むフェイスではないフェイスに含まれていることを示しています。 GetFaceContaining-Point を参照して下さい。
invalid next_right_edge (訳注: 不正な next_right_edge)	不正確なエッジの識別子。	符号付き識別子で次の右エッジを示します。	エッジの右側を進んで遭遇する次のエッジが間違っていることを示しています。
invalid next_left_edge (訳注: 不正な next_left_edge)	不正確なエッジの識別子。	符号付き識別子で次の左エッジを示します。	エッジの左側を進んで遭遇する次のエッジが間違っていることを示しています。
mixed face labeling in ring (訳注: リング内のフェイスのラベルの矛盾)	リングを識別する符号付きのエッジ識別子。 GetRingEdges を参照して下さい。		リング内のエッジが、進む側にあるフェイスが矛盾していることを示しています。これは「サイドロケーションの矛盾」とも言われます。
non-closed ring (訳注: 閉じていないリング)	リングを識別する符号付きのエッジ識別子。 GetRingEdges を参照して下さい。		続く next_left_edge/next_right_edge属性で形成されるエッジのリングの始点と終点が異なります。
face has multiple shells (訳注: 複数の外殻を持つフェイス)	競合するフェイスの識別子。	リングを識別する符号付きのエッジ識別子。 GetRingEdges を参照して下さい。	複数のエッジのリングがその内部で同じフェイスを示しています。

Availability: 1.0.0

Enhanced: 2.0.0 では、より効果的なエッジ交差検出が可能になり、以前の版で残っていた偽陽性を解決しています。

Changed: 2.2.0 エラーの記述と矛盾しないように'edge crosses node' の id1 と id2 の値が入れ替わっています。

Changed: 3.2.0 任意パラメータ bbox を追加し、フェイスラベルとエッジリンクのチェックを追加しました。

例

```
SELECT * FROM topology.ValidateTopology('ma_topo');
```

```
error      | id1 | id2
-----+-----+-----
face without edges |    1 |
```

関連情報

[validate_topology_returntype](#)

9.4.10 ValidateTopologyRelation

ValidateTopologyRelation — 不正なトポロジ関係の行に関する情報を返します。

Synopsis

```
setof record ValidateTopologyRelation(varchar toponame);
```

説明

トポロジの関係テーブルの不正点に関する情報を提供するレコードの集合を返します。

Availability: 3.2.0

関連情報

[ValidateTopology](#)

9.4.11 ValidateTopologyPrecision

ValidateTopologyPrecision — トポロジ内の低精度の頂点を返します。

Synopsis

```
geometry ValidateTopologyPrecision(name toponame, geometry bbox, float8 gridSize);
```

説明

トポロジまたは与えられた **gridSize** に丸められた頂点のすべてを点ジオメトリで返します。任意で、確認範囲を **bbox** パラメータで指定される範囲に制限できます。

Availability: 3.6.0

例

```
SELECT ST_AsEWKT(g)
FROM topology.ValidateTopologyPrecision(
    'city_data',
    gridSize =
> 2,
    bbox =
> ST_MakeEnvelope(0, 0, 20, 20)
) g;
```

```
MULTIPOINT(9 6,9 14)
```

関連情報

[MakeTopologyPrecise](#)

9.4.12 MakeTopologyPrecise

MakeTopologyPrecise — トポロジの頂点を精度グリッドにスナップします。

Synopsis

```
void MakeTopologyPrecise(name toponame, geometry bbox, float8 gridSize);
```

説明

トポロジの全ての頂点をトポロジの精度グリッドにスナップするか、**gridSize** パラメータで指定したサイズのグリッドにスナップします。任意に **bbox** パラメータで指定した領域とインタセクとするものだけ进行处理することもできます。

最小エッジ範囲より大きい **gridSize** を使うと、エッジが空ジオメトリに縮退するのでなく例外が発生します。これにより、精密なスナップ処理の際にプリミティブを完全に削除しないようにすることで、隣接するトポロジー関係を保護します。

**Note**

スナップによってトポロジが不正になることがありえるので、処理の結果を [ValidateTopology](#) で確認することを推奨します。

Availability: 3.6.0

例

```
SELECT topology.MakeTopologyPrecise(  
    'city_data',  
    gridSize =  
> 2  
);
```

関連情報

[ValidateTopologyPrecision](#), [ValidateTopology](#)

9.4.13 FindTopology

FindTopology — 異なる方法でトポロジ行を返します。

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);
```

説明

トポロジ識別子またはトポロジ関連オブジェクトの識別しを取り、`topology.topology` レコードを返します。

Availability: 3.2.0

例

```
SELECT name(FindTopology('features.land_parcels', 'feature'));
```

```
city_data
```

関連情報

[FindLayer](#)

9.4.14 FindLayer

FindLayer — 様々な方法で `topology.layer` 行を返します。

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);
```

説明

トポロジ識別子またはトポロジ関連オブジェクトの識別子を取り、`topology.topology` レコードを返します。

Availability: 3.2.0

例

```
SELECT layer_id(FindLayer('features.land_parcels', 'feature'));
```

```
1
```


関連情報

[FindTopology](#)

9.4.15 TotalTopologySize

TotalTopologySize — 指定したトポロジが使用するディスク容量の総量を得ます。インデックスや TOAST データも含まれます。

Synopsis

```
int8 TotalTopologySize(name toponame);
```

説明

トポロジ名から、そのトポロジのすべてのテーブルのディスク容量の総量を返します。インデックスと TOAST データも含まれます。

Availability: 3.6.0

例

```
SELECT topology.TopologySummary('city_data');
```

```
Topology city_data (329), SRID 4326, precision: 0
22 nodes, 24 edges, 10 faces, 29 topogeoms in 5 layers
Layer 1, type Polygonal (3), 9 topogeoms
  Deploy: features.land_parcel.feature
Layer 2, type Puntal (1), 8 topogeoms
  Deploy: features.traffic_signs.feature
Layer 3, type Lineal (2), 8 topogeoms
  Deploy: features.city_streets.feature
Layer 4, type Polygonal (3), 3 topogeoms
  Hierarchy level 1, child layer 1
  Deploy: features.big_parcel.feature
Layer 5, type Puntal (1), 1 topogeoms
  Hierarchy level 1, child layer 2
  Deploy: features.big_signs.feature
```

9.4.16 UpgradeTopology

UpgradeTopology — 指定したトポロジーに対して、トポロジー ID とプリミティブ ID が大きな数 (int8) に対応するようにアップグレードします。

Synopsis

```
void UpgradeTopology(name toponame);
```

説明

トポロジー名を取って、トポロジー ID とプリミティブ ID が大きな数 (int8) に対応するようアップグレードします。この関数は次のアップグレードを行います: フェイス (face_id カラムを int4 から int8 に、face_id_seq を int4 から int8 に) - ノード (node_id カラムを int4 から int8 に、containing_face カラムを int4 から int8 に、node_id_seq を int4 から int8 に) - edge_data (edge_id カラムを int4 から int8 に、edge_data_edge_id_seq を int4 から int8 に、left_face カラムと right_face カラムを int4 から int8 に、start_node カラムと end_node カラムを int4 から int8 に、next_left_edge カラムと next_right_edge カラムを int4 から int8 に) - relation (topogeo_id カラムを int4 から int8 に、element_id を int4 から int8 に) - topology (useslargeids カラムを TRUE に設定)

Availability: 3.6.0

例

```
SELECT topology.UpgradeTopology('city_data');
```

9.5 トポロジ統計管理

トポロジに要素を追加すると、そのトリガとして、分割されることになる既存のエッジを探索し、ノードを追加し、新しいラインでノードを作成するエッジを更新するために多数のデータベースクエリが発生します。このため、トポロジテーブル内のデータに関する統計情報が最新の状態になっているなら、統計情報を使うと便利です。

PostGIS トポロジーの追加や編集の関数は、自動的に統計情報を更新することはありません。トポロジにおいて逐次変更しては、統計情報の更新が過剰になるためです。処理は呼び出し元の義務となっています。



Note

autovacuum で更新された統計情報は、autovacuum プロセス完了前に始まったトランザクションからは見えないので、更新した統計情報を使うには、実行時間の長いトランザクションでは ANALYZE 自体を実行する必要があります。

9.6 トポロジコンストラクタ

9.6.1 CreateTopology

CreateTopology — 新しいトポロジスキーマを生成し、topology.topology テーブルに登録します。

Synopsis

integer **CreateTopology**(name topology_schema_name, integer srid, double precision prec=0, boolean hasz=false, integer topoid=0, boolean useslargeids=false);

説明

topology_name という名前でトポロジスキーマを新たに生成し、topology.topology テーブルに登録します。トポロジ名は一意でなければなりません。トポロジテーブル (edge_data, face, node, relation) は、このスキーマ内に生成されます。トポロジの ID を返します。

`srid` はトポロジーの空間参照系の SRID です。SRID を指定していない場合のデフォルトは -1 (不明) です。許容値 `prec` は空間参照系の単位で計測されます。許容値はデフォルトでは 0.0 です。

`hasz` を指定しない場合のデフォルトは FALSE です。

`topoid` 任意パラメータ、明示的な識別子 (トポロジー ID の割当を決定しますが一意である必要があります)

`useslargeids` 任意パラメータで、デフォルトは FALSE です。TRUE の場合には、このトポロジーのトポロジー ID とプリミティブ ID は大きな数の ID (int8) に対応します。

これは SQL/MM の `ST_InitTopoGeo` に似ていますが、より高機能です。

Availability: 1.1

Enhanced: 2.0 `hasZ` 対応の追加。

Enhanced: 3.6.0 `topoid` と `usesLargeIDs` パラメーターの追加。

例

`ma_topo` という名前のトポロジスキーマを生成します。これはマサチューセッツ州メートル平面 (SRID 26986) でエッジとノードを保存することとします。空間参照系の単位がメートルなので、許容値は 0.5 メートルとなります。

```
SELECT topology.CreateTopology('ma_topo', 26986, 0.5);
```

`ri_topo` という名前の、空間参照系をフィート平面 (SRID = 3438) としたロードアイランドのトポロジの生成

```
SELECT topology.CreateTopology('ri_topo', 3438) AS topoid;
```

2

関連情報

Section 4.5, `ST_InitTopoGeo`

9.6.2 CopyTopology

`CopyTopology` — トポロジ (ノード、エッジ、フェイス、レイヤと `TopoGeometry`) の新しいスキーマに複写します。

Synopsis

integer **CopyTopology**(varchar existing_topology_name, varchar new_name);

説明

`new_name` という名前のトポロジを作成しますが、SRID と精度は `existing_topology_name` から複写します。`existing_topology_name` のノード、エッジ、フェイスと、レイヤとそれに関連付けられた `TopoGeometry` は、新しいトポロジに複写されます。



Note

`topology.layer` テーブルの新しい行には `schema_name`, `table_name`, `feature_column` の値の合成値が含まれます。`TopoGeometry` オブジェクトは定義だけが存在するのであって、ユーザ定義テーブルで利用できる状態にないからです。

Availability: 2.0.0

例

ma_topo という名前のトポロジのバックアップを作成します。

```
SELECT topology.CopyTopology('ma_topo', 'ma_topo_backup');
```

関連情報

Section 4.5, [CreateTopology](#), [RenameTopology](#)

9.6.3 ST_InitTopoGeo

ST_InitTopoGeo — 新しいトポロジスキーマを生成し、topology.topology テーブルに登録します。

Synopsis

text **ST_InitTopoGeo**(varchar topology_schema_name);

説明

[CreateTopology](#) の SQL/MM 相当の関数です。空間参照系と許容値の任意引数がありません。トポロジ ID でなくトポロジ生成の文字列による説明が返されます。

Availability: 1.1



このメソッドは SQL/MM 仕様の実装です。SQL-MM 3 トポロジ-ジオメトリおよびトポロジ-ネットワーク 3: ルーチン詳細: X.3.17

例

```
SELECT topology.ST_InitTopoGeo('topo_schema_to_create') AS topocreation;
```

```
Topology-Geometry 'topo_schema_to_create' (id:7) created.
```

関連情報

[CreateTopology](#)

9.6.4 ST_CreateTopoGeo

ST_CreateTopoGeo — 空のトポロジにジオメトリのコレクションを追加し、成否を示すメッセージを返します。

Synopsis

text **ST_CreateTopoGeo**(varchar atopoology, geometry acollection);

説明

空のトポロジにジオメトリのコレクションを追加し、成否を示すメッセージを返します。

空トポロジの追加に使えます。

Availability: 2.0

 このメソッドは SQL/MM 仕様の実装です。SQL-MM 3 トポロジ-ジオメトリおよびトポロジ-ネットワーク 3: ルーチン詳細: X.3.18

例

Populate the 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)
  );
```

Topology ri_topo populated

Create the tables and topogeometries.

```
CREATE TABLE ri.roads(gid serial PRIMARY KEY, road_name text);

SELECT topology.AddTopoGeometryColumn('ri_topo', 'ri', 'roads', 'topo', 'LINE');
```

関連情報

[TopoGeo_LoadGeometry](#), [AddTopoGeometryColumn](#), [CreateTopology](#), [DropTopology](#)

9.6.5 TopoGeo_AddPoint

TopoGeo_AddPoint — 許容差を使って既存のトポロジにポイントを追加し、可能ならエッジを分割します。

Synopsis

```
bigint TopoGeo_AddPoint(varchar atopology, geometry apoint, float8 tolerance);
```

説明

既存のトポロジにポイントを追加し、その識別番号を返します。指定されたポイントは、許容差の範囲内で既存のノードまたはエッジにスナップします。既存のエッジはスナップされたポイントで分割されることがあります。

Availability: 2.0.0

関連情報

[TopoGeo_AddLineString](#), [TopoGeo_AddPolygon](#), [TopoGeo_LoadGeometry](#), [AddNode](#), [CreateTopology](#)

9.6.6 TopoGeo_AddLineString

`TopoGeo_AddLineString` — 許容差を使って既存のトポロジにラインストリングを追加し、可能ならエッジ/フェイスを分割します。

Synopsis

SETOF bigint **TopoGeo_AddLineString**(varchar atopology, geometry aline, float8 tolerance, int max_edges)

説明

ラインストリングを既存のトポロジーに追加して、整列させた符号付エッジ識別番号を返します (負数は入力ラインストリングと逆方向であることを示します)。

与えられたラインは与えられた許容値内にある既存のノードまたはエッジにスナップします。既存のエッジとフェイスはラインで分割されることがあります。新規エッジ追加に加えて、新しいノードとフェイスが追加されることがあります。

返されるエッジ識別番号は、入力ラインを可能な限り忠実になるような表現を完全に行うために、既存のエッジまたは新規生成されるエッジのいずれかの番号になります。

新規生成されるエッジ (及んでいなかった範囲の空間に及ぶか、既存エッジの分割から得られた結果) の数は `max_edges` パラメータで制限されます。



Note

この関数を介してロードされたトポロジの統計情報の更新は呼び出し元次第です。 [maintaining statistics during topology editing and population](#) をご覧ください。

Availability: 2.0.0

Enhanced: 3.2.0 符号付き識別子を返す機能が追加されました。

Enhanced: 3.7.0 トポロジー内で生成されるエッジ数の制限機能の追加。

関連情報

[TopoGeo_AddPoint](#), [TopoGeo_AddPolygon](#), [TopoGeo_LoadGeometry](#), [AddEdge](#), [CreateTopology](#)

9.6.7 TopoGeo_AddPolygon

`TopoGeo_AddPolygon` — 許容差を使って既存のトポロジにラインストリングを追加し、可能ならエッジ/フェイスを分割します。エッジ識別番号を返します。

Synopsis

SETOF bigint **TopoGeo_AddPolygon**(varchar atopology, geometry apoly, float8 tolerance);

説明

既存のトポロジにポリゴンを追加し、これを構成するフェイスの識別番号の集合を返します。指定されたポリゴンの境界線は、許容差の範囲内で既存のノードまたはエッジにスナップします。既存のエッジとフェイスはポリゴンの境界線で分割されることがあります。



Note

この関数を介してロードされたトポロジの統計情報の更新は呼び出し元次第です。 **maintaining statistics during topology editing and population** をご覧下さい。

Availability: 2.0.0

関連情報

[TopoGeo_AddPoint](#), [TopoGeo_AddLineString](#), [TopoGeo_LoadGeometry](#), [AddFace](#), [CreateTopology](#)

9.6.8 TopoGeo_LoadGeometry

TopoGeo_LoadGeometry — ジオメトリを既存のトポロジにロードし、必要に応じてスナップと分割を実行します。

Synopsis

void **TopoGeo_LoadGeometry**(varchar atopology, geometry ageom, float8 tolerance);

説明

ジオメトリを既存のトポロジにロードします。与えられたジオメトリは、与えられた許容値の範囲内で、既存のノードまたはエッジにスナップします。既存のエッジとフェイスはロードの結果分割される場合があります。



Note

この関数を介してロードされたトポロジの統計情報の更新は呼び出し元次第です。 **maintaining statistics during topology editing and population** をご覧下さい。

Availability: 3.5.0

関連情報

[TopoGeo_AddPoint](#), [TopoGeo_AddLineString](#), [TopoGeo_AddPolygon](#), [CreateTopology](#)

9.7 トポロジエディタ

9.7.1 ST_AddIsoNode

ST_AddIsoNode — フェイスに孤立ノードを追加し、新しいノードの識別番号を返します。フェイスが **NULL** の場合でもノードは生成されます。

Synopsis

```
bigint ST_AddIsoNode(varchar atopology, bigint aface, geometry apoint);
```

説明

atopology で指定されたトポロジの **aface** の識別番号で示された既存のフェイスに対して、**apoint** で示された位置に孤立ノードを追加します。

ポイントジオメトリの空間参照系 (SRID) がトポロジと同じでない場合、**apoint** がポイントジオメトリでない場合、ポイントが **NULL** である場合、または、ポイントが既存のエッジ (境界も含む) とインタセクトする場合には、例外が投げられます。また、ポイントが既にノードとして存在する場合、例外が投げられます。

aface が **NULL** でなく、かつ **apoint** がフェイス内に無い場合には、例外が投げられます。

Availability: 1.1

 このメソッドは SQL/MM 仕様の実装です。SQL-MM 3 トポロジ-ジオメトリおよびトポロジ-ネットワーク 3: ルーチン詳細: X 3.1

例

関連情報

[AddNode](#), [CreateTopology](#), [DropTopology](#), [ST_Intersects](#)

9.7.2 ST_AddIsoEdge

ST_AddIsoEdge — **anode** と **anothernode** で指定される二つの既存孤立ノードを接続するトポロジに、ジオメトリ **alinestring** で定義される孤立エッジを追加し、新しいエッジの識別番号を返します。

Synopsis

```
bigint ST_AddIsoEdge(varchar atopology, bigint anode, bigint anothernode, geometry alinestring);
```

説明

anode と **anothernode** で指定される二つの既存孤立ノードを接続するトポロジに、ジオメトリ **alinestring** で定義される孤立エッジを追加し、新しいエッジの識別番号を返します。

alinestring ジオメトリとトポロジとで空間参照系 (SRID) が異なる場合、引数が **NULL** である場合、ノードが一つ以上のフェイスに含まれている場合には、二つのノードが既存エッジの始端または終端である場合には、例外が投げられます。

`alinesring` が `anode` と `anothernode` に属するフェイス内に無い場合には、例外が投げられます。
`anode` と `anothernode` が、`alinesring` の始端と終端でない場合には、例外が投げられます。

Availability: 1.1

 このメソッドは SQL/MM 仕様の実装です。SQL-MM 3 トポロジ-ジオメトリおよびトポロジ-ネットワーク 3: ルーチン詳細: X.3.4

例

関連情報

[ST_AddIsoNode](#), [ST_IsSimple](#), [ST_Within](#)

9.7.3 ST_AddEdgeNewFaces

`ST_AddEdgeNewFaces` — 新しいエッジを追加します。新しいエッジがフェイスを分割する場合には、もとのフェイスを削除して、分割した二つのフェイスに置き換えます。

Synopsis

`bigint ST_AddEdgeNewFaces(varchar atopology, bigint anode, bigint anothernode, geometry acurve);`

説明

新しいエッジを追加します。新しいエッジがフェイスを分割する場合には、もとのフェイスを削除して、分割した二つのフェイスに置き換えます。新しいエッジの識別番号を返します。

この関数によって、既存の接続されているエッジとリレーションシップが更新されます。

引数が `NULL` の場合、与えられたノードが不明な場合 (トポロジスキーマの `node` テーブル内に既に存在していなければなりません)、`acurve` が `LINestring` でない場合、`anode` と `anothernode` が `acurve` の始端、終端でない場合には、例外が投げられます。

`acurve` ジオメトリの空間参照系 (SRID) がトポロジと同じでない場合、例外が投げられます。

Availability: 2.0

 このメソッドは SQL/MM 仕様の実装です。SQL-MM 3 トポロジ-ジオメトリおよびトポロジ-ネットワーク 3: ルーチン詳細: X.3.12

例

関連情報

[ST_RemEdgeNewFace](#)

[ST_AddEdgeModFace](#)

9.7.4 ST_AddEdgeModFace

`ST_AddEdgeModFace` — 新しいエッジを追加します。新しいエッジがフェイスを分割する場合には、もとのフェイスを編集し、一つのフェイスを追加します。

Synopsis

bigint **ST_AddEdgeModFace**(varchar atopolology, bigint anode, bigint anothernode, geometry acurve);

説明

新しいエッジを追加します。新しいエッジがフェイスを分割する場合には、もとのフェイスを編集し、一つのフェイスを追加します。



Note

可能なら、新しいフェイスは新しいエッジの左側に作られます。左側のフェイスがユニバースフェイス (境界がない) でなければならない場合には、可能ではありません。

新しく追加されたエッジの識別番号を返します。

この関数によって、既存の接続されているエッジとリレーションシップが更新されます。

引数が NULL の場合、与えられたノードが不明な場合 (トポロジスキーマの `node` テーブル内に既に存在していなければなりません)、`acurve` が `LINestring` でない場合、`anode` と `anothernode` が `acurve` の始端、終端でない場合には、例外が投げられます。

`acurve` ジオメトリの空間参照系 (SRID) がトポロジと同じでない場合、例外が投げられます。

Availability: 2.0



このメソッドは SQL/MM 仕様の実装です。SQL-MM 3 トポロジ-ジオメトリおよびトポロジ-ネットワーク 3: ルーチン詳細: X.3.13

例

関連情報

[ST_RemEdgeModFace](#)

[ST_AddEdgeNewFaces](#)

9.7.5 ST_RemEdgeNewFace

`ST_RemEdgeNewFace` — エッジを削除し、消去対象エッジでフェイスが二つに分割されているなら元の二つのフェイスを削除し、一つの新しいフェイスに置き換えます。

Synopsis

bigint **ST_RemEdgeNewFace**(varchar atopolology, bigint anedge);

説明

エッジを削除し、消去対象エッジでフェイスが二つに分割されているなら元の二つのフェイスを削除し、一つの新しいフェイスに置き換えます。

新しく作成されたフェイスの識別番号を返します。新しいフェイスが生成されない場合には NULL を返します。削除対象エッジがダングルである (訳注: 「ぶらさがる」状態、すなわち一方の端が孤立ノードでなく、かつもう一

方が孤立ノード) 場合か、孤立している場合か、ユニバースフェイスとの境界になっている (おそらく反対側のフェイスにユニバースが侵入します) 場合には、フェイスは生成されません。

この関数によって、既存の接続されているエッジとリレーションシップが更新されます。

既存の **TopoGeometry** の定義に入り込んでいるエッジは削除を拒絶されます。**TopoGeometry** が二つのフェイスのうちひとつだけで定義されている (かつ他方は定義に使われていない) 場合、二つのフェイスの修復は拒絶されます。

引数が **NULL** である場合、与えられたエッジが不明である場合 (トポロジスキーマの **edge** テーブル内に既に存在していなければなりません)、トポロジ名が不正である場合、例外が投げられます。

Availability: 2.0



このメソッドは SQL/MM 仕様の実装です。SQL-MM 3 トポロジ-ジオメトリおよびトポロジ-ネットワーク 3: ルーチン詳細: X.3.14

例

関連情報

[ST_RemEdgeModFace](#)

[ST_AddEdgeNewFaces](#)

9.7.6 ST_RemEdgeModFace

ST_RemEdgeModFace — エッジを削除します。削除されるエッジが二つのフェイスを分割していたなら、一方のフェイスを削除し、もう一方のフェイスを両方の空間を覆うように編集します。

Synopsis

```
bigint ST_RemEdgeModFace(varchar atopology, bigint anedge);
```

説明

エッジを削除します。削除されるエッジが二つのフェイスを分割していたなら、一方のフェイスを削除し、もう一方のフェイスを両方の空間を覆うように編集します。右側のフェイスを優先的に保持します。これは [ST_AddEdgeModFace](#) と一致します。残ったフェイスの ID を返します。

この関数によって、既存の接続されているエッジとリレーションシップが更新されます。

既存の **TopoGeometry** の定義に入り込んでいるエッジは削除を拒絶されます。**TopoGeometry** が二つのフェイスのうちひとつだけで定義されている (かつ他方は定義に使われていない) 場合、二つのフェイスの修復は拒絶されます。

引数が **NULL** である場合、与えられたエッジが不明である場合 (トポロジスキーマの **edge** テーブル内に既に存在していなければなりません)、トポロジ名が不正である場合、例外が投げられます。

Availability: 2.0



このメソッドは SQL/MM 仕様の実装です。SQL-MM 3 トポロジ-ジオメトリおよびトポロジ-ネットワーク 3: ルーチン詳細: X.3.15

例

関連情報

[ST_AddEdgeModFace](#)

[ST_RemEdgeNewFace](#)

9.7.7 ST_ChangeEdgeGeom

ST_ChangeEdgeGeom — トポロジ構造に影響を与えることなくエッジの形状を変更します。

Synopsis

text **ST_ChangeEdgeGeom**(varchar atopology, bigint anedge, geometry acurve);

説明

トポロジ構造に影響を与えることなくエッジの形状を変更します。

全ての引数が NULL であるか、与えられたエッジが **topology** スキーマの **edge** テーブルにないか、**acurve** が **LINestring** でないか、変更で基になるトポロジが変わるか、のいずれかの場合には、エラーが投げられます。

acurve ジオメトリの空間参照系 (SRID) がトポロジと同じでない場合、例外が投げられます。

新しい **acurve** が単純でない場合には、エラーが投げられます。

古い位置から新しい位置へのエッジ移動で障害物にあたった場合はエラーが投げられます。

Availability: 1.1.0

Enhanced: 2.0.0 トポロジ整合性の強制を追加しました。

 このメソッドは SQL/MM 仕様の実装です。SQL-MM 3 トポロジ-ジオメトリおよびトポロジ-ネットワーク 3: ルーチン詳細: X.3.6

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

関連情報

[ST_AddEdgeModFace](#)

[ST_RemEdgeModFace](#)

[ST_ModEdgeSplit](#)

9.7.8 ST_ModEdgeSplit

ST_ModEdgeSplit — 既存のエッジに沿って新しいノードを生成してエッジを分割します。もとのエッジは変更され、新しいエッジが一つ追加されます。

Synopsis

bigint ST_ModEdgeSplit(varchar atopology, bigint anedge, geometry apoint);

説明

既存のエッジに沿って新しいノードを生成してエッジを分割します。もとのエッジは変更され、新しいエッジが一つ追加されます。この関数によって、既存の接続されているエッジとリレーションシップが更新されます。新しく追加されたノードの識別番号が返ります。

Availability: 1.1

Changed: 2.0 - 以前の版では名前を間違えて ST_ModEdgesSplit になっていました。

 このメソッドは SQL/MM 仕様の実装です。SQL-MM 3 トポロジ-ジオメトリおよびトポロジ-ネットワーク 3: ルーチン詳細: X.3.9

例

Add an edge to split.

```
SELECT topology.AddEdge('ma_topo', ST_GeomFromText('LINESTRING(227592 893910, 227600 893910)', 26986)) AS edgeid;
```

3

Split the edge with a new node.

```
SELECT topology.ST_ModEdgeSplit('ma_topo', 3, ST_SetSRID(ST_Point(227594, 893910), 26986)) AS node_id;
```

7

関連情報

[ST_NewEdgesSplit](#), [ST_ModEdgeHeal](#), [ST_NewEdgeHeal](#), [AddEdge](#)

9.7.9 ST_ModEdgeHeal

ST_ModEdgeHeal — 二つのエッジについて、接続しているノードを削除して修復します。1 番目のエッジを編集して、2 番目のエッジを削除します。削除されたノードの識別番号を返します。

Synopsis

bigint ST_ModEdgeHeal(varchar atopology, bigint anedge, bigint anotheredge);

説明

二つのエッジについて、接続しているノードを削除して修復します。1 番目のエッジを編集して、2 番目のエッジを削除します。削除されたノードの識別番号を返します。この関数によって、既存の接続されているエッジとリレーションシップが更新されます。

Availability: 2.0



このメソッドは SQL/MM 仕様の実装です。SQL-MM 3 トポロジ-ジオメトリおよびトポロジ-ネットワーク 3: ルーチン詳細: X.3.9

関連情報

[ST_ModEdgeSplit](#) [ST_NewEdgesSplit](#)

9.7.10 ST_NewEdgeHeal

ST_NewEdgeHeal — 二つのエッジについて、接続しているノードを削除して修復します。両方のエッジを削除し、1 番目のエッジと同じ方向のエッジに置き換えます。

Synopsis

```
bigint ST_NewEdgeHeal(varchar atopology, bigint anedge, bigint anotheredge);
```

説明

二つのエッジについて、接続しているノードを削除して修復します。両方のエッジを削除し、1 番目のエッジと同じ方向のエッジに置き換えます。修復されたエッジに置き換えられた新しいエッジの識別番号を返します。この関数によって、既存の接続されているエッジとリレーションシップが更新されます。

Availability: 2.0



このメソッドは SQL/MM 仕様の実装です。SQL-MM 3 トポロジ-ジオメトリおよびトポロジ-ネットワーク 3: ルーチン詳細: X.3.9

関連情報

[ST_ModEdgeHeal](#) [ST_ModEdgeSplit](#) [ST_NewEdgesSplit](#)

9.7.11 ST_MoveIsoNode

ST_MoveIsoNode — トポロジ内の孤立ノードを別の位置に移動させます。新しい **apoint** ジオメトリがノードとして存在しているなら、エラーが投げられます。移動に関する説明を返します。

Synopsis

```
text ST_MoveIsoNode(varchar atopology, bigint anode, geometry apoint);
```

説明

トポロジ内の孤立ノードを別の位置に移動させます。新しい **apoint** ジオメトリがノードとして存在しているなら、エラーが投げられます。

引数が **NULL** の場合、**apoint** がポイントでない場合、既存のノードが孤立していない (既存のエッジの始端か終端になっている) 場合、新しいノード位置が既存のエッジとインタセクトする場合 (終端であっても)、新しい位置が異なるフェイス内にある場合 (3.2.0 以降) のいずれかの場合には、例外が投げられます。

ポイントジオメトリの空間参照系 (SRID) がトポロジと異なる場合には、例外が投げられます。

Availability: 2.0.0

Enhanced: 3.2.0 ノードが異なるフェイスに移動しないことを保証

 このメソッドは SQL/MM 仕様の実装です。SQL-MM 3 トポロジ-ジオメトリおよびトポロジ-ネットワーク 3: ルーチン詳細: X.3.2

例

Add an isolated node with no face.

```
SELECT topology.ST_AddIsoNode('ma_topo', NULL, ST_GeomFromText('POINT(227579 893916)', 26986)) AS nodeid;
```

7

Move the new node.

```
SELECT topology.ST_MoveIsoNode('ma_topo', 7, ST_GeomFromText('POINT(227579.5 893916.5)', 26986)) AS descrip;
```

Isolated Node 7 moved to location 227579.5,893916.5

関連情報

[ST_AddIsoNode](#)

9.7.12 ST_NewEdgesSplit

ST_NewEdgesSplit — 新しいノードを既存のエッジに沿って作成して、エッジを分割します。もとのエッジは削除され、二つのエッジに置き換えられます。二つの新しいエッジに接続する新しいノードの識別番号を返します。

Synopsis

bigint **ST_NewEdgesSplit**(varchar atopology, bigint anedge, geometry apoint);

説明

anedge で指定される既存のエッジに沿った **apoint** の位置に新しいノードを作成して、エッジを分割します。もとのエッジは削除され、二つのエッジに置き換えられます。二つの新しいエッジに接続する新しいノードの識別番号を返します。この関数によって、既存の接続されているエッジとリレーションシップが更新されます。

ポイントジオメトリの空間参照系 (SRID) がトポロジと異なる場合、**apoint** がポイントジオメトリでない場合、ポイントが NULL の場合、ポイントが既にノードとして存在する場合、エッジが既存のエッジと一致しない場合、ポイントがエッジ内でない場合、例外が投げられます。

Availability: 1.1

 このメソッドは SQL/MM 仕様の実装です。SQL-MM 3 トポロジ-ジオメトリおよびトポロジ-ネットワーク 3: ルーチン詳細: X.3.8

例

Add an edge to split.

```
SELECT topology.AddEdge('ma_topo', ST_GeomFromText('LINESTRING(227575 893917, 227592 893900)', 26986)) AS edgeid;
```

2

Split the new edge.

```
SELECT topology.ST_NewEdgesSplit('ma_topo', 2, ST_GeomFromText('POINT(227578.5 893913.5)', 26986)) AS newnodeid;
```

6

関連情報

[ST_ModEdgeSplit](#) [ST_ModEdgeHeal](#) [ST_NewEdgeHeal](#) [AddEdge](#)

9.7.13 ST_RemoveIsoNode

ST_RemoveIsoNode — 孤立ノードを削除し、実行結果が返されます。ノードが孤立していない (エッジの始端または終端である) 場合には、例外が投げられます。

Synopsis

text **ST_RemoveIsoNode**(varchar atopology, bigint anode);

説明

孤立ノードを削除し、実行結果が返されます。ノードが孤立していない (エッジの始端または終端である) 場合には、例外が投げられます。

Availability: 1.1

 このメソッドは SQL/MM 仕様の実装です。SQL-MM 3 トポロジ-ジオメトリおよびトポロジ-ネットワーク 3: ルーチン詳細: X.3.3

例

Remove an isolated node with no face.

```
SELECT topology.ST_RemoveIsoNode('ma_topo', 7) AS result;
```

```
Isolated node 7 removed
```

関連情報

[ST_AddIsoNode](#)

9.7.14 ST_RemoveIsoEdge

ST_RemoveIsoEdge — 孤立エッジを削除し、実行結果の記述を返します。エッジが孤立していない場合には、例外が投げられます。

Synopsis

```
text ST_RemoveIsoEdge(varchar atopology, bigint anedge);
```

説明

孤立エッジを削除し、実行結果の記述を返します。エッジが孤立していない場合には、例外が投げられます。

Availability: 1.1



このメソッドは SQL/MM 仕様の実装です。SQL-MM 3 トポロジ-ジオメトリおよびトポロジ-ネットワーク 3: ルーチン詳細: X.3.3

例

Remove an isolated edge with no face.

```
SELECT topology.ST_RemoveIsoEdge('ma_topo', 7) AS result;
```

```
Isolated edge 7 removed
```

関連情報

[ST_AddIsoNode](#)

9.8 トポロジアクセサ

9.8.1 GetEdgeByPoint

GetEdgeByPoint — 与えられたポイントにインタセクトするエッジの識別番号を探索します。

Synopsis

bigint **GetEdgeByPoint**(varchar atopology, geometry apoint, float8 tol1);

説明

与えられたポイントにインタセクトするエッジの識別番号を探索します。

この関数は、トポロジ、ポイント、許容差を引数にして、整数 (エッジの識別番号) を返します。許容差が 0 の場合、ポイントはエッジとインタセクトしていなければなりません。

apoint がエッジとインタセクトしない場合には、0 を返します。

許容差 > 0 として、ポイント付近のエッジが一つより多い場合には、例外が投げられます。



Note

許容差が 0 の場合には、ST_Intersects を使い、それ以外では、ST_DWithin を使います。

GEOS モジュールで実現しています。

Availability: 2.0.0

例

本例では **AddEdge** で作ったエッジを使います。

```
SELECT topology.GetEdgeByPoint('ma_topo', geom, 1) As with1mtol, topology.GetEdgeByPoint('ma_topo', geom, 0) As withnotol
FROM ST_GeomFromEWKT('SRID=26986;POINT(227622.6 893843)') As geom;
```

with1mtol	withnotol
2	0

This query errors because two or more edges are found.

```
SELECT topology.GetEdgeByPoint('ma_topo', geom, 1) As nearnode
FROM ST_GeomFromEWKT('SRID=26986;POINT(227591.9 893900.4)') As geom;
```

ERROR: Two or more edges found

関連情報

AddEdge, **GetNodeByPoint**, **GetFaceByPoint**

9.8.2 GetFaceByPoint

GetFaceByPoint — 与えたポイントとインタセクトするフェイスを見つけます。

Synopsis

bigint **GetFaceByPoint**(varchar atopology, geometry apoint, float8 tol1);

説明

ポイントと与えた許容値で参照されるフェイスを見つけます。

この関数は、ポイントを中心に持ち半径が許容値となる円とインタセクトするフェイスを効率的に探します。

与えられたクエリ位置とインタセクトするフェイスが無い場合には、0 を返します (ユニバーサルフェイス)。

1 以上のフェイスがクエリ位置とインタセクトする場合には例外が投げられます。

Availability: 2.0.0

Enhanced: 3.2.0 より効率的な実装とより明確な短縮、不正なトポロジでの動作停止。

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

This query errors because two or more faces are found.

```
SELECT topology.GetFaceByPoint('ma_topo', geom, 1) As nearnode
FROM ST_GeomFromEWKT('POINT(227591.9 893900.4)') As geom;
```

ERROR: Two or more faces found

関連情報

[GetFaceContainingPoint](#), [AddFace](#), [GetNodeByPoint](#), [GetEdgeByPoint](#)

9.8.3 GetFaceContainingPoint

GetFaceContainingPoint — ポイントを含むフェイスを見つけます。

Synopsis

bigint **GetFaceContainingPoint**(text atopology, geometry apoint);

説明

ポイントを含むフェイスの ID を返します。

ポイントがフェイスの境界に落ちた場合には例外が投げられます。



Note

この関数は妥当なトポロジが前提であり、エッジリンクとフェイスラベルを使用して動作します。

Availability: 3.2.0

関連情報

[ST_GetFaceGeometry](#)

9.8.4 GetNodeByPoint

GetNodeByPoint — ポイント位置にあるノードの識別番号を探索します。

Synopsis

bigint **GetNodeByPoint**(varchar atopology, geometry apoint, float8 tol1);

説明

ポイント位置にあるノードの識別番号を探索します。

トポロジ、POINT、許容値が与えられ、整数 (ノード識別番号) が返ります。tolerance = 0 の場合には、確実にインタセクトするノードを取得し、そうでない場合には、指定した間隔の中からノードを取得します。

apoint がノードとインタセクトしない場合には、0 を返します。

許容差 > 0 として、ポイント付近のノードが一つより多い場合には、例外が投げられます。



Note

許容差が 0 の場合には、ST_Intersects を使い、それ以外では、ST_DWithin を使います。

GEOS モジュールで実現しています。

Availability: 2.0.0

例

本例では[AddEdge](#)で作ったエッジを使います。

```
SELECT topology.GetNodeByPoint('ma_topo', geom, 1) As nearnode
FROM ST_GeomFromEWKT('SRID=26986;POINT(227591.9 893900.4)') As geom;
```

2

This query errors because the tolerance matches two or more nodes.

```
SELECT topology.GetNodeByPoint('ma_topo', geom, 1000) As too_much_tolerance
FROM ST_GeomFromEWKT('SRID=26986;POINT(227591.9 893900.4)') As geom;
```

ERROR: Two or more nodes found

関連情報

[AddEdge](#), [GetEdgeByPoint](#), [GetFaceByPoint](#)

9.8.5 GetTopologyID

GetTopologyID — トポロジ名から topology.topology テーブル内にあるトポロジの識別番号を返します。

Synopsis

```
integer GetTopologyID(varchar toponame);
```

説明

トポロジ名から topology.topology テーブル内にあるトポロジの識別番号を返します。

Availability: 1.1

例

```
SELECT topology.GetTopologyID('ma_topo') AS topo_id;
```

```
1
```

関連情報

[CreateTopology](#), [DropTopology](#), [GetTopologyName](#), [GetTopologySRID](#)

9.8.6 GetTopologySRID

GetTopologySRID — トポロジ名から topology.topology テーブル内にあるトポロジの SRID を返します。

Synopsis

```
integer GetTopologyID(varchar toponame);
```

説明

与えられたトポロジ名から topology.topology テーブル内のトポロジの空間参照系識別番号を返します。

Availability: 2.0.0

例

```
SELECT topology.GetTopologySRID('ma_topo') AS srid;
```

```
4326
```

関連情報

[CreateTopology](#), [DropTopology](#), [GetTopologyName](#), [GetTopologyID](#)

9.8.7 GetTopologyName

GetTopologyName — 識別番号からトポロジ (スキーマ) の名前を返します。

Synopsis

```
varchar GetTopologyName(integer topology_id);
```

説明

識別番号からトポロジ (スキーマ) の名前を返します。

Availability: 1.1

例

```
SELECT topology.GetTopologyName(1) AS topo_name;
```

```
ma_topo
```

関連情報

[CreateTopology](#), [DropTopology](#), [GetTopologyID](#), [GetTopologySRID](#)

9.8.8 ST_GetFaceEdges

ST_GetFaceEdges — 順序番号を含む、**aface** の境界となる、整列したエッジの集合を返します。

Synopsis

```
getfaceedges_returntype ST_GetFaceEdges(varchar atopology, bigint aface);
```

説明

順序番号を含む、**aface** の境界となる、整列したエッジの集合を返します。それぞれの出力は、順序番号とエッジ識別番号からなります。順序番号は 1 から始まります。

各リングエッジの列挙は、識別番号が最も小さいものから始まります。エッジの順序は左手の法則に従います (境界フェイスは各有向辺の左側にあるようにします)。

Availability: 2.0



このメソッドは SQL/MM 仕様の実装です。SQL-MM 3 トポロジ-ジオメトリおよびトポロジ-ネットワーク 3: ルーチン詳細: X.3.5

例

This query returns the edges that bound face 1.

```
SELECT (topology.ST_GetFaceEdges('tt', 1)).*;
```

sequence	edge
1	-4
2	5
3	7
4	-6
5	1
6	2
7	3

This query returns the sequence, edge identifier, and geometry of the edges that bound face 1. Use [ST_GetFaceGeometry](#) if only the geometry and sequence are needed.

```
SELECT t.seq, t.edge, geom
FROM topology.ST_GetFaceEdges('tt', 1) As t(seq, edge)
      INNER JOIN tt.edge AS e ON abs(t.edge) = e.edge_id;
```

関連情報

[GetRingEdges](#), [AddFace](#), [ST_GetFaceGeometry](#)

9.8.9 ST_GetFaceGeometry

ST_GetFaceGeometry — 指定されたトポロジの中の、フェイス識別番号で指定されたポリゴンを返します。

Synopsis

geometry **ST_GetFaceGeometry**(varchar atopology, bigint aface);

説明

指定されたトポロジの中の、フェイス識別番号で指定されたポリゴンを返します。フェイスを作るエッジからポリゴンを構築します。

Availability: 1.1

 このメソッドは SQL/MM 仕様の実装です。SQL-MM 3 トポロジ-ジオメトリおよびトポロジ-ネットワーク 3: ルーチン詳細: X.3.16

例

This query returns the WKT of the polygon added with [AddFace](#).

```
SELECT ST_AsText(topology.ST_GetFaceGeometry('ma_topo', 1)) AS facegeomwkt;
```

```
POLYGON((234776.9 899563.7,234896.5 899456.7,234914 899436.4,234946.6 899356.9,
234872.5 899328.7,234891 899285.4,234992.5 899145,234890.6 899069,
234755.2 899255.4,234612.7 899379.4,234776.9 899563.7))
```

関連情報

[AddFace](#)

9.8.10 GetRingEdges

GetRingEdges — 与えられた側を歩いて得られた、正負符号付きエッジ識別番号の集合を、順序通りに返します。

Synopsis

```
getfaceedges_returntype GetRingEdges(varchar atopology, bigint aring, integer max_edges=null);
```

説明

与えられた側を歩いて得られた、正負符号付きエッジ識別番号の集合を、順序通りに返します。出力は順序番号と正負符号付きエッジ識別番号からなります。順序番号は 1 始まりです。

正のエッジ識別番号を渡すと、対応するエッジの左側を歩き、エッジを順方向に進みます。負のエッジ識別番号を渡すと、右側を歩き、エッジを逆方向に進みます。

max_edges が NULL でな場合には、返されるレコードを超えることはありません。これは、不正なトポロジを扱うときの安全確保のためのパラメータであることを意味します。



Note

この関数はメタデータとリンクするエッジ環を使います。

Availability: 2.0.0

関連情報

[ST_GetFaceEdges](#), [GetNodeEdges](#)

9.8.11 GetNodeEdges

GetNodeEdges — 与えられたノードに付随するエッジの集合を整理して返します。

Synopsis

```
getfaceedges_returntype GetNodeEdges(varchar atopology, bigint anode);
```

説明

与えられたノードに付随するエッジの集合を整理して返します。出力は、連続する、正負符号を持つエッジ識別番号からなります。順序番号は 1 から始まります。正のエッジは与えられたノードから始まるものです。負のエッジは与えられたノードで終わるものです。閉じたエッジは 2 回現れます (正と負になります)。並び順は北側から時計回りになります。

**Note**

この関数は、並び順を計算していて、メタデータからのデータを使わないので、連結しているエッジ環を構築するのに使えます。

Availability: 2.0

関連情報

[getfaceedges_returntype](#), [GetRingEdges](#), [ST_Azimuth](#)

9.8.12 FindVertexSegmentPairsBelowDistance

FindVertexSegmentPairsBelowDistance — 許容距離内にあるトポロジーの頂点/辺の組を検索します。

Synopsis

```
setof record FindVertexSegmentPairsBelowDistance(varchar atopology, float8 tolerance);
```

説明

個々のタプルが頂点/辺の組合せで、頂点は辺の終端でなく、かつ頂点と辺の間の距離が指定した許容値より小さくなるリレーションを返します。

Availability: 3.7

関連情報

[ValidateTopologyPrecision](#)

9.9 トポロジ処理

9.9.1 Polygonize

Polygonize — トポロジエッジで定義される全てのフェイスを探索し、追加します。

Synopsis

```
text Polygonize(varchar toponame);
```

説明

トポロジエッジプリミティブで構築することができる全てのフェイスを登録します。
対象トポロジは自己インタセクトするエッジが無いと仮定しています。



Note

既に登録されているフェイスは認識されているので、同じトポロジに対して `Polygonize` を複数回呼んでも問題ありません。



Note

この関数はエッジテーブルの `next_left_edge` と `next_right_edge` フィールドの読み書きを行います。

Availability: 2.0.0

関連情報

[AddFace](#), [ST_Polygonize](#)

9.9.2 AddNode

AddNode — 指定したトポロジスキーマのノードテーブルにポイントノードを追加し、新しいノードの識別番号を返します。指定したポイントに既にノードがある場合は既存のノード識別番号を返します。

Synopsis

```
bigint AddNode(varchar toponame, geometry apoint, boolean allowEdgeSplitting=false, boolean computeContainingFace=false);
```

説明

指定したトポロジスキーマのノードテーブルにポイントノードを追加します。[AddEdge](#)関数は、呼ばれると自動的にエッジの始端と終端のポイントを追加するので、明示的にエッジのノードを追加する必要はありません。
ノードとクロスするエッジが発見された場合は、例外が発生するか、エッジが分割されます。`allowEdgeSplitting` パラメータの値に依存します。

`computeContainingFace` が `true` の場合には、新しく追加されたノードによって、ノードを含む正しいフェイスが計算されます。



Note

`apoint` ジオメトリが既にノードとして存在する場合、ノードは追加されずに、既存ノードの識別番号を返します。

Availability: 2.0.0

例

```
SELECT topology.AddNode('ma_topo', ST_GeomFromText('POINT(227641.6 893816.5)', 26986)) AS nodeid; ↵
```

4

関連情報

[AddEdge](#), [CreateTopology](#)

9.9.3 AddEdge

AddEdge — 指定したラインストリングジオメトリを使って、ラインストリングエッジをエッジテーブルに追加し、指定したトポロジスキーマの始点終点をポイントノードテーブルに追加し、新しい (または既存の) エッジの識別番号を返します。

Synopsis

```
bigint AddEdge(varchar toponame, geometry aline);
```

説明

指定したラインストリングジオメトリを使って、指定した **toponame** スキーマのノードテーブルにノードを追加し、新しいまたは既存のレコードのエッジ識別番号を返します。ラインストリングエッジをエッジテーブルに追加し、指定したトポロジスキーマの始点終点をポイントノードテーブルに追加し、新規または既存のエッジの識別番号を返します。新しく追加されたエッジは両側にユニバースフェイスを持ち、自分自身にリンクしています。



Note

aline ジオメトリが既存のエッジとクロスしたり、既存のエッジをオーバーラップしたり、既存のエッジを包含したり、既存のエッジに包含されたりする場合には、エラーが投げられ、エッジは追加されません。



Note

aline のジオメトリは **srid** がトポロジで指定されたものと同じである必要があります、異なる場合には、不正な空間参照系エラーが投げられます。

GEOS モジュールで実現しています。



Warning

AddEdge は 3.5.0 では非推奨です。 **TopoGeo_AddLineString** を代わりに使います。

Availability: 2.0.0

例

```
SELECT topology.AddEdge('ma_topo', ST_GeomFromText('LINESTRING(227575.8 893917.2, 227591.9 893900.4)', 26986)) AS edgeid;
```

1

```
SELECT topology.AddEdge('ma_topo', ST_GeomFromText('LINESTRING(227591.9 893900.4, 227622.6 893844.2, 227641.6 893816.5, 227704.5 893778.5)', 26986)) AS edgeid;
```

2

This query errors because the edge intersects an existing edge away from the endpoints.

```
SELECT topology.AddEdge('ma_topo', ST_GeomFromText('LINESTRING(227591.2 893900, 227591.9 893900.4, 227704.5 893778.5)', 26986)) AS edgeid;
```

```
ERROR: Edge intersects (not on endpoints) with existing edge 1
```

関連情報

[TopoGeo_AddLineString](#), [CreateTopology](#), [Section 4.5](#)

9.9.4 AddFace

AddFace — フェイスプリミティブをトポロジに登録し、その識別番号を得ます。

Synopsis

bigint **AddFace**(varchar toponame, geometry apolygon, boolean force_new=false);

説明

フェイスプリミティブをトポロジに登録し、その識別番号を得ます。

新しく追加されたフェイスのために、境界を形成するエッジとフェイスに含まれるエッジは `left_face` と `right_face` フィールドに正しい値を持つよう更新されます。フェイスに含まれる孤立ノードも正しい `containing_face` フィールド値を持つよう更新されます。



Note

この関数はエッジテーブルの `next_left_edge` と `next_right_edge` フィールドの読み書きを行いません。

対象トポロジは妥当 (自己インタセクトするエッジが無い) と仮定しています。ポリゴンの境界が既存のエッジでは完全には定義されない場合や、ポリゴンが既存のフェイスにオーバラップする場合には、例外が投げられます。

`apolygon` ジオメトリがフェイスとして既に存在している場合には、`force_new` が FALSE (デフォルト) なら既存フェイスのフェイス ID を返し、`force_new` が TRUE なら新しい ID が新規登録されたフェイスに割り当てられます。

**Note**

既存フェイスの新規登録が実行される時 (force_new=true)、そのエッジ内の既存のフェイスへの参照のダングルを解決しません。また、関連テーブルのノードと既存フェイスのレコードの MBR(訳注: 最小境界矩形) フィールドの更新も行いません。これに対応するのは、この関数を呼び出した側です。

**Note**

apolygon ジオメトリはトポロジと同じ srid である必要があり、異なる場合には、不正な空間参照系エラーが投げられます。

Availability: 2.0.0

例

First add the edges using generate_series as an iterator. This query only works for polygons with fewer than 10000 points because of the maximum value in generate_series.

```
SELECT topology.AddEdge('ma_topo', ST_MakeLine(ST_PointN(geom,i), ST_PointN(geom, i + 1) )) ←
  As edgeid
FROM (SELECT ST_NPoints(geom) AS npt, geom
      FROM
        (SELECT ST_Boundary(ST_GeomFromText('POLYGON((234896.5 899456.7,234914 899436.4,234946.6 899356.9,234872.5 899328.7,
        234891 899285.4,234992.5 899145, 234890.6 899069,234755.2 899255.4,
        234612.7 899379.4,234776.9 899563.7,234896.5 899456.7))', 26986) ) As geom
      ) As geoms) As facen CROSS JOIN generate_series(1,10000) As i
WHERE i < npt;
```

```
3
4
5
6
7
8
9
10
11
12
```

Then add the face.

```
SELECT topology.AddFace(
  'ma_topo',
  ST_GeomFromText(
    'POLYGON((234896.5 899456.7,234914 899436.4,234946.6 899356.9,234872.5 899328.7,
    234891 899285.4,234992.5 899145, 234890.6 899069,234755.2 899255.4,
    234612.7 899379.4,234776.9 899563.7,234896.5 899456.7))',
    26986) ) As faceid;
```

```
1
```

関連情報

[AddEdge](#), [CreateTopology](#), [Section 4.5](#)

9.9.5 ST_Simplify

ST_Simplify — 与えた TopoGeometry を「シンプル化した」ジオメトリを返します。ダグラス-ポーカーのアルゴリズムを使います。

Synopsis

```
geometry ST_Simplify(topogeometry tg, float8 tolerance);
```

説明

与えた TopoGeometry を「シンプル化した」ジオメトリを返します。個々の要素エッジに対してダグラス-ポーカーのアルゴリズムを使います。



Note

返されるジオメトリは単純でなかったり不正であったりする場合があります。
要素エッジの分割によって単純性/妥当性を維持することがあります。

GEOS モジュールで実現しています。

Availability: 2.1.0

関連情報

[ST_Simplify](#) (ジオメトリ), [ST_IsSimple](#), [ST_IsValid](#), [ST_ModEdgeSplit](#)

9.9.6 RemoveUnusedPrimitives

RemoveUnusedPrimitives — 存在する TopoGeometry オブジェクトを定義するのに必要でないトポロジプリミティブを削除します。

Synopsis

```
bigint RemoveUnusedPrimitives(text topology_name, geometry bbox);
```

説明

存在する TopoGeometry オブジェクトを表現するのに厳密には必要でない全てのプリミティブ (nodes, edges, faces) を見つけ、これらを削除し、トポロジの妥当性 (エッジのリンク、フェイスのラベル) を維持します。

新しいプリミティブ識別子が作られませんが、既存のプリミティブはフェイスのマージ (エッジの削除のため) や修復されたエッジ (ノードの削除のため) を含むために拡張されます。

Availability: 3.3.0

関連情報

[ST_ModEdgeHeal](#), [ST_RemEdgeModFace](#)

9.10 TopoGeometry コンストラクタ

9.10.1 CreateTopoGeom

CreateTopoGeom — 新しい TopoGeometry オブジェクトを topo エレメント配列から生成します - tg_type: 1:[multi]point, 2:[multi]line, 3:[multi]poly, 4:collection

Synopsis

```
topogeometry CreateTopoGeom(varchar toponame, integer tg_type, integer layer_id, topoelementarray tg_objs, bigint tg_id);
```

```
topogeometry CreateTopoGeom(varchar toponame, integer tg_type, integer layer_id);
```

説明

layer_id で示されたレイヤで TopoGeometry オブジェクトを生成し、toponame スキーマの関連テーブルに登録します。

tg_type は次の整数値とします: 1:[multi]point (点), 2:[multi]line (線), 3:[multi]poly (面), 4:collection。layer_id は、topology.layer テーブル内のレイヤ識別番号です。

点レイヤはノードの集合から形成され、線レイヤはエッジの集合から形成され、面レイヤはフェイスの集合から形成され、コレクションはノード、エッジ、フェイスの混合から形成されます。

要素の配列を省略した場合、空の TopoGeometry オブジェクトが生成されます。

Availability: 1.1

例: 既存エッジからの形成

ri_topo スキーマ内 (ST_CreateTopoGeo の例でロードしてあります) で、ラインタイプ (整数値で 2) の、layer_id が 2 のレイヤ (ri_roads) の最初のエッジから TopoGeometry を生成します。

```
INSERT INTO ri.ri_roads(road_name, topo) VALUES('Unknown', topology.CreateTopoGeom('ri_topo' ←
',2,2, '{{1,2}}'::topology.topoelementarray);
```

例: 面ジオメトリから最善と推測される TopoGeometry への変換

フェイスのコレクションから形成されるジオメトリがあるとして。blockgroups テーブルがあり、それぞれの区画群の TopoGeometry を知りたいとします。データが完全に整列しているなら、次のようにできます。

Create the topogeometry column.

```
SELECT topology.AddTopoGeometryColumn(
    'topo_boston',
    'boston', 'blockgroups', 'topo', 'POLYGON');
```

1

Update the column assuming everything is perfectly aligned with the edges.

```

UPDATE boston.blockgroups AS bg
    SET topo = topology.CreateTopoGeom(
        'topo_boston',
        3, 1,
        foo.bfaces)
FROM (SELECT b.gid, topology.TopoElementArray_Agg(ARRAY[f.face_id,3]) As bfaces
      FROM boston.blockgroups As b
      INNER JOIN topo_boston.face As f ON b.geom && f.mbr
      WHERE ST_Covers(b.geom, topology.ST_GetFaceGeometry('topo_boston', f.face_id))
      GROUP BY b.gid) As foo
WHERE foo.gid = bg.gid;

```

The world is rarely perfect, so this version allows some error. It counts a face if 50 percent of it falls within the expected block group boundary.

```

UPDATE boston.blockgroups AS bg
    SET topo = topology.CreateTopoGeom(
        'topo_boston',
        3, 1,
        foo.bfaces)
FROM (SELECT b.gid, topology.TopoElementArray_Agg(ARRAY[f.face_id,3]) As bfaces
      FROM boston.blockgroups As b
      INNER JOIN topo_boston.face As f ON b.geom && f.mbr
      WHERE ST_Covers(b.geom, topology.ST_GetFaceGeometry('topo_boston', f.face_id))
      OR
      ( ST_Intersects(b.geom, topology.ST_GetFaceGeometry('topo_boston', f.face_id))
        AND ST_Area(ST_Intersection(b.geom, topology.ST_GetFaceGeometry('topo_boston', ←
          f.face_id) ) ) >
          ST_Area(topology.ST_GetFaceGeometry('topo_boston', f.face_id))*0.5
        )
      GROUP BY b.gid) As foo
WHERE foo.gid = bg.gid;

```

To convert the topogeometry back to a denormalized geometry aligned with faces and edges, cast the topogeometry to a geometry. The resulting geometries are aligned with the TIGER street centerlines.

```

UPDATE boston.blockgroups SET new_geom = topo::geometry;

```

関連情報

[AddTopoGeometryColumn](#), [toTopoGeom](#) [ST_CreateTopoGeo](#), [ST_GetFaceGeometry](#), [TopoElementArray](#), [TopoElementArray_Agg](#)

9.10.2 toTopoGeom

toTopoGeom — 単純なジオメトリから TopoGeometry を生成します。

Synopsis

```

topogeometry toTopoGeom(geometry geom, varchar toponame, integer layer_id, float8 tolerance=-1);

```

```

topogeometry toTopoGeom(geometry geom, topogeometry topogeom, float8 tolerance=-1);

```


説明

単純なジオメトリから **TopoGeometry** を生成します。

入力ジオメトリを表現するために必要なトポロジプリミティブが、下位にあるトポロジに追加されます。既存のものを分割することもあります。 **relation** テーブル内の **TopoGeometry** の出力に紐づきます。

既存の **TopoGeometry** オブジェクトは形状を維持します (**topogeom** が与えられている場合には、それが例外となる可能性があります)。

tolerance が与えられた場合には、入力ジオメトリを既存のプリミティブにスナップします。特別な値である -1 (デフォルト) ではトポロジーの精度を許容値として使うことになります。

1 番目の形式では、新しい **TopoGeometry** は、与えられたトポロジ (**toponame**) の与えられたレイヤ (**layer_id**) に作られます。

2 番目の形式では、変換結果のプリミティブが、既存の **TopoGeometry** (**topogeom**) に追加されます。また、最終の形状にスペースを追加することがあります。新しい形状を完全に持つには、古いものを入れ替えます。 **clearTopoGeom** を参照して下さい。

Availability: 2.0

Enhanced: 2.1.0 版では、既存の **TopoGeometry** を取る形式が追加されました。

Changed: 3.7.0 許容値 0 を実際の 0 として扱い、-1 (新デフォルト値) をそのトポロジーの精度を使うよう求めるように変更されました。

例

これは完全に全て揃ったワークフローです。

Create the topology without allowing any tolerance.

```
SELECT topology.CreateTopology('topo_boston_test', 2249);
```

Create a new table.

```
CREATE TABLE nei_topo(gid serial primary key, nei varchar(30));
```

Add a topogeometry column to the new table.

```
SELECT topology.AddTopoGeometryColumn('topo_boston_test', 'public', 'nei_topo', 'topo', ' ←
MULTIPOLYGON') AS new_layer_id;
```

1

Use the new layer identifier to populate the topogeometry column.

```
INSERT INTO nei_topo(nei, topo)
SELECT nei, topology.toTopoGeom(geom, 'topo_boston_test', 1)
FROM neighborhoods
WHERE gid BETWEEN 1 and 15;
```

Verify what happened.

```
SELECT * FROM
topology.TopologySummary('topo_boston_test');
```

```
Topology topo_boston_test (5), SRID 2249, precision 0
61 nodes, 87 edges, 35 faces, 15 topogeoms in 1 layers
Layer 1, type Polygonal (3), 15 topogeoms
Deploy: public.nei_topo.topo
```

Shrink all TopoGeometry polygons by 10 meters.

```
UPDATE nei_topo SET topo = toTopoGeom(ST_Buffer(topo, -10), clearTopoGeom(topo), 0);
```

Find faces that are left without any TopoGeometry reference after the shrink operation.

```
SELECT ST_GetFaceGeometry('topo_boston_test', f.face_id)
FROM topo_boston_test.face f
WHERE f.face_id
> 0
AND NOT EXISTS (
  SELECT * FROM topo_boston_test.relation
  WHERE layer_id = 1 AND element_id = f.face_id
);
```

関連情報

[CreateTopology](#), [AddTopoGeometryColumn](#), [CreateTopoGeom](#), [TopologySummary](#), [clearTopoGeom](#)

9.10.3 TopoElementArray_Agg

TopoElementArray_Agg — element_id とタイプの配列 (topoelements) からなる topoelementarray を返します。

Synopsis

topoelementarray **TopoElementArray_Agg**(topoelement set tefield);

説明

TopoElementArrayを**TopoElement**の集合から生成するために使います。

Availability: 2.0.0

例

```
SELECT topology.TopoElementArray_Agg(ARRAY[e, t] ORDER BY e, t) As tea
FROM generate_series(1,3) As e CROSS JOIN generate_series(1,4) As t;
```

```
{{1,1},{1,2},{1,3},{1,4},{2,1},{2,2},{2,3},{2,4},{3,1},{3,2},{3,3},{3,4}}
```

関連情報

[TopoElement](#), [TopoElementArray](#)

9.10.4 TopoElement

TopoElement — TopoGeometry を TopoElement に変換します。

Synopsis

topoelement **TopoElement**(topogeometry topo);

説明

TopoGeometryを**TopoElement**に変換します。

Availability: 3.4.0

例

これは完全に全て揃ったワークフローです。

Return the TopoElement representation of each topogeometry.

```
SELECT TopoElement(topo)
FROM neighborhoods;
```

Use a cast to aggregate topogeometries into a TopoElementArray.

```
SELECT topology.TopoElementArray_Agg(topo::topoelement)
FROM neighborhoods
GROUP BY city;
```

関連情報

TopoElementArray_Agg, **TopoGeometry**, **TopoElement**

9.11 TopoGeometry エディタ

9.11.1 clearTopoGeom

clearTopoGeom — TopoGeometry の中身を消去します。

Synopsis

topogeometry **clearTopoGeom**(topogeometry topogeom);

説明

TopoGeometryの中身を消去し、空にします。**toTopoGeom**と併用して、既存オブジェクトと上位にある依存オブジェクトの形状の置換に、だいたい便利です。

Availability: 2.1

例

Shrink all TopoGeometry polygons by 10 meters.

```
UPDATE nei_topo SET topo = toTopoGeom(ST_Buffer(topo, -10), clearTopoGeom(topo), 0);
```

関連情報

[toTopoGeom](#)

9.11.2 TopoGeom_addElement

TopoGeom_addElement — TopoGeometry の定義に要素を追加します。

Synopsis

topogeometry **TopoGeom_addElement**(topogeometry tg, topoelement el);

説明

TopoElementを TopoGeometry オブジェクトの定義に追加します。要素がすでに定義の一部になっていても、エラーは発生しません。

Availability: 2.3

例

Add edge 5 to TopoGeometry tg.

```
UPDATE mylayer SET tg = TopoGeom_addElement(tg, '{5,2}');
```

関連情報

[TopoGeom_remElement](#), [CreateTopoGeom](#)

9.11.3 TopoGeom_remElement

TopoGeom_remElement — TopoGeometry の定義から要素を削除します。

Synopsis

topogeometry **TopoGeom_remElement**(topogeometry tg, topoelement el);

説明

TopoGeometry オブジェクトの定義から**TopoElement**を削除します。

Availability: 2.3

例

Remove face 43 from TopoGeometry tg.

```
UPDATE mylayer SET tg = TopoGeom_remElement(tg, '{43,3}');
```

関連情報

[TopoGeom_addElement](#), [CreateTopoGeom](#)

9.11.4 TopoGeom_addTopoGeom

TopoGeom_addTopoGeom — TopoGeometry の要素を他の TopoGeometry の定義に追加します。

Synopsis

topogeometry **TopoGeom_addTopoGeom**(topogeometry tgt, topogeometry src);

説明

TopoGeometryの要素を他の TopoGeometry の定義に追加します。ソースオブジェクト内にすべての要素を保持する必要がある場合には、キャッシュされたタイプ (**type** 属性) をコレクションに変更する可能性があります。

二つの TopoGeometry オブジェクトは * 同じ * トポロジに対して定義されている必要があります。また、階層的な定義がある場合には、同じ子レイヤの要素で構成されている必要があります。

Availability: 3.2

例

Set an overall TopoGeometry value to be composed of all elements from specific TopoGeometry values.

```
UPDATE mylayer SET tg_overall = TopoGeom_addTopoGeom(
    TopoGeom_addTopoGeom(
        clearTopoGeom(tg_overall),
        tg_specific1
    ),
    tg_specific2
);
```

関連情報

[TopoGeom_addElement](#), [clearTopoGeom](#), [CreateTopoGeom](#)

9.11.5 toTopoGeom

toTopoGeom — ジオメトリの形状を既存の TopoGeometry に追加します。

説明

[toTopoGeom](#)を参照して下さい。

9.12 TopoGeometry アクセサ

9.12.1 GetTopoGeomElementArray

`GetTopoGeomElementArray` — 与えられた `TopoGeometry` (プリミティブ要素) のトポロジ要素とタイプを含む `topoelementarray` (`topoelement` の配列) を返します。

Synopsis

```
topoelementarray GetTopoGeomElementArray(varchar toponame, integer layer_id, bigint tg_id);  
topoelementarray GetTopoGeomElementArray(topogeometry tg);
```

説明

トポロジ要素と与えられた `TopoGeometry` (プリミティブ要素) のタイプを含む **TopoElementArray** を返します。`GetTopoGeomElements` に近いですが、これは、要素群をデータセットでなく配列で返しています。

`tg_id` は、`topology.layer` テーブル内の `layer_id` で指定されたレイヤのトポロジにおける `TopoGeometry` オブジェクトの識別番号です。

Availability: 1.1

例

関連情報

GetTopoGeomElements, **TopoElementArray**

9.12.2 GetTopoGeomElements

`GetTopoGeomElements` — 与えられた `TopoGeometry` (プリミティブ要素) の、トポロジの `element_id` と `element_type` を含む `topoelement` オブジェクトの集合を返します。

Synopsis

```
setof topoelement GetTopoGeomElements(varchar toponame, integer layer_id, bigint tg_id);  
setof topoelement GetTopoGeomElements(topogeometry tg);
```

説明

`toponame` スキーマ内で与えられた `TopoGeometry` オブジェクトを形成するプリミティブなトポロジ要素 **TopoElement** (1: ノード、2: エッジ、3: フェイス) に対応した `element_id`, `element_type` (`topoelements`) の集合を返します。

`tg_id` は、`topology.layer` テーブル内の `layer_id` で指定されたレイヤのトポロジにおける `TopoGeometry` オブジェクトの識別番号です。

Availability: 2.0.0

例

関連情報

[GetTopoGeomElementArray](#), [TopoElement](#), [TopoGeom_addElement](#), [TopoGeom_remElement](#)

9.12.3 ST_SRID

ST_SRID — TopoGeometry の空間参照識別子を返します。

Synopsis

integer **ST_SRID**(topogeometry tg);

説明

ST_Geometry の spatial_ref_sys テーブルで定義されている空間参照系の識別番号を返します。Section 4.5を参照して下さい。



Note

spatial_ref_sys テーブルは PostGIS が知る参照系の全てのカタログを作っていて、ある空間参照系から他の空間参照系に変換するために使われます。ジオメトリの変換を予定している場合は正しい空間参照系の識別番号を持っているか確認することは重要です。

Availability: 3.2.0



このメソッドは SQL/MM 仕様の実装です。SQL-MM 3: 14.1.5

例

```
SELECT ST_SRID(ST_GeomFromText('POINT(-71.1043 42.315)',4326));
         4326
```

関連情報

Section 4.5, [ST_SetSRID](#), [ST_Transform](#), [ST_SRID](#)

9.13 TopoGeometry 出力

9.13.1 AsGML

AsGML — TopoGeometry の GML 表現を返します。

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);
```

説明

GML3 の書式で TopoGeometry の GML 表現を返します。nsprefix_in が指定されていない場合には、gml が使われます。非修飾名前空間を得るには nsprefix に空文字列を渡します。精度 (デフォルトは 15) と options (デフォルトは 1) パラメーターは、与えられた場合には、裏で呼んでいる ST_AsGML にそのまま渡します。

visitedTable パラメーターは、与えられた場合には、訪問したノード要素とエッジ要素のトラックを保持するために使われ、重複定義になるところで相互参照 (xlink:xref) を使います。テーブルは整数カラムである 'element_type' と 'element_id' とを持つことを期待されます。この関数を呼び出したユーザは、このテーブルへの読み込み権限と書き込み権限とが必要です。効率よくするには、element_type と element_id に、この順序でインデックスを定義します。インデックスは一意制約をカラムに追加すると自動的に生成されます。例を示します。

```
CREATE TABLE visited (
  element_type integer, element_id integer,
  unique(element_type, element_id)
);
```

idprefix パラメーターは、指定された場合には、Edge タグ識別子と Node タグ識別子の前に付きます。

gmlver パラメーターは、指定された場合には、裏で呼んでいる ST_AsGML に渡されます。デフォルトは 3 です。

Availability: 2.0.0

例

ここでは **CreateTopoGeom** で生成した TopoGeometry を使用しています。

```
SELECT topology.AsGML(topo) As rdgml
FROM ri.roads
WHERE road_name = 'Unknown';
```

```
<gml:TopoCurve>
  <gml:directedEdge>
    <gml:Edge gml:id="E1">
      <gml:directedNode orientation="-">
        <gml:Node gml:id="N1"/>
      </gml:directedNode>
    </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>

```

上の例から名前空間を取り除いた例です。

```

SELECT topology.AsGML(topo,'') As rdgml
FROM ri.roads
WHERE road_name = 'Unknown';

```

```

<TopoCurve>
  <directedEdge>
    <Edge id="E1">
      <directedNode orientation="-">
        <Node id="N1"/>
      </directedNode>
      <directedNode
></directedNode>
        <curveProperty>
          <Curve srsName="urn:ogc:def:crs:EPSG::3438">
            <segments>
              <LineStringSegment>
                <posList srsDimension="2"
>384744 236928 384750 236923 384769 236911 384799 236895 384811 236890
      384833 236884 384844 236882 384866 236881 384879 236883 384954 ←
      236898 385087 236932 385117 236938
      385167 236938 385203 236941 385224 236946 385233 236950 385241 ←
      236956 385254 236971
      385260 236979 385268 236999 385273 237018 385273 237037 385271 ←
      237047 385267 237057 385225 237125
      385210 237144 385192 237161 385167 237192 385162 237202 385159 ←
      237214 385159 237227 385162 237241
      385166 237256 385196 237324 385209 237345 385234 237375 385237 ←
      237383 385238 237399 385236 237407
      385227 237419 385213 237430 385193 237439 385174 237451 385170 ←
      237455 385169 237460 385171 237475
      385181 237503 385190 237521 385200 237533 385206 237538 385213 ←
      237541 385221 237542 385235 237540 385242 237541
      385249 237544 385260 237555 385270 237570 385289 237584 385292 ←
      237589 385291 237596 385284 237630</posList>
            </LineStringSegment>
          </segments>
        </Curve>
      </curveProperty>
    </Edge>
  </directedEdge>
</TopoCurve>

```

```
        </segments>
      </Curve>
    </curveProperty>
  </Edge>
</directedEdge>
</TopoCurve>
```

関連情報

[CreateTopoGeom](#), [ST_CreateTopoGeo](#)

9.13.2 AsTopoJSON

AsTopoJSON — opoGeometry の TopoJSON 表現を返します。

Synopsis

```
text AsTopoJSON(topogeometry tg, regclass edgeMapTable);
```

説明

TopoGeometry の TopoJSON 表現を返します。**edgeMapTable** が NULL でない場合には、エッジ識別番号とアーク添え字のルックアップと格納のマッピングに使われます。これは、最終的な文書内のコンパクトな“arcs”配列ができるようにするためです。

このテーブルは、与えられた場合には、“serial” 型の“arc_id” カラムと整数型の“edge_id” とを持つことが期待されます。関数はこのテーブルに“edge_id” を問い合わせるので、このカラムにインデックスを追加することが推奨されます。



Note

TopoJSON でのアークのインデックスは 0 始まりですが、“edgeMapTable” テーブルでは 1 始まりです。

完全な TopoJSON 文書は、この関数が返すスニペットだけでなく、実際の arcs メンバと、いくつかのヘッダを含む必要があります。[TopoJSON specification](#)をご覧ください。

Availability: 2.1.0

Enhanced: 2.2.1 点入力に対応するようになりました

関連情報

[ST_AsGeoJSON](#)

例

Create a temporary edge map for the TopoJSON output.

```
CREATE TEMP TABLE edgemap(arc_id serial, edge_id int unique);
```

Build the TopoJSON header, objects, arcs, and footer.

```
SELECT '{ "type": "Topology", "transform": { "scale": [1,1], "translate": [0,0] }, "objects" ←
  ": {'

UNION ALL SELECT '' || feature_name || '": ' || AsTopoJSON(feature, 'edgemap')
FROM features.big_parcel WHERE feature_name = 'P3P4';

WITH edges AS (
  SELECT m.arc_id, e.geom FROM edgemap m, city_data.edge e
  WHERE e.edge_id = m.edge_id
), points AS (
  SELECT arc_id, (ST_DumpPoints(geom)).* FROM edges
), compare AS (
  SELECT p2.arc_id,
    CASE WHEN p1.path IS NULL THEN p2.geom
    ELSE ST_Translate(p2.geom, -ST_X(p1.geom), -ST_Y(p1.geom))
  END AS geom
  FROM points p2 LEFT OUTER JOIN points p1
  ON ( p1.arc_id = p2.arc_id AND p2.path[1] = p1.path[1]+1 )
  ORDER BY arc_id, p2.path
), arcdump AS (
  SELECT arc_id, (regexp_matches(ST_AsGeoJSON(geom), '\[.*\]'))[1] as t
  FROM compare
), arcs AS (
  SELECT arc_id, '[' || array_to_string(array_agg(t), ',') || ']' as a FROM arcdump
  GROUP BY arc_id
  ORDER BY arc_id
)
SELECT '}', "arcs": [' UNION ALL
SELECT array_to_string(array_agg(a), E',\n') from arcs

UNION ALL SELECT '}]':::text as t;
```

```
{ "type": "Topology", "transform": { "scale": [1,1], "translate": [0,0] }, "objects": {
  "P3P4": { "type": "MultiPolygon", "arcs": [[[-1]], [[6,5,-5,-4,-3,1]]]}
}, "arcs": [
  [[25,30],[6,0],[0,10],[-14,0],[0,-10],[8,0]],
  [[35,6],[0,8]],
  [[35,6],[12,0]],
  [[47,6],[0,8]],
  [[47,14],[0,8]],
  [[35,22],[12,0]],
  [[35,14],[0,8]]
]]}
```

9.14 トポロジ空間関係関数

9.14.1 Equals

Equals — 二つの TopoGeometry が同じトポロジプリミティブで成っている場合に **true** を返します。

Synopsis

boolean **Equals**(topogeometry tg1, topogeometry tg2);

説明

二つ TopoGeometry が同じトポロジプリミティブで成っている場合に true を返します。



Note

この関数はジオメトリコレクションの TopoGeometry に対応していません。異なるトポロジからなる TopoGeometry との比較もできません。

Availability: 1.1.0



この関数は 3 次元に対応し、Z 値を削除しません。

例

The following example uses the nei_topo table created in the [toTopoGeom](#) examples.

```
SELECT topology.Equals(a.topo, b.topo)
FROM nei_topo AS a
JOIN nei_topo AS b ON b.gid = a.gid
WHERE a.gid = 1;
```

t

関連情報

[GetTopoGeomElements](#), [ST_Equals](#)

9.14.2 Intersects

Intersects — 二つの TopoGeometry からのプリミティブの組がインタセクトする場合に true を返します。

Synopsis

boolean **Intersects**(topogeometry tg1, topogeometry tg2);

説明

二つの TopoGeometry からのプリミティブがインタセクトする場合に true を返します。



Note

この関数はジオメトリコレクションの TopoGeometry に対応していません。異なるトポロジからなる TopoGeometry との比較もできません。また、現在のところ、階層 TopoGeometry (他の TopoGeometry からなる TopoGeometry) に対応していません。

Availability: 1.1.0



この関数は 3 次元に対応し、Z 値を削除しません。

例

The following example uses the `nei_topo` table created in the [toTopoGeom](#) examples.

```
SELECT topology.Intersects(a.topo, b.topo)
FROM nei_topo AS a
JOIN nei_topo AS b ON b.gid = a.gid
WHERE a.gid = 1;
```

t

関連情報

[ST_Intersects](#)

9.15 トポロジのインポートとエクスポート

トポロジを生成したり、場合によってはトポロジレイヤに関連付けると、バックアップや他のデータベースへの転送のために、ファイルに出力するフォーマットでエクスポートした方がいいでしょう。

トポロジはテーブルの集合 (プリミティブで 4 テーブル、レイヤで任意数) と、メタデータテーブルのレコード (`topology.topology` と `topology.layer`) になっているため、PostgreSQL の標準的なダンプ/レストアのツールでは問題があります。さらに、トポロジの識別子はデータベース間で同じではないため、トポロジのパラメータはレストア時に変更する必要があります。

トポロジのエクスポート/レストアを簡単化するために、二つの実行可能ファイルが提供されています。`pgtopo_export` と `pgtopo_import` です。使用例は次の通りです。

```
pgtopo_export dev_db topo1 | pgtopo_import topo1 | psql staging_db
```

9.15.1 トポロジエクスポートの使用

`pgtopo_export` スクリプトはデータベース名とトポロジ名を取り、トポロジ (と関連するレイヤ) を新しいデータベースにインポートするために使うことができるダンプファイルを出力します。

デフォルトでは `pgtopo_export` はダンプファイルを標準出力に書くので、`pgtopo_import` にパイプで繋げたり、ファイルにリダイレクトする (端末への出力を止める) ことができます。出力ファイル名をコマンドラインスイッチ `-f` で指定することもできます。

デフォルトでは `pgtopo_export` は、指定されたトポロジに対して定義されたレイヤの全てのダンプが含まれます。これは、必要なデータより多い場合もありますし、動作しない場合もあり (レイヤテーブルが複雑な依存関係を持っている場合)、この場合には `--skip-layers` スイッチでレイヤをスキップし、別々に処理することができます。

`pgtopo_export` を `--help` (または短縮の `-h`) スイッチで実行すると、常に短い使用法文字列を印字します。

ダンプファイルフォーマットは、少なくともフォーマットバージョン情報を持つ `pgtopo_dump_version` ファイルを含む `pgtopo_export` ディレクトリの圧縮された tar アーカイブです。バージョン 1 では、ディレクトリにはトポロジのプリミティブテーブル (`node`, `edge_data`, `face`, `relation`) と、プリミティブと関連付けられたトポロジとレイヤとを持つタブ区切り CSV ファイルが含まれ、(`--skip-layers` が与えられない場合) 与えられたトポロジのレイヤとして報告されたテーブルの PostgreSQL ダンプのカスタムフォーマットも含まれます。

9.15.2 トポロジインポータの使用

`pgtopo_import` スクリプトは `pgtopo_export` 書式のトポロジのダンプと、生成するトポロジに与える名前とを取り、そこからトポロジと関連するレイヤを再構築する SQL スクリプトを出力します。

生成された SQL には、与えられた名前でトポロジを生成し、プリミティブなデータをロードし、格納して、`TopoGeometry` 値を確実に正しいトポロジに関連付けることで全てのトポロジレイヤを登録する手続きが含まれます。

デフォルトでは `pgtopo_import` は標準入力からダンプを読むので、`pgtopo_export` をパイプで繋げて併用することができます。入力ファイル名をコマンドラインスイッチ `-f` で指定することもできます。

デフォルトでは `pgtopo_import` の SQL ファイル出力には、ダンプで発見された全てのレイヤを格納するためのコードが含まれます。

これらは、ダンプ内にあるテーブル名が既に対象データベース内のテーブル名で使われている場合には、望ましくないようになるか動作しない可能性があります。この場合には、レイヤをスキップする `--skip-layers` スイッチを使い、別に (または後で) 処理することができます。

ロードして、レイヤを名前付きトポロジにリンクする SQL は `--only-layers` スイッチで生成できます。これは、名前の競合を解決した * 後に * レイヤをロードしたり、異なるトポロジのレイヤ (開始トポロジで空間的に単純化されたもの等が挙げられます) にリンクするのに使えます。

対象トポロジが既に存在していて事前に削除したいなら `--drop-topology` スイッチを渡すことができます (PostGIS 3.6.0 以降)。

Chapter 10

ラスタデータの管理、クエリ、アプリケーション

10.1 ラスタのロードと生成

`raster2pgsql` ラスタローダを使って PostGIS ラスタを既存のラスタファイルからロードするのは、最もよく行われます。

10.1.1 `raster2pgsql` を使ってラスタをロードする

`raster2pgsql` は、GDAL がサポートするラスタ書式を PostGIS ラスタテーブルにロードするのに適切な SQL にする実行ファイルです。ラスタのオーバビューの生成だけでなく、ラスタファイルのフォルダのロードも可能です。

かなり多くの場合、(ご自分の GDAL ライブラリをコンパイルしない限り) `raster2pgsql` は PostGIS の一部としてコンパイルされるので、実行ファイルがサポートするラスタタイプは GDAL の依存ライブラリでコンパイルされたものと同じになります。お手持ちの `raster2pgsql` がサポートするラスタタイプの一覧は、`-G` スイッチで見ることができます。



Note

同じアラインメントを持つラスタの集合から特定の要素のオーバビューを生成する時、オーバビューが同じアラインメントを持たないことがあります。オーバビューが同じアラインメントを持たない例については <http://trac.osgeo.org/postgis/ticket/1764> をご覧下さい。

10.1.1.1 使用例

ローダを用いて入力ファイルを 100x100 のタイルで生成して、データベースにアップロードする例は、次の通りです。

```
# -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**

対象テーブル名の一部としてスキーマを指定しない場合には、テーブルはデータベースのデフォルトスキーマ内かユーザーが接続しているスキーマ内に作られます。

変換とアップロードは UNIX のパイプを使うと一回で実行できます。

```
raster2pgsql -s 4326 -I -C -M *.tif -F -t 100x100 public.demelevation | psql -d gisdb
```

マサチューセッツ州平面のメートル単位の空中写真タイルを **aerial** という名前のスキーマにロードします。元の画像と 2, 4 レベルのオーバビューのテーブルとを生成します。データ格納に **COPY** を使用し (データベースに仲介ファイルなくまっすぐ入ります)、**-e** でトランザクションを指定しないようにします (待たずにテーブルのデータを見たい場合には良いです)。ラスタを 128x128 ピクセルのタイルに分解してラスタ制約を適用します。INSERT モードでなく COPY モードを使用します。**-F** で、カラム名をタイル切り出し元ファイルのファイル名にします。

```
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 コマンドの出力は次のようになります

```
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 オプション

Long options that take values also accept the **--option=value** spelling.

-?, --help ヘルプを表示します。引数を全く指定しない場合にも表示されます。

-G, --gdal-formats サポートされているラスタ書式を印字します。

(c|a|d|p) 相互に排他的なオプションです。

-c 新しいテーブルを生成し、ラスタを入れます。これがデフォルトモードです。

- a 既存のテーブルにラスタを追加します。
- d テーブルを削除し、新しいテーブルを生成し、ラスタを入れます。
- p 準備モード、テーブルを作るだけです。

--if-not-exists Use IF NOT EXISTS for table creation in -c and -p modes. When -I or --create-index is also specified, use IF NOT EXISTS for index creation too. This option cannot be used with -d. Append mode requires an explicit creation action.

Atomic loader actions The short modes are presets over explicit actions. The equivalent long options are --drop-table, --create-table, --load-data, --create-index, --add-constraints, --vacuum, --analyze, and --no-transaction. Add --if-not-exists to make active creation actions use IF NOT EXISTS. --drop-table emits DROP TABLE IF EXISTS before the selected actions. With no mode specified, the default create/load actions still apply.

ラスタ処理: ラスタカタログに適切に登録するための制約の適用

- C, **--add-constraints** SRID やピクセルサイズ等のラスタ制約を適用して、raster_columns ビューで適切な登録ができるようにします。
- x, **--no-extent** Disable setting the max extent constraint. Only applied if -C or --add-constraints is also used.
- r, **--regular-blocking** Set the constraints (spatially unique and coverage tile) for regular blocking. Only applied if -C or --add-constraints is also used.

ラスタ処理: 入力ラスタデータセットの操作に使われる追加的なパラメータ

- s, **--srid** [<FROM_SRID>:]<SRID> Assign the output raster to the specified SRID. When FROM_SRID is also provided, raster2pgsql marks the input raster with FROM_SRID and emits SQL that reprojects the raster to the target SRID. If the source SRID is omitted or is zero, the raster metadata will be checked to determine an appropriate source SRID. Reprojection cannot be used with copy mode.
- b, **--band** BAND ラスタから抽出するバンドのインデックス (1 始まり)。1 より多いバンドを抽出するには、コンマ (,) で区切ります。指定しない場合、全てのバンドが抽出されます。
- t, **--tile-size** TILE_SIZE 行毎に挿入するラスタを切断します。TILE_SIZE は、「幅 x 高さ」で表現しますが、“auto” を指定すると、最初のラスタを使って適切なタイルサイズが計算され、全てのラスタに適用されます。
- P, **--pad** 全てのタイルが同じ幅と高さを持つことを保証するために、右端、下端のタイルに詰め物を施します。
- R, **--register** ファイルシステム (データベース外) ラスタとして、ラスタを登録します。データベースには、ラスタのメタデータとラスタのファイルパスのみ格納されます (ピクセルは格納されません)。
- l, **--overview-factor** OVERVIEW_FACTOR Create overview of the raster. For more than one factor, separate with comma(,). Overview table name follows the pattern of overview_factor_table where overview_factor is a placeholder for numerical overview factor and table is replaced with the base table name. If a source raster already has an overview with matching dimensions, that overview is reused. Created overview is stored in the database and is not affected by -R. Note that your generated sql file will contain both the main table and overview tables.
- N, **--nodata** NODATA NODATA 値を持たないバンドで使用する NODATA 値を設定します。

データベースオブジェクトの操作に使われる追加的なパラメータ

- f, **--raster-column** COLUMN 出力先ラスタカラムの名前を指定します。デフォルトは‘rast’です。
- F, **--filename** ファイル名でカラムを追加します。
- n, **--filename-column** COLUMN ファイル名カラムの名前を指定します。-F を暗に含みます。
- q, **--quote** PostgreSQL 識別子に引用符を付けます。
- I Alias for --create-index.

- create-index** Create a GiST index on the raster column at the end of this raster2pgsql run. With repeated **-a** append runs, create the index on the final run or after loading; add **--if-not-exists** to make reruns tolerate an existing index.
- M** ラスタテーブルに **vacuum analyze** を行います。
- k, --skip-nodata-check** 空タイルを保持してラスタバンドごとの NODATA 値のチェックを省きます。チェック時間を抑えることにはなりますが、データベース内の不要な行がずっと増えて、これらの行が空タイルとされないことに注意して下さい。
- T, --tablespace tablespace** 生成されるテーブルのテーブルスペースを指定します。**-X** フラグを併用しない場合には、インデックス (主キーを含む) はデフォルトのテーブルスペースを使用することにご注意ください。
- X, --index-tablespace tablespace** テーブルの新しいインデックスに使うテーブル空間を指定します。主キーに適用され、**-I** フラグがある場合においては空間インデックスにも適用されます。
- Y, --copy [MAX_ROWS_PER_COPY] COPY** ステートメントを使い、INSERT を使いません。任意で **max_rows_per_copy** を指定します。指定しない場合のデフォルト値は 50 です。
- e, --no-transaction** ステートメント毎に実行して、トランザクションを使用しないようにします。
- E, --endian ENDIAN** 生成されるラスタのバイナリ出力のエンディアンを制御します。XDR (訳注: ビッグエンディアン) の場合は 0 を、NDR (訳注: リトルエンディアン) の場合は 1 を、それぞれ指定します。デフォルトは 1 です。現時点では NDR 出力のみサポートします。
- V, --wkb-version version** 出力書式の版を指定します。デフォルトは 0 です。現時点では 0 のみサポートします。

10.1.2 PostGIS ラスタ関数を用いたラスタの生成

データベース内でラスタやラスタテーブルを生成したい場合が多くあります。これを行うための関数が多数あります。一般的な手順は次の通りです。

1. 新しいラスタ行を保持するためのラスタカラムを持つテーブルを生成します:

