PostGIS 3.4.0rc2 Handbuch

Contents

1 Einführung **1**

1.1 Projektleitung . . . 1

1.2 Aktuelle Kernentwickler . . . 2

1.3 Frühere Kernentwickler . . . 2

1.4 Weitere Mitwirkende . . . 2

2 PostGIS Installation **6**

2.1 Kurzfassung . . . 6

2.2 Kompilierung und Installation des Quellcodes: Detaillierte Beschreibung . . . 6

2.2.1 Nutzung des Quellcodes . . . 7

2.2.2 Systemvoraussetzungen . . . 7

2.2.3 Konfiguration . . . 8

2.2.4 Build-Prozess . . . 10

2.2.5 Build-Prozess für die PostGIS Extensions und deren Bereitstellung . . . 10

2.2.6 Softwaretest . . . 12

2.2.7 Installation . . . 15

2.3 Installation und Verwendung des Adressennormierers . . . 16

2.4 Installing, Upgrading Tiger Geocoder, and loading data . . . 16

2.4.1 Tiger Geocoder Enabling your PostGIS database . . . 17

2.4.2 Die Adressennormierer-Extension zusammen mit dem Tiger Geokodierer verwenden . . . 19

2.4.3 Required tools for tiger data loading . . . 19

2.4.4 Upgrading your Tiger Geocoder Install and Data . . . 20

2.5 Übliche Probleme bei der Installation . . . 20

3 PostGIS Verwaltung **22**

3.1 Leistungsoptimierung . . . 22

3.1.1 Startup . . . 22

3.1.2 Runtime . . . 23

3.2 Configuring raster support . . . 23

3.3 Creating spatial databases . . . 24

3.3.1 Spatially enable database using EXTENSION . . . 24

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

4 Data Management **29**
4.1 Spatial Data Model . . . 29
4.1.1 OGC Geometry . . . 29
4.1.1.1 Point . . . 30
4.1.1.2 LineString . . . 30
4.1.1.3 LinearRing . . . 30
4.1.1.4 Polygon . . . 30
4.1.1.5 MultiPoint . . . 30
4.1.1.6 MultiLineString . . . 30
4.1.1.7 MultiPolygon . . . 31
4.1.1.8 GeometryCollection . . . 31
4.1.1.9 PolyhedralSurface . . . 31
4.1.1.10 Triangle . . . 31
4.1.1.11 TIN . . . 31
4.1.2 SQL/MM Part 3 - Curves . . . 31
4.1.2.1 CircularString . . . 32
4.1.2.2 CompoundCurve . . . 32
4.1.2.3 CurvePolygon . . . 32
4.1.2.4 MultiCurve . . . 32
4.1.2.5 MultiSurface . . . 32
4.1.3 WKT and WKB . . . 33
4.2 Geometry Data Type . . . 34
4.2.1 PostGIS EWKB and EWKT . . . 34
4.3 Geography Data Type . . . 36
4.3.1 Creating Geography Tables . . . 36
4.3.2 Using Geography Tables . . . 37
4.3.3 When to use the Geography data type . . . 38
4.3.4 Fortgeschrittene FAQ's zum geographischen Datentyp . . . 38
4.4 Geometrievalidierung . . . 39
4.4.1 Simple Geometry . . . 39
4.4.2 Valid Geometry . . . 41
4.4.3 Managing Validity . . . 43

4.5 Spatial Reference Systems . . . 44
4.5.1 SPATIAL_REF_SYS Table . . . 45
4.5.2 User-Defined Spatial Reference Systems . . . 46
4.6 Spatial Tables . . . 46
4.6.1 Erstellung einer räumlichen Tabelle . . . 46
4.6.2 GEOMETRY_COLUMNS View . . . 47
4.6.3 Manually Registering Geometry Columns . . . 48
4.7 Loading Spatial Data . . . 50
4.7.1 Using SQL to Load Data . . . 50
4.7.2 Using the Shapefile Loader . . . 50
4.8 Extracting Spatial Data . . . 52
4.8.1 Using SQL to Extract Data . . . 52
4.8.2 Using the Shapefile Dumper . . . 53
4.9 Spatial Indexes . . . 54
4.9.1 GiST-Indizes . . . 54
4.9.2 BRIN Indizes . . . 55
4.9.3 SP-GiST Indizes . . . 56
4.9.4 Tuning Index Usage . . . 57

5 Räumliche Abfrage **58**
5.1 Räumliche Beziehungen feststellen . . . 58
5.1.1 Dimensionally Extended 9-Intersection Model . . . 58
5.1.2 Named Spatial Relationships . . . 60
5.1.3 General Spatial Relationships . . . 61
5.2 Using Spatial Indexes . . . 63
5.3 Examples of Spatial SQL . . . 63

6 Performance Tipps **66**
6.1 Kleine Tabellen mit großen Geometrien . . . 66
6.1.1 Problembeschreibung . . . 66
6.1.2 Umgehungslösung . . . 66
6.2 CLUSTER auf die geometrischen Indizes . . . 67
6.3 Vermeidung von Dimensionsumrechnungen . . . 67

7 Referenz PostGIS **68**
7.1 PostgreSQL und PostGIS Datentypen - Geometry/Geography/Box . . . 68
7.1.1 box2d . . . 68
7.1.2 box3d . . . 69
7.1.3 geometry . . . 69
7.1.4 geometry_dump . . . 70

7.1.5 geography . . . 70
7.2 Geometrische Managementfunktionen . . . 71
7.2.1 AddGeometryColumn . . . 71
7.2.2 DropGeometryColumn . . . 73
7.2.3 DropGeometryTable . . . 73
7.2.4 Find_SRID . . . 74
7.2.5 Populate_Geometry_Columns . . . 75
7.2.6 UpdateGeometrySRID . . . 76
7.3 Geometrische Konstruktoren . . . 77
7.3.1 ST_Collect . . . 77
7.3.2 ST_LineFromMultiPoint . . . 79
7.3.3 ST_MakeEnvelope . . . 80
7.3.4 ST_MakeLine . . . 80
7.3.5 ST_MakePoint . . . 82
7.3.6 ST_MakePointM . . . 83
7.3.7 ST_MakePolygon . . . 84
7.3.8 ST_Point . . . 86
7.3.9 ST_PointZ . . . 87
7.3.10 ST_PointM . . . 88
7.3.11 ST_PointZM . . . 88
7.3.12 ST_Polygon . . . 89
7.3.13 ST_TileEnvelope . . . 90
7.3.14 ST_HexagonGrid . . . 90
7.3.15 ST_Hexagon . . . 93
7.3.16 ST_SquareGrid . . . 94
7.3.17 ST_Square . . . 95
7.3.18 ST_Letters . . . 96
7.4 Geometrische Zugriffsfunktionen . . . 97
7.4.1 GeometryType . . . 97
7.4.2 ST_Boundary . . . 98
7.4.3 ST_BoundingDiagonal . . . 100
7.4.4 ST_CoordDim . . . 101
7.4.5 ST_Dimension . . . 102
7.4.6 ST_Dump . . . 102
7.4.7 ST_DumpPoints . . . 104
7.4.8 ST_DumpSegments . . . 108
7.4.9 ST_DumpRings . . . 110
7.4.10 ST_EndPoint . . . 111
7.4.11 ST_Envelope . . . 112

7.4.12 ST_ExteriorRing . . . 114
7.4.13 ST_GeometryN . . . 115
7.4.14 ST_GeometryType . . . 117
7.4.15 ST_HasArc . . . 118
7.4.16 ST_InteriorRingN . . . 119
7.4.17 ST_IsClosed . . . 120
7.4.18 ST_IsCollection . . . 121
7.4.19 ST_IsEmpty . . . 122
7.4.20 ST_IsPolygonCCW . . . 124
7.4.21 ST_IsPolygonCW . . . 124
7.4.22 ST_IsRing . . . 125
7.4.23 ST_IsSimple . . . 126
7.4.24 ST_M . . . 127
7.4.25 ST_MemSize . . . 127
7.4.26 ST_NDims . . . 129
7.4.27 ST_NPoints . . . 129
7.4.28 ST_NRings . . . 130
7.4.29 ST_NumGeometries . . . 130
7.4.30 ST_NumInteriorRings . . . 131
7.4.31 ST_NumInteriorRing . . . 132
7.4.32 ST_NumPatches . . . 132
7.4.33 ST_NumPoints . . . 133
7.4.34 ST_PatchN . . . 133
7.4.35 ST_PointN . . . 134
7.4.36 ST_Points . . . 136
7.4.37 ST_StartPoint . . . 137
7.4.38 ST_Summary . . . 138
7.4.39 ST_X . . . 139
7.4.40 ST_Y . . . 140
7.4.41 ST_Z . . . 140
7.4.42 ST_Zmflag . . . 141
7.5 Geometrische Editoren . . . 142
7.5.1 ST_AddPoint . . . 142
7.5.2 ST_CollectionExtract . . . 143
7.5.3 ST_CollectionHomogenize . . . 144
7.5.4 ST_CurveToLine . . . 145
7.5.5 ST_Scroll . . . 148
7.5.6 ST_FlipCoordinates . . . 149
7.5.7 ST_Force2D . . . 149

7.5.8 ST_Force3D . . . 150
7.5.9 ST_Force3DZ . . . 151
7.5.10 ST_Force3DM . . . 152
7.5.11 ST_Force4D . . . 152
7.5.12 ST_ForcePolygonCCW . . . 153
7.5.13 ST_ForceCollection . . . 154
7.5.14 ST_ForcePolygonCW . . . 155
7.5.15 ST_ForceSFS . . . 155
7.5.16 ST_ForceRHR . . . 156
7.5.17 ST_ForceCurve . . . 156
7.5.18 ST_LineToCurve . . . 157
7.5.19 ST_Multi . . . 159
7.5.20 ST_LineExtend . . . 159
7.5.21 ST_Normalize . . . 160
7.5.22 ST_Project . . . 160
7.5.23 ST_QuantizeCoordinates . . . 161
7.5.24 ST_RemovePoint . . . 163
7.5.25 ST_RemoveRepeatedPoints . . . 164
7.5.26 ST_Reverse . . . 165
7.5.27 ST_Segmentize . . . 165
7.5.28 ST_SetPoint . . . 167
7.5.29 ST_ShiftLongitude . . . 168
7.5.30 ST_WrapX . . . 169
7.5.31 ST_SnapToGrid . . . 170
7.5.32 ST_Snap . . . 171
7.5.33 ST_SwapOrdinates . . . 174
7.6 Geometrievalidierung . . . 175
7.6.1 ST_IsValid . . . 175
7.6.2 ST_IsValidDetail . . . 176
7.6.3 ST_IsValidReason . . . 178
7.6.4 ST_MakeValid . . . 179
7.7 Spatial Reference System Functions . . . 184
7.7.1 ST_InverseTransformPipeline . . . 184
7.7.2 ST_SetSRID . . . 185
7.7.3 ST_SRID . . . 186
7.7.4 ST_Transform . . . 187
7.7.5 ST_TransformPipeline . . . 189
7.7.6 postgis_srs_codes . . . 191
7.7.7 postgis_srs . . . 191

7.7.8 postgis_srs_all . . . 192
7.7.9 postgis_srs_search . . . 193
7.8 Geometrische Konstruktoren . . . 194
7.8.1 Well-known-Text (WKT) Repräsentation . . . 194
7.8.1.1 ST_BdPolyFromText . . . 194
7.8.1.2 ST_BdMPolyFromText . . . 194
7.8.1.3 ST_GeogFromText . . . 195
7.8.1.4 ST_GeographyFromText . . . 195
7.8.1.5 ST_GeomCollFromText . . . 196
7.8.1.6 ST_GeomFromEWKT . . . 196
7.8.1.7 ST_GeomFromMARC21 . . . 198
7.8.1.8 ST_GeometryFromText . . . 200
7.8.1.9 ST_GeomFromText . . . 201
7.8.1.10 ST_LineFromText . . . 202
7.8.1.11 ST_MLineFromText . . . 203
7.8.1.12 ST_MPointFromText . . . 204
7.8.1.13 ST_MPolyFromText . . . 205
7.8.1.14 ST_PointFromText . . . 205
7.8.1.15 ST_PolygonFromText . . . 206
7.8.1.16 ST_WKTToSQL . . . 207
7.8.2 Well-known-Binary (WKB) Repräsentation . . . 208
7.8.2.1 ST_GeogFromWKB . . . 208
7.8.2.2 ST_GeomFromEWKB . . . 208
7.8.2.3 ST_GeomFromWKB . . . 210
7.8.2.4 ST_LineFromWKB . . . 211
7.8.2.5 ST_LinestringFromWKB . . . 211
7.8.2.6 ST_PointFromWKB . . . 212
7.8.2.7 ST_WKBToSQL . . . 213
7.8.3 Weitere Formate . . . 214
7.8.3.1 ST_Box2dFromGeoHash . . . 214
7.8.3.2 ST_GeomFromGeoHash . . . 214
7.8.3.3 ST_GeomFromGML . . . 215
7.8.3.4 ST_GeomFromGeoJSON . . . 218
7.8.3.5 ST_GeomFromKML . . . 219
7.8.3.6 ST_GeomFromTWKB . . . 220
7.8.3.7 ST_GMLToSQL . . . 220
7.8.3.8 ST_LineFromEncodedPolyline . . . 221
7.8.3.9 ST_PointFromGeoHash . . . 221
7.8.3.10 ST_FromFlatGeobufToTable . . . 222

7.8.3.11 ST_FromFlatGeobuf . . . 223
7.9 Geometrieausgabe . . . 223
7.9.1 Well-known-Text (WKT) Repräsentation . . . 223
7.9.1.1 ST_AsEWKT . . . 223
7.9.1.2 ST_AsText . . . 224
7.9.2 Well-known-Binary (WKB) Repräsentation . . . 226
7.9.2.1 ST_AsBinary . . . 226
7.9.2.2 ST_AsEWKB . . . 227
7.9.2.3 ST_AsHEXEWKB . . . 228
7.9.3 Weitere Formate . . . 229
7.9.3.1 ST_AsEncodedPolyline . . . 229
7.9.3.2 ST_AsFlatGeobuf . . . 230
7.9.3.3 ST_AsGeobuf . . . 230
7.9.3.4 ST_AsGeoJSON . . . 231
7.9.3.5 ST_AsGML . . . 233
7.9.3.6 ST_AsKML . . . 236
7.9.3.7 ST_AsLatLonText . . . 237
7.9.3.8 ST_AsMARC21 . . . 238
7.9.3.9 ST_AsMVTGeom . . . 241
7.9.3.10 ST_AsMVT . . . 242
7.9.3.11 ST_AsSVG . . . 243
7.9.3.12 ST_AsTWKB . . . 244
7.9.3.13 ST_AsX3D . . . 245
7.9.3.14 ST_GeoHash . . . 249
7.10 Operatoren . . . 250
7.10.1 Bounding Box Operators . . . 250
7.10.1.1 && . . . 250
7.10.1.2 &&(geometry,box2df) . . . 251
7.10.1.3 &&(box2df,geometry) . . . 252
7.10.1.4 &&(box2df,box2df) . . . 252
7.10.1.5 &&& . . . 253
7.10.1.6 &&&(geometry,gidx) . . . 254
7.10.1.7 &&&(gidx,geometry) . . . 255
7.10.1.8 &&&(gidx,gidx) . . . 256
7.10.1.9 &< . . . 257
7.10.1.10 &<| . . . 258
7.10.1.11 &> . . . 258
7.10.1.12 << . . . 259
7.10.1.13 <<| . . . 260

7.10.1.14 = . . . 261
7.10.1.15 >> . . . 262
7.10.1.16 @ . . . 263
7.10.1.17 @(geometry,box2df) . . . 264
7.10.1.18 @(box2df,geometry) . . . 264
7.10.1.19 @(box2df,box2df) . . . 265
7.10.1.20 |&> . . . 266
7.10.1.21 |>> . . . 267
7.10.1.22 ~ . . . 267
7.10.1.23 ~(geometry,box2df) . . . 268
7.10.1.24 ~(box2df,geometry) . . . 269
7.10.1.25 ~(box2df,box2df) . . . 270
7.10.1.26 ~= . . . 270
7.10.2 Operatoren . . . 271
7.10.2.1 <-> . . . 271
7.10.2.2 |=| . . . 273
7.10.2.3 <#> . . . 274
7.10.2.4 <<->> . . . 275
7.10.2.5 <<#>> . . . 276
7.11 Lagevergleiche . . . 277
7.11.1 Topologische Beziehungen . . . 277
7.11.1.1 ST_3DIntersects . . . 277
7.11.1.2 ST_Contains . . . 278
7.11.1.3 ST_ContainsProperly . . . 281
7.11.1.4 ST_CoveredBy . . . 283
7.11.1.5 ST_Covers . . . 284
7.11.1.6 ST_Crosses . . . 285
7.11.1.7 ST_Disjoint . . . 287
7.11.1.8 ST_Equals . . . 288
7.11.1.9 ST_Intersects . . . 289
7.11.1.10 ST_LineCrossingDirection . . . 291
7.11.1.11 ST_OrderingEquals . . . 294
7.11.1.12 ST_Overlaps . . . 295
7.11.1.13 ST_Relate . . . 298
7.11.1.14 ST_RelateMatch . . . 300
7.11.1.15 ST_Touches . . . 301
7.11.1.16 ST_Within . . . 303
7.11.2 Distance Relationships . . . 305
7.11.2.1 ST_3DDWithin . . . 305

7.11.2.2 ST_3DDFullyWithin . . . 306
7.11.2.3 ST_DFullyWithin . . . 306
7.11.2.4 ST_DWithin . . . 307
7.11.2.5 ST_PointInsideCircle . . . 308
7.12 Measurement Functions . . . 309
7.12.1 ST_Area . . . 309
7.12.2 ST_Azimuth . . . 311
7.12.3 ST_Angle . . . 312
7.12.4 ST_ClosestPoint . . . 313
7.12.5 ST_3DClosestPoint . . . 315
7.12.6 ST_Distance . . . 316
7.12.7 ST_3DDistance . . . 318
7.12.8 ST_DistanceSphere . . . 319
7.12.9 ST_DistanceSpheroid . . . 320
7.12.10 ST_FrechetDistance . . . 321
7.12.11 ST_HausdorffDistance . . . 322
7.12.12 ST_Length . . . 323
7.12.13 ST_Length2D . . . 325
7.12.14 ST_3DLength . . . 325
7.12.15 ST_LengthSpheroid . . . 326
7.12.16 ST_LongestLine . . . 327
7.12.17 ST_3DLongestLine . . . 329
7.12.18 ST_MaxDistance . . . 330
7.12.19 ST_3DMaxDistance . . . 331
7.12.20 ST_MinimumClearance . . . 332
7.12.21 ST_MinimumClearanceLine . . . 333
7.12.22 ST_Perimeter . . . 333
7.12.23 ST_Perimeter2D . . . 335
7.12.24 ST_3DPerimeter . . . 336
7.12.25 ST_ShortestLine . . . 336
7.12.26 ST_3DShortestLine . . . 338
7.13 Overlay Functions . . . 339
7.13.1 ST_ClipByBox2D . . . 339
7.13.2 ST_Difference . . . 340
7.13.3 ST_Intersection . . . 341
7.13.4 ST_MemUnion . . . 344
7.13.5 ST_Node . . . 344
7.13.6 ST_Split . . . 345
7.13.7 ST_Subdivide . . . 348

7.13.8 ST_SymDifference . . . 350
7.13.9 ST_UnaryUnion . . . 351
7.13.10 ST_Union . . . 352
7.14 Geometrieverarbeitung . . . 354
7.14.1 ST_Buffer . . . 354
7.14.2 ST_BuildArea . . . 359
7.14.3 ST_Centroid . . . 361
7.14.4 ST_ChaikinSmoothing . . . 363
7.14.5 ST_ConcaveHull . . . 364
7.14.6 ST_ConvexHull . . . 368
7.14.7 ST_DelaunayTriangles . . . 369
7.14.8 ST_FilterByM . . . 374
7.14.9 ST_GeneratePoints . . . 375
7.14.10 ST_GeometricMedian . . . 375
7.14.11 ST_LineMerge . . . 377
7.14.12 ST_MaximumInscribedCircle . . . 379
7.14.13 ST_LargestEmptyCircle . . . 381
7.14.14 ST_MinimumBoundingCircle . . . 383
7.14.15 ST_MinimumBoundingRadius . . . 384
7.14.16 ST_OrientedEnvelope . . . 385
7.14.17 ST_OffsetCurve . . . 386
7.14.18 ST_PointOnSurface . . . 390
7.14.19 ST_Polygonize . . . 392
7.14.20 ST_ReducePrecision . . . 394
7.14.21 ST_SharedPaths . . . 395
7.14.22 ST_Simplify . . . 398
7.14.23 ST_SimplifyPreserveTopology . . . 399
7.14.24 ST_SimplifyPolygonHull . . . 399
7.14.25 ST_SimplifyVW . . . 402
7.14.26 ST_SetEffectiveArea . . . 402
7.14.27 ST_TriangulatePolygon . . . 404
7.14.28 ST_VoronoiLines . . . 405
7.14.29 ST_VoronoiPolygons . . . 406
7.15 Coverages . . . 408
7.15.1 ST_CoverageInvalidEdges . . . 408
7.15.2 ST_CoverageSimplify . . . 410
7.15.3 ST_CoverageUnion . . . 411
7.16 Affine Transformations . . . 412
7.16.1 ST_Affine . . . 412

7.16.2 ST_Rotate . . . 414
7.16.3 ST_RotateX . . . 415
7.16.4 ST_RotateY . . . 416
7.16.5 ST_RotateZ . . . 417
7.16.6 ST_Scale . . . 418
7.16.7 ST_Translate . . . 419
7.16.8 ST_TransScale . . . 420
7.17 Clustering Functions . . . 421
7.17.1 ST_ClusterDBSCAN . . . 421
7.17.2 ST_ClusterIntersecting . . . 424
7.17.3 ST_ClusterIntersectingWin . . . 424
7.17.4 ST_ClusterKMeans . . . 425
7.17.5 ST_ClusterWithin . . . 427
7.17.6 ST_ClusterWithinWin . . . 428
7.18 Bounding Box Functions . . . 429
7.18.1 Box2D . . . 429
7.18.2 Box3D . . . 430
7.18.3 ST_EstimatedExtent . . . 430
7.18.4 ST_Expand . . . 431
7.18.5 ST_Extent . . . 433
7.18.6 ST_3DExtent . . . 434
7.18.7 ST_MakeBox2D . . . 435
7.18.8 ST_3DMakeBox . . . 436
7.18.9 ST_XMax . . . 436
7.18.10 ST_XMin . . . 437
7.18.11 ST_YMax . . . 438
7.18.12 ST_YMin . . . 439
7.18.13 ST_ZMax . . . 440
7.18.14 ST_ZMin . . . 441
7.19 Kilometrierung . . . 442
7.19.1 ST_LineInterpolatePoint . . . 442
7.19.2 ST_3DLineInterpolatePoint . . . 444
7.19.3 ST_LineInterpolatePoints . . . 444
7.19.4 ST_LineLocatePoint . . . 445
7.19.5 ST_LineSubstring . . . 446
7.19.6 ST_LocateAlong . . . 449
7.19.7 ST_LocateBetween . . . 449
7.19.8 ST_LocateBetweenElevations . . . 451
7.19.9 ST_InterpolatePoint . . . 452

7.19.10 ST_AddMeasure . . . 452
7.20 Trajectory Functions . . . 453
7.20.1 ST_IsValidTrajectory . . . 453
7.20.2 ST_ClosestPointOfApproach . . . 454
7.20.3 ST_DistanceCPA . . . 455
7.20.4 ST_CPAWithin . . . 456
7.21 SFCGAL Functions . . . 456
7.21.1 postgis_sfcgal_version . . . 456
7.21.2 postgis_sfcgal_full_version . . . 457
7.21.3 ST_3DArea . . . 457
7.21.4 ST_3DConvexHull . . . 458
7.21.5 ST_3DIntersection . . . 459
7.21.6 ST_3DDifference . . . 461
7.21.7 ST_3DUnion . . . 462
7.21.8 ST_AlphaShape . . . 464
7.21.9 ST_ApproximateMedialAxis . . . 466
7.21.10 ST_ConstrainedDelaunayTriangles . . . 467
7.21.11 ST_Extrude . . . 468
7.21.12 ST_ForceLHR . . . 470
7.21.13 ST_IsPlanar . . . 470
7.21.14 ST_IsSolid . . . 471
7.21.15 ST_MakeSolid . . . 471
7.21.16 ST_MinkowskiSum . . . 472
7.21.17 ST_OptimalAlphaShape . . . 473
7.21.18 ST_Orientation . . . 475
7.21.19 ST_StraightSkeleton . . . 476
7.21.20 ST_Tesselate . . . 477
7.21.21 ST_Volume . . . 479
7.22 Unterstützung von lang andauernden Transaktionen/Long Transactions . . . 480
7.22.1 AddAuth . . . 480
7.22.2 CheckAuth . . . 481
7.22.3 DisableLongTransactions . . . 482
7.22.4 EnableLongTransactions . . . 482
7.22.5 LockRow . . . 483
7.22.6 UnlockRows . . . 483
7.23 Version Functions . . . 484
7.23.1 PostGIS_Extensions_Upgrade . . . 484
7.23.2 PostGIS_Full_Version . . . 485
7.23.3 PostGIS_GEOS_Version . . . 485

7.23.4 PostGIS_GEOS_Compiled_Version . . . 486
7.23.5 PostGIS_Liblwgeom_Version . . . 486
7.23.6 PostGIS_LibXML_Version . . . 487
7.23.7 PostGIS_Lib_Build_Date . . . 487
7.23.8 PostGIS_Lib_Version . . . 488
7.23.9 PostGIS_PROJ_Version . . . 488
7.23.10 PostGIS_Wagyu_Version . . . 489
7.23.11 PostGIS_Scripts_Build_Date . . . 489
7.23.12 PostGIS_Scripts_Installed . . . 490
7.23.13 PostGIS_Scripts_Released . . . 490
7.23.14 PostGIS_Version . . . 491
7.24 PostGIS Grand Unified Custom Variables (GUCs) . . . 492
7.24.1 postgis.backend . . . 492
7.24.2 postgis.gdal_datapath . . . 492
7.24.3 postgis.gdal_enabled_drivers . . . 493
7.24.4 postgis.enable_outdb_rasters . . . 494
7.24.5 postgis.gdal_config_options . . . 495
7.25 Troubleshooting Functions . . . 496
7.25.1 PostGIS_AddBBox . . . 496
7.25.2 PostGIS_DropBBox . . . 496
7.25.3 PostGIS_HasBBox . . . 497

8 Topologie **499**
8.1 Topologische Datentypen . . . 499
8.1.1 getfaceedges_returntype . . . 499
8.1.2 TopoGeometry . . . 500
8.1.3 validatetopology_returntype . . . 500
8.2 Topologische Domänen . . . 501
8.2.1 TopoElement . . . 501
8.2.2 TopoElementArray . . . 501
8.3 Verwaltung von Topologie und TopoGeometry . . . 502
8.3.1 AddTopoGeometryColumn . . . 502
8.3.2 RenameTopoGeometryColumn . . . 503
8.3.3 DropTopology . . . 504
8.3.4 RenameTopology . . . 504
8.3.5 DropTopoGeometryColumn . . . 505
8.3.6 Populate_Topology_Layer . . . 505
8.3.7 TopologySummary . . . 506
8.3.8 ValidateTopology . . . 507

8.3.9 ValidateTopologyRelation . . . 509
8.3.10 FindTopology . . . 509
8.3.11 FindLayer . . . 510
8.4 Topology Statistics Management . . . 511
8.5 Topologie Konstruktoren . . . 511
8.5.1 CreateTopology . . . 511
8.5.2 CopyTopology . . . 512
8.5.3 ST_InitTopoGeo . . . 512
8.5.4 ST_CreateTopoGeo . . . 513
8.5.5 TopoGeo_AddPoint . . . 514
8.5.6 TopoGeo_AddLineString . . . 514
8.5.7 TopoGeo_AddPolygon . . . 515
8.6 Topologie Editoren . . . 515
8.6.1 ST_AddIsoNode . . . 515
8.6.2 ST_AddIsoEdge . . . 516
8.6.3 ST_AddEdgeNewFaces . . . 516
8.6.4 ST_AddEdgeModFace . . . 517
8.6.5 ST_RemEdgeNewFace . . . 518
8.6.6 ST_RemEdgeModFace . . . 518
8.6.7 ST_ChangeEdgeGeom . . . 519
8.6.8 ST_ModEdgeSplit . . . 520
8.6.9 ST_ModEdgeHeal . . . 521
8.6.10 ST_NewEdgeHeal . . . 521
8.6.11 ST_MoveIsoNode . . . 522
8.6.12 ST_NewEdgesSplit . . . 522
8.6.13 ST_RemoveIsoNode . . . 523
8.6.14 ST_RemoveIsoEdge . . . 524
8.7 Zugriffsfunktionen zur Topologie . . . 524
8.7.1 GetEdgeByPoint . . . 524
8.7.2 GetFaceByPoint . . . 525
8.7.3 GetFaceContainingPoint . . . 526
8.7.4 GetNodeByPoint . . . 526
8.7.5 GetTopologyID . . . 527
8.7.6 GetTopologySRID . . . 528
8.7.7 GetTopologyName . . . 528
8.7.8 ST_GetFaceEdges . . . 529
8.7.9 ST_GetFaceGeometry . . . 530
8.7.10 GetRingEdges . . . 530
8.7.11 GetNodeEdges . . . 531

8.8 Topologie Verarbeitung . . . 532
8.8.1 Polygonize . . . 532
8.8.2 AddNode . . . 532
8.8.3 AddEdge . . . 533
8.8.4 AddFace . . . 534
8.8.5 ST_Simplify . . . 536
8.8.6 RemoveUnusedPrimitives . . . 536
8.9 TopoGeometry Konstruktoren . . . 537
8.9.1 CreateTopoGeom . . . 537
8.9.2 toTopoGeom . . . 538
8.9.3 TopoElementArray_Agg . . . 540
8.9.4 TopoElement . . . 540
8.10 TopoGeometry Editoren . . . 541
8.10.1 clearTopoGeom . . . 541
8.10.2 TopoGeom_addElement . . . 541
8.10.3 TopoGeom_remElement . . . 542
8.10.4 TopoGeom_addTopoGeom . . . 542
8.10.5 toTopoGeom . . . 543
8.11 TopoGeometry Accessors . . . 543
8.11.1 GetTopoGeomElementArray . . . 543
8.11.2 GetTopoGeomElements . . . 544
8.11.3 ST_SRID . . . 544
8.12 TopoGeometry Ausgabe . . . 545
8.12.1 AsGML . . . 545
8.12.2 AsTopoJSON . . . 547
8.13 Räumliche Beziehungen einer Topologie . . . 549
8.13.1 Equals . . . 549
8.13.2 Intersects . . . 549
8.14 Importing and exporting Topologies . . . 550
8.14.1 Using the Topology exporter . . . 550
8.14.2 Using the Topology importer . . . 550

9 Rasterdatenverwaltung, -abfrage und Anwendungen 552
9.1 Laden und Erstellen von Rastertabellen . . . 552
9.1.1 Verwendung von raster2pgsql zum Laden von Rastern . . . 552
9.1.1.1 Example Usage . . . 552
9.1.1.2 raster2pgsql options . . . 553
9.1.2 Erzeugung von Rastern mit den PostGIS Rasterfunktionen . . . 554
9.1.3 Using "out db" cloud rasters . . . 555

9.2 Raster Katalog . . . 556
9.2.1 Rasterspalten Katalog . . . 556
9.2.2 Raster Übersicht/Raster Overviews . . . 557
9.3 Eigene Anwendungen mit PostGIS Raster erstellen . . . 558
9.3.1 PHP Beispiel: Ausgabe mittels ST_AsPNG in Verbindung mit anderen Rasterfunktionen . . . 558
9.3.2 ASP.NET C# Beispiel: Ausgabe mittels ST_AsPNG in Verbindung mit anderen Rasterfunktionen . . . 559
9.3.3 Applikation für die Java-Konsole, welche eine Rasterabfrage als Bilddatei ausgibt . . . 560
9.3.4 Verwenden Sie PLPython um Bilder via SQL herauszuschreiben . . . 562
9.3.5 Faster mit PSQL ausgeben . . . 562

10 Referenz Raster **564**
10.1 Datentypen zur Unterstützung von Rastern. . . . 565
10.1.1 geomval . . . 565
10.1.2 addbandarg . . . 565
10.1.3 rastbandarg . . . 565
10.1.4 raster . . . 566
10.1.5 reclassarg . . . 566
10.1.6 summarystats . . . 567
10.1.7 unionarg . . . 567
10.2 Rastermanagement . . . 568
10.2.1 AddRasterConstraints . . . 568
10.2.2 DropRasterConstraints . . . 570
10.2.3 AddOverviewConstraints . . . 571
10.2.4 DropOverviewConstraints . . . 572
10.2.5 PostGIS_GDAL_Version . . . 572
10.2.6 PostGIS_Raster_Lib_Build_Date . . . 572
10.2.7 PostGIS_Raster_Lib_Version . . . 573
10.2.8 ST_GDALDrivers . . . 573
10.2.9 ST_Contour . . . 578
10.2.10 ST_InterpolateRaster . . . 579
10.2.11 UpdateRasterSRID . . . 580
10.2.12 ST_CreateOverview . . . 580
10.3 Raster Constructors . . . 581
10.3.1 ST_AddBand . . . 581
10.3.2 ST_AsRaster . . . 583
10.3.3 ST_Band . . . 585
10.3.4 ST_MakeEmptyCoverage . . . 587
10.3.5 ST_MakeEmptyRaster . . . 588
10.3.6 ST_Tile . . . 589

10.3.7 ST_Retile . . . 592
10.3.8 ST_FromGDALRaster . . . 592
10.4 Zugriffsfunktionen auf Raster . . . 593
10.4.1 ST_GeoReference . . . 593
10.4.2 ST_Height . . . 594
10.4.3 ST_IsEmpty . . . 595
10.4.4 ST_MemSize . . . 595
10.4.5 ST_MetaData . . . 596
10.4.6 ST_NumBands . . . 596
10.4.7 ST_PixelHeight . . . 597
10.4.8 ST_PixelWidth . . . 598
10.4.9 ST_ScaleX . . . 599
10.4.10 ST_ScaleY . . . 600
10.4.11 ST_RasterToWorldCoord . . . 600
10.4.12 ST_RasterToWorldCoordX . . . 601
10.4.13 ST_RasterToWorldCoordY . . . 602
10.4.14 ST_Rotation . . . 603
10.4.15 ST_SkewX . . . 604
10.4.16 ST_SkewY . . . 604
10.4.17 ST_SRID . . . 605
10.4.18 ST_Summary . . . 606
10.4.19 ST_UpperLeftX . . . 606
10.4.20 ST_UpperLeftY . . . 607
10.4.21 ST_Width . . . 607
10.4.22 ST_WorldToRasterCoord . . . 608
10.4.23 ST_WorldToRasterCoordX . . . 609
10.4.24 ST_WorldToRasterCoordY . . . 609
10.5 Zugriffsfunktionen auf Rasterbänder . . . 610
10.5.1 ST_BandMetaData . . . 610
10.5.2 ST_BandNoDataValue . . . 611
10.5.3 ST_BandIsNoData . . . 612
10.5.4 ST_BandPath . . . 613
10.5.5 ST_BandFileSize . . . 614
10.5.6 ST_BandFileTimestamp . . . 614
10.5.7 ST_BandPixelType . . . 615
10.5.8 ST_MinPossibleValue . . . 616
10.5.9 ST_HasNoBand . . . 616
10.6 Zugriffsfunktionen und Änderungsmethoden für Rasterpixel . . . 617
10.6.1 ST_PixelAsPolygon . . . 617

10.6.2 ST_PixelAsPolygons . . . 618
10.6.3 ST_PixelAsPoint . . . 619
10.6.4 ST_PixelAsPoints . . . 619
10.6.5 ST_PixelAsCentroid . . . 620
10.6.6 ST_PixelAsCentroids . . . 621
10.6.7 ST_Value . . . 622
10.6.8 ST_NearestValue . . . 625
10.6.9 ST_SetZ . . . 627
10.6.10 ST_SetM . . . 628
10.6.11 ST_Neighborhood . . . 629
10.6.12 ST_SetValue . . . 632
10.6.13 ST_SetValues . . . 632
10.6.14 ST_DumpValues . . . 641
10.6.15 ST_PixelOfValue . . . 642
10.7 Raster Editoren . . . 643
10.7.1 ST_SetGeoReference . . . 643
10.7.2 ST_SetRotation . . . 644
10.7.3 ST_SetScale . . . 645
10.7.4 ST_SetSkew . . . 646
10.7.5 ST_SetSRID . . . 647
10.7.6 ST_SetUpperLeft . . . 647
10.7.7 ST_Resample . . . 648
10.7.8 ST_Rescale . . . 649
10.7.9 ST_Reskew . . . 651
10.7.10 ST_SnapToGrid . . . 652
10.7.11 ST_Resize . . . 653
10.7.12 ST_Transform . . . 654
10.8 Editoren für Rasterbänder . . . 657
10.8.1 ST_SetBandNoDataValue . . . 657
10.8.2 ST_SetBandIsNoData . . . 658
10.8.3 ST_SetBandPath . . . 659
10.8.4 ST_SetBandIndex . . . 661
10.9 Rasterband Statistik und Analytik . . . 663
10.9.1 ST_Count . . . 663
10.9.2 ST_CountAgg . . . 663
10.9.3 ST_Histogram . . . 664
10.9.4 ST_Quantile . . . 666
10.9.5 ST_SummaryStats . . . 668
10.9.6 ST_SummaryStatsAgg . . . 670

10.9.7 ST_ValueCount . . . 671
10.10 Rastereingabe . . . 673
10.10.1 ST_RastFromWKB . . . 673
10.10.2 ST_RastFromHexWKB . . . 674
10.11 Ausgabe von Rastern . . . 675
10.11.1 ST_AsBinary/ST_AsWKB . . . 675
10.11.2 ST_AsHexWKB . . . 676
10.11.3 ST_AsGDALRaster . . . 676
10.11.4 ST_AsJPEG . . . 678
10.11.5 ST_AsPNG . . . 679
10.11.6 ST_AsTIFF . . . 680
10.12 Raster Processing: Map Algebra . . . 680
10.12.1 ST_Clip . . . 680
10.12.2 ST_ColorMap . . . 683
10.12.3 ST_Grayscale . . . 686
10.12.4 ST_Intersection . . . 688
10.12.5 ST_MapAlgebra (callback function version) . . . 690
10.12.6 ST_MapAlgebra (expression version) . . . 696
10.12.7 ST_MapAlgebraExpr . . . 699
10.12.8 ST_MapAlgebraExpr . . . 701
10.12.9 ST_MapAlgebraFct . . . 706
10.12.10 ST_MapAlgebraFct . . . 709
10.12.11 ST_MapAlgebraFctNgb . . . 713
10.12.12 ST_Reclass . . . 715
10.12.13 ST_Union . . . 717
10.13 Integrierte Map Algebra Callback Funktionen . . . 718
10.13.1 ST_Distinct4ma . . . 718
10.13.2 ST_InvDistWeight4ma . . . 719
10.13.3 ST_Max4ma . . . 720
10.13.4 ST_Mean4ma . . . 721
10.13.5 ST_Min4ma . . . 722
10.13.6 ST_MinDist4ma . . . 723
10.13.7 ST_Range4ma . . . 724
10.13.8 ST_StdDev4ma . . . 725
10.13.9 ST_Sum4ma . . . 726
10.14 Raster Processing: DEM (Elevation) . . . 727
10.14.1 ST_Aspect . . . 727
10.14.2 ST_HillShade . . . 729
10.14.3 ST_Roughness . . . 731

10.14.4 ST_Slope . . . 731
10.14.5 ST_TPI . . . 733
10.14.6 ST_TRI . . . 734
10.15 Raster Processing: Raster to Geometry . . . 734
10.15.1 Box3D . . . 734
10.15.2 ST_ConvexHull . . . 735
10.15.3 ST_DumpAsPolygons . . . 736
10.15.4 ST_Envelope . . . 737
10.15.5 ST_MinConvexHull . . . 738
10.15.6 ST_Polygon . . . 739
10.16 Rasteroperatoren . . . 741
10.16.1 && . . . 741
10.16.2 &< . . . 741
10.16.3 &> . . . 742
10.16.4 = . . . 743
10.16.5 @ . . . 743
10.16.6 ~= . . . 744
10.16.7 ~ . . . 744
10.17 Räumliche Beziehungen von Rastern und Rasterbändern . . . 745
10.17.1 ST_Contains . . . 745
10.17.2 ST_ContainsProperly . . . 746
10.17.3 ST_Covers . . . 747
10.17.4 ST_CoveredBy . . . 748
10.17.5 ST_Disjoint . . . 748
10.17.6 ST_Intersects . . . 749
10.17.7 ST_Overlaps . . . 750
10.17.8 ST_Touches . . . 751
10.17.9 ST_SameAlignment . . . 752
10.17.10 ST_NotSameAlignmentReason . . . 753
10.17.11 ST_Within . . . 754
10.17.12 ST_DWithin . . . 755
10.17.13 ST_DFullyWithin . . . 756
10.18 Raster Tipps . . . 757
10.18.1 Out-DB Raster . . . 757
10.18.1.1 Das Verzeichnis enthält eine Vielzahl an Dateien . . . 757
10.18.1.2 Die maximale Anzahl geöffneter Dateien . . . 757
10.18.1.2.1 Die maximale Anzahl geöffneter Dateien für das ganze System . . . 758
10.18.1.2.2 Die maximale Anzahl geöffneter Dateien pro Prozess . . . 758

11 PostGIS Extras **760**

11.1 Adressennormierer . . . 760

11.1.1 Funktionsweise des Parsers . . . 760

11.1.2 Adressennormierer Datentypen . . . 761

11.1.2.1 stdaddr . . . 761

11.1.3 Adressennormierer Tabellen . . . 761

11.1.3.1 rules table . . . 761

11.1.3.2 lex table . . . 764

11.1.3.3 gaz table . . . 765

11.1.4 Adressennormierer Funktionen . . . 765

11.1.4.1 debug_standardize_address . . . 765

11.1.4.2 parse_address . . . 767

11.1.4.3 standardize_address . . . 768

11.2 Tiger Geokoder . . . 769

11.2.1 Drop_Indexes_Generate_Script . . . 770

11.2.2 Drop_Nation_Tables_Generate_Script . . . 771

11.2.3 Drop_State_Tables_Generate_Script . . . 771

11.2.4 Geocode . . . 772

11.2.5 Geocode_Intersection . . . 775

11.2.6 Get_Geocode_Setting . . . 776

11.2.7 Get_Tract . . . 777

11.2.8 Install_Missing_Indexes . . . 777

11.2.9 Loader_Generate_Census_Script . . . 778

11.2.10 Loader_Generate_Script . . . 780

11.2.11 Loader_Generate_Nation_Script . . . 782

11.2.12 Missing_Indexes_Generate_Script . . . 783

11.2.13 Normalize_Address . . . 784

11.2.14 Pagc_Normalize_Address . . . 785

11.2.15 Pprint_Addy . . . 787

11.2.16 Reverse_Geocode . . . 788

11.2.17 Topology_Load_Tiger . . . 790

11.2.18 Set_Geocode_Setting . . . 792

12 PostGIS Special Functions Index **793**

12.1 PostGIS Aggregate Functions . . . 793

12.2 PostGIS Window Functions . . . 794

12.3 PostGIS SQL-MM Compliant Functions . . . 794

12.4 PostGIS Geography Support Functions . . . 811

12.5 PostGIS Raster Support Functions . . . 812

12.6 PostGIS Geometry / Geography / Raster Dump Functions . . . 818
12.7 PostGIS Box Functions . . . 818
12.8 PostGIS Functions that support 3D . . . 820
12.9 PostGIS Curved Geometry Support Functions . . . 825
12.10 PostGIS Polyhedral Surface Support Functions . . . 828
12.11 PostGIS Function Support Matrix . . . 831
12.12 New, Enhanced or changed PostGIS Functions . . . 842
12.12.1 PostGIS Functions new or enhanced in 3.4 . . . 842
12.12.2 PostGIS Functions new or enhanced in 3.3 . . . 843
12.12.3 PostGIS Functions new or enhanced in 3.2 . . . 844
12.12.4 PostGIS Functions new or enhanced in 3.1 . . . 844
12.12.5 PostGIS Functions new or enhanced in 3.0 . . . 846
12.12.6 PostGIS Functions new or enhanced in 2.5 . . . 847
12.12.7 PostGIS Functions new or enhanced in 2.4 . . . 848
12.12.8 PostGIS Functions new or enhanced in 2.3 . . . 849
12.12.9 PostGIS Functions new or enhanced in 2.2 . . . 851
12.12.10 PostGIS Functions new or enhanced in 2.1 . . . 853
12.12.11 PostGIS Functions new or enhanced in 2.0 . . . 855
12.12.12 PostGIS Functions new or enhanced in 1.5 . . . 861
12.12.13 PostGIS Functions new or enhanced in 1.4 . . . 862
12.12.14 PostGIS Functions new or enhanced in 1.3 . . . 863

13 Meldung von Problemen **864**
13.1 Software Bugs melden . . . 864
13.2 Probleme mit der Dokumentation melden . . . 864

A Anhang **866**
A.1 PostGIS 3.4.0 . . . 866
A.1.1 New features . . . 866
A.1.2 Enhancements . . . 867
A.1.3 Breaking Changes . . . 867

Abstract

PostGIS ist eine Erweiterung des objektrelationalen Datenbanksystems PostgreSQL. Es ermöglicht die Speicherung von Geoobjekten eines GIS (Geoinformationssystem) in der Datenbank. PostGIS unterstützt räumliche, GIST-basierte R-Tree Indizes, sowie Funktionen zur Analyse und Bearbeitung von Geoobjekten.

Dieses Handbuch beschreibt die Version 3.4.0rc2

This work is licensed under a Creative Commons Attribution-Share Alike 3.0 License. Feel free to use this material any way you like, but we ask that you attribute credit to the PostGIS Project and wherever possible, a link back to https://postgis.net.

Chapter 1

Einführung

PostGIS erweitert das relationale Datenbanksystem PostgreSQL zu einer Geodatenbank. PostGIS wurde im Rahmen eines Technologieforschungsprojektes zu Geodatenbanken von Refractions Research Inc gegründet. Refractions ist ein Beratungsunternehmen für GIS und Datenbanken in Viktoria, British Columbia, Kanada, spezialisiert auf Datenintegration und Entwicklung von Individualsoftware.

PostGIS ist ein Projekt der OSGeo Foundation. PostGIS wird von vielen FOSS4G-Entwicklern und Unternehmen auf der ganzen Welt laufend verbessert und finanziert. Diese profitieren ihrerseits von der Funktionsvielfalt und Einsatzflexibilität von PostGIS.

Die PostGIS Project Develpment Group beabsichtigt durch die Unterstützung und Weiterentwicklung von PostGIS eine hohe Funktionsvielfalt zu erreichen. Diese soll wichtige GIS-Funktionalitäten, Kompatibilität mit den spatialen Standards OpenGIS und SQL/MM, hochentwickelte topologische Konstrukte (Coverages, Oberflächen, Netzwerke), Datenquellen für Desktop Benutzeroberflächen zum Darstellen und Bearbeiten von GIS Daten, sowie Werkzeuge für den Zugriff via Internettechnologie beinhalten.

1.1 Projektleitung

Das PostGIS Project Steering Committee (PSC) koordiniert die allgemeine Ausrichtung, den Releasezyklus, die Dokumentation und die Öffentlichkeitsarbeit des PostGIS Projektes. Zusätzlich bietet das PSC allgemeine Unterstützung für Anwender, übernimmt und prüft Patches aus der PostGIS Gemeinschaft und stimmt über sonstige Themen, wie Commit-Zugriff für Entwickler, neue PSC Mitglieder oder entscheidende Änderungen an der API, ab.

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

Regina Obe Buildbot Wartung, Kompilierung produktiver und experimenteller Softwarepakete für Windows, Abgleich von PostGIS mit den PostgreSQL Releases, allgemeine Unterstützung von Anwendern auf der PostGIS Newsgroup, Mitarbeit an X3D, Tiger Geokodierer, an Funktionen zur Verwaltung von Geometrien; Smoke testing neuer Funktionalität und wichtige Änderungen am Code.

Darafei Praliaskouski Index Optimierung, Bugfixes und Verbesserungen von Funktionen für den geometrischen/geographischer Datentyp, GitHub Verwalter und Wartung des Travis Bot.

Paul Ramsey (Vorsitzender) Mitbegründer des PostGIS Projektes. Allgemeine Fehlerbehebung, geographische Unterstützung, Indizes zur Unterstützung von Geographie und Geometrie (2D, 3D, nD Index und jegliche räumliche Indizes), grundlegende interne geometrische Strukturen, PointCloud (in Entwicklung), Einbindung von GEOS Funktionalität und Abstimmung mit GEOS Releases, Abglech von PostGIS mit den PostgreSQL Releases, Loader/Dumper und die Shapefile Loader GUI.

Sandro Santilli Bugfixes, Wartung, Git Mirrors Management und Integration neuer GEOS-Funktionalitäten, sowie Abstimmung mit den GEOS Versionen, Topologieunterstützung, Raster Grundstruktur und Funktionen der Low-Level-API.

1.2 Aktuelle Kernentwickler

Nicklas Avén Verbesserung und Erweiterung von Distanzfunktionen (einschließlich 3D-Distanz und Funktionen zu räumlichen Beziehungen), Tiny WKB Ausgabeformat (TWKB) (in Entwicklung) und allgemeine Unterstützung von Anwendern.

Dan Baston Beiträge zu den geometrischen Clusterfunktionen, Verbesserung anderer geometrischer Alorithmen, GEOS Erweiterungen und allgemeine Unterstützung von Anwendern.

Martin Davis GEOS enhancements and documentation

Björn Harrtell MapBox Vector Tile und GeoBuf Funktionen. Gogs Tests und GitLab Experimente.

Aliaksandr Kalenik Geometry Processing, PostgreSQL gist, general bug fixing

1.3 Frühere Kernentwickler

Bborie Park Prior PSC Member. Raster development, integration with GDAL, raster loader, user support, general bug fixing, testing on various OS (Slackware, Mac, Windows, and more)

Mark Cave-Ayland Koordiniert die Wartung und Fehlerbehebung, die Selektivität und die Anbindung von räumlichen Indizes, den Loader/Dumper und die Shapfile Loader GUI, die Einbindung von neuen Funktionen sowie die Verbesserung von neuen Funktionen.

Jorge Arévalo Entwicklung von PostGIS Raster, GDAL-Treiberunterstützung, Lader/loader

Olivier Courtin Ein- und Ausgabefunktionen für XML (KML,GML)/GeoJSON, 3D Unterstützng und Bugfixes.

Chris Hodgson Ehemaliges PSC Mitglied. Allgemeine Entwicklungsarbeit, Wartung von Buildbot und Homepage, OSGeo Inkubationsmanagement.

Mateusz Loskot CMake Unterstützung für PostGIS, Entwicklung des ursprünglichen Raster-Laders in Python und systemnahe Funktionen der Raster-API

Kevin Neufeld Ehemaliges PSC Mitglied. Dokumentation und Werkzeuge zur Dokumentationsunterstützung, Buildbot Wartung, fortgeschrittene Anwenderunterstützung auf der PostGIS Newsgroup, Verbesserungen an den Funktionen zur Verwaltung von Geometrien.

Dave Blasby Der ursprüngliche Entwickler und Mitbegründer von PostGIS. Dave schrieb die serverseitigen Bereiche, wie das Binden von Indizes und viele der serverseitiger analytischer Funktionen.

Jeff Lounsbury Ursprüngliche Entwicklung des Shapefile Loader/Dumper. Aktuell ist er Vertreter der PostGIS Projekt Inhaber.

Mark Leslie Laufende Wartung und Entwicklung der Kernfunktionen. Erweiterte Unterstützung von Kurven. Shapefile Loader GUI.

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

David Zwarg Entwickelt für Raster (in erster Linie analytische Funktionen in Map Algebra)

1.4 Weitere Mitwirkende

Die einzelnen Mitwirkenden

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

Gründungs-Sponsoren Dabei handelt es sich um Unternehmen, die Entwicklungszeit, Hosting, oder direkte finanzielle Förderungen, in das PostGIS Projekt eingebracht haben

- Aiven
- Arrival 3D
- Associazione Italiana per l'Informazione Geografica Libera (GFOSS.it)
- AusVet
- Avencia
- Azavea
- Boundless
- Cadcorp
- Camptocamp
- Carto
- Crunchy Data
- City of Boston (DND)

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

Crowd Funding-Kampagnen Wir starten Crowdfunding Kampagnen, um dringend gewünschte und von vielen Anwendern benötigte Funktionalitäten zu finanzieren. Jede Kampagne konzentriert sich auf eine bestimmte Funktionalität oder eine Gruppe von Funktionen. Jeder Sponsor spendiert einen kleinen Teil des benötigten Geldes und wenn genug Menschen/Organisationen mitmachen, können wir die Arbeit bezahlen, von der dann viele etwas haben. Falls Sie eine Idee für eine Funktionalität haben, bei der Sie glauben, dass viele andere bereit sind diese mitzufinanzieren, dann schicken Sie bitte Ihre Überlegungen an die PostGIS newsgroup - gemeinsam wird es uns gelingen.

PostGIS 2.0.0 war die erste Version, mit der wir diese Strategie verfolgten. Wir benutzten PledgeBank und hatten zwei erfolgreiche Kampagnen.

postgistopology - mehr als 10 Sponsoren förderten mit jeweils $250 USD die Entwicklung von TopoGeometry Funktionen und das Aufmöbeln der Topologie-Unterstützung für 2.0.0.

postgis64windows - 20 Sponsoren förderten die Arbeit an den Problemen mit der 64-bit Version von PostGIS für Windows mit jeweils $100 USD. Es ist tatsächlich geschehen und nun steht eine 64-bit Version von PostGIS 2.0.1 als PostgreSQL Stack-Builder zur Verfügung.

Wichtige Support-Bibliotheken The GEOS geometry operations library

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

The PROJ cartographic projection library

Zu guter Letzt das PostgreSQL DBMS, der Gigant auf dessen Schultern PostGIS steht. Die Geschwindigkeit und Flexibilität von PostGIS wäre ohne die Erweiterbarkeit, den großartigen Anfrageplaner, den GIST Index, und der Unmenge an SQL Funktionen, die von PostgreSQL bereitgestellt werden, nicht möglich.

Chapter 2

PostGIS Installation

Dieses Kapitel erläutert die notwendigen Schritte zur Installation von PostGIS.

2.1 Kurzfassung

Zum Kompilieren müssen die Abhängigkeiten im Suchpfad eingetragen sein:

```
tar -xvfz postgis-3.4.0rc2.tar.gz
cd postgis-3.4.0rc2
./configure
make
make install
```

Nachdem PostGIS installiert ist, muss es in jeder Datenbank-Instanz, in der es verwendet werden soll, aktiviert werden.

2.2 Kompilierung und Installation des Quellcodes: Detaillierte Beschreibung

Note

Viele Betriebssysteme stellen heute bereits vorkompilierte Pakete für PostgreSQL/PostGIS zur Verfügung. Somit ist eine Kompilation nur notwendig, wenn man die aktuellsten Versionen benötigt oder für die Paketverwaltung zuständig ist.

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

Pre-Built Packages for various OS are listed in PostGIS Pre-built Packages

Wenn Sie ein Windowsbenutzer sind, können Sie stabile Kompilationen mittels Stackbuilder oder die PostGIS Windows download site erhalten. Es gibt auch very bleeding-edge windows experimental builds, die ein oder zweimal pro Woche, bzw. anlassweise kompiliert werden. Damit können Sie mit im Aufbau befindlichen PostGIS Releases experimentieren.

The PostGIS module is an extension to the PostgreSQL backend server. As such, PostGIS 3.4.0rc2 *requires* full PostgreSQL server headers access in order to compile. It can be built against PostgreSQL versions 12 - 16. Earlier versions of PostgreSQL are *not* supported.

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

Note

Um die GEOS Funktionen nutzen zu können, muss bei der Installation von PostgreSQL explizit gegen die Standard C++ Bibliothek gelinkt werden:

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

Dies dient als Abhilfe für C++ Fehler bei der Interaktion mit älteren Entwicklungswerkzeugen. Falls eigenartige Probleme auftreten (die Verbindung zum Backend bricht unerwartet ab oder ähnliches) versuchen Sie bitte diesen Trick. Dies verlangt natürlich die Kompilation von PostgreSQL von Grund auf.

Die folgenden Schritte bescheiben die Konfiguration und Kompilation des PostGIS Quellcodes. Sie gelten für Linux Anwender und funktionieren nicht für Windows oder Mac.

2.2.1 Nutzung des Quellcodes

Das PostGIS Quellarchiv kann von der Download Webseite https://download.osgeo.org/postgis/source/postgis-3.4.0rc2.tar.gz bezogen werden.

```
wget https://download.osgeo.org/postgis/source/postgis-3.4.0rc2.tar.gz
tar -xvzf postgis-3.4.0rc2.tar.gz
cd postgis-3.4.0rc2
```

Dadurch wird das Verzeichnis `postgis-3.4.0rc2` im aktuellen Arbeitsverzeichnis erzeugt.

Alternativ kann der Quellcode auch von svn repository http://svn.osgeo.org/postgis/trunk/ bezogen werden.

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

Um die Installation fortzusetzen ist in das neu erstellte Verzeichnis `postgis-3.4.0rc2` zu wechseln.

```
./configure
```

2.2.2 Systemvoraussetzungen

Zur Kompilation und Anwendung stellt PostGIS die folgenden Systemanforderungen:

Notwendige Systemvoraussetzungen

- PostgreSQL 12 - 16. A complete installation of PostgreSQL (including server headers) is required. PostgreSQL is available from https://www.postgresql.org .

  For a full PostgreSQL / PostGIS support matrix and PostGIS/GEOS support matrix refer to https://trac.osgeo.org/postgis/wiki/UsersWikiPostgreSQLPostGIS

- GNU C Compiler (`gcc`). Es können auch andere ANSI C Compiler zur PostGIS Kompilation verwendet werden, aber die Kompilation mit `gcc` macht die geringsten Probleme.

- GNU Make (`gmake` oder `make`). Für viele Systeme ist GNU `make` die Standardversion von make. Überprüfe die Version durch `make -v`. Andere Versionen von `make` können das PostGIS `Makefile` nicht richtig ausführen.

- Proj reprojection library. Proj 6.1 or above is required. The Proj library is used to provide coordinate reprojection support within PostGIS. Proj is available for download from https://proj.org/ .

- GEOS geometry library, version 3.6 or greater, but GEOS 3.12+ is required to take full advantage of all the new functions and features. GEOS is available for download from https://libgeos.org .

- LibXML2, version 2.5.x or higher. LibXML2 is currently used in some imports functions (ST_GeomFromGML and ST_GeomFromKM LibXML2 is available for download from https://gitlab.gnome.org/GNOME/libxml2/-/releases.
- JSON-C, Version 0.9 oder höher. JSON-C wird zurzeit benutzt um GeoJSON über die Funktion ST_GeomFromGeoJson zu importieren. JSON-C kann unter https://github.com/json-c/json-c/releases/ bezogen werden.
- GDAL, version 2+ is required 3+ is preferred. This is required for raster support. https://gdal.org/download.html.
- Wenn mit PostgreSQL+JIT kompiliert wird, ist die LLVM-Version >=6 erforderlich https://trac.osgeo.org/postgis/ticket/4125 .

Optionale Systemanforderungen

- GDAL (pseudo optional) nur wenn Sie kein Rasterunterstützung möchten, können Sie es weglassen. Sorgen Sie außerdem dafür das Treiber, die Sie brauchen wie in Section 3.2 beschrieben, aktiviert sind.
- GTK (benötigt GTK+2.0, 2.8+) um den "shp2pgsql-gui shape file loader" zu kompilieren. http://www.gtk.org/ .
- SFCGAL, version 1.3.1 (or higher), 1.4.1 or higher is recommended and required to be able to use all functionality. SFCGAL can be used to provide additional 2D and 3D advanced analysis functions to PostGIS cf Section 7.21. And also allow to use SFCGAL rather than GEOS for some 2D functions provided by both backends (like ST_Intersection or ST_Area, for instance). A PostgreSQL configuration variable `postgis.backend` allow end user to control which backend he want to use if SFCGAL is installed (GEOS by default). Nota: SFCGAL 1.2 require at least CGAL 4.3 and Boost 1.54 (cf: https://oslandia.gitlab.io/SFCGAL/dev.html) https://gitlab.com/Oslandia/SFCGAL/.
- In order to build the Section 11.1 you will also need PCRE http://www.pcre.org (which generally is already installed on nix systems). Section 11.1 will automatically be built if it detects a PCRE library, or you pass in a valid `--with-pcre-dir=/path/to/p` during configure.
- Um ST_AsMVT verwenden zu können, wird die protobuf-c Bibliothek (für die Anwendung) und der protoc-c Kompiler (für die Kompilation) benötigt. Weiters ist pgk-config erforderlich um die korrekte Minimumversion von protobuf-c zu bestimmen. Siehe protobuf-c.
- CUnit (`CUnit`). Wird für Regressionstest benötigt. http://cunit.sourceforge.net/
- DocBook (`xsltproc`) ist für die Kompilation der Dokumentation notwendig. Docbook steht unter http://www.docbook.org/ zur Verfügung.
- DBLatex (`dblatex`) ist zur Kompilation der Dokumentation im PDF-Format nötig. DBLatex liegt unter http://dblatex.sourceforge.ne vor.
- ImageMagick (`convert`) wird zur Erzeugung von Bildern für die Dokumentation benötigt. ImageMagick kann von http://www.image bezogen werden.

2.2.3 Konfiguration

Wie bei den meisten Installationen auf Linux besteht der erste Schritt in der Erstellung eines Makefiles, welches dann zur Kompilation des Quellcodes verwendet wird. Dies wird durch einen Aufruf des Shell Scripts erreicht.

./configure

Ohne zusätzliche Parameter legt dieser Befehl die Komponenten und Bibliotheken fest, welche für die Kompilation des PostGIS Quellcodes auf Ihrem System benötigt werden. Obwohl dies der häufigste Anwendungsfall von **./configure** ist, akzeptiert das Skript eine Reihe von Parametern, falls sich die benötigten Bibliotheken und Programme nicht in den Standardverzeichnissen befinden.

Die folgende Liste weist nur die am häufigsten verwendeten Parameter auf. Für eine vollständige Liste benutzen Sie bitte **--help** oder **--help=short** .

--with-library-minor-version Starting with PostGIS 3.0, the library files generated by default will no longer have the minor version as part of the file name. This means all PostGIS 3 libs will end in `postgis-3`. This was done to make pg_upgrade easier, with downside that you can only install one version PostGIS 3 series in your server. To get the old behavior of file including the minor version: e.g. `postgis-3.0` add this switch to your configure statement.

--prefix=PREFIX Das Verzeichnis, in dem die PostGIS Bibliotheken und SQL-Skripts installiert werden. Standardmäßig ist dies das Verzeichnis in dem auch PostgreSQL installatiert wurde.

Caution
Dieser Parameter ist zur Zeit defekt; somit kann PostGIS nur in das PostgreSQL Installationsverzeichnis installiert werden. Dieser Bug kann auf http://trac.osgeo.org/postgis/ticket/635 verfolgt werden.

--with-pgconfig=FILE PostgreSQL stellt das Dienstprogramm **pg_config** zur Verfügung um Extensions wie PostGIS die Auffindung des PostgreSQL Installationsverzeichnisses zu ermöglichen. Benutzen Sie bitte diesen Parameter (**--with-pgconfig=/path/to/p** um eine bestmmte PostgreSQL Installation zu definieren, gegen die PostGIS kompiliert werden soll.

--with-gdalconfig=FILE GDAL, eine erforderliche Bibliothek, welche die Funktionalität zur Rasterunterstützung liefert. **gdal-config** um Software Installationen die Auffindung des GDAL Installationsverzeichnis zu ermöglichen. Benutzen Sie bitte diesen Parameter (**--with-gdalconfig=/path/to/gdal-config**) um eine bestimmte GDAL Installation zu definieren, gegen die PostGIS kompiliert werden soll.

--with-geosconfig=FILE GEOS, eine erforderliche Geometriebibliothek, stellt **geos-config** zur Verfügung, um Software Installationen das Auffinden des GEOS Installationsverzeichnisses zu ermöglichen. Benutzen Sie bitte diesen Parameter (**--with-geosconfig=/path/to/geos-config**) um eine bestimmte GEOS Installation zu definieren, gegen die PostGIS kompiliert werden soll.

--with-xml2config=FILE LibXML ist die Bibliothek, welche für die Prozesse GeomFromKML/GML benötigt wird. Falls Sie libxml installiert haben, wird sie üblicherweise gefunden. Falls nicht oder wenn Sie eine bestimmte Version verwenden wollen, müssen Sie PostGIS auf eine bestimmte Konfigurationsdatei `xml2-config` verweisen, damit Softwareinstallationen das Installationsverzeichnis von LibXML finden können. Verwenden Sie bitte diesen Parameter (**>--with-xml2config=/path/to/xml2-config**) um eine bestimmte LibXML Installation anzugeben, gegen die PostGIS kompiliert werden soll.

--with-projdir=DIR Proj4 ist eine Bibliothek, die von PostGIS zur Koordinatentransformation benötigt wird. Benutzen Sie bitte diesen Parameter (**--with-projdir=/path/to/projdir**) um ein bestimmtes Proj4 Installationsverzeichnis anzugeben, für das PostGIS kompiliert werden soll.

--with-libiconv=DIR Das Verzeichnis in dem iconv installiert ist.

--with-jsondir=DIR JSON-C ist eine MIT-lizensierte JSON Bibliothek, die von PostGIS für ST_GeomFromJSON benötigt wird. Benutzen Sie bitte diesen Parameter (**--with-jsondir=/path/to/jsondir**), um ein bestimmtes JSON-C Installationsverzeichnis anzugeben, für das PostGIS kompiliert werden soll.

--with-pcredir=DIR PCRE ist eine BSD-lizensierte Perl compatible Bibliothek für reguläre Ausdrücke, die von der Erweiterung "address_standardizer" benötigt wird. Verwenden Sie diesen Parameter (**--with-pcredir=/path/to/pcredir**), um ein bestimmtes Installationsverzeichnis von PCRE anzugeben, gegen das PostGIS kompiliert werden soll.

--with-gui Kompilieren Sie die Datenimport-GUI (benötigt GTK+2.0). Dies erzeugt die graphische Schnittstelle "shp2pgsql-gui" für shp2pgsql.

--without-raster Ohne Rasterunterstützung kompilieren.

--without-topology Ausschalten der Topologie Unterstützung. Es existiert keine entsprechende Bibliothek, da sich die gesamte benötigte Logik in der postgis-3.4.0rc2 Bibliothek befindet.

--with-gettext=no Standardmäßig versucht PostGIS gettext zu detektieren und kompiliert mit gettext Unterstützung. Wenn es allerdings zu Inkompatibilitätsproblemen kommt, die zu einem Zusammenbrechen des Loader führen, so können Sie das mit diesem Befehl zur Gänze deaktivieren. Siehe Ticket http://trac.osgeo.org/postgis/ticket/748 für ein Beispiel wie dieses Problem gelöst werden kann. Sie verpassen nicht viel, wenn Sie dies deaktivieren, da es für die internationale Hilfe zum GUI Loader/Label verwendet wird, welcher nicht dokumentiert und immer noch experimentell ist.

--with-sfcgal=PATH Ohne diesen Switch wird PostGIS ohne sfcgal Unterstützung installiert. `PATH` ist ein optionaler Parameter, welcher einen alternativen Pfad zu sfcgal-config angibt.

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

Note
Wenn Sie PostGIS vom Code Repository bezogen haben, müssen Sie zu allererst das Skript ausführen **./autogen.sh**
Dieses Skript erzeugt das **configure** Skript, welches seinerseits zur Anpassung der Installation von PostGIS eingesetzt wird.
Falls Sie stattdessen PostGIS als Tarball vorliegen haben, dann ist es nicht notwendig **./autogen.sh** auszuführen, da **configure** bereits erzeugt wurde.

2.2.4 Build-Prozess

Sobald das Makefile erzeugt wurde, ist der Build-Prozess für PostGIS so einfach wie

make

Die letzte Zeile der Ausgabe sollte "`PostGIS was built successfully. Ready to install.`" enthalten

Seit PostGIS v1.4.0 haben alle Funktionen Kommentare, welche aus der Dokumentation erstellt werden. Wenn Sie diese Kommentare später in die räumliche Datenbank importieren wollen, können Sie den Befehl ausführen der "docbook" benötigt. Die Dateien "postgis_comments.sql", "raster_comments.sql" und "topology_comments.sql" sind im Ordner "doc" der "tar.gz"-Distribution mit paketiert, weshalb Sie bei einer Installation vom "tar ball" her, die Kommentare nicht selbst erstellen müssen. Die Kommentare werden auch als Teil der Installation "CREATE EXTENSION" angelegt.

make comments

Eingeführt in PostGIS 2.0. Erzeugt HTML-Spickzettel, die als schnelle Referenz oder als Handzettel für Studenten geeignet sind. Dies benötigt xsltproc zur Kompilation und erzeugt 4 Dateien in dem Ordner "doc": `topology_cheatsheet.html`,`tiger_geocoder_cheatsheet.html`, `raster_cheatsheet.html`, `postgis_cheatsheet.html`

Einige bereits Vorgefertigte können von PostGIS / PostgreSQL Study Guides als HTML oder PDF heruntergeladen werden

make cheatsheets

2.2.5 Build-Prozess für die PostGIS Extensions und deren Bereitstellung

Die PostGIS Erweiterungen/Extensions werden ab PostgreSQL 9.1+ automatisch kompiliert und installiert.

Wenn Sie aus dem Quell-Repository kompilieren, müssen Sie zuerst die Beschreibung der Funktionen kompilieren. Diese lassen sich kompilieren, wenn Sie docbook installiert haben. Sie können sie aber auch händisch mit folgender Anweisung kompilieren:

make comments

Sie müssen die Kommentare nicht kompilieren, wenn sie von einem Format "tar" weg kompilieren, da diese in der tar-Datei bereits vorkompilierten sind.

Wenn Sie gegen PostgreSQL 9.1 kompilieren, sollten die Erweiterungen automatisch als Teil des Prozesses "make install" kompilieren. Falls notwendig, können Sie auch vom Ordner mit den Erweiterungen aus kompilieren, oder die Dateien auf einen anderen Server kopieren.

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

Note

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

Die Erweiterungsdateien sind für dieselbe Version von PostGIS immer ident, unabhängig vom Betriebssystem. Somit ist es in Ordnung, die Erweiterungsdateien von einem Betriebssystem auf ein anderes zu kopieren, solange die Binärdateien von PostGIS bereits installiert sind.

Falls Sie die Erweiterungen händisch auf einen anderen Server installieren wollen, müssen sie folgende Dateien aus dem Erweiterungsordner in den Ordner `PostgreSQL / share / extension` Ihrer PostgreSQL Installation kopieren. Ebenso die benötigten Binärdateien für das reguläre PostGIS, falls sich PostGIS noch nicht auf dem Server befindet.

- Dies sind die Kontrolldateien, welche Information wie die Version der zu installierenden Erweiterung anzeigen, wenn diese nicht angegben ist. `postgis.control, postgis_topology.control`.
- Alle Dateien in dem Ordner "/sql" der jeweiligen Erweiterung. Diese müssen in das Verzeichnis "share/extension" von PostgreSQL `extensions/postgis/sql/*.sql`, `extensions/postgis_topology/sql/*.sql` kopiert werden

Sobald Sie dies ausgeführt haben, sollten Sie `postgis`, `postgis_topology` als verfügbare Erweiterungen in PgAdmin -> extensions sehen.

Falls Sie psql verwenden, können Sie die installierten Erweiterungen folgendermaßen abfragen:

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

             name              | default_version | installed_version
-------------------------------+-----------------+-------------------
 address_standardizer          | 3.4.0rc2        | 3.4.0rc2
 address_standardizer_data_us  | 3.4.0rc2        | 3.4.0rc2
 postgis                       | 3.4.0rc2        | 3.4.0rc2
 postgis_raster                | 3.4.0rc2        | 3.4.0rc2
 postgis_sfcgal                | 3.4.0rc2        |
 postgis_tiger_geocoder        | 3.4.0rc2        | 3.4.0rc2
 postgis_topology              | 3.4.0rc2        |
(6 rows)
```

Wenn Sie in der Datenbank, die Sie abfragen, eine Erweiterung installiert haben, dann sehen Sie einen Hinweis in der Spalte `installed_version`. Wenn Sie keine Datensätze zurückbekommen bedeutet dies, dass Sie überhaupt keine PostGIS Erweiterung auf dem Server installiert haben. PgAdmin III 1.14+ bietet diese Information ebenfalls in der Sparte `extensions` im Navigationsbaum der Datenbankinstanz an und ermöglicht sogar ein Upgrade oder eine Deinstallation über einen Rechtsklick.

Wenn die Erweiterungen vorhanden sind, können Sie die PostGIS-Extension sowohl mit der erweiterten pgAdmin Oberfläche als auch mittels folgender SQL-Befehle in einer beliebigen Datenbank installieren:

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

Sie können psql verwenden, um sich die installierten Versionen und die Datenbankschemen in denen sie installiert sind, anzeigen zu lassen.

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

```
List of installed extensions
-[ RECORD 1 ]-------------------------------------------------
Name        | postgis
Version     | 3.4.0rc2
Schema      | public
Description | PostGIS geometry, geography, and raster spat..
-[ RECORD 2 ]-------------------------------------------------
Name        | postgis_raster
Version     | 3.0.0dev
Schema      | public
Description | PostGIS raster types and functions
-[ RECORD 3 ]-------------------------------------------------
Name        | postgis_tiger_geocoder
Version     | 3.4.0rc2
Schema      | tiger
Description | PostGIS tiger geocoder and reverse geocoder
-[ RECORD 4 ]-------------------------------------------------
Name        | postgis_topology
Version     | 3.4.0rc2
Schema      | topology
Description | PostGIS topology spatial types and functions
```

Warning

Die Erweiterungstabellen `spatial_ref_sys`. `layer` und `topology` können nicht explizit gesichert werden. Sie können nur mit der entsprechenden `postgis` oder `postgis_topology` Erweiterung gesichert werden, was nur geschieht, wenn Sie die ganze Datenbank sichern. Ab PostGIS 2.0.2 werden nur diejenigen Datensätze von SRID beim Backup der Datenbank gesichert, die nicht mit PostGIS paketiert sind. Sie sollten daher keine paketierte SRID ändern und Sie können erwarten, dass Ihre Änderungen gesichert werden. Wenn Sie irgendein Problem finden, reichen Sie bitte ein Ticket ein. Die Struktur der Erweiterungstabellen wird niemals gesichert, da diese mit `CREATE EXTENSION` erstellt wurde und angenommen wird, dass sie für eine bestimmten Version einer Erweiterung gleich ist. Dieses Verhalten ist in dem aktuellen PostgreSQL Extension Model eingebaut, weshalb wir daran nichts ändern können.

Wenn Sie 3.4.0rc2 ohne unser wunderbares Extension System installiert haben, können Sie auf erweiterungsbasiert wechseln, indem Sie folgende Befehle ausführen, welche die Funktionen in ihre entsprechenden Erweiterungen paketieren.

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

2.2.6 Softwaretest

Wenn Sie die Kompilation von PostGIS überprüfen wollen:

make check

Obiger Befehl durchläuft mehere Überprüfungen und Regressionstests, indem er die angelegte Bibliothek in einer aktuellen PostgreSQL Datenbank ausführt.

Note
Falls Sie PostGIS so konfiguriert haben, dass nicht die Standardverzeichnisse für PostgreSQL, GEOS oder Proj4 verwendet werden, kann es sein, dass Sie die Speicherstellen dieser Bibliotheken in der Umgebungsvariablen "LD_LIBRARY_PATH" eintragen müssen.

Caution
Zurzeit beruht **make check** auf die Umgebungsvariablen `PATH` und`PGPORT` beim Ausführen der Überprüfungen - es wird *nicht* die Version von PostgreSQL verwendet, die mit dem Konfigurationsparameter **--with-pgconfig** angegeben wurde. Daher stellen Sie sicher, dass die Variable PATH mit der während der Konfiguration dedektierten Installation von PostgreSQL übereinstimmt, oder seien Sie auf drohende Kopfschmerzen vorbereitet.

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

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

        .
        .
        .

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

        .
        .
        .

Running tests

        .
        .
        .

Run tests: 134
Failed: 0


-- if you build with SFCGAL

        .
        .
        .

Running tests

        .
        .
        .

Run tests: 13
Failed: 0

-- if you built with raster support
```

```
        .
        .
        .

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


        .
        .
        .

Running tests

        .
        .
        .

Run tests: 101
Failed: 0

-- topology regress

.
.
.

Running tests

        .
        .
        .

Run tests: 51
Failed: 0

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

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

        .
        .
        .

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

Die Erweiterungen `postgis_tiger_geocoder` und `address_standardizer` unterstützen zurzeit nur die standardmäßige Installationsüberprüfung von PostgreSQL. Um diese zu überprüfen siehe unterhalb. Anmerkung: "make install" ist nicht notwendig, wenn Sie bereits ein "make install" im Root des Ordners mit dem PostGIS Quellcode durchgeführt haben.

Für den address_standardizer:

```
cd extensions/address_standardizer
make install
make installcheck
```

Die Ausgabe sollte folgendermaßen aussehen:

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

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

Für den Tiger Geokodierer müssen Sie die Erweiterungen "postgis" und "fuzzystrmatch" in Ihrer PostgreSQL Instanz haben. Die Überprüfungen des "address_standardizer" laufen ebenfalls an, wenn Sie postgis mit "address_standardizer" Unterstützung kompiliert haben:

```
cd extensions/postgis_tiger_geocoder
make install
make installcheck
```

Die Ausgabe sollte folgendermaßen aussehen:

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

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

2.2.7 Installation

Um PostGIS zu installieren geben Sie bitte folgendes ein

make install

Dies kopiert die Installationsdateien von PostGIS in das entsprechende Unterverzeichnis, welches durch den Konfigurationsparameter **--prefix** bestimmt wird. Insbesondere:

- Die Binärdateien vom Loader und Dumper sind unter `[prefix]/bin` installiert.
- Die SQL-Dateien, wie `postgis.sql` sind unter `[prefix]/share/contrib` installiert.

- Die PostGIS Bibliotheken sind unter `[prefix]/lib` installiert.

Falls Sie zuvor den Befehl **make comments** ausgeführt haben, um die Dateien `postgis_comments.sql` und `raster_comments.sql` anzulegen, können Sie die SQL-Dateien folgendermaßen installieren:

make comments-install

Note

`postgis_comments.sql`, `raster_comments.sql` und `topology_comments.sql` wurden vom klassischen Build- und Installationsprozess getrennt, da diese mit **xsltproc** eine zusätzliche Abhängigkeit haben.

2.3 Installation und Verwendung des Adressennormierers

Die Erweiterung `address_standardizer` musste als getrenntes Paket heruntergeladen werden. Ab PostGIS 2.2 ist es mitgebündelt. Für weitere Informationen zu dem address_standardizer, was er kann und wie man ihn für spezielle Bedürfnisse konfigurieren kann, siehe Section 11.1.

Dieser Adressennormierer kann in Verbindung mit der in PostGIS paketierten Erweiterung "tiger gecoder" als Ersatz für Normalize_Address verwendet werden. Um diesen als Ersatz zu nutzen, siehe Section 2.4.2. Sie können diesen auch als Baustein für Ihren eigenen Geokodierer verwenden oder für die Normierung von Adressen um diese leichter vergleichbar zu machen.

Der Adressennormierer benötigt PCRE, welches üblicherweise auf Nix-Systemen bereits installiert ist. Sie können die letzte Version aber auch von http://www.pcre.org herunterladen. Wenn PCRE während der Section 2.2.3 gefunden wird, dann wird die Erweiterung "address standardizer" automatisch kompiliert. Wenn Sie stattdessen eine benutzerdefinierte Installation von PCRE verwenden wollen, können Sie `--with-pcredir=/path/to/pcre` an "configure" übergeben, wobei `/path/to/pcre` der Root-Ordner Ihrer Verzeichnisse "include" und "lib" von PCRE ist.

Für Windows Benutzer ist ab PostGIS 2.1+ die Erweiterung "address_standardizer" bereits mitpaketiert. Somit besteht keine Notwendigkeit zu Kompilieren und es kann sofort der Schritt `CREATE EXTENSION` ausgeführt werden.

Sobald die Installation beendet ist, können Sie sich mit Ihrer Datenbank verbinden und folgenden SQL-Befehl ausführen:

```
CREATE EXTENSION address_standardizer;
```

Der folgende Test benötigt keine rules-, gaz- oder lex-Tabellen

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

Die Ausgabe sollte wie folgt sein:

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

2.4 Installing, Upgrading Tiger Geocoder, and loading data

Extras wie den Tiger Geokodierer befinden sich möglicherweise nicht in Ihrer PostGIS Distribution. Wenn Sie die Erweiterung "Tiger Geokodierer" vermissen, oder eine neuere Version installieren wollen, dann können Sie die Dateien `share/extension/postgis_tiger_geocoder.*` aus den Paketen des Abschnitts Windows Unreleased Versions für Ihre Version von PostgreSQL verwenden. Obwohl diese Pakete für Windows sind, funktionieren die Dateien der Erweiterung "postgis_tiger_geocoder" mit jedem Betriebssystem, da die Erweiterung eine reine SQL/plpgsql Anwendung ist.

2.4.1 Tiger Geocoder Enabling your PostGIS database

1. These directions assume your PostgreSQL installation already has the postgis_tiger_geocoder extension installed.

2. Verbinden Sie sich zu Ihrer Datenbank über psql, pgAdmin oder ein anderes Werkzeug und führen Sie die folgenden SQL Befehle aus. Wenn Sie in eine Datenbank installieren, die bereits PostGIS beinhaltet, dann müssen Sie den ersten Schritt nicht ausführen. Wenn Sie auch die Erweiterung `fuzzystrmatch` bereits installiert haben, so müssen Sie auch den zweiten Schritt nicht ausführen.

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

   Wenn Sie bereits die postgis-tiger-geocoder Extension installiert haben und nur auf den letzten Stand updaten wollen:

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

   Wenn benutzerdefinierte Einträge oder Änderungen an `tiger.loader_platform` oder `tiger.loader_variables` gemacht wurden, müssen diese aktualisiert werden.

3. Um die Richtigkeit der Installation festzustellen, führen Sie bitte folgenden SQL-Befehl in Ihrer Datenbank aus:

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

   Dies sollte folgendes ausgeben:

```
address | streetname | streettypeabbrev |  zip
---------+------------+------------------+-------
           1 | Devonshire | Pl               | 02109
```

4. Erstellen Sie einen neuen Datensatz in der Tabelle `tiger.loader_platform`, welcher die Pfade zu Ihren ausführbaren Dateien und zum Server beinhaltet.

   Um zum Beispiel ein Profil mit dem Namen "debbie" anzulegen. welches der `sh` Konvention folgt, können Sie folgendes tun:

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

   Anschließend ändern Sie die Pfade in der Spalte *declare_sect*, so dass diese mit den Speicherpfaden von Debbie's "pg", "nzip", "shp2pgsql", "psql", etc. übereinstimmen.

   Wenn Sie die Tabelle `loader_platform` nicht editieren, so beinhaltet diese lediglich die üblichen Ortsangaben und Sie müssen das erzeugte Skript editieren, nachdem es erzeugt wurde.

5. Ab PostGIS 2.4.1 wurde der Ladevorgang der "Zip code-5 digit tabulation area" `zcta5` überarbeitet, um aktuelle zcta5 Daten zu laden und ist nun ein Teil von Loader_Generate_Nation_Script, falls aktiviert. Standardmäßig ausgeschaltet, da der Ladevorgang ziemlich viel Zeit benötigt (20 bis 60 Minuten), ziemlich viel Festplattenspeicher beansprucht wird und es nur selten verwendet wird.

   Folgendermaßen können Sie deise aktivieren:

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

Falls vorhanden kann die Funktion Geocode diese verwenden, wenn die zips durch einen Boundary Filter begrenzt sind. Die Funktion Reverse_Geocode verwendet dies wenn eine zurückgegebene Adresse keinen zip-Code enthält, was oft bei der inversen Geokodierung von Highways auftritt.

6. Erstellen Sie einen Ordner mit der Bezeichnung `gisdata` im Root des Servers oder auf Ihrem lokalen PC, wenn Sie eine schnelle Netzwerkverbindung zu dem Server haben. In diesen Ordner werden die Dateien von Tiger heruntergeladen und aufbereitet. Wenn Sie den Ordner nicht im Root des Servers haben wollen, oder für die Staging-Umgebung in eine anderen Ordner wechseln wollen, dann können Sie das Attribut `staging_fold` in der Tabelle `tiger.loader_variables` editieren.

7. Erstellen Sie einen Ordner "temp" in dem Ordner `gisdata` oder wo immer Sie `staging_fold` haben wollen. Dies wird der Ordner, in dem der Loader die heruntergeladenen Tigerdaten extrahiert.

8. Anschließend führen Sie die SQL Funktion Loader_Generate_Nation_Script aus, um sicherzustellen dass die Bezeichnung Ihres benutzerdefinierten Profils verwendet wird und kopieren das Skript in eine .sh oder .bat Datei. Um zum Beispiel das Skript zum Laden einer Nation zu erzeugen:

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

9. Führen Sie die erzeugten Skripts zum Laden der Nation auf der Befehlszeile aus.

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

10. Nachdem Sie das "Nation" Skript ausgeführt haben, sollten sich drei Tabellen in dem Schema `tiger_data` befinden und mit Daten befüllt sein. Führen Sie die folgenden Abfragen in "psql" oder "pgAdmin" aus, um dies sicher zu stellen

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

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

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

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

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

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

Alternativ können Sie diese Tabellen nach dem Laden der Länderdaten importieren, indem Sie das Loader_Generate_Census_Scrip verwenden

12. Für jeden Staat, für den Sie Daten laden wollen, müssen Sie ein Skript Loader_Generate_Script erstellen.

Warning
Erstellen Sie das Skript für die Bundesstaaten NICHT bevor die Daten zur Nation geladen wurden, da das Skript die Liste "county" verwendet, welche durch das "nation"-Skript geladen wird.

13.
```
psql -c "SELECT Loader_Generate_Script(ARRAY['MA'], 'debbie')" -d geocoder -tA
> /gisdata/ma_load.sh
```

14. Die vorher erzeugten, befehlszeilenorientierten Skripts ausführen.

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

15. Nachdem Sie mit dem Laden der Daten fertig sind, ist es eine gute Idee ein ANALYZE auf die Tigertabellen auszuführen, um die Datenbankstatistik (inklusive vererbter Statistik) zu aktualisieren

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

2.4.2 Die Adressennormierer-Extension zusammen mit dem Tiger Geokodierer verwenden

Eine von vielen Beschwerden betrifft die Funktion Normalize_Address des Adressennormierers, die eine Adresse vor der Geokodierung vorbereitend standardisiert. Der Normierer ist bei weitem nicht perfekt und der Versuch seine Unvollkommenheit auszubessern nimmt viele Ressourcen in Anspruch. Daher haben wir ein anderes Projekt integriert, welches eine wesentlich bessere Funktionseinheit für den Adressennormierer besitzt. Um diesen neuen Adressennormierer zu nutzen, können Sie die Erweiterung so wie unter Section 2.3 beschrieben kompilieren und als Extension in Ihrer Datenbank installieren.

Sobald Sie diese Extension in der gleichen Datenbank installieren, in der Sie auch `postgis_tiger_geocoder` installiert haben, dann können Sie Pagc_Normalize_Address anstatt Normalize_Address verwenden. Diese Erweiterung ist nicht auf Tiger beschränkt, wodurch sie auch mit anderen Datenquellen, wie internationalen Adressen, genutzt werden kann. Die Tiger Geokodierer Extension enthält eine eigenen Versionen von rules table (`tiger.pagc_rules`), gaz table (`tiger.pagc_gaz`) und lex table (`tiger.pagc_lex`). Diese können Sie hinzufügen und aktualisieren, um die Normierung an die eigenen Bedürfnisse anzupassen.

2.4.3 Required tools for tiger data loading

Der Ladeprozess lädt Daten von der Census Webseite für die jeweiligen Nationsdateien und die angeforderten Bundesstaaten herunter, extrahiert die Dateien und lädt anschließlich jeden Bundesstaat in einen eigenen Satz von Bundesstaattabellen. Jede Bundesstaattabelle erbt von den Tabellen im Schema `tiger`, wodurch es ausreicht nur diese Tabellen abzufragen um auf alle Daten zugreifen zu können. Sie können auch jederzeit Bundesstaattabellen mit Drop_State_Tables_Generate_Script löschen, wenn Sie einen Bundesstaat neu laden müssen oder den Bundesstaat nicht mehr benötigen.

Um Daten laden zu können benötigen Sie folgende Werkzeuge:

- Ein Werkzeug, um die Zip-Dateien der Census Webseite zu entpacken.

  Auf UNIX-ähnlichen Systemen: Das Programm `unzip`, das üblicherweise auf den meisten UNIX-ähnlichen Systemen bereits vorinstalliert ist.

  Auf Windows 7-zip, ein freies Werkzeug zum komprimieren/entkomprimieren, das Sie von http://www.7-zip.org/ herunterladen können.

- Das `shp2pgsql` Kommandozeilenprogramm, welches standardmäßig mit PostGIS mitinstalliert wird.

- `wget`, ein Download-Manager, der üblicherweise auf den meisten UNIX/Linux Systemen vorinstalliert ist.

  Für Windows können Sie vorkompilierte Binärdateien von http://gnuwin32.sourceforge.net/packages/wget.htm herunterladen

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

Wie ein Skript zum Laden der Daten für Ihre Plattform und für die gewünschten Bundesstaaten generiert werden kann siehe Loader_Generate_Script. Sie können diese stückchenweise installieren. Sie müssen nicht alle benötigten Staaten auf einmal laden. Sie können sie laden wenn Sie diese benötigen.

Nachdem die gewünschten Bundesstaaten geladen wurden, führen Sie so wie unter Install_Missing_Indexes beschrieben

```
SELECT install_missing_indexes();
```

aus.

Um zu überprüfen, dass alles funktioniert wie es sollte, können Sie eine Geokodierung über eine Adresse Ihres Staates laufen lassen, indem Sie Geocode verwenden

2.4.4 Upgrading your Tiger Geocoder Install and Data

First upgrade your postgis_tiger_geocoder extension as follows:

```
ALTER EXTENSION postgis_tiger_geocoder UPDATE;
```

Anschließend löschen Sie alle "nation"-Tabellen und laden die Neuen. Erstellen Sie ein "drop"-Skript mit den unter Drop_Nation_Tables_ beschriebenen SQL-Anweisungen

```
SELECT drop_nation_tables_generate_script();
```

Führen Sie die erstellten SQL "drop"-Anweisungen aus.

Die untere SELECT Anweisung erstellt ein Skript zum Laden eines Staates. Details dazu finden Sie unter Loader_Generate_Nation_Scrip

Auf Windows:

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

Auf Unix/Linux:

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

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

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

2.5 Übliche Probleme bei der Installation

Falls Ihre Installation/Upgrade nicht so verläuft wie erwartet, gibt es eine ganze Reihe von Dingen zu überprüfen.

1. Überprüfen Sie, ob Sie PostgreSQL 12 oder neuer installiert haben und dass die Version des PostgreSQL Quellcodes, gegen den Sie kompilieren, mit der Version der laufenden PostgreSQL Datenbank übereinstimmt. Ein Wirrwarr kann dann entstehen, wenn die Linux Distribution bereits PostgreSQL installiert hat, oder wenn Sie PostgreSQL in einem anderen Zusammenhang installiert und darauf vergessen haben. PostGIS funktioniert nur mit PostgreSQL 12 oder jünger und es kommt zu merkwürdigen, unerwarteten Fehlermeldungen, wenn Sie eine ältere Version verwenden. Um die Version Ihrer laufenden PostgreSQL Datenbank zu überprüfen, können Sie sich mittels psql zur Datenbank verbinden und folgende Anfrage ausführen:

   ```
   SELECT version();
   ```

   Falls Sie eine RPM-basierte Distribution am Laufen haben, können Sie nach vorinstallierten Paketen mit dem Befehl **rpm** suchen: **rpm -qa | grep postgresql**

2. Wenn das Upgrade schief geht, stellen Sie bitte sicher, dass PostGIS, in der Datenbank die Sie wiederherstellen wollen, installiert ist.

   ```
   SELECT postgis_full_version();
   ```

Überprüfen Sie bitte auch, ob "configure" den korrekten Speicherort und die korrekte Version von PostgreSQL, sowie der Bibliotheken Proj4 und GEOS gefunden hat.

1. Die Ausgabe von configure wird verwendet, um die Datei `postgis_config.h` zu erstellen. Überprüfen Sie bitte, ob die Variablen `POSTGIS_PGSQL_VERSION`, `POSTGIS_PROJ_VERSION` und `POSTGIS_GEOS_VERSION` korrekt gesetzt sind.

Chapter 3

PostGIS Verwaltung

3.1 Leistungsoptimierung

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

For general details about optimizing PostgreSQL, refer to Tuning your PostgreSQL Server.

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

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

In addition to the Postgres settings, PostGIS has some custom settings which are listed in Section 7.24.

3.1.1 Startup

These settings are configured in `postgresql.conf`:

constraint_exclusion

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

shared_buffers

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

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

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

3.1.2 Runtime

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

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

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

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

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

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

max_parallel_workers_per_gather

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

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

3.2 Configuring raster support

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

As of PostGIS 2.1.3, out-of-db rasters and all raster drivers are disabled by default. In order to re-enable these, you need to set the following environment variables `POSTGIS_GDAL_ENABLED_DRIVERS` and `POSTGIS_ENABLE_OUTDB_RASTERS` in the server environment. For PostGIS 2.2, you can use the more cross-platform approach of setting the corresponding Section 7.24.

If you want to enable offline raster:

```
POSTGIS_ENABLE_OUTDB_RASTERS=1
```

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

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

```
POSTGIS_GDAL_ENABLED_DRIVERS=ENABLE_ALL
```

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

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

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

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

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

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

3.3 Creating spatial databases

3.3.1 Spatially enable database using EXTENSION

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

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

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

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

3.3.2 Spatially enable database without using EXTENSION (discouraged)

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

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

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

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

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

Run the following Shell commands in your terminal:

```
DB=[yourdatabase]
    SCRIPTSDIR=`pg_config --sharedir`/contrib/postgis-3.3/

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

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

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

3.4 Upgrading spatial databases

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

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

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

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

3.4.1 Soft upgrade

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

3.4.1.1 Soft Upgrade 9.1+ using extensions

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

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

```
SELECT postgis_extensions_upgrade();
```

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

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

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

```
ALTER EXTENSION postgis UPDATE TO "3.4.0rc2";
ALTER EXTENSION postgis_topology UPDATE TO "3.4.0rc2";
```

If you get an error notice something like:

```
No migration path defined for ... to 3.4.0rc2
```

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

If you get a notice message like:

```
Version "3.4.0rc2" of extension "postgis" is already installed
```

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

```
ALTER EXTENSION postgis UPDATE TO "3.4.0rc2next";
ALTER EXTENSION postgis_topology UPDATE TO "3.4.0rc2next";
```

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

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

```
DROP EXTENSION postgis_raster;
```

3.4.1.2 Soft Upgrade Pre 9.1+ or without extensions

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

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

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

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

```
ls `pg_config --sharedir`/contrib/postgis-3.4.0rc2/*_upgrade.sql
```

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

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

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

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

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

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

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

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

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

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

3.4.2 Hard upgrade

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

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

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

The Procedure is as follows:

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

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

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

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

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

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

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

Errors may arise in the following cases:

1. Some of your views or functions make use of deprecated PostGIS objects. In order to fix this you may try loading `legacy.sql` script prior to restore or you'll have to restore to a version of PostGIS which still contains those objects and try a migration again after porting your code. If the `legacy.sql` way works for you, don't forget to fix your code to stop using deprecated functions and drop them loading `uninstall_legacy.sql`.

2. Some custom records of spatial_ref_sys in dump file have an invalid SRID value. Valid SRID values are bigger than 0 and smaller than 999000. Values in the 999000.999999 range are reserved for internal use while values > 999999 can't be used at all. All your custom records with invalid SRIDs will be retained, with those > 999999 moved into the reserved range, but the spatial_ref_sys table would lose a check constraint guarding for that invariant to hold and possibly also its primary key (when multiple invalid SRIDS get converted to the same reserved SRID value).

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

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

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

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

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

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

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

   run this command :

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

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

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

Chapter 4

Data Management

4.1 Spatial Data Model

4.1.1 OGC Geometry

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

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

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

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

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

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

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

An important property of geometry values is their spatial **extent** or **bounding box**, which the OGC model calls **envelope**. This is the 2 or 3-dimensional box which encloses the coordinates of a geometry. It is an efficient way to represent a geometry's extent in coordinate space and to check whether two geometries interact.

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

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

4.1.1.1 Point

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

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

4.1.1.2 LineString

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

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

4.1.1.3 LinearRing

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

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

4.1.1.4 Polygon

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

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

4.1.1.5 MultiPoint

A MultiPoint is a collection of Points.

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

4.1.1.6 MultiLineString

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

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

4.1.1.7 MultiPolygon

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

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

4.1.1.8 GeometryCollection

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

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

4.1.1.9 PolyhedralSurface

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

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

4.1.1.10 Triangle

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

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

4.1.1.11 TIN

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

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

4.1.2 SQL/MM Part 3 - Curves

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

Note
Alle Gleitpunkt Vergleiche der SQL-MM Implementierung werden mit einer bestimmten Toleranz ausgeführt, zurzeit 1E-8.

4.1.2.1 CircularString

CircularString is the basic curve type, similar to a LineString in the linear world. A single arc segment is specified by three points: the start and end points (first and third) and some other point on the arc. To specify a closed circle the start and end points are the same and the middle point is the opposite point on the circle diameter (which is the center of the arc). In a sequence of arcs the end point of the previous arc is the start point of the next arc, just like the segments of a LineString. This means that a CircularString must have an odd number of points greater than 1.

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

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

4.1.2.2 CompoundCurve

A CompoundCurve is a single continuous curve that may contain both circular arc segments and linear segments. That means that in addition to having well-formed components, the end point of every component (except the last) must be coincident with the start point of the following component.

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

4.1.2.3 CurvePolygon

A CurvePolygon is like a polygon, with an outer ring and zero or more inner rings. The difference is that a ring can be a CircularString or CompoundCurve as well as a LineString.

Ab PostGIS 1.4 werden zusammengesetzte Kurven/CompoundCurve in einem Kurvenpolygon/CurvePolygon unterstützt.

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

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

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

4.1.2.4 MultiCurve

A MultiCurve is a collection of curves which can include LineStrings, CircularStrings or CompoundCurves.

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

4.1.2.5 MultiSurface

A MultiSurface is a collection of surfaces, which can be (linear) Polygons or CurvePolygons.

```
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 and WKB

The OGC SFA specification defines two formats for representing geometry values for external use: Well-Known Text (WKT) and Well-Known Binary (WKB). Both WKT and WKB include information about the type of the object and the coordinates which define it.

Well-Known Text (WKT) provides a standard textual representation of spatial data. Examples of WKT representations of spatial objects are:

- 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

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

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

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

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

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

- WKT: POINT(1 1)

  WKB: 01010000000000000000F03F000000000000F03
- WKT: LINESTRING (2 2, 9 9)

  WKB: 0102000000020000000000000000000040000000000000004000000000000022400000000000002240

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

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

For example, a statement to create and insert a spatial object from WKB is:

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

4.2 Geometry Data Type

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

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

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

The SFA standard specifies that spatial objects include a Spatial Reference System identifier (SRID). The SRID is required when creating spatial objects for insertion into the database (it may be defaulted to 0). See ST_SRID and Section 4.5

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

4.2.1 PostGIS EWKB and EWKT

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

Because of these limitations PostGIS defined extended EWKB and EWKT formats. They provide 3D (XYZ and XYM) and 4D (XYZM) coordinate support and include SRID information. Including all geometry information allows PostGIS to use EWKB as the format of record (e.g. in DUMP files).

EWKB and EWKT are used for the "canonical forms" of PostGIS data objects. For input, the canonical form for binary data is EWKB, and for text data either EWKB or EWKT is accepted. This allows geometry values to be created by casting a text value in either HEXEWKB or EWKT to a geometry value using `::geometry`. For output, the canonical form for binary is EWKB, and for text it is HEXEWKB (hex-encoded EWKB).

For example this statement creates a geometry by casting from an EWKT text value, and outputs it using the canonical form of HEXEWKB:

```
SELECT 'SRID=4;POINT(0 0)'::geometry;
  geometry
  ----------------------------------------------------
  0101000020040000000000000000000000000000000000000000
```

PostGIS EWKT output has a few differences to OGC WKT:

- For 3DZ geometries the Z qualifier is omitted:
  OGC: POINT Z (1 2 3)
  EWKT: POINT (1 2 3)
- For 3DM geometries the M qualifier is included:
  OGC: POINT M (1 2 3)
  EWKT: POINTM (1 2 3)

- For 4D geometries the ZM qualifier is omitted:
  OGC: POINT ZM (1 2 3 4)
  EWKT: POINT (1 2 3 4)

EWKT avoids over-specifying dimensionality and the inconsistencies that can occur with the OGC/ISO format, such as:

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

Caution
PostGIS extended formats are currently a superset of the OGC ones, so that every valid OGC WKB/WKT is also valid EWKB/EWKT. However, this might vary in the future, if the OGC extends a format in a way that conflicts with the PosGIS definition. Thus you SHOULD NOT rely on this compatibility!

Examples of the EWKT text representation of spatial objects are:

- POINT(0 0 0) -- XYZ
- SRID=32632;POINT(0 0) -- XY mit SRID
- POINTM(0 0 0) -- XYM
- POINT(0 0 0 0) -- XYZM
- SRID=4326;MULTIPOINTM(0 0 0,1 2 1) -- XYM mit SRID
- MULTILINESTRING((0 0 0,1 1 0,1 2 1),(2 3 1,3 2 1,5 4 1))
- POLYGON((0 0 0,4 0 0,4 4 0,0 4 0,0 0 0),(1 1 0,2 1 0,2 2 0,1 2 0,1 1 0))
- MULTIPOLYGON(((0 0 0,4 0 0,4 4 0,0 4 0,0 0 0),(1 1 0,2 1 0,2 2 0,1 2 0,1 1 0)),((-1 -1 0,-1 -2 0,-2 -2 0,-2 -1 0,-1 -1 0)))
- GEOMETRYCOLLECTIONM(POINTM(2 3 9), LINESTRINGM(2 3 4, 3 4 5))
- MULTICURVE((0 0, 5 5), CIRCULARSTRING(4 0, 4 4, 8 4))
- POLYHEDRALSURFACE(((0 0 0, 0 0 1, 0 1 1, 0 1 0, 0 0 0)), ((0 0 0, 0 1 0, 1 1 0, 1 0 0, 0 0 0)), ((0 0 0, 1 0 0, 1 0 1, 0 0 1, 0 0 0)), ((1 1 0, 1 1 1, 1 0 1, 1 0 0, 1 1 0)), ((0 1 0, 0 1 1, 1 1 1, 1 1 0, 0 1 0)), ((0 0 1, 1 0 1, 1 1 1, 0 1 1, 0 0 1)))
- TRIANGLE ((0 0, 0 10, 10 0, 0 0))
- TIN(((0 0 0, 0 0 1, 0 1 0, 0 0 0)), ((0 0 0, 0 1 0, 1 1 0, 0 0 0)))

Input and output using these formats is available using the following functions:

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

For example, a statement to create and insert a PostGIS spatial object using EWKT is:

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

4.3 Geography Data Type

The PostGIS `geography` data type provides native support for spatial features represented on "geographic" coordinates (sometimes called "geodetic" coordinates, or "lat/lon", or "lon/lat"). Geographic coordinates are spherical coordinates expressed in angular units (degrees).

The basis for the PostGIS geometry data type is a plane. The shortest path between two points on the plane is a straight line. That means functions on geometries (areas, distances, lengths, intersections, etc) are calculated using straight line vectors and cartesian mathematics. This makes them simpler to implement and faster to execute, but also makes them inaccurate for data on the spheroidal surface of the earth.

The PostGIS geography data type is based on a spherical model. The shortest path between two points on the sphere is a great circle arc. Functions on geographies (areas, distances, lengths, intersections, etc) are calculated using arcs on the sphere. By taking the spheroidal shape of the world into account, the functions provide more accurate results.

Because the underlying mathematics is more complicated, there are fewer functions defined for the geography type than for the geometry type. Over time, as new algorithms are added the capabilities of the geography type will expand. As a workaround one can convert back and forth between geometry and geography types.

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

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

4.3.1 Creating Geography Tables

You can create a table to store geography data using the CREATE TABLE SQL statement with a column of type `geography`. The following example creates a table with a geography column storing 2D LineStrings in the WGS84 geodetic coordinate system (SRID 4326):

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

The geography type supports two optional type modifiers:

- the spatial type modifier restricts the kind of shapes and dimensions allowed in the column. Values allowed for the spatial type are: POINT, LINESTRING, POLYGON, MULTIPOINT, MULTILINESTRING, MULTIPOLYGON, GEOMETRYCOLLECTION. The geography type does not support curves, TINS, or POLYHEDRALSURFACEs. The modifier supports coordinate dimensionality restrictions by adding suffixes: Z, M and ZM. For example, a modifier of 'LINESTRINGM' only allows linestrings with three dimensions, and treats the third dimension as a measure. Similarly, 'POINTZM' requires four dimensional (XYZM) data.
- the SRID modifier restricts the spatial reference system SRID to a particular number. If omitted, the SRID defaults to 4326 (WGS84 geodetic), and all calculations are performed using WGS84.

Examples of creating tables with geography columns:

- Create a table with 2D POINT geography with the default SRID 4326 (WGS84 long/lat):

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

- Create a table with 2D POINT geography in NAD83 longlat:

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

- Create a table with 3D (XYZ) POINTs and an explicit SRID of 4326:

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

- Create a table with 2D LINESTRING geography with the default SRID 4326:

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

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

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

Geography fields are registered in the `geography_columns` system view. You can query the `geography_columns` view and see that the table is listed:

```
SELECT * FROM geography_columns;
```

Creating a spatial index works the same as for geometry columns. PostGIS will note that the column type is GEOGRAPHY and create an appropriate sphere-based index instead of the usual planar index used for GEOMETRY.

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

4.3.2 Using Geography Tables

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

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

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

```
-- NAD 83 lon/lat
SELECT 'SRID=4269;POINT(-123 34)'::geography;
                    geography
----------------------------------------------------
 0101000020AD10000000000000C05EC00000000000004140
```

```
-- NAD27 lon/lat
SELECT 'SRID=4267;POINT(-123 34)'::geography;
                    geography
----------------------------------------------------
 0101000020AB10000000000000C05EC00000000000004140
```

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

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

Anfrage und Messfunktionen verwenden die Einheit Meter. Daher sollten Entfernungsparameter in Metern ausgedrückt werden und die Rückgabewerte sollten ebenfalls in Meter (oder Quadratmeter für Flächen) erwartet werden.

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

You can see the power of geography in action by calculating how close a plane flying a great circle route from Seattle to London (LINESTRING(-122.33 47.606, 0.0 51.5)) comes to Reykjavik (POINT(-21.96 64.15)) (map the route).

The geography type calculates the true shortest distance of 122.235 km over the sphere between Reykjavik and the great circle flight path between Seattle and London.

```
-- 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
```

The geometry type calculates a meaningless cartesian distance between Reykjavik and the straight line path from Seattle to London plotted on a flat map of the world. The nominal units of the result is "degrees", but the result doesn't correspond to any true angular difference between the points, so even calling them "degrees" is inaccurate.

```
-- 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 When to use the Geography data type

The geography data type allows you to store data in longitude/latitude coordinates, but at a cost: there are fewer functions defined on GEOGRAPHY than there are on GEOMETRY; those functions that are defined take more CPU time to execute.

The data type you choose should be determined by the expected working area of the application you are building. Will your data span the globe or a large continental area, or is it local to a state, county or municipality?

- Wenn sich Ihre Daten in einem kleinen Bereich befinden, werden Sie vermutlich eine passende Projektion wählen und den geometrischen Datentyp verwenden, da dies in Bezug auf die Rechenleistung und die verfügbare Funktionalität die bessere Lösung ist.
- Wenn Ihre Daten global sind oder einen ganzen Kontinent bedecken, ermöglicht der geographische Datentyp ein System aufzubauen, bei dem Sie sich nicht um Projektionsdetails kümmern müssen. Sie speichern die Daten als Länge und Breite und verwenden dann jene Funktionen, die für den geographischen Datentyp definiert sind.
- Wenn Sie keine Ahnung von Projektionen haben, sich nicht näher damit beschäftigen wollen und die Einschränkungen der verfügbaren Funktionalität für den geographischen Datentyp in Kauf nehmen können, ist es vermutlich einfacher für Sie, den geographischen anstatt des geometrischen Datentyps zu verwenden.

Für einen Vergleich, welche Funktionalität von Geography vs. Geometry unterstützt wird, siehe Section 12.11. Für eine kurze Liste mit der Beschreibung der geographischen Funktionen, siehe Section 12.4

4.3.4 Fortgeschrittene FAQ's zum geographischen Datentyp

1. *Werden die Berechnungen auf einer Kugel oder auf einem Rotationsellipsoid durchgeführt?*

   Standardmäßig werden alle Entfernungs- und Flächenberechnungen auf dem Referenzellipsoid ausgeführt. Das Ergebnis der Berechnung sollte in lokalen Gebieten gut mit dem planaren Ergebnis zusammenpassen - eine gut gewählte lokale

Projektion vorausgesetzt. Bei größeren Gebieten ist die Berechnung über das Referenzellipsoid genauer als eine Berechnung die auf der projizierten Ebene ausgeführt wird. Alle geographischen Funktionen verfügen über eine Option um die Berechnung auf einer Kugel durchzuführen. Dies erreicht man, indem der letzte boolesche Eingabewert auf 'FALSE' gesetzt wird. Dies beschleunigt die Berechnung einigermaßen, insbesondere wenn die Geometrie sehr einfach gestaltet ist.

2. *Wie schaut das mit der Datumsgrenze und den Polen aus?*

   Alle diese Berechnungen wissen weder über Datumsgrenzen noch über Pole Bescheid. Da es sich um sphärische Koordinaten handelt (Länge und Breite), unterscheidet sich eine Geometrie, die eine Datumsgrenze überschreitet vom Gesichtspunkt der Berechnung her nicht von irgendeiner anderen Geometrie.

3. *Wie lang kann ein Bogen sein, damit er noch verarbeitet werden kann?*

   Wir verwenden Großkreisbögen als "Interpolationslinie" zwischen zwei Punkten. Das bedeutet, dass es für den Join zwischen zwei Punkten zwei Möglichkeiten gibt, je nachdem, aus welcher Richtung man den Großkreis überquert. Unser gesamter Code setzt voraus, dass die Punkte von der "kürzeren" der beiden Strecken her durch den Großkreis verbunden werden. Als Konsequenz wird eine Geometrie, welche Bögen von mehr als 180 Grad aufweist nicht korrekt modelliert.

4. *Warum dauert es so lange, die Fläche von Europa / Russland / irgendeiner anderen großen geographischen Region zu berechnen?*

   Weil das Polygon so verdammt groß ist! Große Flächen sind aus zwei Gründen schlecht: ihre Begrenzung ist riesig, wodurch der Index dazu tendiert, das Geoobjekt herauszuholen, egal wie Sie die Anfrage ausführen; die Anzahl der Knoten ist riesig, und Tests (wie ST_Distance, ST_Contains) müssen alle Knoten zumindest einmal, manchmal sogar n-mal durchlaufen (wobei N die Anzahl der Knoten im beteiligten Geoobjekt bezeichnet). Wenn es sich um sehr große Polygone handelt, die Abfragen aber nur in kleinen Gebieten stattfinden, empfehlen wir wie beim geometrischen Datentyp, dass Sie die Geometrie in kleinere Stücke "denormalisieren". Dadurch kann der Index effiziente Unterabfragen auf Teile des Geoobjekts ausführen, da eine Abfrage nicht jedesmal das gesamte Geoobjekt herausholen muss. Konsultieren Sie dazu bitte die Dokumentation der FunktionST_Subdivide. Nur weil Sie ganz Europa in einem Polygon speichern *können* heißt das nicht, dass Sie dies auch tun *sollten*.

4.4 Geometrievalidierung

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

4.4.1 Simple Geometry

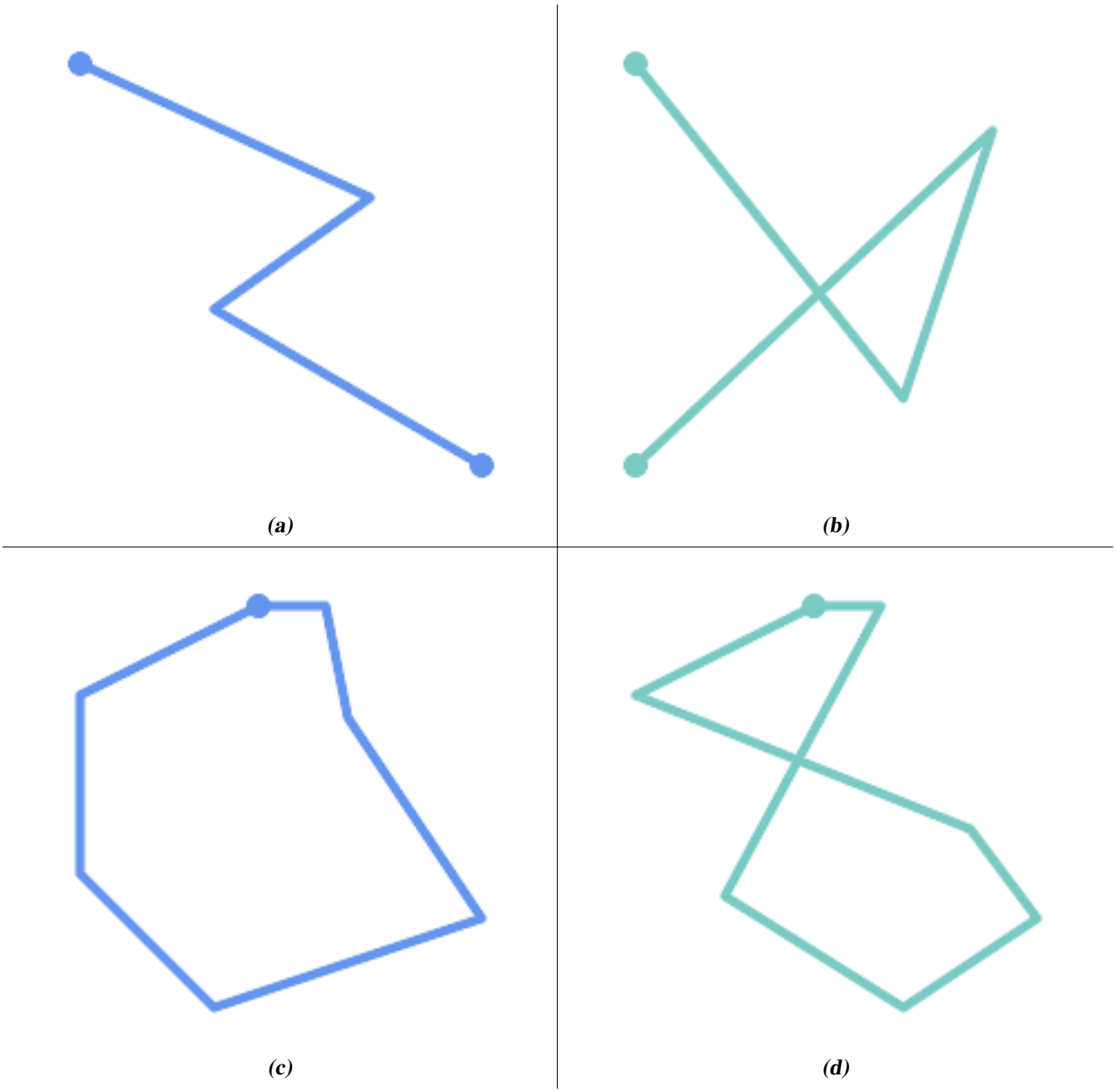
A *simple* geometry is one that has no anomalous geometric points, such as self intersection or self tangency.

A `POINT` is inherently *simple* as a 0-dimensional geometry object.

`MULTIPOINT`s sind *simple*, wenn sich keine zwei Koordinaten (`POINT`s) decken (keine identischen Koordinatenpaare aufweisen).

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

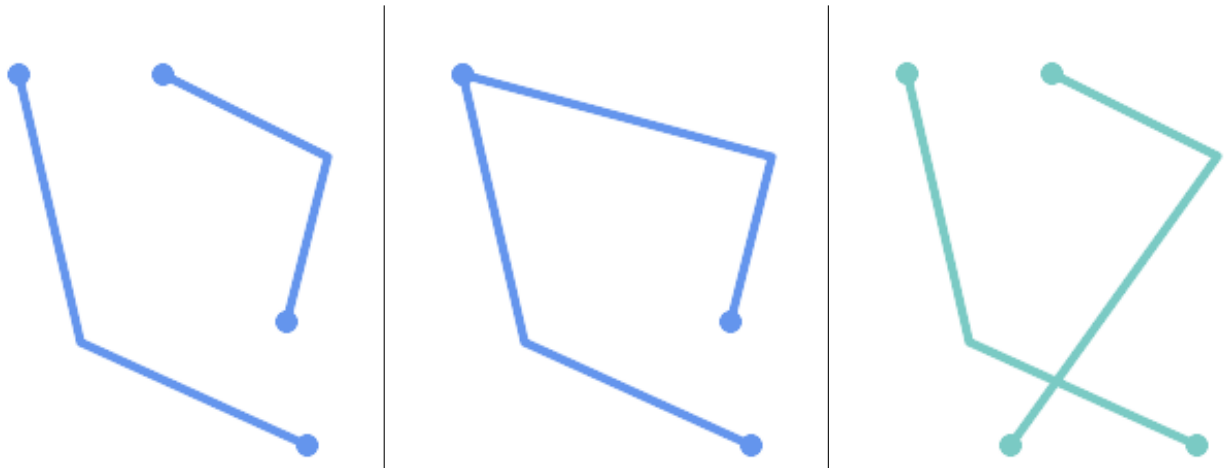
***(a)** and **(c)** are simple `LINESTRING`s. **(b)** and **(d)** are not simple. **(c)** is a closed Linear Ring.*



(a) ***(b)*** ***(c)*** ***(d)***

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

(e) *and* ***(f)*** *are simple* `MULTILINESTRING`*s.* ***(g)*** *is not simple.*



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

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

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

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

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

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

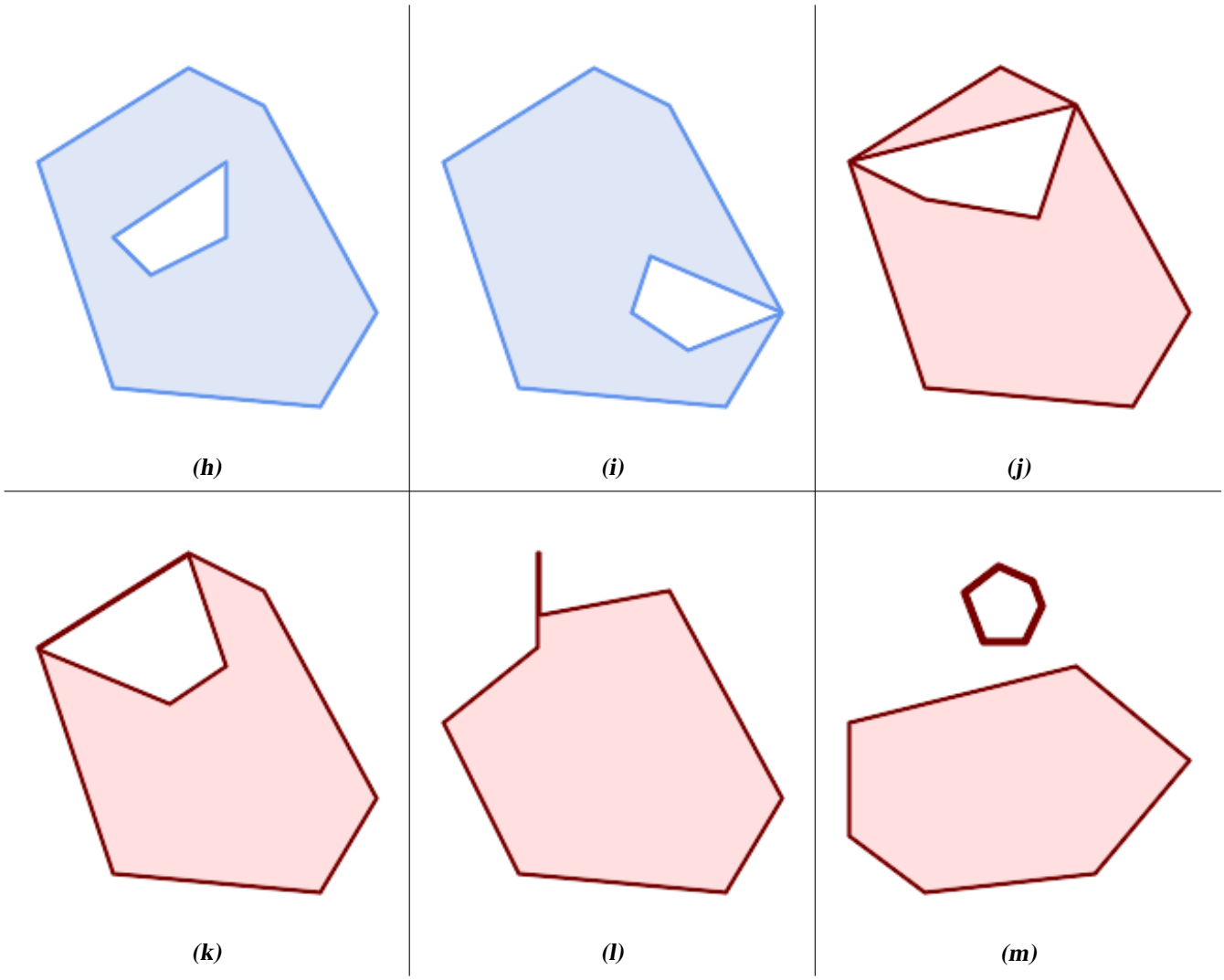
4.4.2 Valid Geometry

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

A POLYGON is *valid* if:

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

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



(h) ***(i)*** ***(j)***

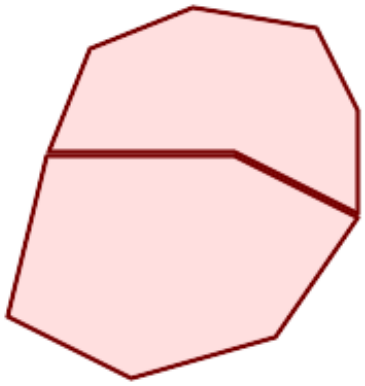
(k) ***(l)*** ***(m)***

A `MULTIPOLYGON` is *valid* if:

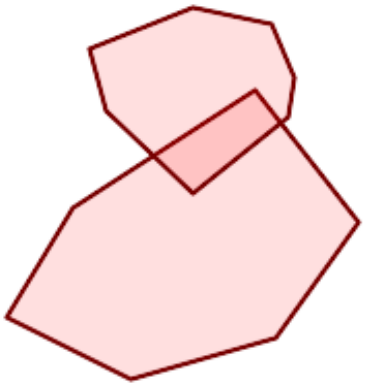
1. its element `POLYGON`s are valid
2. elements do not overlap (i.e. their interiors must not intersect)
3. elements touch only at points (i.e. not along a line)

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

(n)



(o)



(p)

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

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

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

 len_nonzero | len_zero | self_int
-------------+----------+----------
 t           | f        | t
```

POINT and MULTIPOINT geometries have no validity rules.

4.4.3 Managing Validity

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

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

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

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

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

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

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

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

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

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

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

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

4.5 Spatial Reference Systems

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

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

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

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

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

4.5.1 SPATIAL_REF_SYS Table

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

The spatial_ref_sys table definition is:

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

The columns are:

srid An integer code that uniquely identifies the Spatial Reference System (SRS) within the database.

auth_name The name of the standard or standards body that is being cited for this reference system. For example, "EPSG" is a valid auth_name.

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

srtext Die Well-Known-Text Darstellung des Koordinatenreferenzsystems. Ein Beispiel dazu:

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

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

proj4text PostGIS uses the PROJ library to provide coordinate transformation capabilities. The proj4text column contains the PROJ coordinate definition string for a particular SRID. For example:

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

For more information see the PROJ web site. The spatial_ref_sys.sql file contains both srtext and proj4text definitions for all EPSG projections.

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

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

4.5.2 User-Defined Spatial Reference Systems

The PostGIS `spatial_ref_sys` table contains over 3000 of the most common spatial reference system definitions that are handled by the PROJ projection library. But there are many coordinate systems that it does not contain. You can add SRS definitions to the table if you have the required information about the spatial reference system. Or, you can define your own custom spatial reference system if you are familiar with PROJ constructs. Keep in mind that most spatial reference systems are regional and have no meaning when used outside of the bounds they were intended for.

A resource for finding spatial reference systems not defined in the core set is http://spatialreference.org/

Some commonly used spatial reference systems are: 4326 - WGS 84 Long Lat, 4269 - NAD 83 Long Lat, 3395 - WGS 84 World Mercator, 2163 - US National Atlas Equal Area, and the 60 WGS84 UTM zones. UTM zones are one of the most ideal for measurement, but only cover 6-degree regions. (To determine which UTM zone to use for your area of interest, see the utmzone PostGIS plpgsql helper function.)

US states use State Plane spatial reference systems (meter or feet based) - usually one or 2 exists per state. Most of the meter-based ones are in the core set, but many of the feet-based ones or ESRI-created ones will need to be copied from spatialreference.org.

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

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

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

4.6 Spatial Tables

4.6.1 Erstellung einer räumlichen Tabelle

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

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

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

- the **spatial type modifier** restricts the kind of shapes and dimensions allowed in the column. The value can be any of the supported geometry subtypes (e.g. POINT, LINESTRING, POLYGON, MULTIPOINT, MULTILINESTRING, MULTIPOLYGON, GEOMETRYCOLLECTION, etc). The modifier supports coordinate dimensionality restrictions by adding suffixes: Z, M and ZM. For example, a modifier of 'LINESTRINGM' allows only linestrings with three dimensions, and treats the third dimension as a measure. Similarly, 'POINTZM' requires four dimensional (XYZM) data.
- the **SRID modifier** restricts the spatial reference system SRID to a particular number. If omitted, the SRID defaults to 0.

Examples of creating tables with geometry columns:

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

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

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

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

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

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

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

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

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

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

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

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

4.6.2 GEOMETRY_COLUMNS View

The OGC *Simple Features Specification for SQL* defines the `GEOMETRY_COLUMNS` metadata table to describe geometry table structure. In PostGIS `geometry_columns` is a view reading from database system catalog tables. This ensures that the spatial metadata information is always consistent with the currently defined tables and views. The view structure is:

```
\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)  |
```

The columns are:

f_table_catalog, f_table_schema, f_table_name The fully qualified name of the feature table containing the geometry column. There is no PostgreSQL analogue of "catalog" so that column is left blank. For "schema" the PostgreSQL schema name is used (`public` is the default).

f_geometry_column Der Name der Geometriespalte in der Feature-Tabelle.

coord_dimension The coordinate dimension (2, 3 or 4) of the column.

srid The ID of the spatial reference system used for the coordinate geometry in this table. It is a foreign key reference to the `spatial_ref_sys` table (see Section 4.5.1).

type Der Datentyp des Geoobjekts. Um die räumliche Spalte auf einen einzelnen Datentyp zu beschränken, benutzen Sie bitte: POINT, LINESTRING, POLYGON, MULTIPOINT, MULTILINESTRING, MULTIPOLYGON, GEOMETRYCOLLECTION oder die entsprechenden XYM Versionen POINTM, LINESTRINGM, POLYGONM, MULTIPOINTM, MULTILINESTRINGM, MULTIPOLYGONM und GEOMETRYCOLLECTIONM. Für uneinheitliche Kollektionen (gemischete Datentypen) können Sie den Datentyp "GEOMETRY" verwenden.

4.6.3 Manually Registering Geometry Columns

Zwei Fälle bei denen Sie dies benötigen könnten sind SQL-Views und Masseninserts. Beim Fall von Masseninserts können Sie die Registrierung in der Tabelle "geometry_columns" korrigieren, indem Sie auf die Spalte einen CONSTRAINT setzen oder ein "ALTER TABLE" durchführen. Falls Ihre Spalte Typmod basiert ist, geschieht die Registrierung beim Erstellungsprozess auf korrekte Weise, so dass Sie hier nichts tun müssen. Auch Views, bei denen keine räumliche Funktion auf die Geometrie angewendet wird, werden auf gleiche Weise wie die Geometrie der zugrunde liegenden Tabelle registriert.

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

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

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

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

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

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

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

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

Obwohl die alte auf CONSTRAINTs basierte Methode immer noch unterstützt wird, wird eine auf Constraints basierende Geometriespalte, die direkt in einem View verwendet wird, nicht korrekt in geometry_columns registriert. Eine Typmod basierte wird korrekt registriert. Im folgenden Beispiel definieren wir eine Spalte mit Typmod und eine andere mit 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);
```

In psql:

```
\d pois_ny;
```

Wir sehen, das diese Spalten unterschiedlich definiert sind -- eine mittels Typmodifizierer, eine nutzt einen Constraint

```
Table "public.pois_ny"
  Column   |         Type          |                      Modifiers

-----------+-----------------------+-----------------------------------------------------
 gid       | integer               | not null default nextval('pois_ny_gid_seq'::regclass)
 poi_name  | text                  |
 cat       | character varying(20) |
 geom      | geometry(Point,4326)  |
 geom_2160 | geometry              |
Indexes:
    "pois_ny_pkey" PRIMARY KEY, btree (gid)
Check constraints:
    "enforce_dims_geom_2160" CHECK (st_ndims(geom_2160) = 2)
    "enforce_geotype_geom_2160" CHECK (geometrytype(geom_2160) = 'POINT'::text
        OR geom_2160 IS NULL)
    "enforce_srid_geom_2160" CHECK (st_srid(geom_2160) = 2160)
```

Beide registrieren sich korrekt in "geometry_columns"

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

```
f_table_name | f_geometry_column | srid | type
-------------+-------------------+------+-------
pois_ny      | geom              | 4326 | POINT
pois_ny      | geom_2160         | 2160 | POINT
```

Jedoch -- wenn wir einen View auf die folgende Weise erstellen

```
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';
```

Die Typmod basierte geometrische Spalte eines View registriert sich korrekt, die auf Constraint basierende nicht.

```
f_table_name   | f_geometry_column | srid |   type
-----------------+-------------------+------+----------
 vw_pois_ny_parks | geom              | 4326 | POINT
 vw_pois_ny_parks | geom_2160         |    0 | GEOMETRY
```

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

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

```
        FROM geometry_columns
        WHERE f_table_name = 'vw_pois_ny_parks';
```

```
f_table_name   | f_geometry_column | srid | type
-----------------+-------------------+------+-------
 vw_pois_ny_parks | geom              | 4326 | POINT
 vw_pois_ny_parks | geom_2160         | 2160 | POINT
```

4.7 Loading Spatial Data

Once you have created a spatial table, you are ready to upload spatial data to the database. There are two built-in ways to get spatial data into a PostGIS/PostgreSQL database: using formatted SQL statements or using the Shapefile loader.

4.7.1 Using SQL to Load Data

If spatial data can be converted to a text representation (as either WKT or WKB), then using SQL might be the easiest way to get data into PostGIS. Data can be bulk-loaded into PostGIS/PostgreSQL by loading a text file of SQL `INSERT` statements using the `psql` SQL utility.

A SQL load file (`roads.sql` for example) might look like this:

```
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;
```

The SQL file can be loaded into PostgreSQL using `psql`:

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

4.7.2 Using the Shapefile Loader

The `shp2pgsql` data loader converts Shapefiles into SQL suitable for insertion into a PostGIS/PostgreSQL database either in geometry or geography format. The loader has several operating modes selected by command line flags.

There is also a `shp2pgsql-gui` graphical interface with most of the options as the command-line loader. This may be easier to use for one-off non-scripted loading or if you are new to PostGIS. It can also be configured as a plugin to PgAdminIII.

(c|a|d|p) Dies sind sich gegenseitig ausschließende Optionen:

-c Creates a new table and populates it from the Shapefile. *This is the default mode.*

-a Appends data from the Shapefile into the database table. Note that to use this option to load multiple files, the files must have the same attributes and same data types.

-d Drops the database table before creating a new table with the data in the Shapefile.

-p Erzeugt nur den SQL-Code zur Erstellung der Tabelle, ohne irgendwelche Daten hinzuzufügen. Kann verwendet werden, um die Erstellung und das Laden einer Tabelle vollständig zu trennen.

-? Zeigt die Hilfe an.

-D Verwendung des PostgreSQL "dump" Formats für die Datenausgabe. Kann mit -a, -c und -d kombiniert werden. Ist wesentlich schneller als das standardmäßige SQL "insert" Format. Verwenden Sie diese Option wenn Sie sehr große Datensätze haben.

-s [<FROM_SRID>:]<SRID> Erstellt und befüllt die Geometrietabelle mit der angegebenen SRID. Optional kann für das Shapefile eine FROM_SRID angegeben werden, worauf dann die Geometrie in die Ziel-SRID projiziert wird.

-k Erhält die Groß- und Kleinschreibung (Spalte, Schema und Attribute). Beachten Sie bitte, dass die Attributnamen in Shapedateien immer Großbuchstaben haben.

-i Wandeln Sie alle Ganzzahlen in standard 32-bit Integer um, erzeugen Sie keine 64-bit BigInteger, auch nicht dann wenn der DBF-Header dies unterstellt.

-I Einen GIST Index auf die Geometriespalte anlegen.

-m -m `a_file_name` bestimmt eine Datei, in welcher die Abbildungen der (langen) Spaltennamen in die 10 Zeichen langen DBF Spaltennamen festgelegt sind. Der Inhalt der Datei besteht aus einer oder mehreren Zeilen die jeweils zwei, durch Leerzeichen getrennte Namen enthalten, aber weder vorne noch hinten mit Leerzeichen versehen werden dürfen. Zum Beispiel:

```
COLUMNNAME DBFFIELD1
AVERYLONGCOLUMNNAME DBFFIELD2
```

-S Erzeugt eine Einzel- anstatt einer Mehrfachgeometrie. Ist nur erfolgversprechend, wenn die Geometrie auch tatsächlich eine Einzelgeometrie ist (insbesondere gilt das für ein Mehrfachpolygon/MULTIPOLYGON, dass nur aus einer einzelnen Begrenzung besteht, oder für einen Mehrfachpunkt/MULTIPOINT, der nur einen einzigen Knoten aufweist).

-t <dimensionality> Zwingt die Ausgabegeometrie eine bestimmte Dimension anzunehmen. Sie können die folgenden Zeichenfolgen verwenden, um die Dimensionalität anzugeben: 2D, 3DZ, 3DM, 4D.

Wenn die Eingabe weniger Dimensionen aufweist als angegeben, dann werden diese Dimensionen bei der Ausgabe mit Nullen gefüllt. Wenn die Eingabe mehr Dimensionen als angegeben aufweist werden diese abgestreift.

-w Ausgabe im Format WKT anstatt WKB. Beachten Sie bitte, dass es hierbei zu Koordinatenverschiebungen infolge von Genauigkeitsverlusten kommen kann.

-e Jede Anweisung einzeln und nicht in einer Transaktion ausführen. Dies erlaubt den Großteil auch dann zu laden, also die guten Daten, wenn eine Geometrie dabei ist die Fehler verursacht. Beachten Sie bitte das dies nicht gemeinsam mit der -D Flag angegeben werden kann, da das "dump" Format immer eine Transaktion verwendet.

-W <encoding> Gibt die Codierung der Eingabedaten (dbf-Datei) an. Wird die Option verwendet, so werden alle Attribute der dbf-Datei von der angegebenen Codierung nach UTF8 konvertiert. Die resultierende SQL-Ausgabe enthält dann den Befehl `SET CLIENT_ENCODING to UTF8`, damit das Back-end wiederum die Möglichkeit hat, von UTF8 in die, für die interne Nutzung konfigurierte Datenbankcodierung zu decodieren.

-N <policy> Umgang mit NULL-Geometrien (insert*, skip, abort)

-n -n Es wird nur die *.dbf-Datei importiert. Wenn das Shapefile nicht Ihren Daten entspricht, wird automatisch auf diesen Modus geschaltet und nur die *.dbf-Datei geladen. Daher müssen Sie diese Flag nur dann setzen, wenn sie einen vollständigen Shapefile-Satz haben und lediglich die Attributdaten, und nicht die Geometrie, laden wollen.

-G Verwendung des geographischen Datentyps in WGS84 (SRID=4326), anstelle des geometrischen Datentyps (benötigt Längen- und Breitenangaben).

-T <tablespace> Den Tablespace für die neue Tabelle festlegen. Solange der -X Parameter nicht angegeben wird, benutzen die Indizes weiterhin den standardmäßig festgelegten Tablespace. Die PostgreSQL Dokumentation beinhaltet eine gute Beschreibung, wann es sinnvoll ist, eigene Tablespaces zu verwenden.

-X <tablespace> Den Tablespace bestimmen, in dem die neuen Tabellenindizes angelegt werden sollen. Gilt für den Primärschlüsselindex und wenn "-l" verwendet wird, auch für den räumlichen GIST-Index.

-Z When used, this flag will prevent the generation of `ANALYZE` statements. Without the -Z flag (default behavior), the `ANALYZE` statements will be generated.

An example session using the loader to create an input file and loading it might look like this:

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

A conversion and load can be done in one step using UNIX pipes:

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

4.8 Extracting Spatial Data

Spatial data can be extracted from the database using either SQL or the Shapefile dumper. The section on SQL presents some of the functions available to do comparisons and queries on spatial tables.

4.8.1 Using SQL to Extract Data

The most straightforward way of extracting spatial data out of the database is to use a SQL `SELECT` query to define the data set to be extracted and dump the resulting columns into a parsable text file:

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

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

There will be times when some kind of restriction is necessary to cut down the number of records returned. In the case of attribute-based restrictions, use the same SQL syntax as used with a non-spatial table. In the case of spatial restrictions, the following functions are useful:

ST_Intersects Diese Funktion bestimmt ob sich zwei geometrische Objekte einen gemeinsamen Raum teilen

= Überprüft, ob zwei Geoobjekte geometrisch ident sind. Zum Beispiel, ob 'POLYGON((0 0,1 1,1 0,0 0))' ident mit 'POLYGON((0 0,1 1,1 0,0 0))' ist (ist es).

Außerdem können Sie diese Operatoren in Anfragen verwenden. Beachten Sie bitte, wenn Sie eine Geometrie oder eine Box auf der SQL-Befehlszeile eingeben, dass Sie die Zeichensatzdarstellung explizit in eine Geometrie umwandeln müssen. 312 ist ein fiktives Koordinatenreferenzsystem das zu unseren Daten passt. Also, zum Beispiel:

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

Die obere Abfrage würde einen einzelnen Datensatz aus der Tabelle "ROADS_GEOM" zurückgeben, in dem die Geometrie gleich dem angegebenen Wert ist.

Überprüfung ob einige der Strassen in die Polygonfläche hineinreichen:

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

Die häufigsten räumlichen Abfragen werden vermutlich in einem bestimmten Ausschnitt ausgeführt. Insbesondere von Client-Software, wie Datenbrowsern und Kartendiensten, die auf diese Weise die Daten für die Darstellung eines "Kartenausschnitts" erfassen.

Der Operator "&&" kann entweder mit einer BOX3D oder mit einer Geometrie verwendet werden. Allerdings wird auch bei einer Geometrie nur das Umgebungsrechteck für den Vergleich herangezogen.

Die Abfrage zur Verwendung des "BOX3D" Objekts für einen solchen Ausschnitt sieht folgendermaßen aus:

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

Achten Sie auf die Verwendung von SRID=312, welche die Projektion Einhüllenden/Enveloppe bestimmt.

4.8.2 Using the Shapefile Dumper

The `pgsql2shp` table dumper connects to the database and converts a table (possibly defined by a query) into a shape file. The basic syntax is:

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

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

Optionen auf der Befehlszeile:

-f <filename> Ausgabe in eine bestimmte Datei.

-h <host> Der Datenbankserver, mit dem eine Verbindung aufgebaut werden soll.

-p <port> Der Port über den der Verbindungsaufbau mit dem Datenbank Server hergestellt werden soll.

-P <password> Das Passwort, das zum Verbindungsaufbau mit der Datenbank verwendet werden soll.

-u <user> Das Benutzername, der zum Verbindungsaufbau mit der Datenbank verwendet werden soll.

-g <geometry column> Bei Tabellen mit mehreren Geometriespalten jene Geometriespalte, die ins Shapefile geschrieben werden soll.

-b Die Verwendung eines binären Cursors macht die Berechnung schneller; funktioniert aber nur, wenn alle nicht-geometrischen Attribute in den Datentyp "text" umgewandelt werden können.

-r RAW-Modus. Das Attribut `gid` wird nicht verworfen und Spaltennamen werden nicht maskiert.

-m `filename` Bildet die Identifikatoren in Namen mit 10 Zeichen ab. Der Inhalt der Datei besteht aus Zeilen von jeweils zwei durch Leerzeichen getrennten Symbolen, jedoch ohne vor- oder nachgestellte Leerzeichen: VERYLONGSYMBOL SHORTONE ANOTHERVERYLONGSYMBOL SHORTER etc.

4.9 Spatial Indexes

Spatial indexes make using a spatial database for large data sets possible. Without indexing, a search for features requires a sequential scan of every record in the database. Indexing speeds up searching by organizing the data into a structure which can be quickly traversed to find matching records.

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

- **GiST (Generalized Search Tree)** indexes break up data into "things to one side", "things which overlap", "things which are inside" and can be used on a wide range of data-types, including GIS data. PostGIS uses an R-Tree index implemented on top of GiST to index spatial data. GiST is the most commonly-used and versatile spatial index method, and offers very good query performance.
- **BRIN (Block Range Index)** indexes operate by summarizing the spatial extent of ranges of table records. Search is done via a scan of the ranges. BRIN is only appropriate for use for some kinds of data (spatially sorted, with infrequent or no update). But it provides much faster index create time, and much smaller index size.
- **SP-GiST (Space-Partitioned Generalized Search Tree)** is a generic index method that supports partitioned search trees such as quad-trees, k-d trees, and radix trees (tries).

Spatial indexes store only the bounding box of geometries. Spatial queries use the index as a **primary filter** to quickly determine a set of geometries potentially matching the query condition. Most spatial queries require a **secondary filter** that uses a spatial predicate function to test a more specific spatial condition. For more information on queying with spatial predicates see Section 5.2.

See also the PostGIS Workshop section on spatial indexes, and the PostgreSQL manual.

4.9.1 GiST-Indizes

GiST stands for "Generalized Search Tree" and is a generic form of indexing for multi-dimensional data. PostGIS uses an R-Tree index implemented on top of GiST to index spatial data. GiST is the most commonly-used and versatile spatial index method, and offers very good query performance. Other implementations of GiST are used to speed up searches on all kinds of irregular data structures (integer arrays, spectral data, etc) which are not amenable to normal B-Tree indexing. For more information see the PostgreSQL manual.

Once a spatial data table exceeds a few thousand rows, you will want to build an index to speed up spatial searches of the data (unless all your searches are based on attributes, in which case you'll want to build a normal index on the attribute fields).

Die Syntax, mit der ein GIST-Index auf eine Geometriespalte gelegt wird, lautet:

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

Die obere Syntax erzeugt immer einen 2D-Index. Um einen n-dimensionalen Index für den geometrischen Datentyp zu erhalten, können Sie die folgende Syntax verwenden:

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

Die Erstellung eines räumlichen Indizes ist eine rechenintensive Aufgabe. Während der Erstellung wird auch der Schreibzugriff auf die Tabelle blockiert. Bei produktiven Systemen empfiehlt sich daher die langsamere Option CONCURRENTLY:

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

Nachdem ein Index aufgebaut wurde sollte PostgreSQL gezwungen werden die Tabellenstatistik zu sammeln, da diese zur Optmierung der Auswertungspläne verwendet wird:

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

4.9.2 BRIN Indizes

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

Once a spatial table exceeds a few thousand rows, you will want to build an index to speed up spatial searches of the data. GiST indexes are very performant as long as their size doesn't exceed the amount of RAM available for the database, and as long as you can afford the index storage size, and the cost of index update on write. Otherwise, for very large tables BRIN index can be considered as an alternative.

A BRIN index stores the bounding box enclosing all the geometries contained in the rows in a contiguous set of table blocks, called a *block range*. When executing a query using the index the block ranges are scanned to find the ones that intersect the query extent. This is efficient only if the data is physically ordered so that the bounding boxes for block ranges have minimal overlap (and ideally are mutually exclusive). The resulting index is very small in size, but is typically less performant for read than a GiST index over the same data.

Building a BRIN index is much less CPU-intensive than building a GiST index. It's common to find that a BRIN index is ten times faster to build than a GiST index over the same data. And because a BRIN index stores only one bounding box for each range of table blocks, it's common to use up to a thousand times less disk space than a GiST index.

You can choose the number of blocks to summarize in a range. If you decrease this number, the index will be bigger but will probably provide better performance.

For BRIN to be effective, the table data should be stored in a physical order which minimizes the amount of block extent overlap. It may be that the data is already sorted appropriately (for instance, if it is loaded from another dataset that is already sorted in spatial order). Otherwise, this can be accomplished by sorting the data by a one-dimensional spatial key. One way to do this is to create a new table sorted by the geometry values (which in recent PostGIS versions uses an efficient Hilbert curve ordering):

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

Alternatively, data can be sorted in-place by using a GeoHash as a (temporary) index, and clustering on that index:

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

The syntax for building a BRIN index on a `geometry` column is:

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

The above syntax builds a 2D index. To build a 3D-dimensional index, use this syntax:

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

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

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

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

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

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

The `geography` datatype is supported for BRIN indexing. The syntax for building a BRIN index on a geography column is:

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

The above syntax builds a 2D-index for geospatial objects on the spheroid.

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

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

To summarize using BRIN for spatial data:

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

4.9.3 SP-GiST Indizes

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

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

Sobald eine Geodatentabelle einige tausend Zeilen überschreitet, kann es sinnvoll sein einen SP-GIST Index zu erzeugen, um die räumlichen Abfragen auf die Daten zu beschleunigen. Die Syntax zur Erstellung eines SP-GIST Index auf eine "Geometriespalte" lautet:

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

Die obere Syntax erzeugt einen 2D-Index. Ein 3-dimensionaler Index für den geometrischen Datentyp können Sie mit der 3D Operatorklasse erstellen:

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

Die Erstellung eines räumlichen Indizes ist eine rechenintensive Aufgabe. Während der Erstellung wird auch der Schreibzugriff auf die Tabelle blockiert. Bei produktiven Systemen empfiehlt sich daher die langsamere Option CONCURRENTLY:

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

Nachdem ein Index aufgebaut wurde sollte PostgreSQL gezwungen werden die Tabellenstatistik zu sammeln, da diese zur Optmierung der Auswertungspläne verwendet wird:

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

Ein SP-GiST Index kann Abfragen mit folgenden Operatoren beschleunigen:

- <<, &<, &>, >>, <<|, &<|, |&>, |>>, &&, @>, <@, and ~=, für 2-dimensionale Iindices,
- &/&, ~==, @>>, and <<@, für 3-dimensionale Indices.

kNN Suche wird zurzeit nicht unterstützt.

4.9.4 Tuning Index Usage

Ordinarily, indexes invisibly speed up data access: once an index is built, the PostgreSQL query planner automatically decides when to use it to improve query performance. But there are some situations where the planner does not choose to use existing indexes, so queries end up using slow sequential scans instead of a spatial index.

If you find your spatial indexes are not being used, there are a few things you can do:

- Examine the query plan and check your query actually computes the thing you need. An erroneous JOIN, either forgotten or to the wrong table, can unexpectedly retrieve table records multiple times. To get the query plan, execute with `EXPLAIN` in front of the query.
- Make sure statistics are gathered about the number and distributions of values in a table, to provide the query planner with better information to make decisions around index usage. **VACUUM ANALYZE** will compute both.

  You should regularly vacuum your databases anyways. Many PostgreSQL DBAs run **VACUUM** as an off-peak cron job on a regular basis.
- If vacuuming does not help, you can temporarily force the planner to use the index information by using the command **SET ENABLE_SEQSCAN TO OFF;**. This way you can check whether the planner is at all able to generate an index-accelerated query plan for your query. You should only use this command for debugging; generally speaking, the planner knows better than you do about when to use indexes. Once you have run your query, do not forget to run **SET ENABLE_SEQSCAN TO ON;** so that the planner will operate normally for other queries.
- If **SET ENABLE_SEQSCAN TO OFF;** helps your query to run faster, your Postgres is likely not tuned for your hardware. If you find the planner wrong about the cost of sequential versus index scans try reducing the value of `RANDOM_PAGE_COST` in `postgresql.conf`, or use **SET RANDOM_PAGE_COST TO 1.1;**. The default value for `RANDOM_PAGE_COST` is 4.0. Try setting it to 1.1 (for SSD) or 2.0 (for fast magnetic disks). Decreasing the value makes the planner more likely to use index scans.
- If **SET ENABLE_SEQSCAN TO OFF;** does not help your query, the query may be using a SQL construct that the Postgres planner is not yet able to optimize. It may be possible to rewrite the query in a way that the planner is able to handle. For example, a subquery with an inline SELECT may not produce an efficient plan, but could possibly be rewritten using a LATERAL JOIN.

For more information see the Postgres manual section on Query Planning.

Chapter 5

Räumliche Abfrage

Der Sinn von räumlichen Datenbanken liegt darin Abfragen in der Datenbank ausführen zu können, die normalerweise Desktop-GIS-Funktionionalität verlangen würden. Um PostGIS effektiv verwenden zu können muss man die verfügbaren räumlichen Funktionen kennen, wissen wie sie in Abfragen verwendet werden und sicherstellen, dass für gute Performanz die passenden Indizes vorhanden sind.

5.1 Räumliche Beziehungen feststellen

Räumliche Beziehungen geben an wie zwei Geometrien miteinander interagieren. Sie sind die fundamentale Fähigkeit zum Abfragen von Geometrie.

5.1.1 Dimensionally Extended 9-Intersection Model

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

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

Boundary

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

Interior

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

Exterior

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

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

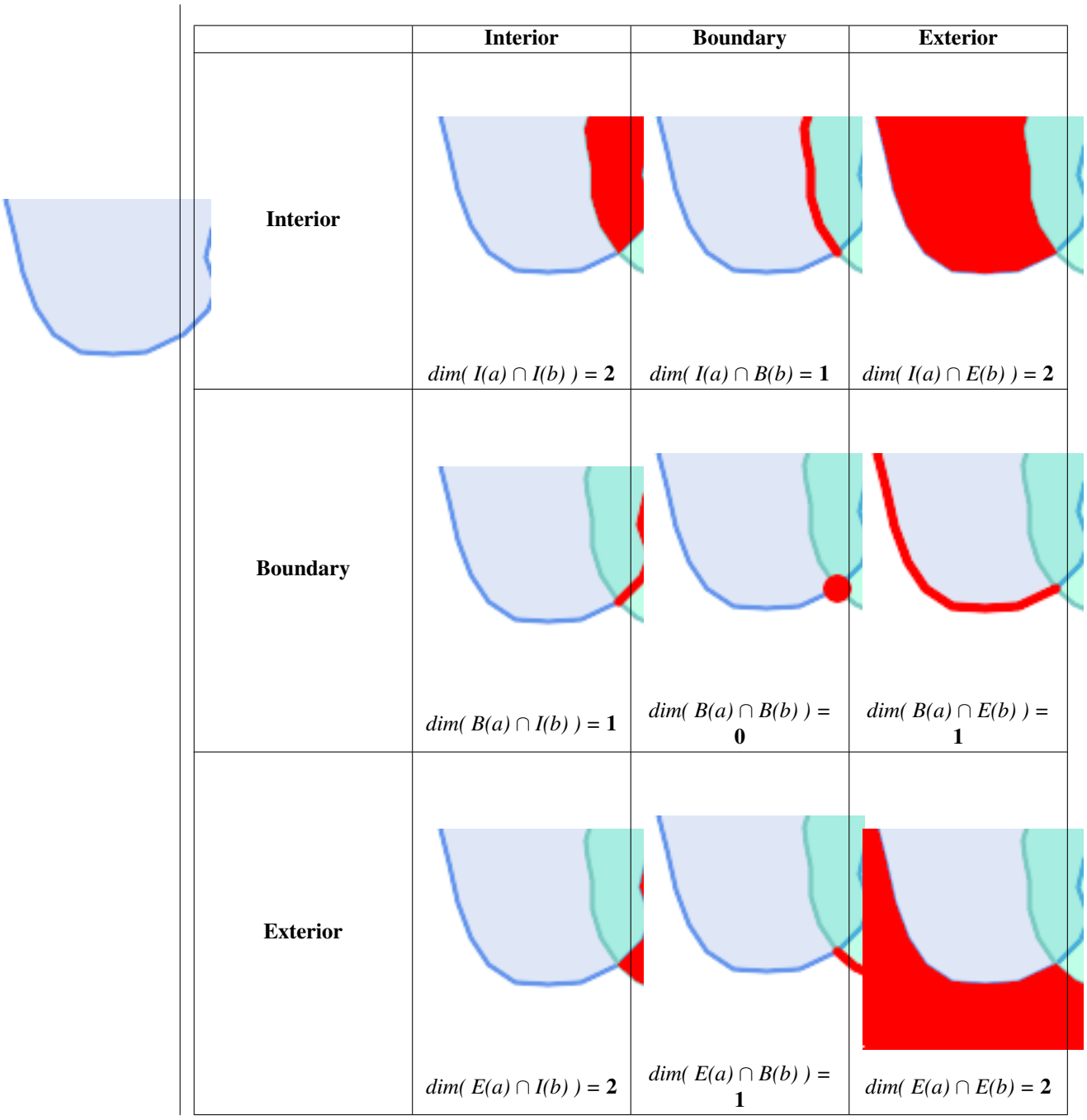
For a geometry *g* the *Interior*, *Boundary*, and *Exterior* are denoted using the notation *I(g)*, *B(g)*, and *E(g)*. Also, *dim(s)* denotes the dimension of a set *s* with the domain of `{0,1,2,F}`:

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

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

	Interior	**Boundary**	**Exterior**
Interior	*dim(I(a) ∩ I(b))*	*dim(I(a) ∩ B(b))*	*dim(I(a) ∩ E(b))*
Boundary	*dim(B(a) ∩ I(b))*	*dim(B(a) ∩ B(b))*	*dim(B(a) ∩ E(b))*
Exterior	*dim(E(a) ∩ I(b))*	*dim(E(a) ∩ B(b))*	*dim(E(a) ∩ E(b))*

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



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

For more information, refer to:

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

5.1.2 Named Spatial Relationships

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

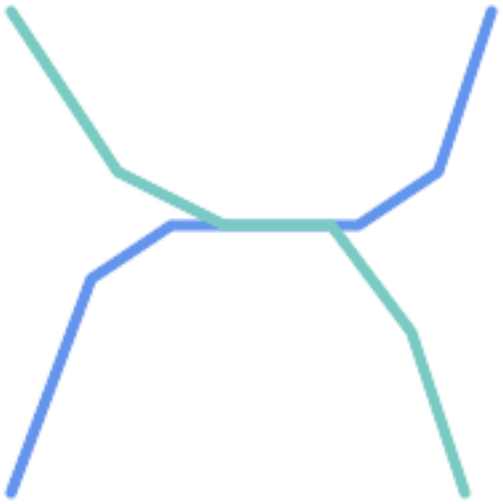
Spatial predicates are usually used as conditions in SQL `WHERE` or `JOIN` clauses. The named spatial predicates automatically use a spatial index if one is available, so there is no need to use the bounding box operator `&&` as well. For example:

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

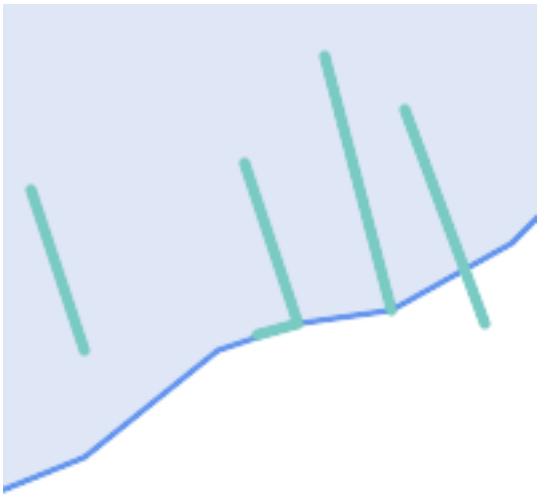
For more details and illustrations, see the PostGIS Workshop.

5.1.3 General Spatial Relationships

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



For example, consider a linear dataset representing a road network. It may be required to identify all road segments that cross each other, not at a point, but in a line (perhaps to validate some business rule). In this case ST_Crosses does not provide the necessary spatial filter, since for linear features it returns `true` only where they cross at a point.
A two-step solution would be to first compute the actual intersection (ST_Intersection) of pairs of road lines that spatially intersect (ST_Intersects), and then check if the intersection's ST_GeometryType is '`LINESTRING`' (properly dealing with cases that return `GEOMETRYCOLLECTION`s of `[MULTI]POINT`s, `[MULTI]LINESTRING`s, etc.).
Clearly, a simpler and faster solution is desirable.



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

- ST_Contains(lake, wharf) = TRUE
- ST_ContainsProperly(lake, wharf) = FALSE
- ST_GeometryType(ST_Intersection(wharf, lake)) = 'LINESTRING'
- ST_NumGeometries(ST_Multi(ST_Intersection(ST_Boundary(wharf), ST_Boundary(lake)))) = 1

  ... but needless to say, this is quite complicated.

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

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

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

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

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

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

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

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

5.2 Using Spatial Indexes

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

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

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

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

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

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

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

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

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

For more information and examples see the PostGIS Workshop.

5.3 Examples of Spatial SQL

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

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

The definition of the `bc_municipality` table is:

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

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

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

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

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

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

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

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

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

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

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

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

   name           | hectares
   ---------------+-----------------
   TUMBLER RIDGE  | 155020.02556131
   ```

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

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

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

   ```
   SELECT
     m.name,
     sum(ST_Length(r.geom))/1000 as roads_km
   FROM bc_roads AS r
   JOIN bc_municipality AS m
   ```

```
  ON ST_Contains(m.geom, r.geom)
GROUP BY m.name
ORDER BY roads_km;

name                         | roads_km
-----------------------------+------------------
SURREY                       | 1539.47553551242
VANCOUVER                    | 1450.33093486576
LANGLEY DISTRICT             | 833.793392535662
BURNABY                      | 773.769091404338
PRINCE GEORGE                | 694.37554369147
...
```

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

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

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

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

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

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

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

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

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

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

Chapter 6

Performance Tipps

6.1 Kleine Tabellen mit großen Geometrien

6.1.1 Problembeschreibung

Aktuelle PostgreSQL Versionen (inklusive 9.6) haben eine Schwäche des Optimizers in Bezug auf TOAST Tabellen. TOAST Tabellen bieten eine Art "Erweiterungsraum", der benutzt wird um große Werte (im Sinne der Datengröße), welche nicht in die üblichen Datenspeicherseiten passen (wie lange Texte, Bilder oder eine komplexe Geometrie mit vielen Stützpunkten) auszulagern, siehe the PostgreSQL Documentation for TOAST für mehr Information).

Das Problem tritt bei Tabellen mit relativ großen Geometrien, aber wenigen Zeilen auf (z.B. eine Tabelle welche die europäischen Ländergrenzen in hoher Auflösung beinhaltet). Dann ist die Tabelle selbst klein, aber sie benützt eine Menge an TOAST Speicherplatz. In unserem Beispiel hat die Tabelle um die 80 Zeilen und nutzt dafür nur 3 Speicherseiten, während die TOAST Tabelle 8225 Speicherseiten benützt.

Stellen Sie sich nun eine Abfrage vor, die den geometrischen Operator && verwendet, um ein Umgebungsrechteck mit nur wenigen Zeilen zu ermitteln. Der Abfrageoptimierer stellt fest, dass die Tabelle nur 3 Speicherseiten und 80 Zeilen aufweist. Er nimmt an, das ein sequentieller Scan bei einer derart kleinen Tabelle wesentlich schneller abläuft als die Verwendung eines Indizes. Und so entscheidet er den GIST Index zu ignorieren. Normalerweise stimmt diese Annahme. Aber in unserem Fall, muss der && Operator die gesamte Geometrie von der Festplatte lesen um den BoundingBox-Vergleich durchführen zu können, wodurch auch alle TOAST-Speicherseiten gelesen werden.

Um zu sehen, ob dieses Problem auftritt, können Sie den "EXPLAIN ANALYZE" Befehl von PostgreSQL anwenden. Mehr Information und die technischen Feinheiten entnehmen Sie bitte dem Thread auf der Postgres Performance Mailing List: http://archives.pos
performance/2005-02/msg00030.php

und einem neueren Thread über PostGIS https://lists.osgeo.org/pipermail/postgis-devel/2017-June/026209.html

6.1.2 Umgehungslösung

Die PostgreSQL Entwickler versuchen das Problem zu lösen, indem sie die Abschätzung der Abfragen TOAST-gewahr machen. Zur Überbrückung zwei Workarounds:

Der erste Workaround besteht darin den Query Planer zu zwingen, den Index zu nutzen. Setzen Sie "SET enable_seqscan TO off;" am Server bevor Sie die Abfrage ausführen. Dies zwingt den Query Planer grundsätzlich dazu sequentielle Scans, wann immer möglich, zu vermeiden. Womit der GIST Index wie üblich verwendet wird. Aber dieser Parameter muss bei jeder Verbindung neu gesetzt werden, und er verursacht das der Query Planer Fehleinschätzungen in anderen Fällen macht. Daher sollte "SET enable_seqscan TO on;" nach der Abfrage ausgeführt werden.

Der zweite Workaround besteht darin, den sequentiellen Scan so schnell zu machen wie der Query Planer annimmt. Dies kann durch eine zusätzliche Spalte, welche die BBOX "zwischenspeichert" und über die abgefragt wird, erreicht werden. In Unserem Beispiel sehen die Befehle dazu folgendermaßen aus:

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

Nun ändern Sie bitte Ihre Abfrage so, das der && Operator gegen die bbox anstelle der geom_column benutzt wird:

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

Selbstverständlich muss man die BBOX synchron halten. Die transparenteste Möglichkeit dies zu erreichen wäre über Trigger. Sie können Ihre Anwendung derart abändern, das die BBOX Spalte aktuell bleibt oder ein UPDATE nach jeder Änderung durchführen.

6.2 CLUSTER auf die geometrischen Indizes

Für Tabelle die hauptsächlich read-only sind und bei denen ein einzelner Index für die Mehrheit der Abfragen verwendet wird, bietet PostgreSQL den CLUSTER Befehl. Dieser Befehl ordnet alle Datenzeilen in derselben Reihenfolge an wie die Kriterien bei der Indexerstellung, was zu zwei Performance Vorteilen führt: Erstens wird für die Index Range Scans die Anzahl der Suchabfragen über die Datentabelle stark reduziert. Zweitens, wenn sich der Arbeitsbereich auf einige kleine Intervale des Index beschränkt ist das Caching effektiver, da die Datenzeilen über weniger data pages verteilt sind. (Sie dürfen sich nun eingeladen fühlen, die Dokumentation über den CLUSTER Befehl in der PostgreSQL Hilfe nachzulesen.)

Die aktuelle PostgreSQL Version erlaubt allerdings kein clustern an Hand von PostGIS GIST Indizes, da GIST Indizes NULL Werte einfach ignorieren. Sie erhalten eine Fehlermeldung wie:

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

Wie die HINT Meldung mitteilt, kann man diesen Mangel umgehen indem man eine "NOT NULL" Bedingung auf die Tabelle setzt:

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

Dies funktioniert natürlich nicht, wenn Sie tatsächlich NULL Werte in Ihrer Geometriespalte benötigen. Außerdem müssen Sie die obere Methode zum Hinzufügen der Bedingung verwenden. Die Verwendung einer CHECK Bedingung wie "ALTER TABLE blubb ADD CHECK (geometry is not null);" wird nicht klappen.

6.3 Vermeidung von Dimensionsumrechnungen

Manchmal kann es vorkommen, das Sie 3D- oder 4D-Daten in Ihrer Tabelle haben, aber immer mit den OpenGIS compliant ST_AsText() oder ST_AsBinary() Funktionen, die lediglich 2D Geometrien ausgeben, zugreifen. Dies geschieht indem intern die ST_Force2D() Funktion aufgerufen wird, welche einen wesentlichen Overhead für große Geometrien aufweist. Um diesen Overhead zu vermeiden kann es praktikabel sein diese zusätzlichen Dimensionen ein für alle mal im Voraus zu löschen:

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

Beachten Sie bitte, falls Sie die Geometriespalte über AddGeometryColumn() hinzugefügt haben, das dadurch eine Bedingung auf die Dimension der Geometrie gesetzt ist. Um dies zu Überbrücken löschen Sie die Bedingung. Vergessen Sie bitte nicht den Eintrag in die geometry_columns Tabelle zu erneuern und die Bedingung anschließend erneut zu erzeugen.

Bei großen Tabellen kann es vernünftig sein, diese UPDATE in mehrere kleinere Portionen aufzuteilen, indem man das UPDATE mittels WHERE Klausel und eines Primärschlüssels, oder eines anderen passenden Kriteriums, beschränkt und ein einfaches "VACUUM;" zwischen den UPDATEs aufruft. Dies verringert den Bedarf an temporären Festplattenspeicher drastisch. Außerdem, falls die Datenbank gemischte Dimensionen der Geometrie aufweist, kann eine Einschränkung des UPDATES mittels "WHERE dimension(the_geom)>2" das wiederholte Schreiben von Geometrien, welche bereits in 2D sind, vermeiden.

Chapter 7

Referenz PostGIS

Nachfolgend sind jene Funktionen aufgeführt, die ein PostGIS Anwender am ehesten benötigt. Es gibt weitere Funktionen, die jedoch keinen Nutzen für den allgemeinen Anwender haben, da es sich um Hilfsfunktionen für PostGIS Objekte handelt.

Note
PostGIS hat begonnen die bestehende Namenskonvention in eine SQL-MM orientierte Konvention zu ändern. Daher wurden die meisten Funktionen, die Sie kennen und lieben gelernt haben, mit dem Standardpräfix (ST) für spatiale Datentypen umbenannt. Vorhergegangene Funktionen sind noch verfügbar; wenn es aber entsprechende aktualisierte Funktionen gibt, dann werden sie in diesem Dokument nicht mehr aufgeführt. Wenn Funktionen kein ST_ Präfix aufweisen und in dieser Dokumentation nicht mehr angeführt sind, dann gelten sie als überholt und werden in einer zukünftigen Release entfernt. Benutzten Sie diese daher BITTE NICHT MEHR.

7.1 PostgreSQL und PostGIS Datentypen - Geometry/Geography/Box

7.1.1 box2d

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

Beschreibung

Box3D ist ein geometrischer Datentyp, der den umschreibenden Quader einer oder mehrerer geometrischer Objekte abbildet. ST_3DExtent gibt ein Box3D-Objekt zurück.

The representation contains the values `xmin, ymin, xmax, ymax`. These are the minimum and maximum values of the X and Y extents.

`box2d` objects have a text representation which looks like `BOX(1 2,5 6)`.

Typumwandlung

Dieser Abschnitt beschreibt sowohl die automatischen, als auch die expliziten Typumwandlungen, die für diesen Datentyp erlaubt sind.

Typumwandlung nach	Verhaltensweise
box3d	automatisch
geometry	automatisch

Siehe auch

Section 12.7

7.1.2 box3d

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

Beschreibung

Box3D ist ein geometrischer Datentyp, der den umschreibenden Quader einer oder mehrerer geometrischer Objekte abbildet. ST_3DExtent gibt ein Box3D-Objekt zurück.

The representation contains the values `xmin, ymin, zmin, xmax, ymax, zmax`. These are the minimum and maxium values of the X, Y and Z extents.

`box3d` objects have a text representation which looks like `BOX3D(1 2 3,5 6 5)`.

Typumwandlung

Dieser Abschnitt beschreibt sowohl die automatischen, als auch die expliziten Typumwandlungen, die für diesen Datentyp erlaubt sind.

Typumwandlung nach	Verhaltensweise
box	automatisch
box2d	automatisch
geometry	automatisch

Siehe auch

Section 12.7

7.1.3 geometry

geometry — Der geographische Datentyp "Geography" wird zur Abbildung eines Geoobjektes im geographischen Kugelkoordinatensystem verwendet.

Beschreibung

Der Datentyp "geometry" ist der elementare räumliche Datentyp von PostGIS zur Abbildung eines Geoobjektes in das kartesische Koordinatensystem.

Alle räumlichen Operationen an einer Geometrie verwenden die Einheiten des Koordinatenreferenzsystems in dem die Geometrie vorliegt.

Typumwandlung

Dieser Abschnitt beschreibt sowohl die automatischen, als auch die expliziten Typumwandlungen, die für diesen Datentyp erlaubt sind.

Typumwandlung nach	Verhaltensweise
box	automatisch
box2d	automatisch

box3d	automatisch
Bytea	automatisch
geography	automatisch
Text	automatisch

Siehe auch

Section 4.1, Section 4.3

7.1.4 geometry_dump

geometry_dump — A composite type used to describe the parts of complex geometry.

Beschreibung

`geometry_dump` is a composite data type containing the fields:

- `geom` - a geometry representing a component of the dumped geometry. The geometry type depends on the originating function.
- `path[]` - an integer array that defines the navigation path within the dumped geometry to the `geom` component. The path array is 1-based (i.e. `path[1]` is the first element.)

It is used by the `ST_Dump*` family of functions as an output type to explode a complex geometry into its constituent parts.

Siehe auch

Section 12.6

7.1.5 geography

geography — The type representing spatial features with geodetic (ellipsoidal) coordinate systems.

Beschreibung

Der geographische Datentyp "Geography" wird zur Abbildung eines Geoobjektes im geographischen Kugelkoordinatensystem verwendet.

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

Typumwandlung

Dieser Abschnitt beschreibt sowohl die automatischen, als auch die expliziten Typumwandlungen, die für diesen Datentyp erlaubt sind.

Typumwandlung nach	Verhaltensweise
geometry	explizit

Siehe auch

Section 4.3, Section 4.3

7.2 Geometrische Managementfunktionen

7.2.1 AddGeometryColumn

AddGeometryColumn — Entfernt eine Geometriespalte aus einer räumlichen Tabelle.

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);

Beschreibung

Fügt eine Geometriespalte zu den Attributen einer bestehende Tabelle hinzu. Der `schema_name` ist der Name des Schemas, in dem sich die Tabelle befindet. Bei der `srid` handelt es sich um eine Ganzzahl, welche auf einen entsprechenden Eintrag in der SPATIAL_REF_SYS Tabelle verweist. Beim `type` handelt es sich um eine Zeichenkette, welche dem Geometrietyp entsprechen muss, z.B.: 'POLYGON' oder 'MULTILINESTRING'. Falls der Name des Schemas nicht existiert (oder im aktuellen search_path nicht sichtbar ist), oder die angegebene SRID, der Geometrietyp, oder die Dimension ungültig sind, wird ein Fehler angezeigt.

Note
Änderung: 2.0.0 Diese Funktion aktualisiert die geometry_columns Tabelle nicht mehr, da geometry_columns jetzt ein View ist, welcher den Systemkatalog ausliest. Standardmäßig werden auch keine Bedingungen/constraints erzeugt, sondern es wird der in PostgreSQL integrierte Typmodifikaor verwendet. So entspricht zum Beispiel die Erzeugung einer wgs84 POINT Spalte mit dieser Funktion: `ALTER TABLE some_table ADD COLUMN geom geometry(Point,4326);`
Änderung: 2.0.0 Falls Sie das alte Verhalten mit Constraints wünschen, setzen Sie bitte `use_typmod` vom standardmäßigen true auf false.

Note
Änderung: 2.0.0 Views können nicht mehr händisch in "geometry_columns" registriert werden. Views auf eine Geometrie in Typmod-Tabellen, bei denen keine Adapterfunktion verwendet wird, registrieren sich selbst auf korrekte Weise, da sie die Typmod-Verhaltensweise von der Spalte der Stammtabelle erben. Views die ein geometrische Funktion ausführen die eine andere Geometrie ausgibt, benötigen die Umwandlung in eine Typmod-Geometrie, damit die Geometrie des Views korrekt in "geometry_columns" registriert wird. Siehe Section 4.6.3.

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

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

This method supports Circular Strings and Curves.

Verbesserung: 2.0.0 use_typmod Argument eingeführt. Standardmäßig wird eine typmod Geometrie anstelle einer Constraint-basierten Geometrie erzeugt.

Beispiele

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

-- Describing the table shows a simple table with a single "id" column.
postgis=# \d my_schema.my_spatial_table
                                                    Table "my_schema.my_spatial_table"
 Column |  Type   |                                  Modifiers
--------+---------+---------------------------------------------------------------------------- ↩

 id     | integer | not null default nextval('my_schema.my_spatial_table_id_seq'::regclass)

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

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

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

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

                                    Table "my_schema.my_spatial_table"
  Column  |         Type         |                                 Modifiers
----------+----------------------+-----------------------------------------------------------------------

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

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

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

Siehe auch

DropGeometryColumn, DropGeometryTable, Section 4.6.2, Section 4.6.3

7.2.2 DropGeometryColumn

DropGeometryColumn — Entfernt eine Geometriespalte aus einer räumlichen Tabelle.

Synopsis

text **DropGeometryColumn**(varchar table_name, varchar column_name);
text **DropGeometryColumn**(varchar schema_name, varchar table_name, varchar column_name);
text **DropGeometryColumn**(varchar catalog_name, varchar schema_name, varchar table_name, varchar column_name);

Beschreibung

Entfernt eine geometrische Spalte aus der Geometrietabelle. Der "schema_name" muss mit dem Feld "f_table_schema" in der Tabelle "geometry_columns" übereinstimmen.

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

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

This method supports Circular Strings and Curves.

Note
Änderung: 2.0.0 Diese Funktion wurde zwecks Abwärtskompatibilität eingeführt. Seit geometry_columns ein View auf den Systemkatalog ist, können Sie die Geometriespalte, so wie jede andere Tabellenspalte, mit `ALTER TABLE` löschen.

Beispiele

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

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

Siehe auch

AddGeometryColumn, DropGeometryTable, Section 4.6.2

7.2.3 DropGeometryTable

DropGeometryTable — Löscht eine Tabelle und alle Referenzen in dem geometry_columns View.

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);

Beschreibung

Löscht eine Tabelle und deren Verweise in "geometry_columns". Anmerkung: verwendet current_schema() wenn kein Schema angegeben wird, eine Schema erkennende pgsql Installation vorausgesetzt.

Note
Änderung: 2.0.0 Diese Funktion wurde zwecks Abwärtskompatibilität eingeführt. Seit geometry_columns ein View auf den Systemkatalog ist, können Sie eine Tabelle mit einer Geometriespalte, so wie jede andere Tabelle, mit `DROP TABLE` löschen.

Beispiele

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

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

Siehe auch

AddGeometryColumn, DropGeometryColumn, Section 4.6.2

7.2.4 Find_SRID

Find_SRID — Returns the SRID defined for a geometry column.

Synopsis

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

Beschreibung

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

Beispiele

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

Siehe auch

ST_SRID

7.2.5 Populate_Geometry_Columns

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

Synopsis

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

Beschreibung

Sorgt dafür, dass die Geometriespalten mit Typmodifikatoren oder mit passenden räumlichen Constraints versehen sind. Dadurch wird die korrekte Registrierung im View `geometry_columns` sichergestellt. Standardmäßig werden alle Geometriespalten, die keinen Typmodifikator aufweisen, mit Typmodifikatoren versehen. Für die alte Verhaltensweise setzen Sie bitte `use_typmod=fal`

Aus Gründen der Abwärtskompatibilität und für räumliche Anwendungen, wie eine Tabellenvererbung bei denen jede Kindtabelle einen anderen geometrischen Datentyp aufweist, wird die alte Verhaltensweise mit Check-Constraints weiter unterstützt. Wenn Sie diese alte Verhaltensweise benötigen, können Sie den neuen Übergabewert auf FALSE setzen - `use_typmod=false`. Wenn Sie dies tun, so werden die Geometriespalten anstelle von Typmodifikatoren mit 3 Constraints erstellt. Insbesondere bedeutet dies, dass jede Geometriespalte, die zu einer Tabelle gehört, mindestens drei Constraints aufweist:

- `enforce_dims_the_geom` - stellt sicher, dass jede Geometrie dieselbe Dimension hat (siehe ST_NDims)
- `enforce_geotype_the_geom` - stellt sicher, dass jede Geometrie vom selben Datentyp ist (siehe GeometryType)
- `enforce_srid_the_geom` - stellt sicher, dass jede Geometrie die selbe Projektion hat (siehe ST_SRID)

Wenn die `oid` einer Tabelle übergeben wird, so versucht diese Funktion, die SRID, die Dimension und den Datentyp der Geometrie in der Tabelle zu bestimmen und fügt, falls notwendig, Constraints hinzu. Bei Erfolg wird eine entsprechende Spalte in die Tabelle "geometry_columns" eingefügt, andernfalls wird der Fehler abgefangen und eine Fehlermeldung ausgegeben, die das Problem beschreibt.

Wenn die `oid` eines Views übergeben wird, so versucht diese Funktion, die SRID, die Dimension und den Datentyp der Geometrie in dem View zu bestimmen und die entsprechenden Einträge in die Tabelle `geometry_columns` vorzunehmen. Constraints werden allerdings nicht erzwungen.

Die parameterlose Variante ist ein einfacher Adapter für die parametrisierte Variante, welche die Tabelle "geometry_columns" zuerst entleert und dann für jede räumliche Tabelle oder View in der Datenbank wiederbefüllt. Wo es passend ist, werden räumliche Constraints auf die Tabellen gelegt. Es wird die Anzahl der in der Datenbank gefundenen Geometriespalten und die Anzahl der in die Tabelle `geometry_columns` eingefügten Zeilen ausgegeben. Die parametrisierte Version gibt lediglich die Anzahl der Zeilen aus, die in die Tabelle `geometry_columns` eingefügt wurden.

Verfügbarkeit: 1.4.0

Änderung: 2.0.0 Standardmäßig werden nun Typmodifikatoren anstelle von Check-Constraints für die Beschränkung des Geometrietyps verwendet. Sie können nach wie vor stattdessen die Verhaltensweise mit Check-Constraints verwenden, indem Sie die neu eingeführte Variable `use_typmod` auf FALSE setzen.

Erweiterung: 2.0.0 Der optionale Übergabewert `use_typmod` wurde eingeführt, um bestimmen zu können, ob die Spalten mit Typmodifikatoren oder mit Check-Constraints erstellt werden sollen.

Beispiele

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

populate_geometry_columns
--------------------------
                         1


\d myspatial_table

                                    Table "public.myspatial_table"
 Column |           Type             |                            Modifiers
--------+----------------------------+-------------------------------------------------------------------

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

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

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

7.2.6 UpdateGeometrySRID

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

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);

Beschreibung

Erneuert die SRID aller Features in einer Geometriespalte; erneuert die Constraints und die Referenz in "geometry_columns". Anmerkung: verwendet current_schema() wenn kein Schema angegeben wird, eine Schema erkennende pgsql Installation vo-

rausgesetzt.

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

This method supports Circular Strings and Curves.

Beispiele

Insert geometries into roads table with a SRID set already using EWKT format:

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

Ändert die SRID der Straßentabelle auf 4326

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

Das vorhergegangene Beispiel ist gleichbedeutend mit diesr DDL Anweisung

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

Falls Sie sich in der Projektion geirrt haben (oder sie als "unknown" importiert haben) und sie in einem Aufwaschen in die Web Mercator Projektion transformieren wollen, so können Sie dies mit DDL bewerkstelligen. Es gibt jedoch keine äquivalente PostGIS Managementfunktion, die dies in einem Schritt bewerkstelligen könnte.

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

Siehe auch

UpdateRasterSRID, ST_SetSRID, ST_Transform

7.3 Geometrische Konstruktoren

7.3.1 ST_Collect

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

Synopsis

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

Beschreibung

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

Variant 1: accepts two input geometries

Variant 2: accepts an array of geometries

Variant 3: aggregate function accepting a rowset of geometries.

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

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

Verfügbarkeit: 1.4.0 - ST_MakeLine(geomarray) wurde eingeführt. ST_MakeLine Aggregatfunktion wurde verbessert, um mehr Punkte schneller handhaben zu können.

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

This method supports Circular Strings and Curves.

Beispiele - Verwendung von XLink

Collect 2D points.

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

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

Collect 3D points.

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

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

Collect curves.

```
SELECT ST_AsText( ST_Collect( 'CIRCULARSTRING(220268 150415,220227 150505,220227 150406)',
                'CIRCULARSTRING(220227 150406,2220227 150407,220227 150406)'));

                st_astext
------------------------------------------------------------------------------
MULTICURVE(CIRCULARSTRING(220268 150415,220227 150505,220227 150406),
 CIRCULARSTRING(220227 150406,2220227 150407,220227 150406))
```

Beispiele: Verwendung der Feld-Version

Using an array constructor for a subquery.

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

Using an array constructor for values.

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

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

Beispiele: Spatiale Aggregatversion

Creating multiple collections by grouping geometries in a table.

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

Siehe auch

ST_Dump, ST_AsBinary

7.3.2 ST_LineFromMultiPoint

ST_LineFromMultiPoint — Erzeugt einen LineString aus einer MultiPoint Geometrie.

Synopsis

geometry **ST_LineFromMultiPoint**(geometry aMultiPoint);

Beschreibung

Erzeugt einen LineString aus einer MultiPoint Geometrie.

Für Punkt mit X-, Y- und M-Koordinaten verwenden Sie bitte ST_MakePointM .

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

Beispiele

Erzeugt einen LineString aus einer MultiPoint Geometrie.

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

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

Siehe auch

ST_AsEWKT, ST_AsKML

7.3.3 ST_MakeEnvelope

ST_MakeEnvelope — Erzeugt ein rechteckiges Polygon aus den gegebenen Minimum- und Maximumwerten. Die Eingabewerte müssen in dem Koordinatenreferenzsystem sein, welches durch die SRID angegeben wird.

Synopsis

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

Beschreibung

Erzeugt ein rechteckiges Polygon das durch die Minima und Maxima angegeben wird. durch die gegebene Hülle. Die Eingabewerte müssen in dem Koordinatenreferenzsystem sein, welches durch die SRID angegeben wird. Wenn keine SRID angegeben ist, so wird das Koordinatenreferenzsystem "unknown" angenommen

Verfügbarkeit: 1.5

Erweiterung: 2.0: es wurde die Möglichkeit eingeführt, eine Einhüllende/Envelope festzulegen, ohne dass die SRID spezifiziert ist.

Beispiel: Ein Umgebungsrechteck Polygon erzeugen

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

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

Siehe auch

ST_MakePoint, ST_MakePoint, ST_Point, ST_SRID

7.3.4 ST_MakeLine

ST_MakeLine — Erzeugt einen Linienzug aus einer Punkt-, Mehrfachpunkt- oder Liniengeometrie.

Synopsis

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

Beschreibung

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

Variant 1: accepts two input geometries

Variant 2: accepts an array of geometries

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

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

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

Verfügbarkeit: Mit 2.3.0 wurde die Unterstützung zur Eingabe von MultiPoint Elementen eingeführt

Verfügbarkeit: 2.0.0 - Unterstützung zur Eingabe von LineString Elementen eingeführt

Verfügbarkeit: 1.4.0 - ST_MakeLine(geomarray) wurde eingeführt. ST_MakeLine Aggregatfunktion wurde verbessert, um mehr Punkte schneller handhaben zu können.

Beispiele: Verwendung der Feld-Version

Create a line composed of two points.

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

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

Erzeugt eine BOX3D, die durch 2 geometrische 3D-Punkte definiert wird.

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

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

Erzeugt einen Linienzug aus einer Punkt-, Mehrfachpunkt- oder Liniengeometrie.

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

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

Beispiele: Verwendung der Feld-Version

Create a line from an array formed by a subquery with ordering.

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

Create a 3D line from an array of 3D points

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

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

Beispiele: Spatiale Aggregatversion

Diese Beispiel nimmt eine Abfolge von GPS Punkten entgegen und erzeugt einen Datensatz für jeden GPS Pfad, wobei das Geometriefeld ein Linienzug ist, welcher in der Reihenfolge der Aufnahmeroute aus den GPS Punkten zusammengesetzt wird.

Using aggregate `ORDER BY` provides a correctly-ordered LineString.

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

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

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

Siehe auch

ST_RemoveRepeatedPoints, ST_AsText, ST_GeomFromText, ST_MakePoint

7.3.5 ST_MakePoint

ST_MakePoint — Erzeugt eine 2D-, 3DZ- oder 4D-Punktgeometrie.

Synopsis

geometry **ST_Point**(float x_lon, float y_lat);

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

geometry **ST_MakePoint**(double precision x, double precision y, double precision z, double precision m);

Beschreibung

Erzeugt eine 2D-, 3DZ- oder 4D-Punktgeometrie.

Für Punkt mit X-, Y- und M-Koordinaten verwenden Sie bitte ST_MakePointM .

Erzeugt eine 2D-, 3DZ- oder 4D-Punktgeometrie (Geometrie mit Kilometrierung). `ST_MakePoint` ist zwar nicht OGC-konform, ist aber im Allgemeinen schneller und genauer als ST_GeomFromText oderST_PointFromText und auch leichter anzuwenden wenn Sie mit rohen Koordinaten anstatt mit WKT arbeiten.

Note
For geodetic coordinates, `X` is longitude and `Y` is latitude

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

Beispiele

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

--Return point marked as WGS 84 long lat
SELECT ST_SetSRID(ST_MakePoint(-71.1043443253471, 42.3150676015829),4326);

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

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

Siehe auch

ST_GeomFromText, ST_PointFromText, ST_SetSRID, ST_MakePointM

7.3.6 ST_MakePointM

ST_MakePointM — Erzeugt einen Punkt mit x, y und measure/Kilometrierungs Koordinaten.

Synopsis

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

Beschreibung

Erzeugt einen Punkt mit x, y und measure/Kilometrierungs Koordinaten.

Für Punkt mit X-, Y- und M-Koordinaten verwenden Sie bitte ST_MakePointM .

Note
For geodetic coordinates, `X` is longitude and `Y` is latitude

Beispiele

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

Create point with unknown SRID.

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

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

Erzeugt einen Punkt mit x, y und measure/Kilometrierungs Koordinaten.

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

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

Get measure of created point.

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

result
-------
10
```

Siehe auch

ST_AsEWKT, ST_MakePoint, ST_SetSRID

7.3.7 ST_MakePolygon

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

Synopsis

geometry **ST_MakePolygon**(geometry linestring);

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

Beschreibung

Erzeugt ein Polygon, das durch die gegebene Hülle gebildet wird. Die Eingabegeometrie muss aus geschlossenen Linienzügen bestehen.

Variant 1: Accepts one shell LineString.

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

Note
Diese Funktion akzeptiert keine MULTILINESTRINGs. Verwenden Sie bitte ST_LineMerge oder ST_Dump um Linienzüge zu erzeugen.

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

Beispiele: Verwendung der Feld-Version

Erzeugt einen LineString aus einem codierten Linienzug.

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

Create a Polygon from an open LineString, using ST_StartPoint and ST_AddPoint to close it.

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

Erzeugt einen LineString aus einem codierten Linienzug.

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

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

Create a Polygon from a LineString with measures

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

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

Beispiele: Außenhülle mit inneren Ringen

Erzeugung eines Donuts mit einem Ameisenloch

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

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

Note
Das Konstrukt mit CASE wird verwendet, da die Übergabe eines NULL-Feldes an ST_MakePolygon NULL ergibt.

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

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

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

Siehe auch

ST_BuildArea ST_Polygon

7.3.8 ST_Point

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

Synopsis

geometry **ST_Point**(float x_lon, float y_lat);

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

Beschreibung

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

Note
For geodetic coordinates, `X` is longitude and `Y` is latitude

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

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

Beispiele: Geometrie

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

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

New in 3.2.0: With SRID specified

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

Beispiele: Geographie

Pre-PostGIS 3.2 syntax

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

3.2 and on you can include the srid

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

PostgreSQL also provides the `::` short-hand for casting

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

If the point coordinates are not in a geodetic coordinate system (such as WGS84), then they must be reprojected before casting to a geography. In this example a point in Pennsylvania State Plane feet (SRID 2273) is projected to WGS84 (SRID 4326).

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

Siehe auch

Section 4.3, ST_MakePoint, ST_SetSRID, ST_Transform, ST_PointZ, ST_PointM, ST_PointZM

7.3.9 ST_PointZ

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

Synopsis

geometry **ST_MakePoint**(double precision x, double precision y, double precision z, double precision m);

Beschreibung

Gibt einen ST_Point mit den gegebenen Koordinatenwerten aus. Ein OGC-Alias für ST_MakePoint.

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

Beispiele

```
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)
```

Siehe auch

ST_MakePoint, ST_PointFromText, ST_SetSRID, ST_MakePointM

7.3.10 ST_PointM

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

Synopsis

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

Beschreibung

Gibt einen ST_Point mit den gegebenen Koordinatenwerten aus. Ein OGC-Alias für ST_MakePoint.

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

Beispiele

```
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)
```

Siehe auch

ST_MakePoint, ST_PointFromText, ST_SetSRID, ST_MakePointM

7.3.11 ST_PointZM

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

Synopsis

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

Beschreibung

Gibt einen ST_Point mit den gegebenen Koordinatenwerten aus. Ein OGC-Alias für ST_MakePoint.

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

Beispiele

```
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)
```

Siehe auch

ST_MakePoint, ST_Point, ST_PointM, ST_PointZ, ST_SetSRID

7.3.12 ST_Polygon

ST_Polygon — Creates a Polygon from a LineString with a specified SRID.

Synopsis

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

Beschreibung

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

ST_Polygon is similar to ST_MakePolygon Variant 1 with the addition of setting the SRID.

, ST_MakePoint, ST_SetSRID

Note
Diese Funktion akzeptiert keine MULTILINESTRINGs. Verwenden Sie bitte ST_LineMerge oder ST_Dump um Linienzüge zu erzeugen.

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

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

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

Beispiele

Create a 2D polygon.

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

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

Create a 3D polygon.

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

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

Siehe auch

ST_AsEWKT, ST_AsText, ST_GeomFromEWKT, ST_GeomFromText, ST_LineMerge, ST_MakePolygon

7.3.13 ST_TileEnvelope

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

Synopsis

geometry **ST_MakePoint**(double precision x, double precision y, double precision z, double precision m);

Beschreibung

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

By default, the tile envelope is in the Web Mercator coordinate system (SRID:3857) using the standard range of the Web Mercator system (-20037508.342789, 20037508.342789). This is the most common coordinate system used for MVT tiles. The optional `bounds` parameter can be used to generate tiles in any coordinate system. It is a geometry that has the SRID and extent of the "Zoom Level zero" square within which the XYZ tile system is inscribed.

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

Erweiterung: 2.0.0 Standardwert für den optionalen Parameter SRID eingefügt.

Verfügbarkeit: 2.1.0

Beispiel: Ein Umgebungsrechteck Polygon erzeugen

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

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

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

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

Siehe auch

ST_MakeEnvelope

7.3.14 ST_HexagonGrid

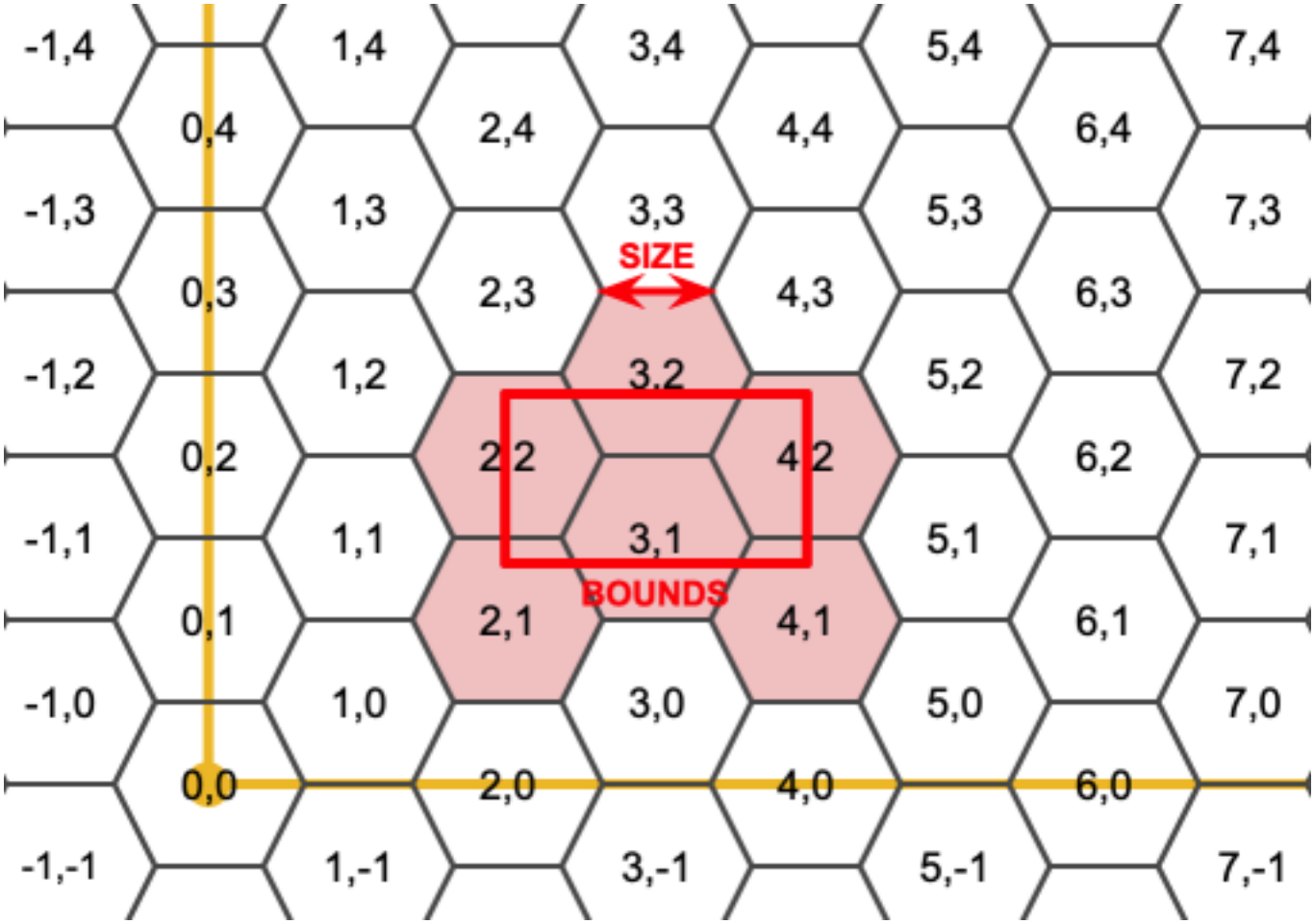
ST_HexagonGrid — Returns a set of hexagons and cell indices that completely cover the bounds of the geometry argument.

Synopsis

geometry **ST_Point**(float x_lon, float y_lat);

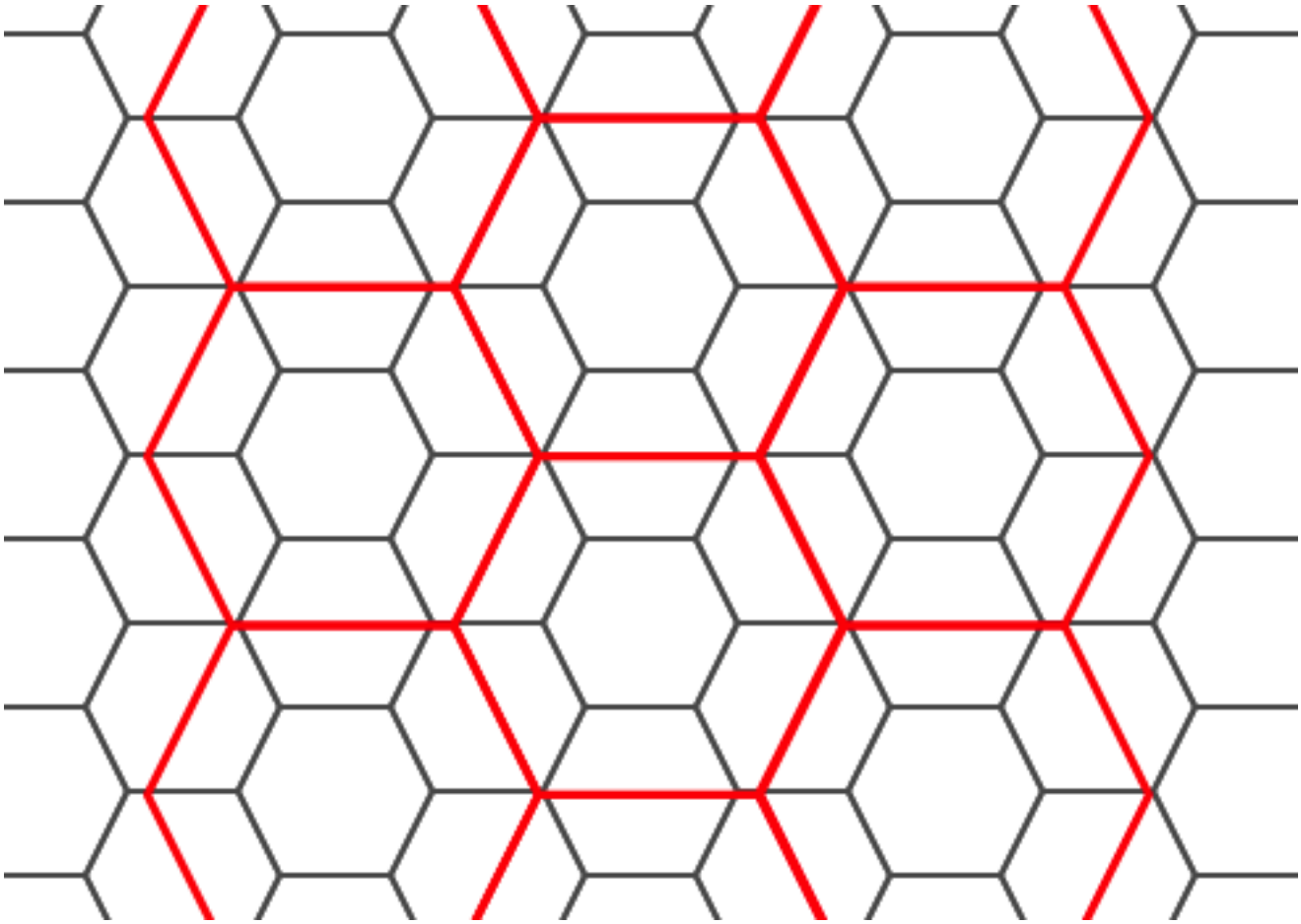
Beschreibung

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



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

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



Verfügbarkeit: 2.1.0

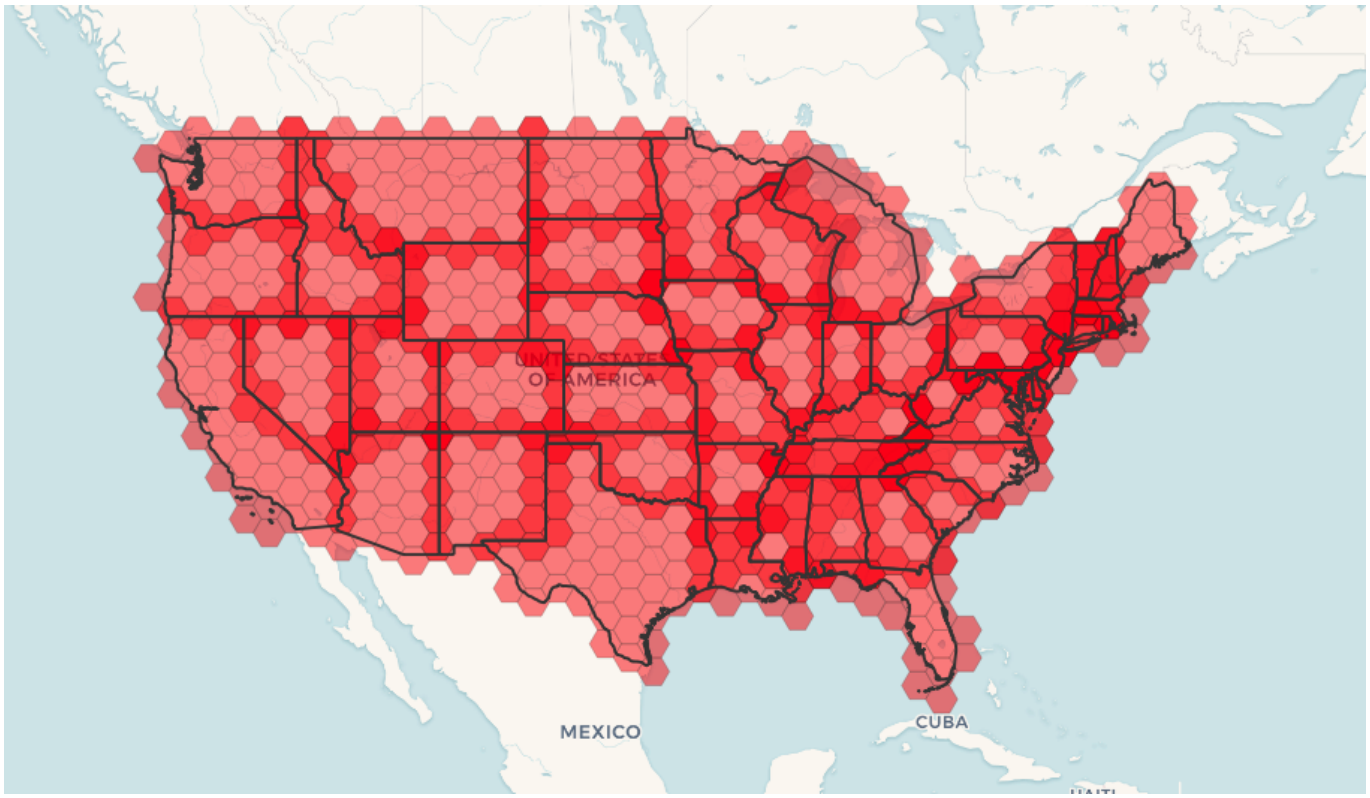
Beispiele: Verwendung der Feld-Version

To do a point summary against a hexagonal tiling, generate a hexagon grid using the extent of the points as the bounds, then spatially join to that grid.

```
SELECT COUNT(*), hexes.geom
FROM
    ST_HexagonGrid(
        10000,
        ST_SetSRID(ST_EstimatedExtent('pointtable', 'geom'), 3857)
    ) AS hexes
    INNER JOIN
    pointtable AS pts
    ON ST_Intersects(pts.geom, hexes.geom)
GROUP BY hexes.geom;
```

Beispiel: Ein Umgebungsrechteck Polygon erzeugen

If we generate a set of hexagons for each polygon boundary and filter out those that do not intersect their hexagons, we end up with a tiling for each polygon.



Tiling states results in a hexagon coverage of each state, and multiple hexagons overlapping at the borders between states.

Note
The LATERAL keyword is implied for set-returning functions when referring to a prior table in the FROM list. So CROSS JOIN LATERAL, CROSS JOIN, or just plain , are equivalent constructs for this example.

```
SELECT admin1.gid, hex.geom
FROM
    admin1
    CROSS JOIN
    ST_HexagonGrid(100000, admin1.geom) AS hex
WHERE
    adm0_a3 = 'USA'
    AND
    ST_Intersects(admin1.geom, hex.geom)
```

Siehe auch

ST_EstimatedExtent, ST_MakePoint, ST_Point, ST_SRID

7.3.15 ST_Hexagon

ST_Hexagon — Returns a single hexagon, using the provided edge size and cell coordinate within the hexagon grid space.

Synopsis

geometry **ST_MakePoint**(double precision x, double precision y, double precision z, double precision m);

Beschreibung

Uses the same hexagon tiling concept as ST_HexagonGrid, but generates just one hexagon at the desired cell coordinate. Optionally, can adjust origin coordinate of the tiling, the default origin is at 0,0.

Hexagons are generated with no SRID set, so use ST_SetSRID to set the SRID to the one you expect.

Verfügbarkeit: 2.1.0

Example: Creating a hexagon at the origin

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

POLYGON((-1 0,-0.5
         -0.866025403784439,0.5
         -0.866025403784439,1
         0,0.5
         0.866025403784439,-0.5
         0.866025403784439,-1 0))
```

Siehe auch

ST_TileEnvelope, ST_MakePoint, ST_SetSRID

7.3.16 ST_SquareGrid

ST_SquareGrid — Returns a set of grid squares and cell indices that completely cover the bounds of the geometry argument.

Synopsis

geometry **ST_Point**(float x_lon, float y_lat);

Beschreibung

Starts with the concept of a square tiling of the plane. For a given planar SRS, and a given edge size, starting at the origin of the SRS, there is one unique square tiling of the plane, Tiling(SRS, Size). This function answers the question: what grids in a given Tiling(SRS, Size) overlap with a given bounds.

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

Doubling or edge size of the square generates a new parent tiling that perfectly fits with the original tiling. Standard web map tilings in mercator are just powers-of-two square grids in the mercator plane.

Verfügbarkeit: 2.1.0

Beispiel: Ein Umgebungsrechteck Polygon erzeugen

The grid will fill the whole bounds of the country, so if you want just squares that touch the country you will have to filter afterwards with ST_Intersects.

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

Example: Counting points in squares (using single chopped grid)

To do a point summary against a square tiling, generate a square grid using the extent of the points as the bounds, then spatially join to that grid. Note the estimated extent might be off from actual extent, so be cautious and at very least make sure you've analyzed your table.