```
CREATE TABLE myrasters(rid serial primary key, rast raster);
```

2. この目標で助けとなる関数は多数あります。他のラスタの派生でないラスタを生成する場合、**ST_MakeEmptyRaster** 順次実行して作業を開始します

ジオメトリからラスタを生成することもできます。**ST_AsRaster**を使います。**ST_Union**や**ST_MapAlgebraFct**や、地図解析関数群等といった、他の関数を組み合わせる場合もあります。

既存テーブルから新しいラスタテーブルを生成するための多数の選択肢があります。たとえば、**ST_Transform**を使って、既存テーブルから異なる投影法のラスタテーブルを生成します

3. はじめにテーブルにデータを入れたら、ラスタカラムに空間インデックスを生成したくなるでしょう:

```
CREATE INDEX myrasters_rast_st_convexhull_idx ON myrasters USING gist(ST_ConvexHull( ← rast));
```

ST_ConvexHullを使用していることに注意して下さい。多くのラスタ演算子はラスタの凸包を元にしています。



Note

2.0 より前の PostGIS ラスタは、エンベロープを基本にして、凸包ではありませんでした。空間インデックスを適切に働かせるには、エンベロープを基本にしたインデックスを削除して、凸包を元にしたインデックスに置き換えます。

4. **AddRasterConstraints**を用いてラスタ制約を適用します

10.1.3 「データベース外」クラウドラスタの使用

`raster2pgsql` ツールは GDAL を使用してラスタデータにアクセスします。また、クラウドの「オブジェクト保存」(AWS S3 や Google Cloud Storage 等) にリモート格納されたラスタを読む能力という、重要な GDAL 機能の利点を享受することができます。

クラウドストレージの効率的な仕様には、「クラウド最適化」された書式の使用が必要です。最も知られて幅広く使われているものは、「cloud optimized GeoTIFF」です。JPEG や非タイル化 TIFF などの非クラウド書式を使うと、発揮される能力は非常に貧弱になります。システムがラスタの一部にアクセスする必要に迫られるたびに全体をダウンロードするためです。

まず、ラスタをあなたが選択したクラウドストレージに保存します。保存したら、アクセスに使う URI が得られます。「http」URI か、時々特別なサービスを示す URI (「s3://bucket/object」等) です。非公開バケットにアクセスするには、接続認証のための GDAL コンフィギュレーションオプションが必要になります。このコマンドはクラウドラスタから読み、データベースに書きます。

```
AWS_ACCESS_KEY_ID=xxxxxxxxxxxxxxxxxxxxxx \
AWS_SECRET_ACCESS_KEY=xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx \
raster2pgsql \
-s 990000 \
-t 256x256 \
-I \
-R \
/vs3/your.bucket.com/your_file.tif \
your_table \
| psql your_db
```

テーブルがロードされたら、データベースにリモートラスタから読むためのアクセス許可を与えます。`postgis.enable_outdb_rasters` と `postgis.gdal_enabled_drivers` の、二つのアクセス許可を設定します。

```
SET postgis.enable_outdb_rasters = true;
SET postgis.gdal_enabled_drivers TO 'ENABLE_ALL';
```

確実に変更するには、データベースに直接設定します。新しい設定を使うには再接続が必要です。

```
ALTER DATABASE your_db SET postgis.enable_outdb_rasters = true;
ALTER DATABASE your_db SET postgis.gdal_enabled_drivers TO 'ENABLE_ALL';
```

非公開ラスタの場合、クラウドラスタから読み取るアクセスキーを与えなければならない場合があります。`raster2pgsql` の書き込みに使用したのと同じキーを、データベース内で `postgis.gdal_vsi_options` 設定と共に使用するように設定できます。`key=value` ペアをスペースで区切ることで、複数のオプションを設定できることに注意してください。

```
SET postgis.gdal_vsi_options = 'AWS_ACCESS_KEY_ID=xxxxxxxxxxxxxxxxxxxxxx
AWS_SECRET_ACCESS_KEY=xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx';
```

データをロードし、パーミッションが設定されると、ラスタテーブルは他のラスタテーブルとおなじように、同じ関数を使って、対話的処理が可能となります。データベースは、ピクセルデータの読み取りが必要なら、クラウドへの接続の全てを扱います。

10.2 ラスタカタログ

PostGIS が生成する、二つのラスタカタログのビューがあります。両方ともラスタテーブルの制約の中に埋め込まれる情報を用いています。結果として、カタログビューは、テーブル内のラスタデータに制約が働いたため、常にラスタデータとの矛盾がありません。

1. `raster_columns` ラスタタイプのデータベースにおける全てのラスタテーブルカラムのカタログです。
2. `raster_overviews` データベース内の、より詳細なテーブルのためのオーバビューを提供するラスタテーブルのカラム全てのカタログです。この種のテーブルは、ロード時に `-l` を指定した時に生成されます。

10.2.1 ラスタカラムカタログ

`raster_columns` は、ラスタタイプのデータベースにおける全てのラスタテーブルカラムのカタログです。テーブルの制約を使ったビューなので、他のデータベースのバックアップからラスタテーブルをリストアしたとしても、情報は常に矛盾がありません。`raster_columns` カタログには次のカラムがあります。

ローダを使わずにテーブルを生成したり、ロード時に `-C` フラグを忘れてしまった場合には、事後に `AddRasterConstraints` で制約を強制でき、`raster_columns` カタログは、ラスタスタイルの共通の情報を登録します。

- `r_table_catalog` テーブルが存在するデータベースです。これは常に現在のデータベースを読みます。
- `r_table_schema` ラスタテーブルが属するデータベーススキーマです。
- `r_table_name` ラスタテーブルです。
- `r_raster_column` ラスタタイプである `r_table_name` テーブルのカラムです。PostGIS には、一つのテーブルに複数のラスタカラムを持つことを妨げません。異なるラスタカラムを持つラスタテーブルが、ラスタカラム毎に複数回出現するテーブルを持つことができます。
- `srid` ラスタの空間参照系識別番号です。Section 4.5にあるエントリであるべきです。
- `scale_x` 地理空間座標とピクセルの間の拡大縮小係数です。これは、ラスタカラムのすべてのタイルが同じ `scale_x` を持ち、制約が適用されている場合のみ出現します。詳細情報については `ST_ScaleX` を参照してください。
- `scale_y` 地理空間座標とピクセルの間の拡大縮小係数です。これは、ラスタカラムのすべてのタイルが同じ `scale_y` を持ち、`scale_y` の制約が適用されている場合のみ出現します。詳細情報については `ST_ScaleY` を参照してください。
- `blocksize_x` ラスタタイルごとの幅 (横方向のピクセル数) です。詳細情報については `ST_Width` を参照してください。
- `blocksize_y` ラスタタイルごとの高さ (縦方向のピクセル数) です。詳細情報については `ST_Height` を参照してください。
- `same_alignment` 全てのラスタタイルが同じアラインメントを持っているかを示す真偽値です。詳細情報については `ST_SameAlignment` を参照してください。
- `regular_blocking` ラスタカラムが空間的に一意かつカバレッジタイルの制約を持つなら、TRUE となります。その他の場合は FALSE になります。
- `num_bands` ラスタ集合のタイルごとのバンド数。これと同じ情報が得られる関数は `ST_NumBands`
- `pixel_types` バンドごとのピクセルタイプを定義する配列です。この配列の要素数はバンド数と同じです。`pixel_types` は、`ST_BandPixelType` で定義されるピクセルタイプの一つを取ります。
- `nodata_values` バンド毎の `nodata_value` を示す倍精度浮動小数点数の配列です。バンド数と同じ配列数となります。これらの値は、バンド毎のほとんどの処理で無視されるべきピクセル値の定義です。これは `ST_BandNoDataValue` で得られる情報と似ています。
- `out_db` ラスタバンドデータがデータベース外で維持されているかを示す真偽値の配列です。この配列の添え字はバンド番号と同じです。
- `extent` ラスタ集合における全てのラスタ行の範囲です。集合の範囲を変更するデータを別途ロードする予定である場合、ロード前に `DropRasterConstraints` 関数を実行して、ロード後に `AddRasterConstraints` で制約を再適用します。
- `spatial_index` 空間インデックスを持っているかどうかを示す真偽値です。

10.2.2 ラスタオーバビュー

`raster_overviews` は、オーバビューで使われるラスタテーブルカラムに関する情報のカタログで、オーバビューを用いる際に知っておくと便利な情報も持ちます。オーバビューテーブルは `raster_columns` と `raster_overviews` の両方のカタログに入れられます。オーバビューもラスタの一つであるのは確かですが、より高い解像度テーブルの解像度を落としたカリカチュアになるという特殊な目的を満たすためでもあるからです。ラスタをロードする際に `-l` スイッチを使うと、オーバビューが主ラスタテーブルと一緒に生成されます。もしくは、**AddOverviewConstraints**を使うと手動で生成できます。

オーバビューテーブルには、他のラスタテーブルと同じ制約と、オーバビュー特有の制約となる追加情報があります。



Note

`raster_overviews` の情報は `raster_columns` とは重複しません。`raster_columns` にあるオーバビューテーブルに関する情報が必要な場合は、`raster_overviews` と `raster_columns` とを結合すると、必要な情報の集合を完全に取得することができます。

オーバビューの主たる理由は次の二つです。

1. ズームアウトした際の地図表示を早くするために、元のテーブルの低解像度表現が一般的に使われます。
2. レコード数が少なく、ピクセル毎の適用範囲が広いため、高解像度の元テーブルより計算が一般的に早くなります。計算は高解像度テーブルより精度は落ちますが、大まかな計算には十分でありえます。

`raster_overviews` カタログには、次の情報のカラムがあります。

- `o_table_catalog` オーバビューテーブルが存在するデータベースです。常に現在のデータベースを読みます。
- `o_table_schema` オーバビューラスタテーブルが属するデータベーススキーマです。
- `o_table_name` ラスタオーバビューテーブル名です。
- `o_raster_column` オーバビューテーブル内のラスタカラムです。
- `r_table_catalog` このオーバビューの元となるラスタテーブルのデータベースです。常に現在のデータベースを読みます。
- `r_table_schema` このオーバビューの元となるラスタテーブルが属するデータベーススキーマです。
- `r_table_name` このオーバビューの元となるラスタテーブルです。
- `r_raster_column` このオーバビューの元となるラスタカラムです。
- `overview_factor` - オーバビューテーブルのピラミッドレベルです。高い数字ほど解像度が低くなります。`raster2pgsql` は、画像のフォルダを渡された場合は、分割して、イメージファイルのオーバビューの計算とロードを行います。レベル 1 は元ファイルと同じです。レベル 2 は、元ファイルの 4 分の 1 になります。たとえば、5000x5000 ピクセルの画像ファイルのフォルダがあるとして、125x125 に分ける場合、画像ファイルごとに $(5000 \times 5000) / (125 \times 125) = 1600$ 行のレコードを持ち、`o_2` テーブル (レベル 2) は $\text{ceiling}(1600 / \text{Power}(2, 2)) = 400$ 行、`o_3` (レベル 3) では $\text{ceiling}(1600 / \text{Power}(2, 3)) = 200$ 行のレコードを持ちます。ピクセルがタイルサイズで割り切れない場合、スクラップタイル (完全には値が入っていない) が得られます。`raster2pgsql` によって生成される個々のオーバビュータイルは、元となるラスタと同じピクセル数を持ち、個々のピクセルの表現範囲 (オリジナルの $\text{Power}(2, \text{overview\_factor})$ ピクセル分) が低い解像度になっている点に注意して下さい。

10.3 PostGIS ラスタを使ったカスタムアプリケーションの構築

PostGIS ラスタに知られている画像書式でラスタをレンダリングするための SQL 関数があります。このことから多数のレンダリングの選択肢が使えます。たとえば、OpenOffice/LibreOffice を使ってレンダリングが可能で、**Rendering PostGIS Raster graphics with LibreOffice Base Reports** にデモンストレーションがあります。さらに、本節で示す通り、多種多様な言語を使用できます。

10.3.1 ST_AsPNG を他の関数とあわせて使った PHP 出力例

本節では、PHP の PostgreSQL ドライバーと **ST_AsGDALRaster** 等の関数を使って、HTML `img` タグに埋め込むことができる PHP リクエストストリームにラスターの 1、2、3 バンドを出力する方法を示します。

サンプルクエリでは、指定した WGS84 バウンディングボックスにインタセクトするタイルを取って、**ST_Union** でインタセクトしたタイルを結合して全てのバンドを返し、**ST_Transform** でユーザー指定投影法に変換し、**ST_AsPNG** を使って PNG で結果を出力するためのラスタ関数群全体をまとめる方法を示します。

次で示すスクリプトは、

```
http://mywebserver/test_raster.php?srid=2249
```

で、マサチューセッツ州平面 (フィート単位) のラスタ画像を取得するものです。

```
<?php
/** contents of test_raster.php */
$conn_str = 'dbname=mydb host=localhost port=5432 user=myuser password=mypwd';
$dbconn = pg_connect($conn_str);
header('Content-Type: image/png');
/**If a particular projection was requested use it otherwise use mass state plane meters ←
**/
if (!empty( $_REQUEST['srid'] ) && is_numeric( $_REQUEST['srid'] )) {
    $input_srid = intval($_REQUEST['srid']);
}
else { $input_srid = 26986; }
/** The set bytea_output ensures bytea values are returned using escape format. */
$sql = "set bytea_output='escape';
SELECT ST_AsPNG(
    ST_Transform(
        ST_AddBand(
            ST_Union(rast, 1),
            ARRAY[ST_Union(rast, 2), ST_Union(rast, 3)]
        ),
        $input_srid
    )
) As new_rast
FROM aerials.boston
WHERE
    ST_Intersects(rast, ST_Transform(ST_MakeEnvelope(-71.1217, 42.227, -71.1210, ←
        42.218,4326),26986) )";
$result = pg_query($sql);
$row = pg_fetch_row($result);
pg_free_result($result);
if ($row === false) return;
echo pg_unescape_bytea($row[0]);
?>
```

10.3.2 ST_AsPNG を他の関数とあわせて使った ASP.NET C# 出力例

本節では、Npgsql PostgreSQL .NET ドライバーと **ST_AsGDALRaster** 等の関数を使って、HTML `img` タグに埋め込むことができるように、ラスターの 1、2、3 バンドを出力する方法を示します。

この例では Npgsql .NET PostgreSQL ドライバーが必要です。最新版は<http://npgsql.projects.postgresql.org/>にあります。最新版をダウンロードして、ASP.NET の bin フォルダに入れるだけでうまくいきます。

サンプルクエリでは、指定した WGS84 バウンディングボックスにインタセクトするタイルを取って、**ST_Union** でインタセクトしたタイルを結合して全てのバンドを返し、**ST_Transform** でユーザー指定投影法に変換し、**ST_AsPNG** を使って PNG で結果を出力するためのラスタ関数群全体をまとめる方法を示します。

この例は C# で実装している点を除いては Section 10.3.1 と同じです。

次で示すスクリプトは、

```
http://mywebserver/TestRaster.ashx?srid=2249
```

で、マサチューセッツ州平面 (フィート単位) のラスタ画像を取得します。

```
-- 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 raster クエリを画像ファイルで出力する Java コンソールアプリケーション

これは、一つの画像を返すクエリを取り、指定したファイルに出力する、簡単な Java コンソールアプリケーションです。

最新の PostgreSQL JDBC ドライバーは<http://jdbc.postgresql.org/download.html>からダウンロードできます。

あとで示すコードをコンパイルします。コマンドは次の通りです。

```

set env CLASSPATH ../\postgresql-9.0-801.jdbc4.jar
javac SaveQueryImage.java
jar cfm SaveQueryImage.jar Manifest.txt *.class

```

次のようにコマンドラインから呼び出します。

```

java -jar SaveQueryImage.jar "SELECT ST_AsPNG(ST_AsRaster(ST_Buffer(ST_Point(1,5),10, ' ↵
    quad_segs=2'),150, 150, '8BUI',100));" "test.png"

```

```

-- Manifest.txt --
Class-Path: postgresql-9.0-801.jdbc4.jar
Main-Class: SaveQueryImage

```

```

// Code for SaveQueryImage.java
import java.sql.Connection;
import java.sql.SQLException;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.io.*;

public class SaveQueryImage {
    public static void main(String[] argv) {
        System.out.println("Checking if Driver is registered with DriverManager.");
    }
}

```



```

try {
    //java.sql.DriverManager.registerDriver (new org.postgresql.Driver());
    Class.forName("org.postgresql.Driver");
}
catch (ClassNotFoundException cnfe) {
    System.out.println("Couldn't find the driver!");
    cnfe.printStackTrace();
    System.exit(1);
}

Connection conn = null;

try {
    conn = DriverManager.getConnection("jdbc:postgresql://localhost:5432/mydb","myuser ←
    ", "mypwd");
    conn.setAutoCommit(false);

    PreparedStatement sGetImg = conn.prepareStatement(argv[0]);

    ResultSet rs = sGetImg.executeQuery();

        FileOutputStream fout;
        try
        {
            rs.next();
            /** Output to file name requested by user */
            fout = new FileOutputStream(new File(argv[1]) );
            fout.write(rs.getBytes(1));
            fout.close();
        }
        catch(Exception e)
        {
            System.out.println("Can't create file");
            e.printStackTrace();
        }

        rs.close();
        sGetImg.close();
        conn.close();
    }
    catch (SQLException se) {
        System.out.println("Couldn't connect: print out a stack trace and exit.");
        se.printStackTrace();
        System.exit(1);
    }
}
}
}

```

10.3.4 PLPython を使って SQL を介して画像をダンプする

これは、サーバディレクトリ内でレコードごとにファイルを生成する Python ストアド関数です。plpython が必要です。plpythonu と plpython3u の両方が正しく動作します。

```

CREATE OR REPLACE FUNCTION write_file (param_bytes bytea, param_filepath text)
RETURNS text
AS $$
f = open(param_filepath, 'wb+')
f.write(param_bytes)
return param_filepath
$$ LANGUAGE plpythonu;

```

```
--write out 5 images to the PostgreSQL server in varying sizes
-- note the postgresql daemon account needs to have write access to folder
-- this echos back the file names created;
SELECT write_file(
    ST_AsPNG(ST_AsRaster(ST_Buffer(ST_Point(1,5),j*5, 'quad_segs=2'),150*j, 150*j, '8 BUI',100)),
    'C:/temp/slices'|| j || '.png')
FROM generate_series(1,5) As j;

write_file
-----
C:/temp/slices1.png
C:/temp/slices2.png
C:/temp/slices3.png
C:/temp/slices4.png
C:/temp/slices5.png
```

10.3.5 PSQL でラスタを出力する

PSQL から組み込み機能を用いてバイナリを出力するのは簡単ではありません。ここで紹介する方法は、レガシーなラージオブジェクトをサポートする PostgreSQL 上に乗っかる、ちょっとしたハックです。まずは、psql を起動して、データベースに接続します。

この方法は Python の場合と違い、ローカル機にファイルが生成されます。

```
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);
```

Chapter 11

ラストリファレンス

raster は、ラスタデータを格納、解析するための PostGIS 型です。

ラスタファイルからラスタをロードするには **Section 10.1**を参照して下さい。

このリファレンスで使うダミーデータのラスタテーブルの作り方は次の通りです:

```
CREATE TABLE dummy_rast(integer, rast raster);
INSERT INTO dummy_rast(rid, rast)
VALUES (1,
('01' -- little endian (uint8 ndr)
||
'0000' -- version (uint16 0)
||
'0000' -- nBands (uint16 0)
||
'000000000000000040' -- scaleX (float64 2)
||
'0000000000000000840' -- scaleY (float64 3)
||
'000000000000E03F' -- ipX (float64 0.5)
||
'000000000000E03F' -- ipY (float64 0.5)
||
'0000000000000000' -- skewX (float64 0)
||
'0000000000000000' -- skewY (float64 0)
||
'00000000' -- SRID (int32 0)
||
'0A00' -- width (uint16 10)
||
'1400' -- height (uint16 20)
)::rast
),
-- Raster: 5 x 5 pixels, 3 bands, PT_8BUI pixel type, NODATA = 0
(2, ('01000003009A9999999999A93F9A999999999A9BF000000E02B274A' ||
'410000000077195641000000000000000000000000000000000000000000000000 ←
FFFFFFFFFF050005000400FDFFDFEFDFEFDFEFDFEFDF9FAFEF' ||
' ←
EFCF9FBFDFFEFDFCFFAFEFEFE04004E627AADD16076B4F9FE6370A9F5FE59637AB0E54F58617087040046566487A1506
')::rast);
```

下に示す関数は **PostGIS** ラスタのユーザにとって必要そうなものです。他にも一般的なユーザにとって興味のないラスタ対応関数があります。

11.1 ラスタサポートデータ型

11.1.1 geomval

geomval — **geom** (ジオメトリオブジェクトを保持) と **val** (ラスタバンドからのピクセル値を倍精度浮動小数点数で保持) の 2 フィールドからなるデータ型。

説明

geomval は **geom** フィールドで参照されるジオメトリオブジェクトと、特定の地理位置におけるラスタバンドのピクセル値を表現する浮動小数点数の **val** からなる複合型です。ポリゴンにラスタバンドを展開するための出力型として、**ST_DumpAsPolygon** とラスタのインタセクション関連関数群に使われます。

関連情報

Section [13.6](#)

11.1.2 addbandarg

addbandarg — **ST_AddBand** の入力に使われる複合型で、新しいバンドの属性と初期値からなります。

説明

ST_AddBand の入力に使われる複合型で、新しいバンドの属性と初期値からなります。

index integer ラスタのバンド群のうち新しいバンドが追加される位置を示す 1 始まりの値。NULL の場合には、新しいバンドはラスタバンド群の末尾に追加されます。

pixeltype text 新しいバンドのピクセルタイプ。定義されたピクセルタイプは **ST_BandPixelType** に記述されています。

initialvalue double precision 新しいバンドに設定する初期値。

nodataval double precision 新しいバンドの NODATA 値。NULL の場合には、新しいバンドに NODATA 値を設定しません。

関連情報

[ST_AddBand](#)

11.1.3 rastbandarg

rastbandarg — ラスタとそのバンドインデックスを表現する必要がある時に使われる複合型。

説明

ラスタとそのバンドインデックスを表現する必要がある時に使われる複合型。

rast raster 問題にするラスタ。

nband integer ラスタのバンドを示す 1 始まりの値。

関連情報

ST_MapAlgebra (callback function version)

11.1.4 raster

raster — ラスタ空間データ型。

説明

raster は、JPEG, TIFF, PNG, デジタル標高モデルといったラスタデータを表現する空間データ型です。ラスタはそれぞれ 1 以上のバンドを持ち、バンドはそれぞれにピクセル値の集合を持ちます。ラスタには地理参照を与えることができます。



Note
PostGIS を GDAL サポート付きでコンパイルする必要があります。現在、ラスタは暗黙にジオメトリ型に変換することができますが、ラスタのST_ConvexHullを返します。この自動キャストは近いうちに削除するかも知れないので、この関数に頼らないようにして下さい。

キャストの挙動

本節では、このデータ型で許容される明示的なキャストと自動キャストの一覧を挙げます。

キャスト先	ふるまい
geometry	自動

関連情報

Chapter 11

11.1.5 reclassarg

reclassarg — ST_Reclass 関数への入力として使われる再分類のふるまいを定義する複合型です。

説明

ST_Reclass関数への入力として使われる再分類のふるまいを定義する複合型です。

nband integer 再分類対象バンドの番号。

reclassexpr text コンマ区切りの範囲表現。古い値から新しい値にマップする方法を定義します。'(' は「これより大きい」、')' は「これより小さい」、'[' は「これ以上」、']' は「これ以下」、をそれぞれ示します。

1. [a-b] = a <= x <= b
2. (a-b) = a < x <= b
3. [a-b) = a <= x < b
4. (a-b) = a < x < b

'(' 標記は必須ではありません。 'a-b' は '(a-b)' と同じ意味になります。

pixeltype text **ST_BandPixelType** の記述にあるピクセルタイプの一つ。

nodataval double precision NODATA として扱う値。透過が可能な画像出力では、この値は空白になります。

例: 2 番バンドを **8BUI** で **255** を **NODATA** に再分類

```
SELECT ROW(2, '0-100:1-10, 101-500:11-150,501 - 10000: 151-254', '8BUI', 255)::reclassarg;
```

例: 1 番バンドを **1BB** で **NODATA** を定義しないで再分類

```
SELECT ROW(1, '0-100]:0, (100-255:1', '1BB', NULL)::reclassarg;
```

関連情報

ST_Reclass

11.1.6 summarystats

summarystats — **ST_SummaryStats** 関数と **ST_SummaryStatsAgg** 関数の出力として使う複合型です。

説明

ST_SummaryStats 関数と **ST_SummaryStatsAgg** 関数の出力として使う複合型です。

count integer 統計情報で使ったピクセルの数。

sum double precision 統計情報で使ったピクセルの値の合計。

mean double precision 統計情報で使ったピクセルの値の平均。

stddev double precision 統計情報で使ったピクセルの値の標準偏差。

min double precision 統計情報で使ったピクセルの値の最小値。

max double precision 統計情報で使ったピクセルの値の最大値。

関連情報

ST_SummaryStats, **ST_SummaryStatsAgg**

11.1.7 unionarg

unionarg — **ST_Union** 関数の入力に使う複合型です。処理するバンドと結合処理の挙動を定義します。

説明

`ST_Union` 関数の入力に使う複合型です。処理するバンドと結合処理の挙動を定義します。

nband integer 処理する入力ラスタごとのバンドを示す 1 始まりの値です。

uniontype text 結合処理の種別。 **ST_Union** に記述がある、定義された種別のうち一つを取ります。

関連情報

ST_Union

11.2 ラスタ管理

11.2.1 AddRasterConstraints

`AddRasterConstraints` — ロードされたラスタテーブルの特定の列にラスタ制約を追加します。制約には空間参照系、スケール、ブロックサイズ、アラインメント、バンド数、バンド型、ラスタ列が規則正しいブロックかどうかを示すフラグがあります。テーブルは制約が推論されるためのデータがロードされなければなりません。制約の設定が完了すると `true` を返し、問題があると通知を返します。

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 );
```

説明

ラスタ列上に、ラスタカタログ `raster_columns` で情報を表示するために使われる制約を生成します。`rastschema` は、テーブルがあるデータベーススキーマの名前です。`srid` は `SPATIAL_REF_SYS` テーブル内のエントリを参照する整数でなければなりません。

`raster2pgsql` はこの関数を使ってラスタテーブルを登録します。

渡すのに妥当な制約名は次の通りです。詳細情報については [Section 10.2.1](#) を参照して下さい。

- `blocksize` ブロックの X と Y 両方のサイズを指定します
- `blocksize_x` X タイル (タイル毎のピクセル幅) を設定します
- `blocksize_y` Y タイル (タイル毎のピクセル幅) を設定します
- `extent` テーブル全体の範囲を計算し、全てのラスタがこの範囲内にある制約を適用します

- `num_bands` バンド数
- `pixel_types` バンドごとにピクセルタイプを読み、全てのバンドが同じピクセルタイプであることを確認します。
- `regular_blocking` 空間的に一意 (二つのラスタが空間的に同じにならない) であり、カバレッジタイル (ラスタがカバレッジに整列する) 制約であるようにします。
- `same_alignment` 同じアラインメントを持つようにします。任意の 2 タイルについて比較すると `true` を返すという意味です。 [ST_SameAlignment](#) を参照して下さい。
- `srid` 全て同じ SRID を持っていることを確認するためのものです。
- その他 -- 上の関数への入力で一覧が挙げられています。

**Note**

この関数はテーブル内に存在するデータから制約を推論します。動作させるには、まずラスタカラムを生成し、その後にデータをロードする必要があります。

**Note**

制約を適用した後にデータをさらにロードする必要がある場合には、データ範囲が変わるなら `DropRasterConstraints` を使います。

Availability: 2.0.0

例: カラムに対して全てのありえる制約をデータに基づいて適用する

```
CREATE TABLE myrasters(rid SERIAL primary key, rast raster);
INSERT INTO myrasters(rast)
SELECT ST_AddBand(ST_MakeEmptyRaster(1000, 1000, 0.3, -0.3, 2, 2, 0, 0,4326), 1, '8BSI'::↵
    text, -129, NULL);

SELECT AddRasterConstraints('myrasters'::name, 'rast'::name);

-- verify if registered correctly in the raster_columns view --
SELECT srid, scale_x, scale_y, blocksize_x, blocksize_y, num_bands, pixel_types, ↵
    nodata_values
FROM raster_columns
WHERE r_table_name = 'myrasters';
```

srid	scale_x	scale_y	blocksize_x	blocksize_y	num_bands	pixel_types	nodata_values
4326	2	2	1000	1000	1	{8BSI}	{0}

例: 単一の制約を適用する


```
CREATE TABLE public.myrasters2(rid SERIAL primary key, rast raster);
INSERT INTO myrasters2(rast)
SELECT ST_AddBand(ST_MakeEmptyRaster(1000, 1000, 0.3, -0.3, 2, 2, 0, 0,4326), 1, '8BSI':: ↵
    text, -129, NULL);

SELECT AddRasterConstraints('public'::name, 'myrasters2'::name, 'rast'::name, ' ↵
    regular_blocking', 'blocksize');
```

```
NOTICE: Adding regular blocking constraint
NOTICE: Adding blocksize-X constraint
NOTICE: Adding blocksize-Y constraint
```

関連情報

Section [10.2.1](#), [ST_AddBand](#), [ST_MakeEmptyRaster](#), [DropRasterConstraints](#), [ST_BandPixelType](#), [ST_SRID](#)

11.2.2 DropRasterConstraints

DropRasterConstraints — ラスタテーブルカラムを参照する PostGIS ラスタ制約を削除します。データの再読み込みやラスタカラムデータの更新の際に使えます。

Synopsis

```
boolean DropRasterConstraints(name rasttable, name rastcolumn, boolean srid, boolean scale_x,
boolean scale_y, boolean blocksize_x, boolean blocksize_y, boolean same_alignment, boolean regular_blocking,
boolean num_bands=true, boolean pixel_types=true, boolean nodata_values=true, boolean out_db=true,
boolean extent=true);
boolean DropRasterConstraints(name rastschema, name rasttable, name rastcolumn, boolean srid=true,
boolean scale_x=true, boolean scale_y=true, boolean blocksize_x=true, boolean blocksize_y=true,
boolean same_alignment=true, boolean regular_blocking=false, boolean num_bands=true, boolean pixel_types=true,
boolean nodata_values=true, boolean out_db=true, boolean extent=true);
boolean DropRasterConstraints(name rastschema, name rasttable, name rastcolumn, text[] constraints);
```

説明

AddRasterConstraintsで追加された、ラスタテーブルカラムを参照する PostGIS ラスタ制約を削除します。データの追加ロードやラスタカラムデータの更新で使えます。ラスタテーブルまたはラスタカラムの **rid** を取得したい場合には使う必要がありません。

ラスタテーブルを削除するには、次の標準的な SQL を使います。

```
DROP TABLE mytable
```

ラスタカラムを削除してテーブルの残りのカラムを置いておくには、次の標準的な SQL を使います。

```
ALTER TABLE mytable DROP COLUMN rast
```

カラムまたはテーブルが削除されると、テーブルは **raster_columns** カタログから見えなくなります。しかしながら、制約だけが削除されたので、ラスタカラムはなお **raster_columns** カタログに残ります。ただし、カラム名とテーブルからの内部に関する他の情報は存在しません。

Availability: 2.0.0

例

```
SELECT DropRasterConstraints ('myrasters','rast');
----RESULT output ---
t

-- verify change in raster_columns --
SELECT srid, scale_x, scale_y, blocksize_x, blocksize_y, num_bands, pixel_types, ↵
      nodata_values
FROM raster_columns
WHERE r_table_name = 'myrasters';
```

srid	scale_x	scale_y	blocksize_x	blocksize_y	num_bands	pixel_types	↵
0							

関連情報

[AddRasterConstraints](#)

11.2.3 AddOverviewConstraints

AddOverviewConstraints — ラスタカラムに対して、他のオーバビューであることをタグ付けします。

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);

説明

raster_overviews ラスタカカタログ内の情報を表示するために使われるラスタカラムに制約を追加します。
ovfactor 引数は、オーバビューカラムの縮尺乗数を示します。縮尺乗数は大きいほど低い解像度になります。
ovschema と **refschema** が省略されると、**search_path** を走査して発見した最初のテーブルを使います。
Availability: 2.0.0

例

```
CREATE TABLE res1 AS SELECT
ST_AddBand(ST_MakeEmptyRaster(1000, 1000, 0, 0, 2),
1, '8BSI'::text, -129, NULL
) r1;

CREATE TABLE res2 AS SELECT
ST_AddBand(ST_MakeEmptyRaster(500, 500, 0, 0, 4),
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)

```

関連情報

Section [10.2.2, DropOverviewConstraints](#), [ST_CreateOverview](#), [AddRasterConstraints](#)

11.2.4 DropOverviewConstraints

DropOverviewConstraints — ラスタカラムに対して他のオーバビューであることをタグ付けしているのを解除します。

Synopsis

```

boolean DropOverviewConstraints(name ovschema, name ovtable, name ovcolumn);
boolean DropOverviewConstraints(name ovtable, name ovcolumn);

```

説明

ラスタカラムから、`raster_overviews` ラスタカタログ内にある、他のラスタのオーバビューであることをと示すために使われる制約を削除します。

`ovschema` が省略されると、`search_path` を走査して発見した最初のテーブルを使います。

Availability: 2.0.0

関連情報

Section [10.2.2, AddOverviewConstraints](#), [DropRasterConstraints](#)

11.2.5 PostGIS_GDAL_Version

PostGIS_GDAL_Version — PostGIS で使っている GDAL ライブラリの版を報告します。

Synopsis

```

text PostGIS_GDAL_Version();

```

説明

PostGIS で使っている GDAL ライブラリの版を報告します。GDAL がデータファイルを発見できるかについてもチェックして、報告します。

例

```
SELECT PostGIS_GDAL_Version();
```

```
GDAL 1.11dev, released 2013/04/13
```

関連情報

[postgis.gdal_datapath](#)

11.2.6 PostGIS_Raster_Lib_Build_Date

PostGIS_Raster_Lib_Build_Date — 完全なラスタライブラリをビルドした日付を報告します。

Synopsis

```
text PostGIS_Raster_Lib_Build_Date();
```

説明

ラスタをビルドした日付を報告します。

例

```
SELECT PostGIS_Raster_Lib_Build_Date();
```

```
2010-04-28 21:15:10
```

関連情報

[PostGIS_Raster_Lib_Version](#)

11.2.7 PostGIS_Raster_Lib_Version

PostGIS_Raster_Lib_Version — 完全なラスタの版とビルドコンフィギュレーション情報を報告します。

Synopsis

```
text PostGIS_Raster_Lib_Version();
```

説明

完全なラスタの版とビルドコンフィギュレーション情報を報告します。

例

```
SELECT PostGIS_Raster_Lib_Version();
```

```
2.0.0
```

関連情報

[PostGIS_Lib_Version](#)

11.2.8 ST_GDALDrivers

ST_GDALDrivers — 使用している GDAL ライブラリが対応するラスタ書式の一覧を返します。この一覧で `can_write=True` となっているものだけが **ST_AsGDALRaster** で使えます。

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);

説明

使用している GDAL ライブラリが対応するラスタ書式の `short_name`, `long_name` と作成オプションの一覧を返します。`short_name` は、**ST_AsGDALRaster** の `format` パラメーターに使います。オプションは GDAL ライブラリのコンパイルに使ったドライバーに依存します。`create_options` は、XML 形式の `CretionOptionList/Option` の集合で、ドライバーごとの生成オプションごとに、名前を持ち、追加情報の `type`, `description` と `VALUE` の集合を持ちます。

Changed: 2.5.0 - `can_read` カラムと `can_write` カラムを追加。

Changed: 2.0.6, 2.1.3 - GUC (訳注: Grand Unified Configuration の略で、動的に変更できる PostgreSQL パラメーター) または環境変数 `gdal_enabled_drivers` が設定されていないデフォルトではドライバーが全て無効になりました。

Availability: 2.0.0 - GDAL 1.6.0 以上が必要です。

例: ドライバー一覧

```
SET postgis.gdal_enabled_drivers = 'ENABLE_ALL';
SELECT short_name, long_name, can_write
FROM ST_GDALDrivers()
ORDER BY short_name;
```

short_name	long_name	can_write
AAIGrid	Arc/Info ASCII Grid	t
ACE2	ACE2	f
ADRG	ARC Digitized Raster Graphics	f
AIG	Arc/Info Binary Grid	f
AirSAR	AirSAR Polarimetric Image	f
ARG	Azavea Raster Grid format	t
BAG	Bathymetry Attributed Grid	f
BIGGIF	Graphics Interchange Format (.gif)	f
BLX	Magellan topo (.blx)	t
BMP	MS Windows Device Independent Bitmap	f
BSB	Maptech BSB Nautical Charts	f
PAux	PCI .aux Labelled	f
PCIDSK	PCIDSK Database File	f
PCRaster	PCRaster Raster File	f
PDF	Geospatial PDF	f
PDS	NASA Planetary Data System	f
PDS4	NASA Planetary Data System 4	t
PLMOSAIC	Planet Labs Mosaics API	f
PLSCENES	Planet Labs Scenes API	f
PNG	Portable Network Graphics	t
PNM	Portable Pixmap Format (netpbm)	f
PRF	Racurs PHOTOMOD PRF	f
R	R Object Data Store	t
Rasterlite	Rasterlite	t
RDA	DigitalGlobe Raster Data Access driver	f
RIK	Swedish Grid RIK (.rik)	f
RMF	Raster Matrix Format	f
ROI_PAC	ROI_PAC raster	f
RPFTOC	Raster Product Format TOC format	f
RRASTER	R Raster	f
RS2	RadarSat 2 XML Product	f
RST	Idrisi Raster A.1	t
SAFE	Sentinel-1 SAR SAFE Product	f
SAGA	SAGA GIS Binary Grid (.sdatt, .sg-grd-z)	t
SAR_CE0S	CE0S SAR Image	f
SDTS	SDTS Raster	f
SENTINEL2	Sentinel 2	f
SGI	SGI Image File Format 1.0	f
SNODAS	Snow Data Assimilation System	f
SRP	Standard Raster Product (ASRP/USRP)	f
SRTMHGT	SRTMHGT File Format	t
Terragen	Terragen heightfield	f
TIL	EarthWatch .TIL	f
TSX	TerraSAR-X Product	f
USGSDem	USGS Optional ASCII DEM (and CDED)	t
VICAR	MIPL VICAR file	f
VRT	Virtual Raster	t
WCS	OGC Web Coverage Service	f
WMS	OGC Web Map Service	t
WMTS	OGC Web Map Tile Service	t
XPM	X11 PixMap Format	t
XYZ	ASCII Gridded XYZ	t
ZMap	ZMap Plus Grid	t

例: ドライバー毎のオプション一覧

Output the create options XML column of JPEG as a table. These creator options can be used in the ST_AsGDALRaster options argument.

```
SELECT (xpath('@name', g.opt))[1]::text As oname,
       (xpath('@type', g.opt))[1]::text As otype,
       (xpath('@description', g.opt))[1]::text As descrip
FROM (SELECT unnest(xpath('/CreationOptionList/Option', create_options::xml)) As opt
FROM ST_GDALDrivers()
WHERE short_name = 'JPEG') As g;
```

oname	otype	descrip
PROGRESSIVE	boolean	whether to generate a progressive JPEG
QUALITY	int	good=100, bad=0, default=75
WORLDFILE	boolean	whether to generate a worldfile
INTERNAL_MASK	boolean	whether to generate a validity mask
COMMENT	string	Comment
SOURCE_ICC_PROFILE	string	ICC profile encoded in Base64
EXIF_THUMBNAIL	boolean	whether to generate an EXIF thumbnail(overview). By default its max dimension will be 128
THUMBNAIL_WIDTH	int	Forced thumbnail width
THUMBNAIL_HEIGHT	int	Forced thumbnail height

(9 rows)

This example returns raw XML output for GeoTIFF creator options.

```
SELECT create_options
FROM ST_GDALDrivers()
WHERE short_name = 'GTiff';

<CreationOptionList>
  <Option name="COMPRESS" type="string-select">
    <Value
>NONE</Value>
    <Value
>LZW</Value>
    <Value
>PACKBITS</Value>
    <Value
>JPEG</Value>
    <Value
>CCITTRLE</Value>
    <Value
>CCITTFAX3</Value>
    <Value
>CCITTFAX4</Value>
    <Value
>DEFLATE</Value>
  </Option>
  <Option name="PREDICTOR" type="int" description="Predictor Type"/>
  <Option name="JPEG_QUALITY" type="int" description="JPEG quality 1-100" default="75"/>
  <Option name="ZLEVEL" type="int" description="DEFLATE compression level 1-9" default ↵
    ="6"/>
  <Option name="NBITS" type="int" description="BITS for sub-byte files (1-7), sub-uint16 ↵
    (9-15), sub-uint32 (17-31)"/>
  <Option name="INTERLEAVE" type="string-select" default="PIXEL">
    <Value
>BAND</Value>
    <Value
```

```
>PIXEL</Value>
  </Option>
  <Option name="TILED" type="boolean" description="Switch to tiled format"/>
  <Option name="TFW" type="boolean" description="Write out world file"/>
  <Option name="RPB" type="boolean" description="Write out .RPB (RPC) file"/>
  <Option name="BLOCKXSIZE" type="int" description="Tile Width"/>
  <Option name="BLOCKYSIZE" type="int" description="Tile/Strip Height"/>
  <Option name="PHOTOMETRIC" type="string-select">
    <Value
>MINISBLACK</Value>
    <Value
>MINISWHITE</Value>
    <Value
>PALETTE</Value>
    <Value
>RGB</Value>
    <Value
>CMYK</Value>
    <Value
>YCBCR</Value>
    <Value
>CIELAB</Value>
    <Value
>ICCLAB</Value>
    <Value
>ITULAB</Value>
  </Option>
  <Option name="SPARSE_OK" type="boolean" description="Can newly created files have ↵
    missing blocks?" default="FALSE"/>
  <Option name="ALPHA" type="boolean" description="Mark first extrasample as being alpha ↵
    "/>
  <Option name="PROFILE" type="string-select" default="GDALGeoTIFF">
    <Value
>GDALGeoTIFF</Value>
    <Value
>GeoTIFF</Value>
    <Value
>BASELINE</Value>
  </Option>
  <Option name="PIXELTYPE" type="string-select">
    <Value
>DEFAULT</Value>
    <Value
>SIGNEDBYTE</Value>
  </Option>
  <Option name="BIGTIFF" type="string-select" description="Force creation of BigTIFF file ↵
    ">
    <Value
>YES</Value>
    <Value
>NO</Value>
    <Value
>IF_NEEDED</Value>
    <Value
>IF_SAFER</Value>
  </Option>
  <Option name="ENDIANNESS" type="string-select" default="NATIVE" description="Force ↵
    endianness of created file. For DEBUG purpose mostly">
    <Value
>NATIVE</Value>
    <Value
>INVERTED</Value>
```



```

    <Value
>LITTLE</Value>
    <Value
>BIG</Value>
  </Option>
  <Option name="COPY_SRC_OVERVIEWS" type="boolean" default="NO" description="Force copy ↵
    of overviews of source dataset (CreateCopy())"/>
</CreationOptionList>

```

```

-- Output the create options XML column for GTiff as a table --
SELECT (xpath('@name', g.opt))[1]::text As oname,
       (xpath('@type', g.opt))[1]::text As otype,
       (xpath('@description', g.opt))[1]::text As descrip,
       array_to_string(xpath('Value/text()', g.opt),', ') As vals
FROM (SELECT unnest(xpath('/CreationOptionList/Option', create_options::xml)) As opt
FROM ST_GDALDrivers()
WHERE short_name = 'GTiff') As g;

```

oname	otype	descrip ↵ vals
COMPRESS	string-select	↵ NONE, LZW, ↵ PACKBITS, JPEG, CCITTRLE, CCITTFAX3, CCITTFAX4, DEFLATE
PREDICTOR	int	Predictor Type ↵
JPEG_QUALITY	int	JPEG quality 1-100 ↵
ZLEVEL	int	DEFLATE compression level 1-9 ↵
NBITS	int	BITS for sub-byte files (1-7), sub-uint16 (9-15), sub ↵
INTERLEAVE	string-select	↵ BAND, PIXEL
TILED	boolean	Switch to tiled format ↵
TFW	boolean	Write out world file ↵
RPB	boolean	Write out .RPB (RPC) file ↵
BLOCKXSIZE	int	Tile Width ↵
BLOCKYSIZE	int	Tile/Strip Height ↵
PHOTOMETRIC	string-select	↵ MINISBLACK, ↵ MINISWHITE, PALETTE, RGB, CMYK, YCBCR, CIELAB, ICCLAB, ITULAB
SPARSE_OK	boolean	Can newly created files have missing blocks? ↵
ALPHA	boolean	Mark first extrasample as being alpha ↵
PROFILE	string-select	↵ GDALGeoTIFF, ↵ GeoTIFF, BASELINE
PIXELTYPE	string-select	↵ DEFAULT, ↵ SIGNEDBYTE
BIGTIFF	string-select	Force creation of BigTIFF file ↵ YES, NO, IF_NEEDED, IF_SAFER
ENDIANNESS	string-select	Force endianness of created file. For DEBUG purpose ↵ mostly NATIVE, INVERTED, LITTLE, BIG

```
COPY_SRC_OVERVIEWS | boolean | Force copy of overviews of source dataset (CreateCopy ←
  ()) |
(19 rows)
```

関連情報

[ST_AsGDALRaster](#), [ST_SRID](#), [postgis.gdal_enabled_drivers](#)

11.2.9 UpdateRasterSRID

`UpdateRasterSRID` — ユーザが指定したカラムとテーブルにあるラスタの全てについて SRID を変更します。

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);
```

説明

ユーザが指定したカラムとテーブルにあるラスタの全てについて SRID を変更します。この関数は全ての適切なカラム制約 (extent, alignment, SRID) を削除してから、指定したカラムのラスタの SRID を変更します。



Note

この関数はラスタのデータ (バンドピクセル値) を触りません。ラスタのメタデータのみ変更します。

Availability: 2.1.0

関連情報

[UpdateGeometrySRID](#)

11.2.10 ST_CreateOverview

`ST_CreateOverview` — 与えられたラスタカバレッジから解像度を落としたものを生成します。

Synopsis

```
regclass ST_CreateOverview(regclass tab, name col, int factor, text algo='NearestNeighbor');
```

説明

元のテーブルからリサンプリングを施したタイルのオーバビューテーブルを生成します。出力タイルは入力タイルと同じサイズで、同じ空間範囲を持ち、入力タイルより低い解像度 (ピクセル数は縦横ともに元のタイルの $1/\text{factor}$ 倍になります) を持ちます。

オーバビューテーブルは `raster_overviews` カタログに出現し、ラスタ制約を持ちます。

アルゴリズム指定オプションは 'NearestNeighbor', 'Bilinear', 'Cubic', 'CubicSpline', 'Lanczos' です。詳細については [GDAL Warp resampling methods](#) をご覧ください。

Availability: 2.2.0

例

高品質ですが遅い生成書式での出力

```
SELECT ST_CreateOverview('mydata.mytable'::regclass, 'rast', 2, 'Lanczos');
```

デフォルトの最近傍補間を使った早い処理による出力

```
SELECT ST_CreateOverview('mydata.mytable'::regclass, 'rast', 2);
```

関連情報

[ST_Retile](#), [AddOverviewConstraints](#), [AddRasterConstraints](#), [Section 10.2.2](#)

11.3 ラスタコンストラクタ

11.3.1 ST_AddBand

`ST_AddBand` — 与えられたタイプで、与えられた初期値にした新しいバンドを、与えられたインデックス位置に追加したラスタを返します。インデックス位置を指定していない場合には、バンドは末尾に追加されます。

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);

説明

指定した位置 (index) に、指定したタイプの、指定した初期値を持ち、指定した NODATA 値を持つ、追加された新しいバンドを持つラスタを返します。インデックス位置を指定していない場合には、バンドは末尾に追加されます。**fromband** が指定されない場合には、1 番バンドと仮定します。ピクセルタイプは **ST_BandPixelType** で指定されているピクセルタイプの文字列表現です。既存のインデックスが指定された場合には、バンドのインデックス $\geq$ 指定インデックスとなるバンドのインデックスは 1 ずつ足されます。初期値としてピクセルタイプの最大値を超えた値が指定された場合には、初期値にピクセルタイプの許容最大値が指定されます。

addbandarg の配列を取る版 (一つ目の版) では、指定した **addbandarg** のインデックス値は、**addbandarg** で示されるバンドがラスタに追加される時のラスタとの相対値です。下の複数バンドの例を参照してください。

ラスタの配列を取る版 (五つ目の版) では、**torast** が NULL なら、配列内のラスタごとの **fromband** のバンドが新しいラスタに集約されます。

outdbfile を取る版 (六つ目と七つ目の版) では、値はラスタファイルへのフルパスを含まなければなりません。また、ファイルは PostgreSQL サーバプロセスがアクセス可能でなければなりません。

Enhanced: 2.1.0 **addbandarg** 対応が追加されました。

Enhanced: 2.1.0 out-db バンドが追加されました。

例: 単一バンド

This example adds another 8-bit unsigned integer band 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;
```

pixeltype	nodatavalue	isoutdb	path
1BB		f	
4BUI		f	

This example outputs metadata for the raster.

```
SELECT (rmd).width, (rmd).height, (rmd).numbands
FROM (SELECT ST_MetaData(rast) As rmd
FROM dummy_rast WHERE rid = 10) AS foo;
```

upperleftx	upperlefty	width	height	scalex	scaley	skewx	skewy	srid	↔
numbands									
0	0	100	100	1	-1	0	0	0	↔
2									

例: 複数の新規バンド

```
SELECT
*
FROM ST_BandMetaData(ST_AddBand(ST_MakeEmptyRaster(10, 10, 0, 0, 1, -1, 0, 0, 0),
ARRAY[
ROW(NULL, '8BUI', 255, 0),
ROW(NULL, '16BUI', 1, 2),
ROW(2, '32BUI', 100, 12),
ROW(2, '32BF', 3.14, -1)
]::addbandarg[]
),
ARRAY[]::integer[]
);
```

bandnum	pixeltype	nodatavalue	isoutdb	path	outdbbandnum	filesize	↔
filetimestamp							
1	8BUI	0	f	null	null	null	↔
null							
2	32BF	-1	f	null	null	null	↔
null							
3	32BUI	12	f	null	null	null	↔
null							
4	16BUI	2	f	null	null	null	↔
null							

Aggregate the 1st band of a table of like rasters into a single raster. With as many bands as there are test_types and as many rows (new rasters) as there are mice. Note that The ORDER BY test_type clause relies on aggregate ORDER BY support. The resulting raster will have a band for each test_type alphabetical by test_type. For mouse lovers: No mice were harmed in this exercise.

```
SELECT
mouse,
ST_AddBand(NULL, array_agg(rast ORDER BY test_type), 1) As rast
FROM mice_studies
GROUP BY mouse;
```

例: データベース外のバンド

```
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

関連情報

[ST_BandMetaData](#), [ST_BandPixelType](#), [ST_MakeEmptyRaster](#), [ST_MetaData](#), [ST_NumBands](#), [ST_Reclass](#)

11.3.2 ST_AsRaster

ST_AsRaster — PostGIS ジオメトリを PostGIS ラスタに変換します。

Synopsis

raster **ST_AsRaster**(geometry geom, raster ref, text pixeltype, double precision value=1, double precision nodataval=0, boolean touched=false);

raster **ST_AsRaster**(geometry geom, raster ref, text[] pixeltype=ARRAY['8BUI'], double precision[] value=ARRAY[1], double precision[] nodataval=ARRAY[0], boolean touched=false);

raster **ST_AsRaster**(geometry geom, double precision scalex, double precision scaley, double precision gridx, double precision gridy, text pixeltype, double precision value=1, double precision nodataval=0, double precision skewx=0, double precision skewy=0, boolean touched=false);

raster **ST_AsRaster**(geometry geom, double precision scalex, double precision scaley, double precision gridx=NULL, double precision gridy=NULL, text[] pixeltype=ARRAY['8BUI'], double precision[] value=ARRAY[1], double precision[] nodataval=ARRAY[0], double precision skewx=0, double precision skewy=0, boolean touched=false);

raster **ST_AsRaster**(geometry geom, double precision scalex, double precision scaley, text pixeltype, double precision value=1, double precision nodataval=0, double precision upperleftx=NULL, double precision upperlefty=NULL, double precision skewx=0, double precision skewy=0, boolean touched=false);

raster **ST_AsRaster**(geometry geom, double precision scalex, double precision scaley, text[] pixeltype, double precision[] value=ARRAY[1], double precision[] nodataval=ARRAY[0], double precision upperleftx=NULL, double precision upperlefty=NULL, double precision skewx=0, double precision skewy=0, boolean touched=false);

raster **ST_AsRaster**(geometry geom, integer width, integer height, double precision gridx, double precision gridy, text pixeltype, double precision value=1, double precision nodataval=0, double precision skewx=0, double precision skewy=0, boolean touched=false);

raster **ST_AsRaster**(geometry geom, integer width, integer height, double precision gridx=NULL, double precision gridy=NULL, text[] pixeltype=ARRAY['8BUI'], double precision[] value=ARRAY[1], double precision[] nodataval=ARRAY[0], double precision skewx=0, double precision skewy=0, boolean touched=false);

raster **ST_AsRaster**(geometry geom, integer width, integer height, text pixeltype, double precision value=1, double precision nodataval=0, double precision upperleftx=NULL, double precision upperlefty=NULL, double precision skewx=0, double precision skewy=0, boolean touched=false);

raster **ST_AsRaster**(geometry geom, integer width, integer height, text[] pixeltype, double precision[] value=ARRAY[1], double precision[] nodataval=ARRAY[0], double precision upperleftx=NULL, double precision upperlefty=NULL, double precision skewx=0, double precision skewy=0, boolean touched=false);

説明

PostGIS ジオメトリを PostGIS ラスタに変換します。多数の形式がありますが、結果ラスタのアラインメントとピクセルサイズを設定する三つの考えられる類型に分かれます。

一つ目の群は、最初の 2 形式です。アラインメント (`scalex`, `scaley`, `gridx`, `gridy`)、ピクセルタイプ、NODATA 値が提供される参照ラスタと同じとなるラスタを生成します。一般的にこの参照ラスタは、ジオメトリを含むテーブルを参照ラスタを含むテーブルに結合して渡します。

二つ目の群は、4 形式あります。ピクセルサイズ (`scalex` & `scaley` と `skewx` & `skewy`) の引数を渡してラスタのピクセル範囲を設定します。結果ラスタの `width` & `height` はジオメトリの範囲にあわせて調整されます。ほとんどの場合、PostgreSQL が正しい形式を選択するために、`scalex` & `scaley` の整数引数を倍精度浮動小数点数にキャストしなければなりません。

三つ目の群は、4 形式あります。ラスタのピクセル範囲 (`width` & `height`) を渡してラスタのピクセル範囲を固定するものです。結果ラスタのピクセルサイズ (`scalex` & `scaley` と `skewx` & `skewy`) の引数はジオメトリの範囲にあわせて調整されます。

二つ目の群と三つ目の群の群内前半 2 形式は、アラインメントグリッド (`gridx` & `gridy`) の適切な隅でアラインメントを特定します。群内後半 2 形式は左上隅 (`upperleftx` & `upperlefty`) を取ります。

これらの群のそれぞれによって、1 バンドまたは複数バンドのラスタを生成することができます。複数バンドのラスタを生成するには、ピクセルタイプ配列 (`pixeltype[]`)、初期値配列 (`value`)、NODATA 値配列 (`nodataval`) を渡す必要があります。ピクセルタイプが渡されていない場合には、デフォルトは 8BUI になり、初期値については 1、NODATA 値については 0 になります。

出力ラスタは元のジオメトリと同じ空間参照系になります。参照ラスタを使う形式は例外で、結果ラスタは参照ラスタと同じ SRID になります。

任意引数 `touched` は、デフォルトでは FALSE になります。GDAL の ALL_TOUCHED ラスタ化オプションに相当し、ラインまたはポリゴンに接触するピクセルが描画されます。ライン上にあるか中心点がポリゴン内部にあるピクセルが描画されるだけではありません。

これは特にジオメトリをデータベースから **ST_AsPNG** や **ST_AsGDALRaster** 系の関数を併用して直接 JPEG や PNG に描画する際に使えます。

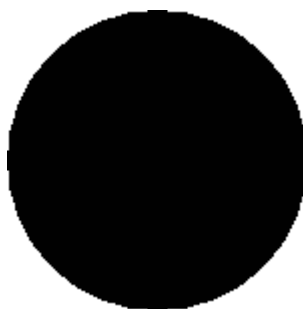
Availability: 2.0.0 - GDAL 1.6.0 以上が必要です。



Note

曲線、TIN、多面体サーフェスのような複雑なジオメトリタイプのレンダリングは、まだできませんが、GDAL ができることは実現できます。

例: ジオメトリを **PNG** ファイルとして出力



黒い円

This example outputs a black circle occupying 150 by 150 pixels.

```
SELECT ST_AsPNG(ST_AsRaster(ST_Buffer(ST_Point(1,5),10),150, 150));
```



PostGIS だけで描画したバッファの例

This example maps bands to RGB channels, producing the teal value (118,154,118).

```
SELECT ST_AsPNG(ST_AsRaster(ST_Buffer(ST_GeomFromText('LINESTRING(50 50,150 150,150 50)'), ←
    10,'join=bevel'),
    200,200,ARRAY['8BUI', '8BUI', '8BUI'], ARRAY[118,154,118], ARRAY[0,0,0]));
```

関連情報

[ST_BandPixelType](#), [ST_Buffer](#), [ST_GDALDrivers](#), [ST_AsGDALRaster](#), [ST_AsPNG](#), [ST_AsJPEG](#), [ST_SRID](#)

11.3.3 ST_AsRasterAgg

ST_AsRasterAgg — 集約関数です。PostGIS ジオメトリを新しいラスタに変換します。

Synopsis

raster **ST_AsRasterAgg**(geometry geom, double precision val, raster ref, text pixeltype, double precision nodataval, text uniontype, boolean touched);

説明

入力ジオメトリとそれに関連付けた値を使ってラスタに変換したもの全てを持つ単一バンドのラスタを返します。

Availability: 3.6.0

例

```
WITH inp(g,v) AS (
    VALUES
        ( ST_Buffer(ST_MakePoint(10,0), 10), 1 ),
        ( ST_Buffer(ST_MakePoint(20,0), 10), 2 )
),
agg AS (
```



```

SELECT ST_AsRasterAgg(
    g,
    v,
    ST_MakeEmptyRaster(0,0,0,0,1.0),
    '8BUI',
    99,
    'SUM',
    true
) r
FROM inp
)
SELECT
    ST_Width(r) w,
    ST_Height(r) h,
    ST_Value(r,'POINT(5 0)') v5_0,
    ST_Value(r,'POINT(15 0)') v15_0,
    ST_Value(r,'POINT(25 0)') v25_0
FROM agg;

```

w	h	v5_0	v15_0	v25_0
30	20	1	3	2

(1 row)

関連情報

[ST_AsRaster](#), [ST_DumpAsPolygons](#), [ST_Union](#)

11.3.4 ST_Band

ST_Band — 既存のラスタの、一つ以上のバンドを新しいラスタとして返します。既存のラスタから新しいラスタを構築する際に使えます。

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=,);

```

説明

既存のラスタの、一つ以上のバンドを新しいラスタとして返します。既存ラスタから新しいラスタを構築したり、ラスタの選択したバンドのみを出力したり、バンドの並びを改める際に使えます。バンドが指定されない場合や指定したバンドがラスタに存在しない場合には、全てのバンドを返します。バンド削除等の様々な関数を補助する関数として使われています。



Warning

nbands を文字列とする版では、デフォルトのデリミタは、です。'1,2,3' と指定できます。異なるデリミタを使いたい場合には、`ST_Band(rast, '1@2@3', '@')` とします。複数バンドを指定する際に、`ST_Band(rast, '{1,2,3}'::int[]);` というような、配列を使うことを強くお勧めします。text によるバンド一覧を取る形式は、PostGIS の将来の版で削除するかも知れません。

Availability: 2.0.0

例

This example creates two rasters: one containing band 1 of `dummy_rast`, and one containing band 2 reclassified as 2BUI.

```
SELECT ST_NumBands(rast1) As numb1, ST_BandPixelType(rast1) As pix1,
       ST_NumBands(rast2) As numb2, ST_BandPixelType(rast2) As pix2
FROM (
  SELECT ST_Band(rast) As rast1, ST_Reclass(ST_Band(rast,3), '100-200':1, [200-254:2', '2 ←
    BUI') As rast2
    FROM dummy_rast
    WHERE rid = 2) As foo;
```

numb1	pix1	numb2	pix2
1	8BUI	1	2BUI

Return bands 2 and 3 using array cast syntax.

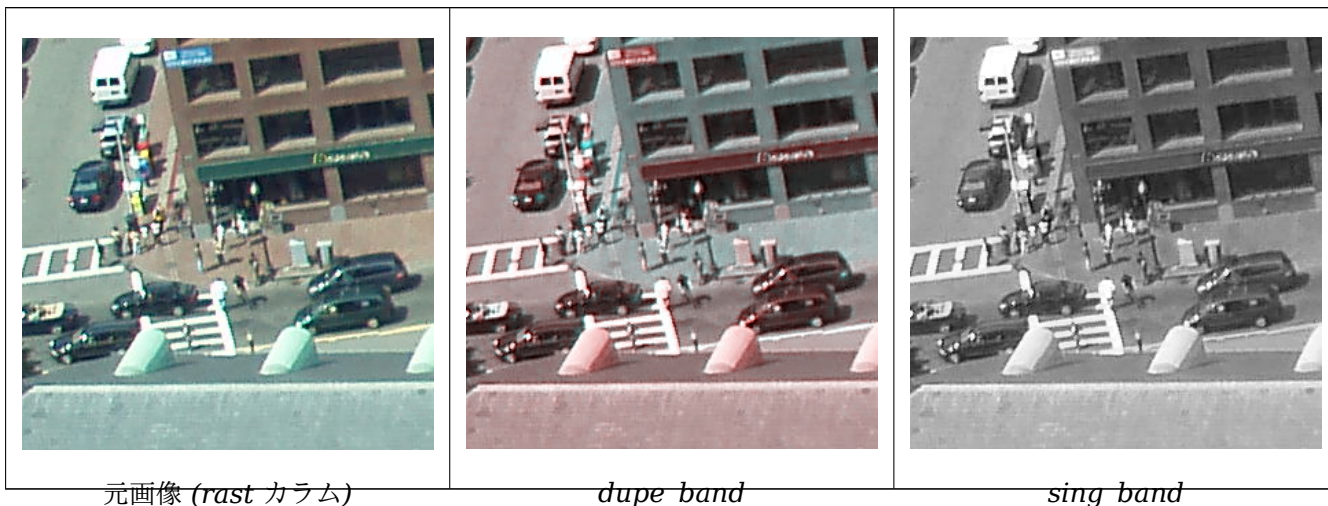
```
SELECT ST_NumBands(ST_Band(rast, '{2,3}'::int[])) As num_bands
FROM dummy_rast WHERE rid=2;
```

2

Return bands 2 and 3 using array syntax.

```
SELECT ST_NumBands(ST_Band(rast, ARRAY[2,3])) As num_bands
FROM dummy_rast
WHERE rid=2;
```

2



```
--Make a new raster with 2nd band of original and 1st band repeated twice,
and another with just the third band
SELECT rast, ST_Band(rast, ARRAY[2,1,1]) As dupe_band,
       ST_Band(rast, 3) As sing_band
FROM samples.than_chunked
WHERE rid=35;
```

関連情報

[ST_AddBand](#), [ST_NumBands](#), [ST_Reclass](#), [Chapter 11](#)

11.3.5 ST_MakeEmptyCoverage

`ST_MakeEmptyCoverage` — 空のラスタタイルのグリッドでジオリファレンスを施されている領域を生成します。

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);

説明

ST_MakeEmptyRasterでラスタ集合を生成します。グリッドの次元は `width` と `height` です。タイルの次元は `tilewidth` と `tileheight` です。領域は左上隅 (`upperleftx`, `upperlefty`) から右下隅 (`upperleftx + width * scalex`, `upperlefty + height * scaley`) で形成されます。



Note

ラスタの `scaley` は一般的に負数で、`scalex` は一般的に正数です。右下隅は、左上隅より、`y` は小さな値、`x` は大きな値になります。

Availability: 2.4.0

基本的な例

WGS84 で左上が (22, 27) で右下が (55, 53) となる範囲の 4x4 グリッドの 16 タイルを生成します。

```
SELECT (ST_MetaData(tile)).* FROM ST_MakeEmptyCoverage(1, 1, 4, 4, 22, 33, (55 - 22)/(4)::float, (33 - 53)/(4)::float, 0., 0., 4326) tile;
```

upperleftx upperlefty width height scalex scaley skewx skewy srid ↩	numbands
22 33 1 1 8.25 -11 0 0 4326 ↩	0
30.25 33 1 1 8.25 -11 0 0 4326 ↩	0
38.5 33 1 1 8.25 -11 0 0 4326 ↩	0
46.75 33 1 1 8.25 -11 0 0 4326 ↩	0
22 22 1 1 8.25 -11 0 0 4326 ↩	0
30.25 22 1 1 8.25 -11 0 0 4326 ↩	0
38.5 22 1 1 8.25 -11 0 0 4326 ↩	0

46.75		22		1		1		8.25		-11		0		0		4326		↔
		0																
22		11		1		1		8.25		-11		0		0		4326		↔
		0																
30.25		11		1		1		8.25		-11		0		0		4326		↔
		0																
38.5		11		1		1		8.25		-11		0		0		4326		↔
		0																
46.75		11		1		1		8.25		-11		0		0		4326		↔
		0																
22		0		1		1		8.25		-11		0		0		4326		↔
		0																
30.25		0		1		1		8.25		-11		0		0		4326		↔
		0																
38.5		0		1		1		8.25		-11		0		0		4326		↔
		0																
46.75		0		1		1		8.25		-11		0		0		4326		↔
		0																

関連情報

[ST_MakeEmptyRaster](#)

11.3.6 ST_MakeEmptyRaster

ST_MakeEmptyRaster — 与えられたピクセル範囲 (width & height)、左上の X,Y、ピクセルサイズ、回転 (scalex, scaley, skewx, skewy) と空間参照系 (srid) が指定された空ラスタ (バンドを持たないラスタ) を返します。ラスタが渡されると、新しいラスタは渡されたラスタと同じサイズ、アラインメント、SRID になります。SRID が指定されていない場合には、空間参照系は不明 (0) とされます。

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);

説明

与えられたピクセル範囲 (width & height) を持ち、かつ、左上隅の X (upperleftx) と Y (upperlefty)、ピクセルサイズと回転 (scalex, scaley, skewx, skewy)、空間参照系 (srid) によって空間 (またはワールド) 座標上で地理参照された空ラスタ (バンドを持たないラスタ) を返します。

最後の形式では、ピクセルサイズを一つの引数 (pixelsize) で指定しています。scalex はこの引数に、scaley は引数の正負逆の数に、それぞれ設定され、skewx と skewy は 0 に設定されます。

既存のラスタを渡すと、同じメタデータ設定 (バンド以外) を持つ新しいラスタが返ります。

SRID が設定されていない場合には、デフォルトは 0 です。空ラスタを生成した後に、バンドの追加や編集を行うことになります。ST_AddBandでバンドを定義し、ST_SetValueで初期ピクセル値を設定します。

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;
```

rid	upperleftx	upperlefty	width	height	scalex	scaley	skewx	skewy	srid	↔
numbands										
3	0.0005	0.0005	100	100	1	1	0	0	↔	
4	0.0005	0.0005	100	100	1	1	0	0	↔	

関連情報

[ST_AddBand](#), [ST_MetaData](#), [ST_ScaleX](#), [ST_ScaleY](#), [ST_SetValue](#), [ST_SkewX](#), , [ST_SkewY](#)

11.3.7 ST_Tile

ST_Tile — 求められた出力ラスタのピクセル数に基づいて入力ラスタを分割した結果のラスタ集合を返します。

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);
```

説明

求められた出力ラスタのピクセル数に基づいて入力ラスタを分割した結果のラスタ集合を返します。

padwithnodata が FALSE である場合には、ラスタの右端と下端のタイルは、他のタイルと異なるピクセル範囲を持つことがあります。**padwithnodata** が TRUE の場合には、全てのタイルは同じピクセル範囲を持ち、端のタイルに NODATA 値が詰められます。ラスタバンドに NODATA 値が指定されていない場合には、**nodataval** で指定することができます。

**Note**

入力ラスタの指定したバンドがデータベース外ラスタの場合には、出力ラスタ内の対応するバンドもデータベース外になります。

Availability: 2.1.0

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;
```

```
(1,"{{1,1,1},{1,1,1},{1,1,1}}")
(2,"{{10,10,10},{10,10,10},{10,10,10}}")
(1,"{{2,2,2},{2,2,2},{2,2,2}}")
(2,"{{20,20,20},{20,20,20},{20,20,20}}")
(1,"{{3,3,3},{3,3,3},{3,3,3}}")
(2,"{{30,30,30},{30,30,30},{30,30,30}}")
(1,"{{4,4,4},{4,4,4},{4,4,4}}")
(2,"{{40,40,40},{40,40,40},{40,40,40}}")
(1,"{{5,5,5},{5,5,5},{5,5,5}}")
(2,"{{50,50,50},{50,50,50},{50,50,50}}")
(1,"{{6,6,6},{6,6,6},{6,6,6}}")
(2,"{{60,60,60},{60,60,60},{60,60,60}}")
(1,"{{7,7,7},{7,7,7},{7,7,7}}")
(2,"{{70,70,70},{70,70,70},{70,70,70}}")
(1,"{{8,8,8},{8,8,8},{8,8,8}}")
(2,"{{80,80,80},{80,80,80},{80,80,80}}")
(1,"{{9,9,9},{9,9,9},{9,9,9}}")
(2,"{{90,90,90},{90,90,90},{90,90,90}}")
(18 rows)
```

```

WITH foo AS (
  SELECT ST_AddBand(ST_AddBand(ST_MakeEmptyRaster(3, 3, 0, 0, 1, -1, 0, 0, 0), 1, '8BUI', 1, 0), 2, '8BUI', 10, 0) AS rast UNION ALL
  SELECT ST_AddBand(ST_AddBand(ST_MakeEmptyRaster(3, 3, 3, 0, 1, -1, 0, 0, 0), 1, '8BUI', 2, 0), 2, '8BUI', 20, 0) AS rast UNION ALL
  SELECT ST_AddBand(ST_AddBand(ST_MakeEmptyRaster(3, 3, 6, 0, 1, -1, 0, 0, 0), 1, '8BUI', 3, 0), 2, '8BUI', 30, 0) AS rast UNION ALL

  SELECT ST_AddBand(ST_AddBand(ST_MakeEmptyRaster(3, 3, 0, -3, 1, -1, 0, 0, 0), 1, '8BUI', 4, 0), 2, '8BUI', 40, 0) AS rast UNION ALL
  SELECT ST_AddBand(ST_AddBand(ST_MakeEmptyRaster(3, 3, 3, -3, 1, -1, 0, 0, 0), 1, '8BUI', 5, 0), 2, '8BUI', 50, 0) AS rast UNION ALL
  SELECT ST_AddBand(ST_AddBand(ST_MakeEmptyRaster(3, 3, 6, -3, 1, -1, 0, 0, 0), 1, '8BUI', 6, 0), 2, '8BUI', 60, 0) AS rast UNION ALL

  SELECT ST_AddBand(ST_AddBand(ST_MakeEmptyRaster(3, 3, 0, -6, 1, -1, 0, 0, 0), 1, '8BUI', 7, 0), 2, '8BUI', 70, 0) AS rast UNION ALL
  SELECT ST_AddBand(ST_AddBand(ST_MakeEmptyRaster(3, 3, 3, -6, 1, -1, 0, 0, 0), 1, '8BUI', 8, 0), 2, '8BUI', 80, 0) AS rast UNION ALL
  SELECT ST_AddBand(ST_AddBand(ST_MakeEmptyRaster(3, 3, 6, -6, 1, -1, 0, 0, 0), 1, '8BUI', 9, 0), 2, '8BUI', 90, 0) AS rast
), bar AS (
  SELECT ST_Union(rast) AS rast FROM foo
), baz AS (
  SELECT ST_Tile(rast, ARRAY[2], 3, 3) AS rast FROM bar
)
SELECT
  ST_DumpValues(rast)
FROM baz;

```

```

(1,"{{10,10,10},{10,10,10},{10,10,10}}")
(1,"{{20,20,20},{20,20,20},{20,20,20}}")
(1,"{{30,30,30},{30,30,30},{30,30,30}}")
(1,"{{40,40,40},{40,40,40},{40,40,40}}")
(1,"{{50,50,50},{50,50,50},{50,50,50}}")
(1,"{{60,60,60},{60,60,60},{60,60,60}}")
(1,"{{70,70,70},{70,70,70},{70,70,70}}")
(1,"{{80,80,80},{80,80,80},{80,80,80}}")
(1,"{{90,90,90},{90,90,90},{90,90,90}}")
(9 rows)

```

関連情報

[ST_Union](#), [ST_Retile](#)

11.3.8 ST_Retile

ST_Retile — 任意のタイル化されたラスタカバレッジから構成されたタイルの集合を返します。

Synopsis

SETOF raster **ST_Retile**(regclass tab, name col, geometry ext, float8 sfx, float8 sfy, int tw, int th, text algo='NearestNeighbor');

説明

指定された縮尺 (*sfx*, *sfy*) と最大サイズ (*tw*, *th*) を持ち、指定された範囲 (*ext*) を包含し、指定されたラスタカバレッジ (*tab*, *col*) からのデータを持つタイルの集合を返します。

アルゴリズム指定オプションは 'NearestNeighbor', 'Bilinear', 'Cubic', 'CubicSpline', 'Lanczos' です。詳細については [GDAL Warp resampling methods](#) をご覧ください。

Availability: 2.2.0

関連情報

[ST_CreateOverview](#)

11.3.9 ST_FromGDALRaster

ST_FromGDALRaster — 対応する GDAL ラスタファイルからラスタを返します。

Synopsis

```
raster ST_FromGDALRaster(bytea gdaldata, integer srid=NULL);
```

説明

対応する GDAL ラスタファイルからラスタを返します。gdaldata は bytea 型で、GDAL ラスタファイルの中身です。

srid が NULL の場合には、この関数は、GDAL ラスタから自動的に SRID を設定しようとします。srid が与えられている場合には、与えられた値が自動的に設定した SRID を上書きします。

Availability: 2.1.0

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)

関連情報

[ST_AsGDALRaster](#)

11.4 ラスタアクセス

11.4.1 ST_GeoReference

ST_GeoReference — GDAL 書式または一般的にワールドファイルでみられる ESRI 書式の地理参照メタデータを返します。デフォルトは GDAL です。

Synopsis

```
text ST_GeoReference(raster rast, text format=GDAL);
```

説明

GDAL 書式または一般的に [world file](#) (英語版 Wikipedia) でみられる ESRI 書式の地理参照メタデータを返します。書式を指定しない場合のデフォルトは GDAL です。書式は 'GDAL' または 'ESRI' の文字列です。

書式の表現の違いは次の通りです。

GDAL:

```
scalex
skewy
skewx
scaley
upperleftx
upperlefty
```

ESRI:

```
scalex
skewy
skewx
scaley
upperleftx + scalex*0.5
upperlefty + scaley*0.5
```

例

```
SELECT ST_GeoReference(rast, 'ESRI') As esri_ref, ST_GeoReference(rast, 'GDAL') As gdal_ref
FROM dummy_rast WHERE rid=1;
```

esri_ref		gdal_ref
2.0000000000		2.0000000000
0.0000000000	:	0.0000000000
0.0000000000	:	0.0000000000
3.0000000000	:	3.0000000000
1.5000000000	:	0.5000000000
2.0000000000	:	0.5000000000

関連情報

[ST_SetGeoReference](#), [ST_ScaleX](#), [ST_ScaleY](#)

11.4.2 ST_Height

ST_Height — ラスタの高さをピクセル単位で返します。

Synopsis

```
integer ST_Height(raster rast);
```

説明

ラスタの高さをピクセル単位で返します。

例

```
SELECT rid, ST_Height(rast) As rastheight
FROM dummy_rast;
```

rid	rastheight
1	20
2	5

関連情報

[ST_Width](#)

11.4.3 ST_IsEmpty

ST_IsEmpty — ラスタが空 (幅が 0 で高さが 0) の場合には TRUE を返します。他の場合には、FALSE を返します。

Synopsis

```
boolean ST_IsEmpty(raster rast);
```

説明

ラスタが空 (幅が 0 で高さが 0) の場合には TRUE を返します。他の場合には、FALSE を返します。

Availability: 2.0.0

例

```
SELECT ST_IsEmpty(ST_MakeEmptyRaster(100, 100, 0, 0, 0, 0, 0, 0, 0))
```

f

```
SELECT ST_IsEmpty(ST_MakeEmptyRaster(0, 0, 0, 0, 0, 0, 0, 0, 0))
```

t

関連情報

[ST_HasNoBand](#)

11.4.4 ST_MemSize

ST_MemSize — ラスタが取る領域の合計をバイト単位で返します。

Synopsis

integer **ST_MemSize**(raster rast);

説明

ラスタが取る領域の合計をバイト単位で返します。

この関数は PostgreSQL ビルトイン関数 `pg_column_size`, `pg_size_pretty`, `pg_relation_size`, `pg_total_relation_size` へのすばらしい賛辞です。

Note



テーブルのバイト単位のサイズを得る `pg_relation_size` は `ST_MemSize` よりも小さい値を返すことがあります。 `pg_relation_size` は TOAST テーブルの寄与分を追加せず、大きなジオメトリは TOAST テーブルに格納されるためです。 `pg_column_size` は圧縮後のサイズを返すので、小さくなることがあります。

`pg_total_relation_size` - テーブル、TOAST テーブル、インデックスを含みます。

Availability: 2.2.0

例

```
SELECT ST_MemSize(ST_AsRaster(ST_Buffer(ST_Point(1,5),10,1000),150, 150, '8BUI')) AS  
      rast_mem;
```

22568

関連情報

11.4.5 ST_MetaData

ST_MetaData — ピクセル数、回転 (スキュー)、左上隅位置等のラスタオブジェクトに関する基本的なメタデータを返します。

Synopsis

record **ST_MetaData**(raster rast);

説明

ピクセルサイズ、回転 (skew)、左上隅位置等のラスタオブジェクトに関する基本的なメタデータを返します。返されるカラムは upperleftx | upperlefty | width | height | scalex | scaley | skewx | skewy | srid | numbands です。

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	↵
	numbands									
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	↵

関連情報

[ST_BandMetaData](#), [ST_NumBands](#)

11.4.6 ST_NumBands

ST_NumBands — ラスタオブジェクトのバンド数を返します。

Synopsis

integer **ST_NumBands**(raster rast);

説明

ラスタオブジェクトのバンド数を返します。

例

```
SELECT rid, ST_NumBands(rast) As numbands
FROM dummy_rast;
```

rid	numbands
1	0
2	3

関連情報

[ST_Value](#)

11.4.7 ST_PixelHeight

ST_PixelHeight — 空間参照系の地理的な単位でのピクセルの高さを返します。

Synopsis

double precision **ST_PixelHeight**(raster rast);

説明

空間参照系の地理的な単位でのピクセルの高さを返します。スキューが無い一般的な状況では、ピクセル高は地理座標とラスタピクセルのスケール率です。

関係について図示したものは[ST_PixelWidth](#)を参照して下さい。

例: スキューの無いラスタ

```
SELECT ST_Height(rast) As rastheight, ST_PixelHeight(rast) As pixheight,
       ST_ScaleX(rast) As scalex, ST_ScaleY(rast) As scaley, ST_SkewX(rast) As skewx,
       ST_SkewY(rast) As skewy
FROM dummy_rast;
```

rastheight	pixheight	scalex	scaley	skewx	skewy
20	3	2	3	0	0
5	0.05	0.05	-0.05	0	0

例: スキューが **0** でないラスタ

```
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

関連情報

[ST_PixelWidth](#), [ST_ScaleX](#), [ST_ScaleY](#), [ST_SkewX](#), [ST_SkewY](#)

11.4.8 ST_PixelWidth

`ST_PixelWidth` — 空間参照系の地理的な単位でのピクセルの幅を返します。

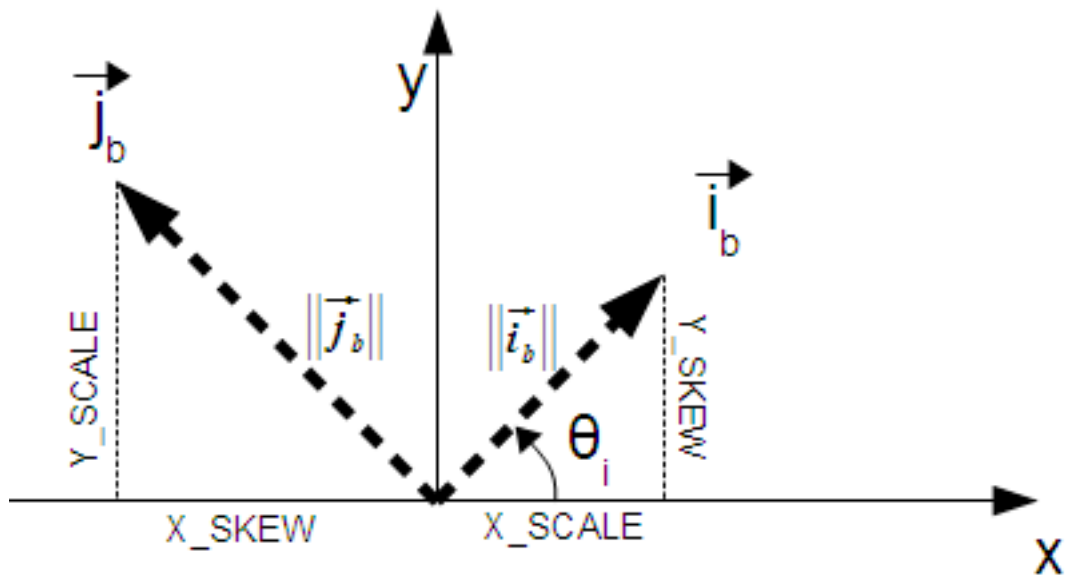
Synopsis

double precision **ST_PixelWidth**(raster rast);

説明

空間参照系の地理的な単位でのピクセルの幅を返します。スキューが無い一般的な状況では、ピクセル幅は地理座標とラスタピクセルのスケール率です。

関係について次の図に示します。



ピクセル幅: i 方向のピクセルサイズ
ピクセル高: j 方向のピクセルサイズ

例: スキューの無いラスタ

```
SELECT ST_Width(rast) As rastwidth, ST_PixelWidth(rast) As pixwidth,
       ST_ScaleX(rast) As scalex, ST_ScaleY(rast) As scaley, ST_SkewX(rast) As skewx,
       ST_SkewY(rast) As skewy
FROM dummy_rast;
```

rastwidth	pixwidth	scalex	scaley	skewx	skewy
10	2	2	3	0	0
5	0.05	0.05	-0.05	0	0

例: スキューが **0** でないラスタ

```
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

関連情報

[ST_PixelHeight](#), [ST_ScaleX](#), [ST_ScaleY](#), [ST_SkewX](#), [ST_SkewY](#)

11.4.9 ST_ScaleX

ST_ScaleX — 空間参照系の地理的な単位でのピクセル幅の X 成分を返します。

Synopsis

```
float8 ST_ScaleX(raster rast);
```

説明

空間参照系の地理的な単位でのピクセル幅の X 成分を返します。詳細については[World File](#) (英語版 Wikipedia) をご覧ください。

Changed: 2.0.0. WKTRaster 版では ST_PixelSizeX と呼ばれていました。

例

```
SELECT rid, ST_ScaleX(rast) As rastpixwidth
FROM dummy_rast;
```

rid	rastpixwidth
1	2
2	0.05

関連情報

[ST_Width](#)

11.4.10 ST_ScaleY

ST_ScaleY — 空間参照系の地理的な単位でのピクセル幅の Y 成分を返します。

Synopsis

```
float8 ST_ScaleY(raster rast);
```

説明

空間参照系の地理的な単位でのピクセル幅の Y 成分を返します。詳細については[World File](#) (英語版 Wikipedia) をご覧ください。

Changed: 2.0.0. WKTRaster 版では ST_PixelSizeY と呼ばれていました。

例

```
SELECT rid, ST_ScaleY(rast) As rastpixheight
FROM dummy_rast;
```

rid	rastpixheight
1	3
2	-0.05

関連情報

[ST_Height](#)

11.4.11 ST_RasterToWorldCoord

ST_RasterToWorldCoord — ラスタの指定した列と行における左上隅の地理座標 X 値と Y 値 (経度と緯度) を返します。列と行の番号は 1 始まりです。

Synopsis

```
record ST_RasterToWorldCoord(raster rast, integer xcolumn, integer yrow);
```

説明

ラスタの指定した列と行における左上隅の地理座標 X 値と Y 値 (経度と緯度) を返します。X 値と Y 値の単位は、地理参照されたラスタの地理単位です。列と行の数字は 1 始まりですが、引数に 0、負数またはラスタのピクセル範囲を超える値が渡されている場合には、ラスタのグリッドがラスタ範囲外に適用できると仮定して、範囲外の座標を返します。

Availability: 2.1.0

例

This example uses a 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

This example uses a 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

関連情報

[ST_RasterToWorldCoordX](#), [ST_RasterToWorldCoordY](#), [ST_SetSkew](#)

11.4.12 ST_RasterToWorldCoordX

ST_RasterToWorldCoordX — ラスタの指定した列と行における左上隅の地理座標の X 値を返します。列と行の番号は 1 始まりです。

Synopsis

```
float8 ST_RasterToWorldCoordX(raster rast, integer xcolumn);
float8 ST_RasterToWorldCoordX(raster rast, integer xcolumn, integer yrow);
```

説明

ラスタの指定した列と行における左上隅の地理座標の X 値を地理参照したラスタの地理単位で返します。列と行の番号は 1 始まりです。負数またはラスタの列数を超える値を渡した場合には、スキューとピクセルサイズが選択したラスタと同じであるという仮定のもとで、ラスタファイルから左または右にはずれた座標値を返します。



Note

スキューの無いラスタでは、X 列を与えれば十分です。スキューのあるラスタの場合には、地理参照のとれた座標は ST_ScaleX と ST_SkewX および行と列の関数となります。スキューのあるラスタで X 列のみ与えた場合にはエラーが発生します。

Changed: 2.1.0 以前の版では ST_Raster2WorldCoordX と呼ばれていました。

例

This example shows that providing a column is sufficient for a non-skewed raster.

```
SELECT rid, ST_RasterToWorldCoordX(rast,1) As x1coord,
       ST_RasterToWorldCoordX(rast,2) As x2coord,
       ST_ScaleX(rast) As pixelx
FROM dummy_rast;
```

rid	x1coord	x2coord	pixelx
1	0.5	2.5	2
2	3427927.75	3427927.8	0.05

For fun let's skew it.

```
SELECT rid, ST_RasterToWorldCoordX(rast, 1, 1) As x1coord,
       ST_RasterToWorldCoordX(rast, 2, 3) As x2coord,
       ST_ScaleX(rast) As pixelx
FROM (SELECT rid, ST_SetSkew(rast, 100.5, 0) As rast FROM dummy_rast) As foo;
```

rid	x1coord	x2coord	pixelx
1	0.5	203.5	2
2	3427927.75	3428128.8	0.05

関連情報

[ST_ScaleX](#), [ST_RasterToWorldCoordY](#), [ST_SetSkew](#), [ST_SkewX](#)

11.4.13 ST_RasterToWorldCoordY

ST_RasterToWorldCoordY — ラスタの指定した列と行における左上隅の地理座標の Y 値を返します。列と行の番号は 1 始まりです。

Synopsis

```
float8 ST_RasterToWorldCoordY(raster rast, integer yrow);
float8 ST_RasterToWorldCoordY(raster rast, integer xcolumn, integer yrow);
```

説明

ラスタの指定した列と行における左上隅の地理座標の Y 値を地理参照したラスタの地理単位で返します。列と行の番号は 1 始まりです。負数またはラスタの列数や行数を超える値を渡した場合には、スキューとピクセルサイズが選択したラスタと同じであるという仮定のもとで、ラスタファイルから左または右にはずれた座標値を返します。



Note

スキューの無いラスタでは、Y 行を与えれば十分です。スキューのあるラスタの場合には、地理参照のとれた座標は `ST_ScaleY` と `ST_SkewY` および行と列の関数となります。スキューのあるラスタで Y 行のみ与えた場合にはエラーが発生します。

Changed: 2.1.0 以前の版では `ST_Raster2WorldCoordY` と呼ばれていました。

例

Non-skewed raster providing row is sufficient.

```
SELECT rid, ST_RasterToWorldCoordY(rast,1) As ylcoord,
       ST_RasterToWorldCoordY(rast,3) As y2coord,
       ST_ScaleY(rast) As pixely
FROM dummy_rast;
```

rid	ylcoord	y2coord	pixely
1	0.5	6.5	3
2	5793244	5793243.9	-0.05

For fun let's skew it.

```
SELECT rid, ST_RasterToWorldCoordY(rast,1,1) As ylcoord,
       ST_RasterToWorldCoordY(rast,2,3) As y2coord,
       ST_ScaleY(rast) As pixely
FROM (SELECT rid, ST_SetSkew(rast,0,100.5) As rast FROM dummy_rast) As foo;
```

rid	ylcoord	y2coord	pixely
1	0.5	107	3
2	5793244	5793344.4	-0.05

関連情報

[ST_ScaleY](#), [ST_RasterToWorldCoordX](#), [ST_SetSkew](#), [ST_SkewY](#)

11.4.14 ST_Rotation

ST_Rotation — ラスタの回転をラジアンで返します。

Synopsis

```
float8 ST_Rotation(raster rast);
```

説明

ラスタの回転をラジアンで返します。ラスタが回転を持っていない場合には、NaN が返されます。詳細については [World File](#) (英語版 Wikipedia) を参照して下さい。

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

関連情報

[ST_SetRotation](#), [ST_SetScale](#), [ST_SetSkew](#)

11.4.15 ST_SkewX

ST_SkewX — 空間参照の X スキュー (回転パラメータ) を返します。

Synopsis

```
float8 ST_SkewX(raster rast);
```

説明

空間参照の X スキュー (回転パラメータ) を返します。詳細については[World File](#) (英語版 Wikipedia) を参照して下さい。

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

関連情報

[ST_GeoReference](#), [ST_SkewY](#), [ST_SetSkew](#)

11.4.16 ST_SkewY

ST_SkewY — 空間参照の Y スキュー (回転パラメータ) を返します。

Synopsis

```
float8 ST_SkewY(raster rast);
```

説明

空間参照の Y スキュー (回転パラメータ) を返します。詳細については[World File](#) (英語版 Wikipedia) を参照して下さい。

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

関連情報

[ST_GeoReference](#), [ST_SkewX](#), [ST_SetSkew](#)

11.4.17 ST_SRID

ST_SRID — ラスタの spatial_ref_sys テーブルで定義されている空間参照系識別番号を返します。

Synopsis

```
integer ST_SRID(raster rast);
```

説明

ラスタの spatial_ref_sys テーブルで定義されている空間参照系識別番号を返します。



Note
地理参照を施していないラスタとジオメトリの SRID は 0 です。

例

```
SELECT ST_SRID(rast) As srid
FROM dummy_rast WHERE rid=1;
```

0

関連情報

Section [4.5](#), [ST_SRID](#)

11.4.18 ST_Summary

ST_Summary — ラスタの中身の概要が文字列で返されます。

Synopsis

text **ST_Summary**(raster rast);

説明

ラスタの中身の概要が文字列で返されます。

Availability: 2.1.0

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
    )
);
```

```
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)
```

関連情報

[ST_MetaData](#), [ST_BandMetaData](#), [ST_Summary](#) [ST_Extent](#)

11.4.19 ST_UpperLeftX

ST_UpperLeftX — 適用されている空間参照系でのラスタの左上隅の X 座標値を返します。

Synopsis

float8 **ST_UpperLeftX**(raster rast);

説明

適用されている空間参照系でのラスタの左上隅の X 座標値を返します。

例

```
SELECT rid, ST_UpperLeftX(rast) As ulx
FROM dummy_rast;
```

rid	ulx
1	0.5
2	3427927.75

関連情報

[ST_UpperLeftY](#), [ST_GeoReference](#), [Box3D](#)

11.4.20 ST_UpperLeftY

ST_UpperLeftY — 適用されている空間参照系でのラスタの左上隅の Y 座標値を返します。

Synopsis

float8 **ST_UpperLeftY**(raster rast);

説明

適用されている空間参照系でのラスタの左上隅の Y 座標値を返します。

例

```
SELECT rid, ST_UpperLeftY(rast) As uly
FROM dummy_rast;
```

rid	uly
1	0.5
2	5793244

関連情報

[ST_UpperLeftX](#), [ST_GeoReference](#), [Box3D](#)

11.4.21 ST_Width

ST_Width — ラスタの幅をピクセル単位で返します。

Synopsis

```
integer ST_Width(raster rast);
```

説明

ラスタの幅をピクセル単位で返します。

例

```
SELECT ST_Width(rast) As rastwidth  
FROM dummy_rast WHERE rid=1;
```

```
10
```

関連情報

[ST_Height](#)

11.4.22 ST_WorldToRasterCoord

ST_WorldToRasterCoord — ラスタの空間参照系による地理座標の X 値と Y 値 (経度と緯度) またはポイントジオメトリに対応するピクセルの左上隅を返します。

Synopsis

```
record ST_WorldToRasterCoord(raster rast, geometry pt);  
record ST_WorldToRasterCoord(raster rast, double precision longitude, double precision latitude);
```

説明

地理座標の X 値と Y 値 (経度と緯度) またはポイントジオメトリに対応するピクセルの左上隅を返します。X 値、Y 値やポイントジオメトリがラスタの範囲外であるかないかにかかわらず動作します。地理座標の X 値と Y 値はラスタの空間参照系で表現しなければなりません。

Availability: 2.1.0

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

関連情報

[ST_WorldToRasterCoordX](#), [ST_WorldToRasterCoordY](#), [ST_RasterToWorldCoordX](#), [ST_RasterToWorldCoordY](#), [ST_SRID](#)

11.4.23 ST_WorldToRasterCoordX

ST_WorldToRasterCoordX — ラスタの空間参照系に基づくポイントジオメトリ (pt) または X,Y 座標値 (xw,yw) に対応するラスタの列を返します。

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);
```

説明

ラスタの空間参照系に基づくポイントジオメトリ (pt) または X,Y 座標値 (xw,yw) に対応するラスタの列を返します。ラスタがスキューされている場合には、ポイントか xw と yw の両方が必要です。ラスタがスキューされていない場合には、xw を指定すれば十分です。ワールド座標系はラスタの空間参照系です。

Changed: 2.1.0 以前の版では ST_World2RasterCoordX と呼ばれていました。

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

関連情報

[ST_RasterToWorldCoordX](#), [ST_RasterToWorldCoordY](#), [ST_SRID](#)

11.4.24 ST_WorldToRasterCoordY

ST_WorldToRasterCoordY — ラスタの空間参照系に基づくポイントジオメトリ (pt) または X,Y 座標値 (xw,yw) に対応するラスタの行を返します。

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);
```

説明

ラスタの空間参照系に基づくポイントジオメトリ (pt) または X,Y 座標値 (xw,yw) に対応するラスタの行を返します。ラスタがスキューされている場合には、ポイントか **xw** と **yw** の両方かが必要です。ラスタがスキューされていない場合には、**yw** を指定すれば十分です。ワールド座標系はラスタの空間参照系です。

Changed: 2.1.0 以前の版では **ST_World2RasterCoordY** と呼ばれていました。

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

関連情報

[ST_RasterToWorldCoordX](#), [ST_RasterToWorldCoordY](#), [ST_SRID](#)

11.5 ラスタバンドアクセス

11.5.1 ST_BandMetaData

ST_BandMetaData — 指定したラスタバンドの基本的なメタデータを返します。バンド番号を指定しない場合には、1番と仮定します。

Synopsis

- (1) record **ST_BandMetaData**(raster rast, integer band=1);
- (2) record **ST_BandMetaData**(raster rast, integer[] band);

説明

指定したラスタブンドの基本的なメタデータを返します。返されるカラムは `pixeltype`, `nodatavalue`, `isoutdb`, `path`, `outdbbandnum`, `filesize`, `filetimestamp` です。



Note

ラスタブンドを持たない場合にはエラーが投げられます。



Note

バンドに NODATA 値が無い場合には、`nodatavalue` は NULL になります。



Note

`isoutdb` が FALSE の場合には、`path`, `outdbbandnum`, `filesize` および `filetimestamp` は NULL です。
`outdb` のアクセスが禁止されているなら、`filesize` と `filetimestamp` が NULL になります。

Enhanced: 2.5.0 `outdb` ラスタに `outdbbandnum`, `filesize` と `filetimestamp` を取り入れました。

例: 一つ目の形式

```
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	

例: 二つ目の形式

```
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
3	8BUI		t		3	12345	1521807257
2	8BUI		t		2	12345	1521807257

関連情報

[ST_MetaData](#), [ST_BandPixelType](#)

11.5.2 ST_BandNoDataValue

ST_BandNoDataValue — 指定されたバンドについてデータが無いことを表現する値を返します。バンド番号を指定しない場合には、1 番と仮定します。

Synopsis

double precision **ST_BandNoDataValue**(raster rast, integer bandnum=1);

説明

指定されたバンドについてデータが無いことを表現する値を返します。

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

関連情報

[ST_NumBands](#)

11.5.3 ST_BandIsNoData

ST_BandIsNoData — 指定したバンドが NODATA 値だけで満たされている場合には、TRUE を返します。

Synopsis

```
boolean ST_BandIsNoData(raster rast, integer band, boolean forceChecking=false);
boolean ST_BandIsNoData(raster rast, boolean forceChecking=false);
```

説明

指定したバンドが NODATA 値だけで満たされている場合には、TRUE を返します。バンド番号を指定しない場合には、1 番と仮定します。最後の引数が TRUE の場合には、全バンドについてピクセル毎に調べます。他の場合には、isnodata フラグの値を返すだけです。この引数を指定しない場合のデフォルト値は FALSE です。

Availability: 2.0.0



Note

If the flag is dirty (this is, the result is different using TRUE as last parameter and not using it) you should update the raster to set this flag to true, by using ST_SetBandIsNoData(), or ST_SetBandNoDataValue() with TRUE as last argument. See [ST_SetBandIsNoData](#).

例

This example creates a dummy table with one raster column.

```
create table dummy_rast (rid integer, rast raster);

-- Add raster with two bands, one pixel/band. In the first band, nodatavalue = pixel value ←
-- = 3.
-- In the second band, nodatavalue = 13, pixel value = 4
insert into dummy_rast values(
1,
(
'01' -- little endian (uint8 ndr)
||
'0000' -- version (uint16 0)
||
'0200' -- nBands (uint16 0)
||
'17263529ED684A3F' -- scaleX (float64 0.000805965234044584)
||
'F9253529ED684ABF' -- scaleY (float64 -0.00080596523404458)
||
'1C9F33CE69E352C0' -- ipX (float64 -75.5533328537098)
||
'718F0E9A27A44840' -- ipY (float64 49.2824585505576)
||
'ED50EB853EC32B3F' -- skewX (float64 0.000211812383858707)
||
'7550EB853EC32B3F' -- skewY (float64 0.000211812383858704)
||
'E6100000' -- SRID (int32 4326)
||
'0100' -- width (uint16 1)
```

```
||
'0100' -- height (uint16 1)
||
'6' -- hasnodatavalue and isnodata value set to true.
||
'2' -- first band type (4BUI)
||
'03' -- novalue==3
||
'03' -- pixel(0,0)==3 (same that nodata)
||
'0' -- hasnodatavalue set to false
||
'5' -- second band type (16BSI)
||
'0D00' -- novalue==13
||
'0400' -- pixel(0,0)==4
)::raster
);

select ST_BandIsNoData(rast, 1) from dummy_rast where rid = 1; -- Expected true
select ST_BandIsNoData(rast, 2) from dummy_rast where rid = 1; -- Expected false
```

関連情報

[ST_BandNoDataValue](#), [ST_NumBands](#), [ST_SetBandNoDataValue](#), [ST_SetBandIsNoData](#)

11.5.4 ST_BandPath

ST_BandPath — ファイルシステムに格納されているバンドのシステムファイルパスを返します。バンド番号が指定されていない場合には、1 番と仮定します。

Synopsis

text **ST_BandPath**(raster rast, integer bandnum=1);

説明

バンドへのシステムファイルパスを返します。データベース内に格納されているバンドでこの関数と呼んだ場合にはエラーが投げられます。

例

関連情報

11.5.5 ST_BandFileSize

ST_BandFileSize — ファイルシステムに格納されているバンドのファイルサイズを返します。バンド番号が指定されていない場合には、1 番と仮定します。

Synopsis

bigint **ST_BandFileSize**(raster rast, integer bandnum=1);

説明

ファイルシステムに格納されているバンドのファイルサイズを返します。データベース内バンドを与えるか、データベース外バンドにアクセスできない場合にはエラーが投げられます。

この関数は一般的に `ST_BandPath()` と `ST_BandFileTimestamp()` とを併用して、データベース外ラスタのファイル名が、この関数から見えるものとサーバから見えるものと同じかどうかを判断することができます。

Availability: 2.5.0

例

```
SELECT ST_BandFileSize(rast,1) FROM dummy_rast WHERE rid = 1;
```

```
240574
```

11.5.6 ST_BandFileTimestamp

`ST_BandFileTimestamp` — ファイルシステムに格納されているバンドのファイルタイムスタンプ返します。バンド番号が指定されていない場合には、1 番と仮定します。

Synopsis

bigint **ST_BandFileTimestamp**(raster rast, integer bandnum=1);

説明

ファイルシステムに保存されているバンドのファイルタイムスタンプを返します (UTC 1970 年 1 月 1 日 0 時 0 分 0 秒からの秒数)。データベース内バンドの場合またはデータベース外ラスタへのアクセスが無効な場合にはエラーが投げられます。

この関数は通常、`ST_BandPath()` および `ST_BandFileSize()` と組み合わせて使用されるため、クライアントは、`outdb` ラスタのファイル名がサーバーによって見られるものと同じであるかどうかを判断できます。

Availability: 2.5.0

例

```
SELECT ST_BandFileTimestamp(rast,1) FROM dummy_rast WHERE rid = 1;
```

```
1521807257
```

11.5.7 ST_BandPixelType

`ST_BandPixelType` — 指定したバンドのピクセルタイプを返します。バンド番号が指定されていない場合には、1 番と仮定します。

Synopsis

text **ST_BandPixelType**(raster rast, integer bandnum=1);

説明

与えられたバンドのセルに格納されるデータの型とサイズを示す名前を返します。
ピクセルタイプは 11 通りあります。対応しているピクセルタイプは次の通りです。

- 1BB - 1 ビット真偽値
- 2BUI - 2 ビット符号なし整数
- 4BUI - 4 ビット符号なし整数
- 8BSI - 8 ビット整数
- 8BUI - 8 ビット符号なし整数
- 16BSI - 16 ビット整数
- 16BUI - 16 ビット符号なし整数
- 32BSI - 32 ビット整数
- 32BUI - 32 ビット符号なし整数
- 32BF - 32 ビット浮動小数点数
- 64BF - 64 ビット浮動小数点数

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
```

関連情報

[ST_NumBands](#)

11.5.8 ST_MinPossibleValue

ST_MinPossibleValue — このピクセルタイプが格納できる最小値を返します。

Synopsis

integer **ST_MinPossibleValue**(text pixeltype);

説明

このピクセルタイプが格納できる最小値を返します。

例

```
SELECT ST_MinPossibleValue('16BSI');
```

```
-32768
```

```
SELECT ST_MinPossibleValue('8BUI');
```

```
0
```

関連情報

[ST_BandPixelType](#)

11.5.9 ST_HasNoBand

ST_HasNoBand — 指定したバンド番号のバンドが無い場合には、TRUE を返します。バンド番号を指定していない場合には、1 番と仮定します。

Synopsis

boolean **ST_HasNoBand**(raster rast, integer bandnum=1);

説明

指定したバンド番号のバンドが無い場合には、TRUE を返します。バンド番号を指定していない場合には、1 番と仮定します。

Availability: 2.0.0

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

関連情報

[ST_NumBands](#)

11.6 ラスタピクセルアクセサとセッター

11.6.1 ST_PixelAsPolygon

ST_PixelAsPolygon — 指定した行と列のピクセルの境界となるジオメトリを返します。

Synopsis

geometry **ST_PixelAsPolygon**(raster rast, integer columnx, integer rowy);

説明

指定した行と列のピクセルの境界となるジオメトリを返します。

Availability: 2.0.0

例

This example returns a raster pixel polygon.

```
SELECT i,j, ST_AsText(ST_PixelAsPolygon(foo.rast, i,j)) As blpgeom
FROM dummy_rast As foo
      CROSS JOIN generate_series(1,2) As i
      CROSS JOIN generate_series(1,1) As j
WHERE rid=2;
```

i	j	blpgeom
1	1	POLYGON((3427927.75 5793244,3427927.8 5793244,3427927.8 5793243.95,...
2	1	POLYGON((3427927.8 5793244,3427927.85 5793244,3427927.85 5793243.95, ..

関連情報

[ST_DumpAsPolygons](#), [ST_PixelAsPolygons](#), [ST_PixelAsPoint](#), [ST_PixelAsPoints](#), [ST_PixelAsCentroid](#), [ST_PixelAsCentroids](#), [ST_Intersection](#), [ST_AsText](#)

11.6.2 ST_PixelAsPolygons

ST_PixelAsPolygons — 全てのピクセルについて境界となるジオメトリを、ピクセルごとのピクセル値とラスタ座標系の X と Y とを付けて返します。

Synopsis

setof record **ST_PixelAsPolygons**(raster rast, integer band=1, boolean exclude_nodata_value=TRUE);

説明

全てのピクセルについて境界となるジオメトリを、ピクセルごとのピクセル値 (倍精度浮動小数点数) とラスタ座標系の X と Y (ともに整数) とを付けて返します。

返されるレコードの書式は、*geom* (**geometry**型)、*val* (倍精度浮動小数点数型)、*x* (整数型)、*y* (整数型) です。



Note
exclude_nodata_value が TRUE の時、値が NODATA でないこれらのピクセルだけをポイントとして返します。



Note
ST_PixelAsPolygons は、ピクセルごとに一つのポリゴンジオメトリを返します。*ST_DumpAsPolygons* とは、一つのジオメトリが同じ値となる一つ以上のピクセルを表現する点で違います。

Availability: 2.0.0
Enhanced: 2.1.0 任意引数 *exclude_nodata_value* が追加されました。
Changed: 2.1.1 *exclude_nodata_value* の挙動を変更しました。

例

This example returns a raster pixel polygon.

```
SELECT (gv).x, (gv).y, (gv).val, ST_AsText((gv).geom) geom
FROM (SELECT ST_PixelAsPolygons(ST_SetValue(ST_SetValue(ST_AddBand(ST_MakeEmptyRaster(2, 2, ←
    0, 0, 0.001, -0.001, 0.001, 0.001, 4269),
                                     '8BUI'::text, 1, 0),
                                     2, 2, 10),
    1, 1, NULL)
) gv
) foo
ORDER BY (gv).x, (gv).y;
```

x	y	val	geom
1	2	1	POLYGON((0.001 -0.001,0.002 0,0.003 -0.001,0.002 -0.002,0.001 -0.001))
2	1	1	POLYGON((0.001 0.001,0.002 0.002,0.003 0.001,0.002 0,0.001 0.001))
2	2	10	POLYGON((0.002 0,0.003 0.001,0.004 0,0.003 -0.001,0.002 0))

関連情報

[ST_DumpAsPolygons](#), [ST_PixelAsPolygon](#), [ST_PixelAsPoint](#), [ST_PixelAsPoints](#), [ST_PixelAsCentroid](#), [ST_PixelAsText](#)

11.6.3 ST_PixelAsPoint

ST_PixelAsPoint — ピクセルの左上隅のポイントジオメトリを返します。

Synopsis

geometry **ST_PixelAsPoint**(raster rast, integer columnx, integer rowy);

説明

ピクセルの左上隅のポイントジオメトリを返します。

Availability: 2.1.0

例

```
SELECT ST_AsText(ST_PixelAsPoint(rast, 1, 1)) FROM dummy_rast WHERE rid = 1;
```

```
POINT(0.5 0.5)
```

関連情報

[ST_DumpAsPolygons](#), [ST_PixelAsPolygon](#), [ST_PixelAsPolygons](#), [ST_PixelAsPoints](#), [ST_PixelAsCentroid](#), [ST_PixelAsCentroids](#)

11.6.4 ST_PixelAsPoints

ST_PixelAsPoints — 全てのピクセルについてポイントジオメトリを、ピクセルごとのピクセル値とラスタ座標系の X と Y とを付けて返します。ポイントジオメトリの座標はピクセルの左上隅です。

Synopsis

setof record **ST_PixelAsPoints**(raster rast, integer band=1, boolean exclude_nodata_value=TRUE);

説明

全てのピクセルについてポイントジオメトリを、ピクセルごとのピクセル値とラスタ座標系の X と Y とを付けて返します。ポイントジオメトリの座標はピクセルの左上隅です。

返されるレコードの書式は、*geom* (**geometry**型)、*val* (倍精度浮動小数点数型)、*x* (整数型)、*y* (整数型) です。



Note

exclude_nodata_value が TRUE の時、値が NODATA でないこれらのピクセルだけをポイントとして返します。

Availability: 2.1.0

Changed: 2.1.1 *exclude_nodata_value* の挙動を変更しました。

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)

関連情報

[ST_DumpAsPolygons](#), [ST_PixelAsPolygon](#), [ST_PixelAsPolygons](#), [ST_PixelAsPoint](#), [ST_PixelAsCentroid](#), [ST_PixelAsCentroids](#)

11.6.5 ST_PixelAsCentroid

ST_PixelAsCentroid — ピクセルで表現される面の重心 (ポイントジオメトリ) を返します。

Synopsis

geometry **ST_PixelAsCentroid**(raster rast, integer x, integer y);

説明

ピクセルで表現される面の重心 (ポイントジオメトリ) を返します。

Enhanced: 3.2.0 C 言語による、より高速な実装。

Availability: 2.1.0

例

```
SELECT ST_AsText(ST_PixelAsCentroid(rast, 1, 1)) FROM dummy_rast WHERE rid = 1;
```

```
POINT(1.5 2)
```

関連情報

[ST_DumpAsPolygons](#), [ST_PixelAsPolygon](#), [ST_PixelAsPolygons](#), [ST_PixelAsPoint](#), [ST_PixelAsPoints](#), [ST_PixelAsCentroids](#)

11.6.6 ST_PixelAsCentroids

ST_PixelAsCentroids — 全てのピクセルについて重心 (ポイントジオメトリ) を、ピクセルごとのピクセル値とラスタ座標系の X と Y とを付けて返します。ポイントジオメトリの座標はピクセルで表現される面の重心です。

Synopsis

```
setof record ST_PixelAsCentroids(raster rast, integer band=1, boolean exclude_nodata_value=TRUE);
```

説明

全てのピクセルについて重心 (ポイントジオメトリ) を、ピクセルごとのピクセル値とラスタ座標系の X と Y とを付けて返します。ポイントジオメトリの座標はピクセルで表現される面の重心です。

返されるレコードの書式は、*geom* (**geometry**型)、*val* (倍精度浮動小数点数型)、*x* (整数型)、*y* (整数型) です。



Note

exclude_nodata_value が TRUE の時、値が NODATA でないこれらのピクセルだけをポイントとして返します。

Enhanced: 3.2.0 C 言語による、より高速な実装。

Changed: 2.1.1 *exclude_nodata_value* の挙動を変更しました。

Availability: 2.1.0

例

This example uses LATERAL syntax.

```
SELECT x, y, val, ST_AsText(geom)
FROM (SELECT dp.* FROM dummy_rast, LATERAL ST_PixelAsCentroids(rast, 1) AS dp WHERE rid <= 2) foo;
```

x	y	val	st_astext
1	1	253	POINT(3427927.775 5793243.975)
2	1	254	POINT(3427927.825 5793243.975)
3	1	253	POINT(3427927.875 5793243.975)
4	1	254	POINT(3427927.925 5793243.975)
5	1	254	POINT(3427927.975 5793243.975)

```

1 | 2 | 253 | POINT(3427927.775 5793243.925)
2 | 2 | 254 | POINT(3427927.825 5793243.925)
3 | 2 | 254 | POINT(3427927.875 5793243.925)
4 | 2 | 253 | POINT(3427927.925 5793243.925)
5 | 2 | 249 | POINT(3427927.975 5793243.925)
1 | 3 | 250 | POINT(3427927.775 5793243.875)
2 | 3 | 254 | POINT(3427927.825 5793243.875)
3 | 3 | 254 | POINT(3427927.875 5793243.875)
4 | 3 | 252 | POINT(3427927.925 5793243.875)
5 | 3 | 249 | POINT(3427927.975 5793243.875)
1 | 4 | 251 | POINT(3427927.775 5793243.825)
2 | 4 | 253 | POINT(3427927.825 5793243.825)
3 | 4 | 254 | POINT(3427927.875 5793243.825)
4 | 4 | 254 | POINT(3427927.925 5793243.825)
5 | 4 | 253 | POINT(3427927.975 5793243.825)
1 | 5 | 252 | POINT(3427927.775 5793243.775)
2 | 5 | 250 | POINT(3427927.825 5793243.775)
3 | 5 | 254 | POINT(3427927.875 5793243.775)
4 | 5 | 254 | POINT(3427927.925 5793243.775)
5 | 5 | 254 | POINT(3427927.975 5793243.775)

```

関連情報

[ST_DumpAsPolygons](#), [ST_PixelAsPolygon](#), [ST_PixelAsPolygons](#), [ST_PixelAsPoint](#), [ST_PixelAsPoints](#), [ST_Pixel](#)

11.6.7 ST_Value

ST_Value — 指定したバンドにおける `columnx`, `rowy` で指定したピクセルまたは指定したジオメトリポイントに対応するピクセルの値を返します。バンド番号は 1 始まりで、指定しない場合には、1 番と仮定します。`exclude_nodata_value` が FALSE に設定された場合には、**NODATA** ピクセルを含む全てのピクセルがインタセクトするかが考慮され、値を返します。`exclude_nodata_value` を渡さない場合には、ラスタのメタデータから読みます。

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);

```

説明

与えたバンドの、与えた列番号、行番号のピクセル、または与えたジオメトリのポイントにおける値を返します。バンド番号は 1 始まりで、指定しない場合には、1 番と仮定します。

`exclude_nodata_value` が TRUE の場合には、**nodata** 値でないピクセルだけを考慮します。`exclude_nodata_value` が FALSE の場合には、全てのピクセルが考慮されます。

The allowed values of the `resample` parameter are "nearest" which performs the default nearest-neighbor resampling, "bilinear" which performs a **bilinear interpolation** to estimate the value between pixel centers, and the nearest-neighbor boundary options "nearest-UL", "nearest-UR", "nearest-LL", and "nearest-LR". Boundary options choose which pixel is returned when the point lies on a horizontal or vertical pixel boundary, including corner intersections.

Enhanced: 3.7.0 resample accepts nearest-neighbor boundary options "nearest-UL", "nearest-UR", "nearest-LL", and "nearest-LR".

Enhanced: 3.2.0 任意引数 resample が追加されました。

Enhanced: 2.0.0 任意引数 exclude_nodata_value が追加されました。

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 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)))
```

関連情報

[ST_DumpValues](#), [ST_SetValue](#), [ST_DumpAsPolygons](#), [ST_NumBands](#), [ST_PixelAsPolygon](#), [ST_ScaleX](#), [ST_ScaleY](#), [ST_UpperLeftX](#), [ST_UpperLeftY](#), [ST_SRID](#), [ST_AsText](#), [ST_Point](#), [ST_MakeEnvelope](#), [ST_Intersects](#), [ST_Intersection](#)

11.6.8 ST_NearestValue

`ST_NearestValue` — 与えられたバンドの、`columnx` と `rowy` で指定されるか、またはラスタと同じ空間参照系で表現されたポイントで指定されたピクセルに最も近い `NODATA` でない値を返します。

Synopsis

```
double precision ST_NearestValue(raster rast, integer bandnum, geometry pt, boolean exclude_nodata_value=false);
double precision ST_NearestValue(raster rast, geometry pt, boolean exclude_nodata_value=true);
double precision ST_NearestValue(raster rast, integer bandnum, integer columnx, integer rowy, boolean exclude_nodata_value=true);
double precision ST_NearestValue(raster rast, integer columnx, integer rowy, boolean exclude_nodata_value=true);
```

説明

与えられたバンドの、`columnx` と `rowy` で指定されるか、またはラスタと同じ空間参照系で表現されたポイントで指定されたピクセルに最も近い **NODATA** でない値を返します。`columnx`, `rowy` ピクセルまたは指定したジオメトリポイントのピクセルが **NODATA** である場合には、この関数は、`columnx`, `rowy` ピクセルかジオメトリポイントのピクセルから最も近い **NODATA**. でないピクセルを探索します。

バンド番号は 1 始まりで、`bandnum` が指定されていない場合には、1 番と仮定します。`exclude_nodata_value` が **FALSE** に設定された場合には、**NODATA** ピクセルを含む全てのピクセルがインタセクトするかが考慮され、値を返します。`exclude_nodata_value` を渡さない場合には、ラスタのメタデータから読みます。

Availability: 2.1.0



Note
`ST_NearestValue` は `ST_Value` と交換可能です。

例

This example reads a value from pixel 2x2.

```
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

This example reads a NODATA value from pixel 2x3.

```
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
```

```

),
2, 3, 0.
),
3, 5, 0.
),
4, 2, 0.
),
5, 4, 0.
) AS rast
) AS foo

```

```

value | nearestvalue
-----+-----
      |          1

```

関連情報

[ST_Neighborhood](#), [ST_Value](#)

11.6.9 ST_SetZ

ST_SetZ — 入力ジオメトリと同じ X/y 座標値と、指定されたりサンプリングアルゴリズムを使ってラスタから複写された Z 値とを持つジオメトリを返します。

Synopsis

geometry **ST_SetZ**(raster rast, geometry geom, text resample=nearest, integer band=1);

説明

入力ジオメトリと同じ X/Y 座標値と、指定されたりサンプリングアルゴリズムを使ってラスタから複写された Z 値とを持つジオメトリを返します。

resample パラメータの取ることができる値は、デフォルトである最近傍探索リサンプリングを行う“nearest”、バイリニア補間 ([WikiPedia 英語版](#)) を行う“bilinear”です。近隣セルも考慮に入れた値を計算するためのものです。

Availability: 3.2.0

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

```

```
LINESTRING Z (1 1.9 38,1 0.2 27)
```

関連情報

[ST_Value](#), [ST_SetM](#)

11.6.10 ST_SetM

ST_SetM — 入力ジオメトリと同じ X/Y 値を持ち、かつ、指定されたりサンプリングアルゴリズムを使ってラスタから複写された M 値を持つジオメトリを返します。

Synopsis

geometry **ST_SetM**(raster rast, geometry geom, text resample=nearest, integer band=1);

説明

入力ジオメトリと同じ X/Y 値を持ち、かつ、指定されたりサンプリングアルゴリズムを使ってラスタから複写された M 値を持つジオメトリを返します。

resample パラメータの取ることができる値は、デフォルトである最近傍探索リサンプリングを行う **"nearest"**、バイリニア補間 ([WikiPedia 英語版](#)) を行う **"bilinear"** です。近隣セルも考慮に入れた値を計算するためのものです。

Availability: 3.2.0

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

```

```
LINESTRING M (1 1.9 38,1 0.2 27)
```

関連情報

[ST_Value](#), [ST_SetZ](#)

11.6.11 ST_Neighborhood

ST_Neighborhood — 与えられたバンドの `columnX`, `columnY` か、ラスタと同じ空間参照系のジオメトリポイントで指定されたピクセルの周囲にある、**NODATA** でない 2 次元倍精度浮動小数点数配列を返します。

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);

```

説明

与えられたバンドの `columnX`, `columnY` か、ラスタと同じ空間参照系のジオメトリポイントで指定されたピクセルの周囲にある、**NODATA** でない 2 次元倍精度浮動小数点数配列を返します。`distanceX` と `distanceY` 引数は、たとえば、対象ピクセルから X 軸に沿って 3 ピクセルとして Y 軸に沿って 2 ピクセルというふうに、指定したピクセルの周囲のピクセル数を、それぞれ X 方向と Y 方向に定義します。2 次元配列の中心の値は `columnX`, `columnY` またはジオメトリポイントで指定したピクセルの値です。

バンド番号は 1 始まりで、`bandnum` が指定されていない場合には、1 番と仮定します。`exclude_nodata_value` が `FALSE` に設定された場合には、**NODATA** ピクセルを含む全てのピクセルがインタセクトするかが考慮され、値を返します。`exclude_nodata_value` を渡さない場合には、ラスタのメタデータから読みます。



Note

返される 2 次元配列の各軸の要素数は $2 * (\text{distanceX}|\text{distanceY}) + 1$ です。`distanceX` と `distanceY` を 1 にすると、3x3 の配列が返ります。



Note

2 次元配列の出力は `ST_Min4ma`, `ST_Sum4ma`, `ST_Mean4ma` といったあらゆるラスタ処理関数に渡すことができます。

Availability: 2.1.0

例

This example reads a value from pixel 2x2.

```
SELECT
  ST_Neighborhood(rast, 2, 2, 1, 1)
FROM (
  SELECT
    ST_SetValues(ST_AddBand(ST_MakeEmptyRaster(5, 5, -2, 2, 1, -1, 0, 0, 0),
      '8BUI'::text, 1, 0
    ),
    1, 1, 1, ARRAY[
      [0, 1, 1, 1, 1],
      [1, 1, 1, 0, 1],
      [1, 0, 1, 1, 1],
      [1, 1, 1, 1, 0],
      [1, 1, 0, 1, 1]
    ]::double precision[],
    1
  ) AS rast
) AS foo
```

```
{{NULL,1,1},{1,1,1},{1,NULL,1}}
```

This example reads a NODATA value from pixel 2x3.

```
SELECT
  ST_Neighborhood(rast, 2, 3, 1, 1)
FROM (
  SELECT
    ST_SetValues(ST_AddBand(ST_MakeEmptyRaster(5, 5, -2, 2, 1, -1, 0, 0, 0),
      '8BUI'::text, 1, 0
```

```

    ),
    1, 1, 1, ARRAY[
        [0, 1, 1, 1, 1],
        [1, 1, 1, 0, 1],
        [1, 0, 1, 1, 1],
        [1, 1, 1, 1, 0],
        [1, 1, 0, 1, 1]
    ]::double precision[],
    1
) AS rast
) AS foo

```

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

```

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

関連情報

[ST_NearestValue](#), [ST_Min4ma](#), [ST_Max4ma](#), [ST_Sum4ma](#), [ST_Mean4ma](#), [ST_Range4ma](#), [ST_Distinct4ma](#), [ST_StdDev4ma](#)

11.6.12 ST_SetValue

ST_SetValue — 与えられたバンドの **columnX**, **columnY** か、ラスタと同じ空間参照系のジオメトリポイントで指定されたピクセルの値または指定したジオメトリとインタセクトするピクセル群の値を設定することから得られる、変更されたラスタを返します。バンド番号は 1 始まりで、指定しない場合には、1 番と仮定します。

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);

```


説明

指定されたバンドの、与えられたラスタの行と列またはジオメトリで指定したピクセルの値を新しい値に設定することで得られる、変更したラスタを返します。バンドが指定されていない場合には、1番と仮定します。

Enhanced: 2.1.0 `ST_SetValue` でジオメトリを用いる形式が、ポイントだけでなくあらゆるジオメトリタイプに対応するようになりました。ジオメトリを用いる形式は `ST_SetValues` の `geomval[]` を用いる形式をラップしたものです。

例

This example uses geometry input.

```
SELECT (foo.geomval).val, ST_AsText(ST_Union((foo.geomval).geom))
FROM (SELECT ST_DumpAsPolygons(ST_SetValue(
    rast,
    1,
    ST_Point(3427927.75, 5793243.95),
    50)
    ) As geomval
FROM dummy_rast
where rid = 2) As foo
WHERE (foo.geomval).val < 250
GROUP BY (foo.geomval).val;
```

val	st_astext
50	POLYGON((3427927.75 5793244,3427927.75 5793243.95,3427927.8 579324 ...
249	POLYGON((3427927.95 5793243.95,3427927.95 5793243.85,3427928 57932 ...

This example stores the changed raster.

```
UPDATE dummy_rast SET rast = ST_SetValue(rast,1, ST_Point(3427927.75, 5793243.95),100)
WHERE rid = 2 ;
```

関連情報

[ST_Value](#), [ST_DumpAsPolygons](#)

11.6.13 ST_SetValues

`ST_SetValues` — 与えられたバンドに複数の値を設定して、変更されたラスタを返します。

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);
```

説明

指定したバンドの指定したピクセルに新しい値を設定した結果として変更されたラスタを返します。`columnx` と `rowy` は 1 始まりです。

`keepnodata` が TRUE の場合には、NODATA 値を持つピクセルは `newvalueset` に一致する値に設定しません。

一つ目の形式では、設定するピクセルを `columnx`, `rowy` のピクセル座標で決定し、範囲を `newvalueset` 配列で決定します。`noset` によって、ピクセルを `newvalueset` 内にある値で、一部を設定されないようにすることができます (PostgreSQL は不調和配列、ジャグ配列を許さないため)。一つ目の形式の例をご覧ください。

二つ目の形式では、一つ目の形式と似ていますが、倍精度浮動小数点数のスカラ値である `nosetvalue` を `noset` 配列の代わりに使う点が違います。`nosetvalue` の値になる `newvalueset` 内の要素の設定は行いません。二つ目の形式の例をご覧ください。

三つ目の形式では、設定するピクセルを `columnx`, `rowy` のピクセル座標と `width`, `height`. で決定します。三つ目の形式の例をご覧ください。

四つ目の形式では、`rast` の 1 番バンドのピクセルを設定すると仮定する点を除いては、三つ目の形式と同じです。

五つ目の形式では、設定するピクセルを `geomval` の配列で決定します。配列内の全てのジオメトリのタイプが POINT または MULTIPOINT である場合には、ポイントごとの経度と緯度がピクセルの設定に直接使うためのショートカットに使われます。他の場合には、ジオメトリはラスタに変換され一つずつ渡されます。五つ目の形式の例をご覧ください。

Availability: 2.1.0

例: 一つ目の形式

```
/*
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	
2	3	9
3	1	9
3	2	9
3	3	9

```
/*
The ST_SetValues() does the following...
```

```

+ - + - + - +      + - + - + - +
| 1 | 1 | 1 |      | 9 | 9 | 9 |
+ - + - + - +      + - + - + - +
| 1 | 1 | 1 |      => | 1 |   | 9 |
+ - + - + - +      + - + - + - +
| 1 | 1 | 1 |      | 9 | 9 | 9 |
+ - + - + - +      + - + - + - +
*/
SELECT
    (poly).x,
    (poly).y,
    (poly).val
FROM (
    SELECT
        ST_PixelAsPolygons(ST_SetValues(ST_AddBand(ST_MakeEmptyRaster(3, 3, 0, 0, 1, -1, 0, 0, ←
            0),
            1, '8BUI', 1, 0
        ),
        1, 1, 1,
        ARRAY[[9, 9, 9], [9, NULL, 9], [9, 9, 9]]::double precision[][],
        ARRAY[[false], [true]]::boolean[][])
    ) AS poly
) foo
ORDER BY 1, 2;

```

```

x | y | val
---+---+---
1 | 1 | 9
1 | 2 | 1
1 | 3 | 9
2 | 1 | 9
2 | 2 |
2 | 3 | 9
3 | 1 | 9
3 | 2 | 9
3 | 3 | 9

```

```

/*
The ST_SetValues() does the following...

+ - + - + - +      + - + - + - +
|   | 1 | 1 |      |   | 9 | 9 |
+ - + - + - +      + - + - + - +
| 1 | 1 | 1 |      => | 1 |   | 9 |
+ - + - + - +      + - + - + - +
| 1 | 1 | 1 |      | 9 | 9 | 9 |
+ - + - + - +      + - + - + - +
*/
SELECT
    (poly).x,
    (poly).y,
    (poly).val
FROM (
    SELECT
        ST_PixelAsPolygons(ST_SetValues(ST_SetValue(ST_AddBand(ST_MakeEmptyRaster(3, 3, 0, 0, ←
            1, -1, 0, 0, 0),
            1, '8BUI', 1, 0
        ),
        1, 1, 1, NULL
    ),

```

```
1, 1, 1,
  ARRAY[[9, 9, 9], [9, NULL, 9], [9, 9, 9]]::double precision[][],
  ARRAY[[false], [true]]::boolean[][],
  TRUE
)
) AS poly
) foo
ORDER BY 1, 2;
```

x	y	val
1	1	
1	2	1
1	3	9
2	1	9
2	2	
2	3	9
3	1	9
3	2	9
3	3	9

例: 二つ目の形式

```
/*
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[][], -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

```

例: 三つ目の形式

```

/*
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...

```

<pre> + - + - + - + 1 1 1 + - + - + - + 1 1 + - + - + - + 1 1 1 + - + - + - + </pre>	=>	<pre> + - + - + - + 1 1 1 + - + - + - + 1 9 + - + - + - + 1 9 9 + - + - + - + </pre>

```

*/
SELECT
    (poly).x,
    (poly).y,
    (poly).val
FROM (
SELECT
    ST_PixelAsPolygons(ST_SetValues(ST_SetValue(ST_AddBand(ST_MakeEmptyRaster(3, 3, 0, 0, 0,
        1, -1, 0, 0, 0),
        1, '8BUI', 1, 0
    ),
        1, 2, 2, NULL
    ),
        1, 2, 2, 2, 2, 9, TRUE
    )
) AS poly
) foo
ORDER BY 1, 2;

```

x	y	val
1	1	1
1	2	1
1	3	1

2	1	1
2	2	
2	3	9
3	1	1
3	2	9
3	3	9

例: 五つ目の形式

```
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, 2, 2, 2, NULL} }")
1	3	(1, "{ {3, 3, 3, 3, 3}, {3, NULL, NULL, NULL, NULL}, {3, NULL, NULL, NULL, NULL}, {3, NULL, NULL, NULL, NULL}, {3, 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, 4} }")
(4 rows)		

配列内における後の geomvals で前の geomvals を上書きできることを示しています。

```
WITH foo AS (  
  SELECT 1 AS rid, ST_AddBand(ST_MakeEmptyRaster(5, 5, 0, 0, 1, -1, 0, 0, 0), 1, '8BUI',  
    0, 0) AS rast  
) , bar AS (  
  SELECT 1 AS gid, 'SRID=0;POINT(2.5 -2.5)::geometry geom UNION ALL  
  SELECT 2 AS gid, 'SRID=0;POLYGON((1 -1, 4 -1, 4 -4, 1 -4, 1 -1))::geometry geom UNION  
    ALL  
  SELECT 3 AS gid, 'SRID=0;POLYGON((0 0, 5 0, 5 -1, 1 -1, 1 -4, 0 -4, 0 0))::geometry  
    geom UNION ALL  
  SELECT 4 AS gid, 'SRID=0;MULTIPOINT(0 0, 4 4, 4 -4)::geometry  
)  
SELECT  
  t1.rid, t2.gid, t3.gid, ST_DumpValues(ST_SetValues(rast, 1, ARRAY[ROW(t2.geom, t2.gid),  
    ROW(t3.geom, t3.gid)]::geomval[]))  
FROM foo t1  
CROSS JOIN bar t2  
CROSS JOIN bar t3  
WHERE t2.gid = 1
```



```
AND t3.gid = 2
ORDER BY t1.rid, t2.gid, t3.gid;
```

rid	gid	gid	st_dumpvalues
1	1	2	(1,"{{NULL,NULL,NULL,NULL,NULL},{NULL,2,2,2,NULL},{NULL,2,2,2,NULL},{ NULL,2,2,2,NULL},{NULL,NULL,NULL,NULL,NULL}}")
(1 row)			

この例は前の例の逆です。

```
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' geom
)
SELECT
  t1.rid, t2.gid, t3.gid, ST_DumpValues(ST_SetValues(rast, 1, ARRAY[ROW(t2.geom, t2.gid),
    ROW(t3.geom, t3.gid)]::geomval[]))
FROM foo t1
CROSS JOIN bar t2
CROSS JOIN bar t3
WHERE t2.gid = 2
AND t3.gid = 1
ORDER BY t1.rid, t2.gid, t3.gid;
```

rid	gid	gid	st_dumpvalues
1	2	1	(1,"{{NULL,NULL,NULL,NULL,NULL},{NULL,2,2,2,NULL},{NULL,2,1,2,NULL},{ NULL,2,2,2,NULL},{NULL,NULL,NULL,NULL,NULL}}")
(1 row)			

関連情報

[ST_Value](#), [ST_SetValue](#), [ST_PixelAsPolygons](#)

11.6.14 ST_DumpValues

ST_DumpValues — 指定したバンドの値を 2 次元で得ます。

Synopsis

```
setof record ST_DumpValues( raster rast , integer[] nband=NULL , boolean exclude_nodata_value=true
);
double precision[][] ST_DumpValues( raster rast , integer nband , boolean exclude_nodata_value=true
);
```

説明

指定したバンドの値を 2 次元で得ます (一つ目の添え字で行、二つ目の添え字で列に、それぞれ対応します)。nband が NULL または指定されていない場合には、全てのラスタブンドが処理されます。

Availability: 2.1.0

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;
```

5

関連情報

[ST_Value](#), [ST_SetValue](#), [ST_SetValues](#)

11.6.15 ST_PixelOfValue

ST_PixelOfValue — 検索値と同じ値を持つピクセルの columnx, rowy ピクセル座標を得ます。

Synopsis

```
setof record ST_PixelOfValue( raster rast , integer nband , double precision[] search , boolean exclude_nodata_value=true );
setof record ST_PixelOfValue( raster rast , double precision[] search , boolean exclude_nodata_value=true );
setof record ST_PixelOfValue( raster rast , integer nband , double precision search , boolean exclude_nodata_value=true );
setof record ST_PixelOfValue( raster rast , double precision search , boolean exclude_nodata_value=true );
```

説明

検索値と同じ値を持つピクセルの columnx, rowy ピクセル座標を得ます。バンドを指定しない場合には、1 番と仮定します。

Availability: 2.1.0

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 ラスタエディタ

11.7.1 ST_SetGeoReference

ST_SetGeoReference — 地理参照 6 パラメーターを一度に設定します。数値は空白で区切ります。GDAL または ESRI 書式の入力を受け付けます。デフォルトは GDAL です。

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);

```

説明

地理参照 6 パラメーターを一度に設定します。'GDAL' または 'ESRI' 書式の入力を受け付けます。デフォルトは GDAL です。6 パラメーターが与えられない場合には、NULL を返します。

書式の表現の違いは次の通りです。

GDAL:

```
scalex skewy skewx scaley upperleftx upperlefty
```

ESRI:

```
scalex skewy skewx scaley upperleftx + scalex*0.5 upperlefty + scaley*0.5
```



Note

ラスタがデータベース外のバンドを持っている場合には、地理参照の変更によって、バンドの外部保存されているデータに正しくアクセスできなくなることがあります。

Enhanced: 2.1.0 **ST_SetGeoReference**(raster, double precision, ...) 形式を追加しました。

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	5	5	10	-10	0	↔
3	0.001	0	1	1	5	5	10	-10	0.001	↔

関連情報

[ST_GeoReference](#), [ST_ScaleX](#), [ST_ScaleY](#), [ST_UpperLeftX](#), [ST_UpperLeftY](#)

11.7.2 ST_SetRotation

ST_SetRotation — ラスタの回転をラジアン単位で設定します。

Synopsis

```
raster ST_SetRotation(raster rast, float8 rotation);
```

説明

ラスタを回転させます。回転はラジアン単位です。詳細については[World File](#) (英語版 Wikipedia) をご覧下さい。

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	st_scalex	↔
-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	

関連情報

[ST_Rotation](#), [ST_ScaleX](#), [ST_ScaleY](#), [ST_SkewX](#), [ST_SkewY](#)

11.7.3 ST_SetScale

ST_SetScale — ピクセルサイズの X 値と Y 値を空間参照系の単位で設定します。数値は単位/ピクセルの幅または高さです。

Synopsis

raster **ST_SetScale**(raster rast, float8 xy);
raster **ST_SetScale**(raster rast, float8 x, float8 y);

説明

ピクセルサイズの X 値と Y 値を空間参照系の単位で設定します。数値は単位/ピクセルの幅または高さです。一つだけ渡した場合には、X 値と Y 値は同じ値に設定されます。



Note
ST_SetScale はラスタの範囲をあわせるためのリサンプリングをしない点で [ST_Rescale](#) と異なります。根本的に誤った設定を行ったのを訂正するためにラスタのメタデータ (地理参照) を変更するだけです。ST_Rescale は、入力ラスタの地理範囲に合わせて計算された幅、高さを持つラスタを返します。ST_SetScale はラスタの幅も高さも変更しません。

Changed: 2.0.0 WKTRaster 版では、ST_SetPixelSize と呼ばれていました。2.0.0 で変更されました。

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)
```

関連情報

[ST_ScaleX](#), [ST_ScaleY](#), [Box3D](#)

11.7.4 ST_SetSkew

ST_SetSkew — 地理参照のスキュー (回転パラメーター) の X 値と Y 値を設定します。一つだけ渡した場合には、X 値と Y 値は同じ値に設定されます。

Synopsis

```
raster ST_SetSkew(raster rast, float8 skewxy);
raster ST_SetSkew(raster rast, float8 skewx, float8 skewy);
```

説明

地理参照のスキュー (回転パラメーター) の X 値と Y 値を設定します。一つだけ渡した場合には、X 値と Y 値は同じ値に設定されます。詳細については [World File](#) (英語版 Wikipedia) をご覧ください。

例

This is the first example.

```
UPDATE dummy_rast SET rast = ST_SetSkew(rast,1,2) WHERE rid = 1;
SELECT rid, ST_SkewX(rast) As skewx, ST_SkewY(rast) As skewy,
       ST_GeoReference(rast) as georef
FROM dummy_rast WHERE rid = 1;
```

```
rid | skewx | skewy | georef
-----+-----+-----+-----
  1 |      1 |      2 | 2.0000000000
      : 2.0000000000
      : 1.0000000000
      : 3.0000000000
      : 0.5000000000
      : 0.5000000000
```

This second example sets both values to the same number.

```
UPDATE dummy_rast SET rast = ST_SetSkew(rast,0) WHERE rid = 1;
SELECT rid, ST_SkewX(rast) As skewx, ST_SkewY(rast) As skewy,
       ST_GeoReference(rast) as georef
FROM dummy_rast WHERE rid = 1;
```

rid	skewx	skewy	georef
1	0	0	2.0000000000 : 0.0000000000 : 0.0000000000 : 3.0000000000 : 0.5000000000 : 0.5000000000

関連情報

[ST_GeoReference](#), [ST_SetGeoReference](#), [ST_SkewX](#), [ST_SkewY](#)

11.7.5 ST_SetSRID

ST_SetSRID — スタの SRID を `spatial_ref_sys` に定義されている特定の整数値に設定します。

Synopsis

raster **ST_SetSRID**(raster rast, integer srid);

説明

ラスタの SRID を特定の整数値に設定します。



Note

ラスタの投影変換は行いません。単にメタデータを空間参照系の定義されている値に設定するだけです。後で投影変換を行う場合に使います。

関連情報

Section [4.5](#), [ST_SRID](#)

11.7.6 ST_SetUpperLeft

ST_SetUpperLeft — ラスタの左上隅の投影座標系の X 値と Y 値を設定します。

Synopsis

raster **ST_SetUpperLeft**(raster rast, double precision x, double precision y);

説明

ラスタの左上隅の投影座標系の X 値と Y 値を設定します。

例

```
SELECT ST_SetUpperLeft(rast, -71.01, 42.37)
FROM dummy_rast
WHERE rid = 2;
```

関連情報

[ST_UpperLeftX](#), [ST_UpperLeftY](#)

11.7.7 ST_Resample

ST_Resample — 指定したリサンプリングアルゴリズム、新しいピクセル範囲、グリッドの隅、定義するか他のラスタから借りてきた地理参照属性を使ってリサンプリングを行います。

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);

説明

指定したリサンプリングアルゴリズム、新しいピクセル範囲 (width & height)、グリッド隅 (gridx & gridy)、定義するか他のラスタから借りてきた地理参照属性 (scalex, scaley, skewx, skewy) を使ってリサンプリングを行います。参照ラスタを使用する場合には、二つのラスタは同じ SRID でなければなりません。

新しいピクセル値は次のリサンプリングアルゴリズムの一つを使って計算されます:

- NearestNeighbor (最近傍補間、英語又は米式綴り方)
- Bilinear (双線形補間)
- Cubic (3 次補完)
- CubicSpline (3 次スプライン補完)
- Lanczos (ランチョス補間)
- Max (最大)
- Min (最小)

デフォルトは `NearestNeighbor` (最近傍補間) です。最も実行速度が速いですが、結果の補間が最も悪くなります。

`maxerr` が指定されていない場合には 0.125 とします。



Note

詳細については [GDAL Warp resampling methods](#) をご覧下さい。

Availability: 2.0.0 GDAL 1.6.1 以上が必要です。

Enhanced: 3.4.0 リサンプリング選択肢に最大と最小を追加

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

関連情報

[ST_Rescale](#), [ST_Resize](#), [ST_Transform](#)

11.7.8 ST_Rescale

`ST_Rescale` — スケール (ピクセルサイズ) だけを調整するリサンプリングを行います。新しいピクセル値のリサンプリングアルゴリズムとして最近傍補間 ('`NearestNeighbor`' (英語または米式綴り方))、双線形補間 ('`Bilinear`'), 3 次補間 ('`Cubic`'), 3 次スプライン補間 ('`CubicSpline`'), ランチョス補間 ('`Lanczos`') を用います。デフォルトは最近傍補間です。

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);

説明

スケール (またはピクセルサイズ) だけを調整するリサンプリングを行います。新しいピクセル値は次のリサンプリングアルゴリズムを使って計算されます:

- NearestNeighbor (最近傍補間、英語又は米式綴り方)
- Bilinear (双線形補間)
- Cubic (3 次補完)
- CubicSpline (3 次スプライン補完)
- Lanczos (ランチョス補間)
- Max (最大)
- Min (最小)

デフォルトは NearestNeighbor (最近傍補間) です。最も実行速度が速いですが、結果の補間が最も悪くなります。

`scalex` と `scaley` で、新しいピクセルサイズを定義します。ラスタを正しい方向にするには、`scaley` は負数でなければならないことがしばしばあります。

新しい `scalex` または `scaley` がラスタの `width` または `height` の除数でない時、結果ラスタの範囲は元のラスタの範囲を含むために拡大されます。 **ST_Resize** をご覧ください。

`maxerr` は、リサンプリングアルゴリズムで変換を行う際に近似を行うかどうかを決めるしきい値です (ピクセル単位)。 `maxerr` を設定しない場合には、デフォルトの 0.125 が使用され、この値は GDAL の `gdalwarp` ユーティリティで使われる値と同じです。0 に設定した場合には、近似は行われません。



Note

詳細については [GDAL Warp resampling methods](#) をご覧下さい。



Note

`ST_Recalc` は、`ST_SetScale` はラスタ範囲に合わせるためのリサンプリングをしない点で、**ST_SetScale** と異なります。`ST_SetScale` は根本的に誤った設定を行ったのを訂正するためにラスタのメタデータ (地理参照) を変更するだけです。`ST_Rescale` は、入力ラスタの地理範囲に合わせて計算された幅、高さを持つラスタを返します。`ST_SetScale` はラスタの幅も高さも変更しません。

Availability: 2.0.0 GDAL 1.6.1 以上が必要です。

Enhanced: 3.4.0 リサンプリング選択肢に最大と最小を追加

Changed: 2.1.0 SRID なしのラスタで動作するようになりました。

例

ラスタのピクセルサイズを 0.001 度から 0.0015 度にスケール再設定を行う例です。

This example returns the original raster pixel size.

```
SELECT ST_PixelWidth(ST_AddBand(ST_MakeEmptyRaster(100, 100, 0, 0, 0.001, -0.001, 0, 0, 4269), '8BUI'::text, 1, 0)) width
```

0.001

This example returns the pixel size of the rescaled raster.

```
SELECT ST_PixelWidth(ST_Rescale(ST_AddBand(ST_MakeEmptyRaster(100, 100, 0, 0, 0.001, ←
-0.001, 0, 0, 4269), '8BUI'::text, 1, 0), 0.0015)) width
```

0.0015

関連情報

[ST_Resize](#), [ST_Resample](#), [ST_SetScale](#), [ST_ScaleX](#), [ST_ScaleY](#), [ST_Transform](#)

11.7.9 ST_Reskew

ST_Reskew — キュー (回転パラメーター) だけを調整するリサンプリングを行います。新しいピクセル値のリサンプリングアルゴリズムとして最近傍補間 ('NearestNeighbor' (米式綴り方))、双線形補間 ('Bilinear')、3 次補間 ('Cubic')、3 次スプライン補間 ('CubicSpline')、ランチョス補間 ('Lanczos') を用います。デフォルトは最近傍補間です。

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);

説明

スキュー (回転パラメーター) だけを調整するリサンプリングを行います。新しいピクセル値のリサンプリングアルゴリズムとして最近傍補間 ('NearestNeighbor' (米式綴り方))、双線形補間 ('Bilinear')、3 次補間 ('Cubic')、3 次スプライン補間 ('CubicSpline')、ランチョス補間 ('Lanczos') を用います。デフォルトは最も早いですが最も悪い内挿を行う最近傍補間です。

skewx と **skewy** によって新しいスキューが定義されます。

新しいラスタの範囲は元のラスタの範囲を含みます。

maxerr が指定されていない場合には 0.125 とします。



Note

詳細については [GDAL Warp resampling methods](#) をご覧下さい。



Note

ST_Reskew は、**ST_SetSkew** がラスタの範囲をあわせるためのリサンプリングをしない点で **ST_SetSkew** と異なります。根本的に誤った設定を行ったのを訂正するためにラスタのメタデータ (地理参照) を変更するだけです。**ST_Reskew** は、入力ラスタの地理範囲に合わせて計算された幅、高さを持つラスタを返します。**ST_SetSkew** はラスタの幅も高さも変更しません。

Availability: 2.0.0 GDAL 1.6.1 以上が必要です。

Changed: 2.1.0 SRID なしのラスタで動作するようになりました。

例

スキューを 0.0 から 0.0015 に再設定する簡単な例です。

This example uses the original non-rotated raster.

```
SELECT ST_Rotation(ST_AddBand(ST_MakeEmptyRaster(100, 100, 0, 0, 0.001, -0.001, 0, 0, 4269) ←
, '8BUI'::text, 1, 0));

0

-- the reskewed raster raster rotation
SELECT ST_Rotation(ST_Reskew(ST_AddBand(ST_MakeEmptyRaster(100, 100, 0, 0, 0.001, -0.001, ←
0, 0, 4269), '8BUI'::text, 1, 0), 0.0015));

-0.982793723247329
```

関連情報

[ST_Resample](#), [ST_Rescale](#), [ST_SetSkew](#), [ST_SetRotation](#), [ST_SkewX](#), [ST_SkewY](#), [ST_Transform](#)

11.7.10 ST_SnapToGrid

ST_SnapToGrid — グリッドにスナップすることでラスタをリサンプリングします。新しいピクセル値のリサンプリングアルゴリズムとして最近傍補間 ('NearestNeighbor' (米式綴り方))、双線形補間 ('Bilinear')、3 次補間 ('Cubic')、3 次スプライン補間 ('CubicSpline')、ランチョス補間 ('Lanczos') を用います。デフォルトは最近傍補間です。

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);
```

説明

任意のピクセル隅 (gridx & gridy) と、任意引数としてピクセルサイズ (scalex & scaley) によって定義されるグリッドにスナップすることでラスタをリサンプリングします。新しいピクセル値のリサンプリングアルゴリズムとして最近傍補間 ('NearestNeighbor' (米式綴り方))、双線形補間 ('Bilinear')、3 次補間 ('Cubic')、3 次スプライン補間 ('CubicSpline')、ランチョス補間 ('Lanczos') を用います。デフォルトは最も早いですが最も悪い内挿を行う最近傍補間です。

gridx と gridy は、新しいグリッドの任意のピクセル隅を定義します。これは新しいラスタの左上隅に必ずなるものではありません。新しいラスタ範囲の内部または境界である必要はありません。

任意引数として、新しいグリッドのピクセルサイズを scalex と scaley で指定することができます。

新しいラスタの範囲は元のラスタの範囲を含みます。

maxerr が指定されていない場合には 0.125 とします。

**Note**

詳細については [GDAL Warp resampling methods](#) をご覧下さい。

**Note**

グリッドパラメーターより多くの制御が必要な場合には、[ST_Resample](#) を使います。

Availability: 2.0.0 GDAL 1.6.1 以上が必要です。

Changed: 2.1.0 SRID なしのラスタで動作するようになりました。

例

ラスタをわずかに異なるグリッドにスナップさせる簡単な例です。

This example returns the upper-left X coordinate of the original raster.

```
SELECT ST_UpperLeftX(ST_AddBand(ST_MakeEmptyRaster(10, 10, 0, 0, 0.001, -0.001, 0, 0, 4269) ←
, '8BUI'::text, 1, 0));
0

-- the upper left of raster after snapping
SELECT ST_UpperLeftX(ST_SnapToGrid(ST_AddBand(ST_MakeEmptyRaster(10, 10, 0, 0, 0.001, ←
-0.001, 0, 0, 4269), '8BUI'::text, 1, 0), 0.0002, 0.0002));
-0.0008
```

関連情報

[ST_Resample](#), [ST_Rescale](#), [ST_UpperLeftX](#), [ST_UpperLeftY](#)

11.7.11 ST_Resize

ST_Resize — ラスタを新しい幅、高さにサイズ再設定を行います。

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);
```

説明

ラスタを新しい幅、高さにサイズ再設定を行います。新しい幅、高さはピクセル数で確実に指定するか、ラスタの幅、高さの比率で指定します。新しいラスタの範囲は元のラスタの範囲と同じです。

新しいピクセル値のリサンプリングアルゴリズムとして最近傍補間 ('NearestNeighbor' (米式綴り方))、双線形補間 ('Bilinear')、3 次補間 ('Cubic')、3 次スプライン補間 ('CubicSpline')、ランチョス補間 ('Lanczos') を用います。デフォルトは最も早いですが最も悪い内挿を行う最近傍補間です。

一つ目の形式では、出力ラスタの実際の幅、高さを予定しています。

二つ目の形式では、0 から 1 の間の値で、入力ラスタの幅、高さに対する比率を指定しています。

三つ目の形式では、出力ラスタの実際の幅、高さを取るか、文字列による入力ラスタの幅、高さに対する百分率 ("20%") を取ります。

Availability: 2.1.0 GDAL 1.6.1 以上が必要です。

例

```
WITH foo AS(
SELECT
  1 AS rid,
  ST_Resize(ST_AddBand(ST_MakeEmptyRaster(1000, 1000, 0, 0, 1, -1, 0, 0, 0),
    1, '8BUI', 255, 0
  ),
  '50%', '500') AS rast
UNION ALL
SELECT
  2 AS rid,
  ST_Resize(ST_AddBand(ST_MakeEmptyRaster(1000, 1000, 0, 0, 1, -1, 0, 0, 0),
    1, '8BUI', 255, 0
  ),
  500, 100) AS rast
UNION ALL
SELECT
  3 AS rid,
  ST_Resize(ST_AddBand(ST_MakeEmptyRaster(1000, 1000, 0, 0, 1, -1, 0, 0, 0),
    1, '8BUI', 255, 0
  ),
  0.25, 0.9) AS rast
), bar AS (
  SELECT rid, ST_MetaData(rast) AS meta, rast FROM foo
)
SELECT rid, (meta).* FROM bar
```

rid	upperleftx	upperlefty	width	height	scalex	scaley	skewx	skewy	srid	↵
numbands										
1	0	0	500	500	2	-2	0	0	0	↵
	1									
2	0	0	500	100	2	-10	0	0	0	↵
	1									
3	0	0	250	900	4	-1.1111111111111112			0	↵
	0	0	1							
(3 rows)										

関連情報

[ST_Resample](#), [ST_Rescale](#), [ST_Reskew](#), [ST_SnapToGrid](#)

11.7.12 ST_Transform

ST_Transform — ラスタを既知の空間参照系から他の既知の空間参照系に、指定したリサンプリングアルゴリズムで投影変換します。新しいピクセル値のリサンプリングアルゴリズムとして最近傍補間 ('NearestNeighbor' (米式綴り方))、双線形補間 ('Bilinear')、3 次補間 ('Cubic')、3 次スプライン補間 ('CubicSpline')、ランチョス補間 ('Lanczos') を用います。デフォルトは最近傍補間です。

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);
```

説明

ラスタを既知の空間参照系から他の既知の空間参照系に、指定したリサンプリングアルゴリズムで投影変換します。algorithm が指定されていない場合には 'NearestNeighbor' とし、maxerror が指定されていない場合には 0.125 とします。

アルゴリズム指定オプションは 'NearestNeighbor', 'Bilinear', 'Cubic', 'CubicSpline', 'Lanczos' です。詳細については [GDAL Warp resampling methods](#) をご覧ください。

ST_Transform はしばしば ST_SetSRID と混同されます。ST_Transform は、ある空間参照系から別の空間参照系に変更する際に、実際にラスタの座標を変更します (かつピクセル値のリサンプリングを行います) が、ST_SetSRID は単にラスタの空間参照系識別子を変更するだけです。

三つ目の形式は、他の形式と違い、alignto として参照ラスタが求められます。投影変換したラスタは参照ラスタの空間参照系 (SRID) に投影変換して、参照ラスタと同じアラインメントを持つようにします (ST_SameAlignment が TRUE になるようにします)。

Note



変換しようとしてうまくいかない場合には、環境変数 PROJSO を使用している PostGIS のビルドが使用している投影ライブラリに設定する必要があるかも知れません。ファイル名を指定します。Windows ではコントロールパネル -> システム -> システムの詳細設定 -> 詳細設定タブ -> 環境変数で PROJSO というシステム変数を追加して libproj.dll へのパスを指定します。変更後は PostgreSQL サービス/デーモンを再起動する必要があります。



Warning

タイルのカバレッジを変換する時には、ほとんどの場合で、「例: 三つ目の形式」で示す通り、同じアラインメントになり、かつタイル内に隙間がなくなるようにするために、参照ラスタを使用したくなります。

Availability: 2.0.0 GDAL 1.6.1 以上が必要です。

Enhanced: 2.1.0 ST_Transform(rast, alignto) の形式を追加しました。

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

 元のラスタはメートル単位のマサチューセッツ州平面 (<i>mass_stm</i>)	 WGS84 経度緯度に変換した後 (<i>wgs_84</i>)	 最近傍補間の代わりに双線形補間を使って WGS84 経度緯度に変換した後 (<i>wgs_84_bilin</i>)

例: 三つ目の形式

次に示す例は、ST_Transform(raster, srid) と ST_Transform(raster, alignto) との違いを示しています。

```
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
```

```

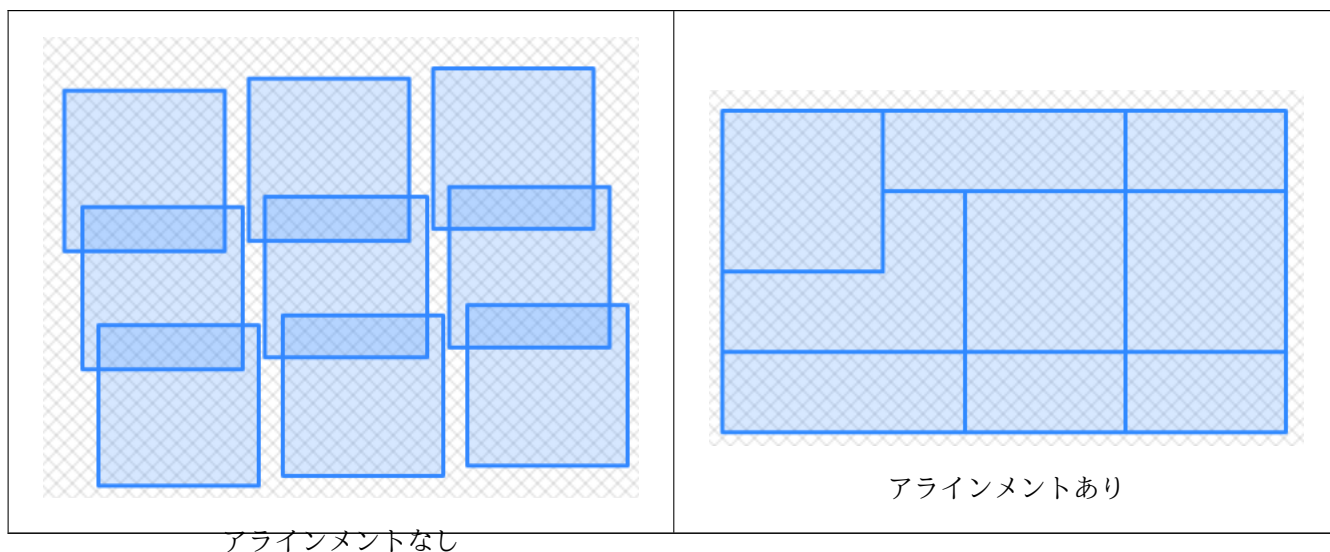
), bar AS (
  SELECT
    ST_Transform(rast, 4269) AS alignto
  FROM foo
  LIMIT 1
), baz AS (
  SELECT
    rid,
    rast,
    ST_Transform(rast, 4269) AS not_aligned,
    ST_Transform(rast, alignto) AS aligned
  FROM foo
  CROSS JOIN bar
)
SELECT
  ST_SameAlignment(rast) AS rast,
  ST_SameAlignment(not_aligned) AS not_aligned,
  ST_SameAlignment(aligned) AS aligned
FROM baz

```

```

rast | not_aligned | aligned
-----+-----+-----
t    | f             | t

```



関連情報

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

11.8 ラスタバンドエディタ

11.8.1 ST_SetBandNoDataValue

`ST_SetBandNoDataValue` — 指定したバンドに `NODATA` を表現する値を設定します。バンドを指定しない場合には、1番と仮定します。`NODATA` 値を持たないようにするには、`nodatavalue` に `NULL` を指定します。

Synopsis

raster **ST_SetBandNoDataValue**(raster rast, double precision nodatavalue);
raster **ST_SetBandNoDataValue**(raster rast, integer band, double precision nodatavalue, boolean forcechecking=false);

説明

指定したバンドに NODATA を表現する値を設定します。バンドを指定しない場合には、1 番と仮定します。この関数は **ST_Polygon**, **ST_DumpAsPolygons**, **ST_PixelAs** 系関数群からの結果に影響を与えます。

例

This example changes only the first band NODATA value.

```
UPDATE dummy_rast
  SET rast = ST_SetBandNoDataValue(rast,1, 254)
WHERE rid = 2;

-- change no data band value of bands 1,2,3
UPDATE dummy_rast
  SET rast =
    ST_SetBandNoDataValue(ST_SetBandNoDataValue(ST_SetBandNoDataValue(rast,1, 254),
    2,99),
    3,108)
  WHERE rid = 2;

-- wipe out the nodata value this will ensure all pixels are considered for all processing ←
  functions
UPDATE dummy_rast
  SET rast = ST_SetBandNoDataValue(rast,1, NULL)
WHERE rid = 2;
```

関連情報

ST_BandNoDataValue, **ST_NumBands**

11.8.2 ST_SetBandIsNoData

ST_SetBandIsNoData — バンドの isnodata フラグを TRUE にします。

Synopsis

raster **ST_SetBandIsNoData**(raster rast, integer band=1);

説明

バンドの isnodata フラグを TRUE にします。バンドを指定しない場合には、1 番と仮定します。isnodata フラグが汚れている場合にのみ呼ぶべきものです。**ST_BandIsNoData**の、最後の引数に TRUE を設定した場合の結果と指定しない場合の結果とで異なっている時です。

Availability: 2.0.0

例

This example creates a dummy table with one raster column.

```
create table dummy_rast (rid integer, rast raster);

-- Add raster with two bands, one pixel/band. In the first band, nodatavalue = pixel value ←
  = 3.
-- In the second band, nodatavalue = 13, pixel value = 4
insert into dummy_rast values(
1,
(
'01' -- little endian (uint8 ndr)
||
'0000' -- version (uint16 0)
||
'0200' -- nBands (uint16 0)
||
'17263529ED684A3F' -- scaleX (float64 0.000805965234044584)
||
'F9253529ED684ABF' -- scaleY (float64 -0.00080596523404458)
||
'1C9F33CE69E352C0' -- ipX (float64 -75.5533328537098)
||
'718F0E9A27A44840' -- ipY (float64 49.2824585505576)
||
'ED50EB853EC32B3F' -- skewX (float64 0.000211812383858707)
||
'7550EB853EC32B3F' -- skewY (float64 0.000211812383858704)
||
'E6100000' -- SRID (int32 4326)
||
'0100' -- width (uint16 1)
||
'0100' -- height (uint16 1)
||
'4' -- hasnodatavalue set to true, isnodata value set to false (when it should be true)
||
'2' -- first band type (4BUI)
||
'03' -- novalue==3
||
'03' -- pixel(0,0)==3 (same that nodata)
||
'0' -- hasnodatavalue set to false
||
'5' -- second band type (16BSI)
||
'0D00' -- novalue==13
||
'0400' -- pixel(0,0)==4
)::raster
);

select ST_BandIsNoData(rast, 1) from dummy_rast where rid = 1; -- Expected false
select ST_BandIsNoData(rast, 1, TRUE) from dummy_rast where rid = 1; -- Expected true

-- The isnodata flag is dirty. We are going to set it to true
update dummy_rast set rast = ST_SetBandIsNoData(rast, 1) where rid = 1;

select ST_BandIsNoData(rast, 1) from dummy_rast where rid = 1; -- Expected true
```

関連情報

[ST_BandNoDataValue](#), [ST_NumBands](#), [ST_SetBandNoDataValue](#), [ST_BandIsNoData](#)

11.8.3 ST_SetBandPath

ST_SetBandPath — データベース外バンドの外部パスとバンド番号を更新します。

Synopsis

raster **ST_SetBandPath**(raster rast, integer band, text outdbpath, integer outdbindex, boolean force=false)

説明

データベース外バンドの外部パスと外部バンド番号を更新します。



Note

force が TRUE の場合には、外部ラスタファイルと PostGIS ラスタとの互換性 (例: アラインメント、ピクセル対応) 確認のためのテストを行いません。このモードは、外部ラスタファイルが存在するファイルシステムの変更を目的としています。

Availability: 2.5.0

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	<-->

関連情報

[ST_BandMetaData](#), [ST_SetBandIndex](#)

11.8.4 ST_SetBandIndex

ST_SetBandIndex — データベース外バンドの外部バンド番号の更新

Synopsis

raster **ST_SetBandIndex**(raster rast, integer band, integer outdbindex, boolean force=false);

説明

データベース外バンドの外部バンド番号を更新します。データベース外バンドと関連する外部ラスタファイルには触れません。



Note
force が TRUE の場合には、外部ラスタファイルと PostGIS ラスタとの互換性 (例 : アラインメント、ピクセル対応) 確認のためのテストを行いません。このモードは、バンドが外部ラスタファイル内で移動する場所を対象としています。



Note
内部的には、このメソッドは、存在するパス情報を更新する替わりに、PostGIS ラスタの band で指定したバンドを新しいバンドに置き換えます。

Availability: 2.5.0

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

関連情報

[ST_BandMetaData](#), [ST_SetBandPath](#)

11.9 ラスタバンド統計情報と解析

11.9.1 ST_Count

ST_Count — ラスタまたはラスタカバレッジの指定したバンドのピクセル数を返します。バンドを指定しない場合には、1 番と仮定します。exclude_nodata_value を TRUE に設定している場合には、NODATA 値と等しくないピクセルのみを数えます。

Synopsis

```
bigint ST_Count(raster rast, integer nband=1, boolean exclude_nodata_value=true);
bigint ST_Count(raster rast, boolean exclude_nodata_value);
```

説明

ラスタまたはラスタカバレッジの指定したバンドのピクセル数を返します。**nband** でバンドを指定しない場合には、1 番と仮定します。



Note

exclude_nodata_value を TRUE に設定している場合には、**nodata** 値と等しくないピクセルのみ数えます。**exclude_nodata_value** を FALSE に設定している場合には、全てのピクセルを数えます。

Changed: 3.1.0 - **ST_Count**(rastertable, rastercolumn, ...) の形式は削除されました。代わりに**ST_CountAgg**を使用します。

Availability: 2.0.0

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

関連情報

ST_CountAgg, **ST_SummaryStats**, **ST_SetBandNoDataValue**

11.9.2 ST_CountAgg

ST_CountAgg — 集約関数です。ラスタ集合の与えられたバンドのピクセル数を返します。バンドが指定されていない場合には、1 番と仮定します。**exclude_nodata_value** を TRUE に設定している場合には、**NODATA** 値と等しくないピクセルのみを数えます。

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);
```


説明

ラスタ集合の与えられたバンドのピクセル数を返します。**nband** でバンドを指定しない場合には、1 番と仮定します。

exclude_nodata_value を TRUE に設定している場合には、**NODATA** 値と等しくないピクセルのみを数えます。**exclude_nodata_value** を FALSE に設定している場合には、全てのピクセルを数えます。

デフォルトでは、全てのピクセルを見ます。より早い応答を得るには、**sample_percent** 値を 0 から 1 の間で設定します。

Availability: 2.2.0

例

```
WITH foo AS (
  SELECT
    rast.rast
  FROM (
    SELECT ST_SetValue(ST_SetValue(ST_SetValue(ST_AddBand(ST_MakeEmptyRaster(10, 10, ↵
      10, 10, 2, 2, 0, 0,0),
      1, '64BF', 0, 0
    ),
      1, 1, 1, -10
    ),
      1, 5, 4, 0
    ),
      1, 5, 5, 3.14159
    ) AS rast
  ) AS rast
  FULL JOIN (
    SELECT generate_series(1, 10) AS id
  ) AS id
  ON 1 = 1
)
SELECT
  ST_CountAgg(rast, 1, TRUE)
FROM foo;
```

20

関連情報

[ST_Count](#), [ST_SummaryStats](#), [ST_SetBandNoDataValue](#)

11.9.3 ST_Histogram

ST_Histogram — ラスタまたはラスタカバレッジのビン範囲で分割したデータ分布をまとめるヒストグラムの集合を返します。ビン数を指定しない場合には自動計算されます。

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);
```

説明

最小値、最大値、合計数、全体から見た割合からなるレコードの集合を返します。nband でバンドを指定しない場合には、1 番と仮定します。



Note

デフォルトでは、nodata と同じピクセル値は考慮に入れません。exclude_nodata_value を FALSE に設定すると全てのピクセルを数えます。

width カテゴリ/ビン毎の幅を示す配列です。ビン数の指定が width 要素数を超える場合には、width を繰り返します。

例: ビン 9 個、width が [a, b, c] では、[a, b, c, a, b, c, a, b, c] が出力されます。

bins 取り出し数 -- 関数から戻そうとするレコード数です。指定しない場合には、自動計算されます。

right ヒストグラムを右から計算します (デフォルトは左からです)。値の評価の優先順位が、X 軸について [a, b) から (a, b] に変わります。

Changed: 3.1.0 ST_Histogram(table_name, column_name) の形式は削除されました。

Availability: 2.0.0

例: 単一ラスタスタイル - ビン数を自動計算とした、1 番、2 番、3 番バンドのヒストグラムの計算

```
SELECT band, (stats).*
FROM (SELECT rid, band, ST_Histogram(rast, band) As stats
      FROM dummy_rast CROSS JOIN generate_series(1,3) As band
      WHERE rid=2) As foo;
```

band	min	max	count	percent
1	249	250	2	0.08
1	250	251	2	0.08
1	251	252	1	0.04
1	252	253	2	0.08
1	253	254	18	0.72
2	78	113.2	11	0.44
2	113.2	148.4	4	0.16
2	148.4	183.6	4	0.16
2	183.6	218.8	1	0.04
2	218.8	254	5	0.2
3	62	100.4	11	0.44
3	100.4	138.8	5	0.2
3	138.8	177.2	4	0.16
3	177.2	215.6	1	0.04
3	215.6	254	4	0.16

例: ピン数を **6** で固定して **2** 番バンドだけを計算

```
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)

関連情報

[ST_Count](#), [ST_SummaryStats](#), [ST_SummaryStatsAgg](#)

11.9.4 ST_Quantile

ST_Quantile — ラスタまたはラスタテーブルカバレッジのサンプルまたは母集団の分位数を計算します。値がラスタの 25%,50%,75% にあるかを調べることができます。

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);
```

説明

ラスタまたはラスタテーブルカバレッジのサンプルまたは母集団の分位数を計算します。値がラスタの 25%,50%,75% にあるかを調べることができます。

**Note**

`exclude_nodata_value` を `FALSE` に設定している場合には、`NODATA` 値となるピクセルも数えます。

Changed: 3.1.0 `ST_Quantile(table_name, column_name)` の形式は削除されました。

Availability: 2.0.0

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;
```

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

関連情報

[ST_Count](#), [ST_SummaryStats](#), [ST_SummaryStatsAgg](#), [ST_SetBandNoDataValue](#)

11.9.5 ST_SummaryStats

`ST_SummaryStats` — 与えられたラスターのバンドまたはラスターカバレッジの `count`, `sum`, `mean`, `stddev`, `min`, `max` からなる統計情報のサマリーです。バンドを指定しない場合には、1 番と仮定します。

Synopsis

```
summarystats ST_SummaryStats(raster rast, boolean exclude_nodata_value);
summarystats ST_SummaryStats(raster rast, integer nband, boolean exclude_nodata_value);
```

説明

ラスタまたはラスタカバレッジの指定したバンドについて、ピクセル数、合計値、平均値、標準偏差、最小値、最大値からなる **summarystats** による統計情報の概要を返します。`nband` でバンドを指定しない場合には、1 番と仮定します。

**Note**
デフォルトでは、`nodata` と同じピクセル値は考慮に入れません。`exclude_nodata_value` を `FALSE` に設定すると全てのピクセルを数えます。

**Note**
デフォルトでは、全てのピクセルを見ます。より早い応答を得るには、`sample_percent` 値を 1 未満で設定します。

Changed: 3.1.0 `ST_SummaryStats(rastertable, rastercolumn, ...)` の形式は削除されました。代わりに **`ST_SummaryStatsAgg`** を使用します。

Availability: 2.0.0

例: 単一ラスタスタイル

```
SELECT rid, band, (stats).*
FROM (SELECT rid, band, ST_SummaryStats(rast, band) As stats
      FROM dummy_rast CROSS JOIN generate_series(1,3) As band
      WHERE rid=2) As foo;
```

rid	band	count	sum	mean	stddev	min	max
2	1	23	5821	253.086957	1.248061	250	254
2	2	25	3682	147.28	59.862188	78	254
2	3	25	3290	131.6	61.647384	62	254

例: 対象とする建物とインタセクトするピクセルの概要

この例は、ボストンの建物の全て (約 102,000 件) と空中写真タイル (150x150 ピクセルで約 134,000 タイル) とで、Windows 64 ビット上の PostGIS で計算したところ、574 ミリ秒かかりました。

```
WITH
-- our features of interest
feat AS (SELECT gid As building_id, geom_26986 As geom FROM buildings AS b
        WHERE gid IN(100, 103,150)
        ),
-- clip band 2 of raster tiles to boundaries of builds
-- then get stats for these clipped regions
b_stats AS
  (SELECT building_id, (stats).*
   FROM (SELECT building_id, ST_SummaryStats(ST_Clip(rast,2,geom)) As stats
        FROM aerials.boston
        INNER JOIN feat
        ON ST_Intersects(feat.geom,rast)
        ) As foo
  )
-- finally summarize stats
SELECT building_id, SUM(count) As num_pixels,
      MIN(min) As min_pval,
      MAX(max) As max_pval,
      SUM(mean*count)/SUM(count) As avg_pval
FROM b_stats
WHERE count
> 0
GROUP BY building_id
ORDER BY building_id;
```

building_id	num_pixels	min_pval	max_pval	avg_pval
100	1090	1	255	61.0697247706422
103	655	7	182	70.5038167938931
150	895	2	252	185.642458100559

例: ラスタカバレッジ

This example returns statistics for each band.

```
SELECT band, (stats).*
FROM (SELECT band, ST_SummaryStats('o_4_boston','rast', band) As stats
      FROM generate_series(1,3) As band) As foo;
```

band	count	sum	mean	stddev	min	max
1	8450000	725799	82.7064349112426	45.6800222638537	0	255
2	8450000	700487	81.4197705325444	44.2161184161765	0	255
3	8450000	575943	74.682739408284	44.2143885481407	0	255

For a table -- will get better speed if set sampling to less than 100%. Here we set to 25% and get a much faster answer.

```
SELECT band, (stats).*
FROM (SELECT band, ST_SummaryStats('o_4_boston','rast', band,true,0.25) As stats
      FROM generate_series(1,3) As band) As foo;
```

band	count	sum	mean	stddev	min	max
1	2112500	180686	82.6890480473373	45.6961043857248	0	255
2	2112500	174571	81.448503668639	44.2252623171821	0	255
3	2112500	144364	74.6765884023669	44.2014869384578	0	255

関連情報

[summarystats](#), [ST_SummaryStatsAgg](#), [ST_Count](#), [ST_Clip](#)

11.9.6 ST_SummaryStatsAgg

ST_SummaryStatsAgg — 集約関数。与えられたラスターのバンドの集合の count, sum, mean, stddev, min, max からなる統計情報のサマリーです。バンドを指定しない場合には、1 番と仮定します。

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);
```

説明

ラスターまたはラスターカバレッジの指定したバンドについて、ピクセル数、合計値、平均値、標準偏差、最小値、最大値からなる **summarystats** による統計情報の概要を返します。nband でバンドを指定しない場合には、1 番と仮定します。



Note

デフォルトでは、NODATA と同じピクセル値は考慮に入れません。exclude_nodata_value を FALSE に設定すると全てのピクセルを数えます。



Note

デフォルトでは全てのピクセルを見ます。より早い応答を得るには、sample_percent を 0 から 1 の間で設定します。

Availability: 2.2.0

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)
```

関連情報

[summarystats](#), [ST_SummaryStats](#), [ST_Count](#), [ST_Clip](#)

11.9.7 ST_ValueCount

ST_ValueCount — ラスタ (またはラスタカバレッジ) の指定されたバンドで、指定した値を持つピクセルを対象として、ピクセルバンド値とピクセル数からなるレコードの集合を返します。バンドを指定しない場合には、1番と仮定します。デフォルトでは **NODATA** 値のピクセルは数えられず、ピクセルの他の値は出力され、ピクセルバンド値は最も近い整数に丸められます。

Synopsis

SETOF record **ST_ValueCount**(raster rast, integer nband=1, boolean exclude_nodata_value=true, double precision[] searchvalues=NULL, double precision roundto=0, double precision OUT value, integer OUT count);

SETOF record **ST_ValueCount**(raster rast, integer nband, double precision[] searchvalues, double precision roundto=0, double precision OUT value, integer OUT count);

SETOF record **ST_ValueCount**(raster rast, double precision[] searchvalues, double precision roundto=0, double precision OUT value, integer OUT count);

bigint **ST_ValueCount**(raster rast, double precision searchvalue, double precision roundto=0);

bigint **ST_ValueCount**(raster rast, integer nband, boolean exclude_nodata_value, double precision searchvalue, double precision roundto=0);

bigint **ST_ValueCount**(raster rast, integer nband, double precision searchvalue, double precision roundto=0);

SETOF record **ST_ValueCount**(text rastertable, text rastercolumn, integer nband=1, boolean exclude_nodata_value=true, double precision[] searchvalues=NULL, double precision roundto=0, double precision OUT value, integer OUT count);

SETOF record **ST_ValueCount**(text rastertable, text rastercolumn, double precision[] searchvalues, double precision roundto=0, double precision OUT value, integer OUT count);

SETOF record **ST_ValueCount**(text rastertable, text rastercolumn, integer nband, double precision[] searchvalues, double precision roundto=0, double precision OUT value, integer OUT count);

bigint **ST_ValueCount**(text rastertable, text rastercolumn, integer nband, boolean exclude_nodata_value, double precision searchvalue, double precision roundto=0);

bigint **ST_ValueCount**(text rastertable, text rastercolumn, double precision searchvalue, double precision roundto=0);

bigint **ST_ValueCount**(text rastertable, text rastercolumn, integer nband, double precision searchvalue, double precision roundto=0);

説明

ラスタタイルまたはラスタカバレッジの指定したバンドにおけるピクセルバンド値とピクセル数にあたる、**value** と **count** からなるレコードの集合を返します。

nband でバンドを指定しない場合には、1 番と仮定します。**searchvalues** が指定されていない場合には、ラスタまたはラスタカバレッジで発見した全てのピクセル値が返ります。**searchvalue** を一つ指定した場合には、指定したピクセルバンド値を持つピクセルの数を示すレコードでなく、整数を返します。



Note

exclude_nodata_value を FALSE に設定している場合には、NODATA 値となるピクセルも数えます。

Availability: 2.0.0

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
```

関連情報

ST Count, ST SetBandNoDataValue

11.10 ラスタ入力

11.10.1 ST_RastFromWKB

ST_RastFromWKB — Well-Known Binary (WKB) ラスタからラスタ値を返します。

Synopsis

```
raster ST_RastFromWKB(bytea wkb);
```

說明

Well-Known Binary (WKB) ラスタが与えられると、ラスタを返します。

Availability: 2.5.0

例

```
SELECT (ST_MetaData(ST_RastFromWKB('001\000\000\000\000\000\000\000\000\000\000@ ↵  
    \000\000\000\000\000\000\010@ ↵  
    \000\000\000\000\000\000\340?\000\000\000\000\000\000\340?\000\000\000\000\000\000\000\000\0  
bytea  
    )  
)).* AS metadata;
```

upperleftx	upperlefty	width	height	scalex	scaley	skewx	skewy	srid	↔
numbands									
0.5	0.5	10	20	2	3	0	0	10	↔
0									

説明

ラスタのバイナリ表現を返します。`outasin` が `TRUE` の場合には、データベース外のバンドがデータベース内のバンドとして扱われます。表現の詳細については、PostGIS ソースフォルダにある `raster/doc/RFC2-WellKnownBinaryFormat` をご覧下さい。

データを文字列表現に変換せずにデータベース外に引き出すためのバイナリカーソルに使用できます。



Note

デフォルトでは、WKB 出力には、データベース外のバンドの外部ファイルパスを含みます。クライアントがデータベース外にあるラスタファイルにアクセスできない場合には、`outasin` を TRUE に設定します。

Enhanced: 2.1.0 outasin の追加

Enhanced: 2.5.0 ST AsWKB の追加

例

```
SELECT ST_AsBinary(rast) AS rastbin FROM dummy_rast WHERE rid=1;
```

```
\001\000\000\000\000\000\000\000\000\000\000@ \000\000\000\000\000\010@ ←  
      \000\000\000\000\000\000\340? \000\000\000\000\000\000\340?\000\000\000\000\000\000\000\000\000\000\000\000\000
```

関連情報

ST RastFromWKB, ST AsHexWKB

11.11.2 ST_AsHexWKB

ST AsHexWKB — Well-Known Binary (WKB) ラスタを 16 進数表現で返します。

Synopsis

```
bytea ST_AsHexWKB(raster rast, boolean outasin=FALSE);
```

說明

ラスタのバイナリの 16 進数表現を返します。`outasin` が TRUE の場合には、データベース外のバンドがデータベース内のバンドとして扱われます。表現の詳細については、PostGIS ソースフォルダにある `raster/doc/RFC2-WellKnownBinaryFormat` をご覧ください。



Note

デフォルトでは、WKB 出力には、データベース外のバンドの外部ファイルパスを含みます。クライアントがデータベース外バンドを持っているラスタファイルにアクセスできない場合には、outasin を TRUE に設定します。

Availability: 2.5.0


```
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 出力の例

```
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;
```

関連情報

Section [10.3](#), [ST_GDALDrivers](#), [ST_SRID](#)

11.11.4 ST_AsJPEG

ST_AsJPEG — ラスタの選択されたバンドを、単一の Joint Photographic Exports Group (JPEG) 画像としてバイト配列で返します。バンドを指定せず、1 バンドか 3 より多いバンドがある場合には、1 番バンドを使用します。3 バンドのみ指定した場合には、3 バンドを使用し、RGB に対応付けます。

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);
```

説明

ラスタの選択されたバンドを、単一の Joint Photographic Exports Group (JPEG) 画像として返します。より一般でないラスタタイプで出力する必要がある場合には、**ST_AsGDALRaster**を使います。バンドを指定せず、1 バンドか 3 より多いバンドがある場合には、1 番バンドを使用します。3 バンドのみ指定した場合には、3 バンドを使用します。この関数には多数の任意引数が付くさまざまな形式があります。引数については次の通りです。

- **nband** 単一バンド出力のためのものです。
- **nbands** 出力バンドの配列 (JPEG では 3 要素が最大です) で、バンドの並び順は RGB です。たとえば `ARRAY[3,2,1]` は、3 番バンドを赤、2 番バンドを緑、1 番バンドを青にそれぞれ対応させます。
- **quality** 0 から 100 の数値です。高いほどしっかりした画像になります。
- **options** JPEG のために定義する GDAL オプションの文字列配列です ([ST_GDALDrivers](#) の `create_options` を見てください)。JPEG の妥当なパラメーターは `PROGRESSIVE` の 'ON' または 'OFF' と、`QUALITY` の 0 から 100 までの数 (デフォルトは 75) です。詳細については [GDAL Raster format options](#) をご覧ください。

Availability: 2.0.0 - GDAL 1.6.0 以上が必要です。

例: 出力

Output the first three bands at 75 percent quality.