```
SELECT COUNT(*), squares.geom
    FROM
    pointtable AS pts
    INNER JOIN
    ST_SquareGrid(
        1000,
        ST_SetSRID(ST_EstimatedExtent('pointtable', 'geom'), 3857)
    ) AS squares
    ON ST_Intersects(pts.geom, squares.geom)
    GROUP BY squares.geom
```

Example: Counting points in squares using set of grid per point

This yields the same result as the first example but will be slower for a large number of points

```
SELECT COUNT(*), squares.geom
    FROM
    pointtable AS pts
    INNER JOIN
    ST_SquareGrid(
        1000,
       pts.geom
    ) AS squares
    ON ST_Intersects(pts.geom, squares.geom)
    GROUP BY squares.geom
```

Siehe auch

ST_TileEnvelope, ST_Point, ST_SetSRID, ST_SRID

7.3.17 ST_Square

ST_Square — Returns a single square, using the provided edge size and cell coordinate within the square grid space.

Synopsis

geometry **ST_MakePoint**(double precision x, double precision y, double precision z, double precision m);

Beschreibung

Uses the same square tiling concept as ST_SquareGrid, but generates just one square at the desired cell coordinate. Optionally, can adjust origin coordinate of the tiling, the default origin is at 0,0.

Squares are generated with no SRID set, so use ST_SetSRID to set the SRID to the one you expect.

Verfügbarkeit: 2.1.0

Example: Creating a square at the origin

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

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

Siehe auch

ST_TileEnvelope, ST_MakeLine, ST_MakePolygon

7.3.18 ST_Letters

ST_Letters — Returns the input letters rendered as geometry with a default start position at the origin and default text height of 100.

Synopsis

geometry **ST_Letters**(text letters, json font);

Beschreibung

Uses a built-in font to render out a string as a multipolygon geometry. The default text height is 100.0, the distance from the bottom of a descender to the top of a capital. The default start position places the start of the baseline at the origin. Over-riding the font involves passing in a json map, with a character as the key, and base64 encoded TWKB for the font shape, with the fonts having a height of 1000 units from the bottom of the descenders to the tops of the capitals.

The text is generated at the origin by default, so to reposition and resize the text, first apply the `ST_Scale` function and then apply the `ST_Translate` function.

Verfügbarkeit: 2.1.0

Beispiel: Ein Umgebungsrechteck Polygon erzeugen

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

Letters generated by ST_Letters

Example: Scaling and moving words

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

Siehe auch

ST_AsTWKB, ST_Scale, ST_Translate

7.4 Geometrische Zugriffsfunktionen

7.4.1 GeometryType

GeometryType — Gibt den Geometrietyp des ST_Geometry Wertes zurück.

Synopsis

text **GeometryType**(geometry geomA);

Beschreibung

Gibt den Geometrietyp als Zeichenkette zurück. z.B.: 'LINESTRING', 'POLYGON', 'MULTIPOINT', etc.

OGC SPEC s2.1.1.1 - Gibt den Namen des instanziierbaren Subtyps der Geometrie zurück, von dem die geometrische Instanz ein Mitglied ist. Der Name des instanziierbaren Subtyps der Geometrie wird als Zeichenkette ausgegeben.

Note
Die Funktion zeigt auch an ob die Geometrie eine Maßzahl aufweist, indem eine Zeichenkette wie 'POINTM' zurückgegeben wird.

Erweiterung: Mit 2.0.0 wurde die Unterstützung für polyedrische Oberflächen, Dreiecke und TIN eingeführt.

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

This method supports Circular Strings and Curves.

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

This function supports Polyhedral surfaces.

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

Beispiele

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

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

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

Siehe auch

ST_GeometryType

7.4.2 ST_Boundary

ST_Boundary — Gibt die abgeschlossene Hülle aus der kombinierten Begrenzung der Geometrie zurück.

Synopsis

geometry **ST_Boundary**(geometry geomA);

Beschreibung

Gibt die abgeschlossene Hülle aus der kombinierten Begrenzung der Geometrie zurück. Die Definition der kombinierte Begrenzung ist in Abschnitt 3.12.3.2 der OGC SPEC beschrieben. Da das Ergebnis dieser Funktion eine abgeschlossene Hülle und daher topologisch geschlossen ist, kann die resultierende Begrenzung durch geometrische Primitive, wie in Abschnitt 3.12.2. der OGC SPEC erörtert, dargestellt werden.

Wird durch das GEOS Modul ausgeführt

Note
Vor 2.0.0 meldete diese Funktion einen Fehler, falls sie auf eine `GEOMETRYCOLLECTION` angewandt wurde. Ab 2.0.0 wird stattdessen NULL (nicht unterstützte Eingabe) zurückgegeben.

This method implements the OGC Simple Features Implementation Specification for SQL 1.1. OGC SPEC s2.1.1.1

This method implements the SQL/MM specification. SQL-MM IEC 13249-3: 5.1.17

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

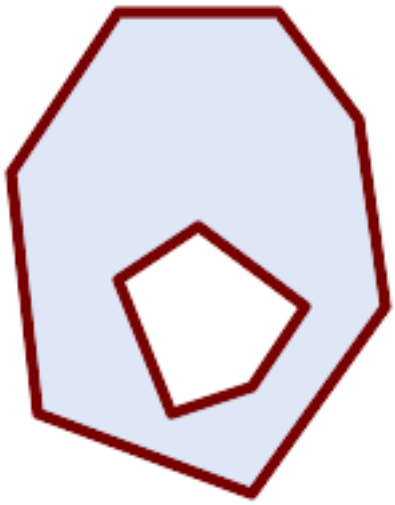
Erweiterung: mit 2.1.0 wurde die Unterstützung von Dreiecken eingeführt

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

Beispiele

Linienzug mit überlagerten Begrenzungspunkten

```
SELECT ST_Boundary(geom)
FROM (SELECT 'LINESTRING(100 150,50 60,  ↩
     70 80, 160 170)'::geometry As geom) As f;

ST_AsText output

MULTIPOINT((100 150),(160 170))
```



Polygon mit Lücke und der Abgrenzung/Boundary als Multilinestring

```
SELECT ST_Boundary(geom)
FROM (SELECT
'POLYGON (( 10 130, 50 190, 110 190, 140  ↩
     150, 150 80, 100 10, 20 40, 10 130 ),
        ( 70 40, 100 50, 120 80, 80 110,  ↩
     50 90, 70 40 ))'::geometry As geom) As f;

ST_AsText output

MULTILINESTRING((10 130,50 190,110  ↩
     190,140 150,150 80,100 10,20 40,10 130),
        (70 40,100 50,120 80,80 110,50  ↩
     90,70 40))
```

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

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

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

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

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

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

Siehe auch

ST_AsText, ST_ExteriorRing, ST_MakePolygon

7.4.3 ST_BoundingDiagonal

ST_BoundingDiagonal — Gibt die Diagonale des Umgebungsdreiecks der angegebenen Geometrie zurück.

Synopsis

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

Beschreibung

Gibt für eine angegebenen Geometrie die Diagonale des Umgebungsrechtecks als Linienzug zurück. Wenn die Geometrie leer ist, so ist auch die Diagonale Linie leer. Anderenfalls wird ein Linienzug aus 2 Punkten mit den kleinsten xy-Werten am Anfangspunkt und den größten xy-Werten am Endpunkt ausgegeben.

Der `fits` Parameter bestimmt ob die bestmögliche Anpassung notwendig ist. Wenn er FALSE ist, so kann auch die Diagonale eines etwas größeren Umgebungsrechtecks akzeptiert werden (dies ist für Geometrien mit vielen Knoten schneller). Auf jeden Fall wird immer die gesamte Eingabegeometrie durch das von der Diagonale bestimmten Umgebungsrechtecks abgedeckt.

Die zurückgegebene Linienzug-Geometrie beinhaltet immer die SRID und die Dimensionalität (Anwesenheit von Z und M) der eingegebenen Geometrie.

Note
Bei Spezialfällen (ein einzelner Knoten als Eingabewert) ist der zurückgegebene Linienzug topologisch ungültig (kein Inneres/Interior). Das Ergebnis ist dadurch jedoch nicht semantisch ungültig.

Verfügbarkeit: 2.2.0

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

This function supports M coordinates.

Beispiele

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

Siehe auch

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

7.4.4 ST_CoordDim

ST_CoordDim — Gibt die Dimension der Koordinaten für den Wert von ST_Geometry zurück.

Synopsis

integer **ST_CoordDim**(geometry geomA);

Beschreibung

Gibt die Dimension der Koordinaten für den Wert von ST_Geometry zurück.

Dies ist der MM konforme Alias für ST_NDims

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

This method implements the SQL/MM specification. SQL-MM 3: 5.1.3

This method supports Circular Strings and Curves.

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

This function supports Polyhedral surfaces.

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

Beispiele

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

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

Siehe auch

ST_NDims

7.4.5 ST_Dimension

ST_Dimension — Gibt die Dimension der Koordinaten für den Wert von ST_Geometry zurück.

Synopsis

integer **ST_Dimension**(geometry g);

Beschreibung

Die inhärente Dimension eines geometrischen Objektes, welche kleiner oder gleich der Dimension der Koordinaten sein muss. Nach OGC SPEC s2.1.1.1 wird 0 für `POINT`, 1 für `LINESTRING`, 2 for `POLYGON`, und die größte Dimension der Teile einer `GEOMETRYCOLLECTION` zurückgegeben. Wenn die Dimension nicht bekannt ist (leere`GEOMETRYCOLLECTION`) wird 0 zurückgegeben.

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

Erweiterung: 2.0.0 - Unterstützung für polyedrische Oberflächen und TIN eingeführt.

Note
Vor 2.0.0 meldete diese Funktion einen Fehler, falls sie auf eine leere Geometrie angewandt wurde.

This function supports Polyhedral surfaces.

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

Beispiele

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

Siehe auch

ST_NDims

7.4.6 ST_Dump

ST_Dump — Returns a set of `geometry_dump` rows for the components of a geometry.

Synopsis

geometry **ST_Envelope**(geometry g1);

Beschreibung

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

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

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

Erweiterung: Mit 2.0.0 wurde die Unterstützung für polyedrische Oberflächen, Dreiecke und TIN eingeführt.

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

Note
Vor 1.3.4 ist diese Funktion abgestürzt, wenn die Geometrien CURVES enthalten. Dies wurde mit 1.3.4+ behoben

---

This method supports Circular Strings and Curves.

This function supports Polyhedral surfaces.

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

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

Standard Beispiele

```
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)
```

Beispiele für polyedrische Oberflächen, TIN und Dreieck

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

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

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

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

Siehe auch

geometry_dump, ST_GeomFromEWKT, ST_Dump, ST_GeometryN, ST_NumGeometries

7.4.7 ST_DumpPoints

ST_DumpPoints — Gibt eine Zusammenfassung des Inhalts einer Geometrie wieder.

Synopsis

geometry **ST_Points**(geometry geom);

Beschreibung

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

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

To obtain a single geometry containing the coordinates use ST_Points.

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

Erweiterung: Mit 2.0.0 wurde die Unterstützung für polyedrische Oberflächen, Dreiecke und TIN eingeführt.

Verfügbarkeit: 1.2.2

This method supports Circular Strings and Curves.

This function supports Polyhedral surfaces.

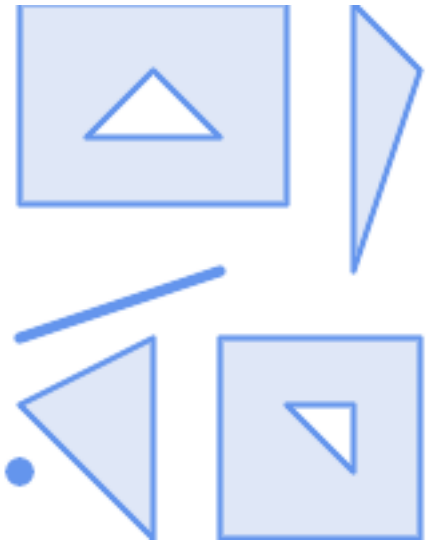
This function supports Triangles and Triangulated Irregular Network Surfaces (TIN).

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

Classic Explode a Table of LineStrings into nodes

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

Standard Beispiele



```
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)
```

Beispiele für polyedrische Oberflächen, TIN und Dreieck

```
-- Polyhedral surface cube --
SELECT (g.gdump).path, ST_AsEWKT((g.gdump).geom) as wkt
  FROM
    (SELECT
       ST_DumpPoints(ST_GeomFromEWKT('POLYHEDRALSURFACE( ((0 0 0, 0 0 1, 0 1 1, 0 1 0, 0 0  ↩
          0)),
((0 0 0, 0 1 0, 1 1 0, 1 0 0, 0 0 0)), ((0 0 0, 1 0 0, 1 0 1, 0 0 1, 0 0 0)),
((1 1 0, 1 1 1, 1 0 1, 1 0 0, 1 1 0)),
((0 1 0, 0 1 1, 1 1 1, 1 1 0, 0 1 0)), ((0 0 1, 1 0 1, 1 1 1, 0 1 1, 0 0 1)) )') ) AS gdump
```

```
    ) AS g;
-- result --
  path   |     wkt
---------+--------------
 {1,1,1} | POINT(0 0 0)
 {1,1,2} | POINT(0 0 1)
 {1,1,3} | POINT(0 1 1)
 {1,1,4} | POINT(0 1 0)
 {1,1,5} | POINT(0 0 0)
 {2,1,1} | POINT(0 0 0)
 {2,1,2} | POINT(0 1 0)
 {2,1,3} | POINT(1 1 0)
 {2,1,4} | POINT(1 0 0)
 {2,1,5} | POINT(0 0 0)
 {3,1,1} | POINT(0 0 0)
 {3,1,2} | POINT(1 0 0)
 {3,1,3} | POINT(1 0 1)
 {3,1,4} | POINT(0 0 1)
 {3,1,5} | POINT(0 0 0)
 {4,1,1} | POINT(1 1 0)
 {4,1,2} | POINT(1 1 1)
 {4,1,3} | POINT(1 0 1)
 {4,1,4} | POINT(1 0 0)
 {4,1,5} | POINT(1 1 0)
 {5,1,1} | POINT(0 1 0)
 {5,1,2} | POINT(0 1 1)
 {5,1,3} | POINT(1 1 1)
 {5,1,4} | POINT(1 1 0)
 {5,1,5} | POINT(0 1 0)
 {6,1,1} | POINT(0 0 1)
 {6,1,2} | POINT(1 0 1)
 {6,1,3} | POINT(1 1 1)
 {6,1,4} | POINT(0 1 1)
 {6,1,5} | POINT(0 0 1)
(30 rows)
```

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

```
-- TIN --
SELECT (g.gdump).path, ST_AsEWKT((g.gdump).geom) as wkt
  FROM
    (SELECT
       ST_DumpPoints( ST_GeomFromEWKT('TIN (((
                0 0 0,
```

```
                0 0 1,
                0 1 0,
                0 0 0
            )), ((
                0 0 0,
                0 1 0,
                1 1 0,
                0 0 0
            ))
            )') ) AS gdump
    ) AS g;
-- result --
  path   |     wkt
---------+--------------
 {1,1,1} | POINT(0 0 0)
 {1,1,2} | POINT(0 0 1)
 {1,1,3} | POINT(0 1 0)
 {1,1,4} | POINT(0 0 0)
 {2,1,1} | POINT(0 0 0)
 {2,1,2} | POINT(0 1 0)
 {2,1,3} | POINT(1 1 0)
 {2,1,4} | POINT(0 0 0)
(8 rows)
```

Siehe auch

geometry_dump, ST_GeomFromEWKT, ST_Dump, ST_GeometryN, ST_NumGeometries

7.4.8 ST_DumpSegments

ST_DumpSegments — Gibt eine Zusammenfassung des Inhalts einer Geometrie wieder.

Synopsis

geometry **ST_Points**(geometry geom);

Beschreibung

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

- Gibt den Wert TRUE zurück, wenn der LINESTRING geschlossen ist und der Simple Feature Spezifikation entspricht.
- the *path* field (an integer[]) is an index enumerating the segment start point positions in the elements of the supplied geometry. The indices are 1-based. For example, for a LINESTRING the paths are {i} where i is the nth segment start point in the LINESTRING. For a POLYGON the paths are {i, j} where i is the ring number (1 is outer; inner rings follow) and j is the segment start point position in the ring.

Verfügbarkeit: 2.2.0

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

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

Standard Beispiele

```
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   &#x2502;      st_astext
---------------------------------
 {1,1}   &#x2502; LINESTRING(1 1,3 3)
 {1,2}   &#x2502; LINESTRING(3 3,4 4)
 {2,1,1} &#x2502; LINESTRING(5 5,6 6)
 {2,1,2} &#x2502; LINESTRING(6 6,7 7)
 {2,1,3} &#x2502; LINESTRING(7 7,5 5)
(5 rows)
```

Beispiele für polyedrische Oberflächen, TIN und Dreieck

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

 path  &#x2502;      st_astext
 ---------------------------------
 {1,1} &#x2502; LINESTRING(0 0,0 9)
 {1,2} &#x2502; LINESTRING(0 9,9 0)
 {1,3} &#x2502; LINESTRING(9 0,0 0)
(3 rows)
```

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

```
        ) AS g
) j;

  path   &#x2502;        st_asewkt
  --------------------------------
 {1,1,1} &#x2502; LINESTRING(0 0 0,0 0 1)
 {1,1,2} &#x2502; LINESTRING(0 0 1,0 1 0)
 {1,1,3} &#x2502; LINESTRING(0 1 0,0 0 0)
 {2,1,1} &#x2502; LINESTRING(0 0 0,0 1 0)
 {2,1,2} &#x2502; LINESTRING(0 1 0,1 1 0)
 {2,1,3} &#x2502; LINESTRING(1 1 0,0 0 0)
(6 rows)
```

Siehe auch

geometry_dump, ST_Collect, ST_Dump, ST_NumInteriorRing,

7.4.9 ST_DumpRings

ST_DumpRings — Returns a set of `geometry_dump` rows for the exterior and interior rings of a Polygon.

Synopsis

geometry **ST_ExteriorRing**(geometry a_polygon);

Beschreibung

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

The *geom* field contains each ring as a POLYGON. The *path* field is an integer array of length 1 containing the polygon ring index. The exterior ring (shell) has index 0. The interior rings (holes) have indices of 1 and higher.

Note
Dies funktioniert nicht mit MULTIPOLYGONen. Verwenden Sie die Funktion bitte in Zusammenhang mit ST_Dump um sie auf MULTIPOLYGONe anzuwenden.

Availability: PostGIS 1.1.3. Requires PostgreSQL 7.3 or higher.

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

Beispiele

General form of query.

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

A polygon with a single hole.

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

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

Siehe auch

geometry_dump, ST_GeomFromEWKT, ST_Dump, ST_GeometryN, ST_NumGeometries

7.4.10 ST_EndPoint

ST_EndPoint — Gibt die Anzahl der Stützpunkte eines ST_LineString oder eines ST_CircularString zurück.

Synopsis

geometry **ST_Points**(geometry geom);

Beschreibung

Gibt den Anfangspunkt einer `LINESTRING` oder `CIRCULARLINESTRING` Geometrie als `POINT` oder `NULL` zurück, falls es sich beim Eingabewert nicht um einen `LINESTRING` oder `CIRCULARLINESTRING` handelt.

This method implements the SQL/MM specification. SQL-MM 3: 7.1.4

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

This method supports Circular Strings and Curves.

Note
Änderung: 2.0.0 unterstützt die Verarbeitung von MultiLinestring's die nur aus einer einzelnen Geometrie bestehen, nicht mehr. In früheren Versionen von PostGIS gab die Funktion bei einem aus einer einzelnen Linie bestehender MultiLinestring den Anfangspunkt zurück. Ab 2.0.0 gibt sie nur NULL zurück, so wie bei jedem anderen MultiLinestring. Die alte Verhaltensweise war undokumentiert, aber Anwender, die annahmen, dass Sie Ihre Daten als LINESTRING vorliegen haben, könnten in 2.0 dieses zurückgegebene NULL bemerken.

Beispiele

Einhüllende von Punkt und Linienzug.

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

End point of a non-LineString is NULL

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

Einhüllende von Punkt und Linienzug.

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

Gibt die Anzahl der Stützpunkte eines ST_LineString oder eines ST_CircularString zurück.

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

Siehe auch

ST_PointN, ST_StartPoint

7.4.11 ST_Envelope

ST_Envelope — Gibt eine Geometrie in doppelter Genauigkeit (float8) zurück, welche das Umgebungsrechteck der beigestellten Geometrie darstellt.

Synopsis

geometry **ST_Envelope**(geometry g1);

Beschreibung

Gibt das kleinstmögliche Umgebungsrechteck der bereitgestellten Geometrie als Geometrie im Float8-Format zurück. Das Polygon wird durch die Eckpunkte des Umgebungsrechteckes beschrieben ((`MINX`, `MINY`), (`MINX`, `MAXY`), (`MAXX`, `MAXY`), (`MAXX`, `MINY`), (`MINX`, `MINY`)). (PostGIS fügt auch die `ZMIN`/`ZMAX` Koordinaten hinzu).

Spezialfälle (vertikale Linien, Punkte) geben eine Geometrie geringerer Dimension zurück als `POLYGON`, insbesondere `POINT` oder `LINESTRING`.

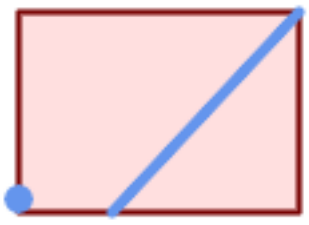
Verfügbarkeit: 1.5.0 Änderung der Verhaltensweise insofern, das die Ausgabe in Double Precision anstelle von Float4 erfolgt

This method implements the OGC Simple Features Implementation Specification for SQL 1.1. s2.1.1.1

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

Beispiele

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


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


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

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



Einhüllende von Punkt und Linienzug.

```
SELECT ST_AsText(ST_Envelope(
                ST_Collect(
                        ST_GeomFromText('LINESTRING(55 75,125 150)'),
                                ST_Point(20, 80))
```

```
                                    )) As wktenv;
wktenv
-----------
POLYGON((20 75,20 150,125 150,125 75,20 75))
```

Siehe auch

Box2D, Box3D, ST_OrientedEnvelope

7.4.12 ST_ExteriorRing

ST_ExteriorRing — Gibt die Anzahl der inneren Ringe einer Polygongeometrie aus.

Synopsis

geometry **ST_ExteriorRing**(geometry a_polygon);

Beschreibung

Gibt einen Linienzug zurück, welcher den äußeren Ring der `POLYGON` Geometrie darstellt. Gibt NULL zurück wenn es sich bei der Geometrie um kein Polygon handelt.

Note
Dies funktioniert nicht mit MULTIPOLYGONen. Verwenden Sie die Funktion bitte in Zusammenhang mit ST_Dump um sie auf MULTIPOLYGONe anzuwenden.

This method implements the OGC Simple Features Implementation Specification for SQL 1.1. 2.1.5.1

This method implements the SQL/MM specification. SQL-MM 3: 8.2.3, 8.3.3

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

Beispiele

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

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

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

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

Siehe auch

ST_InteriorRingN, ST_Boundary, ST_NumInteriorRings

7.4.13 ST_GeometryN

ST_GeometryN — Gibt den Geometrietyp des ST_Geometry Wertes zurück.

Synopsis

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

Beschreibung

Gibt die auf 1-basierende n-te Geometrie zurück, wenn es sich bei der Geometrie um eine GEOMETRYCOLLECTION, (MULTI)POINT (MULTI)LINESTRING, MULTICURVE oder (MULTI)POLYGON, POLYHEDRALSURFACE handelt. Anderenfalls wird NULL zurückgegeben.

Note
Seit Version 0.8.0 basiert der Index auf 1, so wie in der OGC Spezifikation. Vorhergegangene Versionen waren 0-basiert.

Note
Falls Sie alle Geometrien einer Geometrie entnehmen wollen, so ist ST_Dump wesentlich leistungsfähiger und es funktioniert auch mit Einzelgeometrien.

Erweiterung: Mit 2.0.0 wurde die Unterstützung für polyedrische Oberflächen, Dreiecke und TIN eingeführt.

Änderung: 2.0.0 Vorangegangene Versionen geben bei Einzelgeometrien NULL zurück. Dies wurde geändert um die Geometrie für den ST_GeometrieN(..,1) Fall zurückzugeben.

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

This method implements the SQL/MM specification. SQL-MM 3: 9.1.5

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

This method supports Circular Strings and Curves.

This function supports Polyhedral surfaces.

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

Standard Beispiele

```
--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);
```

Beispiele für polyedrische Oberflächen, TIN und Dreieck

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

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

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

```
            )')  AS geom
    ) AS g;
-- result --
                 wkt
------------------------------------
 TRIANGLE((0 0 0,0 1 0,1 1 0,0 0 0))
```

Siehe auch

ST_Dump, ST_NumGeometries

7.4.14 ST_GeometryType

ST_GeometryType — Gibt den Geometrietyp des ST_Geometry Wertes zurück.

Synopsis

text **ST_GeometryType**(geometry g1);

Beschreibung

Gibt den Geometrietyp als Zeichenkette zurück. Z.B.: 'ST_LineString', 'ST_Polygon','ST_MultiPolygon' etc. Diese Funktion unterscheidet sich von GeometryType(geometry) durch den Präfix ST_ und dadurch, das nicht angezeigt wird, ob die Geometrie eine Maßzahl besitzt.

Erweiterung: Mit 2.0.0 wurde die Unterstützung für polyedrische Oberflächen eingeführt.

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

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

This function supports Polyhedral surfaces.

Beispiele

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

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

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

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

Siehe auch

GeometryType

7.4.15 ST_HasArc

ST_HasArc — Tests if a geometry contains a circular arc

Synopsis

boolean **ST_IsEmpty**(geometry geomA);

Beschreibung

Gibt den Wert TRUE zurück, falls es sich bei der Geometrie um eine leere GeometryCollection, Polygon, Point etc. handelt.

Verfügbarkeit: 1.2.2

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

This method supports Circular Strings and Curves.

Beispiele

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

Siehe auch

ST_CurveToLine, ST_PointN

7.4.16 ST_InteriorRingN

ST_InteriorRingN — Gibt die Anzahl der inneren Ringe einer Polygongeometrie aus.

Synopsis

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

Beschreibung

Gibt den Nten innenliegenden Linienzug des Ringes der Polygongeometrie zurück. Gibt NULL zurück, falls es sich bei der Geometrie nicht um ein Polygon handelt, oder sich das angegebene N außerhalb des zulässigen Bereiches befindet.

Note
Dies funktioniert nicht mit MULTIPOLYGONen. Verwenden Sie die Funktion bitte in Zusammenhang mit ST_Dump um sie auf MULTIPOLYGONe anzuwenden.

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

This method implements the SQL/MM specification. SQL-MM 3: 8.2.6, 8.3.5

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

Beispiele

```
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;
```

Siehe auch

ST_ExteriorRing, ST_M, ST_X, ST_Y, ST_ZMax, ST_ZMin

7.4.17 ST_IsClosed

ST_IsClosed — Gibt den Wert TRUE zurück, wenn die Anfangs- und Endpunkte des LINESTRING's zusammenfallen. Bei polyedrischen Oberflächen, wenn sie geschlossen (volumetrisch) sind.

Synopsis

boolean **ST_IsClosed**(geometry g);

Beschreibung

Gibt den Wert TRUE zurück, wenn die Anfangs- und Endpunkte des LINESTRING's zusammenfallen. Bei polyedrischen Oberflächen wird angezeigt, ob die Oberfläche eine Fläche (offen) oder ein Volumen (geschlossen) beschreibt.

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

This method implements the SQL/MM specification. SQL-MM 3: 7.1.5, 9.3.3

Note
SQL-MM gibt vor, daß das Ergebnis von ST_IsClosed(NULL) 0 ergeben soll, während PostGIS NULL zurückgibt.

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

This method supports Circular Strings and Curves.

Erweiterung: Mit 2.0.0 wurde die Unterstützung für polyedrische Oberflächen eingeführt.

This function supports Polyhedral surfaces.

Beispiele für Linienzüge und Punkte

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

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

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

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

```
(1 row)

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

Beispiel für eine polyedrische Oberfläche

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

 st_isclosed
-------------
 t


 -- Same as cube but missing a side --
 SELECT ST_IsClosed(ST_GeomFromEWKT('POLYHEDRALSURFACE( ((0 0 0, 0 0 1, 0 1 1, 0 1 0, 0 0  ↩
     0)),
                ((0 0 0, 0 1 0, 1 1 0, 1 0 0, 0 0 0)), ((0 0 0, 1 0 0, 1 0 1, 0 0 1, 0 0 0) ↩
                    ),
                ((1 1 0, 1 1 1, 1 0 1, 1 0 0, 1 1 0)),
                ((0 1 0, 0 1 1, 1 1 1, 1 1 0, 0 1 0)) )'));

 st_isclosed
-------------
 f
```

Siehe auch

ST_IsRing

7.4.18 ST_IsCollection

ST_IsCollection — Gibt den Wert TRUE zurück, falls es sich bei der Geometrie um eine leere GeometryCollection, Polygon, Point etc. handelt.

Synopsis

boolean **ST_IsCollection**(geometry g);

Beschreibung

Gibt den Wert `TRUE` zurück, wenn der Geometrietyp einer der folgenden Gemetrietypen entspricht:

- GEOMETRYCOLLECTION

- MULTI{POINT,POLYGON,LINESTRING,CURVE,SURFACE}
- COMPOUNDCURVE

Note
Diese Funktion wertet den Geometrietyp aus. D.h.: sie gibt den Wert `TRUE` für Geometriekollektionen zurück, wenn diese leer sind, oder nur ein einziges Element aufweisen.

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

This method supports Circular Strings and Curves.

Beispiele

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

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

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

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

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

Siehe auch

ST_NumGeometries

7.4.19 ST_IsEmpty

ST_IsEmpty — Tests if a geometry is empty.

Synopsis

boolean **ST_IsEmpty**(geometry geomA);

Beschreibung

Gibt den Wert TRUE zurück, wenn es sich um eine leere Geometrie handelt. Falls TRUE, dann repräsentiert diese Geometrie eine leere GeometryCollection, Polygon, Point etc.

Note
SQL-MM gibt vor, daß das Ergebnis von ST_IsEmpty(NULL) der Wert 0 ist, während PostGIS den Wert NULL zurückgibt.

This method implements the OGC Simple Features Implementation Specification for SQL 1.1. s2.1.1.1

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

This method supports Circular Strings and Curves.

Warning
Änderung: 2.0.0 - In Vorgängerversionen von PostGIS war ST_GeomFromText('GEOMETRYCOLLECTION(EMPTY)') erlaubt. Um eine bessere Übereinstimmung mit der SQL/MM Norm zu erreichen, ist dies nun nicht mehr gestattet.

Beispiele

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

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

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

 st_isempty
------------
 f
(1 row)

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

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

7.4.20 ST_IsPolygonCCW

ST_IsPolygonCCW — Gibt TRUE zurück, wenn alle äußeren Ringe gegen den Uhrzeigersinn orientiert sind und alle inneren Ringe im Uhrzeigersinn ausgerichtet sind.

Synopsis

boolean **ST_IsPolygonCCW** (geometry geom);

Beschreibung

Gibt TRUE zurück, wenn für alle Bestandteile der angegebenen Geometrie gilt: die äußeren Ringe sind gegen den Uhrzeigersinn und die inneren Ringe im Uhrzeigersinn ausgerichtet.

Gibt TRUE zurück, wenn die Geometrie keine Polygonbestandteile aufweist.

Note
Da geschlossene Linienzüge nicht als Polygonbestandteile betrachtet werden, erhalten Sie auch dann TRUE, wenn Sie einen einzelnen geschlossenen Linienzug eingeben und zwar unabhängig von dessen Ausrichtung.

Note
Wenn bei einer Polygongeometrie die inneren Ringe nicht entgegengesetzt orientiert sind (insbesondere, wenn einer oder mehrere innere Ringe die selbe Ausrichtung wie die äußeren Ringe haben), dann geben sowohl ST_IsPolygonCW als auch ST_IsPolygonCCW den Wert FALSE zurück.

Verfügbarkeit: 2.2.0

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

This function supports M coordinates.

Siehe auch

ST_ForcePolygonCW , ST_ForcePolygonCCW , ST_IsPolygonCW

7.4.21 ST_IsPolygonCW

ST_IsPolygonCW — Gibt den Wert TRUE zurück, wenn alle äußeren Ringe im Uhrzeigersinn und alle inneren Ringe gegen den Uhrzeigersinn ausgerichtet sind.

Synopsis

boolean **ST_IsPolygonCW** (geometry geom);

Beschreibung

Gibt den Wert TRUE zurück, wenn für alle Polygonbestandteile der eingegebenen Geometrie gilt: die äußeren Ringe sind im Uhrzeigersinn orientiert, die inneren Ringe entgegen dem Uhrzeigersinn.

Gibt TRUE zurück, wenn die Geometrie keine Polygonbestandteile aufweist.

Note
Da geschlossene Linienzüge nicht als Polygonbestandteile betrachtet werden, erhalten Sie auch dann TRUE, wenn Sie einen einzelnen geschlossenen Linienzug eingeben und zwar unabhängig von dessen Ausrichtung.

Note
Wenn bei einer Polygongeometrie die inneren Ringe nicht entgegengesetzt orientiert sind (insbesondere, wenn einer oder mehrere innere Ringe die selbe Ausrichtung wie die äußeren Ringe haben), dann geben sowohl ST_IsPolygonCW als auch ST_IsPolygonCCW den Wert FALSE zurück.

Verfügbarkeit: 2.2.0

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

This function supports M coordinates.

Siehe auch

ST_ForcePolygonCW , ST_ForcePolygonCCW , ST_IsPolygonCW

7.4.22 ST_IsRing

ST_IsRing — Tests if a LineString is closed and simple.

Synopsis

boolean **ST_IsRing**(geometry g);

Beschreibung

Gibt den Wert `TRUE` zurück, wenn der `LINESTRING` sowohl ST_IsClosed (`ST_StartPoint` ((*g*)) ~= `ST_Endpoint` ((*g*))) als auch ST_IsSimple (sich nicht selbst überschneidet) ist.

This method implements the OGC Simple Features Implementation Specification for SQL 1.1. 2.1.5.1

This method implements the SQL/MM specification. SQL-MM 3: 7.1.6

Note
SQL-MM gibt vor, daß das Ergebnis von`ST_IsRing(NULL)` der Wert 0 sein soll, während PostGIS den Wert `NULL` zurückgibt.

Beispiele

```
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)
```

Siehe auch

ST_IsClosed, ST_IsSimple, ST_StartPoint, ST_EndPoint

7.4.23 ST_IsSimple

ST_IsSimple — Gibt den Wert (TRUE) zurück, wenn die Geometrie keine irregulären Stellen, wie Selbstüberschneidungen oder Selbstberührungen, aufweist.

Synopsis

boolean **ST_IsSimple**(geometry geomA);

Beschreibung

Gibt TRUE zurück, wenn keine regelwidrigen geometrischen Merkmale, wie Geometrien die sich selbst kreuzen oder berühren, auftreten. Für weiterführende Information zur OGC-Definition von Simplizität und Gültigkeit von Geometrien, siehe "Ensuring OpenGIS compliancy of geometries"

Note
SQL-MM definiert das Ergebnis von ST_IsSimple(NULL) als 0, während PostGIS NULL zurückgibt.

This method implements the OGC Simple Features Implementation Specification for SQL 1.1. s2.1.1.1

This method implements the SQL/MM specification. SQL-MM 3: 5.1.8

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

Beispiele

```
SELECT ST_IsSimple(ST_GeomFromText('POLYGON((1 2, 3 4, 5 6, 1 2))'));
 st_issimple
-------------
 t
```

```
(1 row)

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

Siehe auch

ST_IsValid

7.4.24 ST_M

ST_M — Returns the M coordinate of a Point.

Synopsis

float **ST_M**(geometry a_point);

Beschreibung

Gibt die M-Koordinate des Punktes zurück, oder NULL wenn keine vorhanden ist. Der Einabewert muss ein Punkt sein.

Note
Dies ist (noch) kein Teil der OGC Spezifikation, wird aber hier aufgeführt um die Liste von Funktionen zum Auslesen von Punktkoordinaten zu vervollständigen.

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

This method implements the SQL/MM specification.

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

Beispiele

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

Siehe auch

ST_GeomFromEWKT, ST_X, ST_Y, ST_Z

7.4.25 ST_MemSize

ST_MemSize — Gibt den Geometrietyp des ST_Geometry Wertes zurück.

Synopsis

integer **ST_NRings**(geometry geomA);

Beschreibung

Gibt den Geometrietyp des ST_Geometry Wertes zurück.

This complements the PostgreSQL built-in database object functions pg_column_size, pg_size_pretty, pg_relation_size, pg_total_relatio

Note

pg_relation_size which gives the byte size of a table may return byte size lower than ST_MemSize. This is because pg_relation_size does not add toasted table contribution and large geometries are stored in TOAST tables.
pg_total_relation_size - includes, the table, the toasted tables, and the indexes.
pg_column_size returns how much space a geometry would take in a column considering compression, so may be lower than ST_MemSize

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

This method supports Circular Strings and Curves.

This function supports Polyhedral surfaces.

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

Changed: 2.2.0 name changed to ST_MemSize to follow naming convention.

Beispiele

```
--Return how much byte space Boston takes up  in our Mass data set
SELECT pg_size_pretty(SUM(ST_MemSize(geom))) as totgeomsum,
pg_size_pretty(SUM(CASE WHEN town = 'BOSTON' THEN ST_MemSize(geom) ELSE 0 END)) As bossum,
CAST(SUM(CASE WHEN town = 'BOSTON' THEN ST_MemSize(geom) ELSE 0 END)*1.00 /
                SUM(ST_MemSize(geom))*100 As numeric(10,2)) As perbos
FROM towns;

totgeomsum        bossum        perbos
----------        ------        ------
1522 kB                30 kB        1.99


SELECT ST_MemSize(ST_GeomFromText('CIRCULARSTRING(220268 150415,220227 150505,220227  ↩
    150406)'));

---
73

--What percentage of our table is taken up by just the geometry
SELECT pg_total_relation_size('public.neighborhoods') As fulltable_size, sum(ST_MemSize( ↩
    geom)) As geomsize,
sum(ST_MemSize(geom))*1.00/pg_total_relation_size('public.neighborhoods')*100 As pergeom
FROM neighborhoods;
fulltable_size geomsize  pergeom
-----------------------------------------------
262144         96238         36.711883544921875000
```

7.4.26 ST_NDims

ST_NDims — Gibt die Dimension der Koordinaten für den Wert von ST_Geometry zurück.

Synopsis

integer **ST_NDims**(geometry g1);

Beschreibung

Gibt die Koordinatendimension der Geometrie zurück. PostGIS unterstützt 2- (x,y), 3- (x,y,z) oder 2D mit Kilometrierung - x,y,m, und 4- dimensionalen Raum - 3D mit Kilometrierung x,y,z,m .

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

Beispiele

```
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
```

Siehe auch

ST_CoordDim, ST_Dimension, ST_GeomFromEWKT

7.4.27 ST_NPoints

ST_NPoints — Gibt die Anzahl der Punkte (Knoten) einer Geometrie zurück.

Synopsis

integer **ST_NPoints**(geometry g1);

Beschreibung

Gibt die Anzahl der Punkte einer Geometrie zurück. Funktioniert für alle Geometrien.

Erweiterung: Mit 2.0.0 wurde die Unterstützung für polyedrische Oberflächen eingeführt.

Note

Vor 1.3.4 ist diese Funktion abgestürzt, wenn die Geometrien CURVES enthalten. Dies wurde mit 1.3.4+ behoben

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

This method supports Circular Strings and Curves.

This function supports Polyhedral surfaces.

Beispiele

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

--Polygon in 3D space
SELECT ST_NPoints(ST_GeomFromEWKT('LINESTRING(77.29 29.07 1,77.42 29.26 0,77.27 29.31  ↩
    -1,77.29 29.07 3)'))
--result
4
```

Siehe auch

ST_NumPoints

7.4.28 ST_NRings

ST_NRings — Gibt die Anzahl der inneren Ringe einer Polygongeometrie aus.

Synopsis

integer **ST_NRings**(geometry geomA);

Beschreibung

Wenn es sich bei der Geometrie um ein Polygon oder um ein MultiPolygon handelt, wird die Anzahl der Ringe zurückgegeben. Anders als NumInteriorRings werden auch die äußeren Ringe gezählt.

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

This method supports Circular Strings and Curves.

Beispiele

```
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)
```

Siehe auch

ST_NumInteriorRings

7.4.29 ST_NumGeometries

ST_NumGeometries — Gibt die Anzahl der Punkte einer Geometrie zurück. Funktioniert für alle Geometrien.

Synopsis

integer **ST_NumGeometries**(geometry geom);

Beschreibung

Gibt die Anzahl an Geometrien aus. Wenn es sich bei der Geometrie um eine GEOMETRYCOLLECTION (oder MULTI*) handelt, wird die Anzahl der Geometrien zurückgegeben, bei Einzelgeometrien wird 1, ansonsten NULL zurückgegeben.

Erweiterung: Mit 2.0.0 wurde die Unterstützung für polyedrische Oberflächen, Dreiecke und TIN eingeführt.

Änderung: 2.0.0 Bei früheren Versionen wurde NULL zurückgegeben, wenn die Geometrie nicht vom Typ GEOMETRYCOLLECTION/MULTI war. 2.0.0+ gibt nun 1 für Einzelgeometrien, wie POLYGON, LINESTRING, POINT zurück.

This method implements the SQL/MM specification. SQL-MM 3: 9.1.4

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

This function supports Polyhedral surfaces.

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

Beispiele

```
--Prior versions would have returned NULL for this -- in 2.0.0 this returns 1
SELECT ST_NumGeometries(ST_GeomFromText('LINESTRING(77.29 29.07,77.42 29.26,77.27  ↩
    29.31,77.29 29.07)'));
--result
1

--Geometry Collection Example - multis count as one geom in a collection
SELECT ST_NumGeometries(ST_GeomFromEWKT('GEOMETRYCOLLECTION(MULTIPOINT((-2 3),(-2 2)),
LINESTRING(5 5 ,10 10),
POLYGON((-7 4.2,-7.1 5,-7.1 4.3,-7 4.2)))'));
--result
3
```

Siehe auch

ST_GeometryN, ST_Multi

7.4.30 ST_NumInteriorRings

ST_NumInteriorRings — Gibt die Anzahl der inneren Ringe einer Polygongeometrie aus.

Synopsis

integer **ST_NumInteriorRings**(geometry a_polygon);

Beschreibung

Gibt die Anzahl der inneren Ringe einer Polygongeometrie aus. Gibt NULL zurück, wenn die Geometrie kein Polygon ist.

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

Änderung: 2.0.0 - In füheren Versionen war ein MULTIPOLYGON als Eingabe erlaubt, wobei die Anzahl der inneren Ringe des ersten Polygons ausgegeben wurde.

Beispiele

```
--If you have a regular polygon
SELECT gid, field1, field2, ST_NumInteriorRings(geom) AS numholes
FROM sometable;

--If you have multipolygons
--And you want to know the total number of interior rings in the MULTIPOLYGON
SELECT gid, field1, field2, SUM(ST_NumInteriorRings(geom)) AS numholes
FROM (SELECT gid, field1, field2, (ST_Dump(geom)).geom As geom
        FROM sometable) As foo
GROUP BY gid, field1,field2;
```

Siehe auch

ST_NumInteriorRing, ST_PointN

7.4.31 ST_NumInteriorRing

ST_NumInteriorRing — Gibt die Anzahl der inneren Ringe eines Polygons in der Geometrie aus. Ist ein Synonym für ST_NumInteriorRi

Synopsis

integer **ST_NumInteriorRing**(geometry a_polygon);

Siehe auch

ST_NumInteriorRings, ST_PointN

7.4.32 ST_NumPatches

ST_NumPatches — Gibt die Anzahl der Maschen einer polyedrischen Oberfläche aus. Gibt NULL zurück, wenn es sich nicht um polyedrische Geometrien handelt.

Synopsis

integer **ST_NumPatches**(geometry g1);

Beschreibung

Gibt die Anzahl der Maschen einer polyedrischen Oberfläche aus. Gibt NULL zurück, wenn es sich um keine polyedrische Geometrie handelt. Ist ein Synonym für ST_NumGeometries zur Unterstützung der MM Namensgebung. Wenn Ihnen die MM-Konvention egal ist, so ist die Verwendung von ST_NumGeometries schneller.

Verfügbarkeit: 2.0.0

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

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

This method implements the SQL/MM specification. SQL-MM ISO/IEC 13249-3: 8.5

This function supports Polyhedral surfaces.

Beispiele

```
SELECT ST_NumPatches(ST_GeomFromEWKT('POLYHEDRALSURFACE( ((0 0 0, 0 0 1, 0 1 1, 0 1 0, 0 0 ↩
    0)),
                ((0 0 0, 0 1 0, 1 1 0, 1 0 0, 0 0 0)), ((0 0 0, 1 0 0, 1 0 1, 0 0 1, 0 0 0) ↩
                    ),
                ((1 1 0, 1 1 1, 1 0 1, 1 0 0, 1 1 0)),
                ((0 1 0, 0 1 1, 1 1 1, 1 1 0, 0 1 0)), ((0 0 1, 1 0 1, 1 1 1, 0 1 1, 0 0 1) ↩
                    ) )'));
                --result
                6
```

Siehe auch

ST_GeomFromEWKT, ST_NumGeometries

7.4.33 ST_NumPoints

ST_NumPoints — Gibt die Anzahl der Stützpunkte eines ST_LineString oder eines ST_CircularString zurück.

Synopsis

integer **ST_NumPoints**(geometry g1);

Beschreibung

Gibt die Anzahl der Stützpunkte eines ST_LineString oder eines ST_CircularString zurück. Vor 1.4 funktionierte dies nur mit ST_LineString, wie von der Spezifikation festgelegt. Ab 1.4 aufwärts handelt es sich um einen Alias für ST_NPoints, das die Anzahl der Knoten nicht nur für Linienzüge ausgibt. Erwägen Sie stattdessen die Verwendung von ST_NPoints, das vielseitig ist und mit vielen Geometrietypen funktioniert.

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

This method implements the SQL/MM specification. SQL-MM 3: 7.2.4

Beispiele

```
SELECT ST_NumPoints(ST_GeomFromText('LINESTRING(77.29 29.07,77.42 29.26,77.27 29.31,77.29 ↩
    29.07)'));
                --result
                4
```

Siehe auch

ST_NPoints

7.4.34 ST_PatchN

ST_PatchN — Gibt den Geometrietyp des ST_Geometry Wertes zurück.

Synopsis

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

Beschreibung

>Gibt die auf 1-basierende n-te Geometrie (Masche) zurück, wenn es sich bei der Geometrie um ein POLYHEDRALSURFACE, oder ein POLYHEDRALSURFACEM handelt. Anderenfalls wird NULL zurückgegeben. Gibt bei polyedrischen Oberflächen das selbe Ergebnis wie ST_GeometryN. Die Verwendung von ST_GeometryN ist schneller.

Note
Der Index ist auf 1 basiert.

Note
Falls Sie alle Geometrien einer Geometrie entnehmen wollen, so ist ST_Dump wesentlich leistungsfähiger.

Verfügbarkeit: 2.0.0

This method implements the SQL/MM specification. SQL-MM ISO/IEC 13249-3: 8.5

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

This function supports Polyhedral surfaces.

Beispiele

```
--Extract the 2nd face of the polyhedral surface
SELECT ST_AsEWKT(ST_PatchN(geom, 2)) As geomewkt
FROM (
VALUES (ST_GeomFromEWKT('POLYHEDRALSURFACE( ((0 0 0, 0 0 1, 0 1 1, 0 1 0, 0 0 0)),
        ((0 0 0, 0 1 0, 1 1 0, 1 0 0, 0 0 0)), ((0 0 0, 1 0 0, 1 0 1, 0 0 1, 0 0 0)),
        ((1 1 0, 1 1 1, 1 0 1, 1 0 0, 1 1 0)),
        ((0 1 0, 0 1 1, 1 1 1, 1 1 0, 0 1 0)), ((0 0 1, 1 0 1, 1 1 1, 0 1 1, 0 0 1)) )')) ) ↩
            As foo(geom);

              geomewkt
---+---------------------------------------
 POLYGON((0 0 0,0 1 0,1 1 0,1 0 0,0 0 0))
```

Siehe auch

ST_AsEWKT, ST_GeomFromEWKT, ST_Dump, ST_GeometryN, ST_NumGeometries

7.4.35 ST_PointN

ST_PointN — Gibt die Anzahl der Stützpunkte eines ST_LineString oder eines ST_CircularString zurück.

Synopsis

geometry **ST_PointN**(geometry a_linestring, integer n);

Beschreibung

Gibt den n-ten Punkt des ersten LineString's oder des kreisförmigen LineStrings's einer Geometrie zurück. Negative Werte werden rückwärts, vom Ende des LineString's her gezählt, sodass -1 der Endpunkt ist. Gibt NULL aus, wenn die Geometrie keinen LineString enthält.

Note
Seit Version 0.8.0 ist der Index 1-basiert, so wie in der OGC Spezifikation. Rückwärtiges Indizieren (negativer Index) findet sich nicht in der OGC Spezifikation. Vorhergegangene Versionen waren 0-basiert.

Note
Falls Sie den n-ten Punkt eines jeden LineString's in einem MultiLinestring wollen, nutzen Sie diese Funktion gemeinsam mit ST_Dump.

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

This method implements the SQL/MM specification. SQL-MM 3: 7.2.5, 7.3.5

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

This method supports Circular Strings and Curves.

Note
Änderung: 2.0.0 arbeitet nicht mehr mit MultiLinestring's, die nur eine einzelne Geometrie enthalten. In früheren Versionen von PostGIS gab die Funktion bei einem, aus einer einzelnen Linie bestehender MultiLinestring, den Anfangspunkt zurück. Ab 2.0.0 wird, so wie bei jedem anderen MultiLinestring auch, NULL zurückgegeben.
Änderung: 2.3.0 : negatives Indizieren verfügbar (-1 entspricht dem Endpunkt)

Beispiele

```
-- Extract all POINTs from a LINESTRING
SELECT ST_AsText(
   ST_PointN(
          column1,
          generate_series(1, ST_NPoints(column1))
   ))
FROM ( VALUES ('LINESTRING(0 0, 1 1, 2 2)'::geometry) ) AS foo;

 st_astext
------------
 POINT(0 0)
 POINT(1 1)
 POINT(2 2)
(3 rows)

--Example circular string
```

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

 st_astext
------------
 POINT(3 2)
(1 row)

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

    st_astext
-----------------
 POINT Z (1 1 1)
(1 row)
```

Siehe auch

ST_NPoints

7.4.36 ST_Points

ST_Points — Gibt einen MultiPoint zurück, welcher alle Koordinaten einer Geometrie enthält.

Synopsis

geometry **ST_Points**(geometry geom);

Beschreibung

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

To obtain information about the position of each coordinate in the parent geometry use ST_DumpPoints.

M and Z coordinates are preserved if present.

This method supports Circular Strings and Curves.

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

Verfügbarkeit: 2.3.0

Beispiele

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

--result
MULTIPOINT Z ((30 10 4),(10 30 5),(40 40 6),(30 10 4))
```

Siehe auch

ST_RemoveRepeatedPoints, ST_PointN

7.4.37 ST_StartPoint

ST_StartPoint — Returns the first point of a LineString.

Synopsis

geometry **ST_StartPoint**(geometry geomA);

Beschreibung

Gibt den Anfangspunkt einer `LINESTRING` oder `CIRCULARLINESTRING` Geometrie als `POINT` oder `NULL` zurück, falls es sich beim Eingabewert nicht um einen `LINESTRING` oder `CIRCULARLINESTRING` handelt.

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

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

This method supports Circular Strings and Curves.

Note

Enhanced: 3.2.0 returns a point for all geometries. Prior behavior returns NULLs if input was not a LineString.
Änderung: 2.0.0 unterstützt die Verarbeitung von MultiLinestring's die nur aus einer einzelnen Geometrie bestehen, nicht mehr. In früheren Versionen von PostGIS gab die Funktion bei einem aus einer einzelnen Linie bestehender MultiLinestring den Anfangspunkt zurück. Ab 2.0.0 gibt sie nur NULL zurück, so wie bei jedem anderen MultiLinestring. Die alte Verhaltensweise war undokumentiert, aber Anwender, die annahmen, dass Sie Ihre Daten als LINESTRING vorliegen haben, könnten in 2.0 dieses zurückgegebene NULL bemerken.

Beispiele

Start point of a LineString

```
SELECT ST_AsText(ST_StartPoint('LINESTRING(0 1, 0 2)'::geometry));
 st_astext
------------
 POINT(0 1)
```

Start point of a non-LineString is NULL

```
SELECT ST_StartPoint('POINT(0 1)'::geometry) IS NULL AS is_null;
  is_null
----------
 t
```

Start point of a 3D LineString

```
SELECT ST_AsEWKT(ST_StartPoint('LINESTRING(0 1 1, 0 2 2)'::geometry));
 st_asewkt
------------
 POINT(0 1 1)
```

Gibt die Anzahl der Stützpunkte eines ST_LineString oder eines ST_CircularString zurück.

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

Siehe auch

ST_EndPoint, ST_PointN

7.4.38 ST_Summary

ST_Summary — Gibt eine Zusammenfassung des Inhalts einer Geometrie wieder.

Synopsis

text **ST_Summary**(geometry g);
text **ST_Summary**(geography g);

Beschreibung

Gibt eine Zusammenfassung des Inhalts einer Geometrie wieder.

Die Bedeutung der Flags, welche in eckigen Klammern hinter dem Geometrietyp angezeigt werden, ist wie folgt:

- M: besitzt eine M-Ordinate
- Z: besitzt eine Z-Ordinate
- B: besitzt ein zwischengespeichertes Umgebungsrechteck
- G: ist geodätisch (Geographie)
- S: besitzt ein räumliches Koordinatenreferenzsystem

This method supports Circular Strings and Curves.

This function supports Polyhedral surfaces.

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

Verfügbarkeit: 1.2.2

Erweiterung: 2.0.0 Unterstützung für geographische Koordinaten hinzugefügt

Erweiterung: 2.1.0 S-Flag, diese zeigt an ob das Koordinatenreferenzsystem bekannt ist

Erweiterung: 2.2.0 Unterstützung für TIN und Kurven

Beispiele

```
=# 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)
```

Siehe auch

PostGIS_DropBBox, PostGIS_AddBBox, ST_Force3DM, ST_Force3DZ, ST_Force2D, geography

ST_IsValid, ST_IsValid, ST_IsValidReason, ST_IsValidDetail

7.4.39 ST_X

ST_X — Returns the X coordinate of a Point.

Synopsis

float **ST_X**(geometry a_point);

Beschreibung

Gibt die X-Koordinate eines Punktes, oder NULL wenn diese nicht vorhanden ist, zurück. Die Eingabe muss ein Punkt sein.

Note
To get the minimum and maximum X value of geometry coordinates use the functions ST_XMin and ST_XMax.

This method implements the SQL/MM specification. SQL-MM 3: 6.1.3

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

Beispiele

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

SELECT ST_Y(ST_Centroid(ST_GeomFromEWKT('LINESTRING(1 2 3 4, 1 1 1 1)')));
 st_y
------
  1.5
(1 row)
```

Siehe auch

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

7.4.40 ST_Y

ST_Y — Returns the Y coordinate of a Point.

Synopsis

float **ST_Y**(geometry a_point);

Beschreibung

Gibt die Y-Koordinate eines Punktes, oder NULL wenn diese nicht vorhanden ist, zurück. Die Eingabe muss ein Punkt sein.

Note
To get the minimum and maximum Y value of geometry coordinates use the functions ST_YMin and ST_YMax.

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

This method implements the SQL/MM specification. SQL-MM 3: 6.1.4

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

Beispiele

```
SELECT ST_Y(ST_GeomFromEWKT('POINT(1 2 3 4)'));
 st_y
------
        2
(1 row)

SELECT ST_Y(ST_Centroid(ST_GeomFromEWKT('LINESTRING(1 2 3 4, 1 1 1 1)')));
 st_y
------
  1.5
(1 row)
```

Siehe auch

ST_Centroid, ST_GeomFromEWKT, ST_M, ST_X, ST_YMax, ST_YMin, ST_Z

7.4.41 ST_Z

ST_Z — Returns the Z coordinate of a Point.

Synopsis

float **ST_Z**(geometry a_point);

Beschreibung

Gibt die Z-Koordinate eines Punktes, oder NULL wenn diese nicht vorhanden ist, zurück. Die Eingabe muss ein Punkt sein.

Note
To get the minimum and maximum Z value of geometry coordinates use the functions ST_ZMin and ST_ZMax.

This method implements the SQL/MM specification.

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

Beispiele

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

Siehe auch

ST_GeomFromEWKT, ST_M, ST_X, ST_Y, ST_ZMax, ST_ZMin

7.4.42 ST_Zmflag

ST_Zmflag — Gibt die Dimension der Koordinaten von ST_Geometry zurück.

Synopsis

smallint **ST_Zmflag**(geometry geomA);

Beschreibung

Gibt die Dimension der Koordinaten für den Wert von ST_Geometry zurück.

Values are: 0 = 2D, 1 = 3D-M, 2 = 3D-Z, 3 = 4D.

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

This method supports Circular Strings and Curves.

Beispiele

```
SELECT ST_Zmflag(ST_GeomFromEWKT('LINESTRING(1 2, 3 4)'));
 st_zmflag
-----------
                 0

SELECT ST_Zmflag(ST_GeomFromEWKT('LINESTRINGM(1 2 3, 3 4 3)'));
 st_zmflag
-----------
                 1

SELECT ST_Zmflag(ST_GeomFromEWKT('CIRCULARSTRING(1 2 3, 3 4 3, 5 6 3)'));
 st_zmflag
-----------
                 2
SELECT ST_Zmflag(ST_GeomFromEWKT('POINT(1 2 3 4)'));
 st_zmflag
-----------
                 3
```

Siehe auch

ST_CoordDim, ST_NDims, ST_Dimension

7.5 Geometrische Editoren

7.5.1 ST_AddPoint

ST_AddPoint — Fügt einem Linienzug einen Punkt hinzu.

Synopsis

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

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

Beschreibung

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

Verfügbarkeit: 1.1.0

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

Beispiele

Add a point to the end of a 3D line

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

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

Guarantee all lines in a table are closed by adding the start point of each line to the end of the line only for those that are not closed.

```
UPDATE sometable
SET geom = ST_AddPoint(geom, ST_StartPoint(geom))
FROM sometable
WHERE ST_IsClosed(geom) = false;
```

Siehe auch

ST_RemovePoint, ST_SetPoint

7.5.2 ST_CollectionExtract

ST_CollectionExtract — Given a geometry collection, returns a multi-geometry containing only elements of a specified type.

Synopsis

geometry **ST_CollectionExtract**(geometry collection);

geometry **ST_CollectionExtract**(geometry collection, integer type);

Beschreibung

Given a geometry collection, returns a homogeneous multi-geometry.

If the *type* is not specified, returns a multi-geometry containing only geometries of the highest dimension. So polygons are preferred over lines, which are preferred over points.

If the *type* is specified, returns a multi-geometry containing only that type. If there are no sub-geometries of the right type, an EMPTY geometry is returned. Only points, lines and polygons are supported. The type numbers are:

- 1 == POINT
- 2 == LINESTRING
- 3 == POLYGON

For atomic geometry inputs, the geometry is retured unchanged if the input type matches the requested type. Otherwise, the result is an EMPTY geometry of the specified type. If required, these can be converted to multi-geometries using ST_Multi.

Warning
MultiPolygon results are not checked for validity. If the polygon components are adjacent or overlapping the result will be invalid. (For example, this can occur when applying this function to an ST_Split result.) This situation can be checked with ST_IsValid and repaired with ST_MakeValid.

Verfügbarkeit: 1.5.0

Note
Prior to 1.5.3 this function returned atomic inputs unchanged, no matter type. In 1.5.3 non-matching single geometries returned a NULL result. In 2.0.0 non-matching single geometries return an EMPTY result of the requested type.

Beispiele

Extract highest-dimension type:

```
SELECT ST_AsText(ST_CollectionExtract(
        'GEOMETRYCOLLECTION( POINT(0 0), LINESTRING(1 1, 2 2) )'));
    st_astext
    ---------------
    MULTILINESTRING((1 1, 2 2))
```

Extract points (type 1 == POINT):

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

Extract lines (type 2 == LINESTRING):

```
SELECT ST_AsText(ST_CollectionExtract(
        'GEOMETRYCOLLECTION(GEOMETRYCOLLECTION(LINESTRING(0 0, 1 1)),LINESTRING(2 2, 3 3)) ↩
            ',
        2 ));
    st_astext
    ---------------
    MULTILINESTRING((0 0, 1 1), (2 2, 3 3))
```

Siehe auch

ST_CollectionHomogenize, ST_Multi, ST_IsValid, ST_MakeValid

7.5.3 ST_CollectionHomogenize

ST_CollectionHomogenize — Returns the simplest representation of a geometry collection.

Synopsis

geometry **ST_CollectionHomogenize**(geometry collection);

Beschreibung

Given a geometry collection, returns the "simplest" representation of the contents.

- Homogeneous (uniform) collections are returned as the appropriate multi-geometry.
- Heterogeneous (mixed) collections are flattened into a single GeometryCollection.
- Collections containing a single atomic element are returned as that element.
- Atomic geometries are returned unchanged. If required, these can be converted to a multi-geometry using ST_Multi.

Warning
This function does not ensure that the result is valid. In particular, a collection containing adjacent or overlapping Polygons will create an invalid MultiPolygon. This situation can be checked with ST_IsValid and repaired with ST_MakeValid.

Verfügbarkeit: 2.0.0

Beispiele

Single-element collection converted to an atomic geometry

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

        st_astext
        ------------
        POINT(0 0)
```

Nested single-element collection converted to an atomic geometry:

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

        st_astext
        ------------
        POINT(0 0)
```

Collection converted to a multi-geometry:

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

        st_astext
        ---------------------
        MULTIPOINT((0 0),(1 1))
```

Nested heterogeneous collection flattened to a GeometryCollection:

```
SELECT ST_AsText(ST_CollectionHomogenize('GEOMETRYCOLLECTION(POINT(0 0), GEOMETRYCOLLECTION ↩
    ( LINESTRING(1 1, 2 2)))'));

        st_astext
        ---------------------
        GEOMETRYCOLLECTION(POINT(0 0),LINESTRING(1 1,2 2))
```

Collection of Polygons converted to an (invalid) MultiPolygon:

```
SELECT ST_AsText(ST_CollectionHomogenize('GEOMETRYCOLLECTION (POLYGON ((10 50, 50 50, 50  ↩
    10, 10 10, 10 50)), POLYGON ((90 50, 90 10, 50 10, 50 50, 90 50)))'));

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

Siehe auch

ST_CollectionExtract, ST_Multi, ST_IsValid, ST_MakeValid

7.5.4 ST_CurveToLine

ST_CurveToLine — Converts a geometry containing curves to a linear geometry.

Synopsis

geometry **ST_CurveToLine**(geometry curveGeom, float tolerance, integer tolerance_type, integer flags);

Beschreibung

Konvertiert einen CIRCULAR STRING in einen normalen LINESTRING, ein CURVEPOLYGON in ein POLYGON und ein MULTISURFACE in ein MULTIPOLYGON. Ist nützlich zur Ausgabe an Geräten, die keine Kreisbögen unterstützen.

Wandelt eine gegebene Geometrie in eine lineare Geometrie um. Jede Kurvengeometrie und jedes Kurvensegment wird in linearer Näherung mit der gegebenen Toleranz und Optionen konvertiert (Standardmäßig 32 Segmenten pro Viertelkreis und keine Optionen).

Der Übergabewert 'tolerance_type' gibt den Toleranztyp an. Er kann die folgenden Werte annehmen:

- 0 (default): die Toleranz wird über die maximale Anzahl der Segmente pro Viertelkreis angegeben.
- 1: Die Toleranz wird als maximale Abweichung der Linie von der Kurve in der Einheit der Herkunftsdaten angegeben.
- 2: Die Toleranz entspricht dem maximalen Winkel zwischen zwei erzeugten Radien.

Der Parameter 'flags' ist ein Bitfeld mit dem Standardwert 0. Es werden folgende Bits unterstützt:

- 1: Symmetrische (orientierungsunabhängige) Ausgabe.
- 2: Erhält den Winkel, vermeidet die Winkel (Segmentlängen) bei der symmetrischen Ausgabe zu reduzieren. Hat keine Auswirkung, wenn die Symmetrie-Flag nicht aktiviert ist.

Verfügbarkeit: 1.3.0

Erweiterung: ab 2.4.0 kann die Toleranz über die 'maximale Abweichung' und den 'maximalen Winkel' angegeben werden. Die symmetrische Ausgabe wurde hinzugefügt.

Erweiterung: 3.0.0 führte eine minimale Anzahl an Segmenten pro linearisierten Bogen ein, um einem topologischen Kollaps vorzubeugen.

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

This method implements the SQL/MM specification. SQL-MM 3: 7.1.7

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

This method supports Circular Strings and Curves.

Beispiele

```
SELECT ST_AsText(ST_CurveToLine(ST_GeomFromText('CIRCULARSTRING(220268 150415,220227  ↩
    150505,220227 150406)')));

--Result --
 LINESTRING(220268 150415,220269.95064912 150416.539364228,220271.823415575  ↩
     150418.17258804,220273.613787707 150419.895736857,
 220275.317452352 150421.704659462,220276.930305234 150423.594998003,220278.448460847  ↩
     150425.562198489,
 220279.868261823 150427.60152176,220281.186287736 150429.708054909,220282.399363347  ↩
     150431.876723113,
 220283.50456625 150434.10230186,220284.499233914 150436.379429536,220285.380970099  ↩
     150438.702620341,220286.147650624 150441.066277505,
 220286.797428488 150443.464706771,220287.328738321 150445.892130112,220287.740300149  ↩
     150448.342699654,
 220288.031122486 150450.810511759,220288.200504713 150453.289621251,220288.248038775  ↩
     150455.77405574,
 220288.173610157 150458.257830005,220287.977398166 150460.734960415,220287.659875492  ↩
     150463.199479347,
```

```
 220287.221807076 150465.64544956,220286.664248262 150468.066978495,220285.988542259  ↩
     150470.458232479,220285.196316903 150472.81345077,
 220284.289480732 150475.126959442,220283.270218395 150477.39318505,220282.140985384  ↩
     150479.606668057,
 220280.90450212 150481.762075989,220279.5637474 150483.85421628,220278.12195122  ↩
     150485.87804878,
 220276.582586992 150487.828697901,220274.949363179 150489.701464356,220273.226214362  ↩
     150491.491836488,
 220271.417291757 150493.195501133,220269.526953216 150494.808354014,220267.559752731  ↩
     150496.326509628,
 220265.520429459 150497.746310603,220263.41389631 150499.064336517,220261.245228106  ↩
     150500.277412127,
 220259.019649359 150501.38261503,220256.742521683 150502.377282695,220254.419330878  ↩
     150503.259018879,
 220252.055673714 150504.025699404,220249.657244448 150504.675477269,220247.229821107  ↩
     150505.206787101,
 220244.779251566 150505.61834893,220242.311439461 150505.909171266,220239.832329968  ↩
     150506.078553494,
 220237.347895479 150506.126087555,220234.864121215 150506.051658938,220232.386990804  ↩
     150505.855446946,
 220229.922471872 150505.537924272,220227.47650166 150505.099855856,220225.054972724  ↩
     150504.542297043,
 220222.663718741 150503.86659104,220220.308500449 150503.074365683,
 220217.994991777 150502.167529512,220215.72876617 150501.148267175,
 220213.515283163 150500.019034164,220211.35987523 150498.7825509,
 220209.267734939 150497.441796181,220207.243902439 150496,
 220205.293253319 150494.460635772,220203.420486864 150492.82741196,220201.630114732  ↩
     150491.104263143,
 220199.926450087 150489.295340538,220198.313597205 150487.405001997,220196.795441592  ↩
     150485.437801511,
 220195.375640616 150483.39847824,220194.057614703 150481.291945091,220192.844539092  ↩
     150479.123276887,220191.739336189 150476.89769814,
 220190.744668525 150474.620570464,220189.86293234 150472.297379659,220189.096251815  ↩
     150469.933722495,
 220188.446473951 150467.535293229,220187.915164118 150465.107869888,220187.50360229  ↩
     150462.657300346,
 220187.212779953 150460.189488241,220187.043397726 150457.710378749,220186.995863664  ↩
     150455.22594426,
 220187.070292282 150452.742169995,220187.266504273 150450.265039585,220187.584026947  ↩
     150447.800520653,
 220188.022095363 150445.35455044,220188.579654177 150442.933021505,220189.25536018  ↩
     150440.541767521,
 220190.047585536 150438.18654923,220190.954421707 150435.873040558,220191.973684044  ↩
     150433.60681495,
 220193.102917055 150431.393331943,220194.339400319 150429.237924011,220195.680155039  ↩
     150427.14578372,220197.12195122 150425.12195122,
 220198.661315447 150423.171302099,220200.29453926 150421.298535644,220202.017688077  ↩
     150419.508163512,220203.826610682 150417.804498867,
 220205.716949223 150416.191645986,220207.684149708 150414.673490372,220209.72347298  ↩
     150413.253689397,220211.830006129 150411.935663483,
 220213.998674333 150410.722587873,220216.22425308 150409.61738497,220218.501380756  ↩
     150408.622717305,220220.824571561 150407.740981121,
 220223.188228725 150406.974300596,220225.586657991 150406.324522731,220227 150406)

--3d example
SELECT ST_AsEWKT(ST_CurveToLine(ST_GeomFromEWKT('CIRCULARSTRING(220268 150415 1,220227  ↩
    150505 2,220227 150406 3)')));
Output
------
 LINESTRING(220268 150415 1,220269.95064912 150416.539364228 1.0181172856673,
 220271.823415575 150418.17258804 1.03623457133459,220273.613787707 150419.895736857  ↩
     1.05435185700189,....AD INFINITUM ....
```

```
    220225.586657991 150406.324522731 1.32611114201132,220227 150406 3)

--use only 2 segments to approximate quarter circle
SELECT ST_AsText(ST_CurveToLine(ST_GeomFromText('CIRCULARSTRING(220268 150415,220227  ↩
    150505,220227 150406)'),2));
st_astext
------------------------------
 LINESTRING(220268 150415,220287.740300149 150448.342699654,220278.12195122  ↩
     150485.87804878,
 220244.779251566 150505.61834893,220207.243902439 150496,220187.50360229 150462.657300346,
 220197.12195122 150425.12195122,220227 150406)

-- Ensure approximated line is no further than 20 units away from
-- original curve, and make the result direction-neutral
SELECT ST_AsText(ST_CurveToLine(
 'CIRCULARSTRING(0 0,100 -100,200 0)'::geometry,
    20, -- Tolerance
    1, -- Above is max distance between curve and line
    1  -- Symmetric flag
));
st_astext
------------------------------------------------------------------------------------------------
 LINESTRING(0 0,50 -86.6025403784438,150 -86.6025403784439,200 -1.1331077795296e-13,200 0)
```

Siehe auch

ST_LineToCurve

7.5.5 ST_Scroll

ST_Scroll — Change start point of a closed LineString.

Synopsis

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

Beschreibung

Changes the start/end point of a closed LineString to the given vertex *point*.

Availability: 3.2.0

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

This function supports M coordinates.

Beispiele

Make e closed line start at its 3rd vertex

```
SELECT ST_AsEWKT(ST_Scroll('SRID=4326;LINESTRING(0 0 0 1, 10 0 2 0, 5 5 4 2,0 0 0 1)', ' ↩
    POINT(5 5 4 2)'));

st_asewkt
```

```
----------
SRID=4326;LINESTRING(5 5 4 2,0 0 0 1,10 0 2 0,5 5 4 2)
```

Siehe auch

ST_Normalize

7.5.6 ST_FlipCoordinates

ST_FlipCoordinates — Returns a version of a geometry with X and Y axis flipped.

Synopsis

geometry **ST_FlipCoordinates**(geometry geom);

Beschreibung

Returns a version of the given geometry with X and Y axis flipped. Useful for fixing geometries which contain coordinates expressed as latitude/longitude (Y,X).

Verfügbarkeit: 2.0.0

This method supports Circular Strings and Curves.

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

This function supports M coordinates.

This function supports Polyhedral surfaces.

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

Beispiel

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

Siehe auch

ST_SwapOrdinates

7.5.7 ST_Force2D

ST_Force2D — Die Geometrien in einen "2-dimensionalen Modus" zwingen.

Synopsis

geometry **ST_Force2D**(geometry geomA);

Beschreibung

Zwingt die Geometrien in einen "2-dimensionalen Modus", sodass in der Ausgabe nur die X- und Y-Koordinaten dargestellt werden. Nützlich um eine OGC-konforme Ausgabe zu erhalten (da OGC nur 2-D Geometrien spezifiziert).

Erweiterung: Mit 2.0.0 wurde die Unterstützung für polyedrische Oberflächen eingeführt.

Änderung: 2.1.0. Bis zu 2.0.x wurde diese Funktion mit ST_Force_2D bezeichnet.

This method supports Circular Strings and Curves.

This function supports Polyhedral surfaces.

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

Beispiele

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

SELECT  ST_AsEWKT(ST_Force2D('POLYGON((0 0 2,0 5 2,5 0 2,0 0 2),(1 1 2,3 1 2,1 3 2,1 1 2)) ↩
   '));

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

Siehe auch

ST_Force3D

7.5.8 ST_Force3D

ST_Force3D — Zwingt die Geometrien in einen XYZ Modus. Dies ist ein Alias für ST_Force3DZ.

Synopsis

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

Beschreibung

Forces the geometries into XYZ mode. This is an alias for ST_Force3DZ. If a geometry has no Z component, then a `Zvalue` Z coordinate is tacked on.

Erweiterung: Mit 2.0.0 wurde die Unterstützung für polyedrische Oberflächen eingeführt.

Änderung: 2.1.0. Bis zu 2.0.x wurde diese Funktion mit ST_Force_3D bezeichnet.

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

This function supports Polyhedral surfaces.

This method supports Circular Strings and Curves.

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

Beispiele

```
--Nothing happens to an already 3D geometry
                SELECT ST_AsEWKT(ST_Force3D(ST_GeomFromEWKT('CIRCULARSTRING(1 1 2, 2 3 2, 4 ↩
                    5 2, 6 7 2, 5 6 2)')));
                                    st_asewkt
-----------------------------------------------
 CIRCULARSTRING(1 1 2,2 3 2,4 5 2,6 7 2,5 6 2)


SELECT  ST_AsEWKT(ST_Force3D('POLYGON((0 0,0 5,5 0,0 0),(1 1,3 1,1 3,1 1))'));

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

Siehe auch

ST_AsEWKT, ST_Force2D, ST_Force3DM, ST_Force3DZ

7.5.9 ST_Force3DZ

ST_Force3DZ — Zwingt die Geometrien in einen XYZ Modus.

Synopsis

geometry **ST_Force3DZ**(geometry geomA, float Zvalue = 0.0);

Beschreibung

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

Erweiterung: Mit 2.0.0 wurde die Unterstützung für polyedrische Oberflächen eingeführt.

Änderung: 2.1.0. Bis zu 2.0.x wurde diese Funktion mit ST_Force_3DZ bezeichnet.

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

This function supports Polyhedral surfaces.

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

This method supports Circular Strings and Curves.

Beispiele

```
--Nothing happens to an already 3D geometry
SELECT ST_AsEWKT(ST_Force3DZ(ST_GeomFromEWKT('CIRCULARSTRING(1 1 2, 2 3 2, 4 5 2, 6 7 2, 5  ↩
    6 2)')));
                                    st_asewkt
-----------------------------------------------
 CIRCULARSTRING(1 1 2,2 3 2,4 5 2,6 7 2,5 6 2)


SELECT  ST_AsEWKT(ST_Force3DZ('POLYGON((0 0,0 5,5 0,0 0),(1 1,3 1,1 3,1 1))'));
```

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

Siehe auch

ST_AsEWKT, ST_Force2D, ST_Force3DM, ST_Force3D

7.5.10 ST_Force3DM

ST_Force3DM — Zwingt die Geometrien in einen XYM Modus.

Synopsis

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

Beschreibung

Forces the geometries into XYM mode. If a geometry has no M component, then a *Mvalue* M coordinate is tacked on. If it has a Z component, then Z is removed

Änderung: 2.1.0. Bis zu 2.0.x wurde diese Funktion mit ST_Force_3DM bezeichnet.

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

This method supports Circular Strings and Curves.

Beispiele

```
--Nothing happens to an already 3D geometry
SELECT ST_AsEWKT(ST_Force3DM(ST_GeomFromEWKT('CIRCULARSTRING(1 1 2, 2 3 2, 4 5 2, 6 7 2, 5  ↩
    6 2)')));
                                    st_asewkt
-----------------------------------------------
 CIRCULARSTRINGM(1 1 0,2 3 0,4 5 0,6 7 0,5 6 0)


SELECT  ST_AsEWKT(ST_Force3DM('POLYGON((0 0 1,0 5 1,5 0 1,0 0 1),(1 1 1,3 1 1,1 3 1,1 1 1)) ↩
    '));

                                                    st_asewkt
--------------------------------------------------------------
 POLYGONM((0 0 0,0 5 0,5 0 0,0 0 0),(1 1 0,3 1 0,1 3 0,1 1 0))
```

Siehe auch

ST_AsEWKT, ST_Force2D, ST_Force3DM, ST_Force3D, ST_GeomFromEWKT

7.5.11 ST_Force4D

ST_Force4D — Zwingt die Geometrien in einen XYZM Modus.

Synopsis

geometry **ST_Force4D**(geometry geomA, float Zvalue = 0.0, float Mvalue = 0.0);

Beschreibung

Forces the geometries into XYZM mode. *Zvalue* and *Mvalue* is tacked on for missing Z and M dimensions, respectively.

Änderung: 2.1.0. Bis zu 2.0.x wurde diese Funktion mit ST_Force_4D bezeichnet.

Changed: 3.1.0. Added support for supplying non-zero Z and M values.

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

This method supports Circular Strings and Curves.

Beispiele

```
--Nothing happens to an already 3D geometry
SELECT ST_AsEWKT(ST_Force4D(ST_GeomFromEWKT('CIRCULARSTRING(1 1 2, 2 3 2, 4 5 2, 6 7 2, 5 6 ↩
    2)')));
                                                st_asewkt
----------------------------------------------------------
 CIRCULARSTRING(1 1 2 0,2 3 2 0,4 5 2 0,6 7 2 0,5 6 2 0)



SELECT  ST_AsEWKT(ST_Force4D('MULTILINESTRINGM((0 0 1,0 5 2,5 0 3,0 0 4),(1 1 1,3 1 1,1 3  ↩
   1,1 1 1))'));

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

Siehe auch

ST_AsEWKT, ST_Force2D, ST_Force3DM, ST_Force3D

7.5.12 ST_ForcePolygonCCW

ST_ForcePolygonCCW — Richtet alle äußeren Ringe gegen den Uhrzeigersinn und alle inneren Ringe mit dem Uhrzeigersinn aus.

Synopsis

geometry **ST_ForcePolygonCCW** (geometry geom);

Beschreibung

Zwingt (Multi)Polygone, den äusseren Ring gegen den Uhrzeigersinn und die inneren Ringe im Uhrzeigersinn zu orientieren. Andere Geometrien werden unverändert zurückgegeben.

Verfügbarkeit: 2.4.0

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

This function supports M coordinates.

Siehe auch

ST_ForcePolygonCW , ST_IsPolygonCCW , ST_IsPolygonCW

7.5.13 ST_ForceCollection

ST_ForceCollection — Wandelt eine Geometrie in eine GEOMETRYCOLLECTION um.

Synopsis

geometry **ST_ForceCollection**(geometry geomA);

Beschreibung

Wandelt eine Geometrie in eine GEOMETRYCOLLECTION um. Nützlich um eine WKB-Darstellung zu vereinfachen.

Erweiterung: Mit 2.0.0 wurde die Unterstützung für polyedrische Oberflächen eingeführt.

Verfügbarkeit: 1.2.2, Vor 1.3.4 ist diese Funktion bei CURVES abgestürzt. Dies wurde mit 1.3.4+ behoben

Änderung: 2.1.0. Bis zu 2.0.x wurde diese Funktion mit ST_Force_Collection bezeichnet.

This function supports Polyhedral surfaces.

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

This method supports Circular Strings and Curves.

Beispiele

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

                                                                                st_asewkt
-------------------------------------------------------------------------------------------
 GEOMETRYCOLLECTION(POLYGON((0 0 1,0 5 1,5 0 1,0 0 1),(1 1 1,3 1 1,1 3 1,1 1 1)))


  SELECT ST_AsText(ST_ForceCollection('CIRCULARSTRING(220227 150406,2220227 150407,220227  ↩
     150406)'));
                                                                                st_astext
-------------------------------------------------------------------------------------------
 GEOMETRYCOLLECTION(CIRCULARSTRING(220227 150406,2220227 150407,220227 150406))
(1 row)
```

```
-- POLYHEDRAL example --
SELECT ST_AsEWKT(ST_ForceCollection('POLYHEDRALSURFACE(((0 0 0,0 0 1,0 1 1,0 1 0,0 0 0)),
 ((0 0 0,0 1 0,1 1 0,1 0 0,0 0 0)),
 ((0 0 0,1 0 0,1 0 1,0 0 1,0 0 0)),
 ((1 1 0,1 1 1,1 0 1,1 0 0,1 1 0)),
 ((0 1 0,0 1 1,1 1 1,1 1 0,0 1 0)),
 ((0 0 1,1 0 1,1 1 1,0 1 1,0 0 1)))'))

                                                                                st_asewkt
-------------------------------------------------------------------------------------------
GEOMETRYCOLLECTION(
```

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

Siehe auch

ST_AsEWKT, ST_Force2D, ST_Force3DM, ST_Force3D, ST_GeomFromEWKT

7.5.14 ST_ForcePolygonCW

ST_ForcePolygonCW — Richtet alle äußeren Ringe im Uhrzeigersinn und alle inneren Ringe gegen den Uhrzeigersinn aus.

Synopsis

geometry **ST_ForcePolygonCW** (geometry geom);

Beschreibung

Zwingt (Multi)Polygone, den äusseren Ring im Uhrzeigersinn und die inneren Ringe gegen den Uhrzeigersinn zu orientieren. Andere Geometrien werden unverändert zurückgegeben.

Verfügbarkeit: 2.4.0

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

This function supports M coordinates.

Siehe auch

ST_ForcePolygonCCW , ST_IsPolygonCCW , ST_IsPolygonCW

7.5.15 ST_ForceSFS

ST_ForceSFS — Erzwingt, dass Geometrien nur vom Typ SFS 1.1 sind.

Synopsis

geometry **ST_ForceSFS**(geometry geomA);
geometry **ST_ForceSFS**(geometry geomA, text version);

Beschreibung

This function supports Polyhedral surfaces.

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

This method supports Circular Strings and Curves.

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

7.5.16 ST_ForceRHR

ST_ForceRHR — Orientiert die Knoten in einem Polygon so, dass sie der Drei-Finger-Regel folgen.

Synopsis

geometry **ST_ForceRHR**(geometry g);

Beschreibung

Orientiert die Knoten in einem Polygon so, dass sie der Drei-Finger-Regel folgen. Dadurch kommt die durch das Polygon begrenzte Fläche auf der rechten Seite der Begrenzung zu liegen. Insbesondere sind der äussere Ring im Uhrzeigersinn und die inneren Ringe gegen den Uhrzeigersinn orientiert. Diese Funktion ist ein Synonym für ST_ForcePolygonCW

Note
Die obere Definition mit der Drei-Finger-Regel widerspricht den Definitionen, die in anderen Zusammenhängen verwendet werden. Um Verwirrung zu vermeiden, wird die Verwendung von ST_ForcePolygonCW empfohlen.

Erweiterung: Mit 2.0.0 wurde die Unterstützung für polyedrische Oberflächen eingeführt.

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

This function supports Polyhedral surfaces.

Beispiele

```
SELECT ST_AsEWKT(
  ST_ForceRHR(
        'POLYGON((0 0 2, 5 0 2, 0 5 2, 0 0 2),(1 1 2, 1 3 2, 3 1 2, 1 1 2))'
  )
);
                                                    st_asewkt
--------------------------------------------------------------
 POLYGON((0 0 2,0 5 2,5 0 2,0 0 2),(1 1 2,3 1 2,1 3 2,1 1 2))
(1 row)
```

Siehe auch

ST_ForcePolygonCCW , ST_ForcePolygonCW , ST_IsPolygonCCW , ST_IsPolygonCW , ST_BuildArea, ST_Polygonize, ST_Reverse

7.5.17 ST_ForceCurve

ST_ForceCurve — Wandelt einen geometrischen in einen Kurven Datentyp um, soweit anwendbar.

Synopsis

geometry **ST_ForceCurve**(geometry g);

Beschreibung

Wandelt eine Geometrie in eine Kurvendarstellung um, soweit anwendbar: Linien werden CompundCurves, MultiLines werden MultiCurves, Polygone werden zu CurvePolygons, Multipolygons werden MultiSurfaces. Wenn die Geometrie bereits in Kurvendarstellung vorliegt, wird sie unverändert zurückgegeben.

Verfügbarkeit: 2.2.0

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

This method supports Circular Strings and Curves.

Beispiele

```
SELECT ST_AsText(
  ST_ForceCurve(
        'POLYGON((0 0 2, 5 0 2, 0 5 2, 0 0 2),(1 1 2, 1 3 2, 3 1 2, 1 1 2))'::geometry
  )
);
                               st_astext
---------------------------------------------------------------------
 CURVEPOLYGON Z ((0 0 2,5 0 2,0 5 2,0 0 2),(1 1 2,1 3 2,3 1 2,1 1 2))
(1 row)
```

Siehe auch

ST_LineToCurve

7.5.18 ST_LineToCurve

ST_LineToCurve — Converts a linear geometry to a curved geometry.

Synopsis

geometry **ST_LineToCurve**(geometry geomANoncircular);

Beschreibung

Wandelt einen einfachen LineString/Polygon in Kreisbögen/CIRCULARSTRINGs und Kurvenpolygone um. Beachten Sie, dass wesentlich weniger Punkte zur Beschreibung des Kurvenäquivalents benötigt werden.

Note
Wenn der gegebene LINESTRING/POLYGON nicht genug gekrümmt ist um eine deutliche Kurve zu repräsentieren, wird die Geometrie von der Funktion unverändert zurückgegeben.

Verfügbarkeit: 1.3.0

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

This method supports Circular Strings and Curves.

Beispiele

```
-- 2D Example
SELECT ST_AsText(ST_LineToCurve(foo.geom)) As curvedastext,ST_AsText(foo.geom) As  ↩
    non_curvedastext
    FROM (SELECT ST_Buffer('POINT(1 3)'::geometry, 3) As geom) As foo;

curvedatext                                                                  non_curvedastext
-----------------------------------------------------------------------------|------------------------------

CURVEPOLYGON(CIRCULARSTRING(4 3,3.12132034355964 0.878679656440359, | POLYGON((4  ↩
    3,3.94235584120969 2.41472903395162,3.77163859753386 1.85194970290473,
1 0,-1.12132034355965 5.12132034355963,4 3))                          |  3.49440883690764  ↩
    1.33328930094119,3.12132034355964 0.878679656440359,
                                                                      |  2.66671069905881  ↩
                                                                          0.505591163092366,2.14805029
                                                                           0.228361402466141,
                                                                      |  1.58527096604839  ↩
                                                                          0.0576441587903094,1 ↩
                                                                           0,
                                                                      |  0.414729033951621  ↩
                                                                          0.0576441587903077,-0.148050
                                                                           0.228361402466137,
                                                                      |  -0.666710699058802  ↩
                                                                          0.505591163092361,-1.121320
                                                                           0.878679656440353,
                                                                      |  -1.49440883690763  ↩
                                                                          1.33328930094119,-1.771638
                                                                           1.85194970290472
                                                                      |  --ETC--  ↩
                                                                          ,3.94235584120969  ↩
                                                                          3.58527096604839,4  ↩
                                                                          3))

--3D example
SELECT ST_AsText(ST_LineToCurve(geom)) As curved, ST_AsText(geom) AS not_curved
FROM (SELECT ST_Translate(ST_Force3D(ST_Boundary(ST_Buffer(ST_Point(1,3), 2,2))),0,0,3) AS  ↩
    geom) AS foo;

                        curved                          |               not_curved
--------------------------------------------------------+-----------------------------------------

 CIRCULARSTRING Z (3 3 3,-1 2.99999999999999 3,3 3 3) | LINESTRING Z (3 3 3,2.4142135623731 ↩
     1.58578643762691 3,1 1 3,
                                                      | -0.414213562373092 1.5857864376269  ↩
                                                          3,-1 2.99999999999999 3,
                                                      | -0.414213562373101 4.41421356237309 ↩
                                                           3,
                                                      | 0.999999999999991 5  ↩
                                                          3,2.41421356237309 4.4142135623731 ↩
                                                           3,3 3 3)
(1 row)
```

Siehe auch

ST_CurveToLine

7.5.19 ST_Multi

ST_Multi — Gibt die Geometrie als MULTI* Geometrie zurück.

Synopsis

geometry **ST_Multi**(geometry geom);

Beschreibung

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

Beispiele

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

Siehe auch

ST_AsText

7.5.20 ST_LineExtend

ST_LineExtend — Returns a line with the last and first segments extended the specified distance(s).

Synopsis

geometry **ST_LineExtend**(geometry line, float distance_forward, float distance_backward=0.0);

Beschreibung

Returns a line with the last and first segments extended the specified distance(s). Distance of zero carries out no extension. Only non-negative distances are allowed. The first (and last) two distinct points in a line are used to determine the direction of projection, duplicate points are ignored.

Availability: 3.4.0

Example: Projected point at 100,000 meters and bearing 45 degrees

```
SELECT ST_AsText(ST_Project('POINT(0 0)'::geography, 100000, radians(45.0)));
-----------------------------------------
 POINT(0.635231029125537 0.639472334729198)
```

Siehe auch

ST_LocateAlong, ST_Project

7.5.21 ST_Normalize

ST_Normalize — Gibt die Geometrie in Normalform zurück.

Synopsis

geometry **ST_Normalize**(geometry geom);

Beschreibung

Gibt die Geometrie in Normalform aus. Möglicherweise werden die Knoten der Polygonringe, die Ringe eines Polygons oder die Elemente eines Komplexes von Mehrfachgeometrien neu gereiht.

Hauptsächlich für Testzwecke sinnvoll (zum Vergleich von erwarteten und erhaltenen Ergebnissen).

Verfügbarkeit: 2.3.0

Beispiele

```
SELECT ST_AsText(ST_Normalize(ST_GeomFromText(
  'GEOMETRYCOLLECTION(
    POINT(2 3),
    MULTILINESTRING((0 0, 1 1),(2 2, 3 3)),
    POLYGON(
      (0 10,0 0,10 0,10 10,0 10),
      (4 2,2 2,2 4,4 4,4 2),
      (6 8,8 8,8 6,6 6,6 8)
    )
  )'
)));
                                                                st_astext
-----------------------------------------------------------------------------------------------------

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

Siehe auch

ST_Equals,

7.5.22 ST_Project

ST_Project — Returns a point projected from a start point by a distance and bearing (azimuth).

Synopsis

geometry **ST_Project**(geometry g1, float distance, float azimuth);
geometry **ST_Project**(geometry g1, geometry g2, float distance);
geography **ST_Project**(geography g1, float distance, float azimuth);
geography **ST_Project**(geography g1, geography g2, float distance);

Beschreibung

Returns a point projected from a point along a geodesic using a given distance and azimuth (bearing). This is known as the direct geodesic problem.

The two-point version uses the path from the first to the second point to implicitly define the azimuth and uses the distance as before.

The distance is given in meters. Negative values are supported.

The azimuth (also known as heading or bearing) is given in radians. It is measured clockwise from true north.

- North is azimuth zero (0 degrees)
- East is azimuth $\pi/2$ (90 degrees)
- South is azimuth π (180 degrees)
- West is azimuth $3\pi/2$ (270 degrees)

Negative azimuth values and values greater than 2π (360 degrees) are supported.

Verfügbarkeit: 2.0.0

Enhanced: 2.4.0 Allow negative distance and non-normalized azimuth.

Enhanced: 3.4.0 Allow geometry arguments and two-point form omitting azimuth.

Example: Projected point at 100,000 meters and bearing 45 degrees

```
SELECT ST_AsText(ST_Project('POINT(0 0)'::geography, 100000, radians(45.0)));
------------------------------------------
 POINT(0.635231029125537 0.639472334729198)
```

Siehe auch

ST_Azimuth, ST_Distance, PostgreSQL function radians()

7.5.23 ST_QuantizeCoordinates

ST_QuantizeCoordinates — Setzt die niedrigwertigsten Bits der Koordinaten auf Null

Synopsis

geometry **ST_QuantizeCoordinates** (geometry g , int prec_x , int prec_y , int prec_z , int prec_m);

Beschreibung

`ST_QuantizeCoordinates` determines the number of bits (`N`) required to represent a coordinate value with a specified number of digits after the decimal point, and then sets all but the `N` most significant bits to zero. The resulting coordinate value will still round to the original value, but will have improved compressiblity. This can result in a significant disk usage reduction provided that the geometry column is using a compressible storage type. The function allows specification of a different number of digits after the decimal point in each dimension; unspecified dimensions are assumed to have the precision of the `x` dimension. Negative digits are interpreted to refer digits to the left of the decimal point, (i.e., `prec_x=-2` will preserve coordinate values to the nearest 100.

Die von `ST_QuantizeCoordinates` erzeugten Koordinaten sind unabhängig von der Geometrie, die diese Koordinaten und die relative Position dieser Koordinaten in der Geometrie enthält. Daher sind vorhandene topologische Beziehungen zwischen Geometrien durch die Verwendung dieser Funktion nicht betroffen. Die Funktion erzeugt möglicherweise ungültige Geometrie, wenn sie mit einer Anzahl von Stellen aufgerufen wird, die Koordinaten innerhalb der Geometrie zusammenfallen lassen.

Verfügbarkeit: 2.5.0

Technischer Hintergrund

PostGIS speichert alle Koordinatenwerte als Gleitkommazahlen mit doppelter Genauigkeit, die 15 signifikante Stellen zuverlässig darstellen können. PostGIS kann jedoch verwendet werden, um Daten zu verwalten, die weniger als 15 signifikante Ziffern enthalten. Ein Beispiel sind TIGER-Daten, die als geografische Koordinaten mit sechs Nachkommastellen zur Verfügung gestellt werden (so dass nur neun signifikante Ziffern des Längengrads und acht signifikante Breitengrade erforderlich sind).

Wenn 15 signifikante Ziffern verfügbar sind, gibt es viele mögliche Darstellungen einer Zahl mit 9 signifikanten Ziffern. Eine Gleitkommazahl mit doppelter Genauigkeit verwendet 52 explizite Bits, um den Mantisse der Koordinate darzustellen. Nur 30 Bits werden benötigt, um eine Mantisse mit 9 signifikanten Ziffern darzustellen, wobei 22 unbedeutende Bits übrig bleiben; Wir können ihren Wert auf alles setzen, was wir wollen, und erhalten trotzdem eine zum Eingabewert passende Zahl. Beispielsweise kann der Wert 100.123456 durch die nächstliegenden Zahlen 100.123456000000, 100.123456000001 und 100.123456432199 dargestellt werden. Alle sind gleichermaßen gültig, da `ST_AsText (geom, 6)` bei allen dieser Eingaben das gleiche Ergebnis liefert. Da wir diese Bits auf einen beliebigen Wert setzen können, setzt `ST_QuantizeCoordinates` die 22 nicht signifikanten Bits auf Null. Für eine lange Koordinatensequenz wird dadurch ein Muster aus Blöcken von aufeinanderfolgenden Nullen erzeugt, das von PostgreSQL effizienter komprimiert wird.

Note
Von `ST_QuantizeCoordinates` ist möglicherweise nur die Größe der Geometrie auf der Festplatte betroffen. ST_MemSize, das die speicherinterne Verwendung der Geometrie meldet, gibt unabhängig vom von einer Geometrie belegten Speicherplatz den gleichen Wert zurück.

Beispiele

```
SELECT ST_AsText(ST_QuantizeCoordinates('POINT (100.123456 0)'::geometry, 4));
st_astext
-------------------------
POINT(100.123455047607 0)
```

```
WITH test AS (SELECT 'POINT (123.456789123456 123.456789123456)'::geometry AS geom)
SELECT
  digits,
  encode(ST_QuantizeCoordinates(geom, digits), 'hex'),
  ST_AsText(ST_QuantizeCoordinates(geom, digits))
FROM test, generate_series(15, -15, -1) AS digits;

digits  |                  encode                  |                st_astext
--------+------------------------------------------+----------------------------------------- ↩

15      | 01010000005f9a72083cdd5e405f9a72083cdd5e40 | POINT(123.456789123456  ↩
    123.456789123456)
14      | 01010000005f9a72083cdd5e405f9a72083cdd5e40 | POINT(123.456789123456  ↩
    123.456789123456)
13      | 01010000005f9a72083cdd5e405f9a72083cdd5e40 | POINT(123.456789123456  ↩
    123.456789123456)
12      | 01010000005c9a72083cdd5e405c9a72083cdd5e40 | POINT(123.456789123456  ↩
    123.456789123456)
11      | 0101000000409a72083cdd5e40409a72083cdd5e40 | POINT(123.456789123456  ↩
    123.456789123456)
```

```
10      | 0101000000009a72083cdd5e40009a72083cdd5e40 | POINT(123.456789123455  ↩
    123.456789123455)
9       | 0101000000009072083cdd5e40009072083cdd5e40 | POINT(123.456789123418  ↩
    123.456789123418)
8       | 0101000000008072083cdd5e40008072083cdd5e40 | POINT(123.45678912336  ↩
    123.45678912336)
7       | 0101000000000070083cdd5e40000070083cdd5e40 | POINT(123.456789121032  ↩
    123.456789121032)
6       | 0101000000000040083cdd5e40000040083cdd5e40 | POINT(123.456789076328  ↩
    123.456789076328)
5       | 0101000000000000083cdd5e40000000083cdd5e40 | POINT(123.456789016724  ↩
    123.456789016724)
4       | 0101000000000000003cdd5e40000000003cdd5e40 | POINT(123.456787109375  ↩
    123.456787109375)
3       | 0101000000000000003cdd5e40000000003cdd5e40 | POINT(123.456787109375  ↩
    123.456787109375)
2       | 0101000000000000038dd5e400000000038dd5e40 | POINT(123.45654296875  ↩
    123.45654296875)
1       | 0101000000000000000dd5e400000000000dd5e40 | POINT(123.453125 123.453125)
0       | 0101000000000000000dc5e400000000000dc5e40 | POINT(123.4375 123.4375)
-1      | 0101000000000000000c05e400000000000c05e40 | POINT(123 123)
-2      | 01010000000000000000005e4000000000000005e40 | POINT(120 120)
-3      | 010100000000000000000058400000000000005840 | POINT(96 96)
-4      | 010100000000000000000058400000000000005840 | POINT(96 96)
-5      | 010100000000000000000058400000000000005840 | POINT(96 96)
-6      | 010100000000000000000058400000000000005840 | POINT(96 96)
-7      | 010100000000000000000058400000000000005840 | POINT(96 96)
-8      | 010100000000000000000058400000000000005840 | POINT(96 96)
-9      | 010100000000000000000058400000000000005840 | POINT(96 96)
-10     | 010100000000000000000058400000000000005840 | POINT(96 96)
-11     | 010100000000000000000058400000000000005840 | POINT(96 96)
-12     | 010100000000000000000058400000000000005840 | POINT(96 96)
-13     | 010100000000000000000058400000000000005840 | POINT(96 96)
-14     | 010100000000000000000058400000000000005840 | POINT(96 96)
-15     | 010100000000000000000058400000000000005840 | POINT(96 96)
```

Siehe auch

ST_SnapToGrid

7.5.24 ST_RemovePoint

ST_RemovePoint — Remove a point from a linestring.

Synopsis

geometry **ST_RemovePoint**(geometry linestring, integer offset);

Beschreibung

Removes a point from a LineString, given its index (0-based). Useful for turning a closed line (ring) into an open linestring.

Enhanced: 3.2.0

Verfügbarkeit: 1.1.0

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

Beispiele

Guarantees no lines are closed by removing the end point of closed lines (rings). Assumes geom is of type LINESTRING

```
UPDATE sometable
        SET geom = ST_RemovePoint(geom, ST_NPoints(geom) - 1)
        FROM sometable
        WHERE ST_IsClosed(geom);
```

Siehe auch

ST_AddPoint, ST_NPoints, ST_NumPoints

7.5.25 ST_RemoveRepeatedPoints

ST_RemoveRepeatedPoints — Returns a version of a geometry with duplicate points removed.

Synopsis

geometry **ST_RemoveRepeatedPoints**(geometry geom, float8 tolerance);

Beschreibung

Returns a version of the given geometry with duplicate consecutive points removed. The function processes only (Multi)LineStrings, (Multi)Polygons and MultiPoints but it can be called with any kind of geometry. Elements of GeometryCollections are processed individually. The endpoints of LineStrings are preserved.

If the *tolerance* parameter is provided, vertices within the tolerance distance of one another are considered to be duplicates.

Enhanced: 3.2.0

Verfügbarkeit: 2.2.0

This function supports Polyhedral surfaces.

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

Beispiele

```
SELECT ST_AsText( ST_RemoveRepeatedPoints( 'MULTIPOINT ((1 1), (2 2), (3 3), (2 2))'));
--------------------------
 MULTIPOINT(1 1,2 2,3 3)
```

```
SELECT ST_AsText( ST_RemoveRepeatedPoints( 'LINESTRING (0 0, 0 0, 1 1, 0 0, 1 1, 2 2)'));
--------------------------------
 LINESTRING(0 0,1 1,0 0,1 1,2 2)
```

Example: Collection elements are processed individually.

```
SELECT ST_AsText( ST_RemoveRepeatedPoints( 'GEOMETRYCOLLECTION (LINESTRING (1 1, 2 2, 2 2,  ↩
    3 3), POINT (4 4), POINT (4 4), POINT (5 5))'));
-----------------------------------------------------------------------------
 GEOMETRYCOLLECTION(LINESTRING(1 1,2 2,3 3),POINT(4 4),POINT(4 4),POINT(5 5))
```

Example: Repeated point removal with a distance tolerance.

```
SELECT ST_AsText( ST_RemoveRepeatedPoints( 'LINESTRING (0 0, 0 0, 1 1, 5 5, 1 1, 2 2)', 2)) ↩
    ;
-------------------------
 LINESTRING(0 0,5 5,2 2)
```

Siehe auch

ST_Simplify

7.5.26 ST_Reverse

ST_Reverse — Gibt die Geometrie in umgekehrter Knotenreihenfolge zurück.

Synopsis

geometry **ST_Reverse**(geometry g1);

Beschreibung

Kann mit jedem geometrischen Datentyp verwendet werden; kehrt die Reihenfolge der Knoten um

Erweiterung: mit 2.4.0 wurde die Unterstützung für Kurven eingeführt.

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

This function supports Polyhedral surfaces.

Beispiele

```
SELECT ST_AsText(geom) as line, ST_AsText(ST_Reverse(geom)) As reverseline
FROM
(SELECT ST_MakeLine(ST_Point(1,2),
                ST_Point(1,10)) As geom) as foo;
--result
                line         |     reverseline
--------------------+----------------------
LINESTRING(1 2,1 10) | LINESTRING(1 10,1 2)
```

7.5.27 ST_Segmentize

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

Synopsis

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

Beschreibung

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

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

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

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

Verfügbarkeit: 1.2.2

Enhanced: 3.0.0 Segmentize geometry now produces equal-length subsegments

Enhanced: 2.3.0 Segmentize geography now produces equal-length subsegments

Erweiterung: mit 2.1.0 wurde die Unterstützung des geographischen Datentyps eingeführt.

Changed: 2.1.0 As a result of the introduction of geography support, the usage `ST_Segmentize('LINESTRING(1 2, 3 4)', 0.5)` causes an ambiguous function error. The input needs to be properly typed as a geometry or geography. Use ST_GeomFromText, ST_GeogFromText or a cast to the required type (e.g. `ST_Segmentize('LINESTRING(1 2, 3 4)'::geometry, 0.5)`)

Beispiele

Segmentizing a line. Long segments are split evenly, and short segments are not split.

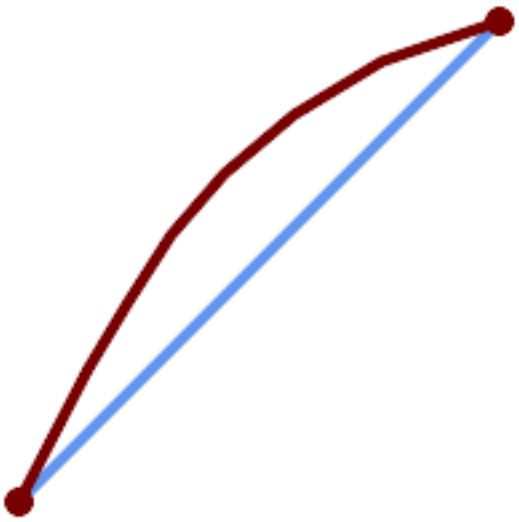
```
SELECT ST_AsText(ST_Segmentize(
    'MULTILINESTRING((0 0, 0 1, 0 9),(1 10, 1 18))'::geometry,
        5 ) );
-----------------------------------------------------
MULTILINESTRING((0 0,0 1,0 5,0 9),(1 10,1 14,1 18))
```

Segmentizing a polygon:

```
SELECT ST_AsText(
        ST_Segmentize(('POLYGON((0 0, 0 8, 30 0, 0 0))'::geometry), 10));
-----------------------------------------------------------
POLYGON((0 0,0 8,7.5 6,15 4,22.5 2,30 0,20 0,10 0,0 0))
```

Segmentizing a geographic line, using a maximum segment length of 2000 kilometers. Vertices are added along the great-circle arc connecting the endpoints.

```
SELECT ST_AsText(
        ST_Segmentize(('LINESTRING (0 0, 60 60)'::geography), 2000000));
------------------------------------------------------------
LINESTRING(0 0,4.252632294621186 8.43596525986862,8.69579947419404  ↩
    16.824093489701564,13.550465473227048 25.107950473646188,19.1066053508691  ↩
    33.21091076089908,25.779290201459894 41.01711439406505,34.188839517966954  ↩
    48.337222885886,45.238153936612264 54.84733442373889,60 60)
```



A geographic line segmentized along a great circle arc

Siehe auch

ST_LineSubstring

7.5.28 ST_SetPoint

ST_SetPoint — Einen Punkt eines Linienzuges durch einen gegebenen Punkt ersetzen.

Synopsis

geometry **ST_SetPoint**(geometry linestring, integer zerobasedposition, geometry point);

Beschreibung

Ersetzt den Punkt N eines Linienzuges mit dem gegebenen Punkt. Der Index beginnt mit 0. Negative Indizes werden rückwärts gezählt, sodasss -1 der letzte Punkt ist. Dies findet insbesondere bei Triggern verwendung, wenn man die Beziehung zwischen den Verbindungsstücken beim Verschieben von Knoten erhalten will

Verfügbarkeit: 1.1.0

Änderung: 2.3.0 : negatives Indizieren

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

Beispiele

```
--Change first point in line string from -1 3 to -1 1
SELECT ST_AsText(ST_SetPoint('LINESTRING(-1 2,-1 3)', 0, 'POINT(-1 1)'));
           st_astext
-----------------------
 LINESTRING(-1 1,-1 3)

---Change last point in a line string (lets play with 3d linestring this time)
SELECT ST_AsEWKT(ST_SetPoint(foo.geom, ST_NumPoints(foo.geom) - 1, ST_GeomFromEWKT('POINT ↩
    (-1 1 3)')))
```

```
FROM (SELECT ST_GeomFromEWKT('LINESTRING(-1 2 3,-1 3 4, 5 6 7)') As geom) As foo;
           st_asewkt
-----------------------
LINESTRING(-1 2 3,-1 3 4,-1 1 3)

SELECT ST_AsText(ST_SetPoint(g, -3, p))
FROM ST_GEomFromText('LINESTRING(0 0, 1 1, 2 2, 3 3, 4 4)') AS g
        , ST_PointN(g,1) as p;
           st_astext
-----------------------
LINESTRING(0 0,1 1,0 0,3 3,4 4)
```

Siehe auch

ST_AddPoint, ST_NPoints, ST_NumPoints, ST_PointN, ST_RemovePoint

7.5.29 ST_ShiftLongitude

ST_ShiftLongitude — Shifts the longitude coordinates of a geometry between -180..180 and 0..360.

Synopsis

geometry **ST_ShiftLongitude**(geometry geom);

Beschreibung

Reads every point/vertex in a geometry, and shifts its longitude coordinate from -180..0 to 180..360 and vice versa if between these ranges. This function is symmetrical so the result is a 0..360 representation of a -180..180 data and a -180..180 representation of a 0..360 data.

Note
This is only useful for data with coordinates in longitude/latitude; e.g. SRID 4326 (WGS 84 geographic)

Warning
Pre-1.3.4 Aufgrund eines Bugs funktionierte dies für MULTIPOINT nicht. Mit 1.3.4+ funktioniert es auch mit MULTIPOINT.

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

Erweiterung: Mit 2.0.0 wurde die Unterstützung für polyedrische Oberflächen und TIN eingeführt.

Anmerkung: Vor 2.2.0 hieß diese Funktion "ST_Shift_Longitude"

This function supports Polyhedral surfaces.

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

Beispiele

```
--single point forward transformation
SELECT ST_AsText(ST_ShiftLongitude('SRID=4326;POINT(270 0)'::geometry))

st_astext
----------
POINT(-90 0)


--single point reverse transformation
SELECT ST_AsText(ST_ShiftLongitude('SRID=4326;POINT(-90 0)'::geometry))

st_astext
----------
POINT(270 0)


--for linestrings the functions affects only to the sufficient coordinates
SELECT ST_AsText(ST_ShiftLongitude('SRID=4326;LINESTRING(174 12, 182 13)'::geometry))

st_astext
----------
LINESTRING(174 12,-178 13)
```

Siehe auch

ST_WrapX

7.5.30 ST_WrapX

ST_WrapX — Versammelt eine Geometrie um einen X-Wert

Synopsis

geometry **ST_WrapX**(geometry geom, float8 wrap, float8 move);

Beschreibung

This function splits the input geometries and then moves every resulting component falling on the right (for negative 'move') or on the left (for positive 'move') of given 'wrap' line in the direction specified by the 'move' parameter, finally re-unioning the pieces together.

Note
Nützlich, um eine Eingabe in Länge und Breite neu zu zentrieren, damit die wesentlichen Geoobjekte nicht von einer Seite bis zur anderen abgebildet werden.

Availability: 2.3.0 requires GEOS

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

Beispiele

```
-- 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);
```

Siehe auch

ST_ShiftLongitude

7.5.31 ST_SnapToGrid

ST_SnapToGrid — Fängt alle Punkte der Eingabegeometrie auf einem regelmäßigen Gitter.

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);

Beschreibung

Variante 1, 2 und 3: Fängt alle Punkte der Eingabegeometrie auf den Gitterpunkten, die durch Ursprung und Gitterkästchengröße festgelegt sind. Aufeinanderfolgende Punkte, die in dasselbe Gitterkästchen fallen, werden gelöscht, wobei NULL zurückgegeben wird, wenn nicht mehr genug Punkte für den jeweiligen geometrischen Datentyp vorhanden sind. Collapsed geometries in a collection are stripped from it. Kollabierte Geometrien einer Kollektion werden von dieser entfernt. Nützlich um die Genauigkeit zu verringern.

Variante 4: wurde mit 1.1.0 eingeführt - Fängt alle Punkte der Eingabegeometrie auf den Gitterpunkten, welche durch den Ursprung des Gitters (der zweite Übergabewert muss ein Punkt sein) und die Gitterkästchengröße bestimmt sind. Geben Sie 0 als Größe für jene Dimension an, die nicht auf den Gitterpunkten gefangen werden soll.

Note
Die zurückgegebene Geometrie kann ihre Simplizität verlieren (siehe ST_IsSimple).

Note
Vor Release 1.1.0 gab diese Funktion immer eine 2D-Geometrie zurück. Ab 1.1.0 hat die zurückgegebene Geometrie dieselbe Dimensionalität wie die Eingabegemetrie, wobei höhere Dimensionen unangetastet bleiben. Verwenden Sie die Version, welche einen zweiten geometrischen Übergabewert annimmt, um sämtliche Grid-Dimensionen zu bestimmen.

Verfügbarkeit: 1.0.0RC1

Verfügbarkeit: 1.1.0, Unterstützung für Z und M

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

Beispiele

```
--Snap your geometries to a precision grid of 10^-3
UPDATE mytable
   SET geom = ST_SnapToGrid(geom, 0.001);

SELECT ST_AsText(ST_SnapToGrid(
                        ST_GeomFromText('LINESTRING(1.1115678 2.123, 4.111111 3.2374897,  ↩
                            4.11112 3.23748667)'),
                        0.001)
                );
                          st_astext
-------------------------------------
 LINESTRING(1.112 2.123,4.111 3.237)
 --Snap a 4d geometry
SELECT ST_AsEWKT(ST_SnapToGrid(
        ST_GeomFromEWKT('LINESTRING(-1.1115678 2.123 2.3456 1.11111,
                4.111111 3.2374897 3.1234 1.1111, -1.11111112 2.123 2.3456 1.1111112)'),
 ST_GeomFromEWKT('POINT(1.12 2.22 3.2 4.4444)'),
 0.1, 0.1, 0.1, 0.01) );
                                                                        st_asewkt
--------------------------------------------------------------------------------
 LINESTRING(-1.08 2.12 2.3 1.1144,4.12 3.22 3.1 1.1144,-1.08 2.12 2.3 1.1144)


--With a 4d geometry - the ST_SnapToGrid(geom,size) only touches x and y coords but keeps m ↩
     and z the same
SELECT ST_AsEWKT(ST_SnapToGrid(ST_GeomFromEWKT('LINESTRING(-1.1115678 2.123 3 2.3456,
                4.111111 3.2374897 3.1234 1.1111)'),
           0.01)      );
                                                 st_asewkt
-----------------------------------------------------------
 LINESTRING(-1.11 2.12 3 2.3456,4.11 3.24 3.1234 1.1111)
```

Siehe auch

ST_Snap, ST_AsEWKT, ST_AsText, ST_GeomFromText, ST_GeomFromEWKT, ST_Simplify

7.5.32 ST_Snap

ST_Snap — Fängt die Segmente und Knoten einer Eingabegeometrie an den Knoten einer Referenzgeometrie.

Synopsis

geometry **ST_Snap**(geometry input, geometry reference, float tolerance);

Beschreibung

Fängt die Knoten und Segmente einer Geometrie an den Knoten einer anderen Geometrie. Eine Entfernungstoleranz bestimmt, wo das Fangen durchgeführt wird. Die Ergebnisgeometrie ist die Eingabegeometrie mit gefangenen Knoten. Wenn kein Fangen auftritt, wird die Eingabegeometrie unverändert ausgegeben..

Eine Geometrie an einer anderen zu fangen, kann die Robustheit von Überlagerungs-Operationen verbessern, indem nahe zusammenfallende Kanten beseitigt werden (diese verursachen Probleme bei der Knoten- und Verschneidungsberechnung).

Übermäßiges Fangen kann zu einer invaliden Topologie führen. Die Anzahl und der Ort an dem Knoten sicher gefangen werden können wird mittels Heuristik bestimmt. Dies kann allerdings dazu führen, dass einige potentielle Knoten nicht gefangen werden.

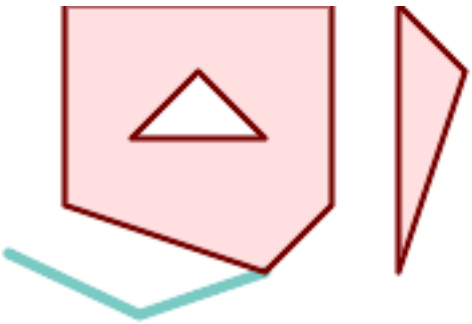
Note
Die zurückgegebene Geometrie kann ihre Simplizität (see ST_IsSimple) und Valididät (see ST_IsValid) verlieren.

Wird vom GEOS Modul ausgeführt

Verfügbarkeit: 2.0.0

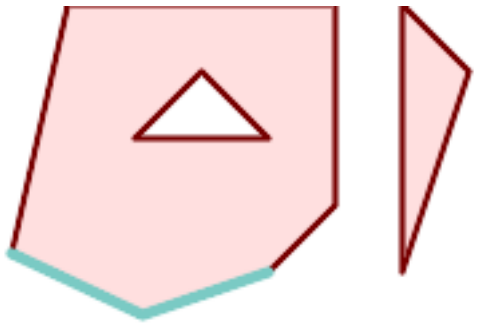
Beispiele

Ein Mehrfachpolygon mit einem Linienzug (vor dem Fangen)



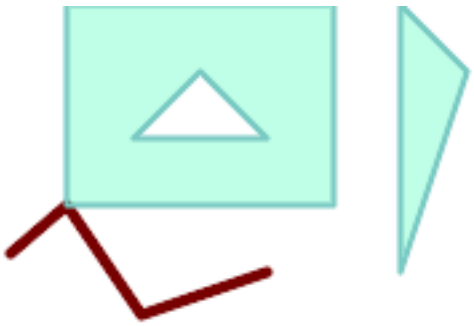
Ein Mehrfachpolygon das an einem Linienzug gefangen wird; die Toleranz beträgt 1.01 der Entfernung. Das neue Mehrfachpolygon wird mit dem betreffenden Linienzug angezeigt.

```
SELECT ST_AsText(ST_Snap(poly,line,  ↩
    ST_Distance(poly,line)*1.01)) AS polysnapped
FROM (SELECT
   ST_GeomFromText('MULTIPOLYGON(
     ((26 125, 26 200, 126 200, 126 125,  ↩
    26 125 ),
      ( 51 150, 101 150, 76 175, 51 150 ) ↩
    ),
      (( 151 100, 151 200, 176 175, 151  ↩
    100 )))') As poly,
       ST_GeomFromText('LINESTRING (5  ↩
    107, 54 84, 101 100)') As line
        ) As foo;

                              polysnapped
-------------------------------------------------------------------------------
 MULTIPOLYGON(((26 125,26 200,126 200,126 ↩
     125,101 100,26 125),
 (51 150,101 150,76 175,51 150)),((151  ↩
    100,151 200,176 175,151 100)))
```



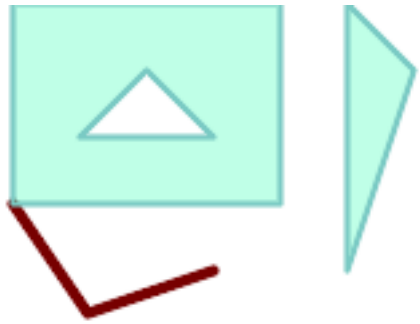
Ein Mehrfachpolygon das an einem Linienzug gefangen wird; die Toleranz beträgt 1.25 der Entfernung. Das neue Mehrfachpolygon wird mit dem betreffenden Linienzug angezeigt.

```
SELECT ST_AsText(
    ST_Snap(poly,line, ST_Distance(poly, ↩
    line)*1.25)
  ) AS polysnapped
FROM (SELECT
  ST_GeomFromText('MULTIPOLYGON(
    (( 26 125, 26 200, 126 200, 126 125,  ↩
    26 125 ),
      ( 51 150, 101 150, 76 175, 51 150 ) ↩
    ),
      (( 151 100, 151 200, 176 175, 151  ↩
    100 )))') As poly,
       ST_GeomFromText('LINESTRING (5  ↩
    107, 54 84, 101 100)') As line
        ) As foo;

                                        ↩   polysnapped
-------------------------------------------------------------------------------

MULTIPOLYGON(((5 107,26 200,126 200,126  ↩
    125,101 100,54 84,5 107),
(51 150,101 150,76 175,51 150)),((151  ↩
    100,151 200,176 175,151 100)))
```



Ein Linienzug der an dem ursprünglichen Mehrfachpolygon gefangen wird; die Toleranz beträgt 1.01 der Entfernung. Das neue Linienzug wird mit dem betreffenden Mehrfachpolygon angezeigt.

```
SELECT ST_AsText(
   ST_Snap(line, poly, ST_Distance(poly, ↩
     line)*1.01)
  ) AS linesnapped
FROM (SELECT
  ST_GeomFromText('MULTIPOLYGON(
     ((26 125, 26 200, 126 200, 126 125,  ↩
    26 125),
      (51 150, 101 150, 76 175, 51 150 )) ↩
    ,
     ((151 100, 151 200, 176 175, 151  ↩
    100)))') As poly,
      ST_GeomFromText('LINESTRING (5  ↩
    107, 54 84, 101 100)') As line
       ) As foo;

              linesnapped
----------------------------------------
 LINESTRING(5 107,26 125,54 84,101 100)
```



Ein Linienzug der an dem ursprünglichen Mehrfachpolygon gefangen wird; die Toleranz beträgt 1.25 der Entfernung. Das neue Linienzug wird mit dem betreffenden Mehrfachpolygon angezeigt.

```
SELECT ST_AsText(
 ST_Snap(line, poly, ST_Distance(poly, ↩
     line)*1.25)
  ) AS linesnapped
FROM (SELECT
  ST_GeomFromText('MULTIPOLYGON(
     (( 26 125, 26 200, 126 200, 126 125, ↩
      26 125 ),
      (51 150, 101 150, 76 175, 51 150 )) ↩
    ,
     ((151 100, 151 200, 176 175, 151  ↩
    100 )))') As poly,
      ST_GeomFromText('LINESTRING (5  ↩
    107, 54 84, 101 100)') As line
       ) As foo;
              linesnapped
----------------------------------------
LINESTRING(26 125,54 84,101 100)
```

Siehe auch

ST_SnapToGrid

7.5.33 ST_SwapOrdinates

ST_SwapOrdinates — Gibt eine Version der Ausgangsgeometrie zurück, in der die angegebenen Ordinatenwerte ausgetauscht werden.

Synopsis

geometry **ST_SwapOrdinates**(geometry geom, cstring ords);

Beschreibung

Gibt eine Version der Ausgangsgeometrie zurück, in der die angegebenen Ordinaten ausgetauscht werden.

Der `ords` Parameter ist eine Zeichenkette aus 2 Zeichen, welche die Ordinate benennt die getauscht werden soll. Gültige Bezeichnungen sind: x,y,z und m.

Verfügbarkeit: 2.2.0

This method supports Circular Strings and Curves.

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

This function supports M coordinates.

This function supports Polyhedral surfaces.

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

Beispiel

```
-- Scale M value by 2
SELECT ST_AsText(
  ST_SwapOrdinates(
    ST_Scale(
      ST_SwapOrdinates(g,'xm'),
      2, 1
    ),
  'xm')
) FROM ( SELECT 'POINT ZM (0 0 0 2)'::geometry g ) foo;
     st_astext
--------------------
 POINT ZM (0 0 0 4)
```

Siehe auch

ST_FlipCoordinates

7.6 Geometrievalidierung

7.6.1 ST_IsValid

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

Synopsis

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

Beschreibung

Tests if an ST_Geometry value is well-formed and valid in 2D according to the OGC rules. For geometries with 3 and 4 dimensions, the validity is still only tested in 2 dimensions. For geometries that are invalid, a PostgreSQL NOTICE is emitted providing details of why it is not valid.

For the version with the `flags` parameter, supported values are documented in ST_IsValidDetail This version does not print a NOTICE explaining invalidity.

For more information on the definition of geometry validity, refer to Section 4.4

Note
SQL-MM defines the result of ST_IsValid(NULL) to be 0, while PostGIS returns NULL.

Performed by the GEOS module.

The version accepting flags is available starting with 2.0.0.

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

This method implements the SQL/MM specification. SQL-MM 3: 5.1.9

Note
Neither OGC-SFS nor SQL-MM specifications include a flag argument for ST_IsValid. The flag is a PostGIS extension.

Examples

```
SELECT ST_IsValid(ST_GeomFromText('LINESTRING(0 0, 1 1)')) As good_line,
        ST_IsValid(ST_GeomFromText('POLYGON((0 0, 1 1, 1 2, 1 1, 0 0))')) As bad_poly
--results
NOTICE:  Self-intersection at or near point 0 0
 good_line | bad_poly
-----------+----------
 t         | f
```

Siehe auch

ST_IsSimple, ST_IsValidReason, ST_IsValidDetail,

7.6.2 ST_IsValidDetail

ST_IsValidDetail — Returns a `valid_detail` row stating if a geometry is valid or if not a reason and a location.

Synopsis

valid_detail **ST_IsValidDetail**(geometry geom, integer flags);

Beschreibung

Returns a `valid_detail` row, containing a boolean (`valid`) stating if a geometry is valid, a varchar (`reason`) stating a reason why it is invalid and a geometry (`location`) pointing out where it is invalid.

Useful to improve on the combination of ST_IsValid and ST_IsValidReason to generate a detailed report of invalid geometries.

The optional `flags` parameter is a bitfield. It can have the following values:

- 0: Use usual OGC SFS validity semantics.
- 1: Consider certain kinds of self-touching rings (inverted shells and exverted holes) as valid. This is also known as "the ESRI flag", since this is the validity model used by those tools. Note that this is invalid under the OGC model.

Performed by the GEOS module.

Verfügbarkeit: 2.0.0

Examples

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

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

 --simple example
SELECT * FROM ST_IsValidDetail('LINESTRING(220227 150406,2220227 150407,222020 150410)');

 valid | reason | location
-------+--------+----------
 t     |        |
```

Siehe auch

ST_IsValid, ST_IsValidReason

7.6.3 ST_IsValidReason

ST_IsValidReason — Returns text stating if a geometry is valid, or a reason for invalidity.

Synopsis

text **ST_IsValidReason**(geometry geomA);
text **ST_IsValidReason**(geometry geomA, integer flags);

Beschreibung

Returns text stating if a geometry is valid, or if invalid a reason why.

Useful in combination with ST_IsValid to generate a detailed report of invalid geometries and reasons.

Allowed `flags` are documented in ST_IsValidDetail.

Performed by the GEOS module.

Availability: 1.4

Availability: 2.0 version taking flags.

Examples

```
-- invalid bow-tie polygon
SELECT ST_IsValidReason(
    'POLYGON ((100 200, 100 100, 200 200,
     200 100, 100 200))'::geometry) as validity_info;
validity_info
--------------------------
Self-intersection[150 150]
```

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

 gid  |      validity_info
------+--------------------------
 5330 | Self-intersection [32 5]
 5340 | Self-intersection [42 5]
 5350 | Self-intersection [52 5]
```

```
 --simple example
SELECT ST_IsValidReason('LINESTRING(220227 150406,2220227 150407,222020 150410)');

 st_isvalidreason
------------------
 Valid Geometry
```

Siehe auch

ST_IsValid, ST_Summary

7.6.4 ST_MakeValid

ST_MakeValid — Attempts to make an invalid geometry valid without losing vertices.

Synopsis

geometry **ST_MakeValid**(geometry input);
geometry **ST_MakeValid**(geometry input, text params);

Beschreibung

The function attempts to create a valid representation of a given invalid geometry without losing any of the input vertices. Valid geometries are returned unchanged.

Supported inputs are: POINTS, MULTIPOINTS, LINESTRINGS, MULTILINESTRINGS, POLYGONS, MULTIPOLYGONS and GEOMETRYCOLLECTIONS containing any mix of them.

In case of full or partial dimensional collapses, the output geometry may be a collection of lower-to-equal dimension geometries, or a geometry of lower dimension.

Single polygons may become multi-geometries in case of self-intersections.

The `params` argument can be used to supply an options string to select the method to use for building valid geometry. The options string is in the format "method=linework|structure keepcollapsed=true|false". If no "params" argument is provided, the "linework" algorithm will be used as the default.

The "method" key has two values.

- "linework" is the original algorithm, and builds valid geometries by first extracting all lines, noding that linework together, then building a value output from the linework.
- "structure" is an algorithm that distinguishes between interior and exterior rings, building new geometry by unioning exterior rings, and then differencing all interior rings.

The "keepcollapsed" key is only valid for the "structure" algorithm, and takes a value of "true" or "false". When set to "false", geometry components that collapse to a lower dimensionality, for example a one-point linestring would be dropped.

Performed by the GEOS module.

Verfügbarkeit: 2.0.0

Enhanced: 2.0.1, speed improvements

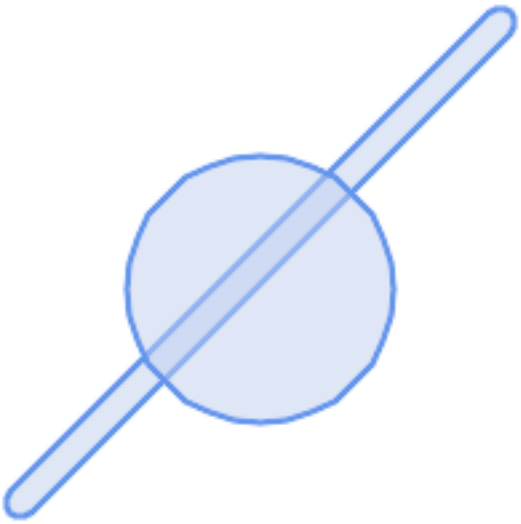
Enhanced: 2.1.0, added support for GEOMETRYCOLLECTION and MULTIPOINT.

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

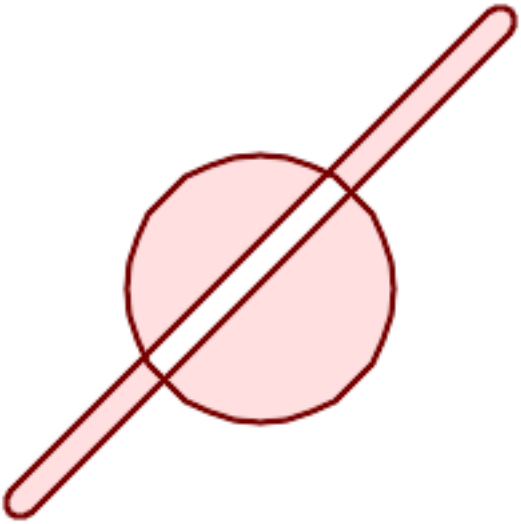
Enhanced: 3.2.0, added algorithm options, 'linework' and 'structure' which requires GEOS >= 3.10.0.

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

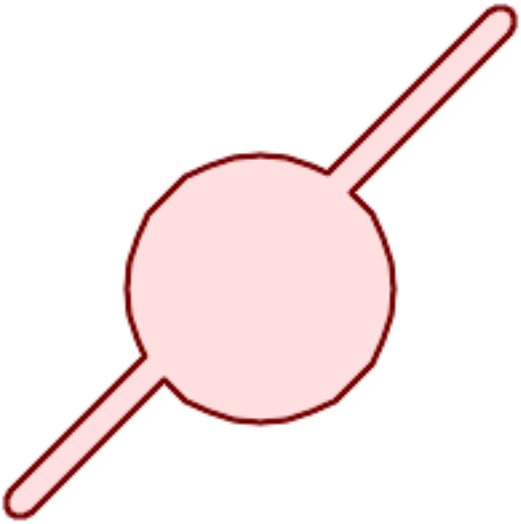
Examples



before_geom: MULTIPOLYGON of 2 overlapping polygons

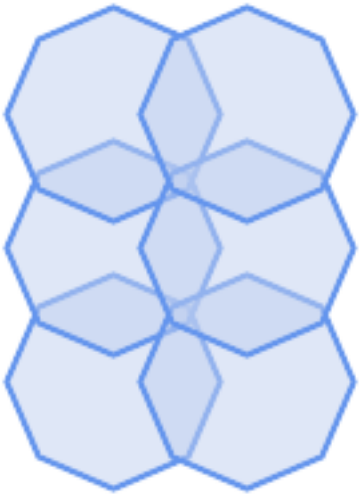


after_geom: MULTIPOLYGON of 4 non-overlapping polygons

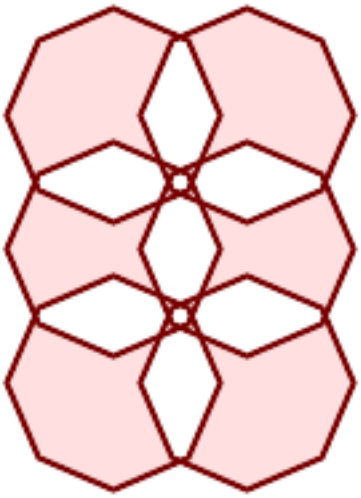


after_geom_structure: MULTIPOLYGON of 1 non-overlapping polygon

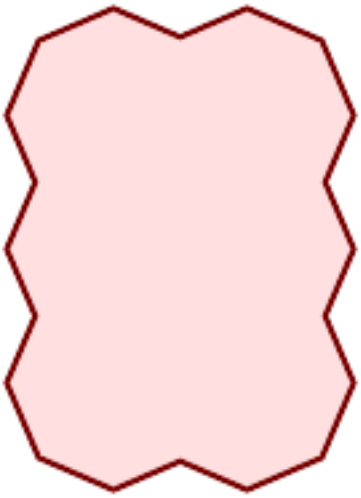
```
SELECT f.geom AS before_geom, ST_MakeValid(f.geom) AS after_geom, ST_MakeValid(f.geom,  ↩
    'method=structure') AS after_geom_structure
FROM (SELECT 'MULTIPOLYGON(((186 194,187 194,188 195,189 195,190 195,
191 195,192 195,193 194,194 194,194 194,193 195,192 195,191
```



before_geom: MULTIPOLYGON of 6 overlapping polygons



after_geom: MULTIPOLYGON of 14 Non-overlapping polygons



after_geom_structure: MULTIPOLYGON of 1 Non-overlapping polygon

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

Examples

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

 st_astext
------------
 POINT(0 0)


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

    st_astext
------------------
 LINESTRING EMPTY
```

Siehe auch

ST_IsValid, ST_Collect, ST_CollectionExtract

7.7 Spatial Reference System Functions

7.7.1 ST_InverseTransformPipeline

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

Synopsis

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

Beschreibung

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

Refer to ST_TransformPipeline for details on writing a transformation pipeline.

Availability: 3.4.0

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

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

Examples

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

```
-- Inverse direction
SELECT ST_AsText(ST_InverseTransformPipeline('POINT(426857.9877165967 5427937.523342293)':: ↩
    geometry,
  'urn:ogc:def:coordinateOperation:EPSG::16031')) AS wgs_geom;

          wgs_geom
----------------------------
 POINT(2 48.99999999999999)
(1 row)
```

GDA2020 example.

```
-- using ST_Transform with automatic selection of a conversion pipeline.
SELECT ST_AsText(ST_Transform('SRID=4939;POINT(143.0 -37.0)'::geometry, 7844)) AS  ↩
    gda2020_auto;

                gda2020_auto
---------------------------------------------
 POINT(143.00000635638918 -36.999986706128176)
(1 row)
```

Siehe auch

ST_Transform, ST_TransformPipeline

7.7.2 ST_SetSRID

ST_SetSRID — Set the SRID on a geometry.

Synopsis

geometry **ST_SetSRID**(geometry geom, integer srid);

Beschreibung

Sets the SRID on a geometry to a particular integer value. Useful in constructing bounding boxes for queries.

Note
This function does not transform the geometry coordinates in any way - it simply sets the meta data defining the spatial reference system the geometry is assumed to be in. Use ST_Transform if you want to transform the geometry into a new projection.

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

This method supports Circular Strings and Curves.

Examples

-- Mark a point as WGS 84 long lat --

```
SELECT ST_SetSRID(ST_Point(-123.365556, 48.428611),4326) As wgs84long_lat;
-- the ewkt representation (wrap with ST_AsEWKT) -
SRID=4326;POINT(-123.365556 48.428611)
```

-- Mark a point as WGS 84 long lat and then transform to web mercator (Spherical Mercator) --

```
SELECT ST_Transform(ST_SetSRID(ST_Point(-123.365556, 48.428611),4326),3785) As spere_merc;
-- the ewkt representation (wrap with ST_AsEWKT) -
SRID=3785;POINT(-13732990.8753491 6178458.96425423)
```

Siehe auch

Section 4.5, ST_SRID, ST_Transform, UpdateGeometrySRID

7.7.3 ST_SRID

ST_SRID — Returns the spatial reference identifier for a geometry.

Synopsis

integer **ST_SRID**(geometry g1);

Beschreibung

Returns the spatial reference identifier for the ST_Geometry as defined in spatial_ref_sys table. Section 4.5

Note
spatial_ref_sys table is a table that catalogs all spatial reference systems known to PostGIS and is used for transformations from one spatial reference system to another. So verifying you have the right spatial reference system identifier is important if you plan to ever transform your geometries.

This method implements the OGC Simple Features Implementation Specification for SQL 1.1. s2.1.1.1

This method implements the SQL/MM specification. SQL-MM 3: 5.1.5

This method supports Circular Strings and Curves.

Examples

```
SELECT ST_SRID(ST_GeomFromText('POINT(-71.1043 42.315)',4326));
    --result
    4326
```

Siehe auch

Section 4.5, ST_SetSRID, ST_Transform, ST_SRID, ST_SRID

7.7.4 ST_Transform

ST_Transform — Return a new geometry with coordinates transformed to a different spatial reference system.

Synopsis

geometry **ST_Transform**(geometry g1, integer srid);
geometry **ST_Transform**(geometry geom, text to_proj);
geometry **ST_Transform**(geometry geom, text from_proj, text to_proj);
geometry **ST_Transform**(geometry geom, text from_proj, integer to_srid);

Beschreibung

Returns a new geometry with its coordinates transformed to a different spatial reference system. The destination spatial reference `to_srid` may be identified by a valid SRID integer parameter (i.e. it must exist in the `spatial_ref_sys` table). Alternatively, a spatial reference defined as a PROJ.4 string can be used for `to_proj` and/or `from_proj`, however these methods are not optimized. If the destination spatial reference system is expressed with a PROJ.4 string instead of an SRID, the SRID of the output geometry will be set to zero. With the exception of functions with `from_proj`, input geometries must have a defined SRID.

ST_Transform is often confused with ST_SetSRID. ST_Transform actually changes the coordinates of a geometry from one spatial reference system to another, while ST_SetSRID() simply changes the SRID identifier of the geometry.

ST_Transform automatically selects a suitable conversion pipeline given the source and target spatial reference systems. To use a specific conversion method, use ST_TransformPipeline.

Note
Requires PostGIS be compiled with PROJ support. Use PostGIS_Full_Version to confirm you have PROJ support compiled in.

Note
If using more than one transformation, it is useful to have a functional index on the commonly used transformations to take advantage of index usage.

Note
Prior to 1.3.4, this function crashes if used with geometries that contain CURVES. This is fixed in 1.3.4+

Enhanced: 2.0.0 support for Polyhedral surfaces was introduced.

Enhanced: 2.3.0 support for direct PROJ.4 text was introduced.

This method implements the SQL/MM specification. SQL-MM 3: 5.1.6

This method supports Circular Strings and Curves.

This function supports Polyhedral surfaces.

Examples

Change Massachusetts state plane US feet geometry to WGS 84 long lat

```
SELECT ST_AsText(ST_Transform(ST_GeomFromText('POLYGON((743238 2967416,743238 2967450,
  743265 2967450,743265.625 2967416,743238 2967416))',2249),4326)) As wgs_geom;

 wgs_geom
---------------------------
 POLYGON((-71.1776848522251 42.3902896512902,-71.1776843766326 42.3903829478009,
-71.1775844305465 42.3903826677917,-71.1775825927231 42.3902893647987,-71.177684
8522251 42.3902896512902));
(1 row)

--3D Circular String example
SELECT ST_AsEWKT(ST_Transform(ST_GeomFromEWKT('SRID=2249;CIRCULARSTRING(743238 2967416  ↩
    1,743238 2967450 2,743265 2967450 3,743265.625 2967416 3,743238 2967416 4)'),4326));

         st_asewkt
---------------------------------------------------------------------------------------
 SRID=4326;CIRCULARSTRING(-71.1776848522251 42.3902896512902 1,-71.1776843766326  ↩
     42.3903829478009 2,
 -71.1775844305465 42.3903826677917 3,
 -71.1775825927231 42.3902893647987 3,-71.1776848522251 42.3902896512902 4)
```

Example of creating a partial functional index. For tables where you are not sure all the geometries will be filled in, its best to use a partial index that leaves out null geometries which will both conserve space and make your index smaller and more efficient.

```
CREATE INDEX idx_geom_26986_parcels
  ON parcels
  USING gist
  (ST_Transform(geom, 26986))
  WHERE geom IS NOT NULL;
```

Examples of using PROJ.4 text to transform with custom spatial references.

```
-- Find intersection of two polygons near the North pole, using a custom Gnomic projection
-- See http://boundlessgeo.com/2012/02/flattening-the-peel/
 WITH data AS (
   SELECT
     ST_GeomFromText('POLYGON((170 50,170 72,-130 72,-130 50,170 50))', 4326) AS p1,
     ST_GeomFromText('POLYGON((-170 68,-170 90,-141 90,-141 68,-170 68))', 4326) AS p2,
     '+proj=gnom +ellps=WGS84 +lat_0=70 +lon_0=-160 +no_defs'::text AS gnom
 )
 SELECT ST_AsText(
   ST_Transform(
     ST_Intersection(ST_Transform(p1, gnom), ST_Transform(p2, gnom)),
   gnom, 4326))
 FROM data;
                                          st_astext
 ------------------------------------------------------------------------------
  POLYGON((-170 74.053793645338,-141 73.4268621378904,-141 68,-170 68,-170 74.053793645338) ↩
      )
```

Configuring transformation behavior

Sometimes coordinate transformation involving a grid-shift can fail, for example if PROJ.4 has not been built with grid-shift files or the coordinate does not lie within the range for which the grid shift is defined. By default, PostGIS will throw an error if a

grid shift file is not present, but this behavior can be configured on a per-SRID basis either by testing different `to_proj` values of PROJ.4 text, or altering the `proj4text` value within the `spatial_ref_sys` table.

For example, the proj4text parameter +datum=NAD87 is a shorthand form for the following +nadgrids parameter:

```
+nadgrids=@conus,@alaska,@ntv2_0.gsb,@ntv1_can.dat
```

The @ prefix means no error is reported if the files are not present, but if the end of the list is reached with no file having been appropriate (ie. found and overlapping) then an error is issued.

If, conversely, you wanted to ensure that at least the standard files were present, but that if all files were scanned without a hit a null transformation is applied you could use:

```
+nadgrids=@conus,@alaska,@ntv2_0.gsb,@ntv1_can.dat,null
```

The null grid shift file is a valid grid shift file covering the whole world and applying no shift. So for a complete example, if you wanted to alter PostGIS so that transformations to SRID 4267 that didn't lie within the correct range did not throw an ERROR, you would use the following:

```
UPDATE spatial_ref_sys SET proj4text = '+proj=longlat +ellps=clrk66 +nadgrids=@conus, ↩
    @alaska,@ntv2_0.gsb,@ntv1_can.dat,null +no_defs' WHERE srid = 4267;
```

Siehe auch

Section 4.5, ST_SetSRID, ST_SRID, UpdateGeometrySRID, ST_TransformPipeline

7.7.5 ST_TransformPipeline

ST_TransformPipeline — Return a new geometry with coordinates transformed to a different spatial reference system using a defined coordinate transformation pipeline.

Synopsis

geometry **ST_TransformPipeline**(geometry g1, text pipeline, integer to_srid);

Beschreibung

Return a new geometry with coordinates transformed to a different spatial reference system using a defined coordinate transformation pipeline.

Transformation pipelines are defined using any of the following string formats:

- `urn:ogc:def:coordinateOperation:AUTHORITY::CODE`. Note that a simple `EPSG:CODE` string does not uniquely identify a coordinate operation: the same EPSG code can be used for a CRS definition.
- A PROJ pipeline string of the form: `+proj=pipeline ...`. Automatic axis normalisation will not be applied, and if necessary the caller will need to add an additional pipeline step, or remove `axisswap` steps.
- Concatenated operations of the form: `urn:ogc:def:coordinateOperation,coordinateOperation:EPSG::3895,c`

Availability: 3.4.0

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

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

Examples

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

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

                  utm_geom
--------------------------------------------
 POINT(426857.9877165967 5427937.523342293)
(1 row)

-- Inverse direction
SELECT ST_AsText(ST_InverseTransformPipeline('POINT(426857.9877165967 5427937.523342293)':: ↩
    geometry,
  'urn:ogc:def:coordinateOperation:EPSG::16031')) AS wgs_geom;

          wgs_geom
-----------------------------
 POINT(2 48.99999999999999)
(1 row)
```

GDA2020 example.

```
-- using ST_Transform with automatic selection of a conversion pipeline.
SELECT ST_AsText(ST_Transform('SRID=4939;POINT(143.0 -37.0)'::geometry, 7844)) AS  ↩
    gda2020_auto;

                 gda2020_auto
-----------------------------------------------
 POINT(143.00000635638918 -36.999986706128176)
(1 row)

-- using a defined conversion (EPSG:8447)
SELECT ST_AsText(ST_TransformPipeline('SRID=4939;POINT(143.0 -37.0)'::geometry,
  'urn:ogc:def:coordinateOperation:EPSG::8447')) AS gda2020_code;

                  gda2020_code
-----------------------------------------------
 POINT(143.0000063280214 -36.999986718287545)
(1 row)

-- using a PROJ pipeline definition matching EPSG:8447, as returned from
-- 'projinfo -s EPSG:4939 -t EPSG:7844'.
-- NOTE: any 'axisswap' steps must be removed.
SELECT ST_AsText(ST_TransformPipeline('SRID=4939;POINT(143.0 -37.0)'::geometry,
  '+proj=pipeline
   +step +proj=unitconvert +xy_in=deg +xy_out=rad
   +step +proj=hgridshift +grids=au_icsm_GDA94_GDA2020_conformal_and_distortion.tif
   +step +proj=unitconvert +xy_in=rad +xy_out=deg')) AS gda2020_pipeline;

                  gda2020_pipeline
-----------------------------------------------
 POINT(143.0000063280214 -36.999986718287545)
(1 row)
```

Siehe auch

ST_Transform, ST_InverseTransformPipeline

7.7.6 postgis_srs_codes

postgis_srs_codes — Return the list of SRS codes associated with the given authority.

Synopsis

setof text **postgis_srs_codes**(text auth_name);

Beschreibung

Returns a set of all `auth_srid` for the given `auth_name`.

Availability: 3.4.0

Proj version 6+

Examples

List the first ten codes associated with the EPSG authority.

```
SELECT * FROM postgis_srs_codes('EPSG') LIMIT 10;

 postgis_srs_codes
-------------------
 2000
 20004
 20005
 20006
 20007
 20008
 20009
 2001
 20010
 20011
```

Siehe auch

postgis_srs, postgis_srs_all, postgis_srs_search

7.7.7 postgis_srs

postgis_srs — Return a metadata record for the requested authority and srid.

Synopsis

setof record **postgis_srs**(text auth_name, text auth_srid);

Beschreibung

Returns a metadata record for the requested `auth_srid` for the given `auth_name`. The record will have the `auth_name`, `auth_srid`, `srname`, `srtext`, `proj4text`, and the corners of the area of usage, `point_sw` and `point_ne`.

Availability: 3.4.0

Proj version 6+

Examples

Get the metadata for EPSG:3005.

```
SELECT * FROM postgis_srs('EPSG', '3005');

auth_name | EPSG
auth_srid | 3005
srname    | NAD83 / BC Albers
srtext    | PROJCS["NAD83 / BC Albers", ... ]]
proj4text | +proj=aea +lat_0=45 +lon_0=-126 +lat_1=50 +lat_2=58.5 +x_0=1000000 +y_0=0 + ↩
    datum=NAD83 +units=m +no_defs +type=crs
point_sw  | 0101000020E6100000E17A14AE476161C00000000000204840
point_ne  | 0101000020E610000085EB51B81E855CC0E17A14AE47014E40
```

Siehe auch

postgis_srs_codes, postgis_srs_all, postgis_srs_search

7.7.8 postgis_srs_all

postgis_srs_all — Return metadata records for every spatial reference system in the underlying Proj database.

Synopsis

setof record **postgis_srs_all**(void);

Beschreibung

Returns a set of all metadata records in the underlying Proj database. The records will have the `auth_name`, `auth_srid`, `srname`, `srtext`, `proj4text`, and the corners of the area of usage, `point_sw` and `point_ne`.

Availability: 3.4.0

Proj version 6+

Examples

Get the first 10 metadata records from the Proj database.

```
SELECT auth_name, auth_srid, srname FROM postgis_srs_all() LIMIT 10;

 auth_name | auth_srid |                  srname
-----------+-----------+-----------------------------------------
 EPSG      | 2000      | Anguilla 1957 / British West Indies Grid
 EPSG      | 20004     | Pulkovo 1995 / Gauss-Kruger zone 4
 EPSG      | 20005     | Pulkovo 1995 / Gauss-Kruger zone 5
 EPSG      | 20006     | Pulkovo 1995 / Gauss-Kruger zone 6
 EPSG      | 20007     | Pulkovo 1995 / Gauss-Kruger zone 7
 EPSG      | 20008     | Pulkovo 1995 / Gauss-Kruger zone 8
 EPSG      | 20009     | Pulkovo 1995 / Gauss-Kruger zone 9
 EPSG      | 2001      | Antigua 1943 / British West Indies Grid
 EPSG      | 20010     | Pulkovo 1995 / Gauss-Kruger zone 10
 EPSG      | 20011     | Pulkovo 1995 / Gauss-Kruger zone 11
```

Siehe auch

postgis_srs_codes, postgis_srs, postgis_srs_search

7.7.9 postgis_srs_search

postgis_srs_search — Return metadata records for projected coordinate systems that have areas of useage that fully contain the bounds parameter.

Synopsis

setof record **postgis_srs_search**(geometry bounds, text auth_name=EPSG);

Beschreibung

Return a set of metadata records for projected coordinate systems that have areas of useage that fully contain the bounds parameter. Each record will have the `auth_name`, `auth_srid`, `srname`, `srtext`, `proj4text`, and the corners of the area of usage, `point_sw` and `point_ne`.

The search only looks for projected coordinate systems, and is intended for users to explore the possible systems that work for the extent of their data.

Availability: 3.4.0

Proj version 6+

Examples

Search for projected coordinate systems in Louisiana.

```
SELECT auth_name, auth_srid, srname,
  ST_AsText(point_sw) AS point_sw,
  ST_AsText(point_ne) AS point_ne
FROM postgis_srs_search('SRID=4326;LINESTRING(-90 30, -91 31)')
LIMIT 3;

 auth_name | auth_srid |                srname                |      point_sw       |       ↩
    point_ne
-----------+-----------+--------------------------------------+---------------------+---------------

 EPSG      | 2801      | NAD83(HARN) / Louisiana South        | POINT(-93.94 28.85) | POINT ↩
    (-88.75 31.07)
 EPSG      | 3452      | NAD83 / Louisiana South (ftUS)       | POINT(-93.94 28.85) | POINT ↩
    (-88.75 31.07)
 EPSG      | 3457      | NAD83(HARN) / Louisiana South (ftUS) | POINT(-93.94 28.85) | POINT ↩
    (-88.75 31.07)
```

Scan a table for max extent and find projected coordinate systems that might suit.

```
WITH ext AS (
  SELECT ST_Extent(geom) AS geom, Max(ST_SRID(geom)) AS srid
  FROM foo
)
SELECT auth_name, auth_srid, srname,
  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;
```

Siehe auch

postgis_srs_codes, postgis_srs_all, postgis_srs

7.8 Geometrische Konstruktoren

7.8.1 Well-known-Text (WKT) Repräsentation

7.8.1.1 ST_BdPolyFromText

ST_BdPolyFromText — Konstruiert ein Polygon aus einer beliebigen Ansammlung von geschlossenen Linienzügen, welche als MultiLineString in der Well-Known Text Darstellung vorliegen müssen.

Synopsis

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

Beschreibung

Konstruiert ein Polygon aus einer beliebigen Ansammlung von geschlossenen Linienzügen, welche als MultiLineString in der Well-Known Text Darstellung vorliegen müssen.

Note
Meldet einen Fehler, wenn es sich bei dem WKT nicht um einen MULTILINESTRING handelt. Meldet einen Fehler, wenn die Ausgabe ein MULTIPOLYGON ist; in diesem Fall verwenden Sie bitte ST_BdMPolyFromText, oder vergleichen ST_BuildArea() für einen PostGIS orientierten Ansatz.

This method implements the OGC Simple Features Implementation Specification for SQL 1.1. s3.2.6.2

Wird vom GEOS Modul ausgeführt

Verfügbarkeit: 1.1.0

Siehe auch

ST_BuildArea, ST_BdMPolyFromText

7.8.1.2 ST_BdMPolyFromText

ST_BdMPolyFromText — Konstruiert ein MultiPolygon aus einer beliebigen Ansammlung von geschlossenen Linienzügen, welche als MultiLineString in der Well-Known Text Darstellung vorliegen müssen.

Synopsis

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

Beschreibung

Konstruiert ein Polygon aus einer beliebigen Ansammlung von geschlossenen Linienzügen, Polygonen und MultiLineStrings, welche in der Well-Known Text Darstellung vorliegen müssen.

Note
Meldet einen Fehler wenn der WKT kein MULTILINESTRING ist. Erzwingt die MULTIPOLYGON Ausgabe sogar dann, wenn das Ergebnis nur aus einem einzelnen POLYGON besteht; verwenden Sie bitte ST_BdPolyFromText , wenn Sie sicher sind, daß nur ein einzelnes POLYGON entsteht, oder vegleichen Sie ST_BuildArea() für einen PostGIS orientierten Ansatz.

This method implements the OGC Simple Features Implementation Specification for SQL 1.1. s3.2.6.2

Wird vom GEOS Modul ausgeführt

Verfügbarkeit: 1.1.0

Siehe auch

ST_BuildArea, ST_BdPolyFromText

7.8.1.3 ST_GeogFromText

ST_GeogFromText — Gibt einen geographischen Datentyp aus einer Well-known-Text (WKT), oder einer erweiterten WKT (EWKT), Darstellung zurück.

Synopsis

geography **ST_GeogFromText**(text EWKT);

Beschreibung

Gibt ein geographisches Objekt in der Well-known-Text oder in der erweiterten Well-known-Text Darstellung zurück. Falls nicht angegeben, wird die SRID 4326 angenommen. Dies ist ein Alias für ST_GeographyFromText. Punkte werden immer in Form von Länge und Breite ausgedrückt.

Beispiele

```
--- converting lon lat coords to geography
ALTER TABLE sometable ADD COLUMN geog geography(POINT,4326);
UPDATE sometable SET geog = ST_GeogFromText('SRID=4326;POINT(' || lon || ' ' || lat || ')') ↩
    ;

--- specify a geography point using EPSG:4267, NAD27
SELECT ST_AsEWKT(ST_GeogFromText('SRID=4267;POINT(-77.0092 38.889588)'));
```

Siehe auch

ST_AsText, ST_GeographyFromText

7.8.1.4 ST_GeographyFromText

ST_GeographyFromText — Gibt einen geographischen Datentyp aus einer Well-known-Text (WKT), oder einer erweiterten WKT (EWKT), Darstellung zurück.

Synopsis

geography **ST_GeographyFromText**(text EWKT);

Beschreibung

Gibt ein geographisches Objekt in der Well-known-Text Darstellung zurück. Falls nicht angegeben, wird die SRID 4326 angenommen.

Siehe auch

ST_GeogFromText, ST_AsText

7.8.1.5 ST_GeomCollFromText

ST_GeomCollFromText — Erzeugt eine Sammelgeometrie mit der gegebenen SRID aus einer WKT-Kollektion. Wenn keine SRID angegeben ist, wird diese standardmäßig auf 0 gesetzt.

Synopsis

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

Beschreibung

Erzeugt eine Sammelgeometrie mit der gegebenen SRID aus einer Well-known-Text (WKT) Darstellung. Wenn keine SRID angegeben ist, wird diese standardmäßig auf 0 gesetzt.

OGC SPEC 3.2.6.2 - Die Option SRID stammt aus dem Konformitätstest

Gibt NULL zurück, wenn der WKT keine GEOMETRYCOLLECTION ist

Note
Verwenden Sie diese Funktion nicht, wenn Sie sich vollkommen sicher sind, dass ihre WKT Geometrie eine Sammelgeometrie ist. Sie ist langsamer als ST_GeomFromText, da sie einen zusätzlicher Validierungsschritt ausführt.

This method implements the OGC Simple Features Implementation Specification for SQL 1.1. s3.2.6.2

This method implements the SQL/MM specification.

Beispiele

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

Siehe auch

ST_GeomFromText, ST_SRID

7.8.1.6 ST_GeomFromEWKT

ST_GeomFromEWKT — Gibt einen spezifizierten ST_Geometry-Wert von einer erweiterten Well-known-Text Darstellung (EWKT) zurück.

Synopsis

geometry **ST_GeomFromEWKT**(text EWKT);

Beschreibung

Erzeugt ein PostGIS ST_Geometry Objekt aus der erweiterten OGC Well-known-Text (EWKT) Darstellung.

Note
EWKT ist kein Format des OGC Standards, sondern ein PostGIS eigenes Format, welches den Identifikator (SRID) des räumlichen Koordinatenreferenzsystem mit einbindet

Erweiterung: Mit 2.0.0 wurde die Unterstützung für polyedrische Oberflächen und TIN eingeführt.

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

This method supports Circular Strings and Curves.

This function supports Polyhedral surfaces.

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

Beispiele

```
SELECT ST_GeomFromEWKT('SRID=4269;LINESTRING(-71.160281 42.258729,-71.160837  ↩
    42.259113,-71.161144 42.25932)');
SELECT ST_GeomFromEWKT('SRID=4269;MULTILINESTRING((-71.160281 42.258729,-71.160837  ↩
    42.259113,-71.161144 42.25932))');

SELECT ST_GeomFromEWKT('SRID=4269;POINT(-71.064544 42.28787)');

SELECT ST_GeomFromEWKT('SRID=4269;POLYGON((-71.1776585052917  ↩
    42.3902909739571,-71.1776820268866 42.3903701743239,
-71.1776063012595 42.3903825660754,-71.1775826583081 42.3903033653531,-71.1776585052917  ↩
    42.3902909739571))');

SELECT ST_GeomFromEWKT('SRID=4269;MULTIPOLYGON(((-71.1031880899493 42.3152774590236,
-71.1031627617667 42.3152960829043,-71.102923838298 42.3149156848307,
-71.1023097974109 42.3151969047397,-71.1019285062273 42.3147384934248,
-71.102505233663 42.3144722937587,-71.10277487471 42.3141658254797,
-71.103113945163 42.3142739188902,-71.10324876416 42.31402489987,
-71.1033002961013 42.3140393340215,-71.1033488797549 42.3139495090772,
-71.103396240451 42.3138632439557,-71.1041521907712 42.3141153348029,
-71.1041411411543 42.3141545014533,-71.1041287795912 42.3142114839058,
-71.1041188134329 42.3142693656241,-71.1041112482575 42.3143272556118,
-71.1041072845732 42.3143851580048,-71.1041057218871 42.3144430686681,
-71.1041065602059 42.3145009876017,-71.1041097995362 42.3145589148055,
-71.1041166403905 42.3146168544148,-71.1041258822717 42.3146748022936,
-71.1041375307579 42.3147318674446,-71.1041492906949 42.3147711126569,
-71.1041598612795 42.314808571739,-71.1042515013869 42.3151287620809,
-71.1041173835118 42.3150739481917,-71.1040809891419 42.3151344119048,
-71.1040438678912 42.3151191367447,-71.1040194562988 42.3151832057859,
-71.1038734225584 42.3151140942995,-71.1038446938243 42.3151006300338,
-71.1038315271889 42.315094347535,-71.1037393329282 42.315054824985,
-71.1035447555574 42.3152608696313,-71.1033436658644 42.3151648370544,
-71.1032580383161 42.3152269126061,-71.103223066939 42.3152517403219,
```

```
-71.1031880899493 42.3152774590236)),
((-71.1043632495873 42.315113108546,-71.1043583974082 42.3151211109857,
-71.1043443253471 42.3150676015829,-71.1043850704575 42.3150793250568,-71.1043632495873 ↩
    42.315113108546)))');
```

```
--3d circular string
SELECT ST_GeomFromEWKT('CIRCULARSTRING(220268 150415 1,220227 150505 2,220227 150406 3)');
```

```
--Polyhedral Surface example
SELECT ST_GeomFromEWKT('POLYHEDRALSURFACE(
        ((0 0 0, 0 0 1, 0 1 1, 0 1 0, 0 0 0)),
        ((0 0 0, 0 1 0, 1 1 0, 1 0 0, 0 0 0)),
        ((0 0 0, 1 0 0, 1 0 1, 0 0 1, 0 0 0)),
        ((1 1 0, 1 1 1, 1 0 1, 1 0 0, 1 1 0)),
        ((0 1 0, 0 1 1, 1 1 1, 1 1 0, 0 1 0)),
        ((0 0 1, 1 0 1, 1 1 1, 0 1 1, 0 0 1))
)');
```

Siehe auch

ST_AsEWKT, ST_GeomFromText

7.8.1.7 ST_GeomFromMARC21

ST_GeomFromMARC21 — Takes MARC21/XML geographic data as input and returns a PostGIS geometry object.

Synopsis

geometry **ST_GeomFromMARC21** (text marcxml);

Beschreibung

This function creates a PostGIS geometry from a MARC21/XML record, which can contain a `POINT` or a `POLYGON`. In case of multiple geographic data entries in the same MARC21/XML record, a `MULTIPOINT` or `MULTIPOLYGON` will be returned. If the record contains mixed geometry types, a `GEOMETRYCOLLECTION` will be returned. It returns NULL if the MARC21/XML record does not contain any geographic data (datafield:034).

LOC MARC21/XML versions supported:

- MARC21/XML 1.1

Availability: 3.3.0, requires libxml2 2.6+

Note
The MARC21/XML Coded Cartographic Mathematical Data currently does not provide any means to describe the Spatial Reference System of the encoded coordinates, so this function will always return a geometry with `SRID 0`.

Note
Returned `POLYGON` geometries will always be clockwise oriented.

Beispiele

Converting MARC21/XML geographic data containing a single `POINT` encoded as `hddd.dddddd`

```
SELECT
                ST_AsText(
                    ST_GeomFromMARC21('
                        <record xmlns="http://www.loc.gov/MARC21/slim">
                            <leader
>00000nz a2200000nc 4500</leader>
                            <controlfield tag="001"
>040277569</controlfield>
                            <datafield tag="034" ind1=" " ind2=" ">
                                <subfield code="d"
>W004.500000</subfield>
                                <subfield code="e"
>W004.500000</subfield>
                                <subfield code="f"
>N054.250000</subfield>
                                <subfield code="g"
>N054.250000</subfield>
                            </datafield>
                        </record
>'));

                st_astext
                -------------------
                POINT(-4.5 54.25)
                (1 row)
```

Converting MARC21/XML geographic data containing a single `POLYGON` encoded as `hdddmmss`

```
SELECT
                ST_AsText(
                    ST_GeomFromMARC21('
                        <record xmlns="http://www.loc.gov/MARC21/slim">
                            <leader
>01062cem a2200241 a 4500</leader>
                            <controlfield tag="001"
>   84696781 </controlfield>
                            <datafield tag="034" ind1="1" ind2=" ">
                                <subfield code="a"
>a</subfield>
                                <subfield code="b"
>50000</subfield>
                                <subfield code="d"
>E0130600</subfield>
                                <subfield code="e"
>E0133100</subfield>
                                <subfield code="f"
>N0523900</subfield>
                                <subfield code="g"
>N0522300</subfield>
                            </datafield>
                        </record
>'));

                st_astext
                ----------------------------------------------------------------------------------

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

```
                (1 row)
```

Converting MARC21/XML geographic data containing a `POLYGON` and a `POINT`:

```
SELECT
                ST_AsText(
                    ST_GeomFromMARC21('
                <record xmlns="http://www.loc.gov/MARC21/slim">
                    <datafield tag="034" ind1="1" ind2=" ">
                        <subfield code="a"
>a</subfield>
                        <subfield code="b"
>50000</subfield>
                        <subfield code="d"
>E0130600</subfield>
                        <subfield code="e"
>E0133100</subfield>
                        <subfield code="f"
>N0523900</subfield>
                        <subfield code="g"
>N0522300</subfield>
                    </datafield>
                    <datafield tag="034" ind1=" " ind2=" ">
                        <subfield code="d"
>W004.500000</subfield>
                        <subfield code="e"
>W004.500000</subfield>
                        <subfield code="f"
>N054.250000</subfield>
                        <subfield code="g"
>N054.250000</subfield>
                    </datafield>
                </record
>'));
                                                                                st_astext ↩

                ----------------------------------------------------------------------------------

                GEOMETRYCOLLECTION(POLYGON((13.1 52.65,13.516666666666667  ↩
                    52.65,13.516666666666667 52.38333333333333,13.1 52.38333333333333,13.1  ↩
                    52.65)),POINT(-4.5 54.25))
                (1 row)
```

Siehe auch

ST_AsMARC21

7.8.1.8 ST_GeometryFromText

ST_GeometryFromText — Gibt einen spezifizierten ST_Geometry-Wert von einer Well-known-Text Darstellung (WKT) zurück. Die Bezeichnung ist ein Alias für ST_GeomFromText

Synopsis

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

Beschreibung

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

This method implements the SQL/MM specification. SQL-MM 3: 5.1.40

Siehe auch

ST_GeomFromText

7.8.1.9 ST_GeomFromText

ST_GeomFromText — Gibt einen spezifizierten ST_Geometry Wert aus einer Well-known-Text Darstellung (WKT) zurück.

Synopsis

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

Beschreibung

Erzeugt ein PostGIS ST_Geometry Objekt aus der OGC Well-known-Text Darstellung.

Note
Die Funktion ST_GeomFromText hat zwei Varianten. Die erste Variante nimmt keine SRID entgegen und gibt eine Geometrie ohne ein bestimmtes Koordinatenreferenzsystem aus (SRID=0) . Die zweite Variante nimmt eine SRID als zweiten Übergabewert entgegen und gibt eine Geometrie zurück, die diese SRID als Teil ihrer Metadaten beinhaltet.

This method implements the OGC Simple Features Implementation Specification for SQL 1.1. s3.2.6.2 - die Option SRID ist vom Konformitätstest.

This method implements the SQL/MM specification. SQL-MM 3: 5.1.40

This method supports Circular Strings and Curves.

Note
While not OGC-compliant, ST_MakePoint is faster than ST_GeomFromText and ST_PointFromText. It is also easier to use for numeric coordinate values. ST_Point is another option similar in speed to ST_MakePoint and is OGC-compliant, but doesn't support anything but 2D points.

Warning
Änderung: 2.0.0 - In Vorgängerversionen von PostGIS war ST_GeomFromText('GEOMETRYCOLLECTION(EMPTY)') erlaubt. Um eine bessere Übereinstimmung mit der SQL/MM Norm zu erreichen, ist dies in PostGIS 2.0.0 nun nicht mehr gestattet. Hier sollte nun ST_GeomFromText('GEOMETRYCOLLECTION EMPTY') geschrieben werden.

Beispiele

```
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)');
```

Siehe auch

ST_GeomFromEWKT, ST_GeomFromWKB, ST_SRID

7.8.1.10 ST_LineFromText

ST_LineFromText — Erzeugt eine Geometrie aus einer WKT Darstellung mit der angegebenen SRID. Wenn keine SRID angegeben wird, wird diese standardmäßig auf 0 gesetzt.

Synopsis

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

Beschreibung

Erzeugt eine Geometrie aus einer WKT Darstellung mit der angegebenen SRID. Wenn keine SRID angegeben wird, wird diese standardmäßig auf 0 gesetzt. Wenn das übergebene WKT kein LineString ist, wird NULL zurückgegeben.

Note
OGC SPEC 3.2.6.2 - die Option SRID ist vom Konformitätstest.

Note
Wenn Sie wissen, dass die Geometrie nur aus LINESTRINGs besteht, ist es effizienter einfach ST_GeomFromText zu verwenden. Diese Funktion ruft auch nur ST_GeomFromText auf und fügt die Information hinzu, dass es sich um einen Linienzug handelt.

This method implements the OGC Simple Features Implementation Specification for SQL 1.1. s3.2.6.2

This method implements the SQL/MM specification. SQL-MM 3: 7.2.8

Beispiele

```
SELECT ST_LineFromText('LINESTRING(1 2, 3 4)') AS aline, ST_LineFromText('POINT(1 2)') AS  ↩
    null_return;
aline                              | null_return
-----------------------------------------------
0102000000020000000000000000F ... | t
```

Siehe auch

ST_GeomFromText

7.8.1.11 ST_MLineFromText

ST_MLineFromText — Liest einen festgelegten ST_MultiLineString Wert von einer WKT-Darstellung aus.

Synopsis

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

Beschreibung

Erzeugt eine Geometrie aus einer Well-known-Text (WKT) Darstellung mit der angegebenen SRID. Wenn keine SRID angegeben wird, wird diese standardmäßig auf 0 gesetzt.

OGC SPEC 3.2.6.2 - Die Option SRID stammt aus dem Konformitätstest

Gibt NULL zurück wenn der WKT kein MULTILINESTRING ist.

Note
Verwenden Sie diese Funktion nicht, wenn Sie sich vollkommen sicher sind, dass ihre WKT Geometrie nur aus Punkten besteht. Sie ist langsamer als ST_GeomFromText, da sie einen zusätzlichen Validierungsschritt hinzufügt.

This method implements the OGC Simple Features Implementation Specification for SQL 1.1. s3.2.6.2

This method implements the SQL/MM specification. SQL-MM 3: 9.4.4

Beispiele

```
SELECT ST_MLineFromText('MULTILINESTRING((1 2, 3 4), (4 5, 6 7))');
```

Siehe auch

ST_GeomFromText

7.8.1.12 ST_MPointFromText

ST_MPointFromText — Erzeugt eine Geometrie aus WKT mit der angegebenen SRID. Wenn keine SRID angegeben wird, wird diese standardmäßig auf 0 gesetzt.

Synopsis

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

Beschreibung

Erzeugt eine Geometrie aus WKT mit der angegebenen SRID. Wenn SRID nicht angegeben ist, wird sie standardmäßig auf 0 gesetzt.

OGC SPEC 3.2.6.2 - Die Option SRID stammt aus dem Konformitätstest

Gibt NULL zurück, wenn der WKT kein MULTIPOINT ist.

Note
Verwenden Sie diese Funktion nicht, wenn Sie sich vollkommen sicher sind, dass ihre WKT Geometrie nur aus Punkten besteht. Sie ist langsamer als ST_GeomFromText, da sie einen zusätzlichen Validierungsschritt hinzufügt.

This method implements the OGC Simple Features Implementation Specification for SQL 1.1. 3.2.6.2

This method implements the SQL/MM specification. SQL-MM 3: 9.2.4

Beispiele

```
SELECT ST_MPointFromText('MULTIPOINT((1 2),(3 4))');
SELECT ST_MPointFromText('MULTIPOINT((-70.9590 42.1180),(-70.9611 42.1223))', 4326);
```

Siehe auch

ST_GeomFromText

7.8.1.13 ST_MPolyFromText

ST_MPolyFromText — Erzeugt eine MultiPolygon Geometrie aus WKT mit der angegebenen SRID. Wenn SRID nicht angegeben ist, wird sie standardmäßig auf 0 gesetzt.

Synopsis

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

Beschreibung

Erzeugt ein MultiPolygon von WKT mit der gegebenen SRID. Wenn SRID nicht angegeben ist, wird sie standardmäßig auf 0 gesetzt.

OGC SPEC 3.2.6.2 - Die Option SRID stammt aus dem Konformitätstest

Meldet einen Fehler, wenn der WKT kein MULTIPOLYGON ist.

Note
Verwenden Sie diese Funktion nicht, wenn Sie sich vollkommen sicher sind, dass ihre WKT Geometrie nur aus Multi-Polygonen besteht. Sie ist langsamer als ST_GeomFromText, da sie einen zusätzlichen Validierungsschritt hinzufügt.

This method implements the OGC Simple Features Implementation Specification for SQL 1.1. s3.2.6.2

This method implements the SQL/MM specification. SQL-MM 3: 9.6.4

Beispiele

```
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);
```

Siehe auch

ST_GeomFromText, ST_SRID

7.8.1.14 ST_PointFromText

ST_PointFromText — Erzeugt eine Punktgeometrie mit gegebener SRID von WKT. Wenn SRID nicht angegeben ist, wird sie standardmäßig auf 0 gesetzt.

Synopsis

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

Beschreibung

Erzeugt ein PostGIS ST_Geometrie Punktobjekt von der OGC Well-known-Text Darstellung. Wenn die SRID nicht angegeben ist, wird sie standardmäßig auf "unknown" (zurzeit 0) gesetzt. Falls die Geometrie nicht in der WKT Punktdarstellung vorliegt, wird NULL zurückgegeben. Bei einer invaliden WKT Darstellung wird eine Fehlermeldung angezeigt.

Note
Die Funktion ST_PointFromText hat zwei Varianten. Die erste Variante nimmt keine SRID entgegen und gibt eine Geometrie ohne ein bestimmtes Koordinatenreferenzsystem aus. Die zweite Variante nimmt eine SRID als zweiten Übergabewert entgegen und gibt eine Geometrie zurück, die diese SRID als Teil ihrer Metadaten beinhaltet. Die SRID muss in der Tabelle "spatial_ref_sys" definiert sein.

Note
Verwenden Sie diese Funktion nicht, wenn Sie sich vollkommen sicher sind, dass ihre WKT Geometrie nur aus Punkten besteht. Sie ist langsamer als ST_GeomFromText, da sie einen zusätzlichen Validierungsschritt hinzufügt. Wenn Sie Punkte aus Koordinaten in Länge und Breite erstellen und mehr auf Rechenleistung und Genauigkeit wertlegen als auf OGC-Konformität, so verwenden Sie bitte ST_MakePoint oder den OGC-konformen Alias ST_Point.

This method implements the OGC Simple Features Implementation Specification for SQL 1.1. s3.2.6.2 - die Option SRID ist vom Konformitätstest.

This method implements the SQL/MM specification. SQL-MM 3: 6.1.8

Beispiele

```
SELECT ST_PointFromText('POINT(-71.064544 42.28787)');
SELECT ST_PointFromText('POINT(-71.064544 42.28787)', 4326);
```

Siehe auch

ST_GeomFromText, ST_MakePoint, ST_Point, ST_SRID

7.8.1.15 ST_PolygonFromText

ST_PolygonFromText — Erzeugt eine Geometrie aus WKT mit der angegebenen SRID. Wenn keine SRID angegeben wird, wird diese standardmäßig auf 0 gesetzt.

Synopsis

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

Beschreibung

Erzeugt eine Geometrie mit gegebener SRID von WKT. Wenn SRID nicht angegeben ist, wird sie standardmäßig auf 0 gesetzt. Gibt NULL zurück, wenn WKT kein Polygon ist.

OGC SPEC 3.2.6.2 - Die Option SRID stammt aus dem Konformitätstest

Note
Verwenden Sie diese Funktion nicht, wenn Sie sich vollkommen sicher sind, dass ihre WKT Geometrie nur aus Polygonen besteht. Sie ist langsamer als ST_GeomFromText, da sie einen zusätzlichen Validierungsschritt hinzufügt.

This method implements the OGC Simple Features Implementation Specification for SQL 1.1. s3.2.6.2

This method implements the SQL/MM specification. SQL-MM 3: 8.3.6

Beispiele

```
SELECT ST_PolygonFromText('POLYGON((-71.1776585052917 42.3902909739571,-71.1776820268866  ↩
    42.3903701743239,
-71.1776063012595 42.3903825660754,-71.1775826583081 42.3903033653531,-71.1776585052917  ↩
    42.3902909739571))');
st_polygonfromtext
------------------
0103000000010000000050000006...


SELECT ST_PolygonFromText('POINT(1 2)') IS NULL as point_is_notpoly;

point_is_not_poly
----------
t
```

Siehe auch

ST_GeomFromText

7.8.1.16 ST_WKTToSQL

ST_WKTToSQL — Gibt einen spezifizierten ST_Geometry-Wert von einer Well-known-Text Darstellung (WKT) zurück. Die Bezeichnung ist ein Alias für ST_GeomFromText

Synopsis

geometry **ST_WKTToSQL**(text WKT);

Beschreibung

This method implements the SQL/MM specification. SQL-MM 3: 5.1.34

Siehe auch

ST_GeomFromText

7.8.2 Well-known-Binary (WKB) Repräsentation

7.8.2.1 ST_GeogFromWKB

ST_GeogFromWKB — Erzeugt ein geographisches Objekt aus der Well-known-Binary (WKB) oder der erweiterten Well-known-Binary (EWKB) Darstellung.

Synopsis

geography **ST_GeogFromWKB**(bytea wkb);

Beschreibung

Die Funktion `ST_GeogFromWKB` empfängt eine Well-known-Binary (WKB) oder eine erweiterte PostGIS WKB (EWKB) Darstellung einer Geometrie und erzeugt eine Instanz des entsprechenden geographischen Datentyps. Diese Funktion übernimmt die Rolle der Geometrie-Factory/Fabrik in SQL.

Wenn die SRID nicht festgelegt ist, wird 4326 (WGS 84) angenommen.

This method supports Circular Strings and Curves.

Beispiele

```
--Although bytea rep contains single \, these need to be escaped when inserting into a  ↩
    table
SELECT ST_AsText(
ST_GeogFromWKB(E'\\001\\002\\000\\000\\000\\002\\000\\000\\000\\037\\205\\353Q ↩
    \\270~\\\\\\300\\323Mb\\020X\\231C@\\020X9\\264\\310~\\\\\\300)\\\\\\217\\302\\365\\230 ↩
    C@')
);
                                          st_astext
-----------------------------------------------------
 LINESTRING(-113.98 39.198,-113.981 39.195)
(1 row)
```

Siehe auch

ST_GeogFromText, ST_AsBinary

7.8.2.2 ST_GeomFromEWKB

ST_GeomFromEWKB — Gibt einen geometrischen Datentyp (ST_Geometry) aus einer Well-known-Binary (WKB) Darstellung zurück.

Synopsis

geometry **ST_GeomFromEWKB**(bytea EWKB);

Beschreibung

Erzeugt ein PostGIS ST_Geometry Objekt aus der erweiterten OGC Well-known-Text (EWKT) Darstellung.

Note
EWKB ist kein Format des OGC Standards, sondern ein PostGIS eigenes Format, welches den Identifikator (SRID) des räumlichen Koordinatenreferenzsystem mit einbindet

Erweiterung: Mit 2.0.0 wurde die Unterstützung für polyedrische Oberflächen und TIN eingeführt.

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

This method supports Circular Strings and Curves.

This function supports Polyhedral surfaces.

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

Beispiele

Binärdarstellung des Linienzuges LINESTRING(-71.160281 42.258729,-71.160837 42.259113,-71.161144 42.25932) in NAD 83 (4269).

Note
ANMERKUNG: Obwohl die Bytefelder durch \ getrennt sind und auch ' aufweisen können, müssen wir beides mit \ maskieren; wenn "standard_conforming_strings" ausgeschaltet ist mit ". Somit sieht diese Darstellung nicht genauso wie die AsEWKB Darstellung aus.

```
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
Ab PostgreSQL 9.1 - ist standard_conforming_strings standardmäßig auf "on" gesetzt. Bei Vorgängerversionen war es "off". Sie können die Standardvorgaben für eine einzelne Abfrage ändern oder auf Datenbank- oder Serverebene setzen. Unterhalb steht, wie Sie dies mit standard_conforming_strings = on umsetzten können. In diesem Fall maskieren wir das ' mit dem ANSI Zeichen ', aber Schrägstriche werden nicht maskiert

```
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')
```

Siehe auch

ST_AsBinary, ST_AsEWKB, ST_GeomFromWKB

7.8.2.3 ST_GeomFromWKB

ST_GeomFromWKB — Erzeugt ein geometrisches Objekt aus der Well-known-Binary (WKB) Darstellung und einer optionalen SRID.

Synopsis

geometry **ST_GeomFromWKB**(bytea geom);
geometry **ST_GeomFromWKB**(bytea geom, integer srid);

Beschreibung

Die Funktion `ST_GeogFromWKB` nimmt eine Well-known-Binary (WKB) Darstellung und eine Id für das Koordinatenreferenzsystem (`SRID`) entgegen und erzeugt eine Instanz des entsprechenden geometrischen Datentyps. Diese Funktion übernimmt die Rolle der Geometrie-Factory in SQL. Ist eine alternative Bezeichnung für ST_WKBToSQL.

Wenn die SRID nicht festgelegt ist, wird sie standardmäßig auf 0 (Unknown) gesetzt.

This method implements the OGC Simple Features Implementation Specification for SQL 1.1. s3.2.7.2 - die optionale SRID kommt vom Konformitätstest.

This method implements the SQL/MM specification. SQL-MM 3: 5.1.41

This method supports Circular Strings and Curves.

Beispiele

```
--Although bytea rep contains single \, these need to be escaped when inserting into a  ↩
    table
                -- unless standard_conforming_strings is set to on.
SELECT ST_AsEWKT(
ST_GeomFromWKB(E'\\001\\002\\000\\000\\000\\002\\000\\000\\000\\037\\205\\353Q ↩
    \\270~\\\\\\300\\323Mb\\020X\\231C@\\020X9\\264\\310~\\\\\\300)\\\\\\217\\302\\365\\230 ↩
    C@',4326)
);
                                          st_asewkt
-----------------------------------------------------
 SRID=4326;LINESTRING(-113.98 39.198,-113.981 39.195)
(1 row)

SELECT
  ST_AsText(
        ST_GeomFromWKB(
          ST_AsEWKB('POINT(2 5)'::geometry)
        )
  );
 st_astext
------------
 POINT(2 5)
(1 row)
```

Siehe auch

ST_WKBToSQL, ST_AsBinary, ST_GeomFromEWKB

7.8.2.4 ST_LineFromWKB

ST_LineFromWKB — Erzeugt einen LINESTRING mit gegebener SRID aus einer WKB-Darstellung

Synopsis

geometry **ST_LineFromWKB**(bytea WKB);
geometry **ST_LineFromWKB**(bytea WKB, integer srid);

Beschreibung

Die Funktion ST_GeogFromWKB nimmt eine Well-known-Binary Darstellung der Geometrie und eine Id für das Koordinatenreferenzsystem (SRID) entgegen und erzeugt eine Instanz des entsprechenden geometrischen Datentyps - in diesem Fall eine Geometrie vom Typ LineString. Diese Funktion übernimmt die Rolle der Geometrie-Factory in SQL.

Wenn keine SRID angegeben ist, wird diese auf 0 gesetzt. NULL wird zurückgegeben, wenn die Eingabe bytea keinen LINESTRING darstellt.

Note
OGC SPEC 3.2.6.2 - die Option SRID ist vom Konformitätstest.

Note
Wenn Sie wissen, dass Ihre Geometrie nur aus LINESTRINGs besteht, ist es effizienter einfach ST_GeomFromWKB zu verwenden. Diese Funktion ruft auch nur ST_GeomFromWKB auf und fügt die Information hinzu, dass es sich um einen Linienzug handelt.

This method implements the OGC Simple Features Implementation Specification for SQL 1.1. s3.2.6.2

This method implements the SQL/MM specification. SQL-MM 3: 7.2.9

Beispiele

```
SELECT ST_LineFromWKB(ST_AsBinary(ST_GeomFromText('LINESTRING(1 2, 3 4)'))) AS aline,
                ST_LineFromWKB(ST_AsBinary(ST_GeomFromText('POINT(1 2)'))) IS NULL AS  ↩
                    null_return;
aline                              | null_return
-----------------------------------------------
0102000000020000000000000000F ... | t
```

Siehe auch

ST_GeomFromWKB, ST_LinestringFromWKB

7.8.2.5 ST_LinestringFromWKB

ST_LinestringFromWKB — Erzeugt eine Geometrie mit gegebener SRID aus einer WKB-Darstellung.

Synopsis

geometry **ST_LinestringFromWKB**(bytea WKB);
geometry **ST_LinestringFromWKB**(bytea WKB, integer srid);

Beschreibung

Die Funktion `ST_LinestringFromWKB` nimmt eine Well-known-Binary Darstellung der Geometrie und eine Id für das Koordinatenreferenzsystem (`SRID`) entgegen und erzeugt eine Instanz des entsprechenden geometrischen Datentyps - in diesem Fall eine Geometrie vom Typ `LineString`. Diese Funktion übernimmt die Rolle der Geometrie-Factory in SQL.

Wenn keine SRID angegeben ist, wird diese standardmäßig auf 0 gesetzt. `NULL` wird zurückgegeben, wenn die Eingabe `bytea` keinen `LINESTRING` darstellt. Ist ein Alias für ST_LineFromWKB.

Note
OGC SPEC 3.2.6.2 - optionale SRID ist vom Konformitätstest.

Note
Wenn Sie wissen, dass Ihre Geometrie nur aus `LINESTRING`s besteht, ist es effizienter einfach ST_GeomFromWKB zu verwenden. Diese Funktion ruft auch nur ST_GeomFromWKB auf und fügt die Information hinzu, dass es sich um einen `LINESTRING` handelt.

This method implements the OGC Simple Features Implementation Specification for SQL 1.1. s3.2.6.2

This method implements the SQL/MM specification. SQL-MM 3: 7.2.9

Beispiele

```
SELECT
  ST_LineStringFromWKB(
        ST_AsBinary(ST_GeomFromText('LINESTRING(1 2, 3 4)'))
  ) AS aline,
  ST_LinestringFromWKB(
        ST_AsBinary(ST_GeomFromText('POINT(1 2)'))
  ) IS NULL AS null_return;
   aline                           | null_return
-------------------------------------------------
0102000000020000000000000000F ... | t
```

Siehe auch

ST_GeomFromWKB, ST_LineFromWKB

7.8.2.6 ST_PointFromWKB

ST_PointFromWKB — Erzeugt eine Geometrie mit gegebener SRID von WKB.

Synopsis

geometry **ST_GeomFromWKB**(bytea geom);
geometry **ST_GeomFromWKB**(bytea geom, integer srid);

Beschreibung

Die Funktion `ST_PointFromWKB` nimmt eine Well-known-Binary Darstellung der Geometrie und eine Id für das Koordinatenreferenzsystem (`SRID`) entgegen und erzeugt eine Instanz des entsprechenden geometrischen Datentyps - in diesem Fall eine Geometrie vom Typ `POINT`. Diese Funktion übernimmt die Rolle der Geometrie-Factory in SQL.

Wenn keine SRID angegeben ist, wird diese standardmäßig auf 0 gesetzt. `NULL` wird zurückgegeben, wenn die Eingabe `bytea` keinen `POINT` darstellt.

This method implements the OGC Simple Features Implementation Specification for SQL 1.1. s3.2.7.2

This method implements the SQL/MM specification. SQL-MM 3: 6.1.9

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

This method supports Circular Strings and Curves.

Beispiele

```
SELECT
  ST_AsText(
        ST_PointFromWKB(
          ST_AsEWKB('POINT(2 5)'::geometry)
        )
  );
 st_astext
------------
 POINT(2 5)
(1 row)

SELECT
  ST_AsText(
        ST_PointFromWKB(
          ST_AsEWKB('LINESTRING(2 5, 2 6)'::geometry)
        )
  );
 st_astext
-----------

(1 row)
```

Siehe auch

ST_GeomFromWKB, ST_LineFromWKB

7.8.2.7 ST_WKBToSQL

ST_WKBToSQL — Gibt einen geometrischen Datentyp (ST_Geometry) aus einer Well-known-Binary (WKB) Darstellung zurück. Ein Synonym für ST_GeomFromWKB, welches jedoch keine SRID annimmt

Synopsis

geometry **ST_WKBToSQL**(bytea WKB);

Beschreibung

This method implements the SQL/MM specification. SQL-MM 3: 5.1.36

Siehe auch

ST_GeomFromWKB

7.8.3 Weitere Formate

7.8.3.1 ST_Box2dFromGeoHash

ST_Box2dFromGeoHash — Gibt die BOX2D einer GeoHash Zeichenkette zurück.

Synopsis

box2d **ST_Box2dFromGeoHash**(text geohash, integer precision=full_precision_of_geohash);

Beschreibung

Gibt die BOX2D einer GeoHash Zeichenkette zurück.

If no `precision` is specified ST_Box2dFromGeoHash returns a BOX2D based on full precision of the input GeoHash string.

Wenn `precision` angegeben wird, verwendet ST_Box2dFromGeoHash entsprechend viele Zeichen des GeoHash um die BOX2D zu erzeugen. Niedrigere Werte erzeugen eine größere BOX2D und höhere Werte erhöhen die Genauigkeit.

Verfügbarkeit: 2.1.0

Beispiele

```
SELECT ST_Box2dFromGeoHash('9qqj7nmxncgyy4d0dbxqz0');

                st_geomfromgeohash
-------------------------------------------------
 BOX(-115.172816 36.114646,-115.172816 36.114646)

SELECT ST_Box2dFromGeoHash('9qqj7nmxncgyy4d0dbxqz0', 0);

 st_box2dfromgeohash
----------------------
 BOX(-180 -90,180 90)

 SELECT ST_Box2dFromGeoHash('9qqj7nmxncgyy4d0dbxqz0', 10);
                           st_box2dfromgeohash
-------------------------------------------------------------------------
 BOX(-115.17282128334 36.1146408319473,-115.172810554504 36.1146461963654)
```

Siehe auch

ST_GeoHash, ST_GeomFromGeoHash, ST_PointFromGeoHash

7.8.3.2 ST_GeomFromGeoHash

ST_GeomFromGeoHash — Gibt die Geometrie einer GeoHash Zeichenfolge zurück.

Synopsis

geometry **ST_GeomFromGeoHash**(text geohash, integer precision=full_precision_of_geohash);

Beschreibung

Gibt die Geometrie einer GeoHash Zeichenfolge zurück. Der geometrische Datentyp ist ein Polygon, das den GeoHash begrenzt.

Wenn keine `precision` angegeben wird, dann gibt ST_GeomFromGeoHash ein Polygon zurück, das auf der vollständigen Genauigkeit der GeoHash Zeichenfolge beruht.

Wenn `precision` angegeben wird, verwendet ST_GeomFromGeoHash entsprechend viele Zeichen des GeoHash, um das Polygon zu erzeugen.

Verfügbarkeit: 2.1.0

Beispiele

```
SELECT ST_AsText(ST_GeomFromGeoHash('9qqj7nmxncgyy4d0dbxqz0'));
                                                    st_astext
------------------------------------------------------------------------------------------------------------

 POLYGON((-115.172816 36.114646,-115.172816 36.114646,-115.172816 36.114646,-115.172816  ↩
     36.114646,-115.172816 36.114646))

SELECT ST_AsText(ST_GeomFromGeoHash('9qqj7nmxncgyy4d0dbxqz0', 4));
                                                      st_astext
------------------------------------------------------------------------------------------------------------

 POLYGON((-115.3125 36.03515625,-115.3125 36.2109375,-114.9609375 36.2109375,-114.9609375  ↩
     36.03515625,-115.3125 36.03515625))

SELECT ST_AsText(ST_GeomFromGeoHash('9qqj7nmxncgyy4d0dbxqz0', 10));
                                                                                st_astext ↩

------------------------------------------------------------------------------------------------------------

 POLYGON((-115.17282128334 36.1146408319473,-115.17282128334  ↩
     36.1146461963654,-115.172810554504 36.1146461963654,-115.172810554504  ↩
     36.1146408319473,-115.17282128334 36.1146408319473))
```

Siehe auch

ST_GeoHash,ST_Box2dFromGeoHash, ST_PointFromGeoHash

7.8.3.3 ST_GeomFromGML

ST_GeomFromGML — Nimmt als Eingabe eine GML-Darstellung der Geometrie und gibt ein geometrisches PostGIS-Objekt aus.

Synopsis

geometry **ST_GeomFromGML**(text geomgml);
geometry **ST_GeomFromGML**(text geomgml, integer srid);

Beschreibung

Erzeugt ein PostGIS ST_Geometry Objekt aus der OGC GML Darstellung.

ST_GeomFromGML funktioniert nur bei Fragmenten von GML-Geometrien. Auf das ganze GML-Dokument angewendet führt zu einer Fehlermeldung.

Unterstützte OGC GML Versionen:

- GML 3.2.1 Namespace
- GML 3.1.1 Simple Features profile SF-2 (inkl. GML 3.1.0 und 3.0.0 Rückwertskompatibilität)
- GML 2.1.2

OGC GML Standards, vgl.: http://www.opengeospatial.org/standards/gml:

Verfügbarkeit: 1.5, benötigt libxml2 1.6+

Erweiterung: Mit 2.0.0 wurde die Unterstützung für polyedrische Oberflächen und TIN eingeführt.

Erweiterung: 2.0.0 Standardwert für den optionalen Parameter SRID eingefügt.

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

This function supports Polyhedral surfaces.

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

GML erlaubt das Mischen von Dimensionen (z.B. 2D und 3D innerhalb der selben MultiGeometry). Da PostGIS Geometrien dies nicht zulassen, wandelt ST_GeomFromGML die gesamte Geometrie in 2D um, sobald eine fehlende Z-Dimension existiert.

GML unterstützt uneinheitliche Koordinatenreferenzsysteme innerhalb derselben Mehrfachgeometrie. Da dies der geometrische Datentyp von PostGIS nicht unterstützt, wird in diesem Fall die Subgeometrie in das Referenzsystem des Knotens, der die Wurzel darstellt, umprojiziert. Wenn kein Attribut "srsName" für den Knoten der GML-Wurzel vorhanden ist, gibt die Funktion eine Fehlermeldung aus.

Die Funktion ST_GeomFromGML ist nicht kleinlich, was die explizite Vergabe eines GML-Namensraums betrifft. Bei üblichen Anwendungen können Sie die explizite Vergabe weglassen. Wenn Sie aber das XLink Feature von GML verwenden wollen, müssen Sie den Namensraum explizit angeben.

Note
SQL/MM Kurvengeometrien werden von der Funktion ST_GeomFromGML nicht unterstützt

---

Beispiele - Eine Einzelgeometrie mit einem srsName

```
SELECT ST_GeomFromGML('
                <gml:LineString srsName="EPSG:4269">
                        <gml:coordinates>
                                -71.16028,42.258729 -71.160837,42.259112  ↩
                                    -71.161143,42.25932
                        </gml:coordinates>
                </gml:LineString
>');
```

Beispiele - Verwendung von XLink

```
SELECT ST_GeomFromGML('
                <gml:LineString xmlns:gml="http://www.opengis.net/gml"
                                xmlns:xlink="http://www.w3.org/1999/xlink"
                                srsName="urn:ogc:def:crs:EPSG::4269">
                        <gml:pointProperty>
                                <gml:Point gml:id="p1"
><gml:pos
>42.258729 -71.16028</gml:pos
></gml:Point>
```

```
                        </gml:pointProperty>
                        <gml:pos
>42.259112 -71.160837</gml:pos>
                        <gml:pointProperty>
                                <gml:Point xlink:type="simple" xlink:href="#p1"/>
                        </gml:pointProperty>
                </gml:LineString
>')););
```

Beispiele - polyedische Oberfläche

```
SELECT ST_AsEWKT(ST_GeomFromGML('
<gml:PolyhedralSurface>
<gml:polygonPatches>
  <gml:PolygonPatch>
    <gml:exterior>
      <gml:LinearRing
><gml:posList srsDimension="3"
>0 0 0 0 0 1 0 1 1 0 1 0 0 0 0</gml:posList
></gml:LinearRing>
    </gml:exterior>
  </gml:PolygonPatch>
  <gml:PolygonPatch>
    <gml:exterior>
        <gml:LinearRing
><gml:posList srsDimension="3"
>0 0 0 0 1 0 1 1 0 1 0 0 0 0 0</gml:posList
></gml:LinearRing>
    </gml:exterior>
  </gml:PolygonPatch>
  <gml:PolygonPatch>
    <gml:exterior>
        <gml:LinearRing
><gml:posList srsDimension="3"
>0 0 0 1 0 0 1 0 1 0 0 1 0 0 0</gml:posList
></gml:LinearRing>
    </gml:exterior>
  </gml:PolygonPatch>
  <gml:PolygonPatch>
    <gml:exterior>
        <gml:LinearRing
><gml:posList srsDimension="3"
>1 1 0 1 1 1 1 0 1 1 0 0 1 1 0</gml:posList
></gml:LinearRing>
    </gml:exterior>
  </gml:PolygonPatch>
  <gml:PolygonPatch>
    <gml:exterior>
        <gml:LinearRing
><gml:posList srsDimension="3"
>0 1 0 0 1 1 1 1 1 1 1 0 0 1 0</gml:posList
></gml:LinearRing>
    </gml:exterior>
  </gml:PolygonPatch>
  <gml:PolygonPatch>
    <gml:exterior>
        <gml:LinearRing
><gml:posList srsDimension="3"
>0 0 1 1 0 1 1 1 1 0 1 1 0 0 1</gml:posList
></gml:LinearRing>
```

```
    </gml:exterior>
  </gml:PolygonPatch>
</gml:polygonPatches>
</gml:PolyhedralSurface
>'));

-- result --
 POLYHEDRALSURFACE(((0 0 0,0 0 1,0 1 1,0 1 0,0 0 0)),
 ((0 0 0,0 1 0,1 1 0,1 0 0,0 0 0)),
 ((0 0 0,1 0 0,1 0 1,0 0 1,0 0 0)),
 ((1 1 0,1 1 1,1 0 1,1 0 0,1 1 0)),
 ((0 1 0,0 1 1,1 1 1,1 1 0,0 1 0)),
 ((0 0 1,1 0 1,1 1 1,0 1 1,0 0 1)))
```

Siehe auch

Section 2.2.3, ST_AsGML, ST_GMLToSQL

7.8.3.4 ST_GeomFromGeoJSON

ST_GeomFromGeoJSON — Nimmt als Eingabe eine GeoJSON-Darstellung der Geometrie und gibt ein geometrisches PostGIS-Objekt aus.

Synopsis

geometry **ST_GeomFromGeoJSON**(text geomjson);
geometry **ST_GeomFromGeoJSON**(json geomjson);
geometry **ST_GeomFromGeoJSON**(jsonb geomjson);

Beschreibung

Erzeugt ein geometrisches PostGIS Objekt aus der GeoJSON Darstellung.

ST_GeomFromGeoJSON funktioniert nur bei Fragmenten von JSON-Geometrien. Auf das ganze JSON-Dokument angewendet führt zu einer Fehlermeldung.

Enhanced: 3.0.0 parsed geometry defaults to SRID=4326 if not specified otherwise.

Erweiterung: 2.5.0 unterstützt nun auch die Eingabe von json und jsonb.

Verfügbarkeit: 2.0.0 benötigt - JSON-C >= 0.9

Note
Wenn Sie die JSON-C Unterstützung nicht aktiviert haben, sehen Sie eine Fehlermeldung anstatt einer Ausgabe. Um JSON-C zu aktivieren, führen Sie bitte configure --with-jsondir=/path/to/json-c aus. Für Einzelheiten siehe Section 2.2.3.

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

Beispiele

```
SELECT ST_AsText(ST_GeomFromGeoJSON('{"type":"Point","coordinates":[-48.23456,20.12345]}')) ↩
     As wkt;
wkt
------
POINT(-48.23456 20.12345)
```

```
-- a 3D linestring
SELECT ST_AsText(ST_GeomFromGeoJSON('{"type":"LineString","coordinates ↩
    ":[[1,2,3],[4,5,6],[7,8,9]]}')) As wkt;

wkt
--------------------
LINESTRING(1 2,4 5,7 8)
```

Siehe auch

ST_AsText, ST_AsGeoJSON, Section 2.2.3

7.8.3.5 ST_GeomFromKML

ST_GeomFromKML — Nimmt als Eingabe eine KML-Darstellung der Geometrie und gibt ein geometrisches PostGIS-Objekt aus.

Synopsis

geometry **ST_GeomFromKML**(text geomkml);

Beschreibung

Erzeugt ein PostGIS ST_Geometry Objekt aus der OGC KML Darstellung.

T_GeomFromKML funktioniert nur bei Fragmenten von KML-Geometrien. Auf das ganze KML-Dokument angewendet führt zu einer Fehlermeldung.

Unterstützte OGC KML Versionen:

- KML 2.2.0 Namespace

OGC KML Standards, vgl.: http://www.opengeospatial.org/standards/kml:

Verfügbarkeit: 1.5, benötigt libxml2 2.6+

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

Note
SQL/MM Kurvengeometrien werden von der Funktion ST_GeomFromKML nicht unterstützt

---

Beispiele - Eine Einzelgeometrie mit einem srsName

```
SELECT ST_GeomFromKML('
                <LineString>
                        <coordinates
>-71.1663,42.2614
                                -71.1667,42.2616</coordinates>
                </LineString
>');
```

Siehe auch

Section 2.2.3, ST_AsKML

7.8.3.6 ST_GeomFromTWKB

ST_GeomFromTWKB — Erzeugt eine Geometrie aus einer TWKB ("Tiny Well-Known Binary") Darstellung.

Synopsis

geometry **ST_GeomFromTWKB**(bytea twkb);

Beschreibung

Die Funktion `ST_GeomFromTWKB` nimmt eine TWKB ("Tiny Well-Known Binary") Darstellung und erzeugt ein Objekt mit dem entsprechenden geometrischen Datentyp.

Beispiele

```
SELECT ST_AsText(ST_GeomFromTWKB(ST_AsTWKB('LINESTRING(126 34, 127 35)'::geometry)));

         st_astext
-----------------------------
 LINESTRING(126 34, 127 35)
(1 row)


SELECT ST_AsEWKT(
  ST_GeomFromTWKB(E'\\x620002f7f40dbce4040105')
);
                                          st_asewkt
------------------------------------------------------
LINESTRING(-113.98 39.198,-113.981 39.195)
(1 row)
```

Siehe auch

ST_AsTWKB

7.8.3.7 ST_GMLToSQL

ST_GMLToSQL — Gibt einen spezifizierten ST_Geometry Wert aus einer GML-Darstellung zurück. Dies ist ein Aliasname für ST_GeomFromGML

Synopsis

geometry **ST_GMLToSQL**(text geomgml);
geometry **ST_GMLToSQL**(text geomgml, integer srid);

Beschreibung

This method implements the SQL/MM specification. SQL-MM 3: 5.1.50 (ausgenommen Unterstützung von Kurven).

Verfügbarkeit: 1.5, benötigt libxml2 1.6+

Erweiterung: Mit 2.0.0 wurde die Unterstützung für polyedrische Oberflächen und TIN eingeführt.

Erweiterung: 2.0.0 Standardwert für den optionalen Parameter SRID eingefügt.

Siehe auch

Section 2.2.3, ST_GeomFromGML, ST_AsGML

7.8.3.8 ST_LineFromEncodedPolyline

ST_LineFromEncodedPolyline — Erzeugt einen LineString aus einem codierten Linienzug.

Synopsis

geometry **ST_LineFromEncodedPolyline**(text polyline, integer precision=5);

Beschreibung

Erzeugt einen LineString aus einem codierten Linienzug.

Der optionale Parameter `precision` gibt an wieviele Dezimalstellen der kodierten Polylinie erhalten bleiben. Dieser Wert sollte beim Dekodieren und beim Kodieren ident sein, sonst entstehen inkorrekte Koordinaten.

Siehe http://developers.google.com/maps/documentation/utilities/polylinealgorithm

Verfügbarkeit: 2.2.0

Beispiele

```
-- Create a line string from a polyline
SELECT ST_AsEWKT(ST_LineFromEncodedPolyline('_p~iF~ps|U_ulLnnqC_mqNvxq`@'));
-- result --
SRID=4326;LINESTRING(-120.2 38.5,-120.95 40.7,-126.453 43.252)

-- Select different precision that was used for polyline encoding
SELECT ST_AsEWKT(ST_LineFromEncodedPolyline('_p~iF~ps|U_ulLnnqC_mqNvxq`@',6));
-- result --
SRID=4326;LINESTRING(-12.02 3.85,-12.095 4.07,-12.6453 4.3252)
```

Siehe auch

ST_AsEncodedPolyline

7.8.3.9 ST_PointFromGeoHash

ST_PointFromGeoHash — Gibt einen Punkt von einer GeoHash Zeichenfolge zurück.

Synopsis

point **ST_PointFromGeoHash**(text geohash, integer precision=full_precision_of_geohash);

Beschreibung

Gibt die Geometrie einer GeoHash Zeichenfolge zurück. Der Punkt entspricht dem Mittelpunkt des GeoHas.

Wenn keine `precision` angegeben wird, dann gibt ST_PointFromGeoHash einen Punkt zurück, der auf der vollständigen Genauigkeit der gegebenen GeoHash Zeichenfolge beruht.

Wenn `precision` angegeben wird, verwendet ST_PointFromGeoHash entsprechend viele Zeichen des GeoHash, um den Punkt zu erzeugen.

Verfügbarkeit: 2.1.0

Beispiele

```
SELECT ST_AsText(ST_PointFromGeoHash('9qqj7nmxncgyy4d0dbxqz0'));
          st_astext
------------------------------
 POINT(-115.172816 36.114646)

SELECT ST_AsText(ST_PointFromGeoHash('9qqj7nmxncgyy4d0dbxqz0', 4));
             st_astext
-----------------------------------
 POINT(-115.13671875 36.123046875)

SELECT ST_AsText(ST_PointFromGeoHash('9qqj7nmxncgyy4d0dbxqz0', 10));
                 st_astext
-------------------------------------------
 POINT(-115.172815918922 36.1146435141563)
```

Siehe auch

ST_GeoHash, ST_Box2dFromGeoHash, ST_GeomFromGeoHash

7.8.3.10 ST_FromFlatGeobufToTable

ST_FromFlatGeobufToTable — Creates a table based on the structure of FlatGeobuf data.

Synopsis

void **ST_FromFlatGeobufToTable**(text schemaname, text tablename, bytea FlatGeobuf input data);

Beschreibung

Creates a table based on the structure of FlatGeobuf data. (http://flatgeobuf.org).

`schema` Schema name.

`table` Table name.

`data` Input FlatGeobuf data.

Availability: 3.2.0

7.8.3.11 ST_FromFlatGeobuf

ST_FromFlatGeobuf — Reads FlatGeobuf data.

Synopsis

setof anyelement **ST_FromFlatGeobuf**(anyelement Table reference, bytea FlatGeobuf input data);

Beschreibung

Reads FlatGeobuf data (http://flatgeobuf.org). NOTE: PostgreSQL bytea cannot exceed 1GB.

`tabletype` reference to a table type.

`data` input FlatGeobuf data.

Availability: 3.2.0

7.9 Geometrieausgabe

7.9.1 Well-known-Text (WKT) Repräsentation

7.9.1.1 ST_AsEWKT

ST_AsEWKT — Gibt die Well-known-Text(WKT) Darstellung der Geometrie mit den SRID-Metadaten zurück.

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);

Beschreibung

Returns the Well-Known Text representation of the geometry prefixed with the SRID. The optional *maxdecimaldigits* argument may be used to reduce the maximum number of decimal digits after floating point used in output (defaults to 15).

To perform the inverse conversion of EWKT representation to PostGIS geometry use ST_GeomFromEWKT.

Warning
Using the *maxdecimaldigits* parameter can cause output geometry to become invalid. To avoid this use ST_ReducePrecision with a suitable gridsize first.

Note
The WKT spec does not include the SRID. To get the OGC WKT format use ST_AsText.

Warning
WKT format does not maintain precision so to prevent floating truncation, use ST_AsBinary or ST_AsEWKB format for transport.

Enhanced: 3.1.0 support for optional precision parameter.

Erweiterung: Mit 2.0.0 wurde die Unterstützung für den geographischen Datentyp, polyedrische Oberflächen, Dreiecke und TIN eingeführt.

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

This method supports Circular Strings and Curves.

This function supports Polyhedral surfaces.

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

Beispiele

```
SELECT ST_AsEWKT('0103000020E6100000010000005000000000000000
                        0000000000000000000000000000000000000000000000000000000000
                        F03F000000000000F03F000000000000F03F000000000000F03
                        F000000000000000000000000000000000000000000000000000'::geometry);

                    st_asewkt
--------------------------------
SRID=4326;POLYGON((0 0,0 1,1 1,1 0,0 0))
(1 row)

SELECT ST_AsEWKT('0108000080030000000000000060 ↩
    E30A4100000000785C0241000000000000F03F0000000018
E20A4100000000485F0241000000000000004000000000018
E20A4100000000305C0241000000000000840')

--st_asewkt---
CIRCULARSTRING(220268 150415 1,220227 150505 2,220227 150406 3)
```

Siehe auch

ST_AsBinary, ST_AsEWKB, ST_AsText, ST_GeomFromEWKT

7.9.1.2 ST_AsText

ST_AsText — Gibt die Well-known-Text(WKT) Darstellung der Geometrie/Geographie ohne die SRID Metadaten zurück.

Synopsis

text **ST_AsText**(geometry g1);
text **ST_AsText**(geometry g1, integer maxdecimaldigits = 15);
text **ST_AsText**(geography g1);
text **ST_AsText**(geography g1, integer maxdecimaldigits = 15);

Beschreibung

Returns the OGC Well-Known Text (WKT) representation of the geometry/geography. The optional *maxdecimaldigits* argument may be used to limit the number of digits after the decimal point in output ordinates (defaults to 15).

To perform the inverse conversion of WKT representation to PostGIS geometry use ST_GeomFromText.

Note
The standard OGC WKT representation does not include the SRID. To include the SRID as part of the output representation, use the non-standard PostGIS function ST_AsEWKT

Warning
The textual representation of numbers in WKT may not maintain full floating-point precision. To ensure full accuracy for data storage or transport it is best to use Well-Known Binary (WKB) format (see ST_AsBinary and *maxdecimaldigits*).

Warning
Using the *maxdecimaldigits* parameter can cause output geometry to become invalid. To avoid this use ST_ReducePrecision with a suitable gridsize first.

Verfügbarkeit: 1.5 - Unterstützung von geograpischen Koordinaten.

Erweiterung: 2.5 - der optionale Parameter "precision" wurde eingeführt.

This method implements the OGC Simple Features Implementation Specification for SQL 1.1. s2.1.1.1

This method implements the SQL/MM specification. SQL-MM 3: 5.1.25

This method supports Circular Strings and Curves.

Beispiele

```
SELECT ST_AsText('0103000000010000000500000000000000000000
0000000000000000000000000000000000000000000000000000000000
F03F000000000000F03F000000000000F03F000000000000F03F
F000000000000000000000000000000000000000000000000000');

    st_astext
------------------------------
 POLYGON((0 0,0 1,1 1,1 0,0 0))
```

Full precision output is the default.

```
SELECT ST_AsText('POINT(111.1111111 1.1111111)'));
    st_astext
------------------------------
 POINT(111.1111111 1.1111111)
```

The *maxdecimaldigits* argument can be used to limit output precision.

```
SELECT ST_AsText('POINT(111.1111111 1.1111111)'), 2);
    st_astext
--------------------
 POINT(111.11 1.11)
```

Siehe auch

ST_AsBinary, ST_AsEWKB, ST_AsEWKT, ST_GeomFromText

7.9.2 Well-known-Binary (WKB) Repräsentation

7.9.2.1 ST_AsBinary

ST_AsBinary — Return the OGC/ISO Well-Known Binary (WKB) representation of the geometry/geography without SRID meta data.

Synopsis

bytea **ST_AsBinary**(geometry g1);
bytea **ST_AsBinary**(geometry g1, text NDR_or_XDR);
bytea **ST_AsBinary**(geography g1);
bytea **ST_AsBinary**(geography g1, text NDR_or_XDR);

Beschreibung

Returns the OGC/ISO Well-Known Binary (WKB) representation of the geometry. The first function variant defaults to encoding using server machine endian. The second function variant takes a text argument specifying the endian encoding, either little-endian ('NDR') or big-endian ('XDR').

WKB format is useful to read geometry data from the database and maintaining full numeric precision. This avoids the precision rounding that can happen with text formats such as WKT.

To perform the inverse conversion of WKB to PostGIS geometry use ST_GeomFromWKB.

Note
The OGC/ISO WKB format does not include the SRID. To get the EWKB format which does include the SRID use ST_AsEWKB

Note
The default behavior in PostgreSQL 9.0 has been changed to output bytea in hex encoding. If your GUI tools require the old behavior, then SET bytea_output='escape' in your database.

Erweiterung: 2.0.0 - Unterstützung für polyedrische Oberflächen, Dreiecke und TIN eingeführt.

Erweiterung: 2.0.0 - Unterstützung für höherdimensionale Koordinatensysteme eingeführt.

Erweiterung: 2.0.0 Unterstützung zum Festlegen des Endian beim geographischen Datentyp eingeführt.

Verfügbarkeit: 1.5.0 Unterstützung von geograpischen Koordinaten.

Änderung: 2.0.0 - Eingabewerte für diese Funktion dürfen nicht "unknown" sein -- es muss sich um eine Geometrie handeln. Konstrukte, wie `ST_AsBinary('POINT(1 2)')`, sind nicht länger gültig und geben folgende Fehlermeldung aus: `n st_asbinary(unknown) is not unique error`. Dieser Code muss in `ST_AsBinary('POINT(1 2)'::geometry)` geändert werden. Falls dies nicht möglich ist, so installieren Sie bitte `legacy.sql`.

This method implements the OGC Simple Features Implementation Specification for SQL 1.1. s2.1.1.1

This method implements the SQL/MM specification. SQL-MM 3: 5.1.37

This method supports Circular Strings and Curves.

This function supports Polyhedral surfaces.

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

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

Beispiele

```
SELECT ST_AsBinary(ST_GeomFromText('POLYGON((0 0,0 1,1 1,1 0,0 0))',4326));

                    st_asbinary
---------------------------------
\x0103000000010000000500000000000000000000000000000000000000000000000000000000000000000000000000000000
000000f03f000000000000f03f000000000000f03f000000000000f03f000000000000000000000000
00000000000000000000000000
```

```
SELECT ST_AsBinary(ST_GeomFromText('POLYGON((0 0,0 1,1 1,1 0,0 0))',4326), 'XDR');
                    st_asbinary
---------------------------------
\x0000000003000000010000000500000000000000000000000000000000000000000000000000000000000000000000000003ff000
00000000003ff0000000000000003ff0000000000000003ff0000000000000000000000000000000000000
00000000000000000000000000
```

Siehe auch

ST_GeomFromWKB, ST_AsEWKB, ST_AsTWKB, ST_AsText,

7.9.2.2 ST_AsEWKB

ST_AsEWKB — Return the Extended Well-Known Binary (EWKB) representation of the geometry with SRID meta data.

Synopsis

bytea **ST_AsEWKB**(geometry g1);
bytea **ST_AsEWKB**(geometry g1, text NDR_or_XDR);

Beschreibung

Returns the Extended Well-Known Binary (EWKB) representation of the geometry with SRID metadata. The first function variant defaults to encoding using server machine endian. The second function variant takes a text argument specifying the endian encoding, either little-endian ('NDR') or big-endian ('XDR').

WKB format is useful to read geometry data from the database and maintaining full numeric precision. This avoids the precision rounding that can happen with text formats such as WKT.

To perform the inverse conversion of EWKB to PostGIS geometry use ST_GeomFromEWKB.

Note
To get the OGC/ISO WKB format use ST_AsBinary. Note that OGC/ISO WKB format does not include the SRID.

Erweiterung: 2.0.0 - Unterstützung für polyedrische Oberflächen, Dreiecke und TIN eingeführt.

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

This method supports Circular Strings and Curves.

This function supports Polyhedral surfaces.

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

Beispiele

```
SELECT ST_AsEWKB(ST_GeomFromText('POLYGON((0 0,0 1,1 1,1 0,0 0))',4326));

                   st_asewkb
--------------------------------
\x0103000020e6100000010000000500000000000000000000000000000000000000000000000000000000000000000000
000000000000000f03f000000000000f03f000000000000f03f000000000000f03f000000000000000000
00000000000000000000000000000000
```

```
SELECT ST_AsEWKB(ST_GeomFromText('POLYGON((0 0,0 1,1 1,1 0,0 0))',4326), 'XDR');
                   st_asewkb
--------------------------------
\x0020000003000010e600000001000000050000000000000000000000000000000000000000000000000000000000000000
003ff0000000000003ff0000000000003ff0000000000003ff0000000000000000000000000000
00000000000000000000000000000000
```

Siehe auch

ST_AsBinary, ST_GeomFromEWKB, ST_SRID

7.9.2.3 ST_AsHEXEWKB

ST_AsHEXEWKB — Gibt eine Geometrie im HEXEWKB Format (als Text) aus; verwendet entweder die Little-Endian (NDR) oder die Big-Endian (XDR) Zeichenkodierung.

Synopsis

text **ST_AsHEXEWKB**(geometry g1, text NDRorXDR);
text **ST_AsHEXEWKB**(geometry g1);

Beschreibung

Gibt eine Geometrie im HEXEWKB Format (als Text) aus; verwendet entweder die Little-Endian (NDR) oder die Big-Endian (XDR) Zeichenkodierung. Wenn keine Zeichenkodierung angegeben wurde, wird NDR verwendet.

Note
Verfügbarkeit: 1.2.2

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

This method supports Circular Strings and Curves.

Beispiele

```
SELECT ST_AsHEXEWKB(ST_GeomFromText('POLYGON((0 0,0 1,1 1,1 0,0 0))',4326));
                which gives same answer as

                SELECT ST_GeomFromText('POLYGON((0 0,0 1,1 1,1 0,0 0))',4326)::text;

                st_ashexewkb
```

```
                --------
                0103000020E6100000010000000500
                00000000000000000000000000000000
                000000000000000000000000000000F03F
                000000000000F03F000000000000F03F000000000000F03
                F000000000000000000000000000000000000000000000000
```

7.9.3 Weitere Formate

7.9.3.1 ST_AsEncodedPolyline

ST_AsEncodedPolyline — Erzeugt eine codierte Polylinie aus einer LineString Geometrie.

Synopsis

text **ST_AsEncodedPolyline**(geometry geom, integer precision=5);

Beschreibung

Gibt die Geometrie als kodierte Polyline aus. Dieses Format wird von "Google Maps" mit precision=5 und von "Open Source Routing Machine" mit precision=5 oder 6 verwendet.

Der optionale Parameter `precision` gibt an wieviele Dezimalstellen der kodierten Polylinie erhalten bleiben. Dieser Wert sollte beim Dekodieren und beim Kodieren ident sein, sonst entstehen inkorrekte Koordinaten.

Verfügbarkeit: 2.2.0

Beispiele

Grundlegendes

```
SELECT ST_AsEncodedPolyline(GeomFromEWKT('SRID=4326;LINESTRING(-120.2 38.5,-120.95  ↩
    40.7,-126.453 43.252)'));
        --result--
        |_p~iF~ps|U_ulLnnqC_mqNvxq`@
```

Anwendung in Verbindung mit LINESTRING und ST_Segmentize für den geographischen Datentyp, und auf Google Maps stellen

```
-- the SQL for Boston to San Francisco, segments every 100 KM
        SELECT ST_AsEncodedPolyline(
                ST_Segmentize(
                        ST_GeogFromText('LINESTRING(-71.0519 42.4935,-122.4483 37.64)'),
                                100000)::geometry) As encodedFlightPath;
```

In JavaScript sieht dies ungefähr wie folgt aus, wobei die $ Variable durch das Abfrageergebnis ersetzt wird

```
<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>
```

Siehe auch

ST_LineFromEncodedPolyline, ST_Segmentize

7.9.3.2 ST_AsFlatGeobuf

ST_AsFlatGeobuf — Return a FlatGeobuf representation of a set of rows.

Synopsis

bytea **ST_AsFlatGeobuf**(anyelement set row);
bytea **ST_AsFlatGeobuf**(anyelement row, bool index);
bytea **ST_AsFlatGeobuf**(anyelement row, bool index, text geom_name);

Beschreibung

Return a FlatGeobuf representation (http://flatgeobuf.org) of a set of rows corresponding to a FeatureCollection. NOTE: PostgreSQL bytea cannot exceed 1GB.

`row` Datenzeilen mit zumindest einer Geometriespalte.

`index` toggle spatial index creation. Default is false.

`geom_name` ist die Bezeichnung der Geometriespalte in den Datenzeilen. Wenn NULL, dann wird standardmäßig die erste aufgefundene Geometriespalte verwendet.

Availability: 3.2.0

7.9.3.3 ST_AsGeobuf

ST_AsGeobuf — Gibt eine Menge an Zeilen in der Geobuf Darstellung aus.

Synopsis

bytea **ST_AsGeobuf**(anyelement set row);
bytea **ST_AsGeobuf**(anyelement row, text geom_name);

Beschreibung

Gibt Zeilen einer FeatureCollection in der Geobuf Darstellung (https://github.com/mapbox/geobuf) aus. Von jeder Eingabegeometrie wird die maximale Genauigkeit analysiert, um eine optimale Speicherung zu erreichen. Anmerkung: In der jetzigen Form kann Geobuf nicht "gestreamt" werden, wodurch die gesamte Ausgabe im Arbeitsspeicher zusammengestellt wird.

`row` Datenzeilen mit zumindest einer Geometriespalte.

`geom_name` ist die Bezeichnung der Geometriespalte in den Datenzeilen. Wenn NULL, dann wird standardmäßig die erste aufgefundene Geometriespalte verwendet.

Verfügbarkeit: 2.4.0

Beispiele

```
SELECT encode(ST_AsGeobuf(q, 'geom'), 'base64')
    FROM (SELECT ST_GeomFromText('POLYGON((0 0,0 1,1 1,1 0,0 0))') AS geom) AS q;
 st_asgeobuf
---------------------------------
 GAAiEAoOCgwIBBoIAAAAAgIAAAE=
```

7.9.3.4 ST_AsGeoJSON

ST_AsGeoJSON — Return a geometry as a GeoJSON element.

Synopsis

text **ST_AsGeoJSON**(record feature, text geomcolumnname, integer maxdecimaldigits=9, boolean pretty_bool=false);
text **ST_AsGeoJSON**(geometry geom, integer maxdecimaldigits=9, integer options=8);
text **ST_AsGeoJSON**(geography geog, integer maxdecimaldigits=9, integer options=0);

Beschreibung

Returns a geometry as a GeoJSON "geometry", or a row as a GeoJSON "feature". (See the GeoJSON specifications RFC 7946). 2D and 3D Geometries are both supported. GeoJSON only support SFS 1.1 geometry types (no curve support for example).

Der Parameter `maxdecimaldigits` kann zur Reduzierung der Nachkommastellen in der Ausgabe verwendet werden (standardmäßig 9). Wenn EPSG:4326 verwendet wird, kann `maxdecimaldigits=6` eine gute Wahl für viele Karten bei der Bildschirmausgabe sein.

Warning
Using the *`maxdecimaldigits`* parameter can cause output geometry to become invalid. To avoid this use ST_ReducePrecision with a suitable gridsize first.

The `options` argument can be used to add BBOX or CRS in GeoJSON output:

- 0: keine option
- 1: GeoJSON BBOX
- 2: GeoJSON CRS-Kurzform (z.B. EPSG:4326)
- 4: GeoJSON CRS-Langform (z.B. urn:ogc:def:crs:EPSG::4326)
- 8: GeoJSON CRS-Kurzform, außer bei EPSG:4326 (default)

The GeoJSON specification states that polygons are oriented using the Right-Hand Rule, and some clients require this orientation. This can be ensured by using ST_ForcePolygonCCW . The specification also requires that geometry be in the WGS84 coordinate system (SRID = 4326). If necessary geometry can be projected into WGS84 using ST_Transform: `ST_Transform( geom, 4326 )`.

GeoJSON can be tested and viewed online at geojson.io and geojsonlint.com. It is widely supported by web mapping frameworks:

- Beispiel für ein OpenLayers GeoJSON
- Beispiel für ein Leaflet GeoJSON
- Beispiel für ein Mapbox GL GeoJSON

Verfügbarkeit: 1.3.4

Verfügbarkeit: 1.5.0 Unterstützung von geograpischen Koordinaten.

Änderung: 2.0.0 Unterstützung für Standardargumente und benannte Argumente.

Änderung: 3.0.0 Unterstützung von Datensätzen bei der Eingabe

Änderung: 3.0.0 Ausgabe der SRID wenn nicht EPSG:4326

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

Beispiele

Generate a FeatureCollection:

```
SELECT json_build_object(
    'type', 'FeatureCollection',
    'features', json_agg(ST_AsGeoJSON(t.*)::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]}, "properties": {"id": 1, "name": "one"}}, {"type": "Feature", " ↩
    geometry": {"type":"Point","coordinates":[2,2]}, "properties": {"id": 2, "name": "two ↩
    "}}, {"type": "Feature", "geometry": {"type":"Point","coordinates":[3,3]}, "properties": ↩
     {"id": 3, "name": "three"}}]}
```

Generate a Feature:

```
SELECT ST_AsGeoJSON(t.*)
FROM (VALUES (1, 'one', 'POINT(1 1)'::geometry)) AS t(id, name, geom);
```

```
st_asgeojson
----------------------------------------------------------------------------------------------------

 {"type": "Feature", "geometry": {"type":"Point","coordinates":[1,1]}, "properties": {"id": ↩
      1, "name": "one"}}
```

An alternate way to generate Features with an `id` property is to use JSONB functions and operators:

```
SELECT jsonb_build_object(
    'type',       'Feature',
    'id',         id,
    'geometry',   ST_AsGeoJSON(geom)::jsonb,
    'properties', to_jsonb( t.* ) - 'id' - 'geom'
    ) AS json
FROM (VALUES (1, 'one', 'POINT(1 1)'::geometry)) AS t(id, name, geom);
```

```
json
----------------------------------------------------------------------------------------------------

 {"id": 1, "type": "Feature", "geometry": {"type": "Point", "coordinates": [1, 1]}, " ↩
     properties": {"name": "one"}}
```

Don't forget to transform your data to WGS84 longitude, latitude to conform with the GeoJSON specification:

```
SELECT ST_AsGeoJSON(ST_Transform(geom,4326)) from fe_edges limit 1;
```

```
st_asgeojson
----------------------------------------------------------------------------------------------------



{"type":"MultiLineString","coordinates":[[[-89.734634999999997,31.492072000000000],
[-89.734955999999997,31.492237999999997]]]}
```

3D geometries are supported:

```
SELECT ST_AsGeoJSON('LINESTRING(1 2 3, 4 5 6)');
```

```
{"type":"LineString","coordinates":[[1,2,3],[4,5,6]]}
```

Siehe auch

ST_GeomFromGeoJSON, ST_ForcePolygonCCW , ST_Transform

7.9.3.5 ST_AsGML

ST_AsGML — Gibt die Geometrie als GML-Element - Version 2 oder 3 - zurück.

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);

Beschreibung

Gibt die Geometrie als ein Geography Markup Language (GML) Element zurück. Ein Versionsparameter kann mit 2 oder 3 angegeben werden. Wenn kein Versionsparameter angegeben ist, wird dieser standardmäßig Version 2 angenommen. Der Parameter `maxdecimaldigits` kann verwendet werden, um die Anzahl der Nachkommastellen bei der Ausgabe zu reduzieren (standardmäßig 15).

Warning
Using the *`maxdecimaldigits`* parameter can cause output geometry to become invalid. To avoid this use ST_ReducePrecision with a suitable gridsize first.

GML 2 verweist auf Version 2.1.2, GML 3 auf Version 3.1.1

Der Übergabewert "options" ist ein Bitfeld. Es kann verwendet werden um das Koordinatenreferenzsystem bei der GML Ausgabe zu bestimmen und um die Daten in Länge/Breite anzugeben.

- 0: GML Kurzform für das CRS (z.B. EPSG:4326), Standardwert
- 1: GML Langform für das CRS (z.B. urn:ogc:def:crs:EPSG::4326)
- 2: Nur für GML 3, entfernt das srsDimension Attribut von der Ausgabe.
- 4: Nur für GML 3, Für Linien verwenden Sie bitte den Tag <LineString> anstatt <Curve>.
- 16: Deklarieren, dass die Daten in Breite/Länge (z.B. SRID=4326) vorliegen. Standardmäßig wird angenommen, dass die Daten planar sind. Diese Option ist nur bei Ausgabe in GML 3.1.1, in Bezug auf die Anordnung der Achsen sinnvoll. Falls Sie diese setzen, werden die Koordinaten von Länge/Breite auf Breite/Länge vertauscht.
- 32: Ausgabe der BBox der Geometrie (Umhüllende/Envelope).

Der Übergabewert 'namespace prefix' kann verwendet werden, um ein benutzerdefiniertes Präfix für den Namensraum anzugeben, oder kein Präfix (wenn leer). Wenn Null oder weggelassen, so wird das Präfix "gml" verwendet.

Verfügbarkeit: 1.3.2

Verfügbarkeit: 1.5.0 Unterstützung von geograpischen Koordinaten.

Erweiterung: 2.0.0 Unterstützung durch Präfix eingeführt. Für GML3 wurde die Option 4 eingeführt, um die Verwendung von LineString anstatt von Kurven für Linien zu erlauben. Ebenfalls wurde die GML3 Unterstützung für polyedrische Oberflächen und TINS eingeführt, sowie die Option 32 zur Ausgabe der BBox.

Änderung: 2.0.0 verwendet standardmäßig benannte Argumente.

Erweiterung: 2.1.0 Für GML 3 wurde die Unterstützung einer ID eingeführt.

Note
Nur die Version 3+ von ST_AsGML unterstützt polyedrische Oberflächen und TINs.

This method implements the SQL/MM specification. SQL-MM IEC 13249-3: 17.2

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

This function supports Polyhedral surfaces.

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

Beispiele: Version 2

```
SELECT ST_AsGML(ST_GeomFromText('POLYGON((0 0,0 1,1 1,1 0,0 0))',4326));
                st_asgml
                --------
                <gml:Polygon srsName="EPSG:4326"
><gml:outerBoundaryIs
><gml:LinearRing
><gml:coordinates
>0,0 0,1 1,1 1,0 0,0</gml:coordinates
></gml:LinearRing
></gml:outerBoundaryIs
></gml:Polygon
>
```

Beispiele: Version 3

```
-- Flip coordinates and output extended EPSG (16 | 1)--
SELECT ST_AsGML(3, ST_GeomFromText('POINT(5.234234233242 6.34534534534)',4326), 5, 17);
                        st_asgml
                        --------
                <gml:Point srsName="urn:ogc:def:crs:EPSG::4326"
><gml:pos
>6.34535 5.23423</gml:pos
></gml:Point
>
```

```
-- Output the envelope (32) --
SELECT ST_AsGML(3, ST_GeomFromText('LINESTRING(1 2, 3 4, 10 20)',4326), 5, 32);
                st_asgml
                --------
        <gml:Envelope srsName="EPSG:4326">
                <gml:lowerCorner
>1 2</gml:lowerCorner>
                <gml:upperCorner
>10 20</gml:upperCorner>
        </gml:Envelope
>
```

```
-- Output the envelope (32) , reverse (lat lon instead of lon lat) (16), long srs (1)= 32 | ↩
     16 | 1 = 49 --
SELECT ST_AsGML(3, ST_GeomFromText('LINESTRING(1 2, 3 4, 10 20)',4326), 5, 49);
        st_asgml
```

```
        --------
<gml:Envelope srsName="urn:ogc:def:crs:EPSG::4326">
        <gml:lowerCorner
>2 1</gml:lowerCorner>
        <gml:upperCorner
>20 10</gml:upperCorner>
</gml:Envelope
>
```

```
-- Polyhedral Example --
SELECT ST_AsGML(3, ST_GeomFromEWKT('POLYHEDRALSURFACE( ((0 0 0, 0 0 1, 0 1 1, 0 1 0, 0 0 0) ↩
    ),
((0 0 0, 0 1 0, 1 1 0, 1 0 0, 0 0 0)), ((0 0 0, 1 0 0, 1 0 1, 0 0 1, 0 0 0)),
((1 1 0, 1 1 1, 1 0 1, 1 0 0, 1 1 0)),
((0 1 0, 0 1 1, 1 1 1, 1 1 0, 0 1 0)), ((0 0 1, 1 0 1, 1 1 1, 0 1 1, 0 0 1)) )'));
        st_asgml
        --------
 <gml:PolyhedralSurface>
<gml:polygonPatches>
   <gml:PolygonPatch>
                <gml:exterior>
                          <gml:LinearRing>
                                    <gml:posList srsDimension="3"
>0 0 0 0 0 1 0 1 1 0 1 0 0 0 0</gml:posList>
                          </gml:LinearRing>
                </gml:exterior>
   </gml:PolygonPatch>
   <gml:PolygonPatch>
                <gml:exterior>
                          <gml:LinearRing>
                                    <gml:posList srsDimension="3"
>0 0 0 0 1 0 1 1 0 1 0 0 0 0 0</gml:posList>
                          </gml:LinearRing>
                </gml:exterior>
   </gml:PolygonPatch>
   <gml:PolygonPatch>
                <gml:exterior>
                          <gml:LinearRing>
                                    <gml:posList srsDimension="3"
>0 0 0 1 0 0 1 0 1 0 0 1 0 0 0</gml:posList>
                          </gml:LinearRing>
                </gml:exterior>
   </gml:PolygonPatch>
   <gml:PolygonPatch>
                <gml:exterior>
                          <gml:LinearRing>
                                    <gml:posList srsDimension="3"
>1 1 0 1 1 1 1 0 1 1 0 0 1 1 0</gml:posList>
                          </gml:LinearRing>
                </gml:exterior>
   </gml:PolygonPatch>
   <gml:PolygonPatch>
                <gml:exterior>
                          <gml:LinearRing>
                                    <gml:posList srsDimension="3"
>0 1 0 0 1 1 1 1 1 1 1 0 0 1 0</gml:posList>
                          </gml:LinearRing>
                </gml:exterior>
   </gml:PolygonPatch>
   <gml:PolygonPatch>
                <gml:exterior>
                          <gml:LinearRing>
```

```
                                    <gml:posList srsDimension="3"
>0 0 1 1 0 1 1 1 1 0 1 1 0 0 1</gml:posList>
                                </gml:LinearRing>
                        </gml:exterior>
    </gml:PolygonPatch>
</gml:polygonPatches>
</gml:PolyhedralSurface
>
```

Siehe auch

ST_GeomFromGML

7.9.3.6 ST_AsKML

ST_AsKML — Gibt die Geometrie als GML-Element - Version 2 oder 3 - zurück.

Synopsis

text **ST_AsKML**(geometry geom, integer maxdecimaldigits=15, text nprefix=NULL);
text **ST_AsKML**(geography geog, integer maxdecimaldigits=15, text nprefix=NULL);

Beschreibung

Gibt die Geometrie als ein Keyhole Markup Language (KML) Element zurück. Diese Funktion verfügt über mehrere Varianten. Die maximale Anzahl der Dezimalstellen die bei der Ausgabe verwendet wird (standardmäßig 15), die Version ist standardmäßig 2 und der Standardnamensraum hat kein Präfix.

Warning
Using the *maxdecimaldigits* parameter can cause output geometry to become invalid. To avoid this use ST_ReducePrecision with a suitable gridsize first.

Note
Setzt voraus, dass PostGIS mit Proj-Unterstützung kompiliert wurde. Verwenden Sie bitte PostGIS_Full_Version, um festzustellen ob mit proj kompiliert wurde.

Note
Verfügbarkeit: 1.2.2 - spätere Varianten ab 1.3.2 nehmen den Versionsparameter mit auf

---

Note
Erweiterung: 2.0.0 - Präfix Namensraum hinzugefügt. Standardmäßig kein Präfix

---

Note
Changed: 3.0.0 - Removed the "versioned" variant signature

Note
Die Ausgabe AsKML funktioniert nicht bei Geometrien ohne SRID

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

Beispiele

```
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>
```

Siehe auch

ST_AsSVG, ST_AsGML

7.9.3.7 ST_AsLatLonText

ST_AsLatLonText — Gibt die "Grad, Minuten, Sekunden"-Darstellung für den angegebenen Punkt aus.

Synopsis

text **ST_AsLatLonText**(geometry pt, text format='');

Beschreibung

Gibt die "Grad, Minuten, Sekunden"-Darstellung des Punktes aus.

Note
Es wird angenommen, dass der Punkt in einer Breite/Länge-Projektion vorliegt. Die X (Länge) und Y (Breite) Koordinaten werden bei der Ausgabe in den "üblichen" Bereich (-180 to +180 für die Länge, -90 to +90 für die Breite) normalisiert.

Der Textparameter ist eine Zeichenkette für die Formatierung der Ausgabe, ähnlich wie die Zeichenkette für die Formatierung der Datumsausgabe. Gültige Zeichen sind "D" für Grad/Degrees, "M" für Minuten, "S" für Sekunden, und "C" für die Himmelsrichtung (NSEW). DMS Zeichen können wiederholt werden, um die gewünschte Zeichenbreite und Genauigkeit anzugeben ("SSS.SSSS" bedeutet z.B. " 1.0023").

"M", "S", und "C" sind optional. Wenn "C" weggelassen wird, werden Grad mit einem "-" Zeichen versehen, wenn Süd oder West. Wenn "S" weggelassen wird, werden die Minuten als Dezimalzahl mit der vorgegebenen Anzahl an Kommastellen angezeigt. Wenn "M" wegggelassen wird, werden die Grad als Dezimalzahl mit der vorgegebenen Anzahl an Kommastellen angezeigt.

Wenn die Zeichenkette für das Ausgabeformat weggelassen wird (oder leer ist) wird ein Standardformat verwendet.

Verfügbarkeit: 2.0

Beispiele

Standardformat.

```
SELECT (ST_AsLatLonText('POINT (-3.2342342 -2.32498)'));
      st_aslatlontext
---------------------------
 2\textdegree{}19'29.928"S 3\textdegree{}14'3.243"W
```

Ein Format angeben (identisch mit Standardformat).

```
SELECT (ST_AsLatLonText('POINT (-3.2342342 -2.32498)', 'D\textdegree{}M''S.SSS"C'));
      st_aslatlontext
---------------------------
 2\textdegree{}19'29.928"S 3\textdegree{}14'3.243"W
```

Andere Zeichen als D, M, S, C und "." werden lediglich durchgereicht.

```
SELECT (ST_AsLatLonText('POINT (-3.2342342 -2.32498)', 'D degrees, M minutes, S seconds to  ↩
    the C'));
                                  st_aslatlontext
-------------------------------------------------------------------------------------
 2 degrees, 19 minutes, 30 seconds to the S 3 degrees, 14 minutes, 3 seconds to the W
```

Grad mit einem Vorzeichen versehen - anstatt der Himmelsrichtung.

```
SELECT (ST_AsLatLonText('POINT (-3.2342342 -2.32498)', 'D\textdegree{}M''S.SSS"'));
      st_aslatlontext
---------------------------
 -2\textdegree{}19'29.928" -3\textdegree{}14'3.243"
```

Dezimalgrad.

```
SELECT (ST_AsLatLonText('POINT (-3.2342342 -2.32498)', 'D.DDDD degrees C'));
          st_aslatlontext
-----------------------------------
 2.3250 degrees S 3.2342 degrees W
```

Überhöhte Werte werden normalisiert.

```
SELECT (ST_AsLatLonText('POINT (-302.2342342 -792.32498)'));
        st_aslatlontext
-----------------------------
 72\textdegree{}19'29.928"S 57\textdegree{}45'56.757"E
```

7.9.3.8 ST_AsMARC21

ST_AsMARC21 — Returns geometry as a MARC21/XML record with a geographic datafield (034).

Synopsis

text **ST_AsMARC21** (geometry geom , text format='hdddmmss');

Beschreibung

This function returns a MARC21/XML record with Coded Cartographic Mathematical Data representing the bounding box of a given geometry. The `format` parameter allows to encode the coordinates in subfields `$d`,`$e`,`$f` and `$g` in all formats supported by the MARC21/XML standard. Valid formats are:

- cardinal direction, degrees, minutes and seconds (default): `hdddmmss`
- decimal degrees with cardinal direction: `hddd.dddddd`
- decimal degrees without cardinal direction: `ddd.dddddd`
- decimal minutes with cardinal direction: `hdddmm.mmmm`
- decimal minutes without cardinal direction: `dddmm.mmmm`
- decimal seconds with cardinal direction: `hdddmmss.sss`

The decimal sign may be also a comma, e.g. `hdddmm,mmmm`.

The precision of decimal formats can be limited by the number of characters after the decimal sign, e.g. `hdddmm.mm` for decimal minutes with a precision of two decimals.

This function ignores the Z and M dimensions.

LOC MARC21/XML versions supported:

- MARC21/XML 1.1

Availability: 3.3.0

Note
This function does not support non lon/lat geometries, as they are not supported by the MARC21/XML standard (Coded Cartographic Mathematical Data).

Note
The MARC21/XML Standard does not provide any means to annotate the spatial reference system for Coded Cartographic Mathematical Data, which means that this information will be lost after conversion to MARC21/XML.

Beispiele

Converting a `POINT` to MARC21/XML formated as hdddmmss (default)

```
                SELECT ST_AsMARC21('SRID=4326;POINT(-4.504289 54.253312)'::geometry);

                                st_asmarc21
                -------------------------------------------------
                <record xmlns="http://www.loc.gov/MARC21/slim">
                    <datafield tag="034" ind1="1" ind2=" ">
                        <subfield code="a"
>a</subfield>
```

```
                        <subfield code="d"
>W0043015</subfield>
                        <subfield code="e"
>W0043015</subfield>
                        <subfield code="f"
>N0541512</subfield>
                        <subfield code="g"
>N0541512</subfield>
                    </datafield>
                </record>
```

Converting a `POLYGON` to MARC21/XML formated in decimal degrees

```
                SELECT ST_AsMARC21('SRID=4326;POLYGON((-4.5792388916015625  ↩
                    54.18172660239091,-4.56756591796875  ↩
                    54.196993557130355,-4.546623229980469  ↩
                    54.18313300502024,-4.5792388916015625 54.18172660239091))'::geometry,' ↩
                    hddd.dddd');

                <record xmlns="http://www.loc.gov/MARC21/slim">
                    <datafield tag="034" ind1="1" ind2=" ">
                        <subfield code="a"
>a</subfield>
                        <subfield code="d"
>W004.5792</subfield>
                        <subfield code="e"
>W004.5466</subfield>
                        <subfield code="f"
>N054.1970</subfield>
                        <subfield code="g"
>N054.1817</subfield>
                    </datafield>
                </record>
```

Converting a `GEOMETRYCOLLECTION` to MARC21/XML formated in decimal minutes. The geometries order in the MARC21/XML output correspond to their order in the collection.

```
                SELECT ST_AsMARC21('SRID=4326;GEOMETRYCOLLECTION(POLYGON((13.1  ↩
                    52.65,13.516666666666667 52.65,13.516666666666667 52.38333333333333,13.1 ↩
                     52.38333333333333,13.1 52.65)),POINT(-4.5 54.25))'::geometry,'hdddmm. ↩
                    mmmm');

                               st_asmarc21
                -------------------------------------------------
                <record xmlns="http://www.loc.gov/MARC21/slim">
                    <datafield tag="034" ind1="1" ind2=" ">
                        <subfield code="a"
>a</subfield>
                        <subfield code="d"
>E01307.0000</subfield>
                        <subfield code="e"
>E01331.0000</subfield>
                        <subfield code="f"
>N05240.0000</subfield>
                        <subfield code="g"
>N05224.0000</subfield>
```

```
                    </datafield>
                    <datafield tag="034" ind1="1" ind2=" ">
                        <subfield code="a"
>a</subfield>
                        <subfield code="d"
>W00430.0000</subfield>
                        <subfield code="e"
>W00430.0000</subfield>
                        <subfield code="f"
>N05415.0000</subfield>
                        <subfield code="g"
>N05415.0000</subfield>
                    </datafield>
                </record>
```

Siehe auch

ST_GeomFromMARC21

7.9.3.9 ST_AsMVTGeom

ST_AsMVTGeom — Transforms a geometry into the coordinate space of a MVT tile.

Synopsis

geometry **ST_AsMVTGeom**(geometry geom, box2d bounds, integer extent=4096, integer buffer=256, boolean clip_geom=true);

Beschreibung

Transforms a geometry into the coordinate space of a MVT (Mapbox Vector Tile) tile, clipping it to the tile bounds if required. The geometry must be in the coordinate system of the target map (using ST_Transform if needed). Commonly this is Web Mercator (SRID:3857).

The function attempts to preserve geometry validity, and corrects it if needed. This may cause the result geometry to collapse to a lower dimension.

The rectangular bounds of the tile in the target map coordinate space must be provided, so the geometry can be transformed, and clipped if required. The bounds can be generated using ST_TileEnvelope.

This function is used to convert geometry into the tile coordinate space required by ST_AsMVT.

`geom` is the geometry to transform, in the coordinate system of the target map.

`bounds` is the rectangular bounds of the tile in map coordinate space, with no buffer.

`extent` is the tile extent size in tile coordinate space as defined by the MVT specification. Defaults to 4096.

`buffer` is the buffer size in tile coordinate space for geometry clippig. Defaults to 256.

`clip_geom` is a boolean to control if geometries are clipped or encoded as-is. Defaults to true.

Verfügbarkeit: 2.4.0

Note
Ab 3.0 kann zum Ausschneiden und Validieren von MVT-Polygonen bei der Konfiguration Wagyu gewählt werden. Diese Bibliothek ist schneller und liefert genauere Ergebnisse als die standardmäßige GEOS-Bibliothek, sie kann aber kleine Polygone verwerfen.

Beispiele

```
SELECT ST_AsText(ST_AsMVTGeom(
        ST_GeomFromText('POLYGON ((0 0, 10 0, 10 5, 0 -5, 0 0))'),
        ST_MakeBox2D(ST_Point(0, 0), ST_Point(4096, 4096)),
        4096, 0, false));
                              st_astext
-----------------------------------------------------------------------
 MULTIPOLYGON(((5 4096,10 4091,10 4096,5 4096)),((5 4096,0 4101,0 4096,5 4096)))
```

Canonical example for a Web Mercator tile using a computed tile bounds to query and clip geometry.

```
SELECT ST_AsMVTGeom(
            ST_Transform( geom, 3857 ),
            ST_TileEnvelope(12, 513, 412), extent =
> 4096, buffer =
> 64) AS geom
  FROM data
  WHERE geom && ST_TileEnvelope(12, 513, 412, margin =
> (64.0 / 4096))
```

Siehe auch

ST_AsMVT, ST_TileEnvelope, PostGIS_Wagyu_Version

7.9.3.10 ST_AsMVT

ST_AsMVT — Aggregate function returning a MVT representation of a set of rows.

Synopsis

bytea **ST_AsMVT**(anyelement set row);
bytea **ST_AsMVT**(anyelement row, text name);
bytea **ST_AsMVT**(anyelement row, text name, integer extent);
bytea **ST_AsMVT**(anyelement row, text name, integer extent, text geom_name);
bytea **ST_AsMVT**(anyelement row, text name, integer extent, text geom_name, text feature_id_name);

Beschreibung

An aggregate function which returns a binary Mapbox Vector Tile representation of a set of rows corresponding to a tile layer. The rows must contain a geometry column which will be encoded as a feature geometry. The geometry must be in tile coordinate space and valid as per the MVT specification. ST_AsMVTGeom can be used to transform geometry into tile coordinate space. Other row columns are encoded as feature attributes.

Das Mapbox Vector Tile Format kann Geoobjekte mit unterschiedlichen Attributen speichern. Um diese Möglichkeit zu nutzen, muss eine JSONB-Spalte in den Datensätzen mitgeliefert werden, welche in einer tieferen Ebene die JSON-Objekte enthält. Die Schlüssel und Werte im JSONB werden als Featureattribute kodiert.

Durch das Aneinanderhängen mehrerer Funktionsaufrufe mittels `||`, können Tiles mit mehreren Ebenen erstellt werden.

Important
Darf nicht mit einer `GEOMETRYCOLLECTION` im Datensatz aufgerufen werden. Es kann aber ST_AsMVTGeom verwendet werden, um eine Sammelgeometrie einzubinden.

`row` Datenzeilen mit zumindest einer Geometriespalte.

`name` ist die Bezeichnung der Ebene. Standardmäßig die Zeichenkette "default".

`extent` ist die Ausdehnung der Tiles in Bildschirmeinheiten. Standardmäßig 4096.

`geom_name` is the name of the geometry column in the row data. Default is the first geometry column. Note that PostgreSQL by default automatically folds unquoted identifiers to lower case, which means that unless the geometry column is quoted, e.g. `"MyMVTGeom"`, this parameter must be provided as lowercase.

`feature_id_name` ist die Bezeichnung der Feature-ID Spalte im Datensatz. Ist der Übergabewert NULL oder negativ, dann wird die Feature-ID nicht gesetzt. Die erste Spaltee mit einem passenden Namen und einem gültigen Datentyp (Smallint, Integer, Bigint) wird als Feature-ID verwendet, alle nachfolgenden Spalten werden als Eigenschaften hinzugefügt. JSON-Properties werden nicht unterstützt.

Erweiterung: 3.0 - Unterstützung für eine Feature-ID.

Erweiterung: 2.5.0 - Unterstützung von nebenläufigen Abfragen.

Verfügbarkeit: 2.4.0

Beispiele

```
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;
```

Siehe auch

ST_AsMVTGeom, ST_TileEnvelope

7.9.3.11 ST_AsSVG

ST_AsSVG — Gibt eine Geometrie als SVG-Pfad aus.

Synopsis

text **ST_AsSVG**(geometry geom, integer rel=0, integer maxdecimaldigits=15);
text **ST_AsSVG**(geography geog, integer rel=0, integer maxdecimaldigits=15);

Beschreibung

Gibt die Geometrie als Skalare Vektor Graphik (SVG-Pfadgeometrie) aus. Verwenden Sie 1 als zweiten Übergabewert um die Pfadgeometrie in relativen Schritten zu implementieren; Standardmäßig (oder 0) verwendet absolute Schritte. Der dritte Übergabewert kann verwendet werden, um die maximale Anzahl der Dezimalstellen bei der Ausgabe einzuschränken (standardmäßig 15). Punktgeometrie wird als cx/cy übersetzt wenn der Übergabewert 'rel' gleich 0 ist, x/y wenn 'rel' 1 ist. Mehrfachgeometrie wird durch Beistriche (",") getrennt, Sammelgeometrie wird durch Strichpunkt (";") getrennt.

For working with PostGIS SVG graphics, checkout pg_svg library which provides plpgsql functions for working with outputs from ST_AsSVG.

Enhanced: 3.4.0 to support all curve types

Änderung: 2.0.0 verwendet Standardargumente und unterstützt benannte Argumente.

Note
Verfügbarkeit: 1.2.2. Änderung: 1.4.0 L-Befehl beim absoluten Pfad aufgenommen, um mit http://www.w3.org/TR/SVG/paths.html#PathDataBNF konform zu sein.

This method supports Circular Strings and Curves.

Beispiele

```
SELECT ST_AsSVG('POLYGON((0 0,0 1,1 1,1 0,0 0))'::geometry);

st_assvg
--------
M 0 0 L 0 -1 1 -1 1 0 Z
```

Circular string

```
SELECT ST_AsSVG( ST_GeomFromText('CIRCULARSTRING(-2 0,0 2,2 0,0 2,2 4)') );

st_assvg
--------
M -2 0 A 2 2 0 0 1 2 0 A 2 2 0 0 1 2 -4
```

Multi-curve

```
SELECT ST_AsSVG('MULTICURVE((5 5,3 5,3 3,0 3),
 CIRCULARSTRING(0 0,2 1,2 2))'::geometry, 0, 0);
 st_assvg
-----------------------------------------------
 M 5 -5 L 3 -5 3 -3 0 -3 M 0 0 A 2 2 0 0 0 2 -2
```

Multi-surface

```
SELECT ST_AsSVG('MULTISURFACE(
CURVEPOLYGON(CIRCULARSTRING(-2 0,-1 -1,0 0,1 -1,2 0,0 2,-2 0),
    (-1 0,0 0.5,1 0,0 1,-1 0)),
((7 8,10 10,6 14,4 11,7 8)))'::geometry, 0, 2);

st_assvg
-------------------------------------------------------
M -2 0 A 1 1 0 0 0 0 0 A 1 1 0 0 0 2 0 A 2 2 0 0 0 -2 0 Z
M -1 0 L 0 -0.5 1 0 0 -1 -1 0 Z
M 7 -8 L 10 -10 6 -14 4 -11 Z
```

7.9.3.12 ST_AsTWKB

ST_AsTWKB — Gibt die Geometrie als TWKB, aka "Tiny Well-known Binary" zurück

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);

Beschreibung

Gibt die Geometrie im TWKB ("Tiny Well-Known Binary") Format aus. TWKB ist ein komprimiertes binäres Format mit dem Schwerpunkt, die Ausgabegröße zu minimieren.

Der Parameter 'decimaldigits' bestimmt die Anzahl der Dezimalstellen bei der Ausgabe. Standardmäßig werden die Werte vor der Zeichenkodierung auf die Einserstelle gerundet. Wenn Sie die Daten mit höherer Genauigkeit übergeben wollen, erhöhen Sie bitte die Anzahl der Dezimalstellen. Zum Beispiel bedeutet ein Wert von 1, dass die erste Dezimalstelle erhalten bleibt.

Die Parameter "sizes" und "bounding_boxes" bestimmen ob zusätzliche Information über die kodierte Länge und die Abgrenzung des Objektes in der Ausgabe eingebunden werden. Standardmäßig passiert dies nicht. Drehen Sie diese bitte nicht auf, solange dies nicht von Ihrer Client-Software benötigt wird, da dies nur unnötig Speicherplatz verbraucht (Einsparen von Speicherplatz ist der Sinn von TWKB).

Das Feld-Eingabeformat dieser Funktion wird verwendet um eine Sammelgeometriee und eindeutige Identifikatoren in eine TWKB-Collection zu konvertieren, welche die Identifikatoren erhält. Dies ist nützlich für Clients, die davon ausgehen, eine Sammelgeometrie auszupacken, um so auf zusätzliche Information über die internen Objekte zuzugreifen. Sie können das Feld mit der Funktion array_agg erstellen. Die anderen Parameter bewirken dasselbe wie bei dem einfachen Format dieser Funktion.

Note
Die Formatspezifikation steht Online unter https://github.com/TWKB/Specification zur Verfügung, und Code zum Aufbau eines JavaScript Clints findet sich unter https://github.com/TWKB/twkb.js.

Erweiterung: 2.4.0 Hauptspeicher- und Geschwindigkeitsverbesserungen.

Verfügbarkeit: 2.2.0

Beispiele

```
SELECT ST_AsTWKB('LINESTRING(1 1,5 5)'::geometry);
                 st_astwkb
-------------------------------------------
\x02000202020808
```

Um ein aggregiertes TWKB-Objekt inklusive Identifikatoren zu erzeugen, fassen Sie bitte die gewünschte Geometrie und Objekte zuerst mittels "array_agg()" zusammen und rufen anschließend die passende TWKB Funktion auf.

```
SELECT ST_AsTWKB(array_agg(geom), array_agg(gid)) FROM mytable;
                 st_astwkb
-------------------------------------------
\x040402020400000202
```

Siehe auch

ST_GeomFromTWKB, ST_AsBinary, ST_AsEWKB, ST_AsEWKT, ST_GeomFromText

7.9.3.13 ST_AsX3D

ST_AsX3D — Gibt eine Geometrie im X3D XML Knotenelement-Format zurück: ISO-IEC-19776-1.2-X3DEncodings-XML

Synopsis

text **ST_AsX3D**(geometry g1, integer maxdecimaldigits=15, integer options=0);

Beschreibung

Gibt eine Geometrie als X3D knotenformatiertes XML Element zurück http://www.web3d.org/standards/number/19776-1. Falls `maxdecimaldigits` (Genauigkeit) nicht angegeben ist, wird sie standardmäßig 15.

Note
Es gibt verschiedene Möglichkeiten eine PostGIS Geometrie in X3D zu übersetzen, da sich der X3D Geometrietyp nicht direkt in den geometrischen Datentyp von PostGIS abbilden lässt. Einige neuere X3D Datentypen, die sich besser abbilden lassen könnten haben wir vermieden, da diese von den meisten Rendering-Tools zurzeit nicht untestützt werden. Dies sind die Abbildungen für die wir uns entschieden haben. Falls Sie Ideen haben, wie wir es den Anwendern ermöglichen können ihre bevorzugten Abbildungen anzugeben, können Sie gerne ein Bug-Ticket senden.
Im Folgenden wird beschrieben, wie der PostGIS 2D/3D Datentyp derzeit in den X3D Datentyp abgebildet wird

Das Argument 'options' ist ein Bitfeld. Ab PostGIS 2.2+ wird dieses verwendet, um anzuzeigen ob die Koordinaten als X3D geospatiale Knoten in GeoKoordinaten dargestellt werden und auch ob X- und Y-Achse vertauscht werden sollen. Standardmäßig erfolgt die Ausgabe durch `ST_AsX3D` im Datenbankformat (Länge, Breite oder X,Y), aber es kann auch der X3D Standard mit Breite/Länge oder Y/X bevorzugt werden.

- 0: X/Y in der Datenbankreihenfolge (z.B. ist Länge/Breite = X,Y die standardmäßige Datenbankreihenfolge), Standardwert, und nicht-spatiale Koordinaten (nur der normale alte Koordinaten-Tag).
- 1: X und Y umdrehen. In Verbindung mit der Option für GeoKoordinaten wird bei der Standardausgabe die Breite zuerst/"latitude_first" ausgegeben und die Koordinaten umgedreht.
- 2: Die Koordinaten werden als geospatiale GeoKoordinaten ausgegeben. Diese Option gibt eine Fehlermeldung aus, falls die Geometrie nicht in WGS 84 Länge/Breite (SRID: 4326) vorliegt. Dies ist zurzeit der einzige GeoKoordinaten-Typ der unterstützt wird.Siehe die X3D Spezifikation für Koordinatenreferenzsysteme. Die Standardausgabe ist `GeoCoordinate geoSystem='"GD" "WE" "longitude_first"'`. Wenn Sie den X3D Standard bevorzugen `GeoCoordinate geoSyste "WE" "latitude_first"'` verwenden Sie bitte `(2+1) = 3`

PostGIS Datentyp	2D X3D Datentyp	3D X3D Datentyp
LINESTRING	zurzeit nicht implementiert - wird PolyLine2D	LineSet
MULTILINESTRING	zurzeit nicht implementiert - wird PolyLine2D	IndexedLineSet
MULTIPOINT	Polypoint2D	PointSet
POINT	gibt leerzeichengetrennte Koordinaten aus	gibt leerzeichengetrennte Koordinaten aus
(MULTI) POLYGON, POLYHEDRALSURFACE	Ungültiges X3D Markup	IndexedFaceSet (die inneren Ringe werden zurzeit als ein weiteres FaceSet abgebildet)
TIN	TriangleSet2D (zurzeit nicht implementiert)	IndexedTriangleSet

Note
Die Unterstützung von 2D-Geometrie ist noch nicht vollständig. Die inneren Ringe werden zur Zeit lediglich als gesonderte Polygone abgebildet. Wir arbeiten daran.

Lots of advancements happening in 3D space particularly with X3D Integration with HTML5

Es gibt auch einen feinen OpenSource X3D Viewer, den Sie benützen können, um Geometrien darzustellen. Free Wrl http://freewrl.sourc
Binärdateien sind für Mac, Linux und Windows verfügbar. Sie können den mitgelieferten FreeWRL_Launcher verwenden, um Gemetrien darzustellen.

Also check out PostGIS minimalist X3D viewer that utilizes this function and x3dDom html/js open source toolkit.

Verfügbarkeit: 2.0.0: ISO-IEC-19776-1.2-X3DEncodings-XML

Erweiterung: 2.2.0: Unterstützung für geographische Koordinaten und Vertauschen der Achsen (x/y, Länge/Breite). Für nähere Details siehe Optionen.

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

This function supports Polyhedral surfaces.

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

Beispiel: Erzeugung eines voll funktionsfähigen X3D Dokuments - Dieses erzeugt einen Würfel, den man sich mit FreeWrl und anderen X3D-Viewern ansehen kann.

```
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 0 1 0 0  ↩
                1 0 1 0 0 1 1 1 0 1 1 1 1 0 1 1 0 0 0 1 0 0 1 1 1 1 1 1 1 0 0 0 1 1 0 1 1 1  ↩
                1 0 1 1' />
      </IndexedFaceSet>
      </Shape>
    </Transform>
  </Scene>
</X3D
>
```

PostGIS buildings

Copy and paste the output of this query to x3d scene viewer and click Show

```
SELECT string_agg('<Shape
>' || ST_AsX3D(ST_Extrude(geom, 0,0, i*0.5)) ||
    '<Appearance>
          <Material diffuseColor="' || (0.01*i)::text || ' 0.8 0.2" specularColor="' ||  ↩
              (0.05*i)::text || ' 0 0.5"/>
        </Appearance>
    </Shape
>', '')
FROM ST_Subdivide(ST_Letters('PostGIS'),20) WITH ORDINALITY AS f(geom,i);
```

Buildings formed by subdividing PostGIS and extrusion

Beispiel: Ein Achteck, um 3 Einheiten gehoben und mit einer dezimalen Genauigkeit von 6

```
SELECT ST_AsX3D(
ST_Translate(
    ST_Force_3d(
        ST_Buffer(ST_Point(10,10),5, 'quad_segs=2')), 0,0,
    3)
  ,6) As x3dfrag;

x3dfrag
--------
<IndexedFaceSet coordIndex="0 1 2 3 4 5 6 7">
    <Coordinate point="15 10 3 13.535534 6.464466 3 10 5 3 6.464466 6.464466 3 5 10 3  ↩
        6.464466 13.535534 3 10 15 3 13.535534 13.535534 3 " />
</IndexedFaceSet
>
```

Beispiel: 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
>
```

Beispiel: Geschlossener MultiLinestring (die Begrenzung eines Polygons mit Lücken)

```
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 — Gibt die Geometrie in der GeoHash Darstellung aus.

Synopsis

text **ST_GeoHash**(geometry geom, integer maxchars=full_precision_of_point);

Beschreibung

Computes a GeoHash representation of a geometry. A GeoHash encodes a geographic Point into a text form that is sortable and searchable based on prefixing. A shorter GeoHash is a less precise representation of a point. It can be thought of as a box that contains the point.

Non-point geometry values with non-zero extent can also be mapped to GeoHash codes. The precision of the code depends on the geographic extent of the geometry.

If `maxchars` is not specified, the returned GeoHash code is for the smallest cell containing the input geometry. Points return a GeoHash with 20 characters of precision (about enough to hold the full double precision of the input). Other geometric types may return a GeoHash with less precision, depending on the extent of the geometry. Larger geometries are represented with less precision, smaller ones with more precision. The box determined by the GeoHash code always contains the input feature.

If `maxchars` is specified the returned GeoHash code has at most that many characters. It maps to a (possibly) lower precision representation of the input geometry. For non-points, the starting point of the calculation is the center of the bounding box of the geometry.

Verfügbarkeit: 1.4.0

Note
ST_GeoHash requires input geometry to be in geographic (lon/lat) coordinates.

This method supports Circular Strings and Curves.

Beispiele

```
SELECT ST_GeoHash( ST_Point(-126,48) );

         st_geohash
----------------------
 c0w3hf1s70w3hf1s70w3

SELECT ST_GeoHash( ST_Point(-126,48), 5);

 st_geohash
------------
 c0w3h

-- This line contains the point, so the GeoHash is a prefix of the point code
SELECT ST_GeoHash('LINESTRING(-126 48, -126.1 48.1)'::geometry);

 st_geohash
------------
 c0w3
```

Siehe auch

ST_GeomFromGeoHash, ST_PointFromGeoHash, ST_Box2dFromGeoHash

7.10 Operatoren

7.10.1 Bounding Box Operators

7.10.1.1 &&

&& — Gibt `TRUE` zurück, wenn die 2D Bounding Box von A die 2D Bounding Box von B schneidet.

Synopsis

boolean **&&**(geometry A , geometry B);
boolean **&&**(geography A , geography B);

Beschreibung

Der `&&` Operator gibt `TRUE` zurück, wenn die 2D Bounding Box von Geometrie A die 2D Bounding Box der Geometrie von B schneidet.

Note
Dieser Operand benützt jeden Index, der für die Geometrie zur Verfügung stellt.

Erweiterung: Mit 2.0.0 wurde die Unterstützung für polyedrische Oberflächen eingeführt.

Verfügbarkeit: Mit 1.5.0 wurde die Unterstützung von geograpischen Koordinaten eingeführt

This method supports Circular Strings and Curves.

This function supports Polyhedral surfaces.

Beispiele

```
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)
```

Siehe auch

ST_Intersects, &>, &<|, &<, ~, @

7.10.1.2 &&(geometry,box2df)

&&(geometry,box2df) — Gibt TRUE zurück, wenn sich die 2D Bounding Box (cached) einer Geometrie mit einer 2D Bounding Box mit Gleitpunktgenauigkeit (BOX2DF) überschneidet.

Synopsis

boolean **&&**(geometry A , box2df B);

Beschreibung

Der && Operator gibt TRUE zurück, wenn die im Cache befindliche 2D Bounding Box der Geometrie A sich mit der 2D Bounding Box von B, unter Verwendung von Gleitpunktgenauigkeit überschneidet. D.h.: falls B eine (double precision) box2d ist, wird diese intern in eine auf Gleitpunkt genaue 2D Bounding Box (BOX2DF) umgewandelt.

Note
Dieser Operand ist eher für die interne Nutzung durch BRIN Indizes, als durch die Anwender, gedacht.

Verfügbarkeit: Mit 2.3.0 wurde die Unterstützung von Block Range INdexes (BRIN) eingeführt. Erfordert PostgreSQL 9.5+.

This method supports Circular Strings and Curves.

This function supports Polyhedral surfaces.

Beispiele

```
SELECT ST_Point(1,1) && ST_MakeBox2D(ST_Point(0,0), ST_Point(2,2)) AS overlaps;

 overlaps
----------
 t
(1 row)
```

Siehe auch

&&(box2df,geometry), &&(box2df,box2df), ~(geometry,box2df), ~(box2df,geometry), ~(box2df,box2df), @(geometry,box2df), @(box2df,geometry), @(box2df,box2df)

7.10.1.3 &&(box2df,geometry)

&&(box2df,geometry) — Gibt `TRUE` zurück, wenn eine 2D float precision bounding box (BOX2DF) eine Geometrie (cached) 2D bounding box schneidet.

Synopsis

boolean **&&**(box2df A , geometry B);

Beschreibung

Der `&&` Operator gibt `TRUE` zurück, wenn die 2D Bounding Box A die zwischengespeicherte 2D Bounding Box der Geometrie B, unter Benutzung von Fließpunktgenauigkeit, schneidet. D.h.: wenn A eine (double precision) box2d ist, wird diese intern in eine float precision 2D bounding box (BOX2DF) umgewandelt.

Note
Dieser Operand ist eher für die interne Nutzung durch BRIN Indizes, als durch die Anwender, gedacht.

Verfügbarkeit: Mit 2.3.0 wurde die Unterstützung von Block Range INdexes (BRIN) eingeführt. Erfordert PostgreSQL 9.5+.

This method supports Circular Strings and Curves.

This function supports Polyhedral surfaces.

Beispiele

```
SELECT ST_MakeBox2D(ST_Point(0,0), ST_Point(2,2)) && ST_Point(1,1) AS overlaps;

 overlaps
----------
 t
(1 row)
```

Siehe auch

&&(geometry,box2df), &&(box2df,box2df), ~(geometry,box2df), ~(box2df,geometry), ~(box2df,box2df), @(geometry,box2df), @(box2df,geometry), @(box2df,box2df)

7.10.1.4 &&(box2df,box2df)

&&(box2df,box2df) — Gibt `TRUE` zurück, wenn sich zwei 2D float precision Bounding Boxes (BOX2DF) überschneiden.

Synopsis

boolean **&&**(box2df A , box2df B);

Beschreibung

Der `&&` Operator gibt `TRUE` zurück, wenn sich zwei 2D Bounding Boxes A und B, unter Benutzung von float precision, gegenseitig überschneiden. D.h.: Wenn A (oder B) eine (double precision) box2d ist, wird diese intern in eine float precision 2D bounding box (BOX2DF) umgewandelt

Note
Dieser Operator ist für die interne Nutzung durch BRIN Indizes, und nicht so sehr durch Anwender, vorgesehen.

Verfügbarkeit: Mit 2.3.0 wurde die Unterstützung von Block Range INdexes (BRIN) eingeführt. Erfordert PostgreSQL 9.5+.

This method supports Circular Strings and Curves.

This function supports Polyhedral surfaces.

Beispiele

```
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)
```

Siehe auch

&&(geometry,box2df), &&(box2df,geometry), ~(geometry,box2df), ~(box2df,geometry), ~(box2df,box2df), @(geometry,box2df), @(box2df,geometry), @(box2df,box2df)

7.10.1.5 &&&

&&& — Gibt `TRUE` zurück, wenn A's n-D bounding box B's n-D bounding box schneidet.

Synopsis

boolean **&&&**(geometry A , geometry B);

Beschreibung

Der `&&&` Operator gibt `TRUE` zurück, wenn die n-D bounding box der Geometrie A die n-D bounding box der Geometrie B schneidet.

Note
Dieser Operand benützt jeden Index, der für die Geometrie zur Verfügung stellt.

Verfügbarkeit: 2.0.0

This method supports Circular Strings and Curves.

This function supports Polyhedral surfaces.

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

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

Beispiele: 3D LineStrings

```
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
```

Beispele: 3M LineStrings

```
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
```

Siehe auch

&&

7.10.1.6 &&&(geometry,gidx)

&&&(geometry,gidx) — Gibt `TRUE` zurück, wenn die (cached) n-D bounding box einer Geometrie eine n-D float precision bounding box (GIDX) schneidet.

Synopsis

boolean **&&&**(geometry A , gidx B);

Beschreibung

Der `&&&` Operator gibt `TRUE` zurück, wenn die zwischengespeicherte n-D bounding box der Geometrie A die n-D bounding box B, unter Benutzung von float precision, schneidet. D.h.: Wenn B eine (double precision) box3d ist, wird diese intern in eine float precision 3D bounding box (GIDX) umgewandelt

Note
Dieser Operator ist für die interne Nutzung durch BRIN Indizes, und nicht so sehr durch Anwender, vorgesehen.

Verfügbarkeit: Mit 2.3.0 wurde die Unterstützung von Block Range INdexes (BRIN) eingeführt. Erfordert PostgreSQL 9.5+.

This method supports Circular Strings and Curves.

This function supports Polyhedral surfaces.

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

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

Beispiele

```
SELECT ST_MakePoint(1,1,1) &&& ST_3DMakeBox(ST_MakePoint(0,0,0), ST_MakePoint(2,2,2)) AS  ↩
    overlaps;

 overlaps
----------
 t
(1 row)
```

Siehe auch

&&&(gidx,geometry), &&&(gidx,gidx)

7.10.1.7 &&&(gidx,geometry)

&&&(gidx,geometry) — Gibt `TRUE` zurück, wenn eine n-D float precision bounding box (GIDX) eine (cached) n-D bounding box einer Geometrie schneidet.

Synopsis

boolean **&&&**(gidx A , geometry B);

Beschreibung

Der `&&&` Operator gibt `TRUE` zurück, wenn die n-D bounding box A die cached n-D bounding box der Geometrie B, unter Benutzung von float precision, schneidet. D.h.: wenn A eine (double precision) box3d ist, wir diese intern in eine float precision 3D bounding box (GIDX) umgewandelt

Note
Dieser Operator ist für die interne Nutzung durch BRIN Indizes, und nicht so sehr durch Anwender, vorgesehen.

Verfügbarkeit: Mit 2.3.0 wurde die Unterstützung von Block Range INdexes (BRIN) eingeführt. Erfordert PostgreSQL 9.5+.

This method supports Circular Strings and Curves.

This function supports Polyhedral surfaces.

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

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

Beispiele

```
SELECT ST_3DMakeBox(ST_MakePoint(0,0,0), ST_MakePoint(2,2,2)) &&& ST_MakePoint(1,1,1) AS  ↩
    overlaps;

 overlaps
----------
 t
(1 row)
```

Siehe auch

&&&(geometry,gidx), &&&(gidx,gidx)

7.10.1.8 &&&(gidx,gidx)

&&&(gidx,gidx) — Gibt `TRUE` zurück, wenn sich zwei n-D float precision bounding boxes (GIDX) gegenseitig überschneiden.

Synopsis

boolean **&&&**(gidx A , gidx B);

Beschreibung

Der `&&&` Operator gibt `TRUE` zurück, wenn sich zwei n-D bounding boxes A und B, unter Benutzung von float precision, gegenseitig überschneiden. D.h.: wenn A (oder B) eine (double precision) box3d ist, wird diese intern in eine float precision 3D bounding box (GIDX) umgewandelt

---

Note
Dieser Operator ist für die interne Nutzung durch BRIN Indizes, und nicht so sehr durch Anwender, vorgesehen.

Verfügbarkeit: Mit 2.3.0 wurde die Unterstützung von Block Range INdexes (BRIN) eingeführt. Erfordert PostgreSQL 9.5+.

This method supports Circular Strings and Curves.

This function supports Polyhedral surfaces.

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

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

Beispiele

```
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)
```

Siehe auch

&&&(geometry,gidx), &&&(gidx,geometry)

7.10.1.9 &<

&< — Gibt TRUE zurück, wenn die bounding box der Geometrie A, die bounding box der Geometrie B überlagert oder links davon liegt.

Synopsis

boolean **&<**(geometry A , geometry B);

Beschreibung

Der &< Operator gibt TRUE zurück, wenn die bounding box der Geometrie A die bounding box der Geometrie B überlagert oder links davon liegt, oder präziser, überlagert und NICHT rechts von der bounding box der Geometrie B liegt.

Note
Dieser Operand benützt jeden Index, der für die Geometrie zur Verfügung stellt.

Beispiele

```
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)
```

Siehe auch

&&, |&>, &>, &<|

7.10.1.10 &<|

&<| — Gibt TRUE zurück, wenn die bounding box von A jene von B überlagert oder unterhalb liegt.

Synopsis

boolean **&<|**(geometry A , geometry B);

Beschreibung

Der &<| Operator gibt TRUE zurück, wenn die Bounding Box der Geometrie A die Bounding Box der Geometrie B überlagert oder unterhalb liegt, oder präziser, überlagert oder NICHT oberhalb der Bounding der Geometrie B liegt.

This method supports Circular Strings and Curves.

This function supports Polyhedral surfaces.

Note
Dieser Operand benützt jeden Index, der für die Geometrie zur Verfügung stellt.

Beispiele

```
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)
```

Siehe auch

&&, |&>, &>, &<

7.10.1.11 &>

&> — Gibt TRUE zurück, wenn die Bounding Box von A jene von B überlagert oder rechts davon liegt.

Synopsis

boolean **&>**(geometry A , geometry B);

Beschreibung

Der &> Operator gibt TRUE zurück, wenn die Bounding Box der Geometrie A die Bounding Box der Geometrie B überlagert oder rechts von ihr liegt, oder präziser, überlagert und NICHT links von der Bounding Box der Geometrie B liegt.

Note
Dieser Operand benützt jeden Index, der für die Geometrie zur Verfügung stellt.

Beispiele

```
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)
```

Siehe auch

&&, |&>, &<|, &<

7.10.1.12 <<

<< — Gibt TRUE zurück, wenn die Bounding Box von A zur Gänze links von der von B liegt.

Synopsis

boolean **<<**(geometry A , geometry B);

Beschreibung

Der << Operator gibt TRUE zurück, wenn die Bounding Box der Geometrie A zur Gänze links der Bounding Box der Geometrie B liegt.

Note
Dieser Operand benützt jeden Index, der für die Geometrie zur Verfügung stellt.

Beispiele

```
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)
```

Siehe auch

>>, |>>, <<|

7.10.1.13 <<|

<<| — Gibt TRUE zurück, wenn A's Bounding Box zur Gänze unterhalb von der von B liegt.

Synopsis

boolean **<<|**(geometry A , geometry B);

Beschreibung

Der <<| Operator gibt TRUE zurück, wenn die Bounding Box der Geometrie A zur Gänze unterhalb der Bounding Box von Geometrie B liegt.

Note
Dieser Operand benützt jeden Index, der für die Geometrie zur Verfügung stellt.

Beispiele

```
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)
```

Siehe auch

<<, >>, |>>

7.10.1.14 =

= — Gibt TRUE zurück, wenn die Koordinaten und die Reihenfolge der Koordinaten der Geometrie/Geographie A und der Geometrie/Geographie B ident sind.

Synopsis

boolean =(geometry A , geometry B);
boolean =(geography A , geography B);

Beschreibung

Der Operator = gibt TRUE zurück, wenn die Koordinaten und die Reihenfolge der Koordinaten der Geometrie/Geographie A und der Geometrie/Geographie B ident sind. PostgreSQL verwendet die =, <, und > Operatoren um die interne Sortierung und den Vergleich von Geometrien durchzuführen (z.B.: in einer GROUP BY oder ORDER BY Klausel).

Note
Nur die Geometrie/Geographie die in allen Gesichtspunkten übereinstimmt, d.h. mit den selben Koordinaten in der gleichen Reihenfolge, werden von diesem Operator als gleich betrachtet. Für "räumliche Gleichheit", bei der Dinge wie die Reihenfolge der Koordinaten außer Acht gelassen werden, und die es ermöglicht Geoobjekte zu erfassen, die denselben räumlichen Bereich mit unterschiedlicher Darstellung abdecken, verwenden Sie bitte ST_OrderingEquals oder ST_Equals

Caution
Dieser Operator verwendet NICHT die Indizes, welche für die Geometrien vorhanden sind. Um eine Überprüfung auf exakte Gleichheit indexgestützt durchzuführen, kombinieren Sie bitte = mit &&.

Änderung: 2.4.0, in Vorgängerversionen war dies die Gleichheit der umschreibenden Rechtecke, nicht die geometrische Gleichheit. Falls Sie auf Gleichheit der umschreibenden Rechtecke prüfen wollen, verwenden Sie stattdesse bitte ~=.

This method supports Circular Strings and Curves.

This function supports Polyhedral surfaces.

Beispiele

```
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
```

Siehe auch

ST_Equals, ST_OrderingEquals, ~=

7.10.1.15 >>

>> — Gibt TRUE zurück, wenn A's bounding box zur Gänze rechts von der von B liegt.

Synopsis

boolean **>>**(geometry A , geometry B);

Beschreibung

Der >> Operator gibt TRUE zurück, wenn die Bounding Box von Geometrie A zur Gänze rechts der Bounding Box von Geometrie B liegt.

Note
Dieser Operand benützt jeden Index, der für die Geometrie zur Verfügung stellt.

Beispiele

```
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)
```

Siehe auch

<<, |>>, <<|

7.10.1.16 @

@ — Gibt TRUE zurück, wenn die Bounding Box von A in jener von B enthalten ist.

Synopsis

boolean @(geometry A , geometry B);

Beschreibung

Der @ Operator gibt TRUE zurück, wenn die Bounding Box der Geometrie A vollstänig in der Bounding Box der Geometrie B enthalten ist.

Note
Dieser Operand benützt jeden Index, der für die Geometrie zur Verfügung stellt.

Beispiele

```
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)
```

Siehe auch

~, &&

7.10.1.17 @(geometry,box2df)

@(geometry,box2df) — Gibt TRUE zurück, wenn die 2D Bounding Box einer Geometrie in einer 2D float precision Bbounding Box (BOX2DF) enthalten ist.

Synopsis

boolean @(geometry A , box2df B);

Beschreibung

Der @ Operator gibt TRUE zurück, wenn die 2D Bounding Box der Geometrie A in der 2D Bounding Box der Geometrie B , unter Benutzung von float precision, enthalten ist. D.h.: wenn B eine (double precision) box2d ist, wird diese intern in eine float precision 2D bounding box (BOX2DF) übersetzt.

Note
Dieser Operand ist eher für die interne Nutzung durch BRIN Indizes, als durch die Anwender, gedacht.

Verfügbarkeit: Mit 2.3.0 wurde die Unterstützung von Block Range INdexes (BRIN) eingeführt. Erfordert PostgreSQL 9.5+.

This method supports Circular Strings and Curves.

This function supports Polyhedral surfaces.

Beispiele

```
SELECT ST_Buffer(ST_GeomFromText('POINT(2 2)'), 1) @ ST_MakeBox2D(ST_Point(0,0), ST_Point ↩
    (5,5)) AS is_contained;

 is_contained
--------------
 t
(1 row)
```

Siehe auch

&&(geometry,box2df), &&(box2df,geometry), &&(box2df,box2df), ~(geometry,box2df), ~(box2df,geometry), ~(box2df,box2df), @(box2df,geometry), @(box2df,box2df)

7.10.1.18 @(box2df,geometry)

@(box2df,geometry) — Gibt TRUE zurück, wenn eine 2D float precision bounding box (BOX2DF) in der 2D Bounding Box einer Geometrie enthalten ist..

Synopsis

boolean @(box2df A , geometry B);

Beschreibung

Der @ Operator gibt TRUE zurück, wenn die 2D bounding box A in der 2D bounding box der Geometrie B, unter Verwendung von float precision, enthalten ist. D.h.: wenn B eine (double precision) box2d ist, wird diese intern in eine float precision 2D bounding box (BOX2DF) umgewandelt

---

Note
Dieser Operand ist eher für die interne Nutzung durch BRIN Indizes, als durch die Anwender, gedacht.

Verfügbarkeit: Mit 2.3.0 wurde die Unterstützung von Block Range INdexes (BRIN) eingeführt. Erfordert PostgreSQL 9.5+.

This method supports Circular Strings and Curves.

This function supports Polyhedral surfaces.

Beispiele

```
SELECT ST_MakeBox2D(ST_Point(2,2), ST_Point(3,3)) @ ST_Buffer(ST_GeomFromText('POINT(1 1)') ↩
    , 10) AS is_contained;

 is_contained
--------------
 t
(1 row)
```

Siehe auch

&&(geometry,box2df), &&(box2df,geometry), &&(box2df,box2df), ~(geometry,box2df), ~(box2df,geometry), ~(box2df,box2df), @(geometry,box2df), @(box2df,box2df)

7.10.1.19 @(box2df,box2df)

@(box2df,box2df) — Gibt TRUE zurück, wenn eine 2D float precision bounding box (BOX2DF) innerhalb einer anderen 2D float precision bounding box enthalten ist.

Synopsis

boolean @(box2df A , box2df B);

Beschreibung

Der @ Operator gibt TRUE zurück, wenn die 2D bounding box A innerhalb der 2D bounding box B, unter Verwendung von float precision, enthalten ist. D.h.: wenn A (oder B) eine (double precision) box2d ist, wird diese intern in eine float precision 2D bounding box (BOX2DF) umgewandelt.

Note
Dieser Operand ist eher für die interne Nutzung durch BRIN Indizes, als durch die Anwender, gedacht.

Verfügbarkeit: Mit 2.3.0 wurde die Unterstützung von Block Range INdexes (BRIN) eingeführt. Erfordert PostgreSQL 9.5+.

This method supports Circular Strings and Curves.

This function supports Polyhedral surfaces.

Beispiele

```
SELECT ST_MakeBox2D(ST_Point(2,2), ST_Point(3,3)) @ ST_MakeBox2D(ST_Point(0,0), ST_Point ↩
    (5,5)) AS is_contained;

 is_contained
--------------
 t
(1 row)
```

Siehe auch

&&(geometry,box2df), &&(box2df,geometry), &&(box2df,box2df), ~(geometry,box2df), ~(box2df,geometry), ~(box2df,box2df), @(geometry,box2df), @(box2df,geometry)

7.10.1.20 |&>

|&> — Gibt TRUE zurück, wenn A's bounding box diejenige von B überlagert oder oberhalb von B liegt.

Synopsis

boolean **|&>**(geometry A , geometry B);

Beschreibung

Der |&> Operator gibt TRUE zurück, wenn die bounding box der Geometrie A die bounding box der Geometrie B überlagert oder oberhalb liegt, oder präziser, überlagert oder NICHT unterhalb der Bounding Box der Geometrie B liegt.

Note
Dieser Operand benützt jeden Index, der für die Geometrie zur Verfügung stellt.

Beispiele

```
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)
```

Siehe auch

&&, &>, &<|, &<

7.10.1.21 |>>

|>> — Gibt TRUE zurück, wenn A's bounding box is zur Gänze oberhalb der von B liegt.

Synopsis

boolean **|>>**(geometry A , geometry B);

Beschreibung

Der Operator |>> gibt TRUE zurück, wenn die Bounding Box der Geometrie A zur Gänze oberhalb der Bounding Box von Geometrie B liegt.

Note
Dieser Operand benützt jeden Index, der für die Geometrie zur Verfügung stellt.

Beispiele

```
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)
```

Siehe auch

<<, >>, <<|

7.10.1.22 ~

~ — Gibt TRUE zurück, wenn A's bounding box die von B enthält.

Synopsis

boolean ~(geometry A , geometry B);

Beschreibung

Der ~ Operator gibt TRUE zurück, wenn die bounding box der Geometrie A zur Gänze die bounding box der Geometrie B enthält.

Note
Dieser Operand benützt jeden Index, der für die Geometrie zur Verfügung stellt.

Beispiele

```
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)
```

Siehe auch

@, &&

7.10.1.23 ~(geometry,box2df)

~(geometry,box2df) — Gibt TRUE zurück, wenn die 2D bounding box einer Geometrie eine 2D float precision bounding box (GIDX) enthält.

Synopsis

boolean ~(geometry A , box2df B);

Beschreibung

Der ~ Operator gibt TRUE zurück, wenn die 2D bounding box einer Geometrie A die 2D bounding box B, unter Verwendung von float precision, enthält. D.h.: wenn B eine (double precision) box2d ist, wird diese intern in eine float precision 2D bounding box (BOX2DF) übersetzt

Note
Dieser Operand ist eher für die interne Nutzung durch BRIN Indizes, als durch die Anwender, gedacht.

Verfügbarkeit: Mit 2.3.0 wurde die Unterstützung von Block Range INdexes (BRIN) eingeführt. Erfordert PostgreSQL 9.5+.

This method supports Circular Strings and Curves.

This function supports Polyhedral surfaces.

Beispiele

```
SELECT ST_Buffer(ST_GeomFromText('POINT(1 1)'), 10) ~ ST_MakeBox2D(ST_Point(0,0), ST_Point ↩
    (2,2)) AS contains;

 contains
----------
 t
(1 row)
```

Siehe auch

&&(geometry,box2df), &&(box2df,geometry), &&(box2df,box2df), ~(box2df,geometry), ~(box2df,box2df), @(geometry,box2df), @(box2df,geometry), @(box2df,box2df)

7.10.1.24 ~(box2df,geometry)

~(box2df,geometry) — Gibt TRUE zurück, wenn eine 2D float precision bounding box (BOX2DF) die 2D Bounding Box einer Geometrie enthält.

Synopsis

boolean ~(box2df A , geometry B);

Beschreibung

Der ~ Operator gibt TRUE zurück, wenn die 2D bounding box A die Bounding Box der Geometrie B, unter Verwendung von float precision, enthält. D.h.: wenn A eine (double precision) box2d ist, wird diese intern in eine float precision 2D bounding box (BOX2DF) umgewandelt.

Note
Dieser Operand ist eher für die interne Nutzung durch BRIN Indizes, als durch die Anwender, gedacht.

Verfügbarkeit: Mit 2.3.0 wurde die Unterstützung von Block Range INdexes (BRIN) eingeführt. Erfordert PostgreSQL 9.5+.

This method supports Circular Strings and Curves.

This function supports Polyhedral surfaces.

Beispiele

```
SELECT ST_MakeBox2D(ST_Point(0,0), ST_Point(5,5)) ~ ST_Buffer(ST_GeomFromText('POINT(2 2)') ↩
    , 1) AS contains;

 contains
----------
 t
(1 row)
```

Siehe auch

&&(geometry,box2df), &&(box2df,geometry), &&(box2df,box2df), ~(geometry,box2df), ~(box2df,box2df), @(geometry,box2df), @(box2df,geometry), @(box2df,box2df)

7.10.1.25 ~(box2df,box2df)

~(box2df,box2df) — Gibt TRUE zurück, wenn eine 2D float precision bounding box (BOX2DF) eine andere 2D float precision bounding box (BOX2DF) enthält.

Synopsis

boolean ~(box2df A , box2df B);

Beschreibung

Der ~ Operator gibt TRUE zurück, wenn die 2D bounding box A die 2D bounding box B, unter Verwendung von float precision, enthält. D.h.: wenn A eine (double precision) box2d ist, wird diese intern in eine float precision 2D bounding box (BOX2DF) umgewandelt

Note
Dieser Operand ist eher für die interne Nutzung durch BRIN Indizes, als durch die Anwender, gedacht.

Verfügbarkeit: Mit 2.3.0 wurde die Unterstützung von Block Range INdexes (BRIN) eingeführt. Erfordert PostgreSQL 9.5+.

This method supports Circular Strings and Curves.

This function supports Polyhedral surfaces.

Beispiele

```
SELECT ST_MakeBox2D(ST_Point(0,0), ST_Point(5,5)) ~ ST_MakeBox2D(ST_Point(2,2), ST_Point ↩
    (3,3)) AS contains;

 contains
----------
 t
(1 row)
```

Siehe auch

&&(geometry,box2df), &&(box2df,geometry), &&(box2df,box2df), ~(geometry,box2df), ~(box2df,geometry), @(geometry,box2df), @(box2df,geometry), @(box2df,box2df)

7.10.1.26 ~=

~= — Gibt TRUE zurück, wenn die bounding box von A ident mit jener von B ist.

Synopsis

boolean ~=(geometry A , geometry B);

Beschreibung

Der ~= Operator gibt `TRUE` zurück, wenn die bounding box der Geometrie/Geographie A ident mit der bounding box der Geometrie/Geographie B ist.

Note
Dieser Operand benützt jeden Index, der für die Geometrie zur Verfügung stellt.

Verfügbarkeit: 1.5.0 "Verhaltensänderung"

This function supports Polyhedral surfaces.

Warning
Dieser Operator verhält sich ab PostGIS 1.5 insofern anders, als er vom Prüfen der Übereinstimmung der tatsächlichen Geometrie auf eine ledigliche Überprüfung der Gleichheit der Bounding Boxes abgeändert wurde. Um die Sache noch weiter zu komplizieren, hängt dieses Verhalten der Datenbank davon ab, ob ein hard oder soft upgrade durchgeführt wurde. Um herauszufinden, wie sich die Datenbank in dieser Beziehung verhält, führen Sie bitte die untere Abfrage aus. Um auf exakte Gleichheit zu prüfen benutzen Sie bitte ST_OrderingEquals oder ST_Equals.

Beispiele

```
select 'LINESTRING(0 0, 1 1)'::geometry ~= 'LINESTRING(0 1, 1 0)'::geometry as equality;
 equality   |
------------------+
          t     |
```

Siehe auch

ST_Equals, ST_OrderingEquals, =

7.10.2 Operatoren

7.10.2.1 <->

<-> — Gibt die 2D Entfernung zwischen A und B zurück.

Synopsis

double precision **<->**(geometry A , geometry B);
double precision **<->**(geography A , geography B);

Beschreibung

Der <-> Operator gibt die 2D Entfernung zwischen zwei Geometrien zurück. Wird er in einer "ORDER BY" Klausel verwendet, so liefert er Index-unterstützte nearest-neighbor Ergebnismengen. PostgreSQL Versionen unter 9.5 geben jedoch lediglich die Entfernung der Centroide der bounding boxes zurück, während PostgreSQL 9.5+ mittels KNN-Methode die tatsächliche Entfernung zwischen den Geometrien, bei geographischen Koordinaten die Entfernung auf der Späre, widergibt.

Note
Dieser Operand verwendet 2D GiST Indizes, falls diese für die Geometrien vorhanden sind. Er unterscheidet sich insofern von anderen Operatoren, die räumliche Indizes verwenden, indem der räumliche Index nur dann verwendet wird, wenn sich der Operator in einer ORDER BY Klausel befindet.

Note
Der Index kommt nur zum Tragen, wenn eine der Geometrien eine Konstante ist (sich nicht in einer Subquery/CTE befindet). Z.B. 'SRID=3005;POINT(1011102 450541)'::geometry und nicht a.geom

Siehe OpenGeo workshop: Nearest-Neighbour Searching für ein praxisbezogenes Anwendungsbeispiel.

Verbesserung: 2.2.0 -- Echtes KNN ("K nearest neighbor") Verhalten für Geometrie und Geographie ab PostgreSQL 9.5+. Beachten Sie bitte, das KNN für Geographie auf der Späre und nicht auf dem Sphäroid beruht. Für PostgreSQL 9.4 und darunter, wird die Berechnung nur auf Basis des Centroids der Box unterstützt.

Änderung: 2.2.0 -- Da für Anwender von PostgreSQL 9.5 der alte hybride Syntax langsamer sein kann, möchten sie diesen Hack eventuell loswerden, falls der Code nur auf PostGIS 2.2+ 9.5+ läuft. Siehe die unteren Beispiele.

Verfügbarkeit: 2.0.0 -- Weak KNN liefert nearest neighbors, welche sich auf die Entfernung der Centroide der Geometrien, anstatt auf den tatsächlichen Entfernungen, stützen. Genaue Ergebnisse für Punkte, ungenau für alle anderen Geometrietypen. Verfügbar ab PostgreSQL 9.1+.

Beispiele

```
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)
```

Then the KNN raw answer:

```
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)
```

Wenn Sie "EXPLAIN ANALYZE" an den zwei Abfragen ausführen, sollte eine Performance Verbesserung im Ausmaß von einer Sekunde auftreten.

Anwender von PostgreSQL < 9.5 können eine hybride Abfrage erstellen, um die echten nearest neighbors aufzufinden. Zuerst eine CTE-Abfrage, welche die Index-unterstützten KNN-Methode anwendet, dann eine exakte Abfrage um eine korrekte Sortierung zu erhalten:

```
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)
```

Siehe auch

ST_DWithin, ST_Distance, <#>

7.10.2.2 |=|

|=| — Gibt die Entfernung zwischen den Trajektorien A und B, am Ort der dichtesten Annäherung, an.

Synopsis

double precision |=|(geometry A , geometry B);

Beschreibung

Der | = | Operator gibt die 3D Entfernung zwischen zwei Trajektorien (Siehe ST_IsValidTrajectory). Dieser entspricht ST_DistanceCPA, da es sich jedoch um einen Operator handelt, kann dieser für nearest neightbor searches mittels eines N-dimensionalen Index verwendet werden (verlangt PostgreSQL 9.5.0 oder höher).

Note
Dieser Operand verwendet die ND GiST Indizes, welche für Geometrien vorhanden sein können. Er unterscheidet sich insofern von anderen Operatoren, die ebenfalls räumliche Indizes verwenden, als der räumliche Index nur dann angewandt wird, wenn sich der Operand in einer ORDER BY Klausel befindet.

Note
Der Index kommt nur zum Tragen, wenn eine der Geometrien eine Konstante ist (sich nicht in einer Subquery/CTE befindet). Z.B. 'SRID=3005;LINESTRINGM(0 0 0,0 0 1)'::geometry und nicht a.geom

Verfügbarkeit: 2.2.0. Index-unterstützt steht erst ab PostgreSQL 9.5+ zur Verfügung.

Beispiele

```
-- Save a literal query trajectory in a psql variable...
\set qt 'ST_AddMeasure(ST_MakeLine(ST_MakePointM(-350,300,0),ST_MakePointM(-410,490,0)) ↩
    ,10,20)'
-- Run the query !
SELECT track_id, dist FROM (
  SELECT track_id, ST_DistanceCPA(tr,:qt) dist
  FROM trajectories
  ORDER BY tr |=| :qt
  LIMIT 5
) foo;
 track_id        dist
----------+-------------------
      395 | 0.576496831518066
      380 |  5.06797130410151
      390 |  7.72262293958322
      385 |   9.8004461358071
      405 |  10.9534397988433
(5 rows)
```

Siehe auch

ST_DistanceCPA, ST_ClosestPointOfApproach, ST_IsValidTrajectory

7.10.2.3 <#>

<#> — Gibt die 2D Entfernung zwischen den Bounding Boxes von A und B zurück

Synopsis

double precision **<#>**(geometry A , geometry B);

Beschreibung

Der <#> Operator gibt die Entfernung zwischen zwei floating point bounding boxes zurück, wobei diese eventuell vom räumlichen Index ausgelesen wird (PostgreSQL 9.1+ vorausgesetzt). Praktikabel falls man eine nearest neighbor Abfrage **approximate** nach der Entfernung sortieren will.

Note
Dieser Operand verwendet sämtliche Indizes, welche für die Geometrien vorhanden sind. Er unterscheidet sich insofern von anderen Operatoren, welche ebenfalls räumliche Indizes verwenden, als der räumliche Index nur dann verwendet wird, falls sich der Operand in einer ORDER BY Klausel befindet.

Note
Der Index kommt nur zum Tragen, wenn eine der Geometrien eine Konstante ist; z.B.: ORDER BY (ST_GeomFromText('POINT(1 2)') <#> geom) anstatt g1.geom <#>.

Verfügbarkeit: 2.0.0 -- KNN steht erst ab PostgreSQL 9.1+ zur Verfügung

Beispiele

```
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)
```

Siehe auch

ST_DWithin, ST_Distance, <->

7.10.2.4 <<->>

<<->> — Gibt die n-D Entfernung zwischen den geometrischen Schwerpunkten der Begrenzungsrechtecke/Bounding Boxes von A und B zurück.

Synopsis

double precision **<<->>**(geometry A , geometry B);

Beschreibung

Der <<−>> Operator gibt die n-D (euklidische) Entfernung zwischen den geometrischen Schwerpunkten der Begrenzungsrechtecke zweier Geometrien zurück. Praktikabel für nearest neighbor **approximate** distance ordering.

Note
Dieser Operator verwendet n-D GiST Indizes, falls diese für die Geometrien vorhanden sind. Er unterscheidet sich insofern von anderen Operatoren, die räumliche Indizes verwenden, indem der räumliche Index nur dann verwendet wird, wenn sich der Operator in einer ORDER BY Klausel befindet.

Note
Der Index kommt nur zum Tragen, wenn eine der Geometrien eine Konstante ist (sich nicht in einer Subquery/CTE befindet). Z.B. 'SRID=3005;POINT(1011102 450541)'::geometry und nicht a.geom

Verfügbarkeit: 2.2.0 -- KNN steht erst ab PostgreSQL 9.1+ zur Verfügung.

Siehe auch

<<#>>, <->

7.10.2.5 <<#>>

<<#>> — Gibt die n-D Entfernung zwischen den Bounding Boxes von A und B zurück.

Synopsis

double precision **<<#>>**(geometry A , geometry B);

Beschreibung

Der <<#>> Operator gibt die Entfernung zwischen zwei floating point bounding boxes zurück, wobei diese eventuell vom räumlichen Index ausgelesen wird (PostgreSQL 9.1+ vorausgesetzt). Praktikabel falls man eine nearest neighbor Abfrage nach der Entfernung sortieren will / **approximate** distance ordering.

Note
Dieser Operand verwendet sämtliche Indizes, welche für die Geometrien vorhanden sind. Er unterscheidet sich insofern von anderen Operatoren, welche ebenfalls räumliche Indizes verwenden, als der räumliche Index nur dann verwendet wird, falls sich der Operand in einer ORDER BY Klausel befindet.

Note
Der Index kommt nur zum Tragen, wenn eine der Geometrien eine Konstante ist; z.B.: ORDER BY (ST_GeomFromText('POINT(1 2)') <<#>> geom) anstatt g1.geom <<#>>.

Verfügbarkeit: 2.2.0 -- KNN steht erst ab PostgreSQL 9.1+ zur Verfügung.

Siehe auch

<<->>, <#>

7.11 Lagevergleiche

7.11.1 Topologische Beziehungen

7.11.1.1 ST_3DIntersects

ST_3DIntersects — Tests if two geometries spatially intersect in 3D - only for points, linestrings, polygons, polyhedral surface (area)

Synopsis

boolean **ST_3DIntersects**(geometry geomA , geometry geomB);

Beschreibung

Overlaps, Touches und Within implizieren räumliche Überschneidung. Wenn irgendeine dieser Eigenschaften TRUE zurückgibt, dann überschneiden sich die geometrischen Objekte auch. Disjoint impliziert FALSE für die räumliche Überschneidung.

Note
This function automatically includes a bounding box comparison that makes use of any spatial indexes that are available on the geometries.

Änderung: 3.0.0 das SFCGAL Back-end wurde entfernt, das GEOS Back-end unterstützt TIN.

Verfügbarkeit: 2.0.0

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

This function supports Polyhedral surfaces.

This function supports Triangles and Triangulated Irregular Network Surfaces (TIN).

This method implements the SQL/MM specification. SQL-MM IEC 13249-3: 5.1

Beispiele mit dem geometrischen Datentyp

```
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)
```

Beispiele mit dem Datentyp TIN

```
SELECT ST_3DIntersects('TIN(((0 0 0,1 0 0,0 1 0,0 0 0)))'::geometry, 'POINT(.1 .1 0)':: ↩
    geometry);
 st_3dintersects
-----------------
 t
```

Siehe auch

ST_Intersects

7.11.1.2 ST_Contains

ST_Contains — Tests if every point of B lies in A, and their interiors have a point in common

Synopsis

boolean **ST_Contains**(geometry geomA, geometry geomB);

Beschreibung

Returns TRUE if geometry A contains geometry B. A contains B if and only if all points of B lie inside (i.e. in the interior or boundary of) A (or equivalently, no points of B lie in the exterior of A), and the interiors of A and B have at least one point in common.

In mathematical terms: $ST\_Contains(A, B) \Leftrightarrow (A \bigcap B = B) \wedge (Int(A) \bigcap Int(B) \neq \varnothing)$

The contains relationship is reflexive: every geometry contains itself. (In contrast, in the ST_ContainsProperly predicate a geometry does *not* properly contain itself.) The relationship is antisymmetric: if `ST_Contains(A,B) = true` and `ST_Contains(B,A) = true`, then the two geometries must be topologically equal (`ST_Equals(A,B) = true`).

ST_Contains is the converse of ST_Within. So, `ST_Contains(A,B) = ST_Within(B,A)`.

Note
Because the interiors must have a common point, a subtlety of the definition is that polygons and lines do *not* contain lines and points lying fully in their boundary. For further details see Subtleties of OGC Covers, Contains, Within. The ST_Covers predicate provides a more inclusive relationship.

Note
This function automatically includes a bounding box comparison that makes use of any spatial indexes that are available on the geometries. Um die Verwendung eines Indices zu vermeiden, kann die Funktion `_ST_Contains` verwendet werden.

Wird durch das GEOS Modul ausgeführt

Enhanced: 2.3.0 Enhancement to PIP short-circuit extended to support MultiPoints with few points. Prior versions only supported point in polygon.

Important
Enhanced: 3.0.0 enabled support for `GEOMETRYCOLLECTION`

Important
Do not use this function with invalid geometries. You will get unexpected results.

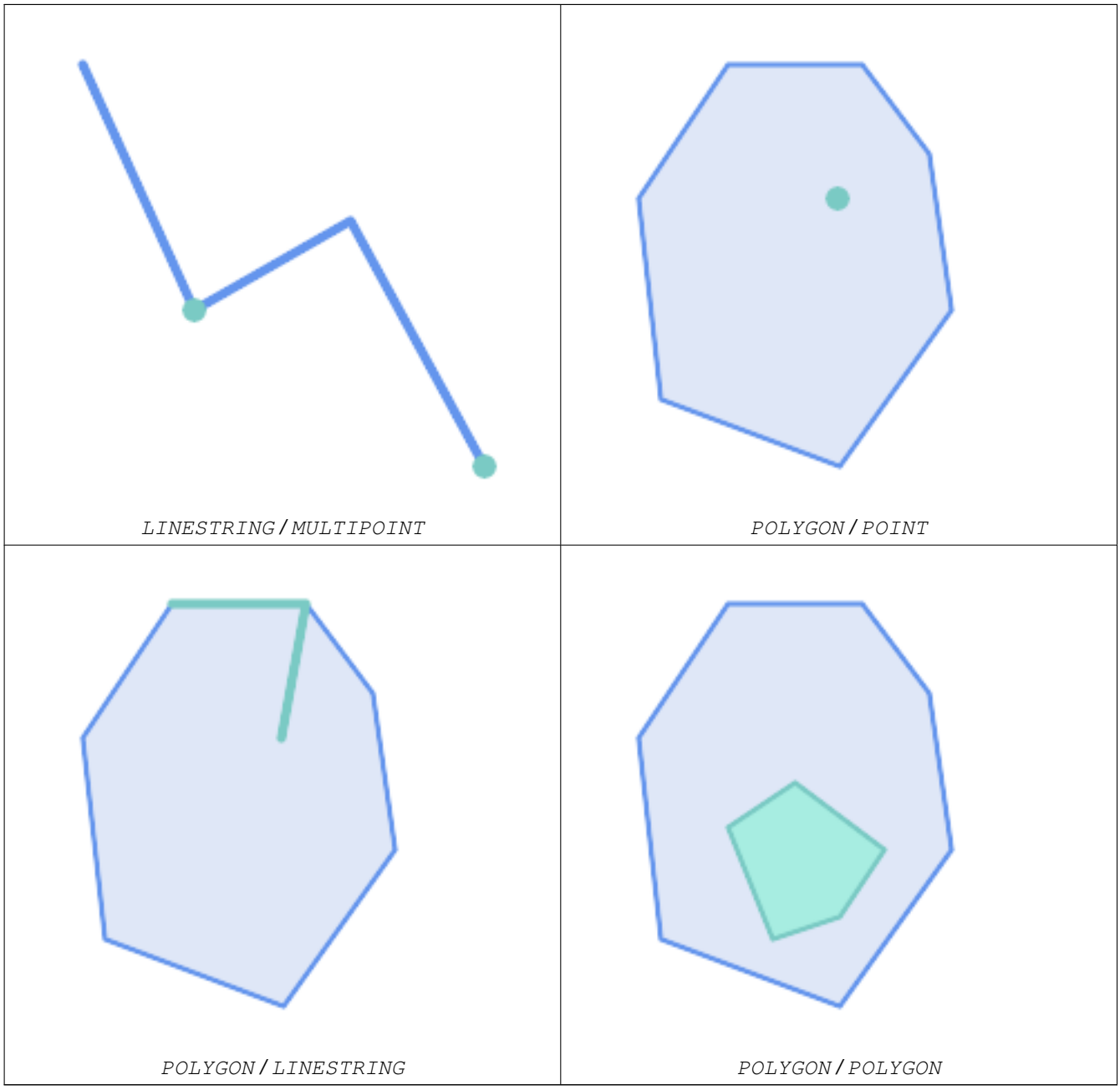
NOTE: this is the "allowable" version that returns a boolean, not an integer.

This method implements the OGC Simple Features Implementation Specification for SQL 1.1. s2.1.1.2 // s2.1.13.3 - same as within(geometry B, geometry A)

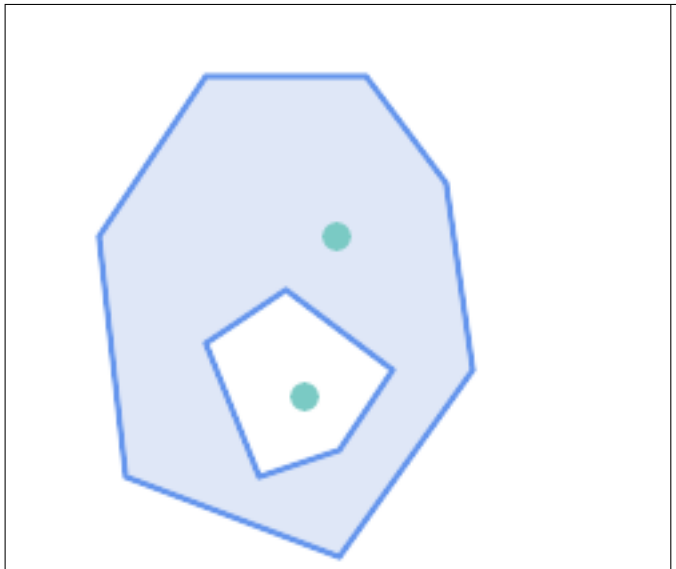
This method implements the SQL/MM specification. SQL-MM 3: 5.1.31

Examples

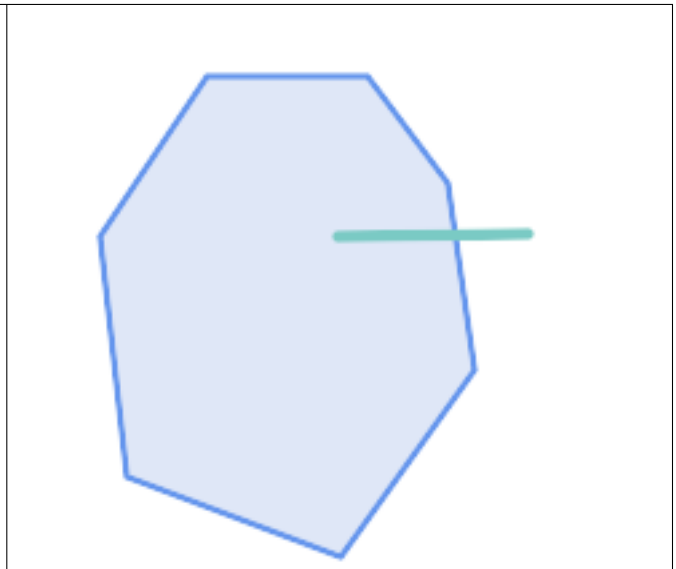
`ST_Contains` returns `TRUE` in the following situations:



`ST_Contains` returns `FALSE` in the following situations:



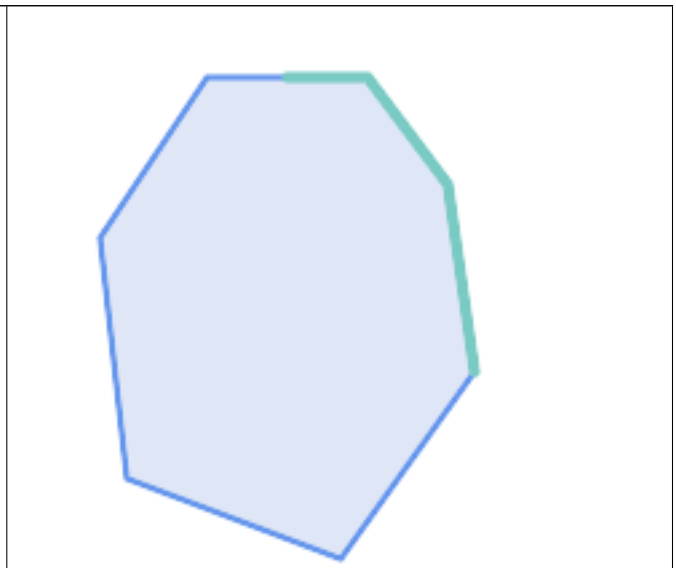
POLYGON/MULTIPOINT



POLYGON/LINESTRING

Due to the interior intersection condition `ST_Contains` returns `FALSE` in the following situations (whereas `ST_Covers` returns `TRUE`):

LINESTRING/POINT



POLYGON/LINESTRING

```
-- A circle within a circle
SELECT ST_Contains(smallc, bigc) As smallcontainsbig,
     ST_Contains(bigc,smallc) As bigcontainssmall,
     ST_Contains(bigc, ST_Union(smallc, bigc)) as bigcontainsunion,
     ST_Equals(bigc, ST_Union(smallc, bigc)) as bigisunion,
     ST_Covers(bigc, ST_ExteriorRing(bigc)) As bigcoversexterior,
     ST_Contains(bigc, ST_ExteriorRing(bigc)) As bigcontainsexterior
FROM (SELECT ST_Buffer(ST_GeomFromText('POINT(1 2)'), 10) As smallc,
       ST_Buffer(ST_GeomFromText('POINT(1 2)'), 20) As bigc) As foo;
```

```
-- Result
  smallcontainsbig | bigcontainssmall | bigcontainsunion | bigisunion | bigcoversexterior | ↩
      bigcontainsexterior
------------------+------------------+------------------+------------+-------------------+----------

 f                | t                | t                | t          | t         | f

-- Example demonstrating difference between contains and contains properly
SELECT ST_GeometryType(geomA) As geomtype, ST_Contains(geomA,geomA) AS acontainsa,  ↩
    ST_ContainsProperly(geomA, geomA) AS acontainspropa,
   ST_Contains(geomA, ST_Boundary(geomA)) As acontainsba, ST_ContainsProperly(geomA,  ↩
       ST_Boundary(geomA)) As acontainspropba
FROM (VALUES ( ST_Buffer(ST_Point(1,1), 5,1) ),
       ( ST_MakeLine(ST_Point(1,1), ST_Point(-1,-1) ) ),
       ( ST_Point(1,1) )
    ) As foo(geomA);

  geomtype    | acontainsa | acontainspropa | acontainsba | acontainspropba
--------------+------------+----------------+-------------+-----------------
ST_Polygon    | t          | f              | f           | f
ST_LineString | t          | f              | f           | f
ST_Point      | t          | t              | f           | f
```

Siehe auch

ST_Boundary, ST_ContainsProperly, ST_Covers, ST_CoveredBy, ST_Equals, ST_Within

7.11.1.3 ST_ContainsProperly

ST_ContainsProperly — Tests if every point of B lies in the interior of A

Synopsis

boolean **ST_ContainsProperly**(geometry geomA, geometry geomB);

Beschreibung

Returns `true` if every point of B lies in the interior of A (or equivalently, no point of B lies in the the boundary or exterior of A).

In mathematical terms: *ST_ContainsProperly(A, B)* $\Leftrightarrow$ $Int(A) \cap B = B$

A contains B properly if the DE-9IM Intersection Matrix for the two geometries matches [T**FF*FF*]

A does not properly contain itself, but does contain itself.

A use for this predicate is computing the intersections of a set of geometries with a large polygonal geometry. Since intersection is a fairly slow operation, it can be more efficient to use containsProperly to filter out test geometries which lie fully inside the area. In these cases the intersection is known a priori to be exactly the original test geometry.

Note
This function automatically includes a bounding box comparison that makes use of any spatial indexes that are available on the geometries. To avoid index use, use the function `_ST_ContainsProperly`.

Note
The advantage of this predicate over ST_Contains and ST_Intersects is that it can be computed more efficiently, with no need to compute topology at individual points.

Performed by the GEOS module.

Availability: 1.4.0

Important
Enhanced: 3.0.0 enabled support for `GEOMETRYCOLLECTION`

Important
Do not use this function with invalid geometries. You will get unexpected results.

Examples

```
--a circle within a circle
  SELECT ST_ContainsProperly(smallc, bigc) As smallcontainspropbig,
  ST_ContainsProperly(bigc,smallc) As bigcontainspropsmall,
  ST_ContainsProperly(bigc, ST_Union(smallc, bigc)) as bigcontainspropunion,
  ST_Equals(bigc, ST_Union(smallc, bigc)) as bigisunion,
  ST_Covers(bigc, ST_ExteriorRing(bigc)) As bigcoversexterior,
  ST_ContainsProperly(bigc, ST_ExteriorRing(bigc)) As bigcontainsexterior
  FROM (SELECT ST_Buffer(ST_GeomFromText('POINT(1 2)'), 10) As smallc,
  ST_Buffer(ST_GeomFromText('POINT(1 2)'), 20) As bigc) As foo;
  --Result
  smallcontainspropbig | bigcontainspropsmall | bigcontainspropunion | bigisunion |  ↩
      bigcoversexterior | bigcontainsexterior
------------------+------------------+------------------+------------+------------------+----------

 f                    | t                   | f                    | t          | t  ↩
                     | f

 --example demonstrating difference between contains and contains properly
 SELECT ST_GeometryType(geomA) As geomtype, ST_Contains(geomA,geomA) AS acontainsa,  ↩
     ST_ContainsProperly(geomA, geomA) AS acontainspropa,
 ST_Contains(geomA, ST_Boundary(geomA)) As acontainsba, ST_ContainsProperly(geomA,  ↩
     ST_Boundary(geomA)) As acontainspropba
 FROM (VALUES ( ST_Buffer(ST_Point(1,1), 5,1) ),
      ( ST_MakeLine(ST_Point(1,1), ST_Point(-1,-1) ) ),
      ( ST_Point(1,1) )
  ) As foo(geomA);

  geomtype    | acontainsa | acontainspropa | acontainsba | acontainspropba
--------------+------------+----------------+-------------+-----------------
ST_Polygon    | t          | f              | f           | f
ST_LineString | t          | f              | f           | f
ST_Point      | t          | t              | f           | f
```

Siehe auch

ST_GeometryType, ST_Boundary, ST_Contains, ST_Covers, ST_CoveredBy, ST_Equals, ST_Relate, ST_Within

7.11.1.4 ST_CoveredBy

ST_CoveredBy — Tests if every point of A lies in B

Synopsis

boolean **ST_CoveredBy**(geometry geomA, geometry geomB);
boolean **ST_CoveredBy**(geography geogA, geography geogB);

Beschreibung

Returns `true` if every point in Geometry/Geography A lies inside (i.e. intersects the interior or boundary of) Geometry/Geography B. Equivalently, tests that no point of A lies outside (in the exterior of) B.

In mathematical terms: *ST_CoveredBy(A, B)* $\Leftrightarrow A \bigcap B = A$

ST_CoveredBy is the converse of ST_Covers. So, `ST_CoveredBy(A,B) = ST_Covers(B,A)`.

Generally this function should be used instead of ST_Within, since it has a simpler definition which does not have the quirk that "boundaries are not within their geometry".

Note
This function automatically includes a bounding box comparison that makes use of any spatial indexes that are available on the geometries. To avoid index use, use the function `_ST_CoveredBy`.

Important
Enhanced: 3.0.0 enabled support for `GEOMETRYCOLLECTION`

Important
Do not use this function with invalid geometries. You will get unexpected results.

Wird durch das GEOS Modul ausgeführt

Availability: 1.2.2

NOTE: this is the "allowable" version that returns a boolean, not an integer.

Not an OGC standard, but Oracle has it too.

Examples

```
--a circle coveredby a circle
SELECT ST_CoveredBy(smallc,smallc) As smallinsmall,
  ST_CoveredBy(smallc, bigc) As smallcoveredbybig,
  ST_CoveredBy(ST_ExteriorRing(bigc), bigc) As exteriorcoveredbybig,
  ST_Within(ST_ExteriorRing(bigc),bigc) As exeriorwithinbig
FROM (SELECT ST_Buffer(ST_GeomFromText('POINT(1 2)'), 10) As smallc,
  ST_Buffer(ST_GeomFromText('POINT(1 2)'), 20) As bigc) As foo;
  --Result
 smallinsmall | smallcoveredbybig | exteriorcoveredbybig | exeriorwithinbig
--------------+-------------------+----------------------+------------------
 t            | t                 | t                    | f
(1 row)
```

Siehe auch

ST_Contains, ST_Covers, ST_ExteriorRing, ST_Within

7.11.1.5 ST_Covers

ST_Covers — Tests if every point of B lies in A

Synopsis

boolean **ST_Covers**(geometry geomA, geometry geomB);
boolean **ST_Covers**(geography geogpolyA, geography geogpointB);

Beschreibung

Returns `true` if every point in Geometry/Geography B lies inside (i.e. intersects the interior or boundary of) Geometry/Geography A. Equivalently, tests that no point of B lies outside (in the exterior of) A.

In mathematical terms: *ST_Covers(A, B)* $\Leftrightarrow A \bigcap B = B$

ST_Covers is the converse of ST_CoveredBy. So, `ST_Covers(A,B) = ST_CoveredBy(B,A)`.

Generally this function should be used instead of ST_Contains, since it has a simpler definition which does not have the quirk that "geometries do not contain their boundary".

Note
This function automatically includes a bounding box comparison that makes use of any spatial indexes that are available on the geometries. To avoid index use, use the function `_ST_Covers`.

Important
Enhanced: 3.0.0 enabled support for `GEOMETRYCOLLECTION`

Important
Do not use this function with invalid geometries. You will get unexpected results.

Wird durch das GEOS Modul ausgeführt

Enhanced: 2.4.0 Support for polygon in polygon and line in polygon added for geography type

Enhanced: 2.3.0 Enhancement to PIP short-circuit for geometry extended to support MultiPoints with few points. Prior versions only supported point in polygon.

Availability: 1.5 - support for geography was introduced.

Availability: 1.2.2

NOTE: this is the "allowable" version that returns a boolean, not an integer.

Not an OGC standard, but Oracle has it too.

Examples

Geometry example

```
--a circle covering a circle
SELECT ST_Covers(smallc,smallc) As smallinsmall,
  ST_Covers(smallc, bigc) As smallcoversbig,
  ST_Covers(bigc, ST_ExteriorRing(bigc)) As bigcoversexterior,
  ST_Contains(bigc, ST_ExteriorRing(bigc)) As bigcontainsexterior
FROM (SELECT ST_Buffer(ST_GeomFromText('POINT(1 2)'), 10) As smallc,
  ST_Buffer(ST_GeomFromText('POINT(1 2)'), 20) As bigc) As foo;
  --Result
 smallinsmall | smallcoversbig | bigcoversexterior | bigcontainsexterior
--------------+----------------+-------------------+---------------------
 t            | f              | t                 | f
(1 row)
```

Geeography Example

```
-- a point with a 300 meter buffer compared to a point, a point and its 10 meter buffer
SELECT ST_Covers(geog_poly, geog_pt) As poly_covers_pt,
  ST_Covers(ST_Buffer(geog_pt,10), geog_pt) As buff_10m_covers_cent
  FROM (SELECT ST_Buffer(ST_GeogFromText('SRID=4326;POINT(-99.327 31.4821)'), 300) As  ↩
      geog_poly,
        ST_GeogFromText('SRID=4326;POINT(-99.33 31.483)') As geog_pt ) As foo;

 poly_covers_pt | buff_10m_covers_cent
----------------+----------------------
 f              | t
```

Siehe auch

ST_Contains, ST_CoveredBy, ST_Within

7.11.1.6 ST_Crosses

ST_Crosses — Tests if two geometries have some, but not all, interior points in common

Synopsis

boolean **ST_Crosses**(geometry g1, geometry g2);

Beschreibung

Compares two geometry objects and returns `true` if their intersection "spatially crosses"; that is, the geometries have some, but not all interior points in common. The intersection of the interiors of the geometries must be non-empty and must have dimension less than the maximum dimension of the two input geometries, and the intersection of the two geometries must not equal either geometry. Otherwise, it returns `false`. The crosses relation is symmetric and irreflexive.

In mathematical terms: $ST\_Crosses(A, B) \Leftrightarrow (dim(\ Int(A) \bigcap Int(B)\ ) < max(\ dim(\ Int(A)\ ), dim(\ Int(B)\ )\ )) \wedge (A \bigcap B \neq A) \wedge (A \bigcap B \neq B)$

Geometries cross if their DE-9IM Intersection Matrix matches:

- `T*T******` for Point/Line, Point/Area, and Line/Area situations
- `T*****T**` for Line/Point, Area/Point, and Area/Line situations

- `0********` for Line/Line situations
- the result is `false` for Point/Point and Area/Area situations

Note
The OpenGIS Simple Features Specification defines this predicate only for Point/Line, Point/Area, Line/Line, and Line/Area situations. JTS / GEOS extends the definition to apply to Line/Point, Area/Point and Area/Line situations as well. This makes the relation symmetric.

Note
This function automatically includes a bounding box comparison that makes use of any spatial indexes that are available on the geometries.

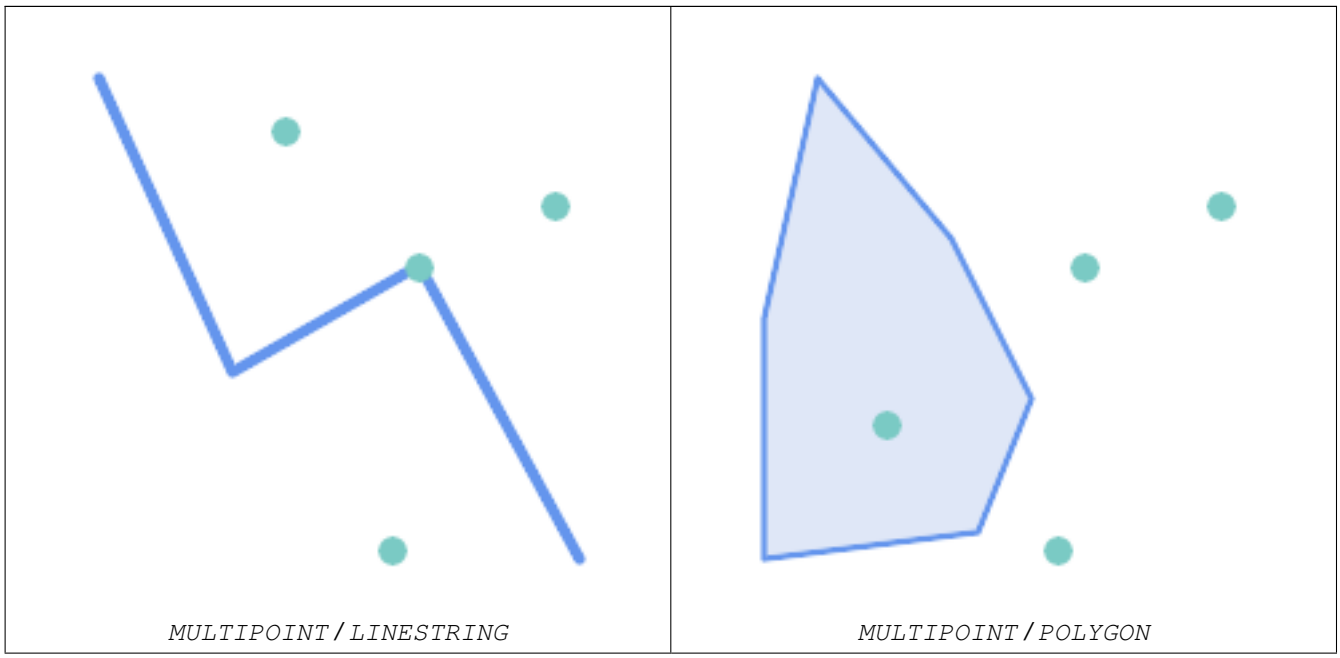
Important
Enhanced: 3.0.0 enabled support for `GEOMETRYCOLLECTION`

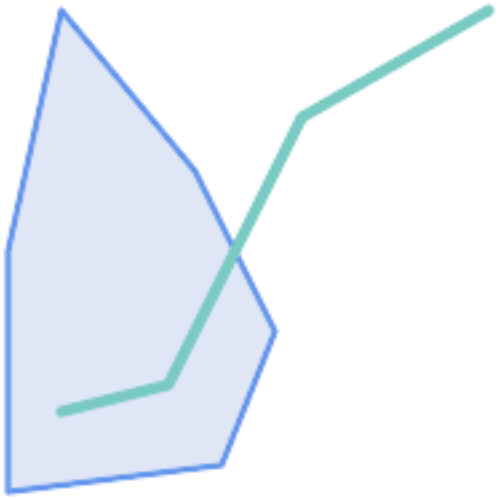
This method implements the OGC Simple Features Implementation Specification for SQL 1.1. s2.1.13.3

This method implements the SQL/MM specification. SQL-MM 3: 5.1.29

Examples

The following situations all return `true`.





LINESTRING/POLYGON

LINESTRING/LINESTRING

Consider a situation where a user has two tables: a table of roads and a table of highways.

```
CREATE TABLE roads (
  id serial NOT NULL,
  geom geometry,
  CONSTRAINT roads_pkey PRIMARY KEY ( ↩
     road_id)
);
```

```
CREATE TABLE highways (
  id serial NOT NULL,
  the_gem geometry,
  CONSTRAINT roads_pkey PRIMARY KEY ( ↩
     road_id)
);
```

To determine a list of roads that cross a highway, use a query similiar to:

```
SELECT roads.id
FROM roads, highways
WHERE ST_Crosses(roads.geom, highways.geom);
```

Siehe auch

ST_Contains, ST_Overlaps

7.11.1.7 ST_Disjoint

ST_Disjoint — Tests if two geometries have no points in common

Synopsis

boolean **ST_Disjoint**(geometry A , geometry B);

Beschreibung

Returns `true` if two geometries are disjoint. Geometries are disjoint if they have no point in common.

If any other spatial relationship is true for a pair of geometries, they are not disjoint. Disjoint implies that ST_Intersects is false.

In mathematical terms: *ST_Disjoint(A, B)* $\Leftrightarrow A \bigcap B = \varnothing$

Important
Enhanced: 3.0.0 enabled support for `GEOMETRYCOLLECTION`

Wird durch das GEOS Modul ausgeführt

Note
This function call does not use indexes. A negated ST_Intersects predicate can be used as a more performant alternative that uses indexes: `ST_Disjoint(A,B) = NOT ST_Intersects(A,B)`

Note
NOTE: this is the "allowable" version that returns a boolean, not an integer.

This method implements the OGC Simple Features Implementation Specification for SQL 1.1. s2.1.1.2 //s2.1.13.3 - a.Relate(b, 'FF*FF****')

This method implements the SQL/MM specification. SQL-MM 3: 5.1.26

Examples

```
SELECT ST_Disjoint('POINT(0 0)'::geometry, 'LINESTRING ( 2 0, 0 2 )'::geometry);
 st_disjoint
---------------
 t
(1 row)
SELECT ST_Disjoint('POINT(0 0)'::geometry, 'LINESTRING ( 0 0, 0 2 )'::geometry);
 st_disjoint
---------------
 f
(1 row)
```

Siehe auch

ST_Intersects

7.11.1.8 ST_Equals

ST_Equals — Tests if two geometries include the same set of points

Synopsis

boolean **ST_Equals**(geometry A, geometry B);

Beschreibung

Returns `true` if the given geometries are "topologically equal". Use this for a 'better' answer than '='. Topological equality means that the geometries have the same dimension, and their point-sets occupy the same space. This means that the order of vertices may be different in topologically equal geometries. To verify the order of points is consistent use ST_OrderingEquals (it must be noted ST_OrderingEquals is a little more stringent than simply verifying order of points are the same).

In mathematical terms: *ST_Equals(A, B)* $\Leftrightarrow$ *A = B*

The following relation holds: *ST_Equals(A, B)* $\Leftrightarrow$ *ST_Within(A,B)* $\wedge$ *ST_Within(B,A)*

Important
Enhanced: 3.0.0 enabled support for `GEOMETRYCOLLECTION`

This method implements the OGC Simple Features Implementation Specification for SQL 1.1. s2.1.1.2

This method implements the SQL/MM specification. SQL-MM 3: 5.1.24

Changed: 2.2.0 Returns true even for invalid geometries if they are binary equal

Examples

```
SELECT ST_Equals(ST_GeomFromText('LINESTRING(0 0, 10 10)'),
    ST_GeomFromText('LINESTRING(0 0, 5 5, 10 10)'));
 st_equals
-----------
 t
(1 row)

SELECT ST_Equals(ST_Reverse(ST_GeomFromText('LINESTRING(0 0, 10 10)')),
    ST_GeomFromText('LINESTRING(0 0, 5 5, 10 10)'));
 st_equals
-----------
 t
(1 row)
```

Siehe auch

ST_IsValid, ST_OrderingEquals, ST_Reverse, ST_Within

7.11.1.9 ST_Intersects

ST_Intersects — Tests if two geometries intersect (they have at least one point in common)

Synopsis

boolean **ST_Intersects**(geometry geomA , geometry geomB);
boolean **ST_Intersects**(geography geogA , geography geogB);

Beschreibung

Returns `true` if two geometries intersect. Geometries intersect if they have any point in common.

For geography, a distance tolerance of 0.00001 meters is used (so points that are very close are considered to intersect).

In mathematical terms: *ST_Intersects(A, B)* $\Leftrightarrow A \bigcap B \neq \varnothing$

Geometries intersect if their DE-9IM Intersection Matrix matches one of:

- `T********`
- `*T*******`
- `***T*****`
- `****T****`

Spatial intersection is implied by all the other spatial relationship tests, except ST_Disjoint, which tests that geometries do NOT intersect.

Note
This function automatically includes a bounding box comparison that makes use of any spatial indexes that are available on the geometries.

Changed: 3.0.0 SFCGAL version removed and native support for 2D TINS added.

Enhanced: 2.5.0 Supports GEOMETRYCOLLECTION.

Enhanced: 2.3.0 Enhancement to PIP short-circuit extended to support MultiPoints with few points. Prior versions only supported point in polygon.

Performed by the GEOS module (for geometry), geography is native

Availability: 1.5 support for geography was introduced.

Note
For geography, this function has a distance tolerance of about 0.00001 meters and uses the sphere rather than spheroid calculation.

Note
NOTE: this is the "allowable" version that returns a boolean, not an integer.

This method implements the OGC Simple Features Implementation Specification for SQL 1.1. s2.1.1.2 //s2.1.13.3 - ST_Intersects(g1, g2) --> Not (ST_Disjoint(g1, g2))

This method implements the SQL/MM specification. SQL-MM 3: 5.1.27

This method supports Circular Strings and Curves.

This function supports Triangles and Triangulated Irregular Network Surfaces (TIN).

Beispiele mit dem geometrischen Datentyp

```
SELECT ST_Intersects('POINT(0 0)'::geometry, 'LINESTRING ( 2 0, 0 2 )'::geometry);
 st_intersects
---------------
 f
(1 row)
SELECT ST_Intersects('POINT(0 0)'::geometry, 'LINESTRING ( 0 0, 0 2 )'::geometry);
 st_intersects
---------------
 t
(1 row)

-- Look up in table. Make sure table has a GiST index on geometry column for faster lookup.
SELECT id, name FROM cities WHERE ST_Intersects(geom, 'SRID=4326;POLYGON((28 53,27.707  ↩
    52.293,27 52,26.293 52.293,26 53,26.293 53.707,27 54,27.707 53.707,28 53))');
 id | name
----+-------
  2 | Minsk
(1 row)
```

Geography Examples

```
SELECT ST_Intersects(
    'SRID=4326;LINESTRING(-43.23456 72.4567,-43.23456 72.4568)'::geography,
    'SRID=4326;POINT(-43.23456 72.4567772)'::geography
    );

 st_intersects
---------------
t
```

Siehe auch

&&, ST_3DIntersects, ST_Disjoint

7.11.1.10 ST_LineCrossingDirection

ST_LineCrossingDirection — Returns a number indicating the crossing behavior of two LineStrings

Synopsis

integer **ST_LineCrossingDirection**(geometry linestringA, geometry linestringB);

Beschreibung

Given two linestrings returns an integer between -3 and 3 indicating what kind of crossing behavior exists between them. 0 indicates no crossing. This is only supported for `LINESTRING`s.

The crossing number has the following meaning:

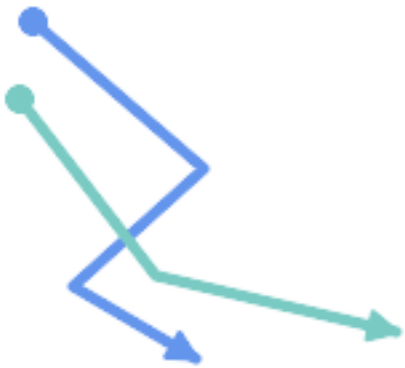
- 0: LINE NO CROSS
- -1: LINE CROSS LEFT
- 1: LINE CROSS RIGHT

- -2: LINE MULTICROSS END LEFT
- 2: LINE MULTICROSS END RIGHT
- -3: LINE MULTICROSS END SAME FIRST LEFT
- 3: LINE MULTICROSS END SAME FIRST RIGHT

Availability: 1.4

Examples

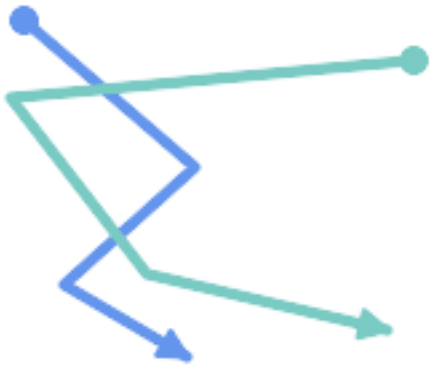
Example: LINE CROSS LEFT and LINE CROSS RIGHT



Blue: Line A; Green: Line B

```
SELECT ST_LineCrossingDirection(lineA, lineB) As A_cross_B,
       ST_LineCrossingDirection(lineB, lineA) As B_cross_A
FROM (SELECT
  ST_GeomFromText('LINESTRING(25 169,89 114,40 70,86 43)') As lineA,
  ST_GeomFromText('LINESTRING (20 140, 71 74, 161 53)') As lineB
  ) As foo;

 A_cross_B | B_cross_A
-----------+-----------
        -1 |         1
```

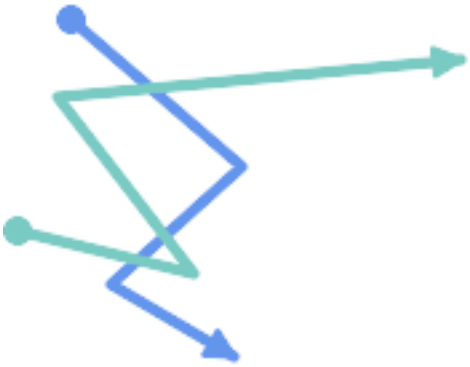
Example: LINE MULTICROSS END SAME FIRST LEFT and LINE MULTICROSS END SAME FIRST RIGHT



Blue: Line A; Green: Line B

```
SELECT ST_LineCrossingDirection(lineA, lineB) As A_cross_B,
       ST_LineCrossingDirection(lineB, lineA) As B_cross_A
FROM (SELECT
 ST_GeomFromText('LINESTRING(25 169,89 114,40 70,86 43)') As lineA,
 ST_GeomFromText('LINESTRING(171 154,20 140,71 74,161 53)') As lineB
  ) As foo;

 A_cross_B | B_cross_A
-----------+-----------
         3 |        -3
```

Example: LINE MULTICROSS END LEFT and LINE MULTICROSS END RIGHT



Blue: Line A; Green: Line B

```
SELECT ST_LineCrossingDirection(lineA, lineB) As A_cross_B,
       ST_LineCrossingDirection(lineB, lineA) As B_cross_A
FROM (SELECT
  ST_GeomFromText('LINESTRING(25 169,89 114,40 70,86 43)') As lineA,
  ST_GeomFromText('LINESTRING(5 90, 71 74, 20 140, 171 154)') As lineB
  ) As foo;
```

```
 A_cross_B | B_cross_A
-----------+-----------
        -2 |         2
```

Example: Finds all streets that cross

```
SELECT s1.gid, s2.gid, ST_LineCrossingDirection(s1.geom, s2.geom)
  FROM streets s1 CROSS JOIN streets s2
         ON (s1.gid != s2.gid AND s1.geom && s2.geom )
WHERE ST_LineCrossingDirection(s1.geom, s2.geom)
> 0;
```

Siehe auch

ST_Crosses

7.11.1.11 ST_OrderingEquals

ST_OrderingEquals — Tests if two geometries represent the same geometry and have points in the same directional order

Synopsis

boolean **ST_OrderingEquals**(geometry A, geometry B);

Beschreibung

ST_OrderingEquals compares two geometries and returns t (TRUE) if the geometries are equal and the coordinates are in the same order; otherwise it returns f (FALSE).

Note
This function is implemented as per the ArcSDE SQL specification rather than SQL-MM. http://edndoc.esri.com/arcsde/9.1/sql_api/sqlapi3.htm#ST_OrderingEquals

This method implements the SQL/MM specification. SQL-MM 3: 5.1.43

Examples

```
SELECT ST_OrderingEquals(ST_GeomFromText('LINESTRING(0 0, 10 10)'),
    ST_GeomFromText('LINESTRING(0 0, 5 5, 10 10)'));
 st_orderingequals
-----------
 f
(1 row)

SELECT ST_OrderingEquals(ST_GeomFromText('LINESTRING(0 0, 10 10)'),
    ST_GeomFromText('LINESTRING(0 0, 0 0, 10 10)'));
 st_orderingequals
-----------
 t
(1 row)

SELECT ST_OrderingEquals(ST_Reverse(ST_GeomFromText('LINESTRING(0 0, 10 10)')),
```

```
    ST_GeomFromText('LINESTRING(0 0, 0 0, 10 10)'));
 st_orderingequals
-----------
 f
(1 row)
```

Siehe auch

&&, ST_Equals, ST_Reverse

7.11.1.12 ST_Overlaps

ST_Overlaps — Tests if two geometries have the same dimension and intersect, but each has at least one point not in the other

Synopsis

boolean **ST_Overlaps**(geometry A, geometry B);

Beschreibung

Returns TRUE if geometry A and B "spatially overlap". Two geometries overlap if they have the same dimension, their interiors intersect in that dimension. and each has at least one point inside the other (or equivalently, neither one covers the other). The overlaps relation is symmetric and irreflexive.

In mathematical terms: $ST\_Overlaps(A, B) \Leftrightarrow (dim(A) = dim(B) = dim(Int(A) \bigcap Int(B))) \wedge (A \bigcap B \neq A) \wedge (A \bigcap B \neq B)$

Note
This function automatically includes a bounding box comparison that makes use of any spatial indexes that are available on the geometries. To avoid index use, use the function `_ST_Overlaps`.

Wird durch das GEOS Modul ausgeführt

Important
Enhanced: 3.0.0 enabled support for `GEOMETRYCOLLECTION`

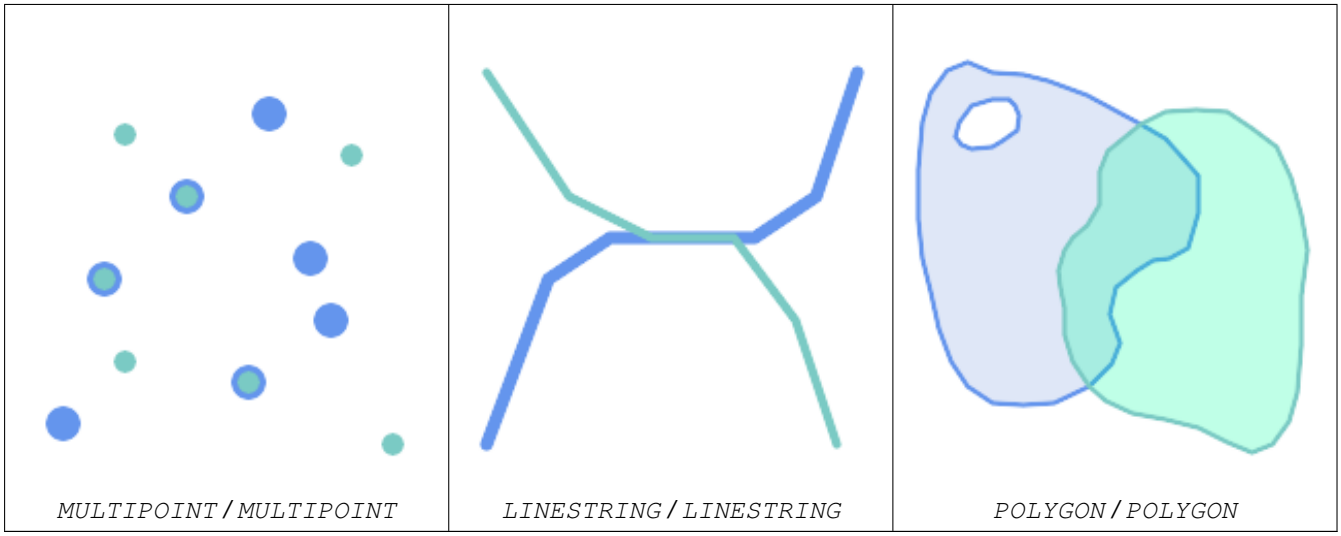
NOTE: this is the "allowable" version that returns a boolean, not an integer.

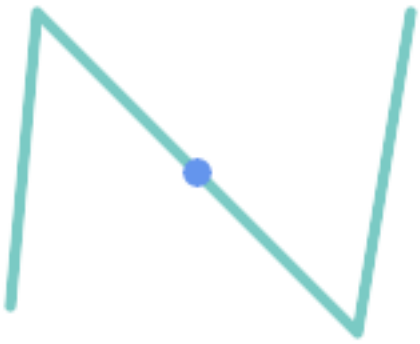
This method implements the OGC Simple Features Implementation Specification for SQL 1.1. s2.1.1.2 // s2.1.13.3

This method implements the SQL/MM specification. SQL-MM 3: 5.1.32

Examples

`ST_Overlaps` returns `TRUE` in the following situations:



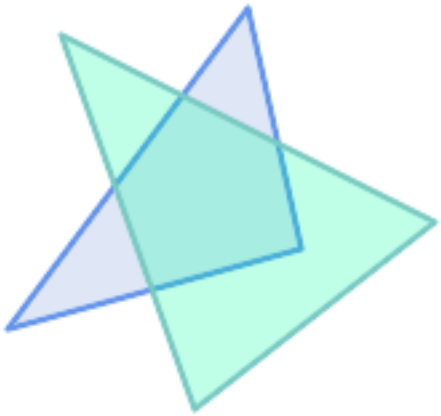


A Point on a LineString is contained, but since it has lower dimension it does not overlap or cross.

```
SELECT ST_Overlaps(a,b) AS overlaps,       ST_Crosses(a,b) AS crosses,
       ST_Intersects(a, b) AS intersects,  ST_Contains(b,a) AS b_contains_a
FROM (SELECT ST_GeomFromText('POINT (100 100)') As a,
             ST_GeomFromText('LINESTRING (30 50, 40 160, 160 40, 180 160)')  AS b) AS t

overlaps | crosses | intersects | b_contains_a
---------+---------------------+--------------
f        | f       | t          | t
```

A LineString that partly covers a Polygon intersects and crosses, but does not overlap since it has different dimension.

```
SELECT ST_Overlaps(a,b) AS overlaps,        ST_Crosses(a,b) AS crosses,
       ST_Intersects(a, b) AS intersects,   ST_Contains(a,b) AS contains
FROM (SELECT ST_GeomFromText('POLYGON ((40 170, 90 30, 180 100, 40 170))') AS a,
             ST_GeomFromText('LINESTRING(10 10, 190 190)') AS b) AS t;

 overlap | crosses | intersects | contains
---------+---------+------------+--------------
 f       | t       | t          | f
```



Two Polygons that intersect but with neither contained by the other overlap, but do not cross because their intersection has the same dimension.

```
SELECT ST_Overlaps(a,b) AS overlaps,       ST_Crosses(a,b) AS crosses,
       ST_Intersects(a, b) AS intersects,  ST_Contains(b, a) AS b_contains_a,
       ST_Dimension(a) AS dim_a, ST_Dimension(b) AS dim_b,
       ST_Dimension(ST_Intersection(a,b)) AS dim_int
FROM (SELECT ST_GeomFromText('POLYGON ((40 170, 90 30, 180 100, 40 170))') AS a,
             ST_GeomFromText('POLYGON ((110 180, 20 60, 130 90, 110 180))') AS b) As t;

 overlaps | crosses | intersects | b_contains_a | dim_a | dim_b | dim_int
```

```
----------+---------+------------+--------------+-------+-------+-----------
 t        | f       | t          | f            |     2 |     2 |         2
```

Siehe auch

ST_Contains, ST_Crosses, ST_Dimension, ST_Intersects

7.11.1.13 ST_Relate

ST_Relate — Tests if two geometries have a topological relationship matching an Intersection Matrix pattern, or computes their Intersection Matrix

Synopsis

boolean **ST_Relate**(geometry geomA, geometry geomB, text intersectionMatrixPattern);
text **ST_Relate**(geometry geomA, geometry geomB);
text **ST_Relate**(geometry geomA, geometry geomB, integer boundaryNodeRule);

Beschreibung

These functions allow testing and evaluating the spatial (topological) relationship between two geometries, as defined by the Dimensionally Extended 9-Intersection Model (DE-9IM).

The DE-9IM is specified as a 9-element matrix indicating the dimension of the intersections between the Interior, Boundary and Exterior of two geometries. It is represented by a 9-character text string using the symbols 'F', '0', '1', '2' (e.g. `'FF1FF0102'`).

A specific kind of spatial relationship can be tested by matching the intersection matrix to an *intersection matrix pattern*. Patterns can include the additional symbols 'T' (meaning "intersection is non-empty") and '*' (meaning "any value"). Common spatial relationships are provided by the named functions ST_Contains, ST_ContainsProperly, ST_Covers, ST_CoveredBy, ST_Crosses, ST_Disjoint, ST_Equals, ST_Intersects, ST_Overlaps, ST_Touches, and ST_Within. Using an explicit pattern allows testing multiple conditions of intersects, crosses, etc in one step. It also allows testing spatial relationships which do not have a named spatial relationship function. For example, the relationship "Interior-Intersects" has the DE-9IM pattern `T********`, which is not evaluated by any named predicate.

For more information refer to Section 5.1.

Variant 1: Tests if two geometries are spatially related according to the given `intersectionMatrixPattern`.

Note
Unlike most of the named spatial relationship predicates, this does NOT automatically include an index call. The reason is that some relationships are true for geometries which do NOT intersect (e.g. Disjoint). If you are using a relationship pattern that requires intersection, then include the && index call.

Note
It is better to use a named relationship function if available, since they automatically use a spatial index where one exists. Also, they may implement performance optimizations which are not available with full relate evalation.

Variant 2: Returns the DE-9IM matrix string for the spatial relationship between the two input geometries. The matrix string can be tested for matching a DE-9IM pattern using ST_RelateMatch.

Variant 3: Like variant 2, but allows specifying a **Boundary Node Rule**. A boundary node rule allows finer control over whether the endpoints of MultiLineStrings are considered to lie in the DE-9IM Interior or Boundary. The `boundaryNodeRule` values are:

- `1`: **OGC-Mod2** - line endpoints are in the Boundary if they occur an odd number of times. This is the rule defined by the OGC SFS standard, and is the default for `ST_Relate`.
- `2`: **Endpoint** - all endpoints are in the Boundary.
- `3`: **MultivalentEndpoint** - endpoints are in the Boundary if they occur more than once. In other words, the boundary is all the "attached" or "inner" endpoints (but not the "unattached/outer" ones).
- `4`: **MonovalentEndpoint** - endpoints are in the Boundary if they occur only once. In other words, the boundary is all the "unattached" or "outer" endpoints.

This function is not in the OGC spec, but is implied. see s2.1.13.2

This method implements the OGC Simple Features Implementation Specification for SQL 1.1. s2.1.1.2 // s2.1.13.3

This method implements the SQL/MM specification. SQL-MM 3: 5.1.25

Wird durch das GEOS Modul ausgeführt

Enhanced: 2.0.0 - added support for specifying boundary node rule.

Important
Enhanced: 3.0.0 enabled support for `GEOMETRYCOLLECTION`

Examples

Using the boolean-valued function to test spatial relationships.

```
SELECT ST_Relate('POINT(1 2)', ST_Buffer( 'POINT(1 2)', 2), '0FFFFF212');
st_relate
-----------
t

SELECT ST_Relate(POINT(1 2)', ST_Buffer( 'POINT(1 2)', 2), '*FF*FF212');
st_relate
-----------
t
```

Testing a custom spatial relationship pattern as a query condition, with `&&` to enable using a spatial index.

```
-- Find compounds that properly intersect (not just touch) a poly (Interior Intersects)

SELECT c.* , p.name As poly_name
    FROM polys AS p
    INNER JOIN compounds As c
          ON c.geom && p.geom
             AND ST_Relate(p.geom, c.geom,'T********');
```

Computing the intersection matrix for spatial relationships.

```
SELECT ST_Relate( 'POINT(1 2)',
                  ST_Buffer( 'POINT(1 2)', 2));
-----------
0FFFFF212

SELECT ST_Relate( 'LINESTRING(1 2, 3 4)',
                  'LINESTRING(5 6, 7 8)' );
-----------
FF1FF0102
```

Using different Boundary Node Rules to compute the spatial relationship between a LineString and a MultiLineString with a duplicate endpoint `(3 3)`:

- Using the **OGC-Mod2** rule (1) the duplicate endpoint is in the **interior** of the MultiLineString, so the DE-9IM matrix entry [aB:bI] is `0` and [aB:bB] is `F`.
- Using the **Endpoint** rule (2) the duplicate endpoint is in the **boundary** of the MultiLineString, so the DE-9IM matrix entry [aB:bI] is `F` and [aB:bB] is `0`.

```
WITH data AS (SELECT
  'LINESTRING(1 1, 3 3)'::geometry AS a_line,
  'MULTILINESTRING((3 3, 3 5), (3 3, 5 3))':: geometry AS b_multiline
)
SELECT ST_Relate( a_line, b_multiline, 1) AS bnr_mod2,
       ST_Relate( a_line, b_multiline, 2) AS bnr_endpoint
    FROM data;

 bnr_mod2  | bnr_endpoint
-----------+--------------
 FF10F0102 | FF1F00102
```

Siehe auch

Section 5.1, ST_RelateMatch, ST_Contains, ST_ContainsProperly, ST_Covers, ST_CoveredBy, ST_Crosses, ST_Disjoint, ST_Equals, ST_Intersects, ST_Overlaps, ST_Touches, ST_Within

7.11.1.14 ST_RelateMatch

ST_RelateMatch — Tests if a DE-9IM Intersection Matrix matches an Intersection Matrix pattern

Synopsis

boolean **ST_RelateMatch**(text intersectionMatrix, text intersectionMatrixPattern);

Beschreibung

Tests if a Dimensionally Extended 9-Intersection Model (DE-9IM) `intersectionMatrix` value satisfies an `intersectionMatr`
Intersection matrix values can be computed by ST_Relate.

For more information refer to Section 5.1.

Wird durch das GEOS Modul ausgeführt

Verfügbarkeit: 2.0.0

Examples

```
SELECT ST_RelateMatch('101202FFF', 'TTTTTTFFF') ;
-- result --
t
```

Patterns for common spatial relationships matched against intersection matrix values, for a line in various positions relative to a polygon

```
SELECT pat.name AS relationship, pat.val AS pattern,
       mat.name AS position, mat.val AS matrix,
       ST_RelateMatch(mat.val, pat.val) AS match
    FROM (VALUES ( 'Equality', 'T1FF1FFF1' ),
                 ( 'Overlaps', 'T*T***T**' ),
                 ( 'Within',   'T*F**F***' ),
                 ( 'Disjoint', 'FF*FF****' )) AS pat(name,val)
    CROSS JOIN
        (VALUES  ('non-intersecting', 'FF1FF0212'),
                 ('overlapping',      '1010F0212'),
                 ('inside',           '1FF0FF212')) AS mat(name,val);

 relationship |  pattern  |     position     |  matrix   | match
--------------+-----------+------------------+-----------+-------
 Equality     | T1FF1FFF1 | non-intersecting | FF1FF0212 | f
 Equality     | T1FF1FFF1 | overlapping      | 1010F0212 | f
 Equality     | T1FF1FFF1 | inside           | 1FF0FF212 | f
 Overlaps     | T*T***T** | non-intersecting | FF1FF0212 | f
 Overlaps     | T*T***T** | overlapping      | 1010F0212 | t
 Overlaps     | T*T***T** | inside           | 1FF0FF212 | f
 Within       | T*F**F*** | non-intersecting | FF1FF0212 | f
 Within       | T*F**F*** | overlapping      | 1010F0212 | f
 Within       | T*F**F*** | inside           | 1FF0FF212 | t
 Disjoint     | FF*FF**** | non-intersecting | FF1FF0212 | t
 Disjoint     | FF*FF**** | overlapping      | 1010F0212 | f
 Disjoint     | FF*FF**** | inside           | 1FF0FF212 | f
```

Siehe auch

Section 5.1, ST_Relate

7.11.1.15 ST_Touches

ST_Touches — Tests if two geometries have at least one point in common, but their interiors do not intersect

Synopsis

boolean **ST_Touches**(geometry A, geometry B);

Beschreibung

Returns `TRUE` if A and B intersect, but their interiors do not intersect. Equivalently, A and B have at least one point in common, and the common points lie in at least one boundary. For Point/Point inputs the relationship is always `FALSE`, since points do not have a boundary.

In mathematical terms: $ST\_Touches(A, B) \Leftrightarrow (Int(A) \bigcap Int(B) \neq \varnothing) \wedge (A \bigcap B \neq \varnothing)$

This relationship holds if the DE-9IM Intersection Matrix for the two geometries matches one of:

- FT*******
- F**T*****
- F***T****

Note
This function automatically includes a bounding box comparison that makes use of any spatial indexes that are available on the geometries. To avoid using an index, use _ST_Touches instead.

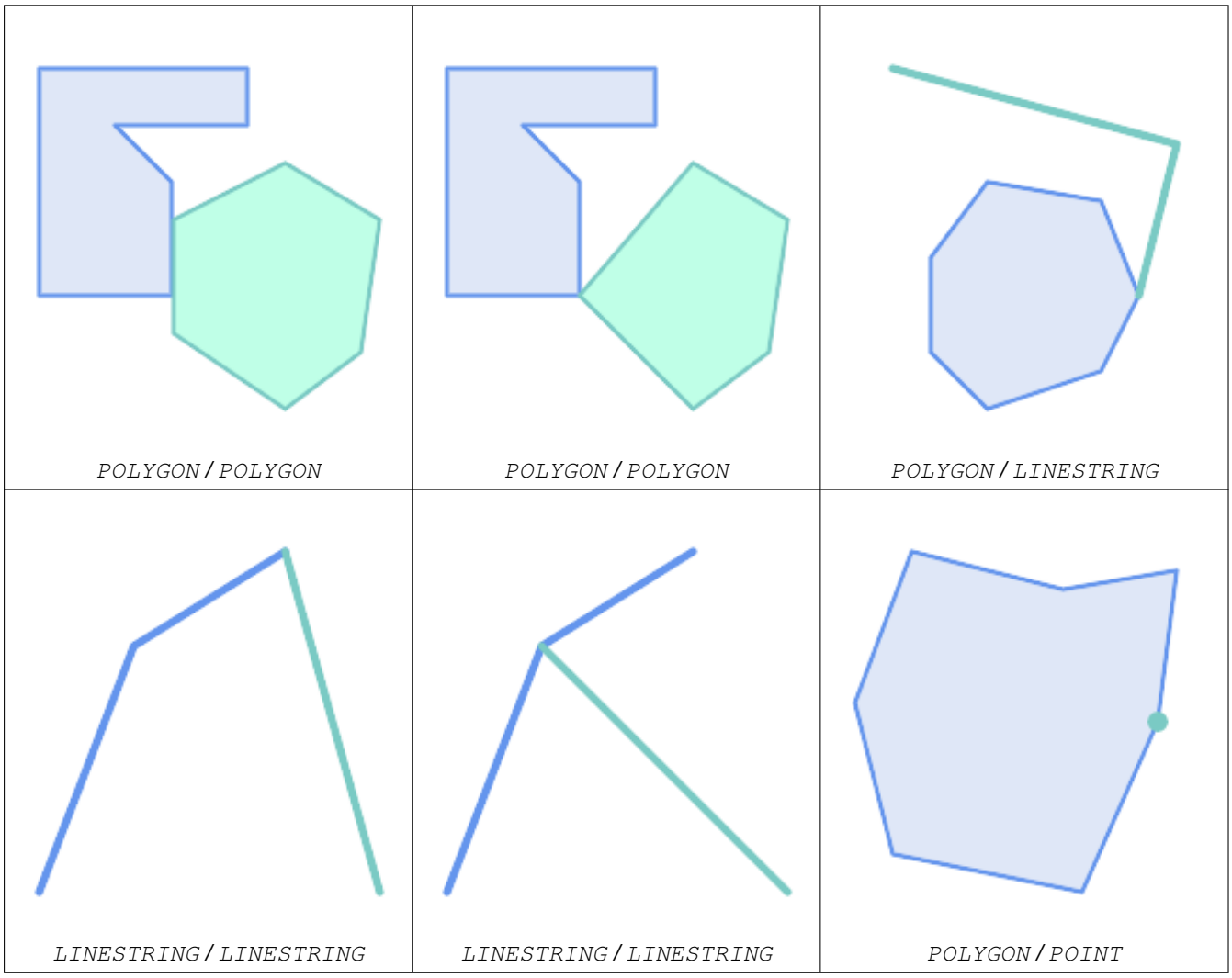
Important
Enhanced: 3.0.0 enabled support for GEOMETRYCOLLECTION

This method implements the OGC Simple Features Implementation Specification for SQL 1.1. s2.1.1.2 // s2.1.13.3

This method implements the SQL/MM specification. SQL-MM 3: 5.1.28

Examples

The ST_Touches predicate returns TRUE in the following examples.




```
SELECT ST_Touches('LINESTRING(0 0, 1 1, 0 2)'::geometry, 'POINT(1 1)'::geometry);
```

```
 st_touches
------------
 f
(1 row)

SELECT ST_Touches('LINESTRING(0 0, 1 1, 0 2)'::geometry, 'POINT(0 2)'::geometry);
 st_touches
------------
 t
(1 row)
```

7.11.1.16 ST_Within

ST_Within — Tests if every point of A lies in B, and their interiors have a point in common

Synopsis

boolean **ST_Within**(geometry A, geometry B);

Beschreibung

Returns TRUE if geometry A is within geometry B. A is within B if and only if all points of A lie inside (i.e. in the interior or boundary of) B (or equivalently, no points of A lie in the exterior of B), and the interiors of A and B have at least one point in common.

For this function to make sense, the source geometries must both be of the same coordinate projection, having the same SRID.

In mathematical terms: $ST\_Within(A, B) \Leftrightarrow (A \bigcap B = A) \wedge (Int(A) \bigcap Int(B) \neq \varnothing)$

The within relation is reflexive: every geometry is within itself. The relation is antisymmetric: if `ST_Within(A,B) = true` and `ST_Within(B,A) = true`, then the two geometries must be topologically equal (`ST_Equals(A,B) = true`).

ST_Within is the converse of ST_Contains. So, `ST_Within(A,B) = ST_Contains(B,A)`.

Note
Because the interiors must have a common point, a subtlety of the definition is that lines and points lying fully in the boundary of polygons or lines are *not* within the geometry. For further details see Subtleties of OGC Covers, Contains, Within. The ST_CoveredBy predicate provides a more inclusive relationship.

Note
This function automatically includes a bounding box comparison that makes use of any spatial indexes that are available on the geometries. To avoid index use, use the function `_ST_Within`.

Wird durch das GEOS Modul ausgeführt

Enhanced: 2.3.0 Enhancement to PIP short-circuit for geometry extended to support MultiPoints with few points. Prior versions only supported point in polygon.

Important
Enhanced: 3.0.0 enabled support for `GEOMETRYCOLLECTION`

Important
Do not use this function with invalid geometries. You will get unexpected results.

NOTE: this is the "allowable" version that returns a boolean, not an integer.

This method implements the OGC Simple Features Implementation Specification for SQL 1.1. s2.1.1.2 // s2.1.13.3 - a.Relate(b, 'T*F**F***')

This method implements the SQL/MM specification. SQL-MM 3: 5.1.30

Examples

```
--a circle within a circle
SELECT ST_Within(smallc,smallc) As smallinsmall,
  ST_Within(smallc, bigc) As smallinbig,
  ST_Within(bigc,smallc) As biginsmall,
  ST_Within(ST_Union(smallc, bigc), bigc) as unioninbig,
  ST_Within(bigc, ST_Union(smallc, bigc)) as biginunion,
  ST_Equals(bigc, ST_Union(smallc, bigc)) as bigisunion
FROM
(
SELECT ST_Buffer(ST_GeomFromText('POINT(50 50)'), 20) As smallc,
  ST_Buffer(ST_GeomFromText('POINT(50 50)'), 40) As bigc) As foo;
--Result
 smallinsmall | smallinbig | biginsmall | unioninbig | biginunion | bigisunion
--------------+------------+------------+------------+------------+------------
 t            | t          | f          | t          | t          | t
(1 row)
```

Siehe auch

ST_Contains, ST_CoveredBy, ST_Equals, ST_IsValid

7.11.2 Distance Relationships

7.11.2.1 ST_3DDWithin

ST_3DDWithin — Tests if two 3D geometries are within a given 3D distance

Synopsis

boolean **ST_3DDWithin**(geometry g1, geometry g2, double precision distance_of_srid);

Beschreibung

Returns true if the 3D distance between two geometry values is no larger than distance `distance_of_srid`. The distance is specified in units defined by the spatial reference system of the geometries. For this function to make sense the source geometries must be in the same coordinate system (have the same SRID).

Note
This function automatically includes a bounding box comparison that makes use of any spatial indexes that are available on the geometries.

This function supports 3d and will not drop the z-index.

This function supports Polyhedral surfaces.

This method implements the SQL/MM specification. SQL-MM ?

Verfügbarkeit: 2.0.0

Examples

```
-- Geometry example - units in meters (SRID: 2163 US National Atlas Equal area) (3D point  ↩
    and line compared 2D point and line)
-- Note: currently no vertical datum support so Z is not transformed and assumed to be same ↩
     units as final.
SELECT ST_3DDWithin(
      ST_Transform(ST_GeomFromEWKT('SRID=4326;POINT(-72.1235 42.3521 4)'),2163),
      ST_Transform(ST_GeomFromEWKT('SRID=4326;LINESTRING(-72.1260 42.45 15, -72.123 42.1546 ↩
          20)'),2163),
      126.8
    ) As within_dist_3d,
ST_DWithin(
      ST_Transform(ST_GeomFromEWKT('SRID=4326;POINT(-72.1235 42.3521 4)'),2163),
      ST_Transform(ST_GeomFromEWKT('SRID=4326;LINESTRING(-72.1260 42.45 15, -72.123 42.1546 ↩
          20)'),2163),
      126.8
    ) As within_dist_2d;

 within_dist_3d | within_dist_2d
----------------+----------------
 f              | t
```

Siehe auch

ST_3DDFullyWithin, ST_DWithin, ST_DFullyWithin, ST_3DDistance, ST_Distance, ST_3DMaxDistance, ST_Transform

7.11.2.2 ST_3DDFullyWithin

ST_3DDFullyWithin — Tests if two 3D geometries are entirely within a given 3D distance

Synopsis

boolean **ST_3DDFullyWithin**(geometry g1, geometry g2, double precision distance);

Beschreibung

Returns true if the 3D geometries are fully within the specified distance of one another. The distance is specified in units defined by the spatial reference system of the geometries. For this function to make sense, the source geometries must both be of the same coordinate projection, having the same SRID.

Note
This function automatically includes a bounding box comparison that makes use of any spatial indexes that are available on the geometries.

Verfügbarkeit: 2.0.0

This function supports 3d and will not drop the z-index.

This function supports Polyhedral surfaces.

Examples

```
-- 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
```

Siehe auch

ST_3DDWithin, ST_DWithin, ST_DFullyWithin, ST_3DMaxDistance

7.11.2.3 ST_DFullyWithin

ST_DFullyWithin — Tests if two geometries are entirely within a given distance

Synopsis

boolean **ST_DFullyWithin**(geometry g1, geometry g2, double precision distance);

Beschreibung

Returns true if the geometries are entirely within the specified distance of one another. The distance is specified in units defined by the spatial reference system of the geometries. For this function to make sense, the source geometries must both be of the same coordinate projection, having the same SRID.

Note
This function automatically includes a bounding box comparison that makes use of any spatial indexes that are available on the geometries.

Availability: 1.5.0

Examples

```
postgis=# 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
     (select ST_GeomFromText('POINT(1 1)') as geom_a,ST_GeomFromText('LINESTRING(1 5, 2 7, 1 ↩
          9, 14 12)') as geom_b) t1;

-----------------
 DFullyWithin10 | DWithin10 | DFullyWithin20 |
---------------+-----------+----------------+
 f              | t         | t              |
```

Siehe auch

ST_MaxDistance, ST_DWithin, ST_3DDWithin, ST_3DDFullyWithin

7.11.2.4 ST_DWithin

ST_DWithin — Tests if two geometries are within a given distance

Synopsis

boolean **ST_DWithin**(geometry g1, geometry g2, double precision distance_of_srid);
boolean **ST_DWithin**(geography gg1, geography gg2, double precision distance_meters, boolean use_spheroid = true);

Beschreibung

Returns true if the geometries are within a given distance

For geometry: The distance is specified in units defined by the spatial reference system of the geometries. For this function to make sense, the source geometries must be in the same coordinate system (have the same SRID).

For geography: units are in meters and distance measurement defaults to `use_spheroid = true`. For faster evaluation use `use_spheroid = false` to measure on the sphere.

Note
Use ST_3DDWithin for 3D geometries.

Note
This function call includes a bounding box comparison that makes use of any indexes that are available on the geometries.

This method implements the OGC Simple Features Implementation Specification for SQL 1.1.

Availability: 1.5.0 support for geography was introduced

Enhanced: 2.1.0 improved speed for geography. See Making Geography faster for details.

Enhanced: 2.1.0 support for curved geometries was introduced.

Prior to 1.3, ST_Expand was commonly used in conjunction with && and ST_Distance to test for distance, and in pre-1.3.4 this function used that logic. From 1.3.4, ST_DWithin uses a faster short-circuit distance function.

Examples

```
-- Find the nearest hospital to each school
-- that is within 3000 units of the school.
--  We do an ST_DWithin search to utilize indexes to limit our search list
--  that the non-indexable ST_Distance needs to process
-- If the units of the spatial reference is meters then units would be meters
SELECT DISTINCT ON (s.gid) s.gid, s.school_name, s.geom, h.hospital_name
  FROM schools s
    LEFT JOIN hospitals h ON ST_DWithin(s.geom, h.geom, 3000)
  ORDER BY s.gid, ST_Distance(s.geom, h.geom);

-- The schools with no close hospitals
-- Find all schools with no hospital within 3000 units
-- away from the school.  Units is in units of spatial ref (e.g. meters, feet, degrees)
SELECT s.gid, s.school_name
  FROM schools s
    LEFT JOIN hospitals h ON ST_DWithin(s.geom, h.geom, 3000)
  WHERE h.gid IS NULL;

-- Find broadcasting towers that receiver with limited range can receive.
-- Data is geometry in Spherical Mercator (SRID=3857), ranges are approximate.

-- Create geometry index that will check proximity limit of user to tower
CREATE INDEX ON broadcasting_towers using gist (geom);

-- Create geometry index that will check proximity limit of tower to user
CREATE INDEX ON broadcasting_towers using gist (ST_Expand(geom, sending_range));

-- Query towers that 4-kilometer receiver in Minsk Hackerspace can get
-- Note: two conditions, because shorter LEAST(b.sending_range, 4000) will not use index.
SELECT b.tower_id, b.geom
  FROM broadcasting_towers b
  WHERE ST_DWithin(b.geom, 'SRID=3857;POINT(3072163.4 7159374.1)', 4000)
    AND ST_DWithin(b.geom, 'SRID=3857;POINT(3072163.4 7159374.1)', b.sending_range);
```

Siehe auch

ST_Distance, ST_3DDWithin

7.11.2.5 ST_PointInsideCircle

ST_PointInsideCircle — Tests if a point geometry is inside a circle defined by a center and radius

Synopsis

boolean **ST_PointInsideCircle**(geometry a_point, float center_x, float center_y, float radius);

Beschreibung

Returns true if the geometry is a point and is inside the circle with center `center_x`,`center_y` and radius `radius`.

Warning
Does not use spatial indexes. Use ST_DWithin instead.

Availability: 1.2

Changed: 2.2.0 In prior versions this was called ST_Point_Inside_Circle

Examples

```
SELECT ST_PointInsideCircle(ST_Point(1,2), 0.5, 2, 3);
 st_pointinsidecircle
----------------------
 t
```

Siehe auch

ST_DWithin

7.12 Measurement Functions

7.12.1 ST_Area

ST_Area — Gibt den geometrischen Schwerpunkt einer Geometrie zurück.

Synopsis

float **ST_Area**(geometry g1);
float **ST_Area**(geography geog, boolean use_spheroid = true);

Beschreibung

Gibt den Flächeninhalt von Polygonen und Mehrfachpolygonen zurück. Gibt den Flächeninhalt der Datentypen "ST_Surface" und "ST_MultiSurface" zurück. Beim geometrischen Datentyp wird die kartesische 2D-Fläche ermittelt und in den Einheiten des SRID ausgegeben. Beim geographischen Datentyp wird die Fläche standardmäßig auf einem Referenzellipsoid ermittelt und in Quadratmeter ausgegeben. Mit ST_Area(geog,false) kann der Flächeninhalt auf einer Kugel ermittelt werden; dies ist zwar schneller aber auch weniger genau.

Erweiterung: Mit 2.0.0 wurde 2D-Unterstützung für polyedrische Oberflächen eingeführt.

Erweiterung: 2.2.0 - die Messung auf dem Referenzellipsoid wird mit der Bibliothek "GeographicLib" durchgeführt. Dadurch wurde die Genauigkeit und die Robustheit erhöht. Um die Vorteile dieser neuen Funktionalität zu nutzen, benötigen Sie Proj >= 4.9.0.

Changed: 3.0.0 - does not depend on SFCGAL anymore.

This method implements the OGC Simple Features Implementation Specification for SQL 1.1.

This method implements the SQL/MM specification. SQL-MM 3: 8.1.2, 9.5.3

This function supports Polyhedral surfaces.

Note
Bei polyedrischen Oberflächen wird nur 2D (nicht 2.5D) unterstützt. Bei 2.5D kann ein Ergebnis ungleich null geliefert werden, wenn die Oberflächen vollständig in der XY-Ebene liegen.

Beispiele

Gibt den Flächeninhalt eines Grundstücks in Massachusetts - in Quadratfuß und konvertiert in Quadratmeter - zurück. Anmerkung: Wegen "Massachusetts State Plane Feet" (EPSG:2249) wird der Flächeninhalt in Quadratfuß ausgegeben

```
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;
&#x250c;&#x2500;&#x2500;&#x2500;&#x2500;&#x2500;&#x2500;&#x2500;&#x2500;&#x2500;&#x252c;&#x2500;&#x2
&#x2502;  sqft   &#x2502;     sqm     &#x2502;
&#x251c;&#x2500;&#x2500;&#x2500;&#x2500;&#x2500;&#x2500;&#x2500;&#x2500;&#x253c;&#x2500;&#x2
&#x2502; 928.625 &#x2502; 86.27208552 &#x2502;
&#x2514;&#x2500;&#x2500;&#x2500;&#x2500;&#x2500;&#x2500;&#x2500;&#x2500;&#x2534;&#x2500;&#x2
```

Gibt den Flächeninhalt in Quadratfuß aus und transformiert nach "Massachusetts state plane meters" (EPSG:26986) um Quadratmeter zu erhalten. Da die Fläche in "Massachusetts State Plane Feet" (EPSG:2249) vorliegt, wird der Flächeninhalt in Quadratfuß ausgegeben. Die transformierte Fläche ist in Quadratmeter, da sie in EPSG:26986 "Massachusetts state plane meters" (EPSG:26986) vorliegt.

```
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;
&#x250c;&#x2500;&#x2500;&#x2500;&#x2500;&#x2500;&#x2500;&#x2500;&#x2500;&#x252c;&#x2500;&#x2
&#x2502;  sqft   &#x2502;       sqm       &#x2502;
&#x251c;&#x2500;&#x2500;&#x2500;&#x2500;&#x2500;&#x2500;&#x2500;&#x2500;&#x253c;&#x2500;&#x2
&#x2502; 928.625 &#x2502; 86.272430607008 &#x2502;
&#x2514;&#x2500;&#x2500;&#x2500;&#x2500;&#x2500;&#x2500;&#x2500;&#x2500;&#x2534;&#x2500;&#x2
```

Gibt den Flächeninhalt in Quadratfuß und in Quadratmeter für den geographischen Datentyp zurück. Beachten Sie bitte, dass wir den geometrischen in den geographischen Datentyp umwandeln (dafür muss die Geometrie in WGS84 lon lat 4326 vorliegen). Beim geographischen Datentyp wird immer in Meter gemessen. Dies ist nur für Vergleichszwecke gedacht, da Ihre Tabelle üblicherweise bereits den geographischen Datentyp aufweisen wird.

```
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;
&#x250c;&#x2500;&#x2500;&#x2500;&#x2500;&#x2500;&#x2500;&#x2500;&#x2500;&#x2500;&#x2500;&#x2500;&#x2500;&#x2
&#x2502;  sqft_spheroid   &#x2502;   sqft_sphere    &#x2502;   sqm_spheroid   &#x2502;
&#x251c;&#x2500;&#x2500;&#x2500;&#x2500;&#x2500;&#x2500;&#x2500;&#x2500;&#x2500;&#x2500;&#x2500;&#x2500;&#x2
&#x2502; 928.684405784452 &#x2502; 927.049336105925 &#x2502; 86.2776044979692 &#x2502;
&#x2514;&#x2500;&#x2500;&#x2500;&#x2500;&#x2500;&#x2500;&#x2500;&#x2500;&#x2500;&#x2500;&#x2500;&#x2500;&#x2
```

If your data is in geography already:

```
select ST_Area(geog) / 0.3048 ^ 2 sqft,
    ST_Area(the_geog) sqm
from somegeogtable;
```

Siehe auch

ST_3DArea, ST_GeomFromEWKT, ST_LengthSpheroid, ST_Perimeter, ST_Transform

7.12.2 ST_Azimuth

ST_Azimuth — Gibt die 2-dimenionale kürzeste Strecke zwischen zwei Geometrien als Linie zurück

Synopsis

float **ST_Azimuth**(geometry origin, geometry target);
float **ST_Azimuth**(geography origin, geography target);

Beschreibung

Returns the azimuth in radians of the target point from the origin point, or NULL if the two points are coincident. The azimuth angle is a positive clockwise angle referenced from the positive Y axis (geometry) or the North meridian (geography): North = 0; Northeast = $\pi/4$; East = $\pi/2$; Southeast = $3\pi/4$; South = π; Southwest $5\pi/4$; West = $3\pi/2$; Northwest = $7\pi/4$.

For the geography type, the azimuth solution is known as the inverse geodesic problem.

The azimuth is a mathematical concept defined as the angle between a reference vector and a point, with angular units in radians. The result value in radians can be converted to degrees using the PostgreSQL function `degrees()`.

Der Azimut ist in Verbindung mit ST_Translate besonders nützlich, weil damit ein Objekt entlang seiner rechtwinkeligen Achse verschoben werden kann. Siehe dazu die Funktion "upgis_lineshift", in dem Abschnitt Plpgsqlfunctions des PostGIS Wiki, für ein Beispiel.

Verfügbarkeit: 1.1.0

Erweiterung: mit 2.0.0 wurde die Unterstützung des geographischen Datentyps eingeführt.

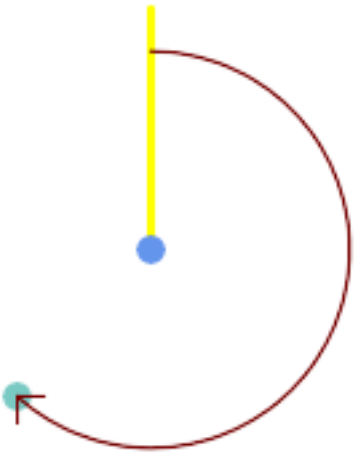
Erweiterung: 2.2.0 die Messungen auf dem Referenzellipsoid werden mit der Bibliothek "GeographicLib" durchgeführt. Dadurch wurde die Genauigkeit und die Robustheit erhöht. Um die Vorteile dieser neuen Funktionalität zu nutzen, benötigen Sie Proj >= 4.9.0.

Beispiele

Geometrischer Datentyp - Azimut in Grad

```
SELECT degrees(ST_Azimuth( ST_Point(25, 45),  ST_Point(75, 100))) AS degA_B,
       degrees(ST_Azimuth( ST_Point(75, 100), ST_Point(25, 45) )) AS degB_A;

      dega_b       |     degb_a
-------------------+-------------------
 42.2736890060937 | 222.273689006094
```

Blue: origin Point(25,45); Green: target Point(75, 100); Yellow: Y axis or North; Red: azimuth angle.



Blue: origin Point(75, 100); Green: target Point(25, 45); Yellow: Y axis or North; Red: azimuth angle.

Siehe auch

ST_Angle, ST_Translate, ST_Project, PostgreSQL Math Functions

7.12.3 ST_Angle

ST_Angle — Gibt den Winkel zwischen 3 Punkten oder zwischen 2 Vektoren (4 Punkte oder 2 Linien) zurück.

Synopsis

float **ST_Angle**(geometry point1, geometry point2, geometry point3, geometry point4);
float **ST_Angle**(geometry line1, geometry line2);

Beschreibung

Gibt die größte 3-dimensionale Distanz zwischen zwei geometrischen Objekten als Linie zurück

Variant 1: computes the angle enclosed by the points P1-P2-P3. If a 4th point provided computes the angle points P1-P2 and P3-P4

Variant 2: computes the angle between two vectors S1-E1 and S2-E2, defined by the start and end points of the input lines

Das Ergebnis wird in Radiant ausgegeben. Wie im folgenden Beispiel gezeigt, kann mit der in PostgreSQL integrierten Funktion "degrees()" von der Einheit Radiant auf die Einheit Grad umgerechnet werden.

ST_Angle(P1,P2,P3) = ST_Angle(P2,P1,P2,P3)

Verfügbarkeit: 2.5.0

Beispiele

Längste Strecke zwischen Polygon und Polygon

```
SELECT degrees( ST_Angle('POINT(0 0)', 'POINT(10 10)', 'POINT(20 0)') );

 degrees
---------
     270
```

Angle between vectors defined by four points

```
SELECT degrees( ST_Angle('POINT (10 10)', 'POINT (0 0)', 'POINT(90 90)', 'POINT (100 80)')  ↩
    );

      degrees
--------------------
 269.9999999999999
```

Angle between vectors defined by the start and end points of lines

```
SELECT degrees( ST_Angle('LINESTRING(0 0, 0.3 0.7, 1 1)', 'LINESTRING(0 0, 0.2 0.5, 1 0)')  ↩
    );

      degrees
--------------
           45
```

Siehe auch

ST_Azimuth

7.12.4 ST_ClosestPoint

ST_ClosestPoint — Returns the 2D point on g1 that is closest to g2. This is the first point of the shortest line from one geometry to the other.

Synopsis

geometry **ST_ClosestPoint**(geometry geom1, geometry geom2);
geography **ST_ClosestPoint**(geography geom1, geography geom2, boolean use_spheroid = true);

Beschreibung

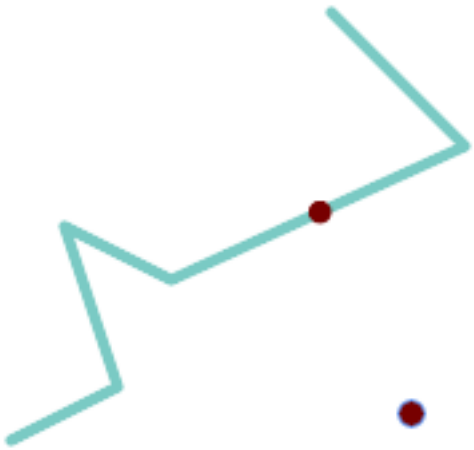
Returns the 2-dimensional point on `geom1` that is closest to `geom2`. This is the first point of the shortest line between the geometries (as computed by ST_ShortestLine).

Note
Falls es sich um eine 3D-Geometrie handelt, sollten Sie ST_3DClosestPoint vorziehen.

Enhanced: 3.4.0 - Support for geography.

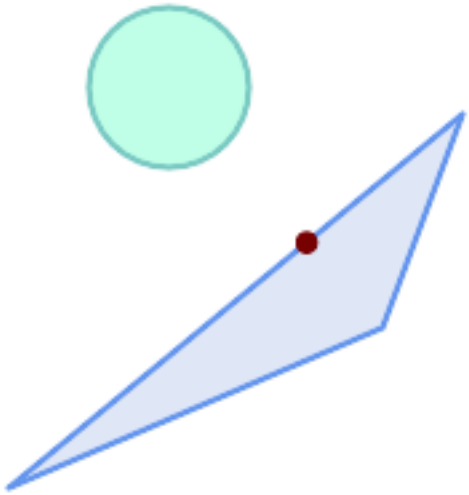
Verfügbarkeit: 1.5.0

Beispiele



The closest point for a Point and a LineString is the point itself. The closest point for a LineString and a Point is a point on the line.

```
SELECT ST_AsText( ST_ClosestPoint(pt,line)) AS cp_pt_line,
       ST_AsText( ST_ClosestPoint(line,pt)) AS cp_line_pt
    FROM (SELECT 'POINT (160 40)'::geometry AS pt,
                 'LINESTRING (10 30, 50 50, 30 110, 70 90, 180 140, 130 190)'::geometry AS  ↩
                    line ) AS t;

   cp_pt_line    |               cp_line_pt
-----------------+-----------------------------------------
 POINT(160 40)   | POINT(125.75342465753425 115.34246575342466)
```



The closest point on polygon A to polygon B

```
SELECT ST_AsText( ST_ClosestPoint(
                'POLYGON ((190 150, 20 10, 160 70, 190 150))',
                ST_Buffer('POINT(80 160)', 30)        )) As ptwkt;
------------------------------------------
 POINT(131.59149149528952 101.89887534906197)
```

Siehe auch

ST_3DClosestPoint, ST_Distance, ST_LongestLine, ST_ShortestLine, ST_MaxDistance

7.12.5 ST_3DClosestPoint

ST_3DClosestPoint — Gibt den 3-dimensionalen Punkt auf g1 zurück, der den kürzesten Abstand zu g2 hat. Dies ist der Anfangspunkt des kürzesten Abstands in 3D.

Synopsis

geometry **ST_3DClosestPoint**(geometry g1, geometry g2);

Beschreibung

Gibt den 3-dimensionalen Punkt auf g1 zurück, der den kürzesten Abstand zu g2 hat. Dies ist der Anfangspunkt des kürzesten Abstands in 3D. Die Länge des kürzesten Abstands in 3D ergibt sich aus der 3D-Distanz.

This function supports 3d and will not drop the z-index.

This function supports Polyhedral surfaces.

Verfügbarkeit: 2.0.0

Änderung: 2.2.0 - Wenn 2 geometrische Objekte in 2D übergegeben werden, wird ein 2D-Punkt zurückgegeben (anstelle wie früher 0 für ein fehlendes Z). Im Falle von 2D und 3D, wird für fehlende Z nicht länger 0 angenommen.

Beispiele

Linienstück und Punkt -- Punkt mit kürzestem Abstand; in 3D und in 2D

```
SELECT ST_AsEWKT(ST_3DClosestPoint(line,pt)) AS cp3d_line_pt,
                ST_AsEWKT(ST_ClosestPoint(line,pt)) As cp2d_line_pt
        FROM (SELECT 'POINT(100 100 30)'::geometry As pt,
                        'LINESTRING (20 80 20, 98 190 1, 110 180 3, 50 75 1000)':: ↩
    geometry As line
                ) As foo;


 cp3d_line_pt                                                  |                ↩
    cp2d_line_pt
---------------------------------------------------------------+-----------------------------------
 POINT(54.6993798867619 128.935022917228 11.5475869506606) | POINT(73.0769230769231  ↩
    115.384615384615)
```

Linienstück und Mehrfachpunkt - Punkt mit kürzestem Abstand; in 3D und in 2D

```
SELECT ST_AsEWKT(ST_3DClosestPoint(line,pt)) AS cp3d_line_pt,
                ST_AsEWKT(ST_ClosestPoint(line,pt)) As cp2d_line_pt
        FROM (SELECT 'MULTIPOINT(100 100 30, 50 74 1000)'::geometry As pt,
                        'LINESTRING (20 80 20, 98 190 1, 110 180 3, 50 75 900)':: ↩
    geometry As line
                ) As foo;


                       cp3d_line_pt                          | cp2d_line_pt
-------------------------------------------------------------+--------------
 POINT(54.6993798867619 128.935022917228 11.5475869506606) | POINT(50 75)
```

Mehrfachlinie und Polygon - Punkt mit kürzestem Abstand; in 3D und in 2D

```
SELECT ST_AsEWKT(ST_3DClosestPoint(poly, mline)) As cp3d,
    ST_AsEWKT(ST_ClosestPoint(poly, mline)) As cp2d
        FROM (SELECT  ST_GeomFromEWKT('POLYGON((175 150 5, 20 40 5, 35 45 5, 50 60 5,  ↩
    100 100 5, 175 150 5))') As poly,
                ST_GeomFromEWKT('MULTILINESTRING((175 155 2, 20 40 20, 50 60 -2, 125  ↩
    100 1, 175 155 1),
                (1 10 2, 5 20 1))') As mline ) As foo;
                   cp3d                    |     cp2d
-------------------------------------------+--------------
 POINT(39.993580415989 54.1889925532825 5) | POINT(20 40)
```

Siehe auch

ST_AsEWKT, ST_ClosestPoint, ST_3DDistance, ST_3DShortestLine

7.12.6 ST_Distance

ST_Distance — Gibt die größte 3-dimensionale Distanz zwischen zwei geometrischen Objekten als Linie zurück

Synopsis

float **ST_Distance**(geometry g1, geometry g2);
float **ST_Distance**(geography geog1, geography geog2, boolean use_spheroid = true);

Beschreibung

Für den geometrischen Datentyp. Es wird der geringste 3-dimensionale kartesische Abstand zwischen zwei geometrischen Objekten in projizierten Einheiten (Einheiten des Koordinatenreferenzsystem) zurückgegeben.

Beim geometrischen Datentyp geometry wird die geringste kartesische Distanz in 2D zwischen zwei geometrischen Objekten - in projizierten Einheiten (Einheiten des Koordinatenreferenzsystem) - zurückgegeben. Beim geographischen Datentyp geography wird standardmäßig die geringste geodätische Distanz zwischen zwei geographischen Objekten in Meter zurückgegeben. Wenn use_speroid FALSE ist, erfolgt eine schnellere Berechnung auf einer Kugel anstatt auf dem Referenzellipsoid.

This method implements the OGC Simple Features Implementation Specification for SQL 1.1.

This method implements the SQL/MM specification. SQL-MM 3: 5.1.23

This method supports Circular Strings and Curves.

Verfügbarkeit: 1.5.0 die Unterstützung des geograpischen Datentyps wurde eingeführt. Geschwindigkeitsverbesserungen bei einer umfangreichen Geometrie und bei einer Geometrie mit vielen Knoten

Enhanced: 2.1.0 Geschwindigkeitsverbesserung beim geographischen Datentyp. Siehe Making Geography faster für Details.

Erweiterung: 2.1.0 - Unterstützung für Kurven beim geometrischen Datentyp eingeführt.

Erweiterung: 2.2.0 - die Messung auf dem Referenzellipsoid wird mit der Bibliothek "GeographicLib" durchgeführt. Dadurch wurde die Genauigkeit und die Robustheit erhöht. Um die Vorteile dieser neuen Funktionalität zu nutzen, benötigen Sie Proj >= 4.9.0.

Changed: 3.0.0 - does not depend on SFCGAL anymore.

Beispiele mit dem geometrischen Datentyp

Geometry example - units in planar degrees 4326 is WGS 84 long lat, units are degrees.

```
SELECT ST_Distance(
    'SRID=4326;POINT(-72.1235 42.3521)'::geometry,
    'SRID=4326;LINESTRING(-72.1260 42.45, -72.123 42.1546)'::geometry );
-----------------
0.00150567726382282
```

Geometry example - units in meters (SRID: 3857, proportional to pixels on popular web maps). Although the value is off, nearby ones can be compared correctly, which makes it a good choice for algorithms like KNN or KMeans.

```
SELECT ST_Distance(
    ST_Transform('SRID=4326;POINT(-72.1235 42.3521)'::geometry, 3857),
    ST_Transform('SRID=4326;LINESTRING(-72.1260 42.45, -72.123 42.1546)'::geometry, 3857) ) ↩
        ;
-----------------
167.441410065196
```

Geometry example - units in meters (SRID: 3857 as above, but corrected by cos(lat) to account for distortion)

```
SELECT ST_Distance(
    ST_Transform('SRID=4326;POINT(-72.1235 42.3521)'::geometry, 3857),
    ST_Transform('SRID=4326;LINESTRING(-72.1260 42.45, -72.123 42.1546)'::geometry, 3857)
                ) * cosd(42.3521);
-----------------
123.742351254151
```

Geometry example - units in meters (SRID: 26986 Massachusetts state plane meters) (most accurate for Massachusetts)

```
SELECT ST_Distance(
    ST_Transform('SRID=4326;POINT(-72.1235 42.3521)'::geometry, 26986),
    ST_Transform('SRID=4326;LINESTRING(-72.1260 42.45, -72.123 42.1546)'::geometry, 26986)  ↩
        );
-----------------
123.797937878454
```

Geometry example - units in meters (SRID: 2163 US National Atlas Equal area) (least accurate)

```
SELECT ST_Distance(
    ST_Transform('SRID=4326;POINT(-72.1235 42.3521)'::geometry, 2163),
    ST_Transform('SRID=4326;LINESTRING(-72.1260 42.45, -72.123 42.1546)'::geometry, 2163) ) ↩
        ;
------------------
126.664256056812
```

Beispiele für den geographischen Datentyp

Same as geometry example but note units in meters - use sphere for slightly faster and less accurate computation.

```
SELECT ST_Distance(gg1, gg2) As spheroid_dist, ST_Distance(gg1, gg2, false) As sphere_dist
FROM (SELECT
    'SRID=4326;POINT(-72.1235 42.3521)'::geography as gg1,
    'SRID=4326;LINESTRING(-72.1260 42.45, -72.123 42.1546)'::geography as gg2
        ) As foo  ;

  spheroid_dist    |   sphere_dist
-------------------+------------------
 123.802076746848 | 123.475736916397
```

Siehe auch

ST_3DDistance, ST_DWithin, ST_DistanceSphere, ST_DistanceSpheroid, ST_MaxDistance, ST_HausdorffDistance, ST_FrechetDistan
ST_Transform

7.12.7 ST_3DDistance

ST_3DDistance — Für den geometrischen Datentyp. Es wird der geringste 3-dimensionale kartesische Abstand (basierend auf dem Koordinatenreferenzsystem) zwischen zwei geometrischen Objekten in projizierten Einheiten zurückgegeben.

Synopsis

float **ST_3DDistance**(geometry g1, geometry g2);

Beschreibung

Für den geometrischen Datentyp. Es wird der geringste 3-dimensionale kartesische Abstand zwischen zwei geometrischen Objekten in projizierten Einheiten (Einheiten des Koordinatenreferenzsystem) zurückgegeben.

This function supports 3d and will not drop the z-index.

This function supports Polyhedral surfaces.

This method implements the SQL/MM specification. SQL-MM ISO/IEC 13249-3

Verfügbarkeit: 2.0.0

Änderung: 2.2.0 - Im Falle von 2D und 3D wird für ein fehlendes Z nicht mehr 0 angenommen.

Changed: 3.0.0 - SFCGAL version removed

Beispiele

```
-- Geometry example - units in meters (SRID: 2163 US National Atlas Equal area) (3D point  ↩
    and line compared 2D point and line)
-- Note: currently no vertical datum support so Z is not transformed and assumed to be same ↩
     units as final.
SELECT ST_3DDistance(
                        ST_Transform('SRID=4326;POINT(-72.1235 42.3521 4)'::geometry,2163),
                        ST_Transform('SRID=4326;LINESTRING(-72.1260 42.45 15, -72.123  ↩
                            42.1546 20)'::geometry,2163)
                ) As dist_3d,
                ST_Distance(
                        ST_Transform('SRID=4326;POINT(-72.1235 42.3521)'::geometry,2163),
                        ST_Transform('SRID=4326;LINESTRING(-72.1260 42.45, -72.123 42.1546) ↩
                            '::geometry,2163)
                ) As dist_2d;

     dist_3d      |     dist_2d
------------------+-----------------
 127.295059324629 | 126.66425605671
```

```
-- Multilinestring and polygon both 3d and 2d distance
-- Same example as 3D closest point example
SELECT ST_3DDistance(poly, mline) As dist3d,
    ST_Distance(poly, mline) As dist2d
        FROM (SELECT  'POLYGON((175 150 5, 20 40 5, 35 45 5, 50 60 5, 100 100 5, 175 150 5) ↩
            )'::geometry as poly,
                'MULTILINESTRING((175 155 2, 20 40 20, 50 60 -2, 125 100 1, 175 155 1), (1  ↩
                    10 2, 5 20 1))'::geometry as mline) as foo;
      dist3d       | dist2d
-------------------+--------
 0.716635696066337 |      0
```

Siehe auch

ST_Distance, ST_3DClosestPoint, ST_3DDWithin, ST_3DMaxDistance, ST_3DShortestLine, ST_Transform

7.12.8 ST_DistanceSphere

ST_DistanceSphere — Gibt die kürzeste Distanz zwischen zwei geometrischen Objekten zurück, die über Länge, Breite und ein bestimmtes Referenzellipsoid gegeben sind. Vorgängerversionen von PostGIS 1.5 unterstützten nur Punkte.

Synopsis

float **ST_DistanceSphere**(geometry geomlonlatA, geometry geomlonlatB, float8 radius=6371008);

Beschreibung

Gibt die kürzeste Distanz zwischen zwei Punkten zurück, die über Länge und Breite gegeben sind. Verwendet die Kugelform für die Erde und den Radius des Referenzellipsoids, der durch die SRID festgelegt ist. Ist schneller als ST_DistanceSpheroid, aber weniger genau. Vorgängerversionen von PostGIS 1.5 unterstützten nur Punkte.

Verfügbarkeit: 1.5 die Unterstützung für weitere geometrische Datentypen neben Punkten eingeführt. Bei Vorgängerversionen wurden nur Punkte unterstützt.

Änderung: 2.2.0 In Vorgängerversionen als ST_Distance_Sphere bezeichet.

Beispiele

```
SELECT round(CAST(ST_DistanceSphere(ST_Centroid(geom), ST_GeomFromText('POINT(-118 38) ↩
    ',4326)) As numeric),2) As dist_meters,
round(CAST(ST_Distance(ST_Transform(ST_Centroid(geom),32611),
                ST_Transform(ST_GeomFromText('POINT(-118 38)', 4326),32611)) As numeric),2) ↩
                    As dist_utm11_meters,
round(CAST(ST_Distance(ST_Centroid(geom), ST_GeomFromText('POINT(-118 38)', 4326)) As  ↩
    numeric),5) As dist_degrees,
round(CAST(ST_Distance(ST_Transform(geom,32611),
                ST_Transform(ST_GeomFromText('POINT(-118 38)', 4326),32611)) As numeric),2) ↩
                    As min_dist_line_point_meters
FROM
        (SELECT ST_GeomFromText('LINESTRING(-118.584 38.374,-118.583 38.5)', 4326) As geom) ↩
            as foo;
         dist_meters | dist_utm11_meters | dist_degrees | min_dist_line_point_meters
        -------------+-------------------+--------------+----------------------------
                70424.47 |          70438.00 |      0.72900 |                   65871.18
```

Siehe auch

ST_Distance, ST_DistanceSpheroid

7.12.9 ST_DistanceSpheroid

ST_DistanceSpheroid — Gibt die kürzeste Distanz zwischen zwei geometrischen Objekten zurück, die über Länge, Breite und ein bestimmtes Referenzellipsoid gegeben sind. Vorgängerversionen von PostGIS 1.5 unterstützten nur Punkte.

Synopsis

float **ST_DistanceSpheroid**(geometry geomlonlatA, geometry geomlonlatB, spheroid measurement_spheroid=WGS84);

Beschreibung

Gibt die kürzeste Distanz zwischen zwei geometrischen Objekten in Meter zurück, die über Länge, Breite und ein bestimmtes Referenzellipsoid gegeben sind. Siehe die Erklärung zu Referenzellipsoiden unter ST_LengthSpheroid. Vorgängerversionen von PostGIS 1.5 unterstützten nur Punkte.

Note
Aktuell schaut diese Funktion nicht auf die SRID der Geometrie, sondern nimmt an, dass die Geometrie in den Koordinaten des gegebenen Referenzellipsoids vorliegt. Vorgängerversionen von PostGIS 1.5 unterstützten nur Punkte.

Verfügbarkeit: 1.5 die Unterstützung für weitere geometrische Datentypen neben Punkten eingeführt. Bei Vorgängerversionen wurden nur Punkte unterstützt.

Änderung: 2.2.0 In Vorgängerversionen als ST_Distance_Spheroid bezeichet.

Beispiele

```
SELECT round(CAST(
                ST_DistanceSpheroid(ST_Centroid(geom), ST_GeomFromText('POINT(-118 38) ↩
                    ',4326), 'SPHEROID["WGS 84",6378137,298.257223563]')
                        As numeric),2) As dist_meters_spheroid,
                round(CAST(ST_DistanceSphere(ST_Centroid(geom), ST_GeomFromText('POINT(-118 ↩
                     38)',4326)) As numeric),2) As dist_meters_sphere,
round(CAST(ST_Distance(ST_Transform(ST_Centroid(geom),32611),
                ST_Transform(ST_GeomFromText('POINT(-118 38)', 4326),32611)) As numeric),2) ↩
                     As dist_utm11_meters
FROM
        (SELECT ST_GeomFromText('LINESTRING(-118.584 38.374,-118.583 38.5)', 4326) As geom) ↩
            as foo;
 dist_meters_spheroid | dist_meters_sphere | dist_utm11_meters
----------------------+--------------------+-------------------
                       70454.92 |           70424.47 |          70438.00
```

Siehe auch

ST_Distance, ST_DistanceSphere

7.12.10 ST_FrechetDistance

ST_FrechetDistance — Gibt den kürzesten 3-dimensionalen Abstand zwischen zwei geometrischen Objekten als Linie zurück

Synopsis

float **ST_FrechetDistance**(geometry g1, geometry g2, float densifyFrac = -1);

Beschreibung

Implementiert einen Algorithmus zur Berechnung der Fréchet-Metrik, der für beide geometrischen Objekte auf diskrete Punkte beschränkt ist und auf Computing Discrete Fréchet Distance beruht. Die Fréchet-Metrik ist ein Maß für die Ähnlichkeit von Kurven, welches die Position und die Reihenfolge der Kurvenstützpunkte mit einbezieht. Daher ist sie oft besser geeignet als die Hausdorff-Metrik.

Wenn der optionale Parameter "densifyFrac" vorgegeben wird, dann führt diese Funktion eine Verdichtung der Linienstücke durch, bevor die diskrete Fréchet-Metrik berechnet wird. Der Parameter "densifyFrac" bestimmt um welchen Anteil die Linienstücke verdichtet werden. Jedes Linienstück wird in gleichlange Teilsegmente zerlegt, deren Anteil an der Gesamtlänge am nächsten an den vorgegebenen Anteil herankommt.

Units are in the units of the spatial reference system of the geometries.

Note
Bei der aktuellen Implementierung können die diskreten Punkte nur Knoten sein. Dies könnte erweitert werden, um eine beliebige Punktdichte zu ermöglichen.

Note
Umso kleiner wir densifyFrac wählen, umso genauer wird die Fréchet-Metrik. Aber die Rechenzeit und der Speicherplatzbedarf steigen quadratisch mit der Anzahl der Teilabschnitte.

Wird durch das GEOS Modul ausgeführt

Verfügbarkeit: 2.4.0 - benötigt GEOS >= 3.7.0

Beispiele

```
postgres=# SELECT st_frechetdistance('LINESTRING (0 0, 100 0)'::geometry, 'LINESTRING (0 0, ↩
     50 50, 100 0)'::geometry);
 st_frechetdistance
--------------------
   70.7106781186548
(1 row)
```

```
SELECT st_frechetdistance('LINESTRING (0 0, 100 0)'::geometry, 'LINESTRING (0 0, 50 50, 100 ↩
     0)'::geometry, 0.5);
 st_frechetdistance
--------------------
                 50
(1 row)
```

Siehe auch

ST_HausdorffDistance

7.12.11 ST_HausdorffDistance

ST_HausdorffDistance — Gibt den kürzesten 3-dimensionalen Abstand zwischen zwei geometrischen Objekten als Linie zurück

Synopsis

float **ST_HausdorffDistance**(geometry g1, geometry g2);
float **ST_HausdorffDistance**(geometry g1, geometry g2, float densifyFrac);

Beschreibung

Returns the Hausdorff distance between two geometries. The Hausdorff distance is a measure of how similar or dissimilar 2 geometries are.

The function actually computes the "Discrete Hausdorff Distance". This is the Hausdorff distance computed at discrete points on the geometries. The *densifyFrac* parameter can be specified, to provide a more accurate answer by densifying segments before computing the discrete Hausdorff distance. Each segment is split into a number of equal-length subsegments whose fraction of the segment length is closest to the given fraction.

Units are in the units of the spatial reference system of the geometries.

Note
This algorithm is NOT equivalent to the standard Hausdorff distance. However, it computes an approximation that is correct for a large subset of useful cases. One important case is Linestrings that are roughly parallel to each other, and roughly equal in length. This is a useful metric for line matching.

Verfügbarkeit: 1.5.0

Beispiele

Hausdorff distance (red) and distance (yellow) between two lines

```
SELECT ST_HausdorffDistance(geomA, geomB),
       ST_Distance(geomA, geomB)
    FROM (SELECT 'LINESTRING (20 70, 70 60, 110 70, 170 70)'::geometry AS geomA,
                 'LINESTRING (20 90, 130 90, 60 100, 190 100)'::geometry AS geomB) AS t;
 st_hausdorffdistance | st_distance
----------------------+-------------
    37.26206567625497 |          20
```

Example: Hausdorff distance with densification.

```
SELECT ST_HausdorffDistance(
            'LINESTRING (130 0, 0 0, 0 150)'::geometry,
            'LINESTRING (10 10, 10 150, 130 10)'::geometry,
            0.5);
 ----------------------
          70
```

Example: For each building, find the parcel that best represents it. First we require that the parcel intersect with the building geometry. `DISTINCT ON` guarantees we get each building listed only once. `ORDER BY ..` `ST_HausdorffDistance` selects the parcel that is most similar to the building.

```
SELECT DISTINCT ON (buildings.gid) buildings.gid, parcels.parcel_id
   FROM buildings
       INNER JOIN parcels
       ON ST_Intersects(buildings.geom, parcels.geom)
   ORDER BY buildings.gid, ST_HausdorffDistance(buildings.geom, parcels.geom);
```

Siehe auch

ST_FrechetDistance

7.12.12 ST_Length

ST_Length — Gibt den geometrischen Schwerpunkt einer Geometrie zurück.

Synopsis

float **ST_Length**(geometry a_2dlinestring);
float **ST_Length**(geography geog, boolean use_spheroid = true);

Beschreibung

Beim geometrischen Datentyp wird die kartesische 2D Länge der Geometrie zurückgegeben. Dabei muss es sich um einen LineString, einen MultiLineString, eine ST_Curve oder eine ST_MultiCurve handeln. Bei einer flächigen Geometrie wird 0 zurückgegeben. Für eine flächige Geometrie können Sie ST_Perimeter verwenden. Bei den geometrischen Datentypen sind die Einheiten für die Längenmessungen durch das Koordinatenreferenzsystem festgelegt.

For geography types: computation is performed using the inverse geodesic calculation. Units of length are in meters. If PostGIS is compiled with PROJ version 4.8.0 or later, the spheroid is specified by the SRID, otherwise it is exclusive to WGS84. If `use_spheroid = false`, then the calculation is based on a sphere instead of a spheroid.

Zur Zeit ein Synonym für ST_Length2D; dies kann sich allerdings ändern, wenn höhere Dimensionen unterstützt werden.

Warning
Änderung: 2.0.0 Wesentliche Änderung -- In früheren Versionen ergab die Anwendung auf ein MULTI/POLYGON vom geographischen Datentyp den Umfang des POLYGON/MULTIPOLYGON. In 2.0.0 wurde dies geändert und es wird jetzt 0 zurückgegeben, damit es mit der Verhaltensweise beim geometrischen Datentyp übereinstimmt. Verwenden Sie bitte ST_Perimeter, wenn Sie den Umfang eines Polygons wissen wollen

Note
Beim geographischer Datentyp werden die Messungen standardmäßig am Referenzellipsoid durchgeführt. Für die schnellere, aber ungenauere Berechnung auf einer Kugel können Sie ST_Length(gg,false) verwenden;

This method implements the OGC Simple Features Implementation Specification for SQL 1.1. s2.1.5.1

This method implements the SQL/MM specification. SQL-MM 3: 7.1.2, 9.3.4

Verfügbarkeit: 1.5.0 Unterstützung von geograpischen Koordinaten.

Beispiele mit dem geometrischen Datentyp

Gibt die Länge eines Linienstücks zurück. Beachten Sie, dass die Einheit Fuß ist, da EPSG:2249 "Massachusetts State Plane Feet" ist

```
SELECT ST_Length(ST_GeomFromText('LINESTRING(743238 2967416,743238 2967450,743265 2967450,
743265.625 2967416,743238 2967416)',2249));

st_length
---------
 122.630744000095


--Transforming WGS 84 LineString to Massachusetts state plane meters
SELECT ST_Length(
        ST_Transform(
                ST_GeomFromEWKT('SRID=4326;LINESTRING(-72.1260 42.45, -72.1240 42.45666,  ↩
                    -72.123 42.1546)'),
                26986
        )
```

```
);

st_length
---------
34309.4563576191
```

Beispiele für den geographischen Datentyp

Gibt die Länge einer Linie zurück, die in geographischen WGS 84 Koordinaten vorliegt.

```
-- the default calculation uses a spheroid
SELECT ST_Length(the_geog) As length_spheroid,  ST_Length(the_geog,false) As length_sphere
FROM (SELECT ST_GeographyFromText(
'SRID=4326;LINESTRING(-72.1260 42.45, -72.1240 42.45666, -72.123 42.1546)') As the_geog)
 As foo;

 length_spheroid  |  length_sphere
------------------+------------------
 34310.5703627288 | 34346.2060960742
```

Siehe auch

ST_GeographyFromText, ST_GeomFromEWKT, ST_LengthSpheroid, ST_Perimeter, ST_Transform

7.12.13 ST_Length2D

ST_Length2D — Gibt die 2-dimensionale Länge einer Linie oder einer Mehrfachlinie zurück. Dies ist ein Alias für `ST_Length`

Synopsis

float **ST_Length2D**(geometry a_2dlinestring);

Beschreibung

Gibt die 2-dimensionale Länge einer Linie oder einer Mehrfachlinie zurück. Dies ist ein Alias für `ST_Length`

Siehe auch

ST_Length, ST_3DLength

7.12.14 ST_3DLength

ST_3DLength — Gibt den geometrischen Schwerpunkt einer Geometrie zurück.

Synopsis

float **ST_3DLength**(geometry a_3dlinestring);

Beschreibung

Gibt die 2- oder 3-dimensionale Länge einer Linie oder einer Mehrfachlinie zurück. Bei einer 2-D Linie wird die Länge nur in 2D zurückgegeben (gleich wie ST_Length und ST_Length2D)

This function supports 3d and will not drop the z-index.

This method implements the SQL/MM specification. SQL-MM IEC 13249-3: 7.1, 10.3

Änderung: 2.0.0 In Vorgängerversionen als ST_Length3D bezeichet.

Beispiele

Gibt die Länge eines 3D-Kabels zurück. Beachten Sie, dass die Einheit Fuß ist, da EPSG:2249 "Massachusetts State Plane Feet" ist

```
SELECT ST_3DLength(ST_GeomFromText('LINESTRING(743238 2967416 1,743238 2967450 1,743265  ↩
    2967450 3,
743265.625 2967416 3,743238 2967416 3)',2249));
ST_3DLength
-----------
122.704716741457
```

Siehe auch

ST_Length, ST_Length2D

7.12.15 ST_LengthSpheroid

ST_LengthSpheroid — Gibt den geometrischen Schwerpunkt einer Geometrie zurück.

Synopsis

float **ST_LengthSpheroid**(geometry a_geometry, spheroid a_spheroid);

Beschreibung

Berechnet die/den Länge/Umfang einer Geometrie auf einem Ellipsoid. Dies ist nützlich wenn die Koordinaten der Geometrie in Länge und Breite vorliegen, und die Länge der Geometrie ohne benötigt wird, ohne dass umprojiziert werden muss. Das Ellipsoid ist ein eigener Datentyp und kann wie folgt erstellt werden:

```
SPHEROID[<NAME>,<SEMI-MAJOR AXIS>,<INVERSE FLATTENING>]
```

Geometrie Beispiel

```
SPHEROID["GRS_1980",6378137,298.257222101]
```

Verfügbarkeit: 1.2.2

Änderung: 2.2.0 In Vorgängerversionen als ST_Length_Spheroid bezeichet.und mit dem Alias "ST_3DLength_Spheroid" versehen

This function supports 3d and will not drop the z-index.

Beispiele

```
SELECT ST_LengthSpheroid( geometry_column,
                          'SPHEROID["GRS_1980",6378137,298.257222101]' )
                          FROM geometry_table;

SELECT ST_LengthSpheroid( geom, sph_m ) As tot_len,
ST_LengthSpheroid(ST_GeometryN(geom,1), sph_m) As len_line1,
ST_LengthSpheroid(ST_GeometryN(geom,2), sph_m) As len_line2
                          FROM (SELECT ST_GeomFromText('MULTILINESTRING((-118.584  ↩
                              38.374,-118.583 38.5),
        (-71.05957 42.3589 , -71.061 43))') As geom,
CAST('SPHEROID["GRS_1980",6378137,298.257222101]' As spheroid) As sph_m)  as foo;
        tot_len      |    len_line1     |    len_line2
-----------------+-----------------+-----------------
 85204.5207562955 | 13986.8725229309 | 71217.6482333646

 --3D
SELECT ST_LengthSpheroid( geom, sph_m ) As tot_len,
ST_LengthSpheroid(ST_GeometryN(geom,1), sph_m) As len_line1,
ST_LengthSpheroid(ST_GeometryN(geom,2), sph_m) As len_line2
                          FROM (SELECT ST_GeomFromEWKT('MULTILINESTRING((-118.584 38.374  ↩
                              20,-118.583 38.5 30),
        (-71.05957 42.3589 75, -71.061 43 90))') As geom,
CAST('SPHEROID["GRS_1980",6378137,298.257222101]' As spheroid) As sph_m)  as foo;

         tot_len      |    len_line1    |    len_line2
------------------+-----------------+------------------
 85204.5259107402 | 13986.876097711 | 71217.6498130292
```

Siehe auch

ST_GeometryN, ST_Length

7.12.16 ST_LongestLine

ST_LongestLine — Gibt die größte 3-dimensionale Distanz zwischen zwei geometrischen Objekten als Linie zurück

Synopsis

geometry **ST_LongestLine**(geometry g1, geometry g2);

Beschreibung

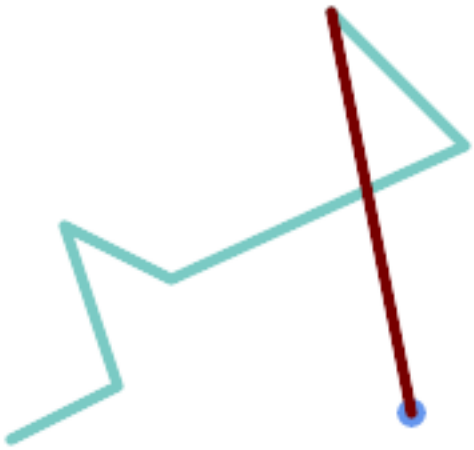
Returns the 2-dimensional longest line between the points of two geometries. The line returned starts on `g1` and ends on `g2`.

The longest line always occurs between two vertices. The function returns the first longest line if more than one is found. The length of the line is equal to the distance returned by ST_MaxDistance.

If g1 and g2 are the same geometry, returns the line between the two vertices farthest apart in the geometry. The endpoints of the line lie on the circle computed by ST_MinimumBoundingCircle.

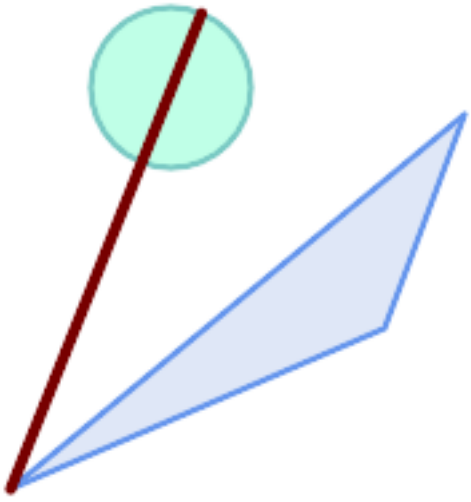
Verfügbarkeit: 1.5.0

Beispiele



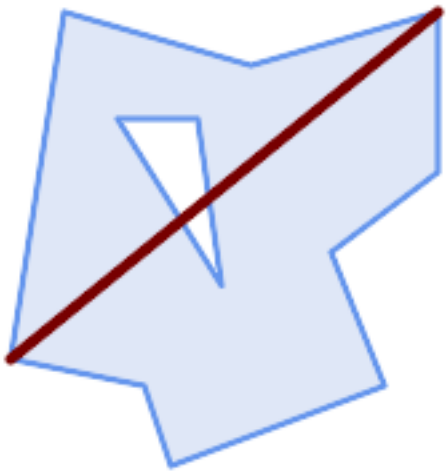
Längste Strecke zwischen Punkt und Linie

```
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)
```



Längste Strecke zwischen Polygon und Polygon

```
SELECT ST_AsText( ST_LongestLine(
        'POLYGON ((190 150, 20 10, 160 70, 190 150))',
        ST_Buffer('POINT(80 160)', 30)
            ) ) AS llinewkt;
-----------------
LINESTRING(20 10,105.3073372946034 186.95518130045156)
```



Longest line across a single geometry. The length of the line is equal to the Maximum Distance. The endpoints of the line lie on the Minimum Bounding Circle.

```
SELECT ST_AsText( ST_LongestLine( geom, geom)) AS llinewkt,
                  ST_MaxDistance( geom, geom) AS max_dist,
                  ST_Length( ST_LongestLine(geom, geom)) AS lenll
FROM (SELECT 'POLYGON ((40 180, 110 160, 180 180, 180 120, 140 90, 160 40, 80 10, 70 40, 20 ↩
     50, 40 180),
              (60 140, 99 77.5, 90 140, 60 140))'::geometry AS geom) AS t;

         llinewkt          |      max_dist      |       lenll
---------------------------+--------------------+--------------------
 LINESTRING(20 50,180 180) | 206.15528128088303 | 206.15528128088303
```

Siehe auch

ST_MaxDistance, ST_ShortestLine, ST_3DLongestLine, ST_MinimumBoundingCircle

7.12.17 ST_3DLongestLine

ST_3DLongestLine — Gibt die größte 3-dimensionale Distanz zwischen zwei geometrischen Objekten als Linie zurück

Synopsis

geometry **ST_3DLongestLine**(geometry g1, geometry g2);

Beschreibung

Gibt den größten 3-dimensionalen Abstand zwischen zwei geometrischen Objekten als Linie zurück. Wenn es mehr als einen größten Abstand gibt, dann wird nur die erste zurückgegeben. Die zurückgegebene Linie fängt immer mit "g1" an und endet mit "g2". Die Länge der 3D-Linie die von dieser Funktion zurückgegeben wird ist immer ident mit der von ST_3DMaxDistance für "g1" und "g2" zurückgegebenen Distanz.

Verfügbarkeit: 2.0.0

Änderung: 2.2.0 - Wenn 2 geometrische Objekte in 2D übergegeben werden, wird ein 2D-Punkt zurückgegeben (anstelle wie früher 0 für ein fehlendes Z). Im Falle von 2D und 3D, wird für fehlende Z nicht länger 0 angenommen.

This function supports 3d and will not drop the z-index.

This function supports Polyhedral surfaces.

Beispiele

Linienstück und Punkt -- größter Abstand in 3D und in 2D

```
SELECT ST_AsEWKT(ST_3DLongestLine(line,pt)) AS lol3d_line_pt,
                ST_AsEWKT(ST_LongestLine(line,pt)) As lol2d_line_pt
        FROM (SELECT 'POINT(100 100 30)'::geometry As pt,
                        'LINESTRING (20 80 20, 98 190 1, 110 180 3, 50 75 1000)':: ↩
     geometry As line
                ) As foo;


           lol3d_line_pt            |       lol2d_line_pt
------------------------------------+---------------------------
 LINESTRING(50 75 1000,100 100 30) | LINESTRING(98 190,100 100)
```

Linienstück und Mehrfachpunkt -- größter Abstand in 3D und in 2D

```
SELECT ST_AsEWKT(ST_3DLongestLine(line,pt)) AS lol3d_line_pt,
                ST_AsEWKT(ST_LongestLine(line,pt)) As lol2d_line_pt
        FROM (SELECT 'MULTIPOINT(100 100 30, 50 74 1000)'::geometry As pt,
                        'LINESTRING (20 80 20, 98 190 1, 110 180 3, 50 75 900)':: ↩
     geometry As line
                ) As foo;


          lol3d_line_pt           |      lol2d_line_pt
----------------------------------+--------------------------
 LINESTRING(98 190 1,50 74 1000) | LINESTRING(98 190,50 74)
```

Mehrfachlinie und Polygon - größter Abstand in 3D und in 2D

```
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)
```

Siehe auch

ST_3DClosestPoint, ST_3DDistance, ST_LongestLine, ST_3DShortestLine, ST_3DMaxDistance

7.12.18 ST_MaxDistance

ST_MaxDistance — Gibt die größte 2-dimensionale Distanz zwischen zwei geometrischen Objekten in projizierten Einheiten zurück.

Synopsis

float **ST_MaxDistance**(geometry g1, geometry g2);

Beschreibung

Gibt die größte 2-dimensionale Distanz zwischen zwei geometrischen Objekten in projizierten Einheiten zurück. Wenn g1 und g2 dieselbe Geometrie sind, dann gibt die Funktion die Distanz zwischen den beiden am weitesten entfernten Knoten in dieser Geometrie zurück.

Gibt die größte 2-dimensionale Distanz zwischen zwei geometrischen Objekten in projizierten Einheiten zurück. Wenn g1 und g2 dieselbe Geometrie sind, dann gibt die Funktion die Distanz zwischen den beiden am weitesten entfernten Knoten in dieser Geometrie zurück.

Verfügbarkeit: 1.5.0

Beispiele

Längste Strecke zwischen Punkt und Linie

```
SELECT ST_MaxDistance('POINT(0 0)'::geometry, 'LINESTRING ( 2 0, 0 2 )'::geometry);
-----------------
 2

SELECT ST_MaxDistance('POINT(0 0)'::geometry, 'LINESTRING ( 2 2, 2 2 )'::geometry);
------------------
 2.82842712474619
```

Maximum distance between vertices of a single geometry.