```
SELECT ST_AsJPEG(rast) As rastjpg
FROM dummy_rast WHERE rid=2;
```

Output only the first band at 90 percent quality.

```
SELECT ST_AsJPEG(rast,1,90) As rastjpg
FROM dummy_rast WHERE rid=2;
```

Output the first three bands, mapping band 2 to red, band 1 to green, and band 3 to blue, using progressive encoding at 90 percent quality.

```
SELECT ST_AsJPEG(rast,ARRAY[2,1,3],ARRAY['QUALITY=90','PROGRESSIVE=ON']) As rastjpg
FROM dummy_rast WHERE rid=2;
```

関連情報

Section [10.3](#), [ST_GDALDrivers](#), [ST_AsGDALRaster](#), [ST_AsPNG](#), [ST_AsTIFF](#)

11.11.5 ST_AsPNG

ST_AsPNG — ラスタの選択されたバンドを、単一の **portable network graphics** (PNG) 画像としてバイト配列で返します。バンドを指定せず、1 バンドか 3 バンドか 4 バンドある場合には、全てのバンドを使用します。バンドを指定せず、2 バンドか 4 より多いバンドがある場合には、1 番バンドを使用します。対象バンドは RGB または RGBA に対応付けられます。

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);
```


説明

ラスタの選択されたバンドを、単一の **portable network graphics (PNG)** 画像として返します。より一般でないラスタタイプで出力する必要がある場合には、**ST_AsGDALRaster**を使います。バンドを指定しない場合には、前から 3 バンドを出力します。この関数には多数の任意引数が付くさまざまな形式があります。**srid** が指定されない場合には、**srid** はラスタが使用している **SRID** を指定します。引数の一覧は次の通りです。

- **nband** 単一バンド出力のためのものです。
- **nbands** 出力バンドの配列 (PNG では 4 要素が最大です) で、バンドの並び順は **RGBA** です。たとえば **ARRAY[3,2,1]** は、3 番バンドを赤、2 番バンドを緑、1 番バンドを青にそれぞれ対応させます。
- **compression** 1 から 9 の数を指定します。大きいほど圧縮効率が上がります。
- **options** PNG のために定義する GDAL オプションの文字列配列です (**ST_GDALDrivers**の **create_options** を見てください)。PNG の妥当なパラメーターは **ZLEVEL** (圧縮に費やす時間の合計で、デフォルトは 6) です。**ARRAY['ZLEVEL=9']** というようにします。この関数が二つの出力を行う必要があるため、ワールドファイルは PNG では許されません。詳細については**GDAL Raster format options**をご覧ください。

Availability: 2.0.0 - GDAL 1.6.0 以上が必要です。

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;
```

関連情報

ST_AsGDALRaster, **ST_ColorMap**, **ST_GDALDrivers**, Section 10.3

11.11.6 ST_AsTIFF

ST_AsTIFF — ラスタの選択されたバンドを、単一の **TIFF** 画像 (バイト配列) として返します。バンドを指定しないか指定したバンドがラスタ内に無い場合には、全てのバンドの使用を試みます。

Synopsis

```
bytea ST_AsTIFF(raster rast, text[] options="", integer srid=sameasource);
bytea ST_AsTIFF(raster rast, text compression="", integer srid=sameasource);
bytea ST_AsTIFF(raster rast, integer[] nbands, text compression="", integer srid=sameasource);
bytea ST_AsTIFF(raster rast, integer[] nbands, text[] options, integer srid=sameasource);
```

説明

ラスタの選択されたバンドを、単一の **Tagged Image File Format (TIFF)** 画像として返します。バンドを指定しない場合には、全てのバンドの使用を試みます。この関数は**ST_AsGDALRaster**のラップです。より一般でないラスタタイプで出力する必要がある場合には、**ST_AsGDALRaster**を使います。この関数には多数の任意引数が付くさまざまな形式があります。空間参照系 (**SRS**) の文字列表現が指定されていない場合には、ラスタの空間参照系を使います。引数については次の通りです。

- **nbands** 出力バンドの配列 (PNG では 3 要素が最大です) で、バンドの並び順は RGB です。たとえば `ARRAY[3,2,1]` は、3 番バンドを赤、2 番バンドを緑、1 番バンドを青にそれぞれ対応させます。
- **compression** 圧縮式 -- JPEG90 (または他のパーセント値), LZW, JPEG, DEFLATE9 のいずれかです。
- **options** GTiff を定義する GDAL オプションの文字列配列です ([ST_GDALDrivers](#) の GTiff 用の `create_options` を見てください)。詳細については [GDAL Raster format options](#) をご覧下さい。
- **srid** ラスタの `spatial_ref_sys` の SRID です。地理参照情報を登録するために使われます。

Availability: 2.0.0 - GDAL 1.6.0 以上が必要です。

例: **JPEG90%** 品質圧縮の使用

```
SELECT ST_AsTIFF(rast, 'JPEG90') As rasttiff
FROM dummy_rast WHERE rid=2;
```

関連情報

[ST_GDALDrivers](#), [ST_AsGDALRaster](#), [ST_SRID](#)

11.12 ラスタ処理: 地図代数

11.12.1 ST_Clip

ST_Clip — 入力ジオメトリで切り取ったラスタを返します。バンドが指定されていない場合には、全てのバンドが処理されます。`crop` が指定されていなかったり `TRUE` となっている場合には、出力ラスタは切り取られます。`touched` が `TRUE` の場合には、接触するピクセルは取り込まれ、`TRUE` でない場合には、中心がジオメトリ内にあるピクセルだけが取り込まれます。

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);
```

説明

入力ジオメトリ `geom` で切り取ったラスタを返します。バンドが指定されていない場合には、全てのバンドが処理されます。

ST_Clip が返すラスタは、バンド毎に一つずつ必ず切り取った領域に適用する `NODATA` 値を持ちます。`NODATA` 値が渡されず、入力ラスタが `NODATA` 値を持たない場合には、結果ラスタの `NODATA` 値は `ST_MinPossibleValue(ST_BandPixelType(rast, band))` に設定されます。配列における `NODATA` 値の要

素数がバンド数より小さい場合には、配列の最後の要素が残りのバンドに適用されます。NODATA 値の要素数がバンド数より多い場合には、超過分は無視されます。全ての NODATA 値配列を受け付ける形式では、バンド毎に適用される単一値も受け付けます。

crop が指定されていない場合には、TRUE として扱われ、出力ラスタは **geom** と **rast** の範囲の共有領域で切り取ります。**crop** が FALSE に指定されている場合には、新しいラスタは **rast** と同じ範囲になります。**touched** が TRUE に指定されている場合には、ジオメトリとインタセクトする **rast** 内の全てのピクセルが選択されます。



Note

デフォルトの振る舞いは **touched=false** で、中心がジオメトリに含まれているピクセルを選択するだけです。

Enhanced: 3.5.0 - 引数 **touched** を追加。

Availability: 2.0.0

Enhanced: 2.1.0 C 言語で記述されました

この例では、MassGIS サイト上の [MassGIS Aerial Orthos](#) にあるマサチューセッツ空中写真データを使っています。

例: 全ての **touched** と **touched** でないとの選択の比較

```
SELECT ST_Count(rast) AS count_pixels_in_orig, ST_Count(rast_touched) AS all_touched_pixels ←
, ST_Count(rast_not_touched) AS default_clip
FROM ST_AsRaster(ST_Letters('R'), scalex =
> 1.0, scaley =
> -1.0) AS r(rast)
INNER JOIN ST_GeomFromText('LINESTRING(0 1, 5 6, 10 10)') AS g(geom)
ON ST_Intersects(r.rast,g.geom)
CROSS JOIN ST_Clip(r.rast, g.geom, touched =
> true) AS rast_touched
CROSS JOIN ST_Clip(r.rast, g.geom, touched =
> false) AS rast_not_touched;
```

count_pixels_in_orig	all_touched_pixels	default_clip
2605	16	10

(1 row)

例: **1** バンドの切り取り (**touched** でない)

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



例: クロップがなく他のバンドは切り取らず **1** バンドを切り取る

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;
```



切り取り前の完全なラスタタイル



切り取り後 - シュールですね

例: 全バンドの切り取り

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;
```



切り取り前の完全なラスタタイル



切り取り後

関連情報

[ST_AddBand](#), [ST_Count](#), [ST_MapAlgebra \(callback function version\)](#), [ST_Intersection](#)

11.12.2 ST_ColorMap

ST_ColorMap — 元のラスタと指定したバンドから 4 個までの 8BUI バンド (grayscale, RGB, RGBA) からなる新しいラスタを生成します。

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);
```

説明

rast の **nband** で示されるバンドに **colormap** を適用し、4 個までの 8BUI バンドからなる新しいラスタを返します。新しいラスタの 8BUI バンドの数は **colormap** で定義された色要素の数で決まります。

nband が指定されていない場合には、1 番と仮定します。

colormap は事前定義された色マップまたは値を定義する行の集合と色要素のキーワードです。

妥当な事前定義された **colormap** キーワードは次の通りです。

- **grayscale** または **greyscale** 1 個の 8BUI バンドからなるラスタで、グレーの陰影です。
- **pseudocolor** 4 個の 8BUI バンド (RGBA) からなるラスタで、青から緑、赤に移るものです。
- **fire** 4 個の 8BUI バンド (RGBA) からなるラスタで、黒から赤、淡黄色に移るものです。
- **bluered** 4 個の 8BUI バンド (RGBA) からなるラスタで、青からパールホワイト、赤に移るものです。

カスタムカラーマップを指定するためにエントリ (1 行 1 エントリ) の集合を **colormap** に渡すことができます。それぞれのエントリは一般的に、ピクセル値、ピクセル値と対応する赤、緑、青、アルファ要素 (0 から 255 の間の色要素) からなる 5 個の値を持ちます。ピクセル値の替わりに百分率値を使うことができ、0% がラスタバンドでの最小値、100% が最大値になります。値はコンマ (','), タブ、コロンの (':'), 空白で区切られます。NODATA 値に対しては、ピクセル値を *nv*, *null*, *nodata* のいずれかに設定できます。例を次に示します。

```
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 の構文は、GDAL ツールの **gdaldem** の起伏モードに似ています。

method の妥当なキーワードは次の通りです。

- **INTERPOLATE** 与えられたピクセル値の間での滑らかな色合成のための線形補間に使います。
- **EXACT** カラーマップで見つかったピクセル値だけを厳格に一致させます。カラーマップエントリに一致しないピクセルに対しては 0 0 0 0 (RGBA) が設定されます。
- **NEAREST** ピクセル値に最も近い値にあうカラーマップエントリを使います。



Note

カラーマップの偉大な参考情報は **ColorBrewer** にあります。



Warning

新しいラスタの結果バンドには NODATA 値が入りません。NODATA 値が必要な場合には、**ST_SetBandNoDataValue**を使って NODATA 値をセットします。

Availability: 2.1.0

例

これは試行のためのがらくたテーブルです。

Setup test raster table.

```
DROP TABLE IF EXISTS funky_shapes;
CREATE TABLE funky_shapes(rast raster);

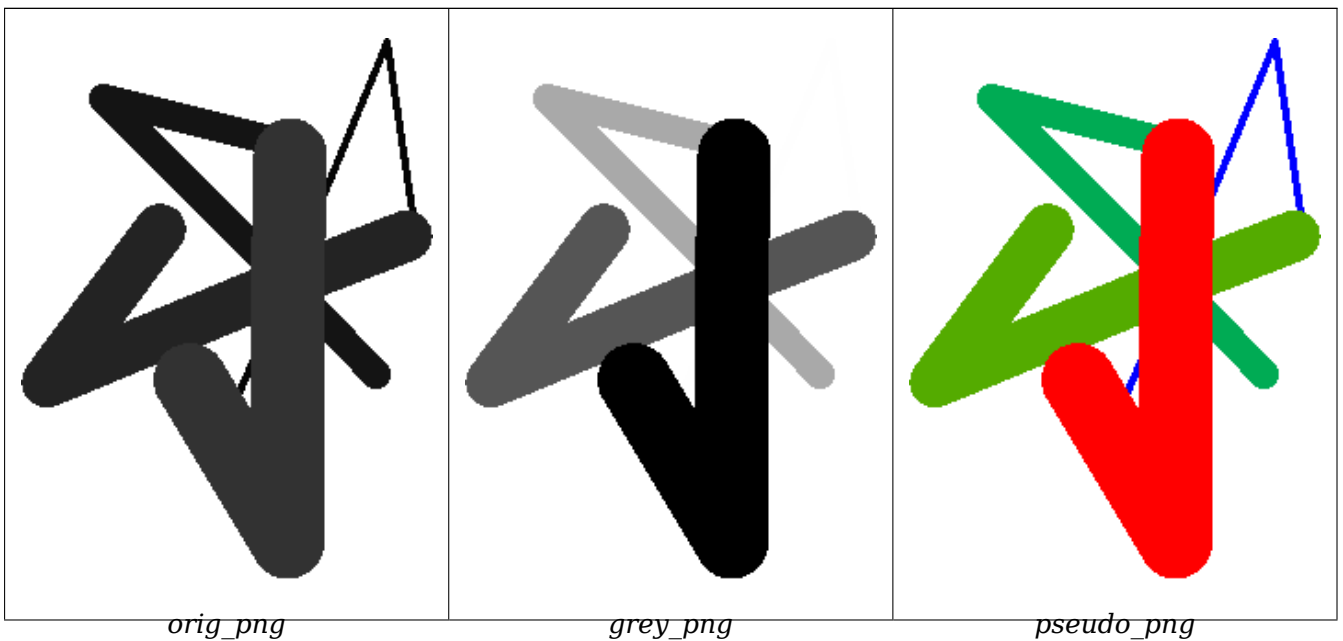
INSERT INTO funky_shapes(rast)
WITH ref AS (
    SELECT ST_MakeEmptyRaster(200, 200, 0, 200, 1, -1, 0, 0) AS rast
)
SELECT
    ST_Union(rast)
FROM (
    SELECT
        ST_AsRaster(ST_Rotate(ST_Buffer(ST_GeomFromText('LINESTRING(0 2,50 50,150 150,125 50)'),
            i*2
        ),
        pi() * i * 0.125, ST_Point(50,50)
    ),
    ref.rast, '8BUI'::text, i * 5
    ) AS rast
    FROM ref
    CROSS JOIN generate_series(1, 10, 3) AS i
) AS shapes;
```

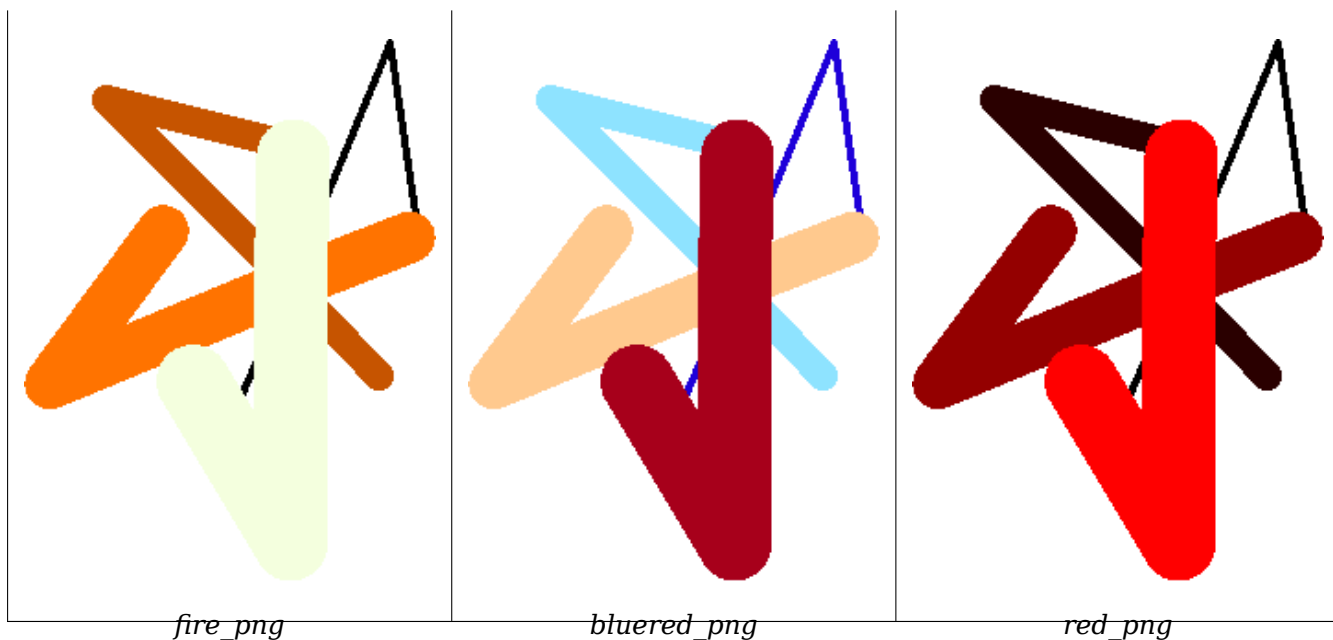
```
SELECT
    ST_NumBands(rast) As n_orig,
    ST_NumBands(ST_ColorMap(rast,1, 'greyscale')) As ngrey,
    ST_NumBands(ST_ColorMap(rast,1, 'pseudocolor')) As npseudo,
    ST_NumBands(ST_ColorMap(rast,1, 'fire')) As nfire,
    ST_NumBands(ST_ColorMap(rast,1, 'bluered')) As nbluered,
    ST_NumBands(ST_ColorMap(rast,1, '
100% 255  0  0
80%  160  0  0
50%  130  0  0
30%   30  0  0
20%   60  0  0
0%    0  0  0
nv 255 255 255
')) As nred
FROM funky_shapes;
```

n_orig	ngrey	npseudo	nfire	nbluered	nred
1	1	4	4	4	3

例: **ST_AsPNG** を使用して異なるカラーマップを比較する

```
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;
```





関連情報

[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 — 元のラスタと指定したバンドを赤、緑、青バンドとして一つの 8BUI バンドを持つラスタを生成します。

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);

説明

三つの入力バンド (一つ以上のラスタ) を与えられ、一つの 8BUI バンドを持つラスタを生成します。タイプが 8BUI でない入力バンドは全て [ST_Reclass](#) で再分類されます。



Note

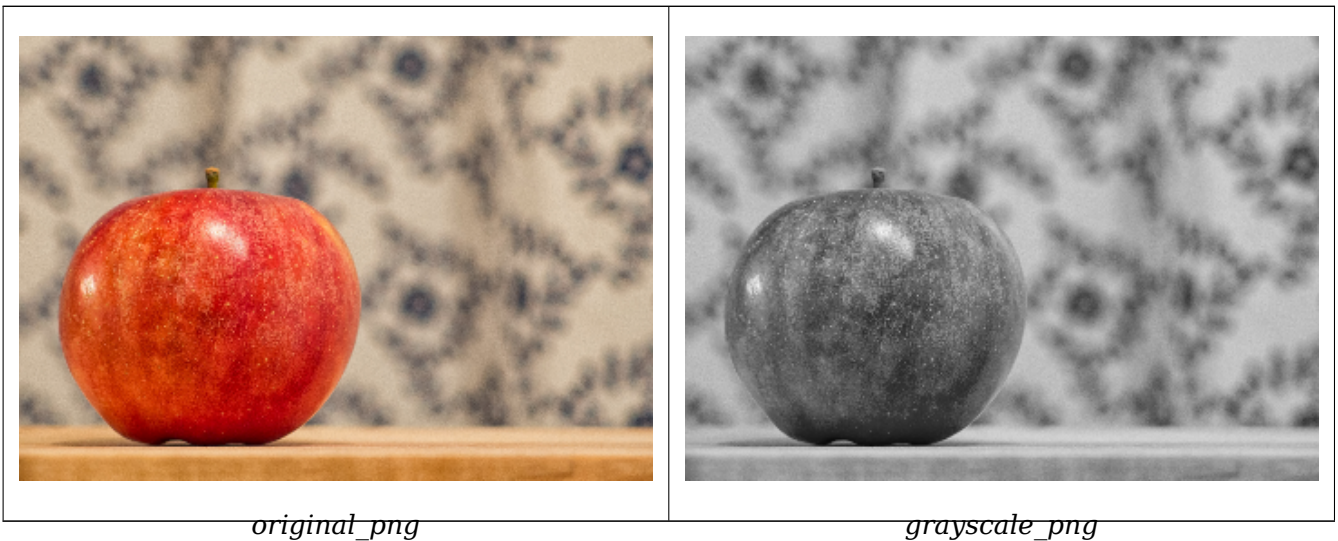
この関数は [ST_ColorMap](#) に `grayscale` キーワードを与えた場合とは違います。 [ST_ColorMap](#) では一つのバンドを扱いますが、この関数では RGB の 3 バンドを期待します。この関数は $0.2989 * \text{赤} + 0.5870 * \text{緑} + 0.1140 * \text{青}$ という式を適用して RGB をグレースケールにしています。

Availability: 2.5.0

例: 一つ目の形式

```
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;
```



例: 二つ目の形式

```
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;
```

関連情報

[ST_AsPNG](#), [ST_Reclass](#), [ST_ColorMap](#)

11.12.4 ST_Intersection

ST_Intersection — 二つのラスタの共有部分またはベクタ化したラスタとジオメトリとのインタセクトした部分を表現する、ラスタまたはジオメトリとピクセル値の組の集合を返します。

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);
```

説明

二つのラスタの共有部分またはベクタ化したラスタとジオメトリとのインタセクトした部分を表現する、ラスタまたはジオメトリとピクセル値の組の集合を返します。

前半の三つの形式は、geomval の集合を返すもので、ベクタ空間で動作します。ラスタは初めに geomval 行の集合にベクトル化されます (**ST_DumpAsPolygons** を使用)。これらの行は PostGIS 関数の **ST_Intersection**(geometry, geometry) を使ってジオメトリとインタセクトさせます。NODATA 値の領域とだけインタセクトするジオメトリについては空ジオメトリを返します。通常は WHERE 節で **ST_Intersects** を使って結果から確実に排除します。

丸括弧でくくって式の末尾に '.geom' や '.val' をつけることにより、geomval の結果集合のジオメトリや値の部分にアクセスすることができます。たとえば (ST_Intersection(rast, geom)).geom 等とします。

他の形式は、ラスタを引数に取り、ラスタを返します。ST_MapAlgebraExpr の二つのラスタを取る形式を使って、インタセクトしている部分を取得します。

結果ラスタの範囲は、二つのラスタの範囲についてインタセクトしている部分です。結果ラスタは、returnband 引数として渡されたものにあわせられた 'BAND1', 'BAND2', 'BOTH' バンドを含みます。どのバンドでも NODATA 値の領域は、結果ラスタの全てのバンドの NODATA 値領域に現れます。言い換えると、あらゆる NODATA 値ピクセルとインタセクトしているピクセルは、結果ラスタでは NODATA 値ピクセルになります。

インタセクトしなかった領域に NODATA 値を入れるために、ST_Intersection からの結果ラスタは、NODATA 値を持たなければなりません。結果ラスタのどのバンドにも、一つか二つの NODATA 値を持つ nodataval[] 配列を与えることで NODATA 値を定義したり置き換えたりできます。この配列は、引数で与えた 'BAND1', 'BAND2', 'BOTH' バンドに依存します。配列の一つ目の値は、一つ目のバンドの NODATA 値を入れ替えるものです。二つ目の値は二つ目のバンドの NODATA 値を入れ替えるものです。入力バンドの一つが NODATA 値を持っておらず、かつ配列を渡さなかった場合には、ST_MinPossibleValue 関数を使って NODATA 値が選ばれます。NODATA 値の配列を受け付ける形式の全てが、単一値を受け付けます。単一値は結果ラスタのそれぞれのバンドに適用されます。

全ての形式で、バンド番号を指定していない場合には、1 番と仮定します。ラスタとジオメトリを引数にとり、ラスタを得たい場合には、**ST_Clip** を参照して下さい。



Note

NODATA 値に遭遇した時の、結果範囲や返された物に関して、より多くの制御を行いたい場合には、**ST_MapAlgebraExpr** の二つのラスタを取る形式を使います。



Note

ラスタバンドとジオメトリとがインタセクトする部分を計算するには、**ST_Clip** を使います。ST_Clip は複数のバンドで動作し、ラスタ化されたジオメトリに従ったバンドを返すことはしません。

**Note**

ST_Intersection は、**ST_Intersects**と組み合わせて、ラスタカラムとジオメトリカラムのインデックスを使うべきです。

Enhanced: 2.0.0 - ラスタ空間のインタセクションが導入されました。2.0.0 より前の版では、ベクタ空間でのインタセクションの計算のみに対応していました。

例: ジオメトリとラスタ -- ジオメトリと値を得る

```
SELECT
  foo.rid,
  foo.gid,
  ST_AsText((foo.geomval).geom) As geomwkt,
  (foo.geomval).val
FROM (
  SELECT
    A.rid,
    g.gid,
    ST_Intersection(A.rast, g.geom) As geomval
  FROM dummy_rast AS A
  CROSS JOIN (
    VALUES
      (1, ST_Point(3427928, 5793243.85) ),
      (2, ST_GeomFromText('LINESTRING(3427927.85 5793243.75,3427927.8 5793243.75,3427927.8 5793243.8)'),),
      (3, ST_GeomFromText('LINESTRING(1 2, 3 4)'))
    ) As g(gid,geom)
  WHERE A.rid = 2
) As foo;
```

rid	gid	geomwkt	val
2	1	POINT(3427928 5793243.85)	249
2	1	POINT(3427928 5793243.85)	253
2	2	POINT(3427927.85 5793243.75)	254
2	2	POINT(3427927.8 5793243.8)	251
2	2	POINT(3427927.8 5793243.8)	253
2	2	LINESTRING(3427927.8 5793243.75,3427927.8 5793243.8)	252
2	2	MULTILINESTRING((3427927.8 5793243.8,3427927.8 5793243.75),...)	250
2	3	GEOMETRYCOLLECTION EMPTY	

関連情報

[geomval](#), [ST_Intersects](#), [ST_MapAlgebraExpr](#), [ST_Clip](#), [ST_AsText](#)

11.12.5 ST_MapAlgebra (callback function version)

ST_MapAlgebra (callback function version) — コールバック関数版 - 一つ以上の入力ラスタ、バンドインデックスと一つのユーザ定義コールバック関数から、一つのバンドからなるラスタを返します。

Synopsis

```
raster ST_MapAlgebra(rastbandarg[] rastbandargset, regprocedure callbackfunc, text pixeltype=NULL,
text extenttype=INTERSECTION, raster customextent=NULL, integer distancex=0, integer distancey=0,
text[] VARIADIC userargs=NULL);
raster ST_MapAlgebra(raster rast, integer[] nband, regprocedure callbackfunc, text pixeltype=NULL,
text extenttype=FIRST, raster customextent=NULL, integer distancex=0, integer distancey=0, text[]
VARIADIC userargs=NULL);
raster ST_MapAlgebra(raster rast, integer nband, regprocedure callbackfunc, text pixeltype=NULL,
text extenttype=FIRST, raster customextent=NULL, integer distancex=0, integer distancey=0, text[]
VARIADIC userargs=NULL);
raster ST_MapAlgebra(raster rast1, integer nband1, raster rast2, integer nband2, regprocedure call-
backfunc, text pixeltype=NULL, text extenttype=INTERSECTION, raster customextent=NULL, inte-
ger distancex=0, integer distancey=0, text[] VARIADIC userargs=NULL);
raster ST_MapAlgebra(raster rast, integer nband, regprocedure callbackfunc, float8[] mask, boolean
weighted, text pixeltype=NULL, text extenttype=INTERSECTION, raster customextent=NULL, text[]
VARIADIC userargs=NULL);
```

説明

一つ以上の入力ラスタ、バンドインデックスと一つのユーザ定義コールバック関数から、一つのバンドからなるラスタを返します。

rast, rast1, rast2, rastbandargset 地図代数処理が行われるラスタです。

rastbandargset によって、多数のラスタと多数のバンドにおいて地図代数処理が使用できます。一つ目の形式の例を見て下さい。

nband, nband1, nband2 処理を行うラスタのバンド番号です。nband はバンドを示す整数スカラまたは整数配列です。2 ラスタ/2 バンドの場合では、nband1 は rast1 のバンド、nband2 は rast2 のバンドです。

callbackfunc callbackfunc 引数は、SQL または PL/pgSQL 関数の名前とシグニチャでなければならず、regprocedure にキャストしなければなりません。PL/pgSQL 関数の例は次の通りです:

```
CREATE OR REPLACE FUNCTION sample_callbackfunc(value double precision[][][], position ←
integer[][], VARIADIC userargs text[])
RETURNS double precision
AS $$
BEGIN
    RETURN 0;
END;
$$ LANGUAGE 'plpgsql' IMMUTABLE;
```

The callbackfunc must have two or three arguments: a 3-dimension double precision array, a 2-dimension integer array and, optionally, a variadic 1-dimension text array. The first argument value is the set of values (as double precision) from all input rasters. The three dimensions (where indexes are 1-based) are: raster #, row y, column x. The second argument position is the set of pixel positions from the output raster and input rasters. The outer dimension (where indexes are 0-based) is the raster #. The position at outer dimension index 0 is the output raster's pixel position. For each outer dimension, there are two elements in the inner dimension for X and Y. The optional third argument userargs is for passing through any user-specified arguments.

regprocedure 引数を SQL 関数に渡す場合には、渡し先の完全な関数シグネチャが必要で、さらに regprocedure にキャストします。上の例の PL/pgSQL 関数を引数として渡すには、引数の SQL は次のようになります:

```
'sample_callbackfunc(double precision[], integer[], text[])':regprocedure
```

引数には関数名が含まれ、関数引数の型、関数名と引数型を引用符で括ったもの、`regprocedure` へのキャストが存在することに注意して下さい。

mask N 次元の数値配列 (行列) で、地図代数コールバック関数に渡すセルを決めるためのフィルタに使われます。0 は近隣セル値を NODATA として扱うべきであることを示し、1 はデータとして扱うべきであることをそれぞれ意味します。**weight** が TRUE に指定されている場合には、この配列の値は、近隣セルのピクセル値に対して掛け算を行うための数になります。

weighted マスク値に重みづけを施す (元の値に対して乗算を行う) か、施さない (マスク処理だけを行う) かを示す真偽値です。

pixeltype `pixeltype` を渡した場合には、新しいラスタの一つのバンドが、そのピクセルタイプになります。`pixeltype` に NULL を渡したり指定しなかった場合には、新しいラスタのピクセルタイプは、一つ目のラスタ (`extenttype` が INTERSECTION, UNION, FIRST, CUSTOM の場合) か、適切なラスタ (`extenttype` が SECOND, LAST の場合) の指定したバンドと同じピクセルタイプになります。疑問を感じたら常に `pixeltype` を渡します。

出力ラスタのピクセルタイプは、必ず `ST_BandPixelType` に挙げられたものの一つになるか、省略されるか、NULL に設定されます。

extenttype INTERSECTION (デフォルト), UNION, FIRST (一つのラスタを取る形式でのデフォルト), SECOND, LAST, CUSTOM のいずれかになります。

customextent `extenttype` が CUSTOM である場合には、ラスタは `customextent` で提供されます。一つ目の形式の例 4 をご覧下さい。

distancex 参照セルからのピクセル単位の距離です。結果として得られる行列の幅は $2 * \text{distancex} + 1$ となります。指定しない場合には、参照セルだけが対象となります (0 の距離の近隣ピクセル)。

distancey 参照セルからのピクセル単位の距離です。結果として得られる行列の高さは $2 * \text{distancey} + 1$ となります。指定しない場合には、参照セルが対象となります (0 の距離の近隣ピクセル)。

userargs If the `callbackfunc` declares a third argument, that argument is a variadic text array. All trailing text arguments are passed through to the specified `callbackfunc`, and are contained in the `userargs` argument.



Note

VARIADIC キーワードに関する詳細情報については、PostgreSQL 文書と [Query Language \(SQL\) Functions](#) (訳注: 日本語版は「[問い合わせ言語 \(SQL\) 関数](#)」です) の "SQL Functions with Variable Numbers of Arguments" (訳注: 日本語版は「[可変長引数を取る SQL 関数](#)」) 節を参照して下さい。

一つ目の形式では、多数のラスタやバンドで地図代数演算が使えるようになるための `rastbandarg` 配列を受け付けます。一つ目の形式の例をご覧下さい。

二つ目と三つ目の形式では、一つのラスタにおける一つ以上のバンドについて演算を行います。二つ目の形式と三つ目の形式の例をご覧下さい。

四つ目の形式では、二つのラスタにおいて、それぞれ一つずつのバンドについて演算を行います。四つ目の形式の例をご覧下さい。

Availability: 2.2.0: マスクが追加されました。

Availability: 2.1.0

例: 一つ目の形式

一つのラスタ、一つのバンド

```
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
```

一つのラスタ、複数のバンド

```
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
```

複数のラスタ、複数のバンド

```
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
```

近隣ピクセルを併用したカバレッジのタイルの完全な例です。

```
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

```

前の例である近隣ピクセルを併用したカバレッジのタイルに似ていますが基本的な SQL 機能だけを使用しています。

```

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;

```

例: 二つ目の形式と三つ目の形式

一つのラスタ、複数のバンド

```

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

```

一つのラスタ、一つのバンド

```

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

```

例: 四つ目の形式

二つのラスタ、二つのバンド

```

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(
        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

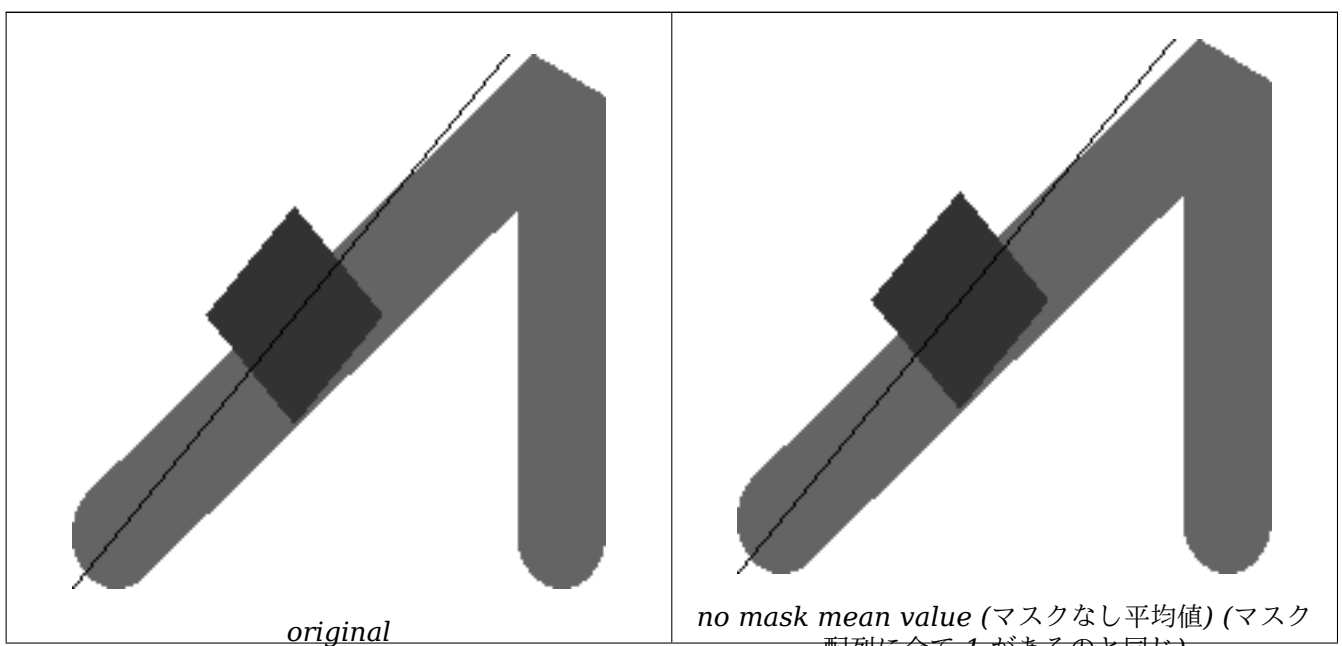
```

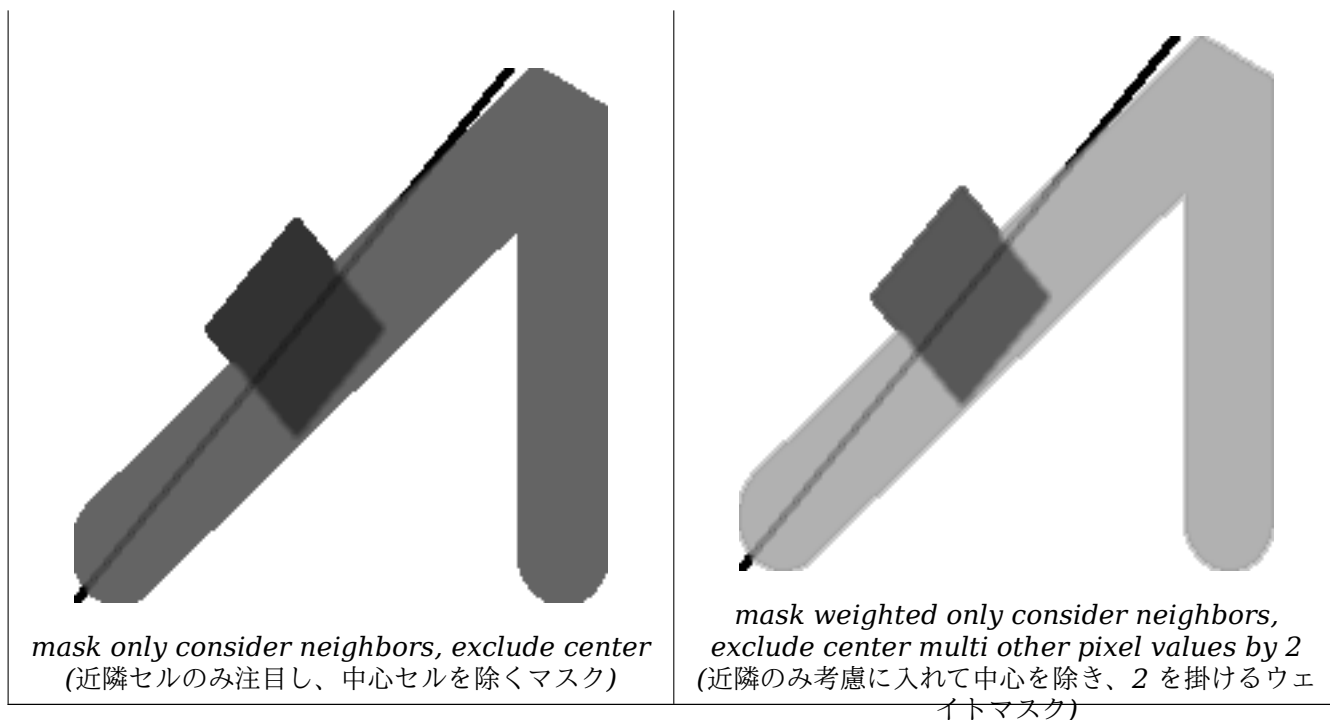
例: マスクの使用

```

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;

```





関連情報

[rastbandarg](#), [ST_Union](#), [ST_MapAlgebra \(expression version\)](#)

11.12.6 ST_MapAlgebra (expression version)

ST_MapAlgebra (expression version) — 数式版 - 一つか二つの入力ラスタ、バンド番号、一つ以上のユーザ定義 SQL 式から一つのバンドを持つラスタを返します。

Synopsis

```
raster ST_MapAlgebra(raster rast, integer nband, text pixeltype, text expression, double precision
nodataval=NULL);
raster ST_MapAlgebra(raster rast, text pixeltype, text expression, double precision nodataval=NULL);
raster ST_MapAlgebra(raster rast1, integer nband1, raster rast2, integer nband2, text expression,
text pixeltype=NULL, text extenttype=INTERSECTION, text nodata1expr=NULL, text nodata2expr=NULL,
double precision nodatanodataval=NULL);
raster ST_MapAlgebra(raster rast1, raster rast2, text expression, text pixeltype=NULL, text extent-
type=INTERSECTION, text nodata1expr=NULL, text nodata2expr=NULL, double precision nodatan-
odataval=NULL);
```

説明

数式版 - 一つか二つの入力ラスタ、バンド番号、一つ以上のユーザ定義 SQL 式から一つのバンドを持つラスタを返します。

Availability: 2.1.0

説明: 一つ目の形式と二つ目の形式 (一つのラスタ)

expression で定義された妥当な PostgreSQL 代数演算を入力ラスタ (**rast**) に適用して、一つのバンドを持つラスタを生成します。**nband** が指定されない場合には、1 番バンドと仮定します。新しいラスタは、元のラスタと同じ地理参照、幅、高さになりますが、バンドは一つだけとなります。

pixeltype が渡された場合には、新しいラスタのバンドは、そのピクセルタイプになります。**pixeltype** に NULL が渡された場合には、新しいラスタは入力 **rast** のバンドのピクセルタイプと同じになります。

- **expression** のキーワードは次の通りです。

1. [**rast**] - 演算対象ピクセルの値
2. [**rast.val**] - 演算対象ピクセルの値
3. [**rast.x**] - 演算対象ピクセルの列 (1 始まり)
4. [**rast.y**] - 演算対象ピクセルの行 (1 始まり)

説明: 三つ目と四つ目の形式 (二つのラスタ)

expression で定義された妥当な二つのバンドへの PostgreSQL 代数演算を入力ラスタ **rast1**, (**rast2**) に適用して、一つのバンドを持つラスタを生成します。**band1**, **band2** が指定されない場合には、1 番バンドと仮定します。新しいラスタは、一つ目のラスタと同じアラインメント (スケール、スキュー、ピクセル隅) を持ちます。新しいラスタは、**extenttype** 引数で定義される範囲になります。

expression 二つのラスタと PostgreSQL 定義済み関数/演算子を含む PostgreSQL 代数式です。関数と演算子は、二つのピクセルがインタセクトするピクセルの値を定めます。たとえば $(([\text{rast1}] + [\text{rast2}]) / 2.0)::\text{integer}$ といったふうになります。

pixeltype 出力ラスタのピクセルタイプです。必ず **ST_BandPixelType** に挙げられたものの一つになるか、省略されるか、NULL に設定されます。引数として渡されないか NULL が渡された場合には、一つ目のラスタのピクセルタイプになります。

extenttype 新しいラスタの範囲を制御します。

1. **INTERSECTION** - 新しいラスタの範囲は二つのラスタのインタセクトした領域です。これがデフォルトです。
2. **UNION** - 新しいラスタの範囲は二つのラスタの結合です。
3. **FIRST** - 新しいラスタの範囲は一つ目のラスタと同じです。
4. **SECOND** - 新しいラスタの範囲は二つ目のラスタと同じです。

nodataexpr **rast1** が NODATA 値で、特に **rast2** ピクセルに値がある時に、**rast2** だけを返すか返すべき値を定義する定数を含む代数式です。

nodata2expr **rast2** が NODATA 値で、特に **rast2** ピクセルに値がある時に、**rast1** だけを返すか返すべき値を定義する定数を含む代数式です。

nodatanodataval **rast1** と **rast2** のピクセルの両方が NOADTA 値になる場合に返すべき定数です。

- 有効な **expression**, **nodataexpr**, **nodata2expr** のキーワードは次の通りです。

1. [**rast1**] - **rast1** の演算対象ピクセルの値
2. [**rast1.val**] - **rast1** の演算対象ピクセルの値
3. [**rast1.x**] - **rast1** の演算対象ピクセルの列 (1 始まり)
4. [**rast1.y**] - **rast1** の演算対象ピクセルの行 (1 始まり)
5. [**rast2**] - **rast2** の演算対象ピクセルの値
6. [**rast2.val**] - **rast2** の演算対象ピクセルの値
7. [**rast2.x**] - **rast2** の演算対象ピクセルの列 (1 始まり)
8. [**rast2.y**] - **rast2** の演算対象ピクセルの行 (1 始まり)

例: 一つ目の形式と二つ目の形式

```
WITH foo AS (
  SELECT ST_AddBand(ST_MakeEmptyRaster(10, 10, 0, 0, 1, 1, 0, 0, 0), '32BF'::text, 1, -1) ←
    AS rast
)
SELECT
  ST_MapAlgebra(rast, 1, NULL, 'ceil([rast]*[rast.x]/[rast.y]+[rast.val])')
FROM foo;
```

例: 三つ目の形式と四つ目の形式

```
WITH foo AS (
  SELECT 1 AS rid, ST_AddBand(ST_AddBand(ST_AddBand(ST_MakeEmptyRaster(2, 2, 0, 0, 1, -1, ←
    0, 0, 0), 1, '16BUI', 1, 0), 2, '8BUI', 10, 0), 3, '32BUI'::text, 100, 0) AS rast ←
    UNION ALL
  SELECT 2 AS rid, ST_AddBand(ST_AddBand(ST_AddBand(ST_MakeEmptyRaster(2, 2, 0, 0, 1, 1, -1, ←
    0, 0, 0), 1, '16BUI', 2, 0), 2, '8BUI', 20, 0), 3, '32BUI'::text, 300, 0) AS rast
)
SELECT
  ST_MapAlgebra(
    t1.rast,
    2,
    t2.rast,
    1,
    '([rast2] + [rast1.val]) / 2'
  ) AS rast
FROM foo t1
CROSS JOIN foo t2
WHERE t1.rid = 1
AND t2.rid = 2;
```

関連情報

[rastbandarg](#), [ST_Union](#), [ST_MapAlgebra \(callback function version\)](#)

11.12.7 ST_MapAlgebraExpr

ST_MapAlgebraExpr — 1 バンド版: 入力バンドに対する妥当な PostgreSQL 代数演算で形成された、指定したピクセルタイプとなる 1 バンドラスタを生成します。バンドを指定しない場合には、1 番を仮定します。

Synopsis

raster **ST_MapAlgebraExpr**(raster rast, integer band, text pixeltype, text expression, double precision nodataval=NULL);

raster **ST_MapAlgebraExpr**(raster rast, text pixeltype, text expression, double precision nodataval=NULL);

説明



Warning

ST_MapAlgebraExpr は 2.1.0 で非推奨になりました。代わりに [ST_MapAlgebra \(expression version\)](#) を使います。

入力ラスタ (**rast**) に対して **expression** で定義される妥当な PostgreSQL 代数演算で形成されるラスタを返します。生成されるラスタは指定したピクセルタイプとなる 1 バンドラスタです。**band** を指定しない場合には、1 番と仮定します。新しいラスタは、元のラスタと同じ地理参照、幅、高さを持ちますが、一つのバンドしか持ちません。

pixeltype が渡された場合には、新しいラスタのバンドは、そのピクセルタイプになります。**pixeltype** に NULL が渡された場合には、新しいラスタは入力 **rast** のバンドのピクセルタイプと同じになります。

数式の中では、**[rast]** で元のバンドのピクセル値を、**[rast.x]** で 1 始まりの列番号、**[rast.y]** で 1 始まりの行番号を、それぞれ参照することができます。

Availability: 2.0.0

例

元のラスタから 1 バンドラスタを生成します。元のラスタバンドの値について 2 で割った余りが入ります。

```
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

ピクセルタイプが 2BUI の 1 バンドラスタを生成します。元のラスタに対して再分類を行った値が入り、NODATA 値を 0 に設定します。

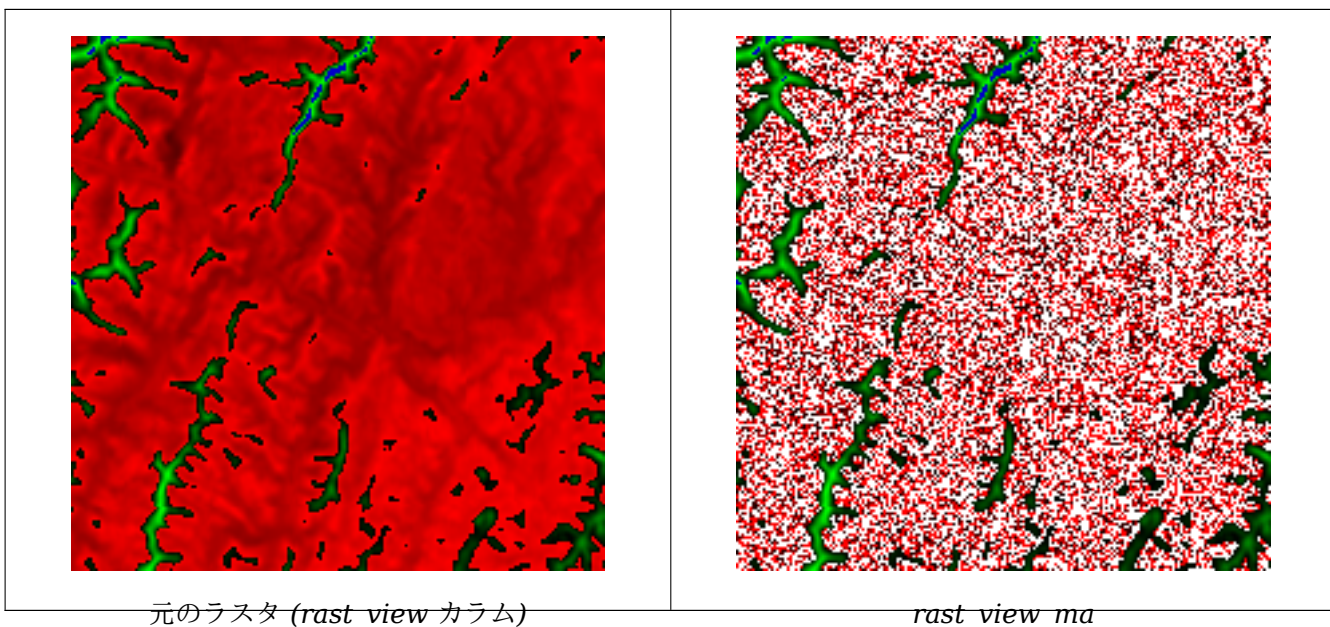
```
ALTER TABLE dummy_rast ADD COLUMN map_rast2 raster;
UPDATE dummy_rast SET
    map_rast2 = ST_MapAlgebraExpr(rast,'2BUI'::text,'CASE WHEN [rast] BETWEEN 100 and 250 ←
        THEN 1 WHEN [rast] = 252 THEN 2 WHEN [rast] BETWEEN 253 and 254 THEN 3 ELSE 0 END':: ←
        text, '0')
WHERE rid = 2;

SELECT DISTINCT
    ST_Value(rast,1,i,j) As origval,
    ST_Value(map_rast2, 1, i, j) As mapval
FROM dummy_rast
CROSS JOIN generate_series(1, 5) AS i
CROSS JOIN generate_series(1,5) AS j
WHERE rid = 2;
```

origval	mapval
249	1
250	1
251	1
252	2
253	3
254	3

```
SELECT
  ST_BandPixelType(map_rast2) As b1pixtyp
FROM dummy_rast
WHERE rid = 2;
```

2BUI



新しいバンドを三つ持つラスタを生成します。元のバンドを三つ持つラスタと同じピクセルタイプです。1 番バンドは地図代数関数によって変更され、残りの二つのバンドは値が代わりません。

```
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;
```

関連情報

[ST_MapAlgebraExpr](#), [ST_MapAlgebraFct](#), [ST_BandPixelType](#), [ST_GeoReference](#), [ST_Value](#)

11.12.8 ST_MapAlgebraExpr

ST_MapAlgebraExpr — 2 バンド版: 二つの入力バンドに対する妥当な PostgreSQL 代数演算で形成された、指定したピクセルタイプとなる 1 バンドラスタを生成します。バンドを指定しない場合には、どちらも 1 番と仮定します。結果ラスタは、一つ目のラスタのアラインメント (スケール、スキュー、ピクセル角位置) にあわされます。範囲は "extenttype" 引数で定義されます。取りうる "extenttype" の値は INTERSECTION, UNION, FIRST, SECOND です。

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);
```

説明



Warning

ST_MapAlgebraExpr は 2.1.0 で非推奨になりました。代わりに **ST_MapAlgebra (expression version)** を使います。

expression で定義された妥当な二つのバンドへの PostgreSQL 代数演算を入力ラスタ **rast1**, (**rast2**) に適用して、一つのバンドを持つラスタを生成します。 **band1**, **band2** が指定されない場合には、1 番バンドと仮定します。新しいラスタは、一つ目のラスタと同じアラインメント (スケール、スキュー、ピクセル隅) を持ちます。新しいラスタは、 **extenttype** 引数で定義される範囲になります。

expression 二つのラスタと PostgreSQL 定義済み関数/演算子を含む PostgreSQL 代数式です。関数と演算子は、二つのピクセルがインタセクトするピクセルの値を定めます。たとえば $(([rast1] + [rast2])/2.0)::integer$ といったふうになります。

pixeltype 出力ラスタのピクセルタイプです。必ず **ST_BandPixelType** に挙げられたものの一つになるか、省略されるか、NULL に設定されます。引数として渡されないか NULL が渡された場合には、一つ目のラスタのピクセルタイプになります。

extenttype 新しいラスタの範囲を制御します。

1. **INTERSECTION** - 新しいラスタの範囲は二つのラスタのインタセクトした領域です。これがデフォルトです。
2. **UNION** - 新しいラスタの範囲は二つのラスタの結合です。
3. **FIRST** - 新しいラスタの範囲は一つ目のラスタと同じです。
4. **SECOND** - 新しいラスタの範囲は二つ目のラスタと同じです。

nodata1expr **rast1** が NODATA 値で、特に **rast2** ピクセルに値がある時に、**rast2** だけを返すか返すべき値を定義する定数を含む代数式です。

nodata2expr **rast2** が NODATA 値で、特に **rast2** ピクセルに値がある時に、**rast1** だけを返すか返すべき値を定義する定数を含む代数式です。

nodatanodataval **rast1** と **rast2** のピクセルの両方が NOADTA 値になる場合に返すべき定数です。

`pixeltype` が渡された場合には、新しいラスタは、指定されたピクセルタイプのバンドを持ちます。`pixeltype` として `NULL` が渡されたりピクセルタイプを指定しない場合には、新しいラスタは `rast1` と同じピクセルタイプになります。

数式の中で使える語は、元バンドのピクセル値を参照する `[rast1.val]`, `[rast2.val]`、1 始まりの列/行インデックスを参照する `[rast1.x]`, `[rast1.y]` などです。

Availability: 2.0.0

例: 2 バンドの共有と結合

元のラスタから 1 バンドラスタを生成します。元のラスタバンドの値について 2 で割った余りが入ります。

Create a cool set of rasters.

```

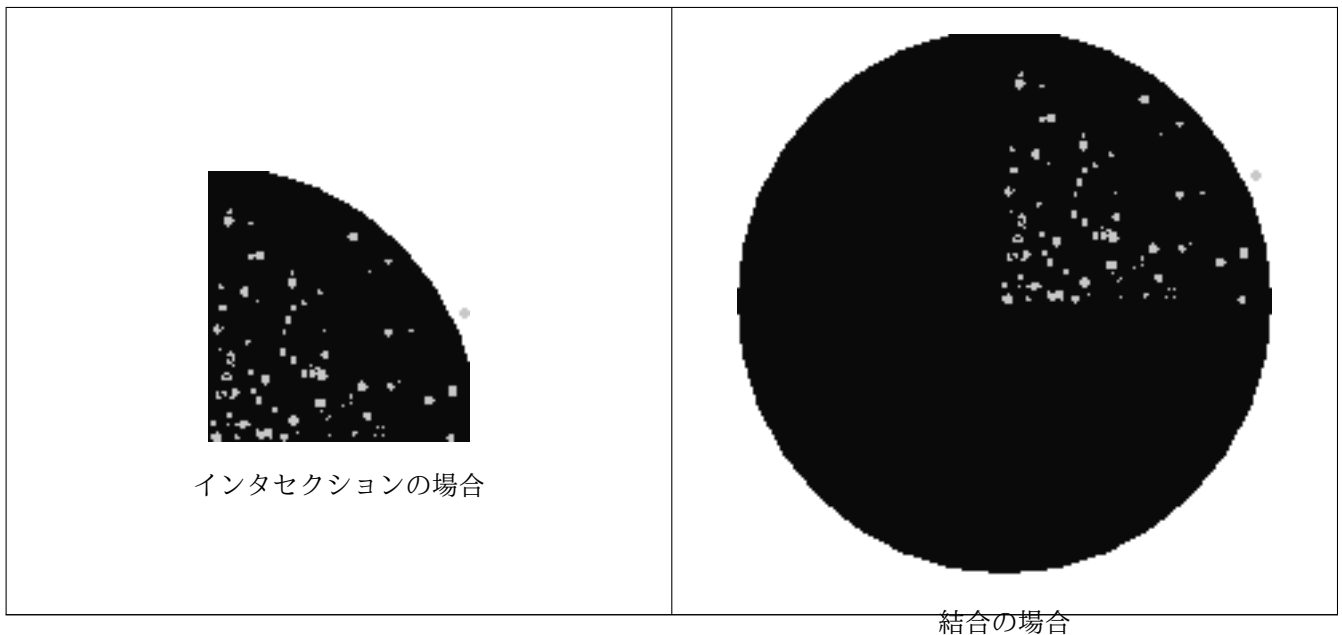
DROP TABLE IF EXISTS fun_shapes;
CREATE TABLE fun_shapes(rid serial PRIMARY KEY, fun_name text, rast raster);

-- 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

```

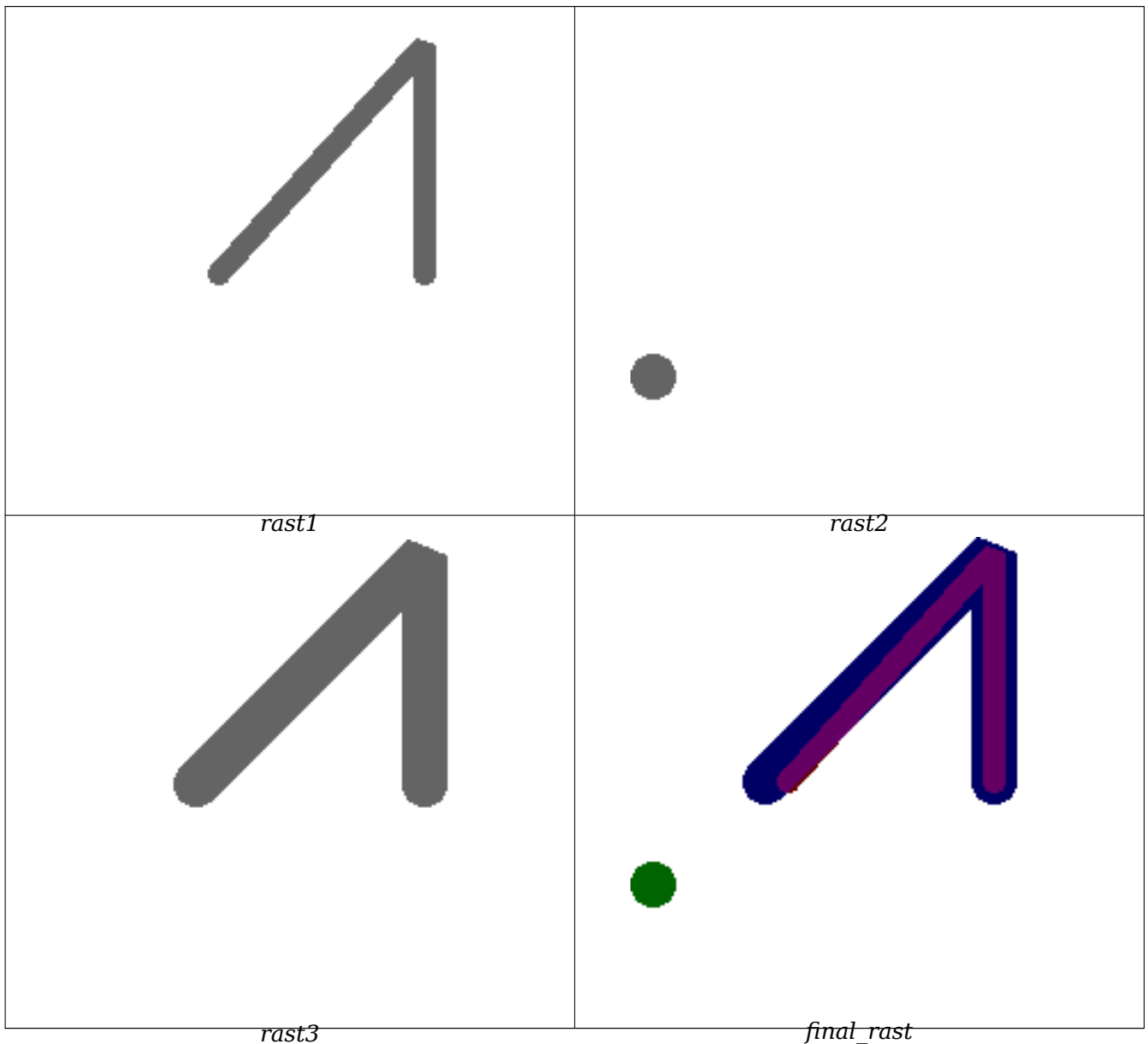


例: 別個バンドとしてキャンバス上にラスタをオーバーレイする

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;
```



例: 選択した区画の **2** メートル幅境界線を空中写真にオーバーレイする

Create new 3 band raster composed of first 2 clipped bands, and overlay of 3rd band with our geometry. This query took 3.6 seconds on PostGIS windows 64-bit install.

```
WITH pr AS
-- Note the order of operation: we clip all the rasters to dimensions of our region
(SELECT ST_Clip(rast,ST_Expand(geom,50) ) As rast, g.geom
  FROM aerals.o_2_boston AS r INNER JOIN
-- union our parcels of interest so they form a single geometry we can later intersect with
  (SELECT ST_Union(ST_Transform(geom,26986)) AS geom
   FROM landparcels WHERE pid IN('0303890000', '0303900000')) As g
  ON ST_Intersects(rast::geometry, ST_Expand(g.geom,50))
),
```

```
-- we then union the raster shards together
-- ST_Union on raster is kinda of slow but much faster the smaller you can get the rasters
-- therefore we want to clip first and then union
prunion AS
(SELECT ST_AddBand(NULL, ARRAY[ST_Union(rast,1),ST_Union(rast,2),ST_Union(rast,3)] ) As ←
    clipped,geom
FROM pr
GROUP BY geom)
-- return our final raster which is the unioned shard with
-- with the overlay of our parcel boundaries
-- add first 2 bands, then mapalgebra of 3rd band + geometry
SELECT ST_AddBand(
    ST_Band(clipped,ARRAY[1,2]),
    ST_MapAlgebraExpr(
        ST_Band(clipped,3),
        ST_AsRaster(ST_Buffer(ST_Boundary(geom),2),clipped, '8BUI',250),
        '[rast2.val]',
        '8BUI',
        'FIRST',
        '[rast2.val]',
        '[rast1.val]') ) As rast
FROM prunion;
```



青線が対象区画の境界です

関連情報

[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 バンド版 - 入力バンドに対する妥当な PostgreSQL 関数で形成された、指定したピクセルタイプとなる 1 バンドラスタを生成します。バンドを指定しない場合には、1 番と仮定します。

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);
```

説明



Warning

ST_MapAlgebraFct は 2.1.0 で非推奨になりました。代わりに **ST_MapAlgebra (callback function version)** を使います。

入力ラスタ (**rast**) に対して **onerasteruserfunc** で指定される妥当な PostgreSQL 関数で形成されたラスタを返します。生成されるラスタは指定したピクセルタイプとなる 1 バンドラスタです。**band** を指定しない場合には、1 番と仮定します。新しいラスタは、元のラスタと同じ地理参照、幅、高さを持ちますが、一つのバンドしか持ちません。

pixeltype が渡された場合には、新しいラスタのバンドは、そのピクセルタイプになります。**pixeltype** に NULL が渡された場合には、新しいラスタは入力 **rast** のバンドのピクセルタイプと同じになります。

onerasteruserfunc 引数は SQL 関数または PL/pgSQL 関数の名前とシグネチャで、**regprocedure** にキャストします。大変単純で本当に使えない PL/pgSQL 関数の例を挙げます:

```
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 は、2 または 3 の引数を受け付けます。すなわち、float8 値、任意引数の整数配列、VARIADIC 文字列配列です。第 1 引数はラスタセルごとの値です (ラスタのデータ型に関係なく)。第 2 引数は現在の処理セルの位置で、'**{x,y}**' であらわされます。第 3 引数は、**ST_MapAlgebraFct** へのパラメータの残っているもの全てが **userfunction** に渡されることを示します。

regprocedure 引数を SQL 関数に渡す場合には、渡し先の完全な関数シグネチャが必要で、さらに **regprocedure** にキャストします。上の例の PL/pgSQL 関数を引数として渡すには、引数の SQL は次のようになります:

```
'simple_function(float,integer[],text[])'::regprocedure
```

引数には関数名が含まれ、関数引数の型、関数名と引数型を引用符で括ったもの、**regprocedure** へのキャストが存在することに注意して下さい。

userfunction の第 3 引数は **variadic text** 配列です。どの **ST_MapAlgebraFct** にもついてくる全ての文字列引数は、指定された **userfunction** に、そのまま渡されて、**args** 引数内に入ります。

**Note**

VARIADIC キーワードに関する詳細情報については、PostgreSQL 文書と [Query Language \(SQL\) Functions](#) (訳注: 日本語版は「[問い合わせ言語 \(SQL\) 関数](#)」です) の “SQL Functions with Variable Numbers of Arguments” (訳注: 日本語版は「[可変長引数を取る SQL 関数](#)」) 節を参照して下さい。

**Note**

text[] 引数は、ユーザ関数 userfunction に処理用の引数を渡すかどうかにかかわらず求められます。

Availability: 2.0.0

例

元のラスタから 1 バンドラスタを生成します。元のラスタバンドの値について 2 で割った余りが入ります。

```
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

ピクセルタイプが 2BUI の 1 バンドラスタを生成します。元のラスタに対して再分類を行った値が入り、NODATA 値をユーザ関数に渡される引数の値 (0) に設定します。

```
ALTER TABLE dummy_rast ADD COLUMN map_rast2 raster;
CREATE FUNCTION classify_fct(pixel float, pos integer[], variadic args text[])
RETURNS float
AS
$$
DECLARE
    nodata float := 0;
BEGIN
    IF NOT args[1] IS NULL THEN
```

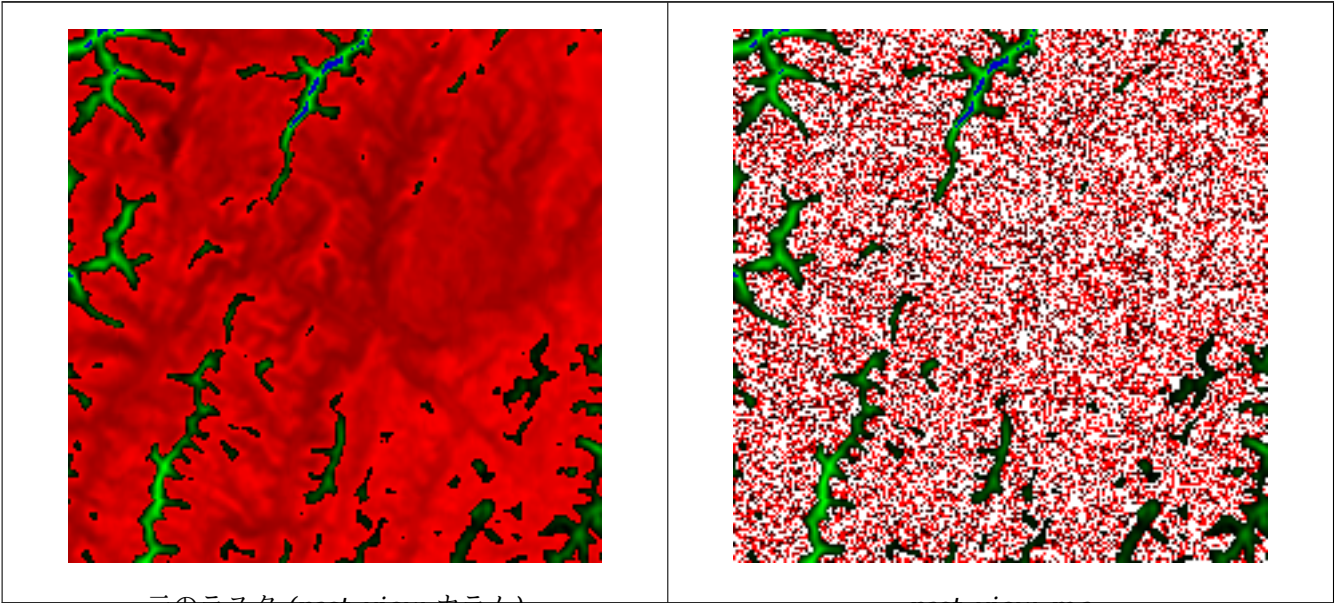
```
        nodata := args[1];
    END IF;
    IF pixel < 251 THEN
        RETURN 1;
    ELSIF pixel = 252 THEN
        RETURN 2;
    ELSIF pixel
> 252 THEN
        RETURN 3;
    ELSE
        RETURN nodata;
    END IF;
END;
$$
LANGUAGE 'plpgsql';
UPDATE dummy_rast SET map_rast2 = ST_MapAlgebraFct(rast,'2BUI','classify_fct(float,integer ↵
[],text[])':::regprocedure, '0') WHERE rid = 2;

SELECT DISTINCT ST_Value(rast,1,i,j) As origval, ST_Value(map_rast2, 1, i, j) As mapval
FROM dummy_rast CROSS JOIN generate_series(1, 5) AS i CROSS JOIN generate_series(1,5) AS j
WHERE rid = 2;

origval | mapval
-----+-----
    249 |      1
    250 |      1
    251 |
    252 |      2
    253 |      3
    254 |      3

SELECT ST_BandPixelType(map_rast2) As b1pixtyp
FROM dummy_rast WHERE rid = 2;

b1pixtyp
-----
2BUI
```



新しいバンドを三つ持つラスタを生成します。元のバンドを三つ持つラスタと同じピクセルタイプです。1 番バン

ドは地図代数関数によって変更され、残りの二つのバンドは値が代わりません。

```
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;
```

関連情報

[ST_MapAlgebraExpr](#), [ST_BandPixelType](#), [ST_GeoReference](#), [ST_SetValue](#)

11.12.10 ST_MapAlgebraFct

ST_MapAlgebraFct — 2 バンド版 - 二つの入力バンドに対する妥当な PostgreSQL 関数で形成された、指定したピクセルタイプとなる 1 バンドラスタを生成します。バンドを指定しない場合には、1 番と仮定します。“extenttype” のデフォルトは INTERSECTION です。

Synopsis

raster **ST_MapAlgebraFct**(raster rast1, raster rast2, regprocedure tworastuserfunc, text pixeltype=same_as_rast1, text extenttype=INTERSECTION, text[] VARIADIC userargs);
raster **ST_MapAlgebraFct**(raster rast1, integer band1, raster rast2, integer band2, regprocedure tworastuserfunc, text pixeltype=same_as_rast1, text extenttype=INTERSECTION, text[] VARIADIC userargs);

説明



Warning

ST_MapAlgebraFct は 2.1.0 で非推奨になりました。代わりに **ST_MapAlgebra (callback function version)** を使います。

二つの入力ラスタ (rast1, rast2) に対して tworastuserfunc で指定される妥当な PostgreSQL 関数で形成されるラスタを返します。band1 または band2 が指定されていない場合には、1 番と仮定します。新しいラスタは、元のラスタと同じ地理参照、幅、高さを持ちますが、一つのバンドしか持ちません。

`pixeltype` が渡された場合には、新しいラスタはそのピクセルタイプのバンドを持ちます。 `pixelteype` として `NULL` が渡されたりピクセルタイプを指定しない場合には、新しいラスタは `rast1` の入力バンドと同じピクセルタイプになります。

`tworastuserfunc` 引数は、SQL または PL/pgSQL 関数の名前とシングニチャでなければならず、`regprocedure` にキャストしなければなりません。PL/pgSQL 関数の例は次の通りです:

```
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` は 3 または 4 の引数を受け付けます。すなわち、一つ目の倍精度浮動小数点数、二つ目の倍精度浮動小数点数、任意引数の整列配列、`VARIADIC` 文字列配列です。第 1 引数は `rast1` のラスタセルごとの値です (ラスタのデータ型に関係なく)。第 2 引数は `rast2` のラスタセルごとの値です。第 3 引数は現在の処理セルの位置で、`{x,y}` であらわされます。第 4 引数は `ST_MapAlgebraFct` へのパラメータの残っているもの全てが `tworastuserfunc` に渡されることを示します。

`regprocedure` 引数を SQL 関数に渡す場合には、渡し先の完全な関数シグネチャが必要で、さらに `regprocedure` にキャストします。上の例の PL/pgSQL 関数を引数として渡すには、引数の SQL は次のようになります:

```
'simple_function(double precision, double precision, integer[], text[])'::regprocedure
```

引数には関数名が含まれ、関数引数の型、関数名と引数型を引用符で括ったもの、`regprocedure` へのキャストが存在することに注意して下さい。

`tworastuserfunc` 引数は `variadic text` 配列です。どの `ST_MapAlgebraFct` にもついてくる全ての文字列引数は、指定された `tworastuserfunc` に、そのまま渡されて、`userargs` 引数内に入ります。



Note

`VARIADIC` キーワードに関する詳細情報については、PostgreSQL 文書と [Query Language \(SQL\) Functions](#) (訳注: 日本語版は「[問い合わせ言語 \(SQL\) 関数](#)」です) の "SQL Functions with Variable Numbers of Arguments" (訳注: 日本語版は「[可変長引数を取る SQL 関数](#)」) 節を参照して下さい。



Note

`text[]` 引数は、ユーザ関数 `tworastuserfunc` に処理用の引数を渡すかどうかにかかわらず求められます。

Availability: 2.0.0

例: 別個のバンドとしてキャンバス上にラスタをオーバーレイする

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;

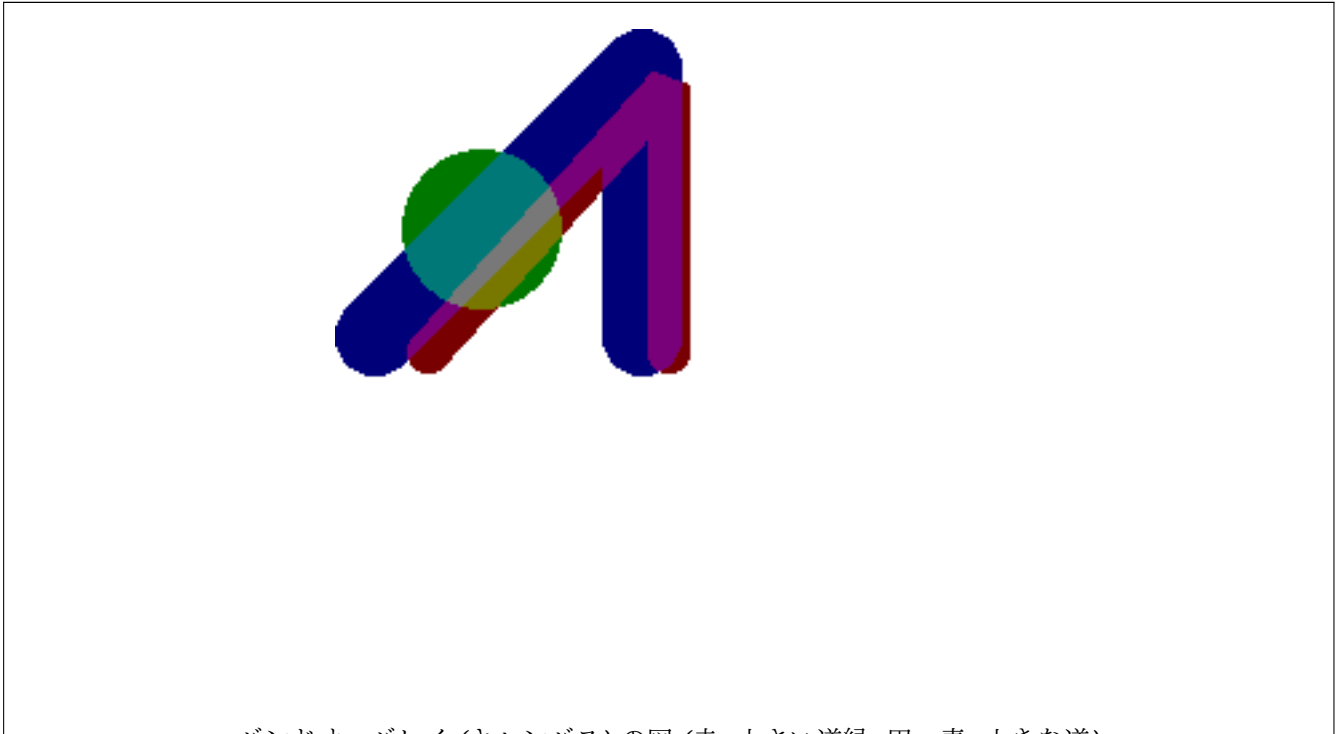
-- Map algebra on single band rasters and then collect with ST_AddBand
INSERT INTO map_shapes(rast,bnum,descrip)
SELECT ST_AddBand(ST_AddBand(rasts[1], rasts[2]),rasts[3]), 4, 'map bands overlay fct union ←
  (canvas)'
  FROM (SELECT ARRAY(SELECT ST_MapAlgebraFct(
    m1.rast,
    m2.rast,

```

```

'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;

```



バンドオーバーレイ (キャンバス) の図 (赤: 小さい道緑: 円、青: 大きな道)

追加引数を取るユーザ定義関数