```
SELECT ST_MaxDistance('POLYGON ((10 10, 10 0, 0 0, 10 10))'::geometry,
                      'POLYGON ((10 10, 10 0, 0 0, 10 10))'::geometry);
------------------
 14.142135623730951
```

Siehe auch

ST_Distance, ST_LongestLine, ST_DFullyWithin

7.12.19 ST_3DMaxDistance

ST_3DMaxDistance — Für den geometrischen Datentyp. Gibt die maximale 3-dimensionale kartesische Distanz (basierend auf dem Koordinatenreferenzsystem) zwischen zwei geometrischen Objekten in projizierten Einheiten zurück.

Synopsis

float **ST_3DMaxDistance**(geometry g1, geometry g2);

Beschreibung

Für den geometrischen Datentyp. Gibt die maximale 3-dimensionale kartesische Distanz zwischen zwei geometrischen Objekten in projizierten Einheiten (Einheiten des Koordinatenreferenzsystem) zurück.

This function supports 3d and will not drop the z-index.

This function supports Polyhedral surfaces.

Verfügbarkeit: 2.0.0

Änderung: 2.2.0 - Im Falle von 2D und 3D wird für ein fehlendes Z nicht mehr 0 angenommen.

Beispiele

```
-- Geometry example - units in meters (SRID: 2163 US National Atlas Equal area) (3D point  ↩
    and line compared 2D point and line)
-- Note: currently no vertical datum support so Z is not transformed and assumed to be same ↩
     units as final.
SELECT ST_3DMaxDistance(
                        ST_Transform(ST_GeomFromEWKT('SRID=4326;POINT(-72.1235 42.3521  ↩
                            10000)'),2163),
                        ST_Transform(ST_GeomFromEWKT('SRID=4326;LINESTRING(-72.1260 42.45  ↩
                            15, -72.123 42.1546 20)'),2163)
                ) As dist_3d,
                ST_MaxDistance(
                        ST_Transform(ST_GeomFromEWKT('SRID=4326;POINT(-72.1235 42.3521  ↩
                            10000)'),2163),
                        ST_Transform(ST_GeomFromEWKT('SRID=4326;LINESTRING(-72.1260 42.45  ↩
                            15, -72.123 42.1546 20)'),2163)
                ) As dist_2d;

     dist_3d      |     dist_2d
------------------+------------------
 24383.7467488441 | 22247.8472107251
```

Siehe auch

ST_Distance, ST_3DDWithin, ST_3DMaxDistance, ST_Transform

7.12.20 ST_MinimumClearance

ST_MinimumClearance — Gibt das Mindestabstandsmaß für eine Geometrie zurück; ein Maß für die Robustheit einer Geometrie.

Synopsis

float **ST_MinimumClearance**(geometry g);

Beschreibung

Es ist nicht ungewöhnlich dass eine Geometrie, welche die Kriterien für Validität gemäß ST_IsValid (Polygone) oder ST_IsSimple (Linien) erfüllt, durch eine geringe Verschiebung von einem Knoten invalid wird. Dies kann während einer Konvertierung in Textformate (wie WKT, KML, GML und GeoJSON) vorkommen, oder bei binären Formaten, welche die Koordinaten nicht als Gleitpunkt-Zahl mit doppelter Genauigkeit (double-precision floating point) abspeichern (MapInfo TAB).

The minimum clearance is a quantitative measure of a geometry's robustness to change in coordinate precision. It is the largest distance by which vertices of the geometry can be moved without creating an invalid geometry. Larger values of minimum clearance indicate greater robustness.

Wenn eine Geometrie ein Mindestabstandsmaß von e hat, dann gilt:

- Zwei sich unterscheidende Knoten der Geometrie sind nicht weniger als e voneinander entfernt.

- Kein Knoten liegt näher als e bei einem Liniensegment, außer es ist ein Endpunkt.

Wenn für eine Geometrie kein Mindestabstandsmaß existiert (zum Beispiel ein einzelner Punkt, oder ein Mehrfachpunkt, dessen Punkte identisch sind), dann gibt ST_MinimumClearance unendlich zurück.

To avoid validity issues caused by precision loss, ST_ReducePrecision can reduce coordinate precision while ensuring that polygonal geometry remains valid.

Verfügbarkeit: 2.3.0

Beispiele

```
SELECT ST_MinimumClearance('POLYGON ((0 0, 1 0, 1 1, 0.5 3.2e-4, 0 0))');
 st_minimumclearance
---------------------
             0.00032
```

Siehe auch

ST_MinimumClearanceLine, ST_Crosses, ST_Dimension, ST_Intersects

7.12.21 ST_MinimumClearanceLine

ST_MinimumClearanceLine — Gibt ein Linienstück mit zwei Punkten zurück, welche sich über das Mindestabstandsmaß erstreckt.

Synopsis

Geometry **ST_MinimumClearanceLine**(geometry g);

Beschreibung

Returns the two-point LineString spanning a geometry's minimum clearance. If the geometry does not have a minimum clearance, `LINESTRING EMPTY` is returned.

Wird durch das GEOS Modul ausgeführt

Verfügbarkeit: 2.3.0 - benötigt GEOS >= 3.6.0

Beispiele

```
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)
```

Siehe auch

ST_MinimumClearance

7.12.22 ST_Perimeter

ST_Perimeter — Returns the length of the boundary of a polygonal geometry or geography.

Synopsis

float **ST_Perimeter**(geometry g1);
float **ST_Perimeter**(geography geog, boolean use_spheroid = true);

Beschreibung

Gibt für die geometrischen/geographischen Datentypen ST_Surface, ST_MultiSurface (Polygon, MultiPolygon) den Umfang in 2D zurück. Bei einer nicht flächigen Geometrie wird 0 zurückgegeben. Für eine lineare Geometrie können Sie ST_Length verwenden. Beim geometrischen Datentyp sind die Einheiten der Umfangsmessung durch das Koordinatenreferenzsystem der Geometrie festgelegt.

For geography types, the calculations are performed using the inverse geodesic problem, where perimeter units are in meters. If PostGIS is compiled with PROJ version 4.8.0 or later, the spheroid is specified by the SRID, otherwise it is exclusive to WGS84. If `use_spheroid = false`, then calculations will approximate a sphere instead of a spheroid.

Zur Zeit ein Synonym für ST_Perimeter2D; dies kann sich allerdings ändern, wenn höhere Dimensionen unterstützt werden.

This method implements the OGC Simple Features Implementation Specification for SQL 1.1. s2.1.5.1

This method implements the SQL/MM specification. SQL-MM 3: 8.1.3, 9.5.4

Verfügbarkeit: Mit 2.0.0 wurde die Unterstützung für geograpischen Koordinaten eingeführt

Beispiele: geometrischer Datentyp

Den Umfang eines Polygons und eines Mehrfachpolygons in Fuß ausgeben. Beachten Sie bitte, dass die Einheit Fuß ist, da EPSG:2249 "Massachusetts State Plane Feet" ist

```
SELECT ST_Perimeter(ST_GeomFromText('POLYGON((743238 2967416,743238 2967450,743265 2967450,
743265.625 2967416,743238 2967416))', 2249));
st_perimeter
---------
 122.630744000095
(1 row)

SELECT ST_Perimeter(ST_GeomFromText('MULTIPOLYGON(((763104.471273676 2949418.44119003,
763104.477769673 2949418.42538203,
763104.189609677 2949418.22343004,763104.471273676 2949418.44119003)),
((763104.471273676 2949418.44119003,763095.804579742 2949436.33850239,
763086.132105649 2949451.46730207,763078.452329651 2949462.11549407,
763075.354136904 2949466.17407812,763064.362142565 2949477.64291974,
763059.953961626 2949481.28983009,762994.637609571 2949532.04103014,
762990.568508415 2949535.06640477,762986.710889563 2949539.61421415,
763117.237897679 2949709.50493431,763235.236617789 2949617.95619822,
763287.718121842 2949562.20592617,763111.553321674 2949423.91664605,
763104.471273676 2949418.44119003)))', 2249));
st_perimeter
---------
 845.227713366825
(1 row)
```

Beispiele: geographischer Datentyp

Gibt den Umfang eines Polygons und eines Mehrfachpolygons in Meter und in Fuß aus. Beachten Sie, dass diese in geographischen Koordinaten (WGS 84 Länge/Breite) vorliegen.

```
SELECT  ST_Perimeter(geog) As per_meters, ST_Perimeter(geog)/0.3048 As per_ft
FROM ST_GeogFromText('POLYGON((-71.1776848522251 42.3902896512902,-71.1776843766326  ↩
    42.3903829478009,
-71.1775844305465 42.3903826677917,-71.1775825927231 42.3902893647987,-71.1776848522251  ↩
    42.3902896512902))') As geog;

   per_meters    |      per_ft
-----------------+------------------
37.3790462565251 | 122.634666195949


-- MultiPolygon example --
SELECT  ST_Perimeter(geog) As per_meters, ST_Perimeter(geog,false) As per_sphere_meters,   ↩
    ST_Perimeter(geog)/0.3048 As per_ft
FROM ST_GeogFromText('MULTIPOLYGON(((-71.1044543107478 42.340674480411,-71.1044542869917  ↩
    42.3406744369506,
-71.1044553562977 42.340673886454,-71.1044543107478 42.340674480411)),
((-71.1044543107478 42.340674480411,-71.1044860600303 42.3407237015564,-71.1045215770124  ↩
    42.3407653385914,
-71.1045498002983 42.3407946553165,-71.1045611902745 42.3408058316308,-71.1046016507427  ↩
    42.340837442371,
-71.104617893173 42.3408475056957,-71.1048586153981 42.3409875993595,-71.1048736143677  ↩
    42.3409959528211,
-71.1048878050242 42.3410084812078,-71.1044020965803 42.3414730072048,
-71.1039672113619 42.3412202916693,-71.1037740497748 42.3410666421308,
-71.1044280218456 42.3406894151355,-71.1044543107478 42.340674480411)))') As geog;

    per_meters    | per_sphere_meters |      per_ft
------------------+-------------------+------------------
 257.634283683311 |  257.412311446337 | 845.256836231335
```

Siehe auch

ST_GeogFromText, ST_GeomFromText, ST_Length

7.12.23 ST_Perimeter2D

ST_Perimeter2D — Returns the 2D perimeter of a polygonal geometry. Alias for `ST_Perimeter`.

Synopsis

float **ST_Perimeter2D**(geometry geomA);

Beschreibung

Gibt den 2-dimensionalen Umfang eines Polygons oder eines Mehrfachpolygons zurück.

Note
Zurzeit ein Alias für ST_Perimeter. In zukünftigen Versionen dürfte ST_Perimeter den Umfang in der höchsten Dimension einer Geometrie zurückgeben. Dies befindet sich jedoch noch im Aufbau.

Siehe auch

ST_Perimeter

7.12.24 ST_3DPerimeter

ST_3DPerimeter — Gibt den geometrischen Schwerpunkt einer Geometrie zurück.

Synopsis

float **ST_3DPerimeter**(geometry geomA);

Beschreibung

Gibt den 3-dimensionalen Umfang eines Polygons oder eines Mehrfachpolygons zurück. Wenn es sich um eine 2-dimensionale Geometrie handelt wird der 2-dimensionale Umfang zurückgegeben.

This function supports 3d and will not drop the z-index.

This method implements the SQL/MM specification. SQL-MM ISO/IEC 13249-3: 8.1, 10.5

Änderung: 2.0.0 In Vorgängerversionen als ST_Perimeter3D bezeichet.

Beispiele

Umfang eines leicht erhöhten Polygons in "Massachusetts state plane feet"

```
SELECT ST_3DPerimeter(geom), ST_Perimeter2d(geom), ST_Perimeter(geom) FROM
                        (SELECT ST_GeomFromEWKT('SRID=2249;POLYGON((743238 2967416 2,743238 ↩
                            2967450 1,
743265.625 2967416 1,743238 2967416 2))') As geom) As foo;

  ST_3DPerimeter   |  st_perimeter2d  |   st_perimeter
-------------------+------------------+------------------
 105.465793597674 | 105.432997272188 | 105.432997272188
```

Siehe auch

ST_GeomFromEWKT, ST_Perimeter, ST_Perimeter2D

7.12.25 ST_ShortestLine

ST_ShortestLine — Gibt die 2-dimenionale kürzeste Strecke zwischen zwei Geometrien als Linie zurück

Synopsis

geometry **ST_ShortestLine**(geometry geom1, geometry geom2);
geography **ST_ShortestLine**(geography geom1, geography geom2, boolean use_spheroid = true);

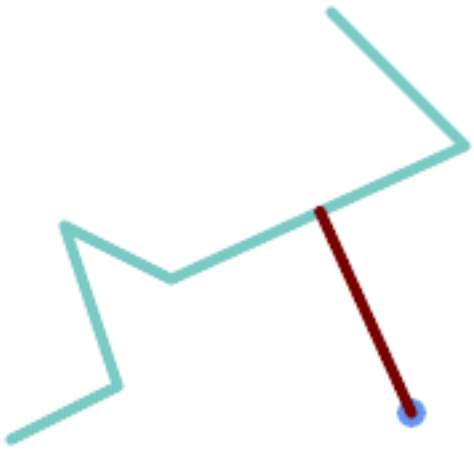
Beschreibung

Returns the 2-dimensional shortest line between two geometries. The line returned starts in `geom1` and ends in `geom2`. If `geom1` and `geom2` intersect the result is a line with start and end at an intersection point. The length of the line is the same as ST_Distance returns for g1 and g2.

Enhanced: 3.4.0 - support for geography.

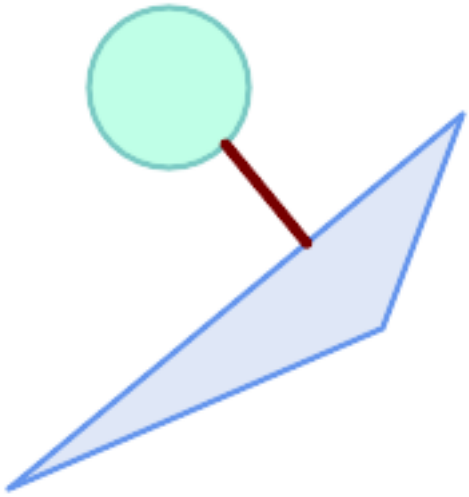
Verfügbarkeit: 1.5.0

Beispiele



Shortest line between Point and LineString

```
SELECT ST_AsText(  ST_ShortestLine(
        'POINT (160 40)',
        'LINESTRING (10 30, 50 50, 30 110, 70 90, 180 140, 130 190)')
        ) As sline;
------------------------------------------------------------
 LINESTRING(160 40,125.75342465753425 115.34246575342466)
```



Shortest line between Polygons

```
SELECT ST_AsText( ST_ShortestLine(
         'POLYGON ((190 150, 20 10, 160 70, 190 150))',
         ST_Buffer('POINT(80 160)', 30)
              ) ) AS llinewkt;
-----------------
LINESTRING(131.59149149528952 101.89887534906197,101.21320343559644 138.78679656440357)
```

Siehe auch

ST_ClosestPoint, ST_Distance, ST_LongestLine, ST_MaxDistance

7.12.26 ST_3DShortestLine

ST_3DShortestLine — Gibt den kürzesten 3-dimensionalen Abstand zwischen zwei geometrischen Objekten als Linie zurück

Synopsis

geometry **ST_3DShortestLine**(geometry g1, geometry g2);

Beschreibung

Gibt den kürzesten 3-dimensionalen Abstand zwischen zwei geometrischen Objekten als Linie zurück. Wenn es mehrere kürzeste Abstände gibt, dann wird nur der erste zurückgegeben, der von der Funktion gefunden wurde. Wenn sich g1 und g2 nur in einem Punkt schneiden, dann gibt die Funktion eine Linie zurück, die ihren Anfang und ihr Ende in dem Schnittpunkt hat. Wenn sich g1 und g2 in mehreren Punkten schneiden, dann gibt die Funktion eine Linie zurück, die Anfang und Ende in irgendeinem der Schnittpunkte hat. Die zurückgegebene Linie beginnt immer mit g1 und endet mit g2. Die Länge der 3D-Linie die von dieser Funktion zurückgegeben wird ist immer ident mit der von ST_3DDistance für g1 und g2 zurückgegebenen Distanz.

Verfügbarkeit: 2.0.0

Änderung: 2.2.0 - Wenn 2 geometrische Objekte in 2D übergegeben werden, wird ein 2D-Punkt zurückgegeben (anstelle wie früher 0 für ein fehlendes Z). Im Falle von 2D und 3D, wird für fehlende Z nicht länger 0 angenommen.

This function supports 3d and will not drop the z-index.

This function supports Polyhedral surfaces.

Beispiele

Linienzug und Punkt -- kürzester Abstand in 3D und in 2D

```
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(54.6993798867619 128.935022917228 11.5475869506606,100 100 30)  |  ↩
    LINESTRING(73.0769230769231 115.384615384615,100 100)
```

Linienstück und Mehrfachpunkt -- kürzester Abstand in 3D und in 2D

```
SELECT ST_AsEWKT(ST_3DShortestLine(line,pt)) AS shl3d_line_pt,
                ST_AsEWKT(ST_ShortestLine(line,pt)) As shl2d_line_pt
        FROM (SELECT 'MULTIPOINT(100 100 30, 50 74 1000)'::geometry As pt,
                        'LINESTRING (20 80 20, 98 190 1, 110 180 3, 50 75 900)':: ↩
    geometry As line
                ) As foo;


                        shl3d_line_pt                                          |  ↩
    shl2d_line_pt
-------------------------------------------------------------------------------+----------------------

 LINESTRING(54.6993798867619 128.935022917228 11.5475869506606,100 100 30) | LINESTRING ↩
    (50 75,50 74)
```

Mehrfachlinienzug und Polygon - kürzester Abstand in 3D und in 2D

```
SELECT ST_AsEWKT(ST_3DShortestLine(poly, mline)) As shl3d,
    ST_AsEWKT(ST_ShortestLine(poly, mline)) As shl2d
        FROM (SELECT  ST_GeomFromEWKT('POLYGON((175 150 5, 20 40 5, 35 45 5, 50 60 5,  ↩
    100 100 5, 175 150 5))') As poly,
                ST_GeomFromEWKT('MULTILINESTRING((175 155 2, 20 40 20, 50 60 -2, 125  ↩
    100 1, 175 155 1),
                (1 10 2, 5 20 1))') As mline ) As foo;
                   shl3d  ↩
                                                                                |     shl2d
-------------------------------------------------------------------------------------------------------

 LINESTRING(39.993580415989 54.1889925532825 5,40.4078575708294 53.6052383805529  ↩
    5.03423778139177) | LINESTRING(20 40,20 40)
```

Siehe auch

ST_3DClosestPoint, ST_3DDistance, ST_LongestLine, ST_ShortestLine, ST_3DMaxDistance

7.13 Overlay Functions

7.13.1 ST_ClipByBox2D

ST_ClipByBox2D — Computes the portion of a geometry falling within a rectangle.

Synopsis

geometry **ST_ClipByBox2D**(geometry geom, box2d box);

Beschreibung

Clips a geometry by a 2D box in a fast and tolerant but possibly invalid way. Topologically invalid input geometries do not result in exceptions being thrown. The output geometry is not guaranteed to be valid (in particular, self-intersections for a polygon may be introduced).

Performed by the GEOS module.

Availability: 2.2.0

Examples

```
-- Rely on implicit cast from geometry to box2d for the second parameter
SELECT ST_ClipByBox2D(geom, ST_MakeEnvelope(0,0,10,10)) FROM mytab;
```

Siehe auch

ST_Intersection, ST_MakeBox2D, ST_MakeEnvelope

7.13.2 ST_Difference

ST_Difference — Computes a geometry representing the part of geometry A that does not intersect geometry B.

Synopsis

geometry **ST_Difference**(geometry geomA, geometry geomB, float8 gridSize = -1);

Beschreibung

Returns a geometry representing the part of geometry A that does not intersect geometry B. This is equivalent to `A - ST_Intersecti`
If A is completely contained in B then an empty atomic geometry of appropriate type is returned.

Note
This is the only overlay function where input order matters. ST_Difference(A, B) always returns a portion of A.

If the optional `gridSize` argument is provided, the inputs are snapped to a grid of the given size, and the result vertices are computed on that same grid. (Requires GEOS-3.9.0 or higher)

Wird durch das GEOS Modul ausgeführt

Enhanced: 3.1.0 accept a gridSize parameter.

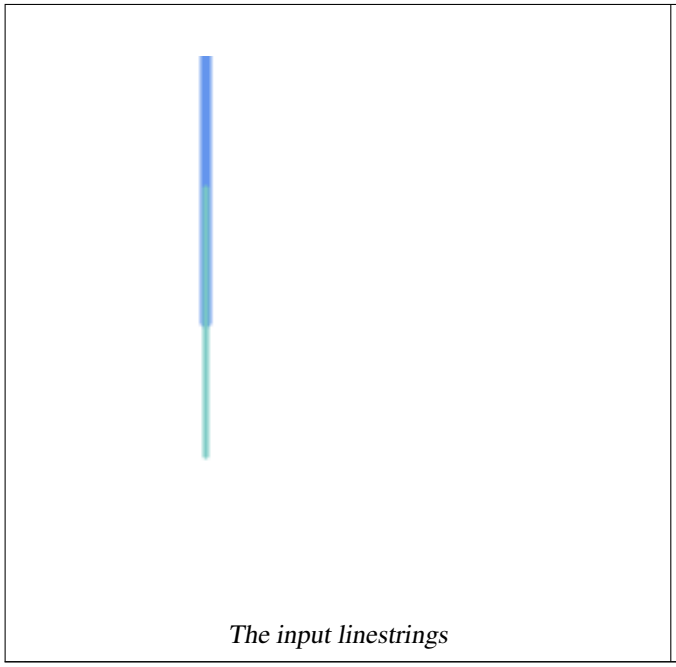
Requires GEOS >= 3.9.0 to use the gridSize parameter.

This method implements the OGC Simple Features Implementation Specification for SQL 1.1. s2.1.1.3

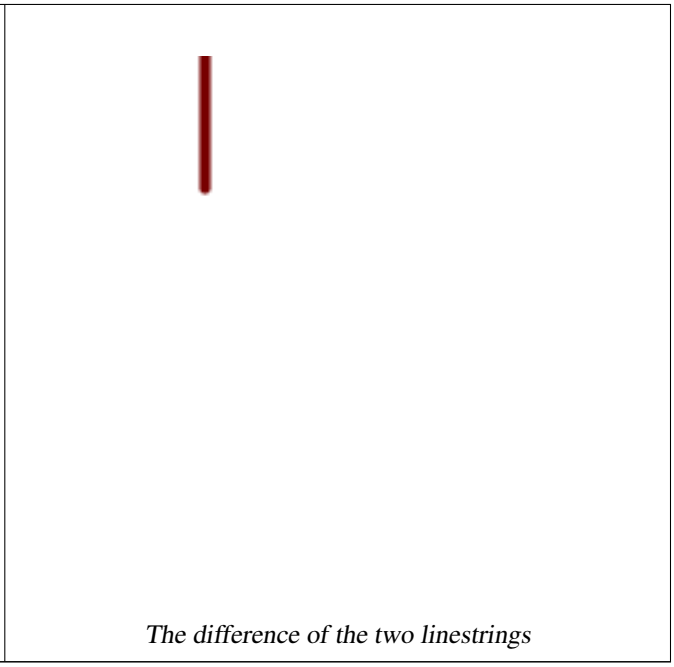
This method implements the SQL/MM specification. SQL-MM 3: 5.1.20

This function supports 3d and will not drop the z-index. However, the result is computed using XY only. The result Z values are copied, averaged or interpolated.

Examples



The input linestrings



The difference of the two linestrings

The difference of 2D linestrings.

```
SELECT ST_AsText(
    ST_Difference(
            'LINESTRING(50 100, 50 200)'::geometry,
            'LINESTRING(50 50, 50 150)'::geometry
        )
    );

st_astext
---------
LINESTRING(50 150,50 200)
```

The difference of 3D points.

```
SELECT ST_AsEWKT( ST_Difference(
                   'MULTIPOINT(-118.58 38.38 5,-118.60 38.329 6,-118.614 38.281 7)' ::  ↩
                       geometry,
                   'POINT(-118.614 38.281 5)' :: geometry
                  ) );

st_asewkt
---------
MULTIPOINT(-118.6 38.329 6,-118.58 38.38 5)
```

Siehe auch

ST_SymDifference, ST_Intersection, ST_Union

7.13.3 ST_Intersection

ST_Intersection — Computes a geometry representing the shared portion of geometries A and B.

Synopsis

geometry **ST_Intersection**(geometry geomA , geometry geomB , float8 gridSize = -1);
geography **ST_Intersection**(geography geogA , geography geogB);

Beschreibung

Returns a geometry representing the point-set intersection of two geometries. In other words, that portion of geometry A and geometry B that is shared between the two geometries.

If the geometries have no points in common (i.e. are disjoint) then an empty atomic geometry of appropriate type is returned.

If the optional `gridSize` argument is provided, the inputs are snapped to a grid of the given size, and the result vertices are computed on that same grid. (Requires GEOS-3.9.0 or higher)

ST_Intersection in conjunction with ST_Intersects is useful for clipping geometries such as in bounding box, buffer, or region queries where you only require the portion of a geometry that is inside a country or region of interest.

Note

For geography this is a thin wrapper around the geometry implementation. It first determines the best SRID that fits the bounding box of the 2 geography objects (if geography objects are within one half zone UTM but not same UTM will pick one of those) (favoring UTM or Lambert Azimuthal Equal Area (LAEA) north/south pole, and falling back on mercator in worst case scenario) and then intersection in that best fit planar spatial ref and retransforms back to WGS84 geography.

Warning

This function will drop the M coordinate values if present.

Warning

If working with 3D geometries, you may want to use SFGCAL based ST_3DIntersection which does a proper 3D intersection for 3D geometries. Although this function works with Z-coordinate, it does an averaging of Z-Coordinate.

Wird durch das GEOS Modul ausgeführt

Enhanced: 3.1.0 accept a gridSize parameter

Requires GEOS >= 3.9.0 to use the gridSize parameter

Changed: 3.0.0 does not depend on SFCGAL.

Availability: 1.5 support for geography data type was introduced.

This method implements the OGC Simple Features Implementation Specification for SQL 1.1. s2.1.1.3

This method implements the SQL/MM specification. SQL-MM 3: 5.1.18

This function supports 3d and will not drop the z-index. However, the result is computed using XY only. The result Z values are copied, averaged or interpolated.

Examples

```
SELECT ST_AsText(ST_Intersection('POINT(0 0)'::geometry, 'LINESTRING ( 2 0, 0 2 )':: ↩
    geometry));
 st_astext
---------------
GEOMETRYCOLLECTION EMPTY

SELECT ST_AsText(ST_Intersection('POINT(0 0)'::geometry, 'LINESTRING ( 0 0, 0 2 )':: ↩
    geometry));
 st_astext
---------------
POINT(0 0)
```

Clip all lines (trails) by country. Here we assume country geom are POLYGON or MULTIPOLYGONS. NOTE: we are only keeping intersections that result in a LINESTRING or MULTILINESTRING because we don't care about trails that just share a point. The dump is needed to expand a geometry collection into individual single MULT* parts. The below is fairly generic and will work for polys, etc. by just changing the where clause.

```
select clipped.gid, clipped.f_name, clipped_geom
from (
         select trails.gid, trails.f_name,
              (ST_Dump(ST_Intersection(country.geom, trails.geom))).geom clipped_geom
         from country
              inner join trails on ST_Intersects(country.geom, trails.geom)
     ) as clipped
where ST_Dimension(clipped.clipped_geom) = 1;
```

For polys e.g. polygon landmarks, you can also use the sometimes faster hack that buffering anything by 0.0 except a polygon results in an empty geometry collection. (So a geometry collection containing polys, lines and points buffered by 0.0 would only leave the polygons and dissolve the collection shell.)

```
select poly.gid,
    ST_Multi(
        ST_Buffer(
            ST_Intersection(country.geom, poly.geom),
            0.0
        )
    ) clipped_geom
from country
     inner join poly on ST_Intersects(country.geom, poly.geom)
where not ST_IsEmpty(ST_Buffer(ST_Intersection(country.geom, poly.geom), 0.0));
```

Examples: 2.5Dish

Note this is not a true intersection, compare to the same example using ST_3DIntersection.

```
select ST_AsText(ST_Intersection(linestring, polygon)) As wkt
from  ST_GeomFromText('LINESTRING Z (2 2 6,1.5 1.5 7,1 1 8,0.5 0.5 8,0 0 10)') AS  ↩
    linestring
 CROSS JOIN ST_GeomFromText('POLYGON((0 0 8, 0 1 8, 1 1 8, 1 0 8, 0 0 8))') AS polygon;

               st_astext
---------------------------------------
 LINESTRING Z (1 1 8,0.5 0.5 8,0 0 10)
```

Siehe auch

ST_3DIntersection, ST_Difference, ST_Union, ST_Dimension, ST_Dump, ST_Force2D, ST_SymDifference, ST_Intersects, ST_Multi

7.13.4 ST_MemUnion

ST_MemUnion — Aggregate function which unions geometries in a memory-efficent but slower way

Synopsis

geometry **ST_MemUnion**(geometry set geomfield);

Beschreibung

An aggregate function that unions the input geometries, merging them to produce a result geometry with no overlaps. The output may be a single geometry, a MultiGeometry, or a Geometry Collection.

Note
Produces the same result as ST_Union, but uses less memory and more processor time. This aggregate function works by unioning the geometries incrementally, as opposed to the ST_Union aggregate which first accumulates an array and then unions the contents using a fast algorithm.

This function supports 3d and will not drop the z-index. However, the result is computed using XY only. The result Z values are copied, averaged or interpolated.

Examples

```
SELECT id,
       ST_MemUnion(geom) as singlegeom
FROM sometable f
GROUP BY id;
```

Siehe auch

ST_Union

7.13.5 ST_Node

ST_Node — Nodes a collection of lines.

Synopsis

geometry **ST_Node**(geometry geom);

Beschreibung

Returns a (Multi)LineString representing the fully noded version of a collection of linestrings. The noding preserves all of the input nodes, and introduces the least possible number of new nodes. The resulting linework is dissolved (duplicate lines are removed).

This is a good way to create fully-noded linework suitable for use as input to ST_Polygonize.

ST_UnaryUnion can also be used to node and dissolve linework. It provides an option to specify a gridSize, which can provide simpler and more robust output. See also ST_Union for an aggregate variant.

This function supports 3d and will not drop the z-index.

Performed by the GEOS module.

Verfügbarkeit: 2.0.0

Changed: 2.4.0 this function uses GEOSNode internally instead of GEOSUnaryUnion. This may cause the resulting linestrings to have a different order and direction compared to PostGIS < 2.4.

Examples

Noding a 3D LineString which self-intersects

```
SELECT ST_AsText(
        ST_Node('LINESTRINGZ(0 0 0, 10 10 10, 0 10 5, 10 0 3)'::geometry)
    ) As  output;
output
-----------
MULTILINESTRING Z ((0 0 0,5 5 4.5),(5 5 4.5,10 10 10,0 10 5,5 5 4.5),(5 5 4.5,10 0 3))
```

Noding two LineStrings which share common linework. Note that the result linework is dissolved.

```
SELECT ST_AsText(
        ST_Node('MULTILINESTRING ((2 5, 2 1, 7 1), (6 1, 4 1, 2 3, 2 5))'::geometry)
    ) As  output;
output
-----------
MULTILINESTRING((2 5,2 3),(2 3,2 1,4 1),(4 1,2 3),(4 1,6 1),(6 1,7 1))
```

Siehe auch

ST_UnaryUnion, ST_Union

7.13.6 ST_Split

ST_Split — Returns a collection of geometries created by splitting a geometry by another geometry.

Synopsis

geometry **ST_Split**(geometry input, geometry blade);

Beschreibung

The function supports splitting a LineString by a (Multi)Point, (Multi)LineString or (Multi)Polygon boundary, or a (Multi)Polygon by a LineString. When a (Multi)Polygon is used as as the blade, its linear components (the boundary) are used for splitting the input. The result geometry is always a collection.

This function is in a sense the opposite of ST_Union. Applying ST_Union to the returned collection should theoretically yield the original geometry (although due to numerical rounding this may not be exactly the case).

Note
If the the input and blade do not intersect due to numerical precision issues, the input may not be split as expected. To avoid this situation it may be necessary to snap the input to the blade first, using ST_Snap with a small tolerance.

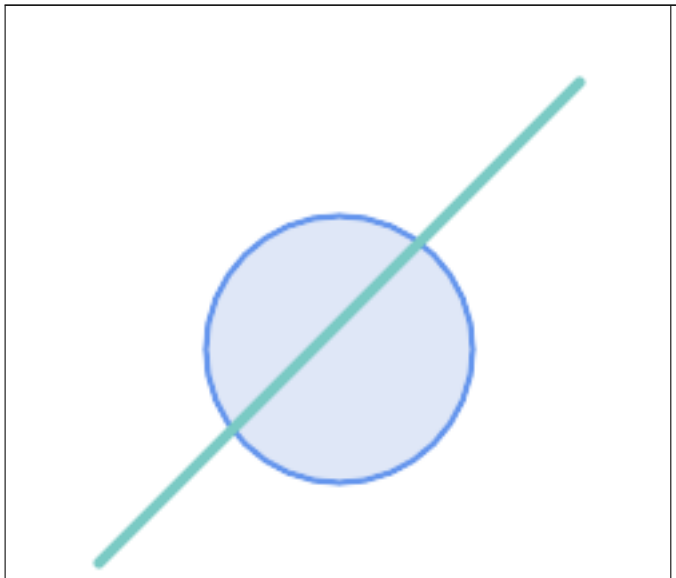
Availability: 2.0.0 requires GEOS

Enhanced: 2.2.0 support for splitting a line by a multiline, a multipoint or (multi)polygon boundary was introduced.

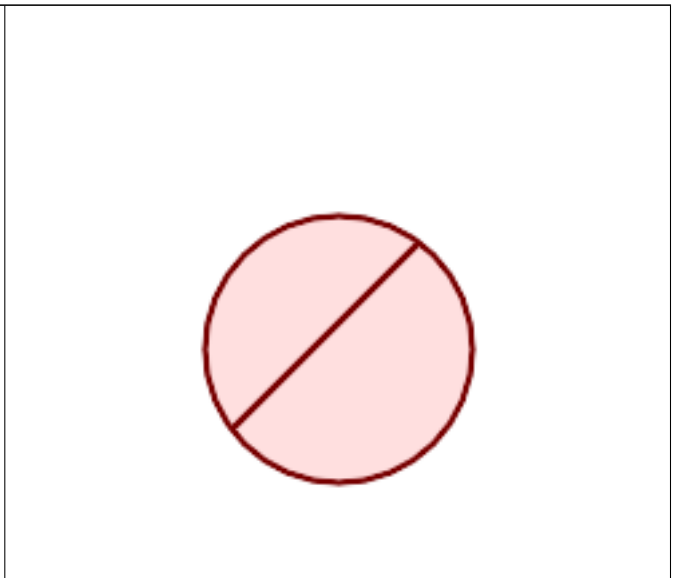
Enhanced: 2.5.0 support for splitting a polygon by a multiline was introduced.

Examples

Split a Polygon by a Line.



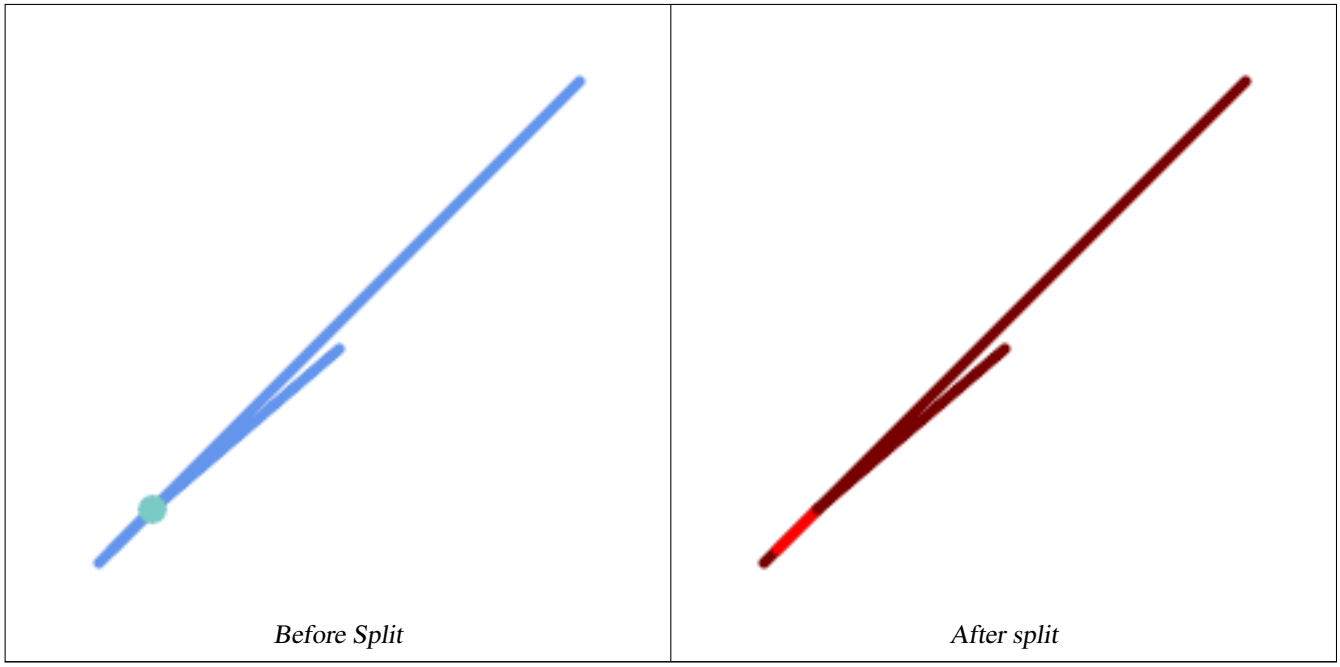
Before Split



After split

```
SELECT ST_AsText( ST_Split(
                ST_Buffer(ST_GeomFromText('POINT(100 90)'), 50), -- circle
                ST_MakeLine(ST_Point(10, 10),ST_Point(190, 190)) -- line
    ));

-- result --
 GEOMETRYCOLLECTION(
            POLYGON((150 90,149.039264020162 80.2454838991936,146.193976625564  ↩
                70.8658283817455,..),
            POLYGON(..))
)
```

Split a MultiLineString a Point, where the point lies exactly on both LineStrings elements.



Before Split | *After split*

```
SELECT ST_AsText(ST_Split(
    'MULTILINESTRING((10 10, 190 191), (15 15, 30 30, 100 90))',
    ST_Point(30,30))) As split;

split
------
GEOMETRYCOLLECTION(
    LINESTRING(10 10,30 30),
    LINESTRING(30 30,190 190),
    LINESTRING(15 15,30 30),
    LINESTRING(30 30,100 90)
)
```

Split a LineString by a Point, where the point does not lie exactly on the line. Shows using ST_Snap to snap the line to the point to allow it to be split.

```
WITH data AS (SELECT
  'LINESTRING(0 0, 100 100)'::geometry AS line,
  'POINT(51 50)':: geometry AS point
)
SELECT ST_AsText( ST_Split( ST_Snap(line, point, 1), point)) AS snapped_split,
       ST_AsText( ST_Split(line, point)) AS not_snapped_not_split
       FROM data;

                            snapped_split                                |             ↩
                                not_snapped_not_split
-------------------------------------------------------------------------+-----------------------------

 GEOMETRYCOLLECTION(LINESTRING(0 0,51 50),LINESTRING(51 50,100 100)) | GEOMETRYCOLLECTION( ↩
     LINESTRING(0 0,100 100))
```

Siehe auch

ST_Snap, ST_Union

7.13.7 ST_Subdivide

ST_Subdivide — Computes a rectilinear subdivision of a geometry.

Synopsis

setof geometry **ST_Subdivide**(geometry geom, integer max_vertices=256, float8 gridSize = -1);

Beschreibung

Returns a set of geometries that are the result of dividing `geom` into parts using rectilinear lines, with each part containing no more than `max_vertices`.

`max_vertices` must be 5 or more, as 5 points are needed to represent a closed box. `gridSize` can be specified to have clipping work in fixed-precision space (requires GEOS-3.9.0+).

Point-in-polygon and other spatial operations are normally faster for indexed subdivided datasets. Since the bounding boxes for the parts usually cover a smaller area than the original geometry bbox, index queries produce fewer "hit" cases. The "hit" cases are faster because the spatial operations executed by the index recheck process fewer points.

Note
This is a set-returning function (SRF) that return a set of rows containing single geometry values. It can be used in a SELECT list or a FROM clause to produce a result set with one record for each result geometry.

Performed by the GEOS module.

Availability: 2.2.0

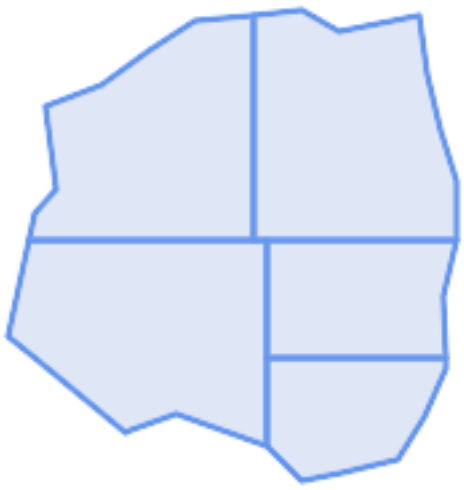
Enhanced: 2.5.0 reuses existing points on polygon split, vertex count is lowered from 8 to 5.

Enhanced: 3.1.0 accept a gridSize parameter.

Requires GEOS >= 3.9.0 to use the gridSize parameter

Examples

Example: Subdivide a polygon into parts with no more than 10 vertices, and assign each part a unique id.



Subdivided to maximum 10 vertices

```
SELECT row_number() OVER() As rn, ST_AsText(geom) As wkt
    FROM (SELECT ST_SubDivide(
        'POLYGON((132 10,119 23,85 35,68 29,66 28,49 42,32 56,22 64,32 110,40 119,36 150,
        57 158,75 171,92 182,114 184,132 186,146 178,176 184,179 162,184 141,190 122,
        190 100,185 79,186 56,186 52,178 34,168 18,147 13,132 10))'::geometry,10))  AS f( ↩
           geom);
```

```
rn &#x2502;                                                          wkt
&#x2500;&#x2500;&#x2500;&#x2500;&#x253c;&#x2500;&#x2500;&#x2500
```

Example: Subdivide the complex geometries of a table in-place. The original geometry records are deleted from the source table, and new records for each subdivided result geometry are inserted.

```
WITH complex_areas_to_subdivide AS (
    DELETE from polygons_table
    WHERE ST_NPoints(geom)
> 255
    RETURNING id, column1, column2, column3, geom
)
INSERT INTO polygons_table (fid, column1, column2, column3, geom)
    SELECT fid, column1, column2, column3,
           ST_Subdivide(geom, 255) as geom
    FROM complex_areas_to_subdivide;
```

Example: Create a new table containing subdivided geometries, retaining the key of the original geometry so that the new table can be joined to the source table. Since ST_Subdivide is a set-returning (table) function that returns a set of single-value rows, this syntax automatically produces a table with one row for each result part.

```
CREATE TABLE subdivided_geoms AS
    SELECT pkey, ST_Subdivide(geom) AS geom
    FROM original_geoms;
```

Siehe auch

ST_ClipByBox2D, ST_Segmentize, ST_Split, ST_NPoints

7.13.8 ST_SymDifference

ST_SymDifference — Computes a geometry representing the portions of geometries A and B that do not intersect.

Synopsis

geometry **ST_SymDifference**(geometry geomA, geometry geomB, float8 gridSize = -1);

Beschreibung

Returns a geometry representing the portions of geonetries A and B that do not intersect. This is equivalent to `ST_Union(A,B) - ST_Intersection(A,B)`. It is called a symmetric difference because `ST_SymDifference(A,B) = ST_SymDifference`

If the optional `gridSize` argument is provided, the inputs are snapped to a grid of the given size, and the result vertices are computed on that same grid. (Requires GEOS-3.9.0 or higher)

Wird durch das GEOS Modul ausgeführt

Enhanced: 3.1.0 accept a gridSize parameter.

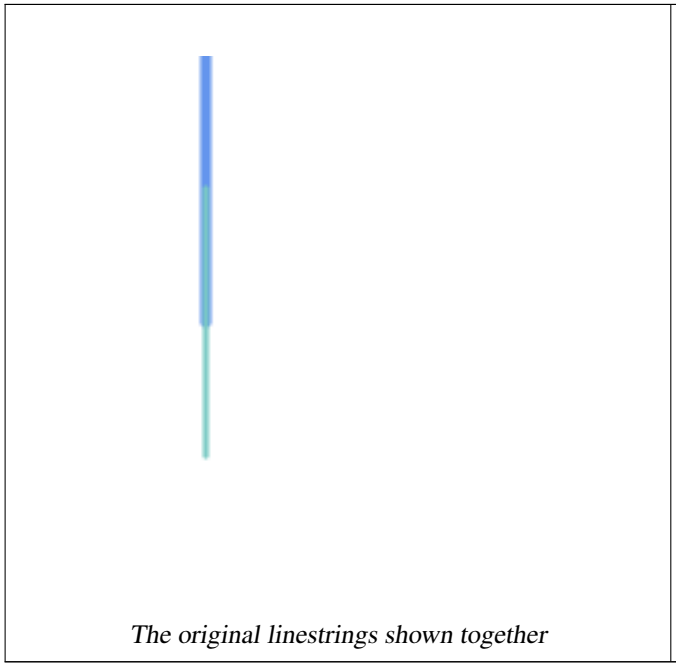
Requires GEOS >= 3.9.0 to use the gridSize parameter

This method implements the OGC Simple Features Implementation Specification for SQL 1.1. s2.1.1.3

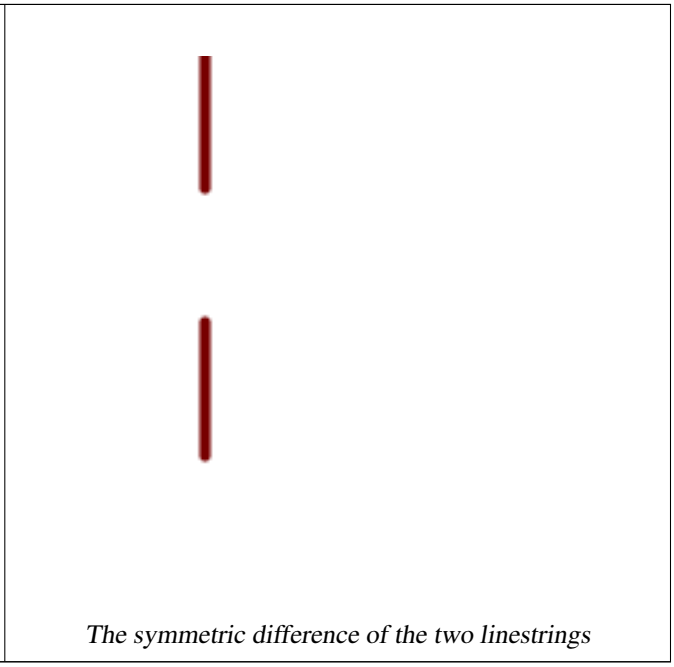
This method implements the SQL/MM specification. SQL-MM 3: 5.1.21

This function supports 3d and will not drop the z-index. However, the result is computed using XY only. The result Z values are copied, averaged or interpolated.

Examples



The original linestrings shown together



The symmetric difference of the two linestrings

```
--Safe for 2d - symmetric difference of 2 linestrings
SELECT ST_AsText(
    ST_SymDifference(
        ST_GeomFromText('LINESTRING(50 100, 50 200)'),
        ST_GeomFromText('LINESTRING(50 50, 50 150)')
    )
);

st_astext
---------
MULTILINESTRING((50 150,50 200),(50 50,50 100))
```

```
--When used in 3d doesn't quite do the right thing
SELECT ST_AsEWKT(ST_SymDifference(ST_GeomFromEWKT('LINESTRING(1 2 1, 1 4 2)'),
    ST_GeomFromEWKT('LINESTRING(1 1 3, 1 3 4)')))

st_astext
------------
MULTILINESTRING((1 3 2.75,1 4 2),(1 1 3,1 2 2.25))
```

Siehe auch

ST_Difference, ST_Intersection, ST_Union

7.13.9 ST_UnaryUnion

ST_UnaryUnion — Computes the union of the components of a single geometry.

Synopsis

geometry **ST_UnaryUnion**(geometry geom, float8 gridSize = -1);

Beschreibung

A single-input variant of ST_Union. The input may be a single geometry, a MultiGeometry, or a GeometryCollection. The union is applied to the individual elements of the input.

This function can be used to fix MultiPolygons which are invalid due to overlapping components. However, the input components must each be valid. An invalid input component such as a bow-tie polygon may cause an error. For this reason it may be better to use ST_MakeValid.

Another use of this function is to node and dissolve a collection of linestrings which cross or overlap to make them simple. (ST_Node also does this, but it does not provide the `gridSize` option.)

It is possible to combine ST_UnaryUnion with ST_Collect to fine-tune how many geometries are be unioned at once. This allows trading off between memory usage and compute time, striking a balance between ST_Union and ST_MemUnion.

If the optional `gridSize` argument is provided, the inputs are snapped to a grid of the given size, and the result vertices are computed on that same grid. (Requires GEOS-3.9.0 or higher)

This function supports 3d and will not drop the z-index. However, the result is computed using XY only. The result Z values are copied, averaged or interpolated.

Enhanced: 3.1.0 accept a gridSize parameter.

Requires GEOS >= 3.9.0 to use the gridSize parameter

Verfügbarkeit: 2.0.0

Siehe auch

ST_Union, ST_MemUnion, ST_MakeValid, ST_Collect, ST_Node

7.13.10 ST_Union

ST_Union — Computes a geometry representing the point-set union of the input geometries.

Synopsis

geometry **ST_Union**(geometry g1, geometry g2);
geometry **ST_Union**(geometry g1, geometry g2, float8 gridSize);
geometry **ST_Union**(geometry[] g1_array);
geometry **ST_Union**(geometry set g1field);
geometry **ST_Union**(geometry set g1field, float8 gridSize);

Beschreibung

Unions the input geometries, merging geometry to produce a result geometry with no overlaps. The output may be an atomic geometry, a MultiGeometry, or a Geometry Collection. Comes in several variants:

Two-input variant: returns a geometry that is the union of two input geometries. If either input is NULL, then NULL is returned.

Array variant: returns a geometry that is the union of an array of geometries.

Aggregate variant: returns a geometry that is the union of a rowset of geometries. The ST_Union() function is an "aggregate" function in the terminology of PostgreSQL. That means that it operates on rows of data, in the same way the SUM() and AVG() functions do and like most aggregates, it also ignores NULL geometries.

See ST_UnaryUnion for a non-aggregate, single-input variant.

The ST_Union array and set variants use the fast Cascaded Union algorithm described in http://blog.cleverelephant.ca/2009/01/-must-faster-unions-in-postgis-14.html

A `gridSize` can be specified to work in fixed-precision space. The inputs are snapped to a grid of the given size, and the result vertices are computed on that same grid. (Requires GEOS-3.9.0 or higher)

Note
ST_Collect may sometimes be used in place of ST_Union, if the result is not required to be non-overlapping. ST_Collect is usually faster than ST_Union because it performs no processing on the collected geometries.

Performed by the GEOS module.

ST_Union creates MultiLineString and does not sew LineStrings into a single LineString. Use ST_LineMerge to sew LineStrings.

NOTE: this function was formerly called GeomUnion(), which was renamed from "Union" because UNION is an SQL reserved word.

Enhanced: 3.1.0 accept a gridSize parameter.

Requires GEOS >= 3.9.0 to use the gridSize parameter

Changed: 3.0.0 does not depend on SFCGAL.

Availability: 1.4.0 - ST_Union was enhanced. ST_Union(geomarray) was introduced and also faster aggregate collection in PostgreSQL.

This method implements the OGC Simple Features Implementation Specification for SQL 1.1. s2.1.1.3

Note
Aggregate version is not explicitly defined in OGC SPEC.

This method implements the SQL/MM specification. SQL-MM 3: 5.1.19 the z-index (elevation) when polygons are involved.

This function supports 3d and will not drop the z-index. However, the result is computed using XY only. The result Z values are copied, averaged or interpolated.

Examples

Aggregate example

```
SELECT id,
       ST_Union(geom) as singlegeom
FROM sometable f
GROUP BY id;
```

Non-Aggregate example

```
select ST_AsText(ST_Union('POINT(1 2)' :: geometry, 'POINT(-2 3)' :: geometry))

st_astext
----------
MULTIPOINT(-2 3,1 2)

select ST_AsText(ST_Union('POINT(1 2)' :: geometry, 'POINT(1 2)' :: geometry))

st_astext
----------
POINT(1 2)
```

3D example - sort of supports 3D (and with mixed dimensions!)

```
select ST_AsEWKT(ST_Union(geom))
from (
         select 'POLYGON((-7 4.2,-7.1 4.2,-7.1 4.3, -7 4.2))'::geometry geom
         union all
         select 'POINT(5 5 5)'::geometry geom
         union all
         select 'POINT(-2 3 1)'::geometry geom
         union all
         select 'LINESTRING(5 5 5, 10 10 10)'::geometry geom
     ) as foo;

st_asewkt
---------
GEOMETRYCOLLECTION(POINT(-2 3 1),LINESTRING(5 5 5,10 10 10),POLYGON((-7 4.2 5,-7.1 4.2  ↩
    5,-7.1 4.3 5,-7 4.2 5)));
```

3d example not mixing dimensions

```
select ST_AsEWKT(ST_Union(geom))
from (
         select 'POLYGON((-7 4.2 2,-7.1 4.2 3,-7.1 4.3 2, -7 4.2 2))'::geometry geom
         union all
         select 'POINT(5 5 5)'::geometry geom
         union all
         select 'POINT(-2 3 1)'::geometry geom
         union all
         select 'LINESTRING(5 5 5, 10 10 10)'::geometry geom
     ) as foo;

st_asewkt
---------
GEOMETRYCOLLECTION(POINT(-2 3 1),LINESTRING(5 5 5,10 10 10),POLYGON((-7 4.2 2,-7.1 4.2  ↩
    3,-7.1 4.3 2,-7 4.2 2)))

--Examples using new Array construct
SELECT ST_Union(ARRAY(SELECT geom FROM sometable));

SELECT ST_AsText(ST_Union(ARRAY[ST_GeomFromText('LINESTRING(1 2, 3 4)'),
            ST_GeomFromText('LINESTRING(3 4, 4 5)')])) As wktunion;

--wktunion---
MULTILINESTRING((3 4,4 5),(1 2,3 4))
```

Siehe auch

ST_Collect, ST_UnaryUnion, ST_MemUnion, ST_Intersection, ST_Difference, ST_SymDifference

7.14 Geometrieverarbeitung

7.14.1 ST_Buffer

ST_Buffer — Computes a geometry covering all points within a given distance from a geometry.

Synopsis

geometry **ST_Buffer**(geometry g1, float radius_of_buffer, text buffer_style_parameters = '');
geometry **ST_Buffer**(geometry g1, float radius_of_buffer, integer num_seg_quarter_circle);
geography **ST_Buffer**(geography g1, float radius_of_buffer, text buffer_style_parameters);
geography **ST_Buffer**(geography g1, float radius_of_buffer, integer num_seg_quarter_circle);

Beschreibung

Computes a POLYGON or MULTIPOLYGON that represents all points whose distance from a geometry/geography is less than or equal to a given distance. A negative distance shrinks the geometry rather than expanding it. A negative distance may shrink a polygon completely, in which case POLYGON EMPTY is returned. For points and lines negative distances always return empty results.

For geometry, the distance is specified in the units of the Spatial Reference System of the geometry. For geography, the distance is specified in meters.

The optional third parameter controls the buffer accuracy and style. The accuracy of circular arcs in the buffer is specified as the number of line segments used to approximate a quarter circle (default is 8). The buffer style can be specifed by providing a list of blank-separated key=value pairs as follows:

- 'quad_segs=#' : number of line segments used to approximate a quarter circle (default is 8).
- 'endcap=round|flat|square' : endcap style (defaults to "round"). 'butt' is accepted as a synonym for 'flat'.
- 'join=round|mitre|bevel' : join style (defaults to "round"). 'miter' is accepted as a synonym for 'mitre'.
- 'mitre_limit=#.#' : mitre ratio limit (only affects mitered join style). 'miter_limit' is accepted as a synonym for 'mitre_limit'.
- 'side=both|left|right' : 'left' or 'right' performs a single-sided buffer on the geometry, with the buffered side relative to the direction of the line. This is only applicable to LINESTRING geometry and does not affect POINT or POLYGON geometries. By default end caps are square.

Note

For geography this is a thin wrapper around the geometry implementation. It determines a planar spatial reference system that best fits the bounding box of the geography object (trying UTM, Lambert Azimuthal Equal Area (LAEA) North/South pole, and finally Mercator). The buffer is computed in the planar space, and then transformed back to WGS84. This may not produce the desired behavior if the input object is much larger than a UTM zone or crosses the dateline

Note

Buffer output is always a valid polygonal geometry. Buffer can handle invalid inputs, so buffering by distance 0 is sometimes used as a way of repairing invalid polygons. ST_MakeValid can also be used for this purpose.

Note

Buffering is sometimes used to perform a within-distance search. For this use case it is more efficient to use ST_DWithin.

Note

This function ignores the Z dimension. It always gives a 2D result even when used on a 3D geometry.

Erweiterung: 2.5.0 - ST_Buffer ermöglicht jetzt auch eine seitliche Pufferzonenberechnung über `side=both|left|right`.

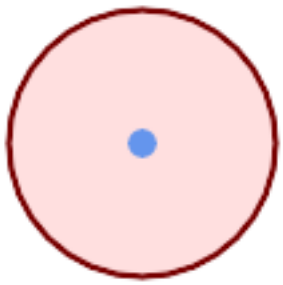
Verfügbarkeit: 1.5 - ST_Buffer wurde um die Unterstützung von Abschlusstücken/endcaps und Join-Typen erweitert. Diese können zum Beispiel dazu verwendet werden, um Linienzüge von Straßen in Straßenpolygone mit flachen oder rechtwinkeligen Abschlüssen anstatt mit runden Enden umzuwandeln. Ein schlanker Adapter für den geographischen Datentyp wurde hinzugefügt.

Wird vom GEOS Modul ausgeführt

This method implements the OGC Simple Features Implementation Specification for SQL 1.1. s2.1.1.3

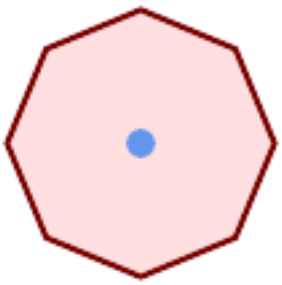
This method implements the SQL/MM specification. SQL-MM IEC 13249-3: 5.1.30

Beispiele



quad_segs=8 (Standardwert)

```
SELECT ST_Buffer(
 ST_GeomFromText('POINT(100 90)'),
 50, 'quad_segs=8');
```



quad_segs=2 (lahme Ente)

```
SELECT ST_Buffer(
 ST_GeomFromText('POINT(100 90)'),
 50, 'quad_segs=2');
```

endcap=round join=round (Standardwert)

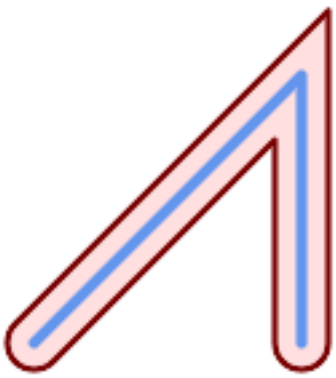
```
SELECT ST_Buffer(
 ST_GeomFromText(
  'LINESTRING(50 50,150 150,150 50)'
 ), 10, 'endcap=round join=round');
```

endcap=square

```
SELECT ST_Buffer(
 ST_GeomFromText(
  'LINESTRING(50 50,150 150,150 50)'
 ), 10, 'endcap=square join=round');
```

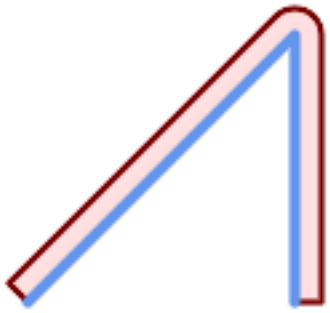
join=bevel

```
SELECT ST_Buffer(
 ST_GeomFromText(
  'LINESTRING(50 50,150 150,150 50)'
 ), 10, 'join=bevel');
```



join=mitre mitre_limit=5.0 (default mitre limit)

```
SELECT ST_Buffer(
 ST_GeomFromText(
  'LINESTRING(50 50,150 150,150 50)'
 ), 10, 'join=mitre mitre_limit=5.0');
```

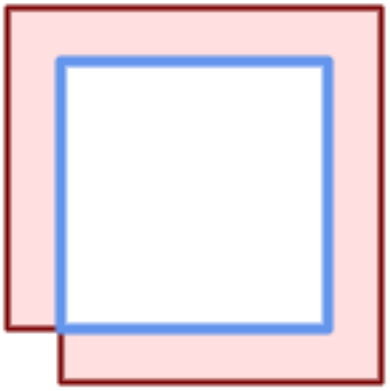


side=left

```
SELECT ST_Buffer(
 ST_GeomFromText(
  'LINESTRING(50 50,150 150,150 50)'
 ), 10, 'side=left');
```

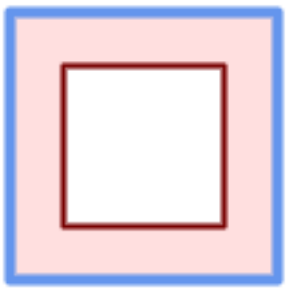
side=right

```
SELECT ST_Buffer(
 ST_GeomFromText(
  'LINESTRING(50 50,150 150,150 50)'
 ), 10, 'side=right');
```



right-hand-winding, polygon boundary side=left

```
SELECT ST_Buffer(
ST_ForceRHR(
ST_Boundary(
 ST_GeomFromText(
'POLYGON ((50 50, 50 150, 150 150, 150  ↩
    50, 50 50))'))),
 ), 20, 'side=left');
```



right-hand-winding, polygon boundary side=right

```
SELECT ST_Buffer(
ST_ForceRHR(
ST_Boundary(
 ST_GeomFromText(
'POLYGON ((50 50, 50 150, 150 150, 150  ↩
    50, 50 50))'))
), 20,'side=right')
```

```
--A buffered point approximates a circle
-- A buffered point forcing approximation of (see diagram)
```

```
-- 2 points per quarter circle is poly with 8 sides (see diagram)
SELECT ST_NPoints(ST_Buffer(ST_GeomFromText('POINT(100 90)'), 50)) As  ↩
    promisingcircle_pcount,
ST_NPoints(ST_Buffer(ST_GeomFromText('POINT(100 90)'), 50, 2)) As lamecircle_pcount;

promisingcircle_pcount | lamecircle_pcount
-----------------------+-------------------
             33 |                 9

--A lighter but lamer circle
-- only 2 points per quarter circle is an octagon
--Below is a 100 meter octagon
-- Note coordinates are in NAD 83 long lat which we transform
to Mass state plane meter and then buffer to get measurements in meters;
SELECT ST_AsText(ST_Buffer(
ST_Transform(
ST_SetSRID(ST_Point(-71.063526, 42.35785),4269), 26986)
,100,2)) As octagon;
----------------------
POLYGON((236057.59057465 900908.759918696,236028.301252769 900838.049240578,235
957.59057465 900808.759918696,235886.879896532 900838.049240578,235857.59057465
900908.759918696,235886.879896532 900979.470596815,235957.59057465 901008.759918
696,236028.301252769 900979.470596815,236057.59057465 900908.759918696))
```

Siehe auch

ST_Collect, ST_DWithin, ST_SetSRID, ST_Transform, ST_Union, ST_MakeValid

7.14.2 ST_BuildArea

ST_BuildArea — Creates a polygonal geometry formed by the linework of a geometry.

Synopsis

geometry **ST_BuildArea**(geometry geom);

Beschreibung

Creates an areal geometry formed by the constituent linework of the input geometry. The input can be a LineString, MultiLineString, Polygon, MultiPolygon or a GeometryCollection. The result is a Polygon or MultiPolygon, depending on input. If the input linework does not form polygons, NULL is returned.

Unlike ST_MakePolygon, this function accepts rings formed by multiple lines, and can form any number of polygons.

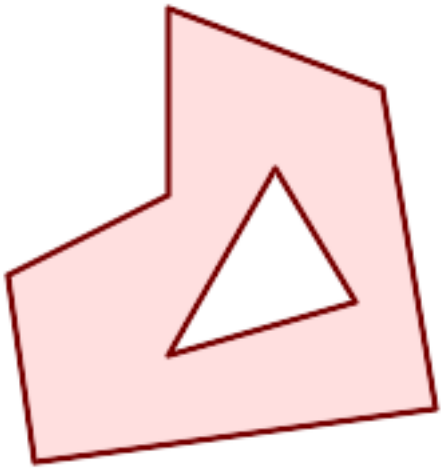
This function converts inner rings into holes. To turn inner rings into polygons as well, use ST_Polygonize.

Note
Input linework must be correctly noded for this function to work properly. ST_Node can be used to node lines.
If the input linework crosses, this function will produce invalid polygons. ST_MakeValid can be used to ensure the output is valid.

Verfügbarkeit: 1.1.0

Beispiele

Input lines



Area result

```
WITH data(geom) AS (VALUES
   ('LINESTRING (180 40, 30 20, 20 90)'::geometry)
  ,('LINESTRING (180 40, 160 160)'::geometry)
  ,('LINESTRING (160 160, 80 190, 80 120, 20 90)'::geometry)
  ,('LINESTRING (80 60, 120 130, 150 80)'::geometry)
  ,('LINESTRING (80 60, 150 80)'::geometry)
)
SELECT ST_AsText( ST_BuildArea( ST_Collect( geom )))
    FROM data;

-----------------------------------------------------------------------------------------
POLYGON((180 40,30 20,20 90,80 120,80 190,160 160,180 40),(150 80,120 130,80 60,150 80))
```

Create a donut from two circular polygons

```
SELECT ST_BuildArea(ST_Collect(inring,outring))
FROM (SELECT
```

```
    ST_Buffer('POINT(100 90)', 25) As inring,
    ST_Buffer('POINT(100 90)', 50) As outring) As t;
```

Siehe auch

ST_Collect, ST_MakePolygon, ST_MakeValid, ST_Node, ST_Polygonize, ST_BdPolyFromText, ST_BdMPolyFromText (wrappers to this function with standard OGC interface)

7.14.3 ST_Centroid

ST_Centroid — Gibt den geometrischen Schwerpunkt einer Geometrie zurück.

Synopsis

geometry **ST_Centroid**(geometry g1);
geography **ST_Centroid**(geography g1, boolean use_spheroid = true);

Beschreibung

Computes a point which is the geometric center of mass of a geometry. For [MULTI]POINTs, the centroid is the arithmetic mean of the input coordinates. For [MULTI]LINESTRINGs, the centroid is computed using the weighted length of each line segment. For [MULTI]POLYGONs, the centroid is computed in terms of area. If an empty geometry is supplied, an empty GEOMETRYCOLLECTION is returned. If NULL is supplied, NULL is returned. If CIRCULARSTRING or COMPOUNDCURVE are supplied, they are converted to linestring with CurveToLine first, then same than for LINESTRING

For mixed-dimension input, the result is equal to the centroid of the component Geometries of highest dimension (since the lower-dimension geometries contribute zero "weight" to the centroid).

Note that for polygonal geometries the centroid does not necessarily lie in the interior of the polygon. For example, see the diagram below of the centroid of a C-shaped polygon. To construct a point guaranteed to lie in the interior of a polygon use ST_PointOnSurface.

New in 2.3.0 : supports CIRCULARSTRING and COMPOUNDCURVE (using CurveToLine)

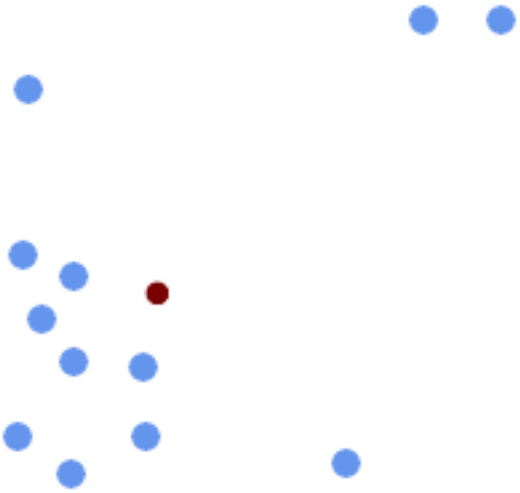
Verfügbarkeit: Mit 2.4.0 wurde die Unterstützung für den geograpischen Datentyp eingeführt.

This method implements the OGC Simple Features Implementation Specification for SQL 1.1.

This method implements the SQL/MM specification. SQL-MM 3: 8.1.4, 9.5.5

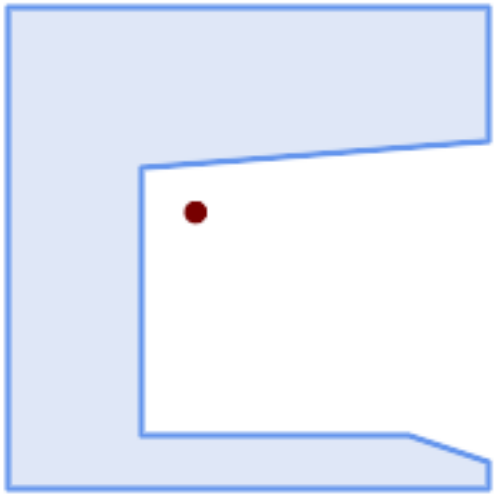
Beispiele

In the following illustrations the red dot is the centroid of the source geometry.

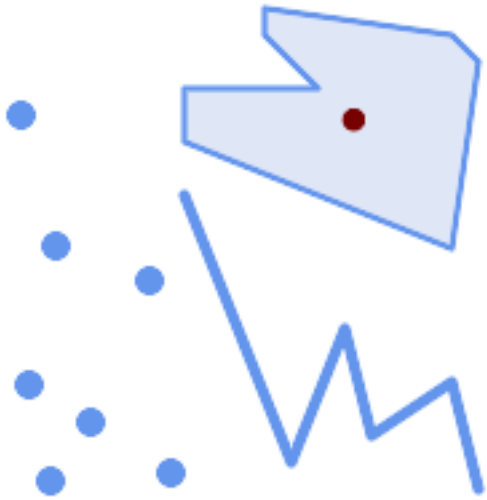


Geometrischer Schwerpunkt eines `MULTIPOINT`

Geometrischer Schwerpunkt eines `LINESTRING`



Geometrischer Schwerpunkt eines `POLYGON`



Der geometrische Schwerpunkt einer `GEOMETRYCOLLECTION`

```
SELECT ST_AsText(ST_Centroid('MULTIPOINT ( -1 0, -1 2, -1 3, -1 4, -1 7, 0 1, 0 3, 1 1, 2  ↩
    0, 6 0, 7 8, 9 8, 10 6 )'));
               st_astext
-----------------------------------------
 POINT(2.30769230769231 3.30769230769231)
(1 row)

SELECT ST_AsText(ST_centroid(g))
FROM  ST_GeomFromText('CIRCULARSTRING(0 2, -1 1,0 0, 0.5 0, 1 0, 2 1, 1 2, 0.5 2, 0 2)')   ↩
    AS g ;
-----------------------------------------
POINT(0.5 1)
```

```
SELECT ST_AsText(ST_centroid(g))
FROM  ST_GeomFromText('COMPOUNDCURVE(CIRCULARSTRING(0 2, -1 1,0 0),(0 0, 0.5 0, 1 0), ↩
    CIRCULARSTRING( 1 0, 2 1, 1 2),(1 2, 0.5 2, 0 2))' ) AS g;
----------------------------------------
POINT(0.5 1)
```

Siehe auch

ST_PointOnSurface, ST_GeometricMedian

7.14.4 ST_ChaikinSmoothing

ST_ChaikinSmoothing — Returns a smoothed version of a geometry, using the Chaikin algorithm

Synopsis

geometry **ST_ChaikinSmoothing**(geometry geom, integer nIterations = 1, boolean preserveEndPoints = false);

Beschreibung

Smoothes a linear or polygonal geometry using Chaikin's algorithm. The degree of smoothing is controlled by the `nIterations` parameter. On each iteration, each interior vertex is replaced by two vertices located at 1/4 of the length of the line segments before and after the vertex. A reasonable degree of smoothing is provided by 3 iterations; the maximum is limited to 5.

If `preserveEndPoints` is true, the endpoints of Polygon rings are not smoothed. The endpoints of LineStrings are always preserved.

Note
The number of vertices doubles with each iteration, so the result geometry may have many more points than the input. To reduce the number of points use a simplification function on the result (see ST_Simplify, ST_SimplifyPreserveTopology and ST_SimplifyVW).

The result has interpolated values for the Z and M dimenions when present.

This function supports 3d and will not drop the z-index.

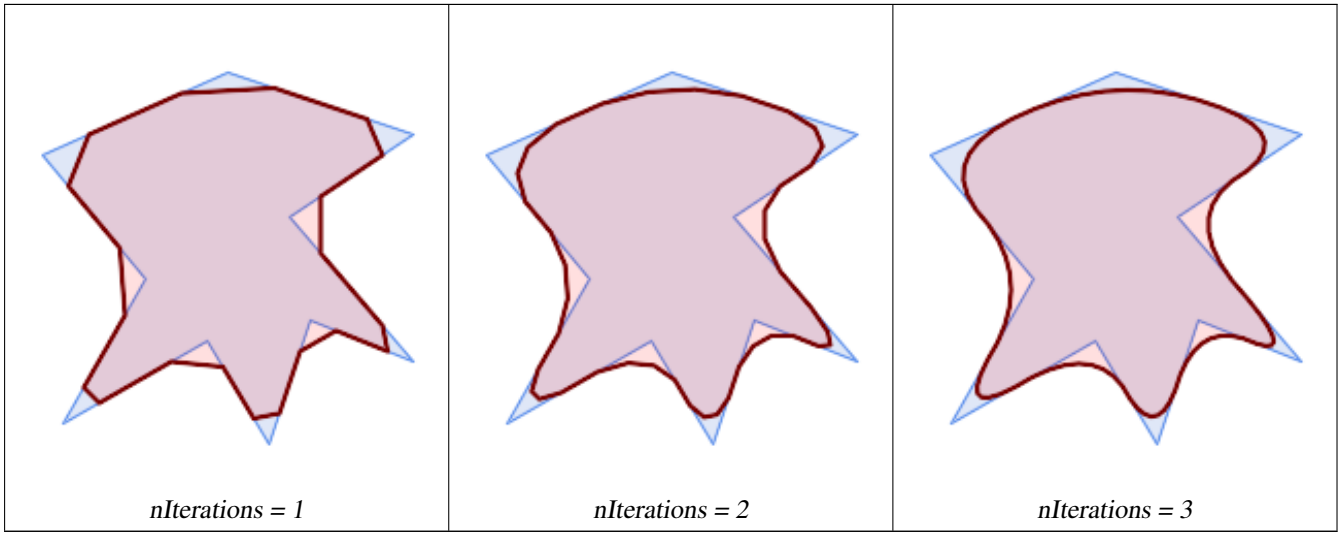
Verfügbarkeit: 2.5.0

Beispiele

Smoothing a triangle:

```
SELECT ST_AsText(ST_ChaikinSmoothing(geom)) smoothed
FROM (SELECT  'POLYGON((0 0, 8 8, 0 16, 0 0))'::geometry geom) AS foo;

                  smoothed
&#x2500;&#x2500;&#x2500;&#x2500;&#x2500;&#x2500;&#x2500;&#x2500;&#x2500;&#x2500;&#x2500;&#x2500;&#x2
 POLYGON((2 2,6 6,6 10,2 14,0 12,0 4,2 2))
```

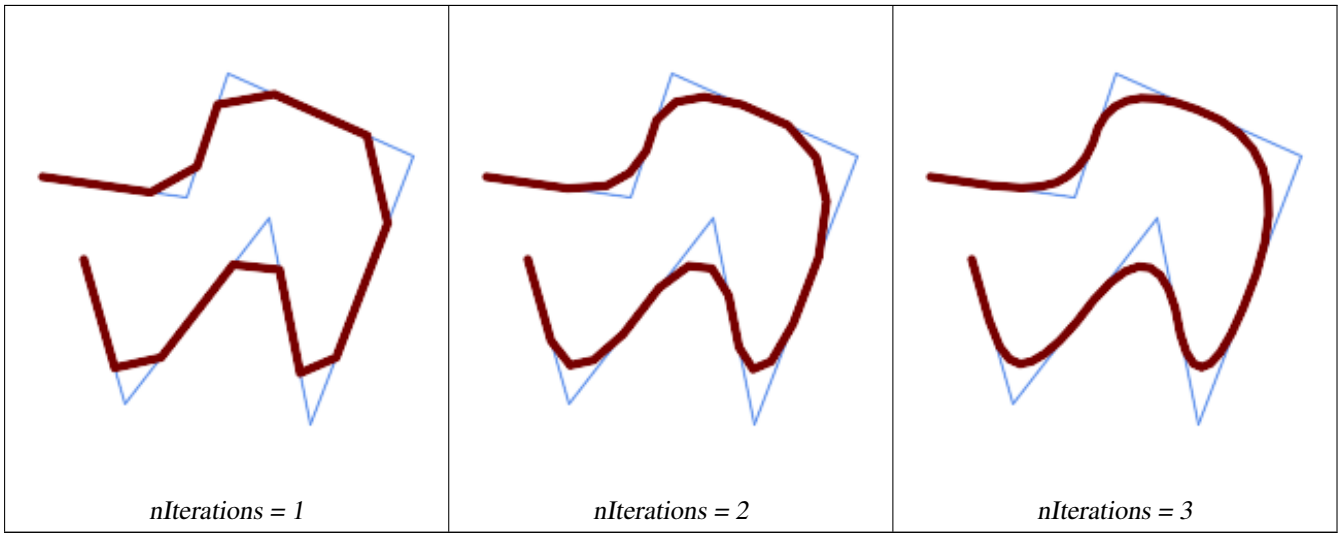
Smoothing a Polygon using 1, 2 and 3 iterations:



nIterations = 1 | *nIterations = 2* | *nIterations = 3*

```
SELECT ST_ChaikinSmoothing(
            'POLYGON ((20 20, 60 90, 10 150, 100 190, 190 160, 130 120, 190 50, 140 70, 120 ↩
                10, 90 60, 20 20))',
            generate_series(1, 3) );
```

Smoothing a LineString using 1, 2 and 3 iterations:



nIterations = 1 | *nIterations = 2* | *nIterations = 3*

```
SELECT ST_ChaikinSmoothing(
            'LINESTRING (10 140, 80 130, 100 190, 190 150, 140 20, 120 120, 50 30, 30 100) ↩
                ',
            generate_series(1, 3) );
```

Siehe auch

ST_Simplify, ST_SimplifyPreserveTopology, ST_SimplifyVW

7.14.5 ST_ConcaveHull

ST_ConcaveHull — Computes a possibly concave geometry that contains all input geometry vertices

Synopsis

geometry **ST_ConcaveHull**(geometry param_geom, float param_pctconvex, boolean param_allow_holes = false);

Beschreibung

A concave hull is a (usually) concave geometry which contains the input, and whose vertices are a subset of the input vertices. In the general case the concave hull is a Polygon. The concave hull of two or more collinear points is a two-point LineString. The concave hull of one or more identical points is a Point. The polygon will not contain holes unless the optional `param_allow_holes` argument is specified as true.

One can think of a concave hull as "shrink-wrapping" a set of points. This is different to the convex hull, which is more like wrapping a rubber band around the points. A concave hull generally has a smaller area and represents a more natural boundary for the input points.

The `param_pctconvex` controls the concaveness of the computed hull. A value of 1 produces the convex hull. Values between 1 and 0 produce hulls of increasing concaveness. A value of 0 produces a hull with maximum concaveness (but still a single polygon). Choosing a suitable value depends on the nature of the input data, but often values between 0.3 and 0.1 produce reasonable results.

Note
Technically, the `param_pctconvex` determines a length as a fraction of the difference between the longest and shortest edges in the Delaunay Triangulation of the input points. Edges longer than this length are "eroded" from the triangulation. The triangles remaining form the concave hull.

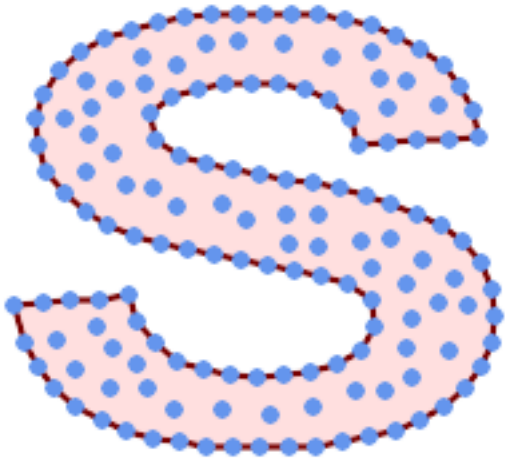
For point and linear inputs, the hull will enclose all the points of the inputs. For polygonal inputs, the hull will enclose all the points of the input *and also* all the areas covered by the input. If you want a point-wise hull of a polygonal input, convert it to points first using ST_Points.

This is not an aggregate function. To compute the concave hull of a set of geometries use ST_Collect (e.g. `ST_ConcaveHull( ST_Collect( geom ), 0.80)`.

Verfügbarkeit: 2.0.0

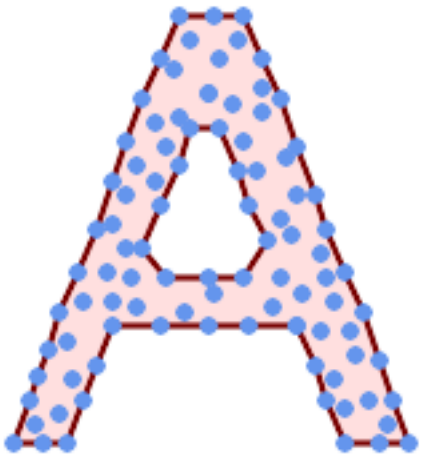
Enhanced: 3.3.0, GEOS native implementation enabled for GEOS 3.11+

Beispiele



Konvexe Hülle eines MultiLineString und eines MultiPoint

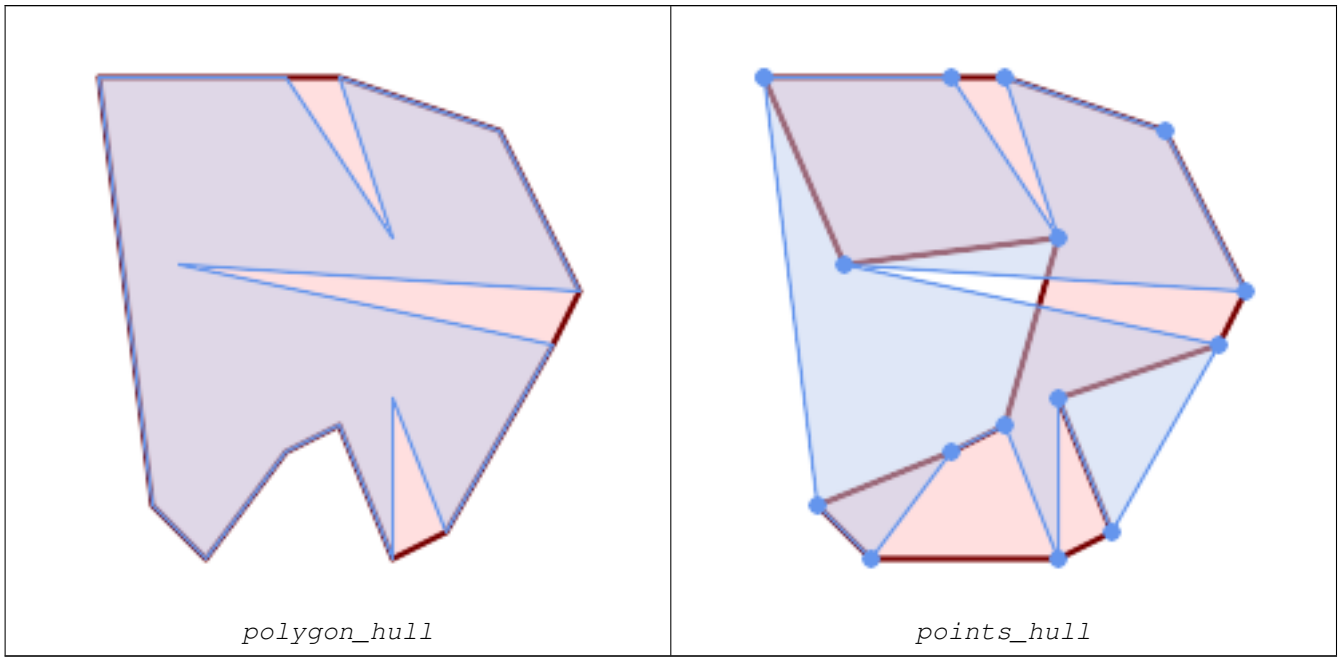
```
SELECT ST_AsText( ST_ConcaveHull(
        'MULTIPOINT ((10 72), (53 76), (56 66), (63 58), (71 51), (81 48), (91 46), (101 ↩
            45), (111 46), (121 47), (131 50), (140 55), (145 64), (144 74), (135 80), (125 ↩
            83), (115 85), (105 87), (95 89), (85 91), (75 93), (65 95), (55 98), (45 102), ↩
            (37 107), (29 114), (22 122), (19 132), (18 142), (21 151), (27 160), (35 167), ↩
            (44 172), (54 175), (64 178), (74 180), (84 181), (94 181), (104 181), (114 181) ↩
            , (124 181), (134 179), (144 177), (153 173), (162 168), (171 162), (177 154), ↩
            (182 145), (184 135), (139 132), (136 142), (128 149), (119 153), (109 155), (99 ↩
             155), (89 155), (79 153), (69 150), (61 144), (63 134), (72 128), (82 125), (92 ↩
             123), (102 121), (112 119), (122 118), (132 116), (142 113), (151 110), (161 ↩
            106), (170 102), (178 96), (185 88), (189 78), (190 68), (189 58), (185 49), ↩
            (179 41), (171 34), (162 29), (153 25), (143 23), (133 21), (123 19), (113 19), ↩
            (102 19), (92 19), (82 19), (72 21), (62 22), (52 25), (43 29), (33 34), (25 41) ↩
            , (19 49), (14 58), (21 73), (31 74), (42 74), (173 134), (161 134), (150 133), ↩
            (97 104), (52 117), (157 156), (94 171), (112 106), (169 73), (58 165), (149 40) ↩
            , (70 33), (147 157), (48 153), (140 96), (47 129), (173 55), (144 86), (159 67) ↩
            , (150 146), (38 136), (111 170), (124 94), (26 59), (60 41), (71 162), (41 64), ↩
             (88 110), (122 34), (151 97), (157 56), (39 146), (88 33), (159 45), (47 56), ↩
            (138 40), (129 165), (33 48), (106 31), (169 147), (37 122), (71 109), (163 89), ↩
             (37 156), (82 170), (180 72), (29 142), (46 41), (59 155), (124 106), (157 80), ↩
             (175 82), (56 50), (62 116), (113 95), (144 167))',
         0.1 ) );
---st_astext--
POLYGON ((18 142, 21 151, 27 160, 35 167, 44 172, 54 175, 64 178, 74 180, 84 181, 94 181, ↩
    104 181, 114 181, 124 181, 134 179, 144 177, 153 173, 162 168, 171 162, 177 154, 182 ↩
    145, 184 135, 173 134, 161 134, 150 133, 139 132, 136 142, 128 149, 119 153, 109 155, 99 ↩
     155, 89 155, 79 153, 69 150, 61 144, 63 134, 72 128, 82 125, 92 123, 102 121, 112 119, ↩
    122 118, 132 116, 142 113, 151 110, 161 106, 170 102, 178 96, 185 88, 189 78, 190 68, ↩
    189 58, 185 49, 179 41, 171 34, 162 29, 153 25, 143 23, 133 21, 123 19, 113 19, 102 19, ↩
    92 19, 82 19, 72 21, 62 22, 52 25, 43 29, 33 34, 25 41, 19 49, 14 58, 10 72, 21 73, 31 ↩
    74, 42 74, 53 76, 56 66, 63 58, 71 51, 81 48, 91 46, 101 45, 111 46, 121 47, 131 50, 140 ↩
     55, 145 64, 144 74, 135 80, 125 83, 115 85, 105 87, 95 89, 85 91, 75 93, 65 95, 55 98, ↩
    45 102, 37 107, 29 114, 22 122, 19 132, 18 142))
```



Konvexe Hülle eines MultiLineString und eines MultiPoint

```
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))
```



polygon_hull | *points_hull*

Comparing a concave hull of a Polygon to the concave hull of the constituent points. The hull respects the boundary of the polygon, whereas the points-based hull does not.

```
WITH data(geom) AS (VALUES
   ('POLYGON ((10 90, 39 85, 61 79, 50 90, 80 80, 95 55, 25 60, 90 45, 70 16, 63 38, 60 10, ↩
       50 30, 43 27, 30 10, 20 20, 10 90))'::geometry)
)
SELECT  ST_ConcaveHull( geom,            0.1) AS polygon_hull,
        ST_ConcaveHull( ST_Points(geom), 0.1) AS points_hull
    FROM data;
```

Verwendung mit ST_Collect zur Berechnung der konvexen Hüllen einer Geometrie.

```
-- Compute estimate of infected area based on point observations
SELECT disease_type,
    ST_ConcaveHull( ST_Collect(obs_pnt), 0.3 ) AS geom
  FROM disease_obs
  GROUP BY disease_type;
```

Siehe auch

ST_ConvexHull, ST_Collect, ST_AlphaShape, ST_OptimalAlphaShape

7.14.6 ST_ConvexHull

ST_ConvexHull — Berechnet die konvexe Hülle einer Geometrie.

Synopsis

geometry **ST_ConvexHull**(geometry geomA);

Beschreibung

Berechnet die konvexe Hülle einer Geometrie. Die konvexe Hülle stellt die kleinste konvexe Geometrie dar, welche die gesamte Geometrie einschließt.

One can think of the convex hull as the geometry obtained by wrapping an rubber band around a set of geometries. This is different from a concave hull which is analogous to "shrink-wrapping" the geometries. A convex hull is often used to determine an affected area based on a set of point observations.

In the general case the convex hull is a Polygon. The convex hull of two or more collinear points is a two-point LineString. The convex hull of one or more identical points is a Point.

This is not an aggregate function. To compute the convex hull of a set of geometries, use ST_Collect to aggregate them into a geometry collection (e.g. `ST_ConvexHull(ST_Collect(geom))`.

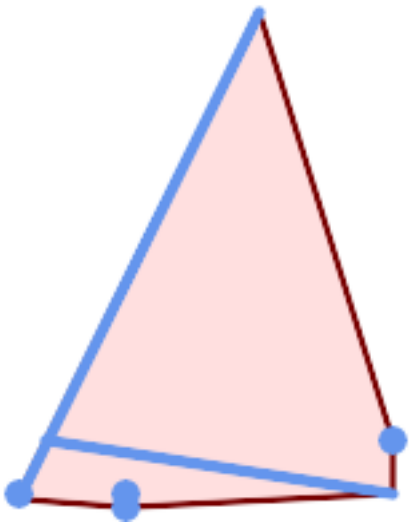
Wird durch das GEOS Modul ausgeführt

This method implements the OGC Simple Features Implementation Specification for SQL 1.1. s2.1.1.3

This method implements the SQL/MM specification. SQL-MM IEC 13249-3: 5.1.16

This function supports 3d and will not drop the z-index.

Beispiele



Konvexe Hülle eines MultiLineString und eines MultiPoint

```
SELECT ST_AsText(ST_ConvexHull(
    ST_Collect(
        ST_GeomFromText('MULTILINESTRING((100 190,10 8),(150 10, 20 30))'),
            ST_GeomFromText('MULTIPOINT(50 5, 150 30, 50 10, 10 10)')
            )) );
---st_astext--
POLYGON((50 5,10 8,10 10,100 190,150 30,150 10,50 5))
```

Verwendung mit ST_Collect zur Berechnung der konvexen Hüllen einer Geometrie.

```
--Get estimate of infected area based on point observations
SELECT d.disease_type,
    ST_ConvexHull(ST_Collect(d.geom)) As geom
    FROM disease_obs As d
    GROUP BY d.disease_type;
```

Siehe auch

ST_Collect, ST_ConcaveHull, ST_MinimumBoundingCircle

7.14.7 ST_DelaunayTriangles

ST_DelaunayTriangles — Returns the Delaunay triangulation of the vertices of a geometry.

Synopsis

geometry **ST_DelaunayTriangles**(geometry g1, float tolerance = 0.0, int4 flags = 0);

Beschreibung

Computes the Delaunay triangulation of the vertices of the input geometry. The optional `tolerance` can be used to snap nearby input vertices together, which improves robustness in some situations. The result geometry is bounded by the convex hull of the input vertices. The result geometry representation is determined by the `flags` code:

- `0` - a GEOMETRYCOLLECTION of triangular POLYGONs (default)
- `1` - a MULTILINESTRING of the edges of the triangulation
- `2` - A TIN of the triangulation

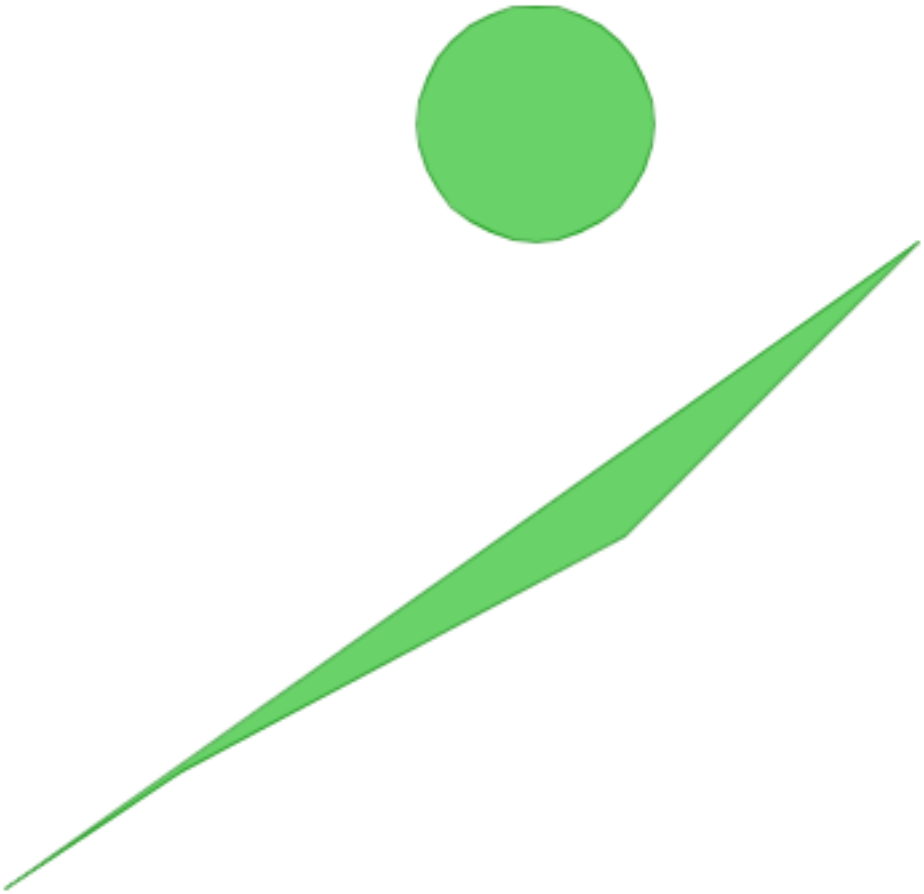
Wird vom GEOS Modul ausgeführt

Verfügbarkeit: 2.1.0

This function supports 3d and will not drop the z-index.

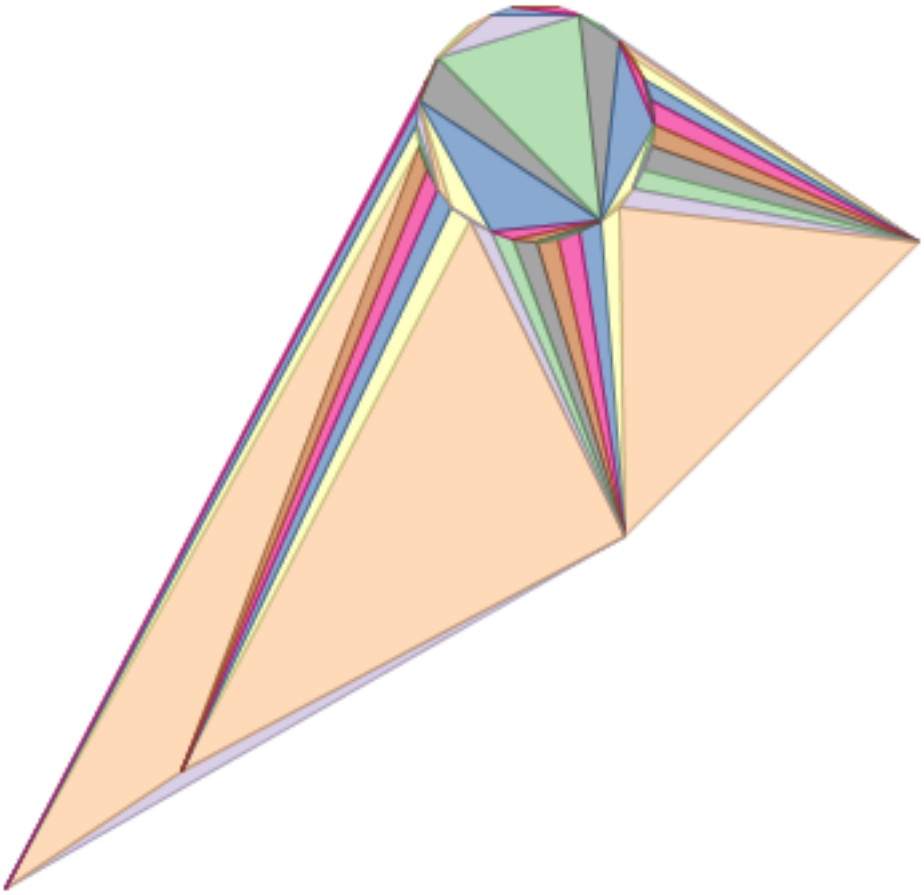
This function supports Triangles and Triangulated Irregular Network Surfaces (TIN).

Beispiele



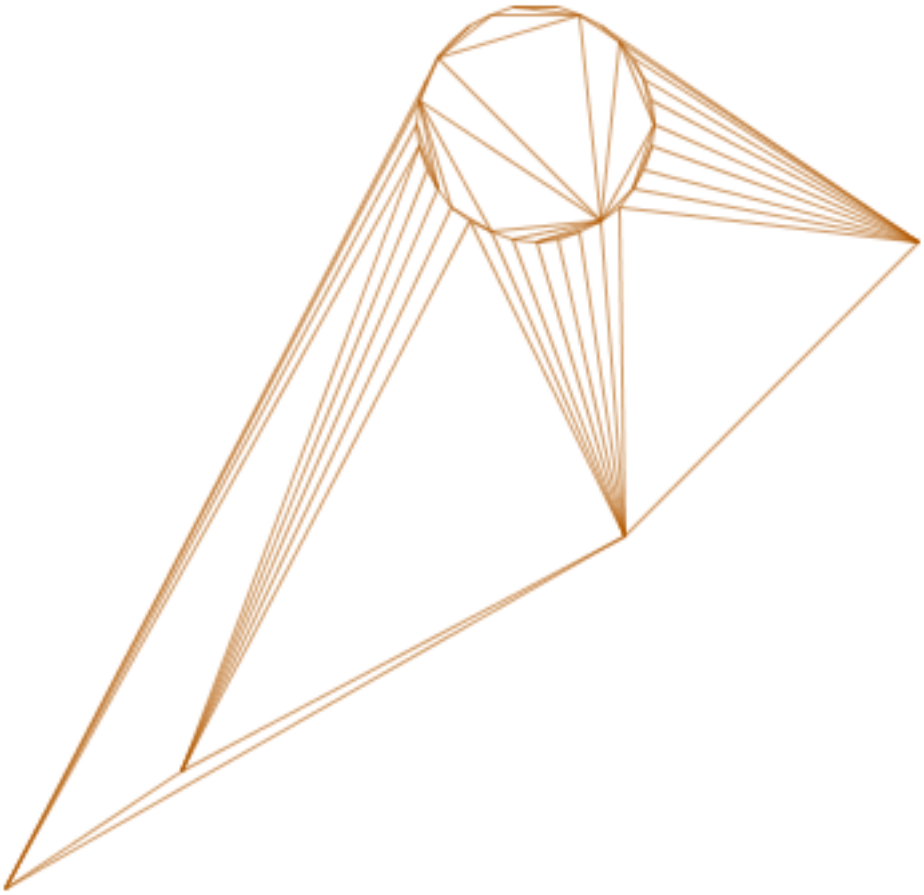
Ursprüngliche Polygone

```
our original geometry
    ST_Union(ST_GeomFromText('POLYGON((175 150, 20 40,
            50 60, 125 100, 175 150))'),
        ST_Buffer(ST_GeomFromText('POINT(110 170)'), 20)
        )
```



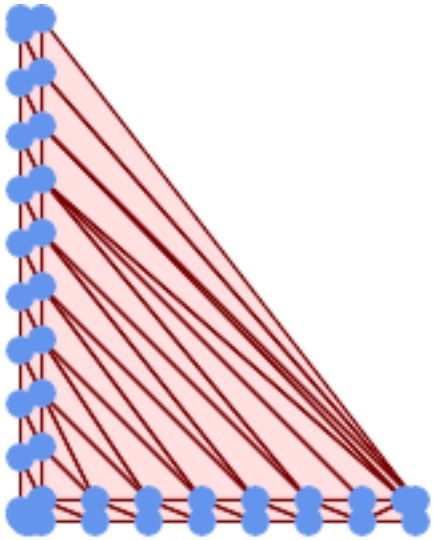
ST_DelaunayTriangles von 2 Polygonen: delaunay triangulierte Polygone, jedes der Dreiecke ist in einer eigenen Farbe dargestellt

```
geometries overlaid multilinestring triangles

SELECT
    ST_DelaunayTriangles(
        ST_Union(ST_GeomFromText('POLYGON((175 150, 20 40,
            50 60, 125 100, 175 150))'),
        ST_Buffer(ST_GeomFromText('POINT(110 170)'), 20)
        ))
     As  dtriag;
```



-- Delaunay-Dreiecke als 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 Dreiecke von 45 Punkten als 55 Dreieckspolygone

```
this produces a table of 42 points that form an L shape

SELECT (ST_DumpPoints(ST_GeomFromText(
'MULTIPOINT(14 14,34 14,54 14,74 14,94 14,114 14,134 14,
150 14,154 14,154 6,134 6,114 6,94 6,74 6,54 6,34 6,
14 6,10 6,8 6,7 7,6 8,6 10,6 30,6 50,6 70,6 90,6 110,6 130,
6 150,6 170,6 190,6 194,14 194,14 174,14 154,14 134,14 114,
14 94,14 74,14 54,14 34,14 14)'))).geom
    INTO TABLE l_shape;

output as individual polygon triangles

SELECT ST_AsText((ST_Dump(geom)).geom) As wkt
FROM ( SELECT ST_DelaunayTriangles(ST_Collect(geom)) As geom
FROM l_shape) As foo;


wkt

POLYGON((6 194,6 190,14 194,6 194))
POLYGON((14 194,6 190,14 174,14 194))
POLYGON((14 194,14 174,154 14,14 194))
POLYGON((154 14,14 174,14 154,154 14))
POLYGON((154 14,14 154,150 14,154 14))
POLYGON((154 14,150 14,154 6,154 14))
```

Example using vertices with Z values.

```
3D multipoint

SELECT ST_AsText(ST_DelaunayTriangles(ST_GeomFromText(
         'MULTIPOINT Z(14 14 10, 150 14 100,34 6 25, 20 10 150)'))) As wkt;


wkt

GEOMETRYCOLLECTION Z (POLYGON Z ((14 14 10,20 10 150,34 6 25,14 14 10))
 ,POLYGON Z ((14 14 10,34 6 25,150 14 100,14 14 10)))
```

Siehe auch

ST_VoronoiPolygons, ST_TriangulatePolygon, ST_ConstrainedDelaunayTriangles, ST_VoronoiLines, ST_ConvexHull

7.14.8 ST_FilterByM

ST_FilterByM — Removes vertices based on their M value

Synopsis

geometry **ST_FilterByM**(geometry geom, double precision min, double precision max = null, boolean returnM = false);

Beschreibung

Filters out vertex points based on their M-value. Returns a geometry with only vertex points that have a M-value larger or equal to the min value and smaller or equal to the max value. If max-value argument is left out only min value is considered. If fourth argument is left out the m-value will not be in the resulting geometry. If resulting geometry have too few vertex points left for its geometry type an empty geometry will be returned. In a geometry collection geometries without enough points will just be left out silently.

This function is mainly intended to be used in conjunction with ST_SetEffectiveArea. ST_EffectiveArea sets the effective area of a vertex in its m-value. With ST_FilterByM it then is possible to get a simplified version of the geometry without any calculations, just by filtering

Note
There is a difference in what ST_SimplifyVW returns when not enough points meet the criteria compared to ST_FilterByM. ST_SimplifyVW returns the geometry with enough points while ST_FilterByM returns an empty geometry

Note
Note that the returned geometry might be invalid

Note
This function returns all dimensions, including the Z and M values

Verfügbarkeit: 2.5.0

Beispiele

Eine gefilterte Linie

```
SELECT ST_AsText(ST_FilterByM(geom,30)) simplified
FROM (SELECT  ST_SetEffectiveArea('LINESTRING(5 2, 3 8, 6 20, 7 25, 10 10)'::geometry) geom ↩
    ) As foo;

result

         simplified
----------------------------
 LINESTRING(5 2,7 25,10 10)
```

Siehe auch

ST_SetEffectiveArea, ST_SimplifyVW

7.14.9 ST_GeneratePoints

ST_GeneratePoints — Generates random points contained in a Polygon or MultiPolygon.

Synopsis

geometry **ST_GeneratePoints**(g geometry , npoints integer);
geometry **ST_GeneratePoints**(geometry g , integer npoints , integer seed = 0);

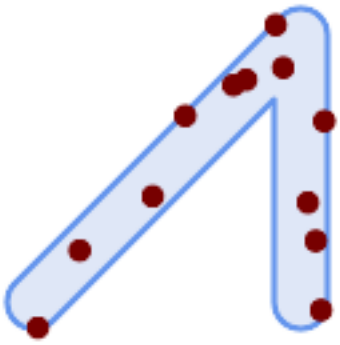
Beschreibung

ST_GeneratePoints generates a given number of pseudo-random points which lie within the input area. The optional `seed` is used to regenerate a deterministic sequence of points, and must be greater than zero.

Verfügbarkeit: 2.3.0

Erweiterung: mit 3.0.0 wurde das Argument "seed" hinzugefügt

Beispiele



Die mit einem zufallsbedingten "seed" Wert von 1996 erzeugten 12 Punkte auf dem Ursprungspolygon

```
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;
```

7.14.10 ST_GeometricMedian

ST_GeometricMedian — Gibt den geometrischen Median eines Mehrfachpunktes zurück.

Synopsis

geometry **ST_GeometricMedian** (geometry geom, float8 tolerance = NULL, int max_iter = 10000, boolean fail_if_not_converged = false);

Beschreibung

Computes the approximate geometric median of a MultiPoint geometry using the Weiszfeld algorithm. The geometric median is the point minimizing the sum of distances to the input points. It provides a centrality measure that is less sensitive to outlier points than the centroid (center of mass).

The algorithm iterates until the distance change between successive iterations is less than the supplied `tolerance` parameter. If this condition has not been met after `max_iterations` iterations, the function produces an error and exits, unless `fail_if_not_converged` is set to `false` (the default).

If a `tolerance` argument is not provided, the tolerance value is calculated based on the extent of the input geometry.

If present, the input point M values are interpreted as their relative weights.

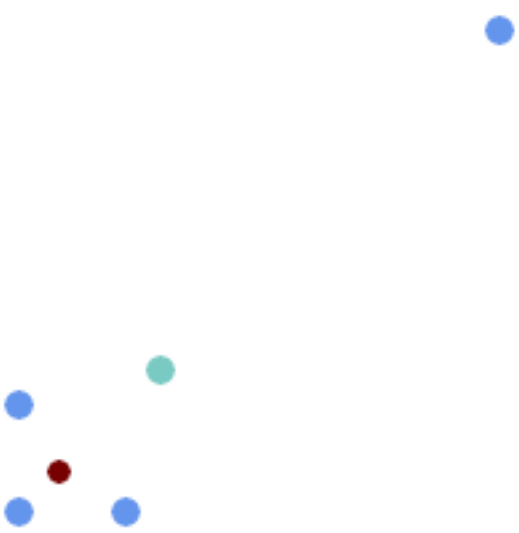
Verfügbarkeit: 2.3.0

Erweiterung: 2.5.0 Unterstützung für M zur Gewichtung nach Punkten.

This function supports 3d and will not drop the z-index.

This function supports M coordinates.

Beispiele



Comparison of the geometric median (red) and centroid (turquoise) of a MultiPoint.

```
WITH test AS (
SELECT 'MULTIPOINT((10 10), (10 40), (40 10), (190 190))'::geometry geom)
SELECT
  ST_AsText(ST_Centroid(geom)) centroid,
  ST_AsText(ST_GeometricMedian(geom)) median
FROM test;

      centroid      |                 median
--------------------+----------------------------------------
   POINT(62.5 62.5) | POINT(25.01778421249728 25.01778421249728)
(1 row)
```

Siehe auch

ST_Centroid

7.14.11 ST_LineMerge

ST_LineMerge — Return the lines formed by sewing together a MultiLineString.

Synopsis

geometry **ST_LineMerge**(geometry amultilinestring);
geometry **ST_LineMerge**(geometry amultilinestring, boolean directed);

Beschreibung

Returns a LineString or MultiLineString formed by joining together the line elements of a MultiLineString. Lines are joined at their endpoints at 2-way intersections. Lines are not joined across intersections of 3-way or greater degree.

If **directed** is TRUE, then ST_LineMerge will not change point order within LineStrings, so lines with opposite directions will not be merged

Note
Only use with MultiLineString/LineStrings. Other geometry types return an empty GeometryCollection

Wird vom GEOS Modul ausgeführt

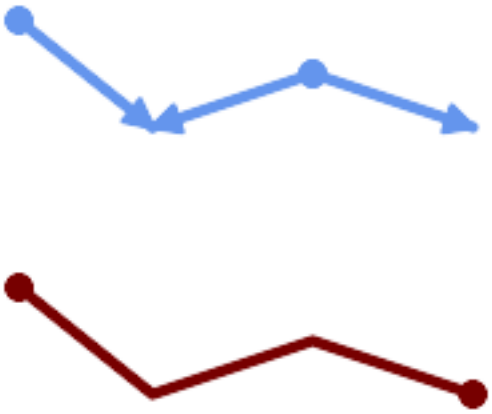
Enhanced: 3.3.0 accept a directed parameter.

Requires GEOS >= 3.11.0 to use the directed parameter.

Verfügbarkeit: 1.1.0

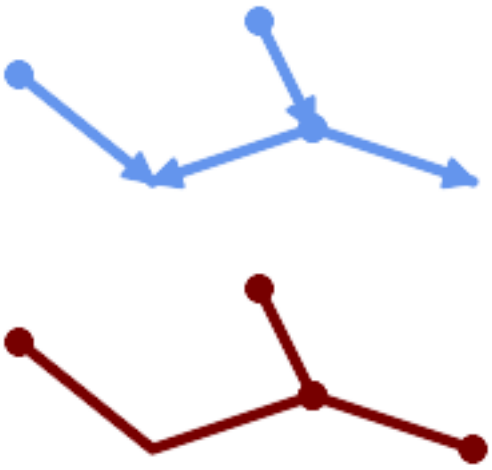
Warning
This function strips the M dimension.

Beispiele



Merging lines with different orientation.

```
SELECT ST_AsText(ST_LineMerge(
'MULTILINESTRING((10 160, 60 120), (120 140, 60 120), (120 140, 180 120))'
                ));
-------------------------------------------
 LINESTRING(10 160,60 120,120 140,180 120)
```

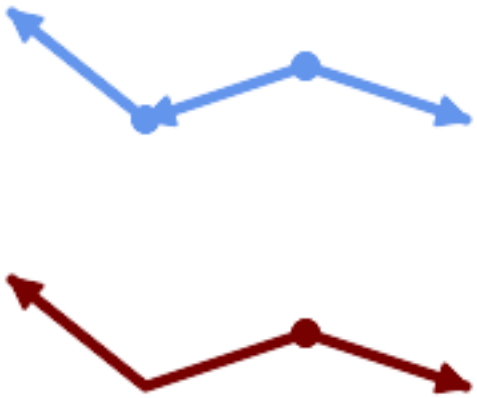


Lines are not merged across intersections with degree > 2.

```
SELECT ST_AsText(ST_LineMerge(
'MULTILINESTRING((10 160, 60 120), (120 140, 60 120), (120 140, 180 120), (100 180, 120  ↩
    140))'
                ));
-------------------------------------------
 MULTILINESTRING((10 160,60 120,120 140),(100 180,120 140),(120 140,180 120))
```

If merging is not possible due to non-touching lines, the original MultiLineString is returned.

```
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))
```



Lines with opposite directions are not merged if directed = TRUE.

```
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))
```

Example showing Z-dimension handling.

```
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)
```

Siehe auch

ST_Segmentize, ST_LineSubstring

7.14.12 ST_MaximumInscribedCircle

ST_MaximumInscribedCircle — Berechnet die konvexe Hülle einer Geometrie.

Synopsis

(geometry, geometry, double precision) **ST_MaximumInscribedCircle**(geometry geom);

Beschreibung

Finds the largest circle that is contained within a (multi)polygon, or which does not overlap any lines and points. Returns a record with fields:

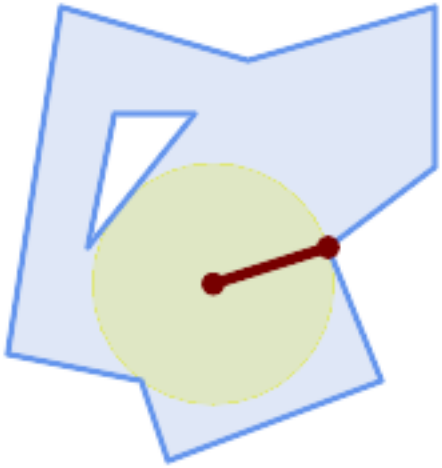
- `center` - center point of the circle
- `nearest` - a point on the geometry nearest to the center
- `radius` - radius of the circle

For polygonal inputs, the circle is inscribed within the boundary rings, using the internal rings as boundaries. For linear and point inputs, the circle is inscribed within the convex hull of the input, using the input lines and points as further boundaries.

Availability: 3.1.0.

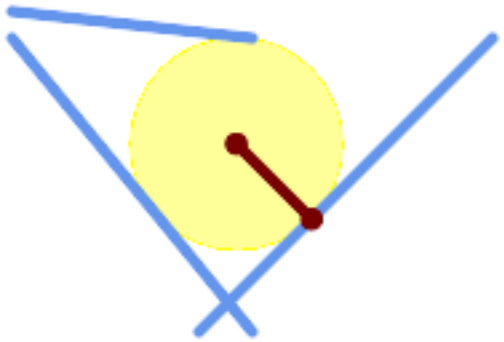
Requires GEOS >= 3.9.0.

Beispiele



Maximum inscribed circle of a polygon. Center, nearest point, and radius are returned.

```
SELECT radius, ST_AsText(center) AS center, ST_AsText(nearest) AS nearest
    FROM ST_MaximumInscribedCircle(
        'POLYGON ((40 180, 110 160, 180 180, 180 120, 140 90, 160 40, 80 10, 70 40, 20 50,  ↩
            40 180),
        (60 140, 50 90, 90 140, 60 140))');

     radius      |           center           |    nearest
-----------------+----------------------------+---------------
 45.165845650018 | POINT(96.953125 76.328125) | POINT(140 90)
```



Maximum inscribed circle of a multi-linestring. Center, nearest point, and radius are returned.

Siehe auch

ST_MinimumBoundingRadius, ST_LargestEmptyCircle

7.14.13 ST_LargestEmptyCircle

ST_LargestEmptyCircle — Computes the largest circle not overlapping a geometry.

Synopsis

(geometry, geometry, double precision) **ST_LargestEmptyCircle**(geometry geom, double precision tolerance=0.0, geometry boundary=POINT EMPTY);

Beschreibung

Finds the largest circle which does not overlap a set of point and line obstacles. (Polygonal geometries may be included as obstacles, but only their boundary lines are used.) The center of the circle is constrained to lie inside a polygonal boundary, which by default is the convex hull of the input geometry. The circle center is the point in the interior of the boundary which has the farthest distance from the obstacles. The circle itself is provided by the center point and a nearest point lying on an obstacle detemining the circle radius.

The circle center is determined to a given accuracy specified by a distance tolerance, using an iterative algorithm. If the accuracy distance is not specified a reasonable default is used.

Returns a record with fields:

- `center` - center point of the circle
- `nearest` - a point on the geometry nearest to the center
- `radius` - radius of the circle

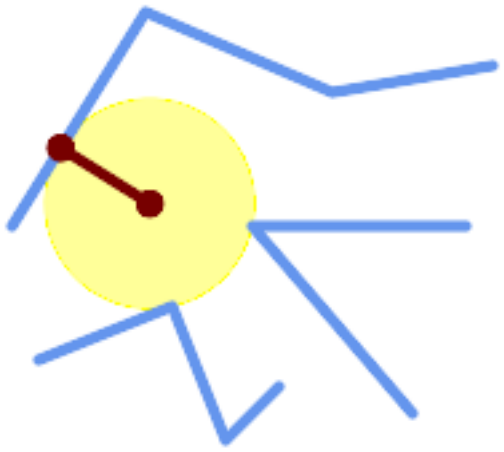
To find the largest empty circle in the interior of a polygon, see ST_MaximumInscribedCircle.

Availability: 3.4.0.

Requires GEOS >= 3.9.0.

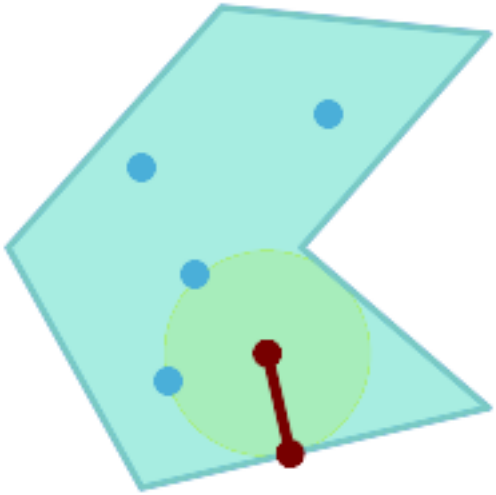
Beispiele

```
SELECT radius,
       ST_AsText(center) AS center,
       ST_AsText(nearest) AS nearest
  FROM ST_LargestEmptyCircle(
        'MULTILINESTRING (
          (10 100, 60 180, 130 150, 190 160),
          (20 50, 70 70, 90 20, 110 40),
          (160 30, 100 100, 180 100))');
```



Largest Empty Circle within a set of lines.

```
SELECT radius,
       ST_AsText(center) AS center,
       ST_AsText(nearest) AS nearest
  FROM ST_LargestEmptyCircle(
         St_Collect(
           'MULTIPOINT ((70 50), (60 130), (130 150), (80 90))',
           'POLYGON ((90 190, 10 100, 60 10, 190 40, 120 100, 190 180, 90 190))'),
         'POLYGON ((90 190, 10 100, 60 10, 190 40, 120 100, 190 180, 90 190))'
       );
```



Largest Empty Circle within a set of points, constrained to lie in a polygon. The constraint polygon boundary must be included as an obstacle, as well as specified as the constraint for the circle center.

Siehe auch

ST_MinimumBoundingRadius

7.14.14 ST_MinimumBoundingCircle

ST_MinimumBoundingCircle — Returns the smallest circle polygon that contains a geometry.

Synopsis

geometry **ST_MinimumBoundingCircle**(geometry geomA, integer num_segs_per_qt_circ=48);

Beschreibung

Returns the smallest circle polygon that contains a geometry.

Note
Der Kreis wird standardmäßig durch ein Polygon mit 48 Segmenten pro Viertelkreis angenähert. Da das Polygon eine Annäherung an den minimalen Umgebungskreis ist, können einige Punkte der Eingabegeometrie nicht in dem Polygon enthalten sein. Die Annäherung kann durch Erhöhung der Anzahl der Segmente mit geringen Einbußen bei der Rechenleistung verbessert werden. Bei Anwendungen wo eine polygonale Annäherung nicht ausreicht, kann ST_MinimumBoundingRadius verwendet werden.

Use with ST_Collect to get the minimum bounding circle of a set of geometries.

To compute two points lying on the mimimum circle (the "maximum diameter") use ST_LongestLine.

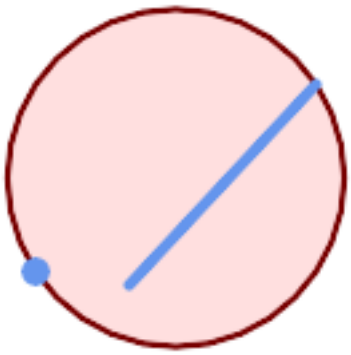
Das Verhältnis zwischen der Fläche des Polygons und der Fläche ihres kleinstmöglichen Umgebungskreises wird öfter als Roeck Test bezeichnet.

Wird vom GEOS Modul ausgeführt

Verfügbarkeit: 1.4.0

Beispiele

```
SELECT d.disease_type,
    ST_MinimumBoundingCircle(ST_Collect(d.geom)) As geom
    FROM disease_obs As d
    GROUP BY d.disease_type;
```



Minimaler Umgebungskreis eines Punktes und eines Linienzuges. Verwendet 8 Segmente um einen Viertelkreis anzunähern.

```
SELECT ST_AsText(ST_MinimumBoundingCircle(
        ST_Collect(
            ST_GeomFromText('LINESTRING(55 75,125 150)'),
                ST_Point(20, 80)), 8
                )) As wktmbc;
wktmbc
-----------
POLYGON((135.59714732062 115,134.384753327498 102.690357210921,130.79416296937  ↩
    90.8537670908995,124.963360620072 79.9451031602111,117.116420743937  ↩
    70.3835792560632,107.554896839789 62.5366393799277,96.6462329091006  ↩
    56.70583703063,84.8096427890789 53.115246672502,72.5000000000001  ↩
    51.9028526793802,60.1903572109213 53.1152466725019,48.3537670908996  ↩
    56.7058370306299,37.4451031602112 62.5366393799276,27.8835792560632  ↩
    70.383579256063,20.0366393799278 79.9451031602109,14.20583703063  ↩
    90.8537670908993,10.615246672502 102.690357210921,9.40285267938019 115,10.6152466725019  ↩
    127.309642789079,14.2058370306299 139.1462329091,20.0366393799275  ↩
    150.054896839789,27.883579256063 159.616420743937,
37.4451031602108 167.463360620072,48.3537670908992 173.29416296937,60.190357210921  ↩
    176.884753327498,
72.4999999999998 178.09714732062,84.8096427890786 176.884753327498,96.6462329091003  ↩
    173.29416296937,107.554896839789 167.463360620072,
117.116420743937 159.616420743937,124.963360620072 150.054896839789,130.79416296937  ↩
    139.146232909101,134.384753327498 127.309642789079,135.59714732062 115))
```

Siehe auch

ST_Collect, ST_MinimumBoundingRadius, ST_LargestEmptyCircle, ST_LongestLine

7.14.15 ST_MinimumBoundingRadius

ST_MinimumBoundingRadius — Returns the center point and radius of the smallest circle that contains a geometry.

Synopsis

(geometry, double precision) **ST_MinimumBoundingRadius**(geometry geom);

Beschreibung

Computes the center point and radius of the smallest circle that contains a geometry. Returns a record with fields:

- `center` - center point of the circle
- `radius` - radius of the circle

Use with ST_Collect to get the minimum bounding circle of a set of geometries.

To compute two points lying on the mimimum circle (the "maximum diameter") use ST_LongestLine.

Verfügbarkeit: 2.3.0

Beispiele

```
SELECT ST_AsText(center), radius FROM ST_MinimumBoundingRadius('POLYGON((26426 65078,26531 ↩
    65242,26075 65136,26096 65427,26426 65078))');

                st_astext                 |      radius
------------------------------------------+------------------
 POINT(26284.8418027133 65267.1145090825) | 247.436045591407
```

Siehe auch

ST_Collect, ST_MinimumBoundingCircle, ST_LongestLine

7.14.16 ST_OrientedEnvelope

ST_OrientedEnvelope — Returns a minimum-area rectangle containing a geometry.

Synopsis

geometry **ST_OrientedEnvelope**(geometry geom);

Beschreibung

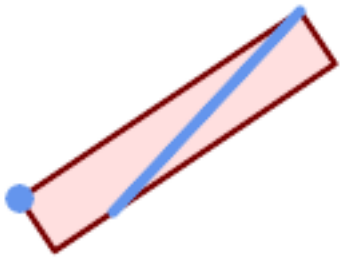
Returns the minimum-area rotated rectangle enclosing a geometry. Note that more than one such rectangle may exist. May return a Point or LineString in the case of degenerate inputs.

Availability: 2.5.0.

Requires GEOS >= 3.6.0.

Beispiele

```
SELECT ST_AsText(ST_OrientedEnvelope('MULTIPOINT ((0 0), (-1 -1), (3 2))'));

                st_astext
                -----------------------------------------------
                POLYGON((3 2,2.88 2.16,-1.12 -0.84,-1 -1,3 2))
```



Die ausgerichtete Einhüllende eines Punktes und einer Linie.

```
SELECT ST_AsText(ST_OrientedEnvelope(
        ST_Collect(
            ST_GeomFromText('LINESTRING(55 75,125 150)'),
                ST_Point(20, 80))
                )) As wktenv;
wktenv
-----------
POLYGON((19.9999999999997 79.9999999999999,33.0769230769229  ↩
    60.3846153846152,138.076923076924 130.384615384616,125.000000000001  ↩
    150.000000000001,19.9999999999997 79.9999999999999))
```

Siehe auch

ST_Envelope ST_MinimumBoundingCircle

7.14.17 ST_OffsetCurve

ST_OffsetCurve — Returns an offset line at a given distance and side from an input line.

Synopsis

geometry **ST_OffsetCurve**(geometry line, float signed_distance, text style_parameters='');

Beschreibung

Return an offset line at a given distance and side from an input line. All points of the returned geometries are not further than the given distance from the input geometry. Useful for computing parallel lines about a center line.

For positive distance the offset is on the left side of the input line and retains the same direction. For a negative distance it is on the right side and in the opposite direction.

Die Einheiten der Entfernung werden in den Einheiten des Koordinatenreferenzsystems gemessen.

Bei einer Puzzlestück-förmigen Geometrie kann die Ausgabe manchmal ein MULTILINESTRING oder EMPTY sein.

Der optionale dritte Parameter ermöglicht es eine Liste von leerzeichengetrennten key=value Paaren anzulegen, um die Berechnungen wie folgt zu optimieren:

- 'quad_segs=#' : Anzahl der Segmente die verwendet werden um einen Viertelkreis anzunähern (standardmäßig 8).
- 'join=round|mitre|bevel' : join style (defaults to "round"). 'miter' kann auch als Synonym für 'mitre' verwendet werden.
- 'mitre_limit=#.#' : Gehrungsobergrenze (beeinflusst nur Gehrungsverbindungen). 'miter_limit' kann auch als Synonym von 'mitre_limit' verwendet werden.

Wird vom GEOS Modul ausgeführt

Verfügbarkeit: 2.0

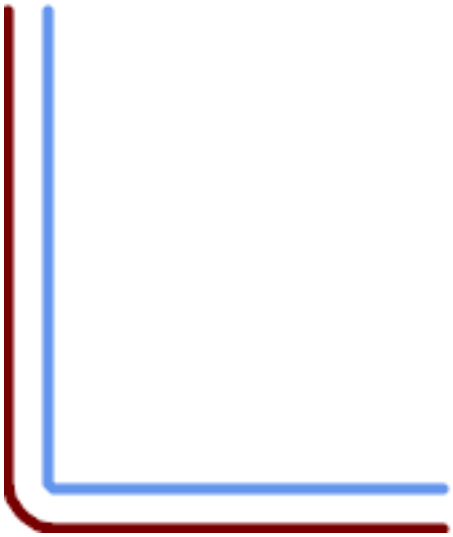
Erweiterung: ab 2.5 wird auch GEOMETRYCOLLECTION und MULTILINESTRING unterstützt.

Note
This function ignores the Z dimension. It always gives a 2D result even when used on a 3D geometry.

Beispiele

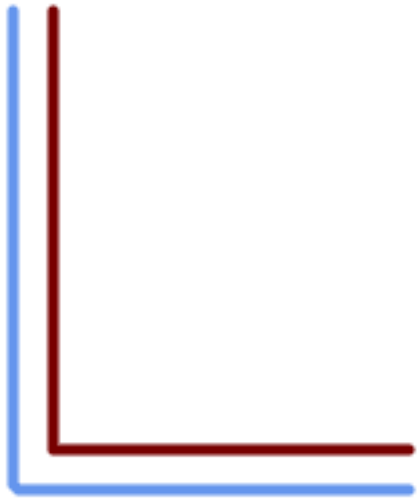
Einen offenen Puffer um die Straßen rechnen

```
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;
```



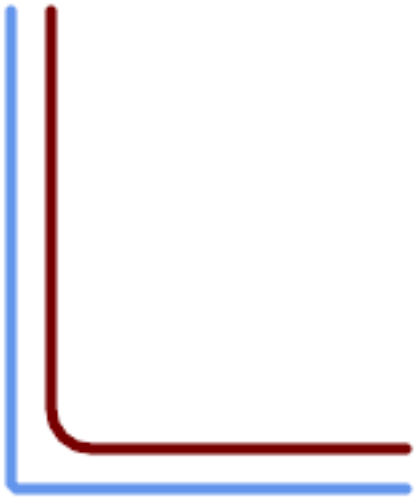
15, 'quad_segs=4 join=round' Ausgangslinie und die um 15 Einheiten versetzte Parallele.

```
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' Ausgangslinie und die um -15 Einheiten versetzte Parallele.

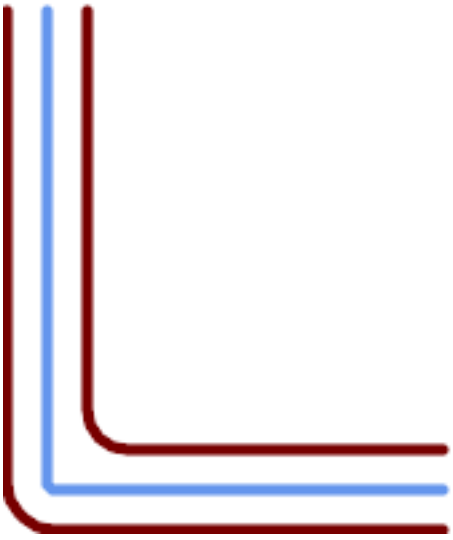
```
SELECT ST_AsText(ST_OffsetCurve(geom,
    -15, 'quad_segs=4 join=round')) As  ↩
    notsocurvy
    FROM ST_GeomFromText(
'LINESTRING(164 16,144 16,124 16,104  ↩
    16,84 16,64 16,
    44 16,24 16,20 16,18 16,17 17,
    16 18,16 20,16 40,16 60,16 80,16 100,
    16 120,16 140,16 160,16 180,16 195)') ↩
     As geom;

notsocurvy

LINESTRING(31 195,31 31,164 31)
```



doppelter Versatz um es kurviger zu bekommen; beachte die Richtungsänderung, also -30 + 15 = -15

```
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;
```

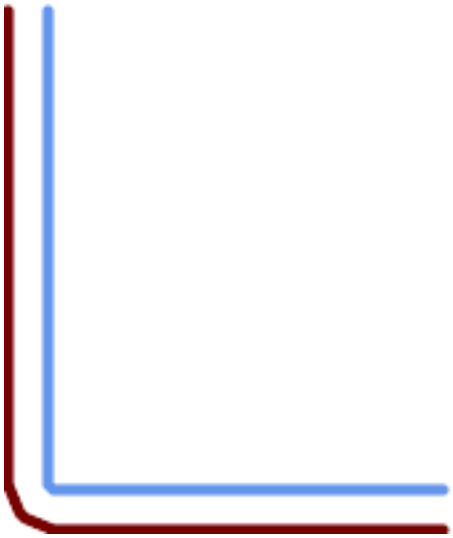
```
morecurvy

LINESTRING(164 31,46 31,40.2597485145236  ↩
    32.1418070123307,
35.3933982822018 35.3933982822018,
32.1418070123307 40.2597485145237,31  ↩
    46,31 195)
```



Doppelter Versatz um es kurviger zu bekommen, kombiniert mit einem normalen Versatz von 15 um parallele Linien zu erhalten. Überlagert mit dem Original.

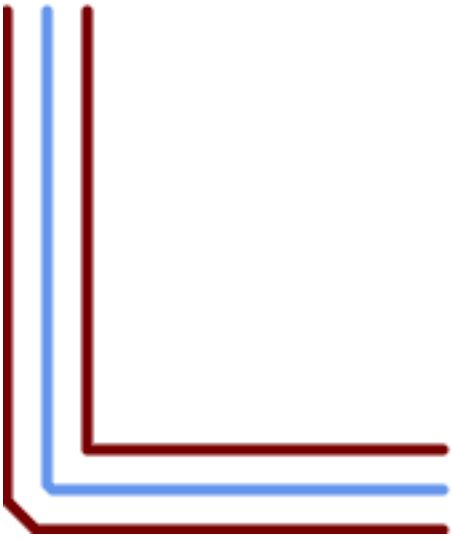
```
SELECT ST_AsText(ST_Collect(
    ST_OffsetCurve(geom, 15, 'quad_segs=4 ↩
     join=round'),
    ST_OffsetCurve(ST_OffsetCurve(geom,
    -30, 'quad_segs=4 join=round'), -15,  ↩
    'quad_segs=4 join=round')
    )
) As parallel_curves
    FROM ST_GeomFromText(
'LINESTRING(164 16,144 16,124 16,104  ↩
    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 7.393398282201
2.14180701233067 12.2597485145237,1 18,1  ↩
    195),
(164 31,46 31,40.2597485145236  ↩
    32.1418070123307,35.3933982822018 35.39339828220
32.1418070123307 40.2597485145237,31  ↩
    46,31 195))
```



15, 'quad_segs=4 join=bevel' gemeinsam mit der Ausgangslinie

```
SELECT ST_AsText(ST_OffsetCurve( ↩
    ST_GeomFromText(
'LINESTRING(164 16,144 16,124 16,104  ↩
    16,84 16,64 16,
    44 16,24 16,20 16,18 16,17 17,
    16 18,16 20,16 40,16 60,16 80,16 100,
    16 120,16 140,16 160,16 180,16 195)') ↩
    ,
        15, 'quad_segs=4 join=bevel'));

output

LINESTRING(164 1,18 1,7.39339828220179  ↩
    5.39339828220179,
    5.39339828220179 7.39339828220179,1  ↩
    18,1 195)
```



15,-15 collected, join=mitre mitre_limit=2.1

```
SELECT ST_AsText(ST_Collect(
    ST_OffsetCurve(geom, 15, 'quad_segs=4 ↩
     join=mitre mitre_limit=2.2'),
    ST_OffsetCurve(geom, -15, 'quad_segs ↩
    =4 join=mitre mitre_limit=2.2')
    ) )
    FROM ST_GeomFromText(
'LINESTRING(164 16,144 16,124 16,104  ↩
    16,84 16,64 16,
    44 16,24 16,20 16,18 16,17 17,
    16 18,16 20,16 40,16 60,16 80,16 100,
    16 120,16 140,16 160,16 180,16 195)') ↩
     As geom;

output

MULTILINESTRING((164 1,11.7867965644036  ↩
    1,1 11.7867965644036,1 195),
    (31 195,31 31,164 31))
```

Siehe auch

ST_Buffer

7.14.18 ST_PointOnSurface

ST_PointOnSurface — Computes a point guaranteed to lie in a polygon, or on a geometry.

Synopsis

geometry **ST_PointOnSurface**(geometry g1);

Beschreibung

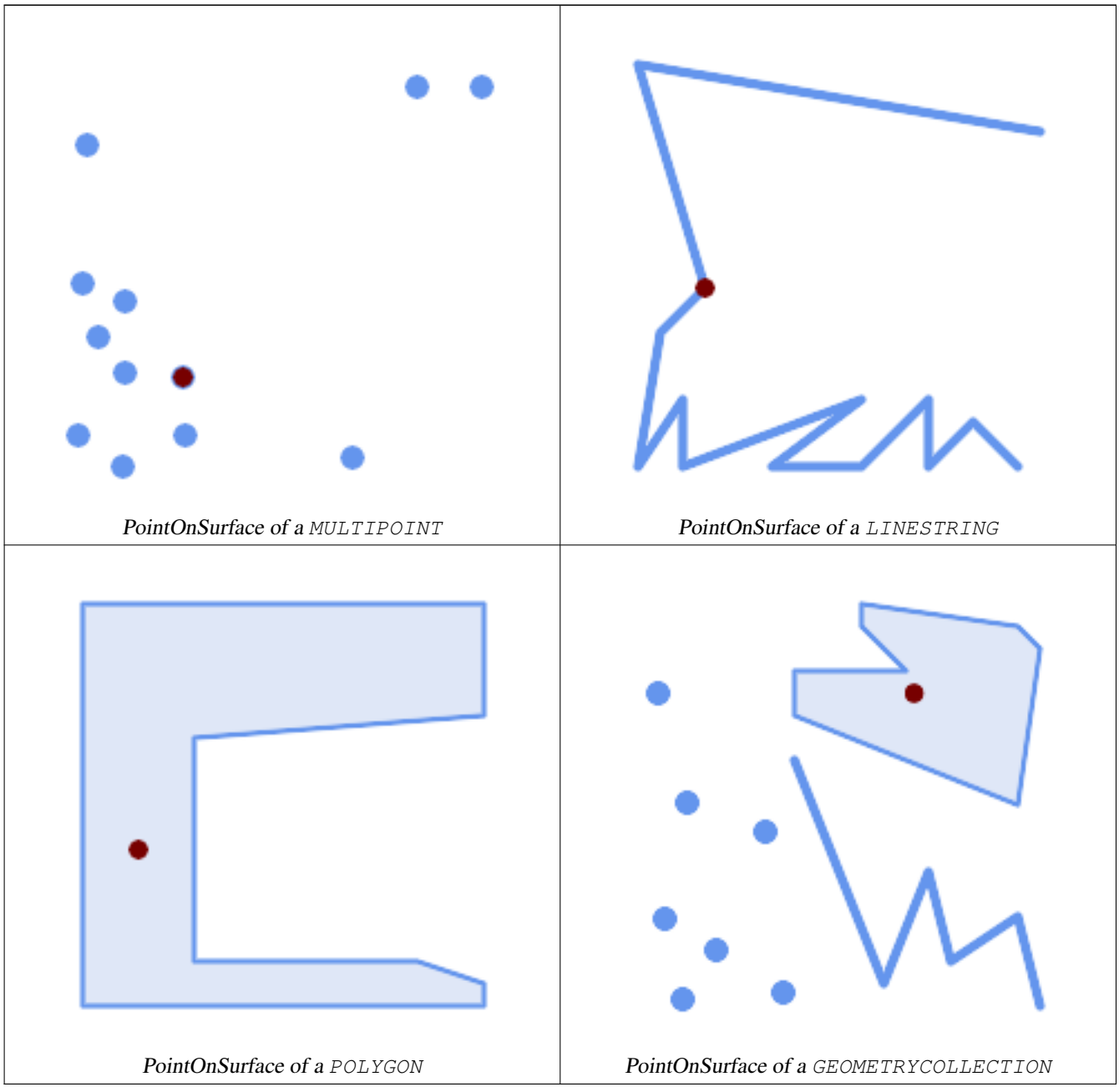
Returns a `POINT` which is guaranteed to lie in the interior of a surface (POLYGON, MULTIPOLYGON, and CURVED POLYGON). In PostGIS this function also works on line and point geometries.

This method implements the OGC Simple Features Implementation Specification for SQL 1.1. s3.2.14.2 // s3.2.18.2

This method implements the SQL/MM specification. SQL-MM 3: 8.1.5, 9.5.6. The specifications define ST_PointOnSurface for surface geometries only. PostGIS extends the function to support all common geometry types. Other databases (Oracle, DB2, ArcSDE) seem to support this function only for surfaces. SQL Server 2008 supports all common geometry types.

This function supports 3d and will not drop the z-index.

Beispiele



PointOnSurface of a `MULTIPOINT`

PointOnSurface of a `LINESTRING`

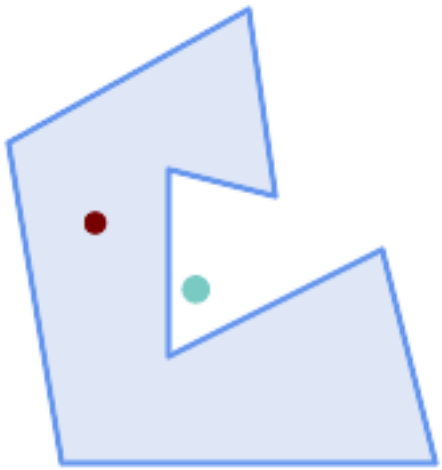
PointOnSurface of a `POLYGON`

PointOnSurface of a `GEOMETRYCOLLECTION`

```
SELECT ST_AsText(ST_PointOnSurface('POINT(0 5)'::geometry));
------------
 POINT(0 5)
```

```
SELECT ST_AsText(ST_PointOnSurface('LINESTRING(0 5, 0 10)'::geometry));
------------
 POINT(0 5)

SELECT ST_AsText(ST_PointOnSurface('POLYGON((0 0, 0 5, 5 5, 5 0, 0 0))'::geometry));
----------------
 POINT(2.5 2.5)

SELECT ST_AsEWKT(ST_PointOnSurface(ST_GeomFromEWKT('LINESTRING(0 5 1, 0 0 1, 0 10 2)')));
----------------
 POINT(0 0 1)
```

Example: The result of ST_PointOnSurface is guaranteed to lie within polygons, whereas the point computed by ST_Centroid may be outside.



Red: point on surface; Green: centroid

```
SELECT ST_AsText(ST_PointOnSurface(geom)) AS pt_on_surf,
       ST_AsText(ST_Centroid(geom)) AS centroid
    FROM (SELECT 'POLYGON ((130 120, 120 190, 30 140, 50 20, 190 20,
                      170 100, 90 60, 90 130, 130 120))'::geometry AS geom) AS t;

   pt_on_surf    |                  centroid
-----------------+-------------------------------------------
 POINT(62.5 110) | POINT(100.18264840182648 85.11415525114155)
```

Siehe auch

ST_Centroid, ST_MaximumInscribedCircle

7.14.19 ST_Polygonize

ST_Polygonize — Computes a collection of polygons formed from the linework of a set of geometries.

Synopsis

geometry **ST_Polygonize**(geometry set geomfield);
geometry **ST_Polygonize**(geometry[] geom_array);

Beschreibung

Creates a GeometryCollection containing the polygons formed by the linework of a set of geometries. If the input linework does not form any polygons, an empty GeometryCollection is returned.

This function creates polygons covering all delimited areas. If the result is intended to form a valid polygonal geometry, use ST_BuildArea to prevent holes being filled.

Note
The input linework must be correctly noded for this function to work properly. To ensure input is noded use ST_Node on the input geometry before polygonizing.

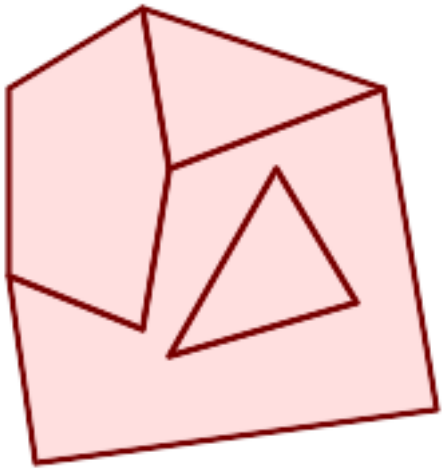
Note
GeometryCollections can be difficult to handle with external tools. Use ST_Dump to convert the polygonized result into separate polygons.

Wird vom GEOS Modul ausgeführt

Verfügbarkeit: 1.0.0RC1

Beispiele

Input lines



Polygonized result

```
WITH data(geom) AS (VALUES
    ('LINESTRING (180 40, 30 20, 20 90)'::geometry)
  ,('LINESTRING (180 40, 160 160)'::geometry)
  ,('LINESTRING (80 60, 120 130, 150 80)'::geometry)
  ,('LINESTRING (80 60, 150 80)'::geometry)
  ,('LINESTRING (20 90, 70 70, 80 130)'::geometry)
  ,('LINESTRING (80 130, 160 160)'::geometry)
  ,('LINESTRING (20 90, 20 160, 70 190)'::geometry)
  ,('LINESTRING (70 190, 80 130)'::geometry)
```

```
  ,('LINESTRING (70 190, 160 160)'::geometry)
)
SELECT ST_AsText( ST_Polygonize( geom ))
    FROM data;

------------------------------------------------------------------------------------------
GEOMETRYCOLLECTION (POLYGON ((180 40, 30 20, 20 90, 70 70, 80 130, 160 160, 180 40), (150  ↩
    80, 120 130, 80 60, 150 80)),
                    POLYGON ((20 90, 20 160, 70 190, 80 130, 70 70, 20 90)),
                    POLYGON ((160 160, 80 130, 70 190, 160 160)),
                    POLYGON ((80 60, 120 130, 150 80, 80 60)))
```

Polygonizing a table of linestrings:

```
SELECT ST_AsEWKT(ST_Polygonize(geom_4269)) As geomtextrep
FROM (SELECT geom_4269 FROM ma.suffolk_edges) As foo;

--------------------------------------
 SRID=4269;GEOMETRYCOLLECTION(POLYGON((-71.040878 42.285678,-71.040943 42.2856,-71.04096  ↩
     42.285752,-71.040878 42.285678)),
 POLYGON((-71.17166 42.353675,-71.172026 42.354044,-71.17239 42.354358,-71.171794  ↩
     42.354971,-71.170511 42.354855,
 -71.17112 42.354238,-71.17166 42.353675)))

--Use ST_Dump to dump out the polygonize geoms into individual polygons
SELECT ST_AsEWKT((ST_Dump(t.polycoll)).geom) AS geomtextrep
FROM (SELECT ST_Polygonize(geom_4269) AS polycoll
    FROM (SELECT geom_4269 FROM ma.suffolk_edges)
        As foo) AS t;

------------------------
 SRID=4269;POLYGON((-71.040878 42.285678,-71.040943 42.2856,-71.04096 42.285752,
-71.040878 42.285678))
 SRID=4269;POLYGON((-71.17166 42.353675,-71.172026 42.354044,-71.17239 42.354358
,-71.171794 42.354971,-71.170511 42.354855,-71.17112 42.354238,-71.17166 42.353675))
```

Siehe auch

ST_BuildArea, ST_Dump, ST_Node

7.14.20 ST_ReducePrecision

ST_ReducePrecision — Returns a valid geometry with points rounded to a grid tolerance.

Synopsis

geometry **ST_ReducePrecision**(geometry g, float8 gridsize);

Beschreibung

Returns a valid geometry with all points rounded to the provided grid tolerance, and features below the tolerance removed.

Unlike ST_SnapToGrid the returned geometry will be valid, with no ring self-intersections or collapsed components.

Precision reduction can be used to:

- match coordinate precision to the data accuracy
- reduce the number of coordinates needed to represent a geometry
- ensure valid geometry output to formats which use lower precision (e.g. text formats such as WKT, GeoJSON or KML when the number of output decimal places is limited).
- export valid geometry to systems which use lower or limited precision (e.g. SDE, Oracle tolerance value)

Availability: 3.1.0.

Requires GEOS >= 3.9.0.

Beispiele

```
SELECT ST_AsText(ST_ReducePrecision('POINT(1.412 19.323)', 0.1));
    st_astext
-----------------
 POINT(1.4 19.3)

SELECT ST_AsText(ST_ReducePrecision('POINT(1.412 19.323)', 1.0));
  st_astext
-------------
 POINT(1 19)

SELECT ST_AsText(ST_ReducePrecision('POINT(1.412 19.323)', 10));
  st_astext
-------------
 POINT(0 20)
```

Precision reduction can reduce number of vertices

```
SELECT ST_AsText(ST_ReducePrecision('LINESTRING (10 10, 19.6 30.1, 20 30, 20.3 30, 40 40)', ↩
     1));
  st_astext
-------------
 LINESTRING (10 10, 20 30, 40 40)
```

Precision reduction splits polygons if needed to ensure validity

```
SELECT ST_AsText(ST_ReducePrecision('POLYGON ((10 10, 60 60.1, 70 30, 40 40, 50 10, 10 10)) ↩
    ', 10));
  st_astext
-------------
 MULTIPOLYGON (((60 60, 70 30, 40 40, 60 60)), ((40 40, 50 10, 10 10, 40 40)))
```

Siehe auch

ST_SnapToGrid, ST_Simplify, ST_SimplifyVW

7.14.21 ST_SharedPaths

ST_SharedPaths — Gibt eine Sammelgeometrie zurück, welche die gemeinsamen Strecken der beiden eingegebenen LineStrings/-MultiLinestrings enthält.

Synopsis

geometry **ST_SharedPaths**(geometry lineal1, geometry lineal2);

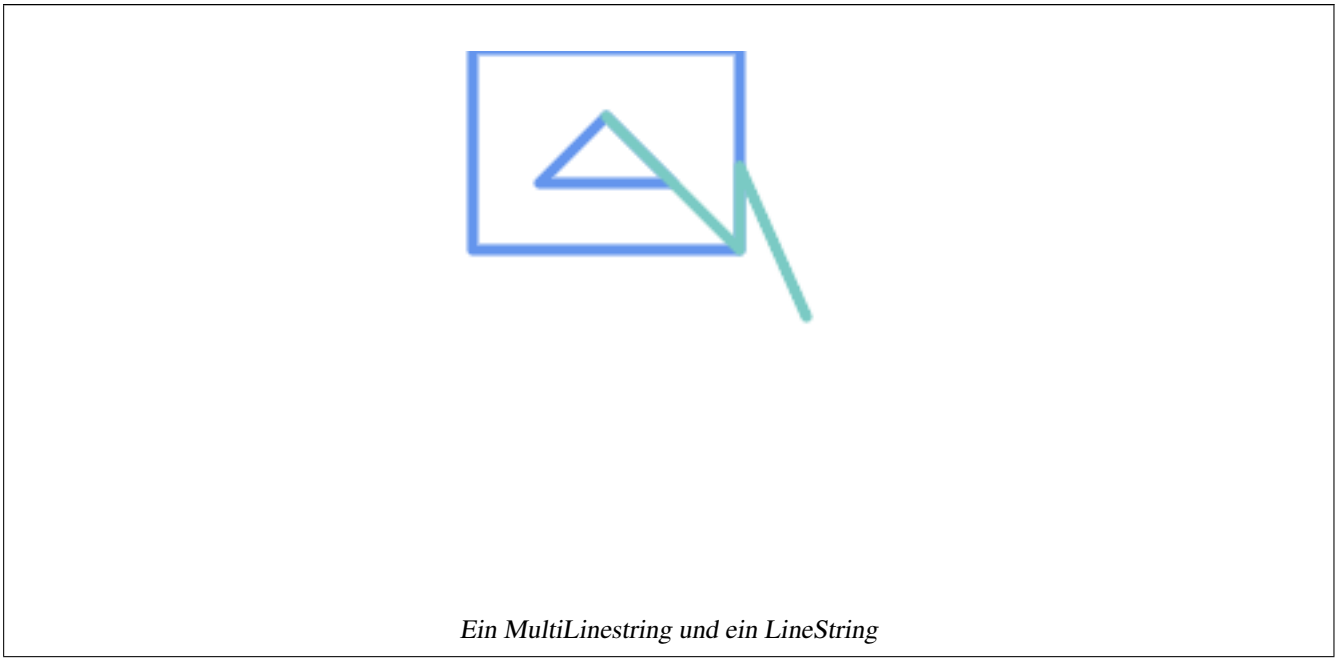
Beschreibung

Gibt eine Sammelgeometrie zurück, die die gemeinsamen Pfade zweier Eingabegeometrie enthält. Jene, die in derselben Richtung orientiert sind, werden im ersten Element der Sammelgeometrie, jene die in die entgegengesetzte Richtung orientiert sind, werden im zweiten Element gespeichert. Die Pfade selbst befinden sich in der ersten Geometrie.

Wird vom GEOS Modul ausgeführt

Verfügbarkeit: 2.0.0

Beispiele: Gemeinsame Strecken finden



Ein MultiLinestring und ein LineString

Die gemeinsame Strecke eines MultiLinestring und Linestring mit überlagerter Ursprungsgeometrie.

```
SELECT ST_AsText(
  ST_SharedPaths(
    ST_GeomFromText('MULTILINESTRING((26 125,26 200,126 200,126 125,26 125),
       (51 150,101 150,76 175,51 150))'),
    ST_GeomFromText('LINESTRING(151 100,126 156.25,126 125,90 161, 76 175)')
    )
  ) As wkt

                                wkt
------------------------------------------------------------
GEOMETRYCOLLECTION(MULTILINESTRING((126 156.25,126 125),
 (101 150,90 161),(90 161,76 175)),MULTILINESTRING EMPTY)
```

```
same example but linestring orientation flipped

SELECT ST_AsText(
  ST_SharedPaths(
   ST_GeomFromText('LINESTRING(76 175,90 161,126 125,126 156.25,151 100)'),
   ST_GeomFromText('MULTILINESTRING((26 125,26 200,126 200,126 125,26 125),
       (51 150,101 150,76 175,51 150))')
    )
  ) As wkt

                                wkt
------------------------------------------------------------
GEOMETRYCOLLECTION(MULTILINESTRING EMPTY,
MULTILINESTRING((76 175,90 161),(90 161,101 150),(126 125,126 156.25)))
```

Siehe auch

ST_Dump, ST_GeometryN, ST_NumGeometries

7.14.22 ST_Simplify

ST_Simplify — Returns a simplified version of a geometry, using the Douglas-Peucker algorithm.

Synopsis

geometry **ST_Simplify**(geometry geomA, float tolerance);
geometry **ST_Simplify**(geometry geomA, float tolerance, boolean preserveCollapsed);

Beschreibung

Gibt eine "vereinfachte" Version der gegebenen Geometrie zurück. Verwendet den Douglas-Peucker Algorithmus. Tut zurzeit nur mit (Multi)Lines und (Multi)Polygons etwas, kann aber gefahrlos mit jedem geometrischen Datentyp verwendet werden. Da die Vereinfachung auf einer Objekt zu Objekt Basis passiert, können Sie diese Funktion auch mit einer Sammelgeometrie speisen.

The "preserve collapsed" flag will retain objects that would otherwise be too small given the tolerance. For example, a 1m long line simplified with a 10m tolerance. If `preserveCollapsed` argument is specified as true, the line will not disappear. This flag is useful for rendering engines, to avoid having large numbers of very small objects disappear from a map leaving surprising gaps.

Note
Bemerke, dass die zurückgegebene Geometrie ihre Simplizität verlieren kann (siehe ST_IsSimple).

Note
Beachten Sie bitte, das die Topologie möglicherweise nicht erhalten bleibt und ungültige Geometrien entstehen können. Benutzen Sie bitte (see ST_SimplifyPreserveTopology) um die Topologie zu erhalten.

Verfügbarkeit: 1.2.2

Beispiele

Ein zu stark vereinfachter Kreis wird zu einem Dreieck, mittelmäßig vereinfacht zum Achteck:

```
SELECT ST_Npoints(geom) AS np_before,
       ST_NPoints(ST_Simplify(geom,0.1)) AS np01_notbadcircle,
       ST_NPoints(ST_Simplify(geom,0.5)) AS np05_notquitecircle,
       ST_NPoints(ST_Simplify(geom,1)) AS np1_octagon,
       ST_NPoints(ST_Simplify(geom,10)) AS np10_triangle,
       (ST_Simplify(geom,100) is null) AS  np100_geometrygoesaway
  FROM
    (SELECT ST_Buffer('POINT(1 3)', 10,12) As geom) AS foo;

 np_before | np01_notbadcircle | np05_notquitecircle | np1_octagon | np10_triangle |  ↩
    np100_geometrygoesaway
-----------+-------------------+---------------------+-------------+---------------+----------------

        49 |                33 |                  17 |           9 |             4 | t
```

Siehe auch

ST_IsSimple, ST_SimplifyPreserveTopology, ST_SimplifyVW, Topology ST_Simplify

7.14.23 ST_SimplifyPreserveTopology

ST_SimplifyPreserveTopology — Returns a simplified and valid version of a geometry, using the Douglas-Peucker algorithm.

Synopsis

geometry **ST_SimplifyPreserveTopology**(geometry geomA, float tolerance);

Beschreibung

Gibt eine "vereinfachte" Version der gegebenen Geometrie zurück. Verwendet den Douglas-Peucker Algorithmus. Vermeidet es, eine invalide Geometrie (insbesondere Polygone) zu erzeugen. Tut zurzeit nur mit (Multi)Lines und (Multi)Polygons etwas, kann aber gefahrlos mit jedem geometrischen Datentyp verwendet werden. Da die Vereinfachung auf einer Objekt zu Objekt Basis passiert, können Sie diese Funktion auch mit einer Sammelgeometrie speisen.

Wird vom GEOS Modul ausgeführt

Verfügbarkeit: 1.3.3

Beispiele

Das gleiche Beispiel wie mit ST_Simplify, aber wir sehen, dass ST_SimplifyPreserveTopology übermäßige Vereinfachung verhindert. Der Kreis kann maximal ein Quadrat werden.

```
SELECT ST_Npoints(geom) As np_before, ST_NPoints(ST_SimplifyPreserveTopology(geom,0.1)) As  ↩
    np01_notbadcircle, ST_NPoints(ST_SimplifyPreserveTopology(geom,0.5)) As  ↩
    np05_notquitecircle,
ST_NPoints(ST_SimplifyPreserveTopology(geom,1)) As np1_octagon, ST_NPoints( ↩
    ST_SimplifyPreserveTopology(geom,10)) As np10_square,
ST_NPoints(ST_SimplifyPreserveTopology(geom,100)) As  np100_stillsquare
FROM (SELECT ST_Buffer('POINT(1 3)', 10,12) As geom) As foo;

--result--
 np_before | np01_notbadcircle | np05_notquitecircle | np1_octagon | np10_square |  ↩
     np100_stillsquare
-----------+-------------------+---------------------+-------------+---------------+----------------

        49 |                33 |                  17 |           9 |             5 |  ↩
                                5
```

Siehe auch

ST_Simplify

7.14.24 ST_SimplifyPolygonHull

ST_SimplifyPolygonHull — Computes a simplifed topology-preserving outer or inner hull of a polygonal geometry.

Synopsis

geometry **ST_SimplifyPolygonHull**(geometry param_geom, float vertex_fraction, boolean is_outer = true);

Beschreibung

Computes a simplified topology-preserving outer or inner hull of a polygonal geometry. An outer hull completely covers the input geometry. An inner hull is completely covered by the input geometry. The result is a polygonal geometry formed by a subset of the input vertices. MultiPolygons and holes are handled and produce a result with the same structure as the input.

The reduction in vertex count is controlled by the `vertex_fraction` parameter, which is a number in the range 0 to 1. Lower values produce simpler results, with smaller vertex count and less concaveness. For both outer and inner hulls a vertex fraction of 1.0 produces the orginal geometry. For outer hulls a value of 0.0 produces the convex hull (for a single polygon); for inner hulls it produces a triangle.

The simplification process operates by progressively removing concave corners that contain the least amount of area, until the vertex count target is reached. It prevents edges from crossing, so the result is always a valid polygonal geometry.

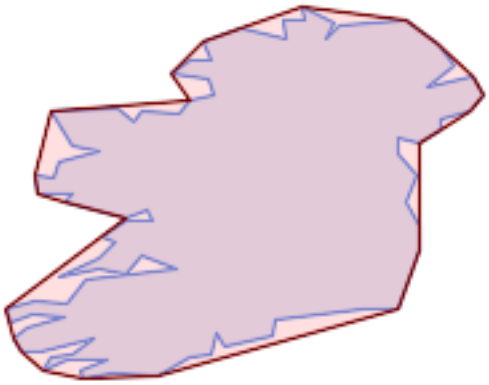
To get better results with geometries that contain relatively long line segments, it might be necessary to "segmentize" the input, as shown below.

Wird vom GEOS Modul ausgeführt

Availability: 3.3.0.

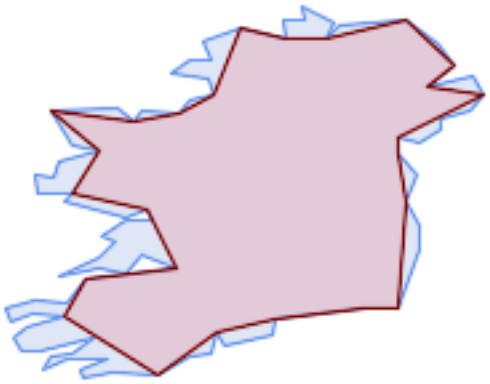
Requires GEOS >= 3.11.0.

Beispiele



Outer hull of a Polygon

```
SELECT ST_SimplifyPolygonHull(
  'POLYGON ((131 158, 136 163, 161 165, 173 156, 179 148, 169 140, 186 144, 190 137, 185  ↩
      131, 174 128, 174 124, 166 119, 158 121, 158 115, 165 107, 161 97, 166 88, 166 79, 158 ↩
       57, 145 57, 112 53, 111 47, 93 43, 90 48, 88 40, 80 39, 68 32, 51 33, 40 31, 39 34,  ↩
      49 38, 34 38, 25 34, 28 39, 36 40, 44 46, 24 41, 17 41, 14 46, 19 50, 33 54, 21 55, 13 ↩
       52, 11 57, 22 60, 34 59, 41 68, 75 72, 62 77, 56 70, 46 72, 31 69, 46 76, 52 82, 47  ↩
      84, 56 90, 66 90, 64 94, 56 91, 33 97, 36 100, 23 100, 22 107, 29 106, 31 112, 46 116, ↩
       36 118, 28 131, 53 132, 59 127, 62 131, 76 130, 80 135, 89 137, 87 143, 73 145, 80  ↩
      150, 88 150, 85 157, 99 162, 116 158, 115 165, 123 165, 122 170, 134 164, 131 158))',
    0.3);
```



Inner hull of a Polygon

```
SELECT ST_SimplifyPolygonHull(
  'POLYGON ((131 158, 136 163, 161 165, 173 156, 179 148, 169 140, 186 144, 190 137, 185 ↩
     131, 174 128, 174 124, 166 119, 158 121, 158 115, 165 107, 161 97, 166 88, 166 79, 158 ↩
      57, 145 57, 112 53, 111 47, 93 43, 90 48, 88 40, 80 39, 68 32, 51 33, 40 31, 39 34, ↩
     49 38, 34 38, 25 34, 28 39, 36 40, 44 46, 24 41, 17 41, 14 46, 19 50, 33 54, 21 55, 13 ↩
      52, 11 57, 22 60, 34 59, 41 68, 75 72, 62 77, 56 70, 46 72, 31 69, 46 76, 52 82, 47 ↩
     84, 56 90, 66 90, 64 94, 56 91, 33 97, 36 100, 23 100, 22 107, 29 106, 31 112, 46 116, ↩
      36 118, 28 131, 53 132, 59 127, 62 131, 76 130, 80 135, 89 137, 87 143, 73 145, 80 ↩
     150, 88 150, 85 157, 99 162, 116 158, 115 165, 123 165, 122 170, 134 164, 131 158))',
    0.3, false);
```

Outer hull simplification of a MultiPolygon, with segmentization

```
SELECT ST_SimplifyPolygonHull(
  ST_Segmentize(ST_Letters('xt'), 2.0),
    0.1);
```

Siehe auch

ST_ConvexHull, ST_SimplifyVW, ST_ConcaveHull, ST_Segmentize

7.14.25 ST_SimplifyVW

ST_SimplifyVW — Returns a simplified version of a geometry, using the Visvalingam-Whyatt algorithm

Synopsis

geometry **ST_SimplifyVW**(geometry geomA, float tolerance);

Beschreibung

Gibt eine "vereinfachte" Version der gegebenen Geometrie zurück. Verwendet den Visvalingam-Whyatt Algorithmus. Tut zurzeit nur mit (Multi)Lines und (Multi)Polygons etwas, kann aber gefahrlos mit jedem geometrischen Datentyp verwendet werden. Da die Vereinfachung auf einer Objekt zu Objekt Basis passiert, können Sie diese Funktion auch mit einer Sammelgeometrie speisen.

Note
Bemerke, dass die zurückgegebene Geometrie ihre Simplizität verlieren kann (siehe ST_IsSimple).

Note
Beachten Sie bitte, das die Topologie möglicherweise nicht erhalten bleibt und ungültige Geometrien entstehen können. Benutzen Sie bitte (see ST_SimplifyPreserveTopology) um die Topologie zu erhalten.

Note
Diese Funktion kann mit 3D umgehen und die dritte Dimension beeinflusst auch das Ergebnis.

Verfügbarkeit: 2.2.0

Beispiele

Ein Linienzug vereinfacht mit einem Schwellenwert von 30 für die minimale Fläche.

```
select ST_AsText(ST_SimplifyVW(geom,30)) simplified
FROM (SELECT  'LINESTRING(5 2, 3 8, 6 20, 7 25, 10 10)'::geometry geom) As foo;
-result
 simplified
------------------------------
LINESTRING(5 2,7 25,10 10)
```

Siehe auch

ST_SetEffectiveArea, ST_Simplify, ST_SimplifyPreserveTopology, Topology ST_Simplify

7.14.26 ST_SetEffectiveArea

ST_SetEffectiveArea — Sets the effective area for each vertex, using the Visvalingam-Whyatt algorithm.

Synopsis

geometry **ST_SetEffectiveArea**(geometry geomA, float threshold = 0, integer set_area = 1);

Beschreibung

Setzt die Nutzfläche für jeden Knoten. Verwendet den Visvalingam-Whyatt Algorithmus. Die Nutzfläche wird als M-Wert des Knoten gespeichert. Wird der optionale Parameter "threshold" verwendet, so wird eine vereinfachte Geometrie zurückgegeben, die nur jene Knoten enthält, deren Nutzfläche größer oder gleich dem Schwellenwert ist.

Diese Funktion kann für die serverseitige Vereinfachung, mittels eines Schwellenwerts verwendet werden. Eine andere Möglichkeit besteht darin, einen Schwellenwert von null anzugeben. In diesem Fall wird die gesamte Geometrie inklusive der Nutzflächen als M-Werte zurückgegeben, welche dann am Client für eine rasche Vereinfachung genutzt werden können.

Tut zurzeit nur mit (Multi)Lines und (Multi)Polygons etwas, kann aber gefahrlos mit jedem geometrischen Datentyp verwendet werden. Da die Vereinfachung auf einer Objekt zu Objekt Basis passiert, können Sie diese Funktion auch mit einer Sammelgeometrie speisen.

Note
Bemerke, dass die zurückgegebene Geometrie ihre Simplizität verlieren kann (siehe ST_IsSimple).

Note
Beachten Sie bitte, das die Topologie möglicherweise nicht erhalten bleibt und ungültige Geometrien entstehen können. Benutzen Sie bitte (see ST_SimplifyPreserveTopology) um die Topologie zu erhalten.

Note
Die Ausgabegeometrie verliert die gesamte vorhandene Information über die M-Werte

Note
Diese Funktion kann mit 3D umgehen und die dritte Dimension beeinflusst auch die tatsächliche Fläche

Verfügbarkeit: 2.2.0

Beispiele

Berechnung der Nutzfläche eines Linienzugs. Da wir einen Schwellenwert von null verwenden, werden alle Knoten der Eingabegeomettrie zurückgegeben.

```
select ST_AsText(ST_SetEffectiveArea(geom)) all_pts, ST_AsText(ST_SetEffectiveArea(geom,30) ↩
    ) thrshld_30
FROM (SELECT  'LINESTRING(5 2, 3 8, 6 20, 7 25, 10 10)'::geometry geom) As foo;
-result
 all_pts | thrshld_30
-----------+-------------------+
LINESTRING M (5 2 3.40282346638529e+38,3 8 29,6 20 1.5,7 25 49.5,10 10 3.40282346638529e ↩
    +38) | LINESTRING M (5 2 3.40282346638529e+38,7 25 49.5,10 10 3.40282346638529e+38)
```

Siehe auch

ST_SimplifyVW

7.14.27 ST_TriangulatePolygon

ST_TriangulatePolygon — Computes the constrained Delaunay triangulation of polygons

Synopsis

geometry **ST_TriangulatePolygon**(geometry geom);

Beschreibung

Computes the constrained Delaunay triangulation of polygons. Holes and Multipolygons are supported.

The "constrained Delaunay triangulation" of a polygon is a set of triangles formed from the vertices of the polygon, and covering it exactly, with the maximum total interior angle over all possible triangulations. It provides the "best quality" triangulation of the polygon.

Availability: 3.3.0.

Requires GEOS >= 3.11.0.

Example

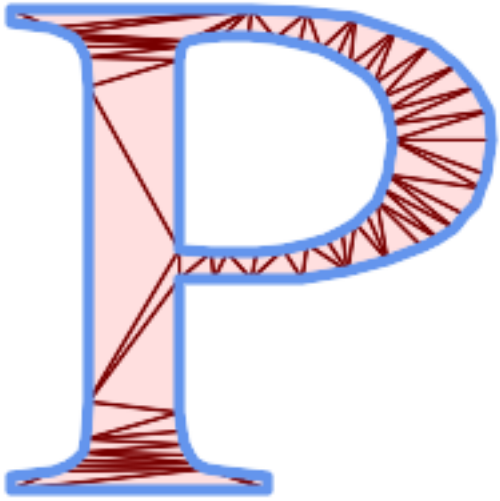
Triangulation of a square.

```
SELECT ST_AsText(
    ST_TriangulatePolygon('POLYGON((0 0, 0 1, 1 1, 1 0, 0 0))'));

                                st_astext
------------------------------------------------------------------------
 GEOMETRYCOLLECTION(POLYGON((0 0,0 1,1 1,0 0)),POLYGON((1 1,1 0,0 0,1 1)))
```

Example

Triangulation of the letter P.

```
SELECT ST_AsText(ST_TriangulatePolygon(
    'POLYGON ((26 17, 31 19, 34 21, 37 24, 38 29, 39 43, 39 161, 38 172, 36 176, 34 179, 30 ↩
         181, 25 183, 10 185, 10 190, 100 190, 121 189, 139 187, 154 182, 167 177, 177 169,  ↩
        184 161, 189 152, 190 141, 188 128, 186 123, 184 117, 180 113, 176 108, 170 104, 164 ↩
         101, 151 96, 136 92, 119 89, 100 89, 86 89, 73 89, 73 39, 74 32, 75 27, 77 23, 79  ↩
        20, 83 18, 89 17, 106 15, 106 10, 10 10, 10 15, 26 17), (152 147, 151 152, 149 157,  ↩
        146 162, 142 166, 137 169, 132 172, 126 175, 118 177, 109 179, 99 180, 89 180, 80  ↩
        179, 76 178, 74 176, 73 171, 73 100, 85 99, 91 99, 102 99, 112 100, 121 102, 128  ↩
        104, 134 107, 139 110, 143 114, 147 118, 149 123, 151 128, 153 141, 152 147))'
    ));
```



Polygon Triangulation

Siehe auch

ST_DelaunayTriangles, ST_ConstrainedDelaunayTriangles, ST_Tesselate

7.14.28 ST_VoronoiLines

ST_VoronoiLines — Returns the boundaries of the Voronoi diagram of the vertices of a geometry.

Synopsis

geometry **ST_VoronoiLines**(geometry geom , float8 tolerance = 0.0 , geometry extend_to = NULL);

Beschreibung

Computes a two-dimensional Voronoi diagram from the vertices of the supplied geometry and returns the boundaries between cells in the diagram as a MultiLineString. Returns null if input geometry is null. Returns an empty geometry collection if the input geometry contains only one vertex. Returns an empty geometry collection if the `extend_to` envelope has zero area.

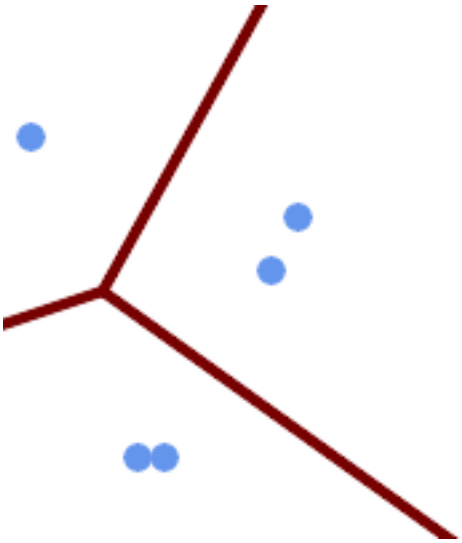
Optionale Parameter:

- `tolerance`: The distance within which vertices will be considered equivalent. Robustness of the algorithm can be improved by supplying a nonzero tolerance distance. (default = 0.0)
- `extend_to`: If present, the diagram is extended to cover the envelope of the supplied geometry, unless smaller than the default envelope (default = NULL, default envelope is the bounding box of the input expanded by about 50%).

Wird vom GEOS Modul ausgeführt

Verfügbarkeit: 2.3.0

Beispiele



Voronoi diagram lines, with tolerance of 30 units

```
SELECT ST_VoronoiLines(
            'MULTIPOINT (50 30, 60 30, 100 100,10 150, 110 120)'::geometry,
            30) AS geom;
```

```
ST_AsText output
MULTILINESTRING((135.555555555556 270,36.8181818181818 92.2727272727273),(36.8181818181818  ↩
    92.2727272727273,-110 43.3333333333333),(230 -45.7142857142858,36.8181818181818  ↩
    92.2727272727273))
```

Siehe auch

ST_DelaunayTriangles, ST_VoronoiPolygons

7.14.29 ST_VoronoiPolygons

ST_VoronoiPolygons — Returns the cells of the Voronoi diagram of the vertices of a geometry.

Synopsis

geometry **ST_VoronoiPolygons**(geometry geom , float8 tolerance = 0.0 , geometry extend_to = NULL);

Beschreibung

Computes a two-dimensional Voronoi diagram from the vertices of the supplied geometry. The result is a GEOMETRYCOLLECTION of POLYGONs that covers an envelope larger than the extent of the input vertices. Returns null if input geometry is null. Returns an empty geometry collection if the input geometry contains only one vertex. Returns an empty geometry collection if the `extend_to` envelope has zero area.

Optionale Parameter:

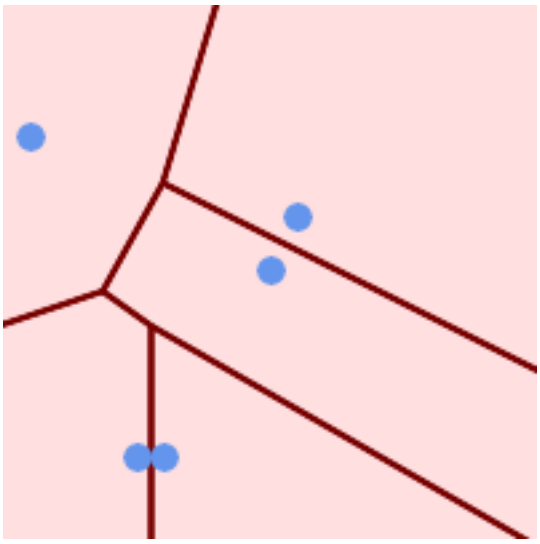
- `tolerance`: The distance within which vertices will be considered equivalent. Robustness of the algorithm can be improved by supplying a nonzero tolerance distance. (default = 0.0)

- `extend_to`: If present, the diagram is extended to cover the envelope of the supplied geometry, unless smaller than the default envelope (default = NULL, default envelope is the bounding box of the input expanded by about 50%).

Wird vom GEOS Modul ausgeführt

Verfügbarkeit: 2.3.0

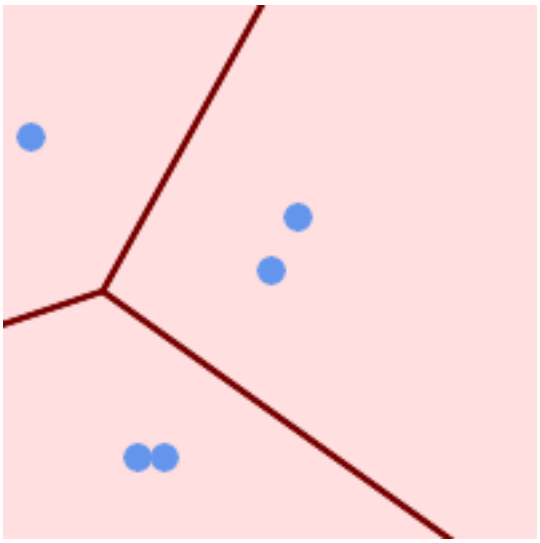
Beispiele



Punkte über dem Voronoi Diagramm

```
SELECT ST_VoronoiPolygons(
                'MULTIPOINT (50 30, 60 30, 100 100,10 150, 110 120)'::geometry
            ) AS geom;
```

```
ST_AsText output
GEOMETRYCOLLECTION(POLYGON((-110 43.3333333333333,-110 270,100.5 270,59.3478260869565  ↩
    132.826086956522,36.8181818181818 92.2727272727273,-110 43.3333333333333)),
POLYGON((55 -90,-110 -90,-110 43.3333333333333,36.8181818181818 92.2727272727273,55  ↩
    79.2857142857143,55 -90)),
POLYGON((230 47.5,230 -20.7142857142857,55 79.2857142857143,36.8181818181818  ↩
    92.2727272727273,59.3478260869565 132.826086956522,230 47.5)),POLYGON((230  ↩
    -20.7142857142857,230 -90,55 -90,55 79.2857142857143,230 -20.7142857142857)),
POLYGON((100.5 270,230 270,230 47.5,59.3478260869565 132.826086956522,100.5 270)))
```



Voronoi diagram, with tolerance of 30 units

```
SELECT ST_VoronoiPolygons(
            'MULTIPOINT (50 30, 60 30, 100 100,10 150, 110 120)'::geometry,
            30) AS geom;
```

```
ST_AsText output
GEOMETRYCOLLECTION(POLYGON((-110 43.3333333333333,-110 270,100.5 270,59.3478260869565  ↩
    132.826086956522,36.8181818181818 92.2727272727273,-110 43.3333333333333)),
POLYGON((230 47.5,230 -45.7142857142858,36.8181818181818 92.2727272727273,59.3478260869565  ↩
    132.826086956522,230 47.5)),POLYGON((230 -45.7142857142858,230 -90,-110 -90,-110  ↩
    43.3333333333333,36.8181818181818 92.2727272727273,230 -45.7142857142858)),
POLYGON((100.5 270,230 270,230 47.5,59.3478260869565 132.826086956522,100.5 270)))
```

Siehe auch

ST_DelaunayTriangles, ST_VoronoiLines

7.15 Coverages

7.15.1 ST_CoverageInvalidEdges

ST_CoverageInvalidEdges — Window function that finds locations where polygons fail to form a valid coverage.

Synopsis

geometry **ST_CoverageInvalidEdges**(geometry winset geom, float8 tolerance = 0);

Description

A window function which checks if the polygons in the window partition form a valid polygonal coverage. It returns linear indicators showing the location of invalid edges (if any) in each polygon.

A set of valid polygons is a valid coverage if the following conditions hold:

- **Non-overlapping** - polygons do not overlap (their interiors do not intersect)

- **Edge-Matched** - vertices along shared edges are identical

As a window function a value is returned for every input polygon. For polygons which violate one or more of the validity conditions the return value is a MULTILINESTRING containing the problematic edges. Coverage-valid polygons return the value NULL. Non-polygonal or empty geometries also produce NULL values.

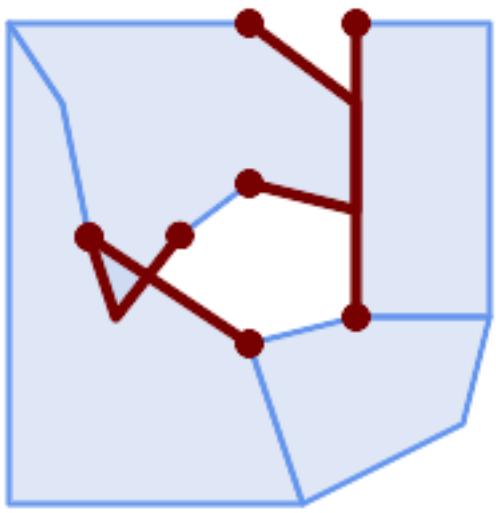
The conditions allow a valid coverage to contain holes (gaps between polygons), as long as the surrounding polygons are edge-matched. However, very narrow gaps are often undesirable. If the *tolerance* parameter is specified with a non-zero distance, edges forming narrower gaps will also be returned as invalid.

The polygons being checked for coverage validity must also be valid geometries. This can be checked with ST_IsValid.

Availability: 3.4.0

Requires GEOS >= 3.12.0

Examples



Invalid edges caused by overlap and non-matching vertices

```
WITH coverage(id, geom) AS (VALUES
  (1, 'POLYGON ((10 190, 30 160, 40 110, 100 70, 120 10, 10 10, 10 190))'::geometry),
  (2, 'POLYGON ((100 190, 10 190, 30 160, 40 110, 50 80, 74 110.5, 100 130, 140 120, 140  ↩
     160, 100 190))'::geometry),
  (3, 'POLYGON ((140 190, 190 190, 190 80, 140 80, 140 190))'::geometry),
  (4, 'POLYGON ((180 40, 120 10, 100 70, 140 80, 190 80, 180 40))'::geometry)
)
SELECT id, ST_AsText(ST_CoverageInvalidEdges(geom) OVER ())
  FROM coverage;

 id |              st_astext
----+---------------------------------------
  1 | LINESTRING (40 110, 100 70)
  2 | MULTILINESTRING ((100 130, 140 120, 140 160, 100 190), (40 110, 50 80, 74 110.5))
  3 | LINESTRING (140 80, 140 190)
  3 | null
```

```
-- Test entire table for coverage validity
SELECT true = ALL (
    SELECT ST_CoverageInvalidEdges(geom) OVER () IS NULL
    FROM coverage
    );
```

See Also

ST_IsValid, ST_CoverageUnion, ST_CoverageSimplify

7.15.2 ST_CoverageSimplify

ST_CoverageSimplify — Window function that simplifies the edges of a polygonal coverage.

Synopsis

geometry **ST_CoverageSimplify**(geometry winset geom, float8 tolerance, boolean simplifyBoundary = true);

Description

A window function which simplifies the edges of polygons in a polygonal coverage. The simplification preserves the coverage topology. This means the simplified output polygons are consisent along shared edges, and still form a valid coverage.

The simplification uses a variant of the Visvalingam–Whyatt algorithm. The `tolerance` parameter has units of distance, and is roughly equal to the square root of triangular areas to be simplified.

To simplify only the "internal" edges of the coverage (those that are shared by two polygons) set the `simplifyBoundary` parameter to false.

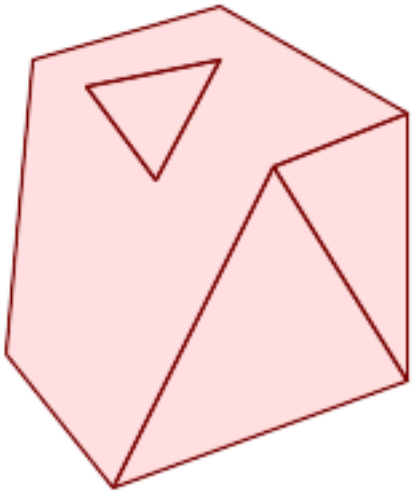
Note
If the input is not a valid coverage there may be unexpected artifacts in the output (such as boundary intersections, or separated boundaries which appeared to be shared). Use ST_CoverageInvalidEdges to determine if a coverage is valid.

Availability: 3.4.0

Requires GEOS >= 3.12.0

Examples

Input coverage



Simplified coverage

```
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))
  3 | POLYGON ((160 150, 160 50, 110 130, 160 150))
```

See Also

ST_CoverageInvalidEdges

7.15.3 ST_CoverageUnion

ST_CoverageUnion — Computes the union of a set of polygons forming a coverage by removing shared edges.

Synopsis

geometry **ST_CoverageUnion**(geometry set geom);

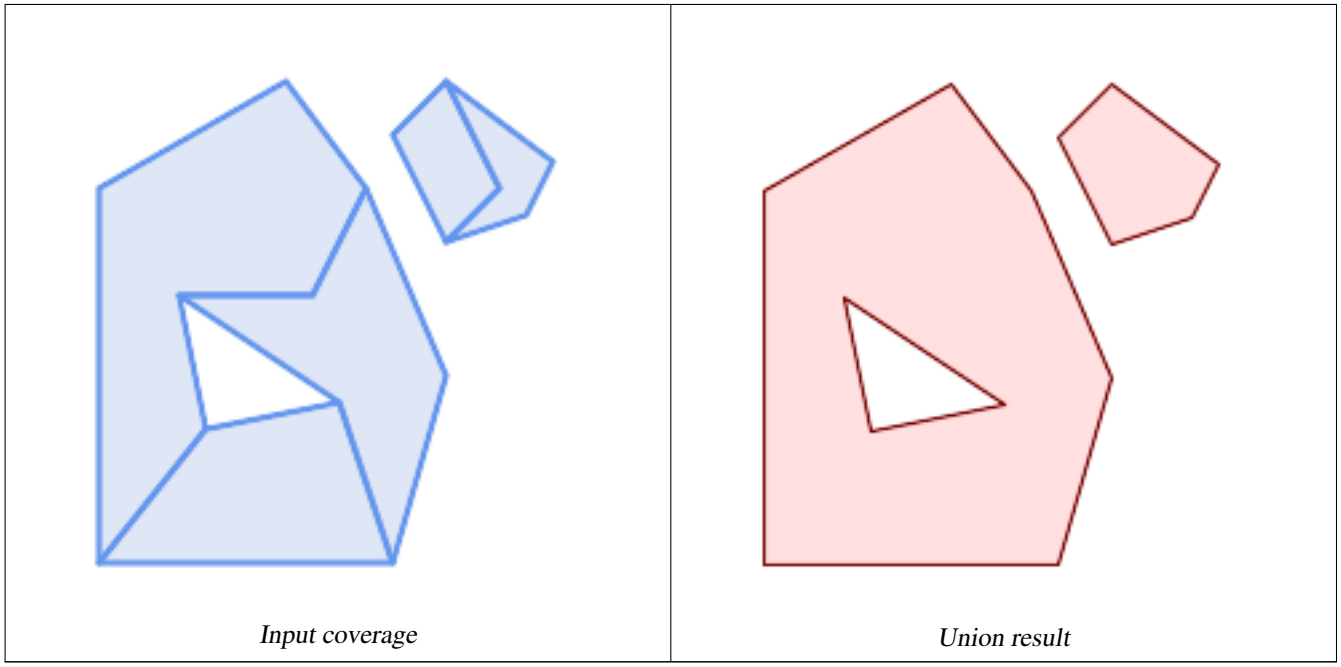
Description

An aggregate function which unions a set of polygons forming a polygonal coverage. The result is a polygonal geometry covering the same area as the coverage. This function produces the same result as ST_Union, but uses the coverage structure to compute the union much faster.

Note
If the input is not a valid coverage there may be unexpected artifacts in the output (such as unmerged or overlapping polygons). Use ST_CoverageInvalidEdges to determine if a coverage is valid.

Availability: 3.4.0 - requires GEOS >= 3.8.0

Examples



Input coverage | *Union result*

```
WITH coverage(id, geom) AS (VALUES
  (1, 'POLYGON ((10 10, 10 150, 80 190, 110 150, 90 110, 40 110, 50 60, 10 10))'::geometry) ↩
      ,
  (2, 'POLYGON ((120 10, 10 10, 50 60, 100 70, 120 10))'::geometry),
  (3, 'POLYGON ((140 80, 120 10, 100 70, 40 110, 90 110, 110 150, 140 80))'::geometry),
  (4, 'POLYGON ((140 190, 120 170, 140 130, 160 150, 140 190))'::geometry),
  (5, 'POLYGON ((180 160, 170 140, 140 130, 160 150, 140 190, 180 160))'::geometry)
)
SELECT ST_AsText(ST_CoverageUnion(geom))
  FROM coverage;

--------------------------------------
MULTIPOLYGON (((10 150, 80 190, 110 150, 140 80, 120 10, 10 10, 10 150), (50 60, 100 70, 40 ↩
     110, 50 60)), ((120 170, 140 190, 180 160, 170 140, 140 130, 120 170)))
```

See Also

ST_CoverageInvalidEdges, ST_Union

7.16 Affine Transformations

7.16.1 ST_Affine

ST_Affine — Apply a 3D affine transformation to a geometry.

Synopsis

geometry **ST_Affine**(geometry geomA, float a, float b, float c, float d, float e, float f, float g, float h, float i, float xoff, float yoff, float zoff);
geometry **ST_Affine**(geometry geomA, float a, float b, float d, float e, float xoff, float yoff);

Beschreibung

Applies a 3D affine transformation to the geometry to do things like translate, rotate, scale in one step.

Version 1: The call

```
ST_Affine(geom, a, b, c, d, e, f, g, h, i, xoff, yoff, zoff)
```

represents the transformation matrix

```
/ a  b  c  xoff \
| d  e  f  yoff |
| g  h  i  zoff |
\ 0  0  0     1 /
```

and the vertices are transformed as follows:

```
x' = a*x + b*y + c*z + xoff
y' = d*x + e*y + f*z + yoff
z' = g*x + h*y + i*z + zoff
```

All of the translate / scale functions below are expressed via such an affine transformation.

Version 2: Applies a 2d affine transformation to the geometry. The call

```
ST_Affine(geom, a, b, d, e, xoff, yoff)
```

represents the transformation matrix

```
/  a  b  0  xoff  \       /  a  b  xoff  \
|  d  e  0  yoff  | rsp.  |  d  e  yoff  |
|  0  0  1     0  |       \  0  0     1  /
\  0  0  0     1  /
```

and the vertices are transformed as follows:

```
x' = a*x + b*y + xoff
y' = d*x + e*y + yoff
z' = z
```

This method is a subcase of the 3D method above.

Enhanced: 2.0.0 support for Polyhedral surfaces, Triangles and TIN was introduced.

Availability: 1.1.2. Name changed from Affine to ST_Affine in 1.2.2

Note

Prior to 1.3.4, this function crashes if used with geometries that contain CURVES. This is fixed in 1.3.4+

This function supports Polyhedral surfaces.

This function supports Triangles and Triangulated Irregular Network Surfaces (TIN).

This function supports 3d and will not drop the z-index.

This method supports Circular Strings and Curves.

Examples

```
--Rotate a 3d line 180 degrees about the z axis.  Note this is long-hand for doing  ↩
    ST_Rotate();
 SELECT ST_AsEWKT(ST_Affine(geom,  cos(pi()), -sin(pi()), 0,  sin(pi()), cos(pi()), 0,  0,  ↩
     0, 1,  0, 0, 0)) As using_affine,
          ST_AsEWKT(ST_Rotate(geom, pi())) As using_rotate
        FROM (SELECT ST_GeomFromEWKT('LINESTRING(1 2 3, 1 4 3)') As geom) As foo;
        using_affine          |        using_rotate
------------------------------+-----------------------------
 LINESTRING(-1 -2 3,-1 -4 3) | LINESTRING(-1 -2 3,-1 -4 3)
(1 row)

--Rotate a 3d line 180 degrees in both the x and z axis
SELECT ST_AsEWKT(ST_Affine(geom, cos(pi()), -sin(pi()), 0, sin(pi()), cos(pi()), -sin(pi()) ↩
    , 0, sin(pi()), cos(pi()), 0, 0, 0))
        FROM (SELECT ST_GeomFromEWKT('LINESTRING(1 2 3, 1 4 3)') As geom) As foo;
           st_asewkt
--------------------------------
 LINESTRING(-1 -2 -3,-1 -4 -3)
(1 row)
```

Siehe auch

ST_Rotate, ST_Scale, ST_Translate, ST_TransScale

7.16.2 ST_Rotate

ST_Rotate — Rotates a geometry about an origin point.

Synopsis

geometry **ST_Rotate**(geometry geomA, float rotRadians);
geometry **ST_Rotate**(geometry geomA, float rotRadians, float x0, float y0);
geometry **ST_Rotate**(geometry geomA, float rotRadians, geometry pointOrigin);

Beschreibung

Rotates geometry rotRadians counter-clockwise about the origin point. The rotation origin can be specified either as a POINT geometry, or as x and y coordinates. If the origin is not specified, the geometry is rotated about POINT(0 0).

Enhanced: 2.0.0 support for Polyhedral surfaces, Triangles and TIN was introduced.

Enhanced: 2.0.0 additional parameters for specifying the origin of rotation were added.

Availability: 1.1.2. Name changed from Rotate to ST_Rotate in 1.2.2

This function supports 3d and will not drop the z-index.

This method supports Circular Strings and Curves.

This function supports Polyhedral surfaces.

This function supports Triangles and Triangulated Irregular Network Surfaces (TIN).

Examples

```
--Rotate 180 degrees
SELECT ST_AsEWKT(ST_Rotate('LINESTRING (50 160, 50 50, 100 50)', pi()));
               st_asewkt
-----------------------------------
 LINESTRING(-50 -160,-50 -50,-100 -50)
(1 row)

--Rotate 30 degrees counter-clockwise at x=50, y=160
SELECT ST_AsEWKT(ST_Rotate('LINESTRING (50 160, 50 50, 100 50)', pi()/6, 50, 160));
                                st_asewkt
--------------------------------------------------------------------------
 LINESTRING(50 160,105 64.7372055837117,148.301270189222 89.7372055837117)
(1 row)

--Rotate 60 degrees clockwise from centroid
SELECT ST_AsEWKT(ST_Rotate(geom, -pi()/3, ST_Centroid(geom)))
FROM (SELECT 'LINESTRING (50 160, 50 50, 100 50)'::geometry AS geom) AS foo;
                            st_asewkt
-----------------------------------------------------------
 LINESTRING(116.4225 130.6721,21.1597 75.6721,46.1597 32.3708)
(1 row)
```

Siehe auch

ST_Affine, ST_RotateX, ST_RotateY, ST_RotateZ

7.16.3 ST_RotateX

ST_RotateX — Rotates a geometry about the X axis.

Synopsis

geometry **ST_RotateX**(geometry geomA, float rotRadians);

Beschreibung

Rotates a geometry geomA - rotRadians about the X axis.

Note
`ST_RotateX(geomA, rotRadians)` is short-hand for `ST_Affine(geomA, 1, 0, 0, 0, cos(rotRadians), -sin(rotRadians), 0, sin(rotRadians), cos(rotRadians), 0, 0, 0).`

Enhanced: 2.0.0 support for Polyhedral surfaces, Triangles and TIN was introduced.

Availability: 1.1.2. Name changed from RotateX to ST_RotateX in 1.2.2

This function supports Polyhedral surfaces.

This function supports 3d and will not drop the z-index.

This function supports Triangles and Triangulated Irregular Network Surfaces (TIN).

Examples

```
--Rotate a line 90 degrees along x-axis
SELECT ST_AsEWKT(ST_RotateX(ST_GeomFromEWKT('LINESTRING(1 2 3, 1 1 1)'), pi()/2));
                 st_asewkt
---------------------------
 LINESTRING(1 -3 2,1 -1 1)
```

Siehe auch

ST_Affine, ST_RotateY, ST_RotateZ

7.16.4 ST_RotateY

ST_RotateY — Rotates a geometry about the Y axis.

Synopsis

geometry **ST_RotateY**(geometry geomA, float rotRadians);

Beschreibung

Rotates a geometry geomA - rotRadians about the y axis.

Note
`ST_RotateY(geomA, rotRadians)` is short-hand for `ST_Affine(geomA, cos(rotRadians), 0, sin(rotRadians), 0, 1, 0, -sin(rotRadians), 0, cos(rotRadians), 0, 0, 0)`.

Availability: 1.1.2. Name changed from RotateY to ST_RotateY in 1.2.2

Enhanced: 2.0.0 support for Polyhedral surfaces, Triangles and TIN was introduced.

This function supports Polyhedral surfaces.

This function supports 3d and will not drop the z-index.

This function supports Triangles and Triangulated Irregular Network Surfaces (TIN).

Examples

```
--Rotate a line 90 degrees along y-axis
 SELECT ST_AsEWKT(ST_RotateY(ST_GeomFromEWKT('LINESTRING(1 2 3, 1 1 1)'), pi()/2));
                 st_asewkt
---------------------------
 LINESTRING(3 2 -1,1 1 -1)
```

Siehe auch

ST_Affine, ST_RotateX, ST_RotateZ

7.16.5 ST_RotateZ

ST_RotateZ — Rotates a geometry about the Z axis.

Synopsis

geometry **ST_RotateZ**(geometry geomA, float rotRadians);

Beschreibung

Rotates a geometry geomA - rotRadians about the Z axis.

Note
This is a synonym for ST_Rotate

Note
`ST_RotateZ(geomA, rotRadians)` is short-hand for `SELECT ST_Affine(geomA, cos(rotRadians), -sin(rotRadians), 0, sin(rotRadians), cos(rotRadians), 0, 0, 0, 1, 0, 0, 0).`

Enhanced: 2.0.0 support for Polyhedral surfaces, Triangles and TIN was introduced.

Availability: 1.1.2. Name changed from RotateZ to ST_RotateZ in 1.2.2

Note
Prior to 1.3.4, this function crashes if used with geometries that contain CURVES. This is fixed in 1.3.4+

This function supports 3d and will not drop the z-index.

This method supports Circular Strings and Curves.

This function supports Polyhedral surfaces.

This function supports Triangles and Triangulated Irregular Network Surfaces (TIN).

Examples

```
--Rotate a line 90 degrees along z-axis
SELECT ST_AsEWKT(ST_RotateZ(ST_GeomFromEWKT('LINESTRING(1 2 3, 1 1 1)'), pi()/2));
                 st_asewkt
---------------------------
 LINESTRING(-2 1 3,-1 1 1)

 --Rotate a curved circle around z-axis
SELECT ST_AsEWKT(ST_RotateZ(geom, pi()/2))
FROM (SELECT ST_LineToCurve(ST_Buffer(ST_GeomFromText('POINT(234 567)'), 3)) As geom) As ↩
    foo;
```

```
----------------------------------------------------------------------------------------------------------------

 CURVEPOLYGON(CIRCULARSTRING(-567 237,-564.87867965644 236.12132034356,-564  ↩
     234,-569.12132034356 231.87867965644,-567 237))
```

Siehe auch

ST_Affine, ST_RotateX, ST_RotateY

7.16.6 ST_Scale

ST_Scale — Scales a geometry by given factors.

Synopsis

geometry **ST_Scale**(geometry geomA, float XFactor, float YFactor, float ZFactor);
geometry **ST_Scale**(geometry geomA, float XFactor, float YFactor);
geometry **ST_Scale**(geometry geom, geometry factor);
geometry **ST_Scale**(geometry geom, geometry factor, geometry origin);

Beschreibung

Scales the geometry to a new size by multiplying the ordinates with the corresponding factor parameters.

The version taking a geometry as the `factor` parameter allows passing a 2d, 3dm, 3dz or 4d point to set scaling factor for all supported dimensions. Missing dimensions in the `factor` point are equivalent to no scaling the corresponding dimension.

The three-geometry variant allows a "false origin" for the scaling to be passed in. This allows "scaling in place", for example using the centroid of the geometry as the false origin. Without a false origin, scaling takes place relative to the actual origin, so all coordinates are just multipled by the scale factor.

Note
Prior to 1.3.4, this function crashes if used with geometries that contain CURVES. This is fixed in 1.3.4+

Availability: 1.1.0.

Enhanced: 2.0.0 support for Polyhedral surfaces, Triangles and TIN was introduced.

Enhanced: 2.2.0 support for scaling all dimension (`factor` parameter) was introduced.

Enhanced: 2.5.0 support for scaling relative to a local origin (`origin` parameter) was introduced.

This function supports Polyhedral surfaces.

This function supports 3d and will not drop the z-index.

This method supports Circular Strings and Curves.

This function supports Triangles and Triangulated Irregular Network Surfaces (TIN).

This function supports M coordinates.

Examples

```
--Version 1: scale X, Y, Z
SELECT ST_AsEWKT(ST_Scale(ST_GeomFromEWKT('LINESTRING(1 2 3, 1 1 1)'), 0.5, 0.75, 0.8));
                           st_asewkt
-------------------------------------
 LINESTRING(0.5 1.5 2.4,0.5 0.75 0.8)

--Version 2: Scale X Y
 SELECT ST_AsEWKT(ST_Scale(ST_GeomFromEWKT('LINESTRING(1 2 3, 1 1 1)'), 0.5, 0.75));
                         st_asewkt
---------------------------------
 LINESTRING(0.5 1.5 3,0.5 0.75 1)

--Version 3: Scale X Y Z M
 SELECT ST_AsEWKT(ST_Scale(ST_GeomFromEWKT('LINESTRING(1 2 3 4, 1 1 1 1)'),
   ST_MakePoint(0.5, 0.75, 2, -1)));
                                st_asewkt
-----------------------------------------
 LINESTRING(0.5 1.5 6 -4,0.5 0.75 2 -1)

--Version 4: Scale X Y using false origin
SELECT ST_AsText(ST_Scale('LINESTRING(1 1, 2 2)', 'POINT(2 2)', 'POINT(1 1)'::geometry));
      st_astext
---------------------
 LINESTRING(1 1,3 3)
```

Siehe auch

ST_Affine, ST_TransScale

7.16.7 ST_Translate

ST_Translate — Translates a geometry by given offsets.

Synopsis

geometry **ST_Translate**(geometry g1, float deltax, float deltay);
geometry **ST_Translate**(geometry g1, float deltax, float deltay, float deltaz);

Beschreibung

Returns a new geometry whose coordinates are translated delta x,delta y,delta z units. Units are based on the units defined in spatial reference (SRID) for this geometry.

Note
Prior to 1.3.4, this function crashes if used with geometries that contain CURVES. This is fixed in 1.3.4+

Availability: 1.2.2

This function supports 3d and will not drop the z-index.

This method supports Circular Strings and Curves.

Examples

Move a point 1 degree longitude

```
SELECT ST_AsText(ST_Translate(ST_GeomFromText('POINT(-71.01 42.37)',4326),1,0)) As  ↩
    wgs_transgeomtxt;

        wgs_transgeomtxt
        ---------------------
        POINT(-70.01 42.37)
```

Move a linestring 1 degree longitude and 1/2 degree latitude

```
SELECT ST_AsText(ST_Translate(ST_GeomFromText('LINESTRING(-71.01 42.37,-71.11 42.38)',4326) ↩
    ,1,0.5)) As wgs_transgeomtxt;
                    wgs_transgeomtxt
        ---------------------------------------
        LINESTRING(-70.01 42.87,-70.11 42.88)
```

Move a 3d point

```
SELECT ST_AsEWKT(ST_Translate(CAST('POINT(0 0 0)' As geometry), 5, 12,3));
        st_asewkt
        ---------
        POINT(5 12 3)
```

Move a curve and a point

```
SELECT ST_AsText(ST_Translate(ST_Collect('CURVEPOLYGON(CIRCULARSTRING(4 3,3.12 0.878,1  ↩
    0,-1.121 5.1213,6 7, 8 9,4 3))','POINT(1 3)'),1,2));


-------------------------------------------------------------------------------------------------------------

 GEOMETRYCOLLECTION(CURVEPOLYGON(CIRCULARSTRING(5 5,4.12 2.878,2 2,-0.121 7.1213,7 9,9 11,5 ↩
     5)),POINT(2 5))
```

Siehe auch

ST_Affine, ST_AsText, ST_GeomFromText

7.16.8 ST_TransScale

ST_TransScale — Translates and scales a geometry by given offsets and factors.

Synopsis

geometry **ST_TransScale**(geometry geomA, float deltaX, float deltaY, float XFactor, float YFactor);

Beschreibung

Translates the geometry using the deltaX and deltaY args, then scales it using the XFactor, YFactor args, working in 2D only.

Note
`ST_TransScale(geomA, deltaX, deltaY, XFactor, YFactor)` is short-hand for `ST_Affine(geomA, XFactor, 0, 0, 0, YFactor, 0, 0, 0, 1, deltaX*XFactor, deltaY*YFactor, 0)`.

Note
Prior to 1.3.4, this function crashes if used with geometries that contain CURVES. This is fixed in 1.3.4+

Availability: 1.1.0.

This function supports 3d and will not drop the z-index.

This method supports Circular Strings and Curves.

Examples

```
SELECT ST_AsEWKT(ST_TransScale(ST_GeomFromEWKT('LINESTRING(1 2 3, 1 1 1)'), 0.5, 1, 1, 2));
                  st_asewkt
-----------------------------
 LINESTRING(1.5 6 3,1.5 4 1)


--Buffer a point to get an approximation of a circle, convert to curve and then translate  ↩
    1,2 and scale it 3,4
  SELECT ST_AsText(ST_Transscale(ST_LineToCurve(ST_Buffer('POINT(234 567)', 3)),1,2,3,4));


----------------------------------------------------------------------------------------------------------------

 CURVEPOLYGON(CIRCULARSTRING(714 2276,711.363961030679 2267.51471862576,705  ↩
     2264,698.636038969321 2284.48528137424,714 2276))
```

Siehe auch

ST_Affine, ST_Translate

7.17 Clustering Functions

7.17.1 ST_ClusterDBSCAN

ST_ClusterDBSCAN — Window function that returns a cluster id for each input geometry using the DBSCAN algorithm.

Synopsis

integer **ST_ClusterDBSCAN**(geometry winset geom, float8 eps, integer minpoints);

Beschreibung

A window function that returns a cluster number for each input geometry, using the 2D Density-based spatial clustering of applications with noise (DBSCAN) algorithm. Unlike ST_ClusterKMeans, it does not require the number of clusters to be specified, but instead uses the desired distance (`eps`) and density (`minpoints`) parameters to determine each cluster.

An input geometry is added to a cluster if it is either:

- A "core" geometry, that is within `eps` distance of at least `minpoints` input geometries (including itself); or
- A "border" geometry, that is within `eps` distance of a core geometry.

Note that border geometries may be within `eps` distance of core geometries in more than one cluster. Either assignment would be correct, so the border geometry will be arbitrarily asssigned to one of the available clusters. In this situation it is possible for a correct cluster to be generated with fewer than `minpoints` geometries. To ensure deterministic assignment of border geometries (so that repeated calls to ST_ClusterDBSCAN will produce identical results) use an `ORDER BY` clause in the window definition. Ambiguous cluster assignments may differ from other DBSCAN implementations.

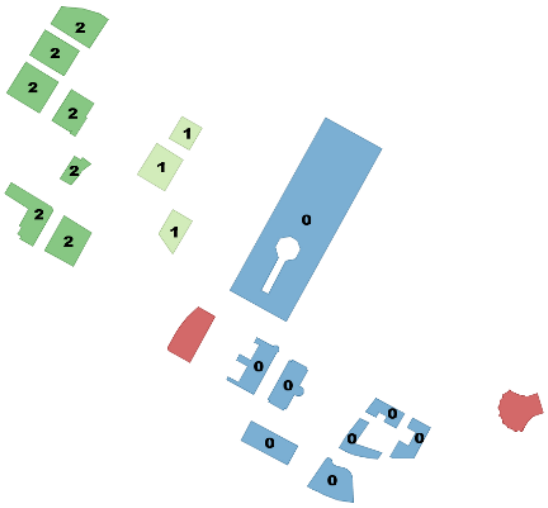
Note
Geometries that do not meet the criteria to join any cluster are assigned a cluster number of NULL.

Availability: 2.3.0

This method supports Circular Strings and Curves.

Examples

Clustering polygon within 50 meters of each other, and requiring at least 2 polygons per cluster.



Clusters within 50 meters with at least 2 items per cluster. Singletons have NULL for cid

```
SELECT name, ST_ClusterDBSCAN(geom, eps  ↩
    := 50, minpoints := 2) over () AS cid
FROM boston_polys
WHERE name
> '' AND building
> ''
        AND ST_DWithin(geom,
        ST_Transform(
            ST_GeomFromText('POINT ↩
    (-71.04054 42.35141)', 4326), 26986),
            500);
```

```
                name                   |  ↩
    bucket
-----------------------------------------+--------- ↩

 Manulife Tower                         |  ↩
          0
 Park Lane Seaport I                    |  ↩
          0
 Park Lane Seaport II                   |  ↩
          0
 Renaissance Boston Waterfront Hotel    |  ↩
          0
 Seaport Boston Hotel                   |  ↩
          0
 Seaport Hotel & World Trade Center     |  ↩
          0
 Waterside Place                        |  ↩
          0
 World Trade Center East                |  ↩
          0
 100 Northern Avenue                    |  ↩
          1
 100 Pier 4                             |  ↩
          1
 The Institute of Contemporary Art      |  ↩
          1
 101 Seaport                            |  ↩
          2
 District Hall                          |  ↩
          2
 One Marina Park Drive                  |  ↩
          2
 Twenty Two Liberty                     |  ↩
          2
 Vertex                                 |  ↩
          2
 Vertex                                 |  ↩
          2
 Watermark Seaport                      |  ↩
          2
 Blue Hills Bank Pavilion               |    ↩
     NULL
 World Trade Center West                |    ↩
     NULL
(20 rows)
```

A example showing combining parcels with the same cluster number into geometry collections.

```
SELECT cid, ST_Collect(geom) AS cluster_geom, array_agg(parcel_id) AS ids_in_cluster FROM (
    SELECT parcel_id, ST_ClusterDBSCAN(geom, eps := 0.5, minpoints := 5) over () AS cid,  ↩
        geom
    FROM parcels) sq
GROUP BY cid;
```

Siehe auch

ST_DWithin, ST_ClusterKMeans, ST_ClusterIntersecting, ST_ClusterIntersectingWin, ST_ClusterWithin, ST_ClusterWithinWin

7.17.2 ST_ClusterIntersecting

ST_ClusterIntersecting — Aggregate function that clusters input geometries into connected sets.

Synopsis

geometry[] **ST_ClusterIntersecting**(geometry set g);

Beschreibung

An aggregate function that returns an array of GeometryCollections partitioning the input geometries into connected clusters that are disjoint. Each geometry in a cluster intersects at least one other geometry in the cluster, and does not intersect any geometry in other clusters.

Availability: 2.2.0

Examples

```
WITH testdata AS
  (SELECT unnest(ARRAY['LINESTRING (0 0, 1 1)'::geometry,
           'LINESTRING (5 5, 4 4)'::geometry,
           'LINESTRING (6 6, 7 7)'::geometry,
           'LINESTRING (0 0, -1 -1)'::geometry,
           'POLYGON ((0 0, 4 0, 4 4, 0 4, 0 0))'::geometry]) AS geom)

SELECT ST_AsText(unnest(ST_ClusterIntersecting(geom))) FROM testdata;

--result

st_astext
---------
GEOMETRYCOLLECTION(LINESTRING(0 0,1 1),LINESTRING(5 5,4 4),LINESTRING(0 0,-1 -1),POLYGON((0 ↩
     0,4 0,4 4,0 4,0 0)))
GEOMETRYCOLLECTION(LINESTRING(6 6,7 7))
```

Siehe auch

ST_ClusterIntersectingWin, ST_ClusterWithin, ST_ClusterWithinWin

7.17.3 ST_ClusterIntersectingWin

ST_ClusterIntersectingWin — Window function that returns a cluster id for each input geometry, clustering input geometries into connected sets.

Synopsis

integer **ST_ClusterIntersectingWin**(geometry winset geom);

Beschreibung

A window function that builds connected clusters of geometries that intersect. It is possible to traverse all geometries in a cluster without leaving the cluster. The return value is the cluster number that the geometry argument participates in, or null for null inputs.

Availability: 3.4.0

Examples

```
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
```

Siehe auch

ST_ClusterIntersecting, ST_ClusterWithin, ST_ClusterWithinWin

7.17.4 ST_ClusterKMeans

ST_ClusterKMeans — Window function that returns a cluster id for each input geometry using the K-means algorithm.

Synopsis

integer **ST_ClusterKMeans**(geometry winset geom, integer number_of_clusters, float max_radius);

Beschreibung

Returns K-means cluster number for each input geometry. The distance used for clustering is the distance between the centroids for 2D geometries, and distance between bounding box centers for 3D geometries. For POINT inputs, M coordinate will be treated as weight of input and has to be larger than 0.

`max_radius`, if set, will cause ST_ClusterKMeans to generate more clusters than `k` ensuring that no cluster in output has radius larger than `max_radius`. This is useful in reachability analysis.

Enhanced: 3.2.0 Support for `max_radius`

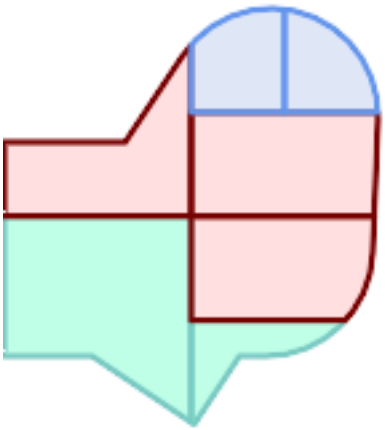
Enhanced: 3.1.0 Support for 3D geometries and weights

Availability: 2.3.0

Examples

Generate dummy set of parcels for examples:

```
CREATE TABLE parcels AS
SELECT lpad((row_number() over())::text,3,'0') As parcel_id, geom,
('{residential, commercial}'::text[])[1 + mod(row_number()OVER(),2)] As type
FROM
    ST_Subdivide(ST_Buffer('SRID=3857;LINESTRING(40 100, 98 100, 100 150, 60 90)'::geometry ↩
        ,
    40, 'endcap=square'),12) As geom;
```



Parcels color-coded by cluster number (cid)

```
SELECT ST_ClusterKMeans(geom, 3) OVER() AS cid, parcel_id, geom
    FROM parcels;
```

```
cid | parcel_id |   geom
-----+-----------+---------------
   0 | 001       | 0103000000...
   0 | 002       | 0103000000...
   1 | 003       | 0103000000...
   0 | 004       | 0103000000...
   1 | 005       | 0103000000...
   2 | 006       | 0103000000...
   2 | 007       | 0103000000...
```

Partitioning parcel clusters by type:

```
SELECT ST_ClusterKMeans(geom, 3) over (PARTITION BY type) AS cid, parcel_id, type
    FROM parcels;
```

```
cid | parcel_id |    type
-----+-----------+-------------
   1 | 005       | commercial
   1 | 003       | commercial
   2 | 007       | commercial
   0 | 001       | commercial
   1 | 004       | residential
   0 | 002       | residential
   2 | 006       | residential
```

Example: Clustering a preaggregated planetary-scale data population dataset using 3D clusering and weighting. Identify at least 20 regions based on Kontur Population Data that do not span more than 3000 km from their center:

```
create table kontur_population_3000km_clusters as
select
    geom,
    ST_ClusterKMeans(
        ST_Force4D(
            ST_Transform(ST_Force3D(geom), 4978), -- cluster in 3D XYZ CRS
            mvalue := population -- set clustering to be weighed by population
        ),
        20,                      -- aim to generate at least 20 clusters
        max_radius := 3000000    -- but generate more to make each under 3000 km radius
    ) over () as cid
from
    kontur_population;
```

World population clustered to above specs produces 46 clusters. Clusters are centered at well-populated regions (New York, Moscow). Greenland is one cluster. There are island clusters that span across the antimeridian. Cluster edges follow Earth's curvature.

Siehe auch

ST_ClusterDBSCAN, ST_ClusterIntersectingWin, ST_ClusterWithinWin, ST_ClusterIntersecting, ST_ClusterWithin, ST_Subdivide, ST_Force3D, ST_Force4D,

7.17.5 ST_ClusterWithin

ST_ClusterWithin — Aggregate function that clusters geometries by separation distance.

Synopsis

geometry[] **ST_ClusterWithin**(geometry set g, float8 distance);

Beschreibung

An aggregate function that returns an array of GeometryCollections, where each collection is a cluster containing some input geometries. Clustering partitions the input geometries into sets in which each geometry is within the specified *`distance`* of at least one other geometry in the same cluster. Distances are Cartesian distances in the units of the SRID.

ST_ClusterWithin is equivalent to running ST_ClusterDBSCAN with `minpoints := 0`.

Availability: 2.2.0

This method supports Circular Strings and Curves.

Examples

```
WITH testdata AS
  (SELECT unnest(ARRAY['LINESTRING (0 0, 1 1)'::geometry,
                       'LINESTRING (5 5, 4 4)'::geometry,
                       'LINESTRING (6 6, 7 7)'::geometry,
                       'LINESTRING (0 0, -1 -1)'::geometry,
                       'POLYGON ((0 0, 4 0, 4 4, 0 4, 0 0))'::geometry]) AS geom)

SELECT ST_AsText(unnest(ST_ClusterWithin(geom, 1.4))) FROM testdata;

--result

st_astext
---------
GEOMETRYCOLLECTION(LINESTRING(0 0,1 1),LINESTRING(5 5,4 4),LINESTRING(0 0,-1 -1),POLYGON((0 ↩
     0,4 0,4 4,0 4,0 0)))
GEOMETRYCOLLECTION(LINESTRING(6 6,7 7))
```

Siehe auch

ST_ClusterWithinWin, ST_ClusterDBSCAN, ST_ClusterIntersecting, ST_ClusterIntersectingWin

7.17.6 ST_ClusterWithinWin

ST_ClusterWithinWin — Window function that returns a cluster id for each input geometry, clustering using separation distance.

Synopsis

integer **ST_ClusterWithinWin**(geometry winset geom, float8 distance);

Beschreibung

A window function that returns a cluster number for each input geometry. Clustering partitions the geometries into sets in which each geometry is within the specified `distance` of at least one other geometry in the same cluster. Distances are Cartesian distances in the units of the SRID.

ST_ClusterWithinWin is equivalent to running ST_ClusterDBSCAN with `minpoints := 0`.

Availability: 3.4.0

This method supports Circular Strings and Curves.

Examples

```
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
```

Siehe auch

ST_ClusterWithin, ST_ClusterDBSCAN, ST_ClusterIntersecting, ST_ClusterIntersectingWin,

7.18 Bounding Box Functions

7.18.1 Box2D

Box2D — Returns a BOX2D representing the 2D extent of a geometry.

Synopsis

box2d **Box2D**(geometry geom);

Beschreibung

Returns a box2d representing the 2D extent of the geometry.

Enhanced: 2.0.0 support for Polyhedral surfaces, Triangles and TIN was introduced.

This method supports Circular Strings and Curves.

This function supports Polyhedral surfaces.

This function supports Triangles and Triangulated Irregular Network Surfaces (TIN).

Examples

```
SELECT Box2D(ST_GeomFromText('LINESTRING(1 2, 3 4, 5 6)'));

box2d
---------
BOX(1 2,5 6)
```

```
SELECT Box2D(ST_GeomFromText('CIRCULARSTRING(220268 150415,220227 150505,220227 150406)'));

box2d
--------
BOX(220186.984375 150406,220288.25 150506.140625)
```

Siehe auch

Box3D, ST_GeomFromText

7.18.2 Box3D

Box3D — Returns a BOX3D representing the 3D extent of a geometry.

Synopsis

box3d **Box3D**(geometry geom);

Beschreibung

Returns a box3d representing the 3D extent of the geometry.

Enhanced: 2.0.0 support for Polyhedral surfaces, Triangles and TIN was introduced.

This method supports Circular Strings and Curves.

This function supports Polyhedral surfaces.

This function supports Triangles and Triangulated Irregular Network Surfaces (TIN).

This function supports 3d and will not drop the z-index.

Examples

```
SELECT Box3D(ST_GeomFromEWKT('LINESTRING(1 2 3, 3 4 5, 5 6 5)'));

Box3d
---------
BOX3D(1 2 3,5 6 5)
```

```
SELECT Box3D(ST_GeomFromEWKT('CIRCULARSTRING(220268 150415 1,220227 150505 1,220227 150406 ↩
    1)'));

Box3d
--------
BOX3D(220227 150406 1,220268 150415 1)
```

Siehe auch

Box2D, ST_GeomFromEWKT

7.18.3 ST_EstimatedExtent

ST_EstimatedExtent — Returns the estimated extent of a spatial table.

Synopsis

box2d **ST_EstimatedExtent**(text schema_name, text table_name, text geocolumn_name, boolean parent_only);
box2d **ST_EstimatedExtent**(text schema_name, text table_name, text geocolumn_name);
box2d **ST_EstimatedExtent**(text table_name, text geocolumn_name);

Beschreibung

Returns the estimated extent of a spatial table as a box2d. The current schema is used if not specified. The estimated extent is taken from the geometry column's statistics. This is usually much faster than computing the exact extent of the table using ST_Extent or ST_3DExtent.

The default behavior is to also use statistics collected from child tables (tables with INHERITS) if available. If `parent_only` is set to TRUE, only statistics for the given table are used and child tables are ignored.

For PostgreSQL >= 8.0.0 statistics are gathered by VACUUM ANALYZE and the result extent will be about 95% of the actual one. For PostgreSQL < 8.0.0 statistics are gathered by running `update_geometry_stats()` and the result extent is exact.

Note
In the absence of statistics (empty table or no ANALYZE called) this function returns NULL. Prior to version 1.5.4 an exception was thrown instead.

Availability: 1.0.0

Changed: 2.1.0. Up to 2.0.x this was called ST_Estimated_Extent.

This method supports Circular Strings and Curves.

Examples

```
SELECT ST_EstimatedExtent('ny', 'edges', 'geom');
--result--
BOX(-8877653 4912316,-8010225.5 5589284)

SELECT ST_EstimatedExtent('feature_poly', 'geom');
--result--
BOX(-124.659652709961 24.6830825805664,-67.7798080444336 49.0012092590332)
```

Siehe auch

ST_Extent, ST_3DExtent

7.18.4 ST_Expand

ST_Expand — Returns a bounding box expanded from another bounding box or a geometry.

Synopsis

geometry **ST_Expand**(geometry geom, float units_to_expand);
geometry **ST_Expand**(geometry geom, float dx, float dy, float dz=0, float dm=0);
box2d **ST_Expand**(box2d box, float units_to_expand);
box2d **ST_Expand**(box2d box, float dx, float dy);
box3d **ST_Expand**(box3d box, float units_to_expand);
box3d **ST_Expand**(box3d box, float dx, float dy, float dz=0);

Beschreibung

Returns a bounding box expanded from the bounding box of the input, either by specifying a single distance with which the box should be expanded on both axes, or by specifying an expansion distance for each axis. Uses double-precision. Can be used for distance queries, or to add a bounding box filter to a query to take advantage of a spatial index.

In addition to the version of ST_Expand accepting and returning a geometry, variants are provided that accept and return box2d and box3d data types.

Distances are in the units of the spatial reference system of the input.

ST_Expand is similar to ST_Buffer, except while buffering expands a geometry in all directions, ST_Expand expands the bounding box along each axis.

Note
Pre version 1.3, ST_Expand was used in conjunction with ST_Distance to do indexable distance queries. For example, `geom && ST_Expand('POINT(10 20)', 10) AND ST_Distance(geom, 'POINT(10 20)') < 10`. This has been replaced by the simpler and more efficient ST_DWithin function.

Availability: 1.5.0 behavior changed to output double precision instead of float4 coordinates.

Enhanced: 2.0.0 support for Polyhedral surfaces, Triangles and TIN was introduced.

Enhanced: 2.3.0 support was added to expand a box by different amounts in different dimensions.

This function supports Polyhedral surfaces.

This function supports Triangles and Triangulated Irregular Network Surfaces (TIN).

Examples

Note
Examples below use US National Atlas Equal Area (SRID=2163) which is a meter projection

```
--10 meter expanded box around bbox of a linestring
SELECT CAST(ST_Expand(ST_GeomFromText('LINESTRING(2312980 110676,2312923 110701,2312892  ↩
    110714)', 2163),10) As box2d);
                                        st_expand
-----------------------------------
 BOX(2312882 110666,2312990 110724)

--10 meter expanded 3D box of a 3D box
SELECT ST_Expand(CAST('BOX3D(778783 2951741 1,794875 2970042.61545891 10)' As box3d),10)
                                                        st_expand
-----------------------------------------------------
 BOX3D(778773 2951731 -9,794885 2970052.61545891 20)

 --10 meter geometry astext rep of a expand box around a point geometry
 SELECT ST_AsEWKT(ST_Expand(ST_GeomFromEWKT('SRID=2163;POINT(2312980 110676)'),10));
                                                                                    st_asewkt ↩

------------------------------------------------------------------------------------------------ ↩

 SRID=2163;POLYGON((2312970 110666,2312970 110686,2312990 110686,2312990 110666,2312970  ↩
     110666))
```

Siehe auch

ST_Buffer, ST_DWithin, ST_SRID

7.18.5 ST_Extent

ST_Extent — Aggregate function that returns the bounding box of geometries.

Synopsis

box2d **ST_Extent**(geometry set geomfield);

Beschreibung

An aggregate function that returns a box2d bounding box that bounds a set of geometries.

The bounding box coordinates are in the spatial reference system of the input geometries.

ST_Extent is similar in concept to Oracle Spatial/Locator's SDO_AGGR_MBR.

Note
ST_Extent returns boxes with only X and Y ordinates even with 3D geometries. To return XYZ ordinates use ST_3DExtent.

Note
The returned `box3d` value does not include a SRID. Use ST_SetSRID to convert it into a geometry with SRID metadata. The SRID is the same as the input geometries.

Enhanced: 2.0.0 support for Polyhedral surfaces, Triangles and TIN was introduced.

This function supports Polyhedral surfaces.

This function supports Triangles and Triangulated Irregular Network Surfaces (TIN).

Examples

Note
Examples below use Massachusetts State Plane ft (SRID=2249)

```
SELECT ST_Extent(geom) as bextent FROM sometable;
                                          st_bextent
------------------------------------
BOX(739651.875 2908247.25,794875.8125 2970042.75)


--Return extent of each category of geometries
SELECT ST_Extent(geom) as bextent
FROM sometable
```

```
GROUP BY category ORDER BY category;

                                          bextent                        |         name
---------------------------------------------------------+----------------
 BOX(778783.5625 2951741.25,794875.8125 2970042.75) | A
 BOX(751315.8125 2919164.75,765202.6875 2935417.25) | B
 BOX(739651.875 2917394.75,756688.375 2935866)      | C

 --Force back into a geometry
 -- and render the extended text representation of that geometry
SELECT ST_SetSRID(ST_Extent(geom),2249) as bextent FROM sometable;

                               bextent
-------------------------------------------------------------------------------
 SRID=2249;POLYGON((739651.875 2908247.25,739651.875 2970042.75,794875.8125 2970042.75,
 794875.8125 2908247.25,739651.875 2908247.25))
```

Siehe auch

ST_EstimatedExtent, ST_3DExtent, ST_SetSRID

7.18.6 ST_3DExtent

ST_3DExtent — Aggregate function that returns the 3D bounding box of geometries.

Synopsis

box3d **ST_3DExtent**(geometry set geomfield);

Beschreibung

An aggregate function that returns a box3d (includes Z ordinate) bounding box that bounds a set of geometries.

The bounding box coordinates are in the spatial reference system of the input geometries.

Note
The returned `box3d` value does not include a SRID. Use ST_SetSRID to convert it into a geometry with SRID metadata. The SRID is the same as the input geometries.

Enhanced: 2.0.0 support for Polyhedral surfaces, Triangles and TIN was introduced.

Changed: 2.0.0 In prior versions this used to be called ST_Extent3D

This function supports 3d and will not drop the z-index.

This method supports Circular Strings and Curves.

This function supports Polyhedral surfaces.

This function supports Triangles and Triangulated Irregular Network Surfaces (TIN).

Examples

```
SELECT ST_3DExtent(foo.geom) As b3extent
FROM (SELECT ST_MakePoint(x,y,z) As geom
        FROM generate_series(1,3) As x
                CROSS JOIN generate_series(1,2) As y
                CROSS JOIN generate_series(0,2) As Z) As foo;
          b3extent
--------------------
 BOX3D(1 1 0,3 2 2)

--Get the extent of various elevated circular strings
SELECT ST_3DExtent(foo.geom) As b3extent
FROM (SELECT ST_Translate(ST_Force_3DZ(ST_LineToCurve(ST_Buffer(ST_Point(x,y),1))),0,0,z)  ↩
    As geom
        FROM generate_series(1,3) As x
                CROSS JOIN generate_series(1,2) As y
                CROSS JOIN generate_series(0,2) As Z) As foo;

        b3extent
--------------------
 BOX3D(1 0 0,4 2 2)
```

Siehe auch

ST_Extent, ST_Force3DZ, ST_SetSRID

7.18.7 ST_MakeBox2D

ST_MakeBox2D — Creates a BOX2D defined by two 2D point geometries.

Synopsis

box2d **ST_MakeBox2D**(geometry pointLowLeft, geometry pointUpRight);

Beschreibung

Creates a box2d defined by two Point geometries. This is useful for doing range queries.

Examples

```
--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)
```

Siehe auch

ST_Point, ST_SetSRID, ST_SRID

7.18.8 ST_3DMakeBox

ST_3DMakeBox — Creates a BOX3D defined by two 3D point geometries.

Synopsis

box3d **ST_3DMakeBox**(geometry point3DLowLeftBottom, geometry point3DUpRightTop);

Beschreibung

Creates a box3d defined by two 3D Point geometries.

This function supports 3D and will not drop the z-index.

Changed: 2.0.0 In prior versions this used to be called ST_MakeBox3D

Examples

```
SELECT ST_3DMakeBox(ST_MakePoint(-989502.1875, 528439.5625, 10),
        ST_MakePoint(-987121.375 ,529933.1875, 10)) As abb3d

--bb3d--
--------
BOX3D(-989502.1875 528439.5625 10,-987121.375 529933.1875 10)
```

Siehe auch

ST_MakePoint, ST_SetSRID, ST_SRID

7.18.9 ST_XMax

ST_XMax — Returns the X maxima of a 2D or 3D bounding box or a geometry.

Synopsis

float **ST_XMax**(box3d aGeomorBox2DorBox3D);

Beschreibung

Returns the X maxima of a 2D or 3D bounding box or a geometry.

Note
Although this function is only defined for box3d, it also works for box2d and geometry values due to automatic casting. However, it will not accept a geometry or box2d text representation, since those do not auto-cast.

This function supports 3d and will not drop the z-index.

This method supports Circular Strings and Curves.

Examples

```
SELECT ST_XMax('BOX3D(1 2 3, 4 5 6)');
st_xmax
-------
4

SELECT ST_XMax(ST_GeomFromText('LINESTRING(1 3 4, 5 6 7)'));
st_xmax
-------
5

SELECT ST_XMax(CAST('BOX(-3 2, 3 4)' As box2d));
st_xmax
-------
3
--Observe THIS DOES NOT WORK because it will try to auto-cast the string representation to ↩
    a BOX3D
SELECT ST_XMax('LINESTRING(1 3, 5 6)');

--ERROR:  BOX3D parser - doesn't start with BOX3D(

SELECT ST_XMax(ST_GeomFromEWKT('CIRCULARSTRING(220268 150415 1,220227 150505 2,220227 ↩
    150406 3)'));
st_xmax
--------
220288.248780547
```

Siehe auch

ST_XMin, ST_YMax, ST_YMin, ST_ZMax, ST_ZMin

7.18.10 ST_XMin

ST_XMin — Returns the X minima of a 2D or 3D bounding box or a geometry.

Synopsis

float **ST_XMin**(box3d aGeomorBox2DorBox3D);

Beschreibung

Returns the X minima of a 2D or 3D bounding box or a geometry.

Note
Although this function is only defined for box3d, it also works for box2d and geometry values due to automatic casting. However it will not accept a geometry or box2d text representation, since those do not auto-cast.

This function supports 3d and will not drop the z-index.

This method supports Circular Strings and Curves.

Examples

```
SELECT ST_XMin('BOX3D(1 2 3, 4 5 6)');
st_xmin
-------
1

SELECT ST_XMin(ST_GeomFromText('LINESTRING(1 3 4, 5 6 7)'));
st_xmin
-------
1

SELECT ST_XMin(CAST('BOX(-3 2, 3 4)' As box2d));
st_xmin
-------
-3
--Observe THIS DOES NOT WORK because it will try to auto-cast the string representation to ↩
    a BOX3D
SELECT ST_XMin('LINESTRING(1 3, 5 6)');

--ERROR:  BOX3D parser - doesn't start with BOX3D(

SELECT ST_XMin(ST_GeomFromEWKT('CIRCULARSTRING(220268 150415 1,220227 150505 2,220227 ↩
    150406 3)'));
st_xmin
--------
220186.995121892
```

Siehe auch

ST_XMax, ST_YMax, ST_YMin, ST_ZMax, ST_ZMin

7.18.11 ST_YMax

ST_YMax — Returns the Y maxima of a 2D or 3D bounding box or a geometry.

Synopsis

float **ST_YMax**(box3d aGeomorBox2DorBox3D);

Beschreibung

Returns the Y maxima of a 2D or 3D bounding box or a geometry.

Note
Although this function is only defined for box3d, it also works for box2d and geometry values due to automatic casting. However it will not accept a geometry or box2d text representation, since those do not auto-cast.

This function supports 3d and will not drop the z-index.

This method supports Circular Strings and Curves.

Examples

```
SELECT ST_YMax('BOX3D(1 2 3, 4 5 6)');
st_ymax
-------
5

SELECT ST_YMax(ST_GeomFromText('LINESTRING(1 3 4, 5 6 7)'));
st_ymax
-------
6

SELECT ST_YMax(CAST('BOX(-3 2, 3 4)' As box2d));
st_ymax
-------
4
--Observe THIS DOES NOT WORK because it will try to auto-cast the string representation to ↩
    a BOX3D
SELECT ST_YMax('LINESTRING(1 3, 5 6)');

--ERROR:  BOX3D parser - doesn't start with BOX3D(

SELECT ST_YMax(ST_GeomFromEWKT('CIRCULARSTRING(220268 150415 1,220227 150505 2,220227 ↩
    150406 3)'));
st_ymax
--------
150506.126829327
```

Siehe auch

ST_XMin, ST_XMax, ST_YMin, ST_ZMax, ST_ZMin

7.18.12 ST_YMin

ST_YMin — Returns the Y minima of a 2D or 3D bounding box or a geometry.

Synopsis

float **ST_YMin**(box3d aGeomorBox2DorBox3D);

Beschreibung

Returns the Y minima of a 2D or 3D bounding box or a geometry.

Note
Although this function is only defined for box3d, it also works for box2d and geometry values due to automatic casting. However it will not accept a geometry or box2d text representation, since those do not auto-cast.

This function supports 3d and will not drop the z-index.

This method supports Circular Strings and Curves.

Examples

```
SELECT ST_YMin('BOX3D(1 2 3, 4 5 6)');
st_ymin
-------
2

SELECT ST_YMin(ST_GeomFromText('LINESTRING(1 3 4, 5 6 7)'));
st_ymin
-------
3

SELECT ST_YMin(CAST('BOX(-3 2, 3 4)' As box2d));
st_ymin
-------
2
--Observe THIS DOES NOT WORK because it will try to auto-cast the string representation to ↩
    a BOX3D
SELECT ST_YMin('LINESTRING(1 3, 5 6)');

--ERROR:  BOX3D parser - doesn't start with BOX3D(

SELECT ST_YMin(ST_GeomFromEWKT('CIRCULARSTRING(220268 150415 1,220227 150505 2,220227  ↩
    150406 3)'));
st_ymin
--------
150406
```

Siehe auch

ST_GeomFromEWKT, ST_XMin, ST_XMax, ST_YMax, ST_ZMax, ST_ZMin

7.18.13 ST_ZMax

ST_ZMax — Returns the Z maxima of a 2D or 3D bounding box or a geometry.

Synopsis

float **ST_ZMax**(box3d aGeomorBox2DorBox3D);

Beschreibung

Returns the Z maxima of a 2D or 3D bounding box or a geometry.

Note
Although this function is only defined for box3d, it also works for box2d and geometry values due to automatic casting. However it will not accept a geometry or box2d text representation, since those do not auto-cast.

This function supports 3d and will not drop the z-index.

This method supports Circular Strings and Curves.

Examples

```
SELECT ST_ZMax('BOX3D(1 2 3, 4 5 6)');
st_zmax
-------
6

SELECT ST_ZMax(ST_GeomFromEWKT('LINESTRING(1 3 4, 5 6 7)'));
st_zmax
-------
7

SELECT ST_ZMax('BOX3D(-3 2 1, 3 4 1)' );
st_zmax
-------
1
--Observe THIS DOES NOT WORK because it will try to auto-cast the string representation to ↩
    a BOX3D
SELECT ST_ZMax('LINESTRING(1 3 4, 5 6 7)');

--ERROR:  BOX3D parser - doesn't start with BOX3D(

SELECT ST_ZMax(ST_GeomFromEWKT('CIRCULARSTRING(220268 150415 1,220227 150505 2,220227  ↩
    150406 3)'));
st_zmax
--------
3
```

Siehe auch

ST_GeomFromEWKT, ST_XMin, ST_XMax, ST_YMax, ST_YMin, ST_ZMax

7.18.14 ST_ZMin

ST_ZMin — Returns the Z minima of a 2D or 3D bounding box or a geometry.

Synopsis

float **ST_ZMin**(box3d aGeomorBox2DorBox3D);

Beschreibung

Returns the Z minima of a 2D or 3D bounding box or a geometry.

Note
Although this function is only defined for box3d, it also works for box2d and geometry values due to automatic casting. However it will not accept a geometry or box2d text representation, since those do not auto-cast.

This function supports 3d and will not drop the z-index.

This method supports Circular Strings and Curves.

Examples

```
SELECT ST_ZMin('BOX3D(1 2 3, 4 5 6)');
st_zmin
-------
3

SELECT ST_ZMin(ST_GeomFromEWKT('LINESTRING(1 3 4, 5 6 7)'));
st_zmin
-------
4

SELECT ST_ZMin('BOX3D(-3 2 1, 3 4 1)' );
st_zmin
-------
1
--Observe THIS DOES NOT WORK because it will try to auto-cast the string representation to ↩
    a BOX3D
SELECT ST_ZMin('LINESTRING(1 3 4, 5 6 7)');

--ERROR:  BOX3D parser - doesn't start with BOX3D(

SELECT ST_ZMin(ST_GeomFromEWKT('CIRCULARSTRING(220268 150415 1,220227 150505 2,220227  ↩
    150406 3)'));
st_zmin
--------
1
```

Siehe auch

ST_GeomFromEWKT, ST_GeomFromText, ST_XMin, ST_XMax, ST_YMax, ST_YMin, ST_ZMax

7.19 Kilometrierung

7.19.1 ST_LineInterpolatePoint

ST_LineInterpolatePoint — Gibt einen oder mehrere, entlang einer Linie interpolierte Punkte zurück.

Synopsis

geometry **ST_LineInterpolatePoint**(geometry a_linestring, float8 a_fraction);
geography **ST_LineInterpolatePoint**(geography a_linestring, float8 a_fraction, boolean use_spheroid = true);

Beschreibung

Fügt einen Punkt entlang einer Linie ein. Der erste Parameter muss einen Linienzug beschreiben. Der zweite Parameter, in Float8-Darstellung mit den Werten von 0 bis 1, gibt jenen Bruchteil der Gesamtlänge des Linienzuges an, wo der Punkt liegen soll.

Siehe ST_LineLocatePoint um die nächstliegende Linie zu einem Punkt zu berechnen.

Note
This function computes points in 2D and then interpolates values for Z and M, while ST_3DLineInterpolatePoint computes points in 3D and only interpolates the M value.

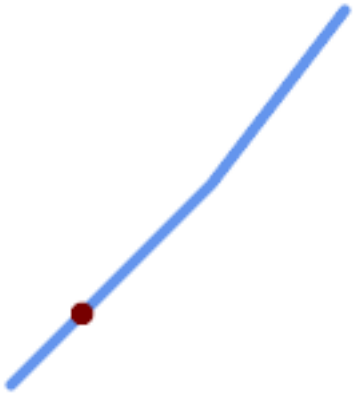
Note
Ab Version 1.1.1 interpoliert diese Funktion auch M- und Z-Werte (falls vorhanden), während frühere Versionen diese Werte auf 0.0 setzten.

Verfügbarkeit: 0.8.2, Z und M Unterstützung wurde mit 1.1.1 hinzugefügt

Änderung: 2.1.0. Bis zu 2.0.x wurde diese Funktion mit ST_Line_Interpolate_Point bezeichnet.

This function supports 3d and will not drop the z-index.

Beispiele



Ein Linienzug mit dem interpolierten Punkt bei Position 0.20 (20%)

```
-- The point 20% along a line

SELECT ST_AsEWKT(  ST_LineInterpolatePoint(
        'LINESTRING(25 50, 100 125, 150 190)',
        0.2 ));
-----------------
 POINT(51.5974135047432 76.5974135047432)
```

Gibt einen oder mehrere, entlang einer Linie interpolierte Punkte zurück.

```
SELECT ST_AsEWKT(  ST_LineInterpolatePoint('
        LINESTRING(1 2 3, 4 5 6, 6 7 8)',
        0.5 ));
--------------------
 POINT(3.5 4.5 5.5)
```

The closest point on a line to a point:

```
SELECT ST_AsText( ST_LineInterpolatePoint( line.geom,
                      ST_LineLocatePoint( line.geom, 'POINT(4 3)')))
FROM (SELECT ST_GeomFromText('LINESTRING(1 2, 4 5, 6 7)') As geom) AS line;
------------
 POINT(3 4)
```

Siehe auch

ST_LineInterpolatePoints, ST_LineInterpolatePoint, ST_LineMerge

7.19.2 ST_3DLineInterpolatePoint

ST_3DLineInterpolatePoint — Gibt einen oder mehrere, entlang einer Linie interpolierte Punkte zurück.

Synopsis

geometry **ST_LineInterpolatePoint**(geometry a_linestring, float8 a_fraction);

Beschreibung

Fügt einen Punkt entlang einer Linie ein. Der erste Parameter muss einen Linienzug beschreiben. Der zweite Parameter, in Float8-Darstellung mit den Werten von 0 bis 1, gibt jenen Bruchteil der Gesamtlänge des Linienzuges an, wo der Punkt liegen soll.

Note
ST_LineInterpolatePoint computes points in 2D and then interpolates the values for Z and M, while this function computes points in 3D and only interpolates the M value.

Verfügbarkeit: 2.0.0

This function supports 3d and will not drop the z-index.

Beispiele

Gibt einen oder mehrere, entlang einer Linie interpolierte Punkte zurück.

```
SELECT ST_AsText(
    ST_3DLineInterpolatePoint('LINESTRING(25 50 70, 100 125 90, 150 190 200)',
        0.20));

   st_asetext
----------------
 POINT Z (59.0675892910822 84.0675892910822 79.0846904776219)
```

Siehe auch

ST_LineInterpolatePoint, ST_LineInterpolatePoint, ST_LineMerge

7.19.3 ST_LineInterpolatePoints

ST_LineInterpolatePoints — Gibt einen oder mehrere, entlang einer Linie interpolierte Punkte zurück.

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);

Beschreibung

Returns one or more points interpolated along a line at a fractional interval. The first argument must be a LINESTRING. The second argument is a float8 between 0 and 1 representing the spacing between the points as a fraction of line length. If the third argument is false, at most one point will be constructed (which is equivalent to ST_LineInterpolatePoint.)

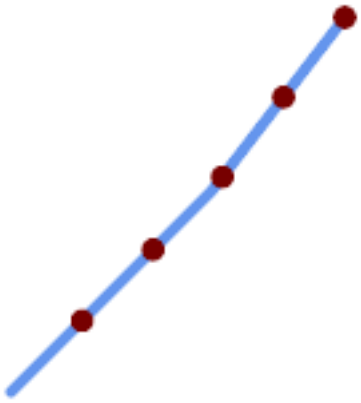
If the result has zero or one points, it is returned as a POINT. If it has two or more points, it is returned as a MULTIPOINT.

Verfügbarkeit: 2.5.0

This function supports 3d and will not drop the z-index.

This function supports M coordinates.

Beispiele



A LineString with points interpolated every 20%

```
--Return points each 20% along a 2D line
SELECT ST_AsText(ST_LineInterpolatePoints('LINESTRING(25 50, 100 125, 150 190)', 0.20))
----------------
 MULTIPOINT((51.5974135047432 76.5974135047432),(78.1948270094864 103.194827009486) ↩
    ,(104.132163186446 130.37181214238),(127.066081593223 160.18590607119),(150 190))
```

Siehe auch

ST_LineInterpolatePoint, ST_LineLocatePoint

7.19.4 ST_LineLocatePoint

ST_LineLocatePoint — Returns the fractional location of the closest point on a line to a point.

Synopsis

float8 **ST_LineLocatePoint**(geometry a_linestring, geometry a_point);
float8 **ST_LineLocatePoint**(geography a_linestring, geography a_point, boolean use_spheroid = true);

Beschreibung

Gibt eine Gleitpunktzahl zwischen 0 und 1 zurück, welche die Lage des Punktes auf einer Linie angibt, der zu einem gegebenen Punkt am nächsten liegt. Die Lage wird als Anteil an der Gesamtlänge der 2D Linie angegeben.

Sie können die zurückgegebene Lage nutzen, um einen Punkt (ST_LineInterpolatePoint) oder eine Teilzeichenfolge (ST_LineSubstring) zu extrahieren.

Nützlich, um die Hausnummern von Adressen anzunähern

Verfügbarkeit: 1.1.0

Änderung: 2.1.0. Bis zu 2.0.x wurde diese Funktion mit ST_Line_Locate_Point bezeichnet.

Beispiele

```
--Rough approximation of finding the street number of a point along the street
--Note the whole foo thing is just to generate dummy data that looks
--like house centroids and street
--We use ST_DWithin to exclude
--houses too far away from the street to be considered on the street
SELECT ST_AsText(house_loc) As as_text_house_loc,
        startstreet_num +
                CAST( (endstreet_num - startstreet_num)
                        * ST_LineLocatePoint(street_line, house_loc) As integer) As  ↩
                            street_num
FROM
(SELECT ST_GeomFromText('LINESTRING(1 2, 3 4)') As street_line,
        ST_Point(x*1.01,y*1.03) As house_loc, 10 As startstreet_num,
                20 As endstreet_num
FROM generate_series(1,3) x CROSS JOIN generate_series(2,4) As y)
As foo
WHERE ST_DWithin(street_line, house_loc, 0.2);

 as_text_house_loc | street_num
-------------------+------------
 POINT(1.01 2.06)  |         10
 POINT(2.02 3.09)  |         15
 POINT(3.03 4.12)  |         20

 --find closest point on a line to a point or other geometry
 SELECT ST_AsText(ST_LineInterpolatePoint(foo.the_line, ST_LineLocatePoint(foo.the_line,  ↩
     ST_GeomFromText('POINT(4 3)'))))
FROM (SELECT ST_GeomFromText('LINESTRING(1 2, 4 5, 6 7)') As the_line) As foo;
   st_astext
----------------
 POINT(3 4)
```

Siehe auch

ST_DWithin, ST_Length2D, ST_LineInterpolatePoint, ST_LineSubstring

7.19.5 ST_LineSubstring

ST_LineSubstring — Returns the part of a line between two fractional locations.

Synopsis

geometry **ST_LineSubstring**(geometry a_linestring, float8 startfraction, float8 endfraction);
geography **ST_LineSubstring**(geography a_linestring, float8 startfraction, float8 endfraction);

Beschreibung

Computes the line which is the section of the input line starting and ending at the given fractional locations. The first argument must be a LINESTRING. The second and third arguments are values in the range [0, 1] representing the start and end locations as fractions of line length. The Z and M values are interpolated for added endpoints if present.

Gleichbedeutend mit ST_LineInterpolatePoint, wenn Anfangswert und Endwert ident sind.

Note
This only works with LINESTRINGs. To use on contiguous MULTILINESTRINGs first join them with ST_LineMerge.

Note
Ab Version 1.1.1 interpoliert diese Funktion auch M- und Z-Werte (falls vorhanden), während frühere Versionen unbestimmte Werte setzten.

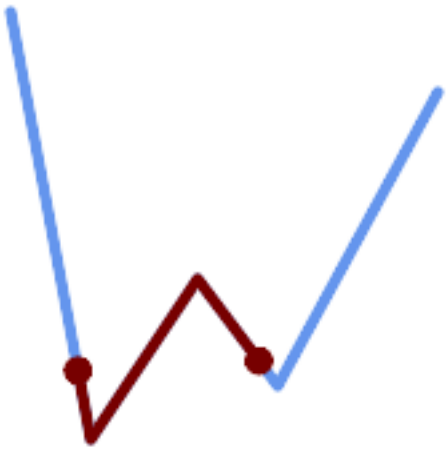
Enhanced: 3.4.0 - Support for geography was introduced.

Änderung: 2.1.0. Bis zu 2.0.x wurde diese Funktion mit ST_Line_Substring bezeichnet.

Verfügbarkeit: 1.1.0, mit 1.1.1 wurde die Unterstützung für Z und M hinzugefügt

This function supports 3d and will not drop the z-index.

Beispiele



Ein Linienstück von einem Mittelstück mit 1/3 Länge überlagert (0.333, 0.666)

```
SELECT ST_AsText(ST_LineSubstring( 'LINESTRING (20 180, 50 20, 90 80, 120 40, 180 150)',  ↩
    0.333, 0.666));
------------------------------------------------------------------------------------------- ↩

LINESTRING (45.17311810399485 45.74337011202746, 50 20, 90 80, 112.97593050157862  ↩
    49.36542599789519)
```

If start and end locations are the same, the result is a POINT.

```
SELECT ST_AsText(ST_LineSubstring( 'LINESTRING(25 50, 100 125, 150 190)', 0.333, 0.333));
----------------------------------------
 POINT(69.2846934853974 94.2846934853974)
```

A query to cut a LineString into sections of length 100 or shorter. It uses `generate_series()` with a CROSS JOIN LATERAL to produce the equivalent of a FOR loop.

```
WITH data(id, geom) AS (VALUES
        ( 'A', 'LINESTRING( 0 0, 200 0)'::geometry ),
        ( 'B', 'LINESTRING( 0 100, 350 100)'::geometry ),
        ( 'C', 'LINESTRING( 0 200, 50 200)'::geometry )
    )
SELECT id, i,
       ST_AsText( ST_LineSubstring( geom, startfrac, LEAST( endfrac, 1 )) ) AS geom
FROM (
    SELECT id, geom, ST_Length(geom) len, 100 sublen FROM data
    ) AS d
CROSS JOIN LATERAL (
    SELECT i, (sublen * i) / len AS startfrac,
              (sublen * (i+1)) / len AS endfrac
    FROM generate_series(0, floor( len / sublen )::integer ) AS t(i)
    -- skip last i if line length is exact multiple of sublen
    WHERE (sublen * i) / len <> 1.0
    ) AS d2;

 id | i |            geom
----+---+-----------------------------
 A  | 0 | LINESTRING(0 0,100 0)
 A  | 1 | LINESTRING(100 0,200 0)
 B  | 0 | LINESTRING(0 100,100 100)
 B  | 1 | LINESTRING(100 100,200 100)
 B  | 2 | LINESTRING(200 100,300 100)
 B  | 3 | LINESTRING(300 100,350 100)
 C  | 0 | LINESTRING(0 200,50 200)
```

Geography implementation measures along a spheroid, geometry along a line

```
SELECT ST_AsText(ST_LineSubstring( 'LINESTRING(-118.2436 34.0522, -71.0570 42.3611)':: ↩
    geography, 0.333, 0.666),6) AS geog_sub
 , ST_AsText(ST_LineSubstring('LINESTRING(-118.2436 34.0522, -71.0570 42.3611)'::geometry,  ↩
     0.333, 0.666),6) AS geom_sub;
-------------------------------------------------------------
geog_sub | LINESTRING(-104.167064 38.854691,-87.674646 41.849854)
geom_sub | LINESTRING(-102.530462 36.819064,-86.817324 39.585927)
```

Siehe auch

ST_Length, ST_LineInterpolatePoint, ST_LineMerge

7.19.6 ST_LocateAlong

ST_LocateAlong — Returns the point(s) on a geometry that match a measure value.

Synopsis

geometry **ST_LocateAlong**(geometry ageom_with_measure, float8 a_measure, float8 offset);

Beschreibung

Returns the location(s) along a measured geometry that have the given measure values. The result is a Point or MultiPoint. Polygonal inputs are not supported.

If `offset` is provided, the result is offset to the left or right of the input line by the specified distance. A positive offset will be to the left, and a negative one to the right.

Note
Use this function only for linear geometries with an M component

The semantic is specified by the *ISO/IEC 13249-3 SQL/MM Spatial* standard.

Verfügbarkeit: 1.1.0 über die alte Bezeichnung ST_Locate_Along_Measure.

Änderung: 2.0.0 In Vorgängerversionen als ST_Locate_Along_Measure bezeichet. Der alte Name ist überholt und wird in der Zukunft entfernt ist aber noch verfügbar.

This function supports M coordinates.

This method implements the SQL/MM specification. SQL-MM IEC 13249-3: 5.1.13

Beispiele

```
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))
```

Siehe auch

ST_LocateBetween, ST_LocateBetweenElevations, ST_InterpolatePoint

7.19.7 ST_LocateBetween

ST_LocateBetween — Returns the portions of a geometry that match a measure range.

Synopsis

geometry **ST_LocateBetween**(geometry geomA, float8 measure_start, float8 measure_end, float8 offset);

Beschreibung

Gibt eine abgeleitete Sammelgeometrie mit jenen Elementen zurück, die in dem gegebenen Kilometrierungsintervall liegen; das Intervall ist unbeschränkt. Polygonale Elemente werden nicht untersützt.

Wenn ein Versatz angegeben ist, werden die Resultierenden um diese Anzahl an Einheiten nach links oder rechts von der gegebenen Linie versetzt. Ein positiver Versatz geschieht nach links, ein negativer nach rechts.

Clipping a non-convex POLYGON may produce invalid geometry.

The semantic is specified by the *ISO/IEC 13249-3 SQL/MM Spatial* standard.

Verfügbarkeit: 1.1.0 über die alte Bezeichnung ST_Locate_Between_Measures.

Änderung: 2.0.0 In Vorgängerversionen als ST_Locate_Along_Measure bezeichet. Der alte Name ist überholt und wird in der Zukunft entfernt ist aber noch verfügbar.

Enhanced: 3.0.0 - added support for POLYGON, TIN, TRIANGLE.

This function supports M coordinates.

This method implements the SQL/MM specification. SQL-MM IEC 13249-3: 5.1

Beispiele

```
SELECT ST_AsText(
  ST_LocateBetween(
       'MULTILINESTRING M ((1 2 3, 3 4 2, 9 4 3),(1 2 3, 5 4 5))':: geometry,
       1.5, 3 ));
------------------------------------------------------------------------
 GEOMETRYCOLLECTION M (LINESTRING M (1 2 3,3 4 2,9 4 3),POINT M (1 2 3))
```

A LineString with the section between measures 2 and 8, offset to the left

```
SELECT ST_AsText( ST_LocateBetween(
  ST_AddMeasure('LINESTRING (20 180, 50 20, 100 120, 180 20)', 0, 10),
  2, 8,
  20
));
------------------------------------------------------------------------
```

```
MULTILINESTRING((54.49835019899045 104.53426957938231,58.70056060327303  ↩
    82.12248075654186,69.16695286779743 103.05526528559065,82.11145618000168  ↩
    128.94427190999915,84.24893681714357 132.32493442618113,87.01636951231555  ↩
    135.21267035596549,90.30307285299679 137.49198684843182,93.97759758337769  ↩
    139.07172433557758,97.89298381958797 139.8887023914453,101.89263860095893  ↩
    139.9102465862721,105.81659870902816 139.13549527600819,109.50792827749828  ↩
    137.5954340631298,112.81899532549731 135.351656550512,115.6173761888606  ↩
    132.49390095108848,145.31017306064817 95.37790486135405))
```

Siehe auch

ST_LocateAlong, ST_LocateAlong, ST_LocateBetween

7.19.8 ST_LocateBetweenElevations

ST_LocateBetweenElevations — Returns the portions of a geometry that lie in an elevation (Z) range.

Synopsis

geometry **ST_LocateBetweenElevations**(geometry geom_mline, float8 elevation_start, float8 elevation_end);

Beschreibung

Returns a geometry (collection) with the portions of a geometry that lie in an elevation (Z) range.

Clipping a non-convex POLYGON may produce invalid geometry.

Verfügbarkeit: 1.4.0

Enhanced: 3.0.0 - added support for POLYGON, TIN, TRIANGLE.

This function supports 3d and will not drop the z-index.

Beispiele

```
SELECT ST_AsText(
  ST_LocateBetweenElevations(
    'LINESTRING(1 2 3, 4 5 6)'::geometry,
    2, 4 ));

             st_astext
-----------------------------------
 MULTILINESTRING Z ((1 2 3,2 3 4))

SELECT ST_AsText(
    ST_LocateBetweenElevations(
      'LINESTRING(1 2 6, 4 5 -1, 7 8 9)',
      6, 9)) As ewelev;

                                ewelev
----------------------------------------------------------------------
 GEOMETRYCOLLECTION Z (POINT Z (1 2 6),LINESTRING Z (6.1 7.1 6,7 8 9))
```

Siehe auch

ST_Dump, ST_LocateAlong, ST_LocateBetween

7.19.9 ST_InterpolatePoint

ST_InterpolatePoint — Für einen gegebenen Punkt wird die Kilometrierung auf dem nächstliegenden Punkt einer Geometrie zurück.

Synopsis

float8 **ST_InterpolatePoint**(geometry linear_geom_with_measure, geometry point);

Beschreibung

Returns an interpolated measure value of a linear measured geometry at the location closest to the given point.

Note
Use this function only for linear geometries with an M component

Verfügbarkeit: 2.0.0

This function supports 3d and will not drop the z-index.

Beispiele

```
SELECT ST_InterpolatePoint('LINESTRING M (0 0 0, 10 0 20)', 'POINT(5 5)');
 ---------------------
         10
```

Siehe auch

ST_AddMeasure, ST_LocateAlong, ST_LocateBetween

7.19.10 ST_AddMeasure

ST_AddMeasure — Interpolates measures along a linear geometry.

Synopsis

geometry **ST_AddMeasure**(geometry geom_mline, float8 measure_start, float8 measure_end);

Beschreibung

Gibt eine abgeleitete Geometrie mit einer zwischen Anfangs- und Endpunkt linear interpolierten Kilometrierung zurück. Wenn die Geometrie keine Dimension für die Kilometrierung aufweist, wird diese hinzugefügt. Wenn die Geometrie eine Dimension für die Kilometrierung hat, wird diese mit den neuen Werten überschrieben. Es werden nur LINESTRINGs und MULTILINESTRINGs unterstützt.

Verfügbarkeit: 1.5.0

This function supports 3d and will not drop the z-index.

Beispiele

```
SELECT ST_AsText(ST_AddMeasure(
ST_GeomFromEWKT('LINESTRING(1 0, 2 0, 4 0)'),1,4)) As ewelev;
           ewelev
-------------------------------
 LINESTRINGM(1 0 1,2 0 2,4 0 4)

SELECT ST_AsText(ST_AddMeasure(
ST_GeomFromEWKT('LINESTRING(1 0 4, 2 0 4, 4 0 4)'),10,40)) As ewelev;
                ewelev
---------------------------------------
 LINESTRING(1 0 4 10,2 0 4 20,4 0 4 40)

SELECT ST_AsText(ST_AddMeasure(
ST_GeomFromEWKT('LINESTRINGM(1 0 4, 2 0 4, 4 0 4)'),10,40)) As ewelev;
                ewelev
---------------------------------------
 LINESTRINGM(1 0 10,2 0 20,4 0 40)

SELECT ST_AsText(ST_AddMeasure(
ST_GeomFromEWKT('MULTILINESTRINGM((1 0 4, 2 0 4, 4 0 4),(1 0 4, 2 0 4, 4 0 4))'),10,70)) As ↩
     ewelev;
                             ewelev
-----------------------------------------------------------------
 MULTILINESTRINGM((1 0 10,2 0 20,4 0 40),(1 0 40,2 0 50,4 0 70))
```

7.20 Trajectory Functions

7.20.1 ST_IsValidTrajectory

ST_IsValidTrajectory — Tests if the geometry is a valid trajectory.

Synopsis

boolean **ST_IsValidTrajectory**(geometry line);

Beschreibung

Tests if a geometry encodes a valid trajectory. A valid trajectory is represented as a `LINESTRING` with measures (M values). The measure values must increase from each vertex to the next.

Valid trajectories are expected as input to spatio-temporal functions like ST_ClosestPointOfApproach

Availability: 2.2.0

This function supports 3d and will not drop the z-index.

Examples

```
-- A valid trajectory
SELECT ST_IsValidTrajectory(ST_MakeLine(
  ST_MakePointM(0,0,1),
  ST_MakePointM(0,1,2))
);
```

```
 t

-- An invalid trajectory
SELECT ST_IsValidTrajectory(ST_MakeLine(ST_MakePointM(0,0,1), ST_MakePointM(0,1,0)));
NOTICE:  Measure of vertex 1 (0) not bigger than measure of vertex 0 (1)
 st_isvalidtrajectory
----------------------
 f
```

Siehe auch

ST_ClosestPointOfApproach

7.20.2 ST_ClosestPointOfApproach

ST_ClosestPointOfApproach — Returns a measure at the closest point of approach of two trajectories.

Synopsis

float8 **ST_ClosestPointOfApproach**(geometry track1, geometry track2);

Beschreibung

Returns the smallest measure at which points interpolated along the given trajectories are at the smallest distance.

Inputs must be valid trajectories as checked by ST_IsValidTrajectory. Null is returned if the trajectories do not overlap in their M ranges.

See ST_LocateAlong for getting the actual points at the given measure.

Availability: 2.2.0

This function supports 3d and will not drop the z-index.

Examples

```
-- Return the time in which two objects moving between 10:00 and 11:00
-- are closest to each other and their distance at that point
WITH inp AS ( SELECT
  ST_AddMeasure('LINESTRING Z (0 0 0, 10 0 5)'::geometry,
    extract(epoch from '2015-05-26 10:00'::timestamptz),
    extract(epoch from '2015-05-26 11:00'::timestamptz)
  ) a,
  ST_AddMeasure('LINESTRING Z (0 2 10, 12 1 2)'::geometry,
    extract(epoch from '2015-05-26 10:00'::timestamptz),
    extract(epoch from '2015-05-26 11:00'::timestamptz)
  ) b
), cpa AS (
  SELECT ST_ClosestPointOfApproach(a,b) m FROM inp
), points AS (
  SELECT ST_Force3DZ(ST_GeometryN(ST_LocateAlong(a,m),1)) pa,
         ST_Force3DZ(ST_GeometryN(ST_LocateAlong(b,m),1)) pb
  FROM inp, cpa
)
SELECT to_timestamp(m) t,
       ST_Distance(pa,pb) distance
```

```
FROM points, cpa;

               t               |     distance
-------------------------------+------------------
 2015-05-26 10:45:31.034483+02 | 1.96036833151395
```

Siehe auch

ST_IsValidTrajectory, ST_DistanceCPA, ST_LocateAlong, ST_AddMeasure

7.20.3 ST_DistanceCPA

ST_DistanceCPA — Returns the distance between the closest point of approach of two trajectories.

Synopsis

float8 **ST_DistanceCPA**(geometry track1, geometry track2);

Beschreibung

Returns the minimum distance two moving objects have ever been each other.

Inputs must be valid trajectories as checked by ST_IsValidTrajectory. Null is returned if the trajectories do not overlap in their M ranges.

Availability: 2.2.0

This function supports 3d and will not drop the z-index.

Examples

```
-- Return the minimum distance of two objects moving between 10:00 and 11:00
WITH inp AS ( SELECT
  ST_AddMeasure('LINESTRING Z (0 0 0, 10 0 5)'::geometry,
    extract(epoch from '2015-05-26 10:00'::timestamptz),
    extract(epoch from '2015-05-26 11:00'::timestamptz)
  ) a,
  ST_AddMeasure('LINESTRING Z (0 2 10, 12 1 2)'::geometry,
    extract(epoch from '2015-05-26 10:00'::timestamptz),
    extract(epoch from '2015-05-26 11:00'::timestamptz)
  ) b
)
SELECT ST_DistanceCPA(a,b) distance FROM inp;

     distance
------------------
 1.96036833151395
```

Siehe auch

ST_IsValidTrajectory, ST_ClosestPointOfApproach, ST_AddMeasure, |=|

7.20.4 ST_CPAWithin

ST_CPAWithin — Tests if the closest point of approach of two trajectories is within the specified distance.

Synopsis

boolean **ST_CPAWithin**(geometry track1, geometry track2, float8 dist);

Beschreibung

Tests whether two moving objects have ever been closer than the specified distance.

Inputs must be valid trajectories as checked by ST_IsValidTrajectory. False is returned if the trajectories do not overlap in their M ranges.

Availability: 2.2.0

This function supports 3d and will not drop the z-index.

Examples

```
WITH inp AS ( SELECT
  ST_AddMeasure('LINESTRING Z (0 0 0, 10 0 5)'::geometry,
    extract(epoch from '2015-05-26 10:00'::timestamptz),
    extract(epoch from '2015-05-26 11:00'::timestamptz)
  ) a,
  ST_AddMeasure('LINESTRING Z (0 2 10, 12 1 2)'::geometry,
    extract(epoch from '2015-05-26 10:00'::timestamptz),
    extract(epoch from '2015-05-26 11:00'::timestamptz)
  ) b
)
SELECT ST_CPAWithin(a,b,2), ST_DistanceCPA(a,b) distance FROM inp;

 st_cpawithin |     distance
--------------+------------------
 t            | 1.96521473776207
```

Siehe auch

ST_IsValidTrajectory, ST_ClosestPointOfApproach, ST_DistanceCPA, |=|

7.21 SFCGAL Functions

7.21.1 postgis_sfcgal_version

postgis_sfcgal_version — Returns the version of SFCGAL in use

Synopsis

text **postgis_sfcgal_version**(void);

Beschreibung

Returns the version of SFCGAL in use

Verfügbarkeit: 2.1.0

This method needs SFCGAL backend.

This function supports 3d and will not drop the z-index.

This function supports Polyhedral surfaces.

This function supports Triangles and Triangulated Irregular Network Surfaces (TIN).

Siehe auch

postgis_sfcgal_full_version

7.21.2 postgis_sfcgal_full_version

postgis_sfcgal_full_version — Returns the full version of SFCGAL in use including CGAL and Boost versions

Synopsis

text **postgis_sfcgal_full_version**(void);

Beschreibung

Returns the full version of SFCGAL in use including CGAL and Boost versions

Verfügbarkeit: 2.3.0

This method needs SFCGAL backend.

This function supports 3d and will not drop the z-index.

This function supports Polyhedral surfaces.

This function supports Triangles and Triangulated Irregular Network Surfaces (TIN).

Siehe auch

postgis_sfcgal_version

7.21.3 ST_3DArea

ST_3DArea — Computes area of 3D surface geometries. Will return 0 for solids.

Synopsis

geometry **ST_ConvexHull**(geometry geomA);

Beschreibung

Verfügbarkeit: 2.1.0

This method needs SFCGAL backend.

This method implements the SQL/MM specification. SQL-MM IEC 13249-3: 8.1, 10.5

This function supports 3d and will not drop the z-index.

This function supports Polyhedral surfaces.

This function supports Triangles and Triangulated Irregular Network Surfaces (TIN).

Beispiele

Note: By default a PolyhedralSurface built from WKT is a surface geometry, not solid. It therefore has surface area. Once converted to a solid, no area.

```
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
```

Siehe auch

ST_Area, ST_ConcaveHull, ST_Dump, ST_Tesselate

7.21.4 ST_3DConvexHull

ST_3DConvexHull — Berechnet die konvexe Hülle einer Geometrie.

Synopsis

geometry **ST_3DConvexHull**(geometry geom1);

Beschreibung

Verfügbarkeit: 2.3.0

This method needs SFCGAL backend.

This function supports 3d and will not drop the z-index.

This function supports Polyhedral surfaces.

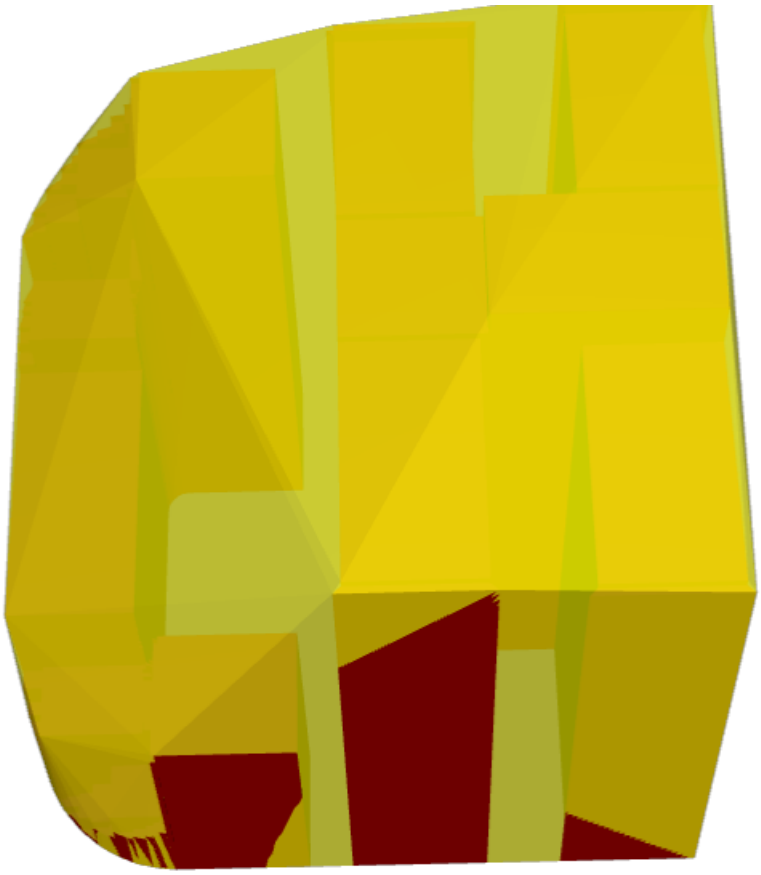
This function supports Triangles and Triangulated Irregular Network Surfaces (TIN).

Beispiele

```
SELECT ST_AsText(ST_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, ST_Extrude(geom, 0,0, i ) AS geom
FROM ST_Subdivide(ST_Letters('CH'),5) WITH ORDINALITY AS sd(geom,i)
      )
      SELECT ST_3DConvexHull(ST_Collect(f.geom) )
      FROM f;
```



Original geometry overlaid with 3D convex hull

Siehe auch

ST_Letters, ST_AsX3D

7.21.5 ST_3DIntersection

ST_3DIntersection — Perform 3D intersection

Synopsis

geometry **ST_SharedPaths**(geometry lineal1, geometry lineal2);

Beschreibung

Return a geometry that is the shared portion between geom1 and geom2.

Verfügbarkeit: 2.1.0

This method needs SFCGAL backend.

This method implements the SQL/MM specification. SQL-MM IEC 13249-3: 5.1

This function supports 3d and will not drop the z-index.

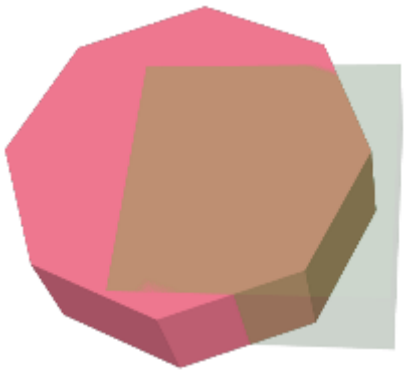
This function supports Polyhedral surfaces.

This function supports Triangles and Triangulated Irregular Network Surfaces (TIN).

Beispiele

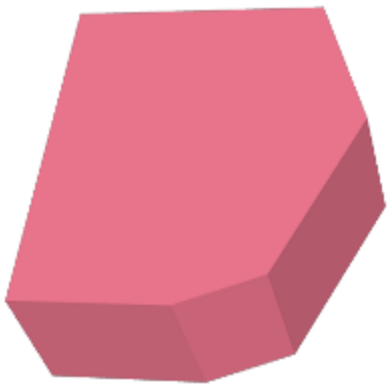
3D images were generated using PostGIS ST_AsX3D and rendering in HTML using X3Dom HTML Javascript rendering library.

```
SELECT ST_Extrude(ST_Buffer( ↩
          ST_GeomFromText('POINT(100 90)'),
 50, 'quad_segs=2'),0,0,30) AS geom1,
        ST_Extrude(ST_Buffer( ↩
          ST_GeomFromText('POINT(80 80)'),
 50, 'quad_segs=1'),0,0,30) AS geom2;
```



Original 3D geometries overlaid. geom2 is shown semi-transparent

```
SELECT ST_3DIntersection(geom1,geom2)
FROM ( SELECT ST_Extrude(ST_Buffer( ↩
          ST_GeomFromText('POINT(100 90)'),
 50, 'quad_segs=2'),0,0,30) AS geom1,
        ST_Extrude(ST_Buffer( ↩
          ST_GeomFromText('POINT(80 80)'),
 50, 'quad_segs=1'),0,0,30) AS geom2 ) As ↩
          t;
```



Intersection of geom1 and geom2

Ein MultiLinestring und ein LineString

```
SELECT ST_AsText(ST_3DIntersection(linestring, polygon)) As wkt
FROM  ST_GeomFromText('LINESTRING Z (2 2 6,1.5 1.5 7,1 1 8,0.5 0.5 8,0 0 10)') AS  ↩
    linestring
 CROSS JOIN ST_GeomFromText('POLYGON((0 0 8, 0 1 8, 1 1 8, 1 0 8, 0 0 8))') AS polygon;
```

```
              wkt
-------------------------------
 LINESTRING Z (1 1 8,0.5 0.5 8)
```

Cube (closed Polyhedral Surface) and Polygon Z

```
SELECT ST_AsText(ST_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)))
```

Intersection of 2 solids that result in volumetric intersection is also a solid (ST_Dimension returns 3)

```
SELECT ST_AsText(ST_3DIntersection( ST_Extrude(ST_Buffer('POINT(10 20)'::geometry,10,1) ↩
    ,0,0,30),
 ST_Extrude(ST_Buffer('POINT(10 20)'::geometry,10,1),2,0,10) ));
```

```
POLYHEDRALSURFACE Z (((13.3333333333333 13.3333333333333 10,20 20 0,20 20  ↩
    10,13.3333333333333 13.3333333333333 10)),
        ((20 20 10,16.6666666666667 23.3333333333333 10,13.3333333333333 13.3333333333333  ↩
            10,20 20 10)),
        ((20 20 0,16.6666666666667 23.3333333333333 10,20 20 10,20 20 0)),
        ((13.3333333333333 13.3333333333333 10,10 10 0,20 20 0,13.3333333333333  ↩
            13.3333333333333 10)),
        ((16.6666666666667 23.3333333333333 10,12 28 10,13.3333333333333 13.3333333333333  ↩
            10,16.6666666666667 23.3333333333333 10)),
        ((20 20 0,9.99999999999995 30 0,16.6666666666667 23.3333333333333 10,20 20 0)),
        ((10 10 0,9.99999999999995 30 0,20 20 0,10 10 0)),((13.3333333333333  ↩
            13.3333333333333 10,12 12 10,10 10 0,13.3333333333333 13.3333333333333 10)),
        ((12 28 10,12 12 10,13.3333333333333 13.3333333333333 10,12 28 10)),
        ((16.6666666666667 23.3333333333333 10,9.99999999999995 30 0,12 28  ↩
            10,16.6666666666667 23.3333333333333 10)),
        ((10 10 0,0 20 0,9.99999999999995 30 0,10 10 0)),
        ((12 12 10,11 11 10,10 10 0,12 12 10)),((12 28 10,11 11 10,12 12 10,12 28 10)),
        ((9.99999999999995 30 0,11 29 10,12 28 10,9.99999999999995 30 0)),((0 20 0,2 20  ↩
            10,9.99999999999995 30 0,0 20 0)),
        ((10 10 0,2 20 10,0 20 0,10 10 0)),((11 11 10,2 20 10,10 10 0,11 11 10)),((12 28  ↩
            10,11 29 10,11 11 10,12 28 10)),
        ((9.99999999999995 30 0,2 20 10,11 29 10,9.99999999999995 30 0)),((11 11 10,11 29  ↩
            10,2 20 10,11 11 10)))
```

7.21.6 ST_3DDifference

ST_3DDifference — Perform 3D difference

Synopsis

geometry **ST_SharedPaths**(geometry lineal1, geometry lineal2);

Beschreibung

Gibt den geometrischen Schwerpunkt einer Geometrie zurück.

Verfügbarkeit: 2.2.0

This method needs SFCGAL backend.

This method implements the SQL/MM specification. SQL-MM IEC 13249-3: 5.1

This function supports 3d and will not drop the z-index.

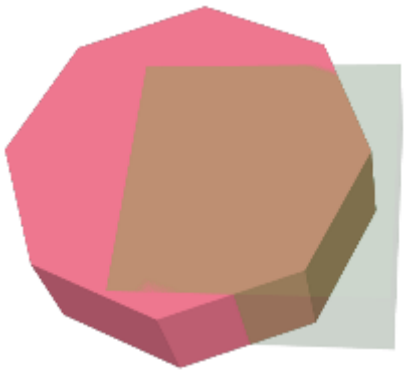
This function supports Polyhedral surfaces.

This function supports Triangles and Triangulated Irregular Network Surfaces (TIN).

Beispiele

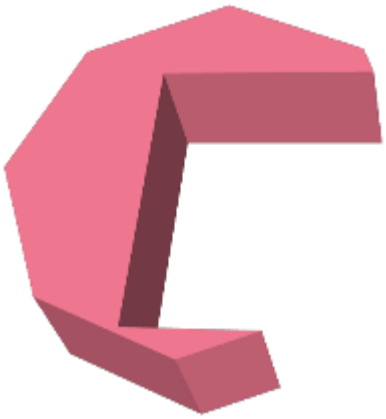
3D images were generated using PostGIS ST_AsX3D and rendering in HTML using X3Dom HTML Javascript rendering library.

```
SELECT ST_Extrude(ST_Buffer( ↩
            ST_GeomFromText('POINT(100 90)'),
 50, 'quad_segs=2'),0,0,30) AS geom1,
        ST_Extrude(ST_Buffer( ↩
            ST_GeomFromText('POINT(80 80)'),
 50, 'quad_segs=1'),0,0,30) AS geom2;
```



Original 3D geometries overlaid. geom2 is the part that will be removed.

```
SELECT ST_3DDifference(geom1,geom2)
FROM ( SELECT ST_Extrude(ST_Buffer( ↩
            ST_GeomFromText('POINT(100 90)'),
 50, 'quad_segs=2'),0,0,30) AS geom1,
        ST_Extrude(ST_Buffer( ↩
            ST_GeomFromText('POINT(80 80)'),
 50, 'quad_segs=1'),0,0,30) AS geom2 ) As ↩
             t;
```



What's left after removing geom2

Siehe auch

ST_Extrude, ST_ConcaveHull, ST_Dump, ST_Tesselate

7.21.7 ST_3DUnion

ST_3DUnion — Perform 3D union.

Synopsis

geometry **ST_3DUnion**(geometry geom1, geometry geom2);
geometry **ST_3DUnion**(geometry set g1field);

Beschreibung

Verfügbarkeit: 2.2.0

Availability: 3.3.0 aggregate variant was added

This method needs SFCGAL backend.

This method implements the SQL/MM specification. SQL-MM IEC 13249-3: 5.1

This function supports 3d and will not drop the z-index.

This function supports Polyhedral surfaces.

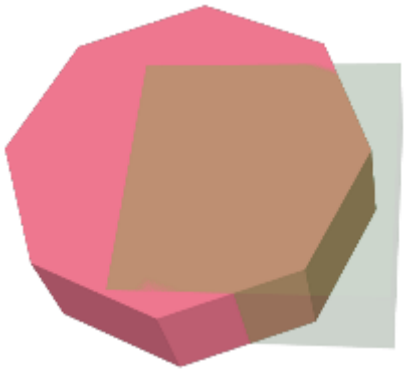
This function supports Triangles and Triangulated Irregular Network Surfaces (TIN).

Aggregate variant: returns a geometry that is the 3D union of a rowset of geometries. The ST_3DUnion() function is an "aggregate" function in the terminology of PostgreSQL. That means that it operates on rows of data, in the same way the SUM() and AVG() functions do and like most aggregates, it also ignores NULL geometries.

Beispiele

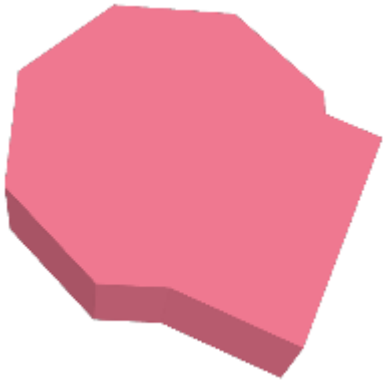
3D images were generated using PostGIS ST_AsX3D and rendering in HTML using X3Dom HTML Javascript rendering library.

```
SELECT ST_Extrude(ST_Buffer( ↩
            ST_GeomFromText('POINT(100 90)'),
 50, 'quad_segs=2'),0,0,30) AS geom1,
       ST_Extrude(ST_Buffer( ↩
            ST_GeomFromText('POINT(80 80)'),
 50, 'quad_segs=1'),0,0,30) AS geom2;
```



Original 3D geometries overlaid. geom2 is the one with transparency.

```
SELECT ST_3DUnion(geom1,geom2)
FROM ( SELECT ST_Extrude(ST_Buffer( ↩
            ST_GeomFromText('POINT(100 90)'),
 50, 'quad_segs=2'),0,0,30) AS geom1,
       ST_Extrude(ST_Buffer( ↩
            ST_GeomFromText('POINT(80 80)'),
 50, 'quad_segs=1'),0,0,30) AS geom2 ) As ↩
          t;
```



Union of geom1 and geom2

Siehe auch

ST_Extrude, ST_ConcaveHull, ST_Dump, ST_Tesselate

7.21.8 ST_AlphaShape

ST_AlphaShape — Computes an Alpha-shape enclosing a geometry

Synopsis

geometry **ST_AlphaShape**(geometry geom, float alpha, boolean allow_holes = false);

Beschreibung

Computes the Alpha-Shape of the points in a geometry. An alpha-shape is a (usually) concave polygonal geometry which contains all the vertices of the input, and whose vertices are a subset of the input vertices. An alpha-shape provides a closer fit to the shape of the input than the shape produced by the convex hull.

The "closeness of fit" is controlled by the `alpha` parameter, which can have values from 0 to infinity. Smaller alpha values produce more concave results. Alpha values greater than some data-dependent value produce the convex hull of the input.

Note
Following the CGAL implementation, the alpha value is the *square* of the radius of the disc used in the Alpha-Shape algorithm to "erode" the Delaunay Triangulation of the input points. See CGAL Alpha-Shapes for more information. This is different from the original definition of alpha-shapes, which defines alpha as the radius of the eroding disc.

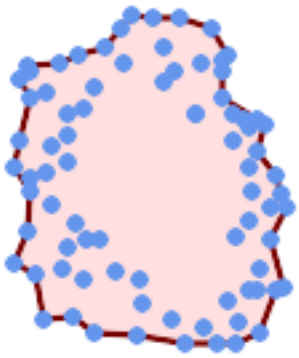
The computed shape does not contain holes unless the optional `allow_holes` argument is specified as true.

This function effectively computes a concave hull of a geometry in a similar way to ST_ConcaveHull, but uses CGAL and a different algorithm.

Availability: 3.3.0 - requires SFCGAL >= 1.4.1.

This method needs SFCGAL backend.

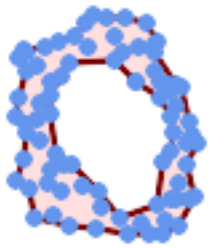
Beispiele



Alpha-shape of a MultiPoint (same example As ST_OptimalAlphaShape)

```
SELECT ST_AsText(ST_AlphaShape('MULTIPOINT((63 84),(76 88),(68 73),(53 18),(91 50),(81 70),
                (88 29),(24 82),(32 51),(37 23),(27 54),(84 19),(75 87),(44 42),(77 67),(90 ↩
                    30),(36 61),(32 65),
                (81 47),(88 58),(68 73),(49 95),(81 60),(87 50),
                (78 16),(79 21),(30 22),(78 43),(26 85),(48 34),(35 35),(36 40),(31 79),(83 ↩
                    29),(27 84),(52 98),(72 95),(85 71),
                (75 84),(75 77),(81 29),(77 73),(41 42),(83 72),(23 36),(89 53),(27 57),(57 ↩
                    97),(27 77),(39 88),(60 81),
                (80 72),(54 32),(55 26),(62 22),(70 20),(76 27),(84 35),(87 42),(82 54),(83 ↩
                    64),(69 86),(60 90),(50 86),(43 80),(36 73),
                (36 68),(40 75),(24 67),(23 60),(26 44),(28 33),(40 32),(43 19),(65 16),(73 ↩
                    16),(38 46),(31 59),(34 86),(45 90),(64 97))'::geometry,80.2));
```

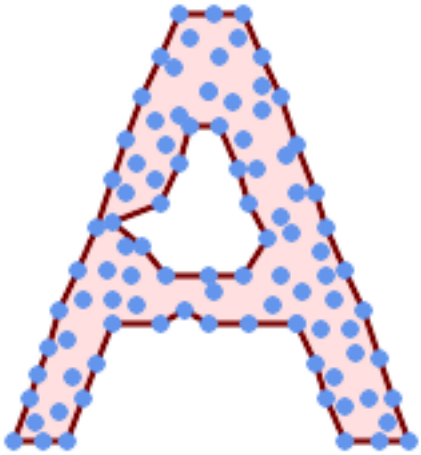
```
POLYGON((89 53,91 50,87 42,90 30,88 29,84 19,78 16,73 16,65 16,53 18,43 19,37 23,30 22,28  ↩
    33,23 36,26 44,27 54,23 60,24 67,
          27 77,24 82,26 85,34 86,39 88,45 90,49 95,52 98,57 97,64 97,72 95,76 88,75 84,83  ↩
              72,85 71,88 58,89 53))
```



Alpha-shape of a MultiPoint, allowing holes (same example as ST_OptimalAlphaShape)

```
SELECT ST_AsText(ST_AlphaShape('MULTIPOINT((63 84),(76 88),(68 73),(53 18),(91 50),(81 70) ↩
    ,(88 29),(24 82),(32 51),(37 23),(27 54),(84 19),(75 87),(44 42),(77 67),(90 30),(36 61) ↩
    ,(32 65),(81 47),(88 58),(68 73),(49 95),(81 60),(87 50),
                (78 16),(79 21),(30 22),(78 43),(26 85),(48 34),(35 35),(36 40),(31 79),(83 ↩
                    29),(27 84),(52 98),(72 95),(85 71),
                (75 84),(75 77),(81 29),(77 73),(41 42),(83 72),(23 36),(89 53),(27 57),(57 ↩
                    97),(27 77),(39 88),(60 81),
                (80 72),(54 32),(55 26),(62 22),(70 20),(76 27),(84 35),(87 42),(82 54),(83 ↩
                    64),(69 86),(60 90),(50 86),(43 80),(36 73),
                (36 68),(40 75),(24 67),(23 60),(26 44),(28 33),(40 32),(43 19),(65 16),(73 ↩
                    16),(38 46),(31 59),(34 86),(45 90),(64 97))'::geometry, 100.1,true))
```

```
POLYGON((89 53,91 50,87 42,90 30,88 29,84 19,78 16,73 16,65 16,53 18,43 19,37 23,30 22,28  ↩
    33,23 36,26 44,27 54,23 60,24 67,27 77,24 82,26 85,34 86,39 88,45 90,49 95,52 98,57  ↩
    97,64 97,72 95,76 88,75 84,83 72,85 71,88 58,89 53),
              (36 61,36 68,40 75,43 80,50 86,60 81,68 73,77 67,81 60,82 54,81 47,78 43,76  ↩
                 27,62 22,54 32,48 34,44 42,38 46,36 61))
```



Alpha-shape of a MultiPoint, allowing holes (same example as ST_ConcaveHull)

```
SELECT ST_AlphaShape(
            'MULTIPOINT ((132 64), (114 64), (99 64), (81 64), (63 64), (57 49), (52 36),  ↩
                (46 20), (37 20), (26 20), (32 36), (39 55), (43 69), (50 84), (57 100), (63 ↩
                 118), (68 133), (74 149), (81 164), (88 180), (101 180), (112 180), (119  ↩
                164), (126 149), (132 131), (139 113), (143 100), (150 84), (157 69), (163  ↩
                51), (168 36), (174 20), (163 20), (150 20), (143 36), (139 49), (132 64),  ↩
                (99 151), (92 138), (88 124), (81 109), (74 93), (70 82), (83 82), (99 82),  ↩
                (112 82), (126 82), (121 96), (114 109), (110 122), (103 138), (99 151), (34 ↩
                 27), (43 31), (48 44), (46 58), (52 73), (63 73), (61 84), (72 71), (90 69) ↩
                , (101 76), (123 71), (141 62), (166 27), (150 33), (159 36), (146 44), (154 ↩
                 53), (152 62), (146 73), (134 76), (143 82), (141 91), (130 98), (126 104), ↩
                 (132 113), (128 127), (117 122), (112 133), (119 144), (108 147), (119 153) ↩
                , (110 171), (103 164), (92 171), (86 160), (88 142), (79 140), (72 124),  ↩
                (83 131), (79 118), (68 113), (63 102), (68 93), (35 45))'::geometry,102.2,  ↩
                true);
```

```
POLYGON((134 80,136 75,130 63,135 45,132 44,126 28,117 24,110 24,98 24,80 27,82 39,72 51,60 ↩
     48,56 34,52 52,42 50,
                34 54,39 66,40 81,34 90,36 100,40 116,36 123,39 128,51 129,58 132,68 135,74 ↩
                     142,78 147,86 146,96 146,
                108 142,114 132,112 126,112 116,116 110,120 108,125 108,128 106,125 96,132  ↩
                    87,134 80))
```

Siehe auch

ST_ConcaveHull, ST_OptimalAlphaShape

7.21.9 ST_ApproximateMedialAxis

ST_ApproximateMedialAxis — Berechnet die konvexe Hülle einer Geometrie.

Synopsis

geometry **ST_OrientedEnvelope**(geometry geom);

Beschreibung

Return an approximate medial axis for the areal input based on its straight skeleton. Uses an SFCGAL specific API when built against a capable version (1.2.0+). Otherwise the function is just a wrapper around ST_StraightSkeleton (slower case).

Verfügbarkeit: 2.2.0

This method needs SFCGAL backend.

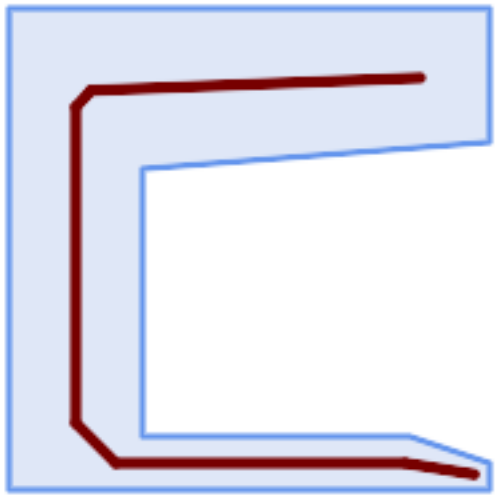
This function supports 3d and will not drop the z-index.

This function supports Polyhedral surfaces.

This function supports Triangles and Triangulated Irregular Network Surfaces (TIN).

Beispiele

```
SELECT ST_ApproximateMedialAxis(ST_GeomFromText('POLYGON (( 190 190, 10 190, 10 10, 190 10, ↩
    190 20, 160 30, 60 30, 60 130, 190 140, 190 190 ))'));
```



A polygon and its approximate medial axis

Siehe auch

ST_StraightSkeleton

7.21.10 ST_ConstrainedDelaunayTriangles

ST_ConstrainedDelaunayTriangles — Return a constrained Delaunay triangulation around the given input geometry.

Synopsis

geometry **ST_PointOnSurface**(geometry g1);

Beschreibung

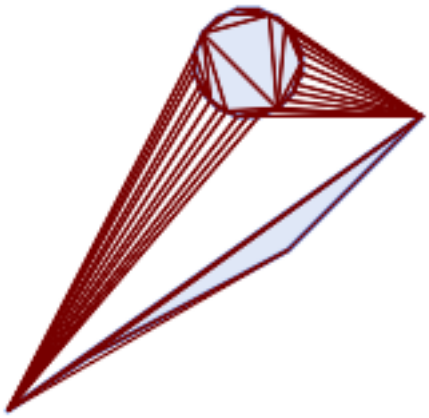
Return a Constrained Delaunay triangulation around the vertices of the input geometry. Output is a TIN.

This method needs SFCGAL backend.

Verfügbarkeit: 2.0.0

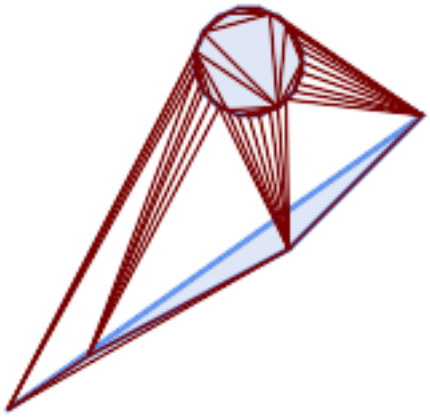
This function supports 3d and will not drop the z-index.

Beispiele



ST_DelaunayTriangles

```
select ST_ConstrainedDelaunayTriangles(
                ST_Union(
                        'POLYGON((175 150, ↩
     20 40, 50 60, 125 100, 175 150))'::geometry,
                        ST_Buffer('POINT ↩
    (110 170)'::geometry, 20)
                    )
            );
```



ST_DelaunayTriangles of 2 polygons. Triangle edges cross polygon boundaries.

```
select ST_DelaunayTriangles(
                ST_Union(
                        'POLYGON((175 150, ↩
     20 40, 50 60, 125 100, 175 150))'::geometry,
                        ST_Buffer('POINT ↩
    (110 170)'::geometry, 20)
                    )
            );
```

Siehe auch

ST_DelaunayTriangles, ST_TriangulatePolygon, ST_Tesselate, ST_ConcaveHull, ST_Dump

7.21.11 ST_Extrude

ST_Extrude — Extrude a surface to a related volume

Synopsis

geometry **ST_FilterByM**(geometry geom, double precision min, double precision max = null, boolean returnM = false);

Beschreibung

Verfügbarkeit: 2.1.0

This method needs SFCGAL backend.

This function supports 3d and will not drop the z-index.

This function supports Polyhedral surfaces.

This function supports Triangles and Triangulated Irregular Network Surfaces (TIN).

Beispiele

3D images were generated using PostGIS ST_AsX3D and rendering in HTML using X3Dom HTML Javascript rendering library.

```
SELECT ST_Buffer(ST_GeomFromText('POINT ↩
    (100 90)'),
  50, 'quad_segs=2'),0,0,30);
```

Original octagon formed from buffering point

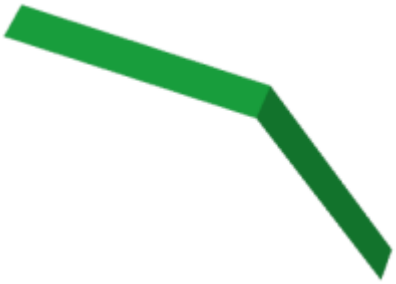
```
ST_Extrude(ST_Buffer(ST_GeomFromText(' ↩
    POINT(100 90)'),
 50, 'quad_segs=2'),0,0,30);
```

Hexagon extruded 30 units along Z produces a PolyhedralSurfaceZ

```
SELECT ST_GeomFromText('LINESTRING(50 50, ↩
    100 90, 95 150)')
```

Ursprüngliche Polygone

```
SELECT ST_Extrude(
 ST_GeomFromText('LINESTRING(50 50, 100  ↩
    90, 95 150)'),0,0,10));
```



LineString Extruded along Z produces a PolyhedralSurfaceZ

Siehe auch

ST_AsX3D

7.21.12 ST_ForceLHR

ST_ForceLHR — Force LHR orientation

Synopsis

geometry **ST_ConvexHull**(geometry geomA);

Beschreibung

Verfügbarkeit: 2.1.0

This method needs SFCGAL backend.

This function supports 3d and will not drop the z-index.

This function supports Polyhedral surfaces.

This function supports Triangles and Triangulated Irregular Network Surfaces (TIN).

7.21.13 ST_IsPlanar

ST_IsPlanar — Check if a surface is or not planar

Synopsis

geometry **ST_ConvexHull**(geometry geomA);

Beschreibung

Availability: 2.2.0: This was documented in 2.1.0 but got accidentally left out in 2.1 release.

This method needs SFCGAL backend.

This function supports 3d and will not drop the z-index.

This function supports Polyhedral surfaces.

This function supports Triangles and Triangulated Irregular Network Surfaces (TIN).

7.21.14 ST_IsSolid

ST_IsSolid — Test if the geometry is a solid. No validity check is performed.

Synopsis

boolean **ST_IsSolid**(geometry geom1);

Beschreibung

Verfügbarkeit: 2.2.0

This method needs SFCGAL backend.

This function supports 3d and will not drop the z-index.

This function supports Polyhedral surfaces.

This function supports Triangles and Triangulated Irregular Network Surfaces (TIN).

7.21.15 ST_MakeSolid

ST_MakeSolid — Cast the geometry into a solid. No check is performed. To obtain a valid solid, the input geometry must be a closed Polyhedral Surface or a closed TIN.

Synopsis

geometry **ST_MakeSolid**(geometry geom1);

Beschreibung

Verfügbarkeit: 2.2.0

This method needs SFCGAL backend.

This function supports 3d and will not drop the z-index.

This function supports Polyhedral surfaces.

This function supports Triangles and Triangulated Irregular Network Surfaces (TIN).

7.21.16 ST_MinkowskiSum

ST_MinkowskiSum — Performs Minkowski sum

Synopsis

geometry **ST_SharedPaths**(geometry lineal1, geometry lineal2);

Beschreibung

This function performs a 2D minkowski sum of a point, line or polygon with a polygon.

A minkowski sum of two geometries A and B is the set of all points that are the sum of any point in A and B. Minkowski sums are often used in motion planning and computer-aided design. More details on Wikipedia Minkowski addition.

The first parameter can be any 2D geometry (point, linestring, polygon). If a 3D geometry is passed, it will be converted to 2D by forcing Z to 0, leading to possible cases of invalidity. The second parameter must be a 2D polygon.

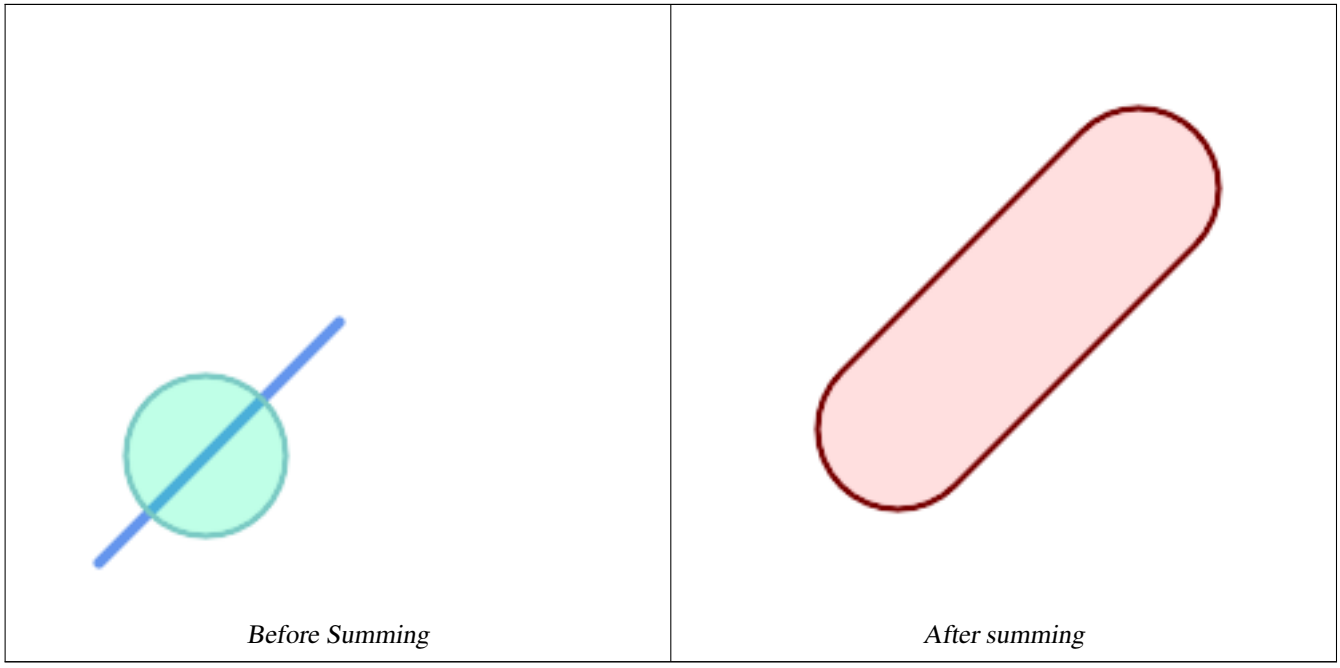
Implementation utilizes CGAL 2D Minkowskisum.

Verfügbarkeit: 2.1.0

This method needs SFCGAL backend.

Beispiele

Minkowski Sum of Linestring and circle polygon where Linestring cuts thru the circle

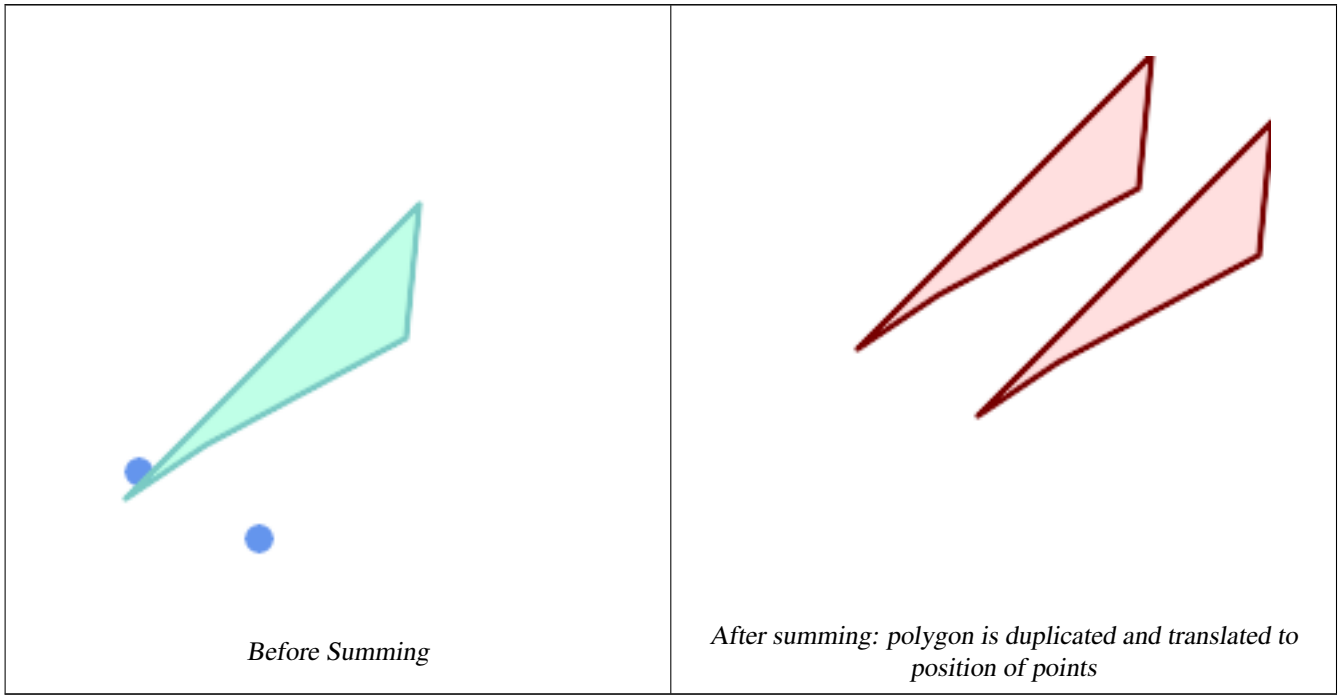


Before Summing | *After summing*

```
SELECT ST_MinkowskiSum(line, circle))
FROM (SELECT
    ST_MakeLine(ST_Point(10, 10),ST_Point(100, 100)) As line,
    ST_Buffer(ST_GeomFromText('POINT(50 50)'), 30) As circle) As foo;

-- wkt --
MULTIPOLYGON(((30 59.9999999999999,30.5764415879031 54.1472903395161,32.2836140246614  ↩
```

```
    48.5194970290472,35.0559116309237 43.3328930094119,38.7867965644036  ↩
    38.7867965644035,43.332893009412 35.0559116309236,48.5194970290474  ↩
    32.2836140246614,54.1472903395162 30.5764415879031,60.0000000000001 30,65.8527096604839  ↩
    30.5764415879031,71.4805029709527 32.2836140246614,76.6671069905881  ↩
    35.0559116309237,81.2132034355964 38.7867965644036,171.213203435596  ↩
    128.786796564404,174.944088369076 133.332893009412,177.716385975339  ↩
    138.519497029047,179.423558412097 144.147290339516,180 150,179.423558412097  ↩
    155.852709660484,177.716385975339 161.480502970953,174.944088369076  ↩
    166.667106990588,171.213203435596 171.213203435596,166.667106990588 174.944088369076,
161.480502970953 177.716385975339,155.852709660484 179.423558412097,150  ↩
    180,144.147290339516 179.423558412097,138.519497029047 177.716385975339,133.332893009412 ↩
     174.944088369076,128.786796564403 171.213203435596,38.7867965644035  ↩
    81.2132034355963,35.0559116309236 76.667106990588,32.2836140246614  ↩
    71.4805029709526,30.5764415879031 65.8527096604838,30 59.9999999999999)))
```

Minkowski Sum of a polygon and multipoint



Before Summing

After summing: polygon is duplicated and translated to position of points

```
SELECT ST_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))
    )
```

7.21.17 ST_OptimalAlphaShape

ST_OptimalAlphaShape — Computes an Alpha-shape enclosing a geometry using an "optimal" alpha value.

Synopsis

geometry **ST_OptimalAlphaShape**(geometry geom, boolean allow_holes = false, integer nb_components = 1);

Beschreibung

Computes the "optimal" alpha-shape of the points in a geometry. The alpha-shape is computed using a value of α chosen so that:

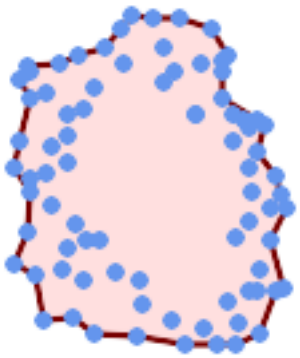
1. the number of polygon elements is equal to or smaller than `nb_components` (which defaults to 1)
2. all input points are contained in the shape

The result will not contain holes unless the optional `allow_holes` argument is specified as true.

Availability: 3.3.0 - requires SFCGAL >= 1.4.1.

This method needs SFCGAL backend.

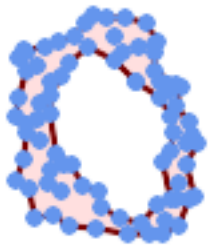
Beispiele



Optimal alpha-shape of a MultiPoint (same example as ST_AlphaShape)

```
SELECT ST_AsText(ST_OptimalAlphaShape('MULTIPOINT((63 84),(76 88),(68 73),(53 18),(91 50) ↩
    ,(81 70),
                (88 29),(24 82),(32 51),(37 23),(27 54),(84 19),(75 87),(44 42),(77 67),(90 ↩
                    30),(36 61),(32 65),
                (81 47),(88 58),(68 73),(49 95),(81 60),(87 50),
                (78 16),(79 21),(30 22),(78 43),(26 85),(48 34),(35 35),(36 40),(31 79),(83 ↩
                    29),(27 84),(52 98),(72 95),(85 71),
                (75 84),(75 77),(81 29),(77 73),(41 42),(83 72),(23 36),(89 53),(27 57),(57 ↩
                    97),(27 77),(39 88),(60 81),
                (80 72),(54 32),(55 26),(62 22),(70 20),(76 27),(84 35),(87 42),(82 54),(83 ↩
                    64),(69 86),(60 90),(50 86),(43 80),(36 73),
                (36 68),(40 75),(24 67),(23 60),(26 44),(28 33),(40 32),(43 19),(65 16),(73 ↩
                    16),(38 46),(31 59),(34 86),(45 90),(64 97))'::geometry));
```

```
POLYGON((89 53,91 50,87 42,90 30,88 29,84 19,78 16,73 16,65 16,53 18,43 19,37 23,30 22,28  ↩
    33,23 36,
                26 44,27 54,23 60,24 67,27 77,24 82,26 85,34 86,39 88,45 90,49 95,52 98,57  ↩
                    97,64 97,72 95,76 88,75 84,75 77,83 72,85 71,83 64,88 58,89 53))
```



Optimal alpha-shape of a MultiPoint, allowing holes (same example as ST_AlphaShape)

```
SELECT ST_AsText(ST_OptimalAlphaShape('MULTIPOINT((63 84),(76 88),(68 73),(53 18),(91 50) ↩
    ,(81 70),(88 29),(24 82),(32 51),(37 23),(27 54),(84 19),(75 87),(44 42),(77 67),(90 30) ↩
    ,(36 61),(32 65),(81 47),(88 58),(68 73),(49 95),(81 60),(87 50),
                (78 16),(79 21),(30 22),(78 43),(26 85),(48 34),(35 35),(36 40),(31 79),(83 ↩
                    29),(27 84),(52 98),(72 95),(85 71),
                (75 84),(75 77),(81 29),(77 73),(41 42),(83 72),(23 36),(89 53),(27 57),(57 ↩
                    97),(27 77),(39 88),(60 81),
                (80 72),(54 32),(55 26),(62 22),(70 20),(76 27),(84 35),(87 42),(82 54),(83 ↩
                    64),(69 86),(60 90),(50 86),(43 80),(36 73),
                (36 68),(40 75),(24 67),(23 60),(26 44),(28 33),(40 32),(43 19),(65 16),(73 ↩
                    16),(38 46),(31 59),(34 86),(45 90),(64 97))'::geometry, allow_holes =
> true));
```

```
POLYGON((89 53,91 50,87 42,90 30,88 29,84 19,78 16,73 16,65 16,53 18,43 19,37 23,30 22,28  ↩
    33,23 36,26 44,27 54,23 60,24 67,27 77,24 82,26 85,34 86,39 88,45 90,49 95,52 98,57  ↩
    97,64 97,72 95,76 88,75 84,75 77,83 72,85 71,83 64,88 58,89 53),(36 61,36 68,40 75,43  ↩
    80,50 86,60 81,68 73,77 67,81 60,82 54,81 47,78 43,81 29,76 27,70 20,62 22,55 26,54  ↩
    32,48 34,44 42,38 46,36 61))
```

Siehe auch

ST_ConcaveHull, ST_AlphaShape

7.21.18 ST_Orientation

ST_Orientation — Determine surface orientation

Synopsis

geometry **ST_OrientedEnvelope**(geometry geom);

Beschreibung

The function only applies to polygons. It returns -1 if the polygon is counterclockwise oriented and 1 if the polygon is clockwise oriented.

Verfügbarkeit: 2.1.0

This method needs SFCGAL backend.

This function supports 3d and will not drop the z-index.

7.21.19 ST_StraightSkeleton

ST_StraightSkeleton — Berechnet die konvexe Hülle einer Geometrie.

Synopsis

geometry **ST_OrientedEnvelope**(geometry geom);

Beschreibung

Verfügbarkeit: 2.1.0

This method needs SFCGAL backend.

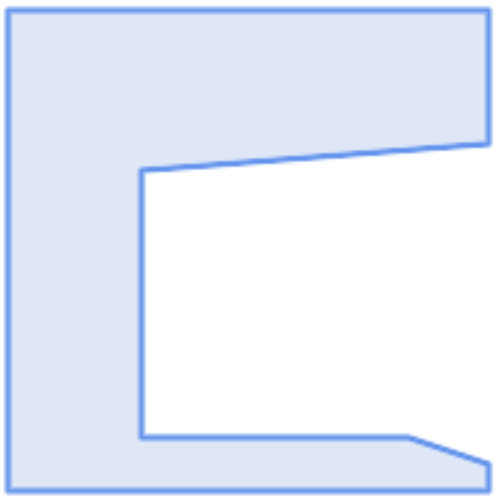
This function supports 3d and will not drop the z-index.

This function supports Polyhedral surfaces.

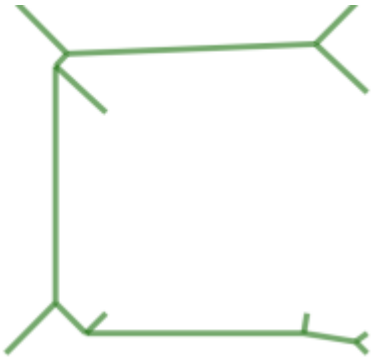
This function supports Triangles and Triangulated Irregular Network Surfaces (TIN).

Beispiele

```
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 ))'));
```



Ursprüngliche Polygone



Straight Skeleton of polygon

7.21.20 ST_Tesselate

ST_Tesselate — Perform surface Tesselation of a polygon or polyhedralsurface and returns as a TIN or collection of TINS

Synopsis

geometry **ST_ConvexHull**(geometry geomA);

Beschreibung

Takes as input a surface such a MULTI(POLYGON) or POLYHEDRALSURFACE and returns a TIN representation via the process of tessellation using triangles.

Verfügbarkeit: 2.1.0

This method needs SFCGAL backend.

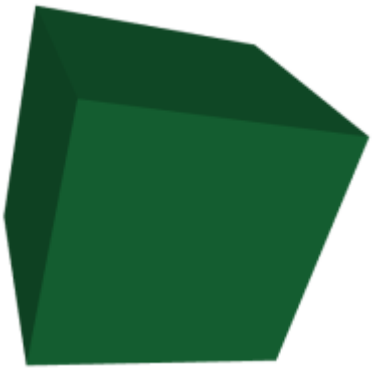
This function supports 3d and will not drop the z-index.

This function supports Polyhedral surfaces.

This function supports Triangles and Triangulated Irregular Network Surfaces (TIN).

Beispiele

```
SELECT ST_GeomFromText('POLYHEDRALSURFACE ↩
     Z( ((0 0 0, 0 0 1, 0 1 1, 0 1 0, 0 0 0)),
                ((0 0 0, 0 1 0, 1 1 0, 1  ↩
    0 0, 0 0 0)), ((0 0 0, 1 0 0, 1 0 1, 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)) )');
```



Ursprüngliches Polygon

```
SELECT ST_Tesselate(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 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_AsText output:

```
TIN Z (((0 0 0,0 0 1,0 1 1,0 0 0)),((0 1  ↩
    0,0 0 0,0 1 1,0 1 0)),
        ((0 0 0,0 1 0,1 1 0,0 0 0)),
        ((1 0 0,0 0 0,1 1 0,1 0 0)),((0 0 ↩
     1,1 0 0,1 0 1,0 0 1)),
        ((0 0 1,0 0 0,1 0 0,0 0 1)),
        ((1 1 0,1 1 1,1 0 1,1 1 0)),((1 0 ↩
     0,1 1 0,1 0 1,1 0 0)),
        ((0 1 0,0 1 1,1 1 1,0 1 0)),((1 1 ↩
     0,0 1 0,1 1 1,1 1 0)),
        ((0 1 1,1 0 1,1 1 1,0 1 1)),((0 1 ↩
     1,0 0 1,1 0 1,0 1 1)))
```

Tesselated Cube with triangles colored

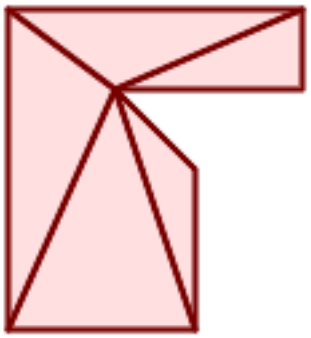
```
SELECT 'POLYGON (( 10 190, 10 70, 80 70,  ↩
    80 130, 50 160, 120 160, 120 190, 10 190 ))'::geometry;
```

Ursprüngliche Polygone

```
SELECT
        ST_Tesselate('POLYGON (( 10 190,  ↩
    10 70, 80 70, 80 130, 50 160, 120 160, 120 190,
```

ST_AsText output

```
TIN(((80 130,50 160,80 70,80 130)),((50  ↩
    160,10 190,10 70,50 160)),
         ((80 70,50 160,10 70,80 70)) ↩
    ,((120 160,120 190,50 160,120 160)),
  ((120 190,10 190,50 160,120 190)))
```



Tesselated Polygon

Siehe auch

ST_ConstrainedDelaunayTriangles, ST_LineSubstring

7.21.21 ST_Volume

ST_Volume — Computes the volume of a 3D solid. If applied to surface (even closed) geometries will return 0.

Synopsis

geometry **ST_ConvexHull**(geometry geomA);

Beschreibung

Verfügbarkeit: 2.2.0

This method needs SFCGAL backend.

This function supports 3d and will not drop the z-index.

This function supports Polyhedral surfaces.

This function supports Triangles and Triangulated Irregular Network Surfaces (TIN).

This method implements the SQL/MM specification. SQL-MM IEC 13249-3: 9.1 (same as ST_3DVolume)

Beispiele

When closed surfaces are created with WKT, they are treated as areal rather than solid. To make them solid, you need to use ST_MakeSolid. Areal geometries have no volume. Here is an example to demonstrate.

```
SELECT ST_Volume(geom) As cube_surface_vol,
        ST_Volume(ST_MakeSolid(geom)) As solid_surface_vol
  FROM (SELECT 'POLYHEDRALSURFACE( ((0 0 0, 0 0 1, 0 1 1, 0 1 0, 0 0 0)),
    ((0 0 0, 0 1 0, 1 1 0, 1 0 0, 0 0 0)),
    ((0 0 0, 1 0 0, 1 0 1, 0 0 1, 0 0 0)),
    ((1 1 0, 1 1 1, 1 0 1, 1 0 0, 1 1 0)),
    ((0 1 0, 0 1 1, 1 1 1, 1 1 0, 0 1 0)),
    ((0 0 1, 1 0 1, 1 1 1, 0 1 1, 0 0 1)) )'::geometry) As f(geom);

 cube_surface_vol | solid_surface_vol
------------------+-------------------
                0 |                 1
```

Siehe auch

ST_3DArea, ST_VoronoiLines, ST_Collect

7.22 Unterstützung von lang andauernden Transaktionen/Long Transactions

Note
Es muss serializable transaction level angewandt werden, da sonst der "Locking" Mechanismus versagt.

7.22.1 AddAuth

AddAuth — Fügt einen Berechtigungsschlüssel für die aktuelle Transaktion hinzu.

Synopsis

boolean **AddAuth**(text auth_token);

Beschreibung

Fügt einen Berechtigungsschlüssel für die aktuelle Transaktion hinzu.

Adds the current transaction identifier and authorization token to a temporary table called `temp_lock_have_table`.

Verfügbarkeit: 1.1.3

Beispiele

```
SELECT LockRow('towns', '353', 'priscilla');
                BEGIN TRANSACTION;
                        SELECT AddAuth('joey');
                        UPDATE towns SET geom = ST_Translate(geom,2,2) WHERE gid = 353;
                COMMIT;


                ---Error--
                ERROR:  UPDATE where "gid" = '353' requires authorization 'priscilla'
```

Siehe auch

LockRow

7.22.2 CheckAuth

CheckAuth — Legt einen Trigger auf eine Tabelle an um, basierend auf einen Token, Updates und Deletes von Zeilen zu verhindern oder zu erlauben.

Synopsis

integer **CheckAuth**(text a_schema_name, text a_table_name, text a_key_column_name);
integer **CheckAuth**(text a_table_name, text a_key_column_name);

Beschreibung

Legt einen Trigger auf eine Tabelle an, um über ein Autorisierungs-Token, Updates und Deletes von Zeilen zu verhindern oder zu erlauben. Identifiziert Zeilen über die Spalte <rowid_col>.

Wenn "a_schema_name" nicht angegeben ist, wird im aktuellen Schema nach der Tabelle gesucht.

Note
Wenn ein Autorisierungstrigger auf dieser Tabelle bereits existiert, gibt die Funktion eine Fehlermeldung aus.
Wenn die Transaktionsunterstützung nicht aktiviert wurde, wird ein Fehler gemeldet.

Verfügbarkeit: 1.1.3

Beispiele

```
SELECT CheckAuth('public', 'towns', 'gid');
                        result
                        ------
                        0
```

Siehe auch

EnableLongTransactions

7.22.3 DisableLongTransactions

DisableLongTransactions — DisableLongTransactions

Synopsis

text **DisableLongTransactions**();

Beschreibung

Deaktiver die Unterstützung von lang andauernden Transaktionen. Diese Funktion entfernt die Metadatentabellen der Unterstützung für lang andauernde Transaktionen und löscht alle Trigger der auf Locks überprüften Tabellen.

Löscht die Metadatentabelle mit der Bezeichnung `authorization_table` und den View mit der Bezeichnung `authorized_tabl` sowie alle Trigger mit der Bezeichnung `checkauthtrigger`

Verfügbarkeit: 1.1.3

Beispiele

```
SELECT DisableLongTransactions();
--result--
Long transactions support disabled
```

Siehe auch

EnableLongTransactions

7.22.4 EnableLongTransactions

EnableLongTransactions — EnableLongTransactions

Synopsis

text **EnableLongTransactions**();

Beschreibung

Akktivert die Unterstützung für lang andauernde Transaktionen. Diese Funktion erzeut die benötigten Metadatentabellen und muss einmal aufgerufen werden bevor die anderen Funktionen dieses Abschnitts angewandt werden. Ein zweimaliger Aufruf ist harmlos.

Erzeugt eine Metatabelle mit der Bezeichnung `authorization_table` und den View `authorized_tables`

Verfügbarkeit: 1.1.3

Beispiele

```
SELECT EnableLongTransactions();
--result--
Long transactions support enabled
```

Siehe auch

DisableLongTransactions

7.22.5 LockRow

LockRow — Setzt einen Lock/Autorisierung auf eine bestimmte Zeile in der Tabelle

Synopsis

integer **LockRow**(text a_schema_name, text a_table_name, text a_row_key, text an_auth_token, timestamp expire_dt);
integer **LockRow**(text a_table_name, text a_row_key, text an_auth_token, timestamp expire_dt);
integer **LockRow**(text a_table_name, text a_row_key, text an_auth_token);

Beschreibung

Setzt einen Lock/eine Autorisierung für eine bestimmte Zeile in der Tabelle <authid> . <authid> ist ein Text, <expires> ist ein Zeitstempel mit dem Standardwert now()+1hour. Gibt 1 aus, wenn ein Lock zugewiesen wurde, ansonsten 0 (bereits über eine andere Authentifizierung gesperrt)

Verfügbarkeit: 1.1.3

Beispiele

```
SELECT LockRow('public', 'towns', '2', 'joey');
LockRow
-------
1

--Joey has already locked the record and Priscilla is out of luck
SELECT LockRow('public', 'towns', '2', 'priscilla');
LockRow
-------
0
```

Siehe auch

UnlockRows

7.22.6 UnlockRows

UnlockRows — Removes all locks held by an authorization token.

Synopsis

integer **UnlockRows**(text auth_token);

Beschreibung

Entfernt alle Locks einer bestimmten Authorisierungs-ID. Gibt die Anzahl der freigegebenen Locks aus.

Verfügbarkeit: 1.1.3

Beispiele

```
SELECT LockRow('towns', '353', 'priscilla');
                SELECT LockRow('towns', '2', 'priscilla');
                SELECT UnLockRows('priscilla');
                UnLockRows
                ------------
                2
```

Siehe auch

LockRow

7.23 Version Functions

7.23.1 PostGIS_Extensions_Upgrade

PostGIS_Extensions_Upgrade — Packages and upgrades PostGIS extensions (e.g. postgis_raster, postgis_topology, postgis_sfcgal) to given or latest version.

Synopsis

text **PostGIS_Extensions_Upgrade**(text target_version=null);

Beschreibung

Packages and upgrades PostGIS extensions to given or latest version. Only extensions you have installed in the database will be packaged and upgraded if needed. Reports full PostGIS version and build configuration infos after. This is short-hand for doing multiple CREATE EXTENSION .. FROM unpackaged and ALTER EXTENSION .. UPDATE for each PostGIS extension. Currently only tries to upgrade extensions postgis, postgis_raster, postgis_sfcgal, postgis_topology, and postgis_tiger_geocoder.

Availability: 2.5.0

Note
Changed: 3.4.0 to add target_version argument.
Changed: 3.3.0 support for upgrades from any PostGIS version. Does not work on all systems.
Changed: 3.0.0 to repackage loose extensions and support postgis_raster.

Examples

```
SELECT PostGIS_Extensions_Upgrade();
```

```
NOTICE:  Packaging extension postgis
NOTICE:  Packaging extension postgis_raster
NOTICE:  Packaging extension postgis_sfcgal
NOTICE:  Extension postgis_topology is not available or not packagable for some reason
NOTICE:  Extension postgis_tiger_geocoder is not available or not packagable for some  ↩
    reason

                    postgis_extensions_upgrade
------------------------------------------------------------------
 Upgrade completed, run SELECT postgis_full_version(); for details
(1 row)
```

Siehe auch

Section 3.4, PostGIS_GEOS_Version, PostGIS_Lib_Version, PostGIS_LibXML_Version, PostGIS_PROJ_Version, PostGIS_Version

7.23.2 PostGIS_Full_Version

PostGIS_Full_Version — Reports full PostGIS version and build configuration infos.

Synopsis

text **PostGIS_Full_Version**();

Beschreibung

Reports full PostGIS version and build configuration infos. Also informs about synchronization between libraries and scripts suggesting upgrades as needed.

Enhanced: 3.4.0 now includes extra PROJ configurations NETWORK_ENABLED, URL_ENDPOINT and DATABASE_PATH of proj.db location

Examples

```
SELECT PostGIS_Full_Version();
                                                                    postgis_full_version
-------------------------------------------------------------------------------------------
POSTGIS="3.4.0dev 3.3.0rc2-993-g61bdf43a7" [EXTENSION] PGSQL="160" GEOS="3.12.0dev-CAPI ↩
    -1.18.0" SFCGAL="1.3.8" PROJ="7.2.1 NETWORK_ENABLED=OFF URL_ENDPOINT=https://cdn.proj. ↩
    org USER_WRITABLE_DIRECTORY=/tmp/proj DATABASE_PATH=/usr/share/proj/proj.db" GDAL="GDAL  ↩
    3.2.2, released 2021/03/05" LIBXML="2.9.10" LIBJSON="0.15" LIBPROTOBUF="1.3.3" WAGYU ↩
    ="0.5.0 (Internal)" TOPOLOGY RASTER
(1 row)
```

Siehe auch

Section 3.4, PostGIS_GEOS_Version, PostGIS_Lib_Version, PostGIS_LibXML_Version, PostGIS_PROJ_Version, PostGIS_Wagyu_Ve
PostGIS_Version

7.23.3 PostGIS_GEOS_Version

PostGIS_GEOS_Version — Returns the version number of the GEOS library.

Synopsis

text **PostGIS_GEOS_Version**();

Beschreibung

Returns the version number of the GEOS library, or `NULL` if GEOS support is not enabled.

Examples

```
SELECT PostGIS_GEOS_Version();
 postgis_geos_version
----------------------
3.12.0dev-CAPI-1.18.0
(1 row)
```

Siehe auch

PostGIS_Full_Version, PostGIS_Lib_Version, PostGIS_LibXML_Version, PostGIS_PROJ_Version, PostGIS_Version

7.23.4 PostGIS_GEOS_Compiled_Version

PostGIS_GEOS_Compiled_Version — Returns the version number of the GEOS library against which PostGIS was built.

Synopsis

text **PostGIS_GEOS_Compiled_Version**();

Beschreibung

Returns the version number of the GEOS library, or against which PostGIS was built.

Availability: 3.4.0

Examples

```
SELECT PostGIS_GEOS_Compiled_Version();
 postgis_geos_compiled_version
-------------------------------
 3.12.0
(1 row)
```

Siehe auch

PostGIS_GEOS_Version, PostGIS_Full_Version

7.23.5 PostGIS_Liblwgeom_Version

PostGIS_Liblwgeom_Version — Returns the version number of the liblwgeom library. This should match the version of PostGIS.

Synopsis

text **PostGIS_Liblwgeom_Version**();

Beschreibung

Returns the version number of the liblwgeom library/

Examples

```
SELECT PostGIS_Liblwgeom_Version();
postgis_liblwgeom_version
--------------------------
3.4.0dev 3.3.0rc2-993-g61bdf43a7
(1 row)
```

Siehe auch

PostGIS_Full_Version, PostGIS_Lib_Version, PostGIS_LibXML_Version, PostGIS_PROJ_Version, PostGIS_Version

7.23.6 PostGIS_LibXML_Version

PostGIS_LibXML_Version — Returns the version number of the libxml2 library.

Synopsis

text **PostGIS_LibXML_Version**();

Beschreibung

Returns the version number of the LibXML2 library.

Availability: 1.5

Examples

```
SELECT PostGIS_LibXML_Version();
 postgis_libxml_version
----------------------
 2.9.10
(1 row)
```

Siehe auch

PostGIS_Full_Version, PostGIS_Lib_Version, PostGIS_PROJ_Version, PostGIS_GEOS_Version, PostGIS_Version

7.23.7 PostGIS_Lib_Build_Date

PostGIS_Lib_Build_Date — Returns build date of the PostGIS library.

Synopsis

text **PostGIS_Lib_Build_Date**();

Beschreibung

Returns build date of the PostGIS library.

Examples

```
SELECT PostGIS_Lib_Build_Date();
 postgis_lib_build_date
------------------------
 2023-06-22 03:56:11
(1 row)
```

7.23.8 PostGIS_Lib_Version

PostGIS_Lib_Version — Returns the version number of the PostGIS library.

Synopsis

text **PostGIS_Lib_Version**();

Beschreibung

Returns the version number of the PostGIS library.

Examples

```
SELECT PostGIS_Lib_Version();
 postgis_lib_version
---------------------
 3.4.0dev
(1 row)
```

Siehe auch

PostGIS_Full_Version, PostGIS_GEOS_Version, PostGIS_LibXML_Version, PostGIS_PROJ_Version, PostGIS_Version

7.23.9 PostGIS_PROJ_Version

PostGIS_PROJ_Version — Returns the version number of the PROJ4 library.

Synopsis

text **PostGIS_PROJ_Version**();

Beschreibung

Returns the version number of the PROJ library and some configuration options of proj.

Enhanced: 3.4.0 now includes NETWORK_ENABLED, URL_ENDPOINT and DATABASE_PATH of proj.db location

Examples

```
SELECT PostGIS_PROJ_Version();
  postgis_proj_version
-------------------------
7.2.1 NETWORK_ENABLED=OFF URL_ENDPOINT=https://cdn.proj.org USER_WRITABLE_DIRECTORY=/tmp/ ↩
    proj DATABASE_PATH=/usr/share/proj/proj.db
(1 row)
```

Siehe auch

PostGIS_Full_Version, PostGIS_GEOS_Version, PostGIS_Lib_Version, PostGIS_LibXML_Version, PostGIS_Version

7.23.10 PostGIS_Wagyu_Version

PostGIS_Wagyu_Version — Returns the version number of the internal Wagyu library.

Synopsis

text **PostGIS_Wagyu_Version**();

Beschreibung

Returns the version number of the internal Wagyu library, or `NULL` if Wagyu support is not enabled.

Examples

```
SELECT PostGIS_Wagyu_Version();
 postgis_wagyu_version
-----------------------
 0.5.0 (Internal)
(1 row)
```

Siehe auch

PostGIS_Full_Version, PostGIS_GEOS_Version, PostGIS_PROJ_Version, PostGIS_Lib_Version, PostGIS_LibXML_Version, PostGIS_Version

7.23.11 PostGIS_Scripts_Build_Date

PostGIS_Scripts_Build_Date — Returns build date of the PostGIS scripts.

Synopsis

text **PostGIS_Scripts_Build_Date**();

Beschreibung

Returns build date of the PostGIS scripts.

Availability: 1.0.0RC1

Examples

```
SELECT PostGIS_Scripts_Build_Date();
  postgis_scripts_build_date
--------------------------
 2023-06-22 03:56:11
(1 row)
```

Siehe auch

PostGIS_Full_Version, PostGIS_GEOS_Version, PostGIS_Lib_Version, PostGIS_LibXML_Version, PostGIS_Version

7.23.12 PostGIS_Scripts_Installed

PostGIS_Scripts_Installed — Returns version of the PostGIS scripts installed in this database.

Synopsis

text **PostGIS_Scripts_Installed**();

Beschreibung

Returns version of the PostGIS scripts installed in this database.

Note
If the output of this function doesn't match the output of PostGIS_Scripts_Released you probably missed to properly upgrade an existing database. See the Upgrading section for more info.

Availability: 0.9.0

Examples

```
SELECT PostGIS_Scripts_Installed();
  postgis_scripts_installed
--------------------------
 3.4.0dev 3.3.0rc2-993-g61bdf43a7
(1 row)
```

Siehe auch

PostGIS_Full_Version, PostGIS_Scripts_Released, PostGIS_Version

7.23.13 PostGIS_Scripts_Released

PostGIS_Scripts_Released — Returns the version number of the postgis.sql script released with the installed PostGIS lib.

Synopsis

text **PostGIS_Scripts_Released**();

Beschreibung

Returns the version number of the postgis.sql script released with the installed PostGIS lib.

Note
Starting with version 1.1.0 this function returns the same value of PostGIS_Lib_Version. Kept for backward compatibility.

Availability: 0.9.0

Examples

```
SELECT PostGIS_Scripts_Released();
  postgis_scripts_released
--------------------------
 3.4.0dev 3.3.0rc2-993-g61bdf43a7
(1 row)
```

Siehe auch

PostGIS_Full_Version, PostGIS_Scripts_Installed, PostGIS_Lib_Version

7.23.14 PostGIS_Version

PostGIS_Version — Returns PostGIS version number and compile-time options.

Synopsis

text **PostGIS_Version**();

Beschreibung

Returns PostGIS version number and compile-time options.

Examples

```
SELECT PostGIS_Version();
                        postgis_version
---------------------------------------
 3.4 USE_GEOS=1 USE_PROJ=1 USE_STATS=1
(1 row)
```

Siehe auch

PostGIS_Full_Version, PostGIS_GEOS_Version, PostGIS_Lib_Version, PostGIS_LibXML_Version, PostGIS_PROJ_Version

7.24 PostGIS Grand Unified Custom Variables (GUCs)

7.24.1 postgis.backend

postgis.backend — Dieses Backend stellt eine Funktion zur Auswahl zwischen GEOS und SFCGAL zur Verfügung.

Beschreibung

Diese GUC hat nur Bedeutung, wenn Sie PostGIS mit SFCGAL Unterstützung kompiliert haben. Funktionen, welche sowohl bei GEOS als auch bei SFCGAL die gleiche Bezeichnung haben, werden standardmäßig mit dem `geos` Backend ausgeführt. Die Standardeinstellung wird mit dieser Variablen überschrieben und SFCGAL für den Aufruf verwendet.

Verfügbarkeit: 2.1.0

Beispiele

Setzt das Backend für die Dauer der Verbindung

```
set postgis.backend = sfcgal;
```

Setzt das Backend für neue Verbindungen zur Datenbank

```
ALTER DATABASE mygisdb SET postgis.backend = sfcgal;
```

Siehe auch

Section 7.21

7.24.2 postgis.gdal_datapath

postgis.gdal_datapath — Eine Konfigurationsmöglichkeit um den Wert von GDAL's GDAL_DATA Option zu setzen. Wenn sie nicht gesetzt ist, wird die Umgebungsvariable GDAL_DATA verwendet.

Beschreibung

Eine PostgreSQL GUC Variable zum setzten von GDAL's GDAL_DATA Option. Der `postgis.gdal_datapath` Wert sollte dem gesamten physischen Pfad zu den Datendateien von GDAL entsprechen.

Diese Konfigurationsmöglichkeit ist am nützlichsten auf Windows Plattformen, wo der Dateipfad von "data" nicht fest kodiert ist. Diese Option sollte auch gesetzt werden, wenn sich die Datendateien nicht in dem von GDAL erwarteten Pfad befinden.

Note
Diese Option kann in der Konfigurationsdatei "postgresql.conf" gesetzt werden. Sie kann auch pro Verbindung oder pro Transaktion gesetzt werden.

Verfügbarkeit: 2.2.0

Note
Zusätzliche Informationen über GDAL_DATA ist unter den Konfigurationsmöglichkeiten für GDAL zu finden.

Beispiele

Den `postgis.gdal_datapath` setzen oder zurücksetzen

```
SET postgis.gdal_datapath TO '/usr/local/share/gdal.hidden';
SET postgis.gdal_datapath TO default;
```

Auf Windows für eine bestimmte Datenbank setzen

```
ALTER DATABASE gisdb
SET postgis.gdal_datapath = 'C:/Program Files/PostgreSQL/9.3/gdal-data';
```

Siehe auch

PostGIS_GDAL_Version, ST_Transform

7.24.3 postgis.gdal_enabled_drivers

postgis.gdal_enabled_drivers — Eine Konfigurationsmöglichkeit um einen GDAL Treiber in der PostGIS Umgebung zu aktivieren. Beeinflusst die Konfigurationsvariable GDAL_SKIP von GDAL.

Beschreibung

Eine Konfigurationsmöglichkeit um einen GDAL Treiber in der PostGIS Umgebung zu aktivieren. Beeinflusst die Konfigurationsvariable GDAL_SKIP von GDAL. Diese Option kann in der PostgreSQL Konfigurationsdatei "postgresql.conf" gesetzt werden. Sie kann aber auch pro Verbindung oder pro Transaktion gesetzt werden.

Der Ausgangswert von `postgis.gdal_enabled_drivers` kann auch beim Startprozess von PostgreSQL gesetzt werden, nämlich durch die Übergabe der Umgebungsvariablen `POSTGIS_GDAL_ENABLED_DRIVERS`, welche die Liste der aktivierten Treiber enthält.

Aktivierte GDAL Treiber können auch über die Kurzbezeichnung oder den Code des Treibers bestimmt werden. Kurzbezeichnungen und Codes für die Treiber finden sich unter GDAL Raster Formate Es können mehrere, durch Leerzeichen getrennte Treiber angegeben werden.

Note

Für `postgis.gdal_enabled_drivers` sind drei spezielle, case-sensitive Codes verfügbar.

- `DISABLE_ALL` deaktiviert alle GDAL-Treiber. Falls vorhanden, überschreibt `DISABLE_ALL` alle anderen Werte in `postgis.gdal_enabled_drivers`.
- `ENABLE_ALL` aktiviert alle GDAL-Treiber.
- `VSICURL` aktiviert GDAL's `/vsicurl/` virtuelles Dateisystem.

Falls `postgis.gdal_enabled_drivers`auf DISABLE_ALL gesetzt ist, kommt es bei der Anwendung von out-db Rastern, ST_FromGDALRaster(), ST_AsGDALRaster(), ST_AsTIFF(), ST_AsJPEG() und ST_AsPNG() zu Fehlermeldungen.

Note

`postgis.gdal_enabled_drivers` wird bei der Standardinstallation von PostGIS auf DISABLE_ALL gesetzt.

Note
Weiterführende Informationen über GDAL_SKIP ist auf GDAL's Configuration Options zu finden.

Verfügbarkeit: 2.2.0

Beispiele

`postgis.gdal_enabled_drivers` setzen und zurücksetzen

Bestimmt das Backend, das für alle neuen Verbindungen zur Datenbank verwendet wird

```
ALTER DATABASE mygisdb SET postgis.gdal_enabled_drivers TO 'GTiff PNG JPEG';
```

Setzt die standardmäßig aktivierten Treiber für alle neuen Verbindungen zum Server. Benötigt Administratorrechte und PostgreSQL 9.4+. Beachten Sie aber bitte, dass die Datenbank-, Sitzungs- und Benutzereinstellungen dies überschreiben.

```
ALTER SYSTEM SET postgis.gdal_enabled_drivers TO 'GTiff PNG JPEG';
SELECT pg_reload_conf();
```

```
SET postgis.gdal_enabled_drivers TO 'GTiff PNG JPEG';
SET postgis.gdal_enabled_drivers = default;
```

Aktiviert alle GDAL-Treiber

```
SET postgis.gdal_enabled_drivers = 'ENABLE_ALL';
```

Deaktiviert alle GDAL-Treiber

```
SET postgis.gdal_enabled_drivers = 'DISABLE_ALL';
```

Siehe auch

ST_FromGDALRaster, ST_AsGDALRaster, ST_AsTIFF, ST_AsPNG, ST_AsJPEG, postgis.enable_outdb_rasters

7.24.4 postgis.enable_outdb_rasters

postgis.enable_outdb_rasters — Eine boolesche Konfigurationsmöglichkeit um den Zugriff auf out-db Rasterbänder zu ermöglichen

Beschreibung

Eine boolesche Konfigurationsmöglichkeit um den Zugriff auf out-db Rasterbänder zu ermöglichen. Diese Option kann in der PostgreSQL Konfigurationsdatei "postgresql.conf" gesetzt werden. Kann aber auch pro Verbindung oder pro Transaktion gesetzt werden.

Der Ausgangswert von `postgis.enable_outdb_rasters` kann auch beim Startprozess von PostgreSQL gesetzt werden, nämlich durch die Übergabe der Umgebungsvariablen `POSTGIS_ENABLE_OUTDB_RASTERS`, welche ungleich null sein muss.

Note
Auch wenn `postgis.enable_outdb_rasters` True ist, bestimmt die GUC `postgis.enable_outdb_rasters` die zugänglichen Rasterformate.

Note
Bei der Standardinstallation von PostGIS ist `postgis.enable_outdb_rasters` auf False gesetzt.

Verfügbarkeit: 2.2.0

Beispiele

`postgis.enable_outdb_rasters` setzen oder zurücksetzen

```
SET postgis.enable_outdb_rasters TO True;
SET postgis.enable_outdb_rasters = default;
SET postgis.enable_outdb_rasters = True;
SET postgis.enable_outdb_rasters = False;
```

Set for specific database

```
ALTER DATABASE gisdb SET postgis.enable_outdb_rasters = true;
```

Setting for whole database cluster. You need to reconnect to the database for changes to take effect.

```
--writes to postgres.auto.conf
ALTER SYSTEM postgis.enable_outdb_rasters = true;
 --Reloads postgres conf
SELECT pg_reload_conf();
```

Siehe auch

postgis.gdal_enabled_drivers postgis.gdal_config_options

7.24.5 postgis.gdal_config_options

postgis.gdal_config_options — Eine boolesche Konfigurationsmöglichkeit um den Zugriff auf out-db Rasterbänder zu ermöglichen

Beschreibung

A string configuration to set options used when working with an out-db raster. Configuration options control things like how much space GDAL allocates to local data cache, whether to read overviews, and what access keys to use for remote out-db data sources.

Verfügbarkeit: 2.2.0

Beispiele

`postgis.enable_outdb_rasters` setzen oder zurücksetzen

```
SET postgis.gdal_config_options = 'AWS_ACCESS_KEY_ID=xxxxxxxxxxxxxxx AWS_SECRET_ACCESS_KEY= ↩
    yyyyyyyyyyyyyyyyyyyyyyyyyy';
```

Set `postgis.gdal_vsi_options` just for the *current transaction* using the `LOCAL` keyword:

```
SET LOCAL postgis.gdal_config_options = 'AWS_ACCESS_KEY_ID=xxxxxxxxxxxxxxx  ↩
    AWS_SECRET_ACCESS_KEY=yyyyyyyyyyyyyyyyyyyyyyyyy';
```

Siehe auch

postgis.enable_outdb_rasters postgis.gdal_enabled_drivers

7.25 Troubleshooting Functions

7.25.1 PostGIS_AddBBox

PostGIS_AddBBox — Add bounding box to the geometry.

Synopsis

geometry **PostGIS_AddBBox**(geometry geomA);

Beschreibung

Add bounding box to the geometry. This would make bounding box based queries faster, but will increase the size of the geometry.

Note
Bounding boxes are automatically added to geometries so in general this is not needed unless the generated bounding box somehow becomes corrupted or you have an old install that is lacking bounding boxes. Then you need to drop the old and readd.

This method supports Circular Strings and Curves.

Examples

```
UPDATE sometable
 SET geom =  PostGIS_AddBBox(geom)
 WHERE PostGIS_HasBBox(geom) = false;
```

Siehe auch

PostGIS_DropBBox, PostGIS_HasBBox

7.25.2 PostGIS_DropBBox

PostGIS_DropBBox — Drop the bounding box cache from the geometry.

Synopsis

geometry **PostGIS_DropBBox**(geometry geomA);

Beschreibung

Drop the bounding box cache from the geometry. This reduces geometry size, but makes bounding-box based queries slower. It is also used to drop a corrupt bounding box. A tale-tell sign of a corrupt cached bounding box is when your ST_Intersects and other relation queries leave out geometries that rightfully should return true.

Note
Bounding boxes are automatically added to geometries and improve speed of queries so in general this is not needed unless the generated bounding box somehow becomes corrupted or you have an old install that is lacking bounding boxes. Then you need to drop the old and readd. This kind of corruption has been observed in 8.3-8.3.6 series whereby cached bboxes were not always recalculated when a geometry changed and upgrading to a newer version without a dump reload will not correct already corrupted boxes. So one can manually correct using below and readd the bbox or do a dump reload.

This method supports Circular Strings and Curves.

Examples

```
--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);
```

Siehe auch

PostGIS_AddBBox, PostGIS_HasBBox, Box2D

7.25.3 PostGIS_HasBBox

PostGIS_HasBBox — Returns TRUE if the bbox of this geometry is cached, FALSE otherwise.

Synopsis

boolean **PostGIS_HasBBox**(geometry geomA);

Beschreibung

Returns TRUE if the bbox of this geometry is cached, FALSE otherwise. Use PostGIS_AddBBox and PostGIS_DropBBox to control caching.

This method supports Circular Strings and Curves.

Examples

```
SELECT geom
FROM sometable WHERE PostGIS_HasBBox(geom) = false;
```

Siehe auch

PostGIS_AddBBox, PostGIS_DropBBox

Chapter 8

Topologie

Die topologischen Datentypen und Funktionen von PostGIS werden für die Verwaltung von topologischen Objekten wie Maschen, Kanten und Knoten verwendet.

Sandro Santilli's Vortrag auf der Tagung "PostGIS Day Paris 2011" liefert eine gute Übersicht über die PostGIS Topologie und deren Perspektiven Topology with PostGIS 2.0 slide deck.

Vincent Picavet provides a good synopsis and overview of what is Topology, how is it used, and various FOSS4G tools that support it in PostGIS Topology PGConf EU 2012.

Ein Beispiel für eine topologische Geodatenbank ist die US Census Topologically Integrated Geographic Encoding and Referencing System (TIGER) Datenbank. Zum Experimentieren mit der PostGIS Topologie stehen unter Topology_Load_Tiger Daten zur Verfügung.

Das PostGIS Modul "Topologie" gab es auch schon in früheren Versionen von PostGIS, es war aber nie Teil der offiziellen PostGIS Dokumentation. In PostGIS 2.0.0 fand eine umfangreiche Überarbeitung statt, um überholte Funktionen zu entfernen, bekannte Probleme mit der Bedienbarkeit zu bereinigen, bessere Dokumentation der Funktionalität, Einführung neuer Funktionen, und eine bessere Übereinstimmung mit den SQL-MM Normen zu erreichen.

Genauere Angaben zu diesem Projekt finden sich unter PostGIS Topology Wiki

Alle Funktionen und Tabellen, die zu diesem Modul gehören, sind im Schema mit der Bezeichnung `topology` installiert.

Funktionen die im SQL/MM Standard definiert sind erhalten das Präfix ST_, PostGIS eigene Funktionen erhalten kein Präfix.

Ab PostGIS 2.0 wird die Topologie Unterstützung standardmäßig mitkompiliert und kann bei der Konfiguration mittels der Konfigurationsoption "--without-topology", wie in Chapter 2 beschrieben, deaktiviert werden.

8.1 Topologische Datentypen

8.1.1 getfaceedges_returntype

getfaceedges_returntype — A composite type that consists of a sequence number and an edge number.

Beschreibung

A composite type that consists of a sequence number and an edge number. This is the return type for `ST_GetFaceEdges` and `GetNodeEdges` functions.

1. `sequence` ist eine Ganzzahl: Sie verweist auf eine Topologie, die in der Tabelle topology.topology definiert ist. In dieser Tabelle sind das Schema, das die Topologie enthält, und die SRID verzeichnet.
2. `edge` ist eine Ganzzahl: Der Identifikator einer Kante.

8.1.2 TopoGeometry

TopoGeometry — Ein zusammengesetzter Typ, der eine topologisch festgelegte Geometrie darstellt.

Beschreibung

Ein zusammengesetzter Datentyp, der auf eine topologische Geometrie in einem bestimmten topologischen Layer verweist und einen spezifischen Datentyp und eine eindeutige ID hat. Folgende Bestandteile bilden die Elemente einer TopoGeometry: topology_id, layer_id, id Ganzzahl, type Ganzzahl.

1. `topology_id` ist eine Ganzzahl: Sie verweist auf eine Topologie, die in der Tabelle topology.topology definiert ist. In dieser Tabelle sind das Schema, das die Topologie enthält, und die SRID verzeichnet.
2. `layer_id` ist eine Ganzzahl: Die layer_id in der Tabelle topology.layer zu der die TopoGeometry gehört. Die Kombination aus topology_id und layer_id liefert eine eindeutige Referenz in der Tabelle topology.layer.
3. `id` ist eine Ganzzahl: Die id ist eine automatisch erzeugte Sequenznummer, welche die TopoGeometry in dem jeweiligen topologischen Layer eindeutig ausweist.
4. `type` ist eine Ganzzahl zwischen 1 und 4, welche den geometrischen Datentyp festlegt: 1:[Multi]Point, 2:[Multi]Line, 3:[Multi]Polygon, 4:GeometryCollection

Verhaltensweise bei der Typumwandlung

In diesem Abschnitt sind die für diesen Datentyp erlaubten impliziten und expliziten Typumwandlungen beschrieben.

Typumwandlung nach	Verhaltensweise
Geometrie	implizit

Siehe auch

CreateTopoGeom

8.1.3 validatetopology_returntype

validatetopology_returntype — Ein zusammengesetzter Datentyp, der aus einer Fehlermeldung und id1 und id2 besteht. id1 und id2 deuten auf die Stelle hin, an der der Fehler auftrat. Dies ist der von `ValidateTopology` zurückgegebene Datentyp.

Beschreibung

Ein zusammengesetzter Datentyp, der aus einer Fehlermeldung und zwei Ganzzahlen besteht. Die Funktion ValidateTopology gibt eine Menge dieser Datentypen zurück, um bei der Validierung gefundene Fehler zu beschreiben. Unter id1 und id2 sind die ids der topologischen Objekte verzeichnet, die an dem Fehler beteiligt sind.

1. `error` ist varchar: Gibt die Art des Fehlers an.

   Aktuell existieren folgende Fehlerbeschreibungen: zusammenfallende Knoten/coincident nodes, Kante ist nicht simple/edge not simple, Kanten- und Endknotengeometrie stimmen nicht überein/edge end node geometry mis-match, Kanten- und Anfangsknotengeometrie stimmen nicht überein/edge start node geometry mismatch, Masche überlappt Masche/face overlaps face, Masche innerhalb einer Masche/face within face.
2. `id1` ist eine ganze Zahl: Gibt den Identifikator einer Kante / Masche / Knoten in der Fehlermeldung an.
3. `id2` ist eine ganze Zahl: Wenn 2 Objekte in den Fehler involviert sind verweist diese Zahl auf die zweite Kante / oder Knoten.

Siehe auch

ValidateTopology

8.2 Topologische Domänen

8.2.1 TopoElement

TopoElement — Ein Feld mit 2 Ganzzahlen, welches in der Regel für die Auffindung einer Komponente einer TopoGeometry dient.

Beschreibung

Ein Feld mit 2 Ganzzahlen, welches einen Bestandteil einer einfachen oder hierarchischen TopoGeometry abbildet.

Im Falle einer einfachen TopoGeometry ist das erste Element des Feldes der Identifikator einer topologischen Elementarstruktur und das zweite Element der Typ (1:Knoten, 2:Kante, 3:Masche). Im Falle einer hierarchischen TopoGeometry ist das erste Element des Feldes der Identifikator der Kind-TopoGeometry und das zweite Element der Identifikator des Layers.

Note
Bei jeder gegebenen hierarchischen TopoGeometry werden alle Kindklassen der TopoGeometrie vom selben Kindlayer abgeleitet, so wie dies in dem Datensatz von "topology.layer" für den Layer der TopoGeometry definiert ist.

Beispiele

```
SELECT te[1] AS id, te[2] AS type FROM
( SELECT ARRAY[1,2]::topology.topoelement AS te ) f;
 id | type
----+------
  1 |    2
```

```
SELECT ARRAY[1,2]::topology.topoelement;
  te
-------
 {1,2}
```

```
--Example of what happens when you try to case a 3 element array to topoelement
-- NOTE: topoement has to be a 2 element array so fails dimension check
SELECT ARRAY[1,2,3]::topology.topoelement;
ERROR:  value for domain topology.topoelement violates check constraint "dimensions"
```

Siehe auch

GetTopoGeomElements, TopoElementArray, TopoGeometry, TopoGeom_addElement, TopoGeom_remElement

8.2.2 TopoElementArray

TopoElementArray — Ein Feld mit TopoElement Objekten.

Beschreibung

Ein Feld mit 1 oder mehreren TopoElement Objekten; wird hauptsächlich verwendet, um Bestandteile einer TopoGeometry herumzureichen.

Beispiele

```
SELECT '{{1,2},{4,3}}'::topology.topoelementarray As tea;
  tea
-------
{{1,2},{4,3}}

-- more verbose equivalent --
SELECT ARRAY[ARRAY[1,2], ARRAY[4,3]]::topology.topoelementarray As tea;

  tea
-------
{{1,2},{4,3}}

--using the array agg function packaged with topology --
SELECT topology.TopoElementArray_Agg(ARRAY[e,t]) As tea
  FROM generate_series(1,4) As e CROSS JOIN generate_series(1,3) As t;
  tea
-----------------------------------------------------------------------
{{1,1},{1,2},{1,3},{2,1},{2,2},{2,3},{3,1},{3,2},{3,3},{4,1},{4,2},{4,3}}
```

```
SELECT '{{1,2,4},{3,4,5}}'::topology.topoelementarray As tea;
ERROR:  value for domain topology.topoelementarray violates check constraint "dimensions"
```

Siehe auch

TopoElement, GetTopoGeomElementArray, TopoElementArray_Agg

8.3 Verwaltung von Topologie und TopoGeometry

8.3.1 AddTopoGeometryColumn

AddTopoGeometryColumn — Fügt ein TopoGeometry Attribut an eine bestehende Tabelle an, registriert dieses neue Attribut als einen Layer in topology.layer und gibt die neue layer_id zurück.

Synopsis

integer **AddTopoGeometryColumn**(varchar topology_name, varchar schema_name, varchar table_name, varchar column_name, varchar feature_type);
integer **AddTopoGeometryColumn**(varchar topology_name, varchar schema_name, varchar table_name, varchar column_name, varchar feature_type, integer child_layer);

Beschreibung

Jedes TopoGeometry Objekt gehört zu einem bestimmten Layer einer bestimmten Topologie. Bevor Sie ein TopoGeometry Objekt erzeugen, müssen Sie dessen topologischen Layer erzeugen. Ein topologischer Layer ist eine Assoziation einer Featuretabelle mit der Topologie. Er enthält auch Angaben zum Typ und zur Hierarchie. Man erzeugt einen Layer mittels der Funktion AddTopoGeometryColumn():

Diese Funktion fügt sowohl das angeforderte Attribut an die Tabelle als auch einen Datensatz mit der angegebenen Information in die Tabelle topology.layer.

Wenn Sie den [child_layer] nicht angeben (oder auf NULL setzen), dann enthält dieser Layer elementare TopoGeometries (aus topologischen Elementarstrukturen zusammengesetzt). Andernfalls enthält dieser Layer hierarchische TopoGeometries (aus den TopoGeometries des child_layer zusammengesetzt).

Sobald der Layer erstellt wurde (seine id von der Funktion AddTopoGeometryColumn zurückgegeben wurde), können Sie TopoGeometry Objekte in ihm erzeugen

Valid `feature_types` are: POINT, MULTIPOINT, LINE, MULTILINE, POLYGON, MULTIPOLYGON, COLLECTION

Availability: 1.1

Beispiele

```
-- Note for this example we created our new table in the ma_topo schema
-- though we could have created it in a different schema -- in which case topology_name and ↩
     schema_name would be different
CREATE SCHEMA ma;
CREATE TABLE ma.parcels(gid serial, parcel_id varchar(20) PRIMARY KEY, address text);
SELECT topology.AddTopoGeometryColumn('ma_topo', 'ma', 'parcels', 'topo', 'POLYGON');
```

```
CREATE SCHEMA ri;
CREATE TABLE ri.roads(gid serial PRIMARY KEY, road_name text);
SELECT topology.AddTopoGeometryColumn('ri_topo', 'ri', 'roads', 'topo', 'LINE');
```

Siehe auch

DropTopoGeometryColumn, toTopoGeom, CreateTopology, CreateTopoGeom

8.3.2 RenameTopoGeometryColumn

RenameTopoGeometryColumn — Renames a topogeometry column

Synopsis

topology.layer **RenameTopoGeometryColumn**(regclass layer_table, name feature_column, name new_name);

Beschreibung

This function changes the name of an existing TopoGeometry column ensuring metadata information about it is updated accordingly.

Availability: 3.4.0

Beispiele

```
SELECT topology.RenameTopoGeometryColumn('public.parcels', 'topogeom', 'tgeom');
```

Siehe auch

AddTopoGeometryColumn, RenameTopology

8.3.3 DropTopology

DropTopology — Bitte mit Vorsicht verwenden: Löscht ein topologisches Schema und dessen Referenz in der Tabelle topology.topology, sowie die Referenzen zu den Tabellen in diesem Schema aus der Tabelle geometry_columns.

Synopsis

integer **DropTopology**(varchar topology_schema_name);

Beschreibung

Löscht ein topologisches Schema und dessen Referenz in der Tabelle topology.topology, sowie die Referenzen zu den Tabellen in diesem Schema aus der Tabelle geometry_columns. Diese Funktion sollte MIT VORSICHT BENUTZT werden, da damit unabsichtlich Daten zerstört werden können. Falls das Schema nicht existiert, werden nur die Referenzeinträge des bezeichneten Schemas gelöscht.

Availability: 1.1

Beispiele

Löscht das Schema "ma_topo" kaskadierend und entfernt alle Referenzen in topology.topology und in geometry_columns.

```
SELECT topology.DropTopology('ma_topo');
```

Siehe auch

DropTopoGeometryColumn

8.3.4 RenameTopology

RenameTopology — Renames a topology

Synopsis

varchar **RenameTopology**(varchar old_name, varchar new_name);

Beschreibung

Renames a topology schema, updating its metadata record in the `topology.topology` table.

Availability: 3.4.0

Beispiele

Rename a topology from `topo_stage` to `topo_prod`.

```
SELECT topology.RenameTopology('topo_stage', 'topo_prod');
```

Siehe auch

CopyTopology, RenameTopoGeometryColumn

8.3.5 DropTopoGeometryColumn

DropTopoGeometryColumn — Entfernt ein TopoGeometry-Attribut aus der Tabelle mit der Bezeichnung `table_name` im Schema `schema_name` und entfernt die Registrierung der Attribute aus der Tabelle "topology.layer".

Synopsis

text **DropTopoGeometryColumn**(varchar schema_name, varchar table_name, varchar column_name);

Beschreibung

Entfernt ein TopoGeometry-Attribut aus der Tabelle mit der Bezeichnung `table_name` im Schema `schema_name` und entfernt die Registrierung der Attribute aus der Tabelle "topology.layer". Gibt eine Zusammenfassung des Löschstatus aus. ANMERKUNG: vor dem Löschen werden alle Werte auf NULL gesetzt, um die Überprüfung der referenziellen Integrität zu umgehen.

Availability: 1.1

Beispiele

```
SELECT topology.DropTopoGeometryColumn('ma_topo', 'parcel_topo', 'topo');
```

Siehe auch

AddTopoGeometryColumn

8.3.6 Populate_Topology_Layer

Populate_Topology_Layer — Fügt fehlende Einträge zu der Tabelle topology.layer hinzu, indem Metadaten aus den topologischen Tabellen ausgelesen werden.

Synopsis

setof record **Populate_Topology_Layer**();

Beschreibung

Trägt fehlende Einträge in der Tabelle `topology.layer` ein, indem die topologischen Constraints und Tabellen inspiziert werden. Diese Funktion ist nach der Wiederherstellung von Schemata mit topologischen Daten sinnvoll, um Einträge im Topologie-Katalog zu reparieren.

Wirft eine Liste der erzeugten Einträge aus. Die zurückgegebenen Attribute sind `schema_name`, `table_name` und `feature_colu`

Verfügbarkeit: 2.3.0

Beispiele

```
SELECT CreateTopology('strk_topo');
CREATE SCHEMA strk;
CREATE TABLE strk.parcels(gid serial, parcel_id varchar(20) PRIMARY KEY, address text);
SELECT topology.AddTopoGeometryColumn('strk_topo', 'strk', 'parcels', 'topo', 'POLYGON');
-- this will return no records because this feature is already registered
SELECT *
  FROM topology.Populate_Topology_Layer();

-- let's rebuild
TRUNCATE TABLE topology.layer;

SELECT *
  FROM topology.Populate_Topology_Layer();

SELECT topology_id,layer_id, schema_name As sn, table_name As tn, feature_column As fc
FROM topology.layer;
```

```
schema_name | table_name | feature_column
-------------+------------+----------------
 strk        | parcels    | topo
(1 row)

 topology_id | layer_id |  sn  |   tn    |  fc
-------------+----------+------+---------+------
           2 |        2 | strk | parcels | topo
(1 row)
```

Siehe auch

AddTopoGeometryColumn

8.3.7 TopologySummary

TopologySummary — Nimmt den Namen einer Topologie und liefert eine Zusammenfassung der Gesamtsummen der Typen und Objekte in der Topologie.

Synopsis

text **TopologySummary**(varchar topology_schema_name);

Beschreibung

Nimmt den Namen einer Topologie und liefert eine Zusammenfassung der Gesamtsummen der Typen und Objekte in der Topologie.

Verfügbarkeit: 2.0.0

Beispiele

```
SELECT topology.topologysummary('city_data');
                    topologysummary
------------------------------------------------------
 Topology city_data (329), SRID 4326, precision: 0
 22 nodes, 24 edges, 10 faces, 29 topogeoms in 5 layers
 Layer 1, type Polygonal (3), 9 topogeoms
  Deploy: features.land_parcels.feature
 Layer 2, type Puntal (1), 8 topogeoms
  Deploy: features.traffic_signs.feature
 Layer 3, type Lineal (2), 8 topogeoms
  Deploy: features.city_streets.feature
 Layer 4, type Polygonal (3), 3 topogeoms
  Hierarchy level 1, child layer 1
  Deploy: features.big_parcels.feature
 Layer 5, type Puntal (1), 1 topogeoms
  Hierarchy level 1, child layer 2
  Deploy: features.big_signs.feature
```

Siehe auch

Topology_Load_Tiger

8.3.8 ValidateTopology

ValidateTopology — Liefert eine Menge validatetopology_returntype Objekte, die Probleme mit der Topologie beschreiben.

Synopsis

setof validatetopology_returntype **ValidateTopology**(varchar toponame, geometry bbox);

Beschreibung

Returns a set of validatetopology_returntype objects detailing issues with topology, optionally limiting the check to the area specified by the `bbox` parameter.

List of possible errors, what they mean and what the returned ids represent are displayed below:

Error	id1	id2	Meaning
coincident nodes	Identifier of first node.	Identifier of second node.	Two nodes have the same geometry.
edge crosses node	Identifier of the edge.	Identifier of the node.	An edge has a node in its interior. See ST_Relate.
invalid edge	Identifier of the edge.		An edge geometry is invalid. See ST_IsValid.
edge not simple	Identifier of the edge.		An edge geometry has self-intersections. See ST_IsSimple.
edge crosses edge	Identifier of first edge.	Identifier of second edge.	Two edges have an interior intersection. See ST_Relate.
edge start node geometry mis-match	Identifier of the edge.	Identifier of the indicated start node.	The geometry of the node indicated as the starting node for an edge does not match the first point of the edge geometry. See ST_StartPoint.

Error	id1	id2	Meaning
edge end node geometry mis-match	Identifier of the edge.	Identifier of the indicated end node.	The geometry of the node indicated as the ending node for an edge does not match the last point of the edge geometry. See ST_EndPoint.
face without edges	Identifier of the orphaned face.		No edge reports an existing face on either of its sides (left_face, right_face).
face has no rings	Identifier of the partially-defined face.		Edges reporting a face on their sides do not form a ring.
face has wrong mbr	Identifier of the face with wrong mbr cache.		Minimum bounding rectangle of a face does not match minimum bounding box of the collection of edges reporting the face on their sides.
hole not in advertised face	Signed identifier of an edge, identifying the ring. See GetRingEdges.		A ring of edges reporting a face on its exterior is contained in different face.
not-isolated node has not-containing_face	Identifier of the ill-defined node.		A node which is reported as being on the boundary of one or more edges is indicating a containing face.
isolated node has containing_face	Identifier of the ill-defined node.		A node which is not reported as being on the boundary of any edges is lacking the indication of a containing face.
isolated node has wrong containing_face	Identifier of the misrepresented node.		A node which is not reported as being on the boundary of any edges indicates a containing face which is not the actual face containing it. See GetFaceContainingPoint.
invalid next_right_edge	Identifier of the misrepresented edge.	Signed id of the edge which should be indicated as the next right edge.	The edge indicated as the next edge encountered walking on the right side of an edge is wrong.
invalid next_left_edge	Identifier of the misrepresented edge.	Signed id of the edge which should be indicated as the next left edge.	The edge indicated as the next edge encountered walking on the left side of an edge is wrong.
mixed face labeling in ring	Signed identifier of an edge, identifying the ring. See GetRingEdges.		Edges in a ring indicate conflicting faces on the walking side. This is also known as a "Side Location Conflict".
non-closed ring	Signed identifier of an edge, identifying the ring. See GetRingEdges.		A ring of edges formed by following next_left_edge/next_right_edge attributes starts and ends on different nodes.

Error	id1	id2	Meaning
face has multiple shells	Identifier of the contended face.	Signed identifier of an edge, identifying the ring. See GetRingEdges.	More than a one ring of edges indicate the same face on its interior.

Verfügbarkeit: 1.0.0

Erweiterung: 2.0.0 effizientere Ermittlung sich überkreuzender Kanten. Falsch positive Fehlmeldungen von früheren Versionen fixiert.

Änderung: 2.2.0 Bei 'edge crosses node' wurden die Werte für id1 und id2 vertauscht, um mit der Fehlerbeschreibung konsistent zu sein.

Changed: 3.2.0 added optional bbox parameter, perform face labeling and edge linking checks.

Beispiele

```
SELECT * FROM  topology.ValidateTopology('ma_topo');
      error        | id1 | id2
-------------------+-----+-----
face without edges |   1 |
```

Siehe auch

validatetopology_returntype, Topology_Load_Tiger

8.3.9 ValidateTopologyRelation

ValidateTopologyRelation — Returns info about invalid topology relation records

Synopsis

setof record **ValidateTopologyRelation**(varchar toponame);

Beschreibung

Returns a set records giving information about invalidities in the relation table of the topology.

Availability: 3.2.0

Siehe auch

ValidateTopology

8.3.10 FindTopology

FindTopology — Returns a topology record by different means.

Synopsis

topology **FindTopology**(TopoGeometry topogeom);
topology **FindTopology**(regclass layerTable, name layerColumn);
topology **FindTopology**(name layerSchema, name layerTable, name layerColumn);
topology **FindTopology**(text topoName);
topology **FindTopology**(int id);

Beschreibung

Takes a topology identifier or the identifier of a topology-related object and returns a topology.topology record.

Availability: 3.2.0

Beispiele

```
SELECT name(findTopology('features.land_parcels', 'feature'));
   name
-----------
 city_data
(1 row)
```

Siehe auch

FindLayer

8.3.11 FindLayer

FindLayer — Returns a topology.layer record by different means.

Synopsis

topology.layer **FindLayer**(TopoGeometry tg);
topology.layer **FindLayer**(regclass layer_table, name feature_column);
topology.layer **FindLayer**(name schema_name, name table_name, name feature_column);
topology.layer **FindLayer**(integer topology_id, integer layer_id);

Beschreibung

Takes a layer identifier or the identifier of a topology-related object and returns a topology.layer record.

Availability: 3.2.0

Beispiele

```
SELECT layer_id(findLayer('features.land_parcels', 'feature'));
 layer_id
----------
        1
(1 row)
```

Siehe auch

FindTopology

8.4 Topology Statistics Management

Adding elements to a topology triggers many database queries for finding existing edges that will be split, adding nodes and updating edges that will node with the new linework. For this reason it is useful that statistics about the data in the topology tables are up-to-date.

PostGIS Topology population and editing functions do not automatically update the statistics because a updating stats after each and every change in a topology would be overkill, so it is the caller's duty to take care of that.

Note
That the statistics updated by autovacuum will NOT be visible to transactions which started before autovacuum process completed, so long-running transactions will need to run ANALYZE themselves, to use updated statistics.

8.5 Topologie Konstruktoren

8.5.1 CreateTopology

CreateTopology — Creates a new topology schema and registers it in the topology.topology table.

Synopsis

integer **CreateTopology**(varchar topology_schema_name);
integer **CreateTopology**(varchar topology_schema_name, integer srid);
integer **CreateTopology**(varchar topology_schema_name, integer srid, double precision prec);
integer **CreateTopology**(varchar topology_schema_name, integer srid, double precision prec, boolean hasz);

Beschreibung

Creates a new topology schema with name `topology_name` and registers it in the `topology.topology` table. Topologies must be uniquely named. The topology tables (`edge_data`, `face`, `node`,and `relation` are created in the schema. It returns the id of the topology.

The `srid` is the spatial reference system SRID for the topology.

The tolerance `prec` is measured in the units of the spatial reference system. The tolerance defaults to 0.

`hasz` defaults to false if not specified.

This is similar to the SQL/MM ST_InitTopoGeo but has more functionality.

Availability: 1.1

Enhanced: 2.0 added the signature accepting hasZ

Beispiele

Create a topology schema called `ma_topo` that stores edges and nodes in Massachusetts State Plane-meters (SRID = 26986). The tolerance represents 0.5 meters since the spatial reference system is meter-based.

```
SELECT topology.CreateTopology('ma_topo', 26986, 0.5);
```

Create a topology for Rhode Island called `ri_topo` in spatial reference system State Plane-feet (SRID = 3438)

```
SELECT topology.CreateTopology('ri_topo', 3438) AS topoid;
topoid
------
2
```

Siehe auch

Section 4.5, ST_InitTopoGeo, Topology_Load_Tiger

8.5.2 CopyTopology

CopyTopology — Makes a copy of a topology (nodes, edges, faces, layers and TopoGeometries) into a new schema

Synopsis

integer **CopyTopology**(varchar existing_topology_name, varchar new_name);

Beschreibung

Creates a new topology with name `new_name`, with SRID and precision copied from `existing_topology_name` The nodes, edges and faces in `existing_topology_name` are copied into the new topology, as well as Layers and their associated TopoGeometries.

Note
The new rows in the `topology.layer` table contain synthetic values for schema_name, table_name and feature_column. This is because the TopoGeometry objects exist only as a definition and are not yet available in a user-defined table.

Verfügbarkeit: 2.0.0

Beispiele

Make a backup of a topology called `ma_topo`.

```
SELECT topology.CopyTopology('ma_topo', 'ma_topo_backup');
```

Siehe auch

Section 4.5, CreateTopology, RenameTopology

8.5.3 ST_InitTopoGeo

ST_InitTopoGeo — Creates a new topology schema and registers it in the topology.topology table.

Synopsis

text **ST_InitTopoGeo**(varchar topology_schema_name);

Beschreibung

This is the SQL-MM equivalent of CreateTopology. It lacks options for spatial reference system and tolerance. it returns a text description of the topology creation, instead of the topology id.

Availability: 1.1

This method implements the SQL/MM specification. SQL-MM 3 Topo-Geo and Topo-Net 3: Routine Details: X.3.17

Beispiele

```
SELECT topology.ST_InitTopoGeo('topo_schema_to_create') AS topocreation;
                      astopocreation
------------------------------------------------------------
 Topology-Geometry 'topo_schema_to_create' (id:7) created.
```

Siehe auch

CreateTopology

8.5.4 ST_CreateTopoGeo

ST_CreateTopoGeo — Fügt eine Sammlung von Geometrien an eine leere Topologie an und gibt eine Bestätigungsmeldung aus.

Synopsis

text **ST_CreateTopoGeo**(varchar atopology, geometry acollection);

Beschreibung

Fügt eine Sammelgeometrie einer leeren Topologie hinzu und gibt eine Bestätigungsmeldung aus.

Nützlich um eine leere Topologie zu befüllen.

Verfügbarkeit: 2.0

This method implements the SQL/MM specification. SQL-MM: Topo-Geo and Topo-Net 3: Routine Details -- X.3.18

Beispiele

```
-- Populate topology --
SELECT topology.ST_CreateTopoGeo('ri_topo',
 ST_GeomFromText('MULTILINESTRING((384744 236928,384750 236923,384769 236911,384799  ↩
     236895,384811 236890,384833 236884,
  384844 236882,384866 236881,384879 236883,384954 236898,385087 236932,385117 236938,
  385167 236938,385203 236941,385224 236946,385233 236950,385241 236956,385254 236971,
  385260 236979,385268 236999,385273 237018,385273 237037,385271 237047,385267 237057,
  385225 237125,385210 237144,385192 237161,385167 237192,385162 237202,385159 237214,
  385159 237227,385162 237241,385166 237256,385196 237324,385209 237345,385234 237375,
  385237 237383,385238 237399,385236 237407,385227 237419,385213 237430,385193 237439,
  385174 237451,385170 237455,385169 237460,385171 237475,385181 237503,385190 237521,
  385200 237533,385206 237538,385213 237541,385221 237542,385235 237540,385242 237541,
  385249 237544,385260 237555,385270 237570,385289 237584,385292 237589,385291  ↩
     237596,385284 237630))',3438)
  );

      st_createtopogeo
----------------------------
 Topology ri_topo populated


-- create tables and topo geometries --
CREATE TABLE ri.roads(gid serial PRIMARY KEY, road_name text);

SELECT topology.AddTopoGeometryColumn('ri_topo', 'ri', 'roads', 'topo', 'LINE');
```

Siehe auch

AddTopoGeometryColumn, CreateTopology, DropTopology

8.5.5 TopoGeo_AddPoint

TopoGeo_AddPoint — Fügt einen Punkt, unter Berücksichtigung einer Toleranz, an eine bestehende Topologie an. Existierende Kanten werden eventuell aufgetrennt.

Synopsis

integer **TopoGeo_AddPoint**(varchar atopology, geometry apoint, float8 tolerance);

Beschreibung

Fügt einen Punkt an eine bestehende Topologie an und gibt dessen Identifikator zurück. Der vorgegebene Punkt wird von bestehenden Knoten oder Kanten, innerhalb einer gegebenen Toleranz, gefangen. Eine existierende Kante kann durch den gefangenen Punkt aufgetrennt werden.

Verfügbarkeit: 2.0.0

Siehe auch

TopoGeo_AddLineString, TopoGeo_AddPolygon, AddNode, CreateTopology

8.5.6 TopoGeo_AddLineString

TopoGeo_AddLineString — Fügt einen Linienzug, unter Berücksichtigung einer Toleranz, an eine bestehende Topologie an. Existierende Kanten/Maschen werden eventuell aufgetrennt. Gibt den Identifikator der Kante aus.

Synopsis

SETOF integer **TopoGeo_AddLineString**(varchar atopology, geometry aline, float8 tolerance);

Beschreibung

Fügt einen Linienzug an eine bestehende Topologie an und gibt die Identifikatoren der Kanten zurück, die diedessen Identifikator zurück. Der vorgegebene Punkt wird von bestehenden Knoten oder Kanten, innerhalb einer gegebenen Toleranz, gefangen. Eine existierende Kante kann durch den gefangenen Punkt aufgetrennt werden.

Note
Updating statistics about topologies being loaded via this function is up to caller, see maintaining statistics during topology editing and population.

Verfügbarkeit: 2.0.0

Siehe auch

TopoGeo_AddPoint, TopoGeo_AddPolygon, AddEdge, CreateTopology

8.5.7 TopoGeo_AddPolygon

TopoGeo_AddPolygon — Fügt ein Polygon, unter Berücksichtigung einer Toleranz, an eine bestehende Topologie an. Existierende Kanten/Maschen werden eventuell aufgetrennt. Gibt den Identifikator der Masche zurück.

Synopsis

SETOF integer **TopoGeo_AddPolygon**(varchar atopology, geometry apoly, float8 tolerance);

Beschreibung

Fügt ein Polygon zu einer existierenden Topologie hinzu und gibt die Identifikatoren der Maschen aus, aus denen es gebildet ist. Die Begrenzung des gegebenen Polygons wird innerhalb der angegebenen Toleranz an bestehenden Knoten und Kanten gefangen. Gegebenenfalls werden existierende Kanten und Maschen an der Begrenzung des neuen Polygons geteilt.

Note
Updating statistics about topologies being loaded via this function is up to caller, see maintaining statistics during topology editing and population.

Verfügbarkeit: 2.0.0

Siehe auch

TopoGeo_AddPoint, TopoGeo_AddLineString, AddFace, CreateTopology

8.6 Topologie Editoren

8.6.1 ST_AddIsoNode

ST_AddIsoNode — Fügt einen isolierten Knoten zu einer Masche in einer Topologie hinzu und gibt die "nodeid" des neuen Knotens aus. Falls die Masche NULL ist, wird der Knoten dennoch erstellt.

Synopsis

integer **ST_AddIsoNode**(varchar atopology, integer aface, geometry apoint);

Beschreibung

Fügt einen isolierten Knoten mit der Punktlage `apoint` zu einer bestehenden Masche mit der "faceid" `aface` zu einer Topologie `atopology` und gibt die "nodeid" des neuen Knoten aus.

Wenn das Koordinatenreferenzsystem (SRID) der Punktgeometrie nicht mit dem der Topologie übereinstimmt, `apoint` keine Punktgeometrie ist, der Punkt NULL ist, oder der Punkt eine bestehende Kante (auch an den Begrenzungen) schneidet, wird eine Fehlermeldung ausgegeben. Falls der Punkt bereits als Knoten existiert, wird ebenfalls eine Fehlermeldung ausgegeben.

Wenn `aface` nicht NULL ist und `apoint` nicht innerhalb der Masche liegt, wird eine Fehlermeldung ausgegeben.

Availability: 1.1

This method implements the SQL/MM specification. SQL-MM: Topo-Net Routines: X+1.3.1

Beispiele

Siehe auch

AddNode, CreateTopology, DropTopology, ST_Intersects

8.6.2 ST_AddIsoEdge

ST_AddIsoEdge — Fügt eine isolierte Kante, die durch die Geometrie `alinestring` festgelegt wird zu einer Topologie hinzu, indem zwei bestehende isolierte Knoten `anode` und `anothernode` verbunden werden. Gibt die "edgeid" der neuen Kante aus.

Synopsis

integer **ST_AddIsoEdge**(varchar atopology, integer anode, integer anothernode, geometry alinestring);

Beschreibung

Fügt eine isolierte Kante, die durch die Geometrie `alinestring` festgelegt wird zu einer Topologie hinzu, indem zwei bestehende isolierte Knoten `anode` und `anothernode` verbunden werden. Gibt die "edgeid" der neuen Kante aus.

Wenn das Koordinatenreferenzsystem (SRID) der Geometrie `alinestring` nicht mit dem der Topologie übereinstimmt, irgendein Eingabewert NULL ist, die Knoten in mehreren Maschen enthalten sind, oder die Knoten Anfangs- oder Endknoten einer bestehenden Kante darstellen, wird eine Fehlermeldung ausgegeben.

Wenn `alinestring` nicht innerhalb der Masche liegt zu der `anode` und `anothernode` gehören, dann wird eine Fehlermeldung ausgegeben.

Wenn `anode` und `anothernode` nicht Anfangs- und Endpunkt von `alinestring` sind, wird eine Fehlermeldung ausgegeben.

Availability: 1.1

This method implements the SQL/MM specification. SQL-MM: Topo-Geo and Topo-Net 3: Routine Details: X.3.4

Beispiele

Siehe auch

ST_AddIsoNode, ST_IsSimple, ST_Within

8.6.3 ST_AddEdgeNewFaces

ST_AddEdgeNewFaces — Fügt eine Kante hinzu. Falls dabei eine Masche aufgetrennt wird, so wird die ursprüngliche Masche gelöscht und durch zwei neue Maschen ersetzt.

Synopsis

integer **ST_AddEdgeNewFaces**(varchar atopology, integer anode, integer anothernode, geometry acurve);

Beschreibung

Fügt eine Kante hinzu. Falls dabei eine Masche aufgetrennt wird, so wird die ursprüngliche Masche gelöscht und durch zwei neue Maschen ersetzt. Gibt die ID der hinzugefügten Kante aus.

Führt ein entsprechendes Update auf alle verbundenen Kanten und Beziehungen durch.

Wenn irgendwelche Übergabewerte NULL sind, die angegebenen Knoten unbekannt sind (müssen bereits in der `node` Tabelle des Schemas "topology" existieren), `acurve` kein `LINESTRING` ist, oder `anode` und `anothernode` nicht die Anfangs- und Endpunkte von `acurve` sind, dann wird eine Fehlermeldung ausgegeben.

Wenn das Koordinatenreferenzsystem (SRID) der Geometrie `acurve` nicht mit jener der Topologie übereinstimmt, wird eine Fehlermeldung ausgegeben.

Verfügbarkeit: 2.0

This method implements the SQL/MM specification. SQL-MM: Topo-Geo and Topo-Net 3: Routine Details: X.3.12

Beispiele

Siehe auch

ST_RemEdgeNewFace

ST_AddEdgeModFace

8.6.4 ST_AddEdgeModFace

ST_AddEdgeModFace — Fügt eine Kante hinzu. Falls dabei eine Masche aufgetrennt wird, so wird die ursprüngliche Masche angepasst und eine weitere Masche hinzugefügt.

Synopsis

integer **ST_AddEdgeModFace**(varchar atopology, integer anode, integer anothernode, geometry acurve);

Beschreibung

Fügt eine Kante hinzu. Falls dabei eine Masche aufgetrennt wird, so wird die ursprüngliche Masche angepasst und eine weitere Masche hinzugefügt.

Note
Wenn möglich, wird die neue Masche auf der linken Seite der neuen Kante erstellt. Dies ist jedoch nicht möglich, wenn die Masche auf der linken Seite die (unbegrenzte) Grundmenge der Maschen darstellt.

Gibt die id der hinzugefügten Kante zurück.

Führt ein entsprechendes Update auf alle verbundenen Kanten und Beziehungen durch.

Wenn irgendwelche Übergabewerte NULL sind, die angegebenen Knoten unbekannt sind (müssen bereits in der `node` Tabelle des Schemas "topology" existieren), `acurve` kein `LINESTRING` ist, oder `anode` und `anothernode` nicht die Anfangs- und Endpunkte von `acurve` sind, dann wird eine Fehlermeldung ausgegeben.

Wenn das Koordinatenreferenzsystem (SRID) der Geometrie `acurve` nicht mit jener der Topologie übereinstimmt, wird eine Fehlermeldung ausgegeben.

Verfügbarkeit: 2.0

This method implements the SQL/MM specification. SQL-MM: Topo-Geo and Topo-Net 3: Routine Details: X.3.13

Beispiele

Siehe auch

ST_RemEdgeModFace

ST_AddEdgeNewFaces

8.6.5 ST_RemEdgeNewFace

ST_RemEdgeNewFace — Entfernt eine Kante. Falls die gelöschte Kante zwei Maschen voneinander getrennt hat, werden die ursprünglichen Maschen gelöscht und durch einer neuen Masche ersetzt.

Synopsis

integer **ST_RemEdgeNewFace**(varchar atopology, integer anedge);

Beschreibung

Entfernt eine Kante. Falls die gelöschte Kante zwei Maschen voneinander getrennt hat, werden die ursprünglichen Maschen gelöscht und durch einer neuen Masche ersetzt.

Gibt die ID der neu erzeugten Masche aus; oder NULL wenn keine Masche erstellt wurde. Es wird keine neue Masche erstellt, wenn die gelöschte Kante defekt oder isoliert ist, oder wenn sie die Begrenzung der Maschengrundmenge darstellt (wodurch die Grundmenge womöglich von der anderen Seite in die Masche fliessen könnte).

Führt ein entsprechendes Update auf alle verbundenen Kanten und Beziehungen durch.

Wenn eine Kante an der Definition einer bestehenden TopoGeometry beteiligt ist, kann sie nicht gelöscht werden. Wenn irgendeine TopoGeometry nur durch eine der beiden Maschen definiert ist (und nicht auch durch die andere), können die beiden Maschen nicht "geheilt" werden.

Wenn irgendwelche Übergabewerte NULL sind, die angegebene Kante unbekannt ist (muss bereits in der `edge` Tabelle des Schemas "topology" existieren), oder die Bezeichnung der Topologie ungültig ist, dann wird eine Fehlermeldung ausgegeben.

Verfügbarkeit: 2.0

This method implements the SQL/MM specification. SQL-MM: Topo-Geo and Topo-Net 3: Routine Details: X.3.14

Beispiele

Siehe auch

ST_RemEdgeModFace

ST_AddEdgeNewFaces

8.6.6 ST_RemEdgeModFace

ST_RemEdgeModFace — Removes an edge, and if the edge separates two faces deletes one face and modifies the other face to cover the space of both.

Synopsis

integer **ST_RemEdgeModFace**(varchar atopology, integer anedge);

Beschreibung

Removes an edge, and if the removed edge separates two faces deletes one face and modifies the other face to cover the space of both. Preferentially keeps the face on the right, to be consistent with ST_AddEdgeModFace. Returns the id of the face which is preserved.

Führt ein entsprechendes Update auf alle verbundenen Kanten und Beziehungen durch.

Wenn eine Kante an der Definition einer bestehenden TopoGeometry beteiligt ist, kann sie nicht gelöscht werden. Wenn irgendeine TopoGeometry nur durch eine der beiden Maschen definiert ist (und nicht auch durch die andere), können die beiden Maschen nicht "geheilt" werden.

Wenn irgendwelche Übergabewerte NULL sind, die angegebene Kante unbekannt ist (muss bereits in der `edge` Tabelle des Schemas "topology" existieren), oder die Bezeichnung der Topologie ungültig ist, dann wird eine Fehlermeldung ausgegeben.

Verfügbarkeit: 2.0

This method implements the SQL/MM specification. SQL-MM: Topo-Geo and Topo-Net 3: Routine Details: X.3.15

Beispiele

Siehe auch

ST_AddEdgeModFace

ST_RemEdgeNewFace

8.6.7 ST_ChangeEdgeGeom

ST_ChangeEdgeGeom — Ändert die geometrische Form einer Kante, ohne sich auf die topologische Struktur auszuwirken.

Synopsis

integer **ST_ChangeEdgeGeom**(varchar atopology, integer anedge, geometry acurve);

Beschreibung

Ändert die geometrische Form der Kante, ohne sich auf topologische Struktur auszuwirken.

If any arguments are null, the given edge does not exist in the `edge` table of the topology schema, the `acurve` is not a `LINESTRING`, or the modification would change the underlying topology then an error is thrown.

Wenn das Koordinatenreferenzsystem (SRID) der Geometrie `acurve` nicht mit jener der Topologie übereinstimmt, wird eine Fehlermeldung ausgegeben.

Wenn die neue `acurve` nicht "simple" ist, wird eine Fehlermeldung ausgegeben.

Wenn beim Verschieben der Kante von der alten auf die neue Position ein Hindernis auftritt, wird eine Fehlermeldung ausgegeben.

Verfügbarkeit: 1.1.0

Erweiterung: 2.0.0 Erzwingung topologischer Konsistenz hinzugefügt

This method implements the SQL/MM specification. SQL-MM: Topo-Geo and Topo-Net 3: Routine Details X.3.6

Beispiele

```
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
```

Siehe auch

ST_AddEdgeModFace

ST_RemEdgeModFace

ST_ModEdgeSplit

8.6.8 ST_ModEdgeSplit

ST_ModEdgeSplit — Trennt eine Kante auf, indem ein neuer Knoten entlang einer bestehenden Kante erstellt wird. Ändert die ursprüngliche Kante und fügt eine neue Kante hinzu.

Synopsis

integer **ST_ModEdgeSplit**(varchar atopology, integer anedge, geometry apoint);

Beschreibung

Trennt eine Kante auf, indem ein neuer Knoten entlang einer bestehenden Kante erstellt wird. Ändert die ursprüngliche Kante und fügt eine neue Kante hinzu. Alle bestehenden Kanten und Beziehungen werden entsprechend aktualisiert. Gibt den Identifikator des neu hinzugefügten Knotens aus.

Availability: 1.1

Änderung: 2.0 - In Vorgängerversionen fälschlicherweise als ST_ModEdgesSplit bezeichnet

This method implements the SQL/MM specification. SQL-MM: Topo-Geo and Topo-Net 3: Routine Details: X.3.9

Beispiele

```
-- Add an edge --
 SELECT topology.AddEdge('ma_topo', ST_GeomFromText('LINESTRING(227592 893910, 227600  ↩
     893910)', 26986) ) As edgeid;

-- edgeid-
3


-- Split the edge  --
SELECT topology.ST_ModEdgeSplit('ma_topo',  3, ST_SetSRID(ST_Point(227594,893910),26986)  ) ↩
     As node_id;
        node_id
-------------------------
7
```

Siehe auch

ST_NewEdgesSplit, ST_ModEdgeHeal, ST_NewEdgeHeal, AddEdge

8.6.9 ST_ModEdgeHeal

ST_ModEdgeHeal — "Heilt" zwei Kanten, indem der verbindende Knoten gelöscht wird, die erste Kante modifiziert und die zweite Kante gelöscht wird. Gibt die ID des gelöschten Knoten zurück.

Synopsis

int **ST_ModEdgeHeal**(varchar atopology, integer anedge, integer anotheredge);

Beschreibung

"Heilt" zwei Kanten, indem der verbindende Knoten gelöscht wird, die erste Kante modifiziert und die zweite Kante gelöscht wird. Gibt die ID des gelöschten Knoten zurück. Aktualisiert alle verbundenen Kanten und Beziehungen dementsprechend.

Verfügbarkeit: 2.0

This method implements the SQL/MM specification. SQL-MM: Topo-Geo and Topo-Net 3: Routine Details: X.3.9

Siehe auch

ST_ModEdgeSplit ST_NewEdgesSplit

8.6.10 ST_NewEdgeHeal

ST_NewEdgeHeal — "Heilt" zwei Kanten, indem der verbindende Knoten und beide Kanten gelöscht werden. Die beiden Kanten werden durch eine Kante ersetzt, welche dieselbe Ausichtung wie die erste Kante hat.

Synopsis

int **ST_NewEdgeHeal**(varchar atopology, integer anedge, integer anotheredge);

Beschreibung

"Heilt" zwei Kanten, indem der verbindende Knoten und beide Kanten gelöscht werden. Die beiden Kanten werden durch eine Kante ersetzt, welche dieselbe Ausichtung wie die erste Kante hat. Gibt die ID der neuen Kante aus. Aktualisiert alle verbundenen Kanten und Beziehungen dementsprechend.

Verfügbarkeit: 2.0

This method implements the SQL/MM specification. SQL-MM: Topo-Geo and Topo-Net 3: Routine Details: X.3.9

Siehe auch

ST_ModEdgeHeal ST_ModEdgeSplit ST_NewEdgesSplit

8.6.11 ST_MoveIsoNode

ST_MoveIsoNode — Verschiebt einen isolierten Knoten in einer Topologie von einer Stelle an eine andere. Falls die neue Geometrie `apoint` bereits als Knoten existiert, wird eine Fehlermeldung ausgegeben. Gibt eine Beschreibung der Verschiebung aus.

Synopsis

text **ST_MoveIsoNode**(varchar atopology, integer anode, geometry apoint);

Beschreibung

Verschiebt einen isolierten Knoten in einer Topologie von einer Stelle an eine andere. Falls die neue Geometrie `apoint` bereits als Knoten existiert, wird eine Fehlermeldung ausgegeben.

If any arguments are null, the `apoint` is not a point, the existing node is not isolated (is a start or end point of an existing edge), new node location intersects an existing edge (even at the end points) or the new location is in a different face (since 3.2.0) then an exception is thrown.

Wenn das Koordinatenreferenzsystem (SRID) der Punktgeometrie nicht mit jener der Topologie übereinstimmt, wird eine Fehlermeldung ausgegeben.

Verfügbarkeit: 2.0.0

Enhanced: 3.2.0 ensures the nod cannot be moved in a different face

This method implements the SQL/MM specification. SQL-MM: Topo-Net Routines: X.3.2

Beispiele

```
-- Add an isolated node with no face  --
SELECT topology.ST_AddIsoNode('ma_topo',  NULL, ST_GeomFromText('POINT(227579 893916)',  ↩
    26986) ) As nodeid;
 nodeid
--------
      7
-- Move the new node --
SELECT topology.ST_MoveIsoNode('ma_topo', 7,  ST_GeomFromText('POINT(227579.5 893916.5)',  ↩
    26986) ) As descrip;
                     descrip
-----------------------------------------------------
Isolated Node 7 moved to location 227579.5,893916.5
```

Siehe auch

ST_AddIsoNode

8.6.12 ST_NewEdgesSplit

ST_NewEdgesSplit — Trennt eine Kante auf, indem ein neuer Knoten entlang einer bestehenden Kante erstellt, die ursprüngliche Kante gelöscht und durch zwei neue Kanten ersetzt wird. Gibt die ID des neu erstellten Knotens aus, der die neuen Kanten verbindet.

Synopsis

integer **ST_NewEdgesSplit**(varchar atopology, integer anedge, geometry apoint);

Beschreibung

Trennt eine Kante mit der Kanten-ID `anedge`auf, indem ein neuer Knoten mit der Punktlage `apoint` entlang der aktuellen Kante erstellt, die ursprüngliche Kante gelöscht und durch zwei neue Kanten ersetzt wird. Gibt die ID des neu erstellten Knotens aus, der die neuen Kanten verbindet. Aktualisiert alle verbundenen Kanten und Beziehungen dementsprechend.

Wenn das Koordinatenreferenzsystem (SRID) der Punktgeometrie nicht mit dem der Topologie übereinstimmt, `apoint` keine Punktgeometrie ist, der Punkt NULL ist, der Punkt bereits als Knoten existiert, die Kante mit einer bestehenden Kante nicht zusammenpasst, oder der Punkt nicht innerhalb der Kante liegt, dann wird eine Fehlermeldung ausgegeben.

Availability: 1.1

This method implements the SQL/MM specification. SQL-MM: Topo-Net Routines: X.3.8

Beispiele

```
-- Add an edge  --
SELECT topology.AddEdge('ma_topo', ST_GeomFromText('LINESTRING(227575 893917,227592 893900) ↩
    ', 26986) ) As edgeid;
-- result-
edgeid
------
        2
-- Split the new edge --
SELECT topology.ST_NewEdgesSplit('ma_topo', 2,  ST_GeomFromText('POINT(227578.5 893913.5)', ↩
     26986) ) As newnodeid;
 newnodeid
---------
       6
```

Siehe auch

ST_ModEdgeSplit ST_ModEdgeHeal ST_NewEdgeHeal AddEdge

8.6.13 ST_RemoveIsoNode

ST_RemoveIsoNode — Löscht einen isolierten Knoten und gibt eine Beschreibung der getroffenen Maßnahmen aus. Falls der Knoten nicht isoliert ist (ist der Anfangs- oder der Endpunkt einer Kante), wird eine Fehlermeldung ausgegeben.

Synopsis

text **ST_RemoveIsoNode**(varchar atopology, integer anode);

Beschreibung

Löscht einen isolierten Knoten und gibt eine Beschreibung der getroffenen Maßnahmen aus. Falls der Knoten nicht isoliert ist (ist der Anfangs- oder der Endpunkt einer Kante), wird eine Fehlermeldung ausgegeben.

Availability: 1.1

This method implements the SQL/MM specification. SQL-MM: Topo-Geo and Topo-Net 3: Routine Details: X+1.3.3

Beispiele

```
-- Remove an isolated node with no face  --
SELECT topology.ST_RemoveIsoNode('ma_topo',  7 ) As result;
         result
-------------------------
 Isolated node 7 removed
```

Siehe auch

ST_AddIsoNode

8.6.14 ST_RemoveIsoEdge

ST_RemoveIsoEdge — Löscht einen isolierten Knoten und gibt eine Beschreibung der getroffenen Maßnahmen aus. Falls der Knoten nicht isoliert ist, wird eine Fehlermeldung ausgegeben.

Synopsis

text **ST_RemoveIsoEdge**(varchar atopology, integer anedge);

Beschreibung

Löscht einen isolierten Knoten und gibt eine Beschreibung der getroffenen Maßnahmen aus. Falls der Knoten nicht isoliert ist, wird eine Fehlermeldung ausgegeben.

Availability: 1.1

This method implements the SQL/MM specification. SQL-MM: Topo-Geo and Topo-Net 3: Routine Details: X+1.3.3

Beispiele

```
-- Remove an isolated node with no face  --
SELECT topology.ST_RemoveIsoNode('ma_topo',  7 ) As result;
         result
-------------------------
 Isolated node 7 removed
```

Siehe auch

ST_AddIsoNode

8.7 Zugriffsfunktionen zur Topologie

8.7.1 GetEdgeByPoint

GetEdgeByPoint — Findet die edge-id einer Kante die einen gegebenen Punkt schneidet.

Synopsis

integer **GetEdgeByPoint**(varchar atopology, geometry apoint, float8 tol1);

Beschreibung

Erfasst die ID einer Kante, die einen Punkt schneidet

Die Funktion gibt eine Ganzzahl (id-edge) für eine Topologie, einen POINT und eine Toleranz aus. Wenn tolerance = 0, dann muss der Punkt die Kante schneiden.

Wenn `apoint` keine Kante schneidet, wird 0 (Null) zurückgegeben.

Wenn eine tolerance > 0 angegeben ist und mehr als eine Kante in diesem Bereich existiert, wird eine Fehlermeldung ausgegeben.

Note
Wenn tolerance = 0, wird von der Funktion ST_Intersects angewendet, ansonsten ST_DWithin.

Wird vom GEOS Modul ausgeführt

Verfügbarkeit: 2.0.0

Beispiele

Die folgenden Beispiele benutzen die Kanten, die wir in AddEdge erzeugt haben

```
SELECT topology.GetEdgeByPoint('ma_topo',geom, 1) As with1mtol, topology.GetEdgeByPoint(' ↩
    ma_topo',geom,0) As withnotol
FROM ST_GeomFromEWKT('SRID=26986;POINT(227622.6 893843)') As geom;
 with1mtol | withnotol
-----------+-----------
         2 |         0
```

```
SELECT topology.GetEdgeByPoint('ma_topo',geom, 1) As nearnode
FROM ST_GeomFromEWKT('SRID=26986;POINT(227591.9 893900.4)') As geom;

-- get error --
ERROR:  Two or more edges found
```

Siehe auch

AddEdge, GetNodeByPoint, GetFaceByPoint

8.7.2 GetFaceByPoint

GetFaceByPoint — Finds face intersecting a given point.

Synopsis

integer **GetFaceByPoint**(varchar atopology, geometry apoint, float8 tol1);

Beschreibung

Finds a face referenced by a Point, with given tolerance.

The function will effectively look for a face intersecting a circle having the point as center and the tolerance as radius.

If no face intersects the given query location, 0 is returned (universal face).

If more than one face intersect the query location an exception is thrown.

Verfügbarkeit: 2.0.0

Enhanced: 3.2.0 more efficient implementation and clearer contract, stops working with invalid topologies.

Beispiele

```
SELECT topology.GetFaceByPoint('ma_topo',geom, 10) As with1mtol, topology.GetFaceByPoint(' ↩
    ma_topo',geom,0) As withnotol
        FROM ST_GeomFromEWKT('POINT(234604.6 899382.0)') As geom;

         with1mtol | withnotol
        -----------+-----------
                         1 |         0
```

```
SELECT topology.GetFaceByPoint('ma_topo',geom, 1) As nearnode
        FROM ST_GeomFromEWKT('POINT(227591.9 893900.4)') As geom;

-- get error --
ERROR:  Two or more faces found
```

Siehe auch

GetFaceContainingPoint, AddFace, GetNodeByPoint, GetEdgeByPoint

8.7.3 GetFaceContainingPoint

GetFaceContainingPoint — Finds the face containing a point.

Synopsis

integer **GetFaceContainingPoint**(text atopology, geometry apoint);

Beschreibung

Returns the id of the face containing a point.

An exception is thrown if the point falls on a face boundary.

Note
The function relies on a valid topology, using edge linking and face labeling.

Availability: 3.2.0

Siehe auch

ST_GetFaceGeometry

8.7.4 GetNodeByPoint

GetNodeByPoint — Findet zu der Lage eines Punktes die node-id eines Knotens.

Synopsis

integer **GetNodeByPoint**(varchar atopology, geometry apoint, float8 tol1);

Beschreibung

Erfasst zu der Lage eines Punktes die ID eines Knotens.

Diese Funktion gibt für eine Topologie, einen POINT und eine Toleranz eine Ganzzahl (id-node) aus. Tolerance = 0 bedeutet exakte Überschneidung, ansonsten wird der Knoten in einem bestimmten Abstand gesucht.

Wenn `apoint` keinen Knoten schneidet, wird 0 (Null) zurückgegeben.

Wenn eine tolerance > 0 angegeben ist und mehr als ein Knoten in dem Bereich des Punktes existiert, wird eine Fehlermeldung ausgegeben.

Note
Wenn tolerance = 0, wird von der Funktion ST_Intersects angewendet, ansonsten ST_DWithin.

Wird vom GEOS Modul ausgeführt

Verfügbarkeit: 2.0.0

Beispiele

Die folgenden Beispiele benutzen die Kanten, die wir in AddEdge erzeugt haben

```
SELECT topology.GetNodeByPoint('ma_topo',geom, 1) As nearnode
 FROM ST_GeomFromEWKT('SRID=26986;POINT(227591.9 893900.4)') As geom;
  nearnode
----------
        2
```

```
SELECT topology.GetNodeByPoint('ma_topo',geom, 1000) As too_much_tolerance
 FROM ST_GeomFromEWKT('SRID=26986;POINT(227591.9 893900.4)') As geom;

 ----get error--
 ERROR:  Two or more nodes found
```

Siehe auch

AddEdge, GetEdgeByPoint, GetFaceByPoint

8.7.5 GetTopologyID

GetTopologyID — Gibt für den Namen einer Topologie die ID der Topologie in der Tabelle "topology.topology" aus.

Synopsis

integer **GetTopologyID**(varchar toponame);

Beschreibung

Gibt für den Namen einer Topologie, die ID der Topologie in der Tabelle "topology.topology" aus.

Availability: 1.1

Beispiele

```
SELECT topology.GetTopologyID('ma_topo') As topo_id;
 topo_id
---------
       1
```

Siehe auch

CreateTopology, DropTopology, GetTopologyName, GetTopologySRID

8.7.6 GetTopologySRID

GetTopologySRID — Gibt für den Namen einer Topologie, die SRID der Topologie in der Tabelle "topology.topology" aus.

Synopsis

integer **GetTopologyID**(varchar toponame);

Beschreibung

Gibt für den Namen einer Topologie, die ID des Koordinatenreferenzsystems in der Tabelle "topology.topology" aus.

Verfügbarkeit: 2.0.0

Beispiele

```
SELECT topology.GetTopologySRID('ma_topo') As SRID;
 SRID
-------
  4326
```

Siehe auch

CreateTopology, DropTopology, GetTopologyName, GetTopologyID

8.7.7 GetTopologyName

GetTopologyName — Gibt für die ID der Topologie, den Namen der Topologie (Schema) zurück.

Synopsis

varchar **GetTopologyName**(integer topology_id);

Beschreibung

Gibt für die ID einer Topologie, den Namen (Schema) der Topologie in der Tabelle "topology.topology" aus.

Availability: 1.1

Beispiele

```
SELECT topology.GetTopologyName(1) As topo_name;
 topo_name
-----------
 ma_topo
```

Siehe auch

CreateTopology, DropTopology, GetTopologyID, GetTopologySRID

8.7.8 ST_GetFaceEdges

ST_GetFaceEdges — Gibt die Kanten, die `aface` begrenzen, sortiert aus.

Synopsis

getfaceedges_returntype **ST_GetFaceEdges**(varchar atopology, integer aface);

Beschreibung

Gibt die Kanten, die `aface` begrenzen, sortiert aus. Jede Ausgabe besteht aus einer Sequenz und einer "edgeid". Die Sequenzzahlen beginnen mit dem Wert 1.

Die Aufzählung der Kanten des Rings beginnt mit der Kante mit dem niedrigsten Identifikator. Die Reihenfolge der Kanten folgt der Drei-Finger-Regel (die begrenzte Masche liegt links von den gerichteten Kanten).

Verfügbarkeit: 2.0

This method implements the SQL/MM specification. SQL-MM 3 Topo-Geo and Topo-Net 3: Routine Details: X.3.5

Beispiele

```
-- Returns the edges bounding face 1
SELECT (topology.ST_GetFaceEdges('tt', 1)).*;
-- result --
 sequence | edge
----------+------
        1 |   -4
        2 |    5
        3 |    7
        4 |   -6
        5 |    1
        6 |    2
        7 |    3
(7 rows)
```

```
-- Returns the sequence, edge id
-- and geometry of the edges that bound face 1
-- If you just need geom and seq, can use ST_GetFaceGeometry
SELECT t.seq, t.edge, geom
FROM topology.ST_GetFaceEdges('tt',1) As t(seq,edge)
        INNER JOIN tt.edge AS e ON abs(t.edge) = e.edge_id;
```

Siehe auch

GetRingEdges, AddFace, ST_GetFaceGeometry

8.7.9 ST_GetFaceGeometry

ST_GetFaceGeometry — Gibt für eine Topologie und eine bestimmte Maschen-ID das Polygon zurück.

Synopsis

geometry **ST_GetFaceGeometry**(varchar atopology, integer aface);

Beschreibung

Gibt für eine Topologie und eine bestimmte Maschen-ID das Polygon zurück. Erstellt das Polygon aus den Kanten, die die Masche aufbauen.

Availability: 1.1

This method implements the SQL/MM specification. SQL-MM 3 Topo-Geo and Topo-Net 3: Routine Details: X.3.16

Beispiele

```
-- Returns the wkt of the polygon added with AddFace
SELECT ST_AsText(topology.ST_GetFaceGeometry('ma_topo', 1)) As facegeomwkt;
-- result --
               facegeomwkt

------------------------------------------------------------------------------
 POLYGON((234776.9 899563.7,234896.5 899456.7,234914 899436.4,234946.6 899356.9,
234872.5 899328.7,234891 899285.4,234992.5 899145,234890.6 899069,
234755.2 899255.4,234612.7 899379.4,234776.9 899563.7))
```

Siehe auch

AddFace

8.7.10 GetRingEdges

GetRingEdges — Gibt eine sortierte Liste von mit Vorzeichen versehenen Identifikatoren der Kanten zurück, die angetroffen werden, wenn man an der Seite der gegebenen Kante entlangwandert.

Synopsis

getfaceedges_returntype **GetRingEdges**(varchar atopology, integer aring, integer max_edges=null);

Beschreibung

Gibt eine sortierte Liste von mit Vorzeichen versehenen Identifikatoren der Kanten zurück, die angetroffen werden, wenn man an der Seite der gegebenen Kante entlangwandert. Jede Ausgabe besteht aus einer Sequenz und einer mit einem Vorzeichen versehenen ID der Kante. Die Sequenz beginnt mit dem Wert 1.

Wenn Sie eine positive ID für die Kante übergeben, beginnt der Weg auf der linken Seite der entsprechenden Kante und folgt der Ausrichtung der Kante. Wenn Sie eine negative ID für die Kante übergeben, beginnt der Weg auf der rechten Seite der Kante und verläuft rückwärts.

Wenn `max_edges` nicht NULL ist, so beschränkt dieser Parameter die Anzahl der von dieser Funktion ausgegebenen Datensätze. Ist als Sicherheitsparameter für den Umgang mit möglicherweise invaliden Topologien gedacht.

Note
Diese Funktion verwendet Metadaten um die Kanten eines Ringes zu verbinden.

Verfügbarkeit: 2.0.0

Siehe auch

ST_GetFaceEdges, GetNodeEdges

8.7.11 GetNodeEdges

GetNodeEdges — Gibt für einen Knoten die sortierte Menge der einfallenden Kanten aus.

Synopsis

getfaceedges_returntype **GetNodeEdges**(varchar atopology, integer anode);

Beschreibung

Gibt für einen Knoten die sortierte Menge der einfallenden Kanten aus. Jede Ausgabe besteht aus einer Sequenz und einer mit Vorzeichen versehenen ID für die Kante. Die Sequenz beginnt mit dem Wert 1. Eine Kante mit positiver ID beginnt an dem gegebenen Knoten. Eine negative Kante endet in dem gegebenen Knoten. Geschlossene Kanten kommen zweimal vor (mit beiden Vorzeichen). Die Sortierung geschieht von Norden ausgehend im Uhrzeigersinn.

Note
Diese Funktion errechnet die Reihenfolge, anstatt sie aus den Metadaten abzuleiten und kann daher verwendet werden, um die Kanten eines Ringes zu verbinden.

Verfügbarkeit: 2.0

Siehe auch

getfaceedges_returntype, GetRingEdges, ST_Azimuth

8.8 Topologie Verarbeitung

8.8.1 Polygonize

Polygonize — Findet und registriert alle Maschen, die durch die Kanten der Topologie festgelegt sind.

Synopsis

text **Polygonize**(varchar toponame);

Beschreibung

Registriert alle Maschen, die aus den Kanten der topologischen Elementarstrukturen erstellt werden können

Von der Zieltopologie wird angenommen, dass sie keine sich selbst überschneidenden Kanten enthält.

Note
Da bereits bekannte Maschen erkannt werden, kann Polygonize gefahrlos mehrere Male auf die selbe Topologie angewendet werden.

Note
Von dieser Funktion werden die Attribute "next_left_edge" und "next_right_edge" der Tabelle "edge" weder verwendet noch gesetzt.

Verfügbarkeit: 2.0.0

Siehe auch

AddFace, ST_Polygonize

8.8.2 AddNode

AddNode — Fügt einen Knotenpunkt zu der Tabelle "node" in dem vorgegebenen topologischen Schema hinzu und gibt die "nodeid" des neuen Knotens aus. Falls der Punkt bereits als Knoten existiert, wird die vorhandene nodeid zurückgegben.

Synopsis

integer **AddNode**(varchar toponame, geometry apoint, boolean allowEdgeSplitting=false, boolean computeContainingFace=false);

Beschreibung

Fügt einen Knotenpunkt zu der Tabelle "node" in dem vorgegebenen topologischen Schema hinzu. Die Funktion AddEdge fügt die Anfangs- und Endpunkte einer Kante automatisch hinzu, wenn sie aufgerufen wird. Deshalb ist es nicht notwendig die Knoten einer Kante explizit anzufügen.

Falls eine Kante aufgefunden wird, die den Knoten kreuzt, dann wird entweder eine Fehlermeldung ausgegeben, oder die Kante aufgetrennt. Dieses Verhalten hängt vom Wert des Parameters `allowEdgeSplitting` ab.

Wenn `computeContainingFace` TRUE ist, dann wird für einen neu hinzugefügten Knoten die berichtigte Begrenzung der Masche berechnet.

Note
Wenn die Geometrie `apoint` bereits als Knoten existiert, dann wird der Knoten nicht hinzugefügt und der bestehende Knoten ausgegeben.

Verfügbarkeit: 2.0.0

Beispiele

```
SELECT topology.AddNode('ma_topo', ST_GeomFromText('POINT(227641.6 893816.5)', 26986) ) As ↩
    nodeid;
-- result --
nodeid
--------
 4
```

Siehe auch

AddEdge, CreateTopology

8.8.3 AddEdge

AddEdge — Fügt die Kante eines Linienzugs in der Tabelle "edge", und die zugehörigen Anfangs- und Endpunkte in die Knotenpunkttabelle, des jeweiligen topologischen Schemas ein. Dabei wird die übergebene Linienzuggeometrie verwendet und die edgeid der neuen (oder bestehenden) Kante ausgegeben.

Synopsis

integer **AddEdge**(varchar toponame, geometry aline);

Beschreibung

Fügt eine Kante in der Tabelle "edge", und die zugehörigen Knoten in die Tabelle "node", des jeweiligen Schemas `toponame` ein. Dabei wird die übergebene Linienzuggeometrie verwendet und die edgeid des neuen oder des bestehenden Datensatzes ausgegeben. Die neu hinzugefügte Kante hat auf beiden Seiten die Masche für die Grundmenge/"Universum" und verweist auf sich selbst.

Note
Wenn die Geometrie `aline` eine bestehende Kante kreuzt, überlagert, beinhaltet oder in ihr enthalten ist, dann wird eine Fehlermeldung ausgegeben und die Kante wird nicht hinzugefügt.

Note
Die Geometrie von `aline` muss dieselbe `SRID` aufweisen wie die Topologie, ansonsten wird die Fehlermeldung "invalid spatial reference sys error" ausgegeben.

Wird vom GEOS Modul ausgeführt

Verfügbarkeit: 2.0.0

Beispiele

```
SELECT topology.AddEdge('ma_topo', ST_GeomFromText('LINESTRING(227575.8 893917.2,227591.9 ↩
    893900.4)', 26986) ) As edgeid;
-- result-
edgeid
--------
 1

SELECT topology.AddEdge('ma_topo', ST_GeomFromText('LINESTRING(227591.9 893900.4,227622.6 ↩
    893844.2,227641.6 893816.5,
 227704.5 893778.5)', 26986) ) As edgeid;
-- result --
edgeid
--------
 2

 SELECT topology.AddEdge('ma_topo', ST_GeomFromText('LINESTRING(227591.2 893900, 227591.9 ↩
     893900.4,
  227704.5 893778.5)', 26986) ) As edgeid;
 -- gives error --
 ERROR:  Edge intersects (not on endpoints) with existing edge 1
```

Siehe auch

TopoGeo_AddLineString, CreateTopology, Section 4.5

8.8.4 AddFace

AddFace — Registriert die Elementarstruktur einer Masche in einer Topologie und gibt den Identifikator der Masche aus.

Synopsis

integer **AddFace**(varchar toponame, geometry apolygon, boolean force_new=false);

Beschreibung

Registriert die Elementarstruktur einer Masche in einer Topologie und gibt den Identifikator der Masche aus.

Bei einer neu hinzugefügten Masche werden die Kanten die ihre Begrenzung bilden und jene die innerhalb der Masche liegen aktualisiert, damit diese die richtigen Werte in den Attributen "left_face" und "right_face" aufweisen. Isolierte Knoten innerhalb der Masche werden ebenfalls aktualisiert, damit das Attribut "containing_face" die richtigen Werte aufweist.

Note
Von dieser Funktion werden die Attribute "next_left_edge" und "next_right_edge" der Tabelle "edge" weder verwendet noch gesetzt.

Es wird angenommen, dass die Zieltopologie valide ist (keine sich selbst überschneidenden Kanten enthält). Eine Fehlermeldung wird ausgegeben, wenn: Die Begrenzung des Polygons nicht vollständig durch bestehende Kanten festgelegt ist oder das Polygon eine bereits bestehende Masche überlappt.

Falls die Geometrie `apolygon` bereits als Masche existiert, dann: wenn `force_new` FALSE (der Standardwert) ist, dann wird die bestehende Masche zurückgegeben; wenn `force_new` TRUE ist, dann wird der neu registrierten Masche eine neue ID zugewiesen.

Note
Wenn eine bestehende Masche neu registriert wird (force_new=true), werden keine Maßnahmen durchgeführt um hängende Verweise auf eine bestehende Masche in den Tabellen "edge", "node" und "relation" zu bereinigen, noch wird der das Attribut MBR der bestehenden Masche aktualisert. Es ist die Aufgabe des Aufrufers sich um dies zu kümmern.

Note
Die Geometrie von `apolygon` muss dieselbe `SRID` aufweisen wie für die Topologie festgelegt, ansonsten wird die Fehlermeldung "invalid spatial reference sys error" ausgegeben.

Verfügbarkeit: 2.0.0

Beispiele

```
-- first add the edges we use generate_series as an iterator (the below
-- will only work for polygons with < 10000 points because of our max in gs)
SELECT topology.AddEdge('ma_topo', ST_MakeLine(ST_PointN(geom,i), ST_PointN(geom, i + 1) )) ↩
     As edgeid
    FROM (SELECT  ST_NPoints(geom) AS npt, geom
            FROM
                (SELECT ST_Boundary(ST_GeomFromText('POLYGON((234896.5 899456.7,234914  ↩
                    899436.4,234946.6 899356.9,234872.5 899328.7,
                234891 899285.4,234992.5 899145, 234890.6 899069,234755.2 899255.4,
                234612.7 899379.4,234776.9 899563.7,234896.5 899456.7))', 26986) )  As geom
            )  As geoms) As facen CROSS JOIN generate_series(1,10000) As i
         WHERE i < npt;
-- result --
 edgeid
--------
      3
      4
      5
      6
      7
      8
      9
     10
     11
     12
(10 rows)
-- then add the face -

SELECT topology.AddFace('ma_topo',
    ST_GeomFromText('POLYGON((234896.5 899456.7,234914 899436.4,234946.6 899356.9,234872.5  ↩
        899328.7,
    234891 899285.4,234992.5 899145, 234890.6 899069,234755.2 899255.4,
    234612.7 899379.4,234776.9 899563.7,234896.5 899456.7))', 26986) ) As faceid;
-- result --
faceid
--------
 1
```

Siehe auch

AddEdge, CreateTopology, Section 4.5

8.8.5 ST_Simplify

ST_Simplify — Gibt für eine TopoGeometry eine "vereinfachte" geometrische Version zurück. Verwendet den Douglas-Peucker Algorithmus.

Synopsis

geometry **ST_Simplify**(TopoGeometry tg, float8 tolerance);

Beschreibung

Gibt für eine TopoGeometry eine "vereinfachte" geometrische Version zurück. Wendet den Douglas-Peucker Algorithmus auf jede Kante an.

Note
Es kann vorkommen, dass die zurückgegebene Geometrie weder "simple" noch valide ist.
Das Auftrennen der Kanten kann helfen, die Simplizität/Validität zu erhalten.

Wird vom GEOS Modul ausgeführt

Verfügbarkeit: 2.1.0

Siehe auch

Geometrie ST_Simplify, ST_IsSimple, ST_IsValid, ST_ModEdgeSplit

8.8.6 RemoveUnusedPrimitives

RemoveUnusedPrimitives — Removes topology primitives which not needed to define existing TopoGeometry objects.

Synopsis

int **RemoveUnusedPrimitives**(text topology_name, geometry bbox);

Beschreibung

Finds all primitives (nodes, edges, faces) that are not strictly needed to represent existing TopoGeometry objects and removes them, maintaining topology validity (edge linking, face labeling) and TopoGeometry space occupation.

No new primitive identifiers are created, but rather existing primitives are expanded to include merged faces (upon removing edges) or healed edges (upon removing nodes).

Availability: 3.3.0

Siehe auch

ST_ModEdgeHeal, ST_RemEdgeModFace

8.9 TopoGeometry Konstruktoren

8.9.1 CreateTopoGeom

CreateTopoGeom — Erzeugt ein neues topologisch geometrisches Objekt aus einem Feld mit topologischen Elementen - 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);
topogeometry **CreateTopoGeom**(varchar toponame, integer tg_type, integer layer_id);

Beschreibung

Erstellt ein TopoGeometry Objekt für den Layer, der über die `layer_id` angegeben wird, und registriert es in der Tabelle "relation" in dem Schema `toponame`.

`tg_type` ist eine Ganzzahl: 1:[multi]point (punktförmig), 2:[multi]line (geradlinig), 3:[multi]poly (flächenhaft), 4:collection. `layer_id` ist der Identifikator des Layers in der Tabelle "topology.layer".

Punktförmige Layer werden aus Knoten gebildet, linienförmige Layer aus Kanten, flächige Layer aus Maschen und Kollektionen können aus einer Mischung von Knoten, Kanten und Maschen gebildet werden.

Wird das Feld mit den Komponenten weggelassen, wird ein leeres TopoGeometrie Objekt erstellt.

Availability: 1.1

Beispiele: Aus bestehenden Kanten bilden

Erstellt eine TopoGeometry im Schema "ri_topo" für den Layer 2 (ri_roads), vom Datentyp (2) LINE, für die erste Kante (die wir unter `ST_CreateTopoGeo` geladen haben).