```

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)';

```



追加引数を持つユーザ定義関数と同じラスタからの異なるバンド

関連情報

[ST_MapAlgebraExpr](#), [ST_BandPixelType](#), [ST_GeoReference](#), [ST_SetValue](#)

11.12.11 ST_MapAlgebraFctNgb

ST_MapAlgebraFctNgb — 1 バンド版: ユーザ定義 PostgreSQL 関数を使用する最近傍地図代数関数です。入力ラスタバンドの近傍の値を与えた PL/pgSQL ユーザ定義関数の結果からなるラスタを返します。

Synopsis

raster **ST_MapAlgebraFctNgb**(raster rast, integer band, text pixeltype, integer ngbwidth, integer ngbheight, regprocedure onerastngbuserfunc, text nodatamode, text[] VARIADIC args);

説明

**Warning**

ST_MapAlgebraFctNgb は 2.1.0 で非推奨になりました。代わりに **ST_MapAlgebra (callback function version)** を使います。

(1 バンド版) 入力ラスタバンドの近傍の値を与えた PL/pgSQL ユーザ定義関数の結果からなるラスタを返します。ユーザ定義関数は近傍のピクセル値を数の配列として取り、ピクセル毎に、ユーザ定義関数からの結果を返し、現在の対象ピクセルのピクセル値を関数の返り値に置き換えます。

rast ユーザ定義関数が評価されるラスタです。

band 評価されるラスタのバンド番号です。デフォルトは 1 です。

pixeltype 出力ラスタのピクセルタイプです。 **ST_BandPixelType** の一覧にあるものか、指定しないか、NULL を指定します。指定しないか NULL を指定した場合には、**rast** のピクセルタイプがデフォルトになります。結果ピクセル値がピクセルタイプが許容する範囲を超える場合には、切り詰められます。

ngbwidth セル単位の近傍の幅です。

ngbheight セル単位の近傍の高さです。

onerastngbuserfunc ラスタの単一バンドの近傍ピクセルに適用される PL/pgSQL ユーザ定義関数です。一つの要素は、近傍ピクセルの四角形を表現する数値の 2 次元配列です。

nodatamode NODATA または NULL となる近傍ピクセルにおける関数に渡す値を定義します。

'ignore': 近傍で遭遇した NODATA 値全てが計算から除外されます。ユーザ定義関数に必ず送られ、ユーザ定義関数が対処方法を決定します。

'NULL': 近傍で遭遇した NODATA 値全てが結果を NULL とします。この場合はユーザ定義関数が呼び出されません。

'value': 近傍で遭遇した NODATA 値全てが参照ピクセル (近傍の中心にあるピクセル) の値に置き換えられます。この値が NODATA になった場合には、'NULL' と同じ挙動を取ります (影響のある近傍について)。

args ユーザ定義関数に渡される引数です。

Availability: 2.0.0

例

単一タイルとしてロードされたカトリーナのラスタを使った例です。 <https://gdal.org/user/drivers/raster/postgisraster.html> に説明があります。また、 **ST_Rescale** の例で準備を行っています

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
```

```

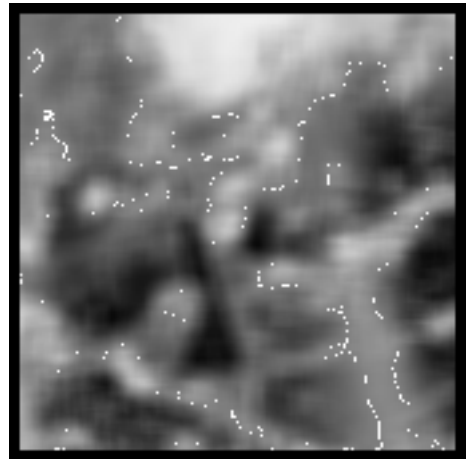
        _matrix := matrix;
        sum := 0;
        FOR x in array_lower(matrix, 1)..array_upper(matrix, 1) LOOP
            FOR y in array_lower(matrix, 2)..array_upper(matrix, 2) LOOP
                sum := sum + _matrix[x][y];
            END LOOP;
        END LOOP;
        RETURN (sum*1.0/(array_upper(matrix,1)*array_upper(matrix,2) ))::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;

```



ラスタの 1 番バンド



ピクセル毎に 4x4 ピクセル内のピクセル平均を計算したラスタ

関連情報

[ST_MapAlgebraFct](#), [ST_MapAlgebraExpr](#), [ST_Rescale](#)

11.12.12 ST_Reclass

ST_Reclass — 元のラスタから再分類したバンドタイプからなるラスタを生成します。**nband** は変更するバンドです。**nband** が指定されていない場合には、1 と仮定します。他の全てのバンドは変更せずに返します。可視画像の書式としてより単純な描画を行うために、16BUI バンドを 8BUI バンドに変換する、等のために使います。

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);
```

説明

reclassexpr によって定義される入力ラスタ (rast) の再分類処理を適用して新しいラスタを生成します。再分類の数式に関する説明は **reclassarg** にあります。band を指定しない場合には、1 番を仮定します。

新しいラスタは元のラスタと同じジオリファレンス、幅、高さになります。新しいラスタのバンドは **pixeltype** で指定したピクセルタイプになります。**reclassargset** を指定すると、しれじれば **reclassarg** は対象バンドのピクセルタイプを定義します。指定しないバンドは変更されずに返ります。

Availability: 2.0.0

例: 基本

元ラスタから新しいラスタを生成しますが、2 番バンドを 8BUI から 4BUI に変換して、101-254 を NODATA 値にします。

```
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

例: 複数の再分類を使った高度な例

元ラスタから新しいラスタを生成しますが、1 番、2 番、3 番バンドを 1BB, 4BUI, 4BUI にそれぞれ変換して再分類します。再分類の識別番号 (理論的には元ラスタと同数のバンド) を入力するために取る **VARIADIC reclassarg** 引数を使っています。

```
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

例: 単一バンドで **32BF** のラスタをの可視ラスタへの高度な対応付け

32BF バンドを一つ持つだけのラスタから三つのバンド (8BUI, 8BUI, 8BUI 可視ラスタ) を生成します

```

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)
        ]
    );

```

関連情報

[ST_AddBand](#), [ST_Band](#), [ST_BandPixelType](#), [ST_MakeEmptyRaster](#), [reclassarg](#), [ST_Value](#)

11.12.13 ST_ReclassExact

ST_ReclassExact — 元ラスターから再分類したバンドからなる新しいラスタを生成します。元ラスターのバンド内の値から新たに生成されるラスターのバンド内の値への 1 対 1 のマッピングを使います。

Synopsis

raster **ST_ReclassExact**(raster rast, double precision[] inputvalues, double precision[] outputvalues, integer bandnumber=1, text pixeltype=32BF, double precision nodatavalue=NULL);

説明

inputvalues 配列と **outputvalues** 配列によって定義される再分類処理を適用することで形成される新しいラスタを生成します。入力配列のピクセル値は対応する出力配列の値にマップされます。他のピクセル値は全て **nodatavalue** にマップされます。

出力ピクセルタイプのデフォルトは浮動小数点数ですが、**pixeltype** パラメータで指定できます。**bandnumber** を指定しない場合には、1 番を仮定します。

新しいラスタは元ラスタと同じジオリファレンス、同じ幅、同じ高さを持ちます。指定されていないバンドは変更されずに返ります。

Availability: 3.6.0

例: 基本

小さいラスタを作ってピクセルに新しい値をマップしていきます。

```
CREATE TABLE reclassexact (
    id integer,
    rast raster
);

--
-- Create a raster with just four pixels
-- [1  2]
-- [3  4]
--
INSERT INTO reclassexact (id, rast)
SELECT 1, ST_SetValues(ST_AddBand(ST_MakeEmptyRaster(2,    -- width in pixels
    2,    -- height in pixels
    0,    -- upper-left x-coordinate
    0,    -- upper-left y-coordinate
    1,    -- pixel size in x-direction
    -1,   -- pixel size in y-direction (negative for north-up)
    0,    -- skew in x-direction
    0,    -- skew in y-direction
    4326  -- SRID (e.g., WGS 84)
),
    '32BUI'::text, -- pixel type (e.g., '32BF' for float, '8BUI' for unsigned 8-bit int)
    0.0,          -- initial value for the band (e.g., 0.0 or a no-data value)
    -99          -- nodatavalue
),
    1, -- band number (usually 1 for single-band rasters)
    1, -- x origin for setting values (usually 1)
    1, -- y origin for setting values (usually 1)
    ARRAY[
        ARRAY[1, 2],
        ARRAY[3, 4]
    ]::double precision[][] -- 2D array of values
);

-- Reclass the values to new values
-- and dump the values of the new raster for display
WITH rc AS (
    SELECT ST_ReclassExact(rast,          -- input raster
        ARRAY[4,3,2,1],                  -- input map
        ARRAY[14,13,12,11],              -- output map
        1,                                -- band number to remap
        '32BUI'                           -- output raster pixtype
    ) AS rast
```

```
FROM reclassexact
WHERE id = 1
)
SELECT 'rce-1', (ST_DumpValues(rc.rast)).*
FROM rc;
```

関連情報

[ST_Reclass](#), [ST_AddBand](#), [ST_Band](#), [ST_MakeEmptyRaster](#)

11.12.14 ST_Union

ST_Union — ラスタスタイルの集合を結合して 1 以上のバンドからなる単一ラスタを返します。

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);
```

説明

ラスタスタイルの集合を結合して少なくとも一つのバンドからなる単一ラスタを返します。結果ラスタの範囲は集合全体の範囲です。インタセクトする場合には、結果値は、LAST (デフォルト), FIRST, MIN, MAX, COUNT, SUM, MEAN, RANGE のいずれかとなる **uniontype** で定義されます。

Note



ラスタを結合するには、全てが同じアラインメントを持たなくてはなりません。[ST_SameAlignment](#)と[ST_NotSameAlignmentReason](#)で詳細情報や助けとなる情報が得られます。アラインメント問題を修正する一つの方法として、[ST_Resample](#)を使い、アラインメントの同じ参照ラスタを使います。

Availability: 2.0.0

Enhanced: 2.1.0 速度が改善されました (完全に C 言語で記述しました)。

Availability: 2.1.0 **ST_Union(rast, unionarg)** の形式が導入されました。

Enhanced: 2.1.0 **ST_Union(rast)** (一つ目の形式) で、全ての入力ラスタの全てのバンドを結合するようになりました。以前の版の PostGIS では、一つ目のバンドと仮定していました。

Enhanced: 2.1.0 **ST_Union(rast, uniontype)** (四つ目の形式) で、全ての入力ラスタの全てのバンドを結合するようになりました。

例: 塊になっているラスタスタイルの単一バンドへの再構成

This creates a single band from first band of raster tiles. That form the original file system tile.

```
SELECT filename, ST_Union(rast,1) As file_rast
FROM sometable WHERE filename IN('dem01','dem02') GROUP BY filename;
```

例: タイルのジオメトリとインタセクトするタイルを結合した複数バンドのラスタ

This creates a multi band raster collecting all the tiles that intersect a line. Note that In 2.0, this would have just returned a single band raster. , new union works on all bands by default. This is equivalent to unionarg: ARRAY[ROW(1, 'LAST'), ROW(2, 'LAST'), ROW(3, 'LAST')]::unionarg[.]

```
SELECT ST_Union(rast)
FROM aeriāls.boston
WHERE ST_Intersects(rast, ST_GeomFromText('LINESTRING(230486 887771, 230500 88772)',26986) ←
);
```

例: タイルのジオメトリとインタセクトするタイルを結合した複数バンドのラスタ

バンドの部分集合が欲しいだけの場合や、バンドの並び順を変更したい場合には、より長い書き方にします。

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) ←
);
```

関連情報

[unionarg](#), [ST_Envelope](#), [ST_ConvexHull](#), [ST_Clip](#), [ST_Union](#)

11.13 組み込み地図代数コールバック関数

11.13.1 ST_Distinct4ma

ST_Distinct4ma — 近隣のピクセル値のうち一意となるものを数えるラスタ処理関数です。

Synopsis

float8 **ST_Distinct4ma**(float8[][] matrix, text nodatamode, text[] VARIADIC args);
double precision **ST_Distinct4ma**(double precision[][][] value, integer[][] pos, text[] VARIADIC user-args);

説明

近隣のピクセル値のうち一意となるものを数えます。



Note

一つ目の形式は、[ST_MapAlgebraFctNgb](#)へのコールバック引数として使用する専用コールバック関数です。



Note

二つ目の形式は、[ST_MapAlgebra \(callback function version\)](#)へのコールバック引数として使用する専用コールバック関数です。

**Warning**

一つ目の形式は、**ST_MapAlgebraFctNgb**が 2.1.0 で非推奨となったので、使用しないようになります。

Availability: 2.0.0

Enhanced: 2.1.0 二つ目の形式の追加

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)
```

関連情報

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 — 近隣のピクセル値の内挿補間を行うラスタ処理関数です。

Synopsis

double precision **ST_InvDistWeight4ma**(double precision[][][] value, integer[][] pos, text[] VARIADIC userargs);

説明

近隣のピクセル値の逆距離加重法 (IDW=Inverse Distance Weighted method) による内挿補間を行います。

userargs に渡される、二つの任意引数があります。一つ目は、逆距離加重法の式に与える 0 から 1 の間を取る冪数 (下の式の変数 k) です。指定しない場合には、デフォルトは 1 とします。二つ目は、対象ピクセルの値が近隣セルからの補間値に含まれている場合にのみ適用される重み率です。指定せず、かつ対象ピクセルが値を持っている場合には、そのピクセルの持つ値が返されます。

基本的な逆距離加重法の方程式は次の通りです。

$$\hat{z}(x_o) = \frac{\sum_{j=1}^m z(x_j) d_{ij}^{-k}}{\sum_{j=1}^m d_{ij}^{-k}}$$

k = 冪数で、0 から 1 の浮動小数点数を取ります。



Note

この関数は **ST_MapAlgebra (callback function version)** へのコールバック引数として使用する専用コールバック関数です。

Availability: 2.1.0

例

```
WITH foo AS (
  SELECT ST_SetValues(ST_AddBand(ST_MakeEmptyRaster(3, 3, 0, 0, 1, -1, 0, 0, 0), 1, '32BF ←
    ', NULL::double precision, -9999),
    1, 1, 1, ARRAY[
      [10, 10, 10],
      [10, NULL, 20],
      [20, 20, 20]
    ]::double precision[][])
) AS rast
)
SELECT ST_Value(
  ST_MapAlgebra(
    rast,
    1,
    'ST_InvDistWeight4ma(double precision[][][],integer[][],text[])'::regprocedure,
    '32BF',
    'FIRST',
    NULL,
    1,
    1
  ),
  2, 2
) AS interpolated_value
FROM foo;
```

15

関連情報

ST_MapAlgebra (callback function version), **ST_MinDist4ma**

11.13.3 ST_Max4ma

ST_Max4ma — 近隣のピクセル値の最大値を計算するラスタ処理関数です。

Synopsis

float8 **ST_Max4ma**(float8[][] matrix, text nodatamode, text[] VARIADIC args);
 double precision **ST_Max4ma**(double precision[][][] value, integer[][] pos, text[] VARIADIC userargs);

説明

近隣のピクセル値の最大値を計算します。

二つ目の形式では、`userargs` に渡す値で `NODATA` ピクセルに代入する値を指定できます。



Note

一つ目の形式は、**ST_MapAlgebraFctNgb**へのコールバック引数として使用する専用コールバック関数です。



Note

二つ目の形式は、**ST_MapAlgebra (callback function version)**へのコールバック引数として使用する専用コールバック関数です。



Warning

一つ目の形式は、**ST_MapAlgebraFctNgb**が 2.1.0 で非推奨となったので、使用しないようになります。

Availability: 2.0.0

Enhanced: 2.1.0 二つ目の形式の追加

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)
```

関連情報

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 — 近隣のピクセル値の平均値を計算するラスタ処理関数です。

Synopsis

```
float8 ST_Mean4ma(float8[][] matrix, text nodatamode, text[] VARIADIC args);
double precision ST_Mean4ma(double precision[][][] value, integer[][] pos, text[] VARIADIC userargs);
```

説明

近隣のピクセル値の平均値を計算します。

二つ目の形式では、`userargs` に渡す値で NODATA ピクセルに代入する値を指定できます。



Note

一つ目の形式は、**ST_MapAlgebraFctNgb**へのコールバック引数として使用する専用コールバック関数です。



Note

二つ目の形式は、**ST_MapAlgebra (callback function version)**へのコールバック引数として使用する専用コールバック関数です。



Warning

一つ目の形式は、**ST_MapAlgebraFctNgb**が 2.1.0 で非推奨となったので、使用しないようになります。

Availability: 2.0.0

Enhanced: 2.1.0 二つ目の形式の追加

例: 一つ目の形式

```
SELECT
  rid,
  ST_Value(ST_MapAlgebraFctNgb(rast, 1, '32BF', 1, 1, 'ST_Mean4ma(float[][],text,text[]) ←
    '::regprocedure, 'ignore', NULL), 2, 2
  )
FROM dummy_rast
WHERE rid = 2;
```

```
rid |      st_value
-----+-----
  2 | 253.222229003906
(1 row)
```

例: 二つ目の形式

```
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)
```

関連情報

[ST_MapAlgebraFctNgb](#), [ST_MapAlgebra \(callback function version\)](#), [ST_Min4ma](#), [ST_Max4ma](#), [ST_Sum4ma](#), [ST_Range4ma](#), [ST_StdDev4ma](#)

11.13.5 ST_Min4ma

ST_Min4ma — 近隣のピクセル値の最小値を計算するラスタ処理関数です。

Synopsis

```
float8 ST_Min4ma(float8[][] matrix, text nodatamode, text[] VARIADIC args);
double precision ST_Min4ma(double precision[][][] value, integer[][] pos, text[] VARIADIC userargs);
```

説明

近隣のピクセル値の最小値を計算します。

二つ目の形式では、`userargs` に渡す値で `NODATA` ピクセルに代入する値を指定できます。



Note

一つ目の形式は、[ST_MapAlgebraFctNgb](#)へのコールバック引数として使用する専用コールバック関数です。



Note

二つ目の形式は、[ST_MapAlgebra \(callback function version\)](#)へのコールバック引数として使用する専用コールバック関数です。



Warning

一つ目の形式は、[ST_MapAlgebraFctNgb](#)が 2.1.0 で非推奨となったので、使用しないようになります。

Availability: 2.0.0

Enhanced: 2.1.0 二つ目の形式の追加

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)
```

関連情報

[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 — 対象ピクセルと値を持つ近隣ピクセルとの最短距離をピクセル単位で返すラスタ処理関数です。

Synopsis

double precision **ST_MinDist4ma**(double precision[][][] value, integer[][] pos, text[] VARIADIC userargs);

説明

対象ピクセルと値を持つ近隣ピクセルとの最短距離をピクセル単位で返します。



Note

この関数の意図は、[ST_InvDistWeight4ma](#)から対象ピクセルの内挿補間された値の有用性を判断することを補助するのに有益なデータポイントを提供することです。この関数は近隣セルの密度が小さい時に特に使えます。



Note

この関数は[ST_MapAlgebra \(callback function version\)](#)へのコールバック引数として使用する専用コールバック関数です。

Availability: 2.1.0

例

```
WITH foo AS (
  SELECT ST_SetValues(ST_AddBand(ST_MakeEmptyRaster(3, 3, 0, 0, 1, -1, 0, 0, 0), 1, '32BF' ←
    ', NULL::double precision, -9999),
    1, 1, 1, ARRAY[
      [10, 10, 10],
      [10, NULL, 20],
      [20, 20, 20]
    ]::double precision[][])
  ) AS rast
)
SELECT ST_Value(
  ST_MapAlgebra(
    rast,
    1,
    'ST_MinDist4ma(double precision[][][],integer[][],text[])'::regprocedure,
    '32BF',
    'FIRST',
    NULL,
    1,
    1
  ),
  2, 2
) AS min_distance
FROM foo;
```

1

関連情報

[ST_MapAlgebra \(callback function version\)](#), [ST_InvDistWeight4ma](#)

11.13.7 ST_Range4ma

ST_Range4ma — 近隣のピクセル値の範囲を計算するラスタ処理関数です。

Synopsis

```
float8 ST_Range4ma(float8[][] matrix, text nodatamode, text[] VARIADIC args);
double precision ST_Range4ma(double precision[][][] value, integer[][] pos, text[] VARIADIC user-
args);
```

説明

近隣のピクセル値の範囲を計算します。

二つ目の形式では、userargs に渡す値で NODATA ピクセルに代入する値を指定できます。



Note

一つ目の形式は、[ST_MapAlgebraFctNgb](#)へのコールバック引数として使用する専用コールバック関数です。

**Note**

二つ目の形式は、**ST_MapAlgebra (callback function version)**へのコールバック引数として使用する専用コールバック関数です。

**Warning**

一つ目の形式は、**ST_MapAlgebraFctNgb**が 2.1.0 で非推奨となったので、使用しないようになります。

Availability: 2.0.0

Enhanced: 2.1.0 二つ目の形式の追加

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)
```

関連情報

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 — 近隣のピクセル値の標準偏差を計算するラスタ処理関数です。

Synopsis

```
float8 ST_StdDev4ma(float8[][] matrix, text nodatamode, text[] VARIADIC args);
double precision ST_StdDev4ma(double precision[][][] value, integer[][] pos, text[] VARIADIC user-args);
```

説明

近隣のピクセル値の標準偏差を計算します。

**Note**

一つ目の形式は、**ST_MapAlgebraFctNgb**へのコールバック引数として使用する専用コールバック関数です。

**Note**

二つ目の形式は、**ST_MapAlgebra (callback function version)**へのコールバック引数として使用する専用コールバック関数です。

**Warning**

一つ目の形式は、**ST_MapAlgebraFctNgb**が 2.1.0 で非推奨となったので、使用しないようになります。

Availability: 2.0.0

Enhanced: 2.1.0 二つ目の形式の追加

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)
```

関連情報

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 — 近隣のピクセル値の合計を計算するラスタ処理関数です。

Synopsis

```
float8 ST_Sum4ma(float8[][] matrix, text nodatamode, text[] VARIADIC args);
double precision ST_Sum4ma(double precision[][][] value, integer[][] pos, text[] VARIADIC user-args);
```

説明

近隣のピクセル値の合計を計算します。

二つ目の形式では、`userargs` に渡す値で `NODATA` ピクセルに代入する値を指定できます。

**Note**

一つ目の形式は、`ST_MapAlgebraFctNgb` へのコールバック引数として使用する専用コールバック関数です。

**Note**

二つ目の形式は、`ST_MapAlgebra (callback function version)` へのコールバック引数として使用する専用コールバック関数です。

**Warning**

一つ目の形式は、`ST_MapAlgebraFctNgb` が 2.1.0 で非推奨となったので、使用しないようになります。

Availability: 2.0.0

Enhanced: 2.1.0 二つ目の形式の追加

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)
```

関連情報

`ST_MapAlgebraFctNgb`, `ST_MapAlgebra (callback function version)`, `ST_Min4ma`, `ST_Max4ma`, `ST_Mean4ma`, `ST_Range4ma`, `ST_Distinct4ma`, `ST_StdDev4ma`

11.14 ラスタ処理: DEM (標高)

11.14.1 ST_Aspect

`ST_Aspect` — 標高ラスタバンドの傾斜方向 (デフォルトの単位は度) を返します。地形解析に使えます。

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);

説明

標高ラスタブンドの傾斜方向 (デフォルトの単位は度) を返します。地図代数を利用して、傾斜方向方程式を隣接ピクセルに適用します。

units は、傾斜方向の単位を示します。取りえる値は **RADIANS**, **DEGREES** (デフォルト) です。

units が **RADIANS** の時、値は 0 から 2π ラジアンの間で、北から時計回りに計ります。

units が **DEGREES** の時、値は 0 から 360 度の間で、北から時計回りに計ります。

ピクセルの傾斜角が 0 の場合には、傾斜方向は-1 とします。



Note

傾斜角、傾斜方および陰影起伏に関する詳細情報については、[ESRI - How hillshade works](#)および[ERDAS Field Guide - Aspect Images](#)を参照して下さい。

Availability: 2.0.0

Enhanced: 2.1.0 **ST_MapAlgebra()** を使用するようにし、**interpolate_nodata** 任意引数を追加しました。

Changed: 2.1.0 以前の版では、戻り値はラジアン単位でした。現在は、デフォルトでは度で返します。

例: 一つ目の形式

```
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)
```

例: 二つ目の形式

カバレッジのタイルの完全な例。

```
WITH foo AS (
  SELECT ST_Tile(ST_SetValues(ST_AddBand(ST_MakeEmptyRaster(6, 6, 0, 0, 1, -1, 0, 0, 0),
    1, '32BF', 0, -9999
  ),
    1, 1, 1, ARRAY[
      [1, 1, 1, 1, 1, 1],
      [1, 1, 1, 1, 2, 1],
      [1, 2, 2, 3, 3, 1],
      [1, 1, 3, 2, 1, 1],
      [1, 2, 2, 1, 2, 1],
      [1, 1, 1, 1, 1, 1]
    ]::double precision[]
  ),
    2, 2
  ) AS rast
)
SELECT
  t1.rast,
  ST_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;
```

関連情報

[ST_MapAlgebra \(callback function version\)](#), [ST_TRI](#), [ST_TPI](#), [ST_Roughness](#), [ST_HillShade](#), [ST_Slope](#)

11.14.2 ST_HillShade

ST_HillShade — 与えられた方位、高度、明度、スケールの入力を使って標高ラスタブンドの仮想照明を返します。

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);

説明

与えられた方位、高度、明度、スケールの入力を使って標高ラスタブンドの仮想照明を返します。地図代数を使って、陰影図方程式を隣接ピクセルに適用します。返されるピクセル値は 0 と 255 の間です。

azimuth は 0 から 360 度の間の値で、北から時計回りに計ります。

altitude は 0 から 90 度の間で、0 が水平、90 が鉛直上向きです。

max_bright は 0 から 255 までの間で、0 は明度なしで、255 が最大明度です。

`scale` は鉛直単位と水平単位との比です。フィート: 経度緯度では `scale=370400` となり、メートル: 経度緯度では `scale=111120` となります。

`interpolate_nodata` が TRUE の場合には、入力ラスタの NODATA ピクセルの値は陰影図を計算する前に `ST_InvDistWeight4ma` を使って内挿を行います。



Note

陰影図に関する詳細情報については [How hillshade works](#) をご覧下さい。

Availability: 2.0.0

Enhanced: 2.1.0 `ST_MapAlgebra()` を使用するようし、`interpolate_nodata` 任意引数を追加しました。

Changed: 2.1.0 以前の版では `azimuth` と `altitude` はラジアン単位で表現しました。現在は `azimuth` と `altitude` は度単位で表現します。

例: 一つ目の形式

```
WITH foo AS (
  SELECT ST_SetValues(ST_AddBand(ST_MakeEmptyRaster(5, 5, 0, 0, 1, -1, 0, 0, 0), 1, '32BF',
    0, -9999),
    1, 1, 1, ARRAY[
      [1, 1, 1, 1, 1],
      [1, 2, 2, 2, 1],
      [1, 2, 3, 2, 1],
      [1, 2, 2, 2, 1],
      [1, 1, 1, 1, 1]
    ]::double precision[])
) AS rast
)
SELECT
  ST_DumpValues(ST_HillShade(rast, 1, '32BF'))
FROM foo
```

```
-----
(1, "{NULL,NULL,NULL,NULL,NULL},{NULL,251.32763671875,220.749786376953,147.224319458008,
  NULL},{NULL,220.749786376953,180.312225341797,67.7497863769531,NULL},{NULL,
  147.224319458008,
  67.7497863769531,43.1210060119629,NULL},{NULL,NULL,NULL,NULL,NULL}}")
(1 row)
```

例: 二つ目の形式

カバレッジのタイルの完全な例。

```
WITH foo AS (
  SELECT ST_Tile(ST_SetValues(ST_AddBand(ST_MakeEmptyRaster(6, 6, 0, 0, 1, -1, 0, 0, 0),
    1, '32BF', 0, -9999),
    ),
    1, 1, 1, ARRAY[
```



```

        [1, 1, 1, 1, 1, 1],
        [1, 1, 1, 1, 2, 1],
        [1, 2, 2, 3, 3, 1],
        [1, 1, 3, 2, 1, 1],
        [1, 2, 2, 1, 2, 1],
        [1, 1, 1, 1, 1, 1]
    ]::double precision[]
),
    2, 2
) AS rast
)
SELECT
    t1.rast,
    ST_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;

```

関連情報

[ST_MapAlgebra \(callback function version\)](#), [ST_TRI](#), [ST_TPI](#), [ST_Roughness](#), [ST_Aspect](#), [ST_Slope](#)

11.14.3 ST_Roughness

ST_Roughness — DEM の「粗度」を計算したラスタを返します。

Synopsis

raster **ST_Roughness**(raster rast, integer nband, raster customextent, text pixelttype="32BF", boolean interpolate_nodata=FALSE);

説明

指定された領域の最大値から最小値を減じることで DEM の「粗度」を計算したラスタを返します。

Availability: 2.1.0

例

```

WITH foo AS (
    SELECT ST_SetValues(ST_AddBand(ST_MakeEmptyRaster(5, 5, 0, 0, 1, -1, 0, 0, 0), 1, '32BF' ←
        ', 0, -9999),
        1, 1, 1, ARRAY[
            [1, 1, 2, 3, 3],
            [1, 2, 4, 15, 3],
            [2, 4, 10, 7, 5],
            [3, 5, 7, 6, 4],
            [3, 3, 5, 4, 4]
        ]::double precision[][])
    ) AS rast
)
SELECT ST_Value(ST_Roughness(rast, 1, '32BF'), 3, 3) AS roughness
FROM foo;

```

関連情報

[ST_MapAlgebra \(callback function version\)](#), [ST_TRI](#), [ST_TPI](#), [ST_Slope](#), [ST_HillShade](#), [ST_Aspect](#)

11.14.4 ST_Slope

ST_Slope — 標高ラスタバンドの傾斜角 (デフォルトでは度単位) を返します。地形解析に使えます。

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);
```

説明

標高ラスタバンドの傾斜角 (デフォルトでは度単位) を返します。地図代数を使って、傾斜角方程式を隣接ピクセルに適用します。

units は傾斜角の単位を示します。取りえる値は **RADIANS**, **DEGREES** (デフォルト), **PERCENT** です。

scale は鉛直単位と水平単位との比です。フィート: 経度緯度では **scale=370400** となり、メートル: 経度緯度では **scale=111120** となります。

interpolate_nodata が **TRUE** の場合には、入力ラスタの **NODATA** ピクセルの値は表面傾斜角を計算する前に [ST_InvDistWeight4ma](#) を使って内挿を行います。



Note

傾斜角、傾斜方および陰影起伏に関する詳細情報については、[ESRI - How hillshade works](#) および [ERDAS Field Guide - Slope Images](#) を参照して下さい。

Availability: 2.0.0

Enhanced: 2.1.0 **ST_MapAlgebra()** を使用するようにし、**units**, **scale**, **interpolate_nodata** 任意引数を追加しました。

Changed: 2.1.0 以前の版では、返り値はラジアン単位でした。現在は、デフォルトでは度で返します。

例: 一つ目の形式

```
WITH foo AS (
  SELECT ST_SetValues(ST_AddBand(ST_MakeEmptyRaster(5, 5, 0, 0, 1, -1, 0, 0, 0), 1, '32BF',
    1, 0, -9999),
    1, 1, 1, ARRAY[
      [1, 1, 1, 1, 1],
      [1, 2, 2, 2, 1],
      [1, 2, 3, 2, 1],
      [1, 2, 2, 2, 1],
    ]
  )
```

```

        [1, 1, 1, 1, 1]
      ]::double precision[][])
    ) AS rast
)
SELECT
  ST_DumpValues(ST_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)

```

例: 二つ目の形式

カバレッジのタイルの完全な例。

```

WITH foo AS (
  SELECT ST_Tile(ST_SetValues(ST_AddBand(ST_MakeEmptyRaster(6, 6, 0, 0, 1, -1, 0, 0, 0),
    1, '32BF', 0, -9999
  ),
    1, 1, 1, ARRAY[
      [1, 1, 1, 1, 1, 1],
      [1, 1, 1, 1, 2, 1],
      [1, 2, 2, 3, 3, 1],
      [1, 1, 3, 2, 1, 1],
      [1, 2, 2, 1, 2, 1],
      [1, 1, 1, 1, 1, 1]
    ]::double precision[]
  ),
    2, 2
  ) AS rast
)
SELECT
  t1.rast,
  ST_Slope(ST_Union(t2.rast), 1, t1.rast)
FROM foo t1
CROSS JOIN foo t2
WHERE ST_Intersects(t1.rast, t2.rast)
GROUP BY t1.rast;

```

関連情報

[ST_MapAlgebra \(callback function version\)](#), [ST_TRI](#), [ST_TPI](#), [ST_Roughness](#), [ST_HillShade](#), [ST_Aspect](#)

11.14.5 ST_TPI

ST_TPI — 地形的位置指数を計算したラスタを返します。

Synopsis

raster **ST_TPI**(raster rast, integer nband, raster customextent, text pixeltype="32BF" , boolean interpolate_nodata=FALSE);

説明

地形的位置指数 (TPI=Topographic Position Index) を計算します。地形的位置指数は、半径 1 における近傍平均値から中心セルの値を引いたものです。



Note

この関数は、近傍平均値半径が 1 についてのみ対応しています。

Availability: 2.1.0

例

```
WITH foo AS (
  SELECT ST_SetValues(ST_AddBand(ST_MakeEmptyRaster(5, 5, 0, 0, 1, -1, 0, 0, 0), 1, '32BF',
    ' ', 0, -9999),
    1, 1, 1, ARRAY[
      [1, 1, 2, 3, 3],
      [1, 2, 4, 15, 3],
      [2, 4, 10, 7, 5],
      [3, 5, 7, 6, 4],
      [3, 3, 5, 4, 4]
    ]::double precision[][])
  ) AS rast
)
SELECT ST_Value(ST_TPI(rast, 1, '32BF'), 3, 3) AS tpi
FROM foo;
```

3.75

関連情報

[ST_MapAlgebra \(callback function version\)](#), [ST_TRI](#), [ST_Roughness](#), [ST_Slope](#), [ST_HillShade](#), [ST_Aspect](#)

11.14.6 ST_TRI

ST_TRI — 起伏指標を計算したラスタを返します。

Synopsis

raster **ST_TRI**(raster rast, integer nband, raster customextent, text pixeltype="32BF" , boolean interpolate_nodata=FALSE);

説明

起伏指標 (TRI=Terrain Ruggedness Index) は、中心ピクセルと隣接ピクセルを、差の絶対値の平均を取って比較することで計算されます。

**Note**

この関数は、近傍平均値半径が 1 についてのみ対応しています。

Availability: 2.1.0

例

```
WITH foo AS (
  SELECT ST_SetValues(ST_AddBand(ST_MakeEmptyRaster(5, 5, 0, 0, 1, -1, 0, 0, 0), 1, '32BF',
    0, 0, -9999),
    1, 1, 1, ARRAY[
      [1, 1, 2, 3, 3],
      [1, 2, 4, 15, 3],
      [2, 4, 10, 7, 5],
      [3, 5, 7, 6, 4],
      [3, 3, 5, 4, 4]
    ]::double precision[][])
) AS rast
SELECT ST_Value(ST_TRI(rast, 1, '32BF'), 3, 3) AS tri
FROM foo;
```

5

関連情報

[ST_MapAlgebra \(callback function version\)](#), [ST_Roughness](#), [ST_TPI](#), [ST_Slope](#), [ST_HillShade](#), [ST_Aspect](#)

11.14.7 ST_InterpolateRaster

ST_InterpolateRaster — X 値と Y 値を使用してグリッド上のポイントを配置し、ポイントの Z 値をサーフェス標高として配置し、3 次元ポイントの入力セットに基づいてグリッドサーフェスを補間します。

Synopsis

raster **ST_InterpolateRaster**(geometry input_points, text algorithm_options, raster template, integer template_band_num=1);

説明

X 値と Y 値を使用してグリッド上のポイントを配置し、ポイントの Z 値をサーフェス標高として配置し、3 次元ポイントの入力セットに基づいてグリッドサーフェスを補間します。逆距離、逆距離最近傍、移動平均、最近傍、線形補間の五つの補間アルゴリズムが使用できます。アルゴリズムとそれらのパラメータに関する詳細情報につ

いては[gdal_grid documentation](#)を参照して下さい。どのように補間計算が行われているかの詳細情報については[GDAL grid tutorial](#)をご覧ください。

入力パラメータは次の通りです:

input_points 補間を駆動するポイント。Z 値を持つジオメトリなら全て受け付けられ、全ての入力ポイントが使用されます。

algorithm_options アルゴリズムとそのオプションを定義する文字列で、[gdal_grid](#)で使われる書式で記述します。たとえば、逆距離補間

template 出力ラスタのジオメトリを駆動するためのラスタテンプレート。幅、高さ、ピクセルサイズ、空間範囲、ピクセルタイプがこのテンプレートから読み込まれます。

template_band_num デフォルトではテンプレートラスタの最初のバンドが出力ラスタの駆動に使われますが、このパラメータで調整できます。

Availability: 3.2.0

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')  
)
```

関連情報

[ST_Contour](#)

11.14.8 ST_Contour

ST_Contour — 与えられたラスタバンドから等高線ベクタを生成します。[GDAL 等高線生成アルゴリズム](#)を使います。

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);

説明

与えられたラスタバンドから等高線ベクタを生成します。[GDAL 等高線生成アルゴリズム](#)を使います。

fixed_levels パラメータが空でない配列である時、**level_interval** と **level_base** とは無視されます。

入力パラメータは次の通りです:

rast 等高線を生成するラスタ

bandnumber 等高線を生成するバンド

level_interval 生成する等高線の標高間隔

level_base 等高線の標高間隔に適用される相対値の「基礎値」です。通常は 0 ですが、そうでない場合もあります。10m 等高線であって、5, 15, 25 ... で等高線を生成する場合には、LEVEL_BASE は 5 になります。

fixed_levels 生成する等高線の標高間隔

polygonize true の場合には、等高線ポリゴンが作られます。ラインストリングではありません。

返り値は次に示す属性を持つ行の集合です。

geom 等高線のジオメトリ。

id GDAL が等高線に与える一意の識別子。

value ラインが表現するラスタ値。標高 DEM 入力の場合、出力等高線の標高となります。

Availability: 3.2.0

例

```
WITH c AS (  
  SELECT (ST_Contour(rast, 1, fixed_levels => ARRAY[100.0, 200.0, 300.0])).*  
  FROM dem_grid WHERE rid = 1  
)  
SELECT ST_AsText(geom), id, value  
FROM c;
```

関連情報

[ST_InterpolateRaster](#)

11.15 ラスタ処理: ラスタからジオメトリ

11.15.1 Box3D

Box3D — ラスタを囲むボックスの box3d 表現を返します。

Synopsis

box3d **Box3D**(raster rast);

説明

ラスタの範囲を表現するボックスを返します。

ポリゴンは、バウンディングボックス ((MINX, MINY), (MAXX, MAXY)) の 4 隅のポイントにより定義されます。

Changed: 2.0.0 以前の版では、box3d でなく box2d を使っていました。box2d は非推奨型となり、box3d に変更しました。

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)

関連情報

[ST_Envelope](#)

11.15.2 ST_ConvexHull

ST_ConvexHull — `BandNoDataValue` と等しいピクセル値を含むラスタの凸包ジオメトリを返します。一般的な形状でスキューのないラスタでは、`ST_Envelope` と同じ結果になります。不規則な形状をしているか回転しているラスタでのみ使います。

Synopsis

geometry **ST_ConvexHull**(raster rast);

説明

`NoDataValue` 値のピクセルを含むラスタの凸包ジオメトリを返します。一般的な形状でスキューのないラスタでは、`ST_Envelope` と同じ結果になります。不規則な形状をしているかスキューのあるラスタでのみ使います。



Note

`ST_Envelope` は、座標値の小数部を切り捨て、ラスタのまわりに小さなバッファを追加します。小数部の切り捨てを行わない `ST_ConvexHull` の答と若干異なります。

例

この図については[PostGIS Raster Specification](#)を参照して下さい。

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)↔
)		

関連情報

[ST_Envelope](#), [ST_MinConvexHull](#), [ST_ConvexHull](#), [ST_AsText](#)

11.15.3 ST_DumpAsPolygons

ST_DumpAsPolygons — 指定されたラスタブンドから geomval (geom,val) 行の集合を返します。バンドを指定しない場合のデフォルトは 1 です。

Synopsis

setof geomval **ST_DumpAsPolygons**(raster rast, integer band_num=1, boolean exclude_nodata_value=TRUE)

説明

集合を返す関数 (SRF=set-returning function) です。geomval 行の集合を返します。geomval はジオメトリ (geom) とピクセルバンド値 (val) からなります。それぞれのポリゴンは、指定したバンドの、val で示される値と同じピクセル値を持っている全てのピクセルの結合です。

ST_DumpAsPolygon はラスタのポリゴン化に使えます。新しい行を生成するので GROUP BY の逆です。たとえば、単一ラスタを複数の POLYGON/MULTIPOLYGON に展開できます。

Changed 3.3.0, 能率向上のため評価と修正が無効になりました。不正なジオメトリを返すことがあります。

Changed 3.7.0, PostgreSQL の処理中断を尊重して、キャンセルや手続時間切れによる処理の停止を即座に行うようになりました。

Availability: GDAL 1.7 以上が必要です。



Note
バンドに NODATA 値が設定されている場合には、exclude_nodata_value=false が設定されている場合を除いて、NODATA 値を持つピクセルは返りません。



Note
ラスタ内の与えられた値を持つピクセルの数にのみ注意する場合にはST_ValueCountを使う方が速いです。



Note
これは、ピクセル値にかかわらずピクセルごとに一つのジオメトリを返す ST_PixelAsPolygons と違います。

例

This syntax uses a LATERAL join.

```
SELECT val, ST_AsText(geom) As geomwkt
FROM (
SELECT dp.*
FROM dummy_rast, LATERAL ST_DumpAsPolygons(rast) AS dp
WHERE rid = 2
) As foo
WHERE val BETWEEN 249 and 251
ORDER BY val;
```

val	geomwkt
249	POLYGON((3427927.95 5793243.95,3427927.95 5793243.85,3427928 5793243.85,3427928 5793243.95,3427927.95 5793243.95))
250	POLYGON((3427927.75 5793243.9,3427927.75 5793243.85,3427927.8 5793243.85,3427927.8 5793243.9,3427927.75 5793243.9))
250	POLYGON((3427927.8 5793243.8,3427927.8 5793243.75,3427927.85 5793243.75,3427927.85 5793243.8, 3427927.8 5793243.8))
251	POLYGON((3427927.75 5793243.85,3427927.75 5793243.8,3427927.8 5793243.8,3427927.8 5793243.85,3427927.75 5793243.85))

関連情報

[geomval](#), [ST_AsRasterAgg](#), [ST_Value](#), [ST_Polygon](#), [ST_ValueCount](#)

11.15.4 ST_Envelope

ST_Envelope — ラスタの範囲のポリゴン表現を返します。

Synopsis

```
geometry ST_Envelope(raster rast);
```

説明

SRID で定義されている空間参照系上でのラスタの範囲のポリゴン表現を返します。ポリゴンで表現される float8 の最小バウンディングボックスです。

ポリゴンはバウンディングボックスの隅のポイント、すなわち ((MINX, MINY), (MINX, MAXY), (MAXX, MAXY), (MAXX, MINY), (MINX, MINY)) です。

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))

関連情報

[ST_Envelope](#), [ST_AsText](#), [ST_SRID](#)

11.15.5 ST_MinConvexHull

ST_MinConvexHull — NODATA 値を除いたラスタの凸包ジオメトリを返します。

Synopsis

geometry **ST_MinConvexHull**(raster rast, integer nband=NULL);

説明

NODATA 値を除いたラスタの凸包ジオメトリを返します。nband が NULL の場合には、ラスタの全てのバンドが考慮されます。

Availability: 2.1.0

例

```
WITH foo AS (
  SELECT
    ST_SetValues(ST_SetValues(ST_AddBand(ST_AddBand(ST_MakeEmptyRaster(9, 9, 0, 0, 1, ←
      -1, 0, 0, 0)), 1, '8BUI', 0, 0), 2, '8BUI', 1, 0),
      1, 1, 1,
      ARRAY[
        [0, 0, 0, 0, 0, 0, 0, 0, 0],
        [0, 0, 0, 0, 0, 0, 0, 0, 0],
        [0, 0, 0, 0, 0, 0, 0, 0, 0],
        [0, 0, 0, 1, 0, 0, 0, 0, 1],
        [0, 0, 0, 1, 1, 0, 0, 0, 0],
        [0, 0, 0, 1, 0, 0, 0, 0, 0],
        [0, 0, 0, 0, 0, 0, 0, 0, 0],
        [0, 0, 0, 0, 0, 0, 0, 0, 0],
        [0, 0, 0, 0, 0, 0, 0, 0, 0]
      ]::double precision[][])
),
2, 1, 1,
ARRAY[
  [0, 0, 0, 0, 0, 0, 0, 0, 0],
  [0, 0, 0, 0, 0, 0, 0, 0, 0],
```

```

        [0, 0, 0, 0, 0, 0, 0, 0, 0],
        [1, 0, 0, 0, 0, 1, 0, 0, 0],
        [0, 0, 0, 0, 1, 1, 0, 0, 0],
        [0, 0, 0, 0, 0, 1, 0, 0, 0],
        [0, 0, 0, 0, 0, 0, 0, 0, 0],
        [0, 0, 0, 0, 0, 0, 0, 0, 0],
        [0, 0, 1, 0, 0, 0, 0, 0, 0]
    ]::double precision[][])
    ) AS rast
)
SELECT
    ST_AsText(ST_ConvexHull(rast)) AS hull,
    ST_AsText(ST_MinConvexHull(rast)) AS mhull,
    ST_AsText(ST_MinConvexHull(rast, 1)) AS mhull_1,
    ST_AsText(ST_MinConvexHull(rast, 2)) AS mhull_2
FROM foo
```

hull		mhull		mhull_1	↔
		mhull_2			
-----+-----					
POLYGON((0 0,9 0,9 -9,0 -9,0 0))		POLYGON((0 -3,9 -3,9 -9,0 -9,0 -3))		POLYGON((3 -3,9	↔
-3,9 -6,3 -6,3 -3))		POLYGON((0 -3,6 -3,6 -9,0 -9,0 -3))			

関連情報

[ST_Envelope](#), [ST_ConvexHull](#), [ST_ConvexHull](#), [ST_AsText](#)

11.15.6 ST_Polygon

ST_Polygon — NODATA 値でないピクセル値を持つピクセルの結合で形成されるマルチポリゴンジオメトリを返します。バンドを指定しない場合のデフォルトは 1 です。

Synopsis

geometry **ST_Polygon**(raster rast, integer band_num=1);

説明

Changed 3.3.0, 能率向上のため評価と修正が無効になりました。不正なジオメトリを返すことがあります。
Availability: 0.1.6 GDAL 1.7 以上が必要です。
Enhanced: 2.1.0 速度を改善し (完全に C 言語で記述しました)、確実に妥当なマルチポリゴンを返すようにしました。
Changed: 2.1.0 以前の版では、時々ポリゴンを返しましたが、常にマルチポリゴンを返すように変更しました。

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;
```

```
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;
```

```
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;
```

```
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)))
```

関連情報

[ST_Value](#), [ST_DumpAsPolygons](#)

11.15.7 ST_IntersectionFractions

ST_IntersectionFractions — ラスタ上の個々のセルのうち与えられたジオメトリで覆われたものの断片を計算します。

Synopsis

raster **ST_IntersectionFractions**(raster rast, geometry geom);

説明

ラスタ上の個々のセルのうち与えられたジオメトリで覆われている割合を計算します。一つ目の引数はラスタで、計算に使うジオメトリグリッドを定義します。範囲とセルサイズはこのラスタから読みます。二つ目の引数はジオメトリです。グリッドを覆っていて、個々のグリッドの値は、ジオメトリが覆うグリッドに基づいて設定されます。ポリゴンについては、個々のセルについて返される値は、ジオメトリに覆われた領域の割合です。ラインストリングについては、個々のセルから返される値はセルに含まれる長さです。

Availability: 3.6.0 GEOS 3.14 以上が必要です。

例

```
CREATE TABLE raster_proportions_rast (
  name text,
  rast raster
);

INSERT INTO raster_proportions_rast (name, rast) VALUES (
  '2x2 raster covering 0,0 to 10,10',
  ST_MakeEmptyRaster(2, 2, -- raster width/height in pixels
    0, 10, -- upper-left corner x/y coordinates
    5, -5, -- pixel width/height in ground units
    0, 0, -- skew x/y
    0 -- SRID
  ));

--
-- This rotated square polygon covers half of each cell in the
-- raster.
--
SELECT name, ST_DumpValues(
  ST_IntersectionFractions(
    rast,
    'POLYGON((5 0, 0 5, 5 10, 10 5, 5 0))'::geometry),
  1)
FROM raster_proportions_rast;
```

```
2x2 raster covering 0,0 to 10,10
-----
{{0.5,0.5},{0.5,0.5}}
```

関連情報

[ST_MakeEmptyRaster](#)

11.16 ラスタ演算子

11.16.1 &&

&& — A のバウンディングボックスが B のバウンディングボックスとインタセクトする場合に TRUE を返します。

Synopsis

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

説明

&& 演算子は、A のバウンディングボックスが B のバウンディングボックスにインタセクトする場合に **TRUE** を返します。



Note

これらのオペランドは、ラスタで使用できるインデックスを使用します。

Availability: 2.0.0

例

```
SELECT A.rid As a_rid, B.rid As b_rid, A.rast && B.rast As intersect
FROM dummy_rast AS A CROSS JOIN dummy_rast AS B LIMIT 3;
```

a_rid	b_rid	intersect
2	2	t
2	3	f
2	1	f

11.16.2 &<

&< — A のバウンディングボックスが B のバウンディングボックスをオーバラップするか、B のバウンディングボックスの左にある場合に **TRUE** を返します。

Synopsis

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

説明

&< 演算子は、A のバウンディングボックスが B のバウンディングボックスをオーバラップするか、B のバウンディングボックスの左にある場合に **TRUE** を返します。条件についてより詳細に言うと、B のバウンディングボックスをオーバラップするか B のバウンディングボックスの右に * ない * 場合です。



Note

これらのオペランドは、ラスタで使用できるインデックスを使用します。

例

```
SELECT A.rid As a_rid, B.rid As b_rid, A.rast &< B.rast As overleft
FROM dummy_rast AS A CROSS JOIN dummy_rast AS B;
```

a_rid	b_rid	overleft
2	2	t
2	3	f
2	1	f
3	2	t
3	3	t
3	1	f
1	2	t
1	3	t
1	1	t

11.16.3 &>

&> — A のバウンディングボックスが B のバウンディングボックスをオーバラップするか、B のバウンディングボックスの右にある場合に **TRUE** を返します。

Synopsis

```
boolean &>( raster A , raster B );
```

説明

&> 演算子は、A のバウンディングボックスが B のバウンディングボックスをオーバラップするか、B のバウンディングボックスの右にある場合に **TRUE** を返します。条件についてより詳細に言うと、B のバウンディングボックスをオーバラップするか B のバウンディングボックスの左に * ない * 場合です。



Note

これらのオペランドは、ジオメトリで使えるインデックスを使用します。

例

```
SELECT A.rid As a_rid, B.rid As b_rid, A.rast &
> B.rast As overright
FROM dummy_rast AS A CROSS JOIN dummy_rast AS B;
```

a_rid	b_rid	overright
2	2	t
2	3	t
2	1	t
3	2	f
3	3	t
3	1	f
1	2	f

1	3 t
1	1 t

11.16.4 =

= — A のバウンディングボックスが B のバウンディングボックスと同じ場合に TRUE を返します。倍精度浮動小数点数のバウンディングボックスを使います。

Synopsis

```
boolean =( raster A , raster B );
```

説明

= 演算子は、A のバウンディングボックスが B のバウンディングボックスと同じ場合に TRUE を返します。PostgreSQL は内部でラスタの順序決定や比較 (GROUP BY 節や ORDER BY 節など) を行うためにラスタ用として定義している =, <, > 演算子を使用します。



Caution

このオペランドは、ラスタで利用できるインデックスを使いません。代わりに **~=** を使います。この演算子はラスタカラムの GROUP BY に使うのがほとんどです。

Availability: 2.1.0

関連情報

~=

11.16.5 @

@ — A のバウンディングボックスが B のバウンディングボックスに含まれる場合に TRUE を返します。倍精度浮動小数点数のバウンディングボックスを使います。

Synopsis

```
boolean @( raster A , raster B );
boolean @( geometry A , raster B );
boolean @( raster B , geometry A );
```

説明

@ は、ラスタまたはジオメトリである A のバウンディングボックスが、ラスタまたはジオメトリである B のバウンディングボックスに含まれる場合に **TRUE** を返します。



Note

このオペランドは、ラスタのインデックスを使用します。

Availability: 2.0.0 raster @ raster, raster @ geometry が導入されました。

Availability: 2.0.5 geometry @ raster が導入されました。

関連情報

~

11.16.6 ~=

~= — A のバウンディングボックスが B のバウンディングボックスと同じ場合に **TRUE** を返します。

Synopsis

boolean ~= (raster A , raster B);

説明

~= 演算子は、A のバウンディングボックスが B のバウンディングボックスと同じ場合に **TRUE** を返します。



Note

このオペランドは、ラスタで利用できるインデックスを使用します。

Availability: 2.0.0

例

大変便利な利用局面として、単一バンドを持つラスタ集合が二つあって、これらは同じデータなのに異なる主題を表現する場合で、これらを取って複数バンドのラスタを生成する局面です。

```
SELECT ST_AddBand(prec.rast, alt.rast) As new_rast
FROM prec INNER JOIN alt ON (prec.rast ~= alt.rast);
```

関連情報

ST_AddBand, **=**

11.16.7 ~

~ — A のバウンディングボックスが B のバウンディングボックスを含む場合に TRUE を返します。倍精度浮動小数点数のバウンディングボックスを使います。

Synopsis

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

説明

~ 演算子は、ラスタまたはジオメトリである A のバウンディングボックスがラスタまたはジオメトリである B のバウンディングボックスを含む場合に TRUE を返します。



Note

このオペランドは、ラスタのインデックスを使用します。

Availability: 2.0.0

関連情報

@

11.17 ラスタとラスタバンドの空間関係関数

11.17.1 ST_Contains

ST_Contains — rastA の外に rastB の点が無く、rastA の内部に rastB の内部の点が一つ以上ある場合に TRUE を返します。

Synopsis

```
boolean ST_Contains( raster rastA , integer nbandA , raster rastB , integer nbandB );  
boolean ST_Contains( raster rastA , raster rastB );
```

説明

rastA の外に rastB の点が無く、rastA の内部に rastB の内部の点が一つ以上ある場合に限って、rastA は rastB を包含しています。バンド番号が指定されていないか NULL に指定されている場合には、ラスタの凸包のみを考慮してテストします。バンド番号が指定されている場合には、値を持つ (NODATA でない) ピクセルについてテストします。

**Note**

この関数はラスタで利用できるインデックスを使用します。

**Note**

ラスタとジオメトリの空間関係をテストするには、`ST_Contains(ST_Polygon(raster), geometry)` または `ST_Contains(geometry, ST_Polygon(raster))` というふうに、ラスタに `ST_Polygon` を使います。

**Note**

`ST_Contains()` は `ST_Within()` の逆です。`ST_Contains(rastA, rastB)` は `ST_Within(rastB, rastA)` を示します。

Availability: 2.1.0

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	
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

関連情報

[ST_Intersects](#), [ST_Within](#)

11.17.2 ST_ContainsProperly

`ST_ContainsProperly` — `rastB` が `rastA` の内部でインタセクトし、かつ `rastA` の境界とも外部ともインタセクトしない場合に `TRUE` を返します。

Synopsis

```
boolean ST_ContainsProperly( raster rastA , integer nbandA , raster rastB , integer nbandB );
boolean ST_ContainsProperly( raster rastA , raster rastB );
```

説明

`rastB` が `rastA` の内部でインタセクトし、かつ `rastA` の境界とも外部ともインタセクトしない場合には、`rastA` は `rastB` に対して `ContainsProperly` (確実に包含している) です。バンド番号が指定されていないか `NULL` に指定されている場合には、ラスタの凸包のみを考慮してテストします。バンド番号が指定されている場合には、値を持つ (NODATA でない) ピクセルについてテストします。

`rastA` は自身には `ContainsProperly` ではありませんが、`Contains` です。



Note

この関数はラスタで利用できるインデックスを使用します。



Note

ラスタとジオメトリの空間関係をテストするには、`ST_ContainsProperly(ST_Polygon(raster), geometry)` または `ST_ContainsProperly(geometry, ST_Polygon(raster))` というふうに、ラスタに `ST_Polygon` を使います。

Availability: 2.1.0

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
```

関連情報

[ST_Intersects](#), [ST_Contains](#)

11.17.3 ST_Covers

`ST_Covers` — `rastB` が `rastA` の外部に点を持たない場合に `TRUE` を返します。

Synopsis

```
boolean ST_Covers( raster rastA , integer nbandA , raster rastB , integer nbandB );
boolean ST_Covers( raster rastA , raster rastB );
```

説明

`rastA` の外部に `rastB` の点が無い場合には、`rastA` は `rastB` に対して **Covers** です。バンド番号が指定されていないか `NULL` に指定されている場合には、ラスタの凸包のみを考慮してテストします。バンド番号が指定されている場合には、値を持つ (`NODATA` でない) ピクセルについてテストします。

**Note**

この関数はラスタで利用できるインデックスを使用します。

**Note**

ラスタとジオメトリの空間関係をテストするには、`ST_Covers(ST_Polygon(raster), geometry)` または `ST_Covers(geometry, ST_Polygon(raster))` というふうに、ラスタに `ST_Polygon` を使います。

Availability: 2.1.0

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
```

関連情報

[ST_Intersects](#), [ST_CoveredBy](#)

11.17.4 ST_CoveredBy

`ST_CoveredBy` — `rastA` が `rastB` の外部に点を持たない場合に `TRUE` を返します。

Synopsis

```
boolean ST_CoveredBy( raster rastA , integer nbandA , raster rastB , integer nbandB );
boolean ST_CoveredBy( raster rastA , raster rastB );
```

説明

`rastA` が `rastB` の外部に点を持たない場合には、`rastA` は `rastB` に対して **CoveredBy** です。バンド番号が指定されていないか `NULL` に指定されている場合には、ラスタの凸包のみを考慮してテストします。バンド番号が指定されている場合には、値を持つ (`NODATA` でない) ピクセルについてテストします。

**Note**

この関数はラスタで利用できるインデックスを使用します。

**Note**

ラスタとジオメトリの空間関係をテストするには、`ST_CoveredBy(ST_Polygon(raster), geometry)` または `ST_CoveredBy(geometry, ST_Polygon(raster))` というふうに、ラスタに `ST_Polygon` を使います。

Availability: 2.1.0

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
```

関連情報

[ST_Intersects](#), [ST_Covers](#)

11.17.5 ST_Disjoint

`ST_Disjoint` — `rastA` が `rastB` とインタセクトしない場合に `TRUE` を返します。

Synopsis

```
boolean ST_Disjoint( raster rastA , integer nbandA , raster rastB , integer nbandB );
boolean ST_Disjoint( raster rastA , raster rastB );
```

説明

`rastA` が `rastB` とインタセクトしない場合には、`rastA` と `rastB` は `Disjoint` です。バンド番号が指定されていないか `NULL` に指定されている場合には、ラスタの凸包のみを考慮してテストします。バンド番号が指定されている場合には、値を持つ (`NODATA` でない) ピクセルについてテストします。

**Note**

この関数はインデックスを * 使いません *。

**Note**

ラスタとジオメトリの空間関係をテストするには、`ST_Disjoint(ST_Polygon(raster), geometry)` というふうに、ラスタに `ST_Polygon` を使います。

Availability: 2.1.0

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

関連情報

[ST_Intersects](#)

11.17.6 ST_Intersects

`ST_Intersects` — `rastA` が `rastB` とインタセクトする場合に `TRUE` を返します。

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 );
```


説明

`rastA` が `rastB` とインタセクトする場合に `TRUE` を返します。バンド番号が指定されていないか `NULL` に指定されている場合には、ラスタの凸包のみを考慮してテストします。バンド番号が指定されている場合には、値を持つ (`NODATA` でない) ピクセルについてテストします。



Note

この関数はラスタで利用できるインデックスを使用します。

Enhanced: 2.0.0 ラスタ/ラスタのインタセクト対応が導入されました。



Warning

Changed: 2.1.0 `ST_Intersects(raster, geometry)` 形式の挙動が `ST_Intersects(geometry, raster)` とあうように変更されました。

例

Different bands of same raster.

```
SELECT ST_Intersects(rast, 2, rast, 3) FROM dummy_rast WHERE rid = 2;
```

t

関連情報

[ST_Intersection](#), [ST_Disjoint](#)

11.17.7 ST_Overlaps

`ST_Overlaps` — `rastA` と `rastB` がインタセクトして、かつ一方がもう一方に完全には包含されない場合には `TRUE` を返します。

Synopsis

```
boolean ST_Overlaps( raster rastA , integer nbandA , raster rastB , integer nbandB );  
boolean ST_Overlaps( raster rastA , raster rastB );
```

説明

`rastA` が `rastB` をオーバーラップする場合に `TRUE` を返します。`rastA` と `rastB` がインタセクトして、かつ一方がもう一方に完全には包含されない場合にはいう意味です。バンド番号が指定されていないか `NULL` に指定されている場合には、ラスタの凸包のみを考慮してテストします。バンド番号が指定されている場合には、値を持つ (`NODATA` でない) ピクセルについてテストします。

**Note**

この関数はラスタで利用できるインデックスを使用します。

**Note**

ラスタとジオメトリの空間関係をテストするには、`ST_Overlaps(ST_Polygon(raster), geometry)` というふうに、ラスタに `ST_Polygon` を使います。

Availability: 2.1.0

例

Comparing different bands of same raster.

```
SELECT ST_Overlaps(rast, 1, rast, 2) FROM dummy_rast WHERE rid = 2;
```

f

関連情報

[ST_Intersects](#)

11.17.8 ST_Touches

`ST_Touches` — `rastA` と `rastB` が少なくとも一つの共通の点を持ち、かつ二つのラスタの内部同士がインタセクトしない場合に `TRUE` を返します。

Synopsis

```
boolean ST_Touches( raster rastA , integer nbandA , raster rastB , integer nbandB );  
boolean ST_Touches( raster rastA , raster rastB );
```

説明

`rastA` が `rastB` に接触する場合に `TRUE` を返します。`rastA` と `rastB` が少なくとも一つの共通の点を持ち、かつ二つのラスタの内部同士がインタセクトしないという意味です。バンド番号が指定されていないか `NULL` に指定されている場合には、ラスタの凸包のみを考慮してテストします。バンド番号が指定されている場合には、値を持つ (`NODATA` でない) ピクセルについてテストします。

**Note**

この関数はラスタで利用できるインデックスを使用します。

**Note**

ラスタとジオメトリの空間関係をテストするには、`ST_Touches(ST_Polygon(raster), geometry)` というふうに、ラスタに `ST_Polygon` を使います。

Availability: 2.1.0

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

関連情報

[ST_Intersects](#)

11.17.9 ST_SameAlignment

ST_SameAlignment — ラスタが同じスキュー、スケール、空間参照系、オフセットを持つ (ピクセルが分割されることなく同じグリッドに置かれている) 場合に TRUE を返し、そうでない場合は問題を詳述する通知とともに FALSE を返します。

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 );
```

説明

非集計版 (一つ目と二つ目の形式): 二つのラスタ (直接与えたラスタでも左上隅、スケール、スキュー、SRID を使って作ったものでも) が同じスキュー、スケール、空間参照系、を持ち、少なくとも 4 隅の一つがもう一方のラスタのグリッド隅に落ちる場合に TRUE を返します。そうでない場合には、問題に言及した警告を出します。

集約関数版 (三つ目の形式): ラスタ集合から、全てのラスタが同じアラインメントを持つ場合に TRUE を返します。ST_SameAlignment() 関数は PostgreSQL の用語で言うところの「集約」関数です。SUM() や AVG() といった複数のデータ行での操作を意味します。

Availability: 2.0.0

Enhanced: 2.1.0 集約関数版の追加

例: ラスタ

```
SELECT ST_SameAlignment(ST_MakeEmptyRaster(1, 1, 0, 0, 1, 1, 0, 0),
    ST_MakeEmptyRaster(1, 1, 0, 0, 1, 1, 0, 0)
) as sm;
```

t

```
SELECT ST_SameAlignment(A.rast,b.rast)
FROM dummy_rast AS A CROSS JOIN dummy_rast AS B;
```

```
NOTICE: The two rasters provided have different SRIDs
NOTICE: The two rasters provided have different SRIDs
t
f
f
f
```

関連情報

Section [10.1](#), [ST_NotSameAlignmentReason](#), [ST_MakeEmptyRaster](#)

11.17.10 ST_NotSameAlignmentReason

ST_NotSameAlignmentReason — ラスタが同じアラインメントを持つかどうか、また、持たない場合にはその理由を示す文字列を返します。

Synopsis

text **ST_NotSameAlignmentReason**(raster rastA, raster rastB);

説明

ラスタが同じアラインメントを持つかどうか、また、持たない場合にはその理由を示す文字列を返します。



Note

ラスタが同じアラインメントを持たない理由が複数ある場合には、一つだけ（最初のテストで失敗した）理由を返します。

Availability: 2.1.0

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_notsamealignmentreason
f	The rasters have different scales on the X axis

(1 row)

関連情報

Section 10.1, [ST_SameAlignment](#)**11.17.11 ST_Within**

ST_Within — `rastA` が `rastB` の外部に点を持たず、`rastA` の内部の少なくとも一つの点が `rastB` の内部にある場合に TRUE を返します。

Synopsis

```
boolean ST_Within( raster rastA , integer nbandA , raster rastB , integer nbandB );
boolean ST_Within( raster rastA , raster rastB );
```

説明

`rastA` が `rastB` の外部に点を持たず、`rastA` の内部の少なくとも一つの点が `rastB` の内部にある場合に `rastA` は `rastB` に対して **Within** です。バンド番号が指定されていないか NULL に指定されている場合には、ラスタの凸包のみを考慮してテストします。バンド番号が指定されている場合には、値を持つ (NODATA でない) ピクセルについてテストします。

**Note**

このオペランドは、ラスタで利用できるインデックスを使用します。

**Note**

ラスタとジオメトリの空間関係をテストするには、`ST_Within(ST_Polygon(raster), geometry)` または `ST_Within(geometry, ST_Polygon(raster))` というふうに、ラスタに `ST_Polygon` を使います。

**Note**

`ST_Within()` は `ST_Contains()` の逆です。`ST_Within(rastA, rastB)` は `ST_Contains(rastB, rastA)` を示します。

Availability: 2.1.0

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

関連情報

[ST_Intersects](#), [ST_Contains](#), [ST_DWithin](#), [ST_DFullyWithin](#)

11.17.12 ST_DWithin

`ST_DWithin` — `rastA` と `rastB` が指定した距離内にある場合に `TRUE` を返します。

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 );
```

説明

`rastA` と `rastB` が指定した距離内にある場合に `TRUE` を返します。バンド番号が指定されていないか `NULL` に指定されている場合には、ラスタの凸包のみを考慮してテストします。バンド番号が指定されている場合には、値を持つ (NODATA でない) ピクセルについてテストします。

`distance` はラスタの空間参照系で定義される単位です。この関数が意味あるものにするには、元のラスタが両方とも同じ空間参照系である、つまり同じ `SRID` を持たなければなりません。



Note

このオペランドは、ラスタで利用できるインデックスを使用します。



Note

ラスタとジオメトリの空間関係をテストするには、`ST_DWithin(ST_Polygon(raster), geometry)` というふうに、ラスタに `ST_Polygon` を使います。

Availability: 2.1.0

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

関連情報

[ST_Within](#), [ST_DFullyWithin](#)

11.17.13 ST_DFullyWithin

ST_DFullyWithin — rastA と rastB が指定した距離内に完全に収まる場合に TRUE を返します。

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 );
```

説明

rastA と rastB が指定した距離内に完全に収まる場合に TRUE を返します。バンド番号が指定されていないか NULL に指定されている場合には、ラスタの凸包のみを考慮してテストします。バンド番号が指定されている場合には、値を持つ (NODATA でない) ピクセルについてテストします。

distance はラスタの空間参照系で定義される単位です。この関数が意味あるものにするには、元のラスタが両方とも同じ空間参照系である、つまり同じ SRID を持たなければなりません。



Note

このオペランドは、ラスタで使えるインデックスを使用します。



Note

ラスタとジオメトリの空間関係をテストするには、ST_DFullyWithin(ST_Polygon(raster), geometry) というふうに、ラスタに ST_Polygon を使います。

Availability: 2.1.0

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
```

関連情報

[ST_Within](#), [ST_DWithin](#)

11.18 ラスタに関する技法

11.18.1 データベース外ラスタ

11.18.1.1 多数のファイルを持つディレクトリ

GDAL はファイルを開く時に、そのファイルのディレクトリを熱心にスキャンして、他のファイルのカタログを構築します。このディレクトリに多数のファイル (千とか万とか) あるとします。一つのファイルを開くと動作が非常に遅くなります (特に NFS のようなネットワークドライブの場合)。

この振る舞いを制御するために、GDAL には **GDAL_DISABLE_READDIR_ON_OPEN** という環境変数があります。GDAL_DISABLE_READDIR_ON_OPEN を TRUE に設定すると、ディレクトリのスキャンを無効にします。

Ubuntu では (Ubuntu 用 PostgreSQL のパッケージを使っていると仮定します)、GDAL_DISABLE_READDIR_ON_OPEN が `/etc/postgresql/POSTGRESQL_VERSION/CLUSTER_NAME/environment` 内で設定できます (POSTGRESQL は 9.6 等の PostgreSQL のバージョンで、CLUSTER_NAME は maindb 等のクラスタ名です)。PostGIS 環境変数もここで同じく設定できます。

```
# 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 開くことができるファイルの最大数

Linux と PostgreSQL が許している、開くことができるファイルの最大数は、通常は増加していません (通常、プロセスあたり 1024 ファイルです)。システムは人間のユーザが使用するという仮定に立っているからです。データベース外ラスタは、一つの妥当なクエリで簡単に制限超過させることができます (例えば、10 年分のラスタからなるデータセットで、個々のラスタは日別最低気温、最大気温を持っていて、データセット内の最大値と最小値を得たい場合などです)。

最も簡単な変更方法は、PostgreSQL 設定 **max_files_per_process** です。デフォルトとして 1000 が設定されていますが、データベース外ラスタとしては非常に低い値です。安全な開始値は 65536 でしょう。ただし、実際には、これはデータベースとデータベースに対して実行されるクエリに依存します。サーバ開始時のみ、かつ PostgreSQL コンフィギュレーションファイル (例: Ubuntu 環境では `/etc/postgresql/POSTGRESQL_VERSION/CLUSTER_NAME/postgresql.conf`) 内のみで設定できます。

```
...
# - Kernel Resource Usage -

max_files_per_process = 65536          # min 25
                                         # (change requires restart)
...
```

主な変更は Linux カーネルのファイルを開く制限です。次の通り、二つに分かれます:

- システム全体で開くことができるファイルの最大数
- プロセスごとに開くことができるファイルの最大数

11.18.1.2.1 システム全体で開くことができるファイルの最大数

次の例で、システム全体の、現在の開くことができるファイルの最大値を調べることができます:

```
$ sysctl -a | grep fs.file-max
fs.file-max = 131072
```

返された値が十分には大きくない場合には、次に示す例に従って、`/etc/sysctl.d/`にファイルを追加します。

```
$ 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 プロセスごとの開けるファイルの最大数

PostgreSQL のサーバプロセスごとに開けるファイルの最大数を増やす必要があります。

現在の PostgreSQL サービスのプロセスは開くことができるファイルの最大数を使っていて、次の例のようになっています (PostgreSQL が実行されていることを確認して下さい)。

```
$ 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

上の例では、プロセスに対する開くことができるファイルの制限が 31718 になっています。どのプロセスがどうするかは問題ではありません。関心を持っている応答は *Max open files* です。

Max open files の *Soft Limit* と *Hard Limit* を PostgreSQL の設定で指定した `max_files_per_process` よりも多くなるようにしたいと思っています。この例では、`max_files_per_process` を 65536 にしています。

Ubuntu (かつ、Ubuntu 用 PostgreSQL パッケージを使用しているものとします) では、*Soft Limit* と *Hard Limit* の変更については、`/etc/init.d/postgresql` (SysV) または `/lib/systemd/system/postgresql*.service` (systemd) を編集するのが簡単な方法です。

まず、SysV Ubuntu を扱います。 **`ulimit -H -n 262144`** と **`ulimit -n 131072`** を `/etc/init.d/postgresql` に追加します。

```
...
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)
    ...
```

Ubuntu の systemd を扱います。 **`LimitNOFILE=131072`** を `/lib/systemd/system/postgresql*.service` の各ファイルの **[Service]** セクション内に記述します。

```
...
[Service]

LimitNOFILE=131072

...

[Install]
WantedBy=multi-user.target
...
```

必要なシステムの変更を行った後、デーモンのリロードを必ず行ってください。