```
INSERT INTO ri.ri_roads(road_name, topo) VALUES('Unknown', topology.CreateTopoGeom('ri_topo ↩
    ',2,2,'{{1,2}}'::topology.topoelementarray);
```

Beispiele: Eine flächige Geometrie in die vermutete TopoGeometry konvertieren

Angenommen wir wollen eine Geometrie aus einer Kollektion von Maschen bilden. Wir haben zum Beispiel die Tabelle "blockgroups" und wollen die TopoGeometry von jeder "blockgroup" wissen. Falls unsere Daten perfekt ausgerichtet sind, können wir folgendes ausführen:

```
-- create our topo geometry column --
SELECT topology.AddTopoGeometryColumn(
        'topo_boston',
        'boston', 'blockgroups', 'topo', 'POLYGON');

-- addtopgeometrycolumn --
1

-- update our column assuming
-- everything is perfectly aligned with our edges
UPDATE boston.blockgroups AS bg
        SET topo = topology.CreateTopoGeom('topo_boston'
        ,3,1
        , foo.bfaces)
FROM (SELECT b.gid,  topology.TopoElementArray_Agg(ARRAY[f.face_id,3]) As bfaces
        FROM boston.blockgroups As b
```

```
            INNER JOIN topo_boston.face As f ON b.geom && f.mbr
        WHERE ST_Covers(b.geom, topology.ST_GetFaceGeometry('topo_boston', f.face_id))
            GROUP BY b.gid) As foo
WHERE foo.gid = bg.gid;
```

```
--the world is rarely perfect allow for some error
--count the face if 50% of it falls
-- within what we think is our blockgroup boundary
UPDATE boston.blockgroups AS bg
        SET topo = topology.CreateTopoGeom('topo_boston'
        ,3,1
        , foo.bfaces)
FROM (SELECT b.gid,  topology.TopoElementArray_Agg(ARRAY[f.face_id,3]) As bfaces
        FROM boston.blockgroups As b
            INNER JOIN topo_boston.face As f ON b.geom && f.mbr
        WHERE ST_Covers(b.geom, topology.ST_GetFaceGeometry('topo_boston', f.face_id))
        OR
 (  ST_Intersects(b.geom, topology.ST_GetFaceGeometry('topo_boston', f.face_id))
            AND ST_Area(ST_Intersection(b.geom, topology.ST_GetFaceGeometry('topo_boston',  ↩
                f.face_id) ) ) >
                ST_Area(topology.ST_GetFaceGeometry('topo_boston', f.face_id))*0.5
                )
            GROUP BY b.gid) As foo
WHERE foo.gid = bg.gid;

-- and if we wanted to convert our topogeometry back
-- to a denormalized geometry aligned with our faces and edges
-- cast the topo to a geometry
-- The really cool thing is my new geometries
-- are now aligned with my tiger street centerlines
UPDATE boston.blockgroups SET new_geom = topo::geometry;
```

Siehe auch

AddTopoGeometryColumn, toTopoGeom ST_CreateTopoGeo, ST_GetFaceGeometry, TopoElementArray, TopoElementArray_Agg

8.9.2 toTopoGeom

toTopoGeom — Wandelt eine einfache Geometrie in eine TopoGeometry um.

Synopsis

topogeometry **toTopoGeom**(geometry geom, varchar toponame, integer layer_id, float8 tolerance);
topogeometry **toTopoGeom**(geometry geom, topogeometry topogeom, float8 tolerance);

Beschreibung

Wandelt eine einfache Geometrie in eine TopoGeometry um.

Die topologischen Elementarstrukturen, die benötigt werden um die Übergabegeometrie darzustellen, werden der zugrunde liegenden Topologie hinzugefügt. Dabei können bestehende Strukturen aufgetrennt werden, die dann mit der ausgegebenen TopoGeometry in der Tabelle `relation` zusammengeführt werden.

Bestehende Objekte einer TopoGeometry (mit der möglichen Ausnahme von `topogeom`, falls angegeben) behalten ihre geometrische Gestalt.

Wenn `tolerance` angegeben ist, wird diese zum Fangen der Eingabegeometrie an bestehenden Elementarstrukturen verwendet.

Bei der ersten Form wird eine neue TopoGeometry für den Layer (`layer_id`) einer Topologie (`toponame`) erstellt.

Bei der zweiten Form werden die aus der Konvertierung enstehenden Elementarstrukturen zu der bestehenden TopoGeometry (`topogeom`) hinzugefügt. Dabei wird möglicherweise zusätzlicher Raum aufgefüllt, um die endgültige geometrische Gestalt zu erreichen. Um die alte geometrische Gestalt zur Gänze durch eine neue zu ersetzen, siehe clearTopoGeom.

Verfügbarkeit: 2.0

Erweiterung: 2.1.0 die Version, welche eine bestehende TopoGeometry entgegennimmt, wurde hinzugefügt.

Beispiele

Dies ist ein in sich selbst vollkommen abgeschlossener Arbeitsablauf

```
-- do this if you don't have a topology setup already
-- creates topology not allowing any tolerance
SELECT topology.CreateTopology('topo_boston_test', 2249);
-- create a new table
CREATE TABLE nei_topo(gid serial primary key, nei varchar(30));
--add a topogeometry column to it
SELECT topology.AddTopoGeometryColumn('topo_boston_test', 'public', 'nei_topo', 'topo', ' ↩
    MULTIPOLYGON') As new_layer_id;
new_layer_id
-----------
1

--use new layer id in populating the new topogeometry column
-- we add the topogeoms to the new layer with 0 tolerance
INSERT INTO nei_topo(nei, topo)
SELECT nei,  topology.toTopoGeom(geom, 'topo_boston_test', 1)
FROM neighborhoods
WHERE gid BETWEEN 1 and 15;

--use to verify what has happened --
SELECT * FROM
    topology.TopologySummary('topo_boston_test');

-- summary--
Topology topo_boston_test (5), SRID 2249, precision 0
61 nodes, 87 edges, 35 faces, 15 topogeoms in 1 layers
Layer 1, type Polygonal (3), 15 topogeoms
 Deploy: public.nei_topo.topo
```

```
-- Shrink all TopoGeometry polygons by 10 meters
UPDATE nei_topo SET topo = ST_Buffer(clearTopoGeom(topo), -10);

-- Get the no-one-lands left by the above operation
-- I think GRASS calls this "polygon0 layer"
SELECT ST_GetFaceGeometry('topo_boston_test', f.face_id)
  FROM topo_boston_test.face f
  WHERE f.face_id
> 0 -- don't consider the universe face
  AND NOT EXISTS ( -- check that no TopoGeometry references the face
    SELECT * FROM topo_boston_test.relation
    WHERE layer_id = 1 AND element_id = f.face_id
  );
```

Siehe auch

CreateTopology, AddTopoGeometryColumn, CreateTopoGeom, TopologySummary, clearTopoGeom

8.9.3 TopoElementArray_Agg

TopoElementArray_Agg — Gibt für eine Menge an element_id, type Feldern (topoelements) ein `topoelementarray` zurück.

Synopsis

topoelementarray **TopoElementArray_Agg**(topoelement set tefield);

Beschreibung

Verwendet um ein TopoElementArray aus einer Menge an TopoElement zu erstellen.

Verfügbarkeit: 2.0.0

Beispiele

```
SELECT topology.TopoElementArray_Agg(ARRAY[e,t]) As tea
  FROM generate_series(1,3) As e CROSS JOIN generate_series(1,4) As t;
  tea
---------------------------------------------------------------------------
{{1,1},{1,2},{1,3},{1,4},{2,1},{2,2},{2,3},{2,4},{3,1},{3,2},{3,3},{3,4}}
```

Siehe auch

TopoElement, TopoElementArray

8.9.4 TopoElement

TopoElement — Converts a topogeometry to a topoelement.

Synopsis

topoelement **TopoElement**(topogeometry topo);

Beschreibung

Converts a TopoGeometry to a TopoElement.

Availability: 3.4.0

Beispiele

Dies ist ein in sich selbst vollkommen abgeschlossener Arbeitsablauf

```
-- do this if you don't have a topology setup already
-- Creates topology not allowing any tolerance
SELECT TopoElement(topo)
FROM neighborhoods;
```

```
-- using as cast
SELECT topology.TopoElementArray_Agg(topo::topoelement)
FROM neighborhoods
GROUP BY city;
```

Siehe auch

TopoElementArray_Agg, TopoGeometry, TopoElement

8.10 TopoGeometry Editoren

8.10.1 clearTopoGeom

clearTopoGeom — Löscht den Inhalt einer TopoGeometry.

Synopsis

topogeometry **clearTopoGeom**(topogeometry topogeom);

Beschreibung

Löscht den Inhalt einer TopoGeometry und wandelt sie in eine leere um. Am nützlichsten in Verbindung mit toTopoGeom, um die geometrische Gestalt bestehende Objekte und alle abhängigen Objekte in höheren hierarchischen Ebenen zu ersetzen.

Verfügbarkeit: 2.1

Beispiele

```
-- Shrink all TopoGeometry polygons by 10 meters
UPDATE nei_topo SET topo = ST_Buffer(clearTopoGeom(topo), -10);
```

Siehe auch

toTopoGeom

8.10.2 TopoGeom_addElement

TopoGeom_addElement — Fügt ein Element zu der Definition einer TopoGeometry hinzu.

Synopsis

topogeometry **TopoGeom_addElement**(topogeometry tg, topoelement el);

Beschreibung

Fügt ein TopoElement zur Definition eines TopoGeometry-Objekts hinzu. Wenn das Element bereits Teil der Definition ist, führt dies zu keinem Fehler.

Verfügbarkeit: 2.3

Beispiele

```
-- Add edge 5 to TopoGeometry tg
UPDATE mylayer SET tg = TopoGeom_addElement(tg, '{5,2}');
```

Siehe auch

TopoGeom_remElement, CreateTopoGeom

8.10.3 TopoGeom_remElement

TopoGeom_remElement — Entfernt ein Element aus der Definition einer TopoGeometry.

Synopsis

topogeometry **TopoGeom_remElement**(topogeometry tg, topoelement el);

Beschreibung

Entfernt ein TopoElement aus der Ausgestaltung des TopoGeometry Objekts.

Verfügbarkeit: 2.3

Beispiele

```
-- Remove face 43 from TopoGeometry tg
UPDATE mylayer SET tg = TopoGeom_remElement(tg, '{43,3}');
```

Siehe auch

TopoGeom_addElement, CreateTopoGeom

8.10.4 TopoGeom_addTopoGeom

TopoGeom_addTopoGeom — Adds element of a TopoGeometry to the definition of another TopoGeometry.

Synopsis

topogeometry **TopoGeom_addTopoGeom**(topogeometry tgt, topogeometry src);

Beschreibung

Adds the elements of a TopoGeometry to the definition of another TopoGeometry, possibly changing its cached type (type attribute) to a collection, if needed to hold all elements in the source object.

The two TopoGeometry objects need be defined against the *same* topology and, if hierarchically defined, need be composed by elements of the same child layer.

Availability: 3.2

Beispiele

```
-- Set an "overall" TopoGeometry value to be composed by all
-- elements of specific TopoGeometry values
UPDATE mylayer SET tg_overall = TopoGeom_addTopogeom(
    TopoGeom_addTopoGeom(
        clearTopoGeom(tg_overall),
        tg_specific1
    ),
    tg_specific2
);
```

Siehe auch

TopoGeom_addElement, clearTopoGeom, CreateTopoGeom

8.10.5 toTopoGeom

toTopoGeom — Fügt eine Geometrie zu einer bestehenden TopoGeometry hinzu.

Beschreibung

Siehe toTopoGeom.

8.11 TopoGeometry Accessors

8.11.1 GetTopoGeomElementArray

GetTopoGeomElementArray — Gibt ein `topoelementarray` (ein Feld von topoelements) zurück, das die topologischen Elemente und den Datentyp der gegebenen TopoGeometry (die Elementarstrukturen) enthält.

Synopsis

topoelementarray **GetTopoGeomElementArray**(varchar toponame, integer layer_id, integer tg_id);

topoelementarray **GetTopoGeomElementArray**(topogeometry tg);

Beschreibung

Gibt ein TopoElementArray zurück, das die topologischen Elemente und den Datentyp der gegebenen TopoGeometry (die Elementarstrukturen) enthält. Dies ist ähnlich dem GetTopoGeomElements, ausser dass die Elemente als Feld statt als Datensatz ausgegeben werden.

tg_id steht für die ID des TopoGeometry Objekts der Topologie eines Layers, der durch die `layer_id` der Tabelle "topology.layer" angegeben wird.

Availability: 1.1

Beispiele

Siehe auch

GetTopoGeomElements, TopoElementArray

8.11.2 GetTopoGeomElements

GetTopoGeomElements — Gibt für eine TopoGeometry (Elementarstrukturen) einen Satz an `topoelement` Objekten zurück, welche die topologische element_id und den element_type beinhalten.

Synopsis

setof topoelement **GetTopoGeomElements**(varchar toponame, integer layer_id, integer tg_id);

setof topoelement **GetTopoGeomElements**(topogeometry tg);

Beschreibung

Returns a set of element_id,element_type (topoelements) corresponding to primitive topology elements TopoElement (1: nodes, 2: edges, 3: faces) that a given topogeometry object in `toponame` schema is composed of.

tg_id steht für die ID des TopoGeometry Objekts der Topologie eines Layers, der durch die `layer_id` der Tabelle "topology.layer" angegeben wird.

Verfügbarkeit: 2.0.0

Beispiele

Siehe auch

GetTopoGeomElementArray, TopoElement, TopoGeom_addElement, TopoGeom_remElement

8.11.3 ST_SRID

ST_SRID — Returns the spatial reference identifier for a topogeometry.

Synopsis

integer **ST_SRID**(topogeometry tg);

Beschreibung

Returns the spatial reference identifier for the ST_Geometry as defined in spatial_ref_sys table. Section 4.5

Note
spatial_ref_sys table is a table that catalogs all spatial reference systems known to PostGIS and is used for transformations from one spatial reference system to another. So verifying you have the right spatial reference system identifier is important if you plan to ever transform your geometries.

Availability: 3.2.0

This method implements the SQL/MM specification. SQL-MM 3: 14.1.5

Beispiele

```
SELECT ST_SRID(ST_GeomFromText('POINT(-71.1043 42.315)',4326));
                --result
                4326
```

Siehe auch

Section 4.5, ST_SetSRID, ST_Transform, ST_SRID

8.12 TopoGeometry Ausgabe

8.12.1 AsGML

AsGML — Gibt die GML-Darstellung einer TopoGeometry zurück.

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);

Beschreibung

Gibt die GML-Darstellung einer TopoGeometry im GML3-Format aus. Wenn kein `nsprefix_in` angegeben ist, dann wird `gml` verwendet. Übergeben sie eine leere Zeichenfolge für "nsprefix" um keinen bestimmten Namensraum festzulegen. Wenn die Parameter "precision" (Standardwert: 15) und "options" (Standardwert: 1) angegeben sind, werden diese unangetastet an den zugrunde liegenden Aufruf von ST_AsGML übergeben.

Wenn der Parameter `visitedTable` angegeben ist, dann wird dieser verwendet um die bereits besuchten Knoten und Kanten über Querverweise (xlink:xref) zu verfolgen, anstatt Definitionen zu vervielfältigen. Die Tabelle muss (zumindest) zwei Integerfelder enthalten: 'element_type' und 'element_id'. Für den Aufruf muss der Anwender sowohl Lese- als auch Schreibrechte auf die Tabelle besitzen. Um die maximale Rechenleistung zu erreichen, sollte ein Index für die Attribute `element_type` und `element_id` - in dieser Reihenfolge - festgelegt werden. Dieser Index wird automatisch erstellt, wenn auf die Attribute ein Unique Constraint gelegt wird. Beispiel:

```
CREATE TABLE visited (
  element_type integer, element_id integer,
  unique(element_type, element_id)
);
```

Wird der Parameter `idprefix` angegeben, so wird dieser den Identifikatoren der Tags von Kanten und Knoten vorangestellt.

Wird der Parameter `gmlver` angegeben, so wird dieser and das zugrunde liegende ST_AsGML übergeben. Standardmäßig wird 3 angenommen.

Verfügbarkeit: 2.0.0

Beispiele

Hier wird die TopoGeometry verwendet, die wir unter CreateTopoGeom erstellt haben

```
SELECT topology.AsGML(topo) As rdgml
  FROM ri.roads
  WHERE road_name = 'Unknown';
```

```
-- rdgml--
<gml:TopoCurve>
    <gml:directedEdge>
        <gml:Edge gml:id="E1">
            <gml:directedNode orientation="-">
                <gml:Node gml:id="N1"/>
            </gml:directedNode>
            <gml:directedNode
></gml: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
>
```

Selbes Beispiel wie das Vorige, aber ohne Namensraum

```
SELECT topology.AsGML(topo,'') As rdgml
  FROM ri.roads
  WHERE road_name = 'Unknown';

-- rdgml--
<TopoCurve>
    <directedEdge>
        <Edge id="E1">
            <directedNode orientation="-">
                <Node id="N1"/>
            </directedNode>
            <directedNode
></directedNode>
            <curveProperty>
                <Curve srsName="urn:ogc:def:crs:EPSG::3438">
                    <segments>
                        <LineStringSegment>
                            <posList srsDimension="2"
>384744 236928 384750 236923 384769 236911 384799 236895 384811 236890
                            384833 236884 384844 236882 384866 236881 384879 236883 384954  ↩
```

```
                                236898 385087 236932 385117 236938
                               385167 236938 385203 236941 385224 236946 385233 236950 385241  ↩
                                   236956 385254 236971
                               385260 236979 385268 236999 385273 237018 385273 237037 385271  ↩
                                   237047 385267 237057 385225 237125
                               385210 237144 385192 237161 385167 237192 385162 237202 385159  ↩
                                   237214 385159 237227 385162 237241
                               385166 237256 385196 237324 385209 237345 385234 237375 385237  ↩
                                   237383 385238 237399 385236 237407
                               385227 237419 385213 237430 385193 237439 385174 237451 385170  ↩
                                   237455 385169 237460 385171 237475
                               385181 237503 385190 237521 385200 237533 385206 237538 385213  ↩
                                   237541 385221 237542 385235 237540 385242 237541
                               385249 237544 385260 237555 385270 237570 385289 237584 385292  ↩
                                   237589 385291 237596 385284 237630</posList>
                           </LineStringSegment>
                       </segments>
                   </Curve>
               </curveProperty>
           </Edge>
       </directedEdge>
</TopoCurve
>
```

Siehe auch

CreateTopoGeom, ST_CreateTopoGeo

8.12.2 AsTopoJSON

AsTopoJSON — Gibt die TopoJSON-Darstellung einer TopoGeometry zurück.

Synopsis

text **AsTopoJSON**(topogeometry tg, regclass edgeMapTable);

Beschreibung

Gibt eine TopoGeometry in der TopoJSON-Darstellung zurück. Wenn `edgeMapTable` nicht NULL ist, wird diese als Lookup/Speicher für die Abbildung der Identifikatoren der Kanten auf die Indizes der Kreisbögen verwendet. Dadurch wird ein kompaktes Feld "arcs" im endgültigen Dokument ermöglicht.

Wenn die Tabelle angegeben ist, wird die Existenz der Attribute "arc_id" vom Datentyp "serial" und "edge_id" vom Typ "integer" vorausgesetzt; da der Code die Tabelle nach der "edge_id" abfragt, sollte ein Index für dieses Attribut erstellt werden.

Note
Die Kreisbögen in der TopoJSON Ausgabe sind von 0 weg indiziert, während sie in der Tabelle "edgeMapTable" 1-basiert sind.

Ein vollständiges TopoJson Dokument benötigt zusätzlich zu den von dieser Funktion ausgegebenen Schnipseln, die tatsächlichen Bögen und einige Header. SiehTopoJSON specification.

Verfügbarkeit: 2.1.0

Erweiterung: 2.2.1 Unterstützung für punktförmige Eingabewerte hinzugefügt

Siehe auch

ST_AsGeoJSON

Beispiele

```
CREATE TEMP TABLE edgemap(arc_id serial, edge_id int unique);

-- header
SELECT '{ "type": "Topology", "transform": { "scale": [1,1], "translate": [0,0] }, "objects ↩
    ": {'

-- objects
UNION ALL SELECT '"' || feature_name || '": ' || AsTopoJSON(feature, 'edgemap')
FROM features.big_parcels WHERE feature_name = 'P3P4';

-- arcs
WITH edges AS (
  SELECT m.arc_id, e.geom FROM edgemap m, city_data.edge e
  WHERE e.edge_id = m.edge_id
), points AS (
  SELECT arc_id, (st_dumppoints(geom)).* FROM edges
), compare AS (
  SELECT p2.arc_id,
         CASE WHEN p1.path IS NULL THEN p2.geom
              ELSE ST_Translate(p2.geom, -ST_X(p1.geom), -ST_Y(p1.geom))
         END AS geom
  FROM points p2 LEFT OUTER JOIN points p1
  ON ( p1.arc_id = p2.arc_id AND p2.path[1] = p1.path[1]+1 )
  ORDER BY arc_id, p2.path
), arcsdump AS (
  SELECT arc_id, (regexp_matches( ST_AsGeoJSON(geom), '\[.*\]'))[1] as t
  FROM compare
), arcs AS (
  SELECT arc_id, '[' || array_to_string(array_agg(t), ',') || ']' as a FROM arcsdump
  GROUP BY arc_id
  ORDER BY arc_id
)
SELECT '}, "arcs": [' UNION ALL
SELECT array_to_string(array_agg(a), E',\n') from arcs

-- footer
UNION ALL SELECT ']}'::text as t;

-- Result:
{ "type": "Topology", "transform": { "scale": [1,1], "translate": [0,0] }, "objects": {
"P3P4": { "type": "MultiPolygon", "arcs": [[[-1]],[[6,5,-5,-4,-3,1]]]}
}, "arcs": [
 [[25,30],[6,0],[0,10],[-14,0],[0,-10],[8,0]],
 [[35,6],[0,8]],
 [[35,6],[12,0]],
 [[47,6],[0,8]],
 [[47,14],[0,8]],
 [[35,22],[12,0]],
 [[35,14],[0,8]]
 ]}
```

8.13 Räumliche Beziehungen einer Topologie

8.13.1 Equals

Equals — Gibt TRUE zurück, wenn zwei TopoGeometry Objekte aus denselben topologischen Elementarstrukturen bestehen.

Synopsis

boolean **Equals**(topogeometry tg1, topogeometry tg2);

Beschreibung

Gibt TRUE zurück, wenn zwei TopoGeometry Objekte aus denselben topologischen Elementarstrukturen: Maschen, Kanten, Knoten, bestehen.

Note
Diese Funktion unterstützt keine TopoGeometry aus einer Sammelgeometrie. Es kann auch keine TopoGeometry Objekte unterschiedlicher Topologien vergleichen

Verfügbarkeit: 1.1.0

This function supports 3d and will not drop the z-index.

Beispiele

Siehe auch

GetTopoGeomElements, ST_Equals

8.13.2 Intersects

Intersects — Gibt TRUE zurück, wenn sich kein beliebiges Paar von Elemtarstrukturen zweier TopoGeometry Objekte überschneidet.

Synopsis

boolean **Intersects**(topogeometry tg1, topogeometry tg2);

Beschreibung

Gibt TRUE zurück, wenn sich kein beliebiges Paar von Elemtarstrukturen zweier TopoGeometry Objekte überschneidet.

Note
This function not supported for topogeometries that are geometry collections. It also can not compare topogeometries from different topologies. Also not currently supported for hierarchical topogeometries (topogeometries composed of other topogeometries).

Verfügbarkeit: 1.1.0

This function supports 3d and will not drop the z-index.

Beispiele

```
```

Siehe auch

ST_Intersects

8.14 Importing and exporting Topologies

Once you have created topologies, and maybe associated topological layers, you might want to export them into a file-based format for backup or transfer into another database.

Using the standard dump/restore tools of PostgreSQL is problematic because topologies are composed by a set of tables (4 for primitives, an arbitrary number for layers) and records in metadata tables (topology.topology and topology.layer). Additionally, topology identifiers are not univoque across databases so that parameter of your topology will need to be changes upon restoring it.

In order to simplify export/restore of topologies a pair of executables are provided: `pgtopo_export` and `pgtopo_import`. Example usage:

```
pgtopo_export dev_db topo1 | pgtopo_import topo1 | psql staging_db
```

8.14.1 Using the Topology exporter

The `pgtopo_export` script takes the name of a database and a topology and outputs a dump file which can be used to import the topology (and associated layers) into a new database.

By default `pgtopo_export` writes the dump file to the standard output so that it can be piped to `pgtopo_import` or redirected to a file (refusing to write to terminal). You can optionally specify an output filename with the `-f` commandline switch.

By default `pgtopo_export` includes a dump of all layers defined against the given topology. This may be more data than you need, or may be non-working (in case your layer tables have complex dependencies) in which case you can request skipping the layers with the `--skip-layers` switch and deal with those separately.

Invoking `pgtopo_export` with the `--help` (or `-h` for short) switch will always print short usage string.

The dump file format is a compressed tar archive of a `pgtopo_export` directory containing at least a `pgtopo_dump_version` file with format version info. As of version `1` the directory contains tab-delimited CSV files with data of the topology primitive tables (node, edge_data, face, relation), the topology and layer records associated with it and (unless `--skip-layers` is given) a custom-format PostgreSQL dump of tables reported as being layers of the given topology.

8.14.2 Using the Topology importer

The `pgtopo_import` script takes a `pgtopo_export` format topology dump and a name to give to the topology to be created and outputs an SQL script reconstructing the topology and associated layers.

The generated SQL file will contain statements that create a topology with the given name, load primitive data in it, restores and registers all topology layers by properly linking all TopoGeometry values to their correct topology.

By default `pgtopo_import` reads the dump from the standard input so that it can be used in conjuction with `pgtopo_export` in a pipeline. You can optionally specify an input filename with the `-f` commandline switch.

By default `pgtopo_import` includes in the output SQL file the code to restore all layers found in the dump.

This may be unwanted or non-working in case your target database already have tables with the same name as the ones in the dump. In that case you can request skipping the layers with the `--skip-layers` switch and deal with those separately (or later).

SQL to only load and link layers to a named topology can be generated using the `--only-layers` switch. This can be useful to load layers AFTER resolving the naming conflicts or to link layers to a different topology (say a spatially-simplified version of the starting topology).

Chapter 9

Rasterdatenverwaltung, -abfrage und Anwendungen

9.1 Laden und Erstellen von Rastertabellen

In den häufigsten Anwendungsfällen werden Sie einen PostGIS-Raster durch das Laden einer bestehenden Rasterdatei, mit Hilfe des Rasterladers `raster2pgsql`, erstellen.

9.1.1 Verwendung von raster2pgsql zum Laden von Rastern

The `raster2pgsql` is a raster loader executable that loads GDAL supported raster formats into SQL suitable for loading into a PostGIS raster table. It is capable of loading folders of raster files as well as creating overviews of rasters.

Since the raster2pgsql is compiled as part of PostGIS most often (unless you compile your own GDAL library), the raster types supported by the executable will be the same as those compiled in the GDAL dependency library. To get a list of raster types your particular `raster2pgsql` supports use the `-G` switch.

Note
Bei einem bestimmten Faktor kann es vorkommen, dass die Raster in der Übersicht/Overview nicht bündig angeordnet sind, obwohl sie die Raster selbst dies sind. Siehe http://trac.osgeo.org/postgis/ticket/1764 für ein solches Beispiel.

9.1.1.1 Example Usage

Eine Beispielssitzung, wo mit dem Lader eine Eingabedatei erstellt und stückchenweise als 100x100 Kacheln hochgeladen wird, könnte so aussehen:

```
# -s use srid 4326
# -I create spatial index
# -C use standard raster constraints
# -M vacuum analyze after load
# *.tif load all these files
# -F include a filename column in the raster table
# -t tile the output 100x100
# public.demelevation load into this table
raster2pgsql -s 4326 -I -C -M -F -t 100x100 *.tif public.demelevation
> elev.sql

# -d connect to this database
# -f read this file after connecting
psql -d gisdb -f elev.sql
```

Note
If you do not specify the schema as part of the target table name, the table will be created in the default schema of the database or user you are connecting with.

Durch die Verwendung von UNIX-Pipes kann die Konvertierung und der Upload in einem Schritt vollzogen werden:

```
raster2pgsql -s 4326 -I -C -M *.tif -F -t 100x100 public.demelevation | psql -d gisdb
```

Luftbildkacheln in "Massachusetts State Plane Meters" in das Schema `aerial` laden. Einen vollständigen View und Übersichtstabellen mit Faktor 2 und 4 erstellen. Verwendet den Modus "copy" für das Insert (keine dazwischengeschaltete Datei, sondern direkt in die Datenbank). Die Option -e bedingt, dass nicht alles innerhalb einer Transaktion abläuft (nützlich, wenn Sie sofort Daten sehen wollen, ohne zu warten). Die Raster werden in 128x128 Pixel große Kacheln zerlegt und Constraints auf die Raster gesetzt. Verwendet den Modus "copy" anstelle eines Tabellen-Inserts. (-F) Erzeugt das Attribut "filename", welches die Bezeichnung der Ausgangsdateien enthält, aus denen die Rasterkacheln ausgeschnitten wurden.

```
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
```

Der -G Befehl gibt eine ähnliche Liste wie die Folgende aus

```
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
```

9.1.1.2 raster2pgsql options

-? Zeigt die Hilfe an, auch dann, wenn keine Argumente übergeben werden.

-G Gibt die unterstützten Rasterformate aus.

(c|a|d|p) Dies sind sich gegenseitig ausschließende Optionen:

- **-c** Eine neue Tabelle anlegen und mit Raster(n) befüllen, *this is the default mode*
- **-a** Raster zu einer bestehende Tabelle hinzufügen.
- **-d** Tabelle löschen, eine Neu erzeugen und mit einem oder mehreren Raster befüllen
- **-p** Beim vorbereitenden Modus wird ledigliche eine Tabelle erstellt.

Raster-Verarbeitung: Anwendung von Constraint's zur ordnungsgemäßen Registrierung im Rasterkatalog

- **-C** Anwendung von Raster-Constraints, wie SRID, Zellgröße etc., um die ordnungsgemäße Registrierung des Rasters in der `raster_columns` View sicherzustellen.
- **-x** Unterbindet das Setzen der "Max-Extent" Bedingung. Wird nur angewandt, wenn auch die -C Flag gesetzt ist.

-r Setzt die Constraints (räumlich eindeutig und die Coverage-Kachel) der regelmäßigen Blöcke. Wird nur angewandt, wenn auch die -C Flag gesetzt ist.

Rasterdaten-Verarbeitung: Optionale Parameter zur Manipulation von Input Raster Datensätzen

-s <SRID> Dem Output-Raster eine bestimmte SRID zuweisen. Wenn keine SRID oder Null angegeben wird, werden die Raster-Metadaten auf eine geignete SRID hin überprüft.

-b BAND Die Kennung (1-basiert) des Bandes, das aus dem Raster entnommen werden soll. Um mehrere Bänder anzugeben, trennen Sie die Kennungen bitte durch ein Komma (,).

-t TILE_SIZE Zerlegt den Raster in Kacheln, um eine Kachel pro Tabellenzeile einzufügen. `TILE_SIZE` wird entweder in BREITExHöhe ausgedrückt, oder auf den Wert "auto" gesetzt, wodurch der Raster-Lader eine passende Kachelgröße an Hand des ersten Raster's ermittelt und diese dann auf die anderen Raster anwendet.

-P Die ganz rechts und ganz unten liegenden Kacheln aufstocken, damit für alle Kacheln gleiche Breite und Höhe sichergestellt ist.

-R, --register Einen im Dateisystem vorliegenden Raster als (out-db) Raster registrieren.

Es werden nur die Metadaten und der Dateipfad des Rasters abgespeichert (nicht die Rasterzellen).

-l OVERVIEW_FACTOR Erzeugt eine Übersicht/Overview des Rasters. Mehrere Faktoren sind durch einen Beistrich(,) zu trennen. Die Benennung der Übersichtstabelle erfolgt dem Muster o_`overview_factor_table`, wobei `overview_factor` ein Platzhalter für den numerischen Wert von "overview_factor" ist und `table` für den zugrundeliegenden Tabellennamen. Die erstellte Übersicht wird in der Datenbank gespeichert, auch wenn die Option -R gesetzt ist. Anmerkung: die erzeugte SQL-Datei enthält sowohl die Haupttabelle, als auch die Übersichtstabellen.

-N NODATA Der NODATA-Wert, der für Bänder verwendet wird, die keinen NODATA-Wert definiert haben.

Optionale Parameter zur Manipulation von Datenbankobjekten

-f COLUMN Gibt den Spaltennamen des Zielrasters an; standardmäßig wird er 'rast' benannt.

-F Eine Spalte mit dem Dateinamen hinzufügen

-n COLUMN Gibt die Bezeichnung für die Spalte mit dem Dateinamen an. Schließt -F" mit ein.

-q Setzt die PostgreSQL-Identifikatoren unter Anführungszeichen.

-I Einen GIST-Index auf die Rasterspalte anlegen.

-M VACUUM ANALYZE auf die Rastertabelle.

-k Keeps empty tiles and skips NODATA value checks for each raster band. Note you save time in checking, but could end up with far more junk rows in your database and those junk rows are not marked as empty tiles.

-T tablespace Bestimmt den Tablespace für die neue Tabelle. Beachten Sie bitte, dass Indizes (einschließlich des Primärschlüssels) weiterhin den standardmäßigen Tablespace nutzen, solange nicht die -X Flag benutzt wird.

-X tablespace Bestimmt den Tablespace für den neuen Index der Tabelle. Dieser gilt sowohl für den Primärschlüssel als auch für den räumlichen Index, falls die -I Flag gesetzt ist.

-Y max_rows_per_copy=50 Use copy statements instead of insert statements. Optionally specify `max_rows_per_copy`; default 50 when not specified.

-e Keine Transaktion verwenden, sondern jede Anweisung einzeln ausführen.

-E ENDIAN Legt die Byte-Reihenfolge des binär erstellten Rasters fest; geben Sie für XDR 0 und für NDR (Standardwert) 1 an; zurzeit wird nur die Ausgabe von NDR unterstützt.

-V version Bestimmt die Version des Ausgabeformats. Voreingestellt ist 0. Zur Zeit wird auch nur 0 unterstützt.

9.1.2 Erzeugung von Rastern mit den PostGIS Rasterfunktionen

Oftmals werden Sie die Raster und die Rastertabellen direkt in der Datenbank erzeugen wollen. Dafür existieren eine Unmenge an Funktionen. Dies verlangt im Allgemeinen die folgende Schritte.

1. Erstellung einer Tabelle mit einer Rasterspalte für die neuen Rasterdatensätze:

```
CREATE TABLE myrasters(rid serial primary key, rast raster);
```

2. Es existieren viele Funktionen die Ihnen helfen dieses Ziel zu erreichen. Wenn Sie einen Raster nicht von anderen Rastern ableiten, sondern selbst erzeugen, können Sie mit ST_MakeEmptyRaster beginnen, gefolgt von ST_AddBand

   Sie können Raster auch aus Geometrien erzeugen. Hierzu können Sie ST_AsRaster verwenden, möglicherweise in Verbindung mit anderen Funktionen, wie ST_Union, ST_MapAlgebraFct oder irgendeiner anderen Map Algebra Funktion.

   Es gibt sogar noch viele andere Möglichkeiten, um eine neue Rastertabelle aus bestehenden Tabellen zu erzeugen. Sie können zum Beispiel mit ST_Transform einen Raster in eine andere Projektion transformieren und so eine neue Rastertabelle erstellen.

3. Wenn Sie mit der Erstbefüllung der Tabelle fertig sind, werden Sie einen räumlichen Index auf die Rasterspalte setzen wollen:

   ```
   CREATE INDEX myrasters_rast_st_convexhull_idx ON myrasters USING gist( ST_ConvexHull( ↩
       rast) );
   ```

   Beachten Sie bitte die Verwendung von ST_ConvexHull; der Grund dafür ist, dass die meisten Rasteroperatoren auf der konvexen Hülle des Rasters beruhen.

   Note
   Vor der Version 2.0 von PostGIS, basierten die Raster auf der Einhüllenden, anstatt auf der konvexen Hülle. Damit die räumlichen Indizes korrekt funktionieren, müssen Sie diese löschen und mit einem auf der konvexen Hülle basierenden Index ersetzen.

4. Mittels AddRasterConstraints Bedingungen auf den Raster legen.

9.1.3 Using "out db" cloud rasters

The `raster2pgsql` tool uses GDAL to access raster data, and can take advantage of a key GDAL feature: the ability to read from rasters that are stored remotely in cloud "object stores" (e.g. AWS S3, Google Cloud Storage).

Efficient use of cloud stored rasters requires the use of a "cloud optimized" format. The most well-known and widely used is the "cloud optimized GeoTIFF" format. Using a non-cloud format, like a JPEG, or an un-tiled TIFF will result in very poor performance, as the system will have to download the entire raster each time it needs to access a subset.

First, load your raster into the cloud storage of your choice. Once it is loaded, you will have a URI to access it with, either an "http" URI, or sometimes a URI specific to the service. (e.g., "s3://bucket/object"). To access non-public buckets, you will need to supply GDAL config options to authenticate your connection. Note that this command is *reading* from the cloud raster and *writing* to the database.

```
AWS_ACCESS_KEY_ID=xxxxxxxxxxxxxxxxxxx \
AWS_SECRET_ACCESS_KEY=xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx \
raster2pgsql \
  -s 990000 \
  -t 256x256 \
  -I \
  -R \
  /vsis3/your.bucket.com/your_file.tif \
  your_table \
  | psql your_db
```

Once the table is loaded, you need to give the database permission to read from remote rasters, by setting two permissions, postgis.enable_outdb_rasters and postgis.gdal_enabled_drivers.

```
SET postgis.enable_outdb_rasters = true;
SET postgis.gdal_enabled_drivers TO 'ENABLE_ALL';
```

To make the changes sticky, set them directly on your database. You will need to re-connect to experience the new settings.

```
ALTER DATABASE your_db SET postgis.enable_outdb_rasters = true;
ALTER DATABASE your_db SET postgis.gdal_enabled_drivers TO 'ENABLE_ALL';
```

For non-public rasters, you may have to provide access keys to read from the cloud rasters. The same keys you used to write the `raster2pgsql` call can be set for use inside the database, with the postgis.gdal_config_options configuration. Note that multiple options can be set by space-separating the `key=value` pairs.

```
SET postgis.gdal_vsi_options = 'AWS_ACCESS_KEY_ID=xxxxxxxxxxxxxxxxxx
AWS_SECRET_ACCESS_KEY=xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx';
```

Once you have the data loaded and permissions set you can interact with the raster table like any other raster table, using the same functions. The database will handle all the mechanics of connecting to the cloud data when it needs to read pixel data.

9.2 Raster Katalog

Mit PostGIS kommen zwei Views des Rasterkatalogs. Beide Views nützen die Information, welche in den Bedingungen/Constraints der Rastertabellen festgelegt ist. Da die Bedingungen zwingend sind, sind die Views des Rasterkatalogs immer konsistent mit den Daten in den Rastertabellen.

1. `raster_columns` diese View/gespeicherte Abfrage katalogisiert alle Rastertabellenspalten Ihrer Datenbank.
2. `raster_overviews` Dieser View katalogisiert all jene Spalten einer Rastertabelle in Ihrer Datenbank, die als Übersicht für Rastertabellen mit höherer Auflösung dienen. Tabellen dieses Typs werden mit der `-l` Option beim Laden erstellt.

9.2.1 Rasterspalten Katalog

`raster_columns` ist ein Katalog mit allen Rasterspalten Ihrer Datenbanktabellen. Es handelt sich dabei um einen View, der die Constraints auf die Tabellen ausnutzt, um so immer konsistent mit dem aktuellen Stand der Datenbank zu bleiben; sogar dann, wenn Sie den Raster aus einem Backup oder einer anderen Datenbank wiederherstellen. Der `raster_columns` Katalog beinhaltet die folgenden Spalten.

Falls Sie Ihre Tabellen nicht mit dem Loader erstellt haben, oder vergessen haben, die `-C` Option während des Ladens anzugeben, können Sie die Constraints auch anschließend erzwingen, indem Sie AddRasterConstraints verwenden, wodurch der `raster_columns` Katalog die Information über Ihre Rasterkacheln, wie üblich abspeichert.

- `r_table_catalog` Die Datenbank, in der sich die Tabelle befindet. Greift immer auf die aktuelle Datenbank zu.
- `r_table_schema` Das Datenbankschema in dem sich die Rastertabelle befindet.
- `r_table_name` Rastertabelle
- `r_raster_column` Die Spalte, in der Tabelle `r_table_name`, die den Datentyp Raster aufweist. In PostGIS gibt es nichts, was Sie daran hindert, mehrere Rasterspalten in einer Tabelle zu haben. Somit ist es möglich auf unterschiedliche Raster(spalten) in einer einzigen Rastertabelle zuzugreifen.
- `srid` Der Identifikator für das Koordinatensystem in dem der Raster vorliegt. Sollte in Section 4.5 eingetragen sein.
- `scale_x` Der Skalierungsfaktor zwischen den Koordinaten der Vektoren und den Pixeln. Dieser steht nur dann zur Verfügung, wenn alle Kacheln der Rasterspalte denselben `scale_x` aufweisen und dieser Constraint auch gesetzt ist. Siehe ST_ScaleX für genauere Angaben.
- `scale_y` Der Skalierungsfaktor zwischen den Koordinaten der Vektoren und den Pixeln. Dieser steht nur dann zur Verfügung, wenn alle Kacheln der Rasterspalte denselben `scale_y` aufweisen und der Constraint `scale_y` auch gesetzt ist. Siehe ST_ScaleY für genauere Angaben.
- `blocksize_x` Die Breite (Anzahl der waagrechten Zellen) einer Rasterkachel. Siehe ST_Width für weitere Details.

- `blocksize_y` Die Höhe (Anzahl der senkrechten Zellen) einer Rasterkachel. Siehe ST_Height für weitere Details.
- `same_alignment` Eine boolesche Variable, die TRUE ist, wenn alle Rasterkacheln dieselbe Ausrichtung haben. Siehe ST_SameAlignment für genauere Angaben.
- `regular_blocking` Wenn auf die Rasterspalte die Constraints für die räumliche Eindeutigkeit und für die Coveragekachel gesetzt sind, ist der Wert TRUE, ansonsten FALSE..
- `num_bands` Die Anzahl der Bänder, die jede Kachel des Rasters aufweist. ST_NumBands gibt die gleiche Information aus. ST_NumBands
- `pixel_types` Ein Feld das den Pixeltyp für die Bänder festlegt. Die Anzahl der Elemente in diesem Feld entspricht der Anzahl der Rasterbänder. Die "pixel_types" sind unter ST_BandPixelType definiert.
- `nodata_values` Ein Feld mit Double Precision Zahlen, welche den `nodata_value` für jedes Band festlegen. Die Anzahl der Elemente in diesem Feld entspricht der Anzahl der Rasterbänder. Diese Zahlen legen den Pixelwert für jedes Rasterband fest, der bei den meisten Operationen ignoriert wir. Eine ähnliche Information erhalten Sie durch ST_BandNoDataValue.
- `out_db` Ein Feld mit booleschen Flags, das anzeigt, ob die Rasterbanddaten außerhalb der Datenbank gehalten werden. Die Anzahl der Elemente in diesem Feld entspricht der Anzahl der Rasterbänder.
- `extent` Die Ausdehnung aller Rasterspalten in Ihrem Rasterdatensatz. Falls Sie vor haben Daten zu laden, welche die Ausdehnung des Datensatzes ändern, sollten Sie die Funktion DropRasterConstraints ausführen, bevor Sie die Daten laden und nach dem Laden die Constraints mit der Funktion AddRasterConstraints erneut setzen.
- `spatial_index` Eine Boolesche Variable, die TRUE anzeigt, wenn ein räumlicher Index auf das Rasterattribut gelegt ist.

9.2.2 Raster Übersicht/Raster Overviews

`raster_overviews` Katalogisiert Information über die Rastertabellenspalten die für die Übersichten/Overviews herangezogen wurden, sowie weitere zusätzliche Information bezüglich Overviews. Die Übersichtstabellen werden sowohl in `raster_columns` als auch in `raster_overviews` registriert, da sie sowohl eigene Raster darstellen, als auch, als niedriger aufgelöstes Zerrbild einer höher aufgelösten Tabelle, einem bestimmten Zweck dienen. Wenn Sie den `-l` Switch beim Laden des Rasters angeben, werden diese gemeinsam mit der Rasterhaupttabelle erstellt; sie können aber auch händisch über AddOverviewConstraints erstellt werden.

Übersichtstabellen enthalten dieselben Constraints wie andere Rastertabellen und zusätzliche informative Constraints, spezifisch für die Übersichten.

Note
Die Information in `raster_overviews` befindet sich nicht in `raster_columns`. Falls Sie die Information der Übersichtstabelle und der `raster_columns` zusammen benötigen, können Sie einen Join auf `raster_overviews` und `raster_columns` ausführen, um die gesamte Information zu erhalten.

Die zwei Hauptgründe für Übersichtsraster sind:

1. Eine niedrig aufgelöste Darstellung der Basistabellen; wird im Allgemeinen zum schnellen Hinauszoomen verwendet.
2. Die Berechnungen laufen grundsätzlich schneller ab, als bei den Stammdaten mit höherer Auflösung, da weniger Datensätze vorhanden sind und die Pixel eine größere Fläche abdecken. Obwohl diese Berechnungen nicht so exakt sind, wie jene auf die hochauflösenden Stammtabellen, sind sie doch für viele Überschlagsrechnunen ausreichend.

Der `raster_overviews` Katalog enthält folgende Attribute an Information.

- `o_table_catalog` Die Datenbank, in der sich die Übersichtstabelle befindet. Liest immer die aktuelle Datenbank.
- `o_table_schema` Das Datenbankschema dem die Rasterübersichtstabelle angehört.

- o_table_name Der Tabellenname der Rasterübersicht
- o_raster_column das Rasterattribut in der Übersichtstabelle.
- r_table_catalog Die Datenbank, in der sich die Rastertabelle befindet, für die diese Übersicht gilt. Greift immer auf die aktuelle Datenbank zu.
- r_table_schema Das Datenbankschema, in dem sich die Rastertabelle befindet, zu der der Übersichtsdienst gehört.
- r_table_name Die Rastertabelle, welche von dieser Übersicht bedient wird.
- r_raster_column Die Rasterspalte, die diese Overviewspalte bedient.
- overview_factor - der Pyramidenlevel der Übersichtstabelle. Umso größer die Zahl ist, desto geringer ist die Auflösung. Wenn ein Ordner für die Bilder angegeben ist, rechnet raster2pgsql eine Übersicht für jede Bilddatei und ladet diese einzeln. Es wird Level 1 und die Ursprungsdatei angenommen. Beim Level 2 repräsentiert jede Kachel 4 Originalkacheln. Angenommen Sie haben einen Ordner mit Bilddateien in einer Auflösung von 5000x5000 Pixel, die Sie auf 125x125 große Kacheln zerlegen wollen. Für jede Bilddattei enthält die Basistabelle (5000*5000)/(125*125) Datensätze = 1600, Ihre (l=2) o_2 Tabelle hat dann eine Obergrenze von (1600/Power(2,2)) = 400 Zeilen, Ihre (l=3) o_3 (1600/Power(2,3)) = 200 Zeilen. Wenn sich die Pixel nicht durch die Größe Ihrer Kacheln teilen lassen, erhalten Sie einige Ausschusskacheln (Kacheln die nicht zur Gänze gefüllt sind). Beachten Sie bitte, dass jede durch raster2pgsql erzeugte Übersichtskachel dieselbe Pixelanzahl hat wie die ursprüngliche Kachel, aber eine geringere Auflösung, wo ein Pixel (Power(2,overview_factor) Pixel der Ursprungsdatei) repräsentiert.

9.3 Eigene Anwendungen mit PostGIS Raster erstellen

The fact that PostGIS raster provides you with SQL functions to render rasters in known image formats gives you a lot of options for rendering them. For example you can use OpenOffice / LibreOffice for rendering as demonstrated in Rendering PostGIS Raster graphics with LibreOffice Base Reports. In addition you can use a wide variety of languages as demonstrated in this section.

9.3.1 PHP Beispiel: Ausgabe mittels ST_AsPNG in Verbindung mit anderen Rasterfunktionen

In diesem Abschnitt zeigen wir die Anwendung des PHP PostgreSQL Treibers und der Funktion ST_AsGDALRaster, um die Bänder 1,2,3 eines Rasters an einen PHP Request-Stream zu übergeben. Dieser kann dann in einen "img src" HTML Tag eingebunden werden.

Dieses Beispiel zeigt, wie Sie ein ganzes Bündel von Rasterfunktionen kombinieren können, um jene Kacheln zu erhalten, die ein bestimmtes WGS84 Umgebungsrechteck schneiden. Anschließend werden alle Bänder dieser Kacheln mit ST_Union vereinigt, mit ST_Transform in die vom Benutzer vorgegebene Projektion transformiert und das Ergebnis mit ST_AsPNG als PNG ausgegeben.

Sie können das unten angeführte Programm über

```
http://mywebserver/test_raster.php?srid=2249
```

aufrufen, um den Raster in "Massachusetts State Plane Feet" zu erhalten.

```
<?php
/** contents of test_raster.php **/
$conn_str ='dbname=mydb host=localhost port=5432 user=myuser password=mypwd';
$dbconn = pg_connect($conn_str);
header('Content-Type: image/png');
/**If a particular projection was requested use it otherwise use mass state plane meters  ↩
    **/
if (!empty( $_REQUEST['srid'] ) && is_numeric( $_REQUEST['srid']) ){
                $input_srid = intval($_REQUEST['srid']);
}
else { $input_srid = 26986; }
```

```
/** The set bytea_output may be needed for PostgreSQL 9.0+, but not for 8.4 **/
$sql = "set bytea_output='escape';
SELECT ST_AsPNG(ST_Transform(
                        ST_AddBand(ST_Union(rast,1), ARRAY[ST_Union(rast,2),ST_Union(rast ↩
                            ,3)])
                                ,$input_srid) ) As new_rast
 FROM aerials.boston
        WHERE
         ST_Intersects(rast, ST_Transform(ST_MakeEnvelope(-71.1217, 42.227, -71.1210,  ↩
             42.218,4326),26986) )";
$result = pg_query($sql);
$row = pg_fetch_row($result);
pg_free_result($result);
if ($row === false) return;
echo pg_unescape_bytea($row[0]);
?>
```

9.3.2 ASP.NET C# Beispiel: Ausgabe mittels ST_AsPNG in Verbindung mit anderen Rasterfunktionen

In diesem Abschnitt zeigen wir die Anwendung des Npgsql PostgreSQL .NET Treibers und der Funktion ST_AsGDALRaster, um die Bänder 1,2,3 eines Rasters an einen PHP Request-Stream zu übergeben. Dieser kann dann in einen "img src" HTML Tag eingebunden werden.

Für dieses Beispiel benötigen Sie den npgsql .NET PostgreSQL Treiber. Um loslegen zu können, reicht es aus, dass Sie die neueste Version von http://npgsql.projects.postgresql.org/ in Ihren ASP.NET Ordner laden.

Dieses Beispiel zeigt, wie Sie ein ganzes Bündel von Rasterfunktionen kombinieren können, um jene Kacheln zu erhalten, die ein bestimmtes WGS84 Umgebungsrechteck schneiden. Anschließend werden alle Bänder dieser Kacheln mit ST_Union vereinigt, mit ST_Transform in die vom Benutzer vorgegebene Projektion transformiert und das Ergebnis mit ST_AsPNG als PNG ausgegeben.

Dasselbe Beispiel wie Section 9.3.1 nur in C# implementiert.

Sie können das unten angeführte Programm über

```
http://mywebserver/TestRaster.ashx?srid=2249
```

aufrufen, um den Raster in "Massachusetts State Plane Feet" zu bekommen.

```
-- 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;
        }
}
```

9.3.3 Applikation für die Java-Konsole, welche eine Rasterabfrage als Bilddatei ausgibt

Eine einfache Java Applikation, die eine Abfrage entgegennimmt, ein Bild erzeugt und in eine bestimmte Datei ausgibt.

Sie können die neuesten PostgreSQL JDBC Treiber unter http://jdbc.postgresql.org/download.html herunterladen.

Sie können den unten angegebenen Code mit einem Befehl wie folgt kompilieren:

```
set env CLASSPATH .:..\postgresql-9.0-801.jdbc4.jar
```

```
javac SaveQueryImage.java
jar cfm SaveQueryImage.jar Manifest.txt *.class
```

Und ihn von der Befehlszeile wie folgt aufrufen:

```
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);
      }
  }
}
```

9.3.4 Verwenden Sie PLPython um Bilder via SQL herauszuschreiben

Diese als plpython gespeicherte Prozedur erzeugt eine Datei pro Datensatz im Serververzeichnis. Benötigt die Installation von plpython. Funktioniert sowohl mit plpythonu als auch mit plpython3u.

```
CREATE OR REPLACE FUNCTION write_file (param_bytes bytea, param_filepath text)
RETURNS text
AS $$
f = open(param_filepath, 'wb+')
f.write(param_bytes)
return param_filepath
$$ LANGUAGE plpythonu;
```

```
--write out 5 images to the PostgreSQL server in varying sizes
-- note the postgresql daemon account needs to have write access to folder
-- this echos back the file names created;
 SELECT write_file(ST_AsPNG(
        ST_AsRaster(ST_Buffer(ST_Point(1,5),j*5, 'quad_segs=2'),150*j, 150*j, '8BUI',100)),
         'C:/temp/slices'|| j || '.png')
         FROM generate_series(1,5) As j;

     write_file
---------------------
 C:/temp/slices1.png
 C:/temp/slices2.png
 C:/temp/slices3.png
 C:/temp/slices4.png
 C:/temp/slices5.png
```

9.3.5 Faster mit PSQL ausgeben

Leider hat PSQL keine einfach zu benützende Funktion für die Ausgabe von Binärdateien eingebaut. Dieser Hack baut auf der etwas veralteten "Large Object" Unterstützung von PostgreSQL auf. Um ihn anzuwenden verbinden Sie sich bitte zuerst über die Befehlszeile "psql" mit Ihrer Datenbank.

Anders als beim Ansatz mit Python, wird die Datei bei diesem Ansatz auf Ihrem lokalen Rechner erzeugt.

```
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 10

Referenz Raster

Im Folgenden sind die gebräuchlichsten Funktionen aufgeführt, die zur Zeit durch PostGIS-Raster zur Verfügung gestellt werden. Es gibt noch zusätzliche Funktionen, die zur Unterstützung von Rasterobjekten benötigt werden und für den Anwender nur geringe Bedeutung haben.

`raster` ist ein neuer PostGIS Datentyp, der zum Speichern und zur Analyse von Rasterdaten verwendet wird.

Um Raster aus Rasterdateien zu laden, siehe Section 9.1

Die Beispiele dieser Referenz benutzen eine Rastertabelle, die mit folgendem Code aus Dummy-Rastern erstellt wurde

```
CREATE TABLE dummy_rast(rid integer, rast raster);
INSERT INTO dummy_rast(rid, rast)
VALUES (1,
('01' -- little endian (uint8 ndr)
||
'0000' -- version (uint16 0)
||
'0000' -- nBands (uint16 0)
||
'0000000000000040' -- scaleX (float64 2)
||
'0000000000000840' -- scaleY (float64 3)
||
'000000000000E03F' -- ipX (float64 0.5)
||
'000000000000E03F' -- ipY (float64 0.5)
||
'0000000000000000' -- skewX (float64 0)
||
'0000000000000000' -- skewY (float64 0)
||
'00000000' -- SRID (int32 0)
||
'0A00' -- width (uint16 10)
||
'1400' -- height (uint16 20)
)::raster
),
-- Raster: 5 x 5 pixels, 3 bands, PT_8BUI pixel type, NODATA = 0
(2,  ('01000003009A9999999999A93F9A9999999999A9BF000000E02B274A' ||
'41000000077195641000000000000000000000000000000000 ↩
    FFFFFFFF050005000400FDFEFDFEFEFDFEFEFDF9FAFEF' ||
' ↩
    EFCF9FBFDFEFEFDFCFAFEFEFE04004E627AADD16076B4F9FE6370A9F5FE59637AB0E54F5861708704004656648 7A1506
    ')::raster);
```

10.1 Datentypen zur Unterstützung von Rastern.

10.1.1 geomval

geomval — Ein räumlicher Datatyp mit zwei Feldern - geom (enthält das geometrische Element) und val (enthält den Zellwert eines Rasterbandes in Doppelter Genauigkeit).

Beschreibung

geomval ist ein a zusammengesetzter Datentyp, der aus einem geometrischen Objekt, auf welches vom Attribut ".geom" her verwiesen wird, und "val" besteht. "val" hält einen Wert in Double Precision, der dem Pixelwert an einer bestimmten geometrischen Stelle in einem Rasterband entspricht. Verwendung findet dieser Datentyp in ST_DumpAsPolygon und als Ausgabetyp in der Familie der Rasterüberlagerungsfunktionen um ein Rasterband in Polygone zu zerlegen.

Siehe auch

Section 12.6

10.1.2 addbandarg

addbandarg — Ein zusammengesetzter Datentyp, der als Eingabewert für die Funktion "ST_AddBand" verwendet wird und sowohl Attribute als auch Initialwert des neuen Bandes festlegt.

Beschreibung

Ein zusammengesetzter Datentyp, der als Eingabewert für die Funktion "ST_AddBand" verwendet wird und sowohl Attribute als auch Initialwert des neuen Bandes festlegt.

index **integer** Ein von 1 aufwärts zählender Wert, der die Positionunter den Rasterbänder angibt, an der das neue Band eingefügt werden soll. Wenn er NULL ist wird das neue Band am Ende der Rasterbänder hinzugefügt.

pixeltype **text** Der Datentyp der Rasterzellen/"Pixel Type" des neuen Bandes. Einer jener Datentypen, die unter ST_BandPixelType definiert sind.

initialvalue **double precision** Der Ausgangswert, auf den die Zellen eines neuen Bandes gesetzt werden.

nodataval **double precision** Der NODATA-Wert des neuen Bandes. Wenn NULL, dann wird für das neue Band kein NODATA-Wert vergeben.

Siehe auch

ST_AddBand

10.1.3 rastbandarg

rastbandarg — Ein zusammengesetzter Datentyp, der verwendet wird um den Raster und, über einen Index, das Band des Rasters anzugeben.

Beschreibung

Ein zusammengesetzter Datentyp, der verwendet wird um den Raster und, über einen Index, das Band des Rasters anzugeben.

rast **raster** Besagter Raster

nband **integer** Der 1-basierte Wert, welcher das betreffende Rasterband kennzeichnet

Siehe auch

ST_MapAlgebra (callback function version)

10.1.4 raster

raster — Der räumliche Datentyp Raster

Beschreibung

Raster ist ein Geodatentyp, der verwendet wird um digitale Bilder darzustellen. Diese Daten können zum Beispiel von JPEGs, TIFFs, PNGs oder digitalen Höhenmodellen stammen. Jeder Raster kann aus 1 oder mehreren Bänder bestehen, diese wiederum bestehen aus einzelnen Zellen denen Werte zugewiesen werden können. Raster können georeferenziert werden.

Note
Verlangt, dass PostGIS mit GDAL-Unterstützung kompiliert wurde. Gegenwärtig können Raster implizit in den Geometry Datentyp umgewandelt werden, allerdings gibt diese Umwandlung die ST_ConvexHull des Rasters zurück. Diese automatische Typumwandlung kann möglicherweise in der nahen Zukunft entfernt werden; verlassen Sie sich daher bitte nicht darauf.

Typumwandlung

Dieser Abschnitt beschreibt die automatischen/impliziten als auch expliziten Typumwandlungen, die für diesen Datentyp erlaubt sind.

Typumwandlung nach	Verhaltensweise
Geometrie	implizit

Siehe auch

Chapter 10

10.1.5 reclassarg

reclassarg — Ein Zusammengesetzter Datentyp, der als Eingabewert für die Funktion "ST_Reclass" dient und die Neuklassifizierung festlegt.

Beschreibung

Ein Zusammengesetzter Datentyp, der als Eingabewert für die Funktion "ST_Reclass" dient und die Neuklassifizierung festlegt.

nband **integer** Die Nummer des Bandes das neu klasssifiziert werden soll.

reclassexpr **text** Ein Ausdruck, der aus "range:map_range" Abbildungen besteht, die als Intervalle dargestellt und durch Beistriche getrennt sind. Der Ausdruck gibt an, wie die alten Zellwerte auf die neuen Zellwerte des Bandes abgebildet werden sollen. "(" bedeutet >, ")" bedeutet <, "]" < oder gleich, "[" > oder gleich

```
1. [a-b] = a <= x <= b

2. (a-b] = a < x <= b

3. [a-b) = a <= x < b
```

```
4. (a-b) = a < x < b
```

Die runden Klammern sind bei dieser Notation optional, d.h. "a-b" ist gleichbedeutend mit "(a-b)".

pixeltype **text** Einer der unter ST_BandPixelType definierten Datentypen

nodataval **double precision** Der Zellwert, der als NODATA/NULL betrachtet werden soll. Bei der Ausgabe von Bildern, die Transparenz unterstützen, bleibt dieser leer.

Beispiel: Band 2 in 8BUI, mit einem NODATA-Wert von 255, umgruppieren.

```
SELECT ROW(2, '0-100:1-10, 101-500:11-150,501 - 10000: 151-254', '8BUI', 255)::reclassarg;
```

Beispiel: Band 1 in 1BB, mit einem unbestimmten NODATA-Wert, umgruppieren.

```
SELECT ROW(1, '0-100]:0, (100-255:1', '1BB', NULL)::reclassarg;
```

Siehe auch

ST_Reclass

10.1.6 summarystats

summarystats — Ein zusammgesetzter Datentyp, welcher von den Funktionen ST_SummaryStats und ST_SummaryStatsAgg zurückgegeben wird.

Beschreibung

Ein zusammgesetzter Datentyp, welcher von ST_SummaryStats und ST_SummaryStatsAgg zurückgegeben wird.

count **integer** Die Summe aller Zellen, die für eine zusammenfassende Statistik abgezählt wurden.

sum **double precision** Die Summe aller abgezählten Zellwerte.

mean **double precision** Der arithmetische Mittelwert aller abgezählten Zellwerte.

stddev **double precision** Die Standardabweichung aller abgezählten Zellwerte.

min **double precision** Der Minimalwert aller abgezählten Zellwerte.

max **double precision** Der Maximalwert aller abgezählten Zellwerte.

Siehe auch

ST_SummaryStats, ST_SummaryStatsAgg

10.1.7 unionarg

unionarg — Ein zusammengesetzter Datentyp, der als Eingabewert für die Funktion "ST_Union" dient. Dieser Datentyp bestimmt die zu behandelnden Bänder, sowie die Verhaltensweise des UNION Operators.

Beschreibung

Ein zusammengesetzter Datentyp, der als Eingabewert für die Funktion "ST_Union" dient. Dieser Datentyp bestimmt die zu behandelnden Bänder, sowie die Verhaltensweise des UNION Operators.

nband **integer** Der 1-basierte Wert, welcher das Band aller Input-Rraster kennzeichnet, die bearbeitet werden sollen.

uniontype **text** Die Variante des UNION Operators. Eine der unter ST_Union definierten Varianten.

Siehe auch

ST_Union

10.2 Rastermanagement

10.2.1 AddRasterConstraints

AddRasterConstraints — Fügt die Raster-Constraints zu einer bestimmten Spalte einer bereits geladenen Rastertabelle hinzu. Diese Constraints beschränken das Koordinatentransformationssystem, den Maßstab, die Blockgröße, die Ausrichtung, die Bänder, den Bandtyp und eine Flag, die anzeigt ob die Rasterspalte regelmäßig geblockt ist. Es müssen bereits Daten in die Tabelle geladen sein, damit die Constraints abgeleitet werden können. Gibt TRUE zurück, wenn das Setzen der Constraints ausgeführt wurde; bei Problemen wird eine Meldung angezeigt.

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 nodata_values=true , boolean out_db=true , boolean extent=true);
boolean **AddRasterConstraints**(name rasttable, name rastcolumn, text[] VARIADIC constraints);
boolean **AddRasterConstraints**(name rastschema, name rasttable, name rastcolumn, text[] VARIADIC constraints);
boolean **AddRasterConstraints**(name rastschema, name rasttable, name rastcolumn, boolean srid=true, boolean scale_x=true, boolean scale_y=true, boolean blocksize_x=true, boolean blocksize_y=true, boolean same_alignment=true, boolean regular_blocking=fa
boolean num_bands=true, boolean pixel_types=true, boolean nodata_values=true , boolean out_db=true , boolean extent=true);

Beschreibung

Setzt die Constraints auf eine Rasterspalte Diese werden verwendet um die Information in dem Rasterkatalog `raster_columns` anzuzeigeen. Der Parameter `rastschema` bezeichnet das Tabellenschema in dem die Tabelle liegt. Die Ganzzahl `srid` verweist auf einen Eintrag in der Tabelle "spatial_ref_sys".

Der `raster2pgsql` Lader verwendet diese Funktion um Rastertabellen zu registrieren.

Gültige Bezeichnungen für Constraints: siehe Section 9.2.1 für weitere Details.

- `blocksize` bestimmt die Datenblockgröße von X und Y
- `blocksize_x` setzt die X-Kachel (die Breite der Kacheln in Pixel)
- `blocksize_y` setzt die Y-Kachel (die Höhe der Kacheln in Pixel)
- `extent` berechnet die räumliche Ausdehnung der ganzen Tabelle und setzt einen Constraint, der alle Raster auf diesen Ausschnitt beschränkt.
- `num_bands` Anzahl der Bänder
- `pixel_types` liest ein Feld mit Pixeltypen für jedes Band ein, und stellt sicher, dass alle Bänder denselben Pixeltyp haben

- `regular_blocking` setzt die Constraints für die räumliche Eindeutigkeit (keine zwei Raster dürfen räumlich ident sein) und für die Coverage-Kachel (der Raster wird an einem Coverage ausgerichtet)
- `same_alignment` stellt sicher, dass alle die selbe Ausrichtung haben, d.h. der Vergleich von zwei beliebigen Kacheln gibt TRUE zurück. Siehe ST_SameAlignment.
- `srid` stellt sicher, dass alle die selbe SRID aufweisen
- Mehr -- alles was als Eingabe in die obere Funktion aufgeführt ist

---

Note
Diese Funktion leitet die Constraints von den Daten ab, die bereits in der Tabelle vorliegen. Damit Sie die Funktion anwenden können, müssen Sie zuerst die Rasterspalte erzeugen in die Sie anschließend die Daten laden.

Note
Falls Sie weitere Daten in Ihre Tabellen laden müssen, nachdem Sie bereits die Constraints gesetzt haben, können Sie DropRasterConstraints ausführen, wenn sich die räumliche Ausdehnung Ihrer Daten geändert hat.

Verfügbarkeit: 2.0.0

Beispiele: Basierend auf den Daten sämtliche Connstraints auf die Spalte setzen

```
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}
```

Beispiele: Einen einzelnen Constraint setzen

```
CREATE TABLE public.myrasters2(rid SERIAL primary key, rast raster);
INSERT INTO myrasters2(rast)
SELECT ST_AddBand(ST_MakeEmptyRaster(1000, 1000, 0.3, -0.3, 2, 2, 0, 0,4326), 1, '8BSI':: ↩
    text, -129, NULL);

SELECT AddRasterConstraints('public'::name, 'myrasters2'::name, 'rast'::name,' ↩
    regular_blocking', 'blocksize');
-- get notice--
NOTICE:  Adding regular blocking constraint
NOTICE:  Adding blocksize-X constraint
NOTICE:  Adding blocksize-Y constraint
```

Siehe auch

Section 9.2.1, ST_AddBand, ST_MakeEmptyRaster, DropRasterConstraints, ST_BandPixelType, ST_SRID

10.2.2 DropRasterConstraints

DropRasterConstraints — Löscht die Constraints eines PostGIS Rasters die sich auf eine Rastertabellenspalte beziehen. Nützlich um Daten erneut zu laden oder um die Daten einer Rasterspalte zu aktualisieren.

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=t
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=fa
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);

Beschreibung

Löscht die Constraints eines PostGIS Rasters die sich auf eine Rastertabellenspalte beziehen und mit AddRasterConstraints hinzugefügt wurden. Nützlich um weitere Daten zu laden oder um die Daten einer Rasterspalte zu aktualisieren. Dies ist nicht notwendig, wenn Sie eine Rastertabelle oder eine Rasterspalte löschen wollen.

Zum Löschen einer Rastertabelle benutzen Sie bitte Standard-SQL:

```
DROP TABLE mytable
```

Um nur eine Rasterspalte zu löschen und den Rest der Tabelle zu belassen, verwenden Sie bitte Standard-SQL

```
ALTER TABLE mytable DROP COLUMN rast
```

Die Tabelle verschwindet aus dem `raster_columns` Katalog, wenn die Tabelle oder die Spalte gelöscht wurde. Wenn allerdings nur die Constraints gelöscht werden, so wird die Rasterspalte weiterhin im `raster_columns` Katalog aufgeführt, aber es ist keine zusätzliche Information mehr vorhanden - abgesehen vom Spalten- und Tabellennamen.

Verfügbarkeit: 2.0.0

Beispiele

```
SELECT DropRasterConstraints ('myrasters','rast');
----RESULT output ---
t

-- verify change in raster_columns --
SELECT srid, scale_x, scale_y, blocksize_x, blocksize_y, num_bands, pixel_types,  ↩
    nodata_values
    FROM raster_columns
    WHERE r_table_name = 'myrasters';

 srid | scale_x | scale_y | blocksize_x | blocksize_y | num_bands | pixel_types|  ↩
     nodata_values
------+---------+---------+-------------+-------------+-----------+-------------+--------------- ↩

    0 |         |         |             |             |           |             |
```

Siehe auch

AddRasterConstraints

10.2.3 AddOverviewConstraints

AddOverviewConstraints — Eine Rasterspalte als Übersicht für eine andere Rasterspalte kennzeichnen.

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);

Beschreibung

Fügt Constraints zu der Rasterspalte hinzu. Diese werden verwendet um die Information im `raster_overviews` Katalog anzuzeigen.

Der Parameter `ovfactorr` gibt den Multiplikator für den Maßstab der Übersichtsspalte an: ein höherer "overview factor" bedingt eine niedrigere Auflösung.

Falls die Parameter `ovschema` und `refschema` weggelassen werden, so wird die erste so benannte Tabelle verwendet, die beim Durchlaufen des `search_path` gefunden wird.

Verfügbarkeit: 2.0.0

Beispiele

```
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)
```

Siehe auch

Section 9.2.2, DropOverviewConstraints, ST_CreateOverview, AddRasterConstraints

10.2.4 DropOverviewConstraints

DropOverviewConstraints — Löscht die Markierung einer Rasterspalte, die festlegt dass sie als Übersicht für eine andere Spalte dient.

Synopsis

boolean **DropOverviewConstraints**(name ovschema, name ovtable, name ovcolumn);
boolean **DropOverviewConstraints**(name ovtable, name ovcolumn);

Beschreibung

Jene Constraints einer Rasterspalte löschen, die sie als Übersicht einer anderen Rasterspalte in dem Rasterkatalog `raster_overviews` ausweisen.

Falls der Parameter `ovschema` weggelassen wird, so wird die erste so benannte Tabelle verwendet, die beim Durchlaufen des `search_path` gefunden wird.

Verfügbarkeit: 2.0.0

Siehe auch

Section 9.2.2, AddOverviewConstraints, DropRasterConstraints

10.2.5 PostGIS_GDAL_Version

PostGIS_GDAL_Version — Gibt die von PostGIS verwendete Version der GDAL-Bibliothek aus.

Synopsis

text **PostGIS_GDAL_Version**();

Beschreibung

Gibt die von PostGIS verwendete Version der GDAL-Bibliothek aus. Überprüft und meldet, ob GDAL die zugehörigen Dateien findet.

Beispiele

```
SELECT PostGIS_GDAL_Version();
       postgis_gdal_version
-----------------------------------
 GDAL 1.11dev, released 2013/04/13
```

Siehe auch

postgis.gdal_datapath

10.2.6 PostGIS_Raster_Lib_Build_Date

PostGIS_Raster_Lib_Build_Date — Gibt einen vollständigen Bericht aus, wann die Rasterbibliothek kompiliert wurde.

Synopsis

text **PostGIS_Raster_Lib_Build_Date**();

Beschreibung

Gibt einen Bericht aus, wann die Rasterbibliothek kompiliert wurde.

Beispiele

```
SELECT PostGIS_Raster_Lib_Build_Date();
postgis_raster_lib_build_date
-----------------------------
2010-04-28 21:15:10
```

Siehe auch

PostGIS_Raster_Lib_Version

10.2.7 PostGIS_Raster_Lib_Version

PostGIS_Raster_Lib_Version — Gibt einen vollständigen Bericht über die Version und das Kompilationsdatum der Rasterbibliothek aus.

Synopsis

text **PostGIS_Raster_Lib_Version**();

Beschreibung

Gibt einen vollständigen Bericht über die Version und das Kompilationsdatum der Rasterbibliothek aus.

Beispiele

```
SELECT PostGIS_Raster_Lib_Version();
postgis_raster_lib_version
-----------------------------
 2.0.0
```

Siehe auch

PostGIS_Lib_Version

10.2.8 ST_GDALDrivers

ST_GDALDrivers — Gibt eine Liste der Rasterformate aus, die von PostGIS über die Bibliothek GDAL unterstützt werden. Nur die Formate mit can_write=True können von ST_AsGDALRaster verwendet werden.

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);

Beschreibung

Gibt eine Liste der Rasterformate aus - short_name, long_name und die Optionen des Urhebers - die von GDAL unterstützt werden. Verwenden Sie den "short_name" als Übergabewert für den Parameter `format` vonST_AsGDALRaster. Die Optionen variieren, je nach dem mit welchen Treibern Ihre "libgdal" kompiliert wurde. `create_options`gibt einen XML-formatierten Satz an CreationOptionList/Option zurück, der aus dem Namen und optional aus `type`, `description`und `VALUE` für jede Option eines bestimmten Treibers besteht.

Änderung: 2.5.0 - die Spalten can_read und can_write hinzugefügt.

Änderung: 2.0.6, 2.1.3 - standardmäßig ist kein Treiber aktiviert, solange die GUC oder die Umgebungsvariable "gdal_enabled_drivers" nicht gesetzt sind.

Verfügbarkeit: 2.0.0 - GDAL >= 1.6.0.

Beispiele: Treiber-Liste

```
SET postgis.gdal_enabled_drivers = 'ENABLE_ALL';
SELECT short_name, long_name, can_write
FROM st_gdaldrivers()
ORDER BY short_name;

   short_name    |                          long_name                          | can_write
-----------------+-------------------------------------------------------------+-----------
 AAIGrid         | Arc/Info ASCII Grid                                         | t
 ACE2            | ACE2                                                        | f
 ADRG            | ARC Digitized Raster Graphics                               | f
 AIG             | Arc/Info Binary Grid                                        | f
 AirSAR          | AirSAR Polarimetric Image                                   | f
 ARG             | Azavea Raster Grid format                                   | t
 BAG             | Bathymetry Attributed Grid                                  | f
 BIGGIF          | Graphics Interchange Format (.gif)                          | f
 BLX             | Magellan topo (.blx)                                        | t
 BMP             | MS Windows Device Independent Bitmap                        | f
 BSB             | Maptech BSB Nautical Charts                                 | f
 PAux            | PCI .aux Labelled                                           | f
 PCIDSK          | PCIDSK Database File                                        | f
 PCRaster        | PCRaster Raster File                                        | f
 PDF             | Geospatial PDF                                              | f
 PDS             | NASA Planetary Data System                                  | f
 PDS4            | NASA Planetary Data System 4                                | t
 PLMOSAIC        | Planet Labs Mosaics API                                     | f
 PLSCENES        | Planet Labs Scenes API                                      | f
 PNG             | Portable Network Graphics                                   | t
 PNM             | Portable Pixmap Format (netpbm)                             | f
 PRF             | Racurs PHOTOMOD PRF                                         | f
 R               | R Object Data Store                                         | t
 Rasterlite      | Rasterlite                                                  | t
 RDA             | DigitalGlobe Raster Data Access driver                      | f
 RIK             | Swedish Grid RIK (.rik)                                     | f
 RMF             | Raster Matrix Format                                        | f
 ROI_PAC         | ROI_PAC raster                                              | f
 RPFTOC          | Raster Product Format TOC format                            | f
 RRASTER         | R Raster                                                    | f
 RS2             | RadarSat 2 XML Product                                      | f
```

```
 RST             | Idrisi Raster A.1                                           | t
 SAFE            | Sentinel-1 SAR SAFE Product                                 | f
 SAGA            | SAGA GIS Binary Grid (.sdat, .sg-grd-z)                     | t
 SAR_CEOS        | CEOS SAR Image                                              | f
 SDTS            | SDTS Raster                                                 | f
 SENTINEL2       | Sentinel 2                                                  | f
 SGI             | SGI Image File Format 1.0                                   | f
 SNODAS          | Snow Data Assimilation System                               | f
 SRP             | Standard Raster Product (ASRP/USRP)                         | f
 SRTMHGT         | SRTMHGT File Format                                         | t
 Terragen        | Terragen heightfield                                        | f
 TIL             | EarthWatch .TIL                                             | f
 TSX             | TerraSAR-X Product                                          | f
 USGSDEM         | USGS Optional ASCII DEM (and CDED)                          | t
 VICAR           | MIPL VICAR file                                             | f
 VRT             | Virtual Raster                                              | t
 WCS             | OGC Web Coverage Service                                    | f
 WMS             | OGC Web Map Service                                         | t
 WMTS            | OGC Web Map Tile Service                                    | t
 XPM             | X11 PixMap Format                                           | t
 XYZ             | ASCII Gridded XYZ                                           | t
 ZMap            | ZMap Plus Grid                                              | t
```

Beispiele: Liste mit den Optionen für jeden Treiber

```
-- Output the create options XML column of JPEG as a table  --
-- Note you can use these creator options in ST_AsGDALRaster options argument
SELECT (xpath('@name', g.opt))[1]::text As oname,
       (xpath('@type', g.opt))[1]::text As otype,
       (xpath('@description', g.opt))[1]::text As descrip
FROM (SELECT unnest(xpath('/CreationOptionList/Option', create_options::xml)) As opt
FROM  st_gdaldrivers()
WHERE short_name = 'JPEG') As g;

       oname        |  otype  |      descrip
--------------------+---------+-------------------------------------------------
 PROGRESSIVE        | boolean | whether to generate a progressive JPEG
 QUALITY            | int     | good=100, bad=0, default=75
 WORLDFILE          | boolean | whether to geneate a worldfile
 INTERNAL_MASK      | boolean | whether to generate a validity mask
 COMMENT            | string  | Comment
 SOURCE_ICC_PROFILE | string  | ICC profile encoded in Base64
 EXIF_THUMBNAIL     | boolean | whether to generate an EXIF thumbnail(overview).
                                By default its max dimension will be 128
 THUMBNAIL_WIDTH    | int     | Forced thumbnail width
 THUMBNAIL_HEIGHT   | int     | Forced thumbnail height
(9 rows)
```

```
-- raw xml output for creator options for GeoTiff --
SELECT create_options
FROM st_gdaldrivers()
WHERE short_name = 'GTiff';

<CreationOptionList>
    <Option name="COMPRESS" type="string-select">
        <Value
>NONE</Value>
        <Value
>LZW</Value>
        <Value
```

```
>PACKBITS</Value>
        <Value
>JPEG</Value>
        <Value
>CCITTRLE</Value>
        <Value
>CCITTFAX3</Value>
        <Value
>CCITTFAX4</Value>
        <Value
>DEFLATE</Value>
    </Option>
    <Option name="PREDICTOR" type="int" description="Predictor Type"/>
    <Option name="JPEG_QUALITY" type="int" description="JPEG quality 1-100" default="75"/>
    <Option name="ZLEVEL" type="int" description="DEFLATE compression level 1-9" default ↩
        ="6"/>
    <Option name="NBITS" type="int" description="BITS for sub-byte files (1-7), sub-uint16  ↩
        (9-15), sub-uint32 (17-31)"/>
    <Option name="INTERLEAVE" type="string-select" default="PIXEL">
        <Value
>BAND</Value>
        <Value
>PIXEL</Value>
    </Option>
    <Option name="TILED" type="boolean" description="Switch to tiled format"/>
    <Option name="TFW" type="boolean" description="Write out world file"/>
    <Option name="RPB" type="boolean" description="Write out .RPB (RPC) file"/>
    <Option name="BLOCKXSIZE" type="int" description="Tile Width"/>
    <Option name="BLOCKYSIZE" type="int" description="Tile/Strip Height"/>
    <Option name="PHOTOMETRIC" type="string-select">
        <Value
>MINISBLACK</Value>
        <Value
>MINISWHITE</Value>
        <Value
>PALETTE</Value>
        <Value
>RGB</Value>
        <Value
>CMYK</Value>
        <Value
>YCBCR</Value>
        <Value
>CIELAB</Value>
        <Value
>ICCLAB</Value>
        <Value
>ITULAB</Value>
    </Option>
    <Option name="SPARSE_OK" type="boolean" description="Can newly created files have  ↩
        missing blocks?" default="FALSE"/>
    <Option name="ALPHA" type="boolean" description="Mark first extrasample as being alpha ↩
        "/>
    <Option name="PROFILE" type="string-select" default="GDALGeoTIFF">
        <Value
>GDALGeoTIFF</Value>
        <Value
>GeoTIFF</Value>
        <Value
>BASELINE</Value>
    </Option>
    <Option name="PIXELTYPE" type="string-select">
```

```
        <Value
>DEFAULT</Value>
        <Value
>SIGNEDBYTE</Value>
    </Option>
    <Option name="BIGTIFF" type="string-select" description="Force creation of BigTIFF file ↩
        ">
        <Value
>YES</Value>
        <Value
>NO</Value>
        <Value
>IF_NEEDED</Value>
        <Value
>IF_SAFER</Value>
    </Option>
    <Option name="ENDIANNESS" type="string-select" default="NATIVE" description="Force  ↩
        endianness of created file. For DEBUG purpose mostly">
        <Value
>NATIVE</Value>
        <Value
>INVERTED</Value>
        <Value
>LITTLE</Value>
        <Value
>BIG</Value>
    </Option>
    <Option name="COPY_SRC_OVERVIEWS" type="boolean" default="NO" description="Force copy  ↩
        of overviews of source dataset (CreateCopy())"/>
</CreationOptionList
>

-- Output the create options XML column for GTiff as a table  --
SELECT (xpath('@name', g.opt))[1]::text As oname,
       (xpath('@type', g.opt))[1]::text As otype,
       (xpath('@description', g.opt))[1]::text As descrip,
       array_to_string(xpath('Value/text()', g.opt),', ') As vals
FROM (SELECT unnest(xpath('/CreationOptionList/Option', create_options::xml)) As opt
FROM  st_gdaldrivers()
WHERE short_name = 'GTiff') As g;

       oname        |     otype     |                              descrip  ↩
                                          |                                   vals
--------------------+---------------+-------------------------------------------------------------

 COMPRESS           | string-select |  ↩
                                                                          | NONE, LZW,  ↩
     PACKBITS, JPEG, CCITTRLE, CCITTFAX3, CCITTFAX4, DEFLATE
 PREDICTOR          | int           | Predictor Type  ↩
                                                          |
 JPEG_QUALITY       | int           | JPEG quality 1-100  ↩
                                                       |
 ZLEVEL             | int           | DEFLATE compression level 1-9  ↩
                                             |
 NBITS              | int           | BITS for sub-byte files (1-7), sub-uint16 (9-15), sub ↩
     -uint32 (17-31) |
 INTERLEAVE         | string-select |  ↩
                                                                          | BAND, PIXEL
 TILED              | boolean       | Switch to tiled format  ↩
                                                   |
 TFW                | boolean       | Write out world file  ↩
                                                     |
```

```
 RPB                | boolean       | Write out .RPB (RPC) file  ↩
                                                                 |
 BLOCKXSIZE         | int           | Tile Width  ↩
                                                                      |
 BLOCKYSIZE         | int           | Tile/Strip Height  ↩
                                                                     |
 PHOTOMETRIC        | string-select |  ↩
                                                                                  | MINISBLACK,  ↩
    MINISWHITE, PALETTE, RGB, CMYK, YCBCR, CIELAB, ICCLAB, ITULAB
 SPARSE_OK          | boolean       | Can newly created files have missing blocks?  ↩
                                |
 ALPHA              | boolean       | Mark first extrasample as being alpha  ↩
                                    |
 PROFILE            | string-select |  ↩
                                                                                  | GDALGeoTIFF,  ↩
    GeoTIFF, BASELINE
 PIXELTYPE          | string-select |  ↩
                                                                                  | DEFAULT,  ↩
    SIGNEDBYTE
 BIGTIFF            | string-select | Force creation of BigTIFF file  ↩
                                          | YES, NO, IF_NEEDED, IF_SAFER
 ENDIANNESS         | string-select | Force endianness of created file. For DEBUG purpose  ↩
    mostly            | NATIVE, INVERTED, LITTLE, BIG
 COPY_SRC_OVERVIEWS | boolean       | Force copy of overviews of source dataset (CreateCopy ↩
    ())             |
(19 rows)
```

Siehe auch

ST_AsGDALRaster, ST_SRID, postgis.gdal_enabled_drivers

10.2.9 ST_Contour

ST_Contour — Generates a set of vector contours from the provided raster band, using the GDAL contouring algorithm.

Synopsis

setof record **ST_Contour**(raster rast, integer bandnumber, double precision level_interval, double precision level_base, double precision[] fixed_levels, boolean polygonize);

Beschreibung

Generates a set of vector contours from the provided raster band, using the GDAL contouring algorithm.

When the `fixed_levels` parameter is a non-empty array, the `level_interval` and `level_base` parameters are ignored.

Return values are a set of records with the following attributes:

geom The geometry of the contour line.

id A unique identifier given to the contour line by GDAL.

value The raster value the line represents. For an elevation DEM input, this would be the elevation of the output contour.

Availability: 3.2.0

Beispiel

```
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;
```

Siehe auch

ST_InterpolateRaster

10.2.10 ST_InterpolateRaster

ST_InterpolateRaster — Interpolates a gridded surface based on an input set of 3-d points, using the X- and Y-values to position the points on the grid and the Z-value of the points as the surface elevation.

Synopsis

raster **ST_InterpolateRaster**(geometry input_points, text algorithm_options, raster template, integer template_band_num=1);

Beschreibung

Interpolates a gridded surface based on an input set of 3-d points, using the X- and Y-values to position the points on the grid and the Z-value of the points as the surface elevation. There are five interpolation algorithms available: inverse distance, inverse distance nearest-neighbor, moving average, nearest neighbor, and linear interpolation. See the gdal_grid documentation for more details on the algorithms and their parameters. For more information on how interpolations are calculated, see the GDAL grid tutorial.

Input parameters are:

input_points The points to drive the interpolation. Any geometry with Z-values is acceptable, all points in the input will be used.

algorithm_options A string defining the algorithm and algorithm options, in the format used by gdal_grid. For example, for an inverse-distance interpolation with a smoothing of 2, you would use "invdist:smoothing=2.0"

template A raster template to drive the geometry of the output raster. The width, height, pixel size, spatial extent and pixel type will be read from this template.

template_band_num By default the first band in the template raster is used to drive the output raster, but that can be adjusted with this parameter.

Availability: 3.2.0

Beispiel

```
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')
)
```

Siehe auch

ST_Contour

10.2.11 UpdateRasterSRID

UpdateRasterSRID — Änderung der SRID aller Raster in der vom Anwender angegebenen Spalte und Tabelle.

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);

Beschreibung

Änderung der SRID aller Raster in der vom Anwender angegebenen Spalte und Tabelle. Diese Funktion löscht alle entsprechenden Constraints (extent, alignment und die SRID) der Spalte bevor die SRID der Rasterspalte geändert wird.

Note
Die Daten des Rasters (Pixelwerte der Bänder) bleiben von dieser Funktion unangetastet. Es werden lediglich die Metadaten des Rasters geändert.

Verfügbarkeit: 2.1.0

Siehe auch

UpdateGeometrySRID

10.2.12 ST_CreateOverview

ST_CreateOverview — Erzeugt eine Version des gegebenen Raster-Coverage mit geringerer Auflösung.

Synopsis

regclass **ST_CreateOverview**(regclass tab, name col, int factor, text algo='NearestNeighbor');

Beschreibung

Erzeugt eine Übersichtstabelle mit skalierten Kacheln der Ursprungstabelle. Die Ausgabekacheln haben dieselbe Größe und haben die gleiche räumliche Ausdehnung wie die Eingabekacheln mit einer niedrigeren Auflösung (die Pixelgröße ist in beiden Richtungen 1/`factor` der Ursprünglichen).

Auf die Übersichtstabelle werden die Constraints gesetzt und sie steht über den Katalog `raster_overviews` zur Verfügung.

Die Optionen für den Algorithmus sind: 'NearestNeighbor', 'Bilinear', 'Cubic', 'CubicSpline' und 'Lanczos'. Siehe GDAL Warp resampling methods für weitere Details.

Verfügbarkeit: 2.2.0

Beispiel

Ausgabe mit besserer Qualität, aber langsamerer Formaterstellung

```
SELECT ST_CreateOverview('mydata.mytable'::regclass, 'rast', 2, 'Lanczos');
```

Schnellere Berechnung der Ausgabe mittels "Nearest Neighbor" (Standardeinstellung)

```
SELECT ST_CreateOverview('mydata.mytable'::regclass, 'rast', 2);
```

Siehe auch

ST_Retile, AddOverviewConstraints, AddRasterConstraints, Section 9.2.2

10.3 Raster Constructors

10.3.1 ST_AddBand

ST_AddBand — Gibt einen Raster mit den neu hinzugefügten Band(Bändern) aus. Der Typ , der Ausgangswert und der Index für den Speicherort des Bandes kann angegeben werden. Wenn kein Index angegeben ist, wird das Band am Ende hinzugefügt.

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);

Beschreibung

Gibt einen Raster mit einem an der gegebenen Position (index) neu hinzugefügten Band zurück. aus. Der Typ, der Ausgangswert und der Wert für NODATA kann übergeben werden. Wenn kein Index angegeben ist, wird das Band am Ende hinzugefügt. Wenn `fromband` nicht angegeben ist, wird Band 1 angenommen. Der Pixeltyp wird als Zeichenfolge übergeben, wie in ST_BandPixelType festgelegt. Falls ein bereits bestehender Index angegeben wird, werden alle folgenden Bänder >= diesem Index um 1 erhöht. Wenn ein größerer Ausgangswert als das Maximum des Pixeltyps angegeben ist, dann wird der Ausgangswert auf den höchsten erlaubten Wert des Pixeltyps gesetzt.

Bei der Variante, welche ein Feld von addbandarg entgegennimmt (Variante 1), ist ein bestimmter Indexwert von "addbandarg" auf den Raster zu dem Zeitpunkt bezogen, als das Band mit diesem "addbandarg" zum Raster hinzugefügt wurde. Siehe das Beispiel "Mehrere neue Bänder" unterhalb.

Bei der Variante, die ein Feld an Rastern (Variante 5) entgegennimmt, wird wenn `torast` NULL ist, das `fromband` Band eines jeden Rasters in diesem Feld, in einem neuen Raster akkumuliert.

Bei den Varianten, die ein `outdbfile` (Varianten 6 und 7) entgegennehmen, muss der Wert den vollständigen Pfad der Rasterdatei enthalten. Die Datei muss auch für die PostgreSQL-Instanz zugänglich sein.

Erweiterung: 2.1.0 - Unterstützung für "addbandarg" hinzugefügt.

Erweiterung: 2.1.0 Unterstützung für die neuen "out-db" Bänder hinzugefügt.

Beispiele: Ein einzelnes, neues Band

```
-- Add another band of type 8 bit unsigned integer with pixels initialized to 200
UPDATE dummy_rast
    SET rast = ST_AddBand(rast,'8BUI'::text,200)
WHERE rid = 1;
```

```
-- Create an empty raster 100x100 units, with upper left  right at 0, add 2 bands (band 1  ↩
    is 0/1 boolean bit switch, band2 allows values 0-15)
-- uses addbandargs
INSERT INTO dummy_rast(rid,rast)
    VALUES(10, ST_AddBand(ST_MakeEmptyRaster(100, 100, 0, 0, 1, -1, 0, 0, 0),
    ARRAY[
        ROW(1, '1BB'::text, 0, NULL),
        ROW(2, '4BUI'::text, 0, NULL)
            ]::addbandarg[]
     )
    );

-- output meta data of raster bands to verify all is right --
SELECT  (bmd).*
FROM (SELECT ST_BandMetaData(rast,generate_series(1,2)) As bmd
    FROM dummy_rast WHERE rid = 10) AS foo;
 --result --
 pixeltype | nodatavalue | isoutdb | path
-----------+-------------+---------+------
 1BB       |             | f       |
 4BUI      |             | f       |


-- output meta data of raster -
SELECT  (rmd).width, (rmd).height, (rmd).numbands
FROM (SELECT ST_MetaData(rast) As rmd
    FROM dummy_rast WHERE rid = 10) AS foo;
-- result --
 upperleftx | upperlefty | width | height | scalex | scaley | skewx | skewy | srid |  ↩
     numbands
------------+------------+-------+--------+--------+--------+-------+-------+------+--------

          0 |          0 |   100 |    100 |      1 |     -1 |     0 |     0 |   0 |         ↩
              2
```

Beispiele: Mehrere neue Bänder

```
SELECT
    *
FROM ST_BandMetadata(
    ST_AddBand(
        ST_MakeEmptyRaster(10, 10, 0, 0, 1, -1, 0, 0, 0),
        ARRAY[
            ROW(NULL, '8BUI', 255, 0),
            ROW(NULL, '16BUI', 1, 2),
            ROW(2, '32BUI', 100, 12),
            ROW(2, '32BF', 3.14, -1)
        ]::addbandarg[]
    ),
    ARRAY[]::integer[]
);
```

```
 bandnum | pixeltype | nodatavalue | isoutdb | path
---------+-----------+-------------+---------+------
       1 | 8BUI      |           0 | f       |
       2 | 32BF      |          -1 | f       |
       3 | 32BUI     |          12 | f       |
       4 | 16BUI     |           2 | f       |
```

```
-- Aggregate the 1st band of a table of like rasters into a single raster
-- with as many bands as there are test_types and as many rows (new rasters) as there are  ↩
    mice
-- NOTE: The ORDER BY test_type is only supported in PostgreSQL 9.0+
-- for 8.4 and below it usually works to order your data in a subselect (but not guaranteed ↩
    )
-- The resulting raster will have a band for each test_type alphabetical by test_type
-- For mouse lovers: No mice were harmed in this exercise
SELECT
    mouse,
    ST_AddBand(NULL, array_agg(rast ORDER BY test_type), 1) As rast
FROM mice_studies
GROUP BY mouse;
```

Beispiele: Neues Out-db Band

```
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
```

Siehe auch

ST_BandMetaData, ST_BandPixelType, ST_MakeEmptyRaster, ST_MetaData, ST_NumBands, ST_Reclass

10.3.2 ST_AsRaster

ST_AsRaster — Konvertiert den geometrischen Datentyp von PostGIS in einen PostGIS Raster.

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);

Beschreibung

Konvertiert den geometrischen Datentyp von PostGIS in einen PostGIS Raster. Es gibt viele Varianten, die jeweils drei Gruppen von Möglichkeiten bieten um die Ausrichtung/alignment und die Pixelgröße des resultierenden Rasters zu bestimmen.

Die erste Gruppe besteht aus den ersten zwei Varianten und erzeugt einen Raster mit derselben Ausrichtung (`scalex`, `scaley`, `gridx` und `gridy`), dem selben Pixeltyp und NODATA Wert wie der gegebene Referenzraster. Üblicherweise wird der Referenzraster über einen Join übergeben, der aus der Tabelle mit der Geometrie und der Tabelle mit dem Referenzraster gebildet wird.

Die zweite Gruppe setzt sich aus vier Varianten zusammen und erlaubt Ihnen, die Dimensionen des Rasters über die Parameter der Pixelgröße festzulegen (`scalex` & `scaley` und `skewx` & `skewy`). Die `width` & `height` des resultierenden Rasters wird an die Ausdehnung der Geometrie angepasst. In den meisten Fällen müssen Sie eine Typumwandlung der Eingabewerte `scalex` & `scaley` von Integer nach Double Precision vornehmen, damit PostgreSQL die richtige Variante auswählt.

Die zweite Gruppe setzt sich aus vier Varianten zusammen und erlaubt Ihnen, die Dimensionen des Rasters über die Dimensionen des Rasters zu fixieren (`width` & `height`). Die Parameter der Pixelgröße (`scalex` & `scaley` und `skewx` & `skewy`). des resultierenden Rasters werden an die Ausdehnung der Geometrie angepasst.

Die ersten zwei Varianten der beiden letzten Gruppen erlauben es Ihnen, an einer beliebigen Ecke des Führungsgitters (`gridx`& `gridy`) auszurichten; und die letzten zwei Varianten nehmen die obere linke Ecke (`upperleftx` & `upperlefty`) entgegen.

Jede Variantengruppe erlaubt die Erstellung eines Rasters sowohl mit einem als auch mit mehreren Bändern. Um einen Raster mit mehreren Bändern zu erstellen, können Sie ein Feld mit Pixeltypen (`pixeltype[]`), ein Feld mit Ausgangswerten (`value`) und ein Feld mit NODATA Werten (`nodataval`) bereitstellen. Falls nicht, werden die Standardwerte - 8BUI für den Pixeltyp, 1 für den Ausgangswert und 0 für NODATA - angenommen.

Der Ausgaberaster hat dieselbe Koordinatenreferenz wie die Ausgangsgeometrie. Die einzige Ausnahme bilden Varianten mit einem Referenzraster. In diesem Fall erhält der resultierende Raster dieselbe SRID wie der Referenzraster.

Der optionale Parameter `touched` ist standardmäßig FALSE und wird auf die GDAL Rasterungsoption ALL_TOUCHED abgebildet, welche bestimmt, ob Pixel die von Linien oder Polygonen nur berührt werden, ebenfalls gerastert werden sollen; und nicht nur die Pixel auf einem Linienzug oder mit dem Mittelpunkt innerhalb des Polygons.

Dies ist insbesondere in Verbindung mit ST_AsPNG und der Funktionsfamilie ST_AsGDALRaster, für die Erstellung von JPEGs oder PNGs aus einer Geometrie direkt von der Datenbank heraus, nützlich.

Verfügbarkeit: 2.0.0 - GDAL >= 1.6.0.

Note
Zur Zeit können komplexe geometrische Datentypen wie Kurven, TINS und polyedrische Oberflächen nicht gerendert werden, was aber möglich sein sollte, sobald GDAL dies kann.

Beispiele: Ausgabe der Geometrien als PNG-Datei

Ein schwarzer Kreis

```
-- this will output a black circle taking up 150 x 150 pixels --
SELECT ST_AsPNG(ST_AsRaster(ST_Buffer(ST_Point(1,5),10),150, 150));
```

Ein Beispiel für einen Puffer; die Grafik ist direkt in PostGIS erstellt

```
-- the bands map to RGB bands - the value (118,154,118) - teal  --
SELECT ST_AsPNG(
    ST_AsRaster(
        ST_Buffer(
            ST_GeomFromText('LINESTRING(50 50,150 150,150 50)'), 10,'join=bevel'),
            200,200,ARRAY['8BUI', '8BUI', '8BUI'], ARRAY[118,154,118], ARRAY[0,0,0]));
```

Siehe auch

ST_BandPixelType, ST_Buffer, ST_GDALDrivers, ST_AsGDALRaster, ST_AsPNG, ST_AsJPEG, ST_SRID

10.3.3 ST_Band

ST_Band — Gibt einen oder mehrere Bänder eines bestehenden Rasters als neuen Raster aus. Nützlich um neue Raster aus bestehenden Rastern abzuleiten.

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=,);

Beschreibung

Gibt einen oder mehrere Bänder eines bestehenden Rasters in Form eines neuen Raster aus. Nützlich um neue Raster aus bestehenden Rastern abzuleiten, um einen Export auf ausgewählte Bänder einzuschränken, oder um die Reihenfolge der Rasterbänder neu zu ordnen. Wenn kein Band angegeben ist, oder keines der spezifizierten Bänder im Raster existiert, dann werden alle Bänder zurückgegeben. Dient auch als Hilfsfunktion für verschiedene Funktionen, wie das Löschen eines Bandes.

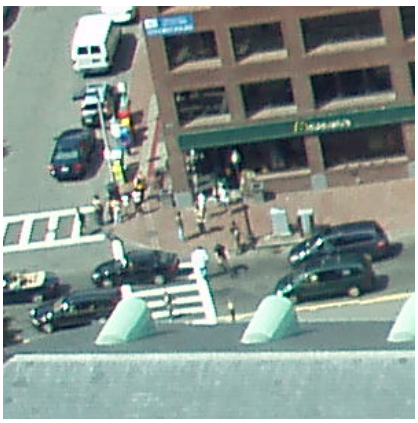
Warning
Bei der Funktionsvariante mit `nbands` als Text, ist das Standardtrennzeichen `,` ; d.h.: Sie können `'1,2,3'` abfragen und falls Sie ein anderes Trennzeichen verwenden wollen `ST_Band(rast, '1@2@3', '@')`. Wenn Sie mehrere Bänder abfragen, empfehlen wir Ihnen unbedingt die Feldvariante der Funktion zu verwenden, z.B. `ST_Band(rast, '{1,2,3}'::int[]);`, da die Variante mit der `text` Liste der Bänder in zukünftigen Versionen von PostGIS entfernt werden könnte.

Verfügbarkeit: 2.0.0

Beispiele

```
-- Make 2 new rasters: 1 containing band 1 of dummy, second containing band 2 of dummy and  ↩
    then reclassified as a 2BUI
SELECT ST_NumBands(rast1) As numb1, ST_BandPixelType(rast1) As pix1,
 ST_NumBands(rast2) As numb2,  ST_BandPixelType(rast2) As pix2
FROM (
    SELECT ST_Band(rast) As rast1, ST_Reclass(ST_Band(rast,3), '100-200):1, [200-254:2', '2 ↩
        BUI') As rast2
        FROM dummy_rast
        WHERE rid = 2) As foo;

 numb1 | pix1 | numb2 | pix2
-------+------+-------+------
     1 | 8BUI |     1 | 2BUI
```

```
-- Return bands 2 and 3. Using array cast syntax
SELECT ST_NumBands(ST_Band(rast, '{2,3}'::int[])) As num_bands
    FROM dummy_rast WHERE rid=2;

num_bands
----------
2

-- Return bands 2 and 3. Use array to define bands
SELECT ST_NumBands(ST_Band(rast, ARRAY[2,3])) As num_bands
    FROM dummy_rast
WHERE rid=2;
```

		
Original (COLUMN rast)	*dupe_band*	*sing_band*

```
--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;
```

Siehe auch

ST_AddBand, ST_NumBands, ST_Reclass, Chapter 10

10.3.4 ST_MakeEmptyCoverage

ST_MakeEmptyCoverage — Bedeckt die georeferenzierte Fläche mit einem Gitter aus leeren Rasterkacheln.

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);

Beschreibung

Erzeugt Rasterkacheln mit ST_MakeEmptyRaster. Die Größe des Gitters wird über `width` & `height` angegeben. Die Kachelgröße über `tilewidth` & `tileheight`. Die abgedeckte georeferenzierte Fläche reicht vom oberen linken Eck (`upperleftx`, `upperlefty`) bis zum unteren rechten Eck (`upperleftx` + `width` * `scalex`, `upperlefty` + `height` * `scaley`).

Note
Beachten Sie bitte, dass bei Rastern "scaley" im Allgemeinen negativ und "scalex" positiv ist. Dadurch hat das untere rechte Eck einen niedrigeren Y-Wert und einen höheren X-Wert als die obere rechte Ecke.

Verfügbarkeit: 2.4.0

Grundlegende Beispiele

Erzeugt ein 4x4 Gitter aus 16 Kacheln, um die WGS84 Fläche vom linken oberen Eck (22, 77) zum rechten unteren Eck (55, 33) abzudecken.

```
SELECT (ST_MetaData(tile)).* FROM ST_MakeEmptyCoverage(1, 1, 4, 4, 22, 33, (55 - 22)/(4):: ↩
    float, (33 - 77)/(4)::float, 0., 0., 4326) tile;

 upperleftx | upperlefty | width | height | scalex | scaley | skewx | skewy | srid |  ↩
     numbands
------------------------------------------------------------------------------------------
         22 |         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
```

Siehe auch

ST_MakeEmptyRaster

10.3.5 ST_MakeEmptyRaster

ST_MakeEmptyRaster — Gibt einen leeren Raster (ohne Bänder), mit den gegebenen Dimensionen (width & height), upperleft X und Y, Pixelgröße, Rotation (scalex, scaley, skewx & skewy) und Koordinatenreferenzsystem (SRID), zurück. Wenn ein Raster übergeben wird, dann wird ein neuer Raster mit der selben Größe, Ausrichtung und SRID zurückgegeben. Wenn SRID nicht angegeben ist, wird das Koordinatenreferenzsystem auf "unknown" (0) gesetzt.

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);

Beschreibung

Gibt einen leeren Raster (ohne Bänder), mit den gegebenen Dimensionen (width & height), georeferenziert in geodätischen (oder geographischen) Koordinaten, oberes linkes X (upperleftx), oberes linkes Y (upperlefty), Pixelgröße, Rotation (scalex, scaley, skewx & skewy) und Koordinatenreferenzsystem (SRID), zurück.

Die letzte Version verwendet einen einzelnen Parameter um die Pixelgröße festzulegen. "scalex" wird auf diesen Übergabewert und "scaley" auf den negativen Wert davon gesetzt. "skewx" und "skewy" werden auf 0 gesetzt.

Wird ein bestehender Raster übergeben, so wird ein neuer Raster mit den gleichen Metadateneinstellungen erstellt (ohne die Bänder).

Wenn die SRID nicht angegeben ist, wird sie standardmäßig auf 0 gesetzt. Nachdem Sie einen leeren Raster erzeugt haben, werden Sie vermutlich Bänder hinzufügen und eventuell auch bearbeiten wollen. Siehe ST_AddBand um die Bänder festzulegen und ST_SetValue um Ausgangswerte für die Pixel zu setzen.

Beispiele

```
INSERT INTO dummy_rast(rid,rast)
VALUES(3, ST_MakeEmptyRaster( 100, 100, 0.0005, 0.0005, 1, 1, 0, 0, 4326) );

--use an existing raster as template for new raster
INSERT INTO dummy_rast(rid,rast)
SELECT 4, ST_MakeEmptyRaster(rast)
FROM dummy_rast WHERE rid = 3;

-- output meta data of rasters we just added
SELECT rid, (md).*
FROM (SELECT rid, ST_MetaData(rast) As md
    FROM dummy_rast
    WHERE rid IN(3,4)) As foo;

-- output --
 rid | upperleftx | upperlefty | width | height | scalex | scaley | skewx | skewy | srid |  ↩
     numbands
-----+------------+------------+-------+--------+------------+------------+-------+-------+------+--
   3 |     0.0005 |     0.0005 |   100 |    100 |          1 |          1 |    0  |     0 | ↩
        4326 |        0
   4 |     0.0005 |     0.0005 |   100 |    100 |          1 |          1 |    0  |     0 | ↩
        4326 |        0
```

Siehe auch

ST_AddBand, ST_MetaData, ST_ScaleX, ST_ScaleY, ST_SetValue, ST_SkewX, , ST_SkewY

10.3.6 ST_Tile

ST_Tile — Gibt Raster, die aus einer Teilungsoperation des Eingaberasters resultieren, mit den gewünschten Dimensionen aus.

Synopsis

setof raster **ST_Tile**(raster rast, int[] nband, integer width, integer height, boolean padwithnodata=FALSE, double precision nodataval=NULL);
setof raster **ST_Tile**(raster rast, integer nband, integer width, integer height, boolean padwithnodata=FALSE, double precision nodataval=NULL);
setof raster **ST_Tile**(raster rast, integer width, integer height, boolean padwithnodata=FALSE, double precision nodataval=NULL);

Beschreibung

Gibt Raster, die aus einer Teilungsoperation des Eingaberasters resultieren, mit den gewünschten Dimensionen aus.

Wenn `padwithnodata` = FALSE, dann können die Eckkacheln auf der rechten Seite und auf der unteren Seite andere Dimensionen als die übrigen Kacheln aufweisen. Wenn `padwithnodata` = TRUE, dann haben alle Kacheln dieselbe Dimension und die Eckkacheln werden eventuell mit NODATA Werten aufgefüllt. Wenn für die Rasterbänder keine NODATA Werte festgelegt sind, können Sie diese mit `nodataval` spezifizieren.

Note
Wenn das Band des Eingaberasters "out-of-db" ist, dann ist das entsprechende Band des Ausgaberasters auch "out-of-db".

Verfügbarkeit: 2.1.0

Beispiele

```
WITH foo AS (
    SELECT ST_AddBand(ST_AddBand(ST_MakeEmptyRaster(3, 3, 0, 0, 1, -1, 0, 0, 0), 1, '8BUI', ↩
         1, 0), 2, '8BUI', 10, 0) AS rast UNION ALL
    SELECT ST_AddBand(ST_AddBand(ST_MakeEmptyRaster(3, 3, 3, 0, 1, -1, 0, 0, 0), 1, '8BUI', ↩
         2, 0), 2, '8BUI', 20, 0) AS rast UNION ALL
    SELECT ST_AddBand(ST_AddBand(ST_MakeEmptyRaster(3, 3, 6, 0, 1, -1, 0, 0, 0), 1, '8BUI', ↩
         3, 0), 2, '8BUI', 30, 0) AS rast UNION ALL

    SELECT ST_AddBand(ST_AddBand(ST_MakeEmptyRaster(3, 3, 0, -3, 1, -1, 0, 0, 0), 1, '8BUI ↩
        ', 4, 0), 2, '8BUI', 40, 0) AS rast UNION ALL
    SELECT ST_AddBand(ST_AddBand(ST_MakeEmptyRaster(3, 3, 3, -3, 1, -1, 0, 0, 0), 1, '8BUI ↩
        ', 5, 0), 2, '8BUI', 50, 0) AS rast UNION ALL
    SELECT ST_AddBand(ST_AddBand(ST_MakeEmptyRaster(3, 3, 6, -3, 1, -1, 0, 0, 0), 1, '8BUI ↩
        ', 6, 0), 2, '8BUI', 60, 0) AS rast UNION ALL

    SELECT ST_AddBand(ST_AddBand(ST_MakeEmptyRaster(3, 3, 0, -6, 1, -1, 0, 0, 0), 1, '8BUI ↩
        ', 7, 0), 2, '8BUI', 70, 0) AS rast UNION ALL
    SELECT ST_AddBand(ST_AddBand(ST_MakeEmptyRaster(3, 3, 3, -6, 1, -1, 0, 0, 0), 1, '8BUI ↩
        ', 8, 0), 2, '8BUI', 80, 0) AS rast UNION ALL
    SELECT ST_AddBand(ST_AddBand(ST_MakeEmptyRaster(3, 3, 6, -6, 1, -1, 0, 0, 0), 1, '8BUI ↩
        ', 9, 0), 2, '8BUI', 90, 0) AS rast
), bar AS (
    SELECT ST_Union(rast) AS rast FROM foo
), baz AS (
    SELECT ST_Tile(rast, 3, 3, TRUE) AS rast FROM bar
)
SELECT
    ST_DumpValues(rast)
FROM baz;

              st_dumpvalues
-----------------------------------------
```

```
 (1,"{{1,1,1},{1,1,1},{1,1,1}}")
 (2,"{{10,10,10},{10,10,10},{10,10,10}}")
 (1,"{{2,2,2},{2,2,2},{2,2,2}}")
 (2,"{{20,20,20},{20,20,20},{20,20,20}}")
 (1,"{{3,3,3},{3,3,3},{3,3,3}}")
 (2,"{{30,30,30},{30,30,30},{30,30,30}}")
 (1,"{{4,4,4},{4,4,4},{4,4,4}}")
 (2,"{{40,40,40},{40,40,40},{40,40,40}}")
 (1,"{{5,5,5},{5,5,5},{5,5,5}}")
 (2,"{{50,50,50},{50,50,50},{50,50,50}}")
 (1,"{{6,6,6},{6,6,6},{6,6,6}}")
 (2,"{{60,60,60},{60,60,60},{60,60,60}}")
 (1,"{{7,7,7},{7,7,7},{7,7,7}}")
 (2,"{{70,70,70},{70,70,70},{70,70,70}}")
 (1,"{{8,8,8},{8,8,8},{8,8,8}}")
 (2,"{{80,80,80},{80,80,80},{80,80,80}}")
 (1,"{{9,9,9},{9,9,9},{9,9,9}}")
 (2,"{{90,90,90},{90,90,90},{90,90,90}}")
(18 rows)
```

```
WITH foo AS (
    SELECT ST_AddBand(ST_AddBand(ST_MakeEmptyRaster(3, 3, 0, 0, 1, -1, 0, 0, 0), 1, '8BUI', ↩
        1, 0), 2, '8BUI', 10, 0) AS rast UNION ALL
    SELECT ST_AddBand(ST_AddBand(ST_MakeEmptyRaster(3, 3, 3, 0, 1, -1, 0, 0, 0), 1, '8BUI', ↩
        2, 0), 2, '8BUI', 20, 0) AS rast UNION ALL
    SELECT ST_AddBand(ST_AddBand(ST_MakeEmptyRaster(3, 3, 6, 0, 1, -1, 0, 0, 0), 1, '8BUI', ↩
        3, 0), 2, '8BUI', 30, 0) AS rast UNION ALL

    SELECT ST_AddBand(ST_AddBand(ST_MakeEmptyRaster(3, 3, 0, -3, 1, -1, 0, 0, 0), 1, '8BUI ↩
        ', 4, 0), 2, '8BUI', 40, 0) AS rast UNION ALL
    SELECT ST_AddBand(ST_AddBand(ST_MakeEmptyRaster(3, 3, 3, -3, 1, -1, 0, 0, 0), 1, '8BUI ↩
        ', 5, 0), 2, '8BUI', 50, 0) AS rast UNION ALL
    SELECT ST_AddBand(ST_AddBand(ST_MakeEmptyRaster(3, 3, 6, -3, 1, -1, 0, 0, 0), 1, '8BUI ↩
        ', 6, 0), 2, '8BUI', 60, 0) AS rast UNION ALL

    SELECT ST_AddBand(ST_AddBand(ST_MakeEmptyRaster(3, 3, 0, -6, 1, -1, 0, 0, 0), 1, '8BUI ↩
        ', 7, 0), 2, '8BUI', 70, 0) AS rast UNION ALL
    SELECT ST_AddBand(ST_AddBand(ST_MakeEmptyRaster(3, 3, 3, -6, 1, -1, 0, 0, 0), 1, '8BUI ↩
        ', 8, 0), 2, '8BUI', 80, 0) AS rast UNION ALL
    SELECT ST_AddBand(ST_AddBand(ST_MakeEmptyRaster(3, 3, 6, -6, 1, -1, 0, 0, 0), 1, '8BUI ↩
        ', 9, 0), 2, '8BUI', 90, 0) AS rast
), bar AS (
    SELECT ST_Union(rast) AS rast FROM foo
), baz AS (
    SELECT ST_Tile(rast, 3, 3, 2) AS rast FROM bar
)
SELECT
    ST_DumpValues(rast)
FROM baz;

              st_dumpvalues
----------------------------------------
 (1,"{{10,10,10},{10,10,10},{10,10,10}}")
 (1,"{{20,20,20},{20,20,20},{20,20,20}}")
 (1,"{{30,30,30},{30,30,30},{30,30,30}}")
 (1,"{{40,40,40},{40,40,40},{40,40,40}}")
 (1,"{{50,50,50},{50,50,50},{50,50,50}}")
 (1,"{{60,60,60},{60,60,60},{60,60,60}}")
 (1,"{{70,70,70},{70,70,70},{70,70,70}}")
 (1,"{{80,80,80},{80,80,80},{80,80,80}}")
 (1,"{{90,90,90},{90,90,90},{90,90,90}}")
```

```
(9 rows)
```

Siehe auch

ST_Union, ST_Retile

10.3.7 ST_Retile

ST_Retile — Gibt konfigurierte Kacheln eines beliebig gekachelten Rastercoverage aus.

Synopsis

SETOF raster **ST_Retile**(regclass tab, name col, geometry ext, float8 sfx, float8 sfy, int tw, int th, text algo='NearestNeighbor');

Beschreibung

Gibt die Kacheln in einem bestimmten Maßstab (`sfx`, `sfy`), maximaler Größe (`tw`, `th`) und räumlicher Ausdehnung (`ext`) mit den Daten des angegebenen Raster-Coverages (`tab`, `col`) zurück.

Die Optionen für den Algorithmus sind: 'NearestNeighbor', 'Bilinear', 'Cubic', 'CubicSpline' und 'Lanczos'. Siehe GDAL Warp resampling methods für weitere Details.

Verfügbarkeit: 2.2.0

Siehe auch

ST_CreateOverview

10.3.8 ST_FromGDALRaster

ST_FromGDALRaster — Erzeugt einen Raster aus einer von GDAL unterstützten Rasterdatei.

Synopsis

raster **ST_FromGDALRaster**(bytea gdaldata, integer srid=NULL);

Beschreibung

Erzeugt einen Raster aus einer von GDAL unterstützten Rasterdatei. `gdaldata` hat den Datentyp BYTEA und den Inhalt der GDAL Rasterdatei.

Wenn `srid` NULL ist, dann versucht die Funktion die SRID aus dem GDAL Raster automatisch zuzuweisen. Wenn `srid` angegeben ist, überschreibt dieser Werrt jede automatisch zugewiesene SRID.

Verfügbarkeit: 2.1.0

Beispiele

```
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)
```

Siehe auch

ST_AsGDALRaster

10.4 Zugriffsfunktionen auf Raster

10.4.1 ST_GeoReference

ST_GeoReference — Gibt die Metadaten der Georeferenzierung, die sich üblicherweise in einem sogenannten "World File befinden, im GDAL oder ESRI Format aus. Die Standardeinstellung ist GDAL.

Synopsis

text **ST_GeoReference**(raster rast, text format=GDAL);

Beschreibung

Gibt die Metadaten der Georeferenzierung, die sich üblicherweise in einem sogenannten "World File befinden, inklusive "Carriage Return" im GDAL oder ESRI Format aus. Die Standardeinstellung ist GDAL. Der Datentyp ist die Zeichenfolge 'GDAL' oder 'ESRI'.

Die Formate unterscheiden sich wie folgt:

GDAL:

```
scalex
skewy
skewx
scaley
upperleftx
upperlefty
```

ESRI:

```
scalex
skewy
skewx
scaley
upperleftx + scalex*0.5
upperlefty + scaley*0.5
```

Beispiele

```
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
```

Siehe auch

ST_SetGeoReference, ST_ScaleX, ST_ScaleY

10.4.2 ST_Height

ST_Height — Gibt die Höhe des Rasters in Pixel aus.

Synopsis

integer **ST_Height**(raster rast);

Beschreibung

Gibt die Höhe des Rasters aus.

Beispiele

```
SELECT rid, ST_Height(rast) As rastheight
FROM dummy_rast;

 rid | rastheight
-----+------------
   1 |         20
   2 |          5
```

Siehe auch

ST_Width

10.4.3 ST_IsEmpty

ST_IsEmpty — Gibt TRUE zurück, wenn der Raster leer ist (width = 0 and height = 0). Andernfalls wird FALSE zurückgegeben.

Synopsis

boolean **ST_IsEmpty**(raster rast);

Beschreibung

Gibt TRUE zurück, wenn der Raster leer ist (width = 0 and height = 0). Andernfalls wird FALSE zurückgegeben.

Verfügbarkeit: 2.0.0

Beispiele

```
SELECT ST_IsEmpty(ST_MakeEmptyRaster(100, 100, 0, 0, 0, 0, 0, 0))
st_isempty |
-----------+
f          |


SELECT ST_IsEmpty(ST_MakeEmptyRaster(0, 0, 0, 0, 0, 0, 0, 0))
st_isempty |
-----------+
t          |
```

Siehe auch

ST_HasNoBand

10.4.4 ST_MemSize

ST_MemSize — Gibt den Platzbedarf des Rasters (in Byte) aus.

Synopsis

integer **ST_MemSize**(raster rast);

Beschreibung

Gibt den Platzbedarf des Rasters (in Byte) aus.

Dies Funktion ist eine schöne Ergänzung zu den in PostgreSQL eingebauten Funktionen pg_column_size, pg_size_pretty, pg_relation_size und pg_total_relation_size.

Note

pg_relation_size, das die Größe einer Tabelle in Byte angibt kann niedrigere Werte liefern als ST_MemSize. Dies kann vorkommen, da pg_relation_size den TOAST-Speicher der Tabellen nicht mitrechnet und eine große Geometrie in TOAST-Tabellen gespeichert wird. pg_column_size kann einen niedrigeren Wert anzeigen, da es die komprimierte Dateigröße ausgibt.
pg_total_relation_size - schließt die Tabelle, die TOAST-Tabellen und di Indizes mit ein.

Verfügbarkeit: 2.2.0

Beispiele

```
SELECT ST_MemSize(ST_AsRaster(ST_Buffer(ST_Point(1,5),10,1000),150, 150, '8BUI')) As  ↩
    rast_mem;

        rast_mem
        --------
        22568
```

Siehe auch

10.4.5 ST_MetaData

ST_MetaData — Gibt die wesentlichen Metadaten eines Rasterobjektes, wie Zellgröße, Rotation (Versatz) etc. aus

Synopsis

record **ST_MetaData**(raster rast);

Beschreibung

Gibt die grundlegenden Metadaten eines Rasters aus, wie Pixelgröße, Rotation (skew), obere, untere linke, etc.. Die zurückgegebenen Spalten sind: upperleftx | upperlefty | width | height | scalex | scaley | skewx | skewy | srid | numbands

Beispiele

```
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 |  ↩
             0
   2 | 3427927.75 |    5793244 |     5 |      5 |   0.05 |   -0.05 |     0 |     0 |    0 |  ↩
             3
```

Siehe auch

ST_BandMetaData, ST_NumBands

10.4.6 ST_NumBands

ST_NumBands — Gibt die Anzahl der Bänder des Rasters aus.

Synopsis

integer **ST_NumBands**(raster rast);

Beschreibung

Gibt die Anzahl der Bänder des Rasters aus.

Beispiele

```
SELECT rid, ST_NumBands(rast) As numbands
FROM dummy_rast;

rid | numbands
----+----------
  1 |        0
  2 |        3
```

Siehe auch

ST_Value

10.4.7 ST_PixelHeight

ST_PixelHeight — Gibt die Pixelhöhe in den Einheiten des Koordinatenreferenzsystem aus.

Synopsis

double precision **ST_PixelHeight**(raster rast);

Beschreibung

Gibt die Höhe eines Pixels in den Einheiten des Koordinatenreferenzsystem aus. Beim üblichen Fall ohne Versatz/skew, entspricht die Pixelhöhe dem Maßstabsverhältnis zwischen den geometrischen Koordinaten und den Rasterpixeln.

Siehe ST_PixelWidth für eine schematische Darstellung der Verhältnisse.

Beispiele: Raster ohne Versatz/skew

```
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
```

Beispiele: Raster mit einem Versatz ungleich 0

```
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
```

Siehe auch

ST_PixelWidth, ST_ScaleX, ST_ScaleY, ST_SkewX, ST_SkewY

10.4.8 ST_PixelWidth

ST_PixelWidth — Gibt die Pixelbreite in den Einheiten des Koordinatenreferenzsystems aus.

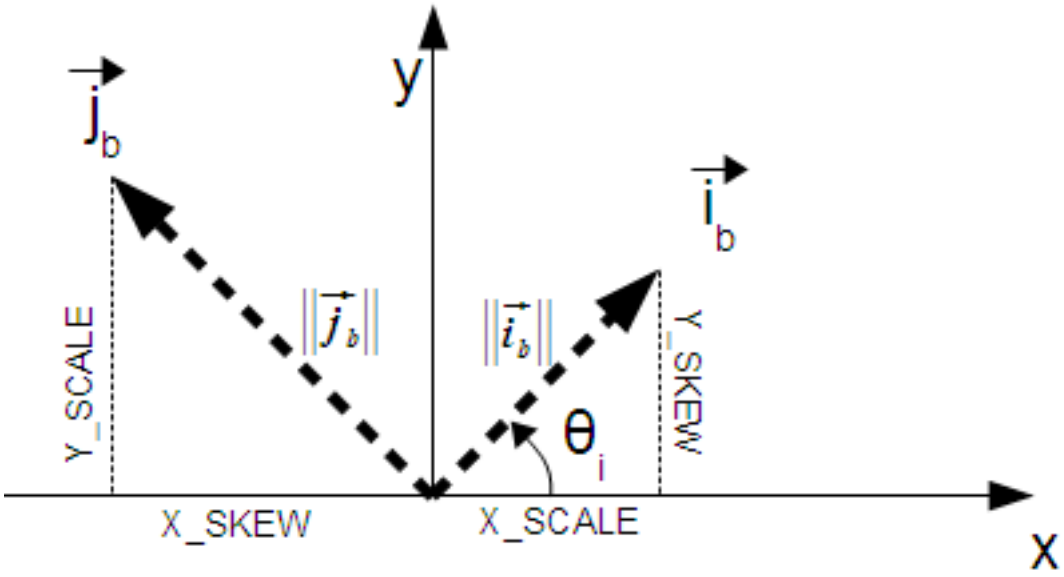
Synopsis

double precision **ST_PixelWidth**(raster rast);

Beschreibung

Gibt die Pixelbreite in den Einheiten des Koordinatenreferenzsystems aus. Beim üblichen Fall ohne Versatz entspricht die Pixelbreite dem Maßstabsverhältnis zwischen den geometrischen Koordinaten und den Rasterpixel.

Das folgende Schema zeigt die Beziehungen:



Pixelbreite: Pixelgröße in "i"-Richtung
Pixelhöhe: Pixelgröße in "j"-Richtung

Beispiele: Raster ohne Versatz/skew

```
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
```

Beispiele: Raster mit einem Versatz ungleich 0

```
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
```

Siehe auch

ST_PixelHeight, ST_ScaleX, ST_ScaleY, ST_SkewX, ST_SkewY

10.4.9 ST_ScaleX

ST_ScaleX — Gibt die X-Komponente der Pixelbreite in den Einheiten des Koordinatenreferenzsystems aus.

Synopsis

float8 **ST_ScaleX**(raster rast);

Beschreibung

Gibt die X-Komponente der Pixelbreite in den Einheiten des Koordinatenreferenzsystems aus. Siehe World File für weitere Details.

Änderung: 2.0.0 In WKTRaster Versionen wurde dies als ST_PixelSizeX bezeichnet.

Beispiele

```
SELECT rid, ST_ScaleX(rast) As rastpixwidth
FROM dummy_rast;

 rid | rastpixwidth
-----+--------------
   1 |            2
   2 |         0.05
```

Siehe auch

ST_Width

10.4.10 ST_ScaleY

ST_ScaleY — Gibt die Y-Komponente der Pixelhöhe in den Einheiten des Koordinatenreferenzsystems aus.

Synopsis

float8 **ST_ScaleY**(raster rast);

Beschreibung

Gibt die Y-Komponente der Pixelhöhe in den Einheiten des Koordinatenreferenzsystems aus. Kann negative Werte annehmen. Siehe World File für weitere Details.

Änderung: 2.0.0. Versionen von WKTRaster haben dies als "ST_PixelSizeY" bezeichnet.

Beispiele

```
SELECT rid, ST_ScaleY(rast) As rastpixheight
FROM dummy_rast;

 rid | rastpixheight
-----+---------------
   1 |             3
   2 |         -0.05
```

Siehe auch

ST_Height

10.4.11 ST_RasterToWorldCoord

ST_RasterToWorldCoord — Gibt die obere linke Ecke des Rasters in geodätischem X und Y (Länge und Breite) für eine gegebene Spalte und Zeile aus. Spalte und Zeile wird von 1 aufwärts gezählt.

Synopsis

record **ST_RasterToWorldCoord**(raster rast, integer xcolumn, integer yrow);

Beschreibung

Gibt die obere linke Ecke einer Spalte und Zeile als geometrisches X und Y (als geographische Länge und Breite) aus. Die zurückgegebenen X und Y sind in den Einheiten des georeferenzierten Rasters. Die Nummerierung der Spalten und Zeilen beginnt mit 1. Allerdings, wenn für einen der Parameter 0, eine negative Zahl, oder eine höhere Zahl als die betreffende Größe des Rasters übergeben wird, dann werden Koordinaten außerhalb des Rasters ausgegeben; dabei wird angenommen, dass das Gitter des Rasters auch außerhalb der Begrenzung des Rasters angewendet werden kann.

Verfügbarkeit: 2.1.0

Beispiele

```
-- non-skewed raster
SELECT
    rid,
    (ST_RasterToWorldCoord(rast,1, 1)).*,
    (ST_RasterToWorldCoord(rast,2, 2)).*
FROM dummy_rast

 rid | longitude  | latitude | longitude |  latitude
-----+------------+----------+-----------+------------
   1 |        0.5 |      0.5 |       2.5 |        3.5
   2 | 3427927.75 |  5793244 | 3427927.8 | 5793243.95
```

```
-- skewed raster
SELECT
    rid,
    (ST_RasterToWorldCoord(rast, 1, 1)).*,
    (ST_RasterToWorldCoord(rast, 2, 3)).*
FROM (
    SELECT
        rid,
        ST_SetSkew(rast, 100.5, 0) As rast
    FROM dummy_rast
) As foo

 rid | longitude  | latitude | longitude | latitude
-----+------------+----------+-----------+-----------
   1 |        0.5 |      0.5 |     203.5 |       6.5
   2 | 3427927.75 |  5793244 | 3428128.8 | 5793243.9
```

Siehe auch

ST_RasterToWorldCoordX, ST_RasterToWorldCoordY, ST_SetSkew

10.4.12 ST_RasterToWorldCoordX

ST_RasterToWorldCoordX — Gibt die geodätische X Koordinate links oberhalb des Rasters, der Spalte und der Zeile aus. Die Nummerierung der Spalten und Zeilen beginnt mit 1.

Synopsis

float8 **ST_RasterToWorldCoordX**(raster rast, integer xcolumn);
float8 **ST_RasterToWorldCoordX**(raster rast, integer xcolumn, integer yrow);

Beschreibung

Gibt die obere linke X-Koordinate einer Rasterzeile in den geometrischen Einheiten des georeferenzierten Rasters aus. Die Nummerierung der Spalten und Zeilen beginnt mit 1. Wenn Sie eine negative Zahl oder eine höhere Zahl als die Anzahl der Spalten des Rasters angeben, dann bekommen Sie die Koordinaten links außerhalb und rechts außerhalb des Rasters zurück; dabei wird angenommen, dass der Versatz und die Pixelgröße gleich wie bei dem ausgewählten Raster sind.

Note
Bei Rastern ohne Versatz, ist die Angabe der X-Spalte ausreichend. Bei Rastern mit Versatz ist die georeferenzierte Koordinate eine Funktion von "ST_ScaleX", "ST_SkewX", der Zeile und der Spalte. Wenn Sie bei einem Raster mit Versatz nur die X-Spalte angeben, dann wird eine Fehlermeldung ausgegeben.

Änderung: 2.1.0 Vorgängerversionen haben dies als "ST_Raster2WorldCoordX" bezeichnet.

Beispiele

```
-- non-skewed raster providing column is sufficient
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 lets skew it
SELECT rid, ST_RasterToWorldCoordX(rast, 1, 1) As x1coord,
    ST_RasterToWorldCoordX(rast, 2, 3) As x2coord,
    ST_ScaleX(rast) As pixelx
FROM (SELECT rid, ST_SetSkew(rast, 100.5, 0) As rast FROM dummy_rast) As foo;

 rid |  x1coord   |  x2coord  | pixelx
-----+------------+-----------+--------
   1 |        0.5 |     203.5 |      2
   2 | 3427927.75 | 3428128.8 |   0.05
```

Siehe auch

ST_ScaleX, ST_RasterToWorldCoordY, ST_SetSkew, ST_SkewX

10.4.13 ST_RasterToWorldCoordY

ST_RasterToWorldCoordY — Gibt die geodätische Y Koordinate links oberhalb des Rasters, der Spalte und der Zeile aus. Die Nummerierung der Spalten und Zeilen beginnt mit 1.

Synopsis

float8 **ST_RasterToWorldCoordY**(raster rast, integer yrow);
float8 **ST_RasterToWorldCoordY**(raster rast, integer xcolumn, integer yrow);

Beschreibung

Gibt die obere linke Y-Koordinate einer Rasterspalte in den Einheiten des georeferenzierten Rasters aus. Die Nummerierung der Spalten und Zeilen beginnt mit 1. Wenn Sie eine negative Zahl oder eine höhere Zahl als die Anzahl der Spalten/Zeilen des Rasters angeben, dann bekommen Sie die Koordinaten außerhalb des Rasters zurück; dabei wird angenommen, dass Versatz und Pixelgröße gleich wie bei der ausgewählten Rasterkachel sind.

Note
Bei Rastern ohne Versatz ist die Angabe der Y-Spalte ausreichend. Bei Rastern mit Versatz entspricht die georeferenzierte Koordinate einer Funktion von ST_ScaleY, ST_SkewY, Zeile und Spalte. Wenn Sie bei einem Raster mit Versatz nur die Y-Spalte angeben, wird eine Fehlermeldung ausgegeben.

Änderung: 2.1.0 In Vorgängerversionen wurde dies als ST_Raster2WorldCoordY bezeichnet

Beispiele

```
-- non-skewed raster providing row is sufficient
SELECT rid, ST_RasterToWorldCoordY(rast,1) As y1coord,
    ST_RasterToWorldCoordY(rast,3) As y2coord,
    ST_ScaleY(rast) As pixely
FROM dummy_rast;

 rid | y1coord |  y2coord  | pixely
-----+---------+-----------+--------
   1 |     0.5 |       6.5 |      3
   2 | 5793244 | 5793243.9 |  -0.05
```

```
-- for fun lets skew it
SELECT rid, ST_RasterToWorldCoordY(rast,1,1) As y1coord,
    ST_RasterToWorldCoordY(rast,2,3) As y2coord,
    ST_ScaleY(rast) As pixely
FROM (SELECT rid, ST_SetSkew(rast,0,100.5) As rast FROM dummy_rast) As foo;

 rid | y1coord |  y2coord  | pixely
-----+---------+-----------+--------
   1 |     0.5 |       107 |      3
   2 | 5793244 | 5793344.4 |  -0.05
```

Siehe auch

ST_ScaleY, ST_RasterToWorldCoordX, ST_SetSkew, ST_SkewY

10.4.14 ST_Rotation

ST_Rotation — Gibt die Rotation des Rasters im Bogenmaß aus.

Synopsis

float8 **ST_Rotation**(raster rast);

Beschreibung

Gibt die einheitliche Rotation des Rasters in Radiant aus. Wenn der Raster keine einheitliche Rotation aufweist wird NaN zurückgegeben. Siehe World File für weitere Details.

Beispiele

```
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
```

Siehe auch

ST_SetRotation, ST_SetScale, ST_SetSkew

10.4.15 ST_SkewX

ST_SkewX — Gibt den georeferenzierten Versatz in X-Richtung (oder den Rotationsparameter) aus.

Synopsis

float8 **ST_SkewX**(raster rast);

Beschreibung

Gibt den georeferenzierten Versatz in X-Richtung (oder den Rotationsparameter) aus. Siehe World File für weitere Details.

Beispiele

```
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
```

Siehe auch

ST_GeoReference, ST_SkewY, ST_SetSkew

10.4.16 ST_SkewY

ST_SkewY — Gibt den georeferenzierten Versatz in Y-Richtung (oder den Rotationsparameter) aus.

Synopsis

float8 **ST_SkewY**(raster rast);

Beschreibung

Gibt den georeferenzierten Versatz in Y-Richtung (oder den Rotationsparameter) aus. Siehe World File für weitere Details.

Beispiele

```
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
```

Siehe auch

ST_GeoReference, ST_SkewX, ST_SetSkew

10.4.17 ST_SRID

ST_SRID — Gibt den Identifikator des Koordinatenreferenzsystems des Rasters aus, das in der Tabelle "spatial_ref_sys" definiert ist.

Synopsis

integer **ST_SRID**(raster rast);

Beschreibung

Gibt den Identifikator des Koordinatenreferenzsystems des Rasters aus, das in der Tabelle "spatial_ref_sys" definiert ist.

Note
Ab PostGIS 2.0 ist die SRID eines nicht georeferenzierten Rasters/Geometrie 0 anstelle wie früher -1.

Beispiele

```
SELECT ST_SRID(rast) As srid
FROM dummy_rast WHERE rid=1;

srid
----------------
0
```

Siehe auch

Section 4.5, ST_SRID

10.4.18 ST_Summary

ST_Summary — Gibt eine textliche Zusammenfassung des Rasterinhalts zurück.

Synopsis

text **ST_Summary**(raster rast);

Beschreibung

Gibt eine textliche Zusammenfassung des Rasterinhalts zurück

Verfügbarkeit: 2.1.0

Beispiele

```
SELECT ST_Summary(
    ST_AddBand(
        ST_AddBand(
            ST_AddBand(
                ST_MakeEmptyRaster(10, 10, 0, 0, 1, -1, 0, 0, 0)
                , 1, '8BUI', 1, 0
            )
            , 2, '32BF', 0, -9999
        )
        , 3, '16BSI', 0, NULL
    )
);

                            st_summary
------------------------------------------------------------------
 Raster of 10x10 pixels has 3 bands and extent of BOX(0 -10,10 0)+
     band 1 of pixtype 8BUI is in-db with NODATA value of 0      +
     band 2 of pixtype 32BF is in-db with NODATA value of -9999  +
     band 3 of pixtype 16BSI is in-db with no NODATA value
(1 row)
```

Siehe auch

ST_MetaData, ST_BandMetaData, ST_Summary ST_Extent

10.4.19 ST_UpperLeftX

ST_UpperLeftX — Gibt die obere linke X-Koordinate des Rasters im Koordinatenprojektionssystem aus.

Synopsis

float8 **ST_UpperLeftX**(raster rast);

Beschreibung

Gibt die obere linke X-Koordinate des Rasters im Koordinatenprojektionssystem aus.

Beispiele

```
SELECt rid, ST_UpperLeftX(rast) As ulx
FROM dummy_rast;

 rid |    ulx
-----+------------
   1 |        0.5
   2 | 3427927.75
```

Siehe auch

ST_UpperLeftY, ST_GeoReference, Box3D

10.4.20 ST_UpperLeftY

ST_UpperLeftY — Gibt die obere linke Y-Koordinate des Rasters im Koordinatenprojektionssystem aus.

Synopsis

float8 **ST_UpperLeftY**(raster rast);

Beschreibung

Gibt die obere linke Y-Koordinate des Rasters im Koordinatenprojektionssystem aus.

Beispiele

```
SELECT rid, ST_UpperLeftY(rast) As uly
FROM dummy_rast;

 rid |   uly
-----+---------
   1 |     0.5
   2 | 5793244
```

Siehe auch

ST_UpperLeftX, ST_GeoReference, Box3D

10.4.21 ST_Width

ST_Width — Gibt die Breite des Rasters in Pixel aus.

Synopsis

integer **ST_Width**(raster rast);

Beschreibung

Gibt die Breite des Rasters in Pixel aus.

Beispiele

```
SELECT ST_Width(rast) As rastwidth
FROM dummy_rast WHERE rid=1;

rastwidth
----------------
10
```

Siehe auch

ST_Height

10.4.22 ST_WorldToRasterCoord

ST_WorldToRasterCoord — Gibt für ein geometrisches X und Y (geographische Länge und Breite) oder für eine Punktgeometrie im Koordinatenreferenzsystem des Rasters, die obere linke Ecke als Spalte und Zeile aus.

Synopsis

record **ST_WorldToRasterCoord**(raster rast, geometry pt);
record **ST_WorldToRasterCoord**(raster rast, double precision longitude, double precision latitude);

Beschreibung

Gibt für ein geometrisches X und Y (geographische Länge und Breite), oder für eine Punktgeometrie, die obere linke Ecke als Spalte und Zeile aus. Diese Funktion funktioniert auch dann, wenn sich X und Y, bzw. die Punktgeometrie außerhalb der Ausdehnung des Rasters befindet. X und Y müssen im Koordinatenreferenzsystem des Rasters angegeben werden.

Verfügbarkeit: 2.1.0

Beispiele

```
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
```

Siehe auch

ST_WorldToRasterCoordX, ST_WorldToRasterCoordY, ST_RasterToWorldCoordX, ST_RasterToWorldCoordY, ST_SRID

10.4.23 ST_WorldToRasterCoordX

ST_WorldToRasterCoordX — Gibt für eine Punktgeometrie (pt) oder eine globale X- und Y-Koordinate (xw, yw) die Rasterspalte im globalen Koordinatenreferenzsystem des Rasters aus.

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);

Beschreibung

Gibt für eine Punktgeometrie (pt) oder eine globale X- und Y-Koordinate (xw, yw) die Rasterspalte aus. Es werden ein Punkt oder sowohl "xw" als auch "yw" in globalen Koordinaten benötigt, wenn der Raster einen Versatz aufweist. Wenn der Raster keinen Versatz aufweist, ist "xw" ausreichend. Die globalen Koordinaten sind im Koordinatenreferenzsystem des Rasters.

Änderung: 2.1.0 In Vorgängerversionen wurde dies als ST_World2RasterCoordX bezeichnet

Beispiele

```
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
```

Siehe auch

ST_RasterToWorldCoordX, ST_RasterToWorldCoordY, ST_SRID

10.4.24 ST_WorldToRasterCoordY

ST_WorldToRasterCoordY — Gibt für eine Punktgeometrie (pt) oder eine globale X- und Y-Koordinate (xw, yw) die Rasterzeile im globalen Koordinatenreferenzsystem des Rasters aus.

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);

Beschreibung

Gibt für eine Punktgeometrie (pt) oder eine globale X- und Y-Koordinate (xw, yw) die Rasterzeile aus. Es werden ein Punkt oder sowohl "xw" als auch "yw" in globalen Koordinaten benötigt, wenn der Raster einen Versatz aufweist. Wenn der Raster keinen Versatz aufweist, ist "xw" ausreichend. Die globalen Koordinaten sind im Koordinatenreferenzsystem des Rasters.

Änderung: 2.1.0 In Vorgängerversionen wurde dies als ST_World2RasterCoordY bezeichnet

Beispiele

```
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
```

Siehe auch

ST_RasterToWorldCoordX, ST_RasterToWorldCoordY, ST_SRID

10.5 Zugriffsfunktionen auf Rasterbänder

10.5.1 ST_BandMetaData

ST_BandMetaData — Gibt die grundlegenden Metadaten eines bestimmten Rasterbandes aus. Wenn der Parameter "bandnum" nicht angegeben ist, wird das 1ste Band angenommen.

Synopsis

(1) record **ST_BandMetaData**(raster rast, integer band=1);
(2) record **ST_BandMetaData**(raster rast, integer[] band);

Beschreibung

Gibt die grundlegenden Metadaten eines Rasterbandes aus. Die zurückgegebenen Spalten sind pixeltype, nodatavalue, isoutdb, path, filesize und filetimestamp.

Note
Wenn der Raster keine Bänder beinhaltet wird ein Fehler gemeldet.

Note
Wenn für das Band kein NODATA-Wert vergeben wurde, wird der NODATA-Wert auf NULL gesetzt.

Note
Wenn isoutdb False ist, dann sind path, outdbbandnum, filesize und filetimestamp NULL. Wenn der Zugriff auf otdb-Raster deaktiviert ist, dann sind filesize und filetimestamp ebenfalls NULL.

Erweiterung: 2.5.0 inkludiert jetzt*outdbbandnum*, *filesize* und *filetimestamp* für outdb Raster.

Beispiele: Variante 1

```
SELECT
    rid,
    (foo.md).*
FROM (
    SELECT
        rid,
        ST_BandMetaData(rast, 1) AS md
    FROM dummy_rast
    WHERE rid=2
) As foo;

 rid | pixeltype | nodatavalue | isoutdb | path | outdbbandnum
-----+-----------+---- --------+---------+------+--------------
   2 | 8BUI      |           0 | f       |      |
```

Beispiele: Variante 2

```
WITH foo AS (
    SELECT
        ST_AddBand(NULL::raster, '/home/pele/devel/geo/postgis-git/raster/test/regress/ ↩
            loader/Projected.tif', NULL::int[]) AS rast
)
SELECT
    *
FROM ST_BandMetadata(
    (SELECT rast FROM foo),
    ARRAY[1,3,2]::int[]
);

 bandnum | pixeltype | nodatavalue | isoutdb |                                        path  ↩
                                          | outdbbandnum  | filesize | filetimestamp |
---------+-----------+-------------+---------+-----------------------------------------------------

       1 | 8BUI      |             | t       | /home/pele/devel/geo/postgis-git/raster/test ↩
           /regress/loader/Projected.tif      |            1  |    12345 |    1521807257 |
       3 | 8BUI      |             | t       | /home/pele/devel/geo/postgis-git/raster/test ↩
           /regress/loader/Projected.tif      |            3  |    12345 |    1521807257 |
       2 | 8BUI      |             | t       | /home/pele/devel/geo/postgis-git/raster/test ↩
           /regress/loader/Projected.tif      |            2  |    12345 |    1521807257 |
```

Siehe auch

ST_MetaData, ST_BandPixelType

10.5.2 ST_BandNoDataValue

ST_BandNoDataValue — Gibt den NODATA Wert des gegebenen Bandes aus. Wenn der Parameter "bandnum" nicht angegeben ist, wird das 1ste Band angenommen.

Synopsis

double precision **ST_BandNoDataValue**(raster rast, integer bandnum=1);

Beschreibung

Gibt den NODATA Wert des Bandes aus.

Beispiele

```
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
```

Siehe auch

ST_NumBands

10.5.3 ST_BandIsNoData

ST_BandIsNoData — Gibt TRUE aus, wenn das Band ausschließlich aus NODATA Werten besteht.

Synopsis

boolean **ST_BandIsNoData**(raster rast, integer band, boolean forceChecking=true);
boolean **ST_BandIsNoData**(raster rast, boolean forceChecking=true);

Beschreibung

Gibt TRUE aus, wenn das Band ausschließlich aus NODATA Werten besteht. Wenn kein Band angegeben ist, wird Band 1 angenommen. Wenn der letzte Übergabewert TRUE ist, dann wird das gesamte Band, Pixel für Pixel, überprüft. Sonst gibt die Funktion nur den Wert der Flag "isnodata" für das Band aus. Wenn dieser Parameter nicht gesetzt wurde, wird der Standardwert "FALSE" angenommen.

Verfügbarkeit: 2.0.0

Note
Wenn die Flag geändert wurde (d.h.: das Ergebnis unterscheidet sich wenn man TRUE als letzten Parameter angibt, von jenem bei dem dieser Parameter nicht verwendet wird), sollten Sie den Raster aktualisieren um diese Flag auf TRUE zu setzen, indem Sie ST_SetBandIsNodata() anwenden, oder ST_SetBandNodataValue() mit TRUE als letzten Übergabewert ausführen. Siehe ST_SetBandIsNoData.

Beispiele

```
-- Create dummy table with one raster column
create table dummy_rast (rid integer, rast raster);

-- Add raster with two bands, one pixel/band. In the first band, nodatavalue = pixel value ↩
    = 3.
-- In the second band, nodatavalue = 13, pixel value = 4
insert into dummy_rast values(1,
```

```
(
'01' -- little endian (uint8 ndr)
||
'0000' -- version (uint16 0)
||
'0200' -- nBands (uint16 0)
||
'17263529ED684A3F' -- scaleX (float64 0.000805965234044584)
||
'F9253529ED684ABF' -- scaleY (float64 -0.00080596523404458)
||
'1C9F33CE69E352C0' -- ipX (float64 -75.5533328537098)
||
'718F0E9A27A44840' -- ipY (float64 49.2824585505576)
||
'ED50EB853EC32B3F' -- skewX (float64 0.000211812383858707)
||
'7550EB853EC32B3F' -- skewY (float64 0.000211812383858704)
||
'E6100000' -- SRID (int32 4326)
||
'0100' -- width (uint16 1)
||
'0100' -- height (uint16 1)
||
'6' -- hasnodatavalue and isnodata value set to true.
||
'2' -- first band type (4BUI)
||
'03' -- novalue==3
||
'03' -- pixel(0,0)==3 (same that nodata)
||
'0' -- hasnodatavalue set to false
||
'5' -- second band type (16BSI)
||
'0D00' -- novalue==13
||
'0400' -- pixel(0,0)==4
)::raster
);

select st_bandisnodata(rast, 1) from dummy_rast where rid = 1; -- Expected true
select st_bandisnodata(rast, 2) from dummy_rast where rid = 1; -- Expected false
```

Siehe auch

ST_BandNoDataValue, ST_NumBands, ST_SetBandNoDataValue, ST_SetBandIsNoData

10.5.4 ST_BandPath

ST_BandPath — Gibt den Dateipfad aus, unter dem das Band im Dateisystem gespeichert ist. Wenn "bandnum" nicht angegeben ist, wird 1 angenommen.

Synopsis

text **ST_BandPath**(raster rast, integer bandnum=1);

Beschreibung

Gibt den Dateipfad des Bandes im System aus. Wenn man die Funktion mit einem "in-db"-Rasterband aufruft, wird eine Fehlermeldung ausgegeben.

Beispiele

Siehe auch

10.5.5 ST_BandFileSize

ST_BandFileSize — Gibt die Dateigröße eines im Dateisystem gespeicherten Bandes aus. Wenn "bandnum" nicht angegeben ist, wird 1 angenommen.

Synopsis

bigint **ST_BandFileSize**(raster rast, integer bandnum=1);

Beschreibung

Gibt die Dateigröße eines im Dateisystem gespeicherten Bandes aus. Wenn die Funktion mit einem indb-Band aufgerufen wird oder der outdb-Zugriff deaktiviert ist, wird eine Fehlermeldung ausgegeben.

Diese Funkction wird üblicherweise gemeinsam imit ST_BandPath() und ST_BandFileTimestamp() verwendet. Auf diese Weise kann ein Client feststellen, ob er die selbe Sicht auf den Dateinamen eines outdb-Rasters hat wie der Server.

Verfügbarkeit: 2.5.0

Beispiele

```
SELECT ST_BandFileSize(rast,1) FROM dummy_rast WHERE rid = 1;

 st_bandfilesize
-----------------
          240574
```

10.5.6 ST_BandFileTimestamp

ST_BandFileTimestamp — Gibt den Zeitstempel eines im Dateisystem gespeicherten Bandes aus. Wenn "bandnum" nicht angegeben ist, wird 1 angenommen.

Synopsis

bigint **ST_BandFileTimestamp**(raster rast, integer bandnum=1);

Beschreibung

Gibt den Zeitstempel (Anzahl der Sekunden seit Jan 1st 1970 00:00:00 UTC) eines im Dateisystem gespeicherten Bandes aus. Wenn die Funktion mit einem indb-Band aufgerufen wird oder der outdb-Zugriff deaktiviert ist, wird eine Fehlermeldung ausgegeben.

Diese Funkction wird üblicherweise gemeinsam imit ST_BandPath() und ST_BandFileSize() verwendet. Auf diese Weise kann ein Client feststellen, ob er die selbe Sicht auf den Dateinamen eines outdb-Rasters hat wie der Server.

Verfügbarkeit: 2.5.0

Beispiele

```
SELECT ST_BandFileTimestamp(rast,1) FROM dummy_rast WHERE rid = 1;

 st_bandfiletimestamp
----------------------
           1521807257
```

10.5.7 ST_BandPixelType

ST_BandPixelType — Gibt den Pixeltyp des angegebenen Bandes aus. Wenn der Parameter "bandnum" nicht angegeben ist, wird das 1ste Band angenommen.

Synopsis

text **ST_BandPixelType**(raster rast, integer bandnum=1);

Beschreibung

Gibt eine Beschreibung des Datentyps und der Größe der Zellwerte in dem gegebenen Band zurück.

Im Folgenden die 11 unterstützten Pixeltypen

- 1BB - 1-Bit Boolean
- 2BUI - 2-Bit vorzeichenlose Ganzzahl
- 4BUI - 4-Bit vorzeichenlose Ganzzahl
- 8BSI - 8-Bit Ganzzahl
- 8BUI - 8-Bit vorzeichenlose Ganzzahl
- 16BSI - 16-Bit Ganzzahl
- 16BUI - 16-bit vorzeichenlose Ganzzahl
- 32BSI - 32-Bit Ganzzahl
- 32BUI - 32-Bit vorzeichenlose Ganzzahl
- 32BF - 32-Bit Float
- 64BF - 64-Bit Float

Beispiele

```
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
```

Siehe auch

ST_NumBands

10.5.8 ST_MinPossibleValue

ST_MinPossibleValue — Returns the minimum value this pixeltype can store.

Synopsis

integer **ST_MinPossibleValue**(text pixeltype);

Beschreibung

Returns the minimum value this pixeltype can store.

Beispiele

```
SELECT ST_MinPossibleValue('16BSI');

 st_minpossiblevalue
---------------------
              -32768


SELECT ST_MinPossibleValue('8BUI');

 st_minpossiblevalue
---------------------
                   0
```

Siehe auch

ST_BandPixelType

10.5.9 ST_HasNoBand

ST_HasNoBand — Gibt TRUE aus, wenn kein Band mit der angegebenen Bandnummer existiert. Gibt den Pixeltyp des angegebenen Bandes aus. Wenn keine Bandnummer angegeben ist, wird das 1ste Band angenommen.

Synopsis

boolean **ST_HasNoBand**(raster rast, integer bandnum=1);

Beschreibung

Gibt TRUE aus, wenn kein Band mit der angegebenen Bandnummer existiert. Gibt den Pixeltyp des angegebenen Bandes aus. Wenn keine Bandnummer angegeben ist, wird das 1ste Band angenommen.

Verfügbarkeit: 2.0.0

Beispiele

```
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
```

Siehe auch

ST_NumBands

10.6 Zugriffsfunktionen und Änderungsmethoden für Rasterpixel

10.6.1 ST_PixelAsPolygon

ST_PixelAsPolygon — Gibt die Polygongeometrie aus, die das Pixel einer bestimmten Zeile und Spalte begrenzt.

Synopsis

geometry **ST_PixelAsPolygon**(raster rast, integer columnx, integer rowy);

Beschreibung

Gibt die Polygongeometrie aus, die das Pixel einer bestimmten Zeile und Spalte begrenzt.

Verfügbarkeit: 2.0.0

Beispiele

```
-- get raster pixel polygon
SELECT i,j, ST_AsText(ST_PixelAsPolygon(foo.rast, i,j)) As b1pgeom
FROM dummy_rast As foo
    CROSS JOIN generate_series(1,2) As i
    CROSS JOIN generate_series(1,1) As j
WHERE rid=2;

 i | j |                                                  b1pgeom
---+---+-----------------------------------------------------------------------------
 1 | 1 | POLYGON((3427927.75 5793244,3427927.8 5793244,3427927.8 5793243.95,...
 2 | 1 | POLYGON((3427927.8 5793244,3427927.85 5793244,3427927.85 5793243.95, ..
```

Siehe auch

ST_DumpAsPolygons, ST_PixelAsPolygons, ST_PixelAsPoint, ST_PixelAsPoints, ST_PixelAsCentroid, ST_PixelAsCentroids, ST_Intersection, ST_AsText

10.6.2 ST_PixelAsPolygons

ST_PixelAsPolygons — Gibt die umhüllende Polygongeometrie, den Zellwert, sowie die X- und Y-Rasterkoordinate für jedes Pixel aus.

Synopsis

setof record **ST_PixelAsPolygons**(raster rast, integer band=1, boolean exclude_nodata_value=TRUE);

Beschreibung

Gibt die umhüllende Polygongeometrie, den Zellwert (double precision), sowie die X- und Y-Rasterkoordinate (Ganzzahl) für jedes Pixel aus.

Datensatzformatausgabe: *geom* geometry, *val* double precision, *x* integer, *y* integers.

Note
Wenn *exclude_nodata_value* = TRUE, dann werden nur jene Pixel als Punkte zurückgegeben deren Zellwerte nicht NODATA sind.

Note
ST_PixelAsPolygons gibt eine Polygongeometrie pro Pixel zurück. Dies unterscheidet sich von ST_DumpAsPolygons, wo jede Geometrie einen oder mehrere Pixel mit gleichem Zellwert darstellt.

Verfügbarkeit: 2.0.0

Erweiterung: 2.1.0 Der optionale Übergabewert "exclude_nodata_value" wurde hinzugefügt.

Änderung: 2.1.1 Die Verhaltensweise von "exclude_nodata_value" wurde geändert.

Beispiele

```
-- get raster pixel polygon
SELECT (gv).x, (gv).y, (gv).val, ST_AsText((gv).geom) geom
FROM (SELECT ST_PixelAsPolygons(
                 ST_SetValue(ST_SetValue(ST_AddBand(ST_MakeEmptyRaster(2, 2, 0, 0, 0.001,  ↩
                     -0.001, 0.001, 0.001, 4269),
                                                 '8BUI'::text, 1, 0),
                                     2, 2, 10),
                             1, 1, NULL)
) gv
) foo;

 x | y | val |                 geom
---+---+-----+--------------------------------------------------------------------------
 1 | 1 |     | POLYGON((0 0,0.001 0.001,0.002 0,0.001 -0.001,0 0))
 1 | 2 |   1 | POLYGON((0.001 -0.001,0.002 0,0.003 -0.001,0.002 -0.002,0.001 -0.001))
 2 | 1 |   1 | POLYGON((0.001 0.001,0.002 0.002,0.003 0.001,0.002 0,0.001 0.001))
 2 | 2 |  10 | POLYGON((0.002 0,0.003 0.001,0.004 0,0.003 -0.001,0.002 0))
```

Siehe auch

ST_DumpAsPolygons, ST_PixelAsPolygon, ST_PixelAsPoint, ST_PixelAsPoints, ST_PixelAsCentroid, ST_PixelAsCentroids, ST_AsText

10.6.3 ST_PixelAsPoint

ST_PixelAsPoint — Gibt eine Punktgeometrie der oberen linken Ecke des Rasters zurück.

Synopsis

geometry **ST_PixelAsPoint**(raster rast, integer columnx, integer rowy);

Beschreibung

Gibt eine Punktgeometrie der oberen linken Ecke des Rasters zurück.

Verfügbarkeit: 2.1.0

Beispiele

```
SELECT ST_AsText(ST_PixelAsPoint(rast, 1, 1)) FROM dummy_rast WHERE rid = 1;

   st_astext
----------------
 POINT(0.5 0.5)
```

Siehe auch

ST_DumpAsPolygons, ST_PixelAsPolygon, ST_PixelAsPolygons, ST_PixelAsPoints, ST_PixelAsCentroid, ST_PixelAsCentroids

10.6.4 ST_PixelAsPoints

ST_PixelAsPoints — Gibt eine Punktgeometrie für jedes Pixel des Rasterbandes zurück, zusammen mit dem Zellwert und den X- und Y-Rasterkoordinaten eines jeden Pixels. Die Koordinaten der Punkte entsprechen dem oberen linken Eck der Pixel.

Synopsis

setof record **ST_PixelAsPoints**(raster rast, integer band=1, boolean exclude_nodata_value=TRUE);

Beschreibung

Gibt eine Punktgeometrie für jedes Pixel des Rasterbandes zurück, zusammen mit dem Zellwert und den X- und Y-Rasterkoordinaten eines jeden Pixels. Die Koordinaten der Punkte entsprechen dem oberen linken Eck der Pixel.

Datensatzformatausgabe: *geom* geometry, *val* double precision, *x* integer, *y* integers.

Note
Wenn *exclude_nodata_value* = TRUE, dann werden nur jene Pixel als Punkte zurückgegeben deren Zellwerte nicht NODATA sind.

Verfügbarkeit: 2.1.0

Änderung: 2.1.1 Die Verhaltensweise von "exclude_nodata_value" wurde geändert.

Beispiele

```
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)
```

Siehe auch

ST_DumpAsPolygons, ST_PixelAsPolygon, ST_PixelAsPolygons, ST_PixelAsPoint, ST_PixelAsCentroid, ST_PixelAsCentroids

10.6.5 ST_PixelAsCentroid

ST_PixelAsCentroid — Gibt den geometrischen Schwerpunkt (Punktgeometrie) der Fläche aus, die durch das Pixel repräsentiert wird.

Synopsis

geometry **ST_PixelAsCentroid**(raster rast, integer x, integer y);

Beschreibung

Gibt den geometrischen Schwerpunkt (Punktgeometrie) der Fläche aus, die durch das Pixel repräsentiert wird.

Enhanced: 3.2.0 Faster now implemented in C.

Verfügbarkeit: 2.1.0

Beispiele

```
SELECT ST_AsText(ST_PixelAsCentroid(rast, 1, 1)) FROM dummy_rast WHERE rid = 1;

  st_astext
--------------
 POINT(1.5 2)
```

Siehe auch

ST_DumpAsPolygons, ST_PixelAsPolygon, ST_PixelAsPolygons, ST_PixelAsPoint, ST_PixelAsPoints, ST_PixelAsCentroids

10.6.6 ST_PixelAsCentroids

ST_PixelAsCentroids — Gibt den geometrischen Schwerpunkt (Punktgeometrie) für jedes Pixel des Rasterbandes zurück, zusammen mit dem Zellwert und den X- und Y-Rasterkoordinaten eines jeden Pixels. Die Koordinaten der Punkte entsprechen dem geometrischen Schwerpunkt der Pixel.

Synopsis

setof record **ST_PixelAsCentroids**(raster rast, integer band=1, boolean exclude_nodata_value=TRUE);

Beschreibung

Gibt den geometrischen Schwerpunkt (Punktgeometrie) für jedes Pixel des Rasterbandes zurück, zusammen mit dem Zellwert und den X- und Y-Rasterkoordinaten eines jeden Pixels. Die Koordinaten der Punkte entsprechen dem geometrischen Schwerpunkt der Pixel.

Datensatzformatausgabe: *geom* geometry, *val* double precision, *x* integer, *y* integers.

Note
Wenn *exclude_nodata_value* = TRUE, dann werden nur jene Pixel als Punkte zurückgegeben deren Zellwerte nicht NODATA sind.

Enhanced: 3.2.0 Faster now implemented in C.

Änderung: 2.1.1 Die Verhaltensweise von "exclude_nodata_value" wurde geändert.

Verfügbarkeit: 2.1.0

Beispiele

```
--LATERAL syntax requires PostgreSQL 9.3+
SELECT x, y, val, ST_AsText(geom)
    FROM (SELECT dp.* FROM dummy_rast, LATERAL ST_PixelAsCentroids(rast, 1) AS dp WHERE rid ↩
        = 2) foo;
 x | y | val |           st_astext
---+---+-----+-------------------------------
 1 | 1 | 253 | POINT(3427927.775 5793243.975)
 2 | 1 | 254 | POINT(3427927.825 5793243.975)
 3 | 1 | 253 | POINT(3427927.875 5793243.975)
 4 | 1 | 254 | POINT(3427927.925 5793243.975)
 5 | 1 | 254 | POINT(3427927.975 5793243.975)
```

```
1 | 2 | 253 | POINT(3427927.775 5793243.925)
2 | 2 | 254 | POINT(3427927.825 5793243.925)
3 | 2 | 254 | POINT(3427927.875 5793243.925)
4 | 2 | 253 | POINT(3427927.925 5793243.925)
5 | 2 | 249 | POINT(3427927.975 5793243.925)
1 | 3 | 250 | POINT(3427927.775 5793243.875)
2 | 3 | 254 | POINT(3427927.825 5793243.875)
3 | 3 | 254 | POINT(3427927.875 5793243.875)
4 | 3 | 252 | POINT(3427927.925 5793243.875)
5 | 3 | 249 | POINT(3427927.975 5793243.875)
1 | 4 | 251 | POINT(3427927.775 5793243.825)
2 | 4 | 253 | POINT(3427927.825 5793243.825)
3 | 4 | 254 | POINT(3427927.875 5793243.825)
4 | 4 | 254 | POINT(3427927.925 5793243.825)
5 | 4 | 253 | POINT(3427927.975 5793243.825)
1 | 5 | 252 | POINT(3427927.775 5793243.775)
2 | 5 | 250 | POINT(3427927.825 5793243.775)
3 | 5 | 254 | POINT(3427927.875 5793243.775)
4 | 5 | 254 | POINT(3427927.925 5793243.775)
5 | 5 | 254 | POINT(3427927.975 5793243.775)
```

Siehe auch

ST_DumpAsPolygons, ST_PixelAsPolygon, ST_PixelAsPolygons, ST_PixelAsPoint, ST_PixelAsPoints, ST_PixelAsCentroid

10.6.7 ST_Value

ST_Value — Gibt den Zellwert eines Pixels aus, das über columnx und rowy oder durch einen bestimmten geometrischen Punkt angegeben wird. Die Bandnummern beginnen mit 1 und wenn keine Bandnummer angegeben ist, dann wird Band 1 angenommen. Wenn `exclude_nodata_value` auf FALSE gesetzt ist, werden auch die Pixel mit einem `nodata` Wert mit einbezogen. Wenn `exclude_nodata_value` nicht übergeben wird, dann wird er über die Metadaten des Rasters ausgelesen.

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);

Beschreibung

Returns the value of a given band in a given columnx, rowy pixel or at a given geometry point. Band numbers start at 1 and band is assumed to be 1 if not specified.

If `exclude_nodata_value` is set to true, then only non `nodata` pixels are considered. If `exclude_nodata_value` is set to false, then all pixels are considered.

The allowed values of the `resample` parameter are "nearest" which performs the default nearest-neighbor resampling, and "bilinear" which performs a bilinear interpolation to estimate the value between pixel centers.

Enhanced: 3.2.0 resample optional argument was added.

Erweiterung: 2.0.0 Der optionale Übergabewert "exclude_nodata_value" wurde hinzugefügt.

Beispiele

```
-- get raster values at particular postgis geometry points
-- the srid of your geometry should be same as for your raster
SELECT rid, ST_Value(rast, foo.pt_geom) As b1pval, ST_Value(rast, 2, foo.pt_geom) As b2pval
FROM dummy_rast CROSS JOIN (SELECT ST_SetSRID(ST_Point(3427927.77, 5793243.76), 0) As ↩
    pt_geom) As foo
WHERE rid=2;

 rid | b1pval | b2pval
-----+--------+--------
   2 |    252 |     79


-- general fictitious example using a real table
SELECT rid, ST_Value(rast, 3, sometable.geom) As b3pval
FROM sometable
WHERE ST_Intersects(rast,sometable.geom);
```

```
SELECT rid, ST_Value(rast, 1, 1, 1) As b1pval,
    ST_Value(rast, 2, 1, 1) As b2pval, ST_Value(rast, 3, 1, 1) As b3pval
FROM dummy_rast
WHERE rid=2;

 rid | b1pval | b2pval | b3pval
-----+--------+--------+--------
   2 |    253 |     78 |     70
```

```
--- Get all values in bands 1,2,3 of each pixel --
SELECT x, y, ST_Value(rast, 1, x, y) As b1val,
    ST_Value(rast, 2, x, y) As b2val, ST_Value(rast, 3, x, y) As b3val
FROM dummy_rast CROSS JOIN
generate_series(1, 1000) As x CROSS JOIN generate_series(1, 1000) As y
WHERE rid =  2 AND x <= ST_Width(rast) AND y <= ST_Height(rast);

 x | y | b1val | b2val | b3val
---+---+-------+-------+-------
 1 | 1 |   253 |    78 |    70
 1 | 2 |   253 |    96 |    80
 1 | 3 |   250 |    99 |    90
 1 | 4 |   251 |    89 |    77
 1 | 5 |   252 |    79 |    62
 2 | 1 |   254 |    98 |    86
 2 | 2 |   254 |   118 |   108
 :
 :
```

```
--- Get all values in bands 1,2,3 of each pixel same as above but returning the upper left  ↩
   point point of each pixel --
SELECT ST_AsText(ST_SetSRID(
    ST_Point(ST_UpperLeftX(rast) + ST_ScaleX(rast)*x,
        ST_UpperLeftY(rast) + ST_ScaleY(rast)*y),
        ST_SRID(rast))) As uplpt
    , ST_Value(rast, 1, x, y) As b1val,
    ST_Value(rast, 2, x, y) As b2val, ST_Value(rast, 3, x, y) As b3val
FROM dummy_rast CROSS JOIN
generate_series(1,1000) As x CROSS JOIN generate_series(1,1000) As y
WHERE rid =  2 AND x <= ST_Width(rast) AND y <= ST_Height(rast);

            uplpt            | b1val | b2val | b3val
-----------------------------+-------+-------+-------
```

```
 POINT(3427929.25 5793245.5) |   253 |    78 |    70
 POINT(3427929.25 5793247)   |   253 |    96 |    80
 POINT(3427929.25 5793248.5) |   250 |    99 |    90
:
```

```
--- Get a polygon formed by union of all pixels
    that fall in a particular value range and intersect particular polygon --
SELECT ST_AsText(ST_Union(pixpolyg)) As shadow
FROM (SELECT ST_Translate(ST_MakeEnvelope(
        ST_UpperLeftX(rast), ST_UpperLeftY(rast),
            ST_UpperLeftX(rast) + ST_ScaleX(rast),
            ST_UpperLeftY(rast) + ST_ScaleY(rast), 0
            ), ST_ScaleX(rast)*x, ST_ScaleY(rast)*y
        ) As pixpolyg, ST_Value(rast, 2, x, y) As b2val
    FROM dummy_rast CROSS JOIN
generate_series(1,1000) As x CROSS JOIN generate_series(1,1000) As y
WHERE rid =  2
    AND x <= ST_Width(rast) AND y <= ST_Height(rast)) As foo
WHERE
    ST_Intersects(
        pixpolyg,
        ST_GeomFromText('POLYGON((3427928 5793244,3427927.75 5793243.75,3427928  ↩
            5793243.75,3427928 5793244))',0)
        ) AND b2val != 254;


        shadow
------------------------------------------------------------------------------
 MULTIPOLYGON(((3427928 5793243.9,3427928 5793243.85,3427927.95 5793243.85,3427927.95  ↩
     5793243.9,
 3427927.95 5793243.95,3427928 5793243.95,3427928.05 5793243.95,3427928.05  ↩
     5793243.9,3427928 5793243.9)),((3427927.95 5793243.9,3427927.95 579324
3.85,3427927.9 5793243.85,3427927.85 5793243.85,3427927.85 5793243.9,3427927.9  ↩
    5793243.9,3427927.9 5793243.95,
3427927.95 5793243.95,3427927.95 5793243.9)),((3427927.85 5793243.75,3427927.85  ↩
    5793243.7,3427927.8 5793243.7,3427927.8 5793243.75
,3427927.8 5793243.8,3427927.8 5793243.85,3427927.85 5793243.85,3427927.85  ↩
    5793243.8,3427927.85 5793243.75)),
((3427928.05 5793243.75,3427928.05 5793243.7,3427928 5793243.7,3427927.95  ↩
    5793243.7,3427927.95 5793243.75,3427927.95 5793243.8,3427
927.95 5793243.85,3427928 5793243.85,3427928 5793243.8,3427928.05 5793243.8,
3427928.05 5793243.75)),((3427927.95 5793243.75,3427927.95 5793243.7,3427927.9  ↩
    5793243.7,3427927.85 5793243.7,
3427927.85 5793243.75,3427927.85 5793243.8,3427927.85 5793243.85,3427927.9 5793243.85,
3427927.95 5793243.85,3427927.95 5793243.8,3427927.95 5793243.75)))
```

```
--- Checking all the pixels of a large raster tile can take a long time.
--- You can dramatically improve speed at some lose of precision by orders of magnitude
--  by sampling pixels using the step optional parameter of generate_series.
--  This next example does the same as previous but by checking 1 for every 4 (2x2) pixels  ↩
    and putting in the last checked
--  putting in the checked pixel as the value for subsequent 4

SELECT ST_AsText(ST_Union(pixpolyg)) As shadow
FROM (SELECT ST_Translate(ST_MakeEnvelope(
        ST_UpperLeftX(rast), ST_UpperLeftY(rast),
            ST_UpperLeftX(rast) + ST_ScaleX(rast)*2,
            ST_UpperLeftY(rast) + ST_ScaleY(rast)*2, 0
            ), ST_ScaleX(rast)*x, ST_ScaleY(rast)*y
        ) As pixpolyg, ST_Value(rast, 2, x, y) As b2val
    FROM dummy_rast CROSS JOIN
```

```
generate_series(1,1000,2) As x CROSS JOIN generate_series(1,1000,2) As y
WHERE rid =  2
    AND x <= ST_Width(rast)  AND y <= ST_Height(rast)  ) As foo
WHERE
    ST_Intersects(
        pixpolyg,
        ST_GeomFromText('POLYGON((3427928 5793244,3427927.75 5793243.75,3427928  ↩
            5793243.75,3427928 5793244))',0)
        ) AND b2val != 254;

        shadow
-------------------------------------------------------------------------------
 MULTIPOLYGON(((3427927.9 5793243.85,3427927.8 5793243.85,3427927.8 5793243.95,
 3427927.9 5793243.95,3427928 5793243.95,3427928.1 5793243.95,3427928.1 5793243.85,3427928  ↩
    5793243.85,3427927.9 5793243.85)),
 ((3427927.9 5793243.65,3427927.8 5793243.65,3427927.8 5793243.75,3427927.8  ↩
    5793243.85,3427927.9 5793243.85,
 3427928 5793243.85,3427928 5793243.75,3427928.1 5793243.75,3427928.1 5793243.65,3427928  ↩
    5793243.65,3427927.9 5793243.65)))
```

Siehe auch

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

10.6.8 ST_NearestValue

ST_NearestValue — Gibt den nächstgelegenen nicht NODATA Wert eines bestimmten Pixels aus, das über "columnx" und "rowy" oder durch eine Punktgeometrie - im gleichen Koordinatenreferenzsystem wie der Raster - ausgewählt wird.

Synopsis

double precision **ST_NearestValue**(raster rast, integer bandnum, geometry pt, boolean exclude_nodata_value=true);
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);

Beschreibung

Gibt den nächstgelegenen, nicht-NODATA Wert eines bestimmten Pixels aus, das über "columnx" und "rowy", oder durch einen geometrischen Punkt angegeben wird. Falls das angegebene Pixel den Wert NODATA hat, findet die Funktion das nächstgelegene Pixel, das nicht den Wert NODATA hat.

Die Bandnummern beginnen mit 1 und wenn keine Bandnummer angegeben ist, dann wird für bandnum 1 angenommen. Wenn exclude_nodata_value auf FALSE gesetzt ist, werden auch die Pixel mit einem nodata Wert einbezogen. Wenn exclude_nodata_value nicht übergeben wird, dann wird er über die Metadaten des Rasters ausgelesen.

Verfügbarkeit: 2.1.0

Note
ST_NearestValue ist eine Alternative zu ST_Value.

Beispiele

```
-- pixel 2x2 has value
SELECT
    ST_Value(rast, 2, 2) AS value,
    ST_NearestValue(rast, 2, 2) AS nearestvalue
FROM (
    SELECT
        ST_SetValue(
            ST_SetValue(
                ST_SetValue(
                    ST_SetValue(
                        ST_SetValue(
                            ST_AddBand(
                                ST_MakeEmptyRaster(5, 5, -2, 2, 1, -1, 0, 0, 0),
                                '8BUI'::text, 1, 0
                            ),
                            1, 1, 0.
                        ),
                        2, 3, 0.
                    ),
                    3, 5, 0.
                ),
                4, 2, 0.
            ),
            5, 4, 0.
        ) AS rast
) AS foo

 value | nearestvalue
-------+--------------
     1 |            1
```

```
-- pixel 2x3 is NODATA
SELECT
    ST_Value(rast, 2, 3) AS value,
    ST_NearestValue(rast, 2, 3) AS nearestvalue
FROM (
    SELECT
        ST_SetValue(
            ST_SetValue(
                ST_SetValue(
                    ST_SetValue(
                        ST_SetValue(
                            ST_AddBand(
                                ST_MakeEmptyRaster(5, 5, -2, 2, 1, -1, 0, 0, 0),
                                '8BUI'::text, 1, 0
                            ),
                            1, 1, 0.
                        ),
                        2, 3, 0.
                    ),
                    3, 5, 0.
                ),
                4, 2, 0.
            ),
            5, 4, 0.
        ) AS rast
) AS foo

 value | nearestvalue
```

```
-------+--------------
       |            1
```

Siehe auch

ST_Neighborhood, ST_Value

10.6.9 ST_SetZ

ST_SetZ — Returns a geometry with the same X/Y coordinates as the input geometry, and values from the raster copied into the Z dimension using the requested resample algorithm.

Synopsis

geometry **ST_SetZ**(raster rast, geometry geom, text resample=nearest, integer band=1);

Beschreibung

Returns a geometry with the same X/Y coordinates as the input geometry, and values from the raster copied into the Z dimensions using the requested resample algorithm.

The `resample` parameter can be set to "nearest" to copy the values from the cell each vertex falls within, or "bilinear" to use bilinear interpolation to calculate a value that takes neighboring cells into account also.

Availability: 3.2.0

Beispiele

```
--
-- 2x2 test raster with values
--
-- 10 50
-- 40 20
--
WITH test_raster AS (
SELECT
ST_SetValues(
  ST_AddBand(
    ST_MakeEmptyRaster(width =
> 2, height =
> 2,
      upperleftx =
> 0, upperlefty =
> 2,
      scalex =
> 1.0, scaley =
> -1.0,
      skewx =
> 0, skewy =
> 0, srid =
> 4326),
    index =
> 1, pixeltype =
> '16BSI',
    initialvalue =
> 0,
```

```
    nodataval =
> -999),
  1,1,1,
  newvalueset =
>ARRAY[ARRAY[10.0::float8, 50.0::float8], ARRAY[40.0::float8, 20.0::float8]]) AS rast
)
SELECT
ST_AsText(
  ST_SetZ(
    rast,
    band =
> 1,
    geom =
> 'SRID=4326;LINESTRING(1.0 1.9, 1.0 0.2)'::geometry,
    resample =
> 'bilinear'
))
FROM test_raster

            st_astext
-----------------------------------
 LINESTRING Z (1 1.9 38,1 0.2 27)
```

Siehe auch

ST_Value, ST_SetM

10.6.10 ST_SetM

ST_SetM — Returns a geometry with the same X/Y coordinates as the input geometry, and values from the raster copied into the M dimension using the requested resample algorithm.

Synopsis

geometry **ST_SetM**(raster rast, geometry geom, text resample=nearest, integer band=1);

Beschreibung

Returns a geometry with the same X/Y coordinates as the input geometry, and values from the raster copied into the M dimensions using the requested resample algorithm.

The `resample` parameter can be set to "nearest" to copy the values from the cell each vertex falls within, or "bilinear" to use bilinear interpolation to calculate a value that takes neighboring cells into account also.

Availability: 3.2.0

Beispiele

```
--
-- 2x2 test raster with values
--
-- 10 50
-- 40 20
--
WITH test_raster AS (
SELECT
```

```
ST_SetValues(
  ST_AddBand(
    ST_MakeEmptyRaster(width =
> 2, height =
> 2,
      upperleftx =
> 0, upperlefty =
> 2,
      scalex =
> 1.0, scaley =
> -1.0,
      skewx =
> 0, skewy =
> 0, srid =
> 4326),
    index =
> 1, pixeltype =
> '16BSI',
    initialvalue =
> 0,
    nodataval =
> -999),
  1,1,1,
  newvalueset =
>ARRAY[ARRAY[10.0::float8, 50.0::float8], ARRAY[40.0::float8, 20.0::float8]]) AS rast
)
SELECT
ST_AsText(
  ST_SetM(
    rast,
    band =
> 1,
    geom =
> 'SRID=4326;LINESTRING(1.0 1.9, 1.0 0.2)'::geometry,
    resample =
> 'bilinear'
))
FROM test_raster

            st_astext
---------------------------------
 LINESTRING M (1 1.9 38,1 0.2 27)
```

Siehe auch

ST_Value, ST_SetZ

10.6.11 ST_Neighborhood

ST_Neighborhood — Gibt ein 2-D Feld in "Double Precision" aus, das sich aus nicht NODATA Werten um ein bestimmtes Pixel herum zusammensetzt. Das Pixel Kann über "columnx" und "rowy" oder über eine Punktgeometrie - im gleichen Koordinatenreferenzsystem wie der Raster - ausgewählt werden.

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=tru

Beschreibung

Gibt für ein Pixel eines Bandes die ringsherum liegenden nicht-NODATA Werte in einem 2D-Feld mit Double Precision zurück. Das Pixel kann entweder über columnX und rowY, oder über einen geometrischen Punkt der im selben Koordinatenreferenzsystem wie der Raster vorliegt übergeben werden. Die Parameter `distanceX` und `distanceY` bestimmen die Anzahl der Pixel auf der X- und Y-Achse, um das festgelegte Pixel herum; z.B.: Sie möchten alle Werte innerhalb einer Entfernung von 3 Pixel entlang der X-Achse und einer Entfernung von 2 Pixel entlang der Y-Achse, um das Pixel von Interesse herum. Der Wert im Zentrum des 2-D Feldes ist der Wert jenes Pixels, das über columnX und rowY oder über einen geometrischen Punkt übergeben wurde.

Die Bandnummern beginnen mit 1 und wenn keine Bandnummer angegeben ist, dann wird für `bandnum` 1 angenommen. Wenn `exclude_nodata_value` auf FALSE gesetzt ist, werden auch die Pixel mit einem `nodata` Wert einbezogen. Wenn `exclude_nodata_value` nicht übergeben wird, dann wird er über die Metadaten des Rasters ausgelesen.

Note
Die Anzahl der Elemente entlang jeder Achse des zurückgegebenen 2D-Feldes ergibt sich aus 2 * (`distanceX`|`distanceY`) + 1. So ergibt sich bei einer `distanceX` und einer `distanceY` von je 1, ein Feld von 3x3.

Note
Das 2-D Feld kann einer beliebigen, integrierten Funktion zur Weiterverarbeitung übergeben werden, wie z.B. an ST_Min4ma, ST_Sum4ma, ST_Mean4ma.

Verfügbarkeit: 2.1.0

Beispiele

```
-- pixel 2x2 has value
SELECT
    ST_Neighborhood(rast, 2, 2, 1, 1)
FROM (
    SELECT
        ST_SetValues(
            ST_AddBand(
                ST_MakeEmptyRaster(5, 5, -2, 2, 1, -1, 0, 0, 0),
                '8BUI'::text, 1, 0
            ),
            1, 1, 1, ARRAY[
                [0, 1, 1, 1, 1],
                [1, 1, 1, 0, 1],
                [1, 0, 1, 1, 1],
                [1, 1, 1, 1, 0],
                [1, 1, 0, 1, 1]
            ]::double precision[],
            1
        ) AS rast
) AS foo
```

```
         st_neighborhood
--------------------------------
{{NULL,1,1},{1,1,1},{1,NULL,1}}
```

```
-- pixel 2x3 is NODATA
SELECT
    ST_Neighborhood(rast, 2, 3, 1, 1)
FROM (
    SELECT
        ST_SetValues(
            ST_AddBand(
                ST_MakeEmptyRaster(5, 5, -2, 2, 1, -1, 0, 0, 0),
                '8BUI'::text, 1, 0
            ),
            1, 1, 1, ARRAY[
                [0, 1, 1, 1, 1],
                [1, 1, 1, 0, 1],
                [1, 0, 1, 1, 1],
                [1, 1, 1, 1, 0],
                [1, 1, 0, 1, 1]
            ]::double precision[],
            1
        ) AS rast
) AS foo

       st_neighborhood
------------------------------
 {{1,1,1},{1,NULL,1},{1,1,1}}
```

```
-- pixel 3x3 has value
-- exclude_nodata_value = FALSE
SELECT
    ST_Neighborhood(rast, 3, 3, 1, 1, false)
FROM ST_SetValues(
            ST_AddBand(
                ST_MakeEmptyRaster(5, 5, -2, 2, 1, -1, 0, 0, 0),
                '8BUI'::text, 1, 0
            ),
            1, 1, 1, ARRAY[
                [0, 1, 1, 1, 1],
                [1, 1, 1, 0, 1],
                [1, 0, 1, 1, 1],
                [1, 1, 1, 1, 0],
                [1, 1, 0, 1, 1]
            ]::double precision[],
            1
        ) AS rast

      st_neighborhood
---------------------------
{{1,1,0},{0,1,1},{1,1,1}}
```

Siehe auch

ST_NearestValue, ST_Min4ma, ST_Max4ma, ST_Sum4ma, ST_Mean4ma, ST_Range4ma, ST_Distinct4ma, ST_StdDev4ma

10.6.12 ST_SetValue

ST_SetValue — Setzt den Wert für ein Pixel eines Bandes, das über columnx und rowy festgelegt wird, oder für die Pixel die eine bestimmte Geometrie schneiden, und gibt den veränderten Raster zurück. Die Bandnummerierung beginnt mit 1; wenn die Bandnummer nicht angegeben ist, wird 1 angenommen.

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);

Beschreibung

Setzt die Werte bestimmter Pixel eines Bandes auf einen neuen Wert und gibt den veränderten Raster zurück. Die Pixel können über die Rasterzeile und die Rasterspalte, oder über eine Geometrie festgelegt werden. Wenn die Bandnummer nicht angegeben ist, wird 1 angenommen.

Erweiterung: 2.1.0 Die geometrische Variante von ST_SetValue() unterstützt nun jeden geometrischen Datentyp, nicht nur POINT. Die geometrische Variante ist ein Adapter für die geomval[] Variante von ST_SetValues()

Beispiele

```
-- Geometry example
SELECT (foo.geomval).val, ST_AsText(ST_Union((foo.geomval).geom))
FROM (SELECT ST_DumpAsPolygons(
        ST_SetValue(rast,1,
                ST_Point(3427927.75, 5793243.95),
                50)
            ) As geomval
FROM dummy_rast
where rid = 2) As foo
WHERE (foo.geomval).val < 250
GROUP BY (foo.geomval).val;

 val |                                                              st_astext
-----+-------------------------------------------------------------------------
  50 | POLYGON((3427927.75 5793244,3427927.75 5793243.95,3427927.8 579324 ...
 249 | POLYGON((3427927.95 5793243.95,3427927.95 5793243.85,3427928 57932 ...
```

```
-- Store the changed raster --
    UPDATE dummy_rast SET rast = ST_SetValue(rast,1, ST_Point(3427927.75, 5793243.95),100)
        WHERE rid = 2   ;
```

Siehe auch

ST_Value, ST_DumpAsPolygons

10.6.13 ST_SetValues

ST_SetValues — Gibt einen Raster zurück, der durch das Setzen der Werte eines bestimmten Bandes verändert wurde.

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);

Beschreibung

Gibt einen veränderten Raster zurück, der durch die Zuweisung neuer Zellwerte auf festgelegte Pixel des ausgewiesenen Rasterbandes entsteht. `columnx` und `rowy` sind bei 1 beginnend indiziert.

Wenn `keepnodata` TRUE ist, dann werden die Pixel mit dem Wert NODATA nicht mit dem entsprechenden Wert in `newvalueset` belegt.

Bei der Variante 1 werden die betreffenden Pixel über die Pixelkoordinaten `columnx` und `rowy`, und durch die Größe des Feldes `newvalueset` festgelegt. Über den Parameter `noset` kann verhindert werden, dass bestimmte, in `newvalueset` auftretende Pixelwerte gesetzt werden (da PostgreSQL keine unregelmäßigen Felder/"ragged arrays" zulässt). Siehe das Beispiel mit der Variante 1.

Variante 2 gleicht Variante 1, aber mit einem einfachen `nosetvalue` in Double Precision anstelle des booleschen Feldes `noset`. Elemente in `newvalueset` mit dem Wert `nosetvalue` werden übersprungen. Siehe das Beispiel mit der Variante 2.

Bei der Variante 3 werden die betreffenden Pixel über die Pixelkoordinaten `columnx` und `rowy`, und durch `width` und `height` festgelegt. Siehe das Beispiel mit der Variante 3.

Variante 4 entspricht Variante3 mit der Ausnahme, dass die Pixel des ersten Bandes von `rast` gesetzt werden.

Bei der Variante 5 wird ein Feld von geomval verwendet um die Pixel zu bestimmen. Wenn die gesamte Geometrie des Feldes vom Datentyp POINT oder MULTIPOINT ist, verwendet die Funktion Länge und Breite eines jeden Punktes um den Wert eines Pixels direkt zu setzen. Andernfalls wird die Geometrie in Raster umgewandelt über die dann in einem Schritt iteriert wird. Siehe das Beispiel mit der Variante 5.

Verfügbarkeit: 2.1.0

Beispiele: Variante 1

```
/*
The ST_SetValues() does the following...

+ - + - + - +          + - + - + - +
| 1 | 1 | 1 |          | 1 | 1 | 1 |
+ - + - + - +          + - + - + - +
| 1 | 1 | 1 |    =
>    | 1 | 9 | 9 |
+ - + - + - +          + - + - + - +
| 1 | 1 | 1 |          | 1 | 9 | 9 |
+ - + - + - +          + - + - + - +
*/
SELECT
    (poly).x,
    (poly).y,
    (poly).val
FROM (
```

```
SELECT
    ST_PixelAsPolygons(
        ST_SetValues(
            ST_AddBand(
                ST_MakeEmptyRaster(3, 3, 0, 0, 1, -1, 0, 0, 0),
                1, '8BUI', 1, 0
            ),
            1, 2, 2, ARRAY[[9, 9], [9, 9]]::double precision[][]
        )
    ) AS poly
) foo
ORDER BY 1, 2;

 x | y | val
---+---+-----
 1 | 1 |   1
 1 | 2 |   1
 1 | 3 |   1
 2 | 1 |   1
 2 | 2 |   9
 2 | 3 |   9
 3 | 1 |   1
 3 | 2 |   9
 3 | 3 |   9
```

```
/*
The ST_SetValues() does the following...

+ - + - + - +          + - + - + - +
| 1 | 1 | 1 |          | 9 | 9 | 9 |
+ - + - + - +          + - + - + - +
| 1 | 1 | 1 |    =
>    | 9 |   | 9 |
+ - + - + - +          + - + - + - +
| 1 | 1 | 1 |          | 9 | 9 | 9 |
+ - + - + - +          + - + - + - +
*/
SELECT
    (poly).x,
    (poly).y,
    (poly).val
FROM (
SELECT
    ST_PixelAsPolygons(
        ST_SetValues(
            ST_AddBand(
                ST_MakeEmptyRaster(3, 3, 0, 0, 1, -1, 0, 0, 0),
                1, '8BUI', 1, 0
            ),
            1, 1, 1, ARRAY[[9, 9, 9], [9, NULL, 9], [9, 9, 9]]::double precision[][]
        )
    ) AS poly
) foo
ORDER BY 1, 2;

 x | y | val
---+---+-----
 1 | 1 |   9
 1 | 2 |   9
 1 | 3 |   9
 2 | 1 |   9
 2 | 2 |
```

```
 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 | 3 |   9
 3 | 2 |   9
```

Beispiele: Variante 2

```
/*
The ST_SetValues() does the following...

+ - + - + - +          + - + - + - +
| 1 | 1 | 1 |          | 1 | 1 | 1 |
+ - + - + - +          + - + - + - +
| 1 | 1 | 1 |    =
>    | 1 | 9 | 9 |
+ - + - + - +          + - + - + - +
| 1 | 1 | 1 |          | 1 | 9 | 9 |
+ - + - + - +          + - + - + - +
*/
SELECT
    (poly).x,
    (poly).y,
    (poly).val
FROM (
SELECT
    ST_PixelAsPolygons(
        ST_SetValues(
            ST_AddBand(
                ST_MakeEmptyRaster(3, 3, 0, 0, 1, -1, 0, 0, 0),
```

```
                1, '8BUI', 1, 0
            ),
            1, 1, 1, ARRAY[[-1, -1, -1], [-1, 9, 9], [-1, 9, 9]]::double precision[][], -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
```

Beispiele: Variante 3

```
/*
The ST_SetValues() does the following...

+ - + - + - +          + - + - + - +
| 1 | 1 | 1 |          | 1 | 1 | 1 |
+ - + - + - +          + - + - + - +
| 1 | 1 | 1 |    =
>    | 1 | 9 | 9 |
+ - + - + - +          + - + - + - +
| 1 | 1 | 1 |          | 1 | 9 | 9 |
+ - + - + - +          + - + - + - +
*/
SELECT
    (poly).x,
    (poly).y,
    (poly).val
FROM (
SELECT
    ST_PixelAsPolygons(
        ST_SetValues(
            ST_AddBand(
                ST_MakeEmptyRaster(3, 3, 0, 0, 1, -1, 0, 0, 0),
                1, '8BUI', 1, 0
            ),
            1, 2, 2, 2, 2, 9
        )
    ) AS poly
) foo
ORDER BY 1, 2;

 x | y | val
---+---+-----
 1 | 1 |   1
 1 | 2 |   1
 1 | 3 |   1
 2 | 1 |   1
 2 | 2 |   9
 2 | 3 |   9
 3 | 1 |   1
 3 | 2 |   9
 3 | 3 |   9
```

```
/*
The ST_SetValues() does the following...

+ - + - + - +          + - + - + - +
| 1 | 1 | 1 |          | 1 | 1 | 1 |
+ - + - + - +          + - + - + - +
| 1 |   | 1 |    =
>    | 1 |   | 9 |
+ - + - + - +          + - + - + - +
| 1 | 1 | 1 |          | 1 | 9 | 9 |
+ - + - + - +          + - + - + - +
*/
SELECT
```

```
    (poly).x,
    (poly).y,
    (poly).val
FROM (
SELECT
    ST_PixelAsPolygons(
        ST_SetValues(
            ST_SetValue(
                ST_AddBand(
                    ST_MakeEmptyRaster(3, 3, 0, 0, 1, -1, 0, 0, 0),
                    1, '8BUI', 1, 0
                ),
                1, 2, 2, NULL
            ),
            1, 2, 2, 2, 2, 9, TRUE
        )
    ) AS poly
) foo
ORDER BY 1, 2;

 x | y | val
---+---+-----
 1 | 1 |   1
 1 | 2 |   1
 1 | 3 |   1
 2 | 1 |   1
 2 | 2 |
 2 | 3 |   9
 3 | 1 |   1
 3 | 2 |   9
 3 | 3 |   9
```

Beispiele: Variante 5

```
WITH foo AS (
    SELECT 1 AS rid, ST_AddBand(ST_MakeEmptyRaster(5, 5, 0, 0, 1, -1, 0, 0, 0), 1, '8BUI',  ↩
        0, 0) AS rast
), bar AS (
    SELECT 1 AS gid, 'SRID=0;POINT(2.5 -2.5)'::geometry geom UNION ALL
    SELECT 2 AS gid, 'SRID=0;POLYGON((1 -1, 4 -1, 4 -4, 1 -4, 1 -1))'::geometry geom UNION  ↩
        ALL
    SELECT 3 AS gid, 'SRID=0;POLYGON((0 0, 5 0, 5 -1, 1 -1, 1 -4, 0 -4, 0 0))'::geometry  ↩
        geom UNION ALL
    SELECT 4 AS gid, 'SRID=0;MULTIPOINT(0 0, 4 4, 4 -4)'::geometry
)
SELECT
    rid, gid, ST_DumpValues(ST_SetValue(rast, 1, geom, gid))
FROM foo t1
CROSS JOIN bar t2
ORDER BY rid, gid;

 rid | gid |                                                
```

```
   1 |    4 | (1,"{{4,NULL,NULL,NULL,NULL},{NULL,NULL,NULL,NULL,NULL},{NULL,NULL,NULL,NULL, ↩
       NULL},{NULL,NULL,NULL,NULL,NULL},{NULL,NULL,NULL,NULL,4}}")
(4 rows)
```

Das folgende Beispiel zeigt, dass bestehende "geomvals" durch ein Feld mit "geomvals" später überschrieben werden können

```
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)
```

Dieses Beispiel ist das Gegenteil des vorigen Beispiels

```
WITH foo AS (
    SELECT 1 AS rid, ST_AddBand(ST_MakeEmptyRaster(5, 5, 0, 0, 1, -1, 0, 0, 0), 1, '8BUI',  ↩
        0, 0) AS rast
), bar AS (
    SELECT 1 AS gid, 'SRID=0;POINT(2.5 -2.5)'::geometry geom UNION ALL
    SELECT 2 AS gid, 'SRID=0;POLYGON((1 -1, 4 -1, 4 -4, 1 -4, 1 -1))'::geometry geom UNION  ↩
        ALL
    SELECT 3 AS gid, 'SRID=0;POLYGON((0 0, 5 0, 5 -1, 1 -1, 1 -4, 0 -4, 0 0))'::geometry  ↩
        geom UNION ALL
    SELECT 4 AS gid, 'SRID=0;MULTIPOINT(0 0, 4 4, 4 -4)'::geometry
)
SELECT
    t1.rid, t2.gid, t3.gid, ST_DumpValues(ST_SetValues(rast, 1, ARRAY[ROW(t2.geom, t2.gid), ↩
         ROW(t3.geom, t3.gid)]::geomval[]))
FROM foo t1
CROSS JOIN bar t2
CROSS JOIN bar t3
WHERE t2.gid = 2
    AND t3.gid = 1
ORDER BY t1.rid, t2.gid, t3.gid;

 rid | gid | gid |                                                    st_dumpvalues
-----+-----+-----+-----------------------------------------------------------------------------------

   1 |    2 |    1 | (1,"{{NULL,NULL,NULL,NULL,NULL},{NULL,2,2,2,NULL},{NULL,2,1,2,NULL},{ ↩
       NULL,2,2,2,NULL},{NULL,NULL,NULL,NULL,NULL}}")
```

```
(1 row)
```

Siehe auch

ST_Value, ST_SetValue, ST_PixelAsPolygons

10.6.14 ST_DumpValues

ST_DumpValues — Gibt die Werte eines bestimmten Bandes als 2-dimensionales Feld aus.

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);

Beschreibung

Gibt die Werte eines bestimmten Bandes als 2-dimensionales Feld (der erste Index entspricht der Zeile, der zweite der Spalte) aus. Wenn `nband` NULL oder nicht gegeben ist, werden alle Rasterbänder abgearbeitet.

Verfügbarkeit: 2.1.0

Beispiele

```
WITH foo AS (
    SELECT ST_AddBand(ST_AddBand(ST_AddBand(ST_MakeEmptyRaster(3, 3, 0, 0, 1, -1, 0, 0, 0), ↩
        1, '8BUI'::text, 1, 0), 2, '32BF'::text, 3, -9999), 3, '16BSI', 0, 0) AS rast
)
SELECT
    (ST_DumpValues(rast)).*
FROM foo;

 nband |                        valarray
-------+-----------------------------------------------------
     1 | {{1,1,1},{1,1,1},{1,1,1}}
     2 | {{3,3,3},{3,3,3},{3,3,3}}
     3 | {{NULL,NULL,NULL},{NULL,NULL,NULL},{NULL,NULL,NULL}}
(3 rows)
```

```
WITH foo AS (
    SELECT ST_AddBand(ST_AddBand(ST_AddBand(ST_MakeEmptyRaster(3, 3, 0, 0, 1, -1, 0, 0, 0), ↩
        1, '8BUI'::text, 1, 0), 2, '32BF'::text, 3, -9999), 3, '16BSI', 0, 0) AS rast
)
SELECT
    (ST_DumpValues(rast, ARRAY[3, 1])).*
FROM foo;

 nband |                        valarray
-------+-----------------------------------------------------
     3 | {{NULL,NULL,NULL},{NULL,NULL,NULL},{NULL,NULL,NULL}}
     1 | {{1,1,1},{1,1,1},{1,1,1}}
(2 rows)
```

```
WITH foo AS (
    SELECT ST_SetValue(ST_AddBand(ST_MakeEmptyRaster(3, 3, 0, 0, 1, -1, 0, 0, 0), 1, '8BUI ↩
        ', 1, 0), 1, 2, 5) AS rast
)
SELECT
    (ST_DumpValues(rast, 1))[2][1]
FROM foo;

 st_dumpvalues
---------------
             5
(1 row)
```

Siehe auch

ST_Value, ST_SetValue, ST_SetValues

10.6.15 ST_PixelOfValue

ST_PixelOfValue — Gibt die columnx- und rowy-Koordinaten jener Pixel aus, deren Zellwert gleich dem gesuchten Wert ist.

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);

Beschreibung

Gibt die columnx- und rowy-Koordinaten jener Pixel aus, deren Zellwert gleich dem gesuchten Wert ist. Wenn kein Band angegeben ist, wird Band 1 angenommen.

Verfügbarkeit: 2.1.0

Beispiele

```
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
```

10.7 Raster Editoren

10.7.1 ST_SetGeoReference

ST_SetGeoReference — Georeferenziert einen Raster über 6 Parameter in einem einzigen Aufruf. Die Zahlen müssen durch Leerzeichen getrennt sein. Die Funktion akzeptiert die Eingabe im Format von 'GDAL' und von 'ESRI'. Der Standardwert ist GDAL.

Synopsis

raster **ST_SetGeoReference**(raster rast, text georefcoords, text format=GDAL);
raster **ST_SetGeoReference**(raster rast, double precision upperleftx, double precision upperlefty, double precision scalex, double precision scaley, double precision skewx, double precision skewy);

Beschreibung

Georeferenziert einen Raster über 6 Parameter in einem einzigen Aufruf. Die Funktion akzeptiert die Eingabe im Format von 'GDAL' und von 'ESRI'. Der Standardwert ist GDAL. Wenn die 6 Parameter nicht angegeben sind, wird NULL zurückgegeben.

Die Formate unterscheiden sich wie folgt:

GDAL:

```
scalex skewy skewx scaley upperleftx upperlefty
```

ESRI:

```
scalex skewy skewx scaley upperleftx + scalex*0.5 upperlefty + scaley*0.5
```

Note

Wenn der Raster out-db Bänder aufweist, dann kann eine Änderung der Georeferenzierung zu einem unkorrekten Zugriff auf die extern gespeicherten Daten des Bandes führen.

Erweiterung: 2.1.0 ST_SetGeoReference(raster, double precision, ...) Variante hinzugefügt

Beispiele

```
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     |     upperlefty     | width | height | scalex | scaley | skewx | ↩
     skewy | srid | numbands
-----+--------------------+--------------------+-------+--------+--------+--------+-------+-------+-
   0 |                  0 |                  0 |     5 |      5 |      1 |     -1 |     0 | ↩
           0 |    0 |        0
   1 |                0.1 |                0.1 |     5 |      5 |     10 |    -10 |     0 | ↩
           0 |    0 |        0
   2 | 0.0999999999999996 | 0.0999999999999996 |     5 |      5 |     10 |    -10 |     0 | ↩
           0 |    0 |        0
   3 |                  1 |                  1 |     5 |      5 |     10 |    -10 | 0.001 | ↩
       0.001 |    0 |        0
```

Siehe auch

ST_GeoReference, ST_ScaleX, ST_ScaleY, ST_UpperLeftX, ST_UpperLeftY

10.7.2 ST_SetRotation

ST_SetRotation — Bestimmt die Rotation des Rasters in Radiant.

Synopsis

raster **ST_SetRotation**(raster rast, float8 rotation);

Beschreibung

Einheitliche Rotation des Rasters. Die Rotation wird in Radiant angegeben. Siehe World File für mehr Details.

Beispiele

```
SELECT
  ST_ScaleX(rast1), ST_ScaleY(rast1), ST_SkewX(rast1), ST_SkewY(rast1),
  ST_ScaleX(rast2), ST_ScaleY(rast2), ST_SkewX(rast2), ST_SkewY(rast2)
FROM (
  SELECT ST_SetRotation(rast, 15) AS rast1, rast as rast2 FROM dummy_rast
) AS foo;
      st_scalex       |      st_scaley       |      st_skewx       |      st_skewy       |  ↩
          st_scalex | st_scaley | st_skewx | st_skewy
--------------------+----------------------+---------------------+---------------------+-----------+--

   -1.51937582571764 |   -2.27906373857646 |   1.95086352047135 |   1.30057568031423 |  ↩
               2 |          3 |         0 |         0
 -0.0379843956429411 | -0.0379843956429411 | 0.0325143920078558 | 0.0325143920078558 |  ↩
          0.05 |      -0.05 |         0 |         0
```

Siehe auch

ST_Rotation, ST_ScaleX, ST_ScaleY, ST_SkewX, ST_SkewY

10.7.3 ST_SetScale

ST_SetScale — Setzt die X- und Y-Größe der Pixel in den Einheiten des Koordinatenreferenzsystems. Entweder eine Zahl pro Pixel oder Breite und Höhe.

Synopsis

raster **ST_SetScale**(raster rast, float8 xy);
raster **ST_SetScale**(raster rast, float8 x, float8 y);

Beschreibung

Setzt die X- und Y-Größe der Pixel in den Einheiten des Koordinatenreferenzsystems. Entweder eine Zahl pro Pixel oder Breite und Höhe. X und Y werden als gleich groß angenommen, wenn nur eine Zahl übergeben wird.

Note
ST_SetScale unterscheidet sich von ST_Rescale dadurch, dass bei ST_SetScale der Raster nicht skaliert wird um mit der Rasterausdehnung übereinzustimmen. Es werden nur die Metadaten (oder die Georeferenz) des Rasters geändert, um eine ursprünglich falsch angegebene Skalierung zu korrigieren. ST_Rescale berechnet die Breite und Höhe eines Rasters so, dass er mit der geographischen Ausdehnung des Rasters übereinstimmt. ST_SetScale verändert weder die Breite noch die Höhe des Rasters.

Änderung: 2.0.0. Versionen von WKTRaster haben dies als "ST_SetPixelSizeY" bezeichnet. Dies wurde mit 2.0.0 geändert.

Beispiele

```
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)
```

Siehe auch

ST_ScaleX, ST_ScaleY, Box3D

10.7.4 ST_SetSkew

ST_SetSkew — Setzt den georeferenzierten X- und Y-Versatz (oder den Rotationsparameter). Wenn nur ein Wert übergeben wird, werden X und Y auf den selben Wert gesetzt.

Synopsis

raster **ST_SetSkew**(raster rast, float8 skewxy);
raster **ST_SetSkew**(raster rast, float8 skewx, float8 skewy);

Beschreibung

Setzt den georeferenzierten X- und Y-Versatz (oder den Rotationsparameter). Wenn nur ein Wert übergeben wird, werden X und Y auf den selben Wert gesetzt. Siehe World File für weitere Details.

Beispiele

```
-- Example 1
UPDATE dummy_rast SET rast = ST_SetSkew(rast,1,2) WHERE rid = 1;
SELECT rid, ST_SkewX(rast) As skewx, ST_SkewY(rast) As skewy,
    ST_GeoReference(rast) as georef
FROM dummy_rast WHERE rid = 1;

rid | skewx | skewy |    georef
----+-------+-------+--------------
```

```
  1 |     1 |     2 | 2.0000000000
                    : 2.0000000000
                    : 1.0000000000
                    : 3.0000000000
                    : 0.5000000000
                    : 0.5000000000
```

```
-- Example 2 set both to same number:
UPDATE dummy_rast SET rast = ST_SetSkew(rast,0) WHERE rid = 1;
SELECT rid, ST_SkewX(rast) As skewx, ST_SkewY(rast) As skewy,
    ST_GeoReference(rast) as georef
FROM dummy_rast WHERE rid = 1;

 rid | skewx | skewy |    georef
-----+-------+-------+--------------
   1 |     0 |     0 | 2.0000000000
                     : 0.0000000000
                     : 0.0000000000
                     : 3.0000000000
                     : 0.5000000000
                     : 0.5000000000
```

Siehe auch

ST_GeoReference, ST_SetGeoReference, ST_SkewX, ST_SkewY

10.7.5 ST_SetSRID

ST_SetSRID — Setzt die SRID eines Rasters auf einen bestimmten Ganzzahlwert. Die SRID wird in der Tabelle "spatial_ref_sys" definiert.

Synopsis

raster **ST_SetSRID**(raster rast, integer srid);

Beschreibung

Weist der SRID des Rasters einen bestimmten Ganzzahlwert zu.

Note
Diese Funktion führt keine Koordinatentransformation des Rasters durch - sie setzt nur die Metadaten, welche das Koordinatenreferenzsystem definieren, in dem der Raster vorliegt. Nützlich für spätere Koordinatentransformationen.

Siehe auch

Section 4.5, ST_SRID

10.7.6 ST_SetUpperLeft

ST_SetUpperLeft — Setzt den Wert der oberen linke Ecke des Rasters auf die projizierten X- und Y-Koordinaten.

Synopsis

raster **ST_SetUpperLeft**(raster rast, double precision x, double precision y);

Beschreibung

Setzt den Wert der oberen linke Ecke des Rasters auf die projizierten X- und Y-Koordinaten.

Beispiele

```
SELECT ST_SetUpperLeft(rast,-71.01,42.37)
FROM dummy_rast
WHERE rid = 2;
```

Siehe auch

ST_UpperLeftX, ST_UpperLeftY

10.7.7 ST_Resample

ST_Resample — Skaliert einen Raster mit einem bestimmten Algorithmus, neuen Dimensionen, einer beliebigen Gitterecke und über Parameter zur Georeferenzierung des Rasters, die angegeben oder von einem anderen Raster übernommen werden können.