```
systemctl daemon-reload
```

Chapter 12

PostGIS 追加機能

本章では、PostGIS のソースアーカイブとソースレポジトリの `extras` フォルダにある機能を記述します。これらは必ず PostGIS バイナリ版に同梱されているものではありませんが、通常は実行可能な PL/pgSQL ベースのものまたは標準的なシェルスクリプトです。

Chapter 13

PostGIS 関数索引

13.1 PostGIS 集約関数

次に示す関数は、PostGIS が提供する空間集約関数です。sum や average といった他の SQL 集約関数のように使います。

- **CG_3DUnion** - postgis_sfcgal を使用して 3 次元結合を実行します。
- **ST_3DExtent** - ジオメトリの 3 次元バウンディングボックスを返す集約関数です。
- **ST_3DUnion** - 3 次元の結合を計算します。
- **ST_AsFlatGeobuf** - 行の集合の FlatGeobuf 表現を返します。
- **ST_AsGeobuf** - 行集合の Geobuf 表現を返します。
- **ST_AsMVT** - 行集合の MVT 表現を返す集約関数です。
- **ST_ClusterDBSCAN** - 入力ジオメトリごとに DBSCAN アルゴリズムを使ってクラスタ番号を返すウィンドウ関数です。
- **ST_ClusterIntersecting** - 入力ジオメトリを接続関係にある集合にクラスタリングする集約関数です。
- **ST_ClusterIntersectingWin** - 入力ジオメトリごとに接続された集合にクラスタリングを行い、クラスタ ID を返すウィンドウ関数です。
- **ST_ClusterKMeans** - 入力ジオメトリごとに k 平均法アルゴリズムを使ってクラスタ番号を返すウィンドウ関数です。
- **ST_ClusterRelateWin** - ジオメトリが接続するかどうかを判定する空間関係パターンを使って、入力ジオメトリを接続されたジオメトリ集合にクラスタリングして、クラスター識別子を返すウィンドウ関数です。
- **ST_ClusterWithin** - 分離距離でジオメトリのクラスタリングを行う集約関数です。
- **ST_ClusterWithinWin** - 入力ジオメトリごとに分離距離を使ったクラスタリングを行い、クラスタ ID を返すウィンドウ関数です。
- **ST_Collect** - ジオメトリの集合からジオメトリコレクションまたはマルチ系ジオメトリを生成します。
- **ST_CoverageClean** - きれいな (エッジが合致し、オーバーラップがなく、隙間がない) ポリゴンカバレッジを、きれいでない入力から計算します。
- **ST_CoverageInvalidEdges** - ポリゴンが妥当なカバレッジの形成に失敗する位置を検索するウィンドウ関数。
- **ST_CoverageSimplify** - ポリゴンカバレッジのエッジを単純化するウィンドウ関数。

- **ST_CoverageUnion** - 共有しているエッジを除去することでカバレッジを形成するポリゴンの集合の結合を計算します。
- **ST_Extent** - ジオメトリのバウンディングボックスを返す集約関数です。
- **ST_MakeLine** - POINT、MULTIPOINT、LINESTRING から LINESTRING を生成します。
- **ST_MemUnion** - ジオメトリを結合する集約関数で、メモリを効率的に使いますが処理時間のかかるものです。
- **ST_MinimumSpanningTree** - 最小全域木アルゴリズムを使って入力ジオメトリを接続された木にクラスタリングを行い、入力ジオメトリごとに木 ID を返すウィンドウ関数です。
- **ST_Polygonize** - ジオメトリ集合のラインワークから形成されるポリゴンのコレクションを計算します。
- **ST_SameAlignment** - ラスタが同じスキュー、スケール、空間参照系、オフセットを持つ (ピクセルが分割されることなく同じグリッドに置かれている) 場合に TRUE を返し、そうでない場合は問題を詳述する通知とともに FALSE を返します。
- **ST_Union** - 入力ジオメトリのポイント集合の結合を表現するジオメトリを返します。
- **ST_Union** - ラスタタイルの集合を結合して 1 以上のバンドからなる単一ラスタを返します。
- **TopoElementArray_Agg** - element_id とタイプの配列 (topoelements) からなる topoelementarray を返します。

13.2 PostGIS ウィンドウ関数

次に示す関数は、PostGIS が提供する空間ウィンドウ関数です。row_number(), lead(), lag() 等のような他の SQL ウィンドウ関数のように使います。全ての関数で、SQL の OVER() 句が必要です。

- **ST_ClusterDBSCAN** - 入力ジオメトリごとに DBSCAN アルゴリズムを使ってクラスタ番号を返すウィンドウ関数です。
- **ST_ClusterIntersectingWin** - 入力ジオメトリごとに接続された集合にクラスタリングを行い、クラスタ ID を返すウィンドウ関数です。
- **ST_ClusterKMeans** - 入力ジオメトリごとに k 平均法アルゴリズムを使ってクラスタ番号を返すウィンドウ関数です。
- **ST_ClusterRelateWin** - ジオメトリが接続するかどうかを判定する空間関係パターンを使って、入力ジオメトリを接続されたジオメトリ集合にクラスタリングして、クラスター識別子を返すウィンドウ関数です。
- **ST_ClusterWithinWin** - 入力ジオメトリごとに分離距離を使ったクラスタリングを行い、クラスタ ID を返すウィンドウ関数です。
- **ST_CoverageClean** - きれいな (エッジが合致し、オーバーラップがなく、隙間がない) ポリゴンカバレッジを、きれいでない入力から計算します。
- **ST_CoverageInvalidEdges** - ポリゴンが妥当なカバレッジの形成に失敗する位置を検索するウィンドウ関数。
- **ST_CoverageSimplify** - ポリゴンカバレッジのエッジを単純化するウィンドウ関数。
- **ST_MinimumSpanningTree** - 最小全域木アルゴリズムを使って入力ジオメトリを接続された木にクラスタリングを行い、入力ジオメトリごとに木 ID を返すウィンドウ関数です。

13.3 PostGIS SQL-MM 準拠関数

次に示す関数は、SQL/MM 3 標準に準拠した PostGIS 関数です。

- **CG_3DArea** - 3 次元の面ジオメトリの面積を計算します。立体の場合は 0 を返します。
- **CG_3DDifference** - 3 次元の差分を計算します。
- **CG_3DIntersection** - 3 次元のインタセクトした (共有する) 部分を計算します。
- **CG_3DUnion** - postgis_sfcgal を使用して 3 次元結合を実行します。
- **CG_Volume** - Computes the volume of a 3D geometry. Closed surfaces can have non-zero volume.
- **ST_3DArea** - 3 次元の面ジオメトリの面積を計算します。立体の場合は 0 を返します。
- **ST_3DDWithin** - 二つの 3 次元ジオメトリが与えられた 3 次元距離内にあるかどうかをテストします。
- **ST_3DDifference** - 3 次元の差分を計算します。
- **ST_3DDistance** - 投影座標系の単位で、二つのジオメトリ間の 3 次元デカルト距離の最小値を返します (空間参照系に基づきます)。
- **ST_3DIntersection** - 3 次元のインタセクトした (共有する) 部分を計算します。
- **ST_3DIntersects** - 二つのジオメトリが 3 次元空間において空間的にインタセクトするかどうかをテストします。ポイント、ラインストリング、ポリゴン、多面体サーフェス (面) についてのみ動作します。
- **ST_3DLength** - 線ジオメトリの 3 次元長を返します。
- **ST_3DPerimeter** - ポリゴンジオメトリの 3 次元周長を返します。
- **ST_3DUnion** - 3 次元の結合を計算します。
- **ST_AddEdgeModFace** - 新しいエッジを追加します。新しいエッジがフェイスを分割する場合には、もとのフェイスを編集し、一つのフェイスを追加します。
- **ST_AddEdgeNewFaces** - 新しいエッジを追加します。新しいエッジがフェイスを分割する場合には、もとのフェイスを削除して、分割した二つのフェイスに置き換えます。
- **ST_AddIsoEdge** - anode と anothernode で指定される二つの既存孤立ノードを接続するトポロジに、ジオメトリ alinestring で定義される孤立エッジを追加し、新しいエッジの識別番号を返します。
- **ST_AddIsoNode** - フェイスに孤立ノードを追加し、新しいノードの識別番号を返します。フェイスが NULL の場合でもノードは生成されます。
- **ST_Area** - ポリゴンジオメトリの面積を返します。
- **ST_AsBinary** - ジオメトリ/ジオグラフィの、SRID メタデータを持たない OGC/ISO Well-Known バイナリ (WKB) 表現を返します。
- **ST_AsGML** - GML 第 2 版または第 3 版としてジオメトリを返します。
- **ST_AsText** - ジオメトリ/ジオグラフィの SRID メタデータのない Well-Known Text (WKT) 表現を返します。
- **ST_Boundary** - ジオメトリの境界を返します。
- **ST_Buffer** - あるジオメトリからの距離が指定された距離以下となる点全ての集合となるジオメトリを返します。
- **ST_Centroid** - ジオメトリの幾何学的重心を返します。
- **ST_ChangeEdgeGeom** - トポロジ構造に影響を与えることなくエッジの形状を変更します。

- **ST_Contains** - B の全てのポイントが A 内にあり、かつ、双方の内部に共有点が存在するかどうかをテストします。
 - **ST_ConvexHull** - ジオメトリの凸包を計算します。
 - **ST_CoordDim** - ジオメトリの座標次元を返します。
 - **ST_CreateTopoGeo** - 空のトポロジにジオメトリのコレクションを追加し、成否を示すメッセージを返します。
 - **ST_Crosses** - 二つのジオメトリが内部に共有ポイントを持ち、かつそれだけにならないようになっているかテストします。
 - **ST_CurveN** - 複合曲線の N 番目の曲線ジオメトリを返します。
 - **ST_CurveToLine** - 曲線を含むジオメトリを線ジオメトリに変換します。
 - **ST_Difference** - ジオメトリ B とインタセクトしていないジオメトリ A の一部を表現するジオメトリを計算します。
 - **ST_Dimension** - ST_Geometry 値の座標次元を返します。
 - **ST_Disjoint** - 二つのジオメトリが共有点を持たないようになっているかテストします。
 - **ST_Distance** - 二つのジオメトリ値またはジオグラフィ値間の距離を返します。
 - **ST_EndPoint** - Returns the last point of a LineString, CircularLineString, or NURBSCurve.
 - **ST_Envelope** - ジオメトリのバウンディングボックスを表現するジオメトリを返します。
 - **ST_Equals** - 二つのジオメトリが同じ点集合になっているかテストします。
 - **ST_ExteriorRing** - ポリゴンの外環を表現するラインストリングを返します。
 - **ST_GMLToSQL** - GML 表現から指定した ST_Geometry 値を返します。これは ST_GeomFromGML の別名です。
 - **ST_GeomCollFromText** - WKT 表現と与えられた SRID からジオメトリのコレクションを生成します。SRID が与えられていない場合は 0 とします。
 - **ST_GeomFromText** - Well-Known Text 表現 (WKT) から指定した ST_Geometry を返します。
 - **ST_GeomFromWKB** - Well-Known Binary ジオメトリ表現 (WKB) と任意パラメーターの SRID からジオメトリインスタンスを生成します。
 - **ST_GeometryFromText** - Well-Known Text 表現 (WKT) から指定した ST_Geometry 値を返します。これは ST_GeomFromText の別名です。
 - **ST_GeometryN** - ジオメトリコレクションの要素を一つ返します。
 - **ST_GeometryType** - ジオメトリの SQL-MM 型を文字列で返します。
 - **ST_GetFaceEdges** - 順序番号を含む、aface の境界となる、整列したエッジの集合を返します。
 - **ST_GetFaceGeometry** - 指定されたトポロジの中の、フェイス識別番号で指定されたポリゴンを返します。
 - **ST_InitTopoGeo** - 新しいトポロジスキーマを生成し、topology.topology テーブルに登録します。
 - **ST_InteriorRingN** - ポリゴンの N 番目の内環 (穴) を返します。
 - **ST_Intersection** - ジオメトリ A とジオメトリ B の共通部分を表現するジオメトリを返します。
 - **ST_Intersects** - 二つのジオメトリがインタセクトしている (少なくとも一つの共有点がある) かどうかテストします。
 - **ST_IsClosed** - ラインストリングの始点と終点が一致しているかをテストします。多面体サーフェスについては閉じているか (立体であるか) をテストします。
-

- **ST_IsEmpty** - ジオメトリが空かをテストします。
 - **ST_IsRing** - ラインストリングが閉じていてかつ単純であるかをテストします。
 - **ST_IsSimple** - ジオメトリが自己インタセクトまたは自己接触となるポイントが無いかをテストします。
 - **ST_IsValid** - ジオメトリが 2 次元で整形されているかのテスト。
 - **ST_Length** - 線系ジオメトリの 2 次元長を返します。
 - **ST_LineFromText** - WKT 表現と与えられた SRID からジオメトリを生成します。SRID が与えられていない場合は 0 (不明) となります。
 - **ST_LineFromWKB** - WKB 表現と与えられた SRID から LINESTRING を生成します。
 - **ST_LinestringFromWKB** - WKB 表現と与えられた SRID からジオメトリを生成します。
 - **ST_LocateAlong** - M 値に一致するジオメトリ上のポイントを返します。
 - **ST_LocateBetween** - M 値の範囲に合致する部分ジオメトリを返します。
 - **ST_M** - ポイントの M 値を返します。
 - **ST_MLineFromText** - WKT 表現から指定した ST_MultiLineString 値を返します。
 - **ST_MPointFromText** - Well-Known Text (WKT) 表現と与えられた SRID からジオメトリを生成します。SRID を与えない場合は 0 (不明) となります。
 - **ST_MPolyFromText** - Well-Known Text (WKT) 表現と与えられた SRID からマルチポリゴン生成します。SRID が与えられていない場合は 0 (不明) となります。
 - **ST_ModEdgeHeal** - 二つのエッジについて、接続しているノードを削除して修復します。1 番目のエッジを編集して、2 番目のエッジを削除します。削除されたノードの識別番号を返します。
 - **ST_ModEdgeSplit** - 既存のエッジに沿って新しいノードを生成してエッジを分割します。もとのエッジは変更され、新しいエッジが一つ追加されます。
 - **ST_MoveIsoNode** - トポロジ内の孤立ノードを別の位置に移動させます。新しい apoint ジオメトリがノードとして存在しているなら、エラーが投げられます。移動に関する説明を返します。
 - **ST_NewEdgeHeal** - 二つのエッジについて、接続しているノードを削除して修復します。両方のエッジを削除し、1 番目のエッジと同じ方向のエッジに置き換えます。
 - **ST_NewEdgesSplit** - 新しいノードを既存のエッジに沿って作成して、エッジを分割します。もとのエッジは削除され、二つのエッジに置き換えられます。二つの新しいエッジに接続する新しいノードの識別番号を返します。
 - **ST_NumCurves** - 複合曲線内の曲線の数を返します。
 - **ST_NumGeometries** - ジオメトリコレクションの要素数を返します。
 - **ST_NumInteriorRings** - ポリゴンの内環 (穴) の数を返します。
 - **ST_NumPatches** - 多面体サーフェスのフェイス数を返します。多面体でないジオメトリの場合には NULL を返します。
 - **ST_NumPoints** - ラインストリングまたは曲線ストリングのポイント数を返します。
 - **ST_OrderingEquals** - 二つのジオメトリが同じジオメトリを表現し、かつ点の並び順が同じかどうかをテストします。
 - **ST_Overlaps** - 二つのジオメトリが同じ次元を持ち、インタセクトして、かつ相手と重ならない点少なくとも一つあるかをテストします。
 - **ST_PatchN** - 多面体サーフェスの N 番目のジオメトリ (フェイス) を返します。
 - **ST_Perimeter** - ポリゴンジオメトリまたはジオグラフィの境界の長さを返します。
-

- **ST_Point** - X, Y と SRID の値からポイントを生成します。
- **ST_PointFromText** - WKT と与えられた SRID からポイントジオメトリを生成します。SRID が与えられていない場合は 0 (不明) とします。
- **ST_PointFromWKB** - WKB と与えられた SRID からジオメトリを生成します。
- **ST_PointN** - ジオメトリの最初のラインストリングまたは曲線ストリングの N 番目のポイントを返します。
- **ST_PointOnSurface** - ポリゴン内またはジオメトリ上にあるのが保証されたポイントを返します。
- **ST_Polygon** - ラインストリングから指定した SRID を持つポリゴンを生成します。
- **ST_PolygonFromText** - Well-Known Text (WKT) 表現と与えられた SRID からジオメトリを生成します。SRID を与えない場合は 0 (不明) となります。
- **ST_Relate** - 二つのジオメトリが与えられた交差行列パターンに合致するトポロジ関係があるかどうかを見るか、交差行列を計算するかします。
- **ST_RemEdgeModFace** - エッジを削除します。削除されるエッジが二つのフェイスを分割していたなら、一方のフェイスを削除し、もう一方のフェイスを両方の空間を覆うように編集します。
- **ST_RemEdgeNewFace** - エッジを削除し、消去対象エッジでフェイスが二つに分割されているなら元の二つのフェイスを削除し、一つの新しいフェイスに置き換えます。
- **ST_RemoveIsoEdge** - 孤立エッジを削除し、実行結果の記述を返します。エッジが孤立していない場合には、例外が投げられます。
- **ST_RemoveIsoNode** - 孤立ノードを削除し、実行結果が返されます。ノードが孤立していない (エッジの始端または終端である) 場合には、例外が投げられます。
- **ST_SRID** - ジオメトリの空間参照系識別子を返します。
- **ST_StartPoint** - Returns the first point of a LineString, CircularLineString, or NURBSCurve.
- **ST_SymDifference** - ジオメトリ A とジオメトリ B がインタセクトしていない部分を表現するジオメトリを返します。
- **ST_Touches** - 二つのジオメトリが少なくとも一つの共有点を持ち、かつ内部でインタセクトしていないようになっているかテストします。
- **ST_Transform** - 異なる空間参照系に投影変換された新しいジオメトリを返します。
- **ST_Union** - 入力ジオメトリのポイント集合の結合を表現するジオメトリを返します。
- **ST_Volume** - Computes the volume of a 3D geometry. Closed surfaces can have non-zero volume.
- **ST_WKBToSQL** - Well-Known Binary 表現 (WKB) から ST_Geometry 値を生成します。これは SRID を取らない ST_GeomFromWKB の別名です。
- **ST_WKTToSQL** - Well-Known Text 表現 (WKT) から指定した ST_Geometry 値を返します。これは ST_GeomFromText の別名です。
- **ST_Within** - A の全てのポイントが B 内にあり、かつ両方の内部が共有点を持つかどうかをテストします。
- **ST_X** - ポイントの X 値を返します。
- **ST_Y** - ポイントの Y 値を返します。
- **ST_Z** - ポイントの Z 値を返します。
- **ST_SRID** - TopoGeometry の空間参照系識別子を返します。

13.4 PostGIS ジオグラフィ対応関数

次に示す関数と演算子は、**ジオグラフィ** データ型を入力または出力に取る PostGIS 関数/演算子です。



Note

(T) の付いた関数はネイティブなジオグラフィ関数ではなく、実行時に ST_Transform を使ってジオメトリとの変換を行います。結果として、日付変更線や極を超えたり、ひとつの UTM ゾーンではカバーできない巨大なジオメトリを使用する場合に、予期しない挙動になることがあります。基本的な変換としては、UTM が望ましいですが、ランベルト正積方位図法 (北/南)、最悪のシナリオでメルカトルに頼ります。

- **ST_Area** - ポリゴンジオメトリの面積を返します。
- **ST_AsBinary** - ジオメトリ/ジオグラフィの、SRID メタデータを持たない OGC/ISO Well-Known バイナリ (WKB) 表現を返します。
- **ST_AsEWKT** - ジオメトリの SRID メタデータが付いた Well-Known Text (WKT) 表現を返します。
- **ST_AsGML** - GML 第 2 版または第 3 版としてジオメトリを返します。
- **ST_AsGeoJSON** - GeoJSON 形式のジオメトリまたは地物を返します。
- **ST_AsKML** - ジオメトリを KML 要素として返します。
- **ST_AsSVG** - ジオメトリから SVG パスデータを返します。
- **ST_AsText** - ジオメトリ/ジオグラフィの SRID メタデータのない Well-Known Text (WKT) 表現を返します。
- **ST_Azimuth** - 北を基準とした 2 点間の線の方位角を返します。
- **ST_Buffer** - あるジオメトリからの距離が指定された距離以下となる点全ての集合となるジオメトリを返します。
- **ST_Centroid** - ジオメトリの幾何学的重心を返します。
- **ST_ClosestPoint** - g1 上にある、g2 と最近傍となる 2 次元ポイントを返します。これは、あるジオメトリから他のジオメトリへの最短ラインの一つ目のポイントです。
- **ST_CoveredBy** - A の全てのポイントが B 内にあるかをテストします。
- **ST_Covers** - B の全ての点が A 内にあるかをテストします。
- **ST_DWithin** - 二つのジオメトリが与えられた距離内にあるかどうかをテストします。
- **ST_Distance** - 二つのジオメトリ値またはジオグラフィ値間の距離を返します。
- **ST_GeogFromText** - Well-Known Text 表現または拡張 WKT から指定したジオグラフィ値を返します。
- **ST_GeogFromWKB** - Well-Known Binary ジオメトリ表現 (WKB) または拡張 WKB(EWKB) からジオグラフィインスタンスを生成します。
- **ST_GeographyFromText** - Well-Known Text 表現または拡張 WKT から指定したジオグラフィ値を返します。
- **=** - ジオメトリ/ジオグラフィ A の座標と座標の並び順がジオメトリ/ジオグラフィ B と同じ場合に TRUE を返します。
- **ST_Intersection** - ジオメトリ A とジオメトリ B の共通部分を表現するジオメトリを返します。
- **ST_Intersects** - 二つのジオメトリがインタセクトしている (少なくとも一つの共有点がある) かどうかテストします。
- **ST_Length** - 線系ジオメトリの 2 次元長を返します。

- **ST_LineInterpolatePoint** - ラインに沿って、割合で示された位置の補間ポイントを返します。
- **ST_LineInterpolatePoints** - ラインに沿って、割合で示された複数の位置の補間ポイントを返します。
- **ST_LineLocatePoint** - ポイントに最も近いライン上のポイントの位置を割合で返します。
- **ST_LineSubstring** - 二つの割合位置からラインの一部を返します。
- **ST_Perimeter** - ポリゴンジオメトリまたはジオグラフィの境界の長さを返します。
- **ST_Project** - 始点から距離と方位で算出されたポイントを返します。
- **ST_Segmentize** - 与えた長さを超える線分を持たないよう変更したジオメトリ/ジオグラフィを返します。
- **ST_ShortestLine** - 二つのジオメトリの 3 次元の最短ラインを返します。
- **ST_Summary** - ジオメトリについての要約文を返します。
- **<->** - A と B の 2 次元距離を返します。
- **&&** - A の 2 次元バウンディングボックスが B の 2 次元バウンディングボックスとインタセクトする場合に TRUE を返します。

13.5 PostGIS ラスタ機能関数

次に示す関数と演算子は、**raster**データ型を入力または出力に取る PostGIS 関数/演算子です。アルファベット順に示します。

- **Box3D** - ラスタを囲むボックスの box3d 表現を返します。
- **@** - A のバウンディングボックスが B のバウンディングボックスに含まれる場合に TRUE を返します。倍精度浮動小数点数のバウンディングボックスを使います。
- **~** - A のバウンディングボックスが B のバウンディングボックスを含む場合に TRUE を返します。倍精度浮動小数点数のバウンディングボックスを使います。
- **=** - A のバウンディングボックスが B のバウンディングボックスと同じ場合に TRUE を返します。倍精度浮動小数点数のバウンディングボックスを使います。
- **&&** - A のバウンディングボックスが B のバウンディングボックスとインタセクトする場合に TRUE を返します。
- **&<** - A のバウンディングボックスが B のバウンディングボックスをオーバーラップするか、B のバウンディングボックスの左にある場合に TRUE を返します。
- **&>** - A のバウンディングボックスが B のバウンディングボックスをオーバーラップするか、B のバウンディングボックスの右にある場合に TRUE を返します。
- **~=** - A のバウンディングボックスが B のバウンディングボックスと同じ場合に TRUE を返します。
- **ST_Retile** - 任意のタイル化されたラスタカバレッジから構成されたタイルの集合を返します。
- **ST_AddBand** - 与えられたタイプで、与えられた初期値にした新しいバンドを、与えられたインデックス位置に追加したラスタを返します。インデックス位置を指定していない場合には、バンドは末尾に追加されます。
- **ST_AsBinary/ST_AsWKB** - ラスタの Well-Known Binary (WKB) 表現を返します。
- **ST_AsGDALRaster** - 指定された GDAL ラスタ書式でラスタタイルを返します。ラスタ書式はコンパイルしたライブラリが対応するものです。ライブラリが対応する書式の一覧を得るには ST_GDALRasters() を使います。
- **ST_AsHexWKB** - Well-Known Binary (WKB) ラスタを 16 進数表現で返します。

- **ST_AsJPEG** - ラスタの選択されたバンドを、単一の Joint Photographic Experts Group (JPEG) 画像としてバイト配列で返します。バンドを指定せず、1 バンドか 3 より多いバンドがある場合には、1 番バンドを使用します。3 バンドのみ指定した場合には、3 バンドを使用し、RGB に対応付けます。
- **ST_AsPNG** - ラスタの選択されたバンドを、単一の portable network graphics (PNG) 画像としてバイト配列で返します。バンドを指定せず、1 バンドか 3 バンドか 4 バンドある場合には、全てのバンドを使用します。バンドを指定せず、2 バンドか 4 より多いバンドがある場合には、1 番バンドを使用します。対象バンドは RGB または RGBA に対応付けられます。
- **ST_AsRaster** - PostGIS ジオメトリを PostGIS ラスタに変換します。
- **ST_AsRasterAgg** - 集約関数です。PostGIS ジオメトリを新しいラスタに変換します。
- **ST_AsTIFF** - ラスタの選択されたバンドを、単一の TIFF 画像 (バイト配列) として返します。バンドを指定しないか指定したバンドがラスタ内に無い場合には、全てのバンドの使用を試みます。
- **ST_Aspect** - 標高ラスタバンドの傾斜方向 (デフォルトの単位は度) を返します。地形解析に使えます。
- **ST_Band** - 既存のラスタの、一つ以上のバンドを新しいラスタとして返します。既存のラスタから新しいラスタを構築する際に使えます。
- **ST_BandFileSize** - ファイルシステムに格納されているバンドのファイルサイズを返します。バンド番号が指定されていない場合には、1 番と仮定します。
- **ST_BandFileTimestamp** - ファイルシステムに格納されているバンドのファイルタイムスタンプ返します。バンド番号が指定されていない場合には、1 番と仮定します。
- **ST_BandIsNoData** - 指定したバンドが NODATA 値だけで満たされている場合には、TRUE を返します。
- **ST_BandMetaData** - 指定したラスタバンドの基本的なメタデータを返します。バンド番号を指定しない場合には、1 番と仮定します。
- **ST_BandNoDataValue** - 指定されたバンドについてデータが無いことを表現する値を返します。バンド番号を指定しない場合には、1 番と仮定します。
- **ST_BandPath** - ファイルシステムに格納されているバンドのシステムファイルパスを返します。バンド番号が指定されていない場合には、1 番と仮定します。
- **ST_BandPixelType** - 指定したバンドのピクセルタイプを返します。バンド番号が指定されていない場合には、1 番と仮定します。
- **ST_Clip** - 入力ジオメトリで切り取ったラスタを返します。バンドが指定されていない場合には、全てのバンドが処理されます。crop が指定されていなかったり TRUE となっている場合には、出力ラスタは切り取られます。touched が TRUE の場合には、接触するピクセルは取り込まれ、TRUE でない場合には、中心がジオメトリ内にあるピクセルだけが取り込まれます。
- **ST_ColorMap** - 元のラスタと指定したバンドから 4 個までの 8BUI バンド (grayscale, RGB, RGBA) からなる新しいラスタを生成します。
- **ST_Contains** - rastA の外に rastB の点が無く、rastA の内部に rastB の内部の点の一つ以上ある場合に TRUE を返します。
- **ST_ContainsProperly** - rastB が rastA の内部でインタセクトし、かつ rastA の境界とも外部ともインタセクトしない場合に TRUE を返します。
- **ST_Contour** - 与えられたラスタバンドから等高線ベクタを生成します。GDAL 等高線生成アルゴリズムを使います。
- **ST_ConvexHull** - BandNoDataValue と等しいピクセル値を含むラスタの凸包ジオメトリを返します。一般的な形状でスキューのないラスタでは、ST_Envelope と同じ結果になります。不規則な形状をしているか回転しているラスタでのみ使います。
- **ST_Count** - ラスタまたはラスタカバレッジの指定したバンドのピクセル数を返します。バンドを指定しない場合には、1 番と仮定します。exclude_nodata_value を TRUE に設定している場合には、NODATA 値と等しくないピクセルのみを数えます。

- **ST_CountAgg** - 集約関数です。ラスタ集合の与えられたバンドのピクセル数を返します。バンドが指定されていない場合には、1 番と仮定します。exclude_nodata_value を TRUE に設定している場合には、NODATA 値と等しくないピクセルのみを数えます。
- **ST_CoveredBy** - rastA が rastB の外部に点を持たない場合に TRUE を返します。
- **ST_Covers** - rastB が rastA の外部に点を持たない場合に TRUE を返します。
- **ST_DFullyWithin** - rastA と rastB が指定した距離内に完全に収まる場合に TRUE を返します。
- **ST_DWithin** - rastA と rastB が指定した距離内にある場合に TRUE を返します。
- **ST_Disjoint** - rastA が rastB とインタセクトしない場合に TRUE を返します。
- **ST_DumpAsPolygons** - 指定されたラスタバンドから geomval (geom,val) 行の集合を返します。バンドを指定しない場合のデフォルトは 1 です。
- **ST_DumpValues** - 指定したバンドの値を 2 次元で得ます。
- **ST_Envelope** - ラスタの範囲のポリゴン表現を返します。
- **ST_FromGDALRaster** - 対応する GDAL ラスタファイルからラスタを返します。
- **ST_GeoReference** - GDAL 書式または一般的にワールドファイルでみられる ESRI 書式の地理参照メタデータを返します。デフォルトは GDAL です。
- **ST_Grayscale** - 元のラスタと指定したバンドを赤、緑、青バンドとして一つの 8BUI バンドを持つラスタを生成します。
- **ST_HasNoBand** - 指定したバンド番号のバンドが無い場合には、TRUE を返します。バンド番号を指定していない場合には、1 番と仮定します。
- **ST_Height** - ラスタの高さをピクセル単位で返します。
- **ST_HillShade** - 与えられた方位、高度、明度、スケールの入力を使って標高ラスタバンドの仮想照明を返します。
- **ST_Histogram** - ラスタまたはラスタカバレッジのビン範囲で分割したデータ分布をまとめるヒストグラムの集合を返します。ビン数を指定しない場合には自動計算されます。
- **ST_InterpolateRaster** - X 値と Y 値を使用してグリッド上のポイントを配置し、ポイントの Z 値をサーフェス標高として配置し、3 次元ポイントの入力セットに基づいてグリッドサーフェスを補間します。
- **ST_Intersection** - 二つのラスタの共有部分またはベクタ化したラスタとジオメトリとのインタセクトした部分を表現する、ラスタまたはジオメトリとピクセル値の組の集合を返します。
- **ST_IntersectionFractions** - ラスタ上の個々のセルのうち与えられたジオメトリで覆われたものの断片を計算します。
- **ST_Intersects** - rastA が rastB とインタセクトする場合に TRUE を返します。
- **ST_IsEmpty** - ラスタが空 (幅が 0 で高さが 0) の場合には TRUE を返します。他の場合には、FALSE を返します。
- **ST_MakeEmptyCoverage** - 空のラスタタイルのグリッドでジオリファレンスを施されている領域を生成します。
- **ST_MakeEmptyRaster** - 与えられたピクセル範囲 (width & height)、左上の X,Y、ピクセルサイズ、回転 (scalex, scaley, skewx, skewy) と空間参照系 (srid) が指定された空ラスタ (バンドを持たないラスタ) を返します。ラスタが渡されると、新しいラスタは渡されたラスタと同じサイズ、アラインメント、SRID になります。SRID が指定されていない場合には、空間参照系は不明 (0) とされます。
- **ST_MapAlgebra (callback function version)** - コールバック関数版 - 一つ以上の入力ラスタ、バンドインデックスと一つのユーザ定義コールバック関数から、一つのバンドからなるラスタを返します。

- **ST_MapAlgebraExpr** - 1 バンド版: 入力バンドに対する妥当な PostgreSQL 代数演算で形成された、指定したピクセルタイプとなる 1 バンドラスタを生成します。バンドを指定しない場合には、1 番を仮定します。
- **ST_MapAlgebraExpr** - 2 バンド版: 二つの入力バンドに対する妥当な PostgreSQL 代数演算で形成された、指定したピクセルタイプとなる 1 バンドラスタを生成します。バンドを指定しない場合には、どちらも 1 番と仮定します。結果ラスタは、一つ目のラスタのアラインメント (スケール、スキュー、ピクセル角位置) にあわされます。範囲は "extenttype" 引数で定義されます。取りうる "extenttype" の値は INTERSECTION, UNION, FIRST, SECOND です。
- **ST_MapAlgebraFct** - 1 バンド版 - 入力バンドに対する妥当な PostgreSQL 関数で形成された、指定したピクセルタイプとなる 1 バンドラスタを生成します。バンドを指定しない場合には、1 番と仮定します。
- **ST_MapAlgebraFct** - 2 バンド版 - 二つの入力バンドに対する妥当な PostgreSQL 関数で形成された、指定したピクセルタイプとなる 1 バンドラスタを生成します。バンドを指定しない場合には、1 番と仮定します。"extenttype" のデフォルトは INTERSECTION です。
- **ST_MapAlgebraFctNgb** - 1 バンド版: ユーザ定義 PostgreSQL 関数を使用する最近傍地図代数関数です。入力ラスタバンドの近傍の値を与えた PL/pgSQL ユーザ定義関数の結果からなるラスタを返します。
- **ST_MapAlgebra (expression version)** - 数式版 - 一つか二つの入力ラスタ、バンド番号、一つ以上のユーザ定義 SQL 式から一つのバンドを持つラスタを返します。
- **ST_MemSize** - ラスタが取る領域の合計をバイト単位で返します。
- **ST_MetaData** - ピクセル数、回転 (スキュー)、左上隅位置等のラスタオブジェクトに関する基本的なメタデータを返します。
- **ST_MinConvexHull** - NODATA 値を除いたラスタの凸包ジオメトリを返します。
- **ST_NearestValue** - 与えられたバンドの、columnx と rowy で指定されるか、またはラスタと同じ空間参照系で表現されたポイントで指定されたピクセルに最も近い NODATA でない値を返します。
- **ST_Neighborhood** - 与えられたバンドの columnX, columnY か、ラスタと同じ空間参照系のジオメトリポイントで指定されたピクセルの周囲にある、NODATA でない 2 次元倍精度浮動小数点数配列を返します。
- **ST_NotSameAlignmentReason** - ラスタが同じアラインメントを持つかどうか、また、持たない場合にはその理由を示す文字列を返します。
- **ST_NumBands** - ラスタオブジェクトのバンド数を返します。
- **ST_Overlaps** - rastA と rastB がインタセクトして、かつ一方がもう一方に完全には包含されない場合には TRUE を返します。
- **ST_PixelAsCentroid** - ピクセルで表現される面の重心 (ポイントジオメトリ) を返します。
- **ST_PixelAsCentroids** - 全てのピクセルについて重心 (ポイントジオメトリ) を、ピクセルごとのピクセル値とラスタ座標系の X と Y とを付けて返します。ポイントジオメトリの座標はピクセルで表現される面の重心です。
- **ST_PixelAsPoint** - ピクセルの左上隅のポイントジオメトリを返します。
- **ST_PixelAsPoints** - 全てのピクセルについてポイントジオメトリを、ピクセルごとのピクセル値とラスタ座標系の X と Y とを付けて返します。ポイントジオメトリの座標はピクセルの左上隅です。
- **ST_PixelAsPolygon** - 指定した行と列のピクセルの境界となるジオメトリを返します。
- **ST_PixelAsPolygons** - 全てのピクセルについて境界となるジオメトリを、ピクセルごとのピクセル値とラスタ座標系の X と Y とを付けて返します。
- **ST_PixelHeight** - 空間参照系の地理的な単位でのピクセルの高さを返します。
- **ST_PixelOfValue** - 検索値と同じ値を持つピクセルの columnx, rowy ピクセル座標を得ます。
- **ST_PixelWidth** - 空間参照系の地理的な単位でのピクセルの幅を返します。

- **ST_Polygon** - NODATA 値でないピクセル値を持つピクセルの結合で形成されるマルチポリゴンジオメトリを返します。バンドを指定しない場合のデフォルトは 1 です。
- **ST_Quantile** - ラスタまたはラスタテーブルカバレッジのサンプルまたは母集団の分位数を計算します。値がラスタの 25%,50%,75% にあるかを調べることができます。
- **ST_RastFromHexWKB** - Well-Known バイナリ (WKB) ラスタの 16 進数表現からラスタを返します。
- **ST_RastFromWKB** - Well-Known Binary (WKB) ラスタからラスタ値を返します。
- **ST_RasterToWorldCoord** - ラスタの指定した列と行における左上隅の地理座標 X 値と Y 値 (経度と緯度) を返します。列と行の番号は 1 始まりです。
- **ST_RasterToWorldCoordX** - ラスタの指定した列と行における左上隅の地理座標の X 値を返します。列と行の番号は 1 始まりです。
- **ST_RasterToWorldCoordY** - ラスタの指定した列と行における左上隅の地理座標の Y 値を返します。列と行の番号は 1 始まりです。
- **ST_Reclass** - 元のラスタから再分類したバンドタイプからなるラスタを生成します。nband は変更するバンドです。nband が指定されていない場合には、1 と仮定します。他の全てのバンドは変更せずに返します。可視画像の書式としてより単純な描画を行うために、16BUI バンドを 8BUI バンドに変換する、等のために使います。
- **ST_ReclassExact** - 元ラスタから再分類したバンドからなる新しいラスタを生成します。元ラスタのバンド内の値から新たに生成されるラスタのバンド内の値への 1 対 1 のマッピングを使います。
- **ST_Resample** - 指定したリサンプリングアルゴリズム、新しいピクセル範囲、グリッドの隅、定義するか他のラスタから借りてきた地理参照属性を使ってリサンプリングを行います。
- **ST_Rescale** - スケール (ピクセルサイズ) だけを調整するリサンプリングを行います。新しいピクセル値のリサンプリングアルゴリズムとして最近傍補間 ('NearestNeighbor' (英語または米式綴り方))、双線形補間 ('Bilinear')、3 次補間 ('Cubic')、3 次スプライン補間 ('CubicSpline')、ランチョス補間 ('Lanczos') を用います。デフォルトは最近傍補間です。
- **ST_Resize** - ラスタを新しい幅、高さにサイズ再設定を行います。
- **ST_Reskew** - キュー (回転パラメーター) だけを調整するリサンプリングを行います。新しいピクセル値のリサンプリングアルゴリズムとして最近傍補間 ('NearestNeighbor' (米式綴り方))、双線形補間 ('Bilinear')、3 次補間 ('Cubic')、3 次スプライン補間 ('CubicSpline')、ランチョス補間 ('Lanczos') を用います。デフォルトは最近傍補間です。
- **ST_Rotation** - ラスタの回転をラジアンで返します。
- **ST_Roughness** - DEM の「粗度」を計算したラスタを返します。
- **ST_SRID** - ラスタの spatial_ref_sys テーブルで定義されている空間参照系識別番号を返します。
- **ST_SameAlignment** - ラスタが同じスキュー、スケール、空間参照系、オフセットを持つ (ピクセルが分割されることなく同じグリッドに置かれている) 場合に TRUE を返し、そうでない場合は問題を詳述する通知とともに FALSE を返します。
- **ST_ScaleX** - 空間参照系の地理的な単位でのピクセル幅の X 成分を返します。
- **ST_ScaleY** - 空間参照系の地理的な単位でのピクセル幅の Y 成分を返します。
- **ST_SetBandIndex** - データベース外バンドの外部バンド番号の更新
- **ST_SetBandIsNoData** - バンドの isnodata フラグを TRUE にします。
- **ST_SetBandNoDataValue** - 指定したバンドに NODATA を表現する値を設定します。バンドを指定しない場合には、1 番と仮定します。NODATA 値を持たないようにするには、nodatavalue に NULL を指定します。
- **ST_SetBandPath** - データベース外バンドの外部パスとバンド番号を更新します。

- **ST_SetGeoReference** - 地理参照 6 パラメーターを一度に設定します。数値は空白で区切ります。GDAL または ESRI 書式の入力を受け付けます。デフォルトは GDAL です。
- **ST_SetM** - 入力ジオメトリと同じ X/Y 値を持ち、かつ、指定されたりサンプリングアルゴリズムを使ってラスタから複写された M 値を持つジオメトリを返します。
- **ST_SetRotation** - ラスタの回転をラジアン単位で設定します。
- **ST_SetSRID** - スタの SRID を spatial_ref_sys に定義されている特定の整数値に設定します。
- **ST_SetScale** - ピクセルサイズの X 値と Y 値を空間参照系の単位で設定します。数値は単位/ピクセルの幅または高さです。
- **ST_SetSkew** - 地理参照のスキュー (回転パラメーター) の X 値と Y 値を設定します。一つだけ渡した場合には、X 値と Y 値は同じ値に設定されます。
- **ST_SetUpperLeft** - ラスタの左上隅の投影座標系の X 値と Y 値を設定します。
- **ST_SetValue** - 与えられたバンドの columnX, columnY か、ラスタと同じ空間参照系のジオメトリポイントで指定されたピクセルの値または指定したジオメトリとインタセクトするピクセル群の値を設定することから得られる、変更されたラスタを返します。バンド番号は 1 始まりで、指定しない場合には、1 番と仮定します。
- **ST_SetValues** - 与えられたバンドに複数の値を設定して、変更されたラスタを返します。
- **ST_SetZ** - 入力ジオメトリと同じ X/y 座標値と、指定されたりサンプリングアルゴリズムを使ってラスタから複写された Z 値とを持つジオメトリを返します。
- **ST_SkewX** - 空間参照の X スキュー (回転パラメータ) を返します。
- **ST_SkewY** - 空間参照の Y スキュー (回転パラメータ) を返します。
- **ST_Slope** - 標高ラスタバンドの傾斜角 (デフォルトでは度単位) を返します。地形解析に使えます。
- **ST_SnapToGrid** - グリッドにスナップすることでラスタをリサンプリングします。新しいピクセル値のリサンプリングアルゴリズムとして最近傍補間 ('NearestNeighbor' (米式綴り方)), 双線形補間 ('Bilinear'), 3 次補間 ('Cubic'), 3 次スプライン補間 ('CubicSpline'), ランチョス補間 ('Lanczos') を用います。デフォルトは最近傍補間です。
- **ST_Summary** - ラスタの中身の概要が文字列で返されます。
- **ST_SummaryStats** - 与えられたラスタのバンドまたはラスタカバレッジの count, sum, mean, stddev, min, max からなる統計情報のサマリーです。バンドを指定しない場合には、1 番と仮定します。
- **ST_SummaryStatsAgg** - 集約関数。与えられたラスタのバンドの集合の count, sum, mean, stddev, min, max からなる統計情報のサマリーです。バンドを指定しない場合には、1 番と仮定します。
- **ST_TPI** - 地形的位置指数を計算したラスタを返します。
- **ST_TRI** - 起伏指標を計算したラスタを返します。
- **ST_Tile** - 求められた出力ラスタのピクセル数に基づいて入力ラスタを分割した結果のラスタ集合を返します。
- **ST_Touches** - rastA と rastB が少なくとも一つの共通の点を持ち、かつ二つのラスタの内部同士がインタセクトしない場合に TRUE を返します。
- **ST_Transform** - ラスタを既知の空間参照系から他の既知の空間参照系に、指定したりサンプリングアルゴリズムで投影変換します。新しいピクセル値のリサンプリングアルゴリズムとして最近傍補間 ('NearestNeighbor' (米式綴り方)), 双線形補間 ('Bilinear'), 3 次補間 ('Cubic'), 3 次スプライン補間 ('CubicSpline'), ランチョス補間 ('Lanczos') を用います。デフォルトは最近傍補間です。
- **ST_Union** - ラスタタイルの集合を結合して 1 以上のバンドからなる単一ラスタを返します。
- **ST_UpperLeftX** - 適用されている空間参照系でのラスタの左上隅の X 座標値を返します。
- **ST_UpperLeftY** - 適用されている空間参照系でのラスタの左上隅の Y 座標値を返します。

- **ST_Value** - 指定したバンドにおける `columnx`, `rowy` で指定したピクセルまたは指定したジオメトリポイントに対応するピクセルの値を返します。バンド番号は 1 始まりで、指定しない場合には、1 番と仮定します。`exclude_nodata_value` が `FALSE` に設定された場合には、`NODATA` ピクセルを含む全てのピクセルがインタセクトするかが考慮され、値を返します。`exclude_nodata_value` を渡さない場合には、ラスタのメタデータから読みます。
- **ST_ValueCount** - ラスタ (またはラスタカバレッジ) の指定されたバンドで、指定した値を持つピクセルを対象として、ピクセルバンド値とピクセル数からなるレコードの集合を返します。バンドを指定しない場合には、1 番と仮定します。デフォルトでは `NODATA` 値のピクセルは数えられず、ピクセルの他の値は出力され、ピクセルバンド値は最も近い整数に丸められます。
- **ST_Width** - ラスタの幅をピクセル単位で返します。
- **ST_Within** - `rastA` が `rastB` の外部に点を持たず、`rastA` の内部の少なくとも一つの点が `rastB` の内部にある場合に `TRUE` を返します。
- **ST_WorldToRasterCoord** - ラスタの空間参照系による地理座標の X 値と Y 値 (経度と緯度) またはポイントジオメトリに対応するピクセルの左上隅を返します。
- **ST_WorldToRasterCoordX** - ラスタの空間参照系に基づくポイントジオメトリ (pt) または X,Y 座標値 (xw,yw) に対応するラスタの列を返します。
- **ST_WorldToRasterCoordY** - ラスタの空間参照系に基づくポイントジオメトリ (pt) または X,Y 座標値 (xw,yw) に対応するラスタの行を返します。
- **UpdateRasterSRID** - ユーザが指定したカラムとテーブルにあるラスタの全てについて `SRID` を変更します。

13.6 PostGIS ジオメトリ/ジオグラフィ/ラスタのダンプ関数

次に示す関数は、**geometry_dump** または **geomval** データ型の集合または単一データを入力または出力に取る PostGIS 関数です。

- **ST_DumpAsPolygons** - 指定されたラスタバンドから `geomval (geom,val)` 行の集合を返します。バンドを指定しない場合のデフォルトは 1 です。
- **ST_Intersection** - 二つのラスタの共有部分またはベクタ化したラスタとジオメトリとのインタセクトした部分を表現する、ラスタまたはジオメトリとピクセル値の組の集合を返します。
- **ST_Dump** - ジオメトリの要素となる `geometry_dump` 行の集合を返します。
- **ST_DumpPoints** - ジオメトリ内の座標の行である `geometry_dump` 行の集合を返します。
- **ST_DumpRings** - ポリゴンのリングごとの `geometry_dump` 行の集合を返します。
- **ST_DumpSegments** - ジオメトリ内の辺の行である `geometry_dump` 行の集合を返します。

13.7 PostGIS ボックス関数

次に示す関数は、PostGIS 空間型の `box` 系の型を入力または出力に取る PostGIS 関数です。ボックス系には **box2d** と **box3d** があります。

- **Box2D** - ジオメトリの 2 次元範囲を表現する `BOX2D` を返します。
- **Box3D** - ジオメトリの 3 次元範囲を表現する `BOX3D` を返します。
- **MakeTopologyPrecise** - トポロジの頂点を精度グリッドにスナップします。

- **Box3D** - ラスタを囲むボックスの **box3d** 表現を返します。
- **ST_3DExtent** - ジオメトリの 3 次元バウンディングボックスを返す集約関数です。
- **ST_3DMakeBox** - 二つの 3 次元のポイントジオメトリで定義される BOX3D を生成します。
- **ST_AsMVTGeom** - ジオメトリを MVT タイルの座標空間に変換します。
- **ST_AsTWKB** - TWKB (Tiny Well-Known Binary) としてジオメトリを出力します。
- **ST_Box2dFromGeoHash** - GeoHash 文字列から BOX2D を返します。
- **ST_ClipByBox2D** - 長方形内に落ちるジオメトリの一部を返します。
- **ST_EstimatedExtent** - 空間テーブルの推定範囲を返します。
- **ST_Expand** - 他のバウンディングボックスまたはジオメトリから拡張されたバウンディングボックスを返します。
- **ST_Extent** - ジオメトリのバウンディングボックスを返す集約関数です。
- **ST_MakeBox2D** - 二つの 2 次元のポイントジオメトリで定義される BOX2D を生成します。
- **ST_RemoveIrrelevantPointsForView** - 指定した矩形表示範囲での描画に関係のないジオメトリのポイントを削除します。
- **ST_XMax** - 2 次元または 3 次元のバウンディングボックスまたはジオメトリの X の最大値を返します。
- **ST_XMin** - 2 次元または 3 次元のバウンディングボックスまたはジオメトリの X の最小値を返します。
- **ST_XSize** - Returns the X size of a 2D or 3D bounding box or a geometry.
- **ST_YMax** - 2 次元または 3 次元のバウンディングボックスまたはジオメトリの Y の最大値を返します。
- **ST_YMin** - 2 次元または 3 次元のバウンディングボックスまたはジオメトリの Y の最小値を返します。
- **ST_YSize** - Returns the Y size of a 2D or 3D bounding box or a geometry.
- **ST_ZMax** - 2 次元または 3 次元のバウンディングボックスまたはジオメトリの Z の最大値を返します。
- **ST_ZMin** - 2 次元または 3 次元のバウンディングボックスまたはジオメトリの Z の最小値を返します。
- **ST_ZSize** - Returns the Z size of a 2D or 3D bounding box or a geometry.
- **RemoveUnusedPrimitives** - 存在する TopoGeometry オブジェクトを定義するのに必要でないトポロジプリミティブを削除します。
- **ValidateTopology** - トポロジの問題についての詳細を示す `validatetopology_returntype` の集合を返します。
- **ValidateTopologyPrecision** - トポロジ内の低精度の頂点を返します。
- **~(box2df,box2df)** - 二つの単精度浮動小数点数による 2 次元バウンディングボックス (BOX2DF) の一方がもう一方を包含する場合に TRUE を返します。
- **~(box2df,geometry)** - 単精度浮動小数点数による 2 次元バウンディングボックス (BOX2DF) をジオメトリの (キャッシュされている)2 次元バウンディングボックスが包含する場合に TRUE を返します。
- **~(geometry,box2df)** - ジオメトリの (キャッシュされている)2 次元バウンディングボックスが単精度浮動小数点数による n 次元バウンディングボックス (GIDX) を包含する場合に TRUE を返します。
- **@(box2df,box2df)** - 二つの単精度浮動小数点数による n 次元バウンディングボックス (GIDX) の一方がもう一方を包含する場合に TRUE を返します。
- **@(box2df,geometry)** - 単精度浮動小数点数による 2 次元バウンディングボックス (BOX2DF) がジオメトリの 2 次元バウンディングボックスに包含される場合に TRUE を返します。

- **@(geometry,box2df)** - ジオメトリの 2 次元バウンディングボックスが単精度浮動小数点数による 2 次元バウンディングボックス (BOX2DF) に包含される場合に TRUE を返します。
- **&&(box2df,box2df)** - 二つの単精度浮動小数点数による 2 次元バウンディングボックス (BOX2DF) が相互にインタセクトする場合に TRUE を返します。
- **&&(box2df,geometry)** - 単精度浮動小数点数による 2 次元バウンディングボックスがジオメトリの (キャッシュされている)2 次元バウンディングボックスとインタセクトする場合に TRUE を返します。
- **&&(geometry,box2df)** - ジオメトリの (キャッシュされている)2 次元バウンディングボックスが単精度浮動小数点数による 2 次元バウンディングボックスとインタセクトする場合に TRUE を返します。

13.8 3 次元対応 PostGIS 関数

次に示す関数は、Z インデックスを放り出さない PostGIS 関数です。

- **AddGeometryColumn** - ジオメトリカラムを既存のテーブルに追加します。
- **Box3D** - ジオメトリの 3 次元範囲を表現する BOX3D を返します。
- **CG_3DAlphaWrapping** - 厳密にジオメトリを囲む 3 次元アルファラッピングを計算します。
- **CG_3DArea** - 3 次元の面ジオメトリの面積を計算します。立体の場合は 0 を返します。
- **CG_3DConvexHull** - ジオメトリの 3 次元の凸包を計算します。
- **CG_3DDifference** - 3 次元の差分を計算します。
- **CG_3DIntersection** - 3 次元のインタセクトした (共有する) 部分を計算します。
- **CG_3DRotate** - ジオメトリを指定したベクトルを軸に 3 次元空間で回転させます。
- **CG_3DScale** - ジオメトリを X,Y,Z 軸に沿って個別の係数で拡大縮小します。
- **CG_3DScaleAroundCenter** - ジオメトリを 3 次元空間で指定した点を中心として拡大縮小します。
- **CG_3DTranslate** - ジオメトリを与えたオフセットで 3 次元で移動させます。
- **CG_3DUnion** - postgis_sfcgal を使用して 3 次元結合を実行します。
- **CG_ConstrainedDelaunayTriangles** - 入力ジオメトリの周りの制約付きドロネー三角形を返します。
- **CG_Extrude** - 関連するボリュームにサーフェスを押し出します。
- **CG_ForceLHR** - LHR (Left Hand Rule) 方向に強制します。
- **CG_IsPlanar** - サーフェスが平面であるかないかをチェックします。
- **CG_IsSolid** - ジオメトリが立体であるかどうかをテストします。妥当性チェックは行いません。
- **CG_MakeSolid** - ジオメトリを立体にキャストします。チェックはしません。妥当な立体を得るには、入力ジオメトリは閉じた多面体サーフェスか閉じた TIN でなければなりません。
- **CG_NurbsCurveApproximate** - Creates an approximating NURBS curve fitting data points within a tolerance
- **CG_NurbsCurveDerivative** - Computes the derivative (tangent, curvature) of a NURBS curve at a given parameter
- **CG_NurbsCurveInterpolate** - Creates an interpolating NURBS curve passing through all given data points
- **CG_Orientation** - サーフェスの方向を判定します。

- **CG_RotateX** - ジオメトリを X 軸を軸に指定した角度で回転させます。
 - **CG_RotateY** - ジオメトリを Y 軸を軸に指定した角度で回転させます。
 - **CG_RotateZ** - ジオメトリを Z 軸を軸に指定した角度で回転させます。
 - **CG_Simplify** - 主要な地物と Z/M 値を維持しつつジオメトリの複雑さを減らします。
 - **CG_StraightSkeleton** - ジオメトリからストレートスケルトンを計算します。
 - **CG_Tessellate** - ポリゴンまたは多面体サーフェスのテッセレーションを計算し、TIN または TIN コレクションを返します。
 - **CG_Visibility** - ポリゴンジオメトリ内のポイント又は辺から可視領域ポリゴンを計算する
 - **CG_Volume** - Computes the volume of a 3D geometry. Closed surfaces can have non-zero volume.
 - **DropGeometryColumn** - ジオメトリカラムを空間テーブルから除去します。
 - **GeometryType** - ジオメトリのタイプを文字列で返します。
 - **ST_3DArea** - 3 次元の面ジオメトリの面積を計算します。立体の場合は 0 を返します。
 - **ST_3DClosestPoint** - g1 上の、g2 に最も近い 3 次元ポイントを返します。これは 3 次元の最短ラインの始点です。
 - **ST_3DConvexHull** - ジオメトリの 3 次元の凸包を計算します。
 - **ST_3DDFullyWithin** - 二つの 3 次元ジオメトリが完全に与えられた 3 次元距離内にあるかどうかをテストします。
 - **ST_3DDWithin** - 二つの 3 次元ジオメトリが与えられた 3 次元距離内にあるかどうかをテストします。
 - **ST_3DDifference** - 3 次元の差分を計算します。
 - **ST_3DDistance** - 投影座標系の単位で、二つのジオメトリ間の 3 次元デカルト距離の最小値を返します (空間参照系に基づきます)。
 - **ST_3DExtent** - ジオメトリの 3 次元バウンディングボックスを返す集約関数です。
 - **ST_3DInterpolatePoint** - 3 次元で最も近いジオメトリの補間 M 値を返します。
 - **ST_3DIntersection** - 3 次元のインタセクトした (共有する) 部分を計算します。
 - **ST_3DIntersects** - 二つのジオメトリが 3 次元空間において空間的にインタセクトするかどうかをテストします。ポイント、ラインストリング、ポリゴン、多面体サーフェス (面) についてのみ動作します。
 - **ST_3DLength** - 線ジオメトリの 3 次元長を返します。
 - **ST_3DLineInterpolatePoint** - 3 次元ラインに沿って、割合で示された位置の補間ポイントを返します。
 - **ST_3DLongestLine** - 二つのジオメトリ間の 3 次元最長ラインを返します。
 - **ST_3DMaxDistance** - 二つのジオメトリ間の 3 次元最大デカルト距離 (空間参照系に基づく) を空間参照系の単位で返します。
 - **ST_3DPerimeter** - ポリゴンジオメトリの 3 次元周長を返します。
 - **ST_3DShortestLine** - 二つのジオメトリの 3 次元の最短ラインを返します。
 - **ST_3DUnion** - 3 次元の結合を計算します。
 - **ST_AddMeasure** - ラインに沿った M 値を補間します。
 - **ST_AddPoint** - ラインストリングにポイントを追加します。
 - **ST_Affine** - ジオメトリに 3 次元アフィン変換を適用します。
-

- **ST_ApproximateMedialAxis** - 面ジオメトリの近似的な中心軸を計算します。
 - **ST_AsBinary** - ジオメトリ/ジオグラフィの、SRID メタデータを持たない OGC/ISO Well-Known バイナリ (WKB) 表現を返します。
 - **ST_AsEWKB** - ジオメトリの、SRID メタデータを持つ Extended Well-Known バイナリ (EWKB) 表現を返します。
 - **ST_AsEWKT** - ジオメトリの SRID メタデータが付いた Well-Known Text (WKT) 表現を返します。
 - **ST_AsGML** - GML 第 2 版または第 3 版としてジオメトリを返します。
 - **ST_AsGeoJSON** - GeoJSON 形式のジオメトリまたは地物を返します。
 - **ST_AsHEXEWKB** - ジオメトリの HEXEWKB 表現を (文字列として) 返します。リトルエンディアン (NDR) またはビッグエンディアン (XDR) のどちらかのエンコーディングを使います。
 - **ST_AsKML** - ジオメトリを KML 要素として返します。
 - **ST_AsX3D** - ジオメトリを X3D ノード要素書式 (ISO-IEC-19776-1.2-X3DEncodings-XML) で返します。
 - **ST_Boundary** - ジオメトリの境界を返します。
 - **ST_BoundingDiagonal** - ジオメトリのバウンディングボックスの対角線を返します。
 - **ST_CPAWithin** - 二つのトラジェクトリの最接近時の距離が指定距離内であるかどうかをテストします。
 - **ST_CatmullRomSmoothing** - Catmull-Rom スプライン曲線を用いて平滑化されたジオメトリを返します。
 - **ST_ChaikinSmoothing** - チャイキンのアルゴリズムを使って、与えられたジオメトリの平滑化されたものを返します。
 - **ST_ClosestPointOfApproach** - 二つのトラジェクトリの最接近時の距離を返します。
 - **ST_Collect** - ジオメトリの集合からジオメトリコレクションまたはマルチ系ジオメトリを生成します。
 - **ST_ConstrainedDelaunayTriangles** - 入力ジオメトリの周りの制約付きドロネー三角形を返します。
 - **ST_ConvexHull** - ジオメトリの凸包を計算します。
 - **ST_CoordDim** - ジオメトリの座標次元を返します。
 - **ST_CurveN** - 複合曲線の N 番目の曲線ジオメトリを返します。
 - **ST_CurveToLine** - 曲線を含むジオメトリを線ジオメトリに変換します。
 - **ST_DelaunayTriangles** - ジオメトリの頂点のドロネー三角形を返します。
 - **ST_Difference** - ジオメトリ B とインタセクトしていないジオメトリ A の一部を表現するジオメトリを計算します。
 - **ST_DistanceCPA** - 二つのトラジェクトリの最接近する時の距離を返します。
 - **ST_Dump** - ジオメトリの要素となる `geometry_dump` 行の集合を返します。
 - **ST_DumpPoints** - ジオメトリ内の座標の行である `geometry_dump` 行の集合を返します。
 - **ST_DumpRings** - ポリゴンのリングごとの `geometry_dump` 行の集合を返します。
 - **ST_DumpSegments** - ジオメトリ内の辺の行である `geometry_dump` 行の集合を返します。
 - **ST_EndPoint** - Returns the last point of a LineString, CircularLineString, or NURBS Curve.
 - **ST_ExteriorRing** - ポリゴンの外環を表現するラインストリングを返します。
 - **ST_Extrude** - 関連するボリュームにサーフェスを押し出します。
 - **ST_FlipCoordinates** - X 値と Y 値を入れ替えたジオメトリを返します。
-

- **ST_Force2D** - ジオメトリを 2 次元モードに強制します。
 - **ST_ForceCurve** - 該当する場合は、ジオメトリを曲線タイプに変換します。
 - **ST_ForceLHR** - LHR (Left Hand Rule) 方向に強制します。
 - **ST_ForcePolygonCCW** - 全ての外環を反時計回りに、全ての内環を時計回りに、それぞれ強制します。
 - **ST_ForcePolygonCW** - 全ての外環を時計回りに、全ての内環を反時計回りに、それぞれ強制します。
 - **ST_ForceRHR** - ポリゴンの頂点の方向を右回りに強制します。
 - **ST_ForceSFS** - SFS 1.1 ジオメトリタイプのみ使うようジオメトリに強制します。
 - **ST_Force3D** - ジオメトリを XYZ モードに強制します。これは ST_Force3DZ の別名です。
 - **ST_Force3DZ** - ジオメトリを XYZ モードに強制します。
 - **ST_Force4D** - ジオメトリを XYZM モードに強制します。
 - **ST_ForceCollection** - ジオメトリをジオメトリコレクションに変換します。
 - **ST_GeomFromEWKB** - 拡張 Well-Known Binary 表現 (EWKB) から指定した ST_Geometry 値を返します。
 - **ST_GeomFromEWKT** - 拡張 Well-Known Text 表現 (EWKT) から指定された ST_Geometry 値を返します。
 - **ST_GeomFromGML** - GML 表現から PostGIS ジオメトリオブジェクトを出力します。
 - **ST_GeomFromGeoJSON** - ジオメトリの GeoJSON 表現を入力として、PostGIS ジオメトリオブジェクトを出力します。
 - **ST_GeomFromKML** - ジオメトリの KML 表現の入力を取り、PostGIS ジオメトリオブジェクトを出力します。
 - **ST_GeometricMedian** - マルチポイントの幾何学的中央値を返します。
 - **ST_GeometryN** - ジオメトリコレクションの要素を一つ返します。
 - **ST_GeometryType** - ジオメトリの SQL-MM 型を文字列で返します。
 - **ST_HasArc** - ジオメトリに円弧が含まれているかどうかテストします。
 - **ST_HasM** - ジオメトリが M 値をもっているかどうかを確認します。
 - **ST_HasZ** - ジオメトリが Z 値を持っているかどうかを確認します。
 - **ST_InteriorRingN** - ポリゴンの N 番目の内環 (穴) を返します。
 - **ST_InterpolatePoint** - ジオメトリのポイントに最も近いポイント上の補間 M 値を返します。
 - **ST_Intersection** - ジオメトリ A とジオメトリ B の共通部分を表現するジオメトリを返します。
 - **ST_IsClosed** - ラインストリングの始点と終点が一致しているかをテストします。多面体サーフェスについては閉じているか (立体であるか) をテストします。
 - **ST_IsCollection** - ジオメトリのタイプがジオメトリコレクションかをテストします。
 - **ST_IsPlanar** - サーフェスが平面であるかないかをチェックします。
 - **ST_IsPolygonCCW** - ポリゴンが反時計回りの外環を持っていて、時計回りの内環を持っているかをテストします。
 - **ST_IsPolygonCW** - ポリゴンが時計回りの外環を持っていて、反時計回りの内環を持っているかをテストします。
 - **ST_IsSimple** - ジオメトリが自己インタセクトまたは自己接触となるポイントが無いかをテストします。
 - **ST_IsSolid** - ジオメトリが立体であるかどうかをテストします。妥当性チェックは行いません。
-

- **ST_IsValidTrajectory** - ジオメトリが妥当なトラジェクトリであるかどうかをテストします。
 - **ST_LengthSpheroid** - 回転楕円体面上の経度緯度のジオメトリの 2 次元または 3 次元の長さ/周長を返します。
 - **ST_LineFromMultiPoint** - マルチポイントジオメトリからラインストリングを生成します。
 - **ST_LineInterpolatePoint** - ラインに沿って、割合で示された位置の補間ポイントを返します。
 - **ST_LineInterpolatePoints** - ラインに沿って、割合で示された複数の位置の補間ポイントを返します。
 - **ST_LineSubstring** - 二つの割合位置からラインの一部を返します。
 - **ST_LineToCurve** - 曲線を含むジオメトリを線ジオメトリに変換します。
 - **ST_LocateBetweenElevations** - 標高 (Z 値) 範囲にある部分ジオメトリを返します。
 - **ST_M** - ポイントの M 値を返します。
 - **ST_MMax** - Returns the M maxima of a geometry.
 - **ST_MMin** - ジオメトリの M 値の最小値を返します。
 - **ST_MSize** - Returns the M size of a geometry.
 - **ST_MakeLine** - POINT、MULTIPOINT、LINESTRING から LINESTRING を生成します。
 - **ST_MakePoint** - 2 次元、3 次元 (XYZ)、4 次元のポイントを生成します。
 - **ST_MakePolygon** - Creates a Polygon or CurvePolygon from a shell and optional list of holes.
 - **ST_MakeSolid** - ジオメトリを立体にキャストします。チェックはしません。妥当な立体を得るには、入力ジオメトリは閉じた多面体サーフェスか閉じた TIN でなければなりません。
 - **ST_MakeValid** - 頂点を失うことなしに不正なジオメトリを妥当なジオメトリにしようと試みます。
 - **ST_MemSize** - ジオメトリが取るメモリ空間の合計を返します。
 - **ST_MemUnion** - ジオメトリを結合する集約関数で、メモリを効率的に使いますが処理時間のかかるものです。
 - **ST_NDims** - ST_Geometry 値の座標次元を返します。
 - **ST_NPoints** - ジオメトリのポイント (頂点) の数を返します。
 - **ST_NRings** - ポリゴンジオメトリのリング数を返します。
 - **ST_Node** - ラインストリングの集合にノードを作成します。
 - **ST_NumCurves** - 複合曲線内の曲線の数値を返します。
 - **ST_NumGeometries** - ジオメトリコレクションの要素数を返します。
 - **ST_NumPatches** - 多面体サーフェスのフェイス数を返します。多面体でないジオメトリの場合には NULL を返します。
 - **ST_Orientation** - サーフェスの方向を判定します。
 - **ST_PatchN** - 多面体サーフェスの N 番目のジオメトリ (フェイス) を返します。
 - **ST_PointFromWKB** - WKB と与えられた SRID からジオメトリを生成します。
 - **ST_PointN** - ジオメトリの最初のラインストリングまたは曲線ストリングの N 番目のポイントを返します。
 - **ST_PointOnSurface** - ポリゴン内またはジオメトリ上にあるのが保証されたポイントを返します。
 - **ST_Points** - ジオメトリの全ての座標を含むマルチポイントを返します。
 - **ST_Polygon** - ラインストリングから指定した SRID を持つポリゴンを生成します。
 - **ST_RemovePoint** - ラインストリングからポイントを削除します。
-

- **ST_RemoveRepeatedPoints** - 重複ポイントを除いたジオメトリを返します。
 - **ST_Reverse** - 頂点の順序を逆にしたジオメトリを返します。
 - **ST_Rotate** - ジオメトリを原点について回転させます。
 - **ST_RotateX** - ジオメトリを X 軸について回転させます。
 - **ST_RotateY** - ジオメトリを Y 軸について回転させます。
 - **ST_RotateZ** - ジオメトリを Z 軸について回転させます。
 - **ST_Scale** - 与えた係数でジオメトリを拡大縮小します。
 - **ST_Scroll** - 閉じた LINESTRING の開始点を変更する。
 - **ST_SetPoint** - ラインストリングのポイントを与えられたポイントに置き換えます。
 - **ST_ShiftLongitude** - 経度座標値を-180 度から 180 度の範囲と 0 度から 360 度の範囲との二つの範囲を行き来するようシフトします。
 - **ST_SnapToGrid** - 入力ジオメトリの全ての点を規則的なグリッドにスナップします。
 - **ST_StartPoint** - Returns the first point of a LineString, CircularLineString, or NURBS Curve.
 - **ST_StraightSkeleton** - ジオメトリからストレートスケルトンを計算します。
 - **ST_SwapOrdinates** - 与えられたジオメトリにおいて与えられた座標の値を入れ替えたジオメトリを返します。
 - **ST_SymDifference** - ジオメトリ A とジオメトリ B がインタセクトしていない部分を表現するジオメトリを返します。
 - **ST_Tessellate** - ポリゴンまたは多面体サーフェスのテッセレーションを計算し、TIN または TIN コレクションを返します。
 - **ST_TransScale** - 与えられた係数とオフセットでジオメトリを変換します。
 - **ST_Translate** - 与えられたオフセットでジオメトリを変換します。
 - **ST_UnaryUnion** - 単一のジオメトリの要素の結合を計算します。
 - **ST_Union** - 入力ジオメトリのポイント集合の結合を表現するジオメトリを返します。
 - **ST_Volume** - Computes the volume of a 3D geometry. Closed surfaces can have non-zero volume.
 - **ST_WrapX** - ジオメトリを X 値で回り込ませます。
 - **ST_X** - ポイントの X 値を返します。
 - **ST_XMax** - 2 次元または 3 次元のバウンディングボックスまたはジオメトリの X の最大値を返します。
 - **ST_XMin** - 2 次元または 3 次元のバウンディングボックスまたはジオメトリの X の最小値を返します。
 - **ST_XSize** - Returns the X size of a 2D or 3D bounding box or a geometry.
 - **ST_Y** - ポイントの Y 値を返します。
 - **ST_YMax** - 2 次元または 3 次元のバウンディングボックスまたはジオメトリの Y の最大値を返します。
 - **ST_YMin** - 2 次元または 3 次元のバウンディングボックスまたはジオメトリの Y の最小値を返します。
 - **ST_YSize** - Returns the Y size of a 2D or 3D bounding box or a geometry.
 - **ST_Z** - ポイントの Z 値を返します。
 - **ST_ZMFlag** - ジオメトリの ZM 座標次元を示す符号を返します。
 - **ST_ZMax** - 2 次元または 3 次元のバウンディングボックスまたはジオメトリの Z の最大値を返します。
-

- **ST_ZMin** - 2 次元または 3 次元のバウンディングボックスまたはジオメトリの Z の最小値を返します。
- **ST_ZSize** - Returns the Z size of a 2D or 3D bounding box or a geometry.
- **Equals** - 二つの TopoGeometry が同じトポロジプリミティブで成っている場合に true を返します。
- **Intersects** - 二つの TopoGeometry からのプリミティブの組がインタセクトする場合に true を返します。
- **UpdateGeometrySRID** - ジオメトリカラム内の全ての地物の SRID を更新し、テーブルのメタデータを更新します。
- **&&&** - A の n 次元バウンディングボックスが B の n 次元バウンディングボックスとインタセクトする場合に TRUE を返します。
- **&&&(geometry,gidx)** - ジオメトリの (キャッシュされている)n 次元バウンディングボックスが単精度浮動小数点数による n 次元バウンディングボックス (GIDX) とインタセクトする場合に TRUE を返します。
- **&&&(gidx,geometry)** - 単精度浮動小数点数による n 次元バウンディングボックス (GIDX) がジオメトリの (キャッシュされている)n 次元バウンディングボックスとインタセクトする場合に TRUE を返します。
- **&&&(gidx,gidx)** - 二つの単精度浮動小数点数による n 次元バウンディングボックス (GIDX) が相互にインタセクトする場合に TRUE を返します。

13.9 PostGIS 曲線ジオメトリ対応関数

次に示す関数は、CIRCULARSTRING, CURVEDPOLYGON 等の曲線ジオメトリ型が使える PostGIS 関数です。

- **AddGeometryColumn** - ジオメトリカラムを既存のテーブルに追加します。
- **Box2D** - ジオメトリの 2 次元範囲を表現する BOX2D を返します。
- **Box3D** - ジオメトリの 3 次元範囲を表現する BOX3D を返します。
- **DropGeometryColumn** - ジオメトリカラムを空間テーブルから除去します。
- **GeometryType** - ジオメトリのタイプを文字列で返します。
- **PostGIS_AddBBox** - ジオメトリにバウンディングボックスを追加します。
- **PostGIS_DropBBox** - ジオメトリからバウンディングボックスのキャッシュを削除します。
- **PostGIS_HasBBox** - ジオメトリのバウンディングボックスがキャッシュされている場合には TRUE を返し、他の場合には FALSE を返します。
- **ST_3DExtent** - ジオメトリの 3 次元バウンディングボックスを返す集約関数です。
- **ST_Affine** - ジオメトリに 3 次元アフィン変換を適用します。
- **ST_AsBinary** - ジオメトリ/ジオグラフィの、SRID メタデータを持たない OGC/ISO Well-Known バイナリ (WKB) 表現を返します。
- **ST_AsEWKB** - ジオメトリの、SRID メタデータを持つ Extended Well-Known バイナリ (EWKB) 表現を返します。
- **ST_AsEWKT** - ジオメトリの SRID メタデータが付いた Well-Known Text (WKT) 表現を返します。
- **ST_AsHEXEWKB** - ジオメトリの HEXEWKB 表現を (文字列として) 返します。リトルエンディアン (NDR) またはビッグエンディアン (XDR) のどちらかのエンコーディングを使います。
- **ST_AsSVG** - ジオメトリから SVG パスデータを返します。
- **ST_AsText** - ジオメトリ/ジオグラフィの SRID メタデータのない Well-Known Text (WKT) 表現を返します。

- **ST_ClusterDBSCAN** - 入力ジオメトリごとに DBSCAN アルゴリズムを使ってクラスタ番号を返すウィンドウ関数です。
- **ST_ClusterWithin** - 分離距離でジオメトリのクラスタリングを行う集約関数です。
- **ST_ClusterWithinWin** - 入力ジオメトリごとに分離距離を使ったクラスタリングを行い、クラスタ ID を返すウィンドウ関数です。
- **ST_Collect** - ジオメトリの集合からジオメトリコレクションまたはマルチ系ジオメトリを生成します。
- **ST_CoordDim** - ジオメトリの座標次元を返します。
- **ST_CurveToLine** - 曲線を含むジオメトリを線ジオメトリに変換します。
- **ST_Distance** - 二つのジオメトリ値またはジオグラフィ値間の距離を返します。
- **ST_Dump** - ジオメトリの要素となる geometry_dump 行の集合を返します。
- **ST_DumpPoints** - ジオメトリ内の座標の行である geometry_dump 行の集合を返します。
- **ST_EndM** - Returns the M coordinate of the last point of a LineString, CircularLineString, or NURBS-Curve.
- **ST_EndPoint** - Returns the last point of a LineString, CircularLineString, or NURBS-Curve.
- **ST_EstimatedExtent** - 空間テーブルの推定範囲を返します。
- **ST_FlipCoordinates** - X 値と Y 値を入れ替えたジオメトリを返します。
- **ST_Force2D** - ジオメトリを 2 次元モードに強制します。
- **ST_ForceCurve** - 該当する場合は、ジオメトリを曲線タイプに変換します。
- **ST_ForceSFS** - SFS 1.1 ジオメトリタイプのみ使うようジオメトリに強制します。
- **ST_Force3D** - ジオメトリを XYZ モードに強制します。これは ST_Force3DZ の別名です。
- **ST_Force3DM** - ジオメトリを XYM モードに強制します。
- **ST_Force3DZ** - ジオメトリを XYZ モードに強制します。
- **ST_Force4D** - ジオメトリを XYZM モードに強制します。
- **ST_ForceCollection** - ジオメトリをジオメトリコレクションに変換します。
- **ST_GeoHash** - ジオメトリの GeoHash 表現を返します。
- **ST_GeogFromWKB** - Well-Known Binary ジオメトリ表現 (WKB) または拡張 WKB(EWKB) からジオグラフィインスタンスを生成します。
- **ST_GeomFromEWKB** - 拡張 Well-Known Binary 表現 (EWKB) から指定した ST_Geometry 値を返します。
- **ST_GeomFromEWKT** - 拡張 Well-Known Text 表現 (EWKT) から指定された ST_Geometry 値を返します。
- **ST_GeomFromText** - Well-Known Text 表現 (WKT) から指定した ST_Geometry を返します。
- **ST_GeomFromWKB** - Well-Known Binary ジオメトリ表現 (WKB) と任意パラメーターの SRID からジオメトリインスタンスを生成します。
- **ST_GeometryN** - ジオメトリコレクションの要素を一つ返します。
- **=** - ジオメトリ/ジオグラフィ A の座標と座標の並び順がジオメトリ/ジオグラフィ B と同じ場合に TRUE を返します。
- **&<|** - A のバウンディングボックスが B のバウンディングボックスをオーバーラップするか、B のバウンディングボックスの下にある場合に TRUE を返します。

- **ST_HasArc** - ジオメトリに円弧が含まれているかどうかテストします。
 - **ST_Intersects** - 二つのジオメトリがインタセクトしている (少なくとも一つの共有点がある) かどうかテストします。
 - **ST_IsClosed** - ラインストリングの始点と終点一致しているかをテストします。多面体サーフェスについては閉じているか (立体であるか) をテストします。
 - **ST_IsCollection** - ジオメトリのタイプがジオメトリコレクションかをテストします。
 - **ST_IsEmpty** - ジオメトリが空かをテストします。
 - **ST_LineToCurve** - 曲線を含むジオメトリを線ジオメトリに変換します。
 - **ST_MMax** - Returns the M maxima of a geometry.
 - **ST_MMin** - ジオメトリの M 値の最小値を返します。
 - **ST_MSize** - Returns the M size of a geometry.
 - **ST_MemSize** - ジオメトリが取るメモリ空間の合計を返します。
 - **ST_NPoints** - ジオメトリのポイント (頂点) の数を返します。
 - **ST_NRings** - ポリゴンジオメトリのリング数を返します。
 - **ST_PointFromWKB** - WKB と与えられた SRID からジオメトリを生成します。
 - **ST_PointN** - ジオメトリの最初のラインストリングまたは曲線ストリングの N 番目のポイントを返します。
 - **ST_Points** - ジオメトリの全ての座標を含むマルチポイントを返します。
 - **ST_Rotate** - ジオメトリを原点について回転させます。
 - **ST_RotateZ** - ジオメトリを Z 軸について回転させます。
 - **ST_SRID** - ジオメトリの空間参照系識別子を返します。
 - **ST_Scale** - 与えた係数でジオメトリを拡大縮小します。
 - **ST_SetEndM** - Sets the M coordinate of the last point of a LineString, CircularLineString, or NURBS-Curve.
 - **ST_SetSRID** - ジオメトリに SRID を設定します。
 - **ST_SetStartM** - Sets the M coordinate of the first point of a LineString, CircularLineString, or NURBS-Curve.
 - **ST_StartM** - Returns the M coordinate of the first point of a LineString, CircularLineString, or NURBS-Curve.
 - **ST_StartPoint** - Returns the first point of a LineString, CircularLineString, or NURBS-Curve.
 - **ST_Summary** - ジオメトリについての要約文を返します。
 - **ST_SwapOrdinates** - 与えられたジオメトリにおいて与えられた座標の値を入れ替えたジオメトリを返します。
 - **ST_TransScale** - 与えられた係数とオフセットでジオメトリを変換します。
 - **ST_Transform** - 異なる空間参照系に投影変換された新しいジオメトリを返します。
 - **ST_Translate** - 与えられたオフセットでジオメトリを変換します。
 - **ST_XMax** - 2 次元または 3 次元のバウンディングボックスまたはジオメトリの X の最大値を返します。
 - **ST_XMin** - 2 次元または 3 次元のバウンディングボックスまたはジオメトリの X の最小値を返します。
 - **ST_XSize** - Returns the X size of a 2D or 3D bounding box or a geometry.
-

- **ST_YMax** - 2 次元または 3 次元のバウンディングボックスまたはジオメトリの Y の最大値を返します。
- **ST_YMin** - 2 次元または 3 次元のバウンディングボックスまたはジオメトリの Y の最小値を返します。
- **ST_YSIZE** - Returns the Y size of a 2D or 3D bounding box or a geometry.
- **ST_ZMFlag** - ジオメトリの ZM 座標次元を示す符号を返します。
- **ST_ZMax** - 2 次元または 3 次元のバウンディングボックスまたはジオメトリの Z の最大値を返します。
- **ST_ZMin** - 2 次元または 3 次元のバウンディングボックスまたはジオメトリの Z の最小値を返します。
- **ST_ZSize** - Returns the Z size of a 2D or 3D bounding box or a geometry.
- **UpdateGeometrySRID** - ジオメトリカラム内の全ての地物の SRID を更新し、テーブルのメタデータを更新します。
- **~(box2df,box2df)** - 二つの単精度浮動小数点数による 2 次元バウンディングボックス (BOX2DF) の一方がもう一方を包含する場合に TRUE を返します。
- **~(box2df,geometry)** - 単精度浮動小数点数による 2 次元バウンディングボックス (BOX2DF) をジオメトリの (キャッシュされている)2 次元バウンディングボックスが包含する場合に TRUE を返します。
- **~(geometry,box2df)** - ジオメトリの (キャッシュされている)2 次元バウンディングボックスが単精度浮動小数点数による n 次元バウンディングボックス (GIDX) を包含する場合に TRUE を返します。
- **&&** - A の 2 次元バウンディングボックスが B の 2 次元バウンディングボックスとインタセクトする場合に TRUE を返します。
- **&&&** - A の n 次元バウンディングボックスが B の n 次元バウンディングボックスとインタセクトする場合に TRUE を返します。
- **@(box2df,box2df)** - 二つの単精度浮動小数点数による n 次元バウンディングボックス (GIDX) の一方がもう一方を包含する場合に TRUE を返します。
- **@(box2df,geometry)** - 単精度浮動小数点数による 2 次元バウンディングボックス (BOX2DF) がジオメトリの 2 次元バウンディングボックスに包含される場合に TRUE を返します。
- **@(geometry,box2df)** - ジオメトリの 2 次元バウンディングボックスが単精度浮動小数点数による 2 次元バウンディングボックス (BOX2DF) に包含される場合に TRUE を返します。
- **&&(box2df,box2df)** - 二つの単精度浮動小数点数による 2 次元バウンディングボックス (BOX2DF) が相互にインタセクトする場合に TRUE を返します。
- **&&(box2df,geometry)** - 単精度浮動小数点数による 2 次元バウンディングボックスがジオメトリの (キャッシュされている)2 次元バウンディングボックスとインタセクトする場合に TRUE を返します。
- **&&(geometry,box2df)** - ジオメトリの (キャッシュされている)2 次元バウンディングボックスが単精度浮動小数点数による 2 次元バウンディングボックスとインタセクトする場合に TRUE を返します。
- **&&&(geometry,gidx)** - ジオメトリの (キャッシュされている)n 次元バウンディングボックスが単精度浮動小数点数による n 次元バウンディングボックス (GIDX) とインタセクトする場合に TRUE を返します。
- **&&&(gidx,geometry)** - 単精度浮動小数点数による n 次元バウンディングボックス (GIDX) がジオメトリの (キャッシュされている)n 次元バウンディングボックスとインタセクトする場合に TRUE を返します。
- **&&&(gidx,gidx)** - 二つの単精度浮動小数点数による n 次元バウンディングボックス (GIDX) が相互にインタセクトする場合に TRUE を返します。

13.10 PostGIS 多面体サーフェス対応関数

次に示す関数は、POLYHEDRALSURFACE, POLYHEDRALSURFACEM ジオメトリが使える PostGIS 関数です。

- **Box2D** - ジオメトリの 2 次元範囲を表現する BOX2D を返します。
- **Box3D** - ジオメトリの 3 次元範囲を表現する BOX3D を返します。
- **CG_3DArea** - 3 次元の面ジオメトリの面積を計算します。立体の場合は 0 を返します。
- **CG_3DConvexHull** - ジオメトリの 3 次元の凸包を計算します。
- **CG_3DDifference** - 3 次元の差分を計算します。
- **CG_3DIntersection** - 3 次元のインタセクトした (共有する) 部分を計算します。
- **CG_3DUnion** - postgis_sfcgal を使用して 3 次元結合を実行します。
- **CG_ApproximateMedialAxis** - 面ジオメトリの近似的な中心軸を計算します。
- **CG_Extrude** - 関連するボリュームにサーフェスを押し出します。
- **CG_ForceLHR** - LHR (Left Hand Rule) 方向に強制します。
- **CG_GenerateFlatRoof** - 接地部ポリゴンから平らな (箱の) 屋根を生成します。
- **CG_GenerateGableRoof** - 接地部ポリゴンから切妻屋根を生成します。
- **CG_GenerateHippedRoof** - 接地部ポリゴンから寄棟屋根を生成する。
- **CG_GenerateRoof** - Generate a roof of the requested type from a footprint polygon.
- **CG_GenerateSkillionRoof** - Generate a skillion (single-slope) roof from a footprint polygon.
- **CG_IsPlanar** - サーフェスが平面であるかないかをチェックします。
- **CG_IsSolid** - ジオメトリが立体であるかどうかをテストします。妥当性チェックは行いません。
- **CG_MakeSolid** - ジオメトリを立体にキャストします。チェックはしません。妥当な立体を得るには、入力ジオメトリは閉じた多面体サーフェスか閉じた TIN でなければなりません。
- **CG_PolygonRepair** - Repair an invalid polygon or multipolygon.
- **CG_StraightSkeleton** - ジオメトリからストレートスケルトンを計算します。
- **CG_Tessellate** - ポリゴンまたは多面体サーフェスのテッセレーションを計算し、TIN または TIN コレクションを返します。
- **CG_Visibility** - ポリゴンジオメトリ内のポイント又は辺から可視領域ポリゴンを計算する
- **CG_Volume** - Computes the volume of a 3D geometry. Closed surfaces can have non-zero volume.
- **GeometryType** - ジオメトリのタイプを文字列で返します。
- **ST_3DArea** - 3 次元の面ジオメトリの面積を計算します。立体の場合は 0 を返します。
- **ST_3DClosestPoint** - g1 上の、g2 に最も近い 3 次元ポイントを返します。これは 3 次元の最短ラインの始点です。
- **ST_3DConvexHull** - ジオメトリの 3 次元の凸包を計算します。
- **ST_3DDFullyWithin** - 二つの 3 次元ジオメトリが完全に与えられた 3 次元距離内にあるかどうかをテストします。
- **ST_3DDWithin** - 二つの 3 次元ジオメトリが与えられた 3 次元距離内にあるかどうかをテストします。

- **ST_3DDifference** - 3 次元の差分を計算します。
 - **ST_3DDistance** - 投影座標系の単位で、二つのジオメトリ間の 3 次元デカルト距離の最小値を返します (空間参照系に基づきます)。
 - **ST_3DExtent** - ジオメトリの 3 次元バウンディングボックスを返す集約関数です。
 - **ST_3DIntersection** - 3 次元のインタセクトした (共有する) 部分を計算します。
 - **ST_3DIntersects** - 二つのジオメトリが 3 次元空間において空間的にインタセクトするかどうかをテストします。ポイント、ラインストリング、ポリゴン、多面体サーフェス (面) についてのみ動作します。
 - **ST_3DLongestLine** - 二つのジオメトリ間の 3 次元最長ラインを返します。
 - **ST_3DMaxDistance** - 二つのジオメトリ間の 3 次元最大デカルト距離 (空間参照系に基づく) を空間参照系の単位で返します。
 - **ST_3DShortestLine** - 二つのジオメトリの 3 次元の最短ラインを返します。
 - **ST_3DUnion** - 3 次元の結合を計算します。
 - **ST_Affine** - ジオメトリに 3 次元アフィン変換を適用します。
 - **ST_ApproximateMedialAxis** - 面ジオメトリの近似的な中心軸を計算します。
 - **ST_Area** - ポリゴンジオメトリの面積を返します。
 - **ST_AsBinary** - ジオメトリ/ジオグラフィの、SRID メタデータを持たない OGC/ISO Well-Known バイナリ (WKB) 表現を返します。
 - **ST_AsEWKB** - ジオメトリの、SRID メタデータを持つ Extended Well-Known バイナリ (EWKB) 表現を返します。
 - **ST_AsEWKT** - ジオメトリの SRID メタデータが付いた Well-Known Text (WKT) 表現を返します。
 - **ST_AsGML** - GML 第 2 版または第 3 版としてジオメトリを返します。
 - **ST_AsX3D** - ジオメトリを X3D ノード要素書式 (ISO-IEC-19776-1.2-X3DEncodings-XML) で返します。
 - **ST_CoordDim** - ジオメトリの座標次元を返します。
 - **ST_Dimension** - ST_Geometry 値の座標次元を返します。
 - **ST_Dump** - ジオメトリの要素となる geometry_dump 行の集合を返します。
 - **ST_DumpPoints** - ジオメトリ内の座標の行である geometry_dump 行の集合を返します。
 - **ST_Expand** - 他のバウンディングボックスまたはジオメトリから拡張されたバウンディングボックスを返します。
 - **ST_Extent** - ジオメトリのバウンディングボックスを返す集約関数です。
 - **ST_Extrude** - 関連するポリウムにサーフェスを押し出します。
 - **ST_FlipCoordinates** - X 値と Y 値を入れ替えたジオメトリを返します。
 - **ST_Force2D** - ジオメトリを 2 次元モードに強制します。
 - **ST_ForceLHR** - LHR (Left Hand Rule) 方向に強制します。
 - **ST_ForceRHR** - ポリゴンの頂点の方向を右回りに強制します。
 - **ST_ForceSFS** - SFS 1.1 ジオメトリタイプのみ使うようジオメトリに強制します。
 - **ST_Force3D** - ジオメトリを XYZ モードに強制します。これは ST_Force3DZ の別名です。
 - **ST_Force3DZ** - ジオメトリを XYZ モードに強制します。
-

- **ST_ForceCollection** - ジオメトリをジオメトリコレクションに変換します。
 - **ST_GeomFromEWKB** - 拡張 Well-Known Binary 表現 (EWKB) から指定した ST_Geometry 値を返します。
 - **ST_GeomFromEWKT** - 拡張 Well-Known Text 表現 (EWKT) から指定された ST_Geometry 値を返します。
 - **ST_GeomFromGML** - GML 表現から PostGIS ジオメトリオブジェクトを出力します。
 - **ST_GeometryN** - ジオメトリコレクションの要素を一つ返します。
 - **ST_GeometryType** - ジオメトリの SQL-MM 型を文字列で返します。
 - **=** - ジオメトリ/ジオグラフィ A の座標と座標の並び順がジオメトリ/ジオグラフィ B と同じ場合に TRUE を返します。
 - **&<|** - A のバウンディングボックスが B のバウンディングボックスをオーバーラップするか、B のバウンディングボックスの下にある場合に TRUE を返します。
 - **~=** - A のバウンディングボックスが B のバウンディングボックスと同じ場合に TRUE を返します。
 - **ST_IsClosed** - ラインストリングの始点と終点が一致しているかをテストします。多面体サーフェスについては閉じているか (立体であるか) をテストします。
 - **ST_IsPlanar** - サーフェスが平面であるかないかをチェックします。
 - **ST_IsSolid** - ジオメトリが立体であるかどうかをテストします。妥当性チェックは行いません。
 - **ST_MakeSolid** - ジオメトリを立体にキャストします。チェックはしません。妥当な立体を得るには、入力ジオメトリは閉じた多面体サーフェスか閉じた TIN でなければなりません。
 - **ST_MemSize** - ジオメトリが取るメモリ空間の合計を返します。
 - **ST_NPoints** - ジオメトリのポイント (頂点) の数を返します。
 - **ST_NumGeometries** - ジオメトリコレクションの要素数を返します。
 - **ST_NumPatches** - 多面体サーフェスのフェイス数を返します。多面体でないジオメトリの場合には NULL を返します。
 - **ST_PatchN** - 多面体サーフェスの N 番目のジオメトリ (フェイス) を返します。
 - **ST_RemoveRepeatedPoints** - 重複ポイントを除いたジオメトリを返します。
 - **ST_Reverse** - 頂点の順序を逆にしたジオメトリを返します。
 - **ST_Rotate** - ジオメトリを原点について回転させます。
 - **ST_RotateX** - ジオメトリを X 軸について回転させます。
 - **ST_RotateY** - ジオメトリを Y 軸について回転させます。
 - **ST_RotateZ** - ジオメトリを Z 軸について回転させます。
 - **ST_Scale** - 与えた係数でジオメトリを拡大縮小します。
 - **ST_ShiftLongitude** - 経度座標値を-180 度から 180 度の範囲と 0 度から 360 度の範囲との二つの範囲を行き来するようシフトします。
 - **ST_StraightSkeleton** - ジオメトリからストレートスケルトンを計算します。
 - **ST_Summary** - ジオメトリについての要約文を返します。
 - **ST_SwapOrdinates** - 与えられたジオメトリにおいて与えられた座標の値を入れ替えたジオメトリを返します。
 - **ST_Tessellate** - ポリゴンまたは多面体サーフェスのテッセレーションを計算し、TIN または TIN コレクションを返します。
-

- **ST_Transform** - 異なる空間参照系に投影変換された新しいジオメトリを返します。
- **ST_Volume** - Computes the volume of a 3D geometry. Closed surfaces can have non-zero volume.
- **~(box2df,box2df)** - 二つの単精度浮動小数点数による 2 次元バウンディングボックス (BOX2DF) の一方がもう一方を包含する場合に TRUE を返します。
- **~(box2df,geometry)** - 単精度浮動小数点数による 2 次元バウンディングボックス (BOX2DF) をジオメトリの (キャッシュされている)2 次元バウンディングボックスが包含する場合に TRUE を返します。
- **~(geometry,box2df)** - ジオメトリの (キャッシュされている)2 次元バウンディングボックスが単精度浮動小数点数による n 次元バウンディングボックス (GIDX) を包含する場合に TRUE を返します。
- **&&** - A の 2 次元バウンディングボックスが B の 2 次元バウンディングボックスとインタセクトする場合に TRUE を返します。
- **&&&** - A の n 次元バウンディングボックスが B の n 次元バウンディングボックスとインタセクトする場合に TRUE を返します。
- **@(box2df,box2df)** - 二つの単精度浮動小数点数による n 次元バウンディングボックス (GIDX) の一方がもう一方を包含する場合に TRUE を返します。
- **@(box2df,geometry)** - 単精度浮動小数点数による 2 次元バウンディングボックス (BOX2DF) がジオメトリの 2 次元バウンディングボックスに包含される場合に TRUE を返します。
- **@(geometry,box2df)** - ジオメトリの 2 次元バウンディングボックスが単精度浮動小数点数による 2 次元バウンディングボックス (BOX2DF) に包含される場合に TRUE を返します。
- **&&(box2df,box2df)** - 二つの単精度浮動小数点数による 2 次元バウンディングボックス (BOX2DF) が相互にインタセクトする場合に TRUE を返します。
- **&&(box2df,geometry)** - 単精度浮動小数点数による 2 次元バウンディングボックスがジオメトリの (キャッシュされている)2 次元バウンディングボックスとインタセクトする場合に TRUE を返します。
- **&&(geometry,box2df)** - ジオメトリの (キャッシュされている)2 次元バウンディングボックスが単精度浮動小数点数による 2 次元バウンディングボックスとインタセクトする場合に TRUE を返します。
- **&&&(geometry,gidx)** - ジオメトリの (キャッシュされている)n 次元バウンディングボックスが単精度浮動小数点数による n 次元バウンディングボックス (GIDX) とインタセクトする場合に TRUE を返します。
- **&&&(gidx,geometry)** - 単精度浮動小数点数による n 次元バウンディングボックス (GIDX) がジオメトリの (キャッシュされている)n 次元バウンディングボックスとインタセクトする場合に TRUE を返します。
- **&&&(gidx,gidx)** - 二つの単精度浮動小数点数による n 次元バウンディングボックス (GIDX) が相互にインタセクトする場合に TRUE を返します。

13.11 PostGIS 関数対応マトリクス

次に示す表は、アルファベット順に並べた PostGIS 空間関数と、動作する空間タイプの種類、対応しようとしている OGC/SQL 準拠を示しています。

-  は、ネイティブで、その型と派生型とに対応しています。
-  は、動作しますが、ジオメトリにキャストして「最善の SRID」に投影変換したうえでジオグラフィに戻す、組み込み変換キャストを使います。大きな面積の領域や、極にある領域については、予期しない結果になることがありますし、浮動小数点数のごみを蓄積することがあります。
-  は、直接的な対応でなく box3d への変換といった他からの自動キャストで動作します。



- は、PostGIS を SFCGAL 対応でコンパイルした場合にのみ利用可能な関数です。
- **geom** - 基本的な 2 次元ジオメトリ (x,y) に対応しています。
- **geog** - 基本的な 2 次元ジオグラフィ (x,y) に対応しています。
- **2.5D** - 3 次元/4 次元 (Z または M 座標を持つ) 空間内の基本的な 2 次元ジオメトリに対応しています。
- **PS** - 多面体サーフェス (Polyhedral Surface) に対応しています
- **T** - 三角形と不規則三角網 (TIN) に対応しています。

関数	geom	geog	2.5D	曲線	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	✓						
ST_MakeNurbstove	✓						
GeometryType	✓		✓	✓		✓	✓
ST_Boundary	✓		✓		✓		
ST_BoundingDiagonal	✓		✓				
ST_CoordDim	✓		✓	✓	✓	✓	✓
ST_Dimension	✓				✓	✓	✓
ST_Dump	✓		✓	✓		✓	✓
ST_DumpPoints	✓		✓	✓		✓	✓

関数	geom	geog	2.5D	曲線	SQL MM	PS	T
ST_DumpSegments	✓		✓				✓
ST_DumpRings	✓		✓				
ST_EndPoint	✓		✓	✓	✓		
ST_Envelope	✓				✓		
ST_ExteriorRing	✓		✓		✓		
ST_GeometryN	✓		✓	✓	✓	✓	✓
ST_GeometryType	✓		✓		✓	✓	
ST_HasArc	✓		✓	✓			
ST_InteriorRingN	✓		✓		✓		
ST_NumCurves	✓		✓		✓		
ST_CurveN	✓		✓		✓		
ST_IsClosed	✓		✓	✓	✓	✓	
ST_IsCollection	✓		✓	✓			
ST_IsEmpty	✓			✓	✓		
ST_IsPolygonCCW	✓		✓				
ST_IsPolygonCW	✓		✓				
ST_IsRing	✓				✓		
ST_IsSimple	✓		✓		✓		
ST_M	✓		✓		✓		
ST_MemSize	✓		✓	✓		✓	✓
ST_NDims	✓		✓				
ST_NPoints	✓		✓	✓		✓	
ST_NRings	✓		✓	✓			
ST_NumGeometries	✓		✓		✓	✓	✓
ST_NumInteriorRings	✓				✓		
ST_NumInteriorRing	✓						
ST_NumPatches	✓		✓		✓	✓	
ST_NumPoints	✓				✓		
ST_PatchN	✓		✓		✓	✓	
ST_PointN	✓		✓	✓	✓		
ST_Points	✓		✓	✓			
ST_StartPoint	✓		✓	✓	✓		
ST_StartM	✓			✓			
ST_EndM	✓			✓			

関数	geom	geog	2.5D	曲線	SQL MM	PS	T
ST_SetStartM	✓			✓			
ST_SetEndM	✓			✓			
ST_Summary	✓	✓		✓		✓	✓
ST_X	✓		✓		✓		
ST_Y	✓		✓		✓		
ST_Z	✓		✓		✓		
ST_ZMFlag	✓		✓	✓			
ST_HasZ	✓		✓				
ST_HasM	✓		✓				
ST_ControlPoint	✓						
ST_Degree	✓						
ST_Weights	✓						
ST_Knots	✓						
ST_NumControlPoints	✓						
ST_NurbsCurve	✓	ational					
ST_IsNurbsCurve	✓						
ST_Evaluate	✓						
ST_NurbsToLine	✓	ring					
ST_AddPoint	✓		✓				
ST_CollectionExtract	✓	ct					
ST_CollectionHomogenize	✓	ogenize					
ST_CurveToLine	✓		✓	✓	✓		
ST_Scroll	✓		✓				
ST_FlipCoordinates	✓	s	✓	✓		✓	✓
ST_Force2D	✓		✓	✓		✓	
ST_Force3D	✓		✓	✓		✓	
ST_Force3DZ	✓		✓	✓		✓	
ST_Force3DM	✓			✓			
ST_Force4D	✓		✓	✓			
ST_ForceCollection	✓	i	✓	✓		✓	
ST_ForceCurve	✓		✓	✓			
ST_ForcePolygonCW	✓	CW	✓				
ST_ForcePolygonCCW	✓	W	✓				
ST_ForceSFS	✓		✓	✓		✓	✓

関数	geom	geog	2.5D	曲線	SQL MM	PS	T
ST_ForceRHR	✓		✓			✓	
ST_LineExtend	✓						
ST_LineToCurve	✓		✓	✓			
ST_Multi	✓						
ST_Normalize	✓						
ST_Project	✓	✓					
ST_QuantizeCoordinates	✓						
ST_RemovePoint	✓		✓				
ST_RemoveRepeatedPoints	✓		✓			✓	
ST_RemoveIrrelevantPointsForView	✓						
ST_RemoveSmallParts	✓						
ST_Reverse	✓		✓			✓	
ST_Segmentize	✓	✓					
ST_SetPoint	✓		✓				
ST_ShiftLongitude	✓		✓			✓	✓
ST_WrapX	✓		✓				
ST_SnapToGrid	✓		✓				
ST_Snap	✓						
ST_SwapOrdinate	✓		✓	✓		✓	✓
ST_IsValid	✓				✓		
ST_IsValidDetail	✓						
ST_IsValidReason	✓						
ST_MakeValid	✓		✓				
ST_InverseTransformPipeline	✓						
ST_SetSRID	✓			✓			
ST_SRID	✓			✓	✓		
ST_Transform	✓			✓	✓	✓	
ST_TransformPipeline	✓						
postgis_srs_codes							
postgis_srs							
postgis_srs_all							
postgis_srs_search	✓						
ST_BdPolyFromText	✓						
ST_BdMPolyFromText	✓						
ST_GeogFromText		✓					

関数	geom	geog	2.5D	曲線	SQL MM	PS	T
ST_GeographyFromText	✓	✓					
ST_GeomCollFromText	✓				✓		
ST_GeomFromEWKT	✓		✓	✓		✓	✓
ST_GeomFromMVC21	✓						
ST_GeometryFromText	✓				✓		
ST_GeomFromInterpolated	✓			✓	✓		
ST_LineFromText	✓				✓		
ST_MLineFromText	✓				✓		
ST_MPointFromText	✓				✓		
ST_MPolyFromText	✓				✓		
ST_PointFromText	✓				✓		
ST_PolygonFromText	✓				✓		
ST_WKTToSQL	✓				✓		
ST_GeogFromWKB		✓		✓			
ST_GeomFromEWKB	✓		✓	✓		✓	✓
ST_GeomFromV3	✓			✓	✓		
ST_LineFromWKB	✓				✓		
ST_LineStringFromWKB	✓				✓		
ST_PointFromWKB	✓		✓	✓	✓		
ST_WKBToSQL	✓				✓		
ST_Box2dFromGeoHash	✓						
ST_GeomFromGeoHash	✓						
ST_GeomFromGeoJSON	✓		✓			✓	✓
ST_GeomFromGeoJSON	✓		✓				
ST_GeomFromKML	✓		✓				
ST_GeomFromInterpolated	✓						
ST_GMLToSQL	✓				✓		
ST_LineFromEncodedPolyline	✓						
ST_PointFromGeoHash							
ST_FromFlatGeobufToTable							
ST_FromFlatGeobuf							
ST_AsEWKT	✓	✓	✓	✓		✓	✓
ST_AsText	✓	✓		✓	✓		
ST_AsBinary	✓	✓	✓	✓	✓	✓	✓
ST_AsEWKB	✓		✓	✓		✓	✓

関数	geom	geog	2.5D	曲線	SQL MM	PS	T
ST_AsHEXEWKB	✓		✓	✓			
ST_AsEncodedPolyline	✓ line						
ST_AsFlatGeobuf	✗						
ST_AsGeobuf	✗						
ST_AsGeoJSON	✓	✓	✓				
ST_AsGML	✓	✓	✓		✓	✓	✓
ST_AsKML	✓	✓	✓				
ST_AsLatLonText	✓						
ST_AsMARC21	✓						
ST_AsMVTGeom	✓						
ST_AsMVT	✗						
ST_AsSVG	✓	✓		✓			
ST_AsTWKB	✓						
ST_AsX3D	✓		✓			✓	✓
ST_GeoHash	✓			✓			
&&	✓	✓		✓		✓	
&&(geometry,box2df)	✓			✓		✓	
&&(box2df,geometry)	✓			✓		✓	
&&(box2df,box2df)	✗			✓		✓	
&&&	✓		✓	✓		✓	✓
&&&(geometry,box2x)	✓		✓	✓		✓	✓
&&&(gidx,geometry)	✓		✓	✓		✓	✓
&&&(gidx,gidx)			✓	✓		✓	✓
&<	✓						
&<	✓			✓		✓	
&>	✓						
<<	✓						
<<	✓						
=	✓	✓		✓		✓	
>>	✓						
@	✓						
@(geometry,box2df)	✓			✓		✓	
@(box2df,geometry)	✓			✓		✓	
@(box2df,box2df)	✗			✓		✓	

関数	geom	geog	2.5D	曲線	SQL MM	PS	T
&>	✓						
>>	✓						
~	✓						
~(geometry,box2df)	✓			✓		✓	
~(box2df,geometry)	✓			✓		✓	
~(box2df,box2df)	✓			✓		✓	
~=	✓					✓	
<->	✓	✓					
=	✓						
<#>	✓						
<<->>	✓						
ST_3DIntersects	✓		✓		✓	✓	✓
ST_Contains	✓				✓		
ST_ContainsProperly	✓						
ST_CoveredBy	✓	✓					
ST_Covers	✓	✓					
ST_Crosses	✓				✓		
ST_Disjoint	✓				✓		
ST_Equals	✓				✓		
ST_Intersects	✓	✓		✓	✓		✓
ST_LineCrossingDirection	✓						
ST_OrderingEquivalent	✓				✓		
ST_Overlaps	✓				✓		
ST_Relate	✓				✓		
ST_RelateMatch							
ST_Touches	✓				✓		
ST_Within	✓				✓		
ST_3DDWithin	✓		✓		✓	✓	
ST_3DDFullyWithin	✓		✓			✓	
ST_DFullyWithin	✓						
ST_DWithin	✓	✓					
ST_PointInsideCircle	✓						
ST_Area	✓	✓			✓	✓	
ST_Azimuth	✓	✓					

関数	geom	geog	2.5D	曲線	SQL MM	PS	T
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_LengthSphere	✓		✓				
ST_LongestLine	✓						
ST_3DLongestLine	✓		✓			✓	
ST_MaxDistance	✓						
ST_3DMaxDistance	✓		✓			✓	
ST_MinimumClearance	✓						
ST_MinimumClearanceLine	✓						
ST_Perimeter	✓	✓			✓		
ST_Perimeter2D	✓						
ST_3DPerimeter	✓		✓		✓		
ST_ShortestLine	✓	✓					
ST_3DShortestLine	✓		✓			✓	
ST_ClipByBox2D	✓						
ST_Difference	✓		✓		✓		
ST_Intersection	✓	😬	✓		✓		
ST_MemUnion	✓		✓				
ST_Node	✓		✓				
ST_Split	✓						
ST_Subdivide	✓						
ST_SymDifference	✓		✓		✓		
ST_UnaryUnion	✓		✓				
ST_Union	✓		✓		✓		

関数	geom	geog	2.5D	曲線	SQL MM	PS	T
ST_Buffer	✓	😄			✓		
ST_BuildArea	✓						
ST_Centroid	✓	✓			✓		
ST_ChaikinSmoothing	✓		✓				
ST_CatmullRomSmoothing	✓		✓				
ST_ConcaveHull	✓						
ST_ConvexHull	✓		✓		✓		
ST_DelaunayTriangles	✓		✓				✓
ST_FilterByM	✓						
ST_GeneratePoints	✓						
ST_GeometricMean	✓		✓				
ST_LineMerge	✓						
ST_MaximumInscribedCircle	✓						
ST_LargestEmptyCircle	✓						
ST_MinimumBoundingCircle	✓						
ST_MinimumBoundingRadius	✓						
ST_OrientedEnvelope	✓						
ST_OffsetCurve	✓						
ST_PointOnSurface	✓		✓		✓		
ST_Polygonize	✓						
ST_ReducePrecision	✓						
ST_SharedPaths	✓						
ST_Simplify	✓						
ST_SimplifyPreserveTopology	✓						
ST_SimplifyPolygonHull	✓						
ST_SimplifyVW	✓						
ST_SetEffectiveArea	✓						
ST_TriangulatePolygon	✓						
ST_VoronoiLines	✓						
ST_VoronoiPolygons	✓						
ST_CoverageIntersectionEdges	✓						
ST_CoverageClean	✓						
ST_CoverageSimplify	✓						
ST_CoverageEdge	✓						

関数	geom	geog	2.5D	曲線	SQL MM	PS	T
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_ClusterRelatWin	✓						
ST_MinimumSpanningTree	✓						
ST_ClusterKMeans	✓						
ST_ClusterWithin	✓			✓			
ST_ClusterWithin/in	✓			✓			
Box2D	✓			✓		✓	✓
Box3D	✓		✓	✓		✓	✓
ST_EstimatedExtent	✗			✓			
ST_Expand	✓					✓	✓
ST_Extent	✓					✓	✓
ST_3DExtent	✓		✓	✓		✓	✓
ST_MakeBox2D	✓						
ST_3DMakeBox	✓						
ST_XMax	✗		✓	✓			
ST_XMin	✗		✓	✓			
ST_YMax	✗		✓	✓			
ST_YMin	✗		✓	✓			
ST_ZMax	✗		✓	✓			
ST_ZMin	✗		✓	✓			
ST_XSize	✗		✓	✓			
ST_YSize	✗		✓	✓			
ST_ZSize	✗		✓	✓			

関数	geom	geog	2.5D	曲線	SQL MM	PS	T
ST_MMin	✓		✓	✓			
ST_MMax	✓		✓	✓			
ST_MSize	✓		✓	✓			
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_3DInterpola	✓ oint		✓				
ST_AddMeasure	✓		✓				
ST_IsValidTraje	✓ ry		✓				
ST_ClosestPoint	✓ Approach		✓				
ST_DistanceCPA	✓		✓				
ST_CPAWithin	✓		✓				
postgis.gdal_datapath							
postgis.gdal_enabled_drivers							
postgis.enable_outdb_rasters							
postgis.gdal_vsi_options							
postgis.gdal_cpl_debug							
PostGIS_AddBB	✓			✓			
PostGIS_DropBB	✓			✓			
PostGIS_HasBB	✓			✓			
postgis_sfcgal_version							
postgis_sfcgal_full_version							
CG_ForceLHR							
CG_IsPlanar							
CG_IsSolid							
CG_MakeSolid							
CG_Orientation							
CG_Area							

関数	geom	geog	2.5D	曲線	SQL MM	PS	T
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_3DDifferen							
CG_3DDifferen							
CG_Distance							
CG_3DDistance							
ST_3DConvexH							
CG_3DConvexH							
ST_3DIntersec							
CG_3DIntersec							
CG_Union							
ST_3DUnion							
CG_3DUnion							

関数	geom	geog	2.5D	曲線	SQL MM	PS	T
ST_AlphaShape	✓						
CG_AlphaShap							
CG_ApproxCon		artition					
ST_Approximat		dialAxis					
CG_Approxima		dialAxis					
CG_GenerateFl		of					
CG_GenerateH		dRoof					
CG_GenerateG		Roof					
CG_GenerateSl		nRoof					
CG_GenerateR							
CG_PolygonRej							
ST_Constrained		aunayTriangles					
CG_Constraine		aunayTriangles					
ST_Extrude							
CG_Extrude							
CG_ExtrudeStr		tSkeleton					
CG_GreeneApp		onvexPartition					
ST_Minkowski							
CG_Minkowski							
ST_OptimalAlp		ape					
CG_OptimalAlp		ape					
CG_OptimalCo		Partition					
CG_StraightSk		n					
ST_StraightSke		n					

関数	geom	geog	2.5D	曲線	SQL MM	PS	T
ST_Tessellate							
CG_Tessellate							
CG_Triangulate							
CG_Visibility							
CG_YMonotone		ition					
CG_StraightSk		nPartition					
CG_3DBuffer							
CG_Rotate							
CG_2DRotate							
CG_3DRotate							
CG_RotateX							
CG_RotateY							
CG_RotateZ							
CG_Scale							
CG_3DScale							
CG_3DScaleAr		Center					
CG_Translate							
CG_3DTranslat							
CG_Simplify							
CG_3DAlphaW		ng					
CG_NurbsCurv		erpolate					
CG_NurbsCurv		roximate					
CG_NurbsCurv		ivative					
getfaceedges_returntype							
TopoGeometry							
validatetopology_returntype							

関数	geom	geog	2.5D	曲線	SQL MM	PS	T
TopoElement							
TopoElementArray							
AddTopoGeometryColumn							
RenameTopoGeometryColumn							
DropTopology							
RenameTopology							
DropTopoGeometryColumn							
FixCorruptTopoGeometryColumn							
Populate_Topology_Layer							
TopologySummary							
ValidateTopolog✓							
ValidateTopologyRelation							
ValidateTopolog✓recision							
MakeTopologyP✓ise							
FindTopology ✓							
FindLayer ✓							
TotalTopologySize							
UpgradeTopology							
CreateTopology							
CopyTopology							
ST_InitTopoGeo					✓		
ST_CreateTopoC✓					✓		
TopoGeo_AddPc✓							
TopoGeo_AddLi✓string							
TopoGeo_AddPc✓on							
TopoGeo_LoadC✓metry							
ST_AddIsoNode✓					✓		
ST_AddIsoEdge✓					✓		
ST_AddEdgeNe✓aces					✓		
ST_AddEdgeMo✓ace					✓		
ST_RemEdgeNewFace					✓		
ST_RemEdgeModFace					✓		
ST_ChangeEdge✓om					✓		
ST_ModEdgeSp✓					✓		
ST_ModEdgeHeal					✓		
ST_NewEdgeHeal					✓		
ST_MoveIsoNoc✓					✓		
ST_NewEdgesS✓					✓		
ST_RemoveIsoNode					✓		
ST_RemoveIsoEdge					✓		

関数	geom	geog	2.5D	曲線	SQL MM	PS	T
GetEdgeByPoint	✓						
GetFaceByPoint	✓						
GetFaceContains	✓ Point						
GetNodeByPoint	✓						
GetTopologyID							
GetTopologySRID							
GetTopologyName							
ST_GetFaceEdges					✓		
ST_GetFaceGeometry	✓ try				✓		
GetRingEdges							
GetNodeEdges							
FindVertexSegmentPairsBelowDistance							
Polygonize							
AddNode	✓						
AddEdge	✓						
AddFace	✓						
ST_Simplify	✓						
RemoveUnusedNodes	✓ natives						
CreateTopology	✓						
toTopoGeom	✓						
TopoElementArray_Agg							
TopoElement	✓						
clearTopoGeom	✓						
TopoGeom_addLineament	✓						
TopoGeom_removeLineament	✓						
TopoGeom_addTopoGeom	✓						
toTopoGeom							
GetTopoGeomElementArray							
GetTopoGeomElements							
ST_SRID	✓				✓		
AsGML	✓						
AsTopoJSON	✓						
Equals	✓		✓				
Intersects	✓		✓				
geomval							
addbandarg							
rastbandarg							
raster							
reclassarg							
summarystats							
unionarg							

関数	geom	geog	2.5D	曲線	SQL MM	PS	T
AddRasterConstraints							
DropRasterConstraints							
AddOverviewConstraints							
DropOverviewConstraints							
PostGIS_GDAL_Version							
PostGIS_Raster_Lib_Build_Date							
PostGIS_Raster_Lib_Version							
ST_GDALDrivers							
UpdateRasterSRID							
ST_CreateOverview							
ST_AddBand							
ST_AsRaster	✓						
ST_AsRasterAgg	✓						
ST_Band							
ST_MakeEmptyCoverage							
ST_MakeEmptyRaster							
ST_Tile							
ST_Retile	✓						
ST_FromGDALRaster							
ST_GeoReference							
ST_Height							
ST_IsEmpty							
ST_MemSize							
ST_MetaData							
ST_NumBands							
ST_PixelHeight							
ST_PixelWidth							
ST_ScaleX							
ST_ScaleY							
ST_RasterToWorldCoord							
ST_RasterToWorldCoordX							
ST_RasterToWorldCoordY							
ST_Rotation							
ST_SkewX							
ST_SkewY							
ST_SRID							
ST_Summary							
ST_UpperLeftX							
ST_UpperLeftY							
ST_Width							
ST_WorldToRasterCoord	✓						
ST_WorldToRasterCoordX	✓						
ST_WorldToRasterCoordY	✓						
ST_BandMetaData							
ST_BandNoDataValue							
ST_BandIsNoData							
ST_BandPath							
ST_BandFileSize							
ST_BandFileTimestamp							
ST_BandPixelType							
ST_MinPossibleValue							
ST_HasNoBand							

関数	geom	geog	2.5D	曲線	SQL MM	PS	T
ST_PixelAsPolygon	✓						
ST_PixelAsPolygons							
ST_PixelAsPoint	✓						
ST_PixelAsPoints							
ST_PixelAsCentroid	✓ 1						
ST_PixelAsCentroids							
ST_Value	✓						
ST_NearestValue	✓						
ST_SetZ	✓						
ST_SetM	✓						
ST_Neighborhood	✓						
ST_SetValue	✓						
ST_SetValues							
ST_DumpValues							
ST_PixelOfValue							
ST_SetGeoReference							
ST_SetRotation							
ST_SetScale							
ST_SetSkew							
ST_SetSRID							
ST_SetUpperLeft							
ST_Resample							
ST_Rescale							
ST_Reskew							
ST_SnapToGrid							
ST_Resize							
ST_Transform							
ST_SetBandNoDataValue							
ST_SetBandIsNoData							
ST_SetBandPath							
ST_SetBandIndex							
ST_Count							
ST_CountAgg							
ST_Histogram							
ST_Quantile							
ST_SummaryStats							
ST_SummaryStatsAgg							
ST_ValueCount							
ST_RastFromWKB							
ST_RastFromHexWKB							
ST_AsBinary/ST_AsWKB							
ST_AsHexWKB							
ST_AsGDALRaster							
ST_AsJPEG							
ST_AsPNG							
ST_AsTIFF							
ST_Clip	✓						
ST_ColorMap							
ST_Grayscale							

関数	geom	geog	2.5D	曲線	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_ReclassExact							
ST_Union							
ST_Distinct4ma							
ST_InvDistWeight4ma							
ST_Max4ma							
ST_Mean4ma							
ST_Min4ma							
ST_MinDist4ma							
ST_Range4ma							
ST_StdDev4ma							
ST_Sum4ma							
ST_Aspect							
ST_HillShade							
ST_Roughness							
ST_Slope							
ST_TPI							
ST_TRI							
ST_InterpolateF	✓er						
ST_Contour							
Box3D							
ST_ConvexHull	✓						
ST_DumpAsPolygons							
ST_Envelope	✓						
ST_MinConvexH	✓						
ST_Polygon	✓						
ST_Intersection	✓ctions						
&&	✓						
&<							
&>							
=							
@	✓						
~=							
~	✓						
ST_Contains							

関数	geom	geog	2.5D	曲線	SQL MM	PS	T
ST_ContainsProperly							
ST_Covers							
ST_CoveredBy							
ST_Disjoint							
ST_Intersects ✓							
ST_Overlaps							
ST_Touches							
ST_SameAlignment							
ST_NotSameAlignmentReason							
ST_Within							
ST_DWithin							
ST_DFullyWithin							

13.12 新規作成/機能強化/変更された PostGIS 関数

13.12.1 PostGIS 3.7 で新規作成/機能強化された関数

次に示す関数は、新規作成または機能強化された PostGIS 関数です。

PostGIS 3.7 で新規作成された関数

- **CG_ApproximateMedialAxis** - Availability: 3.7.0 - projected パラメーター。投影された結果を得るには SFCGAL >= 2.3.0 が必要。古いバージョンでは通知が出力され、非投影の結果になります。面ジオメトリの近似的な中心軸を計算します。
- **CG_GenerateFlatRoof** - Availability: 3.7.0 - SFCGAL >= 2.3.0 が必要。接地部ポリゴンから平らな (箱の) 屋根を生成します。
- **CG_GenerateGableRoof** - Availability: 3.7.0 - SFCGAL >= 2.3.0 が必要。接地部ポリゴンから切妻屋根を生成します。
- **CG_GenerateHippedRoof** - Availability: 3.7.0 - SFCGAL >= 2.3.0 が必要。接地部ポリゴンから寄棟屋根を生成する。
- **CG_GenerateRoof** - Availability: 3.7.0 - SFCGAL >= 2.3.0 が必要。Generate a roof of the requested type from a footprint polygon.
- **CG_GenerateSkillionRoof** - Availability: 3.7.0 - SFCGAL >= 2.3.0 が必要。Generate a skillion (single-slope) roof from a footprint polygon.
- **CG_NurbsCurveApproximate** - Availability: 3.7.0 - SFCGAL >= 2.3.0 が必要。Creates an approximating NURBS curve fitting data points within a tolerance
- **CG_NurbsCurveDerivative** - Availability: 3.7.0 - SFCGAL >= 2.3.0 が必要。Computes the derivative (tangent, curvature) of a NURBS curve at a given parameter
- **CG_NurbsCurveInterpolate** - Availability: 3.7.0 - SFCGAL >= 2.3.0 が必要。Creates an interpolating NURBS curve passing through all given data points
- **CG_PolygonRepair** - Availability: 3.7.0 - requires SFCGAL >= 2.3.0 and CGAL >= 6.0. Repair an invalid polygon or multipolygon.
- **FindVertexSegmentPairsBelowDistance** - Availability: 3.7 許容距離内にあるトポロジーの頂点/辺の組を検索します。
- **ST_3DInterpolatePoint** - Availability: 3.7.0 3次元で最も近いジオメトリの補間 M 値を返します。

- **ST_ClusterRelateWin** - Availability: 3.7.0 ジオメトリが接続するかどうかを判定する空間関係パターンを使って、入力ジオメトリを接続されたジオメトリ集合にクラスタリングして、クラスター識別子を返すウィンドウ関数です。
- **ST_ControlPoints** - Availability: 3.7.0 Returns the control points of a NURBS curve as a MULTIPOINT geometry.
- **ST_CoverageEdges** - Availability: 3.7.0 ポリゴンカバレッジの重複の無いエッジを計算します。
- **ST_Degree** - Availability: 3.7.0 Returns the polynomial degree of a NURBS curve.
- **ST_Evaluate** - Availability: 3.7.0 Evaluates a NURBS curve at a specific parameter value and returns the resulting point.
- **ST_IsNurbsCurve** - Availability: 3.7.0 Checks if a geometry is a NURBS curve.
- **ST_Knots** - Availability: 3.7.0 Returns the knot vector of a NURBS curve.
- **ST_MSize** - Availability: 3.7.0 Returns the M size of a geometry.
- **ST_MakeLine** - Availability: 3.7.0 - マルチラインストリング入力要素への対応が導入されました POINT、MULTIPOINT、LINESTRING から LINESTRING を生成します。
- **ST_MakeNurbsCurve** - Availability: 3.7.0 Creates a NURBS (Non-Uniform Rational B-Spline) curve from control points, optional weights, and optional knot vector.
- **ST_MinimumSpanningTree** - Availability: 3.7.0 最小全域木アルゴリズムを使って入力ジオメトリを接続された木にクラスタリングを行い、入力ジオメトリごとに木 ID を返すウィンドウ関数です。
- **ST_NumControlPoints** - Availability: 3.7.0 Returns the number of control points in a NURBS curve.
- **ST_NurbsCurveIsRational** - Availability: 3.7.0 Checks if a NURBS curve is rational (has weights).
- **ST_NurbsToLineString** - Availability: 3.7.0 Converts a NURBS curve to a LINESTRING by uniform sampling.
- **ST_Weights** - Availability: 3.7.0 Returns the weight array of a rational NURBS curve.
- **ST_XSize** - Availability: 3.7.0 Returns the X size of a 2D or 3D bounding box or a geometry.
- **ST_YSize** - Availability: 3.7.0 Returns the Y size of a 2D or 3D bounding box or a geometry.
- **ST_ZSize** - Availability: 3.7.0 Returns the Z size of a 2D or 3D bounding box or a geometry.

PostGIS 3.7 で機能強化された関数

- **ST_LongestLine** - Enhanced: 3.7.0 - support for a single geometry input. 二つのジオメトリ間の 2 次元最長ラインを返します。
- **ST_MakePolygon** - Enhanced: 3.7.0 - Support for curved input rings was introduced. Creates a Polygon or CurvePolygon from a shell and optional list of holes.
- **ST_MaxDistance** - Enhanced: 3.7.0 - support for a single geometry input. 二つのジオメトリ間の 2 次元最長距離を空間参照系の単位で返します。
- **ST_Value** - Enhanced: 3.7.0 resample accepts nearest-neighbor boundary options "nearest-UL", "nearest-UR", "nearest-LL", and "nearest-LR". 指定したバンドにおける columnx, rowy で指定したピクセルまたは指定したジオメトリポイントに対応するピクセルの値を返します。バンド番号は 1 始まりで、指定しない場合には、1 番と仮定します。exclude_nodata_value が FALSE に設定された場合には、NODATA ピクセルを含む全てのピクセルがインタセクトするかが考慮され、値を返します。exclude_nodata_value を渡さない場合には、ラスタのメタデータから読みます。
- **TopoGeo_AddLineString** - Enhanced: 3.7.0 トポロジー内で生成されるエッジ数の制限機能の追加。許容差を使って既存のトポロジーにラインストリングを追加し、可能ならエッジ/フェイスを分割します。

PostGIS 3.7 で変更された関数

- **ST_AsGeoJSON** - Changed: 3.7.0 キー重複に関する警告の追加 GeoJSON 形式のジオメトリまたは地物を返します。
- **toTopoGeom** - Changed: 3.7.0 許容値 0 を実際の 0 として扱い、-1 (新デフォルト値) をそのトポロジーの精度を使うよう求めるように変更されました。単純なジオメトリから TopoGeometry を生成します。

13.12.2 PostGIS 3.6 で新規作成/機能強化された関数

次に示す関数は、新規作成または機能強化された PostGIS 関数です。

PostGIS 3.6 で新規作成された関数

- **CG_2DRotate** - Availability: 3.6.0 - SFCGAL $\geq$ 2.0.0 が必要ですジオメトリを指定した 2 次元ポイントを中心に指定した角度で回転させます。
- **CG_3DAlphaWrapping** - Availability: 3.6.0 - SFCGAL $\geq$ 2.1.0 が必要厳密にジオメトリを囲む 3 次元アルファラッピングを計算します。
- **CG_3DBuffer** - Availability: 3.6.0 - SFCGAL $\geq$ 2.0.0 が必要ですジオメトリの周りの 3 次元バッファを計算します。
- **CG_3DRotate** - Availability: 3.6.0 - SFCGAL $\geq$ 2.0.0 が必要ですジオメトリを指定したベクトルを軸に 3 次元空間で回転させます。
- **CG_3DScale** - Availability: 3.6.0 - SFCGAL $\geq$ 2.0.0 が必要ですジオメトリを X,Y,Z 軸に沿って個別の係数で拡大縮小します。
- **CG_3DScaleAroundCenter** - Availability: 3.6.0 - SFCGAL $\geq$ 2.0.0 が必要ですジオメトリを 3 次元空間で指定した点を中心として拡大縮小します。
- **CG_3DTranslate** - Availability: 3.6.0 - SFCGAL $\geq$ 2.0.0 が必要ですジオメトリを与えたオフセットで 3 次元で移動させます。
- **CG_Rotate** - Availability: 3.6.0 - SFCGAL $\geq$ 2.0.0 が必要ですジオメトリを (0,0) を中心に指定した角度で回転させます。
- **CG_RotateX** - Availability: 3.6.0 - SFCGAL $\geq$ 2.0.0 が必要ですジオメトリを X 軸を軸に指定した角度で回転させます。
- **CG_RotateY** - Availability: 3.6.0 - SFCGAL $\geq$ 2.0.0 が必要ですジオメトリを Y 軸を軸に指定した角度で回転させます。
- **CG_RotateZ** - Availability: 3.6.0 - SFCGAL $\geq$ 2.0.0 が必要ですジオメトリを Z 軸を軸に指定した角度で回転させます。
- **CG_Scale** - Availability: 3.6.0 - SFCGAL $\geq$ 2.0.0 が必要ですジオメトリを全ての次元で与えられた係数で均一に拡大縮小します。
- **CG_Simplify** - Availability: 3.6.0 - SFCGAL $\geq$ 2.1.0 が必要主要な地物と Z/M 値を維持しつつジオメトリの複雑さを減らします。
- **CG_StraightSkeletonPartition** - Availability: 3.6.0 - SFCGAL $\geq$ 2.0.0 が必要です。ポリゴンのストレートスケルトン分割を計算します。
- **CG_Translate** - Availability: 3.6.0 - SFCGAL $\geq$ 2.0.0 が必要ですジオメトリを与えたオフセットで 2 次元で移動させます。
- **FixCorruptTopoGeometryColumn** - Availability: 3.6.1 postgis_topology を 3.6.0 以上にアップグレードする際に発生するポジオメトリーの間違いを訂正します。

- **MakeTopologyPrecise** - Availability: 3.6.0 トポロジの頂点を精度グリッドにスナップします。
- **ST_AsRasterAgg** - Availability: 3.6.0 集約関数です。PostGIS ジオメトリを新しいラスタに変換します。
- **ST_CatmullRomSmoothing** - Availability: 3.6.0 Catmull-Rom スプライン曲線を用いて平滑化されたジオメトリーを返します。
- **ST_CoverageClean** - Availability: 3.6.0 - GEOS $\geq 3.14.0$ が必要ですきれいな (エッジが合致し、オーバーラップがなく、隙間がない) ポリゴンカバレッジを、きれいでない入力から計算します。
- **ST_IntersectionFractions** - Availability: 3.6.0 GEOS 3.14 以上が必要です。ラスタ上の個々のセルのうち与えられたジオメトリで覆われたものの断片を計算します。
- **ST_ReclassExact** - Availability: 3.6.0 元ラスターから再分類したバンドからなる新しいラスタを生成します。元ラスターのバンド内の値から新たに生成されるラスターのバンド内の値への 1 対 1 のマッピングを使います。
- **TotalTopologySize** - Availability: 3.6.0 指定したトポロジが使用するディスク容量の総量を得ます。インデックスや TOAST データも含まれます。
- **UpgradeTopology** - Availability: 3.6.0 指定したトポロジーに対して、トポロジー ID とプリミティブ ID が大きな数 (int8) に対応するようアップグレードします。
- **ValidateTopologyPrecision** - Availability: 3.6.0 トポロジ内の低精度の頂点を返します。
- **postgis.gdal_cpl_debug** - Availability: 3.6.0 GDAL デバッグメッセージのログを取るか取らないかを設定する真偽値。

PostGIS 3.6 で機能強化された関数

- **CreateTopology** - Enhanced: 3.6.0 `topoid` と `usesLargeIDs` パラメーターの追加。新しいトポロジスキーマを生成し、`topology.topology` テーブルに登録します。

13.12.3 PostGIS 3.5 で新規作成/機能強化された関数

次に示す関数は、新規作成または機能強化された PostGIS 関数です。

PostGIS 3.5 で新規作成された関数

- **CG_3DArea** - Availability: 3.5.0 3 次元の面ジオメトリの面積を計算します。立体の場合は 0 を返します。
- **CG_3DConvexHull** - Availability: 3.5.0 ジオメトリの 3 次元の凸包を計算します。
- **CG_3DDifference** - Availability: 3.5.0 3 次元の差分を計算します。
- **CG_3DDistance** - Availability: 3.5.0 二つのジオメトリの最小 3 次元距離を返します
- **CG_3DIntersection** - Availability: 3.5.0 3 次元のインタセクトした (共有する) 部分を計算します。
- **CG_3DIntersects** - Availability: 3.5.0 二つの 3 次元ジオメトリがインタセクトするかどうかをテストします
- **CG_3DUnion** - Availability: 3.5.0 `postgis_sfcgal` を使用して 3 次元結合を実行します。
- **CG_AlphaShape** - Availability: 3.5.0 - SFCGAL $\geq 1.4.1$ が必要です。ジオメトリを囲むアルファシェイプを計算します。
- **CG_ApproxConvexPartition** - Availability: 3.5.0 - SFCGAL $\geq 1.5.0$ が必要です。ポリゴンジオメトリの近似凸分割を計算します
- **CG_ApproximateMedialAxis** - Availability: 3.5.0 面ジオメトリの近似的な中心軸を計算します。
- **CG_Area** - Availability: 3.5.0 ジオメトリの面積を計算します
- **CG_Difference** - Availability: 3.5.0 二つのジオメトリの幾何学的な差を計算します

- **CG_Distance** - Availability: 3.5.0 二つのジオメトリの最小距離を返します
- **CG_Extrude** - Availability: 3.5.0 関連するポリュームにサーフェスを押し出します。
- **CG_ExtrudeStraightSkeleton** - Availability: 3.5.0 - SFCGAL $\geq 1.5.0$ が必要です。ストレートスケルトンの押し出し
- **CG_ForceLHR** - Availability: 3.5.0 LHR (Left Hand Rule) 方向に強制します。
- **CG_GreeneApproxConvexPartition** - Availability: 3.5.0 - SFCGAL $\geq 1.5.0$ が必要です。ポリゴンジオメトリの近似凸分割を計算します
- **CG_Intersection** - Availability: 3.5.0 二つのジオメトリのインタセクトする (共有する) 部分を計算します
- **CG_Intersects** - Availability: 3.5.0 二つのジオメトリがインタセクトしている (少なくとも一つの共有点がある) かどうかテストします。
- **CG_IsPlanar** - Availability: 3.5.0 サーフェスが平面であるかないかをチェックします。
- **CG_IsSolid** - Availability: 3.5.0 ジオメトリが立体であるかどうかをテストします。妥当性チェックは行いません。
- **CG_MakeSolid** - Availability: 3.5.0 ジオメトリを立体にキャストします。チェックはしません。妥当な立体を得るには、入力ジオメトリは閉じた多面体サーフェスか閉じた TIN でなければなりません。
- **CG_MinkowskiSum** - Availability: 3.5.0 ミンコフスキー和を求めます。
- **CG_OptimalAlphaShape** - Availability: 3.5.0 - SFCGAL $\geq 1.4.1$ が必要です。「最適」アルファ値を使ってジオメトリを囲むアルファシェイプを計算します。
- **CG_OptimalConvexPartition** - Availability: 3.5.0 - SFCGAL $\geq 1.5.0$ が必要です。ポリゴンジオメトリの最適凸分割を計算します
- **CG_Orientation** - Availability: 3.5.0 サーフェスの方向を判定します。
- **CG_StraightSkeleton** - Availability: 3.5.0 ジオメトリからストレートスケルトンを計算します。
- **CG_Tessellate** - Availability: 3.5.0 ポリゴンまたは多面体サーフェスのテッセレーションを計算し、TIN または TIN コレクションを返します。
- **CG_Triangulate** - Availability: 3.5.0 ポリゴンジオメトリを三角形にします
- **CG_Union** - Availability: 3.5.0 二つのジオメトリの結合を計算します
- **CG_Visibility** - Availability: 3.5.0 - SFCGAL $\geq 1.5.0$ が必要です。ポリゴンジオメトリ内のポイント又は辺から可視領域ポリゴンを計算する
- **CG_Volume** - Availability: 3.5.0 Computes the volume of a 3D geometry. Closed surfaces can have non-zero volume.
- **CG_YMonotonePartition** - Availability: 3.5.0 - SFCGAL $\geq 1.5.0$ が必要です。ポリゴンジオメトリの Y 単調分割を計算します
- **ST_HasM** - Availability: 3.5.0 ジオメトリが M 値を持っているかどうかを確認します。
- **ST_HasZ** - Availability: 3.5.0 ジオメトリが Z 値を持っているかどうかを確認します。
- **ST_RemoveIrrelevantPointsForView** - Availability: 3.5.0 指定した矩形表示範囲での描画に関係のないジオメトリのポイントを削除します。
- **ST_RemoveSmallParts** - Availability: 3.5.0 ジオメトリの全ての小さな部位 (ポリゴンのリングまたはラインストリング) を削除します。
- **TopoGeo_LoadGeometry** - Availability: 3.5.0 ジオメトリを既存のトポロジにロードし、必要に応じてスナップと分割を実行します。

PostGIS 3.5 で機能強化された関数

- **ST_Clip** - Enhanced: 3.5.0 - 引数 `touched` を追加。入力ジオメトリで切り取ったラスタを返します。バンドが指定されていない場合には、全てのバンドが処理されます。`crop` が指定されていなかったり `TRUE` となっている場合には、出力ラスタは切り取られます。`touched` が `TRUE` の場合には、接触するピクセルは取り込まれ、`TRUE` でない場合には、中心がジオメトリ内にあるピクセルだけが取り込まれます。

PostGIS 3.5 で変更された関数

- **ST_AsGeoJSON** - Changed: 3.5.0 地物の `id` を含むカラムを指定できるようになりました `GeoJSON` 形式のジオメトリまたは地物を返します。
- **ST_DFullyWithin** - Changed: 3.5.0 : この関数のロジックとしては、今のところ、バッファ内に包含するかどうかのテストを使っています。 `ST_MaxDistance` アルゴリズムではありません。前の版と結果が異なる可能性があります、ユーザの期待に近づくはずですが。ジオメトリが完全に他のジオメトリの指定距離内にあるかどうかをテストします

13.12.4 PostGIS 3.4 で新規作成/機能強化された関数

次に示す関数は、新規作成または機能強化された PostGIS 関数です。

PostGIS 3.4 で新規作成された関数

- **PostGIS_GEOS_Compiled_Version** - Availability: 3.4.0 PostGIS のビルドに使われた GEOS ライブラリのバージョン番号を返します。
- **PostGIS_PROJ_Compiled_Version** - Availability: 3.5.0 PostGIS のビルドに使われた PROJ ライブラリのバージョン番号を返します。
- **RenameTopoGeometryColumn** - Availability: 3.4.0 TopoGeometry カラムの名前を変更します
- **RenameTopology** - Availability: 3.4.0 トポロジ名を変更します
- **ST_ClusterIntersectingWin** - Availability: 3.4.0 入力ジオメトリごとに接続された集合にクラスタリングを行い、クラスタ ID を返すウィンドウ関数です。
- **ST_ClusterWithinWin** - Availability: 3.4.0 入力ジオメトリごとに分離距離を使ったクラスタリングを行い、クラスタ ID を返すウィンドウ関数です。
- **ST_CoverageInvalidEdges** - Availability: 3.4.0 ポリゴンが妥当なカバレッジの形成に失敗する位置を検索するウィンドウ関数。
- **ST_CoverageSimplify** - Availability: 3.4.0 ポリゴンカバレッジのエッジを単純化するウィンドウ関数。
- **ST_CoverageUnion** - Availability: 3.4.0 - GEOS 3.8.0 以上が必要です共有しているエッジを除去することでカバレッジを形成するポリゴンの集合の結合を計算します。
- **ST_InverseTransformPipeline** - Availability: 3.4.0 定義した座標変換パイプラインの逆変換を使って、異なる空間参照系に座標値を変換した新しいジオメトリを返します。
- **ST_LargestEmptyCircle** - Availability: 3.4.0. ジオメトリとオーバーラップ市内最大の円を計算します。
- **ST_LineExtend** - Availability: 3.4.0 指定距離ぶん前後に延長されたラインを返します。
- **ST_TransformPipeline** - Availability: 3.4.0 定義されている座標変換パイプラインを使用して異なる空間参照系に変換された新しいジオメトリを返します。
- **TopoElement** - Availability: 3.4.0 TopoGeometry を TopoElement に変換します。
- **postgis_srs** - Availability: 3.4.0 求める機関と空間参照識別子に関するメタデータレコードを返します。

- **postgis_srs_all** - Availability: 3.4.0 Proj データベース内のあらゆる空間参照系のメタデータレコードを返します。
- **postgis_srs_codes** - Availability: 3.4.0 指定した機関に関連付けられた SRS コードの一覧を返します。
- **postgis_srs_search** - Availability: 3.4.0 bounds パラメータを完全に含む適用範囲を持つ投影座標系のメタデータレコードを返します。

PostGIS 3.4 で機能強化された関数

- **PostGIS_Full_Version** - Enhanced: 3.4.0 現在、外部 PROJ 設定の NETWORK_ENABLED、URL_ENDPOINT、proj.db 位置の DATABASE_PATH があります完全な PostGIS のバージョン情報とコンフィギュレーション情報を報告します。
- **PostGIS_PROJ_Version** - Enhanced: 3.4.0 現在、NETWORK_ENABLED、URL_ENDPOINT、proj.db 位置の DATABASE_PATH があります PROJ4 のバージョン番号を返します。
- **ST_AsSVG** - Enhanced: 3.4.0 全ての曲線タイプに対応しましたジオメトリから SVG パスデータを返します。
- **ST_ClosestPoint** - Enhanced: 3.4.0 - ジオグラフィに対応しました。g1 上にある、g2 と最近傍となる 2 次元ポイントを返します。これは、あるジオメトリから他のジオメトリへの最短ラインの一つ目のポイントです。
- **ST_LineSubstring** - Enhanced: 3.4.0 ジオグラフィ対応が導入されました。二つの割合位置からラインの一部を返します。
- **ST_Project** - Enhanced: 3.4.0 ジオメトリ引数と、azimuth を省略した 2 ポイント形式を許します。始点から距離と方位で算出されたポイントを返します。
- **ST_Resample** - Enhanced: 3.4.0 リサンプリング選択肢に最大と最小を追加指定したりサンプリングアルゴリズム、新しいピクセル範囲、グリッドの隅、定義するか他のラスタから借りてきた地理参照属性を使ってリサンプリングを行います。
- **ST_Rescale** - Enhanced: 3.4.0 リサンプリング選択肢に最大と最小を追加スケール (ピクセルサイズ) だけを調整するリサンプリングを行います。新しいピクセル値のリサンプリングアルゴリズムとして最近傍補間 ('NearestNeighbor' (英語または米式綴り方))、双線形補間 ('Bilinear')、3 次補間 ('Cubic')、3 次スプライン補間 ('CubicSpline')、ランチョス補間 ('Lanczos') を用います。デフォルトは最近傍補間です。
- **ST_ShortestLine** - Enhanced: 3.4.0 - ジオグラフィに対応しました。二つのジオメトリの 3 次元の最短ラインを返します。

PostGIS 3.4 で変更された関数

- **PostGIS_Extensions_Upgrade** - Changed: 3.4.0 target_version 引数が追加されました。PostGIS エクステンション (例: postgis_raster, postgis_topology, postgis_sfcgal) について、指定したバージョンまたは最新版にパッケージ化し、アップグレードします。

13.12.5 PostGIS 3.3 で新規作成/機能強化された関数

次に示す関数は、新規作成または機能強化された PostGIS 関数です。

PostGIS 3.3 で新規作成された関数

- **RemoveUnusedPrimitives** - Availability: 3.3.0 存在する TopoGeometry オブジェクトを定義するのに必要でないトポロジプリミティブを削除します。
- **ST_3DConvexHull** - Availability: 3.3.0 ジオメトリの 3 次元の凸包を計算します。
- **ST_3DUnion** - Availability: 3.3.0 集約関数の形式が追加されました 3 次元の結合を計算します。
- **ST_AsMARC21** - Availability: 3.3.0 ジオメトリを、地理データフィールド (034) を持つ MARC21/XML データとして返します。

- **ST_GeomFromMARC21** - Availability: 3.3.0 libxml2 2.6 以上が必要です。MARC21/XML 地理データを入力に取り、PostGIS ジオメトリオブジェクトを返します。
- **ST_Letters** - Availability: 3.3.0 デフォルトの開始位置を原点とし、デフォルトの高さを 100 とする、ジオメトリとして描画された文字を返します。
- **ST_OptimalAlphaShape** - Availability: 3.3.0 - SFCGAL $\geq$ 1.4.1 が必要です。「最適」アルファ値を使ってジオメトリを囲むアルファシェイプを計算します。
- **ST_SimplifyPolygonHull** - Availability: 3.3.0. ポリゴンジオメトリに対してトポロジを保存した状態で簡略化した外側または内側の凹包を計算します。
- **ST_TriangulatePolygon** - Availability: 3.3.0. ポリゴンの制約付きドロネー三角分割を計算します。
- **postgis_sfcgal_full_version** - Availability: 3.3.0 CGAL と Boost のバージョンを含む、使用している SFCGAL の完全なバージョンを返します

PostGIS 3.3 で機能強化された関数

- **ST_ConcaveHull** - Enhanced: 3.3.0, GEOS 3.11 から GEOS ネイティブ実装が有効になりました全ての入力ジオメトリの頂点を含む凹ジオメトリを計算します。
- **ST_LineMerge** - Enhanced: 3.3.0 directed パラメータを付け付けるようになりました。MULTILINESTRING を縫い合わせて形成したラインを返します。

PostGIS 3.3 で変更された関数

- **PostGIS_Extensions_Upgrade** - Changed: 3.3.0 どの PostGIS のバージョンからでもアップグレードできるようになりました。ただし全てのシステムで動作するわけではありません。PostGIS エクステンション (例: postgis_raster, postgis_topology, postgis_sfcgal) について、指定したバージョンまたは最新版にパッケージ化し、アップグレードします。

13.12.6 PostGIS 3.2 で新規作成/機能強化された関数

次に示す関数は、新規作成または機能強化された PostGIS 関数です。

PostGIS 3.2 で新規作成された関数

- **FindLayer** - Availability: 3.2.0 様々な方法で topology.layer 行を返します。
- **FindTopology** - Availability: 3.2.0 異なる方法でトポロジ行を返します。
- **GetFaceContainingPoint** - Availability: 3.2.0 ポイントを含むフェイスを見つけます。
- **ST_AsFlatGeobuf** - Availability: 3.2.0 行の集合の FlatGeobuf 表現を返します。
- **ST_Contour** - Availability: 3.2.0 与えられたラスタブンドから等高線ベクタを生成します。GDAL 等高線生成アルゴリズムを使います。
- **ST_DumpSegments** - Availability: 3.2.0 ジオメトリ内の辺の行である geometry_dump 行の集合を返します。
- **ST_FromFlatGeobuf** - Availability: 3.2.0 FlatGeobuf データを読みます。
- **ST_FromFlatGeobufToTable** - Availability: 3.2.0 FlatGeobuf データの構造に基づいてテーブルを生成します。
- **ST_InterpolateRaster** - Availability: 3.2.0 X 値と Y 値を使用してグリッド上のポイントを配置し、ポイントの Z 値をサーフェス標高として配置し、3 次元ポイントの入力セットに基づいてグリッドサーフェスを補間します。

- **ST_SRID** - Availability: 3.2.0 TopoGeometry の空間参照識別子を返します。
- **ST_Scroll** - Availability: 3.2.0 閉じた LINESTRING の開始点を変更する。
- **ST_SetM** - Availability: 3.2.0 入力ジオメトリと同じ X/Y 値を持ち、かつ、指定されたりサンプリングアルゴリズムを使ってラスタから複写された M 値を持つジオメトリを返します。
- **ST_SetZ** - Availability: 3.2.0 入力ジオメトリと同じ X/y 座標値と、指定されたりサンプリングアルゴリズムを使ってラスタから複写された Z 値とを持つジオメトリを返します。
- **TopoGeom_addTopoGeom** - Availability: 3.2 TopoGeometry の要素を他の TopoGeometry の定義に追加します。
- **ValidateTopologyRelation** - Availability: 3.2.0 不正なトポロジ関係の行に関する情報を返します。
- **postgis.gdal_vsi_options** - Availability: 3.2.0 データベース外ラスタを操作する時に使用するオプションを設定するためのコンフィギュレーション。

PostGIS 3.2 で機能強化された関数

- **GetFaceByPoint** - Enhanced: 3.2.0 より効率的な実装とより明確な短縮、不正なトポロジでの動作停止。与えたポイントとインタセクトするフェイスを見つけます。
- **ST_ClusterKMeans** - Enhanced: 3.2.0 max_radius パラメータに対応しました入力ジオメトリごとに k 平均法アルゴリズムを使ってクラスター番号を返すウィンドウ関数です。
- **ST_MakeValid** - Enhanced: 3.2.0, アルゴリズムに関する任意パラメータ 'linework' と 'structure' が追加されました。GEOS 3.10.0 以上が必要です。頂点を失うことなしに不正なジオメトリを妥当なジオメトリにしようと試みます。
- **ST_PixelAsCentroid** - Enhanced: 3.2.0 C 言語による、より高速な実装。ピクセルで表現される面の重心 (ポイントジオメトリ) を返します。
- **ST_PixelAsCentroids** - Enhanced: 3.2.0 C 言語による、より高速な実装。全てのピクセルについて重心 (ポイントジオメトリ) を、ピクセルごとのピクセル値とラスタ座標系の X と Y とを付けて返します。ポイントジオメトリの座標はピクセルで表現される面の重心です。
- **ST_Point** - Enhanced: 3.2.0 SRID 任意引数が追加されました。古いバージョンでは、ジオメトリに SRID を与えるには ST_SetSRID を併用しなければなりませんでした。X, Y と SRID の値からポイントを生成します。
- **ST_PointM** - Enhanced: 3.2.0 SRID 任意引数が追加されました。古いバージョンでは、ジオメトリに SRID を与えるには ST_SetSRID を併用しなければなりませんでした。X, Y, M と SRID の値からポイントを生成します。
- **ST_PointZ** - Enhanced: 3.2.0 SRID 任意引数が追加されました。古いバージョンでは、ジオメトリに SRID を与えるには ST_SetSRID を併用しなければなりませんでした。X, Y, Z と SRID の値からポイントを生成します。
- **ST_PointZM** - Enhanced: 3.2.0 SRID 任意引数が追加されました。古いバージョンでは、ジオメトリに SRID を与えるには ST_SetSRID を併用しなければなりませんでした。X, Y, Z, M と SRID の値からポイントを生成します。
- **ST_RemovePoint** - Enhanced: 3.2.0 ラインストリングからポイントを削除します。
- **ST_RemoveRepeatedPoints** - Enhanced: 3.2.0 重複ポイントを除いたジオメトリを返します。
- **ST_StartPoint** - Enhanced: 3.2.0 全てのジオメトリのポイントを返すようになりました。以前のバージョンではラインストリング以外では NULL を返していました。Returns the first point of a LineString, CircularLineString, or NURBS Curve.

- **ST_Value** - Enhanced: 3.2.0 任意引数 `resample` が追加されました。指定したバンドにおける `columnx`, `rowy` で指定したピクセルまたは指定したジオメトリポイントに対応するピクセルの値を返します。バンド番号は 1 始まりで、指定しない場合には、1 番と仮定します。`exclude_nodata_value` が `FALSE` に設定された場合には、NODATA ピクセルを含む全てのピクセルがインタセクトするかが考慮され、値を返します。`exclude_nodata_value` を渡さない場合には、ラスタのメタデータから読みます。
- **TopoGeo_AddLineString** - Enhanced: 3.2.0 符号付き識別子を返す機能が追加されました。許容差を使って既存のトポロジにラインストリングを追加し、可能ならエッジ/フェイスを分割します。

PostGIS 3.2 で変更された関数

- **ST_Boundary** - Changed: 3.2.0 TIN に対応しました。GEOS を使いません。曲線を線形化しません。ジオメトリの境界を返します。
- **ValidateTopology** - Changed: 3.2.0 任意パラメータ `bbox` を追加し、フェイスラベルとエッジリンクのチェックを追加しました。トポロジの問題についての詳細を示す `validatetopology_returntype` の集合を返します。

13.12.7 PostGIS 3.1 で新規作成/機能強化された関数

次に示す関数は、新規作成または機能強化された PostGIS 関数です。

PostGIS 3.1 で新規作成された関数

- **ST_Hexagon** - Availability: 3.1.0 与えられたエッジサイズと六角形グリッド空間内のセル座標を使って単一の六角形を返します。
- **ST_HexagonGrid** - Availability: 3.1.0 引数ジオメトリの境界を完全にカバーする六角形とセルインデックスを返します。
- **ST_MaximumInscribedCircle** - Availability: 3.1.0. ジオメトリに含まれる最大の円を計算します。
- **ST_ReducePrecision** - Availability: 3.1.0. 全ての与えられたグリッド許容値に丸められたポイントからなる妥当なジオメトリを返します。
- **ST_Square** - Availability: 3.1.0 与えられたエッジサイズと六角形グリッド空間内のセル座標を使って単一の正方形を返します。
- **ST_SquareGrid** - Availability: 3.1.0 引数ジオメトリの境界を完全にカバーするグリッド正方形とセルインデックスを返します。

PostGIS 3.1 で機能強化された関数

- **ST_AsEWKT** - Enhanced: 3.1.0 精度パラメータ (任意) に対応しました。ジオメトリの SRID メタデータが付いた Well-Known Text (WKT) 表現を返します。
- **ST_ClusterKMeans** - Enhanced: 3.1.0 3 次元ジオメトリと重みに対応するようになりました入力ジオメトリごとに k 平均法アルゴリズムを使ってクラス番号を返すウィンドウ関数です。
- **ST_Difference** - Enhanced: 3.1.0 `gridSize` パラメータを受け付けるようになりました。ジオメトリ B とインタセクトしていないジオメトリ A の一部を表現するジオメトリを計算します。
- **ST_Intersection** - Enhanced: 3.1.0 `gridSize` パラメータを受け付けるようになりましたジオメトリ A とジオメトリ B の共通部分を表現するジオメトリを返します。
- **ST_MakeValid** - Enhanced: 3.1.0 NaN 値を持つ座標の削除が追加されました。頂点を失うことなしに不正なジオメトリを妥当なジオメトリにしようと試みます。
- **ST_Subdivide** - Enhanced: 3.1.0 `gridSize` パラメータを受け付けるようになりました。ジオメトリの線の分割を計算します。

- **ST_SymDifference** - Enhanced: 3.1.0 gridSize パラメータを受け付けるようになりました。ジオメトリ A とジオメトリ B がインタセクトしていない部分を表現するジオメトリを返します。
- **ST_TileEnvelope** - Enhanced: 3.1.0 margin パラメータが追加されました。Web メルカトル (SRID:3857) 上で XYZ タイルを使った矩形ポリゴンを生成します。
- **ST_UnaryUnion** - Enhanced: 3.1.0 gridSize パラメータを受け付けるようになりました。単一のジオメトリの要素の結合を計算します。
- **ST_Union** - Enhanced: 3.1.0 gridSize パラメータを受け付けるようになりました。入力ジオメトリのポイント集合の結合を表現するジオメトリを返します。

PostGIS 3.1 で変更された関数

- **ST_Count** - Changed: 3.1.0 - ST_Count(rastertable, rastercolumn, ...) の形式は削除されました。代わりにを使用します。ラスタまたはラスタカバレッジの指定したバンドのピクセル数を返します。バンドを指定しない場合には、1 番と仮定します。exclude_nodata_value を TRUE に設定している場合には、NODATA 値と等しくないピクセルのみを数えます。
- **ST_Force3D** - Changed: 3.1.0. 0 でない Z 値を指定できるようになりました。ジオメトリを XYZ モードに強制します。これは ST_Force3DZ の別名です。
- **ST_Force3DM** - Changed: 3.1.0. 0 でない M 値を指定できるようになりました。ジオメトリを XYM モードに強制します。
- **ST_Force3DZ** - Changed: 3.1.0. 0 でない Z 値を指定できるようになりました。ジオメトリを XYZ モードに強制します。
- **ST_Force4D** - Changed: 3.1.0. 0 でない Z 値と M 値を指定できるようになりました。ジオメトリを XYZM モードに強制します。
- **ST_Histogram** - Changed: 3.1.0 ST_Histogram(table_name, column_name) の形式は削除されました。ラスタまたはラスタカバレッジのビン範囲で分割したデータ分布をまとめるヒストグラムの集合を返します。ビン数を指定しない場合には自動計算されます。
- **ST_Quantile** - Changed: 3.1.0 ST_Quantile(table_name, column_name) の形式は削除されました。ラスタまたはラスタテーブルカバレッジのサンプルまたは母集団の分位数を計算します。値がラスタの 25%,50%,75% にあるかを調べることができます。
- **ST_SummaryStats** - Changed: 3.1.0 ST_SummaryStats(rastertable, rastercolumn, ...) の形式は削除されました。代わりにを使用します。与えられたラスターのバンドまたはラスターカバレッジの count, sum, mean, stddev, min, max からなる統計情報のサマリーです。バンドを指定しない場合には、1 番と仮定します。

13.12.8 PostGIS 3.0 で新規作成/機能強化された関数

次に示す関数は、新規作成または機能強化された PostGIS 関数です。

PostGIS 3.0 で新規作成された関数

- **CG_ConstrainedDelaunayTriangles** - Availability: 3.0.0 入力ジオメトリの周りの制約付きドロネー三角形を返します。
- **ST_3DLineInterpolatePoint** - Availability: 3.0.0 3 次元ラインに沿って、割合で示された位置の補間ポイントを返します。
- **ST_ConstrainedDelaunayTriangles** - Availability: 3.0.0 入力ジオメトリの周りの制約付きドロネー三角形を返します。
- **ST_TileEnvelope** - Availability: 3.0.0 Web メルカトル (SRID:3857) 上で XYZ タイルを使った矩形ポリゴンを生成します。

PostGIS 3.0 で機能強化された関数

- **ST_AsMVT** - Enhanced: 3.0 - 地物 ID への対応を追加。行集合の MVT 表現を返す集約関数です。
- **ST_Contains** - Enhanced: 3.0.0 GEOMETRYCOLLECTION への対応が可能となりました B の全てのポイントが A 内にあり、かつ、双方の内部に共有点が存在するかどうかをテストします。
- **ST_ContainsProperly** - Enhanced: 3.0.0 GEOMETRYCOLLECTION への対応が可能となりました B の全てのポイントが A の内部にあるかをテストします。
- **ST_CoveredBy** - Enhanced: 3.0.0 GEOMETRYCOLLECTION への対応が可能となりました A の全てのポイントが B 内にあるかをテストします。
- **ST_Covers** - Enhanced: 3.0.0 GEOMETRYCOLLECTION への対応が可能となりました B の全ての点が A 内にあるかをテストします。
- **ST_Crosses** - Enhanced: 3.0.0 GEOMETRYCOLLECTION への対応が可能となりました二つのジオメトリが内部に共有ポイントを持ち、かつそれだけにならないようになっているかテストします。
- **ST_CurveToLine** - Enhanced: 3.0.0 線形化した弧ごとの最小線分数を実装しました。トポロジ的な崩壊を防ぐためです。曲線を含むジオメトリを線ジオメトリに変換します。
- **ST_Disjoint** - Enhanced: 3.0.0 GEOMETRYCOLLECTION への対応が可能となりました二つのジオメトリが共有点を持たないようになっているかテストします。
- **ST_Equals** - Enhanced: 3.0.0 GEOMETRYCOLLECTION への対応が可能となりました二つのジオメトリが同じ点集合になっているかテストします。
- **ST_GeneratePoints** - Enhanced: 3.0.0 seed パラメータの追加ポリゴン内やマルチポリゴン内にランダムなマルチポイントを生成します。
- **ST_GeomFromGeoJSON** - Enhanced: 3.0.0 パースされたジオメトリのデフォルトの SRID は、他に指定していない場合には 4326 となります。ジオメトリの GeoJSON 表現を入力として、PostGIS ジオメトリオブジェクトを出力します。
- **ST_LocateBetween** - Enhanced: 3.0.0 - POLYGON, TIN, TRIANGLE への対応が追加されました。M 値の範囲に合致する部分ジオメトリを返します。
- **ST_LocateBetweenElevations** - Enhanced: 3.0.0 - POLYGON, TIN, TRIANGLE への対応が追加されました。標高 (Z 値) 範囲にある部分ジオメトリを返します。
- **ST_Overlaps** - Enhanced: 3.0.0 GEOMETRYCOLLECTION への対応が可能となりました二つのジオメトリが同じ次元を持ち、インタセクトして、かつ相手と重ならない点少なくとも一つあるかをテストします。
- **ST_Relate** - Enhanced: 3.0.0 GEOMETRYCOLLECTION への対応が可能となりました二つのジオメトリが与えられた交差行列パターンに合致するトポロジ関係があるかどうかを見るか、交差行列を計算するかします。
- **ST_Segmentize** - Enhanced: 3.0.0 ジオメトリの分割において、現在は、同じ長さに分割しています与えた長さを超える線分を持たないよう変更したジオメトリ/ジオグラフィを返します。
- **ST_Touches** - Enhanced: 3.0.0 GEOMETRYCOLLECTION への対応が可能となりました二つのジオメトリが少なくとも一つの共有点を持ち、かつ内部でインタセクトしていないようになっているかテストします。
- **ST_Within** - Enhanced: 3.0.0 GEOMETRYCOLLECTION への対応が可能となりました A の全てのポイントが B 内にあり、かつ両方の内部に共有点を持つかどうかをテストします。

PostGIS 3.0 で変更された関数

- **PostGIS_Extensions_Upgrade** - Changed: 3.0.0 緩いエクステンションを再パッケージし、また、postgis_raster に対応しました。PostGIS エクステンション (例: postgis_raster, postgis_topology, postgis_sfcgal) について、指定したバージョンまたは最新版にパッケージ化し、アップグレードします。

- **ST_3DDistance** - Changed: 3.0.0 - SFCGAL 版は削除されました投影座標系の単位で、二つのジオメトリ間の 3 次元デカルト距離の最小値を返します (空間参照系に基づきます)。
- **ST_3DIntersects** - Changed: 3.0.0 SFCGAL バックエンドが削除され、GEOS バックエンドでは TIN に対応しました。二つのジオメトリが 3 次元空間において空間的にインタセクトするかどうかをテストします。ポイント、ラインストリング、ポリゴン、多面体サーフェス (面) についてのみ動作します。
- **ST_Area** - Changed: 3.0.0 - SFCGAL に依存しなくなりました。ポリゴンジオメトリの面積を返します。
- **ST_AsGeoJSON** - Changed: 3.0.0 レコードの入力に対応しました GeoJSON 形式のジオメトリまたは地物を返します。
- **ST_AsGeoJSON** - Changed: 3.0.0 EPSG:4326 以外の場合の SRID 出力。GeoJSON 形式のジオメトリまたは地物を返します。
- **ST_AsKML** - Changed: 3.0.0 - "version" の付いた形式の削除ジオメトリを KML 要素として返します。
- **ST_Distance** - Changed: 3.0.0 - SFCGAL に依存しなくなりました。二つのジオメトリ値またはジオグラフィ値間の距離を返します。
- **ST_Intersection** - Changed: 3.0.0 SFCGAL 非依存になりました。ジオメトリ A とジオメトリ B の共通部分を表現するジオメトリを返します。
- **ST_Intersects** - Changed: 3.0.0 SFCGAL 版を削除し、2 次元 TIN のネイティブ対応を追加しました。二つのジオメトリがインタセクトしている (少なくとも一つの共有点がある) かどうかテストします。
- **ST_Union** - Changed: 3.0.0 SFCGAL 非依存になりました。入力ジオメトリのポイント集合の結合を表現するジオメトリを返します。

13.12.9 PostGIS 2.5 で新規作成/機能強化された関数

次に示す関数は、新規作成または機能強化された PostGIS 関数です。

PostGIS 2.5 で新規作成された関数

- **PostGIS_Extensions_Upgrade** - Availability: 2.5.0 PostGIS エクステンション (例: `postgis_raster`, `postgis_topology`, `postgis_sfcgal`) について、指定したバージョンまたは最新版にパッケージ化し、アップグレードします。
- **ST_Angle** - Availability: 2.5.0 3 点もしくは 4 点、または 2 線で定義される二つのベクタ間の角度を返します。
- **ST_AsHexWKB** - Availability: 2.5.0 Well-Known Binary (WKB) ラスタを 16 進数表現で返します。
- **ST_BandFileSize** - Availability: 2.5.0 ファイルシステムに格納されているバンドのファイルサイズを返します。バンド番号が指定されていない場合には、1 番と仮定します。
- **ST_BandFileTimestamp** - Availability: 2.5.0 ファイルシステムに格納されているバンドのファイルタイムスタンプ返します。バンド番号が指定されていない場合には、1 番と仮定します。
- **ST_ChaikinSmoothing** - Availability: 2.5.0 チャイキンのアルゴリズムを使って、与えられたジオメトリの平滑化されたものを返します。
- **ST_FilterByM** - Availability: 2.5.0 M 値に基づいて頂点を削除します。
- **ST_Grayscale** - Availability: 2.5.0 元のラスタと指定したバンドを赤、緑、青バンドとして一つの 8BUI バンドを持つラスタを生成します。
- **ST_LineInterpolatePoints** - Availability: 2.5.0 ラインに沿って、割合で示された複数の位置の補間ポイントを返します。
- **ST_OrientedEnvelope** - Availability: 2.5.0. ジオメトリを囲む最小の回転四角形を返します。

- **ST_QuantizeCoordinates** - Availability: 2.5.0 座標値の最下位ビットを 0 にします。
- **ST_RastFromHexWKB** - Availability: 2.5.0 Well-Known バイナリ (WKB) ラスタの 16 進数表現からラスタを返します。
- **ST_RastFromWKB** - Availability: 2.5.0 Well-Known Binary (WKB) ラスタからラスタ値を返します。
- **ST_SetBandIndex** - Availability: 2.5.0 データベース外バンドの外部バンド番号の更新
- **ST_SetBandPath** - Availability: 2.5.0 データベース外バンドの外部パスとバンド番号を更新します。

PostGIS 2.5 で機能強化された関数

- **ST_AsBinary/ST_AsWKB** - Enhanced: 2.5.0 ST_AsWKB の追加ラスタの Well-Known Binary (WKB) 表現を返します。
- **ST_AsMVT** - Enhanced: 2.5.0 - パラレルクエリへの対応の追加。行集合の MVT 表現を返す集約関数です。
- **ST_AsText** - Enhanced: 2.5 - 精度の任意引数が導入されました。ジオメトリ/ジオグラフィの SRID メタデータのない Well-Known Text (WKT) 表現を返します。
- **ST_BandMetaData** - Enhanced: 2.5.0 outdb ラスタに outdbbandnum, filesize と filetimestamp を取り入れました。指定したラスタバンドの基本的なメタデータを返します。バンド番号を指定しない場合には、1 番と仮定します。
- **ST_Buffer** - Enhanced: 2.5.0 - ST_Buffer のジオメトリ対応版が強化され、バッファを施す側を side=both|left|right で指定できるようになりました。あるジオメトリからの距離が指定された距離以下となる点全ての集合となるジオメトリを返します。
- **ST_GeomFromGeoJSON** - Enhanced: 2.5.0 JSON と JSONB の入力を受け付けるようになりました。ジオメトリの GeoJSON 表現を入力として、PostGIS ジオメトリオブジェクトを出力します。
- **ST_GeometricMedian** - Enhanced: 2.5.0 ポイントの重みとしての M 値の対応が追加されました。マルチポイントの幾何学的中央値を返します。
- **ST_Intersects** - Enhanced: 2.5.0 ジオメトリコレクションに対応しました。二つのジオメトリがインタセクトしている (少なくとも一つの共有点がある) かどうかテストします。
- **ST_OffsetCurve** - Enhanced: 2.5 - GEOMETRYCOLLECTION と MULTILINESTRING への対応追加与えられた距離と方面に入力ラインをずらしたラインを返します。
- **ST_Scale** - Enhanced: 2.5.0 局所原点 (origin パラメータ) を使った拡大縮小への対応を導入しました。与えた係数でジオメトリを拡大縮小します。
- **ST_Split** - Enhanced: 2.5.0 マルチラインによるポリゴンの分割に対応するようになりました。ジオメトリを他のジオメトリで分割してできたジオメトリのコレクションを返します。
- **ST_Subdivide** - Enhanced: 2.5.0 ポリゴン分割で存在するポイントを再利用して頂点数の最小値を 8 から 5 に変更。ジオメトリの線の分割を計算します。

PostGIS 2.5 で変更された関数

- **ST_GDALDrivers** - Changed: 2.5.0 - can_read カラムと can_write カラムを追加。使用している GDAL ライブラリが対応するラスタ書式の一覧を返します。この一覧で can_write=True となっているものだけが ST_AsGDALRaster で使えます。

13.12.10 PostGIS 2.4 で新規作成/機能強化された関数

次に示す関数は、新規作成または機能強化された PostGIS 関数です。

PostGIS 2.4 で新規作成された関数

- **ST_AsGeobuf** - Availability: 2.4.0 行集合の Geobuf 表現を返します。
- **ST_AsMVT** - Availability: 2.4.0 行集合の MVT 表現を返す集約関数です。
- **ST_AsMVTGeom** - Availability: 2.4.0 ジオメトリを MVT タイルの座標空間に変換します。
- **ST_Centroid** - Availability: 2.4.0 ジオグラフィが導入されました。ジオメトリの幾何学的重心を返します。
- **ST_ForcePolygonCCW** - Availability: 2.4.0 全ての外環を反時計回りに、全ての内環を時計回りに、それぞれ強制します。
- **ST_ForcePolygonCW** - Availability: 2.4.0 全ての外環を時計回りに、全ての内環を反時計回りに、それぞれ強制します。
- **ST_FrechetDistance** - Availability: 2.4.0 - GEOS >= 3.7.0 が必要です二つのジオメトリのフレシェ距離を返します。
- **ST_IsPolygonCCW** - Availability: 2.4.0 ポリゴンが反時計回りの外環を持っていて、時計回りの内環を持っているかをテストします。
- **ST_IsPolygonCW** - Availability: 2.4.0 ポリゴンが時計回りの外環を持っていて、反時計回りの内環を持っているかをテストします。
- **ST_MakeEmptyCoverage** - Availability: 2.4.0 空のラスタタイルのグリッドでジオリファレンスを施されている領域を生成します。

PostGIS 2.4 で機能強化された関数

- **ST_AsTWKB** - Enhanced: 2.4.0 メモリと速度の改善。TWKB (Tiny Well-Known Binary) としてジオメトリを出力します。
- **ST_Covers** - Enhanced: 2.4.0 ジオグラフィ型を使う形式においてポリゴンの中のポリゴンとポリゴンの中のラインストリングへの対応を追加 B の全ての点が A 内にあるかをテストします。
- **ST_CurveToLine** - Enhanced: 2.4.0 最大距離差による許容範囲と最大角度による許容範囲に対応し、対称出力に対応しました。曲線を含むジオメトリを線ジオメトリに変換します。
- **ST_Project** - Enhanced: 2.4.0 負の距離と非正規化方位を許容するようになりました。始点から距離と方位で算出されたポイントを返します。
- **ST_Reverse** - Enhanced: 2.4.0 曲線対応が導入されました。頂点の順序を逆にしたジオメトリを返します。

PostGIS 2.4 で変更された関数

- **=** - Changed: 2.4.0, 以前の版では、ジオメトリ自体の等価性でなくバウンディングボックスが等価かどうかを見ていました。バウンディングボックスが等価かどうかを知る必要がある場合には、替わりに `ST_Equals` を使います。ジオメトリ/ジオグラフィ A の座標と座標の並び順がジオメトリ/ジオグラフィ B と同じ場合に `TRUE` を返します。
- **ST_Node** - Changed: 2.4.0 この関数は内部で `GEOSUnaryUnion` の代わりに `GEOSNode` を使用しています。ラインストリングの並び順と方向が PostGIS 2.4 より前のものと違うことになるかも知れません。ラインストリングの集合にノードを作成します。

13.12.11 PostGIS 2.3 で新規作成/機能強化された関数

次に示す関数は、新規作成または機能強化された PostGIS 関数です。

PostGIS 2.3 で新規作成された関数

- **&&(geometry,gidx)** - Availability: 2.3.0 BRIN (Block Range INdexes) が導入されました。ジオメトリの (キャッシュされている)n 次元バウンディングボックスが単精度浮動小数点数による n 次元バウンディングボックス (GIDX) とインタセクトする場合に TRUE を返します。
- **&&(gidx,geometry)** - Availability: 2.3.0 BRIN (Block Range INdexes) が導入されました。単精度浮動小数点数による n 次元バウンディングボックス (GIDX) がジオメトリの (キャッシュされている)n 次元バウンディングボックスとインタセクトする場合に TRUE を返します。
- **&&(gidx,gidx)** - Availability: 2.3.0 BRIN (Block Range INdexes) が導入されました。二つの単精度浮動小数点数による n 次元バウンディングボックス (GIDX) が相互にインタセクトする場合に TRUE を返します。
- **&&(box2df,box2df)** - Availability: 2.3.0 BRIN (Block Range INdexes) が導入されました。二つの単精度浮動小数点数による 2 次元バウンディングボックス (BOX2DF) が相互にインタセクトする場合に TRUE を返します。
- **&&(box2df,geometry)** - Availability: 2.3.0 BRIN (Block Range INdexes) が導入されました。単精度浮動小数点数による 2 次元バウンディングボックスがジオメトリの (キャッシュされている)2 次元バウンディングボックスとインタセクトする場合に TRUE を返します。
- **&&(geometry,box2df)** - Availability: 2.3.0 BRIN (Block Range INdexes) が導入されました。ジオメトリの (キャッシュされている)2 次元バウンディングボックスが単精度浮動小数点数による 2 次元バウンディングボックスとインタセクトする場合に TRUE を返します。
- **@(box2df,box2df)** - Availability: 2.3.0 BRIN (Block Range INdexes) が導入されました。二つの単精度浮動小数点数による n 次元バウンディングボックス (GIDX) の一方がもう一方を包含する場合に TRUE を返します。
- **@(box2df,geometry)** - Availability: 2.3.0 BRIN (Block Range INdexes) が導入されました。単精度浮動小数点数による 2 次元バウンディングボックス (BOX2DF) がジオメトリの 2 次元バウンディングボックスに包含される場合に TRUE を返します。
- **@(geometry,box2df)** - Availability: 2.3.0 BRIN (Block Range INdexes) が導入されました。ジオメトリの 2 次元バウンディングボックスが単精度浮動小数点数による 2 次元バウンディングボックス (BOX2DF) に包含される場合に TRUE を返します。
- **Populate_Topology_Layer** - Availability: 2.3.0 テーブルからメタデータを読み、topology.layer テーブルに不足しているものを追加します。
- **ST_ClusterDBSCAN** - Availability: 2.3.0 入力ジオメトリごとに DBSCAN アルゴリズムを使ってクラスター番号を返すウィンドウ関数です。
- **ST_ClusterKMeans** - Availability: 2.3.0 入力ジオメトリごとに k 平均法アルゴリズムを使ってクラスター番号を返すウィンドウ関数です。
- **ST_GeneratePoints** - Availability: 2.3.0 ポリゴン内やマルチポリゴン内にランダムなマルチポイントを生成します。
- **ST_GeometricMedian** - Availability: 2.3.0 マルチポイントの幾何学的中央値を返します。
- **ST_MakeLine** - Availability: 2.3.0 - MULTIPOINT 入力要素への対応が導入されました POINT、MULTIPOINT、LINESTRING から LINESTRING を生成します。
- **ST_MinimumBoundingRadius** - Availability: 2.3.0 ジオメトリを完全に包含する最小円の中心ポイントと半径を返します。
- **ST_MinimumClearance** - Availability: 2.3.0 ジオメトリのクリアランスの最小値を返します。この値はジオメトリのロバスト性を示すものです。

- **ST_MinimumClearanceLine** - Availability: 2.3.0 - GEOS 3.6.0 以上が必要です。ジオメトリの最小クリアランスを示す、2 点のラインストリングを返します。
- **ST_Normalize** - Availability: 2.3.0 標準的な形式に変えたジオメトリを返します。
- **ST_Points** - Availability: 2.3.0 ジオメトリの全ての座標を含むマルチポイントを返します。
- **ST_VoronoiLines** - Availability: 2.3.0 ジオメトリの頂点からボロノイ図のセルを返します。
- **ST_VoronoiPolygons** - Availability: 2.3.0 ジオメトリの頂点からボロノイ図のセルを返します。
- **ST_WrapX** - Availability: 2.3.0 GEOS が必要です。ジオメトリを X 値で回り込ませます。
- **TopoGeom_addElement** - Availability: 2.3 TopoGeometry の定義に要素を追加します。
- **TopoGeom_remElement** - Availability: 2.3 TopoGeometry の定義から要素を削除します。
- **~(box2df,box2df)** - Availability: 2.3.0 BRIN (Block Range INDEXes) が導入されました。二つの単精度浮動小数点数による 2 次元バウンディングボックス (BOX2DF) の一方がもう一方を包含する場合に TRUE を返します。
- **~(box2df,geometry)** - Availability: 2.3.0 BRIN (Block Range INDEXes) が導入されました。単精度浮動小数点数による 2 次元バウンディングボックス (BOX2DF) をジオメトリの (キャッシュされている)2 次元バウンディングボックスが包含する場合に TRUE を返します。
- **~(geometry,box2df)** - Availability: 2.3.0 BRIN (Block Range INDEXes) が導入されました。ジオメトリの (キャッシュされている)2 次元バウンディングボックスが単精度浮動小数点数による n 次元バウンディングボックス (GIDX) を包含する場合に TRUE を返します。

PostGIS 2.3 で機能強化された関数

- **ST_Contains** - Enhanced: 2.3.0 PIP short-circuit (ポリゴンとポイントに限定した高速判定) を少ないポイントからなるマルチポイントに対応することができるよう拡張しました。以前の版ではポリゴンとポイントの組み合わせにだけ対応していました。B の全てのポイントが A 内にあり、かつ、双方の内部に共有点が存在するかどうかをテストします。
- **ST_Covers** - Enhanced: 2.3.0 ジオメトリについて、PIP short-circuit (ポリゴンとポイントに限定した高速判定) を少ないポイントからなるマルチポイントに対応することができるよう拡張しました。以前の版ではポリゴンとポイントの組み合わせにだけ対応していました。B の全ての点が A 内にあるかをテストします。
- **ST_Expand** - Enhanced: 2.3.0 異なる次元の異なる量によるボックスの拡張に対応するようになりました。他のバウンディングボックスまたはジオメトリから拡張されたバウンディングボックスを返します。
- **ST_Intersects** - Enhanced: 2.3.0 PIP short-circuit (ポリゴンとポイントに限定した高速判定) を少ないポイントからなるマルチポイントに対応することができるよう拡張しました。以前の版ではポリゴンとポイントの組み合わせにだけ対応していました。二つのジオメトリがインタセクトしている (少なくとも一つの共有点がある) かどうかテストします。
- **ST_Segmentize** - Enhanced: 2.3.0 ジオグラフィの分割において、現在は、同じ長さに分割しています与えた長さを超える線分を持たないよう変更したジオメトリ/ジオグラフィを返します。
- **ST_Transform** - Enhanced: 2.3.0 直接の PROJ.4 文字列への対応が導入されました。異なる空間参照系に投影変換された新しいジオメトリを返します。
- **ST_Within** - Enhanced: 2.3.0 ジオメトリについて、PIP short-circuit (ポリゴンとポイントに限定した高速判定) を少ないポイントからなるマルチポイントに対応することができるよう拡張しました。以前の版ではポリゴンとポイントの組み合わせにだけ対応していました。A の全てのポイントが B 内にあり、かつ両方の内部に共有点を持つかどうかをテストします。

PostGIS 2.3 で変更された関数

- **ST_PointN** - Changed: 2.3.0 : 負数インデックスが有効になりました (-1 は終端を指します) ジオメトリの最初のラインストリングまたは曲線ストリングの N 番目のポイントを返します。

13.12.12 PostGIS 2.2 で新規作成/機能強化された関数

次に示す関数は、新規作成または機能強化された PostGIS 関数です。

PostGIS 2.2 で新規作成された関数

- **<<->>** - Availability: 2.2.0. A と B の間またはそれらのバウンディングボックスの間の n 次元距離を返します
- **ST_3DDifference** - Availability: 2.2.0 3 次元の差分を計算します。
- **ST_3DUnion** - Availability: 2.2.0 3 次元の結合を計算します。
- **ST_ApproximateMedialAxis** - Availability: 2.2.0 面ジオメトリの近似的な中心軸を計算します。
- **ST_AsEncodedPolyline** - Availability: 2.2.0 ラインストリングジオメトリから符号化したポリラインを返します。
- **ST_AsTWKB** - Availability: 2.2.0 TWKB (Tiny Well-Known Binary) としてジオメトリを出力します。
- **ST_BoundingDiagonal** - Availability: 2.2.0 ジオメトリのバウンディングボックスの対角線を返します。
- **ST_CPAWithin** - Availability: 2.2.0 二つのトラジェクトリの最接近時の距離が指定距離内であるかどうかをテストします。
- **ST_ClipByBox2D** - Availability: 2.2.0 長方形内に落ちるジオメトリの一部を返します。
- **ST_ClosestPointOfApproach** - Availability: 2.2.0 二つのトラジェクトリの最接近時の距離を返します。
- **ST_ClusterIntersecting** - Availability: 2.2.0 入力ジオメトリを接続関係にある集合にクラスタリングする集約関数です。
- **ST_ClusterWithin** - Availability: 2.2.0 分離距離でジオメトリのクラスタリングを行う集約関数です。
- **ST_CountAgg** - Availability: 2.2.0 集約関数です。ラスタ集合の与えられたバンドのピクセル数を返します。バンドが指定されていない場合には、1 番と仮定します。exclude_nodata_value を TRUE に設定している場合には、NODATA 値と等しくないピクセルのみを数えます。
- **ST_CreateOverview** - Availability: 2.2.0 与えられたラスタカバレッジから解像度を落としたものを生成します。
- **ST_DistanceCPA** - Availability: 2.2.0 二つのトラジェクトリの最接近する時の距離を返します。
- **ST_ForceCurve** - Availability: 2.2.0 該当する場合は、ジオメトリを曲線タイプに変換します。
- **ST_IsPlanar** - Availability: 2.2.0: これは 2.1.0 のマニュアルに記述されていましたが、2.1 版では偶然に外れてしまいました。サーフェスが平面であるかないかをチェックします。
- **ST_IsSolid** - Availability: 2.2.0 ジオメトリが立体であるかどうかをテストします。妥当性チェックは行いません。
- **ST_IsValidTrajectory** - Availability: 2.2.0 ジオメトリが妥当なトラジェクトリであるかどうかをテストします。
- **ST_LineFromEncodedPolyline** - Availability: 2.2.0 エンコード化ポリラインからラインストリングを生成します。
- **ST_MakeSolid** - Availability: 2.2.0 ジオメトリを立体にキャストします。チェックはしません。妥当な立体を得るには、入力ジオメトリは閉じた多面体サーフェスが閉じた TIN でなければなりません。
- **ST_MapAlgebra (callback function version)** - Availability: 2.2.0: マスクが追加されました。コールバック関数版 - 一つ以上の入力ラスタ、バンドインデックスと一つのユーザ定義コールバック関数から、一つのバンドからなるラスタを返します。
- **ST_MemSize** - Availability: 2.2.0 ラスタが取る領域の合計をバイト単位で返します。

- **ST_RemoveRepeatedPoints** - Availability: 2.2.0 重複ポイントを除いたジオメトリを返します。
- **ST_Retile** - Availability: 2.2.0 任意のタイル化されたラスタカバレッジから構成されたタイルの集合を返します。
- **ST_SetEffectiveArea** - Availability: 2.2.0 Visvalingam-Whyatt アルゴリズムを使って有効範囲となる個々の頂点を置きます。
- **ST_SimplifyVW** - Availability: 2.2.0 Visvalingam-Whyatt アルゴリズムを使用して、入力ジオメトリを簡略化したジオメトリを返します。
- **ST_Subdivide** - Availability: 2.2.0 ジオメトリの線の分割を計算します。
- **ST_SummaryStatsAgg** - Availability: 2.2.0 集約関数。与えられたラスターのバンドの集合の count, sum, mean, stddev, min, max からなる統計情報のサマリーです。バンドを指定しない場合には、1 番と仮定します。
- **ST_SwapOrdinates** - Availability: 2.2.0 与えられたジオメトリにおいて与えられた座標の値を入れ替えたジオメトリを返します。
- **ST_Volume** - Availability: 2.2.0 Computes the volume of a 3D geometry. Closed surfaces can have non-zero volume.
- **postgis.enable_outdb_rasters** - Availability: 2.2.0 データベース外ラスターのバンドにアクセスできるようにする、真偽型のコンフィギュレーションオプション。
- **postgis.gdal_datapath** - Availability: 2.2.0 GDAL の GDAL_DATA オプションの値を設定するためのコンフィギュレーションオプションです。設定しない場合には、GDAL_DATA 環境変数が使われます。
- **postgis.gdal_enabled_drivers** - Availability: 2.2.0 PostGIS 環境で GDAL ドライバーを有効にするコンフィギュレーションオプションです。GDAL コンフィギュレーション変数 GDAL_SKIP に影響を与えます。
- **|=|** - Availability: 2.2.0. A トラジェクトリと B トラジェクトリとの最接近する時の距離を返します。

PostGIS 2.2 で機能強化された関数

- **<->** - Enhanced: 2.2.0 -- 真の KNN (k 近傍法) のジオメトリとジオグラフィーのふるまい。ジオグラフィーの KNN は回転楕円体でなく球面に基づきます。A と B の 2 次元距離を返します。
- **AsTopoJSON** - Enhanced: 2.2.1 点入力に対応するようになりました opoGeometry の TopoJSON 表現を返します。
- **ST_Area** - Enhanced: 2.2.0 - 精度とロバスト性の向上のために GeographicLib を使って回転楕円体面上での計測を行うようにしています。この新機能を使うには、Proj 4.9.0 以上が必要です。ポリゴンジオメトリの面積を返します。
- **ST_AsX3D** - Enhanced: 2.2.0: GeoCoordinates と軸 (x/y, 経度/緯度) の反転に対応しました。詳細は options を見て下さい。ジオメトリを X3D ノード要素書式 (ISO-IEC-19776-1.2-X3DEncodings-XML) で返します。
- **ST_Azimuth** - Enhanced: 2.2.0 - 精度とロバスト性の向上のために GeographicLib を使って回転楕円体面上での計測を行うようにしています。この新機能を使うには、Proj 4.9.0 以上が必要です。北を基準とした 2 点間の線の方位角を返します。
- **ST_Distance** - Enhanced: 2.2.0 - 精度とロバスト性の向上のために GeographicLib を使って回転楕円体面上での計測を行うようにしています。この新機能を使うには、Proj 4.9.0 以上が必要です。二つのジオメトリ値またはジオグラフィ値間の距離を返します。
- **ST_Scale** - Enhanced: 2.2.0 全ての次元の拡大縮小 (factor パラメータ) への対応が導入されました。与えた係数でジオメトリを拡大縮小します。
- **ST_Split** - Enhanced: 2.2.0 ライン分割をマルチライン、マルチポイントまたはポリゴンもしくはマルチポリゴンの境界で行えるようにしました。ジオメトリを他のジオメトリで分割してできたジオメトリのコレクションを返します。

- **ST_Summary** - Enhanced: 2.2.0 TIN と曲線の対応が追加されました。ジオメトリについての要約文を返します。

PostGIS 2.2 で変更された関数

- **<->** - Changed: 2.2.0 -- 真の KNN が有効になると、古いハイブリッド法による回避策は遅くなることがあります。下の例を見てください。A と B の 2 次元距離を返します。
- **ST_3DClosestPoint** - Changed: 2.2.0 - 二つの 2 次元ジオメトリが入力である場合には、2 次元ポイントが返ります (古い挙動では、存在しない Z の値について 0 を仮定していました)。2 次元と 3 次元の場合には、もはや、存在しない Z の値について 0 を仮定しません。g1 上の、g2 に最も近い 3 次元ポイントを返します。これは 3 次元の最短ラインの始点です。
- **ST_3DDistance** - Changed: 2.2.0 - 2 次元と 3 次元の場合には、もはや、存在しない Z の値について 0 を仮定しません。投影座標系の単位で、二つのジオメトリ間の 3 次元デカルト距離の最小値を返します (空間参照系に基づきます)。
- **ST_3DLongestLine** - Changed: 2.2.0 - 二つの 2 次元ジオメトリが入力である場合には、2 次元ポイントが返ります (古い挙動では、存在しない Z の値について 0 を仮定していました)。2 次元と 3 次元の場合には、もはや、存在しない Z の値について 0 を仮定しません。二つのジオメトリ間の 3 次元最長ラインを返します。
- **ST_3DMaxDistance** - Changed: 2.2.0 - 2 次元と 3 次元の場合には、もはや、存在しない Z の値について 0 を仮定しません。二つのジオメトリ間の 3 次元最大デカルト距離 (空間参照系に基づく) を空間参照系の単位で返します。
- **ST_3DShortestLine** - Changed: 2.2.0 - 二つの 2 次元ジオメトリが入力である場合には、2 次元ポイントが返ります (古い挙動では、存在しない Z の値について 0 を仮定していました)。2 次元と 3 次元の場合には、もはや、存在しない Z の値について 0 を仮定しません。二つのジオメトリの 3 次元の最短ラインを返します。
- **ST_DistanceSphere** - Changed: 2.2.0 前の版ではこの関数は ST_Distance_Sphere と呼ばれていました球面の地球モデルを使って、二つの経度/緯度ジオメトリの最小距離をメートル単位で返します。
- **ST_DistanceSpheroid** - Changed: 2.2.0 前の版ではこの関数は ST_Distance_Sphere と呼ばれていました回転楕円体面の地球モデルを使って、二つの経度/緯度ジオメトリの最小距離を返します。
- **ST_Equals** - Changed: 2.2.0 この関数は、どちらのジオメトリも不正であっても、バイナリで同じ場合なら TRUE を返します。二つのジオメトリが同じ点集合になっているかテストします。
- **ST_LengthSpheroid** - Changed: 2.2.0 これより前の版では、これは ST_Length_Spheroid と呼ばれ、ST_3DLength_Spheroid という別名を持っていました。回転楕円体面上の経度緯度のジオメトリの 2 次元または 3 次元の長さ/周長を返します。
- **ST_MemSize** - Changed: 2.2.0 命名規則に従うために ST_MemSize に変更しました。ジオメトリが取るメモリ空間の合計を返します。
- **ST_PointInsideCircle** - Changed: 2.2.0 前のバージョンでは ST_Point_Inside_Circle と呼ばれていました。ポイントジオメトリが中心と半径で定められた円の内側にあるかをテストします。
- **ValidateTopology** - Changed: 2.2.0 エラーの記述と矛盾しないように 'edge crosses node' の id1 と id2 の値が入れ替わっています。トポロジの問題についての詳細を示す validate_topology_return_type の集合を返します。

13.12.13 PostGIS 2.1 で新規作成/機能強化された関数

次に示す関数は、新規作成または機能強化された PostGIS 関数です。

PostGIS 2.1 で新規作成された関数

- **=** - Availability: 2.1.0 A のバウンディングボックスが B のバウンディングボックスと同じ場合に TRUE を返します。倍精度浮動小数点数のバウンディングボックスを使います。

- **AsTopoJSON** - Availability: 2.1.0 opoGeometry の TopoJSON 表現を返します。
 - **ST_3DArea** - Availability: 2.1.0 3 次元の面ジオメトリの面積を計算します。立体の場合は 0 を返します。
 - **ST_3DIntersection** - Availability: 2.1.0 3 次元のインタセクトした (共有する) 部分を計算します。
 - **ST_Box2dFromGeoHash** - Availability: 2.1.0 GeoHash 文字列から BOX2D を返します。
 - **ST_ColorMap** - Availability: 2.1.0 元のラスタと指定したバンドから 4 個までの 8BUI バンド (grayscale, RGB, RGBA) からなる新しいラスタを生成します。
 - **ST_Contains** - Availability: 2.1.0 rastA の外に rastB の点が無く、rastA の内部に rastB の内部の点の一つ以上ある場合に TRUE を返します。
 - **ST_ContainsProperly** - Availability: 2.1.0 rastB が rastA の内部でインタセクトし、かつ rastA の境界とも外部ともインタセクトしない場合に TRUE を返します。
 - **ST_CoveredBy** - Availability: 2.1.0 rastA が rastB の外部に点を持たない場合に TRUE を返します。
 - **ST_Covers** - Availability: 2.1.0 rastB が rastA の外部に点を持たない場合に TRUE を返します。
 - **ST_DFullyWithin** - Availability: 2.1.0 rastA と rastB が指定した距離内に完全に収まる場合に TRUE を返します。
 - **ST_DWithin** - Availability: 2.1.0 rastA と rastB が指定した距離内にある場合に TRUE を返します。
 - **ST_DelaunayTriangles** - Availability: 2.1.0 ジオメトリの頂点のドロネ三角形を返します。
 - **ST_Disjoint** - Availability: 2.1.0 rastA が rastB とインタセクトしない場合に TRUE を返します。
 - **ST_DumpValues** - Availability: 2.1.0 指定したバンドの値を 2 次元で得ます。
 - **ST_Extrude** - Availability: 2.1.0 関連するボリュームにサーフェスを押し出します。
 - **ST_ForceLHR** - Availability: 2.1.0 LHR (Left Hand Rule) 方向に強制します。
 - **ST_FromGDALRaster** - Availability: 2.1.0 対応する GDAL ラスタファイルからラスタを返します。
 - **ST_GeomFromGeoHash** - Availability: 2.1.0 GeoHash 文字列からジオメトリを返します。
 - **ST_InvDistWeight4ma** - Availability: 2.1.0 近隣のピクセル値の内挿補間を行うラスタ処理関数です。
 - **ST_MapAlgebra (callback function version)** - Availability: 2.1.0 コールバック関数版 - 一つ以上の入力ラスタ、バンドインデックスと一つのユーザ定義コールバック関数から、一つのバンドからなるラスタを返します。
 - **ST_MapAlgebra (expression version)** - Availability: 2.1.0 数式版 - 一つか二つの入力ラスタ、バンド番号、一つ以上のユーザ定義 SQL 式から一つのバンドを持つラスタを返します。
 - **ST_MinConvexHull** - Availability: 2.1.0 NODATA 値を除いたラスタの凸包ジオメトリを返します。
 - **ST_MinDist4ma** - Availability: 2.1.0 対象ピクセルと値を持つ近隣ピクセルとの最短距離をピクセル単位で返すラスタ処理関数です。
 - **ST_MinkowskiSum** - Availability: 2.1.0 ミンコフスキー和を求めます。
 - **ST_NearestValue** - Availability: 2.1.0 与えられたバンドの、columnx と rowy で指定されるか、またはラスタと同じ空間参照系で表現されたポイントで指定されたピクセルに最も近い NODATA でない値を返します。
 - **ST_Neighborhood** - Availability: 2.1.0 与えられたバンドの columnX, columnY か、ラスタと同じ空間参照系のジオメトリポイントで指定されたピクセルの周囲にある、NODATA でない 2 次元倍精度浮動小数点数配列を返します。
 - **ST_NotSameAlignmentReason** - Availability: 2.1.0 ラスタが同じアラインメントを持つかどうか、また、持たない場合にはその理由を示す文字列を返します。
 - **ST_Orientation** - Availability: 2.1.0 サーフェスの方向を判定します。
-

- **ST_Overlaps** - Availability: 2.1.0 rastA と rastB がインタセクトして、かつ一方がもう一方に完全には包含されない場合には TRUE を返します。
 - **ST_PixelAsCentroid** - Availability: 2.1.0 ピクセルで表現される面の重心 (ポイントジオメトリ) を返します。
 - **ST_PixelAsCentroids** - Availability: 2.1.0 全てのピクセルについて重心 (ポイントジオメトリ) を、ピクセルごとのピクセル値とラスタ座標系の X と Y とを付けて返します。ポイントジオメトリの座標はピクセルで表現される面の重心です。
 - **ST_PixelAsPoint** - Availability: 2.1.0 ピクセルの左上隅のポイントジオメトリを返します。
 - **ST_PixelAsPoints** - Availability: 2.1.0 全てのピクセルについてポイントジオメトリを、ピクセルごとのピクセル値とラスタ座標系の X と Y とを付けて返します。ポイントジオメトリの座標はピクセルの左上隅です。
 - **ST_PixelOfValue** - Availability: 2.1.0 検索値と同じ値を持つピクセルの columnx, rowy ピクセル座標を得ます。
 - **ST_PointFromGeoHash** - Availability: 2.1.0 GeoHash 文字列からポイントを返します。
 - **ST_RasterToWorldCoord** - Availability: 2.1.0 ラスタの指定した列と行における左上隅の地理座標 X 値と Y 値 (経度と緯度) を返します。列と行の番号は 1 始まりです。
 - **ST_Resize** - Availability: 2.1.0 GDAL 1.6.1 以上が必要です。ラスタを新しい幅、高さにサイズ再設定を行います。
 - **ST_Roughness** - Availability: 2.1.0 DEM の「粗度」を計算したラスタを返します。
 - **ST_SetValues** - Availability: 2.1.0 与えられたバンドに複数の値を設定して、変更されたラスタを返します。
 - **ST_Simplify** - Availability: 2.1.0 与えた TopoGeometry を「シンプル化した」ジオメトリを返します。ダグラス-ポーカーのアルゴリズムを使います。
 - **ST_StraightSkeleton** - Availability: 2.1.0 ジオメトリからストレートスケルトンを計算します。
 - **ST_Summary** - Availability: 2.1.0 ラスタの中身の概要が文字列で返されます。
 - **ST_TPI** - Availability: 2.1.0 地形的位置指数を計算したラスタを返します。
 - **ST_TRI** - Availability: 2.1.0 起伏指標を計算したラスタを返します。
 - **ST_Tessellate** - Availability: 2.1.0 ポリゴンまたは多面体サーフェスのテッセレーションを計算し、TIN または TIN コレクションを返します。
 - **ST_Tile** - Availability: 2.1.0 求められた出力ラスタのピクセル数に基づいて入力ラスタを分割した結果のラスタ集合を返します。
 - **ST_Touches** - Availability: 2.1.0 rastA と rastB が少なくとも一つの共通の点を持ち、かつ二つのラスタの内部同士がインタセクトしない場合に TRUE を返します。
 - **ST_Union** - Availability: 2.1.0 ST_Union(rast, unionarg) の形式が導入されました。ラスタタイルの集合を結合して 1 以上のバンドからなる単一ラスタを返します。
 - **ST_Within** - Availability: 2.1.0 rastA が rastB の外部に点を持たず、rastA の内部の少なくとも一つの点が rastB の内部にある場合に TRUE を返します。
 - **ST_WorldToRasterCoord** - Availability: 2.1.0 ラスタの空間参照系による地理座標の X 値と Y 値 (経度と緯度) またはポイントジオメトリに対応するピクセルの左上隅を返します。
 - **UpdateRasterSRID** - Availability: 2.1.0 ユーザが指定したカラムとテーブルにあるラスタの全てについて SRID を変更します。
 - **clearTopoGeom** - Availability: 2.1 TopoGeometry の中身を消去します。
 - **postgis_sfcgal_version** - Availability: 2.1.0 使用している SFCGAL のバージョンを返します
-

PostGIS 2.1 で機能強化された関数

- **ST_AddBand** - Enhanced: 2.1.0 `addbandarg` 対応が追加されました。与えられたタイプで、与えられた初期値にした新しいバンドを、与えられたインデックス位置に追加したラスタを返します。インデックス位置を指定していない場合には、バンドは末尾に追加されます。
- **ST_AddBand** - Enhanced: 2.1.0 `out-db` バンドが追加されました。与えられたタイプで、与えられた初期値にした新しいバンドを、与えられたインデックス位置に追加したラスタを返します。インデックス位置を指定していない場合には、バンドは末尾に追加されます。
- **ST_AsBinary/ST_AsWKB** - Enhanced: 2.1.0 `outasin` の追加ラスタの Well-Known Binary (WKB) 表現を返します。
- **ST_AsGML** - Enhanced: 2.1.0 GML 3 用に `id` が導入されました。GML 第 2 版または第 3 版としてジオメトリを返します。
- **ST_Aspect** - Enhanced: 2.1.0 `ST_MapAlgebra()` を使用するようし、`interpolate_nodata` 任意引数を追加しました。標高ラスタバンドの傾斜方向 (デフォルトの単位は度) を返します。地形解析に使えます。
- **ST_Boundary** - Enhanced: 2.1.0 三角対応が導入されました。ジオメトリの境界を返します。
- **ST_Clip** - Enhanced: 2.1.0 C 言語で記述されました入力ジオメトリで切り取ったラスタを返します。バンドが指定されていない場合には、全てのバンドが処理されます。`crop` が指定されていなかったり `TRUE` となっている場合には、出力ラスタは切り取られます。`touched` が `TRUE` の場合には、接触するピクセルは取り込まれ、`TRUE` でない場合には、中心がジオメトリ内にあるピクセルだけが取り込まれます。
- **ST_DWithin** - Enhanced: 2.1.0 ジオグラフィ版の速度が改善しました。詳細については [Making Geography faster](#) をご覧下さい。二つのジオメトリが与えられた距離内にあるかどうかをテストします。
- **ST_DWithin** - Enhanced: 2.1.0 曲線ジオメトリ対応が導入されました。二つのジオメトリが与えられた距離内にあるかどうかをテストします。
- **ST_Distance** - Enhanced: 2.1.0 ジオグラフィ版の速度が改善しました。詳細については [Making Geography faster](#) をご覧下さい。二つのジオメトリ値またはジオグラフィ値間の距離を返します。
- **ST_Distance** - Enhanced: 2.1.0 - 曲線ジオメトリ対応が導入されました。二つのジオメトリ値またはジオグラフィ値間の距離を返します。
- **ST_Distinct4ma** - Enhanced: 2.1.0 二つ目の形式の追加近隣のピクセル値のうち一意となるものを数えるラスタ処理関数です。
- **ST_DumpPoints** - Enhanced: 2.1.0 速度向上しました。C 言語で実装しなおしました。ジオメトリ内の座標の行である `geometry_dump` 行の集合を返します。
- **ST_HillShade** - Enhanced: 2.1.0 `ST_MapAlgebra()` を使用するようし、`interpolate_nodata` 任意引数を追加しました。与えられた方位、高度、明度、スケールの入力を使って標高ラスタバンドの仮想照明を返します。
- **ST_MakeValid** - Enhanced: 2.1.0 `GEOMETRYCOLLECTION` と `MULTIPOINT` の対応の追加頂点を失うことなしに不正なジオメトリを妥当なジオメトリにしようと試みます。
- **ST_Max4ma** - Enhanced: 2.1.0 二つ目の形式の追加近隣のピクセル値の最大値を計算するラスタ処理関数です。
- **ST_Mean4ma** - Enhanced: 2.1.0 二つ目の形式の追加近隣のピクセル値の平均値を計算するラスタ処理関数です。
- **ST_Min4ma** - Enhanced: 2.1.0 二つ目の形式の追加近隣のピクセル値の最小値を計算するラスタ処理関数です。
- **ST_PixelAsPolygons** - Enhanced: 2.1.0 任意引数 `exclude_nodata_value` が追加されました。全てのピクセルについて境界となるジオメトリを、ピクセルごとのピクセル値とラスタ座標系の X と Y とを付けて返します。

- **ST_Polygon** - Enhanced: 2.1.0 速度を改善し (完全に C 言語で記述しました)、確実に妥当なマルチポリゴンを返すようにしました。NODATA 値でないピクセル値を持つピクセルの結合で形成されるマルチポリゴンジオメトリを返します。バンドを指定しない場合のデフォルトは 1 です。
- **ST_Range4ma** - Enhanced: 2.1.0 二つ目の形式の追加近隣のピクセル値の範囲を計算するラスタ処理関数です。
- **ST_SameAlignment** - Enhanced: 2.1.0 集約関数版の追加ラスタが同じスキュー、スケール、空間参照系、オフセットを持つ (ピクセルが分割されることなく同じグリッドに置かれている) 場合に TRUE を返し、そうでない場合は問題を詳述する通知とともに FALSE を返します。
- **ST_Segmentize** - Enhanced: 2.1.0 ジオグラフィ対応が導入されました。与えた長さを超える線分を持たないよう変更したジオメトリ/ジオグラフィを返します。
- **ST_SetGeoReference** - Enhanced: 2.1.0 ST_SetGeoReference(raster, double precision, ...) 形式を追加しました。地理参照 6 パラメーターを一度に設定します。数値は空白で区切ります。GDAL または ESRI 書式の入力を受け付けます。デフォルトは GDAL です。
- **ST_SetValue** - Enhanced: 2.1.0 ST_SetValue でジオメトリを用いる形式が、ポイントだけでなくあらゆるジオメトリタイプに対応するようになりました。ジオメトリを用いる形式は ST_SetValues の geomval[] を用いる形式をラップしたものです。与えられたバンドの columnX, columnY か、ラスタと同じ空間参照系のジオメトリポイントで指定されたピクセルの値または指定したジオメトリとインタセクトするピクセル群の値を設定することから得られる、変更されたラスタを返します。バンド番号は 1 始まりで、指定しない場合には、1 番と仮定します。
- **ST_Slope** - Enhanced: 2.1.0 ST_MapAlgebra() を使用するようにし、units, scale, interpolate_nodata 任意引数を追加しました。標高ラスタバンドの傾斜角 (デフォルトでは度単位) を返します。地形解析に使えます。
- **ST_StdDev4ma** - Enhanced: 2.1.0 二つ目の形式の追加近隣のピクセル値の標準偏差を計算するラスタ処理関数です。
- **ST_Sum4ma** - Enhanced: 2.1.0 二つ目の形式の追加近隣のピクセル値の合計を計算するラスタ処理関数です。
- **ST_Summary** - Enhanced: 2.1.0 空間参照系を持つかを示す S フラグが追加されました。ジオメトリについての要約文を返します。
- **ST_Transform** - Enhanced: 2.1.0 ST_Transform(rast, alignto) の形式を追加しました。ラスタを既知の空間参照系から他の既知の空間参照系に、指定したリサンプリングアルゴリズムで投影変換します。新しいピクセル値のリサンプリングアルゴリズムとして最近傍補間 ('NearestNeighbor' (米式綴り方))、双線形補間 ('Bilinear')、3 次補間 ('Cubic')、3 次スプライン補間 ('CubicSpline')、ランチョス補間 ('Lanczos') を用います。デフォルトは最近傍補間です。
- **ST_Union** - Enhanced: 2.1.0 速度が改善されました (完全に C 言語で記述しました)。ラスタタイルの集合を結合して 1 以上のバンドからなる単一ラスタを返します。
- **ST_Union** - Enhanced: 2.1.0 ST_Union(rast) (一つ目の形式) で、全ての入力ラスタの全てのバンドを結合するようになりました。以前の版の PostGIS では、一つ目のバンドと仮定していました。ラスタタイルの集合を結合して 1 以上のバンドからなる単一ラスタを返します。
- **ST_Union** - Enhanced: 2.1.0 ST_Union(rast, uniontype) (四つ目の形式) で、全ての入力ラスタの全てのバンドを結合するようになりました。ラスタタイルの集合を結合して 1 以上のバンドからなる単一ラスタを返します。
- **toTopoGeom** - Enhanced: 2.1.0 版では、既存の TopoGeometry を取る形式が追加されました。単純なジオメトリから TopoGeometry を生成します。

PostGIS 2.1 で変更された関数

- **ST_Aspect** - Changed: 2.1.0 以前の版では、返り値はラジアン単位でした。現在は、デフォルトでは度で返します。標高ラスタバンドの傾斜方向 (デフォルトの単位は度) を返します。地形解析に使えます。

- **ST_EstimatedExtent** - Changed: 2.1.0 2.0.x までは ST_Estimated_Extent と呼ばれていました。空間テーブルの推定範囲を返します。
- **ST_Force2D** - Changed: 2.1.0 2.0.x の間は ST_Force_2D と呼ばれていました。ジオメトリを 2 次元モードに強制します。
- **ST_Force3D** - Changed: 2.1.0 2.0.x の間は ST_Force_3D と呼ばれていました。ジオメトリを XYZ モードに強制します。これは ST_Force3DZ の別名です。
- **ST_Force3DM** - Changed: 2.1.0 2.0.x の間は ST_Force_3DM と呼ばれていました。ジオメトリを XYM モードに強制します。
- **ST_Force3DZ** - Changed: 2.1.0 2.0.x の間は ST_Force_3DZ と呼ばれていました。ジオメトリを XYZ モードに強制します。
- **ST_Force4D** - Changed: 2.1.0 2.0.x の間は ST_Force_4D と呼ばれていました。ジオメトリを XYZM モードに強制します。
- **ST_ForceCollection** - Changed: 2.1.0 2.0.x の間は ST_Force_Collection と呼ばれていました。ジオメトリをジオメトリコレクションに変換します。
- **ST_HillShade** - Changed: 2.1.0 以前の版では azimuth と altitude はラジアン単位で表現しました。現在は azimuth と altitude は度単位で表現します。与えられた方位、高度、明度、スケールの入力を使って標高ラスタバンドの仮想照明を返します。
- **ST_LineInterpolatePoint** - Changed: 2.1.0 2.0.x までは ST_Line_Interpolate_Point と呼んでいました。ラインに沿って、割合で示された位置の補間ポイントを返します。
- **ST_LineLocatePoint** - Changed: 2.1.0 2.0.x までは ST_Line_Locate_Point と呼んでいました。ポイントに最も近いライン上のポイントの位置を割合で返します。
- **ST_LineSubstring** - Changed: 2.1.0 2.0.x では ST_Line_Substring と呼ばれていました。二つの割合位置からラインの一部を返します。
- **ST_PixelAsCentroids** - Changed: 2.1.1 exclude_nodata_value の挙動を変更しました。全てのピクセルについて重心 (ポイントジオメトリ) を、ピクセルごとのピクセル値とラスタ座標系の X と Y とを付けて返します。ポイントジオメトリの座標はピクセルで表現される面の重心です。
- **ST_PixelAsPoints** - Changed: 2.1.1 exclude_nodata_value の挙動を変更しました。全てのピクセルについてポイントジオメトリを、ピクセルごとのピクセル値とラスタ座標系の X と Y とを付けて返します。ポイントジオメトリの座標はピクセルの左上隅です。
- **ST_PixelAsPolygons** - Changed: 2.1.1 exclude_nodata_value の挙動を変更しました。全てのピクセルについて境界となるジオメトリを、ピクセルごとのピクセル値とラスタ座標系の X と Y とを付けて返します。
- **ST_Polygon** - Changed: 2.1.0 以前の版では、時々ポリゴンを返しましたが、常にマルチポリゴンを返すように変更しました。NODATA 値でないピクセル値を持つピクセルの結合で形成されるマルチポリゴンジオメトリを返します。バンドを指定しない場合のデフォルトは 1 です。
- **ST_RasterToWorldCoordX** - Changed: 2.1.0 以前の版では ST_Raster2WorldCoordX と呼ばれていました。ラスタの指定した列と行における左上隅の地理座標の X 値を返します。列と行の番号は 1 始まりです。
- **ST_RasterToWorldCoordY** - Changed: 2.1.0 以前の版では ST_Raster2WorldCoordY と呼ばれていました。ラスタの指定した列と行における左上隅の地理座標の Y 値を返します。列と行の番号は 1 始まりです。
- **ST_Rescale** - Changed: 2.1.0 SRID なしのラスタで動作するようになりました。スケール (ピクセルサイズ) だけを調整するリサンプリングを行います。新しいピクセル値のリサンプリングアルゴリズムとして最近傍補間 ('NearestNeighbor' (英語または米式綴り方))、双線形補間 ('Bilinear')、3 次補間 ('Cubic')、3 次スプライン補間 ('CubicSpline')、ランチョス補間 ('Lanczos') を用います。デフォルトは最近傍補間です。
- **ST_Reskew** - Changed: 2.1.0 SRID なしのラスタで動作するようになりました。キュー (回転パラメータ) だけを調整するリサンプリングを行います。新しいピクセル値のリサンプリングアルゴリズムとして最近傍補間 ('NearestNeighbor' (米式綴り方))、双線形補間 ('Bilinear')、3 次補間 ('Cubic')、3 次スプライン補間 ('CubicSpline')、ランチョス補間 ('Lanczos') を用います。デフォルトは最近傍補間です。

- **ST_Segmentize** - Changed: 2.1.0 ジオグラフィ対応の導入の結果、ST_Segmentize('LINESTRING(1 2, 3 4)', 0.5) とすると、あいまい関数エラーが発生します。入力ではジオメトリかジオグラフィかを確実に指定する必要があります。ST_GeomFromText、ST_GeogFromText、使いたい型へのキャスト (例: ST_Segmentize('LINESTRING(1 2, 3 4)::geometry, 0.5)) を行います与えた長さを超える線分を持たないよう変更したジオメトリ/ジオグラフィを返します。
- **ST_Slope** - Changed: 2.1.0 以前の版では、返り値はラジアン単位でした。現在は、デフォルトでは度で返します。標高ラスタブンドの傾斜角 (デフォルトでは度単位) を返します。地形解析に使えます。
- **ST_SnapToGrid** - Changed: 2.1.0 SRID なしのラスタで動作するようになりました。グリッドにスナップすることでラスタをリサンプリングします。新しいピクセル値のリサンプリングアルゴリズムとして最近傍補間 ('NearestNeighbor' (米式綴り方))、双線形補間 ('Bilinear')、3 次補間 ('Cubic')、3 次スプライン補間 ('CubicSpline')、ランチョス補間 ('Lanczos') を用います。デフォルトは最近傍補間です。
- **ST_WorldToRasterCoordX** - Changed: 2.1.0 以前の版では ST_World2RasterCoordX と呼ばれていました。ラスタの空間参照系に基づくポイントジオメトリ (pt) または X,Y 座標値 (xw,yw) に対応するラスタの列を返します。
- **ST_WorldToRasterCoordY** - Changed: 2.1.0 以前の版では ST_World2RasterCoordY と呼ばれていました。ラスタの空間参照系に基づくポイントジオメトリ (pt) または X,Y 座標値 (xw,yw) に対応するラスタの行を返します。

13.12.14 PostGIS 2.0 で新規作成/機能強化された関数

次に示す関数は、新規作成または機能強化された PostGIS 関数です。

PostGIS 2.0 で新規作成された関数

- **&&** - Availability: 2.0.0 A のバウンディングボックスが B のバウンディングボックスとインタセクトする場合に TRUE を返します。
- **&&&** - Availability: 2.0.0 A の n 次元バウンディングボックスが B の n 次元バウンディングボックスとインタセクトする場合に TRUE を返します。
- **<#>** - Availability: 2.0.0. A のバウンディングボックスと B のバウンディングボックスの 2 次元距離を返します。
- **<->** - Availability: 2.0.0 -- 弱い KNN によって、実際の距離の代わりにジオメトリの重心に基づく近傍が得られます。ポイントは確実な結果を得て、他のタイプは全て不確実な結果を得ます。A と B の 2 次元距離を返します。
- **@** - Availability: 2.0.0 raster @ raster, raster @ geometry が導入されました。A のバウンディングボックスが B のバウンディングボックスに含まれる場合に TRUE を返します。倍精度浮動小数点数のバウンディングボックスを使います。
- **@** - Availability: 2.0.5 geometry @ raster が導入されました。A のバウンディングボックスが B のバウンディングボックスに含まれる場合に TRUE を返します。倍精度浮動小数点数のバウンディングボックスを使います。
- **AddEdge** - Availability: 2.0.0 指定したラインストリングジオメトリを使って、ラインストリングエッジをエッジテーブルに追加し、指定したトポロジスキーマの始点終点をポイントノードテーブルに追加し、新しい (または既存の) エッジの識別番号を返します。
- **AddFace** - Availability: 2.0.0 フェイスプリミティブをトポロジに登録し、その識別番号を得ます。
- **AddNode** - Availability: 2.0.0 指定したトポロジスキーマのノードテーブルにポイントノードを追加し、新しいノードの識別番号を返します。指定したポイントに既にノードがある場合は既存のノード識別番号を返します。
- **AddOverviewConstraints** - Availability: 2.0.0 ラスタカラムに対して、他のオーバビューであることをタグ付けします。

- **AddRasterConstraints** - Availability: 2.0.0 ロードされたラスタテーブルの特定の列にラスタ制約を追加します。制約には空間参照系、スケール、ブロックサイズ、アラインメント、バンド数、バンド型、ラスタ列が規則正しいブロックかどうかを示すフラグがあります。テーブルは制約が推論されるためのデータがロードされなければなりません。制約の設定が完了すると `true` を返し、問題があると通知を返します。
 - **AsGML** - Availability: 2.0.0 TopoGeometry の GML 表現を返します。
 - **CopyTopology** - Availability: 2.0.0 トポロジ (ノード、エッジ、フェイス、レイヤと TopoGeometry) の新しいスキーマに複製します。
 - **DropOverviewConstraints** - Availability: 2.0.0 ラスタ列に対して他のオーバビューであることをタグ付けしているのを解除します。
 - **DropRasterConstraints** - Availability: 2.0.0 ラスタテーブル列を参照する PostGIS ラスタ制約を削除します。データの再読み込みやラスタ列データの更新の際に使えます。
 - **GetEdgeByPoint** - Availability: 2.0.0 与えられたポイントにインタセクトするエッジの識別番号を探索します。
 - **GetFaceByPoint** - Availability: 2.0.0 与えたポイントとインタセクトするフェイスを見つけます。
 - **GetNodeByPoint** - Availability: 2.0.0 ポイント位置にあるノードの識別番号を探索します。
 - **GetNodeEdges** - Availability: 2.0 与えられたノードに付随するエッジの集合を整理して返します。
 - **GetRingEdges** - Availability: 2.0.0 与えられた側を歩いて得られた、正負符号付きエッジ識別番号の集合を、順序通りに返します。
 - **GetTopoGeomElements** - Availability: 2.0.0 与えられた TopoGeometry (プリミティブ要素) の、トポロジの `element_id` と `element_type` を含む `topoelement` オブジェクトの集合を返します。
 - **GetTopologySRID** - Availability: 2.0.0 トポロジ名から `topology.topology` テーブル内にあるトポロジの SRID を返します。
 - **Polygonize** - Availability: 2.0.0 トポロジエッジで定義される全てのフェイスを探索し、追加します。
 - **ST_3DClosestPoint** - Availability: 2.0.0 g1 上の、g2 に最も近い 3 次元ポイントを返します。これは 3 次元の最短ラインの始点です。
 - **ST_3DDFullyWithin** - Availability: 2.0.0 二つの 3 次元ジオメトリが完全に与えられた 3 次元距離内にあるかどうかをテストします。
 - **ST_3DDWithin** - Availability: 2.0.0 二つの 3 次元ジオメトリが与えられた 3 次元距離内にあるかどうかをテストします。
 - **ST_3DDistance** - Availability: 2.0.0 投影座標系の単位で、二つのジオメトリ間の 3 次元デカルト距離の最小値を返します (空間参照系に基づきます)。
 - **ST_3DIntersects** - Availability: 2.0.0 二つのジオメトリが 3 次元空間において空間的にインタセクトするかどうかをテストします。ポイント、ラインストリング、ポリゴン、多面体サーフェス (面) についてのみ動作します。
 - **ST_3DLongestLine** - Availability: 2.0.0 二つのジオメトリ間の 3 次元最長ラインを返します。
 - **ST_3DMaxDistance** - Availability: 2.0.0 二つのジオメトリ間の 3 次元最大デカルト距離 (空間参照系に基づく) を空間参照系の単位で返します。
 - **ST_3DShortestLine** - Availability: 2.0.0 二つのジオメトリの 3 次元の最短ラインを返します。
 - **ST_AddEdgeModFace** - Availability: 2.0 新しいエッジを追加します。新しいエッジがフェイスを分割する場合には、もとのフェイスを編集し、一つのフェイスを追加します。
 - **ST_AddEdgeNewFaces** - Availability: 2.0 新しいエッジを追加します。新しいエッジがフェイスを分割する場合には、もとのフェイスを削除して、分割した二つのフェイスに置き換えます。
-

- **ST_AsGDALRaster** - Availability: 2.0.0 - GDAL 1.6.0 以上が必要です。指定された GDAL ラスタ書式でラスタスタイルを返します。ラスタ書式はコンパイルしたライブラリが対応するものです。ライブラリが対応する書式の一覧を得るには `ST_GDALRasters()` を使います。
- **ST_AsJPEG** - Availability: 2.0.0 - GDAL 1.6.0 以上が必要です。ラスタの選択されたバンドを、単一の Joint Photographic Experts Group (JPEG) 画像としてバイト配列で返します。バンドを指定せず、1 バンドか 3 より多いバンドがある場合には、1 番バンドを使用します。3 バンドのみ指定した場合には、3 バンドを使用し、RGB に対応付けます。
- **ST_AsLatLonText** - Availability: 2.0 与えられたポイントの度・分・秒表現を返します。
- **ST_AsPNG** - Availability: 2.0.0 - GDAL 1.6.0 以上が必要です。ラスタの選択されたバンドを、単一の portable network graphics (PNG) 画像としてバイト配列で返します。バンドを指定せず、1 バンドか 3 バンドか 4 バンドある場合には、全てのバンドを使用します。バンドを指定せず、2 バンドか 4 より多いバンドがある場合には、1 番バンドを使用します。対象バンドは RGB または RGBA に対応付けられます。
- **ST_AsRaster** - Availability: 2.0.0 - GDAL 1.6.0 以上が必要です。PostGIS ジオメトリを PostGIS ラスタに変換します。
- **ST_AsTIFF** - Availability: 2.0.0 - GDAL 1.6.0 以上が必要です。ラスタの選択されたバンドを、単一の TIFF 画像 (バイト配列) として返します。バンドを指定しないか指定したバンドがラスタ内に無い場合には、全てのバンドの使用を試みます。
- **ST_AsX3D** - Availability: 2.0.0: ISO-IEC-19776-1.2-X3DEncodings-XML ジオメトリを X3D ノード要素書式 (ISO-IEC-19776-1.2-X3DEncodings-XML) で返します。
- **ST_Aspect** - Availability: 2.0.0 標高ラスタバンドの傾斜方向 (デフォルトの単位は度) を返します。地形解析に使えます。
- **ST_Band** - Availability: 2.0.0 既存のラスタの、一つ以上のバンドを新しいラスタとして返します。既存のラスタから新しいラスタを構築する際に使えます。
- **ST_BandIsNoData** - Availability: 2.0.0 指定したバンドが NODATA 値だけで満たされている場合には、TRUE を返します。
- **ST_Clip** - Availability: 2.0.0 入力ジオメトリで切り取ったラスタを返します。バンドが指定されていない場合には、全てのバンドが処理されます。`crop` が指定されていなかったり TRUE となっている場合には、出力ラスタは切り取られます。`touched` が TRUE の場合には、接触するピクセルは取り込まれ、TRUE でない場合には、中心がジオメトリ内にあるピクセルだけが取り込まれます。
- **ST_CollectionHomogenize** - Availability: 2.0.0 ジオメトリコレクションを与えると、最も単純な表現を返します。
- **ST_ConcaveHull** - Availability: 2.0.0 全ての入力ジオメトリの頂点を含む凹ジオメトリを計算します。
- **ST_Count** - Availability: 2.0.0 ラスタまたはラスタカバレッジの指定したバンドのピクセル数を返します。バンドを指定しない場合には、1 番と仮定します。`exclude_nodata_value` を TRUE に設定している場合には、NODATA 値と等しくないピクセルのみを数えます。
- **ST_CreateTopoGeo** - Availability: 2.0 空のトポロジにジオメトリのコレクションを追加し、成否を示すメッセージを返します。
- **ST_Distinct4ma** - Availability: 2.0.0 近隣のピクセル値のうち一意となるものを数えるラスタ処理関数です。
- **ST_FlipCoordinates** - Availability: 2.0.0 X 値と Y 値を入れ替えたジオメトリを返します。
- **ST_GDALDrivers** - Availability: 2.0.0 - GDAL 1.6.0 以上が必要です。使用している GDAL ライブラリが対応するラスタ書式の一覧を返します。この一覧で `can_write=True` となっているものだけが **ST_AsGDALRaster** で使えます。
- **ST_GeomFromGeoJSON** - Availability: 2.0.0 JSON-C 0.9 以上が必要です。ジオメトリの GeoJSON 表現を入力として、PostGIS ジオメトリオブジェクトを出力します。

- **ST_GetFaceEdges** - Availability: 2.0 順序番号を含む、`aface` の境界となる、整列したエッジの集合を返します。
- **ST_HasNoBand** - Availability: 2.0.0 指定したバンド番号のバンドが無い場合には、TRUE を返します。バンド番号を指定していない場合には、1 番と仮定します。
- **ST_HillShade** - Availability: 2.0.0 与えられた方位、高度、明度、スケールの入力を使って標高ラスタバンドの仮想照明を返します。
- **ST_Histogram** - Availability: 2.0.0 ラスタまたはラスタカバレッジのビン範囲で分割したデータ分布をまとめるヒストグラムの集合を返します。ビン数を指定しない場合には自動計算されます。
- **ST_InterpolatePoint** - Availability: 2.0.0 ジオメトリのポイントに最も近いポイント上の補間 M 値を返します。
- **ST_IsEmpty** - Availability: 2.0.0 ラスタが空 (幅が 0 で高さが 0) の場合には TRUE を返します。他の場合には、FALSE を返します。
- **ST_IsValidDetail** - Availability: 2.0.0 ジオメトリが妥当か、妥当でないなら理由と位置をそれぞれ示す `valid_detail` 行を返します。
- **ST_IsValidReason** - Availability: 2.0 フラグを取る形式。ジオメトリが妥当か否かを示す文字列を返し、不正な場合は理由を返します。
- **ST_MakeLine** - Availability: 2.0.0 - LINESTRING 入力要素への対応が導入されました POINT、MULTIPOINT、LINESTRING から LINESTRING を生成します。
- **ST_MakeValid** - Availability: 2.0.0 頂点を失うことなしに不正なジオメトリを妥当なジオメトリにしようと試みます。
- **ST_MapAlgebraExpr** - Availability: 2.0.0 1 バンド版: 入力バンドに対する妥当な PostgreSQL 代数演算で形成された、指定したピクセルタイプとなる 1 バンドラスタを生成します。バンドを指定しない場合には、1 番を仮定します。
- **ST_MapAlgebraExpr** - Availability: 2.0.0 2 バンド版: 二つの入力バンドに対する妥当な PostgreSQL 代数演算で形成された、指定したピクセルタイプとなる 1 バンドラスタを生成します。バンドを指定しない場合には、どちらも 1 番と仮定します。結果ラスタは、一つ目のラスタのアラインメント (スケール、スキュー、ピクセル角位置) にあわされます。範囲は "extenttype" 引数で定義されます。取りうる "extenttype" の値は INTERSECTION, UNION, FIRST, SECOND です。
- **ST_MapAlgebraFct** - Availability: 2.0.0 1 バンド版 - 入力バンドに対する妥当な PostgreSQL 関数で形成された、指定したピクセルタイプとなる 1 バンドラスタを生成します。バンドを指定しない場合には、1 番と仮定します。
- **ST_MapAlgebraFct** - Availability: 2.0.0 2 バンド版 - 二つの入力バンドに対する妥当な PostgreSQL 関数で形成された、指定したピクセルタイプとなる 1 バンドラスタを生成します。バンドを指定しない場合には、1 番と仮定します。"extenttype" のデフォルトは INTERSECTION です。
- **ST_MapAlgebraFctNgb** - Availability: 2.0.0 1 バンド版: ユーザ定義 PostgreSQL 関数を使用する最近傍地図代数関数です。入力ラスタバンドの近傍の値を与えた PL/pgSQL ユーザ定義関数の結果からなるラスタを返します。
- **ST_Max4ma** - Availability: 2.0.0 近隣のピクセル値の最大値を計算するラスタ処理関数です。
- **ST_Mean4ma** - Availability: 2.0.0 近隣のピクセル値の平均値を計算するラスタ処理関数です。
- **ST_Min4ma** - Availability: 2.0.0 近隣のピクセル値の最小値を計算するラスタ処理関数です。
- **ST_ModEdgeHeal** - Availability: 2.0 二つのエッジについて、接続しているノードを削除して修復します。1 番目のエッジを編集して、2 番目のエッジを削除します。削除されたノードの識別番号を返します。
- **ST_MoveIsoNode** - Availability: 2.0.0 トポロジ内の孤立ノードを別の位置に移動させます。新しい `apoint` ジオメトリがノードとして存在しているなら、エラーが投げられます。移動に関する説明を返します。

- **ST_NewEdgeHeal** - Availability: 2.0 二つのエッジについて、接続しているノードを削除して修復します。両方のエッジを削除し、1 番目のエッジと同じ方向のエッジに置き換えます。
- **ST_Node** - Availability: 2.0.0 ラインストリングの集合にノードを作成します。
- **ST_NumPatches** - Availability: 2.0.0 多面体サーフェスのフェイス数を返します。多面体でないジオメトリの場合には NULL を返します。
- **ST_OffsetCurve** - Availability: 2.0 与えられた距離と方面に入力ラインをずらしたラインを返します。
- **ST_PatchN** - Availability: 2.0.0 多面体サーフェスの N 番目のジオメトリ (フェイス) を返します。
- **ST_Perimeter** - Availability: 2.0.0 ジオグラフィ対応が導入されました。ポリゴンジオメトリまたはジオグラフィの境界の長さを返します。
- **ST_PixelAsPolygon** - Availability: 2.0.0 指定した行と列のピクセルの境界となるジオメトリを返します。
- **ST_PixelAsPolygons** - Availability: 2.0.0 全てのピクセルについて境界となるジオメトリを、ピクセルごとのピクセル値とラスタ座標系の X と Y とを付けて返します。
- **ST_Project** - Availability: 2.0.0 始点から距離と方位で算出されたポイントを返します。
- **ST_Quantile** - Availability: 2.0.0 ラスタまたはラスタテーブルカバレッジのサンプルまたは母集団の分位数を計算します。値がラスタの 25%, 50%, 75% にあるかを調べることができます。
- **ST_Range4ma** - Availability: 2.0.0 近隣のピクセル値の範囲を計算するラスタ処理関数です。
- **ST_Reclass** - Availability: 2.0.0 元のラスタから再分類したバンドタイプからなるラスタを生成します。nband は変更するバンドです。nband が指定されていない場合には、1 と仮定します。他の全てのバンドは変更せずに返します。可視画像の書式としてより単純な描画を行うために、16BUI バンドを 8BUI バンドに変換する、等のために使います。
- **ST_RelateMatch** - Availability: 2.0.0 DE-9IM 交差行列が交差行列パターンに合致するかどうかを見ます。
- **ST_RemEdgeModFace** - Availability: 2.0 エッジを削除します。削除されるエッジが二つのフェイスを分割していたなら、一方のフェイスを削除し、もう一方のフェイスを両方の空間を覆うように編集します。
- **ST_RemEdgeNewFace** - Availability: 2.0 エッジを削除し、消去対象エッジでフェイスが二つに分割されているなら元の二つのフェイスを削除し、一つの新しいフェイスに置き換えます。
- **ST_Resample** - Availability: 2.0.0 GDAL 1.6.1 以上が必要です。指定したリサンプリングアルゴリズム、新しいピクセル範囲、グリッドの隅、定義するか他のラスタから借りてきた地理参照属性を使ってリサンプリングを行います。
- **ST_Rescale** - Availability: 2.0.0 GDAL 1.6.1 以上が必要です。スケール (ピクセルサイズ) だけを調整するリサンプリングを行います。新しいピクセル値のリサンプリングアルゴリズムとして最近傍補間 ('NearestNeighbor' (英語または米式綴り方))、双線形補間 ('Bilinear')、3 次補間 ('Cubic')、3 次スプライン補間 ('CubicSpline')、ランチョス補間 ('Lanczos') を用います。デフォルトは最近傍補間です。
- **ST_Reskew** - Availability: 2.0.0 GDAL 1.6.1 以上が必要です。キュー (回転パラメーター) だけを調整するリサンプリングを行います。新しいピクセル値のリサンプリングアルゴリズムとして最近傍補間 ('NearestNeighbor' (米式綴り方))、双線形補間 ('Bilinear')、3 次補間 ('Cubic')、3 次スプライン補間 ('CubicSpline')、ランチョス補間 ('Lanczos') を用います。デフォルトは最近傍補間です。
- **ST_SameAlignment** - Availability: 2.0.0 ラスタが同じスキュー、スケール、空間参照系、オフセットを持つ (ピクセルが分割されることなく同じグリッドに置かれている) 場合に TRUE を返し、そうでない場合は問題を詳述する通知とともに FALSE を返します。
- **ST_SetBandIsNoData** - Availability: 2.0.0 バンドの isnodata フラグを TRUE にします。
- **ST_SharedPaths** - Availability: 2.0.0 二つの LINESTRING/MULTILINESTRING の入力共有するパスのコレクションを返します。
- **ST_Slope** - Availability: 2.0.0 標高ラスタバンドの傾斜角 (デフォルトでは度単位) を返します。地形解析に使えます。

- **ST_Snap** - Availability: 2.0.0 入力ジオメトリの辺と頂点を参照ジオメトリの頂点にスナップします。
- **ST_SnapToGrid** - Availability: 2.0.0 GDAL 1.6.1 以上が必要です。グリッドにスナップすることでラスタをリサンプリングします。新しいピクセル値のリサンプリングアルゴリズムとして最近傍補間 ('NearestNeighbor' (米式綴り方)), 双線形補間 ('Bilinear'), 3 次補間 ('Cubic'), 3 次スプライン補間 ('CubicSpline'), ランチョス補間 ('Lanczos') を用います。デフォルトは最近傍補間です。
- **ST_Split** - Availability: 2.0.0 GEOS が必要ですジオメトリを他のジオメトリで分割してできたジオメトリのコレクションを返します。
- **ST_StdDev4ma** - Availability: 2.0.0 近隣のピクセル値の標準偏差を計算するラスタ処理関数です。
- **ST_Sum4ma** - Availability: 2.0.0 近隣のピクセル値の合計を計算するラスタ処理関数です。
- **ST_SummaryStats** - Availability: 2.0.0 与えられたラスターのバンドまたはラスターカバレッジの count, sum, mean, stddev, min, max からなる統計情報のサマリーです。バンドを指定しない場合には、1 番と仮定します。
- **ST_Transform** - Availability: 2.0.0 GDAL 1.6.1 以上が必要です。ラスタを既知の空間参照系から他の既知の空間参照系に、指定したリサンプリングアルゴリズムで投影変換します。新しいピクセル値のリサンプリングアルゴリズムとして最近傍補間 ('NearestNeighbor' (米式綴り方)), 双線形補間 ('Bilinear'), 3 次補間 ('Cubic'), 3 次スプライン補間 ('CubicSpline'), ランチョス補間 ('Lanczos') を用います。デフォルトは最近傍補間です。
- **ST_UnaryUnion** - Availability: 2.0.0 単一のジオメトリの要素の結合を計算します。
- **ST_Union** - Availability: 2.0.0 ラスタタイルの集合を結合して 1 以上のバンドからなる単一ラスタを返します。
- **ST_ValueCount** - Availability: 2.0.0 ラスタ (またはラスターカバレッジ) の指定されたバンドで、指定した値を持つピクセルを対象として、ピクセルバンド値とピクセル数からなるレコードの集合を返します。バンドを指定しない場合には、1 番と仮定します。デフォルトでは NODATA 値のピクセルは数えられず、ピクセルの他の値は出力され、ピクセルバンド値は最も近い整数に丸められます。
- **TopoElementArray_Agg** - Availability: 2.0.0 element_id とタイプの配列 (topoelements) からなる topoelementarray を返します。
- **TopoGeo_AddLineString** - Availability: 2.0.0 許容差を使って既存のトポロジにラインストリングを追加し、可能ならエッジ/フェイスを分割します。
- **TopoGeo_AddPoint** - Availability: 2.0.0 許容差を使って既存のトポロジにポイントを追加し、可能ならエッジを分割します。
- **TopoGeo_AddPolygon** - Availability: 2.0.0 許容差を使って既存のトポロジにラインストリングを追加し、可能ならエッジ/フェイスを分割します。エッジ識別番号を返します。
- **TopologySummary** - Availability: 2.0.0 トポロジ名を取り、トポロジ内のオブジェクトの型に関する概要の全体を提供します。
- **toTopoGeom** - Availability: 2.0 単純なジオメトリから TopoGeometry を生成します。
- **~** - Availability: 2.0.0 A のバウンディングボックスが B のバウンディングボックスを含む場合に TRUE を返します。倍精度浮動小数点数のバウンディングボックスを使います。
- **~=** - Availability: 2.0.0 A のバウンディングボックスが B のバウンディングボックスと同じ場合に TRUE を返します。

PostGIS 2.0 で機能強化された関数

- **&&** - Enhanced: 2.0.0 多面体サーフェス対応が導入されました。A の 2 次元バウンディングボックスが B の 2 次元バウンディングボックスとインタセクトする場合に TRUE を返します。

- **AddGeometryColumn** - Enhanced: 2.0.0 use `typmod` 引数が導入されました。デフォルトでは制約を基にしたものでなく `typmod` ジオメトリカラムが生成されます。ジオメトリカラムを既存のテーブルに追加します。
 - **Box2D** - Enhanced: 2.0.0 多面体サーフェス対応、三角対応、TIN 対応が導入されました。ジオメトリの 2 次元範囲を表現する BOX2D を返します。
 - **Box3D** - Enhanced: 2.0.0 多面体サーフェス対応、三角対応、TIN 対応が導入されました。ジオメトリの 3 次元範囲を表現する BOX3D を返します。
 - **CreateTopology** - Enhanced: 2.0 hasZ 対応の追加。新しいトポロジスキーマを生成し、`topology.topology` テーブルに登録します。
 - **GeometryType** - Enhanced: 2.0.0 多面体サーフェス対応、三角対応、TIN 対応が導入されました。ジオメトリのタイプを文字列で返します。
 - **Populate_Geometry_Columns** - Enhanced: 2.0.0 use `typmod` 任意引数が導入されました。カラムが型修飾子で生成されるか制約チェックで作られるかの制御ができます。ジオメトリカラムが型修飾子で定義されるか、適切な空間制約を持つようにします。
 - **ST_3DExtent** - Enhanced: 2.0.0 多面体サーフェス対応、三角対応、TIN 対応が導入されました。ジオメトリの 3 次元バウンディングボックスを返す集約関数です。
 - **ST_Affine** - Enhanced: 2.0.0 多面体サーフェス対応、三角対応、TIN 対応が導入されました。ジオメトリに 3 次元アフィン変換を適用します。
 - **ST_Area** - Enhanced: 2.0.0 - 2 次元多面体サーフェス対応が導入されました。ポリゴンジオメトリの面積を返します。
 - **ST_AsBinary** - Enhanced: 2.0.0 多面体サーフェス対応、三角対応、TIN 対応が導入されました。ジオメトリ/ジオグラフィの、SRID メタデータを持たない OGC/ISO Well-Known バイナリ (WKB) 表現を返します。
 - **ST_AsBinary** - Enhanced: 2.0.0 高次元が導入されました。ジオメトリ/ジオグラフィの、SRID メタデータを持たない OGC/ISO Well-Known バイナリ (WKB) 表現を返します。
 - **ST_AsBinary** - Enhanced: 2.0.0 ジオグラフィでのエンディアン指定が導入されました。ジオメトリ/ジオグラフィの、SRID メタデータを持たない OGC/ISO Well-Known バイナリ (WKB) 表現を返します。
 - **ST_AsEWKB** - Enhanced: 2.0.0 多面体サーフェス対応、三角対応、TIN 対応が導入されました。ジオメトリの、SRID メタデータを持つ Extended Well-Known バイナリ (EWKB) 表現を返します。
 - **ST_AsEWKT** - Enhanced: 2.0.0 ジオグラフィ対応、多面体サーフェス対応、三角形対応、TIN 対応が導入されました。ジオメトリの SRID メタデータが付いた Well-Known Text (WKT) 表現を返します。
 - **ST_AsGML** - Enhanced: 2.0.0 プレフィクスが導入されました。GML 3 用である `options` の 4 は、曲線のかわりにラインストリングを使えるようにするためのものです。GML 3 の多面体サーフェスと TIN が導入されました。`options` の 32 はボックスを出力するために導入されました。GML 第 2 版または第 3 版としてジオメトリを返します。
 - **ST_AsKML** - Enhanced: 2.0.0 - プレフィクスの名前空間の追加、デフォルト値と名前付き引数の追加ジオメトリを KML 要素として返します。
 - **ST_Azimuth** - Enhanced: 2.0.0 ジオグラフィ対応が導入されました。北を基準とした 2 点間の線の方角角を返します。
 - **ST_Dimension** - Enhanced: 2.0.0 多面体サーフェス対応と TIN 対応が導入されました。空ジオメトリを与えた場合に例外を投げなくなりました。`ST_Geometry` 値の座標次元を返します。
 - **ST_Dump** - Enhanced: 2.0.0 多面体サーフェス対応、三角対応、TIN 対応が導入されました。ジオメトリの要素となる `geometry_dump` 行の集合を返します。
 - **ST_DumpPoints** - Enhanced: 2.0.0 多面体サーフェス対応、三角対応、TIN 対応が導入されました。ジオメトリ内の座標の行である `geometry_dump` 行の集合を返します。
-

- **ST_Expand** - Enhanced: 2.0.0 多面体サーフェス対応、三角対応、TIN 対応が導入されました。他のバウンディングボックスまたはジオメトリから拡張されたバウンディングボックスを返します。
- **ST_Extent** - Enhanced: 2.0.0 多面体サーフェス対応、三角対応、TIN 対応が導入されました。ジオメトリのバウンディングボックスを返す集約関数です。
- **ST_Force2D** - Enhanced: 2.0.0 多面体サーフェス対応が導入されました。ジオメトリを 2 次元モードに強制します。
- **ST_Force3D** - Enhanced: 2.0.0 多面体サーフェス対応が導入されました。ジオメトリを XYZ モードに強制します。これは ST_Force3DZ の別名です。
- **ST_Force3DZ** - Enhanced: 2.0.0 多面体サーフェス対応が導入されました。ジオメトリを XYZ モードに強制します。
- **ST_ForceCollection** - Enhanced: 2.0.0 多面体サーフェス対応が導入されました。ジオメトリをジオメトリコレクションに変換します。
- **ST_ForceRHR** - Enhanced: 2.0.0 多面体サーフェス対応が導入されました。ポリゴンの頂点の方向を右回りに強制します。
- **ST_GMLToSQL** - Enhanced: 2.0.0 多面体サーフェス対応と TIN 対応が導入されました。GML 表現から指定した ST_Geometry 値を返します。これは ST_GeomFromGML の別名です。
- **ST_GMLToSQL** - Enhanced: 2.0.0 SRID 任意引数が追加されました。GML 表現から指定した ST_Geometry 値を返します。これは ST_GeomFromGML の別名です。
- **ST_GeomFromEWKB** - Enhanced: 2.0.0 多面体サーフェス対応と TIN 対応が導入されました。拡張 Well-Known Binary 表現 (EWKB) から指定した ST_Geometry 値を返します。
- **ST_GeomFromEWKT** - Enhanced: 2.0.0 多面体サーフェス対応と TIN 対応が導入されました。拡張 Well-Known Text 表現 (EWKT) から指定された ST_Geometry 値を返します。
- **ST_GeomFromGML** - Enhanced: 2.0.0 多面体サーフェス対応と TIN 対応が導入されました。GML 表現から PostGIS ジオメトリオブジェクトを出力します。
- **ST_GeomFromGML** - Enhanced: 2.0.0 SRID 任意引数が追加されました。GML 表現から PostGIS ジオメトリオブジェクトを出力します。
- **ST_GeometryN** - Enhanced: 2.0.0 多面体サーフェス対応、三角対応、TIN 対応が導入されました。ジオメトリコレクションの要素の一つを返します。
- **ST_GeometryType** - Enhanced: 2.0.0 多面体サーフェス対応が導入されました。ジオメトリの SQL-MM 型を文字列で返します。
- **ST_IsClosed** - Enhanced: 2.0.0 多面体サーフェス対応が導入されました。ラインストリングの始点と終点が一致しているかをテストします。多面体サーフェスについては閉じているか (立体であるか) をテストします。
- **ST_MakeEnvelope** - Enhanced: 2.0 SRID 指定なしでエンベロープを指定できるようになりました。座標値の最小値と最大値から矩形ポリゴンを生成します。
- **ST_MakeValid** - Enhanced: 2.0.1 速度の改善頂点を失うことなしに不正なジオメトリを妥当なジオメトリにしようと試みます。
- **ST_NPoints** - Enhanced: 2.0.0 多面体サーフェス対応が導入されました。ジオメトリのポイント (頂点) の数を返します。
- **ST_NumGeometries** - Enhanced: 2.0.0 多面体サーフェス対応、三角対応、TIN 対応が導入されました。ジオメトリコレクションの要素数を返します。
- **ST_Relate** - Enhanced: 2.0.0 - 境界ノード規則が追加されました。二つのジオメトリが与えられた交差行列パターンに合致するトポロジ関係があるかどうかを見るか、交差行列を計算するかします。
- **ST_Rotate** - Enhanced: 2.0.0 多面体サーフェス対応、三角対応、TIN 対応が導入されました。ジオメトリを原点について回転させます。

- **ST_Rotate** - Enhanced: 2.0.0 回転の原点を指定するパラメーターを追加しました。ジオメトリを原点について回転させます。
- **ST_RotateX** - Enhanced: 2.0.0 多面体サーフェス対応、三角対応、TIN 対応が導入されました。ジオメトリを X 軸について回転させます。
- **ST_RotateY** - Enhanced: 2.0.0 多面体サーフェス対応、三角対応、TIN 対応が導入されました。ジオメトリを Y 軸について回転させます。
- **ST_RotateZ** - Enhanced: 2.0.0 多面体サーフェス対応、三角対応、TIN 対応が導入されました。ジオメトリを Z 軸について回転させます。
- **ST_Scale** - Enhanced: 2.0.0 多面体サーフェス対応、三角対応、TIN 対応が導入されました。与えた係数でジオメトリを拡大縮小します。
- **ST_ShiftLongitude** - Enhanced: 2.0.0 多面体サーフェス対応と TIN 対応が導入されました。経度座標値を -180 度から 180 度の範囲と 0 度から 360 度の範囲との二つの範囲を行き来するようシフトします。
- **ST_Summary** - Enhanced: 2.0.0 でジオグラフィ対応が追加されました。ジオメトリについての要約文を返します。
- **ST_Transform** - Enhanced: 2.0.0 多面体サーフェス対応が導入されました。異なる空間参照系に投影変換された新しいジオメトリを返します。
- **ST_Value** - Enhanced: 2.0.0 任意引数 `exclude_nodata_value` が追加されました。指定したバンドにおける `columnx`, `rowy` で指定したピクセルまたは指定したジオメトリポイントに対応するピクセルの値を返します。バンド番号は 1 始まりで、指定しない場合には、1 番と仮定します。`exclude_nodata_value` が FALSE に設定された場合には、NODATA ピクセルを含む全てのピクセルがインタセクトするかが考慮され、値を返します。`exclude_nodata_value` を渡さない場合には、ラスタのメタデータから読みます。
- **ValidateTopology** - Enhanced: 2.0.0 では、より効果的なエッジ交差検出が可能になり、以前の版で残っていた偽陽性を解決しています。トポロジの問題についての詳細を示す `validatetopology_returntype` の集合を返します。

PostGIS 2.0 で変更された関数

- **AddGeometryColumn** - Changed: 2.0.0 `geometry_columns` がシステムカタログを読むビューになったため、`geometry_columns` を更新しないようになりました。デフォルトでは制約を生成せず、PostgreSQL の型修飾子を使います。この関数による WGS 84 の POINT カラムの構築と `ALTER TABLE some_table ADD COLUMN geom geometry(Point,4326);` とは等価です。ジオメトリカラムを既存のテーブルに追加します。
- **AddGeometryColumn** - Changed: 2.0.0 制約を使う必要がある場合には、`use_typmod` を FALSE にします。ジオメトリカラムを既存のテーブルに追加します。
- **AddGeometryColumn** - Changed: 2.0.0 ビューについては、`geometry_columns` への手動登録はできなくなりました。しかし、`typmod` テーブルジオメトリに対して構築されていて、かつラップ関数が無いビューは、親テーブルカラムの `typmod` の挙動を継承するので、正しく登録されます。他のジオメトリを出力するジオメトリ関数を使うビューについては、ビューのジオメトリカラムが正しく登録されるようにするため、`typmod` ジオメトリへのキャストが必要です。を参照して下さい。ジオメトリカラムを既存のテーブルに追加します。
- **Box3D** - Changed: 2.0.0 以前の版では、`box3d` でなく `box2d` を使っていました。`box2d` は非推奨型となり、`box3d` に変更しました。ラスタを囲むボックスの `box3d` 表現を返します。
- **DropGeometryColumn** - Changed: 2.0.0 この関数は後方互換のためのものです。`geometry_columns` は現在はシステムカタログに対するビューですので、他のテーブルのカラムと同じように `ALTER TABLE` を使った削除が可能です。ジオメトリカラムを空間テーブルから除去します。
- **DropGeometryTable** - Changed: 2.0.0 でこの関数は後方互換のためのものです。`geometry_columns` は現在はシステムカタログに対するビューですので、他のテーブルのカラムと同じように `DROP TABLE` を使った削除が可能です。テーブルと `geometry_columns` の当該テーブルへの参照の全てを削除します。

- **Populate_Geometry_Columns** - Changed: 2.0.0 デフォルトでは、ジオメトリタイプの制限について、制約を確認する代わりに型修飾子を使います。新しい `use_typmod` を `FALSE` に設定して使うことで、制約確認を使用することができます。ジオメトリカラムが型修飾子で定義されるか、適切な空間制約を持つようにします。
- **ST_3DExtent** - Changed: 2.0.0 以前の版では `ST_Extent3D` と呼ばれていました。ジオメトリの 3 次元バウンディングボックスを返す集約関数です。
- **ST_3DLength** - Changed: 2.0.0 以前の版では `ST_Length3D` と呼ばれていました線ジオメトリの 3 次元長を返します。
- **ST_3DMakeBox** - Changed: 2.0.0 以前の版では `ST_MakeBox3D` と呼ばれていました。二つの 3 次元のポイントジオメトリで定義される `BOX3D` を生成します。
- **ST_3DPerimeter** - Changed: 2.0.0 以前の版では `ST_Perimeter3D` と呼ばれていました。ポリゴンジオメトリの 3 次元周長を返します。
- **ST_AsBinary** - Changed: 2.0.0 Inputs to this function can not be unknown -- must be geometry. Constructs such as `ST_AsBinary('POINT(1 2)')` are no longer valid and you will get an `n ST_AsBinary(unknown) is not unique` error. Code like that needs to be changed to `ST_AsBinary('POINT(1 2)::geometry')`. If that is not possible, then install `legacy.sql`. ジオメトリ/ジオグラフィの、SRID メタデータを持たない OGC/ISO Well-Known バイナリ (WKB) 表現を返します。
- **ST_AsGML** - Changed: 2.0.0 デフォルトの名前付き引数を使います。GML 第 2 版または第 3 版としてジオメトリを返します。
- **ST_AsGeoJSON** - Changed: 2.0.0 デフォルト引数と名前付き引数に対応しました。GeoJSON 形式のジオメトリまたは地物を返します。
- **ST_AsSVG** - Changed: 2.0.0 - デフォルト引数と名前付き引数に対応しました。ジオメトリから SVG パスデータを返します。
- **ST_EndPoint** - Changed: 2.0.0 一つのジオメトリマルチラインストリングで動作しなくなりました。PostGIS の古いバージョンでは、この関数は一つのマルチラインストリングで動作し、終端ポイントを返します。2.0.0 では、他のマルチラインストリングと同様に `NULL` を返します。古い動作は文書化されていない機能でしたが、データを `LINESTRING` として格納していると思われるユーザーは、2.0.0 で `NULL` が返されることを経験するかも知れません。Returns the last point of a LineString, CircularLineString, or NURBSCurve.
- **ST_GDALDrivers** - Changed: 2.0.6, 2.1.3 - GUC (訳注: Grand Unified Configuration の略で、動的に変更できる PostgreSQL パラメーター) または環境変数 `gdal_enabled_drivers` が設定されていないデフォルトではドライバが全て無効になりました。使用している GDAL ライブラリが対応するラスタ書式の一覧を返します。この一覧で `can_write=True` となっているものだけが `ST_AsGDALRaster` で使えます。
- **ST_GeomFromText** - Changed: 2.0.0 前の版では `ST_GeomFromText('GEOMETRYCOLLECTION(EMPTY)')` が許されていました。SQL/MM 標準への適合のため PostGIS 2.0.0 では不正とされます。今は `ST_GeomFromText('GEOMETRYCOLLECTION(EMPTY)')` となります。Well-Known Text 表現 (WKT) から指定した `ST_Geometry` を返します。
- **ST_GeometryN** - Changed: 2.0.0 以前の版では非マルチのジオメトリでは `NULL` が返りました。`ST_GeometryN(..)` の場合にはジオメトリを返すよう変更されました。ジオメトリコレクションの要素の一つを返します。
- **ST_IsEmpty** - Changed: 2.0.0 以前の版の PostGIS では `ST_GeomFromText('GEOMETRYCOLLECTION(EMPTY)')` を許しました。PostGIS 2.0.0 では、SQL/MM 標準により準拠させるため、これは不正となります。ジオメトリが空かをテストします。
- **ST_Length** - Changed: 2.0.0 大幅な変更 -- 以前の版ではジオグラフィの `POLYGON` や `MULTIPOLYGON` への適用によって `POLYGON` や `MULTIPOLYGON` の周囲長を返しました。2.0.0 版ではジオメトリの挙動に従うため `0` を返すように変更しました。ポリゴンの周囲長を求める場合は、`ST_Perimeter` を使います線系ジオメトリの 2 次元長を返します。
- **ST_LocateAlong** - Changed: 2.0.0 以前の版では `ST_Locate_Along_Measure` と呼ばれていました。M 値に一致するジオメトリ上のポイントを返します。
- **ST_LocateBetween** - Changed: 2.0.0 以前の版では `ST_Locate_Between_Measures` と呼ばれていました。M 値の範囲に合致する部分ジオメトリを返します。

- **ST_ModEdgeSplit** - Changed: 2.0 - 以前の版では名前を間違えて ST_ModEdgesSplit になっていました。既存のエッジに沿って新しいノードを生成してエッジを分割します。もとのエッジは変更され、新しいエッジが一つ追加されます。
- **ST_NumGeometries** - Changed: 2.0.0 前の版では、ジオメトリがコレクション/マルチ系でない場合には NULL を返しました。2.0.0 以上では、POLYGON, LINESTRING, POINT といった単一ジオメトリについては 1 を返します。ジオメトリコレクションの要素数を返します。
- **ST_NumInteriorRings** - Changed: 2.0.0 - 以前の版では、MULTIPOLYGON を渡して最初の POLYGON の内環の数を返すことができました。ポリゴンの内環 (穴) の数を返します。
- **ST_PointN** - Changed: 2.0.0 単一ジオメトリの MULTILINESTRING で動作しなくなりました。単一のラインストリングからなる MULTILINESTRING については幸運にも動いていて、最初のポイントを返していました。2.0.0 では他の MULTILINESTRING と同様に NULL を返すようになりました。ジオメトリの最初のラインストリングまたは曲線ストリングの N 番目のポイントを返します。
- **ST_ScaleX** - Changed: 2.0.0. WKTRaster 版では ST_PixelSizeX と呼ばれていました。空間参照系の地理的な単位でのピクセル幅の X 成分を返します。
- **ST_ScaleY** - Changed: 2.0.0. WKTRaster 版では ST_PixelSizeY と呼ばれていました。空間参照系の地理的な単位でのピクセル幅の Y 成分を返します。
- **ST_SetScale** - Changed: 2.0.0 WKTRaster 版では、ST_SetPixelSize と呼ばれていました。2.0.0 で変更されました。ピクセルサイズの X 値と Y 値を空間参照系の単位で設定します。数値は単位/ピクセルの幅または高さです。
- **ST_StartPoint** - Changed: 2.0.0 一つの MULTILINESTRING で動作しなくなりました。PostGIS の古いバージョンでは、この関数は、一つのラインストリングからなる MULTILINESTRING については幸運にも動いていて、始端ポイントを返していました。2.0.0 では他の MULTILINESTRING と同様に NULL を返すようになりました。古い動作は文書化されていない機能でしたが、データを LINESTRING として格納していると思われるユーザーは、2.0.0 で NULL が返されることを経験するかも知れません。Returns the first point of a LineString, CircularLineString, or NURBSCurve.

13.12.15 PostGIS 1.5 で新規作成/機能強化された関数

次に示す関数は、新規作成または機能強化された PostGIS 関数です。

PostGIS 1.5 で新規作成された関数

- **&&** - Availability: 1.5.0 ジオグラフィ対応が導入されました。A の 2 次元バウンディングボックスが B の 2 次元バウンディングボックスとインタセクトする場合に TRUE を返します。
- **PostGIS_LibXML_Version** - Availability: 1.5 LibXML2 ライブラリのバージョン番号を返します。
- **ST_AddMeasure** - Availability: 1.5.0 ラインに沿った M 値を補間します。
- **ST_AsBinary** - Availability: 1.5.0 ジオグラフィが導入されました。ジオメトリ/ジオグラフィの、SRID メタデータを持たない OGC/ISO Well-Known バイナリ (WKB) 表現を返します。
- **ST_AsGML** - Availability: 1.5.0 ジオグラフィが導入されました。GML 第 2 版または第 3 版としてジオメトリを返します。
- **ST_AsGeoJSON** - Availability: 1.5.0 ジオグラフィが導入されました。GeoJSON 形式のジオメトリまたは地物を返します。
- **ST_AsText** - Availability: 1.5 - ジオグラフィ対応が導入されました。ジオメトリ/ジオグラフィの SRID メタデータのない Well-Known Text (WKT) 表現を返します。
- **ST_Buffer** - Availability: 1.5 - ST_Buffer が強化され、様々な終端と継ぎ目に対応するようになりました。たとえば、道路ラインストリングを道路ポリゴンに変換する際に終端を丸でなく平面や四角で処理したい場合などに使えます。ジオグラフィ用の薄いラッパが追加されました。あるジオメトリからの距離が指定された距離以下となる点全ての集合となるジオメトリを返します。

- **ST_ClosestPoint** - Availability: 1.5.0 g1 上にある、g2 と最近傍となる 2 次元ポイントを返します。これは、あるジオメトリから他のジオメトリへの最短ラインの一つ目のポイントです。
 - **ST_CollectionExtract** - Availability: 1.5.0 ジオメトリコレクションを与えると、指定されたタイプの要素だけからなるマルチジオメトリを返します。
 - **ST_Covers** - Availability: 1.5 - ジオグラフィ対応が導入されました。B の全ての点が A 内にあるかをテストします。
 - **ST_DFullyWithin** - Availability: 1.5.0 ジオメトリが完全に他のジオメトリの指定距離内にあるかどうかをテストします
 - **ST_DWithin** - Availability: 1.5.0 ジオグラフィが導入されました。二つのジオメトリが与えられた距離内にあるかどうかをテストします。
 - **ST_Distance** - Availability: 1.5.0 1.5 でジオグラフィ対応が導入されました。大きいジオメトリや頂点の多いジオメトリについての速度が改善しました二つのジオメトリ値またはジオグラフィ値間の距離を返します。
 - **ST_DistanceSphere** - Availability: 1.5 - ポイント以外のジオメトリが導入されました。以前の版ではポイントでのみ動作しました。球面の地球モデルを使って、二つの経度/緯度ジオメトリの最小距離をメートル単位で返します。
 - **ST_DistanceSpheroid** - Availability: 1.5 - ポイント以外のジオメトリが導入されました。以前の版ではポイントでのみ動作しました。回転楕円体面の地球モデルを使って、二つの経度/緯度ジオメトリの最小距離を返します。
 - **ST_DumpPoints** - Availability: 1.5.0 ジオメトリ内の座標の行である `geometry_dump` 行の集合を返します。
 - **ST_Envelope** - Availability: 1.5.0 挙動が変更され出力が `float4` から `float8` になりました。ジオメトリのバウンディングボックスを表現するジオメトリを返します。
 - **ST_Expand** - Availability: 1.5.0 出力を `float4` 座標値から倍精度に変更しました。他のバウンディングボックスまたはジオメトリから拡張されたバウンディングボックスを返します。
 - **ST_GMLToSQL** - Availability: 1.5 libxml2 1.6+ が必要です。GML 表現から指定した ST_Geometry 値を返します。これは ST_GeomFromGML の別名です。
 - **ST_GeomFromGML** - Availability: 1.5 libxml2 1.6+ が必要です。GML 表現から PostGIS ジオメトリオブジェクトを出力します。
 - **ST_GeomFromKML** - Availability: 1.5 libxml2 2.6 以上が必要です。ジオメトリの KML 表現の入力を取り、PostGIS ジオメトリオブジェクトを出力します。
 - **ST_HausdorffDistance** - Availability: 1.5.0 二つのジオメトリ間のハウスドルフ距離を返します。
 - **ST_Intersection** - Availability: 1.5 ジオグラフィ型が導入されました。ジオメトリ A とジオメトリ B の共通部分を表現するジオメトリを返します。
 - **ST_Intersects** - Availability: 1.5 ジオグラフィ対応が導入されました。二つのジオメトリがインタセクトしている (少なくとも一つの共有点がある) かどうかテストします。
 - **ST_Length** - Availability: 1.5.0 ジオグラフィ t 対応が導入されました。線系ジオメトリの 2 次元長を返します。
 - **ST_LongestLine** - Availability: 1.5.0 二つのジオメトリ間の 2 次元最長ラインを返します。
 - **ST_MakeEnvelope** - Availability: 1.5 座標値の最小値と最大値から矩形ポリゴンを生成します。
 - **ST_MaxDistance** - Availability: 1.5.0 二つのジオメトリ間の 2 次元最長距離を空間参照系の単位で返します。
 - **ST_ShortestLine** - Availability: 1.5.0 二つのジオメトリの 3 次元の最短ラインを返します。
 - **~=** - Availability: 1.5.0 挙動が変更されました A のバウンディングボックスが B のバウンディングボックスと同じ場合に TRUE を返します。
-

13.12.16 PostGIS 1.4 で新規作成/機能強化された関数

次に示す関数は、新規作成または機能強化された PostGIS 関数です。

PostGIS 1.4 で新規作成された関数

- **Populate Geometry Columns** - Availability: 1.4.0 ジオメトリカラムが型修飾子で定義されるか、適切な空間制約を持つようにします。
- **ST_Collect** - Availability: 1.4.0 - **ST_Collect(geometry)** が導入されました。ST_Collect がより多くのジオメトリをより早く扱えるよう強化されました。ジオメトリの集合からジオメトリコレクションまたはマルチ系ジオメトリを生成します。
- **ST_ContainsProperly** - Availability: 1.4.0 B の全てのポイントが A の内部にあるかをテストします。
- **ST_GeoHash** - Availability: 1.4.0 ジオメトリの GeoHash 表現を返します。
- **ST_IsValidReason** - Availability: 1.4 ジオメトリが妥当か否かを示す文字列を返し、不正な場合は理由を返します。
- **ST_LineCrossingDirection** - Availability: 1.4 二つのラインストリングがどのように交差しているかを示す数字を返します。
- **ST_LocateBetweenElevations** - Availability: 1.4.0 標高 (Z 値) 範囲にある部分ジオメトリを返します。
- **ST_MakeLine** - Availability: 1.4.0 - **ST_MakeLine(geomarray)** が導入されました。ST_MakeLine 集約関数はより多くのポイントをより早く扱うための強化が施されています。POINT、MULTIPOINT、LINESTRING から LINESTRING を生成します。
- **ST_MinimumBoundingCircle** - Availability: 1.4.0 入力ジオメトリを含む最小の円を返します。
- **ST_Union** - Availability: 1.4.0 - ST_Union が機能強化されました。ST_Union(geomarray) が導入され、PostgreSQL の高速なコレクションの集約が導入されました。入力ジオメトリのポイント集合の結合を表現するジオメトリを返します。

13.12.17 PostGIS 1.3 で新規作成/機能強化された関数

次に示す関数は、新規作成または機能強化された PostGIS 関数です。

PostGIS 1.3 で新規作成された関数

- **ST_AsGML** - Availability: 1.3.2 GML 第 2 版または第 3 版としてジオメトリを返します。
- **ST_AsGeoJSON** - Availability: 1.3.4 GeoJSON 形式のジオメトリまたは地物を返します。
- **ST_CurveToLine** - Availability: 1.3.0 曲線を含むジオメトリを線ジオメトリに変換します。
- **ST_LineToCurve** - Availability: 1.3.0 曲線を含むジオメトリを線ジオメトリに変換します。
- **ST_SimplifyPreserveTopology** - Availability: 1.3.3 Douglas-Peucker アルゴリズムを使用して、単純化した妥当なジオメトリを返します。

Chapter 14

問題を報告する

14.1 ソフトウェアのバグを報告する

効率的なバグの報告は PostGIS の開発を助ける本質的な方法です。最も効率的なバグ報告は、PostGIS 開発者がそれを再現できるようにすることで、その引き金となったスクリプトと検出された環境に沿った全ての情報を含んでいるのが理想です。SELECT postgis_full_version() [PostGIS] と SELECT version() [PostgreSQL] とを実行することで十分に良い情報を得ることができます。

最新版を使っていない場合には [release changelog](#) をまず見て、既にバグフィクスされていないかを探すのは価値のあることです。

[PostGIS bug tracker](#) を使うと、レポートが捨てられず、その対応プロセスが通知されることを保証します。新しいバグを報告する前にデータベースに問い合わせ、既知のバグかどうかを見て下さい。既知のものでしたら、それに関して持っているあらゆる新しい情報を追加して下さい。

新しいレポートを記入する前に Simon Tatham さんの [How to Report Bugs Effectively](#) に関するページを読むと良いでしょう。

14.2 文書の問題を報告する

文書は、ソフトウェアの機能と挙動を正確に反映するべきものです。正確でない場合は、ソフトウェアのバグがあるか、または文書に誤り若しくは不十分な箇所があることが考えられます。

文書の問題も [PostGIS bug tracker](#) に報告することができます。

訂正が小さいものなら、バグトラッカの新しい問題の中に、文書内の位置を特定して記述して下さい。

変更が大きい場合は、パッチが確実に好まれます。Unix 上で次の 4 ステップの処理を行います (既に [git](#) をインストールしていると仮定します)。

1. PostGIS の git リポジトリを複製します。UNIX では次のように入力します。