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);

Beschreibung

Skaliert einen Raster mit einem bestimmten Algorithmus, neuen Dimensionen (width & height), einer Gitterecke (gridx & gridy) und über Parameter zur Georeferenzierung des Rasters (scalex, scaley, skewx & skewy), die angegeben oder von einem anderen Raster übernommen werden können. Wenn Sie einen Referenzraster verwenden, müssen beide Raster die selbe SRID haben.

New pixel values are computed using one of the following resampling algorithms:

- NearestNeighbor (english or american spelling)
- Bilinear
- Cubic
- CubicSpline
- Lanczos
- Max
- Min

The default is NearestNeighbor which is the fastest but results in the worst interpolation.

Wenn `maxerr` nicht angegeben ist, wird der maximale Fehler mit 0.125 Prozent angesetzt.

Note
Siehe GDAL Warp resampling methods für mehr Details.

Verfügbarkeit: 2.0.0 benötigt GDAL 1.6.1+

Enhanced: 3.4.0 max and min resampling options added

Beispiele

```
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
```

Siehe auch

ST_Rescale, ST_Resize, ST_Transform

10.7.8 ST_Rescale

ST_Rescale — Resample a raster by adjusting only its scale (or pixel size). New pixel values are computed using the NearestNeighbor (english or american spelling), Bilinear, Cubic, CubicSpline, Lanczos, Max or Min resampling algorithm. Default is NearestNeighbor.

Synopsis

raster **ST_Rescale**(raster rast, double precision scalexy, text algorithm=NearestNeighbor, double precision maxerr=0.125);
raster **ST_Rescale**(raster rast, double precision scalex, double precision scaley, text algorithm=NearestNeighbor, double precision maxerr=0.125);

Beschreibung

Resample a raster by adjusting only its scale (or pixel size). New pixel values are computed using one of the following resampling algorithms:

- NearestNeighbor (english or american spelling)
- Bilinear
- Cubic
- CubicSpline
- Lanczos
- Max
- Min

The default is NearestNeighbor which is the fastest but results in the worst interpolation.

`scalex` und `scaley` bestimmen die neue Pixelgröße. Der Wert von "scaley" muss oftmals negativ sein, um einen ordnungsgemäß ausgerichteten Raster zu erhalten.

Wenn "scalex" oder "scaley" teilerfremd zur Breite oder Höhe des Rasters sind, dann wird der Zielraster auf die Ausdehnung des Ausgangsrasters erweitert. Um die exakte Ausdehnung des Ausgangsrasters sicher zu erhalten, siehe ST_Resize

`maxerr` ist der Schwellenwert bei der Näherungstransformation des Skalierungsalgorithmus (in Pixeleinheiten). Ein Standardwert von 0.125 wird verwendet, wenn kein `maxerr` angegeben wird. Dies ist dergleiche Wert, welcher von gdalwarp verwendet wird. Wenn der Wert auf Null gesetzt ist, wird keine Annährung ausgeführt.

Note
Siehe GDAL Warp resampling methods für mehr Details.

Note
ST_Rescale unterscheidet sich von ST_SetScale darin, dass ST_SetScale den Raster nicht skaliert um mit der Ausdehnung des Ausgangsrasters übereinzustimmen. ST_SetScale ändert lediglich die Metadaten (oder die Georeferenz) des Rasters, um eine ursprünglich falsch angegebene Skalierung zu korrigieren. ST_Rescale berechnet die Breite und Höhe eines Rasters so, dass er mit der geographischen Ausdehnung des Ausgangsraster übereinstimmt. ST_SetScale verändert weder die Breite noch die Höhe des Rasters.

Verfügbarkeit: 2.0.0 benötigt GDAL 1.6.1+

Enhanced: 3.4.0 max and min resampling options added

Änderung: 2.1.0 Funktioniert jetzt auch mit Raster ohne SRID

Beispiele

Ein einfaches Beispiel, das die Pixelgröße eines Raster von 0.001 Grad auf 0.0015 Grad ändert.

```
-- the original raster pixel size
SELECT ST_PixelWidth(ST_AddBand(ST_MakeEmptyRaster(100, 100, 0, 0, 0.001, -0.001, 0, 0,  ↩
    4269), '8BUI'::text, 1, 0)) width

   width
----------
```

```
0.001

-- the rescaled raster raster pixel size
SELECT ST_PixelWidth(ST_Rescale(ST_AddBand(ST_MakeEmptyRaster(100, 100, 0, 0, 0.001,  ↩
    -0.001, 0, 0, 4269), '8BUI'::text, 1, 0), 0.0015)) width

   width
----------
0.0015
```

Siehe auch

ST_Resize, ST_Resample, ST_SetScale, ST_ScaleX, ST_ScaleY, ST_Transform

10.7.9 ST_Reskew

ST_Reskew — Skaliert einen Raster, indem lediglich der Versatz (oder Rotationsparameter) angepasst wird. Neue Pixelwerte werden über NearestNeighbor, bilinear, kubisch, CubicSpline oder mit dem Lanczos-Filter errechnet. Die Standardeinstellung ist NearestNeighbor.

Synopsis

raster **ST_Reskew**(raster rast, double precision skewxy, text algorithm=NearestNeighbor, double precision maxerr=0.125);
raster **ST_Reskew**(raster rast, double precision skewx, double precision skewy, text algorithm=NearestNeighbor, double precision maxerr=0.125);

Beschreibung

Skaliert einen Raster, indem lediglich der Versatz (oder Rotationsparameter) angepasst wird. Neue Pixelwerte werden über NearestNeighbor, bilinear, kubisch, CubicSpline oder mit dem Lanczos-Filter errechnet. Die Standardeinstellung "NearestNeighbor" ist am schnellsten, ergibt aber die schlechteste Interpolation.

`skewx` und `skewy` legen den neuen Versatz fest.

Die Ausdehnung des Zielrasters erfasst die Ausdehnung des Ausgangsrasters.

Wenn `maxerr` nicht angegeben ist, wird der maximale Fehler mit 0.125 Prozent angesetzt.

Note
Siehe GDAL Warp resampling methods für mehr Details.

Note
ST_Reskew unterscheidet sich von ST_SetSkew darin, dass ST_SetSkew den Raster nicht skaliert um mit der Ausdehnung des Ausgangsrasters übereinzustimmen. ST_SetSkew ändert lediglich die Metadaten (oder die Georeferenz) des Rasters, um einen ursprünglich falsch angegebenen Versatz zu korrigieren. ST_Reskew ergibt einen Raster mit geänderter Breite und Höhe, da die Berechnung so durchgeführt wird, dass die geographischen Ausdehnung des Zielrasters mit der des Ausgangsrasters übereinstimmt. ST_SetSkew verändert weder die Breite noch die Höhe des Rasters.

Verfügbarkeit: 2.0.0 benötigt GDAL 1.6.1+

Änderung: 2.1.0 Funktioniert jetzt auch mit Raster ohne SRID

Beispiele

Ein einfaches Beispiel, das den Versatz eines Rasters von 0.0 auf 0.0015 ändert.

```
-- the original raster non-rotated
SELECT ST_Rotation(ST_AddBand(ST_MakeEmptyRaster(100, 100, 0, 0, 0.001, -0.001, 0, 0, 4269) ↩
    , '8BUI'::text, 1, 0));

-- result
0

-- the reskewed raster raster rotation
SELECT ST_Rotation(ST_Reskew(ST_AddBand(ST_MakeEmptyRaster(100, 100, 0, 0, 0.001, -0.001,  ↩
    0, 0, 4269), '8BUI'::text, 1, 0), 0.0015));

-- result
-0.982793723247329
```

Siehe auch

ST_Resample, ST_Rescale, ST_SetSkew, ST_SetRotation, ST_SkewX, ST_SkewY, ST_Transform

10.7.10 ST_SnapToGrid

ST_SnapToGrid — Skaliert einen Raster durch Fangen an einem Führungsgitter. Neue Pixelwerte werden über NearestNeighbor, bilinear, kubisch, CubicSpline oder mit dem Lanczos-Filter errechnet. Die Standardeinstellung ist NearestNeighbor.

Synopsis

raster **ST_SnapToGrid**(raster rast, double precision gridx, double precision gridy, text algorithm=NearestNeighbor, double precision maxerr=0.125, double precision scalex=DEFAULT 0, double precision scaley=DEFAULT 0);
raster **ST_SnapToGrid**(raster rast, double precision gridx, double precision gridy, double precision scalex, double precision scaley, text algorithm=NearestNeighbor, double precision maxerr=0.125);
raster **ST_SnapToGrid**(raster rast, double precision gridx, double precision gridy, double precision scalexy, text algorithm=NearestNeigh
double precision maxerr=0.125);

Beschreibung

Skaliert einen Raster durch Fangen an einem Führungsgitter, welches durch einen beliebige Pixeleckpunkt (gridx & gridy) und eine optionale Pixelgröße (scalex & scaley) definiert ist. Neue Pixelwerte werden über NearestNeighbor, bilinear, kubisch, CubicSpline oder mit dem Lanczos-Filter errechnet. Die Standardeinstellung "NearestNeighbor" ist am schnellsten, ergibt aber die schlechteste Interpolation.

`gridx` und `gridy` bestimmen einen beliebigen Pixeleckpunkt des neuen Gitters. Dies muss nicht unbedingt die obere linke Ecke des Zielrasters sein, darf aber nicht innerhalb oder am Rand der Ausdehnung des Zielrasters liegen.

Optional können Sie die Pixelgröße des neuen Gitters mit `scalex` und `scaley` festlegen.

Die Ausdehnung des Zielrasters erfasst die Ausdehnung des Ausgangsrasters.

Wenn `maxerr` nicht angegeben ist, wird der maximale Fehler mit 0.125 Prozent angesetzt.

Note
Siehe GDAL Warp resampling methods für mehr Details.

Note
Verwenden Sie bitte ST_Resample, wenn Sie eine bessere Kontrolle über die Gitterparameter benötigen.

Verfügbarkeit: 2.0.0 benötigt GDAL 1.6.1+

Änderung: 2.1.0 Funktioniert jetzt auch mit Raster ohne SRID

Beispiele

Ein einfaches Beispiel, bei dem ein Raster an einem sich geringfügig unterscheidenden Gitter gefangen wird.

```
-- the original raster upper left X
SELECT ST_UpperLeftX(ST_AddBand(ST_MakeEmptyRaster(10, 10, 0, 0, 0.001, -0.001, 0, 0, 4269) ↩
    , '8BUI'::text, 1, 0));
-- result
0

-- the upper left of raster after snapping
SELECT ST_UpperLeftX(ST_SnapToGrid(ST_AddBand(ST_MakeEmptyRaster(10, 10, 0, 0, 0.001,  ↩
    -0.001, 0, 0, 4269), '8BUI'::text, 1, 0), 0.0002, 0.0002));

--result
-0.0008
```

Siehe auch

ST_Resample, ST_Rescale, ST_UpperLeftX, ST_UpperLeftY

10.7.11 ST_Resize

ST_Resize — Ändert die Zellgröße - width/height - eines Rasters

Synopsis

raster **ST_Resize**(raster rast, integer width, integer height, text algorithm=NearestNeighbor, double precision maxerr=0.125);
raster **ST_Resize**(raster rast, double precision percentwidth, double precision percentheight, text algorithm=NearestNeighbor, double precision maxerr=0.125);
raster **ST_Resize**(raster rast, text width, text height, text algorithm=NearestNeighbor, double precision maxerr=0.125);

Beschreibung

Passt die Größe des Rasters an eine neue Breite/Höhe an. Die neue Breite/Höhe kann durch die genaue Anzahl der Pixel, oder durch einen Prozentsatz der Breite/Höhe des Rasters, angegeben werden. Die Ausdehnung des Zielrasters ist mit der Ausdehnung des Ausgangsrasters ident.

Neue Pixelwerte werden über NearestNeighbor, bilinear, kubisch, CubicSpline oder mit dem Lanczos-Filter errechnet. Die Standardeinstellung "NearestNeighbor" ist am schnellsten, erzeugt aber auch die schlechteste Interpolation.

Variante 1 erwartet die tatsächliche width/height des Ausgaberasters.

Variante 2 erwartet Dezimalwerte zwischen null (0) und eins (1), welche das Verhältnis zur Pixelbreite und zur Pixelhöhe des Eingaberasters angeben.

Variante 3 nimmt entweder die tatsächliche Breite/Höhe des Zielrasters oder einen Prozentsatz der Breite/Höhe des Ausgangsrasters als Zeichenfolge ("20%") entgegen.

Verfügbarkeit: 2.1.0 benötigt GDAL 1.6.1+

Beispiele

```
WITH foo AS(
SELECT
    1 AS rid,
    ST_Resize(
        ST_AddBand(
            ST_MakeEmptyRaster(1000, 1000, 0, 0, 1, -1, 0, 0, 0)
            , 1, '8BUI', 255, 0
        )
    , '50%', '500') AS rast
UNION ALL
SELECT
    2 AS rid,
    ST_Resize(
        ST_AddBand(
            ST_MakeEmptyRaster(1000, 1000, 0, 0, 1, -1, 0, 0, 0)
            , 1, '8BUI', 255, 0
        )
    , 500, 100) AS rast
UNION ALL
SELECT
    3 AS rid,
    ST_Resize(
        ST_AddBand(
            ST_MakeEmptyRaster(1000, 1000, 0, 0, 1, -1, 0, 0, 0)
            , 1, '8BUI', 255, 0
        )
    , 0.25, 0.9) AS rast
), bar AS (
    SELECT rid, ST_Metadata(rast) AS meta, rast FROM foo
)
SELECT rid, (meta).* FROM bar

 rid | upperleftx | upperlefty | width | height | scalex | scaley | skewx | skewy | srid |  ↩
     numbands
-----+------------+------------+-------+--------+--------+--------+-------+-------+------+----------

   1 |          0 |          0 |   500 |    500 |      1 |     -1 |     0 |     0 |    0 |  ↩
             1
   2 |          0 |          0 |   500 |    100 |      1 |     -1 |     0 |     0 |    0 |  ↩
             1
   3 |          0 |          0 |   250 |    900 |      1 |     -1 |     0 |     0 |    0 |  ↩
             1
(3 rows)
```

Siehe auch

ST_Resample, ST_Rescale, ST_Reskew, ST_SnapToGrid

10.7.12 ST_Transform

ST_Transform — Projiziert einen Raster von einem bekannten Koordinatenreferenzsystem in ein anderes bekanntes Koordinatenreferenzsystem um. Die Optionen für die Skalierung sind NearestNeighbor, Bilinear, Cubisch, CubicSpline und der Lanczos-Filter, die Standardeinstellung ist NearestNeighbor.

Synopsis

raster **ST_Transform**(raster rast, integer srid, text algorithm=NearestNeighbor, double precision maxerr=0.125, double precision scalex, double precision scaley);
raster **ST_Transform**(raster rast, integer srid, double precision scalex, double precision scaley, text algorithm=NearestNeighbor, double precision maxerr=0.125);
raster **ST_Transform**(raster rast, raster alignto, text algorithm=NearestNeighbor, double precision maxerr=0.125);

Beschreibung

Projiziert einen Raster von einem bekannten Koordinatenreferenzsystem in ein anderes bekanntes Koordinatenreferenzsystem um. Verwendet den angegebenen Algorithmus für die Interpolation der Pixelwerte. Wenn kein Algorithmus angegeben ist, wird 'NearestNeighbor' verwendet. Wenn "maxerr" nicht angegeben ist, wird ein Prozentsatz von 0.125 angenommen.

Die Optionen für den Algorithmus sind: 'NearestNeighbor', 'Bilinear', 'Cubic', 'CubicSpline' und 'Lanczos'. Siehe GDAL Warp resampling methods für weitere Details.

ST_Transform wird oft mit ST_SetSRID() verwechselt. ST_Transform ändert die Koordinaten der Geometrie tatsächlich (und die Pixelwerte mittels "resampling") von einem Koordinatenreferenzsystem auf ein anderes, während ST_SetSRID() nur den Identifikator "SRID" des Rasters ändert.

Anders als die anderen Varianten, benötigt Variante 3 einen Referenzraster für den Parameter `alignto`. Der Raster wird in das Koordinatenreferenzsystem (SRID) des Referenzrasters transformiert und an dem Referenzraster ausgerichtet (ST_SameAlignment = TRUE).

Note
Falls Sie herausfinden, dass die Koordinatentransformation nicht richtig unterstützt wird, müssen Sie vermutlich die Umgebungsvariable PROJSO auf jene Projektionsbibliothek ".so" oder ".dll" setzen, die von Ihrer PostGIS Installation verwendet wird. Hierbei muss nur der Name der Datei angegeben werden. Auf Windows zum Beispiel würden Sie unter Control Panel -> System -> Environment Variables eine Systemvariable mit dem Namen `PROJSO` hinzufügen und auf `libproj.dll` setzen (falls Sie Proj 4.6.1 verwenden). Nach dieser Änderung müssen Sie den PostgreSQL-Dienst/Dämon neu starten.

Warning
When transforming a coverage of tiles, you almost always want to use a reference raster to insure same alignment and no gaps in your tiles as demonstrated in example: Variant 3.

Verfügbarkeit: 2.0.0 benötigt GDAL 1.6.1+

Erweiterung: 2.1.0 Variante ST_Transform(rast, alignto) hinzugefügt

Beispiele

```
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
```

ursprüngliche "Mass State Plane Meters" (mass_stm)

Nach der Transformation in WGS 84 Länge Breite (wgs_84)

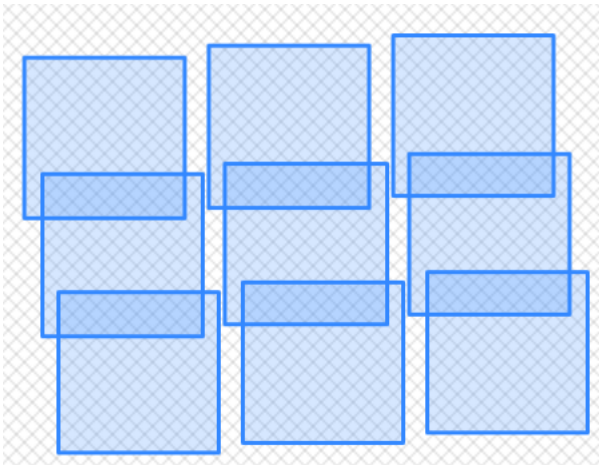
Koordinatentransformation nach WGS 84 mit dem bilinearen Algorithmus anstelle der Standardeinstellung NN (wgs_84_bilin)

Beispiele: Variante 3

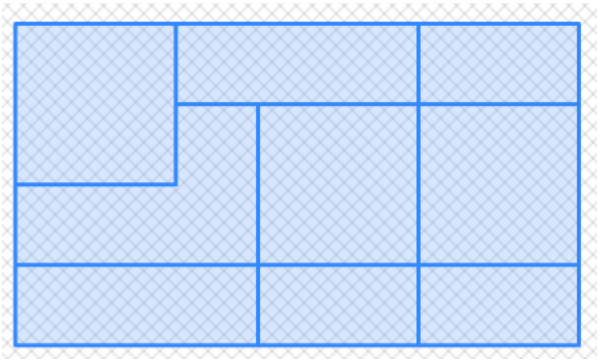
Im Folgenden wird der Unterschied zwischen der Verwendung von ST_Transform(raster, srid) und ST_Transform(raster, alignto) gezeigt

```
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
```



not_aligned



aligned

Siehe auch

ST_Transform, ST_SetSRID

10.8 Editoren für Rasterbänder

10.8.1 ST_SetBandNoDataValue

ST_SetBandNoDataValue — Setzt den NODATA Wert eines Bandes. Wenn kein Band angegeben ist, wird Band 1 angenommen. Falls ein Band keinen NODATA Wert aufweisen soll, übergeben Sie bitte für den Parameter "nodatavalue" NULL.

Synopsis

raster **ST_SetBandNoDataValue**(raster rast, double precision nodatavalue);
raster **ST_SetBandNoDataValue**(raster rast, integer band, double precision nodatavalue, boolean forcechecking=false);

Beschreibung

Setzt den NODATA Wert eines Bandes. Wenn kein Band angegeben ist, wird Band 1 angenommen. Dies beeinflusst die Ergebnisse von ST_Polygon, ST_DumpAsPolygons und die Funktionen ST_PixelAs...().

Beispiele

```
-- change just first band no data value
UPDATE dummy_rast
    SET rast = ST_SetBandNoDataValue(rast,1, 254)
WHERE rid = 2;

-- change no data band value of bands 1,2,3
UPDATE dummy_rast
    SET rast =
        ST_SetBandNoDataValue(
            ST_SetBandNoDataValue(
                ST_SetBandNoDataValue(
                    rast,1, 254)
                ,2,99),
                3,108)
        WHERE rid = 2;

-- wipe out the nodata value this will ensure all pixels are considered for all processing ↩
    functions
UPDATE dummy_rast
    SET rast = ST_SetBandNoDataValue(rast,1, NULL)
WHERE rid = 2;
```

Siehe auch

ST_BandNoDataValue, ST_NumBands

10.8.2 ST_SetBandIsNoData

ST_SetBandIsNoData — Setzt die Flag "isnodata" für das Band auf TRUE.

Synopsis

raster **ST_SetBandIsNoData**(raster rast, integer band=1);

Beschreibung

Setzt die Flag "isnodata" des Bandes auf TRUE. Wenn kein Band angegeben ist, wird Band 1 angenommen. Diese Funktion sollte nur aufgerufen werden, wenn sich die Flag verändert hat. Dies ist der Fall, wenn sich die Ergebnisse von ST_BandIsNoData unterscheiden, da einmal mit TRUE als letzten Übergabewert und ein anderes mal ohne diesen aufgerufen wurde.

Verfügbarkeit: 2.0.0

Beispiele

```
-- Create dummy table with one raster column
create table dummy_rast (rid integer, rast raster);

-- Add raster with two bands, one pixel/band. In the first band, nodatavalue = pixel value ↩
    = 3.
-- In the second band, nodatavalue = 13, pixel value = 4
insert into dummy_rast values(1,
(
'01' -- little endian (uint8 ndr)
```

```
||
'0000' -- version (uint16 0)
||
'0200' -- nBands (uint16 0)
||
'17263529ED684A3F' -- scaleX (float64 0.000805965234044584)
||
'F9253529ED684ABF' -- scaleY (float64 -0.00080596523404458)
||
'1C9F33CE69E352C0' -- ipX (float64 -75.5533328537098)
||
'718F0E9A27A44840' -- ipY (float64 49.2824585505576)
||
'ED50EB853EC32B3F' -- skewX (float64 0.000211812383858707)
||
'7550EB853EC32B3F' -- skewY (float64 0.000211812383858704)
||
'E6100000' -- SRID (int32 4326)
||
'0100' -- width (uint16 1)
||
'0100' -- height (uint16 1)
||
'4' -- hasnodatavalue set to true, isnodata value set to false (when it should be true)
||
'2' -- first band type (4BUI)
||
'03' -- novalue==3
||
'03' -- pixel(0,0)==3 (same that nodata)
||
'0' -- hasnodatavalue set to false
||
'5' -- second band type (16BSI)
||
'0D00' -- novalue==13
||
'0400' -- pixel(0,0)==4
)::raster
);

select st_bandisnodata(rast, 1) from dummy_rast where rid = 1; -- Expected false
select st_bandisnodata(rast, 1, TRUE) from dummy_rast where rid = 1; -- Expected true

-- The isnodata flag is dirty. We are going to set it to true
update dummy_rast set rast = st_setbandisnodata(rast, 1) where rid = 1;


select st_bandisnodata(rast, 1) from dummy_rast where rid = 1; -- Expected true
```

Siehe auch

ST_BandNoDataValue, ST_NumBands, ST_SetBandNoDataValue, ST_BandIsNoData

10.8.3 ST_SetBandPath

ST_SetBandPath — Aktualisiert den externen Dateipfad und die Bandnummer eines out-db Bandes.

Synopsis

raster **ST_SetBandPath**(raster rast, integer band, text outdbpath, integer outdbindex, boolean force=false);

Beschreibung

Aktualisiert den externen Rasterdateipfad und die externe Bandnummer eines out-db Bandes.

Note
Wenn `force` auf TRUE gesetzt ist, wird die Kompatibilität (z.B. Ausrichtung, Pixelunterstützung) zwischen der externen Rasterdatei und dem PostGIS Raster nicht überprüft. Dieser Modus ist für Dateisystemänderungen vorgesehen, bei denen der externe Raster bestehen bleibt.

Note
Intern wird bei dieser Methode das PostGIS Raster Band mit dem Index `band` durch ein neues Band ersetzt, ohne dass die existierende Pfadinformation aktualisiert wird.

Verfügbarkeit: 2.5.0

Beispiele

```
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
```

Siehe auch

ST_BandMetaData, ST_SetBandIndex

10.8.4 ST_SetBandIndex

ST_SetBandIndex — Aktualisiert die externe Bandnummer eines out-db Bandes.

Synopsis

raster **ST_SetBandIndex**(raster rast, integer band, integer outdbindex, boolean force=false);

Beschreibung

Aktualisiert die externe Bandnummer eines out-db Bandes. Die mit dem out-db Band assoziierte externe Rasterdatei wird davon nicht betroffen

Note
Wenn `force` auf TRUE gesetzt ist, wird die Kompatibilität (z.B. Ausrichtung, Pixelunterstützung) zwischen der externen Rasterdatei und dem PostGIS Raster nicht überprüft. Dieser Modus ist für das Verschieben von Bändern in einer externen Rasterdatei vorgesehen.

Note
Intern wird bei dieser Methode das PostGIS Raster Band mit dem Index `band` durch ein neues Band ersetzt, ohne dass die existierende Pfadinformation aktualisiert wird.

Verfügbarkeit: 2.5.0

Beispiele

```
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
```

Siehe auch

ST_BandMetaData, ST_SetBandPath

10.9 Rasterband Statistik und Analytik

10.9.1 ST_Count

ST_Count — Gibt die Anzahl der Pixel für ein Band eines Rasters oder eines Raster-Coverage zurück. Wenn kein Band angegeben ist, wird standardmäßig Band 1 gewählt. Wenn der Parameter "exclude_nodata_value" auf TRUE gesetzt ist, werden nur Pixel mit Werten ungleich NODATA gezählt.

Synopsis

bigint **ST_Count**(raster rast, integer nband=1, boolean exclude_nodata_value=true);
bigint **ST_Count**(raster rast, boolean exclude_nodata_value);

Beschreibung

Gibt die Anzahl der Pixel für ein Band eines Rasters oder eines Raster-Coverage zurück. Wenn kein Band angegeben ist, wird für `nband` 1 vorausgesetzt.

Note
Wenn der Parameter `exclude_nodata_value` auf TRUE gesetzt ist, werden nur die Pixel des Rasters gezählt, deren Werte ungleich `nodata` sind. Setzen Sie bitte `exclude_nodata_value` auf FALSE um die Anzahl aller Pixel zu erhalten

Changed: 3.1.0 - The ST_Count(rastertable, rastercolumn, ...) variants removed. Use ST_CountAgg instead.

Verfügbarkeit: 2.0.0

Beispiele

```
--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
```

Siehe auch

ST_CountAgg, ST_SummaryStats, ST_SetBandNoDataValue

10.9.2 ST_CountAgg

ST_CountAgg — Aggregatfunktion. Gibt die Anzahl der Pixel in einem bestimmten Band der Raster aus. Wenn kein Band angegeben ist, wird Band 1 angenommen. Wenn "exclude_nodata_value" TRUE ist, werden nur die Pixel ohne NODATA Werte gezählt.

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);

Beschreibung

Gibt die Anzahl der Pixel in einem bestimmten Band der Raster aus. Wenn kein Band angegeben ist, wird `nband` standardmäßig auf 1 gesetzt.

Wenn der Parameter `exclude_nodata_value` auf TRUE gesetzt ist, werden nur die Pixel des Rasters gezählt, deren Werte ungleich `NODATA` sind. Setzen Sie bitte `exclude_nodata_value` auf FALSE um die Anzahl aller Pixel zu erhalten

Standardmäßig werden alle Pixel abgetastet. Um eine schnellere Rückmeldung zu bekommen, können Sie den Parameter `sample_percent` auf einen Wert zwischen null (0) und eins (1) setzen.

Verfügbarkeit: 2.2.0

Beispiele

```
WITH foo AS (
    SELECT
        rast.rast
    FROM (
        SELECT ST_SetValue(
            ST_SetValue(
                ST_SetValue(
                    ST_AddBand(
                        ST_MakeEmptyRaster(10, 10, 10, 10, 2, 2, 0, 0,0)
                        , 1, '64BF', 0, 0
                    )
                    , 1, 1, 1, -10
                )
                , 1, 5, 4, 0
            )
            , 1, 5, 5, 3.14159
        ) AS rast
    ) AS rast
    FULL JOIN (
        SELECT generate_series(1, 10) AS id
    ) AS id
        ON 1 = 1
)
SELECT
    ST_CountAgg(rast, 1, TRUE)
FROM foo;

 st_countagg
-------------
          20
(1 row)
```

Siehe auch

ST_Count, ST_SummaryStats, ST_SetBandNoDataValue

10.9.3 ST_Histogram

ST_Histogram — Gibt Datensätze aus, welche die Verteilung der Daten eines Rasters oder eines Rastercoverage darstellen. Dabei wird die Wertemenge in Klassen aufgeteilt und für jede Klasse zusammengefasst. Wenn die Anzahl der Klassen nicht angegeben ist, wird sie automatisch berechnet.

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);

Beschreibung

Gibt Datensätze aus, die das Minimum, das Maximum, die Anzahl und die Prozent der Werte des Rasterbandes für jede Klasse enthalten. Wenn kein Band angegeben ist, wird `nband` standardmäßig auf 1 gesetzt.

Note
Standardmäßig werden nur jene Pixelwerte berücksichtigt, die nicht den Wert `NODATA` haben. Setzen Sie bitte `exclude_nodata_value` auf FALSE um die Anzahl sämtlicher Pixel zu erhalten.

width **double precision[]** Breite: ein Feld das die Breite für jede Kategorie/Klasse angibt. Wenn die Anzahl der Klassen größer ist als die Anzahl der Breiten, werden die Breiten wiederholt.

Beispiel: 9 Klassen, die Breiten sind [a, b, c] und werden als [a, b, c, a, b, c, a, b, c] übergegeben.

bins **integer** Anzahl der Klassen - die Anzahl der Datensätze die von der Funktion ausgegeben werden. Wenn die Anzahl der Klassen nicht angegeben ist, wird sie automatisch berechnet.

right **boolean** Berechnet das Histogramm von rechts anstatt von links (Standardwert). Dies ändert das Auswahlkriterium für einen Wert x von [a,b) auf (a,b].

Changed: 3.1.0 Removed ST_Histogram(table_name, column_name) variant.

Verfügbarkeit: 2.0.0

Beispiel: Einzelne Rasterkachel - Berechnung des Histogramm für die Bänder 1, 2, 3 und automatische Einteilung der Wertemenge in Klassen

```
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
```

Beispiel: Nur Band 2 und 6 Klassen

```
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)
```

Siehe auch

ST_Count, ST_SummaryStats, ST_SummaryStatsAgg

10.9.4 ST_Quantile

ST_Quantile — Berechnet die Quantile eines Rasters oder einer Rastercoverage Tabelle im Kontext von Stichproben oder Bevölkerung. Dadurch kann untersucht werden, ob ein Wert bei 25%, 50% oder 75% Perzentil des Rasters liegt.

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);

Beschreibung

Berechnet die Quantile eines Rasters oder einer Rastercoverage Tabelle im Kontext von Stichproben oder Bevölkerung. Dadurch kann untersucht werden, ob ein Wert bei 25%, 50% oder 75% Perzentil des Rasters liegt.

Note
Wenn `exclude_nodata_value` auf FALSE gesetzt ist, werden auch die Pixel mit NODATA Werten gezählt.

Changed: 3.1.0 Removed ST_Quantile(table_name, column_name) variant.

Verfügbarkeit: 2.0.0

Beispiele

```
UPDATE dummy_rast SET rast = ST_SetBandNoDataValue(rast,249) WHERE rid=2;
--Example will consider only pixels of band 1 that are not 249 and in named quantiles --

SELECT (pvq).*
FROM (SELECT ST_Quantile(rast, ARRAY[0.25,0.75]) As pvq
    FROM dummy_rast WHERE rid=2) As foo
    ORDER BY (pvq).quantile;

 quantile | value
----------+-------
     0.25 |   253
     0.75 |   254

SELECT ST_Quantile(rast, 0.75) As value
    FROM dummy_rast WHERE rid=2;

value
------
  254
```

```
--real live example.  Quantile of all pixels in band 2 intersecting a geometry
SELECT rid, (ST_Quantile(rast,2)).* As pvc
    FROM o_4_boston
        WHERE ST_Intersects(rast,
            ST_GeomFromText('POLYGON((224486 892151,224486 892200,224706 892200,224706  ↩
                892151,224486 892151))',26986)
            )
ORDER BY value, quantile,rid
;


 rid | quantile | value
-----+----------+-------
   1 |        0 |     0
   2 |        0 |     0
  14 |        0 |     1
  15 |        0 |     2
  14 |     0.25 |    37
   1 |     0.25 |    42
  15 |     0.25 |    47
   2 |     0.25 |    50
  14 |      0.5 |    56
   1 |      0.5 |    64
  15 |      0.5 |    66
   2 |      0.5 |    77
  14 |     0.75 |    81
  15 |     0.75 |    87
   1 |     0.75 |    94
   2 |     0.75 |   106
```

```
  14 |         1 |   199
   1 |         1 |   244
   2 |         1 |   255
  15 |         1 |   255
```

Siehe auch

ST_Count, ST_SummaryStats, ST_SummaryStatsAgg, ST_SetBandNoDataValue

10.9.5 ST_SummaryStats

ST_SummaryStats — Gibt eine zusammenfassende Statistik aus, bestehend aus der Anzahl, der Summe, dem arithmetischen Mittel, der Standardabweichung, dem Minimum und dem Maximum der Werte eines Rasterbandes oder eines Rastercoverage. Wenn kein Band angegeben ist, wird Band 1 angenommen.

Synopsis

summarystats **ST_SummaryStats**(raster rast, boolean exclude_nodata_value);
summarystats **ST_SummaryStats**(raster rast, integer nband, boolean exclude_nodata_value);

Beschreibung

Gibt summarystats aus, bestehend aus der Anzahl, der Summe, dem arithmetischen Mittel, der Standardabweichung, dem Minimum und dem Maximum der Werte eines Rasterbandes oder eines Rastercoverage. Wenn kein Band angegeben ist, wird Band 1 angenommen.

Note
Standardmäßig werden nur jene Pixelwerte berücksichtigt, die nicht den Wert `NODATA` haben. Setzen Sie bitte `exclude_nodata_value` auf FALSE um die Anzahl sämtlicher Pixel zu erhalten.

Note
Standardmäßig werden alle Pixel abgetastet. Um eine schnellere Rückmeldung zu bekommen, können Sie den Parameter `sample_percent` auf einen Wert kleiner als 1 setzen.

Changed: 3.1.0 ST_SummaryStats(rastertable, rastercolumn, ...) variants are removed. Use ST_SummaryStatsAgg instead.

Verfügbarkeit: 2.0.0

Beispiel: Einzelne Rasterkachel

```
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
```

Beispiel: Zusammenfassen der Pixel, die interessante Bauwerke schneiden

Dieses Beispiel benötigte 574ms in PostGIS unter Windows 64-Bit, mit allen Bauwerken und Luftbildkacheln von Boston (Kacheln jeweils 150x150 Pixel ~ 134.000 Kacheln; ~ 102.000 Datensätze mit Bauwerken)

```
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
```

Beispiel: Rastercoverage

```
-- stats for each band --
SELECT band, (stats).*
FROM (SELECT band, ST_SummaryStats('o_4_boston','rast', band) As stats
    FROM generate_series(1,3) As band) As foo;

 band |  count  |  sum   |       mean       |      stddev      | min | max
------+---------+--------+------------------+------------------+-----+-----
    1 | 8450000 | 725799 | 82.7064349112426 | 45.6800222638537 |   0 | 255
    2 | 8450000 | 700487 | 81.4197705325444 | 44.2161184161765 |   0 | 255
    3 | 8450000 | 575943 |  74.682739408284 | 44.2143885481407 |   0 | 255

-- For a table -- will get better speed if set sampling to less than 100%
-- Here we set to 25% and get a much faster answer
SELECT band, (stats).*
FROM (SELECT band, ST_SummaryStats('o_4_boston','rast', band,true,0.25) As stats
    FROM generate_series(1,3) As band) As foo;

 band |  count  |  sum   |       mean       |      stddev      | min | max
------+---------+--------+------------------+------------------+-----+-----
    1 | 2112500 | 180686 | 82.6890480473373 | 45.6961043857248 |   0 | 255
    2 | 2112500 | 174571 |  81.448503668639 | 44.2252623171821 |   0 | 255
    3 | 2112500 | 144364 | 74.6765884023669 | 44.2014869384578 |   0 | 255
```

Siehe auch

summarystats, ST_SummaryStatsAgg, ST_Count, ST_Clip

10.9.6 ST_SummaryStatsAgg

ST_SummaryStatsAgg — Aggregatfunktion. Gibt eine zusammenfassende Statistik aus, die aus der Anzahl, der Summe, dem arithmetischen Mittel, dem Minimum und dem Maximum der Werte eines bestimmten Bandes eines Rastersatzes besteht. Wenn kein Band angegeben ist, wird Band 1 angenommen.

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);

Beschreibung

Gibt summarystats aus, bestehend aus der Anzahl, der Summe, dem arithmetischen Mittel, der Standardabweichung, dem Minimum und dem Maximum der Werte eines Rasterbandes oder eines Rastercoverage. Wenn kein Band angegeben ist, wird Band 1 angenommen.

Note
Standardmäßig werden nur jene Pixelwerte berücksichtigt, die nicht `NODATA` sind. Setzen Sie bitte `exclude_nodata_value` auf FALSE um die Anzahl sämtlicher Pixel zu erhalten.

Note
Standardmäßig werden alle Pixel abgetastet. Um eine schnellere Rückmeldung zu bekommen, können Sie den Parameter `sample_percent` auf einen Wert zwischen 0 und 1 setzen.

Verfügbarkeit: 2.2.0

Beispiele

```
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)
```

Siehe auch

summarystats, ST_SummaryStats, ST_Count, ST_Clip

10.9.7 ST_ValueCount

ST_ValueCount — Gibt Datensätze aus, die den Zellwert und die Anzahl der Pixel eines Rasterbandes (oder Rastercoverageban-des) für gegebene Werte enthalten. Wenn kein Band angegeben ist, wird Band 1 angenommen. Pixel mit dem Wert NODATA werden standardmäßig nicht gezählt; alle anderen Pixelwerte des Bandes werden ausgegeben und auf die nächste Ganzzahl gerundet.

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);

Beschreibung

Gibt Datensätze mit den Spalten `value` und `count` aus, welche die Pixelwerte und die Anzahl der Pixel im angegebenen Band der Rasterkachel oder des Rastercoverage enthalten.

Wenn kein Band angegeben ist, wird `nband` standardmäßig auf 1 gesetzt. Wenn keine `searchvalues` angegeben sind, werden alle Pixelwerte des Rasters oder des Rastercoverage ausgeben. Wenn nur ein Suchwert angegeben ist, wird eine Ganzzahl ausgegeben - anstelle von Datensätzen mit der Pixelanzahl eines jeden Pixelwertes für das Bandes.

Note
Wenn `exclude_nodata_value` auf FALSE gesetzt ist, werden auch die Pixel mit NODATA Werten gezählt.

Verfügbarkeit: 2.0.0

Beispiele

```
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 coount 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
```

Siehe auch

ST_Count, ST_SetBandNoDataValue

10.10 Rastereingabe

10.10.1 ST_RastFromWKB

ST_RastFromWKB — Gibt einen Rasterwert von einer Well-known-Binary (WKB) Darstellung eines Rasters zurück.

Synopsis

raster **ST_RastFromWKB**(bytea wkb);

Beschreibung

Gibt einen Raster von einem Raster in der Well-known-Binary (WKB) Darstellung zurück.

Verfügbarkeit: 2.5.0

Beispiele

```
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\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
```

Siehe auch

ST_MetaData, ST_RastFromHexWKB, ST_AsBinary/ST_AsWKB, ST_AsHexWKB

10.10.2 ST_RastFromHexWKB

ST_RastFromHexWKB — Gibt einen Rasterwert von einer Well-known-Binary (WKB) Hex-Darstellung eines Rasters zurück.

Synopsis

raster **ST_RastFromHexWKB**(text wkb);

Beschreibung

Gibt einen Raster von einem Raster in der Well-known-Binary (WKB) Hex-Darstellung zurück.

Verfügbarkeit: 2.5.0

Beispiele

```
SELECT (ST_Metadata(
    ST_RastFromHexWKB(
        '0100000000000000000000004000000000000008400000000000000 ↩
            E03F000000000000E03F0000000000000000000000000000000A0000000A001400'
    )
)).* AS metadata;
```

```
 upperleftx | upperlefty | width | height | scalex | scaley | skewx | skewy | srid |  ↩
    numbands
------------+------------+-------+--------+--------+--------+-------+-------+------+---------- ↩

        0.5 |        0.5 |    10 |     20 |      2 |      3 |     0 |     0 |   10 |  ↩
                  0
```

Siehe auch

ST_MetaData, ST_RastFromWKB, ST_AsBinary/ST_AsWKB, ST_AsHexWKB

10.11 Ausgabe von Rastern

10.11.1 ST_AsBinary/ST_AsWKB

ST_AsBinary/ST_AsWKB — Gibt die Well-known-Binary (WKB) Darstellung eines Rasters zurück.

Synopsis

bytea **ST_AsBinary**(raster rast, boolean outasin=FALSE);
bytea **ST_AsWKB**(raster rast, boolean outasin=FALSE);

Beschreibung

Gibt den Raster in binärer Darstellung zurück. Wenn `outasin` TRUE ist, werden Bänder die außerhalb der Datenbank liegen (out-db) gleich behandelt wie Bänder, die direkt in der Datenbank gespeichert (in-db) sind. Siehe "raster/doc/RFC2-WellKnownBinaryFormat" im Quellverzeichnis von PostGIS für Details zu dieser Darstellung.

Dies ist nützlich, um mit einem "binary" Cursor die Daten direkt als Binärdatei von der Datenbank abzurufen - ohne Konvertierung in eine Textdarstellung.

Note
Standardmäßig ist in der WKB-Ausgabe der Pfad der out-db Bänder enthalten. Wenn ein Client keinen Zugriff auf die Rasterdatei eines "out-db" Bandes hat, können Sie `outasin` auf TRUE setzen.

Erweiterung: 2.1.0 `outasin` hinzugefügt

Erweiterung: 2.5.0 `ST_AsWKB` hinzugefügt

Beispiele

```
SELECT ST_AsBinary(rast) As rastbin FROM dummy_rast WHERE rid=1;

                     rastbin
--------------------------------------------------------------------------------
\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\00
```

Siehe auch

ST_RastFromWKB, ST_AsHexWKB

10.11.2 ST_AsHexWKB

ST_AsHexWKB — Gibt die Well-known-Binary (WKB) Hex-Darstellung eines Rasters zurück.

Synopsis

bytea **ST_AsHexWKB**(raster rast, boolean outasin=FALSE);

Beschreibung

Gibt den Raster in binärer Hex-Darstellung zurück. Wenn `outasin` TRUE ist, werden Bänder die außerhalb der Datenbank liegen (out-db) gleich behandelt wie Bänder, die direkt in der Datenbank gespeichert (in-db) sind. Siehe "raster/doc/RFC2-WellKnownBinaryFormat" im Quellverzeichnis von PostGIS für Details zu dieser Darstellung.

> **Note**
> Standardmäßig ist in der Hex-WKB-Ausgabe der Pfad der out-db Bänder enthalten. Wenn ein Client keinen Zugriff auf die Rasterdatei eines "out-db" Bandes hat, können Sie `outasin` auf TRUE setzen.

Verfügbarkeit: 2.5.0

Beispiele

```
SELECT ST_AsHexWKB(rast) As rastbin FROM dummy_rast WHERE rid=1;

                                                        st_ashexwkb
-----------------------------------------------------------------------------------------------------

 0100000000000000000000004000000000000008400000000000000 ↩
     E03F000000000000E03F00000000000000000000000000000000A0000000A001400
```

Siehe auch

ST_RastFromHexWKB, ST_AsBinary/ST_AsWKB

10.11.3 ST_AsGDALRaster

ST_AsGDALRaster — Gibt die Rasterkachel in dem ausgewiesenen Rasterformat von GDAL aus. Sie können jedes Rasterformat angeben, das von Ihrer Bibliothek unterstützt wird. Um eine Liste mit den unterstützten Formaten auszugeben, verwenden Sie bitte ST_GDALDrivers().

Synopsis

bytea **ST_AsGDALRaster**(raster rast, text format, text[] options=NULL, integer srid=sameassource);

Beschreibung

Gibt die Rasterkachel im dem ausgewiesenen Format zurück. Die Übergabewerte sind:

- `format` das Format das abgerufen wird. Dies ist abhängig von den Treibern die mit Ihrer Bibliothek "libgdal" kompiliert wurden. Üblicherweise stehen 'JPEG', 'GTIff' und 'PNG' zur Verfügung. Verwenden Sie bitte ST_GDALDrivers, um eine Liste der von Ihrer Bibliothek unterstützten Formate zu erhalten.
- `options` ein Textfeld mit Optionen für GDAL. Gültige Optionen sind formatabhängig. Siehe GDAL Raster format options für weitere Details.
- `srs` Der Text von "proj4text" oder "srtext" (aus der Tabelle "spatial_ref_sys"), der in das Rasterbild eingebettet werden soll

Verfügbarkeit: 2.0.0 - GDAL >= 1.6.0.

Beispiel für eine JPEG Ausgabe; mehrere Kacheln als einzelner Raster

```
SELECT ST_AsGDALRaster(ST_Union(rast), 'JPEG', ARRAY['QUALITY=50']) As rastjpg
FROM dummy_rast
WHERE rast && ST_MakeEnvelope(10, 10, 11, 11);
```

Raster mit dem "Large Object Support" von PostgreSQL exportieren

Eine Möglichkeit um Raster in einem anderen Format zu exportieren ist die Verwendung der PostgreSQL Large Object Export Functions. Wir erweitern das vorherige Beispiel mit einem Export. Dafür benötigen Sie Administratorrechte für die Datenbank, da serverseitige Funktionen "Large Objects" verwendet werden. Es kann auch auf einen Server im Netzwerk exportiert werden. Wenn Sie einen lokalen Export benötigen, verwenden Sie bitte die äquivalenten "lo_"-Funktionen von psql, welche auf das lokale Dateisystem anstatt auf das des Servers exportieren.

```
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;
```

Beispiel für die Ausgabe von GTIFF

```
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;
```

Siehe auch

Section 9.3, ST_GDALDrivers, ST_SRID

10.11.4 ST_AsJPEG

ST_AsJPEG — Gibt die ausgewählten Bänder der Rasterkachel als einzelnes Bild (Byte-Array) im Format "Joint Photographic Exports Group" (JPEG) aus. Wenn kein Band angegeben ist und 1 oder mehr als 3 Bänder ausgewählt wurden, dann wird nur das erste Band verwendet. Wenn 3 Bänder ausgewählt wurden, werden alle 3 Bänder verwendet und auf RGB abgebildet.

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);

Beschreibung

Gibt die ausgewählten Bänder der Rasterkachel als einzelnes Bild im Format "Joint Photographic Exports Group" (JPEG) aus. Sie können ST_AsGDALRaster verwenden, wenn Sie in weniger gebräuchliche Rasterformate exportieren wollen. Wenn kein Band angegeben ist und 1 oder mehr als 3 Bänder ausgewählt wurden, dann wird nur das erste Band verwendet. Wenn 3 Bänder ausgewählt wurden, werden alle 3 Bänder verwendet. Die Funktion hat viele Varianten mit vielen Optionen. Diese sind unterhalb angeführt:

- `nband` für den Export einzelner Bänder.
- `nbands` Ein Feld mit den Bändern die exportiert werden sollen (bei JPEG maximal 3). Die Reihenfolge der Bänder ist RBG; z.B. ARRAY[3,2,1] bedutet, dass Band 3 auf Rot, Band 2 auf Grün und Band 1 auf Blau abgebildet wird.
- `quality` Eine Zahl zwischen 0 und 100. Umso höher die Zahl ist, umso schärfer ist das Bild.
- `options` Ein Textfeld mit GDAL Optionen für JPEG (siehe "create_options" für JPEG unter ST_GDALDrivers). Gültige Optionen für JPEG sind `PROGRESSIVE` ON oder OFF und `QUALITY` zwischen 0 und 100 mit dem Standardwert 75. Siehe GDAL Raster format options für weitere Details.

Verfügbarkeit: 2.0.0 - GDAL >= 1.6.0.

Beispiele: Ausgabe

```
-- output first 3 bands 75% quality
SELECT ST_AsJPEG(rast) As rastjpg
    FROM dummy_rast WHERE rid=2;

-- output only first band as 90% quality
SELECT ST_AsJPEG(rast,1,90) As rastjpg
    FROM dummy_rast WHERE rid=2;

-- output first 3 bands (but make band 2 Red, band 1 green, and band 3 blue, progressive ↩
    and 90% quality
SELECT ST_AsJPEG(rast,ARRAY[2,1,3],ARRAY['QUALITY=90','PROGRESSIVE=ON']) As rastjpg
    FROM dummy_rast WHERE rid=2;
```

Siehe auch

Section 9.3, ST_GDALDrivers, ST_AsGDALRaster, ST_AsPNG, ST_AsTIFF

10.11.5 ST_AsPNG

ST_AsPNG — Gibt die ausgewählten Bänder der Rasterkachel als einzelnes, übertragbares Netzwerkgraphik (PNG) Bild (Byte-Feld) aus. Wenn der Raster 1,3 oder 4 Bänder hat und keine Bänder angegeben sind, dann werden alle Bänder verwendet. Wenn der Raster 2 oder mehr als 4 Bänder hat und keine Bänder angegeben sind, dann wird nur Band 1 verwendet. Die Bänder werden in den RGB- oder den RGBA-Raum abgebildet.

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);

Beschreibung

Gibt die ausgewählten Bänder des Raster als einzelnes Bild im Format "Portable Network Graphics Image" (PNG) aus. Sie können ST_AsGDALRaster verwenden, wenn Sie in weniger gebräuchliche Rasterformate exportieren wollen. Wenn kein Band angegeben ist, werden die ersten 3 Bänder exportiert. Wenn `SRID` nicht angegeben ist, wird die SRID des Ausgangsraster verwendet. Die Funktion hat viele Varianten mit vielen Optionen. Diese sind unterhalb angeführt:

- `nband` für den Export einzelner Bänder.
- `nbands` Ein Feld mit den Bändern die exportiert werden sollen (bei PNG maximal 4). Die Reihenfolge der Bänder ist RBGA; z.B. ARRAY[3,2,1] bedutet, dass Band 3 auf Rot, Band 2 auf Grün und Band 1 auf Blau abgebildet wird.
- `compression` Eine Zahl zwischen 1 und 9. Umso höher die Zahl umso größer die Komprimierung.
- `options` Ein Textfeld mit GDAL Optionen für PNG (siehe "create_options" für PNG unter ST_GDALDrivers). Für PNG gibt es nur eine gültige Option "ZLEVEL" (wieviel Zeit für die Komprimierung aufgewendet werden soll - die Standardeinstellung ist 6); z.B.: ARRAY['ZLEVEL=9']. Ein WORLDFILE ist nicht erlaubt, da die Funktion sonst zwei Ausgaben machen müsste. Siehe GDAL Raster format options für weitere Details.

Verfügbarkeit: 2.0.0 - GDAL >= 1.6.0.

Beispiele

```
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;
```

Siehe auch

ST_AsGDALRaster, ST_ColorMap, ST_GDALDrivers, Section 9.3

10.11.6 ST_AsTIFF

ST_AsTIFF — Gibt die ausgewählten Bänder des Raster als einzelnes TIFF Bild (Byte-Feld) zurück. Wenn kein Band angegeben ist oder keines der angegebenen Bänder im Raster existiert, werden alle Bänder verwendet.

Synopsis

bytea **ST_AsTIFF**(raster rast, text[] options='', integer srid=sameassource);
bytea **ST_AsTIFF**(raster rast, text compression='', integer srid=sameassource);
bytea **ST_AsTIFF**(raster rast, integer[] nbands, text compression='', integer srid=sameassource);
bytea **ST_AsTIFF**(raster rast, integer[] nbands, text[] options, integer srid=sameassource);

Beschreibung

Gibt die ausgewählten Bänder des Raster als einzelnes Bild im Format " Tagged Image File Format" (TIFF) aus. Wenn kein Band angegeben ist, wird versucht alle Bänder zu verwenden. Diese Funktion ist ein Adapter für ST_AsGDALRaster. Verwenden Sie bitte ST_AsGDALRaster, wenn Sie in weniger gebräuchliche Rasterformate exportieren wollen. Wenn kein SRS-Text für die Georeferenz vorhanden ist, wird das Koordinatenreferenzsystem des Ausgangsraster verwendet. Die Funktion hat viele Varianten mit vielen Optionen. Diese sind unterhalb angeführt:

- `nbands` Ein Feld mit den Bändern die exportiert werden sollen (bei PNG maximal 3). Die Reihenfolge der Bänder ist RBG; z.B. ARRAY[3,2,1] bedutet, dass Band 3 auf Rot, Band 2 auf Grün und Band 1 auf Blau abgebildet wird.
- `compression` Ein Ausdruck für die Art der Datenkompression -- JPEG90 (oder eine andere Prozentangabe), LZW, JPEG, DEFLATE9.
- `options` Ein Textfeld mit GDAL Optionen für die Erstellung eines GTiff (siehe "create_options" für GTiff unter ST_GDALDrivers oder GDAL Raster format options für weitere Details.
- `srid` Die SRID des Koordinatenreferenzsystems des Raster. Wird als Information zur Georeferenzierung verwendet.

Verfügbarkeit: 2.0.0 - GDAL >= 1.6.0.

Beispiele: 90%ige JPEG Komprimierung

```
SELECT ST_AsTIFF(rast, 'JPEG90') As rasttiff
FROM dummy_rast WHERE rid=2;
```

Siehe auch

ST_GDALDrivers, ST_AsGDALRaster, ST_SRID

10.12 Raster Processing: Map Algebra

10.12.1 ST_Clip

ST_Clip — Schneidet den Raster nach der Eingabegeometrie. Wenn die Bandnummer nicht angegeben ist, werden alle Bänder bearbeitet. Wenn `crop` nicht angegeben oder TRUE ist, wird der Ausgaberaster abgeschnitten.

Synopsis

raster **ST_Clip**(raster rast, integer[] nband, geometry geom, double precision[] nodataval=NULL, boolean crop=TRUE);
raster **ST_Clip**(raster rast, integer nband, geometry geom, double precision nodataval, boolean crop=TRUE);
raster **ST_Clip**(raster rast, integer nband, geometry geom, boolean crop);
raster **ST_Clip**(raster rast, geometry geom, double precision[] nodataval=NULL, boolean crop=TRUE);
raster **ST_Clip**(raster rast, geometry geom, double precision nodataval, boolean crop=TRUE);
raster **ST_Clip**(raster rast, geometry geom, boolean crop);

Beschreibung

Gibt einen Raster aus, der nach der Eingabegeometrie `geom` ausgeschnitten wird. Wird kein Band angegeben, so werden alle Bänder bearbeitet.

Raster die aus einer Operation mit ST_Clip resultieren, müssen in den Bereichen wo sie ausgeschnitten werden, einen NODATA-Wert für jedes Band aufweisen. Wenn keine Werte übergeben werden und für den Ausgangsraster kein NODATA-Wert festgelegt wurde, dann werden die NODATA-Werte des Zielrasters auf ST_MinPossibleValue(ST_BandPixelType(rast, band)) gesetzt. Wenn die Anzahl der NODATA-Werte in dem übergebenen Feld kleiner als die Anzahl der Bänder ist, wird für die restlichen Bänder der letzte Wert des Feldes verwendet. Wenn die Anzahl der NODATA-Werte größer als die Anzahl der Bänder ist, so werden die zusätzlichen NODATA-Werte übergangen. Alle Varianten, die ein Feld mit NODATA-Werten akzeptieren, nehmen auch einen einzelnen Wert für alle Bänder entgegen.

Wenn `crop` nicht angegeben ist, wird TRUE angenommen. Dies bedeutet, dass der Ergebnisraster auf die Ausdehnung der Verschneidung von `geom` und `rast` zugeschnitten wird. Wenn `crop` auf FALSE gesetzt ist, hat der neue Raster dieselbe Ausdehnung wie `rast`.

Verfügbarkeit: 2.0.0

Erweiterung: 2.1.0 neu geschrieben in C

Diese Beispiele nutzen Luftbilddaten aus Massachusetts, die von MassGIS Aerial Orthos heruntergeladen werden können. Die Koordinaten liegen in "Massachusetts State Plane Meters" vor.

Beispiele: 1 Band ausschneiden

```
-- 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
```

Die gesamte Rasterkachel vor dem Ausschneiden

Nach dem Ausschneiden

Beispiele: 1 Band ohne "crop" ausschneiden und die anderen Bänder ungeändert erneut hinzufügen

```
-- 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;
```

Die gesamte Rasterkachel vor dem Ausschneiden

Nach dem Abschneiden - unwirklich

Beispiele: Alle Bänder ausschneiden

```
-- 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;
```

Die gesamte Rasterkachel vor dem Ausschneiden

Nach dem Ausschneiden

Siehe auch

ST_AddBand, ST_MapAlgebra (callback function version), ST_Intersection

10.12.2 ST_ColorMap

ST_ColorMap — Erzeugt aus einem bestimmten Band des Ausgangsrasters einen neuen Raster mit bis zu vier 8BUI-Bändern (Grauwert, RGB, RGBA). Wenn kein Band angegeben ist, wird Band 1 angenommen.

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);

Beschreibung

Wendet eine `colormap` auf das Band `nband` von `rast` an, wodurch ein neuer Raster aus bis zu vier 8BUI-Bändern erstellt wird. Die Anzahl der 8BUI-Bänder des neuen Raster wird durch die Anzahl der Farbkomponenten bestimmt, die in der `colormap` definiert sind.

Wenn `nband` nicht angegeben ist, wird Band 1 angenommen.

`colormap` kann ein Schlüsselwort, ein vordefiniertes Farbschema, oder Zeilen in denen der Zellwert und die Farbkomponenten festgelegt sind.

Gültige, vordefinierte Schlüsselwörter von `colormap`:

- `grayscale` oder `greyscale` für Grauabstufungen in einem 8BUI-Rasterband.

- `pseudocolor` für vier 8BUI-Rasterbänder (RGBA) mit Farbverläufen von Blau zu Grün zu Rot.
- `fire` für vier 8BUI-Rasterbänder (RGBA) mit Farbverläufen von Blau zu Rot zu blassem Gelb.
- `bluered` für vier 8BUI-Rasterbänder (RGBA) mit Farbverläufen von Blau zu blassem Weiß zu Rot.

Anwender können mehrere Einträge (einen pro Zeile) an `colormap` übergeben, um bestimmte Farbschemata zu spezifizieren. Üblicherweise besteht jeder Eintrag aus fünf Werten: der Pixelwert und die zugehörigen Rot-, Grün-, Blau- und Alpha-Komponenten (Farbkomponenten zwischen 0 und 255). Anstelle der Pixelwerte können auch Prozentwerte verwendet werden, wobei 0% und 100% die minimalen und maximalen Werte des Rasterbandes sind. Die Werte können durch Beistriche (','), Tabulatoren, Doppelpunkte (':') und/oder durch Leerzeichen getrennt werden. Für die NODATA-Werte kann der Pixelwert mit *nv*, *NULL* oder auf *NODATA* angegeben werden. Ein Beispiel finden Sie unterhalb.

```
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
```

Die Syntax von `colormap` ist ähnlich wie bei dem Modus "color-relief" bei gdaldem von GDAL.

Gültige Schlüsselwörter für `method`:

- `INTERPOLATE` verwendet eine lineare Interpolation um einen kontinuierlichen Übergang der Farben zwischen den Pixelwerten zu erhalten
- `EXACT` genaue Entsprechung der Pixelwerte mit der "colormap". Pixel, deren Werte keinen Eintrag in "colormap" haben, werden mit "0 0 0 0" (RGBA) eingefärbt
- `NEAREST` verwendet die Einträge der "colormap", deren Wert dem Pixelwert am nächsten kommt

---

Note
Eine großartige Hilfestellung für "colormaps" bietet ColorBrewer

---

Warning
Bei den resultierenden Bänder des neuen Rasters sind keine NODATA-Werte gesetzt. Verwenden Sie bitte ST_SetBandNoDataValue um einen NODATA-Wert zu setzen, falls dieser benötigt wird.

Verfügbarkeit: 2.1.0

Beispiele

Eine "Junk"-Tabelle zum Herumspielen

```
-- 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)
```

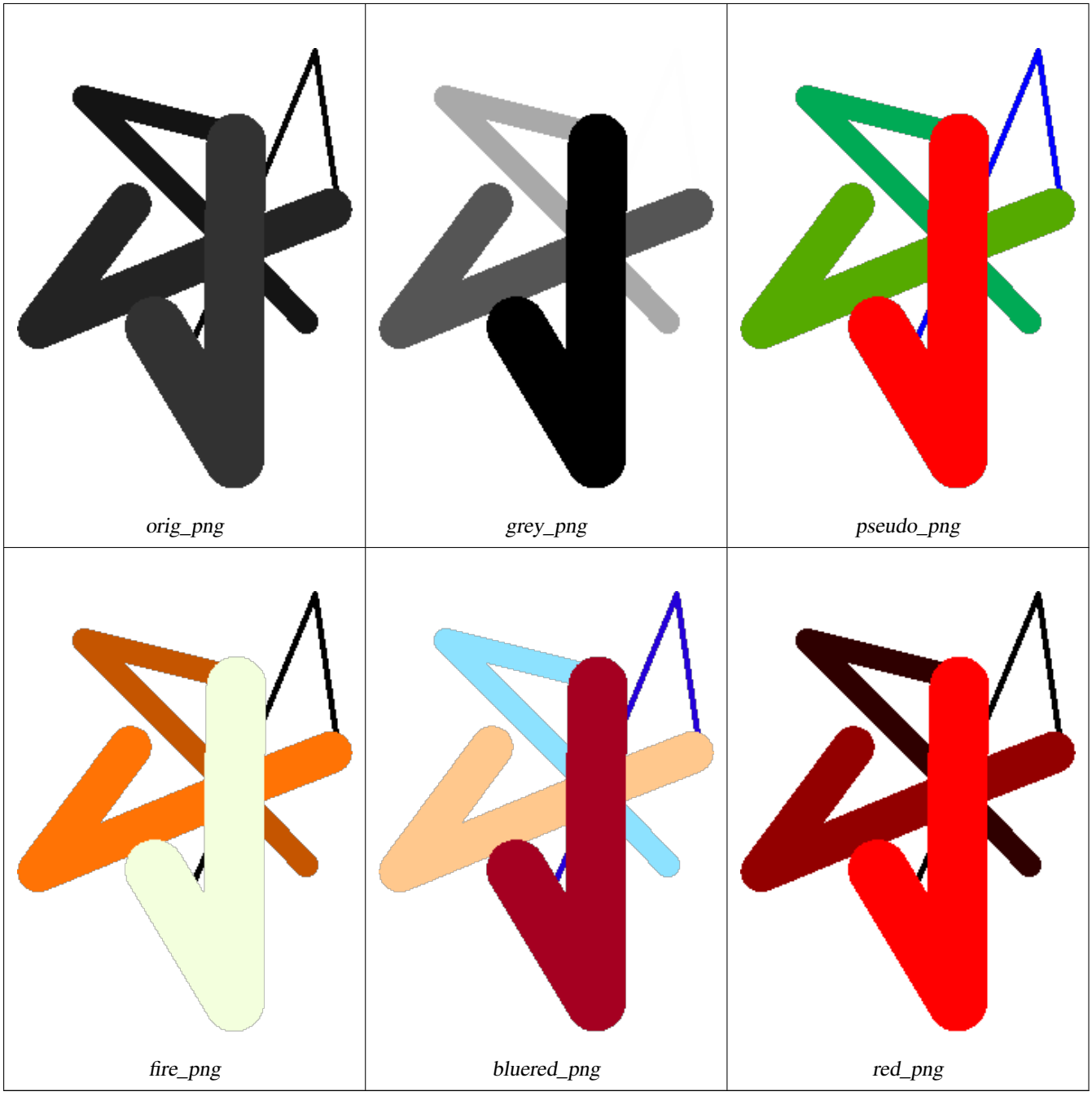
```
FROM (
    SELECT
        ST_AsRaster(
            ST_Rotate(
                ST_Buffer(
                    ST_GeomFromText('LINESTRING(0 2,50 50,150 150,125 50)'),
                    i*2
                ),
                pi() * i * 0.125, ST_Point(50,50)
            ),
            ref.rast, '8BUI'::text, 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
```

Beispiele: Vergleich verschiedener Farbtafeln mit ST_AsPNG

```
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;
```



orig_png | *grey_png* | *pseudo_png*

fire_png | *bluered_png* | *red_png*

Siehe auch

ST_AsPNG, ST_AsRaster ST_MapAlgebra (callback function version), ST_Grayscale ST_NumBands, ST_Reclass, ST_SetBandNoDat ST_Union

10.12.3 ST_Grayscale

ST_Grayscale — Erzeugt einen neuen Raster mit einem 8BUI-Band aus dem Ausgangsrasters und den angegebenen Bändern für Rot, Grün und Blau

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);

Beschreibung

Erzeugt einen Raster mit einem 8BUI-Band aus drei Eingabebändern (von einem oder mehreren Raster). Bänder die nicht den Pixeltyp 8BUI haben werden mit ST_Reclass neu klassifiziert.

Note
Diese Funktion unterscheidet sich insofern von ST_ColorMap mit dem Schlüsselwort `grayscale`, da ST_ColorMap lediglich ein Band bearbeitet, während diese Funktion drei Bänder für RGB erwartet. Diese Funktion verwendet folgende Gleichung zur Konvertierung von RGB auf Graustufen: 0.2989 * RED + 0.5870 * GREEN + 0.1140 * BLUE

Verfügbarkeit: 2.5.0

Beispiele: Variante 1

```
SET postgis.gdal_enabled_drivers = 'ENABLE_ALL';
SET postgis.enable_outdb_rasters = True;

WITH apple AS (
    SELECT ST_AddBand(
        ST_MakeEmptyRaster(350, 246, 0, 0, 1, -1, 0, 0, 0),
        '/tmp/apple.png'::text,
        NULL::int[]
    ) AS rast
)
SELECT
    ST_AsPNG(rast) AS original_png,
    ST_AsPNG(ST_Grayscale(rast)) AS grayscale_png
FROM apple;
```

original_png

grayscale_png

Beispiele: Variante 2

```
SET postgis.gdal_enabled_drivers = 'ENABLE_ALL';
SET postgis.enable_outdb_rasters = True;

WITH apple AS (
    SELECT ST_AddBand(
        ST_MakeEmptyRaster(350, 246, 0, 0, 1, -1, 0, 0, 0),
        '/tmp/apple.png'::text,
        NULL::int[]
    ) AS rast
)
SELECT
    ST_AsPNG(rast) AS original_png,
    ST_AsPNG(ST_Grayscale(
        ARRAY[
            ROW(rast, 1)::rastbandarg, -- red
            ROW(rast, 2)::rastbandarg, -- green
            ROW(rast, 3)::rastbandarg, -- blue
        ]::rastbandarg[]
    )) AS grayscale_png
FROM apple;
```

Siehe auch

ST_AsPNG, ST_Reclass, ST_ColorMap

10.12.4 ST_Intersection

ST_Intersection — Gibt Geometry-PixelValue Paare, oder einen Raster aus, der durch die Schnittmenge der beiden Raster bestimmt wird, oder durch die geometrische Verschneidung einer Vektorisierung des Rasters mit einem geometrischen Datentyp.

Synopsis

setof geomval **ST_Intersection**(geometry geom, raster rast, integer band_num=1);
setof geomval **ST_Intersection**(raster rast, geometry geom);
setof geomval **ST_Intersection**(raster rast, integer band, geometry geomin);
raster **ST_Intersection**(raster rast1, raster rast2, double precision[] nodataval);
raster **ST_Intersection**(raster rast1, raster rast2, text returnband, double precision[] nodataval);
raster **ST_Intersection**(raster rast1, integer band1, raster rast2, integer band2, double precision[] nodataval);
raster **ST_Intersection**(raster rast1, integer band1, raster rast2, integer band2, text returnband, double precision[] nodataval);

Beschreibung

Gibt Geometry-PixelValue Paare, oder einen Raster aus, der durch die Schnittmenge der beiden Raster bestimmt wird, oder durch die geometrische Verschneidung einer Vektorisierung des Rasters mit einem geometrischen Datentyp.

Die ersten drei Varianten, die ein "setof geomval" zurückgeben, führen die Berechnungen im Vektorraum aus. Der Raster wird zuerst in eine Menge von "geomval"-Zeilen vektorisiert (mit ST_DumpAsPolygons) und anschließend mit der Geometrie über die PostGIS Funktion ST_Intersection(geometry, geometry) verschnitten. Wenn die verschnittene Geometrie nur aus NODATA Werten besteht, wird eine leere Geometrie zurückgegeben. Diese wird üblicherweise durch die richtige Verwendung von ST_Intersects in der WHERE-Klausel ausgeschlossen.

Sie können auf die Geometriebestandteile und die Werte der erzeugten "geomvals" zugreifen, indem Sie diese mit Klammern versehen und '.geom' oder '.val' am Ende des Ausdrucks hinzufügen; z.B. (ST_Intersection(rast, geom)).geom

Die anderen Varianten, welche einen Raster zurückgeben, führen die Berechnungen im Rasterraum aus. Sie verwenden die Version mit den zwei Rastern von ST_MapAlgebraExpr um die Verschneidung durchzuführen.

Die Ausdehnung des resultierenden Raster entspricht der Ausdehnung des geometrischen Durchschnitts der beiden Raster. Der resultierende Raster enthält die Bänder 'BAND1', 'BAND2' und 'BOTH', gefolgt von dem Parameter `returnband`. Wenn irgendein Band Bereiche mit NODATA-Werten enthält, so werden diese Bereiche in allen Bändern des resultierenden Raster zu NODATA. Anders ausgedückt, jedes Pixel, das ein Pixel mit NODATA-Wert schneidet wird im Ergebnis selbst zu einem Pixel mit NODATA-Wert.

Raster die aus einer Operation mit ST_Intersection resultieren, müssen in den Bereichen wo sie sich nicht schneiden, einen NODATA-Wert aufweisen. Sie können den NODATA Wert eines jeden resultierenden Bandes festlegen oder ersetzen, indem Sie ein Feld `nodataval[]` übergeben, das einen oder zwei NODATA Werte - 'BAND1', 'BAND2' oder 'BOTH' - enthält. Der erste Wert in dem Feld ersetzt den NODATA Wert im ersten Band, der zweite Wert ersetzt den NODATA Wert im zweiten Band. Wenn für ein übergebenes Band kein NODATA Wert festgelegt wurde und auch keiner als Feld übergeben wurde, dann wird der Wert mit der Funktion ST_MinPossibleValue ausgewählt. Alle Varianten, die ein Feld mit NODATA-Werten akzeptieren, nehmen auch einen einzelnen Wert entgegen, welcher dann auf alle verlangten Bänder übertragen wird.

Bei sämtlichen Varianten wird Band 1 angenommen, wenn keine Bandnummer angegeben ist. Wenn Sie die Verschneidung zwischen einem Raster und einer Geometrie als Raster ausgegeben haben wollen, sehen Sie bitte ST_Clip.

Note
Um über die resultierende Ausdehnung, oder über das was für einen NODATA-Wert zurückgeben werden soll, eine bessere Kontrolle zu haben, können Sie die Variante von ST_MapAlgebraExpr mit den zwei Raster verwenden.

Note
Um eine Verschneidung von einem Rasterband mit einer Geometrie durchzuführen, verwenden Sie bitte ST_Clip. ST_Clip arbeitet mit Raster mit mehreren Bändern und gibt kein Band mit der gerasterten Geometrie zurück.

Note
ST_Intersection sollte in Verbindung mit ST_Intersects und einem Index auf die Rasterspalte und/oder auf die Geometriespalte angewendet werden.

Enhanced: 2.0.0 - Verschneidungsoperation im Rasterraum eingeführt. In Vorgängerversionen von 2.0.0 wurde lediglich die Verschneidung im Vektorraum unterstützt.

Beispiele: Geometrie, Raster -- das Ergebnis sind "geomvals"

```
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
```

Siehe auch

geomval, ST_Intersects, ST_MapAlgebraExpr, ST_Clip, ST_AsText

10.12.5 ST_MapAlgebra (callback function version)

ST_MapAlgebra (callback function version) — Die Version mit der Rückruffunktion - Gibt für einen oder mehrere Eingaberaster einen Raster mit einem Band, den Bandindizes und einer vom Anwender vorgegebenen Rückruffunktion zurück.

Synopsis

raster **ST_MapAlgebra**(rastbandarg[] rastbandargset, regprocedure callbackfunc, text pixeltype=NULL, text extenttype=INTERSECTIO
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 callbackfunc, text pixeltype=NULL,
text extenttype=INTERSECTION, raster customextent=NULL, integer distancex=0, integer distancey=0, text[] VARIADIC user-
args=NULL);
raster **ST_MapAlgebra**(raster rast, integer nband, regprocedure callbackfunc, float8[] mask, boolean weighted, text pixel-
type=NULL, text extenttype=INTERSECTION, raster customextent=NULL, text[] VARIADIC userargs=NULL);

Beschreibung

Gibt für einen oder mehrere Eingaberaster einen Raster mit einem Band, den Bandindizes und einer vom Anwender vorgegebenen Rückruffunktion zurück.

rast,rast1,rast2, rastbandargset Raster die mit der Map Algebra Operation ausgewertet werden.

`rastbandargset` ermöglicht die Anwendung einer Map Algebra Operation auf viele Raster und/oder viele Bänder. Siehe das Beispiel zu Variante 1.

nband, nband1, nband2 Die Nummern der Rasterbänder, die ausgewertet werden sollen. Die Bänder können über "nband" als Integer oder Integer[] angegeben werden. Bei der Variante mit 2 Raster steht "nband1" für die Bänder von "rast1" und nband2 für die Bänder von rast2.

callbackfunc The `callbackfunc` parameter must be the name and signature of an SQL or PL/pgSQL function, cast to a regprocedure. An example PL/pgSQL function example is:

```
CREATE OR REPLACE FUNCTION sample_callbackfunc(value double precision[][][], position ↩
    integer[][], VARIADIC userargs text[])
    RETURNS double precision
    AS $$
    BEGIN
        RETURN 0;
    END;
    $$ LANGUAGE 'plpgsql' IMMUTABLE;
```

The `callbackfunc` must have three arguments: a 3-dimension double precision array, a 2-dimension integer array and a variadic 1-dimension text array. The first argument `value` is the set of values (as double precision) from all input rasters. The three dimensions (where indexes are 1-based) are: raster #, row y, column x. The second argument `position` is the set of pixel positions from the output raster and input rasters. The outer dimension (where indexes are 0-based) is the raster #. The position at outer dimension index 0 is the output raster's pixel position. For each outer dimension, there are two elements in the inner dimension for X and Y. The third argument `userargs` is for passing through any user-specified arguments.

Passing a regprocedure argument to a SQL function requires the full function signature to be passed, then cast to a regprocedure type. To pass the above example PL/pgSQL function as an argument, the SQL for the argument is:

```
'sample_callbackfunc(double precision[], integer[], text[])'::regprocedure
```

Note that the argument contains the name of the function, the types of the function arguments, quotes around the name and argument types, and a cast to a regprocedure.

mask Ein n-dimensionales Feld (Matrix) von Zahlen, das verwendet wird um Zellen für die "Map Algebra"-Rückruffunktion zu filtern. 0 bedeutet, dass ein Nachbarzellwert wie NODATA behandelt werden soll. 1 bedeutet, dass dieser Wert als Datum behandelt werden soll. Wenn der Parameter "weighted" (gewichtet) TRUE ist, dann werden diese Werte als Multiplikatoren für die Pixelwerte in der Nachbarschaft verwendet.

weighted Boolesche Variable (TRUE/FALSE), die angibt ob ein Wert der Maske gewichtet (mit dem ursprünglichen Wert multipliziert) werden soll oder nicht (gilt nur für den ersten, der die Maske übernimmt).

pixeltype Wenn `pixeltype` angegeben ist, dann hat das Band des neuen Rasters diesen Pixeltyp. Wenn für den Pixeltyp NULL oder kein Wert übergeben wird, dann hat das neue Rasterband denselben Pixeltyp wie das angegebene Band des ersten Raster (für die Lagevergleiche: INTERSECTION, UNION, FIRST, CUSTOM), oder wie das spezifizierte Band des entsprechenden Rasters (für die Lagevergleiche:SECOND, LAST). Im Zweifelsfall sollten Sie den `pixeltype` immer angeben.

Der resultierende Pixeltyp des Zielraster muss entweder aus ST_BandPixelType sein, weggelassen oder auf NULL gesetzt werden.

extenttype Mögliche Werte sind INTERSECTION (standardmäßig), UNION, FIRST (standardmäßig für Einzelraster-Varianten), SECOND, LAST, CUSTOM.

customextent Wenn `extentype` CUSTOM ist, dann muss ein Raster für `customextent` übergeben werden. Siehe Beispiel 4 von Variante 1.

distancex Der Abstand in Pixel von der Referenzzelle in X-Richtung. Die Breite der resultierenden Matrix ergibt sich aus `2*distancex + 1`. Wenn nicht angegeben, dann wird nur die Referenzzelle berücksichtigt (keine Nachbarschaft/"neighborhood of 0").

distancey Die Entfernung der Pixel zur Referenzzelle in Y-Richtung. Die Höhe der resultierenden Matrix ergibt sich aus `2*distancey + 1`. Wenn nicht angegeben, dann wird nur die Referenzzelle berücksichtigt (keine Nachbarschaft/"neighborhood of 0").

userargs Der dritte Übergabewert an die Rückruffunktion `callbackfunc` ist ein Feld mit dem Datentyp variadic text. Die an diesen Datentyp angehängten Parameter werden an die `callbackfunc` durchgereicht und sind in dem Übergabewert `userargs` enthalten.

Note
Für nähere Information zu dem Schlüsselwort VARIADIC, sehen Sie bitte die PostgreSQL Dokumentation und den Abschnitt "SQL Functions with Variable Numbers of Arguments" unter Query Language (SQL) Functions.

Note
Der Übergabewert text[] an die Rückruffunktion `callbackfunc` ist auch dann erforderlich, wenn Sie sonst keine Argumente für die Auswertung übergeben.

Variante 1 nimmt ein Feld an `rastbandarg` entgegen, wodurch die Anwendung einer Map Algebra Operation auf mehrere Raster und/oder mehrere Bänder ermöglicht wird. Siehe das Beispiel zu Variante 1.

Die Varianten 2 und 3 führen die Operation auf einem oder mehreren Bändern eines Raster aus. Siehe die Beispiele mit Variante 2 und 3.

Die Variante 4 führt die Operationen an zwei Raster mit einem Band pro Raster aus. Siehe das Beispiel mit Variante 4.

Verfügbarkeit: 2.2.0: Möglichkeit eine Maske hinzuzufügen

Verfügbarkeit: 2.1.0

Beispiele: Variante 1

Ein Raster, ein Band

```
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
```

Ein Raster, mehrere Bänder

```
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
```

Mehrere Raster, mehrere Bänder

```
WITH foo AS (
    SELECT 1 AS rid, ST_AddBand(ST_AddBand(ST_AddBand(ST_MakeEmptyRaster(2, 2, 0, 0, 1, -1, ↩
        0, 0, 0), 1, '16BUI', 1, 0), 2, '8BUI', 10, 0), 3, '32BUI', 100, 0) AS rast UNION  ↩
       ALL
    SELECT 2 AS rid, ST_AddBand(ST_AddBand(ST_AddBand(ST_MakeEmptyRaster(2, 2, 0, 1, 1, -1, ↩
        0, 0, 0), 1, '16BUI', 2, 0), 2, '8BUI', 20, 0), 3, '32BUI', 300, 0) AS rast
)
```

```
SELECT
    ST_MapAlgebra(
        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
```

Ein vollständiges Beispiel mit Coveragekacheln und Nachbarschaft. Diese Abfrage funktioniert nur mit PostgreSQL 9.1 oder höher.

```
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
```

Das selbe Beispiel wie vorher, mit Coveragekacheln und Nachbarschaft, das aber auch mit PostgreSQL 9.0 funktioniert.

```
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;
```

Beispiele: Varianten 2 und 3

Ein Raster, mehrere Bänder

```
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
```

Ein Raster, ein Band

```
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
```

Beispiele: Variante 4

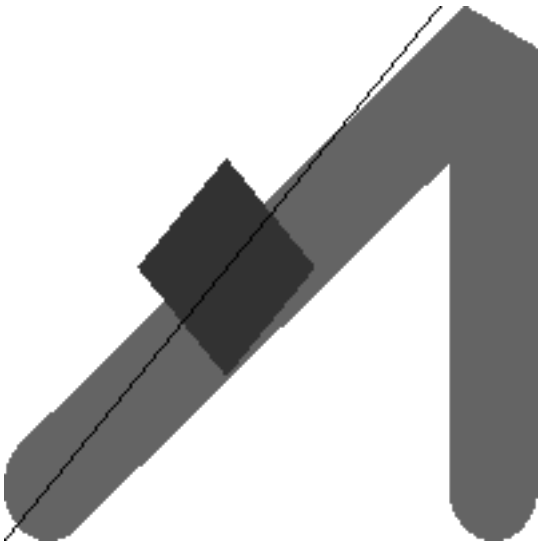
Zwei Raster, zwei Bänder

```
WITH foo AS (
    SELECT 1 AS rid, ST_AddBand(ST_AddBand(ST_AddBand(ST_MakeEmptyRaster(2, 2, 0, 0, 1, -1, ↩
         0, 0, 0), 1, '16BUI', 1, 0), 2, '8BUI', 10, 0), 3, '32BUI', 100, 0) AS rast UNION  ↩
        ALL
    SELECT 2 AS rid, ST_AddBand(ST_AddBand(ST_AddBand(ST_MakeEmptyRaster(2, 2, 0, 1, 1, -1, ↩
         0, 0, 0), 1, '16BUI', 2, 0), 2, '8BUI', 20, 0), 3, '32BUI', 300, 0) AS rast
)
SELECT
    ST_MapAlgebra(
        t1.rast, 2,
        t2.rast, 1,
        'sample_callbackfunc(double precision[], int[], text[])'::regprocedure
    ) AS rast
FROM foo t1
CROSS JOIN foo t2
WHERE t1.rid = 1
    AND t2.rid = 2
```

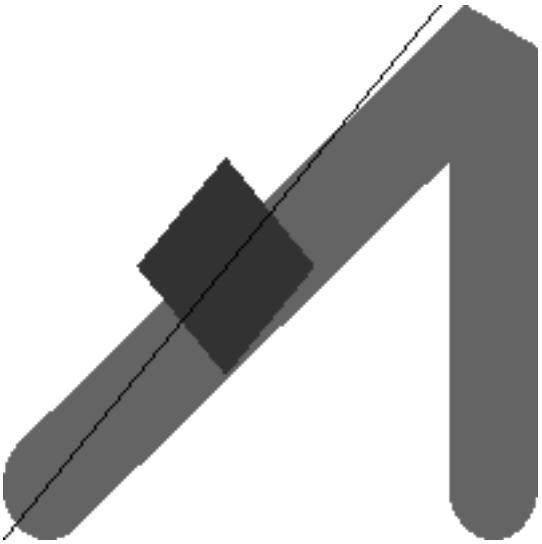
Beispele: Verwendung von Masken

```
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 otehr 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;
```

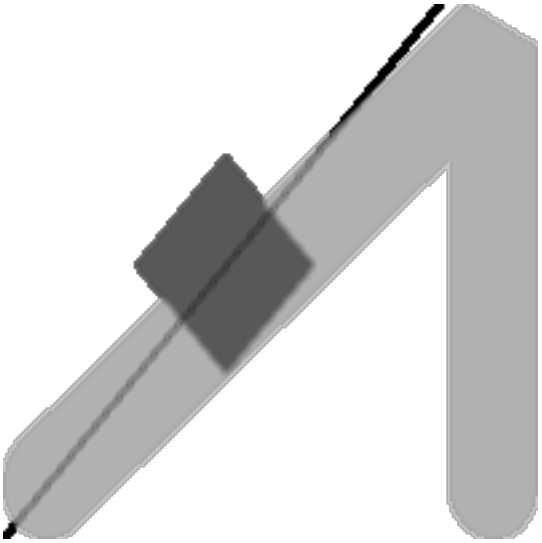


Original



Maske hat keinen Mittelwert (dasselbe wie lauter 1en in der Maskenmatrix)

Maske berücksichtigt nur die Nachbarn und schließt den Zentroid aus



Die Maske berücksichtigt nur die Nachbarn, schließt den Zentroid aus und multipliziert die anderen Pixel mit 2

Siehe auch

rastbandarg, ST_Union, ST_MapAlgebra (expression version)

10.12.6 ST_MapAlgebra (expression version)

ST_MapAlgebra (expression version) — Version mit Ausdrücken - Gibt für einen oder zwei Ausgangsraster, Bandindizes und einer oder mehreren vom Anwender vorgegebenen SQL-Ausdrücken, einen Raster mit einem Band zurück.

Synopsis

raster **ST_MapAlgebra**(raster rast, integer nband, text pixeltype, text expression, double precision nodataval=NULL);
raster **ST_MapAlgebra**(raster rast, text pixeltype, text expression, double precision nodataval=NULL);
raster **ST_MapAlgebra**(raster rast1, integer nband1, raster rast2, integer nband2, text expression, text pixeltype=NULL, text extenttype=INTERSECTION, text nodata1expr=NULL, text nodata2expr=NULL, double precision nodatanodataval=NULL);
raster **ST_MapAlgebra**(raster rast1, raster rast2, text expression, text pixeltype=NULL, text extenttype=INTERSECTION, text nodata1expr=NULL, text nodata2expr=NULL, double precision nodatanodataval=NULL);

Beschreibung

Version mit Ausdrücken - Gibt für einen oder zwei Ausgangsraster, Bandindizes und einer oder mehreren vom Anwender vorgegebenen SQL-Ausdrücken, einen Raster mit einem Band zurück.

Verfügbarkeit: 2.1.0

Beschreibung: Variante 1 und 2 (ein Raster)

Erstellt ein neues Rasterband indem eine gültige algebraische PostgreSQL Operation (`expression`) auf den Ausgangsraster (`rast`) angewendet wird. Wenn `nband` nicht angegeben ist, wird Band 1 angenommen. Der Zielraster hat die selbe Georeferenzierung, Breite und Höhe wie der ursprüngliche Raster, hat aber nur ein Band.

Wenn `pixeltype` angegeben ist, dann hat das Band des neuen Rasters diesen Pixeltyp. Wenn für den Pixeltyp NULL übergeben wird, dann hat das neue Rasterband denselben Pixeltyp wie das gegebene Band von `rast`

- Erlaubte Schlüsselwörter für `expression`
  1. `[rast]` - Zellwert der Pixel von Interesse
  2. `[rast.val]` - Zellwert der Pixel von Interesse
  3. `[rast.x]` - Rasterspalte (von 1 wegzählend) der Pixel von Interesse
  4. `[rast.y]` - Rasterszeile (von 1 wegzählend) der Pixel von Interesse

Beschreibung: Variante 3 und 4 (zwei Raster)

Erstellt einen neuen Raster mit einem Band, indem eine gültige algebraische PostgreSQL Operation auf die beiden, durch den Ausdruck `expression` bestimmten Ausgangsrasterbänder `rast1` und `rast2`) angewendet wird. Wenn `band1` oder `band2` nicht angegeben ist, wird Band 1 angenommen. Der Zielraster wird an dem Gitter des ersten Raster ausgerichtet (Größe, Versatz und Eckpunkte der Pixel). Die Ausdehnung des Zielrasters wird durch den Parameter `extenttype` bestimmt.

expression Ein algebraischer PostgreSQL Ausdruck, der zwei Raster und in PostgreSQL definierte Funktionen/Operatoren einbezieht, die den Pixelwert für sich schneidende Pixel festlegt. z.B. (([rast1] + [rast2])/2.0)::integer

pixeltype Der resultierende Pixeltyp des Zielraster muss entweder aus ST_BandPixelType sein, weggelassen oder auf NULL gesetzt werden. Wenn er nicht übergeben wird oder auf NULL gesetzt ist, wird er standardmäßig auf den Pixeltyp des ersten Raster gesetzt.

extenttype Bestimmt die Ausdehnung des resultierenden Raster

1. `INTERSECTION` - Die Ausdehnung des neuen Raster entspricht der Schnittmenge der beiden Raster. Die Standardeinstellung.
2. `UNION` - Die Ausdehnung des neuen Raster entspricht der Vereinigungsmenge der beiden Raster.
3. `FIRST` - Die Ausdehnung des neuen Raster entspricht jener des ersten Raster.
4. `SECOND` - Die Ausdehnung des neuen Raster entspricht jender des zweiten Raster.

nodata1expr Ein algebraischer Ausdruck der nur `rast2` oder eine Konstante einbezieht. Dieser bestimmt was zurückgegeben wird, wenn die Pixel von `rast1` NODATA Werte sind und die räumlich übereinstimmenden Pixel von rast2 Werte aufweisen.

nodata2expr Ein algebraischer Ausdruck der nur `rast1` oder eine Konstante einbezieht. Dieser bestimmt was zurückgegeben wird, wenn die Pixel von `rast2` NODATA Werte haben und die räumlich übereinstimmenden Pixel von rast1 Werte aufweisen.

nodatanodataval Eine numerische Konstante die ausgegeben wird, wenn die übereinstimmenden Pixel der beiden Raster "rast1" und "rast2" nur NODATA Werte enthalten.

- Zugelassene Schlüsselwörter in `expression`, `nodata1expr` und `nodata2expr`
  1. `[rast1]` - Zellwert der Pixel von Interesse von `rast1`
  2. `[rast1.val]` - Zellwert der Pixel von Interesse von `rast1`
  3. `[rast1.x]` - Rasterspalte (von 1 wegzählend) der Pixel von Interesse von `rast1`
  4. `[rast1.y]` - Rasterzeile (von 1 wegzählend) der Pixel von Interesse von `rast1`
  5. `[rast2]` - Zellwert der Pixel von Interesse von `rast2`
  6. `[rast2.val]` - Zellwert der Pixel von Interesse von `rast2`
  7. `[rast2.x]` - Rasterspalte (von 1 wegzählend) der Pixel von Interesse von `rast2`
  8. `[rast2.y]` - Rasterzeile (von 1 wegzählend) der Pixel von Interesse von `rast2`

Beispiele: Varianten 1 und 2

```
WITH foo AS (
    SELECT ST_AddBand(ST_MakeEmptyRaster(10, 10, 0, 0, 1, 1, 0, 0, 0), '32BF'::text, 1, -1) ↩
        AS rast
)
SELECT
    ST_MapAlgebra(rast, 1, NULL, 'ceil([rast]*[rast.x]/[rast.y]+[rast.val])')
FROM foo;
```

Beispiele: Varianten 3 und 4

```
WITH foo AS (
    SELECT 1 AS rid, ST_AddBand(ST_AddBand(ST_AddBand(ST_MakeEmptyRaster(2, 2, 0, 0, 1, -1, ↩
        0, 0, 0), 1, '16BUI', 1, 0), 2, '8BUI', 10, 0), 3, '32BUI'::text, 100, 0) AS rast  ↩
       UNION ALL
    SELECT 2 AS rid, ST_AddBand(ST_AddBand(ST_AddBand(ST_MakeEmptyRaster(2, 2, 0, 1, 1, -1, ↩
        0, 0, 0), 1, '16BUI', 2, 0), 2, '8BUI', 20, 0), 3, '32BUI'::text, 300, 0) AS rast
)
SELECT
    ST_MapAlgebra(
        t1.rast, 2,
        t2.rast, 1,
        '([rast2] + [rast1.val]) / 2'
    ) AS rast
FROM foo t1
CROSS JOIN foo t2
WHERE t1.rid = 1
    AND t2.rid = 2;
```

Siehe auch

rastbandarg, ST_Union, ST_MapAlgebra (callback function version)

10.12.7 ST_MapAlgebraExpr

ST_MapAlgebraExpr — Version mit 1 Rasterband: Erzeugt ein neues Rasterband, dass über eine gültige, algebraische PostgreSQL Operation für ein Rasterband mit gegebenen Pixeltyp erstellt wird. Wenn kein Band bestimmt ist, wird Band 1 angenommen.

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);

Beschreibung

Warning
ST_MapAlgebraExpr Seit der Version 2.1.0 überholt. Benutzen Sie stattdessen bitte ST_MapAlgebra (expression version) .

Erstellt ein neues Rasterband indem eine gültige algebraische PostgreSQL Operation (`expression`) auf den Ausgangsraster (`rast`) angewendet wird. Wenn kein `band` festgelegt ist, wird Band 1 angenommen. Der Zielraster hat die selbe Georeferenzierung, Breite und Höhe wie der ursprüngliche Raster, hat aber nur ein Band.

Wenn `pixeltype` angegeben ist, dann hat das Band des neuen Rasters diesen Pixeltyp. Wenn für den Pixeltyp NULL übergeben wird, dann hat das neue Rasterband denselben Pixeltyp wie das gegebene Band von `rast`

Im Ausdruck können Sie folgende Terme verwenden: `[rast]` für den Zellwert des ursprünglichen Bandes, `[rast.x]` für den von 1 wegzählenden Index der Rasterspalten, `[rast.y]` für den von 1 wegzählenden Index der Rasterzeilen.

Verfügbarkeit: 2.0.0

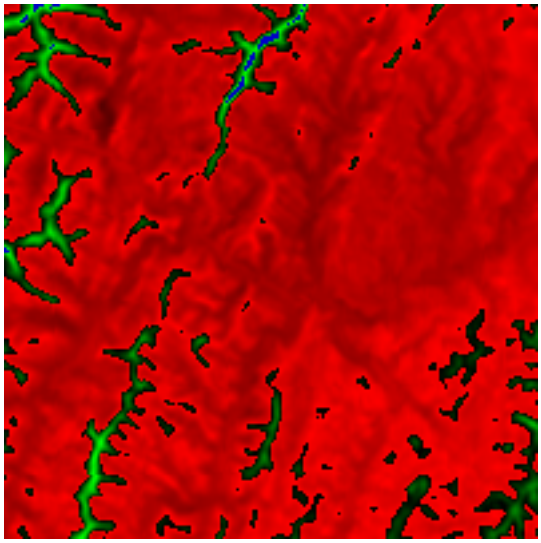
Beispiele

Erzeugt einen Einzelbandraster, der sich durch die Anwendung der Funktion "modulo 2" auf das ursprüngliche Rasterband ergibt.

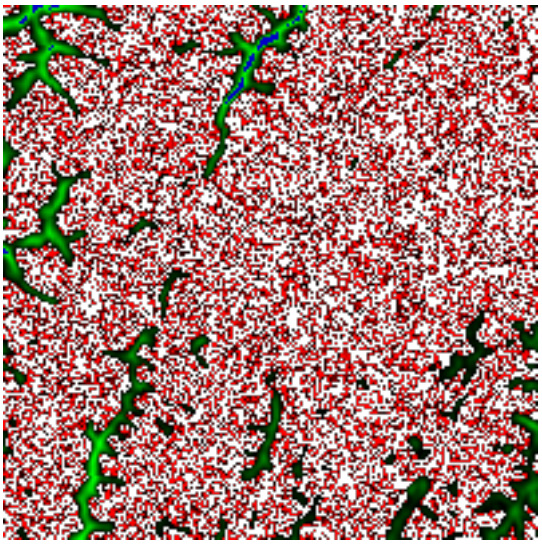
```
ALTER TABLE dummy_rast ADD COLUMN map_rast raster;
UPDATE dummy_rast SET map_rast = ST_MapAlgebraExpr(rast,NULL,'mod([rast]::numeric,2)')  ↩
    WHERE rid = 2;

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
```

Erzeugt einen neuen Einzelbandraster mit dem Pixeltyp 2BUI. Der ursprüngliche Raster wird dabei neu klassifiziert und mit einem NODATA Wert von 0 versehen.

```
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 |
     252 |      2
     253 |      3
     254 |      3

SELECT
    ST_BandPixelType(map_rast2) As b1pixtyp
FROM dummy_rast
WHERE rid = 2;

 b1pixtyp
----------
 2BUI
```



Original (Spalte "rast_view")



rast_view_ma

Erzeugt einen neuen Raster mit 3 Bändern und demselben Pixeltyp als unser ursprünglicher Raster mit 3 Bändern. Das erste

Band wird über einen Map Algebra Ausdruck geändert, die verbleibenden 2 Bänder sind unverändert.

```
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;
```

Siehe auch

ST_MapAlgebraExpr, ST_MapAlgebraFct, ST_BandPixelType, ST_GeoReference, ST_Value

10.12.8 ST_MapAlgebraExpr

ST_MapAlgebraExpr — Version mit 2 Rasterbändern: Erstellt einen neuen Einzelbandraster, indem eine gültige algebraische PostgreSQL Funktion auf die zwei Ausgangsrasterbänder und den entsprechenden Pixeltyp angewendet wird. Wenn keine Bandnummern angegeben sind, wird von jedem Raste Band 1 angenommen. Der Ergebnisraster wird nach dem Gitter des ersten Raster ausgerichtet (Skalierung, Versatz und Eckpunkte der Pixel) und hat die Ausdehnung, welche durch den Parameter "extenttype" definiert ist. Der Parameter "extenttype" kann die Werte INTERSECTION, UNION, FIRST, SECOND annehmen.

Synopsis

raster **ST_MapAlgebraExpr**(raster rast1, raster rast2, text expression, text pixeltype=same_as_rast1_band, text extenttype=INTERSEC
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);

Beschreibung

Warning
ST_MapAlgebraExpr Seit der Version 2.1.0 überholt. Benutzen Sie stattdessen bitte ST_MapAlgebra (expression version) .

Erstellt einen neuen Raster mit einem Band, indem eine gültige algebraische PostgreSQL Operation auf die beiden, durch den Ausdruck `expression` bestimmten Ausgangsrasterbänder `rast1` und `rast2`) angewendet wird. Wenn `band1` oder `band2` nicht angegeben ist, wird Band 1 angenommen. Der Zielraster wird an dem Gitter des ersten Raster ausgerichtet (Größe, Versatz und Eckpunkte der Pixel). Die Ausdehnung des Zielrasters wird durch den Parameter `extenttype` bestimmt.

expression Ein algebraischer PostgreSQL Ausdruck, der zwei Raster und in PostgreSQL definierte Funktionen/Operatoren einbezieht, die den Pixelwert für sich schneidende Pixel festlegt. z.B. (([rast1] + [rast2])/2.0)::integer

pixeltype Der resultierende Pixeltyp des Zielraster muss entweder aus ST_BandPixelType sein, weggelassen oder auf NULL gesetzt werden. Wenn er nicht übergeben wird oder auf NULL gesetzt ist, wird er standardmäßig auf den Pixeltyp des ersten Raster gesetzt.

extenttype Bestimmt die Ausdehnung des resultierenden Raster

1. `INTERSECTION` - Die Ausdehnung des neuen Raster entspricht der Schnittmenge der beiden Raster. Die Standardeinstellung.
2. `UNION` - Die Ausdehnung des neuen Raster entspricht der Vereinigungsmenge der beiden Raster.
3. `FIRST` - Die Ausdehnung des neuen Raster entspricht jener des ersten Raster.
4. `SECOND` - Die Ausdehnung des neuen Raster entspricht jender des zweiten Raster.

nodata1expr Ein algebraischer Ausdruck der nur `rast2` oder eine Konstante einbezieht. Dieser bestimmt was zurückgegeben wird, wenn die Pixel von `rast1` NODATA Werte sind und die räumlich übereinstimmenden Pixel von rast2 Werte aufweisen.

nodata2expr Ein algebraischer Ausdruck der nur `rast1` oder eine Konstante einbezieht. Dieser bestimmt was zurückgegeben wird, wenn die Pixel von `rast2` NODATA Werte haben und die räumlich übereinstimmenden Pixel von rast1 Werte aufweisen.

nodatanodataval Eine numerische Konstante die ausgegeben wird, wenn die übereinstimmenden Pixel der beiden Raster "rast1" und "rast2" nur NODATA Werte enthalten.

Wenn `pixeltype` angegeben ist, dann hat das Band des neuen Rasters diesen Pixeltyp. Wenn für den Pixeltyp NULL übergeben wird oder der Pixeltyp nicht festgelegt ist, dann hat das neue Rasterband denselben Pixeltyp wie das angegebene Band von `rast1`

Sie können den Ausdruck `[rast1.val] [rast2.val]` verwenden, um auf die Pixelwerte der ursprünglichen Rasterbänder zu verweisen, und `[rast1.x]`, `[rast1.y]` etc. um auf die Positionen der Pixel über die Rasterspalten / Rasterzeilen zu verweisen.

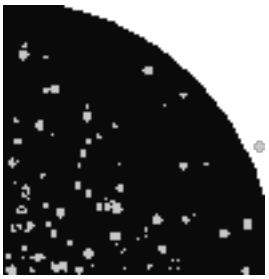
Verfügbarkeit: 2.0.0

Beispiel: Verschneidung und Union von 2 Bändern

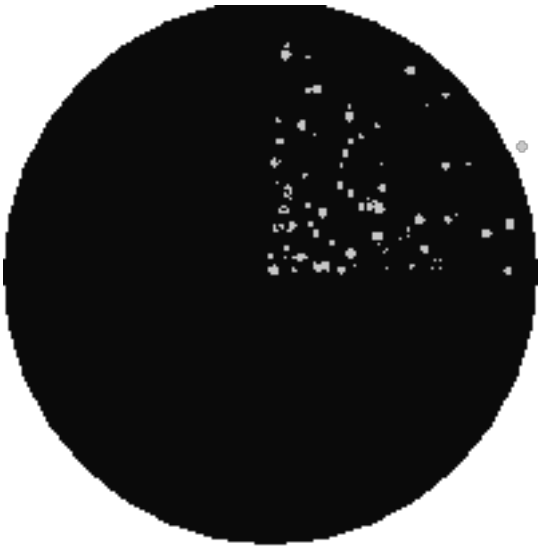
Erzeugt einen Einzelbandraster, der sich durch die Anwendung der Funktion "modulo 2" auf das ursprüngliche Rasterband ergibt.

```
--Create a cool set of rasters --
DROP TABLE IF EXISTS fun_shapes;
CREATE TABLE fun_shapes(rid serial PRIMARY KEY, fun_name text, rast raster);

-- Insert some cool shapes around Boston in Massachusetts state plane meters --
INSERT INTO fun_shapes(fun_name, rast)
VALUES ('ref', ST_AsRaster(ST_MakeEnvelope(235229, 899970, 237229, 901930,26986),200,200,'8 ↩
    BUI',0,0));

INSERT INTO fun_shapes(fun_name,rast)
WITH ref(rast) AS (SELECT rast FROM fun_shapes WHERE fun_name = 'ref' )
SELECT 'area' AS fun_name, ST_AsRaster(ST_Buffer(ST_SetSRID(ST_Point(236229, 900930),26986) ↩
    , 1000),
            ref.rast,'8BUI', 10, 0) As rast
FROM ref
UNION ALL
SELECT 'rand bubbles',
            ST_AsRaster(
            (SELECT ST_Collect(geom)
    FROM (SELECT ST_Buffer(ST_SetSRID(ST_Point(236229 + i*random()*100, 900930 + j*random() ↩
        *100),26986), random()*20) As geom
            FROM generate_series(1,10) As i, generate_series(1,10) As j
            ) As foo ), ref.rast,'8BUI', 200, 0)
FROM ref;

--map them -
SELECT  ST_MapAlgebraExpr(
        area.rast, bub.rast, '[rast2.val]', '8BUI', 'INTERSECTION', '[rast2.val]', '[rast1. ↩
            val]') As interrast,
```

```
        ST_MapAlgebraExpr(
            area.rast, bub.rast, '[rast2.val]', '8BUI', 'UNION', '[rast2.val]', '[rast1.val ↩
                ]') As unionrast
FROM
  (SELECT rast FROM fun_shapes WHERE
 fun_name = 'area') As area
CROSS JOIN  (SELECT rast
FROM fun_shapes WHERE
 fun_name = 'rand bubbles') As bub
```



Map Algebra Verschneidung

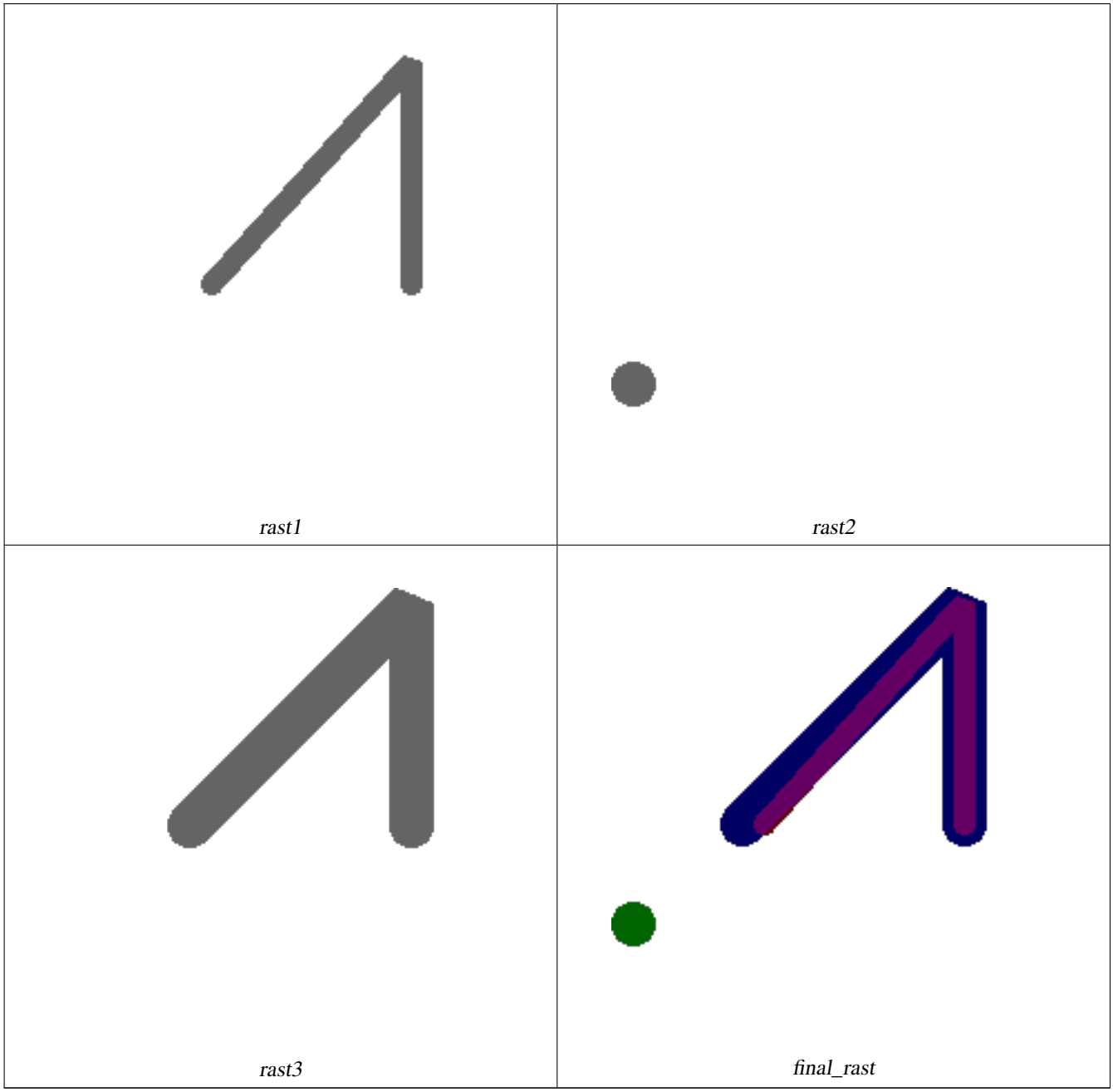


Map Algebra Vereinigung

Beispiel: Überlagerung von Rastern als Einzelbänder am Canvas

```
-- 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;
```



rast1 | *rast2*

rast3 | *final_rast*

Beispiel: Ein Orthophoto mit einer 2 Meter breiten Begrenzung der ausgewählten Grundstücke überlagern

```
-- Create new 3 band raster composed of first 2 clipped bands, and overlay of 3rd band with ↩
     our geometry
-- This query took 3.6 seconds on PostGIS windows 64-bit install
WITH pr AS
```

```
-- Note the order of operation: we clip all the rasters to dimensions of our region
(SELECT ST_Clip(rast,ST_Expand(geom,50) ) As rast, g.geom
    FROM aerials.o_2_boston AS r INNER JOIN
-- union our parcels of interest so they form a single geometry we can later intersect with
        (SELECT ST_Union(ST_Transform(geom,26986)) AS geom
          FROM landparcels WHERE pid IN('0303890000', '0303900000')) As g
        ON ST_Intersects(rast::geometry, ST_Expand(g.geom,50))
),
-- we then union the raster shards together
-- ST_Union on raster is kinda of slow but much faster the smaller you can get the rasters
-- therefore we want to clip first and then union
prunion AS
(SELECT ST_AddBand(NULL, ARRAY[ST_Union(rast,1),ST_Union(rast,2),ST_Union(rast,3)] ) As  ↩
    clipped,geom
FROM pr
GROUP BY geom)
-- return our final raster which is the unioned shard with
-- with the overlay of our parcel boundaries
-- add first 2 bands, then mapalgebra of 3rd band + geometry
SELECT ST_AddBand(ST_Band(clipped,ARRAY[1,2])
    , ST_MapAlgebraExpr(ST_Band(clipped,3), ST_AsRaster(ST_Buffer(ST_Boundary(geom),2), ↩
        clipped, '8BUI',250),
     '[rast2.val]', '8BUI', 'FIRST', '[rast2.val]', '[rast1.val]') ) As rast
FROM prunion;
```

Die blauen Linien sind die Ränder der ausgewählten Grundstücke.

Siehe auch

ST_MapAlgebraExpr, ST_AddBand, ST_AsPNG, ST_AsRaster, ST_MapAlgebraFct, ST_BandPixelType, ST_GeoReference, ST_Value, ST_Union, ST_Union

10.12.9 ST_MapAlgebraFct

ST_MapAlgebraFct — Version mit 1 Rasterband: Erzeugt ein neues Rasterband, dass über eine gültige PostgreSQL Funktion für ein gegebenes Rasterband und Pixeltyp erstellt wird. Wenn kein Band bestimmt ist, wird Band 1 angenommen.

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);

Beschreibung

Warning
ST_MapAlgebraFct Seit der Version 2.1.0 überholt. Benutzen Sie stattdessen bitte ST_MapAlgebra (callback function version)

Erstellt einen neuen Raster mit einem Band, indem eine gültige PostgreSQL Funktion durch `tworastuserfunc` spezifiziert und auf den gegebenen Raster (`rast`) angewendet wird. Wenn kein `Band` angegeben ist, wird Band 1 angenommen. Der Zielraster hat die selbe Georeferenzierung, Breite und Höhe wie der ursprüngliche Raster, hat aber nur ein Band.

Wenn `pixeltype` angegeben ist, dann hat das Band des neuen Rasters diesen Pixeltyp. Wenn für den Pixeltyp NULL übergeben wird, dann hat das neue Rasterband denselben Pixeltyp wie das gegebene Band von `rast`

The `onerasteruserfunc` parameter must be the name and signature of a SQL or PL/pgSQL function, cast to a regprocedure. A very simple and quite useless PL/pgSQL function example is:

```
CREATE OR REPLACE FUNCTION simple_function(pixel FLOAT, pos INTEGER[], VARIADIC args TEXT ↩
    [])
    RETURNS FLOAT
    AS $$ BEGIN
        RETURN 0.0;
    END; $$
    LANGUAGE 'plpgsql' IMMUTABLE;
```

The `userfunction` may accept two or three arguments: a float value, an optional integer array, and a variadic text array. The first argument is the value of an individual raster cell (regardless of the raster datatype). The second argument is the position of the current processing cell in the form '{x,y}'. The third argument indicates that all remaining parameters to ST_MapAlgebraFct shall be passed through to the `userfunction`.

Passing a regprodedure argument to a SQL function requires the full function signature to be passed, then cast to a regprocedure type. To pass the above example PL/pgSQL function as an argument, the SQL for the argument is:

```
'simple_function(float,integer[],text[])'::regprocedure
```

Note that the argument contains the name of the function, the types of the function arguments, quotes around the name and argument types, and a cast to a regprocedure.

Der dritte Übergabewert an die Funktion `userfunction` ist ein Feld vom Datentyp variadic text. Alle an einen Funktionsaufruf von ST_MapAlgebraFct angehängten Parameter werden an die spezifizierte `userfunction` durchgereicht und sind in dem Übergabewert `args` enthalten.

Note
Für nähere Information zu dem Schlüsselwort VARIADIC, sehen Sie bitte die PostgreSQL Dokumentation und den Abschnitt "SQL Functions with Variable Numbers of Arguments" unter Query Language (SQL) Functions.

Note
Der Übergabewert text[] an die `userfunction` ist auch dann erforderlich, wenn Sie sonst keine Argumente für die Auswertung an an "userfunction" übergeben.

Verfügbarkeit: 2.0.0

Beispiele

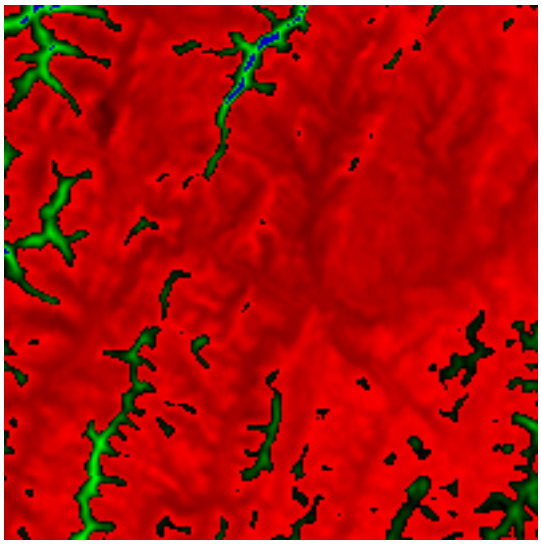
Erzeugt einen Einzelbandraster, der sich durch die Anwendung der Funktion "modulo 2" auf das ursprüngliche Rasterband ergibt.

```
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
```

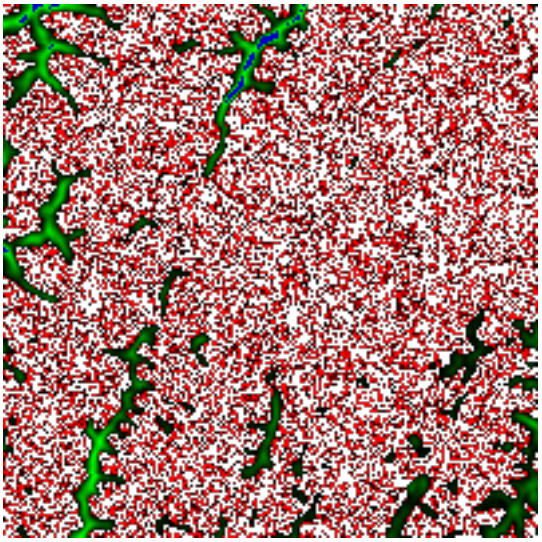
Erzeugt einen neuen Raster mit einem Band und mit dem Pixeltyp 2BUI. Der ursprüngliche Raster wird dabei neu klassifiziert und der NODATA Wert wird an die "user function" (0) übergeben.

```
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
```



Original (Spalte "rast_view")



rast_view_ma

Erzeugt einen neuen Raster mit 3 Bändern und demselben Pixeltyp als unser ursprünglicher Raster mit 3 Bändern. Das erste

Band wird über einen Map Algebra Ausdruck geändert, die verbleibenden 2 Bänder sind unverändert.

```
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;
```

Siehe auch

ST_MapAlgebraExpr, ST_BandPixelType, ST_GeoReference, ST_SetValue

10.12.10 ST_MapAlgebraFct

ST_MapAlgebraFct — Version mit 2 Rasterbändern: Erstellt einen neuen Einzelbandraster, indem eine gültige PostgreSQL Funktion auf die 2 gegebenen Rasterbänder und den entsprechenden Pixeltyp angewendet wird. Wenn kein Band bestimmt ist, wird Band 1 angenommen. Wenn der "Extent"-Typ nicht angegeben ist, wird standardmäßig INTERSECTION angenommen.

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);

Beschreibung

Warning
ST_MapAlgebraFct Seit der Version 2.1.0 überholt. Benutzen Sie stattdessen bitte ST_MapAlgebra (callback function version)

Erstellt einen neuen Raster mit einem Band, indem eine gültige PostgreSQL Funktion (`tworastuserfunc`) auf die Ausgangsraster (`rast1`, `rast2`) angewendet wird. Wenn `band1` oder `band2` nicht angegeben ist, wird Band 1 angenommen. Der Zielraster hat die selbe Georeferenzierung, Breite und Höhe wie der ursprüngliche Raster, hat aber nur ein Band.

Wenn `pixeltype` angegeben ist, dann hat das Band des neuen Rasters diesen Pixeltyp. Wenn für den Pixeltyp NULL oder kein Wert übergeben wird, dann hat das neue Rasterband denselben Pixeltyp wie das angegebene Band von `rast1`

The `tworastuserfunc` parameter must be the name and signature of an SQL or PL/pgSQL function, cast to a regprocedure. An example PL/pgSQL function example is:

```
CREATE OR REPLACE FUNCTION simple_function_for_two_rasters(pixel1 FLOAT, pixel2 FLOAT, pos ↩
    INTEGER[], VARIADIC args TEXT[])
    RETURNS FLOAT
    AS $$ BEGIN
        RETURN 0.0;
    END; $$
    LANGUAGE 'plpgsql' IMMUTABLE;
```

The `tworastuserfunc` may accept three or four arguments: a double precision value, a double precision value, an optional integer array, and a variadic text array. The first argument is the value of an individual raster cell in `rast1` (regardless of the raster datatype). The second argument is an individual raster cell value in `rast2`. The third argument is the position of the current processing cell in the form '{x,y}'. The fourth argument indicates that all remaining parameters to ST_MapAlgebraFct shall be passed through to the `tworastuserfunc`.

Passing a regprodedure argument to a SQL function requires the full function signature to be passed, then cast to a regprocedure type. To pass the above example PL/pgSQL function as an argument, the SQL for the argument is:

```
'simple_function(double precision, double precision, integer[], text[])'::regprocedure
```

Note that the argument contains the name of the function, the types of the function arguments, quotes around the name and argument types, and a cast to a regprocedure.

Der vierte Übergabewert an die Funktion `tworastuserfunc` ist ein Feld mit dem Datentyp variadic text. Alle an eine Funktion ST_MapAlgebraFct angehängten Parameter werden an die `tworastuserfunc` durchgereicht und sind in dem Übergabewert `userargs` enthalten.

Note
Für nähere Information zu dem Schlüsselwort VARIADIC, sehen Sie bitte die PostgreSQL Dokumentation und den Abschnitt "SQL Functions with Variable Numbers of Arguments" unter Query Language (SQL) Functions.

Note
Der Übergabewert text[] an die `tworastuserfunc` ist auch dann erforderlich, wenn Sie sonst keine Argumente für die Auswertung an an "userfunction" übergeben.

Verfügbarkeit: 2.0.0

Beispiel: Überlagerung von Rastern als Einzelbänder am Canvas

```
-- 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;
```

map bands overlay (Canvas) (R: schmale Straße, G: Kreis, B: breite Straße)

Eine benutzerdefinierte Funktion, die zusätzliche Übergabewerte entgegennimmt

```
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)';
```

Eine benutzerdefinierte Funktion mit zusätzlichen Übergabewerten und unterschiedlichen Bändern des gleichen Raster

Siehe auch

ST_MapAlgebraExpr, ST_BandPixelType, ST_GeoReference, ST_SetValue

10.12.11 ST_MapAlgebraFctNgb

ST_MapAlgebraFctNgb — Version mit 1em Band: Map Algebra Nearest Neighbor mit einer benutzerdefinierten PostgreSQL Funktion. Gibt einen Raster zurück, dessen Werte sich aus einer benutzerdefinierte PL/pgsql Funktion ergeben, welche die Nachbarschaftswerte des Ausgangsrasterbandes einbezieht.

Synopsis

raster **ST_MapAlgebraFctNgb**(raster rast, integer band, text pixeltype, integer ngbwidth, integer ngbheight, regprocedure onerastngbuserfunc, text nodatamode, text[] VARIADIC args);

Beschreibung

Warning
ST_MapAlgebraFctNgb Seit der Version 2.1.0 überholt. Benutzen Sie stattdessen bitte ST_MapAlgebra (callback function version)

(Version mit einem Raster) Gibt einen Raster zurück, dessen Werte sich aus einer benutzerdefinierte PL/pgSQL Funktion errechnen, welche die Nachbarschaftswerte des Ausgangsrasterbandes mit einbezieht. Die benutzerdefinierte Funktion nimmt die Werte der Nachbarschaftspixel als Zahlenfeld entgegen und gibt für jedes Pixel das Ergebnis der Funktion zurück, indem die aktuellen Werte der betrachteten Pixel mit dem Ergebnis der benutzerdefinierten Funktion ersetzt werden.

rast Raster der durch die benutzerdefinierte Funktion ausgewertet werden soll.

band Die Bandnummer des Raster, die ausgewertet werden soll. Die Standardeinstellung ist 1.

pixeltype Der resultierende Pixeltyp des Zielraster muss entweder aus ST_BandPixelType sein, weggelassen oder auf NULL gesetzt werden. Wenn er nicht übergeben wird oder auf NULL gesetzt ist, wird er standardmäßig auf den Pixeltyp von `rast` gesetzt. Die Ergebnisse werden abgeschnitten, wenn sie größer als für den Pixeltyp erlaubt sind.

ngbwidth Die Breite der Nachbarschaft in Zellen.

ngbheight Die Höhe der Nachbarschaft in Zellen.

onerastngbuserfunc Eine benutzerdefinierte Funktion in PLPGSQL/psql, die auf die Nachbarschaftspixel in einem einzelnen Rasterband angewandt wird. Das erste Element ist ein 2-dimensionales Feld mit Zahlen, welche das Rechteck mit den Nachbarschaftspixel beschreiben

nodatamode Bestimmt welcher Wert an die Funktion übergeben werden soll, wenn ein Nachbarschaftspixel NODATA oder NULL ist

'ignore': NODATA Werte in der Nachbarschaft werden bei der Berechnung ignoriert -- diese Flag muss an die Rückruffunktion gesendet werden und die benutzerdefinierte Funktion entscheidet dann, wie diese Werte ignoriert werden sollen.

'NULL': NODATA Werte in der Nachbarschaft bedingen, dass die resultierenden Pixel ebenfalls NULL annehmen - die benutzerdefinierte Rückruffunktion wird bei diesem Fall übersprungen.

'value': NODATA Werte in der Nachbarschaft werden durch den Wert des Referenzpixel (das Pixel im Zentrum der Nachbarschaft) ersetzt. Anmerkung: Wenn dieser Wert NODATA ist, dann ist die Verhaltensweise gleich wie bei 'NULL' (für die betroffene Nachbarschaft)

args Übergabewerte für die benutzerdefinierte Funktion.

Verfügbarkeit: 2.0.0

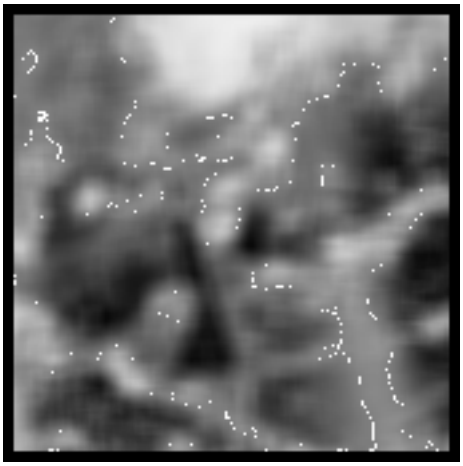
Beispiele

Die Beispiele nutzen den Raster "Katrina", der als einzelne Kachel geladen wird, wie unter http://trac.osgeo.org/gdal/wiki/frmts_wtkraster.html beschrieben; und in den Beispielen von ST_Rescale aufbereitet wurde.

```
--
-- 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;
```

Das erste Band Unseres Rasters



neuer Raster mit dem Durchschnittswert der Pixel innerhalb von 4x4 Pixel

Siehe auch

ST_MapAlgebraFct, ST_MapAlgebraExpr, ST_Rescale

10.12.12 ST_Reclass

ST_Reclass — Erstellt einen neuen Raster, der aus neu klassifizierten Bändern des Originalraster besteht. Das Band "nband" ist jenes das verändert werden soll. Wenn "nband" nicht angegeben ist, wird "Band 1" angenommen. Alle anderen Bänder bleiben unverändert. Anwendungsfall: zwecks einfacherer Visualisierung ein 16BUI-Band in ein 8BUI-Band konvertieren und so weiter.

Synopsis

raster **ST_Reclass**(raster rast, integer nband, text reclassexpr, text pixeltype, double precision nodataval=NULL);
raster **ST_Reclass**(raster rast, reclassarg[] VARIADIC reclassargset);
raster **ST_Reclass**(raster rast, text reclassexpr, text pixeltype);

Beschreibung

Erstellt einen neuen Raster, indem eine gültige algebraische PostgreSQL Operation die durch den Ausdruck `reclassexpr` festgelegt ird auf den Ausgangsraster (`rast`) angewendet wird. Wenn `nband` nicht angegeben ist, wird Band 1 angenommen. Der Zielraster hat die selbe Georeferenzierung, Breite und Höhe wie der ursprüngliche Raster. Nicht ausgewiesene Bänder werden unverändert zurückgegeben. Siehe reclassarg für eine Beschreibung der gültigen Ausdrücke zur Neuklassifizierung.

Die Bänder des Zielraster haben den Pixeltyp `pixeltype`. Wenn `reclassargset` übergeben wird, dann bestimmt ein "regclassarg" wie das jeweilige Band erstellt werden soll.

Verfügbarkeit: 2.0.0

Grundlegende Beispiele

Erzeugt einen neuen Raster aus dem Original, indem Band 2 von 8BUI auf 4BUI und alle Werte von 101-254 auf NODATA gesetzt werden.

```
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
```

Beispiel: Erweiterte Verwendung mit mehreren "reclassargs"

Erstellt einen neuen Raster aus dem Ursprungsraster, wobei die Bänder 1, 2 und 3 in 1BB, 4BUI und 4BUI konvertiert und neu klassifiziert werden. Verwendet das variadische Argument `reclassarg`, welches eine unbegrenzte Anzahl von "reclassargs" entgegennehmen kann (theoretisch so viele wie Bänder vorhanden sind)

```
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
```

Beispiel: Abbildung eines Einzelbandrasters (32BF) auf mehrere darstellbare Bänder

Erstellung eines neuen Raster mit 3 Bändern (8BUI,8BUI,8BUI darstellbarer Raster) aus einem Raster mit nur einem 32bf-Band

```
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)
    ]
    );
```

Siehe auch

ST_AddBand, ST_Band, ST_BandPixelType, ST_MakeEmptyRaster, reclassarg, ST_Value

10.12.13 ST_Union

ST_Union — Gibt die Vereinigung mehrerer Rasterkacheln in einem einzelnen Raster mit mehreren Bändern zurück.

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);

Beschreibung

Gibt die Vereinigung von Rasterkacheln in einem einzelnen Raster zurück, der mindestens aus einem Band besteht. Die räumliche Ausdehnung des Zielraster entspricht der Gesamtausdehnung. Im Fall von Überschneidungen wird der resultierende Wert durch `uniontype` festgelegt, welcher folgende Werte annehmen kann: LAST (Standardwert), FIRST, MIN, MAX, COUNT, SUM, MEAN, RANGE.

Note
Damit Raster vereinigt werden können, müssen sie alle gleich ausgerichtet sein. Siehe ST_SameAlignment und ST_NotSameAlignmentReason für weitere Details und Hilfe. Eine Möglichkeit, um Probleme mit der Ausrichtung zu beheben ist ST_Resample und die Verwendung eines gemeinsamen Referenzraster zur Anpassung.

Verfügbarkeit: 2.0.0

Erweiterung: 2.1.0 Geschwindigkeit verbessert (zur Gänze C-basiert)

Verfügbarkeit: 2.1.0 ST_Union(rast, unionarg) Variante wurde eingeführt.

Erweiterung: 2.1.0 ST_Union(rast) (Variante 1) vereinigt alle Bänder aller Ausgangsraster. Vorherige Versionen von PostGIS setzten das erste Band voraus.

Erweiterung: 2.1.0 ST_Union(rast, uniontype) (Variante 4) vereinigt alle Bänder aller Ausgangsraster.

Beispiele: Wiederherstellung eines einzelnen Bandes einer Rasterkachel

```
-- 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;
```

Beispiele: Vereinigt Kacheln, die eine Geometrie schneiden, zu einem Raster mit mehreren Bändern

```
-- this creates a multi band raster collecting all the tiles that intersect a line
-- Note: In 2.0, this would have just returned a single band raster
-- , new union works on all bands by default
-- this is equivalent to unionarg: ARRAY[ROW(1, 'LAST'), ROW(2, 'LAST'), ROW(3, 'LAST')]:: ↩
    unionarg[]
SELECT ST_Union(rast)
FROM aerials.boston
WHERE ST_Intersects(rast,  ST_GeomFromText('LINESTRING(230486 887771, 230500 88772)',26986) ↩
    );
```

Beispiele: Vereinigt Kacheln, die eine Geometrie schneiden, zu einem Raster mit mehreren Bändern

Um nur eine Teilmenge der Bänder zu erhalten, oder die Reihenfolge der Bänder zu ändern, können wir die längere Syntax verwenden

```
-- this creates a multi band raster collecting all the tiles that intersect a line
SELECT ST_Union(rast,ARRAY[ROW(2, 'LAST'), ROW(1, 'LAST'), ROW(3, 'LAST')]::unionarg[])
FROM aerials.boston
WHERE ST_Intersects(rast,  ST_GeomFromText('LINESTRING(230486 887771, 230500 88772)',26986) ↩
    );
```

Siehe auch

unionarg, ST_Envelope, ST_ConvexHull, ST_Clip, ST_Union

10.13 Integrierte Map Algebra Callback Funktionen

10.13.1 ST_Distinct4ma

ST_Distinct4ma — Funktion zur Rasterdatenverarbeitung, welche die Anzahl der einzelnen Pixelwerte in der Nachbarschaft errechnet.

Synopsis

float8 **ST_Distinct4ma**(float8[][] matrix, text nodatamode, text[] VARIADIC args);
double precision **ST_Distinct4ma**(double precision[][][] value, integer[][] pos, text[] VARIADIC userargs);

Beschreibung

Berechnet die Anzahl der einzelnen Pixelwerte in der Nachbarschaft von Pixel.

Note
Variante 1 ist eine spezialisierte Rückruffunktion, die als Rückrufparameter für ST_MapAlgebraFctNgb verwendet werden kann.

Note
Variante 2 ist eine spezialisierte Rückruffunktion, die als Rückrufparameter für ST_MapAlgebra (callback function version) verwendet werden kann.

Warning
Von der Verwendung der Variante 1 wird abgeraten, da ST_MapAlgebraFctNgb seit 2.1.0 überholt ist.

Verfügbarkeit: 2.0.0

Erweiterung: 2.1.0 Variante 2 wurde hinzugefügt

Beispiele

```
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)
```

Siehe auch

ST_MapAlgebraFctNgb, ST_MapAlgebra (callback function version), ST_Min4ma, ST_Max4ma, ST_Sum4ma, ST_Mean4ma, ST_Distinct4ma, ST_StdDev4ma

10.13.2 ST_InvDistWeight4ma

ST_InvDistWeight4ma — Funktion zur Rasterdatenverarbeitung, die den Wert eines Pixel aus den Pixel der Nachbarschaft interpoliert.

Synopsis

double precision **ST_InvDistWeight4ma**(double precision[][][] value, integer[][] pos, text[] VARIADIC userargs);

Beschreibung

Interpoliert den Wert eines Pixel über die Methode der inversen Distanzwichtung.

Es gibt zwei optionale Parameter, die über `userargs` übergeben werden können. Der erste Parameter ist der Exponent (die Variable "k" in unterer Gleichung); dieser liegt zwischen 0 und 1 und wird in der Gleichung für die Inverse Distanzwichtung verwendet. Wenn er nicht angegeben ist, wird standardmäßig 1 angenommen. Der zweite Parameter entspricht der Gewichtung in Prozent und wird nur verwendet, wenn der Wert der betrachteten Pixel einem aus der Nachbarschaft interpolierten Wert entspricht. Wenn er nicht angegeben ist und das betrachtete Pixel einen Wert hat, so wird dieser Wert zurückgegeben.

Die grundlegende Gleichung der inversen Distanzwichtung ist:

$$\hat{z}(x_o) = \frac{\sum_{j=1}^{m} z(x_j) d_{ij}^{-k}}{\sum_{j=1}^{m} d_{ij}^{-k}}$$

k = Exponent, eine reelle Zahl zwischen 0 und 1

Note
Diese Funktion ist eine spezialisierte Rückruffunktion, die als Rückrufparameter für ST_MapAlgebra (callback function version) verwendet werden kann.

Verfügbarkeit: 2.1.0

Beispiele

```
-- NEEDS EXAMPLE
```

Siehe auch

ST_MapAlgebra (callback function version), ST_MinDist4ma

10.13.3 ST_Max4ma

ST_Max4ma — Funktion zur Rasterdatenverarbeitung, die den maximalen Zellwert in der Nachbarschaft eines Pixel errechnet.

Synopsis

float8 **ST_Max4ma**(float8[][] matrix, text nodatamode, text[] VARIADIC args);
double precision **ST_Max4ma**(double precision[][][] value, integer[][] pos, text[] VARIADIC userargs);

Beschreibung

Berechnet den maximalen Zellwert in der Nachbarschaft von Pixel.

Bei Variante 2 kann ein Substitutionswert für NODATA an "userargs" übergeben werden.

Note
Variante 1 ist eine spezialisierte Rückruffunktion, die als Rückrufparameter für ST_MapAlgebraFctNgb verwendet werden kann.

Note
Variante 2 ist eine spezialisierte Rückruffunktion, die als Rückrufparameter für ST_MapAlgebra (callback function version) verwendet werden kann.

Warning
Von der Verwendung der Variante 1 wird abgeraten, da ST_MapAlgebraFctNgb seit 2.1.0 überholt ist.

Verfügbarkeit: 2.0.0

Erweiterung: 2.1.0 Variante 2 wurde hinzugefügt

Beispiele

```
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)
```

Siehe auch

ST_MapAlgebraFctNgb, ST_MapAlgebra (callback function version), ST_Min4ma, ST_Sum4ma, ST_Mean4ma, ST_Range4ma, ST_Distinct4ma, ST_StdDev4ma

10.13.4 ST_Mean4ma

ST_Mean4ma — Funktion zur Rasterdatenverarbeitung, die den mittleren Zellwert in der Nachbarschaft von Pixel errechnet.

Synopsis

float8 **ST_Mean4ma**(float8[][] matrix, text nodatamode, text[] VARIADIC args);
double precision **ST_Mean4ma**(double precision[][][] value, integer[][] pos, text[] VARIADIC userargs);

Beschreibung

Berechnet den mittleren Zellwert in der Nachbarschaft von Pixel.

Bei Variante 2 kann ein Substitutionswert für NODATA an "userargs" übergeben werden.

Note
Variante 1 ist eine spezialisierte Rückruffunktion, die als Rückrufparameter für ST_MapAlgebraFctNgb verwendet werden kann.

Note
Variante 2 ist eine spezialisierte Rückruffunktion, die als Rückrufparameter für ST_MapAlgebra (callback function version) verwendet werden kann.

Warning
Von der Verwendung der Variante 1 wird abgeraten, da ST_MapAlgebraFctNgb seit 2.1.0 überholt ist.

Verfügbarkeit: 2.0.0

Erweiterung: 2.1.0 Variante 2 wurde hinzugefügt

Beispiele: Variante 1

```
SELECT
    rid,
    st_value(
        st_mapalgebrafctngb(rast, 1, '32BF', 1, 1, 'st_mean4ma(float[][],text,text[])':: ↩
            regprocedure, 'ignore', NULL), 2, 2
    )
FROM dummy_rast
WHERE rid = 2;
 rid |     st_value
-----+------------------
   2 | 253.222229003906
(1 row)
```

Beispiele: Variante 2

```
SELECT
    rid,
    st_value(
               ST_MapAlgebra(rast, 1, 'st_mean4ma(double precision[][][], integer[][], text ↩
                   [])'::regprocedure,'32BF', 'FIRST', NULL, 1, 1)
       ,  2, 2)
  FROM dummy_rast
   WHERE rid = 2;
 rid |     st_value
-----+------------------
   2 | 253.222229003906
(1 row)
```

Siehe auch

ST_MapAlgebraFctNgb, ST_MapAlgebra (callback function version), ST_Min4ma, ST_Max4ma, ST_Sum4ma, ST_Range4ma, ST_StdDev4ma

10.13.5 ST_Min4ma

ST_Min4ma — Funktion zur Rasterdatenverarbeitung, die den minimalen Zellwert in der Nachbarschaft von Pixel errechnet.

Synopsis

float8 **ST_Min4ma**(float8[][] matrix, text nodatamode, text[] VARIADIC args);
double precision **ST_Min4ma**(double precision[][][] value, integer[][] pos, text[] VARIADIC userargs);

Beschreibung

Berechnet den minimalen Zellwert in der Nachbarschaft von Pixel.

Bei Variante 2 kann ein Substitutionswert für NODATA an "userargs" übergeben werden.

Note
Variante 1 ist eine spezialisierte Rückruffunktion, die als Rückrufparameter für ST_MapAlgebraFctNgb verwendet werden kann.

Note
Variante 2 ist eine spezialisierte Rückruffunktion, die als Rückrufparameter für ST_MapAlgebra (callback function version) verwendet werden kann.

Warning
Von der Verwendung der Variante 1 wird abgeraten, da ST_MapAlgebraFctNgb seit 2.1.0 überholt ist.

Verfügbarkeit: 2.0.0

Erweiterung: 2.1.0 Variante 2 wurde hinzugefügt

Beispiele

```
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)
```

Siehe auch

ST_MapAlgebraFctNgb, ST_MapAlgebra (callback function version), ST_Max4ma, ST_Sum4ma, ST_Mean4ma, ST_Range4ma, ST_Distinct4ma, ST_StdDev4ma

10.13.6 ST_MinDist4ma

ST_MinDist4ma — Funktion zur Rasterdatenverarbeitung, welche die kürzeste Entfernung (in Pixel) zwischen dem Pixel von Interesse und einem benachbarten Pixel mit Zellwert zurückgibt.

Synopsis

double precision **ST_MinDist4ma**(double precision[][][] value, integer[][] pos, text[] VARIADIC userargs);

Beschreibung

Gibt die kürzeste Entfernung (Pixelanzahl) zwischen dem Pixel von Interesse und dem nächstgelegenen Pixel mit Zellwert in der Nachbarschaft zurück.

Note
Diese Funktion soll einen Anhaltspunkt liefern, um die Sinnhaftigkeit des mittels ST_InvDistWeight4ma interpolierten Wertes für das betrachtete Pixel abzuleiten. Diese Funktion ist besonders nützlich wenn die Nachbarschaft des Pixel nur spärlich besetzt ist.

Note
Diese Funktion ist eine spezialisierte Rückruffunktion, die als Rückrufparameter für ST_MapAlgebra (callback function version) verwendet werden kann.

Verfügbarkeit: 2.1.0

Beispiele

```
-- NEEDS EXAMPLE
```

Siehe auch

ST_MapAlgebra (callback function version), ST_InvDistWeight4ma

10.13.7 ST_Range4ma

ST_Range4ma — Funktion zur Rasterdatenverarbeitung, die den Wertebereich der Pixel in einer Nachbarschaft errechnet.

Synopsis

float8 **ST_Range4ma**(float8[][] matrix, text nodatamode, text[] VARIADIC args);
double precision **ST_Range4ma**(double precision[][][] value, integer[][] pos, text[] VARIADIC userargs);

Beschreibung

Berechnet den Wertebereich der Pixel in einer Nachbarschaft von Pixel.

Bei Variante 2 kann ein Substitutionswert für NODATA an "userargs" übergeben werden.

Note
Variante 1 ist eine spezialisierte Rückruffunktion, die als Rückrufparameter für ST_MapAlgebraFctNgb verwendet werden kann.

Note
Variante 2 ist eine spezialisierte Rückruffunktion, die als Rückrufparameter für ST_MapAlgebra (callback function version) verwendet werden kann.

Warning
Von der Verwendung der Variante 1 wird abgeraten, da ST_MapAlgebraFctNgb seit 2.1.0 überholt ist.

Verfügbarkeit: 2.0.0

Erweiterung: 2.1.0 Variante 2 wurde hinzugefügt

Beispiele

```
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)
```

Siehe auch

ST_MapAlgebraFctNgb, ST_MapAlgebra (callback function version), ST_Min4ma, ST_Max4ma, ST_Sum4ma, ST_Mean4ma, ST_Distinct4ma, ST_StdDev4ma

10.13.8 ST_StdDev4ma

ST_StdDev4ma — Funktion zur Rasterdatenverarbeitung, welche die Standardabweichung der Zellwerte in der Nachbarschaft von Pixel errechnet.

Synopsis

float8 **ST_StdDev4ma**(float8[][] matrix, text nodatamode, text[] VARIADIC args);
double precision **ST_StdDev4ma**(double precision[][][] value, integer[][] pos, text[] VARIADIC userargs);

Beschreibung

Berechnet die Standardabweichung der Zellwerte in der Nachbarschaft von Pixel.

Note
Variante 1 ist eine spezialisierte Rückruffunktion, die als Rückrufparameter für ST_MapAlgebraFctNgb verwendet werden kann.

Note
Variante 2 ist eine spezialisierte Rückruffunktion, die als Rückrufparameter für ST_MapAlgebra (callback function version) verwendet werden kann.

Warning
Von der Verwendung der Variante 1 wird abgeraten, da ST_MapAlgebraFctNgb seit 2.1.0 überholt ist.

Verfügbarkeit: 2.0.0

Erweiterung: 2.1.0 Variante 2 wurde hinzugefügt

Beispiele

```
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)
```

Siehe auch

ST_MapAlgebraFctNgb, ST_MapAlgebra (callback function version), ST_Min4ma, ST_Max4ma, ST_Sum4ma, ST_Mean4ma, ST_Distinct4ma, ST_StdDev4ma

10.13.9 ST_Sum4ma

ST_Sum4ma — Funktion zur Rasterdatenverarbeitung, die die Summe aller Zellwerte in der Nachbarschaft von Pixel errechnet.

Synopsis

float8 **ST_Sum4ma**(float8[][] matrix, text nodatamode, text[] VARIADIC args);
double precision **ST_Sum4ma**(double precision[][][] value, integer[][] pos, text[] VARIADIC userargs);

Beschreibung

Berechnet die Summe aller Zellwerte in der Nachbarschaft von Pixel.

Bei Variante 2 kann ein Substitutionswert für NODATA an "userargs" übergeben werden.

Note
Variante 1 ist eine spezialisierte Rückruffunktion, die als Rückrufparameter für ST_MapAlgebraFctNgb verwendet werden kann.

Note
Variante 2 ist eine spezialisierte Rückruffunktion, die als Rückrufparameter für ST_MapAlgebra (callback function version) verwendet werden kann.

Warning
Von der Verwendung der Variante 1 wird abgeraten, da ST_MapAlgebraFctNgb seit 2.1.0 überholt ist.

Verfügbarkeit: 2.0.0

Erweiterung: 2.1.0 Variante 2 wurde hinzugefügt

Beispiele

```
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)
```

Siehe auch

ST_MapAlgebraFctNgb, ST_MapAlgebra (callback function version), ST_Min4ma, ST_Max4ma, ST_Mean4ma, ST_Range4ma, ST_Distinct4ma, ST_StdDev4ma

10.14 Raster Processing: DEM (Elevation)

10.14.1 ST_Aspect

ST_Aspect — Gibt die Exposition (standardmäßig in Grad) eines Rasterbandes mit Höhen aus. Nützlich für Terrain-Analysen.

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);

Beschreibung

Gibt die Exposition (standardmäßig in Grad) eines Höhenrasterbandes zurück. Verwendet Map Algebra und wendet die Expositionsgleichung auf benachbarte Pixel an.

`units` die Einheiten für die Exposition. Mögliche Werte sind: RADIANS, DEGREES (Standardeinstellung).

Wenn `units` = RADIANS, dann liegen die Werte zwischen 0 und 2 * pi Radiant und werden im Uhrzeigersinn von Norden her angegeben.

Wenn `units` = DEGREES, dann liegen die Werte zwischen 0 und 360 Grad und sind im Uhrzeigersinn von Norden her angegeben.

Wenn die Neigung einer Zelle null ist, dann ist die Exposition des Pixel -1.

> **Note**
> Für weitere Information über Neigung, Exposition und Schummerung siehe ESRI - How hillshade works und ERDAS Field Guide - Aspect Images.

Verfügbarkeit: 2.0.0

Erweiterung: 2.1.0 Verwendet ST_MapAlgebra() und der optionale Funktionsparameter `interpolate_nodata` wurde hinzugefügt

Änderung: 2.1.0 In Vorgängerversionen wurden die Werte in Radiant ausgegeben. Nun werden die Werte standardmäßig in Grad ausgegeben.

Beispiele: Variante 1

```
WITH foo AS (
    SELECT ST_SetValues(
        ST_AddBand(ST_MakeEmptyRaster(5, 5, 0, 0, 1, -1, 0, 0, 0), 1, '32BF', 0, -9999),
        1, 1, 1, ARRAY[
            [1, 1, 1, 1, 1],
            [1, 2, 2, 2, 1],
            [1, 2, 3, 2, 1],
            [1, 2, 2, 2, 1],
            [1, 1, 1, 1, 1]
        ]::double precision[][]
    ) AS rast
)
SELECT
    ST_DumpValues(ST_Aspect(rast, 1, '32BF'))
FROM foo




----------------------------------------------------------------------------------------------------------------------

----------------------------------
 (1,"{{315,341.565063476562,0,18.4349479675293,45},{288.434936523438,315,0,45,71.5650482177734},{270

2227,180,161.565048217773,135}}")
(1 row)
```

Beispiele: Variante 2

Ein vollständiges Beispiel mit Coveragekacheln. Diese Abfrage funktioniert nur mit PostgreSQL 9.1 oder höher.

```
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;
```

Siehe auch

ST_MapAlgebra (callback function version), ST_TRI, ST_TPI, ST_Roughness, ST_HillShade, ST_Slope

10.14.2 ST_HillShade

ST_HillShade — Gibt für gegebenen Horizontalwinkel, Höhenwinkel, Helligkeit und Maßstabsverhältnis die hypothetische Beleuchtung eines Höhenrasterbandes zurück.

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);

Beschreibung

Gibt für gegebenen Horizontalwinkel, Höhenwinkel, Helligkeit und Maßstabsverhältnis die hypothetische Beleuchtung eines Höhenrasterbandes zurück. Verwendet Map Algebra und wendet die Schummerungsgleichung auf die Nachbarpixel an. Die zurückgegebenen Pixelwerte liegen zwischen 0 und 255.

`azimuth` der Richtungswinkel zwischen 0 und 360 Grad, im Uhrzeigersinn von Norden gemessen.

`altitude` der Höhenwinkel zwischen 0 und 90 Grad, wobei 0 Grad am Horizont und 90 Grad direkt über Kopf liegt.

`max_bright` ein Wert zwischen 0 und 255, wobei 0 keine Helligkeit und 255 maximale Helligkeit bedeutet.

`scale` ist das Verhältnis zwischen vertikalen und horizontalen Einheiten. Für Feet:LatLon verwenden Sie bitte scale=370400, für Meter:LatLon scale=111120.

Wenn `interpolate_nodata` TRUE ist, dann werden die NODATA Pixel des Ausgangsraster mit ST_InvDistWeight4ma interpoliert, bevor die Schummerung berechnet wird.

Note
Für weitere Information zur Schummerung siehe How hillshade works.

Verfügbarkeit: 2.0.0

Erweiterung: 2.1.0 Verwendet ST_MapAlgebra() und der optionale Funktionsparameter `interpolate_nodata` wurde hinzugefügt

Änderung: 2.1.0 In Vorgängerversionen wurden Richtungswinkel und Höhenwinkel in Radiant angegeben. Nun werden die Werte in Grad ausgedrückt.

Beispiele: Variante 1

```
WITH foo AS (
    SELECT ST_SetValues(
        ST_AddBand(ST_MakeEmptyRaster(5, 5, 0, 0, 1, -1, 0, 0, 0), 1, '32BF', 0, -9999),
        1, 1, 1, ARRAY[
            [1, 1, 1, 1, 1],
            [1, 2, 2, 2, 1],
            [1, 2, 3, 2, 1],
            [1, 2, 2, 2, 1],
            [1, 1, 1, 1, 1]
        ]::double precision[][]
    ) AS rast
)
SELECT
    ST_DumpValues(ST_Hillshade(rast, 1, '32BF'))
FROM foo




----------------------------------------------------------------------------------------------------------------------------


-------------------------------------------------------------------------
 (1,"{{NULL,NULL,NULL,NULL,NULL},{NULL,251.32763671875,220.749786376953,147.224319458008, ↩
     NULL},{NULL,220.749786376953,180.312225341797,67.7497863769531,NULL},{NULL ↩
     ,147.224319458008
,67.7497863769531,43.1210060119629,NULL},{NULL,NULL,NULL,NULL,NULL}}")
(1 row)
```

Beispiele: Variante 2

Ein vollständiges Beispiel mit Coveragekacheln. Diese Abfrage funktioniert nur mit PostgreSQL 9.1 oder höher.

```
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;
```

Siehe auch

ST_MapAlgebra (callback function version), ST_TRI, ST_TPI, ST_Roughness, ST_Aspect, ST_Slope

10.14.3 ST_Roughness

ST_Roughness — Gibt einen Raster mit der berechneten "Rauhigkeit" des DHM zurück.

Synopsis

raster **ST_Roughness**(raster rast, integer nband, raster customextent, text pixeltype="32BF" , boolean interpolate_nodata=FALSE);

Beschreibung

Berechnet die "Rauigkeit" eines DHM, indem für ein bestimmtes Gebiet das Maximum vom Minimum abgezogen wird..

Verfügbarkeit: 2.1.0

Beispiele

```
-- needs examples
```

Siehe auch

ST_MapAlgebra (callback function version), ST_TRI, ST_TPI, ST_Slope, ST_HillShade, ST_Aspect

10.14.4 ST_Slope

ST_Slope — Gibt die Neigung (standardmäßig in Grad) eines Höhenrasterbandes zurück. Nützlich für Terrain-Analysen.

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);

Beschreibung

Gibt die Neigung (standardmäßig in Grad) eines Höhenrasterbandes zurück. Verwendet Map Algebra und wendet die Neigungsgleichung auf benachbarte Pixel an.

`units` die Einheiten für die Neigung. Mögliche Werte sind: RADIANS, DEGREES (Standardeinstellung) und Prozent.

`scale` ist das Verhältnis zwischen vertikalen und horizontalen Einheiten. Für Feet:LatLon verwenden Sie bitte scale=370400, für Meter:LatLon scale=111120.

Wenn `interpolate_nodata` TRUE ist, dann werden die NODATA Pixel des Ausgangsraster mit ST_InvDistWeight4ma interpoliert, bevor die Neigung der Oberfläche berechnet wird.

Note
Für weitere Information über Neigung, Exposition und Schummerung siehe ESRI - How hillshade works und ERDAS Field Guide - Slope Images.

Verfügbarkeit: 2.0.0

Erweiterung: 2.1.0 Verwendet ST_MapAlgebra() und es wurden die optionalen Funktionsparameter `units`, `scale` und `interpolat` hinzugefügt

Änderung: 2.1.0 In Vorgängerversionen wurden die Werte in Radiant ausgegeben. Nun werden die Werte standardmäßig in Grad ausgegeben.

Beispiele: Variante 1

```
WITH foo AS (
    SELECT ST_SetValues(
        ST_AddBand(ST_MakeEmptyRaster(5, 5, 0, 0, 1, -1, 0, 0, 0), 1, '32BF', 0, -9999),
        1, 1, 1, ARRAY[
            [1, 1, 1, 1, 1],
            [1, 2, 2, 2, 1],
            [1, 2, 3, 2, 1],
            [1, 2, 2, 2, 1],
            [1, 1, 1, 1, 1]
        ]::double precision[][]
    ) AS rast
)
SELECT
    ST_DumpValues(ST_Slope(rast, 1, '32BF'))
FROM foo

                           st_dumpvalues

------------------------------------------------------------------------------------------------------------

------------------------------------------------------------------------------------------------------------

-----------------------------------------------------------------------
```

```
 (1,"{{10.0249881744385,21.5681285858154,26.5650520324707,21.5681285858154,10.0249881744385},{21.568

{26.5650520324707,36.8698959350586,0,36.8698959350586,26.5650520324707},{21.5681285858154,35.2643890

5681285858154,26.5650520324707,21.5681285858154,10.0249881744385}}")
(1 row)
```

Beispiele: Variante 2

Ein vollständiges Beispiel mit Coveragekacheln. Diese Abfrage funktioniert nur mit PostgreSQL 9.1 oder höher.

```
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;
```

Siehe auch

ST_MapAlgebra (callback function version), ST_TRI, ST_TPI, ST_Roughness, ST_HillShade, ST_Aspect

10.14.5 ST_TPI

ST_TPI — Berechnet den "Topographic Position Index" eines Raster.

Synopsis

raster **ST_TPI**(raster rast, integer nband, raster customextent, text pixeltype="32BF" , boolean interpolate_nodata=FALSE);

Beschreibung

Berechnet den Topographischen Lageindex, welcher als fokaler Mittelwert mit dem Radius eins, abzüglich der mittigen Zelle, definiert ist.

Note
Diese Funktion unterstützt lediglich einen "focalmean"-Radius von Eins.

Verfügbarkeit: 2.1.0

Beispiele

```
-- needs examples
```

Siehe auch

ST_MapAlgebra (callback function version), ST_TRI, ST_Roughness, ST_Slope, ST_HillShade, ST_Aspect

10.14.6 ST_TRI

ST_TRI — Gibt einen Raster mit errechneten Geländerauheitsindex aus.

Synopsis

raster **ST_TRI**(raster rast, integer nband, raster customextent, text pixeltype="32BF" , boolean interpolate_nodata=FALSE);

Beschreibung

Der Geländerauheitsindex wird durch den Vergleich eines zentralen Pixel mit seinen Nachbarn berechnet. Dabei werden die Differenzen der absoluten Werte (Beträge) genommen und für das Ergebnis gemittelt.

Note
Diese Funktion unterstützt lediglich einen "focalmean"-Radius von Eins.

Verfügbarkeit: 2.1.0

Beispiele

```
-- needs examples
```

Siehe auch

ST_MapAlgebra (callback function version), ST_Roughness, ST_TPI, ST_Slope, ST_HillShade, ST_Aspect

10.15 Raster Processing: Raster to Geometry

10.15.1 Box3D

Box3D — Stellt das umschreibende Rechteck eines Raster als Box3D dar.

Synopsis

box3d **Box3D**(raster rast);

Beschreibung

Gibt ein Rechteck mit der Ausdehnung des Raster zurück.

Das Polygon ist durch die Eckpunkte des umschreibenden Rechtecks ((`MINX`, `MINY`), (`MAXX`, `MAXY`)) bestimmt

Änderung: 2.0.0 In Versionen vor 2.0 war dies üblicherweise Box2D anstelle von Box3D. Da Box2D überholt ist, wurde dies zu Box3D geändert.

Beispiele

```
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)
```

Siehe auch

ST_Envelope

10.15.2 ST_ConvexHull

ST_ConvexHull — Gibt die Geometrie der konvexen Hülle des Raster, inklusive der Pixel deren Werte gleich BandNoDataValue sind. Bei regelmäßig geformten und nicht rotierten Raster ist das Ergebnis ident mit ST_Envelope. Diese Funktion ist deshalb nur bei unregelmäßig geformten oder rotierten Raster nützlich.

Synopsis

geometry **ST_ConvexHull**(raster rast);

Beschreibung

Gibt die Geometrie der konvexen Hülle des Raster, inklusive der Pixel NoDataBandValue des Bandes. Bei regelmäßig geformten und nicht rotierten Raster ist das Ergebnis mehr oder weniger das von ST_Envelope. Diese Funktion ist deshalb nur bei unregelmäßig geformten oder rotierten Raster nützlich.

Note
ST_Envelope rundet die Koordinaten und fügt so einen kleinen Puffer um den Raster hinzu. Dadurch unterscheidet sich das Ergebnis gering von ST_ConvexHull, das nicht rundet.

Beispiele

Siehe PostGIS Raster Specification für eine entsprechende Darstellung.

```
-- 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) ↩
    )
```

Siehe auch

ST_Envelope, ST_MinConvexHull, ST_ConvexHull, ST_AsText

10.15.3 ST_DumpAsPolygons

ST_DumpAsPolygons — Gibt geomval (geom,val) Zeilen eines Rasterbandes zurück. Wenn kein Band angegeben ist, wird die Bandnummer standardmäßig auf 1 gesetzt.

Synopsis

setof geomval **ST_DumpAsPolygons**(raster rast, integer band_num=1, boolean exclude_nodata_value=TRUE);

Beschreibung

Dies ist eine Funktion mit Ergebnismenge (SRF von set-returning function). Sie gibt eine Menge an geomval Zeilen für das Band zurück, die aus einer Geometrie (geom) und einem Pixelwert (val) gebildet werden. Ein Polygon entspricht der Vereinigung der Pixel in dem Band, die denselben Pixelwert "val" aufweisen.

ST_DumpAsPolygon ist nützlich um Raster in Polygone zu vektorisieren. Im Gegensatz zu GROUP BY werden hier neue Zeilen erzeugt. Die Funktion kann zum Beispiel verwendet werden, um einen einzelnen Raster in mehrere Polygone/Mehrfach-Polygone zu expandieren.

Changed 3.3.0, validation and fixing is disabled to improve performance. May result invalid geometries.

Verfügbarkeit: Benötigt GDAL 1.7+

Note
Wenn das Band einen NODATA-Wert aufweist, dann werden die Pixel mit diesem Wert nicht zurückgegeben, außer wenn exclude_nodata_value=false.

Note
Wenn Sie nur die Anzahl der Pixel des Raster benötigen, die einen bestimmten Zellwert haben, dann ist ST_ValueCount schneller.

Note
Dies unterscheidet sich von ST_PixelAsPolygons, wo eine Geometrie für jedes Pixel zurückgegeben wird, unabhängig vom Pixelwert.

Beispiele

```
-- this syntax requires PostgreSQL 9.3+
SELECT val, ST_AsText(geom) As geomwkt
FROM (
SELECT dp.*
FROM dummy_rast, LATERAL ST_DumpAsPolygons(rast) AS dp
WHERE rid = 2
) As foo
WHERE val BETWEEN 249 and 251
ORDER BY val;

 val |                                                                        geomwkt
-----+---------------------------------------------------------------------------------
 249 | POLYGON((3427927.95 5793243.95,3427927.95 5793243.85,3427928 5793243.85,
       3427928 5793243.95,3427927.95 5793243.95))
 250 | POLYGON((3427927.75 5793243.9,3427927.75 5793243.85,3427927.8 5793243.85,
       3427927.8 5793243.9,3427927.75 5793243.9))
 250 | POLYGON((3427927.8 5793243.8,3427927.8 5793243.75,3427927.85 5793243.75,
       3427927.85 5793243.8, 3427927.8 5793243.8))
 251 | POLYGON((3427927.75 5793243.85,3427927.75 5793243.8,3427927.8 5793243.8,
       3427927.8 5793243.85,3427927.75 5793243.85))
```

Siehe auch

geomval, ST_Value, ST_Polygon, ST_ValueCount

10.15.4 ST_Envelope

ST_Envelope — Stellt die Ausdehnung des Raster als Polygon dar.

Synopsis

geometry **ST_Envelope**(raster rast);

Beschreibung

Gibt eine Polygondarstellung der Rasterausdehnung zurück. Dies ist das minimale umschreibendes Rechteck in Float8, das in dem durch die SRID festgelegten Koordinatenreferenzsystem als Polygon dargestellt wird.

Das Polygon wird durch die Eckpunkte des umschreibenden Rechtecks ((MINX, MINY), (MINX, MAXY), (MAXX, MAXY), (MAXX, MINY), (MINX, MINY)) festgelegt.

Beispiele

```
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))
```

Siehe auch

ST_Envelope, ST_AsText, ST_SRID

10.15.5 ST_MinConvexHull

ST_MinConvexHull — Gibt die Geometrie der konvexen Hülle des Raster aus, wobei Pixel mit NODATA ausgenommen werden.

Synopsis

geometry **ST_MinConvexHull**(raster rast, integer nband=NULL);

Beschreibung

Gibt die Geometrie der konvexen Hülle des Raster aus, wobei Pixel mit NODATA ausgenommen werden. Wenn nband NULL ist, dann werden alle Bänder des Raster berücksichtigt.

Verfügbarkeit: 2.1.0

Beispiele

```
WITH foo AS (
    SELECT
        ST_SetValues(
            ST_SetValues(
                ST_AddBand(ST_AddBand(ST_MakeEmptyRaster(9, 9, 0, 0, 1, -1, 0, 0, 0), 1, '8 ↩
                    BUI', 0, 0), 2, '8BUI', 1, 0),
                1, 1, 1,
                ARRAY[
                    [0, 0, 0, 0, 0, 0, 0, 0, 0],
                    [0, 0, 0, 0, 0, 0, 0, 0, 0],
                    [0, 0, 0, 0, 0, 0, 0, 0, 0],
                    [0, 0, 0, 1, 0, 0, 0, 0, 1],
                    [0, 0, 0, 1, 1, 0, 0, 0, 0],
                    [0, 0, 0, 1, 0, 0, 0, 0, 0],
                    [0, 0, 0, 0, 0, 0, 0, 0, 0],
```

```
                    [0, 0, 0, 0, 0, 0, 0, 0, 0],
                    [0, 0, 0, 0, 0, 0, 0, 0, 0]
                ]::double precision[][]
            ),
            2, 1, 1,
            ARRAY[
                [0, 0, 0, 0, 0, 0, 0, 0, 0],
                [0, 0, 0, 0, 0, 0, 0, 0, 0],
                [0, 0, 0, 0, 0, 0, 0, 0, 0],
                [1, 0, 0, 0, 0, 1, 0, 0, 0],
                [0, 0, 0, 0, 1, 1, 0, 0, 0],
                [0, 0, 0, 0, 0, 1, 0, 0, 0],
                [0, 0, 0, 0, 0, 0, 0, 0, 0],
                [0, 0, 0, 0, 0, 0, 0, 0, 0],
                [0, 0, 1, 0, 0, 0, 0, 0, 0]
            ]::double precision[][]
        ) AS rast
)
SELECT
    ST_AsText(ST_ConvexHull(rast)) AS hull,
    ST_AsText(ST_MinConvexHull(rast)) AS mhull,
    ST_AsText(ST_MinConvexHull(rast, 1)) AS mhull_1,
    ST_AsText(ST_MinConvexHull(rast, 2)) AS mhull_2
FROM foo

               hull                |                mhull                |                ↩
                   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))
```

Siehe auch

ST_Envelope, ST_ConvexHull, ST_ConvexHull, ST_AsText

10.15.6 ST_Polygon

ST_Polygon — Gibt eine Geometrie mit Mehrfachpolygonen zurück, die aus der Vereinigung von Pixel mit demselben Zellwert gebildet werden. Pixel mit NODATA Werten werden nicht berücksichtigt. Wenn keine Band angegeben ist, wird die Bandnummer standardmäßig auf 1 gesetzt.

Synopsis

geometry **ST_Polygon**(raster rast, integer band_num=1);

Beschreibung

Changed 3.3.0, validation and fixing is disabled to improve performance. May result invalid geometries.

Verfügbarkeit: 0.1.6 - benötigt GDAL 1.7+

Erweiterung: 2.1.0 Geschwindigkeit verbessert (zur Gänze C-basiert) und es wird sichergestellt, dass das zurückgegebenen Mehrfachpolygon valide ist.

Änderung: 2.1.0 In Vorgängerversionen wurde manchmal ein Polygon zurückgegeben; dies wurde geändert so dass jetzt immer ein Mehrfachpolygon zurückgegeben wird.

Beispiele

```
-- by default no data band value is 0 or not set, so polygon will return a square polygon
SELECT ST_AsText(ST_Polygon(rast)) As geomwkt
FROM dummy_rast
WHERE rid = 2;

geomwkt
-----------------------------------------
MULTIPOLYGON(((3427927.75 5793244,3427928 5793244,3427928 5793243.75,3427927.75  ↩
    5793243.75,3427927.75 5793244)))


-- now we change the no data value of first band
UPDATE dummy_rast SET rast = ST_SetBandNoDataValue(rast,1,254)
WHERE rid = 2;
SELECt rid, ST_BandNoDataValue(rast)
from dummy_rast where rid = 2;

-- ST_Polygon excludes the pixel value 254 and returns a multipolygon
SELECT ST_AsText(ST_Polygon(rast)) As geomwkt
FROM dummy_rast
WHERE rid = 2;

geomwkt
---------------------------------------------------------
MULTIPOLYGON(((3427927.9 5793243.95,3427927.85 5793243.95,3427927.85 5793244,3427927.9  ↩
    5793244,3427927.9 5793243.95)),((3427928 5793243.85,3427928 5793243.8,3427927.95  ↩
    5793243.8,3427927.95 5793243.85,3427927.9 5793243.85,3427927.9 5793243.9,3427927.9  ↩
    5793243.95,3427927.95 5793243.95,3427928 5793243.95,3427928 5793243.85)),((3427927.8  ↩
    5793243.75,3427927.75 5793243.75,3427927.75 5793243.8,3427927.75 5793243.85,3427927.75  ↩
    5793243.9,3427927.75 5793244,3427927.8 5793244,3427927.8 5793243.9,3427927.8  ↩
    5793243.85,3427927.85 5793243.85,3427927.85 5793243.8,3427927.85 5793243.75,3427927.8  ↩
    5793243.75)))

-- Or if you want the no data value different for just one time

SELECT ST_AsText(
    ST_Polygon(
        ST_SetBandNoDataValue(rast,1,252)
        )
    ) As geomwkt
FROM dummy_rast
WHERE rid =2;

geomwkt
---------------------------------
MULTIPOLYGON(((3427928 5793243.85,3427928 5793243.8,3427928 5793243.75,3427927.85  ↩
    5793243.75,3427927.8 5793243.75,3427927.8 5793243.8,3427927.75 5793243.8,3427927.75  ↩
    5793243.85,3427927.75 5793243.9,3427927.75 5793244,3427927.8 5793244,3427927.85  ↩
    5793244,3427927.9 5793244,3427928 5793244,3427928 5793243.95,3427928 5793243.85) ↩
    ,(3427927.9 5793243.9,3427927.9 5793243.85,3427927.95 5793243.85,3427927.95  ↩
    5793243.9,3427927.9 5793243.9)))
```

Siehe auch

ST_Value, ST_DumpAsPolygons

10.16 Rasteroperatoren

10.16.1 &&

&& — Gibt TRUE zurück, wenn das umschreibende Rechteck von A das umschreibende Rechteck von B schneidet.

Synopsis

boolean **&&**(raster A , raster B);
boolean **&&**(raster A , geometry B);
boolean **&&**(geometry B , raster A);

Beschreibung

Der Operator && gibt TRUE zurück, wenn das umschreibende Rechteck von Raster/Geometrie A das umschreibende Rechteck von Raster/Geometrie B schneidet.

Note
Dieser Operand benützt jeden Index, der für den Raster zur Verfügung stellt.

Verfügbarkeit: 2.0.0

Beispiele

```
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
```

10.16.2 &<

&< — Gibt TRUE zurück, wenn das umschreibende Rechteck von A links von dem von B liegt.

Synopsis

boolean **&<**(raster A , raster B);

Beschreibung

Der &< Operator gibt TRUE zurück, wenn das umschreibende Rechteck von Raster A das umschreibende Rechteck von Raster B überlagert oder links davon liegt, oder präziser, überlagert und NICHT rechts des umschreibenden Rechtecks von Raster B liegt.

Note
Dieser Operand benützt jeden Index, der für den Raster zur Verfügung stellt.

Beispiele

```
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
```

10.16.3 &>

&> — Gibt TRUE zurück, wenn das umschreibende Rechteck von A rechts von dem von B liegt.

Synopsis

boolean **&>**(raster A , raster B);

Beschreibung

Der &> Operator gibt TRUE zurück, wenn das umschreibende Rechteck von Raster A das umschreibende Rechteck von Raster B überlagert oder rechts davon liegt, oder präziser, überlagert und NICHT links des umschreibenden Rechtecks von Raster B liegt.

Note
Dieser Operand benützt jeden Index, der für die Geometrie zur Verfügung stellt.

Beispiele

```
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
```

10.16.4 =

= — Gibt TRUE zurück, wenn die umschreibenden Rechtecke von A und B ident sind. Das umschreibende Rechteck ist in Double Precision.

Synopsis

boolean =(raster A , raster B);

Beschreibung

Der = Operator gibt TRUE zurück, wenn das umschreibende Rechteck von Raster A ident mit dem umschreibenden Rechteck von Raster B ist. PostgreSQL verwendet die =, <, und > Operatoren um die interne Sortierung und den Vergleich von Raster durchzuführen (z.B.: in einer GROUP BY oder ORDER BY Klausel).

Caution
Dieser Operator verwendet NICHT die für Raster verfügbaren Indizes. Verwenden Sie bitte ~= stattdessen. Dieser Operator existiert meistens, so dass man nach der Rasterspalte gruppieren kann.

Verfügbarkeit: 2.1.0

Siehe auch

~=

10.16.5 @

@ — Gibt TRUE zurück, wenn das umschreibende Rechteck von A in jenem von B enthalten ist. Das umschreibende Rechteck ist in Double Precision.

Synopsis

boolean @(raster A , raster B);
boolean @(geometry A , raster B);
boolean @(raster B , geometry A);

Beschreibung

Der @ Operator gibt TRUE zurück, wenn das umschreibende Rechteck von Raster/Geometrie A in dem umschreibenden Rechteck von Raster/Geometrie B enthalten ist.

Note
Dieser Operand verwendet die räumichen Indizes der Raster.

Verfügbarkeit: 2.0.0 raster @ raster, und raster @ geometry eingeführt

Verfügbarkeit: 2.0.5 "geometry @ raster" eingeführt

Siehe auch

~

10.16.6 ~=

~= — Gibt TRUE zurück wenn die Umgebungsrechtecke von "A" und "B" ident sind.

Synopsis

boolean **~=**(raster A , raster B);

Beschreibung

Der Operator ~= gibt TRUE zurück, wenn die Umgebungsrechtecke der Raster "A" und "B" ident sind.

Note
Dieser Operand benützt jeden Index, der für den Raster zur Verfügung stellt.

Verfügbarkeit: 2.0.0

Beispiele

Ein sinnvoller Anwendungsfall wäre, aus zwei Einzelbandraster mit den gleichen Eigenschaften aber unterschiedlicher Thematik, einen Multi-Bandraster zu erstellen.

```
SELECT ST_AddBand(prec.rast, alt.rast) As new_rast
    FROM prec INNER JOIN alt ON (prec.rast ~= alt.rast);
```

Siehe auch

ST_AddBand, =

10.16.7 ~

~ — Gibt TRUE zurück, wenn das umschreibende Rechteck von A jenes von B enthält. Das umschreibende Rechteck ist in Double Precision.

Synopsis

boolean ~(raster A , raster B);
boolean ~(geometry A , raster B);
boolean ~(raster B , geometry A);

Beschreibung

Der Operator ~ gibt `TRUE` zurück, wenn das umschreibende Rechteck von Raster/Geometrie A das umschreibende Rechteck von Raster/Geometrie B enthält.

Note
Dieser Operand verwendet die räumichen Indizes der Raster.

Verfügbarkeit: 2.0.0

Siehe auch

@

10.17 Räumliche Beziehungen von Rastern und Rasterbändern

10.17.1 ST_Contains

ST_Contains — Gibt TRUE zurück, wenn kein Punkt des Rasters "rastB" im Äußeren des Rasters "rastA" liegt und zumindest ein Punkt im Inneren von "rastB" auch im Inneren von "rastA" liegt.

Synopsis

boolean **ST_Contains**(raster rastA , integer nbandA , raster rastB , integer nbandB);
boolean **ST_Contains**(raster rastA , raster rastB);

Beschreibung

Raster "rastA" enthält dann und nur dann "rastB", wenn sich kein Punkt von "rastB" im Äußeren des Rasters "rastA" befindet und zumindest ein Punkt im Inneren von "rastB" auch im Inneren von "rastA" liegt. Wenn die Bandnummer nicht angegeben ist (oder auf NULL gesetzt), dann wird nur die konvexe Hülle des Rasters zur Überprüfung herangezogen. Wenn die Bandnummer angegeben ist, werden nur die Pixel mit einem Wert (nicht NODATA) bei der Überprüfung herangezogen.

Note
Diese Funktion verwendet die für Raster verfügbaren Indizes.

Note
Um die räumliche Beziehung zwischen einem Raster und einem geometrischen Datentyp zu überprüfen, verwenden Sie bitte ST_Polygon für den Raster, z.B. ST_Contains(ST_Polygon(raster), geometry) or ST_Contains(geometry, ST_Polygon(raster)).

Note
ST_Contains() ist das Gegenstück zuf ST_Within(). Daher impliziert ST_Contains(rastA, rastB) ST_Within(rastB, rastA).

Verfügbarkeit: 2.1.0

Beispiele

```
-- 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
```

Siehe auch

ST_Intersects, ST_Within

10.17.2 ST_ContainsProperly

ST_ContainsProperly — Gibt TRUE zurück, wenn "rastB" das Innere von "rastA" schneidet, aber nicht die Begrenzung oder das Äußere von "rastA".

Synopsis

boolean **ST_ContainsProperly**(raster rastA , integer nbandA , raster rastB , integer nbandB);
boolean **ST_ContainsProperly**(raster rastA , raster rastB);

Beschreibung

Raster "rastA" enthält "rastB" vollständig, wenn "rastB" nur das Innere von "rastA" schneidet, die Begrenzung und das Äußere von "rastA" jedoch nicht. Wenn die Bandnummer nicht angegeben ist (oder auf NULL gesetzt), dann wird nur die konvexe Hülle des Rasters zur Überprüfung herangezogen. Wenn die Bandnummer angegeben ist, werden nur die Pixel mit einem Wert (nicht NODATA) bei der Überprüfung herangezogen.

Raster "rastA" enthält sich selbst, aber enthält sich nicht zur Gänze selbst.

Note
Diese Funktion verwendet die für Raster verfügbaren Indizes.

Note
Um die räumliche Beziehung zwischen einem Raster und einem geometrischen Datentyp zu überprüfen, verwenden Sie bitte ST_Polygon für den Raster, z.B. ST_ContainsProperly(ST_Polygon(raster), geometry) or ST_ContainsProperly(geometry, ST_Polygon(raster)).

Verfügbarkeit: 2.1.0

Beispiele

```
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
```

Siehe auch

ST_Intersects, ST_Contains

10.17.3 ST_Covers

ST_Covers — Gibt TRUE zurück, wenn kein Punkt des Rasters "rastB" außerhalb des Rasters "rastA" liegt.

Synopsis

boolean **ST_Covers**(raster rastA , integer nbandA , raster rastB , integer nbandB);
boolean **ST_Covers**(raster rastA , raster rastB);

Beschreibung

Raster "rastA" deckt "rastB" dann und nur dann ab, wenn kein Punkt von "rastB" im Äußeren von "rastA" liegt. Wenn die Bandnummer nicht angegeben ist (oder auf NULL gesetzt), dann wird nur die konvexe Hülle des Rasters zur Überprüfung herangezogen. Wenn die Bandnummer angegeben ist, werden nur die Pixel mit einem Wert (nicht NODATA) bei der Überprüfung herangezogen.

Note
Diese Funktion verwendet die für Raster verfügbaren Indizes.

Note
Um die räumliche Beziehung zwischen einem Raster und einem geometrischen Datentyp zu überprüfen, verwenden Sie bitte ST_Polygon für den Raster, z.B. ST_Covers(ST_Polygon(raster), geometry) or ST_Covers(geometry, ST_Polygon(raster)).

Verfügbarkeit: 2.1.0

Beispiele

```
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
```

Siehe auch

ST_Intersects, ST_CoveredBy

10.17.4 ST_CoveredBy

ST_CoveredBy — Gibt TRUE zurück, wenn kein Punkt des Rasters "rastA" außerhalb des Rasters "rastB" liegt.

Synopsis

boolean **ST_CoveredBy**(raster rastA , integer nbandA , raster rastB , integer nbandB);
boolean **ST_CoveredBy**(raster rastA , raster rastB);

Beschreibung

Raster "rastA" wird von "rastB" dann und nur dann abgedeckt, wenn kein Punkt von "rastA" im Äußeren von "rastB" liegt. Wenn die Bandnummer nicht angegeben ist (oder auf NULL gesetzt), dann wird nur die konvexe Hülle des Rasters zur Überprüfung herangezogen. Wenn die Bandnummer angegeben ist, werden nur die Pixel mit einem Wert (nicht NODATA) bei der Überprüfung herangezogen.

Note
Diese Funktion verwendet die für Raster verfügbaren Indizes.

Note
Um die räumliche Beziehung zwischen einem Raster und einem geometrischen Datentyp zu überprüfen, verwenden Sie bitte ST_Polygon für den Raster, z.B. ST_CoveredBy(ST_Polygon(raster), geometry) or ST_CoveredBy(geometry, ST_Polygon(raster)).

Verfügbarkeit: 2.1.0

Beispiele

```
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
```

Siehe auch

ST_Intersects, ST_Covers

10.17.5 ST_Disjoint

ST_Disjoint — Gibt TRUE zurück, wenn sich die Raster "rastA" und "rastB" räumlich nicht überschneiden.

Synopsis

boolean **ST_Disjoint**(raster rastA , integer nbandA , raster rastB , integer nbandB);
boolean **ST_Disjoint**(raster rastA , raster rastB);

Beschreibung

Raster "rastA" und "rastB" sind getrennt voneinander (disjoint) wenn sie sich keinen gemeinsamen Raum teilen. Wenn die Bandnummer nicht angegeben ist (oder auf NULL gesetzt), dann wird nur die konvexe Hülle des Rasters zur Überprüfung herangezogen. Wenn die Bandnummer angegeben ist, werden nur die Pixel mit einem Wert (nicht NODATA) bei der Überprüfung herangezogen.

Note
Diese Funktion verwendet keine Indizes.

Note
Um die räumliche Beziehung zwischen einem Raster und einem geometrischen Datentyp zu überprüfen, verwenden Sie bitte ST_Polygon für den Raster, z.B. ST_Disjoint(ST_Polygon(raster), geometry).

Verfügbarkeit: 2.1.0

Beispiele

```
-- 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
```

Siehe auch

ST_Intersects

10.17.6 ST_Intersects

ST_Intersects — Gibt TRUE zurück, wenn sich die Raster "rastA" und "rastB" nicht räumlich überschneiden.

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);

Beschreibung

Gibt TRUE zurück, wenn der Raster "rastA" den Raster "rastB" räumlich schneidet. Wenn die Bandnummer nicht angegeben ist (oder auf NULL gesetzt), dann wird nur die konvexe Hülle des Rasters zur Überprüfung herangezogen. Wenn die Bandnummer angegeben ist, werden nur die Pixel mit einem Wert (nicht NODATA) bei der Überprüfung herangezogen.

Note
Diese Funktion verwendet die für Raster verfügbaren Indizes.

Erweiterung: 2.0.0 Unterstützung für Raster/Raster Verschneidungen eingeführt.

Warning
Änderung: 2.1.0 Die Verhaltensweise der Varianten von ST_Intersects(raster, geometry) wurde geändert um mit dem von ST_Intersects(geometry, raster) übereinzustimmen.

Beispiele

```
-- different bands of same raster
SELECT ST_Intersects(rast, 2, rast, 3) FROM dummy_rast WHERE rid = 2;

 st_intersects
---------------
 t
```

Siehe auch

ST_Intersection, ST_Disjoint

10.17.7 ST_Overlaps

ST_Overlaps — Gibt TRUE zurück, wenn sich die Raster "rastA" und "rastB" schneiden, aber ein Raster den anderen nicht zur Gänze enthält.

Synopsis

boolean **ST_Overlaps**(raster rastA , integer nbandA , raster rastB , integer nbandB);
boolean **ST_Overlaps**(raster rastA , raster rastB);

Beschreibung

Gibt TRUE zurück, wenn der Raster "rastA" den Raster "rastB" überlagert. Dies bedeutet, das rastA und rastB sich schneiden aber einer den anderen nicht zur Gänze enthält. Wenn die Bandnummer nicht angegeben ist (oder auf NULL gesetzt), dann wird nur die konvexe Hülle des Rasters zur Überprüfung herangezogen. Wenn die Bandnummer angegeben ist, werden nur die Pixel mit einem Wert (nicht NODATA) bei der Überprüfung herangezogen.

Note
Diese Funktion verwendet die für Raster verfügbaren Indizes.

Note
Um die räumliche Beziehung zwischen einem Raster und einem geometrischen Datentyp zu überprüfen, verwenden Sie bitte ST_Polygon für den Raster, z.B. ST_Overlaps(ST_Polygon(raster), geometry).

Verfügbarkeit: 2.1.0

Beispiele

```
-- comparing different bands of same raster
SELECT ST_Overlaps(rast, 1, rast, 2) FROM dummy_rast WHERE rid = 2;

 st_overlaps
-------------
 f
```

Siehe auch

ST_Intersects

10.17.8 ST_Touches

ST_Touches — Gibt TRUE zurück, wenn rastA und rastB zumindest einen Punkt gemeinsam haben sich aber nicht überschneiden.

Synopsis

boolean **ST_Touches**(raster rastA , integer nbandA , raster rastB , integer nbandB);
boolean **ST_Touches**(raster rastA , raster rastB);

Beschreibung

Gibt TRUE zurück, wenn der Raster "rastA" den Raster "rastB" räumlich berührt. Dies bedeutet, das rastA und rastB zumindest einen Punkt gemeinsam haben, ihr Inneres sich aber nicht überschneidet. Wenn die Bandnummer nicht angegeben ist (oder auf NULL gesetzt), dann wird nur die konvexe Hülle des Rasters zur Überprüfung herangezogen. Wenn die Bandnummer angegeben ist, werden nur die Pixel mit einem Wert (nicht NODATA) bei der Überprüfung herangezogen.

Note
Diese Funktion verwendet die für Raster verfügbaren Indizes.

Note
Um die räumliche Beziehung zwischen einem Raster und einem geometrischen Datentyp zu überprüfen, verwenden Sie bitte ST_Polygon für den Raster, z.B. ST_Touches(ST_Polygon(raster), geometry).

Verfügbarkeit: 2.1.0

Beispiele

```
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
```

Siehe auch

ST_Intersects

10.17.9 ST_SameAlignment

ST_SameAlignment — Gibt TRUE zurück, wenn die Raster die selbe Rotation, Skalierung, Koordinatenreferenzsystem und Versatz (Pixel können auf dasselbe Gitter gelegt werden, ohne dass die Gitterlinien durch die Pixel schneiden) aufweisen. Wenn nicht, wird FALSE und eine Bechreibung des Problems ausgegeben.

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);

Beschreibung

Nicht-Aggregat Version (Versionen 1 und 2): Gibt TRUE zurück, wenn die zwei Raster (die entweder direkt übergeben oder mit Werten für UpperLeft, Scale, Skew und SRID erstellt werden) dieselbe Skalierung, Rotation und SRID haben und zumindest eine der vier Ecken eines Pixel des einen Raster auf eine Ecke des anderen Rastergitters fällt. Wenn nicht, wird FALSE und eine Problembeschreibung ausgegeben.

Aggregat Version (Variante 3): Gibt TRUE zurück, wenn alle übergebenen Raster gleich ausgerichtet sind. Die Funktion ST_SameAlignment() ist in der Terminologie von PostgreSQL eine Aggregatfunktion. Dies bedeutet, dass sie so wie die Funktionen SUM() und AVG() mit Datenzeilen arbeitet.

Verfügbarkeit: 2.0.0

Erweiterung: 2.1.0 die Variante mit der Aggregatfunktion hinzugefügt

Beispiele: Raster

```
SELECT ST_SameAlignment(
    ST_MakeEmptyRaster(1, 1, 0, 0, 1, 1, 0, 0),
    ST_MakeEmptyRaster(1, 1, 0, 0, 1, 1, 0, 0)
) as sm;

sm
----
t
```

```
SELECT ST_SameAlignment(A.rast,b.rast)
 FROM dummy_rast AS A CROSS JOIN dummy_rast AS B;

 NOTICE:  The two rasters provided have different SRIDs
NOTICE:  The two rasters provided have different SRIDs
 st_samealignment
------------------
 t
 f
 f
 f
```

Siehe auch

Section 9.1, ST_NotSameAlignmentReason, ST_MakeEmptyRaster

10.17.10 ST_NotSameAlignmentReason

ST_NotSameAlignmentReason — Gibt eine Meldung aus, die angibt ob die Raster untereinander ausgerichtet sind oder nicht und warum wenn nicht.

Synopsis

text **ST_NotSameAlignmentReason**(raster rastA, raster rastB);

Beschreibung

Gibt eine Meldung aus, die angibt ob die Raster untereinander ausgerichtet sind oder nicht und warum wenn nicht.

Note
Falls es mehrere Gründe gibt, warum die Raster nicht untereinander ausgerichtet sind, so wird nur der erste Grund (die erste fehlgeschlagene Überprüfung) ausgegeben.

Verfügbarkeit: 2.1.0

Beispiele

```
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)
```

Siehe auch

Section 9.1, ST_SameAlignment

10.17.11 ST_Within

ST_Within — Gibt TRUE zurück, wenn kein Punkt des Rasters "rastA" außerhalb des Rasters "rastB" liegt und zumindest ein Punkt im Inneren von "rastA" auch im Inneren von "rastB" liegt.

Synopsis

boolean **ST_Within**(raster rastA , integer nbandA , raster rastB , integer nbandB);
boolean **ST_Within**(raster rastA , raster rastB);

Beschreibung

Raster "rastA" liegt dann und nur dann "within"/innerhalb von "rastB", wenn sich kein Punkt von "rastA" im Äußeren des Rasters "rastB" befindet und zumindest ein Punkt im Inneren von "rastA" auch im Inneren von "rastB" liegt. Wenn die Bandnummer nicht angegeben ist (oder auf NULL gesetzt), dann wird nur die konvexe Hülle des Rasters zur Überprüfung herangezogen. Wenn die Bandnummer angegeben ist, werden nur die Pixel mit einem Wert (nicht NODATA) bei der Überprüfung herangezogen.

Note
Dieser Operand benützt jeden Index, der für den Raster zur Verfügung stellt.

Note
Um die räumliche Beziehung zwischen einem Raster und einer Geometrie zu überprüfen, verwenden Sie bitte ST_Polygon für den Raster, z.B. ST_Within(ST_Polygon(raster), geometry) or ST_Within(geometry, ST_Polygon(raster)).

Note
ST_Within() ist die Umkehrfunktion von ST_Contains(). Es gilt daher: ST_Within(rastA, rastB) impliziert ST_Contains(rastB, rastA).

Verfügbarkeit: 2.1.0

Beispiele

```
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
```

Siehe auch

ST_Intersects, ST_Contains, ST_DWithin, ST_DFullyWithin

10.17.12 ST_DWithin

ST_DWithin — Gibt TRUE zurück, wenn die Raster "rastA" und "rastB" innerhalb der angegebenen Entfernung voneinander liegen.

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);

Beschreibung

Gibt TRUE zurück, wenn die Raster "rastA" und "rastB" innerhalb der angegebenen Distanz zueinander liegen. Wenn die Bandnummer nicht angegeben ist (oder auf NULL gesetzt), dann wird nur die konvexe Hülle des Rasters zur Überprüfung herangezogen. Wenn die Bandnummer angegeben ist, werden nur die Pixel mit einem Wert (nicht NODATA) bei der Überprüfung herangezogen.

Die Distanz wird in den Einheiten des Koordinatenreferenzsystems des Rasters angegeben. Damit diese Funktion Sinn hat, müssen die Quellraster dieselbe Projektion und die gleiche SRID haben.

Note
Dieser Operand benützt jeden Index, der für den Raster zur Verfügung stellt.

Note
Um die räumliche Beziehung zwischen einem Raster und einer Geometrie zu untersuchen, wenden Sie bitte ST_Polygon auf den Raster an, z.B.: ST_DWithin(ST_Polygon(raster), geometry).

Verfügbarkeit: 2.1.0

Beispiele

```
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
```

Siehe auch

ST_Within, ST_DFullyWithin

10.17.13 ST_DFullyWithin

ST_DFullyWithin — Gibt TRUE zurück, wenn die Raster "rastA" und "rastB" zur Gänze innerhalb der angegebenen Distanz zueinander liegen.

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);

Beschreibung

Gibt TRUE zurück, wenn die Raster "rastA" und "rastB" zur Gänze innerhalb der angegebenen Distanz zueinander liegen. Wenn die Bandnummer nicht angegeben ist (oder auf NULL gesetzt), dann wird nur die konvexe Hülle des Rasters zur Überprüfung herangezogen. Wenn die Bandnummer angegeben ist, werden nur die Pixel mit einem Wert (nicht NODATA) bei der Überprüfung herangezogen.

Die Distanz wird in den Einheiten des Koordinatenreferenzsystems des Rasters angegeben. Damit diese Funktion Sinn hat, müssen die Quellraster dieselbe Projektion und die gleiche SRID haben.

Note
Dieser Operand benützt jeden Index, der für den Raster zur Verfügung stellt.

Note
Um die räumliche Relation zwischen einem Raster und einer Geometrie zu untersuchen, wenden Sie bitte ST_Polygon auf den Raster an, z.B.: ST_DFullyWithin(ST_Polygon(raster), geometry).

Verfügbarkeit: 2.1.0

Beispiele

```
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
```

Siehe auch

ST_Within, ST_DWithin

10.18 Raster Tipps

10.18.1 Out-DB Raster

10.18.1.1 Das Verzeichnis enthält eine Vielzahl an Dateien

Wenn GDAL eine Datei öffnet, dann liest es eifrig das gesamte Verzeichnis in dem sich die Datei befindet um einen Katalog mit den weieren Dateien zu erstellen. Wenn dieses Verzeichnis viele Dateien (z.B.: Tausende, Millionen) enthält, kann das Öffnen dieser Datei extrem lange dauern (insbesondere wenn sich die Datei auf einem Netzlaufwerk, wie einem NFS befindet).

Dieses Verhalten kann durch folgende Umgebungsvariable von GDAL beeinflusst werden: GDAL_DISABLE_READDIR_ON_OPEN. Setzen Sie `GDAL_DISABLE_READDIR_ON_OPEN` auf `TRUE` um das Scannen von Verzeichnissen zu verhindern.

Auf Ubuntu (angenommen Sie verwenden ein PostgreSQL Paket für Ubuntu), kann `GDAL_DISABLE_READDIR_ON_OPEN` in */etc/postgresql/POSTGRESQL_VERSION/CLUSTER_NAME/environment* gesetzt werden (wobei POSTGRESQL_VERSION der Version von PostgreSQL entspricht, z.B. 9.6 und CLUSTER_NAME der Bezeichnung des Datenbankclusters, z.B. maindb). Sie können hier ebenso die Umgebungsvariablen von PostGIS setzen.

```
# 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'
```

10.18.1.2 Die maximale Anzahl geöffneter Dateien

Die Einstellungen von Linux und PostgreSQL bezüglich der maximal erlaubten Anzahl von offenen Dateien sind üblicherweise sehr konservativ (normalerweise 1024 offene Dateien pro Prozess), da sie unter der Annahme getroffen wurden, dass das System von Menschen genutzt wird. Bei Out-DB Rastern kann eine einzelne Abfrage spielend dieses Limit überschreiten (z.B. ein Datensatz mit Rasterwerten über 10 Jahre, wobei ein Raster die Tageswerte der niedrigsten und höchsten Temperaturwerte enthält und wir den absoluten Mindest- und Höchstwert des Datensatzes abfragen wollen).

Am einfachsten kann dies über die PostgreSQL Einstellung max_files_per_process geändert werden. Der Standardwert von 1000 ist für Out-DB Raster viel zu niedrig. Ein zuverlässiger Anfangswert könnte 65536 sein, wobei dies jedoch sehr stark von den verwendeten Datensätzen abhängt und den Abfragen die Sie auf diese ausführen wollen. Diese Einstellung muss vor dem Starten des Servers gesetzt werden und kann vermutlich nur in der Konfigurationsdatei von PostgreSQL (z.B. /etc/postgresql/-POSTGRESQL_VERSION/CLUSTER_NAME/postgresql.conf auf Ubuntu) vorgenommen werden.

```
...
# - Kernel Resource Usage -

max_files_per_process = 65536           # min 25
                                        # (change requires restart)
...
```

Die wesentliche Änderung muss an den Limits des Linux Kernels für offene Dateien vorgenommen werden. Dies umfasst zwei Teile:

- Die maximale Anzahl geöffneter Dateien für das ganze System
- Die maximale Anzahl geöffneter Dateien pro Prozess

10.18.1.2.1 Die maximale Anzahl geöffneter Dateien für das ganze System

Das folgende Beispiel zeigt, wie Sie die aktuelle Einstellung zu der maximalen Anzahl geöffneter Dateien für das ganze System anzeigen können:

```
$ sysctl -a | grep fs.file-max
fs.file-max = 131072
```

Wenn der ausgegebene Wert zu niedrig ist, können Sie wie im folgenden Beispiel gezeigt wird, eine Datei zu */etc/sysctl.d/* hinzufügen:

```
$ 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
```

10.18.1.2.2 Die maximale Anzahl geöffneter Dateien pro Prozess

Die maximale Anzahl geöffneter Dateien pro Process für die Serverprozesse von PostgreSQL sollten geändert werden.

Um die maximale Anzahl geöffneter Dateien herauszufinden, welche von den Prozessen des PostgreSQL Dienstes genutzt werden, können Sie folgendes ausführen (stellen Sie sicher, dass PostgreSQL läuft):

```
$ 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
```

Im oberen Beispiel haben wir die Limits für geöffnete Dateien für den Prozess 31718 ausgelesen. Es spielt dabei keine Rolle welcher Prozess von PostgreSQL, es genügt jeder. Von Interesse ist dabei die Rückmeldung von *Max open files*.

Wir wollen das *Soft Limit* und das *Hard Limit* für die *Max open files* so erhöhen, dass sie über dem Wert liegen den wir in der Einstellung von PostgreSQL für `max_files_per_process` angegeben haben. In unserem Beispiel haben wir `max_files_per_pro` mit 65536 angegeben.

Auf Ubuntu (angenommen Sie verwenden ein PostgreSQL Paket für Ubuntu), kann das *Soft Limit* und das *Hard Limit* am einfachsten durch editieren von **/etc/init.d/postgresql** (SysV) oder **/lib/systemd/system/postgresql*.service** (systemd) geändert werden.

Befassen wir uns zuerst mit dem Fall SysV auf Ubuntu, bei dem wir **ulimit -H -n 262144** und **ulimit -n 131072** zu **/etc/init.d/postgresql** hinzufügen.

```
...
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)
...
```

Nun der Fall mit systemd unter Ubuntu. Wir fügen **LimitNOFILE=131072** in jeder Datei **/lib/systemd/system/postgresql*.service** in dem Abschnitt **[Service]** ein.

```
...
[Service]

LimitNOFILE=131072

...

[Install]
WantedBy=multi-user.target
...
```

Nach den erforderlichen systemd Änderungen müssen Sie den Dämon neu laden

```
systemctl daemon-reload
```

Chapter 11

PostGIS Extras

Dieses Kapitel beschreibt Funktionen, die sich in dem Verzeichnis "extras" des PostGIS Quellcodes (Tarball oder Repository) befinden. Diese sind nicht immer mit der binären PostGIS Release paketiert, es handelt sich dabei aber üblicherweise um Pl/Pgsql- oder Shell-Skripts, die direkt aufgerufen werden können.

11.1 Adressennormierer

Dies ist ein Entwicklungszweig des PAGC Adressennormierers (der Code für diesen Teilbereich beruht auf dem PAGC Adressennormierer für PostgreSQL).

Der Adressennormierer ist ein Parser für einzeilige Adressen. Eine gegebene Adresse wird anhand von in einer Tabelle abgelegten Regeln und den Hilfstabellen "lex" und "gaz" normiert.

Der Code befindet sich in einer einzelnen PostgreSQL Erweiterungsbibliothek mit der Bezeichnung `address_standardizer` und kann mittels `CREATE EXTENSION address_standardizer;` installiert werden. Zusätzlich zu der Erweiterung "address_standardizer" gibt es auch die Erweiterung `address_standardizer_data_us`, welche die Tabellen "gaz", "lex" und "rules" für Daten der USA enthält. Diese Erweiterung kann mittels `CREATE EXTENSION address_standardizer_data_us;` installiert werden.

Der Code für diese Erweiterung befindet sich unter PostGIS in `extensions/address_standardizer` und ist zurzeit self-contained ("unabhängig").

Für eine Installationsanleitung siehe: Section 2.3.

11.1.1 Funktionsweise des Parsers

Der Parser arbeitet von rechts nach links und betrachtet zunächst die Makroelemente Postleitzahl, Staat/Provinz, Stadt. Anschließend werden die Mikroelemente untersucht, um festzustellen ob es sich um eine Husnummer, eine Kreuzung oder eine Wegmarkierung handelt. Zur Zeit schaut der Parser nicht auf die Landeskennzahl oder -namen, dies kann aber möglicherweise noch implementiert werden.

Country code Wird als US oder CA basiert angenommen: Postleitzahl als US oder Kanada, state/province als US oder Kanada, sonst US

Postcode/zipcode Diese werden über Perl-kompatible reguläre Ausdrücke erkannt. Die Regexs befinden sich in "parseaddress-api.c" und können bei Bedarf relativ leicht angepasst werden.

State/province Diese werden über Perl-kompatible reguläre Ausdrücke erkannt. Die Regexs befinden sich zurzeit in "parseaddress-api.c", könnten zukünftig aber zwecks leichterer Wartbarkeit in die "includes" verschoben werden.

11.1.2 Adressennormierer Datentypen

11.1.2.1 stdaddr

stdaddr — Ein zusammengesetzter Datentyp, der aus den Elementen einer Adresse besteht. Dies ist der zurückgegebene Datentyp der `standardize_address` Funktion.

Beschreibung

Ein zusammengesetzter Datentyp, der aus den Elementen einer Adresse besteht. Dies ist der Datentyp, der von standardize_address zurückgegeben wird. Einige Elementbeschreibungen wurden von PAGC Postal Attributes übernommen.

Die Token-Nummern geben die Referenznummer der Ausgabe in der rules table an.

This method needs address_standardizer extension.

building ist ein Text (Token-Nummer `0`): Verweist auf die Hausnummer oder Namen. Gebäude Identifikatoren und Typen nicht geparst. Bei den meisten Adressen üblicherweise leer.

house_num ist ein Text (Token-Nummer `1`): Die Hausnummer einer Straße. Beispiel *75* in `75 State Street`.

predir ist ein Text (Token-Nummer `2`): STREET NAME PRE-DIRECTIONAL, wie Nord, Süd, Ost, West etc.

qual ist ein Text (Token-Nummer `3`): STREET NAME PRE-MODIFIER Beispiel *OLD* in `3715 OLD HIGHWAY 99`.

pretype ist ein Text (Token-Nummer `4`): STREET PREFIX TYPE

name ist ein Text (Token-Nummer `5`): STREET NAME

suftype ist ein Text (Token-Nummer `6`): STREET POST TYPE z.B. St, Ave, Cir. Ein dem Straßennamen angehänter Straßentyp. Beispiel *STREET* in `75 State Street`.

sufdir ist ein Text (Token-Nummer `7`): STREET POST-DIRECTIONAL Eine Richtungsangabe, die dem Straßennamen folg. Beispiel *WEST* in `3715 TENTH AVENUE WEST`.

ruralroute ist ein Text (Token-Nummer `8`): RURAL ROUTE . Beispiel: *7* in `RR 7`.

extra ist ein Text: Zusätzliche Information, wie die Geschossnummer/Stockwerk.

city ist ein Text (Token-Nummer `10`): Beispiel Boston.

state ist ein Text (Token-Nummer `11`): Beispiel `MASSACHUSETTS`

country ist ein Text (Token-Nummer `12`): Beispiel `USA`

postcode ist ein Text POSTAL CODE (ZIP CODE) (Token-Nummer `13`): Beispiel `02109`

box ist ein Text POSTAL BOX NUMBER (Token-Nummer `14 und 15`): Beispiel `02109`

unit ist ein Text Wohnungs- oder Suite-Nummer (Token-Nummer `17`): Beispiel *3B* in `APT 3B`.

11.1.3 Adressennormierer Tabellen

11.1.3.1 rules table

rules table — Die Tabelle "rules" enthält die Regeln, nach denen die Token der Eingabesequenz der Adresse in eine standardisierte Ausgabesequenz abgebildet werden. Eine Regel besteht aus einem Satz Eingabetoken, gefolgt von -1 (Terminator), gefolgt von einem Satz Ausgabetoken, gefolgt von -1, gefolgt von einer Zahl zur Kennzeichnung des Regeltyps, gefolgt von der Rangordnung der Regel.

Beschreibung

Eine "rules" Tabelle muss mindestens die folgenden Spalten aufweisen, es können aber zusätzliche Spalten für den Eigenbedarf hinzugefügt werden.

id Der Primärschlüssel der Tabelle

rule Ein Textfeld, das die Regel festlegt. Details unter PAGC Address Standardizer Rule records.

Eine Regel besteht aus positiven ganzen Zahlen, den Eingabetoken, die durch ein -1 abgeschlossen werden, gefolgt von der gleichen Anzahl an positiven ganzen Zahlen, den Postattributen, die ebenfalls mit -1 abgeschlossen werden, gefolgt von einer ganzen Zahl, die den Regeltyp kennzeichnet, gefolgt von einer ganzen Zahl, welche die Rangordnung der Regel festlegt. Die Regeln werden von 0 (niedrigster Rang) bis 17 (höchster) gereiht.

So wird zum Beispiel durch die Regel `2 0 2 22 3 -1 5 5 6 7 3 -1 2 6` die Abfolge von Ausgabetoken *TYPE NUMBER TYPE DIRECT QUALIF* auf die Ausgabesequenz *STREET STREET SUFTYP SUFDIR QUALIF* abgebildet. Dies ist eine ARC_C Regel vom Rang 6.

Die Nummern der entsprechenden Ausgabe-Token sind unter stdaddr aufgeführt.

Eingabe-Token

Jede Regel beginnt mit einer Menge an Eingabetoken, gefolgt bei der Abschlussanweisung `-1`. Im Folgenden ein Auszug von gültigen Eingabetoken aus PAGC Input Tokens:

Formbasierte Eingabezeichen

AMPERS (13). Das kaufmännische Und (&) wid häufig zur Abkürzung des Wortes "und" verwendet.

DASH (9). Ein Satzzeichen.

DOUBLE (21). Eine Sequenz mit zwei Buchstaben. Wird oft als Identifikator verwendet.

FRACT (25). Brüche kommen manchmal bei Hausnummern oder Blocknummern vor.

MIXED (23). Eine alphanumerische Zeichenkette, die aus Buchstaben und Ziffern besteht. Wird als Identifikator verwendet.

NUMBER (0). Eine Folge von Ziffern.

ORD (15). Bezeichnungen wie "First" oder 1st. Wird häufig bei Straßennamen benutzt.

ORD (18). Ein einzelner Buchstabe.

WORD (1). Ein Wort ist eine Zeichenfolge beliebiger Länge. Ein einzelnes Zeichen kann sowohl ein SINGLE als auch ein WORD sein.

Funktionsbasierte Eingabezeichen

BOXH (14). Ein Text zur Kennzeichnung von Postfächern. Zum Beispiel *Box* oder *PO Box*.

BUILDH (19). Wörter zur Bezeichnung von Gebäuden und Gebäudekomplexen - üblicherweise als Präfix. Zum Beispiel: *Tower* in *Tower 7A*.

BUILDT (24). Wörter und Abkürzungen zur Bezeichnung von Gebäuden und Gebäudekomplexen - üblicherweise als Suffix. Zum Beispiel: *Shopping Centre*.

DIRECT (22). Text zur Richtungsangabe, zum Beispiel*North*.

MILE (20). Wörter zur Bezeichnung von Milepost Adressen.

ROAD (6). Wörter und Abkürzungen für die Bezeichnung von Autobahnen und Straßen. Zum Beispiel *Interstate* in *Interstate 5*.

RR (8). Wörter und Abkürzungen für Postwege im ländlichen Gebiet - "Rural Routes". *RR*.

TYPE (2). Begriffe und Abkürzungen für Straßentypen. Zum Beispiel: *ST* oder *AVE*.

UNITH (16). Begriffe und Abkürzungen für zusätzliche Adressangaben. Zum Beispiel *APT* oder *UNIT*.

Eingabezeichen für den Postleitzahltyp

QUINT (28). Eine 5-stellige Nummer. Gibt den Zip Code an

QUAD (29). Eine 4-stellige Nummer. Gibt den ZIP4 Code an.

PCH (27). Eine 3 Zeichen lange Abfolge von Buchstabe - Zahl - Buchstabe. Kennzeichnet eine FSA, die ersten 3 Zeichen des kanadischen Postleitzahl.

PCT (26). Eine 3 Zeichen lange Abfolge von Zahl -Buchstabe - Zahl. Kennzeichnet eine LDU, die letzten 3 Zeichen des kanadischen Postleitzahl.

Stoppwörter

Stoppwörter werden mit Wörtern kombiniert. In den Regeln wird eine Zeichenkette aus mehreren Wörtern und Stoppwörtern durch einen einzelnen WORD-Token dargestellt.

STOPWORD (7). Ein Wort mit geringer semantischer Bedeutung, das bei der Analyse weggelassen werden kann. Zum Beispiel: *THE*.

Ausgabe-Token

Nach dem ersten -1 (Abschlussanweisung) folgen die Ausgabetoken und deren Reihenfolge, gefolgt bei einer Abschlussanweisung `-1`. Die Nummern der entsprechenden Ausgabetoken sind unter stdaddr aufgeführt. Welche Token zulässig sind hängt von der Art der Regel ab. Die gültigen Ausgabetoken für die jeweiligen Regeln sind unter the section called “Regel Typen und Rang” aufgelistet.

Regel Typen und Rang

Den Schlussteil der Regel bildet der Regeltyp. Dieser wird, gefolgt von einem Rang für die Regel, durch eines der folgenden Wörter angegeben. Die Regeln sind von 0 (niedrigster Rang) bis 17 (höchster Rang) gereiht.

MACRO_C

(Token-Nummer = "**0**"). Die Klassenregeln um MACRO Klauseln, wie *PLACE STATE ZIP*, zu parsen.

MACRO_C Ausgabe-Token (ein Auszug von http://www.pagcgeo.org/docs/html/pagc-12.html#--r-typ--.

CITY (Token-Nummer "10"). Beispiel "Albanien"

STATE (Token-Nummer "11"). Beispiel "NY"

NATION (Token Nummer "12"). Dieses Attribut wird in den meisten Referenzdateien nicht verwendet. Beispiel "USA"

POSTAL (Token Nummer "13"). (SADS Elemente "ZIP CODE" , "PLUS 4"). Dieses Attribut wird für die Postleitzahlen-Codes der USA (ZIP-Code) und Kanada (Postal Code) verwendet.

MICRO_C

(Token Nummer = "**1**"). Die Regelklasse zum Parsen ganzer MICRO Klauseln (wie House, street, sufdir, predir, pretyp, suftype, qualif) (insbesondere ARC_C plus CIVIC_C). Diese Regeln werden bei der Aufbauphase nicht benutzt.

MICRO_C Ausgabe-Token (ein Auszug von http://www.pagcgeo.org/docs/html/pagc-12.html#--r-typ--.

HOUSE ist ein Text (Token-Nummer `1`): Die Hausnummer einer Straße. Beispiel *75* in `75 State Street`.

predir ist ein Text (Token-Nummer `2`): STREET NAME PRE-DIRECTIONAL, wie Nord, Süd, Ost, West etc.

qual ist ein Text (Token-Nummer 3): STREET NAME PRE-MODIFIER Beispiel *OLD* in 3715 OLD HIGHWAY 99.

pretype ist ein Text (Token-Nummer 4): STREET PREFIX TYPE

street ist ein Text (Token-Nummer 5): STREET NAME

suftype ist ein Text (Token-Nummer 6): STREET POST TYPE z.B. St, Ave, Cir. Ein dem Straßennamen angehänter Straßentyp. Beispiel *STREET* in 75 State Street.

sufdir ist ein Text (Token-Nummer 7): STREET POST-DIRECTIONAL Eine Richtungsangabe, die dem Straßennamen folg. Beispiel *WEST* in 3715 TENTH AVENUE WEST.

ARC_C

(Token Nummer = "**2**"). Die Regelklasse zum Parsen von MICRO Klauseln ausgenommen dem Attribut "HOUSE". Verwendet dieselben Ausgabetoken wie MICRO_C, abzüglich dem HOUSE Token.