```
git clone https://git.osgeo.org/gitea/postgis/postgis.git
```

これで postgis ディレクトリに格納されます

2. お使いのテキストエディタで文書に変更を加えます。Unix では、たとえば次のようにします。

```
vim doc/postgis.xml
```

文書は HTML でなく DocBook XML で書かれていますので、慣れていないなら、残りの文書の例にならって下さい。

3. 文書のマスタコピーからパッチファイルを作成します。Unix では次のように入力します。

```
git diff doc/postgis.xml > doc.patch
```

4. バグトラッカ内の新しい問題にパッチが取り付けられます。

Appendix A

付録

A.1 PostGIS 3.7.0alpha1

This version requires GEOS 3.10+, PostgreSQL 14-19beta1, Proj 6.1+, libgmp. To take advantage of all features postgis extension features, GEOS 3.15+ is needed. To take advantage of all postgis_sfcgal extension features SFCGAL 2.3+ is needed.

A.1.1 大幅な変更

#5688, [topology] topology building functions now interpret tolerance 0 (old default) to really mean 0 and want -1 (new default) to mean "use topology precision" (Sandro Santilli)

#6024, Remove support for PostgreSQL 12 and 13 (Regina Obe)

#6040, ST_GeoHash round trip through ST_GeomFromGeoHash (Paul Ramsey)

#6053, [address_standardizer] Extension removed and moved to https://github.com/postgis/address_standardizer (Paul Ramsey)

#6052, [tiger_geocoder] Extension removed and moved to https://git.osgeo.org/postgis/postgis_tiger_geocoder (Regina Obe)

#6079, Remove support for GEOS < 3.10 (Sandro Santilli)

#1793, Remove ST_MemCollect aggregate (Darafei Praliaskouski)

GH-848, CG_AlphaShape now returns a MultiPolygon (Jean Felder)

A.1.2 新機能

#1124, **#2935**, **#3033**, **#4658**, **#4659**, [loader] Rework loader arguments: shp2pgsql can create UNLOGGED tables and choose the feature id column name, shp2pgsql --drop-table can emit DROP TABLE before prepare output, raster2pgsql/shp2pgsql expose loader actions as long options, both loaders accept long aliases for existing options, and

-if-not-exists makes creation actions idempotent (Darafei Praliaskouski)

#4208, Add single-geometry variants of ST_MaxDistance and ST_LongestLine (Darafei Praliaskouski)

[topology] FindVertexSegmentPairsBelowDistance function (Sandro Santilli)

ST_CoverageEdges, returns MultiLinestring of distinct shared edges in polygonal coverage (Paul Ramsey)

ST_MinimumSpanningTree, window function to calculate MST (Paul Ramsey)

#5993, [topology] Add max_edges parameter to TopoGeo_AddLinestring (Sandro Santilli)

#6001, support MultiLineString in ST_MakeLine (Paul Ramsey)

#2858, ST_MMin and ST_MMax (Paul Ramsey)

#5941, [raster] Support GDT_Float16 pixel type (Darafei Praliaskouski)

#5992, Optimize GiST index for repeated edge subdivision in topology (Darafei Praliaskouski)

#5702, Allow the compiler to detect the parallelism -fpto=auto (Darafei Praliaskouski)

#4798, ST_AsGeoJSON warns about duplicate property keys (Darafei Praliaskouski)

#5950, Document POSTGIS_REGRESS_DB_OWNER for sandboxed regression roles (Darafei Praliaskouski)

#4332, Clarify the scope of several Function Reference categories (Darafei Praliaskouski)

#4222, [raster] ST_DumpAsPolygons honours PostgreSQL interrupts (Darafei Praliaskouski)

#5109, Document the meaning of topology.next_left_edge and topology.next_right_edge links (Darafei Praliaskouski)

#5889, [topology] Include representative locations in topology build errors (Darafei Praliaskouski)

#2614, Use GEOSPreparedDistance and caching to accelerate ST_DWithin (Paul Ramsey)

#6022, Document ST_AsMVTGeom tile-coordinate simplification (Darafei Praliaskouski)

GH-839, ST_Multi support for TIN and surfaces (Loïc Bartoletti)

#4398, Add ST_CatmullSmoothing smooths but retains original vertices (Paul Ramsey)

#4560, ST_3DInterpolatePoint for M interpolation from XYZ inputs (Paul Ramsey)

#1291, ST_MakePolygon support for curved rings (Darafei Praliaskouski)

GT-270, Add NURBSCurve (Loïc Bartoletti)

#2863, Add ST_XSize, ST_YSize, ST_ZSize, and ST_MSize dimension helpers (Darafei Praliaskouski)

[fuzzers] Centralize OSS-Fuzz build/dependency logic in PostGIS scripts, prefer requested CXX over pg_config --cc for internal C++ checks, and make fuzzer linking/runtime dependency packaging portable (Darafei Praliaskouski)

A.1.3 性能強化

#2045, pgsql2shp query dumps no longer require temporary table privileges (Darafei Praliaskouski)

#1577, [loader] Report unsupported .zip archive input before trying to open shapefile sidecar files (Darafei Praliaskouski)

#3158, Move geometry and geography typmod bit helpers out of the public liblwgeom header (Darafei Praliaskouski)

#4678, [raster] Move st_sum4ma neighborhood callback to C (Darafei Praliaskouski)

#2116, [raster] Add ST_Value nearest-neighbor boundary options (Darafei Praliaskouski)

#2804, #4315, [raster] Support two-argument ST_MapAlgebra callbacks and pass callback call data as actual arguments (Darafei Praliaskouski)

#2898, Document SQL function cost tiers for contributors

#6062, [topology] Stop using recursive snapping, for improved robustness (Sandro Santilli)

#6065, Improved winding order computation robustness for rings having collapsed elements (Sandro Santilli)

- [#3743](#), [#4385](#), [raster loader] Document raster2pgsql -s FROM_SRID:SRID reprojection support and clarify index creation timing for append loads (Darafei Praliaskouski)
 - [#2137](#), [raster] Reuse matching source raster overviews in raster2pgsql -l (Darafei Praliaskouski)
 - [#4749](#), Use point-in-polygon predicate fast paths for point-only GeometryCollections (Darafei Praliaskouski)
 - [#5532](#), Validate the manual against DocBook XMLSchema in check-xml (Darafei Praliaskouski)
 - [#6087](#), Add a repository-owned CI status checker and static dashboard generator (Darafei Praliaskouski)
 - [#2807](#), [raster] Preserve missing NODATA values in ST_MapAlgebra outputs (Darafei Praliaskouski)
 - [#2832](#), [raster] Avoid padding single-tile raster2pgsql overviews when padding is not requested (Darafei Praliaskouski)
 - [#2623](#), Generate a pgsql2shp GID field when exporting geometry-only shapefiles (Darafei Praliaskouski)
 - [#2850](#), Preserve pgsql2shp numeric precision and scale in DBF metadata (Darafei Praliaskouski)
 - [#2386](#), Add raster2pgsql man page (Darafei Praliaskouski)
- Build PostgreSQL extension modules with `-fno-semantic-interposition` when available so LTO can optimize same-DSO calls to exported PostGIS entry points directly instead of routing through the module PLT; in a local PostgreSQL 18 `pgbench` microbenchmark over 100k four-point groups this reduced `ST_Collect` aggregate latency by about 4% (Darafei Praliaskouski)
- [#5973](#), Install PostGIS utilities under `pg_config --bindir` by default and document `pg_config`-derived install paths more accurately (Darafei Praliaskouski)
 - [#2808](#), Clarify `postgis_full_version()` output when installed component scripts are newer than core scripts (Darafei Praliaskouski)
 - [#5593](#), Docs: render manual images with GraphicsMagick without temporary files, enabling parallel builds, keeping ImageMagick fallbacks working, and documenting the generator workflow for contributors (Darafei Praliaskouski)
 - [#5645](#), Docs: keep code operators (`"=>"`) intact in translated manuals by enforcing verbatim CSS (Darafei Praliaskouski)
 - [#1408](#), Docs: flatten transparent manual images onto white backgrounds for PDF builds while keeping HTML images transparent (Darafei Praliaskouski)

A.1.4 Bug Fixes

- [#4616](#), Detect pgsql2shp query keywords case-insensitively (Darafei Praliaskouski)
 - [#2838](#), Emit valid X3D Coordinate nodes for 2D polygon output (Darafei Praliaskouski)
- Fix WKB and TWKB parser resource exhaustion on malformed input (Darafei Praliaskouski)
- [#6083](#), Pass configured dependency include paths to fuzzer smoke builds (Darafei Praliaskouski)
 - [#3103](#), Add regression coverage for exact-schema `find_srid` and `geometry_columns` lookups (Darafei Praliaskouski)
 - [#1705](#), Infer constraint metadata for direct view and materialized view geometry columns (Darafei Praliaskouski)
 - [#6038](#), Avoid stale relation lookups in `geometry_columns` after topology objects are dropped (Darafei Praliaskouski)
 - [#5655](#), [doc] Skip XML validation during `make check` when `xsltproc` is unavailable (Darafei Praliaskouski)

- #5487**, Improve error detail for repeated upgrades after an incomplete deprecated-function cleanup (Darafei Praliaskouski)
- GH-899**, [raster] Honor PostgreSQL interrupts in long-running GDAL progress callbacks (Darafei Praliaskouski)
- #5975**, Initialize skipped union-find cluster ids deterministically (Darafei Praliaskouski)
- #5904**, [topology] Avoid MinGW format warnings when logging topology ids (Darafei Praliaskouski)
- GH-885**, [topology] Avoid GMP overflow in ring orientation for very large finite coordinates (Darafei Praliaskouski)
- #6073**, **#6074**, Stabilize geography LRS endpoint interpolation and pole centroid regressions on 32-bit systems (Darafei Praliaskouski)
- #6082**, Fix Japanese PDF build by avoiding non-ASCII arrows in SQL program listings (Darafei Praliaskouski)
- GH-888**, [sfcgal] Avoid stale detoasted geometry access in CG_Visibility empty-input handling for SFCGAL < 2.2 (Darafei Praliaskouski)
- #3179**, Mark truncated libpgcommon PostgreSQL messages instead of silently dropping the tail (Darafei Praliaskouski)
- #6003**, Avoid size_t formats in liblwgeom logging/error wrappers for MinGW builds (Darafei Praliaskouski)
- GH-886**, ST_ClusterRelateWin validates NULL relate matrices before detoasting them (Darafei Praliaskouski)
- GH-889**, ST_Relate pattern inversion avoids out-of-bounds access on short DE-9IM patterns (Darafei Praliaskouski)
- GH-890**, Avoid GIDX BRIN summary merges across mismatched dimensions (Darafei Praliaskouski)
- #5948**, [topology] Prevent MakeTopologyPrecise from erasing edges when grid size exceeds their extent (Darafei Praliaskouski)
- #5959**, Prevent histogram target overflow when analysing massive tables (Darafei Praliaskouski)
- #6080**, Avoid out-of-bounds hex WKB lookup for high-bit input bytes (Darafei Praliaskouski)
- GH-895**, Qualify pg_class lookup during SRID/PROJ validation (Darafei Praliaskouski)
- #4828**, geometry_columns handles NOT VALID SRID checks without errors (Darafei Praliaskouski)
- #6048**, [raster] ST_Clip no longer crashes when clipping sparse band selections (Darafei Praliaskouski)
- GH-894**, [raster] Fix Float16 import, mixed nodata warping, band replacement cleanup, GDAL credential redaction, and Python GDAL compatibility (Darafei Praliaskouski)
- Flatgeobuf schema mismatch vulnerability (NeuroWinter)
- #5899**, pg_upgrade issue for non-standard geography SRID (Paul Ramsey)
- #2583**, Preserve Z/ZM dimensional qualifiers in ST_AsEWKT output for TIN (Darafei Praliaskouski)
-