CIVIC_C

(Token-Nummer = "**3**"). Die Klassenregeln zum parsen des HOUSE Attributs.

EXTRA_C

(token number = "**4**"). Die Regelklasse zum Parsen von zusätzlichen Attributen - Attribute die von der Geokodierung ausgeschlossen sind. Diese Regeln werden bei der Aufbauphase nicht benutzt.

EXTRA_C Ausgabe-Token (ein Auszug von http://www.pagcgeo.org/docs/html/pagc-12.html#--r-typ--.

BLDNG (Token Nummer 0): Ungeparste Gebäudeidentifikatoren und Gebäudetypen.

BOXH (Token-Nummer 14): Die **BOX** in BOX 3B

BOXT (Token-Nummer 15): **3B** in BOX 3B

RR (Token-Nummer 8): **RR** in RR 7

UNITH (Token-Nummer 16): **APT** in APT 3B

UNITT (Token-Nummer 17): **3B** in APT 3B

UNKNWN (Token-Nummer 9): Eine nicht näher klassifizierte Ausgabe.

11.1.3.2 lex table

lex table — Eine "lex" Tabelle wird verwendet, um eine alphanumerische Eingabe einzustufen und mit (a) Eingabe-Tokens (siehe the section called “Eingabe-Token”) und (b) normierten Darstellungen zu verbinden.

Beschreibung

Eine lex (abgekürzt für Lexikon) Tabelle wird verwendet um alphanumerische Eingaben zu gliedern, und die Eingabe mit the section called “Eingabe-Token” und (b) genormten Darstellungen zu verbinden. In diesen Tabellen finden Sie Dinge wie ONE abgebildet auf stdword: 1.

Eine "lex" Tabelle muss zumindest die folgenden Spalten aufweisen.

id Der Primärschlüssel der Tabelle

seq Integer: Definitionsnummer?

word text: das Eingabewort

stdword text: das normierte Ersatzwort

token Integer: die Art des Wortes. Wird nur in diesem Zusammenhang ersetzt. Siehe PAGC Tokens.

11.1.3.3 gaz table

gaz table — Eine "gaz" Tabelle wird verwendet, um Ortsnamen zu normieren und um diese mit (a) Eingabe-Token (siehe the section called “Eingabe-Token”) und (b) normierten Darstellungen zu verbinden.

Beschreibung

Eine "gaz" (Abkürzung für Gazeteer) Tabelle wird verwendet, um Ortsnamen zu normieren und um diese mit the section called “Eingabe-Token” und (b) normierten Darstellungen zu verbinden. Wenn Sie zum Beispiel in der USA sind, können Sie die Namen der Bundesstaaten und die zugehörigen Abkürzungen in diese Tabelle laden.

Eine "gaz" Tabelle muss zumindest die folgenden Spalten aufweisen, es können aber zusätzliche Spalten für den Eigenbedarf hinzugefügt werden.

id Der Primärschlüssel der Tabelle

seq Integer: Kennzahl? - Kennung die für diese Instanz des Wortes verwendet wird.

word text: das Eingabewort

stdword text: das normierte Ersatzwort

token Integer: die Art des Wortes. Wird nur in diesem Zusammenhang ersetzt. Siehe PAGC Tokens.

11.1.4 Adressennormierer Funktionen

11.1.4.1 debug_standardize_address

debug_standardize_address — Returns a json formatted text listing the parse tokens and standardizations

Synopsis

text **debug_standardize_address**(text lextab, text gaztab, text rultab, text micro, text macro=NULL);

Beschreibung

This is a function for debugging address standardizer rules and lex/gaz mappings. It returns a json formatted text that includes the matching rules, mapping of tokens, and best standardized address stdaddr form of an input address utilizing lex table table name, gaz table, and rules table table names and an address.

For single line addresses use just `micro`

For two line address A `micro` consisting of standard first line of postal address e.g. `house_num street`, and a macro consisting of standard postal second line of an address e.g `city, state postal_code country`.

Elements returned in the json document are

input_tokens For each word in the input address, returns the position of the word, token categorization of the word, and the standard word it is mapped to. Note that for some input words, you might get back multiple records because some inputs can be categorized as more than one thing.

rules The set of rules matching the input and the corresponding score for each. The first rule (highest scoring) is what is used for standardization

stdaddr The standardized address elements stdaddr that would be returned when running standardize_address

Availability: 3.4.0

This method needs address_standardizer extension.

Beispiele

Verwendung der address_standardizer_data_us Erweiterung

```
CREATE EXTENSION address_standardizer_data_us; -- only needs to be done once
```

Variant 1: Single line address and returning the input tokens

```
SELECT it->
>'pos' AS position, it->
>'word' AS word, it->
>'stdword' AS standardized_word,
            it->
>'token' AS token, it->
>'token-code' AS token_code
    FROM jsonb(
            debug_standardize_address('us_lex',
                'us_gaz', 'us_rules', 'One Devonshire Place, PH 301, Boston, MA 02109')
                 ) AS s, jsonb_array_elements(s->'input_tokens') AS it;
```

```
position |    word    | standardized_word | token  | token_code
---------+------------+-------------------+--------+------------
 0       | ONE        | 1                 | NUMBER | 0
 0       | ONE        | 1                 | WORD   | 1
 1       | DEVONSHIRE | DEVONSHIRE        | WORD   | 1
 2       | PLACE      | PLACE             | TYPE   | 2
 3       | PH         | PATH              | TYPE   | 2
 3       | PH         | PENTHOUSE         | UNITT  | 17
 4       | 301        | 301               | NUMBER | 0
(7 rows)
```

Variant 2: Multi line address and returning first rule input mappings and score

```
SELECT (s->'rules'->0->
>'score')::numeric AS score, it->
>'pos' AS position,
        it->
>'input-word' AS word, it->
>'input-token' AS input_token, it->
>'mapped-word' AS standardized_word,
            it->
>'output-token' AS output_token
    FROM jsonb(
            debug_standardize_address('us_lex',
                'us_gaz', 'us_rules', 'One Devonshire Place, PH 301', 'Boston, MA 02109')
                 ) AS s, jsonb_array_elements(s->'rules'->0->'rule_tokens') AS it;
```

```
score    | position |    word    | input_token | standardized_word | output_token
---------+----------+------------+-------------+-------------------+--------------
 0.876250 | 0        | ONE        | NUMBER      | 1                 | HOUSE
 0.876250 | 1        | DEVONSHIRE | WORD        | DEVONSHIRE        | STREET
 0.876250 | 2        | PLACE      | TYPE        | PLACE             | SUFTYP
 0.876250 | 3        | PH         | UNITT       | PENTHOUSE         | UNITT
 0.876250 | 4        | 301        | NUMBER      | 301               | UNITT
(5 rows)
```

Siehe auch

stdaddr, rules table, lex table, gaz table, Pagc_Normalize_Address

11.1.4.2 parse_address

parse_address — Nimmt eine 1-zeilige Adresse entgegen und zerlegt sie in die Einzelteile

Synopsis

record **parse_address**(text address);

Beschreibung

Nimmt eine Adresse entgegen und gibt einen Datensatz mit den folgenden Attributen zurück: *num*, *street*, *street2*, *address1*, *city*, *state*, *zip*, *zipplus* und *country*.

Verfügbarkeit: 2.2.0

This method needs address_standardizer extension.

Beispiele

Einzelne Adresse

```
SELECT num, street, city, zip, zipplus
        FROM parse_address('1 Devonshire Place, Boston, MA 02109-1234') AS a;
```

```
num |      street      |  city  |  zip  | zipplus
-----+------------------+--------+-------+---------
 1   | Devonshire Place | Boston | 02109 | 1234
```

Tabelle mit Adressen

```
-- basic table
CREATE TABLE places(addid serial PRIMARY KEY, address text);

INSERT INTO places(address)
VALUES ('529 Main Street, Boston MA, 02129'),
 ('77 Massachusetts Avenue, Cambridge, MA 02139'),
 ('25 Wizard of Oz, Walaford, KS 99912323'),
 ('26 Capen Street, Medford, MA'),
 ('124 Mount Auburn St, Cambridge, Massachusetts 02138'),
 ('950 Main Street, Worcester, MA 01610');

 -- parse the addresses
 -- if you want all fields you can use (a).*
SELECT addid, (a).num, (a).street, (a).city, (a).state, (a).zip, (a).zipplus
FROM (SELECT addid, parse_address(address) As a
 FROM places) AS p;
```

```
addid | num |        street        |   city    | state |  zip  | zipplus
-------+-----+----------------------+-----------+-------+-------+---------
     1 | 529 | Main Street          | Boston    | MA    | 02129 |
     2 | 77  | Massachusetts Avenue | Cambridge | MA    | 02139 |
     3 | 25  | Wizard of Oz         | Walaford  | KS    | 99912 | 323
     4 | 26  | Capen Street         | Medford   | MA    |       |
     5 | 124 | Mount Auburn St      | Cambridge | MA    | 02138 |
     6 | 950 | Main Street          | Worcester | MA    | 01610 |
(6 rows)
```

Siehe auch

11.1.4.3 standardize_address

standardize_address — Gibt eine gegebene Adresse in der Form "stdaddr" zurück. Verwendet die Tabellen "lex", "gaz" und "rule".

Synopsis

stdaddr **standardize_address**(text lextab, text gaztab, text rultab, text address);
stdaddr **standardize_address**(text lextab, text gaztab, text rultab, text micro, text macro);

Beschreibung

Gibt eine gegebene Adresse in der Form stdaddr zurück. Verwendet die Tabellennamen lex table, gaz table und rules table und eine Adresse.

Variante 1: Nimmt eine einzeilige Adresse entgegen.

Variante 2: Nimmt eine Adresse in 2 Teilen entgegen. Ein `micro` Teil, der aus der normierten ersten Zeile einer Postadresse besteht; z.B. `house_num street`. Ein "macro"-Teil, der aus der normierten zweiten Zeile einer Adresse besteht; z.B. `city, state postal_code country`.

Verfügbarkeit: 2.2.0

This method needs address_standardizer extension.

Beispiele

Verwendung der address_standardizer_data_us Erweiterung

```
CREATE EXTENSION address_standardizer_data_us; -- only needs to be done once
```

Variante 1: Einzeilige Adresse. Dies funktioniert nicht gut mit Adressen außerhalb der US

```
SELECT house_num, name, suftype, city, country, state, unit  FROM standardize_address(' ↩
    us_lex',
                            'us_gaz', 'us_rules', 'One Devonshire Place, PH 301, Boston, MA  ↩
                               02109');
```

```
house_num |    name    | suftype |  city   | country |     state     |      unit
----------+------------+---------+---------+---------+---------------+-----------------
1         | DEVONSHIRE | PLACE   | BOSTON | USA     | MASSACHUSETTS | # PENTHOUSE 301
```

Verwendung der Tabellen, die mit dem Tiger Geokodierer paketiert sind. Dieses Beispiel funktioniert nur, wenn Sie `postgis_tiger_` installiert haben.

```
SELECT *  FROM standardize_address('tiger.pagc_lex',
         'tiger.pagc_gaz', 'tiger.pagc_rules', 'One Devonshire Place, PH 301, Boston, MA  ↩
            02109-1234');
```

Die Ausgabe über einen Dump mit der Erweiterung "hstore" ist leichter lesbar. Die Erweiterung hstore muss mittels "CREATE EXTENSION hstore;" installiert sein.

```
SELECT (each(hstore(p))).*
 FROM standardize_address('tiger.pagc_lex', 'tiger.pagc_gaz',
   'tiger.pagc_rules', 'One Devonshire Place, PH 301, Boston, MA 02109') As p;
```

```
key     |      value
------------+-----------------
 box        |
 city       | BOSTON
 name       | DEVONSHIRE
 qual       |
 unit       | # PENTHOUSE 301
 extra      |
 state      | MA
 predir     |
 sufdir     |
 country    | USA
 pretype    |
 suftype    | PL
 building   |
 postcode   | 02109
 house_num  | 1
 ruralroute |
(16 rows)
```

Variante 2: Adresse aus zwei Teilen.

```
SELECT (each(hstore(p))).*
 FROM standardize_address('tiger.pagc_lex', 'tiger.pagc_gaz',
   'tiger.pagc_rules', 'One Devonshire Place, PH 301', 'Boston, MA 02109, US') As p;
```

```
key     |      value
------------+-----------------
 box        |
 city       | BOSTON
 name       | DEVONSHIRE
 qual       |
 unit       | # PENTHOUSE 301
 extra      |
 state      | MA
 predir     |
 sufdir     |
 country    | USA
 pretype    |
 suftype    | PL
 building   |
 postcode   | 02109
 house_num  | 1
 ruralroute |
(16 rows)
```

Siehe auch

stdaddr, rules table, lex table, gaz table, Pagc_Normalize_Address

11.2 Tiger Geokoder

Es existieren eine Reihe weiterer Open Source Geokodierer füf PostGIS, welche im Gegensatz zu dem Tiger Geokodierer den Vorteil haben, dass sie mehrere Länder unterstützen

- Nominatim verwendet Daten von OpenStreetMap, die mittels Gazetteer formatiert werden. Es benötigt osm2pgsql zum Laden der Daten, PostgreSQL 8.4+ und PostGIS 1.5+ um zu funktionieren. Es ist als Webinterface paketiert und wird vermutlich als

Webservice aufgerufen. So wie der Tiger Geokodierer besteht es aus einem Geokodierer und einer inversen Komponente des Geokodierers. Aus der Dokumentation ist nicht ersichtlich, ob es wie der Tiger Geokodierer auch eine reine SQL Schnittstelle aufweist, oder ob ein größerer Anteil der Logik in das Webinterface implementiert wurde.

- GIS Graphy nützt ebenfalls PostGIS und arbeitet so wie Nominatim ebenfalls mit Daten von OpenStreetMap (OSM). Es beinhaltet einen Loader, um OSM-Daten zu importieren. Ähnlich wie Nominatim kann es auch für die Geokodierung außerhalb der USA verwendet werden. So wie Nominatim läuft es als Webservice und benötigt Java 1.5, Servlet Apps und Solr. GisGraphy kann plattformübergreifend genutzt werden und hat ebenfalls einen invertierten Geokodierer zusammen mit anderen geschickten Funktionen.

11.2.1 Drop_Indexes_Generate_Script

Drop_Indexes_Generate_Script — Erzeugt ein Skript, welches alle Indizes aus dem Datenbankschema "Tiger" oder aus einem vom Anwender angegebenen Schema löscht, wenn die Indizes nicht auf den Primärschlüssel gelegt und nicht "unique" sind. Wenn kein Schema angegeben ist wird standardmäßig auf das `tiger_data` Schema zugegriffen.

Synopsis

text **Drop_Indexes_Generate_Script**(text param_schema=tiger_data);

Beschreibung

Erzeugt ein Skript, welches alle Indizes aus dem Datenbankschema "Tiger" oder aus einem vom Anwender angegebenen Schema löscht, wenn die Indizes nicht auf den Primärschlüssel gelegt und nicht "unique" sind. Wenn kein Schema angegeben ist wird standardmäßig auf das `tiger_data` Schema zugegriffen.

Dies kann verwendet werden, damit sich die Indizes nicht aufblähen und dadurch den Anfrageoptimierer irritieren oder unnötigen Speicherplatz belegen. Sie können das Skript in Verbindung mit Install_Missing_Indexes verwenden um nur jene Indizes zu erstellen die der Gekodierer benötigt.

Verfügbarkeit: 2.0.0

Beispiele

```
SELECT drop_indexes_generate_script() As actionsql;
actionsql
-----------------------------------------------------
DROP INDEX tiger.idx_tiger_countysub_lookup_lower_name;
DROP INDEX tiger.idx_tiger_edges_countyfp;
DROP INDEX tiger.idx_tiger_faces_countyfp;
DROP INDEX tiger.tiger_place_the_geom_gist;
DROP INDEX tiger.tiger_edges_the_geom_gist;
DROP INDEX tiger.tiger_state_the_geom_gist;
DROP INDEX tiger.idx_tiger_addr_least_address;
DROP INDEX tiger.idx_tiger_addr_tlid;
DROP INDEX tiger.idx_tiger_addr_zip;
DROP INDEX tiger.idx_tiger_county_countyfp;
DROP INDEX tiger.idx_tiger_county_lookup_lower_name;
DROP INDEX tiger.idx_tiger_county_lookup_snd_name;
DROP INDEX tiger.idx_tiger_county_lower_name;
DROP INDEX tiger.idx_tiger_county_snd_name;
DROP INDEX tiger.idx_tiger_county_the_geom_gist;
DROP INDEX tiger.idx_tiger_countysub_lookup_snd_name;
DROP INDEX tiger.idx_tiger_cousub_countyfp;
DROP INDEX tiger.idx_tiger_cousub_cousubfp;
DROP INDEX tiger.idx_tiger_cousub_lower_name;
DROP INDEX tiger.idx_tiger_cousub_snd_name;
```

```
DROP INDEX tiger.idx_tiger_cousub_the_geom_gist;
DROP INDEX tiger_data.idx_tiger_data_ma_addr_least_address;
DROP INDEX tiger_data.idx_tiger_data_ma_addr_tlid;
DROP INDEX tiger_data.idx_tiger_data_ma_addr_zip;
DROP INDEX tiger_data.idx_tiger_data_ma_county_countyfp;
DROP INDEX tiger_data.idx_tiger_data_ma_county_lookup_lower_name;
DROP INDEX tiger_data.idx_tiger_data_ma_county_lookup_snd_name;
DROP INDEX tiger_data.idx_tiger_data_ma_county_lower_name;
DROP INDEX tiger_data.idx_tiger_data_ma_county_snd_name;
:
:
```

Siehe auch

Install_Missing_Indexes, Missing_Indexes_Generate_Script

11.2.2 Drop_Nation_Tables_Generate_Script

Drop_Nation_Tables_Generate_Script — Erzeugt ein Skript, welches alle Tabellen in dem angegebenen Schema löscht, die mit `county_all`, `state_all` oder dem Ländercode gefolgt von `county` oder `state` beginnen.

Synopsis

text **Drop_Nation_Tables_Generate_Script**(text param_schema=tiger_data);

Beschreibung

Erzeugt ein Skript, welches alle Tabellen in dem angegebenen Schema löscht, die mit `county_all`, `state_all` oder dem Ländercode gefolgt von `county` oder `state` beginnen. Dies ist dann notwendig, wenn Sie von `tiger_2010` auf `tiger_2011` Daten upgraden.

Verfügbarkeit: 2.1.0

Beispiele

```
SELECT drop_nation_tables_generate_script();
DROP TABLE tiger_data.county_all;
DROP TABLE tiger_data.county_all_lookup;
DROP TABLE tiger_data.state_all;
DROP TABLE tiger_data.ma_county;
DROP TABLE tiger_data.ma_state;
```

Siehe auch

Loader_Generate_Nation_Script

11.2.3 Drop_State_Tables_Generate_Script

Drop_State_Tables_Generate_Script — Erzeugt ein Skript, dass alle Tabellen in dem angegebenen Schema löscht, die als Präfix einen Ländercode haben. Wenn kein Schema angegeben ist wird standardmäßig auf das `tiger_data` Schema zugegriffen.

Synopsis

text **Drop_State_Tables_Generate_Script**(text param_state, text param_schema=tiger_data);

Beschreibung

Erzeugt ein Skript, dass alle Tabellen in dem angegebenen Schema löscht, die als Präfix einen Ländercode haben. Wenn kein Schema angegeben ist wird standardmäßig auf das `tiger_data` Schema zugegriffen. Wenn beim Import etwas schiefgegangen ist, können mit dieser Funktion die Tabellen eines Staates unmittelbar vor dem erneuten Import, gelöscht werden.

Verfügbarkeit: 2.0.0

Beispiele

```
SELECT drop_state_tables_generate_script('PA');
DROP TABLE tiger_data.pa_addr;
DROP TABLE tiger_data.pa_county;
DROP TABLE tiger_data.pa_county_lookup;
DROP TABLE tiger_data.pa_cousub;
DROP TABLE tiger_data.pa_edges;
DROP TABLE tiger_data.pa_faces;
DROP TABLE tiger_data.pa_featnames;
DROP TABLE tiger_data.pa_place;
DROP TABLE tiger_data.pa_state;
DROP TABLE tiger_data.pa_zip_lookup_base;
DROP TABLE tiger_data.pa_zip_state;
DROP TABLE tiger_data.pa_zip_state_loc;
```

Siehe auch

Loader_Generate_Script

11.2.4 Geocode

Geocode — Nimmt eine Adresse als Zeichenkette (oder eine bereits standardisierte Adresse) entgegen und gibt die möglichen Punktlagen zurück. Die Ausgabe beinhaltet eine Punktgeometrie in NAD 83 Länge/Breite, eine standardisierte Adresse und eine Rangfolge (Rating) für jede Punktlage. Umso niedriger die Rangfolge ist, um so wahrscheinlicher ist die Übereinstimmung. Die Ergebnisse werden mit aufsteigender Rangfolge sortiert - dar niedrigste Rang zuerst. Optional kann die maximale Anzahl der Ergebnisse angegeben werden (Standardeinstellung ist 10) und der Bereich mit restrict_region beschränkt werden (Standardeinstellung ist NULL)

Synopsis

setof record **geocode**(varchar address, integer max_results=10, geometry restrict_region=NULL, norm_addy OUT addy, geometry OUT geomout, integer OUT rating);
setof record **geocode**(norm_addy in_addy, integer max_results=10, geometry restrict_region=NULL, norm_addy OUT addy, geometry OUT geomout, integer OUT rating);

Beschreibung

Nimmt eine Adresse als Zeichenkette (oder eine bereits standardisierte Adresse) entgegen und gibt die möglichen Punktlagen zurück. Die Ausgabe beinhaltet eine Punktgeometrie in NAD 83 Länge/Breite, eine standardisierte Adresse `normalized_address` (addy) und eine Rangfolge (Rating) für jede Punktlage. Umso niedriger die Rangfolge ist, um so wahrscheinlicher ist die Übereinstimmung. Die Ergebnisse werden nach dem Rating aufsteigend sortiert - das niedrigste Rating zuerst. Verwendet Tiger Daten

(Kanten, Maschen, Adressen), Fuzzy String Matching (soundex, levenshtein) von PostgreSQL und PostGIS Funktionen zur Interpolation entlang von Linien, um die Adressen entlang der Kanten von TIGER zu interpolieren. Umso höher das Rating, umso unwahrscheinlicher ist es, dass die Geokodierung richtig liegt. Der geokodierte Punkt wird dort, wo sich die Adresse befindet, standardmäßig um 10 Meter von der Mittellinie auf die Seite (L/R) versetzt. Optional kann die maximale Anzahl der Ergebnisse angegeben werden (Standardeinstellung ist 10) und der Bereich mit restrict_region beschränkt werden (Standardeinstellung ist NULL)

Erweiterung: 2.0.0 Unterstützung von strukturierten Daten von TIGER 2010. Weiters wurde die Logik überarbeitet, um die Rechengeschwindigkeit und die Genauigkeit der Geokodierung zu erhöhen, und den Versatz von der Mittellinie auf die Straßenseite zu ermöglichen. Der neue Parameter `max_results` kann verwendet werden, um die Anzahl der besten Ergebnisse zu beschränken oder um nur das beste Ergebnis zu erhalten.

Beispiele: Grundlagen

Die Zeitangaben für die unteren Beispiele beziehen sich auf einen 3.0 GHZ Prozessor mit Windows 7, 2GB RAM, PostgreSQL 9.1rc1/PostGIS 2.0 und den geladenen TIGER-Daten der Staaten MA, MN, CA und RI.

Genaue Übereinstimmungen haben eine kürzere Rechenzeit (61ms)

```
SELECT g.rating, ST_X(g.geomout) As lon, ST_Y(g.geomout) As lat,
    (addy).address As stno, (addy).streetname As street,
    (addy).streettypeabbrev As styp, (addy).location As city, (addy).stateabbrev As st,( ↩
        addy).zip
    FROM geocode('75 State Street, Boston MA 02109', 1) As g;
 rating |        lon        |      lat       | stno | street | styp |  city  | st |  zip
--------+-------------------+----------------+------+--------+------+--------+----+-------
      0 | -71.0557505845646 | 42.35897920691 |   75 | State  | St   | Boston | MA | 02109
```

Sogar wenn der Zip-Code nicht übergeben wird, kann ihn der Geokodierer erraten (dauerte ca. 122-150ms)

```
SELECT g.rating, ST_AsText(ST_SnapToGrid(g.geomout,0.00001)) As wktlonlat,
    (addy).address As stno, (addy).streetname As street,
    (addy).streettypeabbrev As styp, (addy).location As city, (addy).stateabbrev As st,( ↩
        addy).zip
    FROM geocode('226 Hanover Street, Boston, MA',1) As g;
 rating |         wktlonlat          | stno | street  | styp |  city  | st |  zip
--------+----------------------------+------+---------+------+--------+----+-------
      1 | POINT(-71.05528 42.36316) |  226 | Hanover | St   | Boston | MA | 02113
```

Kann Rechtschreibfehler behandeln und liefert mehre mögliche Lösungen mit Einstufungen, hat allerdings eine längere Laufzeit (500ms).

```
SELECT g.rating, ST_AsText(ST_SnapToGrid(g.geomout,0.00001)) As wktlonlat,
    (addy).address As stno, (addy).streetname As street,
    (addy).streettypeabbrev As styp, (addy).location As city, (addy).stateabbrev As st,( ↩
        addy).zip
    FROM geocode('31 - 37 Stewart Street, Boston, MA 02116',1) As g;
 rating |         wktlonlat          | stno | street | styp |  city  | st |  zip
--------+----------------------------+------+--------+------+--------+----+-------
     70 | POINT(-71.06466 42.35114) |   31 | Stuart | St   | Boston | MA | 02116
```

Verwendet um die Adresskodierung in enier Stapelverarbeitung auszuführen. Am einfachsten ist es `max_results=1` zu setzen. Berechnet nur die Fälle, die noch nicht geokodiert wurden (keine Einstufung haben).

```
CREATE TABLE addresses_to_geocode(addid serial PRIMARY KEY, address text,
        lon numeric, lat numeric, new_address text, rating integer);

INSERT INTO addresses_to_geocode(address)
VALUES ('529 Main Street, Boston MA, 02129'),
 ('77 Massachusetts Avenue, Cambridge, MA 02139'),
 ('25 Wizard of Oz, Walaford, KS 99912323'),
```

```
 ('26 Capen Street, Medford, MA'),
 ('124 Mount Auburn St, Cambridge, Massachusetts 02138'),
 ('950 Main Street, Worcester, MA 01610');

-- only update the first 3 addresses (323-704 ms -  there are caching and shared memory  ↩
    effects so first geocode you do is always slower) --
-- for large numbers of addresses you don't want to update all at once
-- since the whole geocode must commit at once
-- For this example we rejoin with LEFT JOIN
-- and set to rating to -1 rating if no match
-- to ensure we don't regeocode a bad address
UPDATE addresses_to_geocode
  SET  (rating, new_address, lon, lat)
    = ( COALESCE(g.rating,-1), pprint_addy(g.addy),
       ST_X(g.geomout)::numeric(8,5), ST_Y(g.geomout)::numeric(8,5) )
FROM (SELECT addid, address
    FROM addresses_to_geocode
    WHERE rating IS NULL ORDER BY addid LIMIT 3) As a
    LEFT JOIN LATERAL geocode(a.address,1) As g ON true
WHERE a.addid = addresses_to_geocode.addid;

result
-----
Query returned successfully: 3 rows affected, 480 ms execution time.

SELECT * FROM addresses_to_geocode WHERE rating is not null;

 addid |                   address                    |    lon    |   lat    |  ↩
                  new_address                | rating
-------+----------------------------------------------+-----------+----------+-----------------------

     1 | 529 Main Street, Boston MA, 02129            | -71.07177 | 42.38357 | 529 Main St, ↩
          Boston, MA 02129             |      0
     2 | 77 Massachusetts Avenue, Cambridge, MA 02139 | -71.09396 | 42.35961 | 77  ↩
         Massachusetts Ave, Cambridge, MA 02139 |      0
     3 | 25 Wizard of Oz, Walaford, KS 99912323       | -97.92913 | 38.12717 | Willowbrook, ↩
          KS 67502                     |    108
(3 rows)
```

Beispiele: Verwendung eines Geometrie-Filters

```
SELECT g.rating, ST_AsText(ST_SnapToGrid(g.geomout,0.00001)) As wktlonlat,
    (addy).address As stno, (addy).streetname As street,
    (addy).streettypeabbrev As styp,
    (addy).location As city, (addy).stateabbrev As st,(addy).zip
  FROM geocode('100 Federal Street, MA',
        3,
        (SELECT ST_Union(the_geom)
            FROM place WHERE statefp = '25' AND name = 'Lynn')::geometry
        ) As g;

 rating |         wktlonlat          | stno | street  | styp | city | st |  zip
--------+----------------------------+------+---------+------+------+----+-------
      7 | POINT(-70.96796 42.4659)   |  100 | Federal | St   | Lynn | MA | 01905
     16 | POINT(-70.96786 42.46853)  | NULL | Federal | St   | Lynn | MA | 01905
(2 rows)

Time: 622.939 ms
```

Siehe auch

Normalize_Address, Pprint_Addy, ST_AsText, ST_SnapToGrid, ST_X, ST_Y

11.2.5 Geocode_Intersection

Geocode_Intersection — Nimmt 2 sich kreuzende Straßen, einen Bundesstaat, eine Stadt und einen ZIP-Code entgegen und gibt die möglichen Punktlagen an der ersten Querstraße an der Kreuzung zurück. Die Ausgabe beinhaltet auch die Geometrie "geomout" in NAD 83 Länge/Breite, eine standardisierte Adresse `normalized_address` (addy) für jede Punktage, sowie die Rangfolge. Umso niedriger die Rangfolge ist, um so wahrscheinlicher ist die Übereinstimmung. Die Ergebnisse werden mit aufsteigender Rangfolge sortiert - dar niedrigste Rang zuerst. Optional kann die maximale Anzahl der Ergebnisse angegeben werden (Standardeinstellung ist 10). Verwendet TIGER Daten (Kanten, Maschen, Adressen) und Fuzzy String Matching (soundex, levenshtein) von PostgreSQL.

Synopsis

setof record **geocode_intersection**(text roadway1, text roadway2, text in_state, text in_city, text in_zip, integer max_results=10, norm_addy OUT addy, geometry OUT geomout, integer OUT rating);

Beschreibung

Nimmt 2 sich kreuzende Straßen, einen Bundesstaat, eine Stadt und einen ZIP-Code entgegen und gibt die möglichen Punktlagen an der ersten Querstraße bei der Kreuzung zurück. Die Ausgabe beinhaltet auch eine Punktgeometrie in NAD 83 Länge/Breite, eine standardisierte Adresse für jede Punktage, sowie die Rangfolge. Umso niedriger die Rangfolge ist, um so wahrscheinlicher ist die Übereinstimmung. Die Ergebnisse werden mit aufsteigender Rangfolge sortiert - dar niedrigste Rang zuerst. Optional kann die maximale Anzahl der Ergebnisse angegeben werden (Standardeinstellung ist 10). Gibt für jede Punktlage die standardisierte Adresse `normalized_address` (addy), die Punktgeometrie "geomout" in NAD 83 Länge/Breite und ein Rating zurück. Verwendet TIGER Daten (Kanten, Maschen, Adressen) und Fuzzy String Matching (soundex, levenshtein) von PostgreSQL.

Verfügbarkeit: 2.0.0

Beispiele: Grundlagen

Die Zeitangaben für die unteren Beispiele beziehen sich auf einen 3.0 GHZ Prozessor mit Windows 7, 2GB RAM, PostgreSQL 9.0/PostGIS 1.5 und den geladenen TIGER-Daten des Staates MA. Zurzeit ein bißchen langsam (3000ms)

Testlauf auf Windows 2003 64-bit 8GB mit PostGIS 2.0, PostgreSQL 64-bit und geladenen TIGER-Daten von 2011 -- (41ms)

```
SELECT pprint_addy(addy), st_astext(geomout),rating
            FROM geocode_intersection( 'Haverford St','Germania St', 'MA', 'Boston',  ↩
                '02130',1);
           pprint_addy            |         st_astext          | rating
----------------------------------+----------------------------+--------
98 Haverford St, Boston, MA 02130 | POINT(-71.101375 42.31376) |      0
```

Sogar wenn der Zip-Code nicht angegeben ist, kann der Geokodierer diesen erraten (benötigte 3500ms auf einem Windows 7 Rechner, 741 ms auf Windows 2003 64-bit)

```
SELECT pprint_addy(addy), st_astext(geomout),rating
                FROM geocode_intersection('Weld', 'School', 'MA', 'Boston');
          pprint_addy          |        st_astext         | rating
-------------------------------+--------------------------+--------
 98 Weld Ave, Boston, MA 02119 | POINT(-71.099 42.314234) |      3
 99 Weld Ave, Boston, MA 02119 | POINT(-71.099 42.314234) |      3
```

Siehe auch

Geocode, Pprint_Addy, ST_AsText

11.2.6 Get_Geocode_Setting

Get_Geocode_Setting — Gibt die in der Tabelle "tiger.geocode_settings" gespeicherten Einstellungen zurück.

Synopsis

text **Get_Geocode_Setting**(text setting_name);

Beschreibung

Gibt die in der Tabelle "tiger.geocode_settings" gespeicherten Einstellungen zurück. Die Einstellungen erlauben auf "debugging" der Funktionen umzuschalten. Für später ist geplant auch die Ratings über die Einstellungen zu kontrollieren. Die aktuellen Einstellungen sind wie folgt:

```
name              | setting |  unit   | category  |  ↩
                                                                  short_desc
--------------------------------+---------+---------+-----------+-----------------------------------

 debug_geocode_address          | false   | boolean | debug     | outputs debug information ↩
      in notice log such as queries when geocode_address is called if true
 debug_geocode_intersection     | false   | boolean | debug     | outputs debug information ↩
      in notice log such as queries when geocode_intersection is called if true
 debug_normalize_address        | false   | boolean | debug     | outputs debug information ↩
      in notice log such as queries and intermediate expressions when normalize_address is  ↩
     called if true
 debug_reverse_geocode          | false   | boolean | debug     | if true, outputs debug  ↩
     information in notice log such as queries and intermediate expressions when  ↩
     reverse_geocode
 reverse_geocode_numbered_roads | 0       | integer | rating    | For state and county  ↩
     highways, 0 - no preference in name,
                                                                   1 - prefer the numbered  ↩
                                                                       highway name, 2 -  ↩
                                                                       prefer local state/ ↩
                                                                       county name
 use_pagc_address_parser        | false   | boolean | normalize | If set to true, will try  ↩
     to use the address_standardizer extension (via pagc_normalize_address)
                                                                  instead of tiger  ↩
                                                                     normalize_address built ↩
                                                                       one
```

Änderung: 2.2.0 : die Standardeinstellungen befinnden sich nun in der Tabelle "geocode_settings_default". Die vom Anwender angepassten Einstellungen - und nur diese - befinden sich in der Tabelle "geocode_settings".

Verfügbarkeit: 2.1.0

Das Beispiel gibt die "debugging" Einstellungen aus

```
SELECT get_geocode_setting('debug_geocode_address) As result;
result
---------
false
```

Siehe auch

Set_Geocode_Setting

11.2.7 Get_Tract

Get_Tract — Gibt für die Lage einer Geometrie die Census Area oder ein Feld der tract-Tabelle zurück. Standardmäßig wird die Kurzbezeichnung der Census Area ausgegeben.

Synopsis

text **get_tract**(geometry loc_geom, text output_field=name);

Beschreibung

Für eine gegebene Geometrie wird der Zählsprengel zurückgeben, in dem sich die Geometrie befindet. Wenn das Koordinatenreferenzsystem unbestimmt ist, dann wird NAD 83 in Länge und Breite angenommen.

Note
Diese Funktion verwendet den Census `tract`, welcher standardmäßig nicht geladen wird. Wenn Sie bereits die Tabelle mit den Bundesstaaten geladen haben, können Sie mit dem Skript Loader_Generate_Census_Script sowohl "tract" als auch "bg" und "tabblock" laden.
Wenn Sie die Daten der Bundesstaaten noch nicht geladen haben, können Sie diese zusätzlichen Tabellen wie folgt laden

```
UPDATE tiger.loader_lookuptables SET load = true WHERE load = false AND lookup_name ↩
    IN('tract', 'bg', 'tabblock');
```

dann werden diese in dem Loader_Generate_Script mit einbezogen.

Verfügbarkeit: 2.0.0

Beispiele: Grundlagen

```
SELECT get_tract(ST_Point(-71.101375, 42.31376) ) As tract_name;
tract_name
---------
1203.01
```

```
--this one returns the tiger geoid
SELECT get_tract(ST_Point(-71.101375, 42.31376), 'tract_id' ) As tract_id;
tract_id
---------
25025120301
```

Siehe auch

Geocode >

11.2.8 Install_Missing_Indexes

Install_Missing_Indexes — Findet alle Tabellen mit Schlüsselspalten, die für JOINs und Filterbedingungen vom Geokodierer verwendet werden und keinen Index aufweisen; die fehlenden Indizes werden hinzugefügt.

Synopsis

boolean **Install_Missing_Indexes**();

Beschreibung

Findet alle Tabellen in den Schemata `tiger` und `tiger_data`, bei denen die Schlüsselspalten, die für Joins und Filter vom Geokodierer verwendet werden keine Indizes aufweisen. Weiters wird ein SQL-DDL Skript ausgegeben, das die Indizes für diese Tabellen festlegt und anschließend ausgeführt wird. Dabei handelt es sich um eine Hilfsfunktion, die Abfragen schneller macht, indem die benötigten Indizes, die während des Imports gefehlt haben, neu hinzufügt. Diese Funktion ist verwandt mit Missing_Indexes_Generate_Script, wobei zusätzlich zur Erstellung des "CREATE INDEX"-Skripts dieses auch ausgeführt wird. Es wird als Teil des Upgrade-Skripts `update_geocode.sql` aufgerufen.

Verfügbarkeit: 2.0.0

Beispiele

```
SELECT install_missing_indexes();
         install_missing_indexes
-------------------------
 t
```

Siehe auch

Loader_Generate_Script, Missing_Indexes_Generate_Script

11.2.9 Loader_Generate_Census_Script

Loader_Generate_Census_Script — Erzeugt für gegebene Plattform und Bundesstaaten ein Shellskript, das die TIGER Datentabellen "tract", "bg" und "tabblocks" herunterlädt, bereitstellt und in das Schema `tiger_data` importiert. Jedes Bundesstaat-Skript wird in einem eigenen Datensatz ausgegeben.

Synopsis

setof text **loader_generate_census_script**(text[] param_states, text os);

Beschreibung

Erzeugt für gegebene Plattform und Bundesstaaten ein Shellskript, das die TIGER Datentabellen Census Area `tract`, block groups `bg` und `tabblocks` herunterlädt, bereitstellt und in das Schema `tiger_data` importiert. Jedes Bundesstaat-Skript wird in einem eigenen Datensatz ausgegeben.

Zum Herunterladen wird auf Linux unzip (auf Windows standardmäßig 7-zip) und wget verwendet. Es verwendet Section 4.7.2 zum Laden der Daten. Die kleinste Einheit, die bearbeitet wird ist ein ganzer Bundesstaat. Es werden nur die Dateien in den Ordnern "staging" und "temp" bearbeitet.

Verwendet die folgenden Kontrolltabellen, um den Verarbeitungsprozess und die verschiedenen Variationen der Betriebssysteme in Bezug auf den Shellsyntax zu überprüfen.

1. `loader_variables` behält den Überblick über verschiedenen Variablen, wie Census Site, Jahr, Daten- und "staging"-Schemata
2. `loader_platform` Profile von verschiedenen Plattformen und Speicherplätze der ausführbaren Programme. Beinhaltet Windows und Linux. Weitere können hinzugefügt werden.

3. `loader_lookuptables` jeder Datensatz definiert einen bestimmten Tabellentyp (state, county), um Datensätze zu bearbeiten und zu importieren. Legt die Schritte fest, die notwendig sind, um Daten zu importieren, bereitzustellen und hinzuzufügen, und um Spalten, Indizes und Constraints zu löschen. Jede Tabelle wird mit einem Präfix des Bundesstaates versehen und erbt von einer Tabelle in dem Schema TIGER. z.B. wird die Tabelle `tiger_data.ma_faces` erstellt, welche von `tiger.faces` erbt

Verfügbarkeit: 2.0.0

Note
Loader_Generate_Script beinhaltet diese Logik; wenn Sie aber den TIGER Geokodierer vor PostGIS 2.0.0 alpha5 installiert haben, müssen Sie dies für bereits importierte Bundesstaaten ausführen, um die zusätzlichen Tabellen zu erhalten.

Beispiele

Erzeugt ein Skript um Daten für die ausgewählten Länder im Windows Shell Script Format zu laden.

```
SELECT loader_generate_census_script(ARRAY['MA'], 'windows');
-- result --
set STATEDIR="\gisdata\www2.census.gov\geo\pvs\tiger2010st\25_Massachusetts"
set TMPDIR=\gisdata\temp\
set UNZIPTOOL="C:\Program Files\7-Zip\7z.exe"
set WGETTOOL="C:\wget\wget.exe"
set PGBIN=C:\projects\pg\pg91win\bin\
set PGPORT=5432
set PGHOST=localhost
set PGUSER=postgres
set PGPASSWORD=yourpasswordhere
set PGDATABASE=tiger_postgis20
set PSQL="%PGBIN%psql"
set SHP2PGSQL="%PGBIN%shp2pgsql"
cd \gisdata

%WGETTOOL% http://www2.census.gov/geo/pvs/tiger2010st/25_Massachusetts/25/ --no-parent -- ↩
    relative --accept=*bg10.zip,*tract10.zip,*tabblock10.zip --mirror --reject=html
del %TMPDIR%\*.* /Q
%PSQL% -c "DROP SCHEMA tiger_staging CASCADE;"
%PSQL% -c "CREATE SCHEMA tiger_staging;"
cd %STATEDIR%
for /r %%z in (*.zip) do %UNZIPTOOL% e %%z  -o%TMPDIR%
cd %TMPDIR%
%PSQL% -c "CREATE TABLE tiger_data.MA_tract(CONSTRAINT pk_MA_tract PRIMARY KEY (tract_id) ) ↩
     INHERITS(tiger.tract); "
%SHP2PGSQL% -c -s 4269 -g the_geom   -W "latin1" tl_2010_25_tract10.dbf tiger_staging. ↩
    ma_tract10 | %PSQL%
%PSQL% -c "ALTER TABLE tiger_staging.MA_tract10 RENAME geoid10 TO tract_id;  SELECT  ↩
    loader_load_staged_data(lower('MA_tract10'), lower('MA_tract')); "
%PSQL% -c "CREATE INDEX tiger_data_MA_tract_the_geom_gist ON tiger_data.MA_tract USING gist ↩
    (the_geom);"
%PSQL% -c "VACUUM ANALYZE tiger_data.MA_tract;"
%PSQL% -c "ALTER TABLE tiger_data.MA_tract ADD CONSTRAINT chk_statefp CHECK (statefp =  ↩
    '25');"
:
```

Erzeugt ein Shell-Skript

```
STATEDIR="/gisdata/www2.census.gov/geo/pvs/tiger2010st/25_Massachusetts"
TMPDIR="/gisdata/temp/"
UNZIPTOOL=unzip
```

```
WGETTOOL="/usr/bin/wget"
export PGBIN=/usr/pgsql-9.0/bin
export PGPORT=5432
export PGHOST=localhost
export PGUSER=postgres
export PGPASSWORD=yourpasswordhere
export PGDATABASE=geocoder
PSQL=${PGBIN}/psql
SHP2PGSQL=${PGBIN}/shp2pgsql
cd /gisdata

wget http://www2.census.gov/geo/pvs/tiger2010st/25_Massachusetts/25/ --no-parent --relative ↩
     --accept=*bg10.zip,*tract10.zip,*tabblock10.zip --mirror --reject=html
rm -f ${TMPDIR}/*.*
${PSQL} -c "DROP SCHEMA tiger_staging CASCADE;"
${PSQL} -c "CREATE SCHEMA tiger_staging;"
cd $STATEDIR
for z in *.zip; do $UNZIPTOOL -o -d $TMPDIR $z; done
:
:
```

Siehe auch

Loader_Generate_Script

11.2.10 Loader_Generate_Script

Loader_Generate_Script — Erzeugt für gegebene Plattform und Bundesstaaten ein Shellskript, das die TIGER Daten herunterlädt, bereitstellt und in das Schema `tiger_data` importiert. Jedes Bundesstaat-Skript wird in einem eigenen Datensatz ausgegeben. Die neueste Version unterstützt die geänderte Struktur von Tiger 2010 und lädt ebenfalls die Census Tract, Block Groups und Blocks Tabellen.

Synopsis

setof text **loader_generate_script**(text[] param_states, text os);

Beschreibung

Erzeugt für gegebene Plattform und Bundesstaaten ein Shellskript, das die TIGER Daten herunterlädt, bereitstellt und in das Schema `tiger_data` importiert. Jedes Bundesstaat-Skript wird in einem eigenen Datensatz ausgegeben.

Zum Herunterladen wird auf Linux unzip (auf Windows standardmäßig 7-zip) und wget verwendet. Es verwendet Section 4.7.2 um die Daten zu laden. Die kleinste Einheit, die bearbeitet wird ist ein ganzer Bundesstaat. Es werden nur die Dateien in den Ordnern "staging" und "temp" bearbeitet.

Verwendet die folgenden Kontrolltabellen, um den Verarbeitungsprozess und die verschiedenen Variationen der Betriebssysteme in Bezug auf den Shellsyntax zu überprüfen.

1. `loader_variables` behält den Überblick über verschiedenen Variablen, wie Census Site, Jahr, Daten- und "staging"-Schemata
2. `loader_platform` Profile von verschiedenen Plattformen und Speicherplätze der ausführbaren Programme. Beinhaltet Windows und Linux. Weitere können hinzugefügt werden.

3. `loader_lookuptables` jeder Datensatz definiert einen bestimmten Tabellentyp (state, county), um Datensätze zu bearbeiten und zu importieren. Legt die Schritte fest, die notwendig sind, um Daten zu importieren, bereitzustellen und hinzuzufügen, und um Spalten, Indizes und Constraints zu löschen. Jede Tabelle wird mit einem Präfix des Bundesstaates versehen und erbt von einer Tabelle in dem Schema TIGER. z.B. wird die Tabelle `tiger_data.ma_faces` erstellt, welche von `tiger.faces` erbt

Verfügbarkeit: 2.0.0 unterstützt die strukturierten Daten von Tiger 2010 und ladet die Tabellen "tract" (Census Area), "bg" (Census Block Groups) und "tabblocks" (Census Blocks).

Note
Wenn Sie pgAdmin3 verwenden, dann seien Sie gewarnt, dass pgAdmin3 langen Text abschneidet. Um dies zu beheben, können Sie *File -> Options -> Query Tool -> Query Editor - > Max. characters per column* auf mehr als 50000 Zeichen setzen.

Beispiele

Verwendung von psql; die Datenbank ist "gistest" und `/gisdata/data_load.sh` ist die Datei mit den Shell-Befehlen die ausgeführt werden sollen.

```
psql -U postgres -h localhost -d gistest -A -t \
 -c "SELECT Loader_Generate_Script(ARRAY['MA'], 'gistest')"
> /gisdata/data_load.sh;
```

Erzeugt ein Skript, das Daten von 2 Staaten im Windows Shell Script Format ladet.

```
SELECT loader_generate_script(ARRAY['MA','RI'], 'windows') AS result;
-- result --
set TMPDIR=\gisdata\temp\
set UNZIPTOOL="C:\Program Files\7-Zip\7z.exe"
set WGETTOOL="C:\wget\wget.exe"
set PGBIN=C:\Program Files\PostgreSQL\9.4\bin\
set PGPORT=5432
set PGHOST=localhost
set PGUSER=postgres
set PGPASSWORD=yourpasswordhere
set PGDATABASE=geocoder
set PSQL="%PGBIN%psql"
set SHP2PGSQL="%PGBIN%shp2pgsql"
cd \gisdata

cd \gisdata
%WGETTOOL% ftp://ftp2.census.gov/geo/tiger/TIGER2015/PLACE/tl_*_25_* --no-parent --relative ↩
     --recursive --level=2 --accept=zip --mirror --reject=html
cd \gisdata/ftp2.census.gov/geo/tiger/TIGER2015/PLACE
:
:
```

Erzeugt ein Shell-Skript

```
SELECT loader_generate_script(ARRAY['MA','RI'], 'sh') AS result;
-- result --
TMPDIR="/gisdata/temp/"
UNZIPTOOL=unzip
WGETTOOL="/usr/bin/wget"
export PGBIN=/usr/lib/postgresql/9.4/bin
-- variables used by psql: https://www.postgresql.org/docs/current/static/libpq-envars.html
export PGPORT=5432
export PGHOST=localhost
```

```
export PGUSER=postgres
export PGPASSWORD=yourpasswordhere
export PGDATABASE=geocoder
PSQL=${PGBIN}/psql
SHP2PGSQL=${PGBIN}/shp2pgsql
cd /gisdata

cd /gisdata
wget ftp://ftp2.census.gov/geo/tiger/TIGER2015/PLACE/tl_*_25_* --no-parent --relative -- ↩
    recursive --level=2 --accept=zip --mirror --reject=html
cd /gisdata/ftp2.census.gov/geo/tiger/TIGER2015/PLACE
rm -f ${TMPDIR}/*.*
:
:
```

Siehe auch

Section 2.4.1, Loader_Generate_Nation_Script

11.2.11 Loader_Generate_Nation_Script

Loader_Generate_Nation_Script — Erzeugt für die angegebene Plattform ein Shell-Skript, welches die County und State Lookup Tabellen ladet.

Synopsis

text **loader_generate_nation_script**(text os);

Beschreibung

Erstellt für die gegebene Plattform ein Shell Skript, dass die Tabellen `county_all`, `county_all_lookup` und `state_all` in das Schema `tiger_data` lädt. Diese Tabellen erben jeweils von den Tabellen `county`, `county_lookup` und `state`, die sich im Schema `tiger` befinden.

Verwendet unzip auf Linux (auf Windows standardmäßig 7--zip) und wget zum Herunterladen. Verwendet Section 4.7.2 um die Daten zu laden.

Verwendet die Kontrolltabellen `tiger.loader_platform`, `tiger.loader_variables` und `tiger.loader_lookuptabl` um den Verarbeitungsprozess zu überprüfen, sowie unterschiedliche Varianten für die Syntax verschiedener Betriebssysteme.

1. `loader_variables` behält den Überblick über verschiedenen Variablen, wie Census Site, Jahr, Daten- und "staging"-Schemata
2. `loader_platform` Profile von verschiedenen Plattformen und Speicherplätze der ausführbaren Programme. Beinhaltet Windows und Linux/Unix. Weitere können hinzugefügt werden.
3. `loader_lookuptables` jeder Datensatz definiert einen bestimmten Tabellentyp (state, county), um Datensätze zu bearbeiten und zu importieren. Legt die Schritte fest, die notwendig sind, um Daten zu importieren, bereitzustellen und hinzuzufügen, und um Spalten, Indizes und Constraints zu löschen. Jede Tabelle wird mit einem Präfix des Bundesstaates versehen und erbt von einer Tabelle in dem Schema TIGER. z.B. wird die Tabelle `tiger_data.ma_faces` erstellt, welche von `tiger.faces` erbt

Enhanced: 2.4.1 zip code 5 tabulation area (zcta5) load step was fixed and when enabled, zcta5 data is loaded as a single table called zcta5_all as part of the nation script load.

Verfügbarkeit: 2.1.0

Note
If you want zip code 5 tabulation area (zcta5) to be included in your nation script load, do the following:

```
UPDATE tiger.loader_lookuptables SET load = true WHERE table_name = 'zcta510';
```

Note
Falls Sie die Version `tiger_2010` haben und `tiger_2011` laden wollen, dann müssen Sie als allererstes das Skript "loader_generate_nation_script" und die Löschanweisungen Drop_Nation_Tables_Generate_Script ausführen, bevor Sie dieses Skript laufen lassen.

Beispiele

Erzeugt ein Script um die Daten einer Nation in Windows zu laden.

```
SELECT loader_generate_nation_script('windows');
```

Erzeugt ein Script um die Daten auf Linux/Unix Systemen zu laden.

```
SELECT loader_generate_nation_script('sh');
```

Siehe auch

Loader_Generate_Script, Missing_Indexes_Generate_Script

11.2.12 Missing_Indexes_Generate_Script

Missing_Indexes_Generate_Script — Findet alle Tabellen mit Schlüsselspalten, die für JOINs vom Geokodierer verwendet werden und keinen Index aufweisen; gibt ein DDL (SQL) aus, dass die Indizes für diese Tabellen festlegt.

Synopsis

text **Missing_Indexes_Generate_Script**();

Beschreibung

Findet alle Tabellen in den Schemata `tiger` und `tiger_data`, bei denen die Schlüsselspalten, die für Joins vom Geokodierer verwendet werden keine Indizes aufweisen. Weiters wird ein SQL-DDL Skript ausgegeben, das die Indizes für diese Tabellen festlegt. Dabei handelt es sich um eine Hilfsfunktion, die Abfragen schneller macht, indem die benötigten Indizes, die während des Imports gefehlt haben, neu hinzufügt werden. Wenn der Geocodierer verbessert wird, dann wird diese Funktion aktualisiert um sie für die Verwendung neuer Indizes anzupassen. Wenn diese Funktion nichts zurückgibt, so bedeutet dies, dass alle Tabellen entsprechend befüllt und die Schlüsselindizes bereits vorhanden sind.

Verfügbarkeit: 2.0.0

Beispiele

```
SELECT missing_indexes_generate_script();
-- output: This was run on a database that was created before many corrections were made to ↩
     the loading script ---
CREATE INDEX idx_tiger_county_countyfp ON tiger.county USING btree(countyfp);
CREATE INDEX idx_tiger_cousub_countyfp ON tiger.cousub USING btree(countyfp);
CREATE INDEX idx_tiger_edges_tfidr ON tiger.edges USING btree(tfidr);
CREATE INDEX idx_tiger_edges_tfidl ON tiger.edges USING btree(tfidl);
CREATE INDEX idx_tiger_zip_lookup_all_zip ON tiger.zip_lookup_all USING btree(zip);
CREATE INDEX idx_tiger_data_ma_county_countyfp ON tiger_data.ma_county USING btree(countyfp ↩
    );
CREATE INDEX idx_tiger_data_ma_cousub_countyfp ON tiger_data.ma_cousub USING btree(countyfp ↩
    );
CREATE INDEX idx_tiger_data_ma_edges_countyfp ON tiger_data.ma_edges USING btree(countyfp);
CREATE INDEX idx_tiger_data_ma_faces_countyfp ON tiger_data.ma_faces USING btree(countyfp);
```

Siehe auch

Loader_Generate_Script, Install_Missing_Indexes

11.2.13 Normalize_Address

Normalize_Address — Für einen gegebenen Adressentext wird der zusammengesetzte Datentyp `norm_addy` zurückgeben, der ein Suffix und ein Präfix für die Straße, einen normierten Datentyp, die Straße, den Straßennamen etc. enthält und diese einzelnen Attributen zuweist. Diese Funktion benötigt lediglich die "lookup data", die mit dem Tiger Geokodierer paketiert sind (Tiger Census Daten werden nicht benötigt).

Synopsis

norm_addy **normalize_address**(varchar in_address);

Beschreibung

Für einen gegebenen Adressentext wird der zusammengesetzte Datentyp `norm_addy` zurückgeben, der ein Suffix und ein Präfix für die Straße, einen normierten Datentyp, die Straße, den Straßennamen etc. enthält und diese einzelnen Attributen zuweist. Dies ist der erste Schritt beim Geokodieren, der alle Adressen in eine standardisierte Postform bringt. Es werden keine anderen Daten, außer jenen die mit dem Geokodierer paketiert sind, benötigt.

Diese Funktion verwendet lediglich die verschiedenen Lookup-Tabellen "direction/state/suffix/", die mit tiger_gecoder vorinstalliert wurden und sich im Schema `tiger` befinden. Es ist deshalb nicht nötig Tiger Census Daten oder sonstige zusätzliche Daten herunterzuladen um diese Funktion zu verwenden.

Verwendet verschiedene Kontrolltabellen im Schema `tiger` zum Normalisieren der Eingabeadressen.

Die Attribute des Objekttyps `norm_addy`, die in dieser Reihenfolge von der Funktion zurückgegeben werden, wobei () für ein verbindliches Attribut und [] für ein optionales Attribut steht:

(address) [predirAbbrev] (streetName) [streetTypeAbbrev] [postdirAbbrev] [internal] [location] [stateAbbrev] [zip] [parsed] [zip4] [address_alphanumeric]

Erweiterung: 2.4.0 die zusätzlichen Felder "zip4" und "address_alphanumeric" wurden zum Objekt "norm_addy" hinzugefügt.

1. `address` ist eine Ganzzahl: Die Hausnummer
2. `predirAbbrev` ist ein Textfeld variabler Länge: Ein Präfix für die Straßenrichtung, wie N, S, E, W etc. Wird durch die `direction_lookup` Tabelle gesteuert.
3. `streetName` varchar

4. Das Textfeld `streetTypeAbbrev` mit variabler Länge beinhaltet eine abgekürzte Version der Straßentypen: z.B. St, Ave, Cir. Diese werden über die Tabelle `street_type_lookup` kontrolliert.
5. Das Textfeld `postdirAbbrev` mit variabler Länge beinhaltet Suffixe für die Abkürzungen der Straßenrichtung; N, S, E, W etc. Diese werden über die Tabelle `direction_lookup` kontrolliert.
6. `internal` ein Textfeld variabler Länge mit einer zusätzlichen Adressangabe, wie Apartment- oder Suitenummer.
7. `location` ein Textfeld variabler Länge, üblicherweise eine Stadt oder eine autonome Provinz.
8. `stateAbbrev` ein Textfeld variabler Länge mit dem zwei Zeichen langen Bundesstaat der USA. z.B. MA, NY, MI. Diese werden durch die Tabelle `state_lookup` beschränkt.
9. Das Textfeld `zip` mit variabler Länge enthält den 5-Zeichen langen Zip-Code. z.B. 02109
10. `parsed` boolesche Variable - zeigt an, ob eine Adresse durch Normalisierung erstellt wurde. Die Funktion "normalize_address" setzt diese Variable auf TRUE, bevor die Adresse zurückgegeben wird.
11. `zip4` die letzten 4 Zeichen des 9 Zeichen langen Zip-Codes. Verfügbarkeit: PostGIS 2.4.0.
12. `address_alphanumeric` Vollständige Hausnummer, auch wenn sie Buchstaben wie bei "17R" enthält. Kann mit der Funktion Pagc_Normalize_Address besser geparst werden. Verfügbarkeit: PostGIS 2.4.0.

Beispiele

Gibt die ausgewählten Felder aus. Verwenden Sie bitte Pprint_Addy wenn Sie einen sauber formatierten Ausgabetext benötigen.

```
SELECT address As orig, (g.na).streetname, (g.na).streettypeabbrev
 FROM (SELECT address, normalize_address(address) As na
        FROM addresses_to_geocode) As g;

                        orig                          |  streetname   | streettypeabbrev
------------------------------------------------------+---------------+------------------
 28 Capen Street, Medford, MA                         | Capen         | St
 124 Mount Auburn St, Cambridge, Massachusetts 02138  | Mount Auburn  | St
 950 Main Street, Worcester, MA 01610                 | Main          | St
 529 Main Street, Boston MA, 02129                    | Main          | St
 77 Massachusetts Avenue, Cambridge, MA 02139         | Massachusetts | Ave
 25 Wizard of Oz, Walaford, KS 99912323               | Wizard of Oz  |
```

Siehe auch

Geocode, Pprint_Addy

11.2.14 Pagc_Normalize_Address

Pagc_Normalize_Address — Für einen gegebenen Adressentext wird der zusammengesetzte Datentyp `norm_addy` zurückgeben, der ein Suffix und ein Präfix für die Straße, einen normierten Datentyp, die Straße, den Straßennamen etc. enthält und diese einzelnen Attributen zuweist. Diese Funktion benötigt lediglich die "lookup data", die mit dem Tiger Geokodierer paketiert sind (Tiger Census Daten werden nicht benötigt). Benötigt die Erweiterung "address_standardizer".

Synopsis

norm_addy **pagc_normalize_address**(varchar in_address);

Beschreibung

Für einen gegebenen Adressentext wird der zusammengesetzte Datentyp `norm_addy` zurückgeben, der ein Suffix und ein Präfix für die Straße, einen normierten Datentyp, die Straße, den Straßennamen etc. enthält und diese einzelnen Attributen zuweist. Dies ist der erste Schritt beim Geokodieren, der alle Adressen in eine standardisierte Postform bringt. Es werden keine anderen Daten, außer jenen die mit dem Geokodierer paketiert sind, benötigt.

Diese Funktion verwendet lediglich die verschiedenen Lookup-Tabellen "pagc_*", die mit tiger_gecoder vorinstalliert wurden und sich im Schema `tiger` befinden. Es ist deshalb nicht nötig Tiger Census Daten oder sonstige zusätzliche Daten herunterzuladen um diese Funktion zu verwenden. Möglicherweise stellt sich heraus, dass Sie Lookup-Tabellen im Schema `tiger` um weitere Abkürzungen und alternative Namensgebungen erweitern müssen.

Verwendet verschiedene Kontrolltabellen im Schema `tiger` zum Normalisieren der Eingabeadressen.

Die Attribute des Objekttyps `norm_addy`, die in dieser Reihenfolge von der Funktion zurückgegeben werden, wobei () für ein verbindliches Attribut und [] für ein optionales Attribut steht:

Bei Normalize_Address gibt es geringfügige Abweichungen beim Format und bei der Groß- und Kleinschreibung.

Verfügbarkeit: 2.1.0

This method needs address_standardizer extension.

(address) [predirAbbrev] (streetName) [streetTypeAbbrev] [postdirAbbrev] [internal] [location] [stateAbbrev] [zip]

Die native "standardaddr" der Erweiterung "address_standardizer" ist ein bisschen reichhaltiger als "norm_addy", da es für die Unterstützung internationaler Adressen (inklusive Länder) entworfen wurde. Die entsprechenden Attribute von "standardaddr" sind:

house_num,predir, name, suftype, sufdir, unit, city, state, postcode

Erweiterung: 2.4.0 die zusätzlichen Felder "zip4" und "address_alphanumeric" wurden zum Objekt "norm_addy" hinzugefügt.

1. `address` ist eine Ganzzahl: Die Hausnummer
2. `predirAbbrev` ist ein Textfeld variabler Länge: Ein Präfix für die Straßenrichtung, wie N, S, E, W etc. Wird durch die `direction_lookup` Tabelle gesteuert.
3. `streetName` varchar
4. Das Textfeld `streetTypeAbbrev` mit variabler Länge beinhaltet eine abgekürzte Version der Straßentypen: z.B. St, Ave, Cir. Diese werden über die Tabelle `street_type_lookup` kontrolliert.
5. Das Textfeld `postdirAbbrev` mit variabler Länge beinhaltet Suffixe für die Abkürzungen der Straßenrichtung; N, S, E, W etc. Diese werden über die Tabelle `direction_lookup` kontrolliert.
6. `internal` ein Textfeld variabler Länge mit einer zusätzlichen Adressangabe, wie Apartment- oder Suitenummer.
7. `location` ein Textfeld variabler Länge, üblicherweise eine Stadt oder eine autonome Provinz.
8. `stateAbbrev` ein Textfeld variabler Länge mit dem zwei Zeichen langen Bundesstaat der USA. z.B. MA, NY, MI. Diese werden durch die Tabelle `state_lookup` beschränkt.
9. Das Textfeld `zip` mit variabler Länge enthält den 5-Zeichen langen Zip-Code. z.B. 02109
10. `parsed` boolesche Variable - zeigt an, ob eine Adresse durch Normalisierung erstellt wurde. Die Funktion "normalize_address" setzt diese Variable auf TRUE, bevor die Adresse zurückgegeben wird.
11. `zip4` die letzten 4 Zeichen des 9 Zeichen langen Zip-Codes. Verfügbarkeit: PostGIS 2.4.0.
12. `address_alphanumeric` Vollständige Hausnummer, auch wenn sie Buchstaben wie bei "17R" enthält. Kann mit der Funktion Pagc_Normalize_Address besser geparst werden. Verfügbarkeit: PostGIS 2.4.0.

Beispiele

Beispiel für einen Einzelaufruf

```
SELECT addy.*
FROM pagc_normalize_address('9000 E ROO ST STE 999, Springfield, CO') AS addy;


 address | predirabbrev | streetname | streettypeabbrev | postdirabbrev | internal  |   ↩
    location    | stateabbrev | zip | parsed
---------+--------------+------------+------------------+---------------+-----------+-------------+-

    9000 | E            | ROO        | ST               |               | SUITE 999 |  ↩
       SPRINGFIELD | CO          |     | t
```

Batch call (Stapelverarbeitung). Zurzeit gibt es Geschwindigkeitsprobleme bezüglich des Wrappers des postgis_tiger_geocoder für den address_standardizer. Dies wird hoffentlich in späteren Versionen behoben. Wenn Sie Geschwindigkeit für den Aufruf einer Stapelverarbeitung zur Erstellung von normaddy benötigen, können Sie diese Probleme umgehen, indem Sie die Funktion "standardize_address" des address_standardizer direkt aufrufen. Dies wird nachfolgend gezeigt und entspricht ungefähr der Aufgabe unter Normalize_Address wo Daten verwendet werden, die unter Geocode erstellt wurden.

```
WITH g AS (SELECT address, ROW((sa).house_num, (sa).predir, (sa).name
  , (sa).suftype, (sa).sufdir, (sa).unit , (sa).city, (sa).state, (sa).postcode, true):: ↩
     norm_addy As na
 FROM (SELECT address, standardize_address('tiger.pagc_lex'
       , 'tiger.pagc_gaz'
       , 'tiger.pagc_rules', address) As sa
        FROM addresses_to_geocode) As g)
SELECT address As orig, (g.na).streetname, (g.na).streettypeabbrev
 FROM  g;

 orig                                                 |  streetname   | streettypeabbrev
------------------------------------------------------+---------------+------------------
 529 Main Street, Boston MA, 02129                    | MAIN          | ST
 77 Massachusetts Avenue, Cambridge, MA 02139         | MASSACHUSETTS | AVE
 25 Wizard of Oz, Walaford, KS 99912323               | WIZARD OF     |
 26 Capen Street, Medford, MA                         | CAPEN         | ST
 124 Mount Auburn St, Cambridge, Massachusetts 02138 | MOUNT AUBURN  | ST
 950 Main Street, Worcester, MA 01610                 | MAIN          | ST
```

Siehe auch

Normalize_Address, Geocode

11.2.15 Pprint_Addy

Pprint_Addy — Für einen zusammengesetzten Objekttyp `norm_addy` wird eine formatierte Darstellung zurückgegeben. Wird üblicherweise in Verbindung mit normalize_address verwendet.

Synopsis

varchar **pprint_addy**(norm_addy in_addy);

Beschreibung

Für einen zusammengesetzten Objekttyp `norm_addy` wird eine formatierte Darstellung zurückgegeben. Außer den Daten die mit dem Geokodierer paketiert sind, werden keine weiteren Daten benötigt.

Wird üblicherweise in Verbindung mit Normalize_Address verwendet.

Beispiele

Formatiert eine einzelne Adresse

```
SELECT pprint_addy(normalize_address('202 East Fremont Street, Las Vegas, Nevada 89101'))  ↩
    As pretty_address;
           pretty_address
---------------------------------------
 202 E Fremont St, Las Vegas, NV 89101
```

Adressen einer Tabelle formatieren

```
SELECT address As orig, pprint_addy(normalize_address(address)) As pretty_address
        FROM addresses_to_geocode;

                        orig                         |              pretty_address
-----------------------------------------------------+------------------------------------------ ↩

 529 Main Street, Boston MA, 02129                   | 529 Main St, Boston MA, 02129
 77 Massachusetts Avenue, Cambridge, MA 02139        | 77 Massachusetts Ave, Cambridge, MA  ↩
    02139
 28 Capen Street, Medford, MA                        | 28 Capen St, Medford, MA
 124 Mount Auburn St, Cambridge, Massachusetts 02138 | 124 Mount Auburn St, Cambridge, MA  ↩
    02138
 950 Main Street, Worcester, MA 01610                | 950 Main St, Worcester, MA 01610
```

Siehe auch

Normalize_Address

11.2.16 Reverse_Geocode

Reverse_Geocode — Nimmt einen geometrischen Punkt in einem bekannten Koordinatenreferenzsystem entgegen und gibt einen Datensatz zurück, das ein Feld mit theoretisch möglichen Adressen und ein Feld mit Straßenkreuzungen beinhaltet. Wenn include_strnum_range = true, dann beinhalten die Straßenkreuzungen den "Street Range" (Kennung des Straßenabschnitts).

Synopsis

record **Reverse_Geocode**(geometry pt, boolean include_strnum_range=false, geometry[] OUT intpt, norm_addy[] OUT addy, varchar[] OUT street);

Beschreibung

Nimmt einen geometrischen Punkt in einem bekannten Koordinatenreferenzsystem entgegen und gibt einen Datensatz zurück, das ein Feld mit theoretisch möglichen Adressen und ein Feld mit Straßenkreuzungen beinhaltet. Wenn include_strnum_range = true, dann beinhalten die Straßenkreuzungen den "Street Range" (Kennung des Straßenabschnitts). Die Standardeinstellung für include_strnum_range ist FALSE. Die Adressen werden nach dem Abstand des Punktes zu den jeweiligen Straßen gereiht, so dass die erste Adresse höchstwahrscheinlich die Richtige ist.

Warum sprechen wir von theoretischen anstatt von tatsächlichen Adressen. Die Daten von TIGER haben keine echten Adressen, sondern nur Straßenabschnitte (Street Ranges). Daher ist die theoretische Adresse eine anhand der Straßenabschnitte interpolierte Adresse. So gibt zum Beispiel die Interpolation einer Adresse "26 Court St." und "26 Court Sq." zurück, obwohl keine Adresse mit der Bezeichnung "26 Court Sq." existiert. Dies beruht darauf, dass ein Punkt an einem Eckpunkt von 2 Straßen liegen kann und die Logik entlang beider Straßen interpoliert. Die Logik nimmt auch an, dass sich die Adressen in einem regelmäßigen Abstand entlang der Straße befinden, was natürlich falsch ist, da ein öffentliches Gebäude einen großen Bereich eines Straßenabschnitts (street range) einnehmen kann und der Rest der Gebäude sich lediglich am Ende befinden.

Anmerkung: diese Funktion benötigt TIGER-Daten. Wenn Sie für diesen Bereich keine Daten geladen haben, dann bekommen Sie einen Datensatz mit lauter NULLs zurück.

Die zurückgelieferten Elemente des Datensatzes lauten wie folgt:

1. `intpt` ist ein Feld mit Punkten: Dies sind Punkte auf der Mittellinie der Straße, die dem gegebenen Punkt am nächsten liegt. Es beinhaltet soviele Punkte wie es Adressen gibt.
2. `addy` ist ein Feld mit normalisierten Adressen (norm_addy): Diese sind ein Feld mit möglichen Adressen die zu dem gegebenen Punkt passen. Die erste in dem Feld ist die wahrscheinlichste Adresse. Üblicherweise sollte dies nur eine Adresse sein, ausgenommen dem Fall wo ein Punkt an der Ecke von 2 oder 3 Straßen liegt, oder wenn der Punkt irgendwo auf der Straße und nicht auf der Seite liegt.
3. `street` ein Feld mit varchar (Textfeld variabler Länge): Dies sind Querstraßen (oder die Straße) (sich kreuzende Straßen oder die Straße auf der der Punkt liegt).

Enhanced: 2.4.1 if optional zcta5 dataset is loaded, the reverse_geocode function can resolve to state and zip even if the specific state data is not loaded. Refer to Loader_Generate_Nation_Script for details on loading zcta5 data.

Verfügbarkeit: 2.0.0

Beispiele

Beispiel für eine Position an der Ecke von zwei Straßen, aber näher bei einer. Näherungsweise die Lage des MIT: 77 Massachusetts Ave, Cambridge, MA 02139. Beachten Sie, dass obwohl wir keine 3 Straßen haben, PostgreSQL nur NULL für die Einträge oberhalb unserer oberen Begrenzung zurückgibt; kann also gefahrlos verwendet werden. Dieses Beispiel verwendet Adressenbereiche

```
SELECT pprint_addy(r.addy[1]) As st1, pprint_addy(r.addy[2]) As st2, pprint_addy(r.addy[3]) ↩
    As st3,
           array_to_string(r.street, ',') As cross_streets
       FROM reverse_geocode(ST_GeomFromText('POINT(-71.093902 42.359446)',4269),true) As r ↩
           ;

 result
 ------
      st1                                    | st2 | st3 |                cross_streets
-----------------------------------------+-----+-----+-------------------------------------------

 67 Massachusetts Ave, Cambridge, MA 02139 |     |     | 67 - 127 Massachusetts Ave,32 - 88 ↩
     Vassar St
```

Hier haben wir die Adressenbereiche für die Querstraßen nicht inkludiert und eine Position ausgewählt, die sehr sehr nahe an einer Ecke von 2 Straßen liegt, damit diese (Position) von zwei unterschiedlichen Adressen erkannt werden könnte.

```
SELECT pprint_addy(r.addy[1]) As st1, pprint_addy(r.addy[2]) As st2,
pprint_addy(r.addy[3]) As st3, array_to_string(r.street, ',') As cross_str
FROM reverse_geocode(ST_GeomFromText('POINT(-71.06941 42.34225)',4269)) As r;

result
--------
               st1                |               st2                | st3 | cross_str
----------------------------------+----------------------------------+-----+------------------------ ←

 5 Bradford St, Boston, MA 02118 | 49 Waltham St, Boston, MA 02118 |     | Waltham St
```

Bei diesem Beispiel wird das Beispiel von Geocode wiederverwendet und wir wollen nur die primäre Adresse und höchstens 2 Straßenkreuzungen.

```
SELECT actual_addr, lon, lat, pprint_addy((rg).addy[1]) As int_addr1,
    (rg).street[1] As cross1, (rg).street[2] As cross2
FROM (SELECT address As actual_addr, lon, lat,
    reverse_geocode( ST_SetSRID(ST_Point(lon,lat),4326) ) As rg
    FROM addresses_to_geocode WHERE rating
> -1) As foo;

                    actual_addr                     |    lon    |   lat    |  ↩
                                        int_addr1                 |     cross1      |    ↩
                        cross2
-----------------------------------------------------+-----------+----------+-----------------------

 529 Main Street, Boston MA, 02129                   | -71.07181 | 42.38359 | 527 Main St,  ↩
     Boston, MA 02129             | Medford St      |
 77 Massachusetts Avenue, Cambridge, MA 02139        | -71.09428 | 42.35988 | 77  ↩
     Massachusetts Ave, Cambridge, MA 02139 | Vassar St       |
 26 Capen Street, Medford, MA                        | -71.12377 | 42.41101 | 9 Edison Ave, ↩
      Medford, MA 02155           | Capen St        | Tesla Ave
 124 Mount Auburn St, Cambridge, Massachusetts 02138 | -71.12304 | 42.37328 | 3 University  ↩
     Rd, Cambridge, MA 02138      | Mount Auburn St |
 950 Main Street, Worcester, MA 01610                | -71.82368 | 42.24956 | 3 Maywood St, ↩
      Worcester, MA 01603         | Main St         | Maywood Pl
```

Siehe auch

Pprint_Addy, Loader_Generate_Nation_Script

11.2.17 Topology_Load_Tiger

Topology_Load_Tiger — Lädt die Tiger-Daten einer bestimmte Region in die PostGIS Topologie, transformiert sie in das Koordinatenreferenzsystem der Topologie und fängt sie entsprechend der Genauigkeitstoleranz der Topologie.

Synopsis

text **Topology_Load_Tiger**(varchar topo_name, varchar region_type, varchar region_id);

Beschreibung

Lädt die Tiger Daten einer bestimmten Region in die PostGIS Topologie. Die Maschen, Knoten und Kanten werden in das Koordinatenreferenzsystem der Zieltopologie transformiert und die Knoten werden entsprechend der Toleranz der Zieltopologie gefangen. Die erzeugten Maschen, Knoten und Kanten behalten die ids der ursprünglichen Maschen, Knoten und Kanten der Tiger Daten, wodurch die Daten zukünftig leichter mit neuen Tiger Daten abgeglichen werden können. Gibt eine Zusammenfassung des Prozessablaufs aus.

Dies ist zum Beispiel nützlich, wenn Daten neu strukturiert werden müssen, bei denen die neu ausgebildeten Polygone an den Mittellinien der Strassen ausgerichtet sein sollen und die erzeugten Polygone sich nicht überlappen dürfen.

> **Note**
> Diese Funktion benötigt Tiger Daten und das Topologiemodul von PostGIS. Für weiterführende Information siehe Chapter 8 und Section 2.2.3. Wenn keine Daten für den betreffenden Bereich geladen wurden, dann werden auch keine topologischen Datensätze erstellt. Diese Funktion versagt auch dann, wenn keine Topologie mit den topologischen Funktionen aufgebaut wurde.

Note
Die meisten topologischen Überprüfungsfehler entstehen durch Toleranzprobleme, wenn nach der Transformation die Kanten nicht genau abgeglichen sind oder sich überlappen. Um dies zu beheben, können Sie bei topologischen Überprüfungsfehlern die Präzision erhöhen oder erniedrigen.

Benötigte Parameter:

1. `topo_name` Die Bezeichnung einer bestehenden PostGIS Topologie, in die Daten geladen werden.
2. `region_type` Der Typ des begrenzten Bereichs. Zurzeit wird nur `place` und `county` unterstützt. Geplant sind noch einige weitere Typen. In dieser Tabelle werden die Begrenzungen definiert. z.B. `tiger.place`, `tiger.county`
3. `region_id` Entspricht der "geoid" von TIGER und ist ein eindeutige Identifikator für die Region in der Tabelle. Für Plätze ist es die Spalte `plcidfp` in `tiger.place`. Für "county" ist es die Spalte `cntyidfp` in `tiger.county`

Verfügbarkeit: 2.0.0

Beispiel: Boston, Massachusetts Topologie

Erstellt eine Topologie für Boston, Massachusetts in "Mass State Plane Feet" (2249) mit einer Toleranz von 0.25 Meter und lädt anschließend die Maschen, Kanten und Knoten von Tiger für Boston City.

```
SELECT topology.CreateTopology('topo_boston', 2249, 0.25);
createtopology
--------------
   15
-- 60,902 ms ~ 1 minute on windows 7 desktop running 9.1 (with 5 states tiger data loaded)
SELECT tiger.topology_load_tiger('topo_boston', 'place', '2507000');
-- topology_loader_tiger --
29722 edges holding in temporary. 11108 faces added. 1875 edges of faces added.  20576  ↩
    nodes added.
19962 nodes contained in a face.  0 edge start end corrected.  31597 edges added.

-- 41 ms --
SELECT topology.TopologySummary('topo_boston');
 -- topologysummary--
Topology topo_boston (15), SRID 2249, precision 0.25
20576 nodes, 31597 edges, 11109 faces, 0 topogeoms in 0 layers

-- 28,797 ms to validate yeh returned no errors --
SELECT * FROM
    topology.ValidateTopology('topo_boston');

       error       |   id1    |    id2
-------------------+----------+-----------
```

Beispiel: Suffolk, Massachusetts Topologie

Erstellt eine Topologie für Suffolk, Massachusetts in "Mass State Plane Meters" (26986) mit einer Toleranz von 0.25 Meter und lädt anschließend die Maschen, Kanten und Knoten von Tiger für Suffolk County.

```
SELECT topology.CreateTopology('topo_suffolk', 26986, 0.25);
-- this took 56,275 ms ~ 1 minute on Windows 7 32-bit with 5 states of tiger loaded
-- must have been warmed up after loading boston
SELECT tiger.topology_load_tiger('topo_suffolk', 'county', '25025');
-- topology_loader_tiger --
 36003 edges holding in temporary. 13518 faces added. 2172 edges of faces added.
```

```
 24761 nodes added.  24075 nodes contained in a face.  0 edge start end corrected.  38175  ↩
    edges added.
-- 31 ms --
SELECT topology.TopologySummary('topo_suffolk');
 -- topologysummary--
 Topology topo_suffolk (14), SRID 26986, precision 0.25
24761 nodes, 38175 edges, 13519 faces, 0 topogeoms in 0 layers

-- 33,606 ms to validate --
SELECT * FROM
    topology.ValidateTopology('topo_suffolk');

       error       |   id1    |    id2
-------------------+----------+-----------
 coincident nodes  | 81045651 |  81064553
 edge crosses node | 81045651 |  85737793
 edge crosses node | 81045651 |  85742215
 edge crosses node | 81045651 | 620628939
 edge crosses node | 81064553 |  85697815
 edge crosses node | 81064553 |  85728168
 edge crosses node | 81064553 |  85733413
```

Siehe auch

CreateTopology, CreateTopoGeom, TopologySummary, ValidateTopology

11.2.18 Set_Geocode_Setting

Set_Geocode_Setting — Setzt die Einstellungen, welche das Verhalten der Funktionen des Geokodierers beeinflussen.

Synopsis

text **Set_Geocode_Setting**(text setting_name, text setting_value);

Beschreibung

Setzt bestimmte Einstellwerte, die in der Tabelle "tiger.geocode_settings" gespeichert werden. Die Einstellungen erlauben auf "debugging" der Funktionen umzuschalten. Für später ist geplant auch die Rangordnung über die Einstellungen zu kontrollieren. Eine aktuelle Liste der Einstellungen ist unter Get_Geocode_Setting aufgeführt.

Verfügbarkeit: 2.1.0

Das Beispiel gibt die "debugging" Einstellungen aus

Wenn Sie Geocode ausführen und diese Funktion TRUE ist, dann werden die Abfragen und die Zeitmessung von dem Log "NOTICE" ausgegeben.

```
SELECT set_geocode_setting('debug_geocode_address', 'true') As result;
result
---------
true
```

Siehe auch

Get_Geocode_Setting

Chapter 12

PostGIS Special Functions Index

12.1 PostGIS Aggregate Functions

The functions below are spatial aggregate functions that are used in the same way as SQL aggregate function such as `sum` and `average`.

- ST_3DExtent - Aggregate function that returns the 3D bounding box of geometries.
- ST_3DUnion - Perform 3D union.
- ST_AsFlatGeobuf - Return a FlatGeobuf representation of a set of rows.
- ST_AsGeobuf - Gibt eine Menge an Zeilen in der Geobuf Darstellung aus.
- ST_AsMVT - Aggregate function returning a MVT representation of a set of rows.
- ST_ClusterIntersecting - Aggregate function that clusters input geometries into connected sets.
- ST_ClusterWithin - Aggregate function that clusters geometries by separation distance.
- ST_Collect - Creates a GeometryCollection or Multi* geometry from a set of geometries.
- ST_CoverageUnion - Computes the union of a set of polygons forming a coverage by removing shared edges.
- ST_Extent - Aggregate function that returns the bounding box of geometries.
- ST_MakeLine - Erzeugt einen Linienzug aus einer Punkt-, Mehrfachpunkt- oder Liniengeometrie.
- ST_MemUnion - Aggregate function which unions geometries in a memory-efficent but slower way
- ST_Polygonize - Computes a collection of polygons formed from the linework of a set of geometries.
- ST_SameAlignment - Gibt TRUE zurück, wenn die Raster die selbe Rotation, Skalierung, Koordinatenreferenzsystem und Versatz (Pixel können auf dasselbe Gitter gelegt werden, ohne dass die Gitterlinien durch die Pixel schneiden) aufweisen. Wenn nicht, wird FALSE und eine Bechreibung des Problems ausgegeben.
- ST_Union - Computes a geometry representing the point-set union of the input geometries.
- TopoElementArray_Agg - Gibt für eine Menge an element_id, type Feldern (topoelements) ein topoelementarray zurück.

12.2 PostGIS Window Functions

The functions below are spatial window functions that are used in the same way as SQL window functions such as `row_number()`, `lead()`, and `lag()`. They must be followed by an `OVER()` clause.

- ST_ClusterDBSCAN - Window function that returns a cluster id for each input geometry using the DBSCAN algorithm.
- ST_ClusterIntersectingWin - Window function that returns a cluster id for each input geometry, clustering input geometries into connected sets.
- ST_ClusterKMeans - Window function that returns a cluster id for each input geometry using the K-means algorithm.
- ST_ClusterWithinWin - Window function that returns a cluster id for each input geometry, clustering using separation distance.
- ST_CoverageInvalidEdges - Window function that finds locations where polygons fail to form a valid coverage.
- ST_CoverageSimplify - Window function that simplifies the edges of a polygonal coverage.

12.3 PostGIS SQL-MM Compliant Functions

The functions given below are PostGIS functions that conform to the SQL/MM 3 standard

- ST_3DArea - Computes area of 3D surface geometries. Will return 0 for solids. Beschreibung Verfügbarkeit: 2.1.0 This method needs SFCGAL backend. This method implements the SQL/MM specification. SQL-MM IEC 13249-3: 8.1, 10.5 This function supports 3d and will not drop the z-index. This function supports Polyhedral surfaces. This function supports Triangles and Triangulated Irregular Network Surfaces (TIN).
- ST_3DDWithin - Tests if two 3D geometries are within a given 3D distance Beschreibung Returns true if the 3D distance between two geometry values is no larger than distance distance_of_srid. The distance is specified in units defined by the spatial reference system of the geometries. For this function to make sense the source geometries must be in the same coordinate system (have the same SRID). This function automatically includes a bounding box comparison that makes use of any spatial indexes that are available on the geometries. This function supports 3d and will not drop the z-index. This function supports Polyhedral surfaces. This method implements the SQL/MM specification. SQL-MM ? Verfügbarkeit: 2.0.0
- ST_3DDifference - Perform 3D difference Beschreibung Gibt den geometrischen Schwerpunkt einer Geometrie zurück. Verfügbarkeit: 2.2.0 This method needs SFCGAL backend. This method implements the SQL/MM specification. SQL-MM IEC 13249-3: 5.1 This function supports 3d and will not drop the z-index. This function supports Polyhedral surfaces. This function supports Triangles and Triangulated Irregular Network Surfaces (TIN).
- ST_3DDistance - Für den geometrischen Datentyp. Es wird der geringste 3-dimensionale kartesische Abstand (basierend auf dem Koordinatenreferenzsystem) zwischen zwei geometrischen Objekten in projizierten Einheiten zurückgegeben. Beschreibung Für den geometrischen Datentyp. Es wird der geringste 3-dimensionale kartesische Abstand zwischen zwei geometrischen Objekten in projizierten Einheiten (Einheiten des Koordinatenreferenzsystem) zurückgegeben. This function supports 3d and will not drop the z-index. This function supports Polyhedral surfaces. This method implements the SQL/MM specification. SQL-MM ISO/IEC 13249-3 Verfügbarkeit: 2.0.0 Änderung: 2.2.0 - Im Falle von 2D und 3D wird für ein fehlendes Z nicht mehr 0 angenommen. Changed: 3.0.0 - SFCGAL version removed
- ST_3DIntersection - Perform 3D intersection Beschreibung Return a geometry that is the shared portion between geom1 and geom2. Verfügbarkeit: 2.1.0 This method needs SFCGAL backend. This method implements the SQL/MM specification. SQL-MM IEC 13249-3: 5.1 This function supports 3d and will not drop the z-index. This function supports Polyhedral surfaces. This function supports Triangles and Triangulated Irregular Network Surfaces (TIN).
- ST_3DIntersects - Tests if two geometries spatially intersect in 3D - only for points, linestrings, polygons, polyhedral surface (area) Beschreibung Overlaps, Touches und Within implizieren räumliche Überschneidung. Wenn irgendeine dieser Eigenschaften TRUE zurückgibt, dann überschneiden sich die geometrischen Objekte auch. Disjoint impliziert FALSE für die räumliche Überschneidung. This function automatically includes a bounding box comparison that makes use of any spatial indexes that are available on the geometries. Änderung: 3.0.0 das SFCGAL Back-end wurde entfernt, das GEOS Back-end unterstützt TIN. Verfügbarkeit: 2.0.0 This function supports 3d and will not drop the z-index. This function supports Polyhedral surfaces. This function supports Triangles and Triangulated Irregular Network Surfaces (TIN). This method implements the SQL/MM specification. SQL-MM IEC 13249-3: 5.1

- ST_3DLength - Gibt den geometrischen Schwerpunkt einer Geometrie zurück. Beschreibung Gibt die 2- oder 3-dimensionale Länge einer Linie oder einer Mehrfachlinie zurück. Bei einer 2-D Linie wird die Länge nur in 2D zurückgegeben (gleich wie ST_Length und ST_Length2D) This function supports 3d and will not drop the z-index. This method implements the SQL/MM specification. SQL-MM IEC 13249-3: 7.1, 10.3 Änderung: 2.0.0 In Vorgängerversionen als ST_Length3D bezeichet.

- ST_3DPerimeter - Gibt den geometrischen Schwerpunkt einer Geometrie zurück. Beschreibung Gibt den 3-dimensionalen Umfang eines Polygons oder eines Mehrfachpolygons zurück. Wenn es sich um eine 2-dimensionale Geometrie handelt wird der 2-dimensionale Umfang zurückgegeben. This function supports 3d and will not drop the z-index. This method implements the SQL/MM specification. SQL-MM ISO/IEC 13249-3: 8.1, 10.5 Änderung: 2.0.0 In Vorgängerversionen als ST_Perimeter3D bezeichet.

- ST_3DUnion - Perform 3D union. Beschreibung Verfügbarkeit: 2.2.0 Availability: 3.3.0 aggregate variant was added This method needs SFCGAL backend. This method implements the SQL/MM specification. SQL-MM IEC 13249-3: 5.1 This function supports 3d and will not drop the z-index. This function supports Polyhedral surfaces. This function supports Triangles and Triangulated Irregular Network Surfaces (TIN). Aggregate variant: returns a geometry that is the 3D union of a rowset of geometries. The ST_3DUnion() function is an "aggregate" function in the terminology of PostgreSQL. That means that it operates on rows of data, in the same way the SUM() and AVG() functions do and like most aggregates, it also ignores NULL geometries.

- ST_AddEdgeModFace - Fügt eine Kante hinzu. Falls dabei eine Masche aufgetrennt wird, so wird die ursprüngliche Masche angepasst und eine weitere Masche hinzugefügt. Beschreibung Fügt eine Kante hinzu. Falls dabei eine Masche aufgetrennt wird, so wird die ursprüngliche Masche angepasst und eine weitere Masche hinzugefügt. Wenn möglich, wird die neue Masche auf der linken Seite der neuen Kante erstellt. Dies ist jedoch nicht möglich, wenn die Masche auf der linken Seite die (unbegrenzte) Grundmenge der Maschen darstellt. Gibt die id der hinzugefügten Kante zurück. Führt ein entsprechendes Update auf alle verbundenen Kanten und Beziehungen durch. Wenn irgendwelche Übergabewerte NULL sind, die angegebenen Knoten unbekannt sind (müssen bereits in der node Tabelle des Schemas "topology" existieren), acurve kein LINESTRING ist, oder anode und anothernode nicht die Anfangs- und Endpunkte von acurve sind, dann wird eine Fehlermeldung ausgegeben. Wenn das Koordinatenreferenzsystem (SRID) der Geometrie acurve nicht mit jener der Topologie übereinstimmt, wird eine Fehlermeldung ausgegeben. Verfügbarkeit: 2.0 This method implements the SQL/MM specification. SQL-MM: Topo-Geo and Topo-Net 3: Routine Details: X.3.13

- ST_AddEdgeNewFaces - Fügt eine Kante hinzu. Falls dabei eine Masche aufgetrennt wird, so wird die ursprüngliche Masche gelöscht und durch zwei neue Maschen ersetzt. Beschreibung Fügt eine Kante hinzu. Falls dabei eine Masche aufgetrennt wird, so wird die ursprüngliche Masche gelöscht und durch zwei neue Maschen ersetzt. Gibt die ID der hinzugefügten Kante aus. Führt ein entsprechendes Update auf alle verbundenen Kanten und Beziehungen durch. Wenn irgendwelche Übergabewerte NULL sind, die angegebenen Knoten unbekannt sind (müssen bereits in der node Tabelle des Schemas "topology" existieren), acurve kein LINESTRING ist, oder anode und anothernode nicht die Anfangs- und Endpunkte von acurve sind, dann wird eine Fehlermeldung ausgegeben. Wenn das Koordinatenreferenzsystem (SRID) der Geometrie acurve nicht mit jener der Topologie übereinstimmt, wird eine Fehlermeldung ausgegeben. Verfügbarkeit: 2.0 This method implements the SQL/MM specification. SQL-MM: Topo-Geo and Topo-Net 3: Routine Details: X.3.12

- ST_AddIsoEdge - Fügt eine isolierte Kante, die durch die Geometrie alinestring festgelegt wird zu einer Topologie hinzu, indem zwei bestehende isolierte Knoten anode und anothernode verbunden werden. Gibt die "edgeid" der neuen Kante aus. Beschreibung Fügt eine isolierte Kante, die durch die Geometrie alinestring festgelegt wird zu einer Topologie hinzu, indem zwei bestehende isolierte Knoten anode und anothernode verbunden werden. Gibt die "edgeid" der neuen Kante aus. Wenn das Koordinatenreferenzsystem (SRID) der Geometrie alinestring nicht mit dem der Topologie übereinstimmt, irgendein Eingabewert NULL ist, die Knoten in mehreren Maschen enthalten sind, oder die Knoten Anfangs- oder Endknoten einer bestehenden Kante darstellen, wird eine Fehlermeldung ausgegeben. Wenn alinestring nicht innerhalb der Masche liegt zu der anode und anothernode gehören, dann wird eine Fehlermeldung ausgegeben. Wenn anode und anothernode nicht Anfangs- und Endpunkt von alinestring sind, wird eine Fehlermeldung ausgegeben. Availability: 1.1 This method implements the SQL/MM specification. SQL-MM: Topo-Geo and Topo-Net 3: Routine Details: X.3.4

- ST_AddIsoNode - Fügt einen isolierten Knoten zu einer Masche in einer Topologie hinzu und gibt die "nodeid" des neuen Knotens aus. Falls die Masche NULL ist, wird der Knoten dennoch erstellt. Beschreibung Fügt einen isolierten Knoten mit der Punktlage apoint zu einer bestehenden Masche mit der "faceid" aface zu einer Topologie atopology und gibt die "nodeid" des neuen Knoten aus. Wenn das Koordinatenreferenzsystem (SRID) der Punktgeometrie nicht mit dem der Topologie übereinstimmt, apoint keine Punktgeometrie ist, der Punkt NULL ist, oder der Punkt eine bestehende Kante (auch an den Begrenzungen) schneidet, wird eine Fehlermeldung ausgegeben. Falls der Punkt bereits als Knoten existiert, wird ebenfalls eine

Fehlermeldung ausgegeben. Wenn aface nicht NULL ist und apoint nicht innerhalb der Masche liegt, wird eine Fehlermeldung ausgegeben. Availability: 1.1 This method implements the SQL/MM specification. SQL-MM: Topo-Net Routines: X+1.3.1

- ST_Area - Gibt den geometrischen Schwerpunkt einer Geometrie zurück. Beschreibung Gibt den Flächeninhalt von Polygonen und Mehrfachpolygonen zurück. Gibt den Flächeninhalt der Datentypen "ST_Surface" und "ST_MultiSurface" zurück. Beim geometrischen Datentyp wird die kartesische 2D-Fläche ermittelt und in den Einheiten des SRID ausgegeben. Beim geographischen Datentyp wird die Fläche standardmäßig auf einem Referenzellipsoid ermittelt und in Quadratmeter ausgegeben. Mit ST_Area(geog,false) kann der Flächeninhalt auf einer Kugel ermittelt werden; dies ist zwar schneller aber auch weniger genau. Erweiterung: Mit 2.0.0 wurde 2D-Unterstützung für polyedrische Oberflächen eingeführt. Erweiterung: 2.2.0 - die Messung auf dem Referenzellipsoid wird mit der Bibliothek "GeographicLib" durchgeführt. Dadurch wurde die Genauigkeit und die Robustheit erhöht. Um die Vorteile dieser neuen Funktionalität zu nutzen, benötigen Sie Proj >= 4.9.0. Changed: 3.0.0 - does not depend on SFCGAL anymore. This method implements the OGC Simple Features Implementation Specification for SQL 1.1. This method implements the SQL/MM specification. SQL-MM 3: 8.1.2, 9.5.3 This function supports Polyhedral surfaces. Bei polyedrischen Oberflächen wird nur 2D (nicht 2.5D) unterstützt. Bei 2.5D kann ein Ergebnis ungleich null geliefert werden, wenn die Oberflächen vollständig in der XY-Ebene liegen.

- ST_AsBinary - Return the OGC/ISO Well-Known Binary (WKB) representation of the geometry/geography without SRID meta data. Beschreibung Returns the OGC/ISO Well-Known Binary (WKB) representation of the geometry. The first function variant defaults to encoding using server machine endian. The second function variant takes a text argument specifying the endian encoding, either little-endian ('NDR') or big-endian ('XDR'). WKB format is useful to read geometry data from the database and maintaining full numeric precision. This avoids the precision rounding that can happen with text formats such as WKT. To perform the inverse conversion of WKB to PostGIS geometry use . The OGC/ISO WKB format does not include the SRID. To get the EWKB format which does include the SRID use The default behavior in PostgreSQL 9.0 has been changed to output bytea in hex encoding. If your GUI tools require the old behavior, then SET bytea_output='escape' in your database. Erweiterung: 2.0.0 - Unterstützung für polyedrische Oberflächen, Dreiecke und TIN eingeführt. Erweiterung: 2.0.0 - Unterstützung für höherdimensionale Koordinatensysteme eingeführt. Erweiterung: 2.0.0 Unterstützung zum Festlegen des Endian beim geographischen Datentyp eingeführt. Verfügbarkeit: 1.5.0 Unterstützung von geograpischen Koordinaten. Änderung: 2.0.0 - Eingabewerte für diese Funktion dürfen nicht "unknown" sein -- es muss sich um eine Geometrie handeln. Konstrukte, wie ST_AsBinary('POINT(1 2)'), sind nicht länger gültig und geben folgende Fehlermeldung aus: n st_asbinary(unknown) is not unique error. Dieser Code muss in ST_AsBinary('POINT(1 2)'::geometry); geändert werden. Falls dies nicht möglich ist, so installieren Sie bitte legacy.sql. This method implements the OGC Simple Features Implementation Specification for SQL 1.1. s2.1.1.1 This method implements the SQL/MM specification. SQL-MM 3: 5.1.37 This method supports Circular Strings and Curves. This function supports Polyhedral surfaces. This function supports Triangles and Triangulated Irregular Network Surfaces (TIN). This function supports 3d and will not drop the z-index.

- ST_AsGML - Gibt die Geometrie als GML-Element - Version 2 oder 3 - zurück. Beschreibung Gibt die Geometrie als ein Geography Markup Language (GML) Element zurück. Ein Versionsparameter kann mit 2 oder 3 angegeben werden. Wenn kein Versionsparameter angegeben ist, wird dieser standardmäßig Version 2 angenommen. Der Parameter maxdecimaldigits kann verwendet werden, um die Anzahl der Nachkommastellen bei der Ausgabe zu reduzieren (standardmäßig 15). Using the maxdecimaldigits parameter can cause output geometry to become invalid. To avoid this use with a suitable gridsize first. GML 2 verweist auf Version 2.1.2, GML 3 auf Version 3.1.1 Der Übergabewert "options" ist ein Bitfeld. Es kann verwendet werden um das Koordinatenreferenzsystem bei der GML Ausgabe zu bestimmen und um die Daten in Länge/Breite anzugeben. 0: GML Kurzform für das CRS (z.B. EPSG:4326), Standardwert 1: GML Langform für das CRS (z.B. urn:ogc:def:crs:EPSG::4326) 2: Nur für GML 3, entfernt das srsDimension Attribut von der Ausgabe. 4: Nur für GML 3, Für Linien verwenden Sie bitte den Tag <LineString> anstatt <Curve>. 16: Deklarieren, dass die Daten in Breite/Länge (z.B. SRID=4326) vorliegen. Standardmäßig wird angenommen, dass die Daten planar sind. Diese Option ist nur bei Ausgabe in GML 3.1.1, in Bezug auf die Anordnung der Achsen sinnvoll. Falls Sie diese setzen, werden die Koordinaten von Länge/Breite auf Breite/Länge vertauscht. 32: Ausgabe der BBox der Geometrie (Umhüllende/Envelope). Der Übergabewert 'namespace prefix' kann verwendet werden, um ein benutzerdefiniertes Präfix für den Namensraum anzugeben, oder kein Präfix (wenn leer). Wenn Null oder weggelassen, so wird das Präfix "gml" verwendet. Verfügbarkeit: 1.3.2 Verfügbarkeit: 1.5.0 Unterstützung von geograpischen Koordinaten. Erweiterung: 2.0.0 Unterstützung durch Präfix eingeführt. Für GML3 wurde die Option 4 eingeführt, um die Verwendung von LineString anstatt von Kurven für Linien zu erlauben. Ebenfalls wurde die GML3 Unterstützung für polyedrische Oberflächen und TINS eingeführt, sowie die Option 32 zur Ausgabe der BBox. Änderung: 2.0.0 verwendet standardmäßig benannte Argumente. Erweiterung: 2.1.0 Für GML 3 wurde die Unterstützung einer ID eingeführt. Nur die Version 3+ von ST_AsGML unterstützt polyedrische Oberflächen und TINs. This method implements the SQL/MM specification. SQL-MM IEC 13249-3: 17.2 This function supports 3d and will not drop the z-index. This function supports Polyhedral surfaces. This function supports Triangles and Triangulated Irregular Network Surfaces (TIN).

- ST_AsText - Gibt die Well-known-Text(WKT) Darstellung der Geometrie/Geographie ohne die SRID Metadaten zurück. Beschreibung Returns the OGC Well-Known Text (WKT) representation of the geometry/geography. The optional maxdecimaldigits argument may be used to limit the number of digits after the decimal point in output ordinates (defaults to 15). To perform the inverse conversion of WKT representation to PostGIS geometry use . The standard OGC WKT representation does not include the SRID. To include the SRID as part of the output representation, use the non-standard PostGIS function The textual representation of numbers in WKT may not maintain full floating-point precision. To ensure full accuracy for data storage or transport it is best to use Well-Known Binary (WKB) format (see and maxdecimaldigits). Using the maxdecimaldigits parameter can cause output geometry to become invalid. To avoid this use with a suitable gridsize first. Verfügbarkeit: 1.5 - Unterstützung von geograpischen Koordinaten. Erweiterung: 2.5 - der optionale Parameter "precision" wurde eingeführt. This method implements the OGC Simple Features Implementation Specification for SQL 1.1. s2.1.1.1 This method implements the SQL/MM specification. SQL-MM 3: 5.1.25 This method supports Circular Strings and Curves.

- ST_Boundary - Gibt die abgeschlossene Hülle aus der kombinierten Begrenzung der Geometrie zurück. Beschreibung Gibt die abgeschlossene Hülle aus der kombinierten Begrenzung der Geometrie zurück. Die Definition der kombinierte Begrenzung ist in Abschnitt 3.12.3.2 der OGC SPEC beschrieben. Da das Ergebnis dieser Funktion eine abgeschlossene Hülle und daher topologisch geschlossen ist, kann die resultierende Begrenzung durch geometrische Primitive, wie in Abschnitt 3.12.2. der OGC SPEC erörtert, dargestellt werden. Wird durch das GEOS Modul ausgeführt Vor 2.0.0 meldete diese Funktion einen Fehler, falls sie auf eine GEOMETRYCOLLECTION angewandt wurde. Ab 2.0.0 wird stattdessen NULL (nicht unterstützte Eingabe) zurückgegeben. This method implements the OGC Simple Features Implementation Specification for SQL 1.1. OGC SPEC s2.1.1.1 This method implements the SQL/MM specification. SQL-MM IEC 13249-3: 5.1.17 This function supports 3d and will not drop the z-index. Erweiterung: mit 2.1.0 wurde die Unterstützung von Dreiecken eingeführt Changed: 3.2.0 support for TIN, does not use geos, does not linearize curves

- ST_Buffer - Computes a geometry covering all points within a given distance from a geometry. Beschreibung Computes a POLYGON or MULTIPOLYGON that represents all points whose distance from a geometry/geography is less than or equal to a given distance. A negative distance shrinks the geometry rather than expanding it. A negative distance may shrink a polygon completely, in which case POLYGON EMPTY is returned. For points and lines negative distances always return empty results. For geometry, the distance is specified in the units of the Spatial Reference System of the geometry. For geography, the distance is specified in meters. The optional third parameter controls the buffer accuracy and style. The accuracy of circular arcs in the buffer is specified as the number of line segments used to approximate a quarter circle (default is 8). The buffer style can be specifed by providing a list of blank-separated key=value pairs as follows: 'quad_segs=#' : number of line segments used to approximate a quarter circle (default is 8). 'endcap=round|flat|square' : endcap style (defaults to "round"). 'butt' is accepted as a synonym for 'flat'. 'join=round|mitre|bevel' : join style (defaults to "round"). 'miter' is accepted as a synonym for 'mitre'. 'mitre_limit=#.#' : mitre ratio limit (only affects mitered join style). 'miter_limit' is accepted as a synonym for 'mitre_limit'. 'side=both|left|right' : 'left' or 'right' performs a single-sided buffer on the geometry, with the buffered side relative to the direction of the line. This is only applicable to LINESTRING geometry and does not affect POINT or POLYGON geometries. By default end caps are square. For geography this is a thin wrapper around the geometry implementation. It determines a planar spatial reference system that best fits the bounding box of the geography object (trying UTM, Lambert Azimuthal Equal Area (LAEA) North/South pole, and finally Mercator). The buffer is computed in the planar space, and then transformed back to WGS84. This may not produce the desired behavior if the input object is much larger than a UTM zone or crosses the dateline Buffer output is always a valid polygonal geometry. Buffer can handle invalid inputs, so buffering by distance 0 is sometimes used as a way of repairing invalid polygons. can also be used for this purpose. Buffering is sometimes used to perform a within-distance search. For this use case it is more efficient to use . This function ignores the Z dimension. It always gives a 2D result even when used on a 3D geometry. Erweiterung: 2.5.0 - ST_Buffer ermöglicht jetzt auch eine seitliche Pufferzonenberechnung über side=both|left|right. Verfügbarkeit: 1.5 - ST_Buffer wurde um die Unterstützung von Abschlusstücken/endcaps und Join-Typen erweitert. Diese können zum Beispiel dazu verwendet werden, um Linienzüge von Straßen in Straßenpolygone mit flachen oder rechtwinkeligen Abschlüssen anstatt mit runden Enden umzuwandeln. Ein schlanker Adapter für den geographischen Datentyp wurde hinzugefügt. Wird vom GEOS Modul ausgeführt This method implements the OGC Simple Features Implementation Specification for SQL 1.1. s2.1.1.3 This method implements the SQL/MM specification. SQL-MM IEC 13249-3: 5.1.30

- ST_Centroid - Gibt den geometrischen Schwerpunkt einer Geometrie zurück. Beschreibung Computes a point which is the geometric center of mass of a geometry. For [MULTI]POINTs, the centroid is the arithmetic mean of the input coordinates. For [MULTI]LINESTRINGs, the centroid is computed using the weighted length of each line segment. For [MULTI]POLYGONs, the centroid is computed in terms of area. If an empty geometry is supplied, an empty GEOMETRYCOLLECTION is returned. If NULL is supplied, NULL is returned. If CIRCULARSTRING or COMPOUNDCURVE are supplied, they are converted to linestring with CurveToLine first, then same than for LINESTRING For mixed-dimension input, the result is equal to the centroid of the component Geometries of highest dimension (since the lower-dimension geometries contribute zero "weight"

to the centroid). Note that for polygonal geometries the centroid does not necessarily lie in the interior of the polygon. For example, see the diagram below of the centroid of a C-shaped polygon. To construct a point guaranteed to lie in the interior of a polygon use . New in 2.3.0 : supports CIRCULARSTRING and COMPOUNDCURVE (using CurveToLine) Verfügbarkeit: Mit 2.4.0 wurde die Unterstützung für den geograpischen Datentyp eingeführt. This method implements the OGC Simple Features Implementation Specification for SQL 1.1. This method implements the SQL/MM specification. SQL-MM 3: 8.1.4, 9.5.5

- ST_ChangeEdgeGeom - Ändert die geometrische Form einer Kante, ohne sich auf die topologische Struktur auszuwirken. Beschreibung Ändert die geometrische Form der Kante, ohne sich auf topologische Struktur auszuwirken. If any arguments are null, the given edge does not exist in the edge table of the topology schema, the acurve is not a LINESTRING, or the modification would change the underlying topology then an error is thrown. Wenn das Koordinatenreferenzsystem (SRID) der Geometrie acurve nicht mit jener der Topologie übereinstimmt, wird eine Fehlermeldung ausgegeben. Wenn die neue acurve nicht "simple" ist, wird eine Fehlermeldung ausgegeben. Wenn beim Verschieben der Kante von der alten auf die neue Position ein Hindernis auftritt, wird eine Fehlermeldung ausgegeben. Verfügbarkeit: 1.1.0 Erweiterung: 2.0.0 Erzwingung topologischer Konsistenz hinzugefügt This method implements the SQL/MM specification. SQL-MM: Topo-Geo and Topo-Net 3: Routine Details X.3.6

- ST_Contains - Tests if every point of B lies in A, and their interiors have a point in common Beschreibung Returns TRUE if geometry A contains geometry B. A contains B if and only if all points of B lie inside (i.e. in the interior or boundary of) A (or equivalently, no points of B lie in the exterior of A), and the interiors of A and B have at least one point in common. In mathematical terms: ST_Contains(A, B) $\Leftrightarrow$ (A $\bigcap$ B = B) $\wedge$ (Int(A) $\bigcap$ Int(B) $\neq \varnothing$) The contains relationship is reflexive: every geometry contains itself. (In contrast, in the predicate a geometry does not properly contain itself.) The relationship is antisymmetric: if ST_Contains(A,B) = true and ST_Contains(B,A) = true, then the two geometries must be topologically equal (ST_Equals(A,B) = true). ST_Contains is the converse of . So, ST_Contains(A,B) = ST_Within(B,A). Because the interiors must have a common point, a subtlety of the definition is that polygons and lines do not contain lines and points lying fully in their boundary. For further details see Subtleties of OGC Covers, Contains, Within. The predicate provides a more inclusive relationship. This function automatically includes a bounding box comparison that makes use of any spatial indexes that are available on the geometries. Um die Verwendung eines Indices zu vermeiden, kann die Funktion _ST_Contains verwendet werden. Wird durch das GEOS Modul ausgeführt Enhanced: 2.3.0 Enhancement to PIP short-circuit extended to support MultiPoints with few points. Prior versions only supported point in polygon. Enhanced: 3.0.0 enabled support for GEOMETRYCOLLECTION Do not use this function with invalid geometries. You will get unexpected results. NOTE: this is the "allowable" version that returns a boolean, not an integer. This method implements the OGC Simple Features Implementation Specification for SQL 1.1. s2.1.1.2 // s2.1.13.3 - same as within(geometry B, geometry A) This method implements the SQL/MM specification. SQL-MM 3: 5.1.31

- ST_ConvexHull - Berechnet die konvexe Hülle einer Geometrie. Beschreibung Berechnet die konvexe Hülle einer Geometrie. Die konvexe Hülle stellt die kleinste konvexe Geometrie dar, welche die gesamte Geometrie einschließt. One can think of the convex hull as the geometry obtained by wrapping an rubber band around a set of geometries. This is different from a concave hull which is analogous to "shrink-wrapping" the geometries. A convex hull is often used to determine an affected area based on a set of point observations. In the general case the convex hull is a Polygon. The convex hull of two or more collinear points is a two-point LineString. The convex hull of one or more identical points is a Point. This is not an aggregate function. To compute the convex hull of a set of geometries, use to aggregate them into a geometry collection (e.g. ST_ConvexHull(ST_Collect(geom)). Wird durch das GEOS Modul ausgeführt This method implements the OGC Simple Features Implementation Specification for SQL 1.1. s2.1.1.3 This method implements the SQL/MM specification. SQL-MM IEC 13249-3: 5.1.16 This function supports 3d and will not drop the z-index.

- ST_CoordDim - Gibt die Dimension der Koordinaten für den Wert von ST_Geometry zurück. Beschreibung Gibt die Dimension der Koordinaten für den Wert von ST_Geometry zurück. Dies ist der MM konforme Alias für This method implements the OGC Simple Features Implementation Specification for SQL 1.1. This method implements the SQL/MM specification. SQL-MM 3: 5.1.3 This method supports Circular Strings and Curves. This function supports 3d and will not drop the z-index. This function supports Polyhedral surfaces. This function supports Triangles and Triangulated Irregular Network Surfaces (TIN).

- ST_CreateTopoGeo - Fügt eine Sammlung von Geometrien an eine leere Topologie an und gibt eine Bestätigungsmeldung aus. Beschreibung Fügt eine Sammelgeometrie einer leeren Topologie hinzu und gibt eine Bestätigungsmeldung aus. Nützlich um eine leere Topologie zu befüllen. Verfügbarkeit: 2.0 This method implements the SQL/MM specification. SQL-MM: Topo-Geo and Topo-Net 3: Routine Details -- X.3.18

- ST_Crosses - Tests if two geometries have some, but not all, interior points in common Beschreibung Compares two geometry objects and returns true if their intersection "spatially crosses"; that is, the geometries have some, but not all interior points in common. The intersection of the interiors of the geometries must be non-empty and must have dimension less than the maximum dimension of the two input geometries, and the intersection of the two geometries must not equal either geometry. Otherwise, it returns false. The crosses relation is symmetric and irreflexive. In mathematical terms: ST_Crosses(A, B) $\Leftrightarrow$ (dim(Int(A) $\bigcap$ Int(B)) < max(dim(Int(A)), dim(Int(B)))) $\wedge$ (A $\bigcap$ B $\neq$ A) $\wedge$ (A $\bigcap$ B $\neq$ B) Geometries cross if their DE-9IM Intersection Matrix matches: T*T****** for Point/Line, Point/Area, and Line/Area situations T*****T** for Line/Point, Area/Point, and Area/Line situations 0******** for Line/Line situations the result is false for Point/Point and Area/Area situations The OpenGIS Simple Features Specification defines this predicate only for Point/Line, Point/Area, Line/Line, and Line/Area situations. JTS / GEOS extends the definition to apply to Line/Point, Area/Point and Area/Line situations as well. This makes the relation symmetric. This function automatically includes a bounding box comparison that makes use of any spatial indexes that are available on the geometries. Enhanced: 3.0.0 enabled support for GEOMETRYCOLLECTION This method implements the OGC Simple Features Implementation Specification for SQL 1.1. s2.1.13.3 This method implements the SQL/MM specification. SQL-MM 3: 5.1.29

- ST_CurveToLine - Converts a geometry containing curves to a linear geometry. Beschreibung Konvertiert einen CIRCULAR STRING in einen normalen LINESTRING, ein CURVEPOLYGON in ein POLYGON und ein MULTISURFACE in ein MULTIPOLYGON. Ist nützlich zur Ausgabe an Geräten, die keine Kreisbögen unterstützen. Wandelt eine gegebene Geometrie in eine lineare Geometrie um. Jede Kurvengeometrie und jedes Kurvensegment wird in linearer Näherung mit der gegebenen Toleranz und Optionen konvertiert (Standardmäßig 32 Segmenten pro Viertelkreis und keine Optionen). Der Übergabewert 'tolerance_type' gibt den Toleranztyp an. Er kann die folgenden Werte annehmen: 0 (default): die Toleranz wird über die maximale Anzahl der Segmente pro Viertelkreis angegeben. 1: Die Toleranz wird als maximale Abweichung der Linie von der Kurve in der Einheit der Herkunftsdaten angegeben. 2: Die Toleranz entspricht dem maximalen Winkel zwischen zwei erzeugten Radien. Der Parameter 'flags' ist ein Bitfeld mit dem Standardwert 0. Es werden folgende Bits unterstützt: 1: Symmetrische (orientierungsunabhängige) Ausgabe. 2: Erhält den Winkel, vermeidet die Winkel (Segmentlängen) bei der symmetrischen Ausgabe zu reduzieren. Hat keine Auswirkung, wenn die Symmetrie-Flag nicht aktiviert ist. Verfügbarkeit: 1.3.0 Erweiterung: ab 2.4.0 kann die Toleranz über die 'maximale Abweichung' und den 'maximalen Winkel' angegeben werden. Die symmetrische Ausgabe wurde hinzugefügt. Erweiterung: 3.0.0 führte eine minimale Anzahl an Segmenten pro linearisierten Bogen ein, um einem topologischen Kollaps vorzubeugen. This method implements the OGC Simple Features Implementation Specification for SQL 1.1. This method implements the SQL/MM specification. SQL-MM 3: 7.1.7 This function supports 3d and will not drop the z-index. This method supports Circular Strings and Curves.

- ST_Difference - Computes a geometry representing the part of geometry A that does not intersect geometry B. Beschreibung Returns a geometry representing the part of geometry A that does not intersect geometry B. This is equivalent to A - ST_Intersection(A,B). If A is completely contained in B then an empty atomic geometry of appropriate type is returned. This is the only overlay function where input order matters. ST_Difference(A, B) always returns a portion of A. If the optional gridSize argument is provided, the inputs are snapped to a grid of the given size, and the result vertices are computed on that same grid. (Requires GEOS-3.9.0 or higher) Wird durch das GEOS Modul ausgeführt Enhanced: 3.1.0 accept a gridSize parameter. Requires GEOS >= 3.9.0 to use the gridSize parameter. This method implements the OGC Simple Features Implementation Specification for SQL 1.1. s2.1.1.3 This method implements the SQL/MM specification. SQL-MM 3: 5.1.20 This function supports 3d and will not drop the z-index. However, the result is computed using XY only. The result Z values are copied, averaged or interpolated.

- ST_Dimension - Gibt die Dimension der Koordinaten für den Wert von ST_Geometry zurück. Beschreibung Die inhärente Dimension eines geometrischen Objektes, welche kleiner oder gleich der Dimension der Koordinaten sein muss. Nach OGC SPEC s2.1.1.1 wird 0 für POINT, 1 für LINESTRING, 2 for POLYGON, und die größte Dimension der Teile einer GEOMETRYCOLLECTION zurückgegeben. Wenn die Dimension nicht bekannt ist (leereGEOMETRYCOLLECTION) wird 0 zurückgegeben. This method implements the SQL/MM specification. SQL-MM 3: 5.1.2 Erweiterung: 2.0.0 - Unterstützung für polyedrische Oberflächen und TIN eingeführt. Vor 2.0.0 meldete diese Funktion einen Fehler, falls sie auf eine leere Geometrie angewandt wurde. This function supports Polyhedral surfaces. This function supports Triangles and Triangulated Irregular Network Surfaces (TIN).

- ST_Disjoint - Tests if two geometries have no points in common Beschreibung Returns true if two geometries are disjoint. Geometries are disjoint if they have no point in common. If any other spatial relationship is true for a pair of geometries, they are not disjoint. Disjoint implies that is false. In mathematical terms: ST_Disjoint(A, B) $\Leftrightarrow$ A $\bigcap$ B = $\varnothing$ Enhanced: 3.0.0 enabled support for GEOMETRYCOLLECTION Wird durch das GEOS Modul ausgeführt This function call does not use indexes. A negated predicate can be used as a more performant alternative that uses indexes: ST_Disjoint(A,B) = NOT ST_Intersects(A,B) NOTE: this is the "allowable" version that returns a boolean, not an integer. This method implements the

OGC Simple Features Implementation Specification for SQL 1.1. s2.1.1.2 //s2.1.13.3 - a.Relate(b, 'FF*FF****') This method implements the SQL/MM specification. SQL-MM 3: 5.1.26

- ST_Distance - Gibt die größte 3-dimensionale Distanz zwischen zwei geometrischen Objekten als Linie zurück Beschreibung Für den geometrischen Datentyp. Es wird der geringste 3-dimensionale kartesische Abstand zwischen zwei geometrischen Objekten in projizierten Einheiten (Einheiten des Koordinatenreferenzsystem) zurückgegeben. Beim geometrischen Datentyp wird die geringste kartesische Distanz in 2D zwischen zwei geometrischen Objekten - in projizierten Einheiten (Einheiten des Koordinatenreferenzsystem) - zurückgegeben. Beim geographischen Datentyp wird standardmäßig die geringste geodätische Distanz zwischen zwei geographischen Objekten in Meter zurückgegeben. Wenn use_speroid FALSE ist, erfolgt eine schnellere Berechnung auf einer Kugel anstatt auf dem Referenzellipsoid. This method implements the OGC Simple Features Implementation Specification for SQL 1.1. This method implements the SQL/MM specification. SQL-MM 3: 5.1.23 This method supports Circular Strings and Curves. Verfügbarkeit: 1.5.0 die Unterstützung des geograpischen Datentyps wurde eingeführt. Geschwindigkeitsverbesserungen bei einer umfangreichen Geometrie und bei einer Geometrie mit vielen Knoten Enhanced: 2.1.0 Geschwindigkeitsverbesserung beim geographischen Datentyp. Siehe Making Geography faster für Details. Erweiterung: 2.1.0 - Unterstützung für Kurven beim geometrischen Datentyp eingeführt. Erweiterung: 2.2.0 - die Messung auf dem Referenzellipsoid wird mit der Bibliothek "GeographicLib" durchgeführt. Dadurch wurde die Genauigkeit und die Robustheit erhöht. Um die Vorteile dieser neuen Funktionalität zu nutzen, benötigen Sie Proj >= 4.9.0. Changed: 3.0.0 - does not depend on SFCGAL anymore.

- ST_EndPoint - Gibt die Anzahl der Stützpunkte eines ST_LineString oder eines ST_CircularString zurück. Beschreibung Gibt den Anfangspunkt einer LINESTRING oder CIRCULARLINESTRING Geometrie als POINT oder NULL zurück, falls es sich beim Eingabewert nicht um einen LINESTRING oder CIRCULARLINESTRING handelt. This method implements the SQL/MM specification. SQL-MM 3: 7.1.4 This function supports 3d and will not drop the z-index. This method supports Circular Strings and Curves. Änderung: 2.0.0 unterstützt die Verarbeitung von MultiLinestring's die nur aus einer einzelnen Geometrie bestehen, nicht mehr. In früheren Versionen von PostGIS gab die Funktion bei einem aus einer einzelnen Linie bestehender MultiLinestring den Anfangspunkt zurück. Ab 2.0.0 gibt sie nur NULL zurück, so wie bei jedem anderen MultiLinestring. Die alte Verhaltensweise war undokumentiert, aber Anwender, die annahmen, dass Sie Ihre Daten als LINESTRING vorliegen haben, könnten in 2.0 dieses zurückgegebene NULL bemerken.

- ST_Envelope - Gibt eine Geometrie in doppelter Genauigkeit (float8) zurück, welche das Umgebungsrechteck der beigestellten Geometrie darstellt. Beschreibung Gibt das kleinstmögliche Umgebungsrechteck der bereitgestellten Geometrie als Geometrie im Float8-Format zurück. Das Polygon wird durch die Eckpunkte des Umgebungsrechteckes beschrieben ((MINX, MINY), (MINX, MAXY), (MAXX, MAXY), (MAXX, MINY), (MINX, MINY)). (PostGIS fügt auch die ZMIN/ZMAX Koordinaten hinzu). Spezialfälle (vertikale Linien, Punkte) geben eine Geometrie geringerer Dimension zurück als POLYGON, insbesondere POINT oder LINESTRING. Verfügbarkeit: 1.5.0 Änderung der Verhaltensweise insofern, das die Ausgabe in Double Precision anstelle von Float4 erfolgt This method implements the OGC Simple Features Implementation Specification for SQL 1.1. s2.1.1.1 This method implements the SQL/MM specification. SQL-MM 3: 5.1.19

- ST_Equals - Tests if two geometries include the same set of points Beschreibung Returns true if the given geometries are "topologically equal". Use this for a 'better' answer than '='. Topological equality means that the geometries have the same dimension, and their point-sets occupy the same space. This means that the order of vertices may be different in topologically equal geometries. To verify the order of points is consistent use (it must be noted ST_OrderingEquals is a little more stringent than simply verifying order of points are the same). In mathematical terms: ST_Equals(A, B) $\Leftrightarrow$ A = B The following relation holds: ST_Equals(A, B) $\Leftrightarrow$ ST_Within(A,B) $\wedge$ ST_Within(B,A) Enhanced: 3.0.0 enabled support for GEOMETRYCOLLECTION This method implements the OGC Simple Features Implementation Specification for SQL 1.1. s2.1.1.2 This method implements the SQL/MM specification. SQL-MM 3: 5.1.24 Changed: 2.2.0 Returns true even for invalid geometries if they are binary equal

- ST_ExteriorRing - Gibt die Anzahl der inneren Ringe einer Polygongeometrie aus. Beschreibung Gibt einen Linienzug zurück, welcher den äußeren Ring der POLYGON Geometrie darstellt. Gibt NULL zurück wenn es sich bei der Geometrie um kein Polygon handelt. Dies funktioniert nicht mit MULTIPOLYGONen. Verwenden Sie die Funktion bitte in Zusammenhang mit ST_Dump um sie auf MULTIPOLYGONe anzuwenden. This method implements the OGC Simple Features Implementation Specification for SQL 1.1. 2.1.5.1 This method implements the SQL/MM specification. SQL-MM 3: 8.2.3, 8.3.3 This function supports 3d and will not drop the z-index.

- ST_GMLToSQL - Gibt einen spezifizierten ST_Geometry Wert aus einer GML-Darstellung zurück. Dies ist ein Aliasname für ST_GeomFromGML Beschreibung This method implements the SQL/MM specification. SQL-MM 3: 5.1.50 (ausgenommen Unterstützung von Kurven). Verfügbarkeit: 1.5, benötigt libxml2 1.6+ Erweiterung: Mit 2.0.0 wurde die Unterstützung für polyedrische Oberflächen und TIN eingeführt. Erweiterung: 2.0.0 Standardwert für den optionalen Parameter SRID eingefügt.

- ST_GeomCollFromText - Erzeugt eine Sammelgeometrie mit der gegebenen SRID aus einer WKT-Kollektion. Wenn keine SRID angegeben ist, wird diese standardmäßig auf 0 gesetzt. Beschreibung Erzeugt eine Sammelgeometrie mit der gegebenen SRID aus einer Well-known-Text (WKT) Darstellung. Wenn keine SRID angegeben ist, wird diese standardmäßig auf 0 gesetzt. OGC SPEC 3.2.6.2 - Die Option SRID stammt aus dem Konformitätstest Gibt NULL zurück, wenn der WKT keine GEOMETRYCOLLECTION ist Verwenden Sie diese Funktion nicht, wenn Sie sich vollkommen sicher sind, dass ihre WKT Geometrie eine Sammelgeometrie ist. Sie ist langsamer als ST_GeomFromText, da sie einen zusätzlicher Validierungsschritt ausführt. This method implements the OGC Simple Features Implementation Specification for SQL 1.1. s3.2.6.2 This method implements the SQL/MM specification.

- ST_GeomFromText - Gibt einen spezifizierten ST_Geometry Wert aus einer Well-known-Text Darstellung (WKT) zurück. Beschreibung Erzeugt ein PostGIS ST_Geometry Objekt aus der OGC Well-known-Text Darstellung. Die Funktion ST_GeomFromTex hat zwei Varianten. Die erste Variante nimmt keine SRID entgegen und gibt eine Geometrie ohne ein bestimmtes Koordinatenreferenzsystem aus (SRID=0) . Die zweite Variante nimmt eine SRID als zweiten Übergabewert entgegen und gibt eine Geometrie zurück, die diese SRID als Teil ihrer Metadaten beinhaltet. This method implements the OGC Simple Features Implementation Specification for SQL 1.1. s3.2.6.2 - die Option SRID ist vom Konformitätstest. This method implements the SQL/MM specification. SQL-MM 3: 5.1.40 This method supports Circular Strings and Curves. While not OGC-compliant, is faster than ST_GeomFromText and ST_PointFromText. It is also easier to use for numeric coordinate values. is another option similar in speed to and is OGC-compliant, but doesn't support anything but 2D points. Änderung: 2.0.0 - In Vorgängerversionen von PostGIS war ST_GeomFromText('GEOMETRYCOLLECTION(EMPTY)') erlaubt. Um eine bessere Übereinstimmung mit der SQL/MM Norm zu erreichen, ist dies in PostGIS 2.0.0 nun nicht mehr gestattet. Hier sollte nun ST_GeomFromText('GEOMETRYCOLLECTION EMPTY') geschrieben werden.

- ST_GeomFromWKB - Erzeugt ein geometrisches Objekt aus der Well-known-Binary (WKB) Darstellung und einer optionalen SRID. Beschreibung Die Funktion ST_GeogFromWKB nimmt eine Well-known-Binary (WKB) Darstellung und eine Id für das Koordinatenreferenzsystem (SRID) entgegen und erzeugt eine Instanz des entsprechenden geometrischen Datentyps. Diese Funktion übernimmt die Rolle der Geometrie-Factory in SQL. Ist eine alternative Bezeichnung für ST_WKBToSQL. Wenn die SRID nicht festgelegt ist, wird sie standardmäßig auf 0 (Unknown) gesetzt. This method implements the OGC Simple Features Implementation Specification for SQL 1.1. s3.2.7.2 - die optionale SRID kommt vom Konformitätstest. This method implements the SQL/MM specification. SQL-MM 3: 5.1.41 This method supports Circular Strings and Curves.

- ST_GeometryFromText - Gibt einen spezifizierten ST_Geometry-Wert von einer Well-known-Text Darstellung (WKT) zurück. Die Bezeichnung ist ein Alias für ST_GeomFromText Beschreibung This method implements the OGC Simple Features Implementation Specification for SQL 1.1. This method implements the SQL/MM specification. SQL-MM 3: 5.1.40

- ST_GeometryN - Gibt den Geometrietyp des ST_Geometry Wertes zurück. Beschreibung Gibt die auf 1-basierende n-te Geometrie zurück, wenn es sich bei der Geometrie um eine GEOMETRYCOLLECTION, (MULTI)POINT, (MULTI)LINESTRING, MULTICURVE oder (MULTI)POLYGON, POLYHEDRALSURFACE handelt. Anderenfalls wird NULL zurückgegeben. Seit Version 0.8.0 basiert der Index auf 1, so wie in der OGC Spezifikation. Vorhergegangene Versionen waren 0-basiert. Falls Sie alle Geometrien einer Geometrie entnehmen wollen, so ist ST_Dump wesentlich leistungsfähiger und es funktioniert auch mit Einzelgeometrien. Erweiterung: Mit 2.0.0 wurde die Unterstützung für polyedrische Oberflächen, Dreiecke und TIN eingeführt. Änderung: 2.0.0 Vorangegangene Versionen geben bei Einzelgeometrien NULL zurück. Dies wurde geändert um die Geometrie für den ST_GeometrieN(..,1) Fall zurückzugeben. This method implements the OGC Simple Features Implementation Specification for SQL 1.1. This method implements the SQL/MM specification. SQL-MM 3: 9.1.5 This function supports 3d and will not drop the z-index. This method supports Circular Strings and Curves. This function supports Polyhedral surfaces. This function supports Triangles and Triangulated Irregular Network Surfaces (TIN).

- ST_GeometryType - Gibt den Geometrietyp des ST_Geometry Wertes zurück. Beschreibung Gibt den Geometrietyp als Zeichenkette zurück. Z.B.: 'ST_LineString', 'ST_Polygon','ST_MultiPolygon' etc. Diese Funktion unterscheidet sich von GeometryType(geometry) durch den Präfix ST_ und dadurch, das nicht angezeigt wird, ob die Geometrie eine Maßzahl besitzt. Erweiterung: Mit 2.0.0 wurde die Unterstützung für polyedrische Oberflächen eingeführt. This method implements the SQL/MM specification. SQL-MM 3: 5.1.4 This function supports 3d and will not drop the z-index. This function supports Polyhedral surfaces.

- ST_GetFaceEdges - Gibt die Kanten, die aface begrenzen, sortiert aus. Beschreibung Gibt die Kanten, die aface begrenzen, sortiert aus. Jede Ausgabe besteht aus einer Sequenz und einer "edgeid". Die Sequenzzahlen beginnen mit dem Wert 1. Die Aufzählung der Kanten des Rings beginnt mit der Kante mit dem niedrigsten Identifikator. Die Reihenfolge der Kanten folgt der Drei-Finger-Regel (die begrenzte Masche liegt links von den gerichteten Kanten). Verfügbarkeit: 2.0 This method implements the SQL/MM specification. SQL-MM 3 Topo-Geo and Topo-Net 3: Routine Details: X.3.5

- ST_GetFaceGeometry - Gibt für eine Topologie und eine bestimmte Maschen-ID das Polygon zurück. Beschreibung Gibt für eine Topologie und eine bestimmte Maschen-ID das Polygon zurück. Erstellt das Polygon aus den Kanten, die die Masche aufbauen. Availability: 1.1 This method implements the SQL/MM specification. SQL-MM 3 Topo-Geo and Topo-Net 3: Routine Details: X.3.16

- ST_InitTopoGeo - Creates a new topology schema and registers it in the topology.topology table. Beschreibung This is the SQL-MM equivalent of . It lacks options for spatial reference system and tolerance. it returns a text description of the topology creation, instead of the topology id. Availability: 1.1 This method implements the SQL/MM specification. SQL-MM 3 Topo-Geo and Topo-Net 3: Routine Details: X.3.17

- ST_InteriorRingN - Gibt die Anzahl der inneren Ringe einer Polygongeometrie aus. Beschreibung Gibt den Nten innenliegenden Linienzug des Ringes der Polygongeometrie zurück. Gibt NULL zurück, falls es sich bei der Geometrie nicht um ein Polygon handelt, oder sich das angegebene N außerhalb des zulässigen Bereiches befindet. Dies funktioniert nicht mit MULTIPOLYGONen. Verwenden Sie die Funktion bitte in Zusammenhang mit ST_Dump um sie auf MULTIPOLYGONe anzuwenden. This method implements the OGC Simple Features Implementation Specification for SQL 1.1. This method implements the SQL/MM specification. SQL-MM 3: 8.2.6, 8.3.5 This function supports 3d and will not drop the z-index.

- ST_Intersection - Computes a geometry representing the shared portion of geometries A and B. Beschreibung Returns a geometry representing the point-set intersection of two geometries. In other words, that portion of geometry A and geometry B that is shared between the two geometries. If the geometries have no points in common (i.e. are disjoint) then an empty atomic geometry of appropriate type is returned. If the optional gridSize argument is provided, the inputs are snapped to a grid of the given size, and the result vertices are computed on that same grid. (Requires GEOS-3.9.0 or higher) ST_Intersection in conjunction with is useful for clipping geometries such as in bounding box, buffer, or region queries where you only require the portion of a geometry that is inside a country or region of interest. For geography this is a thin wrapper around the geometry implementation. It first determines the best SRID that fits the bounding box of the 2 geography objects (if geography objects are within one half zone UTM but not same UTM will pick one of those) (favoring UTM or Lambert Azimuthal Equal Area (LAEA) north/south pole, and falling back on mercator in worst case scenario) and then intersection in that best fit planar spatial ref and retransforms back to WGS84 geography. This function will drop the M coordinate values if present. If working with 3D geometries, you may want to use SFGCAL based which does a proper 3D intersection for 3D geometries. Although this function works with Z-coordinate, it does an averaging of Z-Coordinate. Wird durch das GEOS Modul ausgeführt Enhanced: 3.1.0 accept a gridSize parameter Requires GEOS >= 3.9.0 to use the gridSize parameter Changed: 3.0.0 does not depend on SFCGAL. Availability: 1.5 support for geography data type was introduced. This method implements the OGC Simple Features Implementation Specification for SQL 1.1. s2.1.1.3 This method implements the SQL/MM specification. SQL-MM 3: 5.1.18 This function supports 3d and will not drop the z-index. However, the result is computed using XY only. The result Z values are copied, averaged or interpolated.

- ST_Intersects - Tests if two geometries intersect (they have at least one point in common) Beschreibung Returns true if two geometries intersect. Geometries intersect if they have any point in common. For geography, a distance tolerance of 0.00001 meters is used (so points that are very close are considered to intersect). In mathematical terms: ST_Intersects(A, B) $\Leftrightarrow$ A $\bigcap$ B $\neq \varnothing$ Geometries intersect if their DE-9IM Intersection Matrix matches one of: T******** *T******* ***T***** ****T**** Spatial intersection is implied by all the other spatial relationship tests, except , which tests that geometries do NOT intersect. This function automatically includes a bounding box comparison that makes use of any spatial indexes that are available on the geometries. Changed: 3.0.0 SFCGAL version removed and native support for 2D TINS added. Enhanced: 2.5.0 Supports GEOMETRYCOLLECTION. Enhanced: 2.3.0 Enhancement to PIP short-circuit extended to support MultiPoints with few points. Prior versions only supported point in polygon. Performed by the GEOS module (for geometry), geography is native Availability: 1.5 support for geography was introduced. For geography, this function has a distance tolerance of about 0.00001 meters and uses the sphere rather than spheroid calculation. NOTE: this is the "allowable" version that returns a boolean, not an integer. This method implements the OGC Simple Features Implementation Specification for SQL 1.1. s2.1.1.2 //s2.1.13.3 - ST_Intersects(g1, g2) --> Not (ST_Disjoint(g1, g2)) This method implements the SQL/MM specification. SQL-MM 3: 5.1.27 This method supports Circular Strings and Curves. This function supports Triangles and Triangulated Irregular Network Surfaces (TIN).

- ST_IsClosed - Gibt den Wert TRUE zurück, wenn die Anfangs- und Endpunkte des LINESTRING's zusammenfallen. Bei polyedrischen Oberflächen, wenn sie geschlossen (volumetrisch) sind. Beschreibung Gibt den Wert TRUE zurück, wenn die Anfangs- und Endpunkte des LINESTRING's zusammenfallen. Bei polyedrischen Oberflächen wird angezeigt, ob die Oberfläche eine Fläche (offen) oder ein Volumen (geschlossen) beschreibt. This method implements the OGC Simple Features Implementation Specification for SQL 1.1. This method implements the SQL/MM specification. SQL-MM 3: 7.1.5, 9.3.3 SQL-MM gibt vor, daß das Ergebnis von ST_IsClosed(NULL) 0 ergeben soll, während PostGIS NULL zurückgibt. This

function supports 3d and will not drop the z-index. This method supports Circular Strings and Curves. Erweiterung: Mit 2.0.0 wurde die Unterstützung für polyedrische Oberflächen eingeführt. This function supports Polyhedral surfaces.

- ST_IsEmpty - Tests if a geometry is empty. Beschreibung Gibt den Wert TRUE zurück, wenn es sich um eine leere Geometrie handelt. Falls TRUE, dann repräsentiert diese Geometrie eine leere GeometryCollection, Polygon, Point etc. SQL-MM gibt vor, daß das Ergebnis von ST_IsEmpty(NULL) der Wert 0 ist, während PostGIS den Wert NULL zurückgibt. This method implements the OGC Simple Features Implementation Specification for SQL 1.1. s2.1.1.1 This method implements the SQL/MM specification. SQL-MM 3: 5.1.7 This method supports Circular Strings and Curves. Änderung: 2.0.0 - In Vorgängerversionen von PostGIS war ST_GeomFromText('GEOMETRYCOLLECTION(EMPTY)') erlaubt. Um eine bessere Übereinstimmung mit der SQL/MM Norm zu erreichen, ist dies nun nicht mehr gestattet.
- ST_IsRing - Tests if a LineString is closed and simple. Beschreibung Gibt den Wert TRUE zurück, wenn der LINESTRING sowohl (ST_StartPoint(g) ~= ST_Endpoint(g)) als auch (sich nicht selbst überschneidet) ist. This method implements the OGC Simple Features Implementation Specification for SQL 1.1. 2.1.5.1 This method implements the SQL/MM specification. SQL-MM 3: 7.1.6 SQL-MM gibt vor, daß das Ergebnis vonST_IsRing(NULL) der Wert 0 sein soll, während PostGIS den Wert NULL zurückgibt.
- ST_IsSimple - Gibt den Wert (TRUE) zurück, wenn die Geometrie keine irregulären Stellen, wie Selbstüberschneidungen oder Selbstberührungen, aufweist. Beschreibung Gibt TRUE zurück, wenn keine regelwidrigen geometrischen Merkmale, wie Geometrien die sich selbst kreuzen oder berühren, auftreten. Für weiterführende Information zur OGC-Definition von Simplizität und Gültigkeit von Geometrien, siehe "Ensuring OpenGIS compliancy of geometries" SQL-MM definiert das Ergebnis von ST_IsSimple(NULL) als 0, während PostGIS NULL zurückgibt. This method implements the OGC Simple Features Implementation Specification for SQL 1.1. s2.1.1.1 This method implements the SQL/MM specification. SQL-MM 3: 5.1.8 This function supports 3d and will not drop the z-index.
- ST_IsValid - Tests if a geometry is well-formed in 2D. Beschreibung Tests if an ST_Geometry value is well-formed and valid in 2D according to the OGC rules. For geometries with 3 and 4 dimensions, the validity is still only tested in 2 dimensions. For geometries that are invalid, a PostgreSQL NOTICE is emitted providing details of why it is not valid. For the version with the flags parameter, supported values are documented in This version does not print a NOTICE explaining invalidity. For more information on the definition of geometry validity, refer to SQL-MM defines the result of ST_IsValid(NULL) to be 0, while PostGIS returns NULL. Performed by the GEOS module. The version accepting flags is available starting with 2.0.0. This method implements the OGC Simple Features Implementation Specification for SQL 1.1. This method implements the SQL/MM specification. SQL-MM 3: 5.1.9 Neither OGC-SFS nor SQL-MM specifications include a flag argument for ST_IsValid. The flag is a PostGIS extension.
- ST_Length - Gibt den geometrischen Schwerpunkt einer Geometrie zurück. Beschreibung Beim geometrischen Datentyp wird die kartesische 2D Länge der Geometrie zurückgegeben. Dabei muss es sich um einen LineString, einen MultiLineString, eine ST_Curve oder eine ST_MultiCurve handeln. Bei einer flächigen Geometrie wird 0 zurückgegeben. Für eine flächige Geometrie können Sie verwenden. Bei den geometrischen Datentypen sind die Einheiten für die Längenmessungen durch das Koordinatenreferenzsystem festgelegt. For geography types: computation is performed using the inverse geodesic calculation. Units of length are in meters. If PostGIS is compiled with PROJ version 4.8.0 or later, the spheroid is specified by the SRID, otherwise it is exclusive to WGS84. If use_spheroid = false, then the calculation is based on a sphere instead of a spheroid. Zur Zeit ein Synonym für ST_Length2D; dies kann sich allerdings ändern, wenn höhere Dimensionen unterstützt werden. Änderung: 2.0.0 Wesentliche Änderung -- In früheren Versionen ergab die Anwendung auf ein MULTI/POLYGON vom geographischen Datentyp den Umfang des POLYGON/MULTIPOLYGON. In 2.0.0 wurde dies geändert und es wird jetzt 0 zurückgegeben, damit es mit der Verhaltensweise beim geometrischen Datentyp übereinstimmt. Verwenden Sie bitte ST_Perimeter, wenn Sie den Umfang eines Polygons wissen wollen Beim geographischer Datentyp werden die Messungen standardmäßig am Referenzellipsoid durchgeführt. Für die schnellere, aber ungenauere Berechnung auf einer Kugel können Sie ST_Length(gg,false) verwenden; This method implements the OGC Simple Features Implementation Specification for SQL 1.1. s2.1.5.1 This method implements the SQL/MM specification. SQL-MM 3: 7.1.2, 9.3.4 Verfügbarkeit: 1.5.0 Unterstützung von geograpischen Koordinaten.
- ST_LineFromText - Erzeugt eine Geometrie aus einer WKT Darstellung mit der angegebenen SRID. Wenn keine SRID angegeben wird, wird diese standardmäßig auf 0 gesetzt. Beschreibung Erzeugt eine Geometrie aus einer WKT Darstellung mit der angegebenen SRID. Wenn keine SRID angegeben wird, wird diese standardmäßig auf 0 gesetzt. Wenn das übergebene WKT kein LineString ist, wird NULL zurückgegeben. OGC SPEC 3.2.6.2 - die Option SRID ist vom Konformitätstest. Wenn Sie wissen, dass die Geometrie nur aus LINESTRINGs besteht, ist es effizienter einfach ST_GeomFromText zu verwenden. Diese Funktion ruft auch nur ST_GeomFromText auf und fügt die Information hinzu, dass es sich um einen Linienzug handelt. This method implements the OGC Simple Features Implementation Specification for SQL 1.1. s3.2.6.2 This method implements the SQL/MM specification. SQL-MM 3: 7.2.8

- ST_LineFromWKB - Erzeugt einen LINESTRING mit gegebener SRID aus einer WKB-Darstellung Beschreibung Die Funktion ST_GeogFromWKB nimmt eine Well-known-Binary Darstellung der Geometrie und eine Id für das Koordinatenreferenzsystem (SRID) entgegen und erzeugt eine Instanz des entsprechenden geometrischen Datentyps - in diesem Fall eine Geometrie vom Typ LineString. Diese Funktion übernimmt die Rolle der Geometrie-Factory in SQL. Wenn keine SRID angegeben ist, wird diese auf 0 gesetzt. NULL wird zurückgegeben, wenn die Eingabe bytea keinen LINESTRING darstellt. OGC SPEC 3.2.6.2 - die Option SRID ist vom Konformitätstest. Wenn Sie wissen, dass Ihre Geometrie nur aus LINESTRINGs besteht, ist es effizienter einfach zu verwenden. Diese Funktion ruft auch nur auf und fügt die Information hinzu, dass es sich um einen Linienzug handelt. This method implements the OGC Simple Features Implementation Specification for SQL 1.1. s3.2.6.2 This method implements the SQL/MM specification. SQL-MM 3: 7.2.9

- ST_LinestringFromWKB - Erzeugt eine Geometrie mit gegebener SRID aus einer WKB-Darstellung. Beschreibung Die Funktion ST_LinestringFromWKB nimmt eine Well-known-Binary Darstellung der Geometrie und eine Id für das Koordinatenreferenzsystem (SRID) entgegen und erzeugt eine Instanz des entsprechenden geometrischen Datentyps - in diesem Fall eine Geometrie vom Typ LineString. Diese Funktion übernimmt die Rolle der Geometrie-Factory in SQL. Wenn keine SRID angegeben ist, wird diese standardmäßig auf 0 gesetzt. NULL wird zurückgegeben, wenn die Eingabe bytea keinen LINESTRING darstellt. Ist ein Alias für . OGC SPEC 3.2.6.2 - optionale SRID ist vom Konformitätstest. Wenn Sie wissen, dass Ihre Geometrie nur aus LINESTRINGs besteht, ist es effizienter einfach zu verwenden. Diese Funktion ruft auch nur auf und fügt die Information hinzu, dass es sich um einen LINESTRING handelt. This method implements the OGC Simple Features Implementation Specification for SQL 1.1. s3.2.6.2 This method implements the SQL/MM specification. SQL-MM 3: 7.2.9

- ST_LocateAlong - Returns the point(s) on a geometry that match a measure value. Beschreibung Returns the location(s) along a measured geometry that have the given measure values. The result is a Point or MultiPoint. Polygonal inputs are not supported. If offset is provided, the result is offset to the left or right of the input line by the specified distance. A positive offset will be to the left, and a negative one to the right. Use this function only for linear geometries with an M component The semantic is specified by the ISO/IEC 13249-3 SQL/MM Spatial standard. Verfügbarkeit: 1.1.0 über die alte Bezeichnung ST_Locate_Along_Measure. Änderung: 2.0.0 In Vorgängerversionen als ST_Locate_Along_Measure bezeichet. Der alte Name ist überholt und wird in der Zukunft entfernt ist aber noch verfügbar. This function supports M coordinates. This method implements the SQL/MM specification. SQL-MM IEC 13249-3: 5.1.13

- ST_LocateBetween - Returns the portions of a geometry that match a measure range. Beschreibung Gibt eine abgeleitete Sammelgeometrie mit jenen Elementen zurück, die in dem gegebenen Kilometrierungsintervall liegen; das Intervall ist unbeschränkt. Polygonale Elemente werden nicht untersützt. Wenn ein Versatz angegeben ist, werden die Resultierenden um diese Anzahl an Einheiten nach links oder rechts von der gegebenen Linie versetzt. Ein positiver Versatz geschieht nach links, ein negativer nach rechts. Clipping a non-convex POLYGON may produce invalid geometry. The semantic is specified by the ISO/IEC 13249-3 SQL/MM Spatial standard. Verfügbarkeit: 1.1.0 über die alte Bezeichnung ST_Locate_Between_Measures. Änderung: 2.0.0 In Vorgängerversionen als ST_Locate_Along_Measure bezeichet. Der alte Name ist überholt und wird in der Zukunft entfernt ist aber noch verfügbar. Enhanced: 3.0.0 - added support for POLYGON, TIN, TRIANGLE. This function supports M coordinates. This method implements the SQL/MM specification. SQL-MM IEC 13249-3: 5.1

- ST_M - Returns the M coordinate of a Point. Beschreibung Gibt die M-Koordinate des Punktes zurück, oder NULL wenn keine vorhanden ist. Der Einabewert muss ein Punkt sein. Dies ist (noch) kein Teil der OGC Spezifikation, wird aber hier aufgeführt um die Liste von Funktionen zum Auslesen von Punktkoordinaten zu vervollständigen. This method implements the OGC Simple Features Implementation Specification for SQL 1.1. This method implements the SQL/MM specification. This function supports 3d and will not drop the z-index.

- ST_MLineFromText - Liest einen festgelegten ST_MultiLineString Wert von einer WKT-Darstellung aus. Beschreibung Erzeugt eine Geometrie aus einer Well-known-Text (WKT) Darstellung mit der angegebenen SRID. Wenn keine SRID angegeben wird, wird diese standardmäßig auf 0 gesetzt. OGC SPEC 3.2.6.2 - Die Option SRID stammt aus dem Konformitätstest Gibt NULL zurück wenn der WKT kein MULTILINESTRING ist. Verwenden Sie diese Funktion nicht, wenn Sie sich vollkommen sicher sind, dass ihre WKT Geometrie nur aus Punkten besteht. Sie ist langsamer als ST_GeomFromText, da sie einen zusätzlichen Validierungsschritt hinzufügt. This method implements the OGC Simple Features Implementation Specification for SQL 1.1. s3.2.6.2 This method implements the SQL/MM specification. SQL-MM 3: 9.4.4

- ST_MPointFromText - Erzeugt eine Geometrie aus WKT mit der angegebenen SRID. Wenn keine SRID angegeben wird, wird diese standardmäßig auf 0 gesetzt. Beschreibung Erzeugt eine Geometrie aus WKT mit der angegebenen SRID. Wenn SRID nicht angegeben ist, wird sie standardmäßig auf 0 gesetzt. OGC SPEC 3.2.6.2 - Die Option SRID stammt aus dem Konformitätstest Gibt NULL zurück, wenn der WKT kein MULTIPOINT ist. Verwenden Sie diese Funktion nicht, wenn Sie sich vollkommen sicher sind, dass ihre WKT Geometrie nur aus Punkten besteht. Sie ist langsamer als ST_GeomFromText,

da sie einen zusätzlichen Validierungsschritt hinzufügt. This method implements the OGC Simple Features Implementation Specification for SQL 1.1. 3.2.6.2 This method implements the SQL/MM specification. SQL-MM 3: 9.2.4

- ST_MPolyFromText - Erzeugt eine MultiPolygon Geometrie aus WKT mit der angegebenen SRID. Wenn SRID nicht angegeben ist, wird sie standardmäßig auf 0 gesetzt. Beschreibung Erzeugt ein MultiPolygon von WKT mit der gegebenen SRID. Wenn SRID nicht angegeben ist, wird sie standardmäßig auf 0 gesetzt. OGC SPEC 3.2.6.2 - Die Option SRID stammt aus dem Konformitätstest Meldet einen Fehler, wenn der WKT kein MULTIPOLYGON ist. Verwenden Sie diese Funktion nicht, wenn Sie sich vollkommen sicher sind, dass ihre WKT Geometrie nur aus MultiPolygonen besteht. Sie ist langsamer als ST_GeomFromText, da sie einen zusätzlichen Validierungsschritt hinzufügt. This method implements the OGC Simple Features Implementation Specification for SQL 1.1. s3.2.6.2 This method implements the SQL/MM specification. SQL-MM 3: 9.6.4

- ST_ModEdgeHeal - "Heilt" zwei Kanten, indem der verbindende Knoten gelöscht wird, die erste Kante modifiziert und die zweite Kante gelöscht wird. Gibt die ID des gelöschten Knoten zurück. Beschreibung "Heilt" zwei Kanten, indem der verbindende Knoten gelöscht wird, die erste Kante modifiziert und die zweite Kante gelöscht wird. Gibt die ID des gelöschten Knoten zurück. Aktualisiert alle verbundenen Kanten und Beziehungen dementsprechend. Verfügbarkeit: 2.0 This method implements the SQL/MM specification. SQL-MM: Topo-Geo and Topo-Net 3: Routine Details: X.3.9

- ST_ModEdgeSplit - Trennt eine Kante auf, indem ein neuer Knoten entlang einer bestehenden Kante erstellt wird. Ändert die ursprüngliche Kante und fügt eine neue Kante hinzu. Beschreibung Trennt eine Kante auf, indem ein neuer Knoten entlang einer bestehenden Kante erstellt wird. Ändert die ursprüngliche Kante und fügt eine neue Kante hinzu. Alle bestehenden Kanten und Beziehungen werden entsprechend aktualisiert. Gibt den Identifikator des neu hinzugefügten Knotens aus. Availability: 1.1 Änderung: 2.0 - In Vorgängerversionen fälschlicherweise als ST_ModEdgesSplit bezeichnet This method implements the SQL/MM specification. SQL-MM: Topo-Geo and Topo-Net 3: Routine Details: X.3.9

- ST_MoveIsoNode - Verschiebt einen isolierten Knoten in einer Topologie von einer Stelle an eine andere. Falls die neue Geometrie apoint bereits als Knoten existiert, wird eine Fehlermeldung ausgegeben. Gibt eine Beschreibung der Verschiebung aus. Beschreibung Verschiebt einen isolierten Knoten in einer Topologie von einer Stelle an eine andere. Falls die neue Geometrie apoint bereits als Knoten existiert, wird eine Fehlermeldung ausgegeben. If any arguments are null, the apoint is not a point, the existing node is not isolated (is a start or end point of an existing edge), new node location intersects an existing edge (even at the end points) or the new location is in a different face (since 3.2.0) then an exception is thrown. Wenn das Koordinatenreferenzsystem (SRID) der Punktgeometrie nicht mit jener der Topologie übereinstimmt, wird eine Fehlermeldung ausgegeben. Verfügbarkeit: 2.0.0 Enhanced: 3.2.0 ensures the nod cannot be moved in a different face This method implements the SQL/MM specification. SQL-MM: Topo-Net Routines: X.3.2

- ST_NewEdgeHeal - "Heilt" zwei Kanten, indem der verbindende Knoten und beide Kanten gelöscht werden. Die beiden Kanten werden durch eine Kante ersetzt, welche dieselbe Ausichtung wie die erste Kante hat. Beschreibung "Heilt" zwei Kanten, indem der verbindende Knoten und beide Kanten gelöscht werden. Die beiden Kanten werden durch eine Kante ersetzt, welche dieselbe Ausichtung wie die erste Kante hat. Gibt die ID der neuen Kante aus. Aktualisiert alle verbundenen Kanten und Beziehungen dementsprechend. Verfügbarkeit: 2.0 This method implements the SQL/MM specification. SQL-MM: Topo-Geo and Topo-Net 3: Routine Details: X.3.9

- ST_NewEdgesSplit - Trennt eine Kante auf, indem ein neuer Knoten entlang einer bestehenden Kante erstellt, die ursprüngliche Kante gelöscht und durch zwei neue Kanten ersetzt wird. Gibt die ID des neu erstellten Knotens aus, der die neuen Kanten verbindet. Beschreibung Trennt eine Kante mit der Kanten-ID anedgeauf, indem ein neuer Knoten mit der Punktlage apoint entlang der aktuellen Kante erstellt, die ursprüngliche Kante gelöscht und durch zwei neue Kanten ersetzt wird. Gibt die ID des neu erstellten Knotens aus, der die neuen Kanten verbindet. Aktualisiert alle verbundenen Kanten und Beziehungen dementsprechend. Wenn das Koordinatenreferenzsystem (SRID) der Punktgeometrie nicht mit dem der Topologie übereinstimmt, apoint keine Punktgeometrie ist, der Punkt NULL ist, der Punkt bereits als Knoten existiert, die Kante mit einer bestehenden Kante nicht zusammenpasst, oder der Punkt nicht innerhalb der Kante liegt, dann wird eine Fehlermeldung ausgegeben. Availability: 1.1 This method implements the SQL/MM specification. SQL-MM: Topo-Net Routines: X.3.8

- ST_NumGeometries - Gibt die Anzahl der Punkte einer Geometrie zurück. Funktioniert für alle Geometrien. Beschreibung Gibt die Anzahl an Geometrien aus. Wenn es sich bei der Geometrie um eine GEOMETRYCOLLECTION (oder MULTI*) handelt, wird die Anzahl der Geometrien zurückgegeben, bei Einzelgeometrien wird 1, ansonsten NULL zurückgegeben. Erweiterung: Mit 2.0.0 wurde die Unterstützung für polyedrische Oberflächen, Dreiecke und TIN eingeführt. Änderung: 2.0.0 Bei früheren Versionen wurde NULL zurückgegeben, wenn die Geometrie nicht vom Typ GEOMETRYCOLLECTION/-MULTI war. 2.0.0+ gibt nun 1 für Einzelgeometrien, wie POLYGON, LINESTRING, POINT zurück. This method implements the SQL/MM specification. SQL-MM 3: 9.1.4 This function supports 3d and will not drop the z-index. This function supports Polyhedral surfaces. This function supports Triangles and Triangulated Irregular Network Surfaces (TIN).

- ST_NumInteriorRings - Gibt die Anzahl der inneren Ringe einer Polygongeometrie aus. Beschreibung Gibt die Anzahl der inneren Ringe einer Polygongeometrie aus. Gibt NULL zurück, wenn die Geometrie kein Polygon ist. This method implements the SQL/MM specification. SQL-MM 3: 8.2.5 Änderung: 2.0.0 - In füheren Versionen war ein MULTIPOLYGON als Eingabe erlaubt, wobei die Anzahl der inneren Ringe des ersten Polygons ausgegeben wurde.

- ST_NumPatches - Gibt die Anzahl der Maschen einer polyedrischen Oberfläche aus. Gibt NULL zurück, wenn es sich nicht um polyedrische Geometrien handelt. Beschreibung Gibt die Anzahl der Maschen einer polyedrischen Oberfläche aus. Gibt NULL zurück, wenn es sich um keine polyedrische Geometrie handelt. Ist ein Synonym für ST_NumGeometries zur Unterstützung der MM Namensgebung. Wenn Ihnen die MM-Konvention egal ist, so ist die Verwendung von ST_NumGeometries schneller. Verfügbarkeit: 2.0.0 This function supports 3d and will not drop the z-index. This method implements the OGC Simple Features Implementation Specification for SQL 1.1. This method implements the SQL/MM specification. SQL-MM ISO/IEC 13249-3: 8.5 This function supports Polyhedral surfaces.

- ST_NumPoints - Gibt die Anzahl der Stützpunkte eines ST_LineString oder eines ST_CircularString zurück. Beschreibung Gibt die Anzahl der Stützpunkte eines ST_LineString oder eines ST_CircularString zurück. Vor 1.4 funktionierte dies nur mit ST_LineString, wie von der Spezifikation festgelegt. Ab 1.4 aufwärts handelt es sich um einen Alias für ST_NPoints, das die Anzahl der Knoten nicht nur für Linienzüge ausgibt. Erwägen Sie stattdessen die Verwendung von ST_NPoints, das vielseitig ist und mit vielen Geometrietypen funktioniert. This method implements the OGC Simple Features Implementation Specification for SQL 1.1. This method implements the SQL/MM specification. SQL-MM 3: 7.2.4

- ST_OrderingEquals - Tests if two geometries represent the same geometry and have points in the same directional order Beschreibung ST_OrderingEquals compares two geometries and returns t (TRUE) if the geometries are equal and the coordinates are in the same order; otherwise it returns f (FALSE). This function is implemented as per the ArcSDE SQL specification rather than SQL-MM. http://edndoc.esri.com/arcsde/9.1/sql_api/sqlapi3.htm#ST_OrderingEquals This method implements the SQL/MM specification. SQL-MM 3: 5.1.43

- ST_Overlaps - Tests if two geometries have the same dimension and intersect, but each has at least one point not in the other Beschreibung Returns TRUE if geometry A and B "spatially overlap". Two geometries overlap if they have the same dimension, their interiors intersect in that dimension. and each has at least one point inside the other (or equivalently, neither one covers the other). The overlaps relation is symmetric and irreflexive. In mathematical terms: ST_Overlaps(A, B) $\Leftrightarrow$ (dim(A) = dim(B) = dim(Int(A) $\bigcap$ Int(B))) $\wedge$ (A $\bigcap$ B $\neq$ A) $\wedge$ (A $\bigcap$ B $\neq$ B) This function automatically includes a bounding box comparison that makes use of any spatial indexes that are available on the geometries. To avoid index use, use the function _ST_Overlaps. Wird durch das GEOS Modul ausgeführt Enhanced: 3.0.0 enabled support for GEOMETRYCOLLECTION NOTE: this is the "allowable" version that returns a boolean, not an integer. This method implements the OGC Simple Features Implementation Specification for SQL 1.1. s2.1.1.2 // s2.1.13.3 This method implements the SQL/MM specification. SQL-MM 3: 5.1.32

- ST_PatchN - Gibt den Geometrietyp des ST_Geometry Wertes zurück. Beschreibung >Gibt die auf 1-basierende n-te Geometrie (Masche) zurück, wenn es sich bei der Geometrie um ein POLYHEDRALSURFACE, oder ein POLYHEDRALSURFACEM handelt. Anderenfalls wird NULL zurückgegeben. Gibt bei polyedrischen Oberflächen das selbe Ergebnis wie ST_GeometryN. Die Verwendung von ST_GeometryN ist schneller. Der Index ist auf 1 basiert. Falls Sie alle Geometrien einer Geometrie entnehmen wollen, so ist ST_Dump wesentlich leistungsfähiger. Verfügbarkeit: 2.0.0 This method implements the SQL/MM specification. SQL-MM ISO/IEC 13249-3: 8.5 This function supports 3d and will not drop the z-index. This function supports Polyhedral surfaces.

- ST_Perimeter - Returns the length of the boundary of a polygonal geometry or geography. Beschreibung Gibt für die geometrischen/geographischen Datentypen ST_Surface, ST_MultiSurface (Polygon, MultiPolygon) den Umfang in 2D zurück. Bei einer nicht flächigen Geometrie wird 0 zurückgegeben. Für eine lineare Geometrie können Sie verwenden. Beim geometrischen Datentyp sind die Einheiten der Umfangsmessung durch das Koordinatenreferenzsystem der Geometrie festgelegt. For geography types, the calculations are performed using the inverse geodesic problem, where perimeter units are in meters. If PostGIS is compiled with PROJ version 4.8.0 or later, the spheroid is specified by the SRID, otherwise it is exclusive to WGS84. If use_spheroid = false, then calculations will approximate a sphere instead of a spheroid. Zur Zeit ein Synonym für ST_Perimeter2D; dies kann sich allerdings ändern, wenn höhere Dimensionen unterstützt werden. This method implements the OGC Simple Features Implementation Specification for SQL 1.1. s2.1.5.1 This method implements the SQL/MM specification. SQL-MM 3: 8.1.3, 9.5.4 Verfügbarkeit: Mit 2.0.0 wurde die Unterstützung für geograpischen Koordinaten eingeführt

- ST_Point - Creates a Point with X, Y and SRID values. Beschreibung Returns a Point with the given X and Y coordinate values. This is the SQL-MM equivalent for that takes just X and Y. For geodetic coordinates, X is longitude and Y is latitude Enhanced: 3.2.0 srid as an extra optional argument was added. Older installs require combining with ST_SetSRID to mark the srid on the geometry. This method implements the SQL/MM specification. SQL-MM 3: 6.1.2

- ST_PointFromText - Erzeugt eine Punktgeometrie mit gegebener SRID von WKT. Wenn SRID nicht angegeben ist, wird sie standardmäßig auf 0 gesetzt. Beschreibung Erzeugt ein PostGIS ST_Geometrie Punktobjekt von der OGC Well-known-Text Darstellung. Wenn die SRID nicht angegeben ist, wird sie standardmäßig auf "unknown" (zurzeit 0) gesetzt. Falls die Geometrie nicht in der WKT Punktdarstellung vorliegt, wird NULL zurückgegeben. Bei einer invaliden WKT Darstellung wird eine Fehlermeldung angezeigt. Die Funktion ST_PointFromText hat zwei Varianten. Die erste Variante nimmt keine SRID entgegen und gibt eine Geometrie ohne ein bestimmtes Koordinatenreferenzsystem aus. Die zweite Variante nimmt eine SRID als zweiten Übergabewert entgegen und gibt eine Geometrie zurück, die diese SRID als Teil ihrer Metadaten beinhaltet. Die SRID muss in der Tabelle "spatial_ref_sys" definiert sein. Verwenden Sie diese Funktion nicht, wenn Sie sich vollkommen sicher sind, dass ihre WKT Geometrie nur aus Punkten besteht. Sie ist langsamer als ST_GeomFromText, da sie einen zusätzlichen Validierungsschritt hinzufügt. Wenn Sie Punkte aus Koordinaten in Länge und Breite erstellen und mehr auf Rechenleistung und Genauigkeit wertlegen als auf OGC-Konformität, so verwenden Sie bitte oder den OGC-konformen Alias . This method implements the OGC Simple Features Implementation Specification for SQL 1.1. s3.2.6.2 - die Option SRID ist vom Konformitätstest. This method implements the SQL/MM specification. SQL-MM 3: 6.1.8

- ST_PointFromWKB - Erzeugt eine Geometrie mit gegebener SRID von WKB. Beschreibung Die Funktion ST_PointFromWKB nimmt eine Well-known-Binary Darstellung der Geometrie und eine Id für das Koordinatenreferenzsystem (SRID) entgegen und erzeugt eine Instanz des entsprechenden geometrischen Datentyps - in diesem Fall eine Geometrie vom Typ POINT. Diese Funktion übernimmt die Rolle der Geometrie-Factory in SQL. Wenn keine SRID angegeben ist, wird diese standardmäßig auf 0 gesetzt. NULL wird zurückgegeben, wenn die Eingabe bytea keinen POINT darstellt. This method implements the OGC Simple Features Implementation Specification for SQL 1.1. s3.2.7.2 This method implements the SQL/MM specification. SQL-MM 3: 6.1.9 This function supports 3d and will not drop the z-index. This method supports Circular Strings and Curves.

- ST_PointN - Gibt die Anzahl der Stützpunkte eines ST_LineString oder eines ST_CircularString zurück. Beschreibung Gibt den n-ten Punkt des ersten LineString's oder des kreisförmigen LineStrings's einer Geometrie zurück. Negative Werte werden rückwärts, vom Ende des LineString's her gezählt, sodass -1 der Endpunkt ist. Gibt NULL aus, wenn die Geometrie keinen LineString enthält. Seit Version 0.8.0 ist der Index 1-basiert, so wie in der OGC Spezifikation. Rückwärtiges Indizieren (negativer Index) findet sich nicht in der OGC Spezifikation. Vorhergegangene Versionen waren 0-basiert. Falls Sie den n-ten Punkt eines jeden LineString's in einem MultiLinestring wollen, nutzen Sie diese Funktion gemeinsam mit ST_Dump. This method implements the OGC Simple Features Implementation Specification for SQL 1.1. This method implements the SQL/MM specification. SQL-MM 3: 7.2.5, 7.3.5 This function supports 3d and will not drop the z-index. This method supports Circular Strings and Curves. Änderung: 2.0.0 arbeitet nicht mehr mit MultiLinestring's, die nur eine einzelne Geometrie enthalten. In früheren Versionen von PostGIS gab die Funktion bei einem, aus einer einzelnen Linie bestehender MultiLinestring, den Anfangspunkt zurück. Ab 2.0.0 wird, so wie bei jedem anderen MultiLinestring auch, NULL zurückgegeben. Änderung: 2.3.0 : negatives Indizieren verfügbar (-1 entspricht dem Endpunkt)

- ST_PointOnSurface - Computes a point guaranteed to lie in a polygon, or on a geometry. Beschreibung Returns a POINT which is guaranteed to lie in the interior of a surface (POLYGON, MULTIPOLYGON, and CURVED POLYGON). In PostGIS this function also works on line and point geometries. This method implements the OGC Simple Features Implementation Specification for SQL 1.1. s3.2.14.2 // s3.2.18.2 This method implements the SQL/MM specification. SQL-MM 3: 8.1.5, 9.5.6. The specifications define ST_PointOnSurface for surface geometries only. PostGIS extends the function to support all common geometry types. Other databases (Oracle, DB2, ArcSDE) seem to support this function only for surfaces. SQL Server 2008 supports all common geometry types. This function supports 3d and will not drop the z-index.

- ST_Polygon - Creates a Polygon from a LineString with a specified SRID. Beschreibung Returns a polygon built from the given LineString and sets the spatial reference system from the srid. ST_Polygon is similar to Variant 1 with the addition of setting the SRID. , , Diese Funktion akzeptiert keine MULTILINESTRINGs. Verwenden Sie bitte oder um Linienzüge zu erzeugen. This method implements the OGC Simple Features Implementation Specification for SQL 1.1. This method implements the SQL/MM specification. SQL-MM 3: 8.3.2 This function supports 3d and will not drop the z-index.

- ST_PolygonFromText - Erzeugt eine Geometrie aus WKT mit der angegebenen SRID. Wenn keine SRID angegeben wird, wird diese standardmäßig auf 0 gesetzt. Beschreibung Erzeugt eine Geometrie mit gegebener SRID von WKT. Wenn SRID nicht angegeben ist, wird sie standardmäßig auf 0 gesetzt. Gibt NULL zurück, wenn WKT kein Polygon ist. OGC SPEC 3.2.6.2 - Die Option SRID stammt aus dem Konformitätstest Verwenden Sie diese Funktion nicht, wenn Sie sich vollkommen sicher sind, dass ihre WKT Geometrie nur aus Polygonen besteht. Sie ist langsamer als ST_GeomFromText, da sie einen zusätzlichen Validierungsschritt hinzufügt. This method implements the OGC Simple Features Implementation Specification for SQL 1.1. s3.2.6.2 This method implements the SQL/MM specification. SQL-MM 3: 8.3.6

- ST_Relate - Tests if two geometries have a topological relationship matching an Intersection Matrix pattern, or computes their Intersection Matrix Beschreibung These functions allow testing and evaluating the spatial (topological) relationship between

two geometries, as defined by the Dimensionally Extended 9-Intersection Model (DE-9IM). The DE-9IM is specified as a 9-element matrix indicating the dimension of the intersections between the Interior, Boundary and Exterior of two geometries. It is represented by a 9-character text string using the symbols 'F', '0', '1', '2' (e.g. 'FF1FF0102'). A specific kind of spatial relationship can be tested by matching the intersection matrix to an intersection matrix pattern. Patterns can include the additional symbols 'T' (meaning "intersection is non-empty") and '*' (meaning "any value"). Common spatial relationships are provided by the named functions , , , , , , , , , , and . Using an explicit pattern allows testing multiple conditions of intersects, crosses, etc in one step. It also allows testing spatial relationships which do not have a named spatial relationship function. For example, the relationship "Interior-Intersects" has the DE-9IM pattern T********, which is not evaluated by any named predicate. For more information refer to . Variant 1: Tests if two geometries are spatially related according to the given intersectionMatrixPattern. Unlike most of the named spatial relationship predicates, this does NOT automatically include an index call. The reason is that some relationships are true for geometries which do NOT intersect (e.g. Disjoint). If you are using a relationship pattern that requires intersection, then include the && index call. It is better to use a named relationship function if available, since they automatically use a spatial index where one exists. Also, they may implement performance optimizations which are not available with full relate evalation. Variant 2: Returns the DE-9IM matrix string for the spatial relationship between the two input geometries. The matrix string can be tested for matching a DE-9IM pattern using . Variant 3: Like variant 2, but allows specifying a Boundary Node Rule. A boundary node rule allows finer control over whether the endpoints of MultiLineStrings are considered to lie in the DE-9IM Interior or Boundary. The boundaryNodeRule values are: 1: OGC-Mod2 - line endpoints are in the Boundary if they occur an odd number of times. This is the rule defined by the OGC SFS standard, and is the default for ST_Relate. 2: Endpoint - all endpoints are in the Boundary. 3: MultivalentEndpoint - endpoints are in the Boundary if they occur more than once. In other words, the boundary is all the "attached" or "inner" endpoints (but not the "unattached/outer" ones). 4: MonovalentEndpoint - endpoints are in the Boundary if they occur only once. In other words, the boundary is all the "unattached" or "outer" endpoints. This function is not in the OGC spec, but is implied. see s2.1.13.2 This method implements the OGC Simple Features Implementation Specification for SQL 1.1. s2.1.1.2 // s2.1.13.3 This method implements the SQL/MM specification. SQL-MM 3: 5.1.25 Wird durch das GEOS Modul ausgeführt Enhanced: 2.0.0 - added support for specifying boundary node rule. Enhanced: 3.0.0 enabled support for GEOMETRYCOLLECTION

- ST_RemEdgeModFace - Removes an edge, and if the edge separates two faces deletes one face and modifies the other face to cover the space of both. Beschreibung Removes an edge, and if the removed edge separates two faces deletes one face and modifies the other face to cover the space of both. Preferentially keeps the face on the right, to be consistent with . Returns the id of the face which is preserved. Führt ein entsprechendes Update auf alle verbundenen Kanten und Beziehungen durch. Wenn eine Kante an der Definition einer bestehenden TopoGeometry beteiligt ist, kann sie nicht gelöscht werden. Wenn irgendeine TopoGeometry nur durch eine der beiden Maschen definiert ist (und nicht auch durch die andere), können die beiden Maschen nicht "geheilt" werden. Wenn irgendwelche Übergabewerte NULL sind, die angegebene Kante unbekannt ist (muss bereits in der edge Tabelle des Schemas "topology" existieren), oder die Bezeichnung der Topologie ungültig ist, dann wird eine Fehlermeldung ausgegeben. Verfügbarkeit: 2.0 This method implements the SQL/MM specification. SQL-MM: Topo-Geo and Topo-Net 3: Routine Details: X.3.15

- ST_RemEdgeNewFace - Entfernt eine Kante. Falls die gelöschte Kante zwei Maschen voneinander getrennt hat, werden die ursprünglichen Maschen gelöscht und durch einer neuen Masche ersetzt. Beschreibung Entfernt eine Kante. Falls die gelöschte Kante zwei Maschen voneinander getrennt hat, werden die ursprünglichen Maschen gelöscht und durch einer neuen Masche ersetzt. Gibt die ID der neu erzeugten Masche aus; oder NULL wenn keine Masche erstellt wurde. Es wird keine neue Masche erstellt, wenn die gelöschte Kante defekt oder isoliert ist, oder wenn sie die Begrenzung der Maschengrundmenge darstellt (wodurch die Grundmenge womöglich von der anderen Seite in die Masche fliessen könnte). Führt ein entsprechendes Update auf alle verbundenen Kanten und Beziehungen durch. Wenn eine Kante an der Definition einer bestehenden TopoGeometry beteiligt ist, kann sie nicht gelöscht werden. Wenn irgendeine TopoGeometry nur durch eine der beiden Maschen definiert ist (und nicht auch durch die andere), können die beiden Maschen nicht "geheilt" werden. Wenn irgendwelche Übergabewerte NULL sind, die angegebene Kante unbekannt ist (muss bereits in der edge Tabelle des Schemas "topology" existieren), oder die Bezeichnung der Topologie ungültig ist, dann wird eine Fehlermeldung ausgegeben. Verfügbarkeit: 2.0 This method implements the SQL/MM specification. SQL-MM: Topo-Geo and Topo-Net 3: Routine Details: X.3.14

- ST_RemoveIsoEdge - Löscht einen isolierten Knoten und gibt eine Beschreibung der getroffenen Maßnahmen aus. Falls der Knoten nicht isoliert ist, wird eine Fehlermeldung ausgegeben. Beschreibung Löscht einen isolierten Knoten und gibt eine Beschreibung der getroffenen Maßnahmen aus. Falls der Knoten nicht isoliert ist, wird eine Fehlermeldung ausgegeben. Availability: 1.1 This method implements the SQL/MM specification. SQL-MM: Topo-Geo and Topo-Net 3: Routine Details: X+1.3.3

- ST_RemoveIsoNode - Löscht einen isolierten Knoten und gibt eine Beschreibung der getroffenen Maßnahmen aus. Falls der Knoten nicht isoliert ist (ist der Anfangs- oder der Endpunkt einer Kante), wird eine Fehlermeldung ausgegeben. Beschreibung

Löscht einen isolierten Knoten und gibt eine Beschreibung der getroffenen Maßnahmen aus. Falls der Knoten nicht isoliert ist (ist der Anfangs- oder der Endpunkt einer Kante), wird eine Fehlermeldung ausgegeben. Availability: 1.1 This method implements the SQL/MM specification. SQL-MM: Topo-Geo and Topo-Net 3: Routine Details: X+1.3.3

- ST_SRID - Returns the spatial reference identifier for a geometry. Beschreibung Returns the spatial reference identifier for the ST_Geometry as defined in spatial_ref_sys table. spatial_ref_sys table is a table that catalogs all spatial reference systems known to PostGIS and is used for transformations from one spatial reference system to another. So verifying you have the right spatial reference system identifier is important if you plan to ever transform your geometries. This method implements the OGC Simple Features Implementation Specification for SQL 1.1. s2.1.1.1 This method implements the SQL/MM specification. SQL-MM 3: 5.1.5 This method supports Circular Strings and Curves.

- ST_StartPoint - Returns the first point of a LineString. Beschreibung Gibt den Anfangspunkt einer LINESTRING oder CIRCULARLINESTRING Geometrie als POINT oder NULL zurück, falls es sich beim Eingabewert nicht um einen LINESTRING oder CIRCULARLINESTRING handelt. This method implements the SQL/MM specification. SQL-MM 3: 7.1.3 This function supports 3d and will not drop the z-index. This method supports Circular Strings and Curves. Enhanced: 3.2.0 returns a point for all geometries. Prior behavior returns NULLs if input was not a LineString. Änderung: 2.0.0 unterstützt die Verarbeitung von MultiLinestring's die nur aus einer einzelnen Geometrie bestehen, nicht mehr. In früheren Versionen von PostGIS gab die Funktion bei einem aus einer einzelnen Linie bestehender MultiLinestring den Anfangspunkt zurück. Ab 2.0.0 gibt sie nur NULL zurück, so wie bei jedem anderen MultiLinestring. Die alte Verhaltensweise war undokumentiert, aber Anwender, die annahmen, dass Sie Ihre Daten als LINESTRING vorliegen haben, könnten in 2.0 dieses zurückgegebene NULL bemerken.

- ST_SymDifference - Computes a geometry representing the portions of geometries A and B that do not intersect. Beschreibung Returns a geometry representing the portions of geonetries A and B that do not intersect. This is equivalent to ST_Union(A,B) - ST_Intersection(A,B). It is called a symmetric difference because ST_SymDifference(A,B) = ST_SymDifference(B,A). If the optional gridSize argument is provided, the inputs are snapped to a grid of the given size, and the result vertices are computed on that same grid. (Requires GEOS-3.9.0 or higher) Wird durch das GEOS Modul ausgeführt Enhanced: 3.1.0 accept a gridSize parameter. Requires GEOS >= 3.9.0 to use the gridSize parameter This method implements the OGC Simple Features Implementation Specification for SQL 1.1. s2.1.1.3 This method implements the SQL/MM specification. SQL-MM 3: 5.1.21 This function supports 3d and will not drop the z-index. However, the result is computed using XY only. The result Z values are copied, averaged or interpolated.

- ST_Touches - Tests if two geometries have at least one point in common, but their interiors do not intersect Beschreibung Returns TRUE if A and B intersect, but their interiors do not intersect. Equivalently, A and B have at least one point in common, and the common points lie in at least one boundary. For Point/Point inputs the relationship is always FALSE, since points do not have a boundary. In mathematical terms: ST_Touches(A, B) $\Leftrightarrow$ (Int(A) $\bigcap$ Int(B) $\neq \varnothing$) $\wedge$ (A $\bigcap$ B $\neq \varnothing$) This relationship holds if the DE-9IM Intersection Matrix for the two geometries matches one of: FT******* F**T***** F***T**** This function automatically includes a bounding box comparison that makes use of any spatial indexes that are available on the geometries. To avoid using an index, use _ST_Touches instead. Enhanced: 3.0.0 enabled support for GEOMETRYCOLLECTION This method implements the OGC Simple Features Implementation Specification for SQL 1.1. s2.1.1.2 // s2.1.13.3 This method implements the SQL/MM specification. SQL-MM 3: 5.1.28

- ST_Transform - Return a new geometry with coordinates transformed to a different spatial reference system. Beschreibung Returns a new geometry with its coordinates transformed to a different spatial reference system. The destination spatial reference to_srid may be identified by a valid SRID integer parameter (i.e. it must exist in the spatial_ref_sys table). Alternatively, a spatial reference defined as a PROJ.4 string can be used for to_proj and/or from_proj, however these methods are not optimized. If the destination spatial reference system is expressed with a PROJ.4 string instead of an SRID, the SRID of the output geometry will be set to zero. With the exception of functions with from_proj, input geometries must have a defined SRID. ST_Transform is often confused with . ST_Transform actually changes the coordinates of a geometry from one spatial reference system to another, while ST_SetSRID() simply changes the SRID identifier of the geometry. ST_Transform automatically selects a suitable conversion pipeline given the source and target spatial reference systems. To use a specific conversion method, use . Requires PostGIS be compiled with PROJ support. Use to confirm you have PROJ support compiled in. If using more than one transformation, it is useful to have a functional index on the commonly used transformations to take advantage of index usage. Prior to 1.3.4, this function crashes if used with geometries that contain CURVES. This is fixed in 1.3.4+ Enhanced: 2.0.0 support for Polyhedral surfaces was introduced. Enhanced: 2.3.0 support for direct PROJ.4 text was introduced. This method implements the SQL/MM specification. SQL-MM 3: 5.1.6 This method supports Circular Strings and Curves. This function supports Polyhedral surfaces.

- ST_Union - Computes a geometry representing the point-set union of the input geometries. Beschreibung Unions the input geometries, merging geometry to produce a result geometry with no overlaps. The output may be an atomic geometry,

a MultiGeometry, or a Geometry Collection. Comes in several variants: Two-input variant: returns a geometry that is the union of two input geometries. If either input is NULL, then NULL is returned. Array variant: returns a geometry that is the union of an array of geometries. Aggregate variant: returns a geometry that is the union of a rowset of geometries. The ST_Union() function is an "aggregate" function in the terminology of PostgreSQL. That means that it operates on rows of data, in the same way the SUM() and AVG() functions do and like most aggregates, it also ignores NULL geometries. See for a non-aggregate, single-input variant. The ST_Union array and set variants use the fast Cascaded Union algorithm described in http://blog.cleverelephant.ca/2009/01/must-faster-unions-in-postgis-14.html A gridSize can be specified to work in fixed-precision space. The inputs are snapped to a grid of the given size, and the result vertices are computed on that same grid. (Requires GEOS-3.9.0 or higher) may sometimes be used in place of ST_Union, if the result is not required to be non-overlapping. ST_Collect is usually faster than ST_Union because it performs no processing on the collected geometries. Performed by the GEOS module. ST_Union creates MultiLineString and does not sew LineStrings into a single LineString. Use to sew LineStrings. NOTE: this function was formerly called GeomUnion(), which was renamed from "Union" because UNION is an SQL reserved word. Enhanced: 3.1.0 accept a gridSize parameter. Requires GEOS >= 3.9.0 to use the gridSize parameter Changed: 3.0.0 does not depend on SFCGAL. Availability: 1.4.0 - ST_Union was enhanced. ST_Union(geomarray) was introduced and also faster aggregate collection in PostgreSQL. This method implements the OGC Simple Features Implementation Specification for SQL 1.1. s2.1.1.3 Aggregate version is not explicitly defined in OGC SPEC. This method implements the SQL/MM specification. SQL-MM 3: 5.1.19 the z-index (elevation) when polygons are involved. This function supports 3d and will not drop the z-index. However, the result is computed using XY only. The result Z values are copied, averaged or interpolated.

- ST_Volume - Computes the volume of a 3D solid. If applied to surface (even closed) geometries will return 0. Beschreibung Verfügbarkeit: 2.2.0 This method needs SFCGAL backend. This function supports 3d and will not drop the z-index. This function supports Polyhedral surfaces. This function supports Triangles and Triangulated Irregular Network Surfaces (TIN). This method implements the SQL/MM specification. SQL-MM IEC 13249-3: 9.1 (same as ST_3DVolume)

- ST_WKBToSQL - Gibt einen geometrischen Datentyp (ST_Geometry) aus einer Well-known-Binary (WKB) Darstellung zurück. Ein Synonym für ST_GeomFromWKB, welches jedoch keine SRID annimmt Beschreibung This method implements the SQL/MM specification. SQL-MM 3: 5.1.36

- ST_WKTToSQL - Gibt einen spezifizierten ST_Geometry-Wert von einer Well-known-Text Darstellung (WKT) zurück. Die Bezeichnung ist ein Alias für ST_GeomFromText Beschreibung This method implements the SQL/MM specification. SQL-MM 3: 5.1.34

- ST_Within - Tests if every point of A lies in B, and their interiors have a point in common Beschreibung Returns TRUE if geometry A is within geometry B. A is within B if and only if all points of A lie inside (i.e. in the interior or boundary of) B (or equivalently, no points of A lie in the exterior of B), and the interiors of A and B have at least one point in common. For this function to make sense, the source geometries must both be of the same coordinate projection, having the same SRID. In mathematical terms: ST_Within(A, B) $\Leftrightarrow$ (A $\bigcap$ B = A) $\wedge$ (Int(A) $\bigcap$ Int(B) $\neq \varnothing$) The within relation is reflexive: every geometry is within itself. The relation is antisymmetric: if ST_Within(A,B) = true and ST_Within(B,A) = true, then the two geometries must be topologically equal (ST_Equals(A,B) = true). ST_Within is the converse of . So, ST_Within(A,B) = ST_Contains(B,A). Because the interiors must have a common point, a subtlety of the definition is that lines and points lying fully in the boundary of polygons or lines are not within the geometry. For further details see Subtleties of OGC Covers, Contains, Within. The predicate provides a more inclusive relationship. This function automatically includes a bounding box comparison that makes use of any spatial indexes that are available on the geometries. To avoid index use, use the function _ST_Within. Wird durch das GEOS Modul ausgeführt Enhanced: 2.3.0 Enhancement to PIP short-circuit for geometry extended to support MultiPoints with few points. Prior versions only supported point in polygon. Enhanced: 3.0.0 enabled support for GEOMETRYCOLLECTION Do not use this function with invalid geometries. You will get unexpected results. NOTE: this is the "allowable" version that returns a boolean, not an integer. This method implements the OGC Simple Features Implementation Specification for SQL 1.1. s2.1.1.2 // s2.1.13.3 - a.Relate(b, 'T*F**F***') This method implements the SQL/MM specification. SQL-MM 3: 5.1.30

- ST_X - Returns the X coordinate of a Point. Beschreibung Gibt die X-Koordinate eines Punktes, oder NULL wenn diese nicht vorhanden ist, zurück. Die Eingabe muss ein Punkt sein. To get the minimum and maximum X value of geometry coordinates use the functions and . This method implements the SQL/MM specification. SQL-MM 3: 6.1.3 This function supports 3d and will not drop the z-index.

- ST_Y - Returns the Y coordinate of a Point. Beschreibung Gibt die Y-Koordinate eines Punktes, oder NULL wenn diese nicht vorhanden ist, zurück. Die Eingabe muss ein Punkt sein. To get the minimum and maximum Y value of geometry coordinates

use the functions and . This method implements the OGC Simple Features Implementation Specification for SQL 1.1. This method implements the SQL/MM specification. SQL-MM 3: 6.1.4 This function supports 3d and will not drop the z-index.

- ST_Z - Returns the Z coordinate of a Point. Beschreibung Gibt die Z-Koordinate eines Punktes, oder NULL wenn diese nicht vorhanden ist, zurück. Die Eingabe muss ein Punkt sein. To get the minimum and maximum Z value of geometry coordinates use the functions and . This method implements the SQL/MM specification. This function supports 3d and will not drop the z-index.
- TG_ST_SRID - Returns the spatial reference identifier for a topogeometry. Beschreibung Returns the spatial reference identifier for the ST_Geometry as defined in spatial_ref_sys table. spatial_ref_sys table is a table that catalogs all spatial reference systems known to PostGIS and is used for transformations from one spatial reference system to another. So verifying you have the right spatial reference system identifier is important if you plan to ever transform your geometries. Availability: 3.2.0 This method implements the SQL/MM specification. SQL-MM 3: 14.1.5

12.4 PostGIS Geography Support Functions

The functions and operators given below are PostGIS functions/operators that take as input or return as output a geography data type object.

Note
Functions with a (T) are not native geodetic functions, and use a ST_Transform call to and from geometry to do the operation. As a result, they may not behave as expected when going over dateline, poles, and for large geometries or geometry pairs that cover more than one UTM zone. Basic transform - (favoring UTM, Lambert Azimuthal (North/South), and falling back on mercator in worst case scenario)

- ST_Area - Gibt den geometrischen Schwerpunkt einer Geometrie zurück.
- ST_AsBinary - Return the OGC/ISO Well-Known Binary (WKB) representation of the geometry/geography without SRID meta data.
- ST_AsEWKT - Gibt die Well-known-Text(WKT) Darstellung der Geometrie mit den SRID-Metadaten zurück.
- ST_AsGML - Gibt die Geometrie als GML-Element - Version 2 oder 3 - zurück.
- ST_AsGeoJSON - Return a geometry as a GeoJSON element.
- ST_AsKML - Gibt die Geometrie als GML-Element - Version 2 oder 3 - zurück.
- ST_AsSVG - Gibt eine Geometrie als SVG-Pfad aus.
- ST_AsText - Gibt die Well-known-Text(WKT) Darstellung der Geometrie/Geographie ohne die SRID Metadaten zurück.
- ST_Azimuth - Gibt die 2-dimenionale kürzeste Strecke zwischen zwei Geometrien als Linie zurück
- ST_Buffer - Computes a geometry covering all points within a given distance from a geometry.
- ST_Centroid - Gibt den geometrischen Schwerpunkt einer Geometrie zurück.
- ST_ClosestPoint - Returns the 2D point on g1 that is closest to g2. This is the first point of the shortest line from one geometry to the other.
- ST_CoveredBy - Tests if every point of A lies in B
- ST_Covers - Tests if every point of B lies in A
- ST_DWithin - Tests if two geometries are within a given distance
- ST_Distance - Gibt die größte 3-dimensionale Distanz zwischen zwei geometrischen Objekten als Linie zurück

- ST_GeogFromText - Gibt einen geographischen Datentyp aus einer Well-known-Text (WKT), oder einer erweiterten WKT (EWKT), Darstellung zurück.
- ST_GeogFromWKB - Erzeugt ein geographisches Objekt aus der Well-known-Binary (WKB) oder der erweiterten Well-known-Binary (EWKB) Darstellung.
- ST_GeographyFromText - Gibt einen geographischen Datentyp aus einer Well-known-Text (WKT), oder einer erweiterten WKT (EWKT), Darstellung zurück.
- = - Gibt TRUE zurück, wenn die Koordinaten und die Reihenfolge der Koordinaten der Geometrie/Geographie A und der Geometrie/Geographie B ident sind.
- ST_Intersection - Computes a geometry representing the shared portion of geometries A and B.
- ST_Intersects - Tests if two geometries intersect (they have at least one point in common)
- ST_Length - Gibt den geometrischen Schwerpunkt einer Geometrie zurück.
- ST_LineInterpolatePoint - Gibt einen oder mehrere, entlang einer Linie interpolierte Punkte zurück.
- ST_LineInterpolatePoints - Gibt einen oder mehrere, entlang einer Linie interpolierte Punkte zurück.
- ST_LineLocatePoint - Returns the fractional location of the closest point on a line to a point.
- ST_LineSubstring - Returns the part of a line between two fractional locations.
- ST_Perimeter - Returns the length of the boundary of a polygonal geometry or geography.
- ST_Project - Returns a point projected from a start point by a distance and bearing (azimuth).
- ST_Segmentize - Returns a modified geometry/geography having no segment longer than a given distance.
- ST_ShortestLine - Gibt die 2-dimenionale kürzeste Strecke zwischen zwei Geometrien als Linie zurück
- ST_Summary - Gibt eine Zusammenfassung des Inhalts einer Geometrie wieder.
- <-> - Gibt die 2D Entfernung zwischen A und B zurück.
- && - Gibt TRUE zurück, wenn die 2D Bounding Box von A die 2D Bounding Box von B schneidet.

12.5 PostGIS Raster Support Functions

The functions and operators given below are PostGIS functions/operators that take as input or return as output a raster data type object. Listed in alphabetical order.

- Box3D - Stellt das umschreibende Rechteck eines Raster als Box3D dar.
- @ - Gibt TRUE zurück, wenn das umschreibende Rechteck von A in jenem von B enthalten ist. Das umschreibende Rechteck ist in Double Precision.
- ~ - Gibt TRUE zurück, wenn das umschreibende Rechteck von A jenes von B enthält. Das umschreibende Rechteck ist in Double Precision.
- = - Gibt TRUE zurück, wenn die umschreibenden Rechtecke von A und B ident sind. Das umschreibende Rechteck ist in Double Precision.
- && - Gibt TRUE zurück, wenn das umschreibende Rechteck von A das umschreibende Rechteck von B schneidet.
- &< - Gibt TRUE zurück, wenn das umschreibende Rechteck von A links von dem von B liegt.
- &> - Gibt TRUE zurück, wenn das umschreibende Rechteck von A rechts von dem von B liegt.
- ~= - Gibt TRUE zurück wenn die Umgebungsrechtecke von "A" und "B" ident sind.

- ST_Retile - Gibt konfigurierte Kacheln eines beliebig gekachelten Rastercoverage aus.
- ST_AddBand - Gibt einen Raster mit den neu hinzugefügten Band(Bändern) aus. Der Typ , der Ausgangswert und der Index für den Speicherort des Bandes kann angegeben werden. Wenn kein Index angegeben ist, wird das Band am Ende hinzugefügt.
- ST_AsBinary/ST_AsWKB - Gibt die Well-known-Binary (WKB) Darstellung eines Rasters zurück.
- ST_AsGDALRaster - Gibt die Rasterkachel in dem ausgewiesenen Rasterformat von GDAL aus. Sie können jedes Rasterformat angeben, das von Ihrer Bibliothek unterstützt wird. Um eine Liste mit den unterstützten Formaten auszugeben, verwenden Sie bitte ST_GDALDrivers().
- ST_AsHexWKB - Gibt die Well-known-Binary (WKB) Hex-Darstellung eines Rasters zurück.
- ST_AsJPEG - Gibt die ausgewählten Bänder der Rasterkachel als einzelnes Bild (Byte-Array) im Format "Joint Photographic Exports Group" (JPEG) aus. Wenn kein Band angegeben ist und 1 oder mehr als 3 Bänder ausgewählt wurden, dann wird nur das erste Band verwendet. Wenn 3 Bänder ausgewählt wurden, werden alle 3 Bänder verwendet und auf RGB abgebildet.
- ST_AsPNG - Gibt die ausgewählten Bänder der Rasterkachel als einzelnes, übertragbares Netzwerkgraphik (PNG) Bild (Byte-Feld) aus. Wenn der Raster 1,3 oder 4 Bänder hat und keine Bänder angegeben sind, dann werden alle Bänder verwendet. Wenn der Raster 2 oder mehr als 4 Bänder hat und keine Bänder angegeben sind, dann wird nur Band 1 verwendet. Die Bänder werden in den RGB- oder den RGBA-Raum abgebildet.
- ST_AsRaster - Konvertiert den geometrischen Datentyp von PostGIS in einen PostGIS Raster.
- ST_AsTIFF - Gibt die ausgewählten Bänder des Raster als einzelnes TIFF Bild (Byte-Feld) zurück. Wenn kein Band angegeben ist oder keines der angegebenen Bänder im Raster existiert, werden alle Bänder verwendet.
- ST_Aspect - Gibt die Exposition (standardmäßig in Grad) eines Rasterbandes mit Höhen aus. Nützlich für Terrain-Analysen.
- ST_Band - Gibt einen oder mehrere Bänder eines bestehenden Rasters als neuen Raster aus. Nützlich um neue Raster aus bestehenden Rastern abzuleiten.
- ST_BandFileSize - Gibt die Dateigröße eines im Dateisystem gespeicherten Bandes aus. Wenn "bandnum" nicht angegeben ist, wird 1 angenommen.
- ST_BandFileTimestamp - Gibt den Zeitstempel eines im Dateisystem gespeicherten Bandes aus. Wenn "bandnum" nicht angegeben ist, wird 1 angenommen.
- ST_BandIsNoData - Gibt TRUE aus, wenn das Band ausschließlich aus NODATA Werten besteht.
- ST_BandMetaData - Gibt die grundlegenden Metadaten eines bestimmten Rasterbandes aus. Wenn der Parameter "bandnum" nicht angegeben ist, wird das 1ste Band angenommen.
- ST_BandNoDataValue - Gibt den NODATA Wert des gegebenen Bandes aus. Wenn der Parameter "bandnum" nicht angegeben ist, wird das 1ste Band angenommen.
- ST_BandPath - Gibt den Dateipfad aus, unter dem das Band im Dateisystem gespeichert ist. Wenn "bandnum" nicht angegeben ist, wird 1 angenommen.
- ST_BandPixelType - Gibt den Pixeltyp des angegebenen Bandes aus. Wenn der Parameter "bandnum" nicht angegeben ist, wird das 1ste Band angenommen.
- ST_Clip - Schneidet den Raster nach der Eingabegeometrie. Wenn die Bandnummer nicht angegeben ist, werden alle Bänder bearbeitet. Wenn crop nicht angegeben oder TRUE ist, wird der Ausgaberaster abgeschnitten.
- ST_ColorMap - Erzeugt aus einem bestimmten Band des Ausgangsrasters einen neuen Raster mit bis zu vier 8BUI-Bändern (Grauwert, RGB, RGBA). Wenn kein Band angegeben ist, wird Band 1 angenommen.
- ST_Contains - Gibt TRUE zurück, wenn kein Punkt des Rasters "rastB" im Äußeren des Rasters "rastA" liegt und zumindest ein Punkt im Inneren von "rastB" auch im Inneren von "rastA" liegt.
- ST_ContainsProperly - Gibt TRUE zurück, wenn "rastB" das Innere von "rastA" schneidet, aber nicht die Begrenzung oder das Äußere von "rastA".
- ST_Contour - Generates a set of vector contours from the provided raster band, using the GDAL contouring algorithm.

- ST_ConvexHull - Gibt die Geometrie der konvexen Hülle des Raster, inklusive der Pixel deren Werte gleich BandNoDataValue sind. Bei regelmäßig geformten und nicht rotierten Raster ist das Ergebnis ident mit ST_Envelope. Diese Funktion ist deshalb nur bei unregelmäßig geformten oder rotierten Raster nützlich.
- ST_Count - Gibt die Anzahl der Pixel für ein Band eines Rasters oder eines Raster-Coverage zurück. Wenn kein Band angegeben ist, wird standardmäßig Band 1 gewählt. Wenn der Parameter "exclude_nodata_value" auf TRUE gesetzt ist, werden nur Pixel mit Werten ungleich NODATA gezählt.
- ST_CountAgg - Aggregatfunktion. Gibt die Anzahl der Pixel in einem bestimmten Band der Raster aus. Wenn kein Band angegeben ist, wird Band 1 angenommen. Wenn "exclude_nodata_value" TRUE ist, werden nur die Pixel ohne NODATA Werte gezählt.
- ST_CoveredBy - Gibt TRUE zurück, wenn kein Punkt des Rasters "rastA" außerhalb des Rasters "rastB" liegt.
- ST_Covers - Gibt TRUE zurück, wenn kein Punkt des Rasters "rastB" außerhalb des Rasters "rastA" liegt.
- ST_DFullyWithin - Gibt TRUE zurück, wenn die Raster "rastA" und "rastB" zur Gänze innerhalb der angegebenen Distanz zueinander liegen.
- ST_DWithin - Gibt TRUE zurück, wenn die Raster "rastA" und "rastB" innerhalb der angegebenen Entfernung voneinander liegen.
- ST_Disjoint - Gibt TRUE zurück, wenn sich die Raster "rastA" und "rastB" räumlich nicht überschneiden.
- ST_DumpAsPolygons - Gibt geomval (geom,val) Zeilen eines Rasterbandes zurück. Wenn kein Band angegeben ist, wird die Bandnummer standardmäßig auf 1 gesetzt.
- ST_DumpValues - Gibt die Werte eines bestimmten Bandes als 2-dimensionales Feld aus.
- ST_Envelope - Stellt die Ausdehnung des Raster als Polygon dar.
- ST_FromGDALRaster - Erzeugt einen Raster aus einer von GDAL unterstützten Rasterdatei.
- ST_GeoReference - Gibt die Metadaten der Georeferenzierung, die sich üblicherweise in einem sogenannten "World File befinden, im GDAL oder ESRI Format aus. Die Standardeinstellung ist GDAL.
- ST_Grayscale - Erzeugt einen neuen Raster mit einem 8BUI-Band aus dem Ausgangsrasters und den angegebenen Bändern für Rot, Grün und Blau
- ST_HasNoBand - Gibt TRUE aus, wenn kein Band mit der angegebenen Bandnummer existiert.Gibt den Pixeltyp des angegebenen Bandes aus. Wenn keine Bandnummer angegeben ist, wird das 1ste Band angenommen.
- ST_Height - Gibt die Höhe des Rasters in Pixel aus.
- ST_HillShade - Gibt für gegebenen Horizontalwinkel, Höhenwinkel, Helligkeit und Maßstabsverhältnis die hypothetische Beleuchtung eines Höhenrasterbandes zurück.
- ST_Histogram - Gibt Datensätze aus, welche die Verteilung der Daten eines Rasters oder eines Rastercoverage darstellen. Dabei wird die Wertemenge in Klassen aufgeteilt und für jede Klasse zusammengefasst. Wenn die Anzahl der Klassen nicht angegeben ist, wird sie automatisch berechnet.
- ST_InterpolateRaster - Interpolates a gridded surface based on an input set of 3-d points, using the X- and Y-values to position the points on the grid and the Z-value of the points as the surface elevation.
- ST_Intersection - Gibt Geometry-PixelValue Paare, oder einen Raster aus, der durch die Schnittmenge der beiden Raster bestimmt wird, oder durch die geometrische Verschneidung einer Vektorisierung des Rasters mit einem geometrischen Datentyp.
- ST_Intersects - Gibt TRUE zurück, wenn sich die Raster "rastA" und "rastB" nicht räumlich überschneiden.
- ST_IsEmpty - Gibt TRUE zurück, wenn der Raster leer ist (width = 0 and height = 0). Andernfalls wird FALSE zurückgegeben.
- ST_MakeEmptyCoverage - Bedeckt die georeferenzierte Fläche mit einem Gitter aus leeren Rasterkacheln.

- ST_MakeEmptyRaster - Gibt einen leeren Raster (ohne Bänder), mit den gegebenen Dimensionen (width & height), upperleft X und Y, Pixelgröße, Rotation (scalex, scaley, skewx & skewy) und Koordinatenreferenzsystem (SRID), zurück. Wenn ein Raster übergeben wird, dann wird ein neuer Raster mit der selben Größe, Ausrichtung und SRID zurückgegeben. Wenn SRID nicht angegeben ist, wird das Koordinatenreferenzsystem auf "unknown" (0) gesetzt.
- ST_MapAlgebra (callback function version) - Die Version mit der Rückruffunktion - Gibt für einen oder mehrere Eingaberaster einen Raster mit einem Band, den Bandindizes und einer vom Anwender vorgegebenen Rückruffunktion zurück.
- ST_MapAlgebraExpr - Version mit 1 Rasterband: Erzeugt ein neues Rasterband, dass über eine gültige, algebraische PostgreSQL Operation für ein Rasterband mit gegebenen Pixeltyp erstellt wird. Wenn kein Band bestimmt ist, wird Band 1 angenommen.
- ST_MapAlgebraExpr - Version mit 2 Rasterbändern: Erstellt einen neuen Einzelbandraster, indem eine gültige algebraische PostgreSQL Funktion auf die zwei Ausgangsrasterbänder und den entsprechenden Pixeltyp angewendet wird. Wenn keine Bandnummern angegeben sind, wird von jedem Raste Band 1 angenommen. Der Ergebnisraster wird nach dem Gitter des ersten Raster ausgerichtet (Skalierung, Versatz und Eckpunkte der Pixel) und hat die Ausdehnung, welche durch den Parameter "extenttype" definiert ist. Der Parameter "extenttype" kann die Werte INTERSECTION, UNION, FIRST, SECOND annehmen.
- ST_MapAlgebraFct - Version mit 1 Rasterband: Erzeugt ein neues Rasterband, dass über eine gültige PostgreSQL Funktion für ein gegebenes Rasterband und Pixeltyp erstellt wird. Wenn kein Band bestimmt ist, wird Band 1 angenommen.
- ST_MapAlgebraFct - Version mit 2 Rasterbändern: Erstellt einen neuen Einzelbandraster, indem eine gültige PostgreSQL Funktion auf die 2 gegebenen Rasterbänder und den entsprechenden Pixeltyp angewendet wird. Wenn kein Band bestimmt ist, wird Band 1 angenommen. Wenn der "Extent"-Typ nicht angegeben ist, wird standardmäßig INTERSECTION angenommen.
- ST_MapAlgebraFctNgb - Version mit 1em Band: Map Algebra Nearest Neighbor mit einer benutzerdefinierten PostgreSQL Funktion. Gibt einen Raster zurück, dessen Werte sich aus einer benutzerdefinierte PL/pgsql Funktion ergeben, welche die Nachbarschaftswerte des Ausgangsrasterbandes einbezieht.
- ST_MapAlgebra (expression version) - Version mit Ausdrücken - Gibt für einen oder zwei Ausgangsraster, Bandindizes und einer oder mehreren vom Anwender vorgegebenen SQL-Ausdrücken, einen Raster mit einem Band zurück.
- ST_MemSize - Gibt den Platzbedarf des Rasters (in Byte) aus.
- ST_MetaData - Gibt die wesentlichen Metadaten eines Rasterobjektes, wie Zellgröße, Rotation (Versatz) etc. aus
- ST_MinConvexHull - Gibt die Geometrie der konvexen Hülle des Raster aus, wobei Pixel mit NODATA ausgenommen werden.
- ST_NearestValue - Gibt den nächstgelegenen nicht NODATA Wert eines bestimmten Pixels aus, das über "columnx" und "rowy" oder durch eine Punktgeometrie - im gleichen Koordinatenreferenzsystem wie der Raster - ausgewählt wird.
- ST_Neighborhood - Gibt ein 2-D Feld in "Double Precision" aus, das sich aus nicht NODATA Werten um ein bestimmtes Pixel herum zusammensetzt. Das Pixel Kann über "columnx" und "rowy" oder über eine Punktgeometrie - im gleichen Koordinatenreferenzsystem wie der Raster - ausgewählt werden.
- ST_NotSameAlignmentReason - Gibt eine Meldung aus, die angibt ob die Raster untereinander ausgerichtet sind oder nicht und warum wenn nicht.
- ST_NumBands - Gibt die Anzahl der Bänder des Rasters aus.
- ST_Overlaps - Gibt TRUE zurück, wenn sich die Raster "rastA" und "rastB" schneiden, aber ein Raster den anderen nicht zur Gänze enthält.
- ST_PixelAsCentroid - Gibt den geometrischen Schwerpunkt (Punktgeometrie) der Fläche aus, die durch das Pixel repräsentiert wird.
- ST_PixelAsCentroids - Gibt den geometrischen Schwerpunkt (Punktgeometrie) für jedes Pixel des Rasterbandes zurück, zusammen mit dem Zellwert und den X- und Y-Rasterkoordinaten eines jeden Pixels. Die Koordinaten der Punkte entsprechen dem geometrischen Schwerpunkt der Pixel.
- ST_PixelAsPoint - Gibt eine Punktgeometrie der oberen linken Ecke des Rasters zurück.
- ST_PixelAsPoints - Gibt eine Punktgeometrie für jedes Pixel des Rasterbandes zurück, zusammen mit dem Zellwert und den X- und Y-Rasterkoordinaten eines jeden Pixels. Die Koordinaten der Punkte entsprechen dem oberen linken Eck der Pixel.

- ST_PixelAsPolygon - Gibt die Polygongeometrie aus, die das Pixel einer bestimmten Zeile und Spalte begrenzt.
- ST_PixelAsPolygons - Gibt die umhüllende Polygongeometrie, den Zellwert, sowie die X- und Y-Rasterkoordinate für jedes Pixel aus.
- ST_PixelHeight - Gibt die Pixelhöhe in den Einheiten des Koordinatenreferenzsystem aus.
- ST_PixelOfValue - Gibt die columnx- und rowy-Koordinaten jener Pixel aus, deren Zellwert gleich dem gesuchten Wert ist.
- ST_PixelWidth - Gibt die Pixelbreite in den Einheiten des Koordinatenreferenzsystems aus.
- ST_Polygon - Gibt eine Geometrie mit Mehrfachpolygonen zurück, die aus der Vereinigung von Pixel mit demselben Zellwert gebildet werden. Pixel mit NODATA Werten werden nicht berücksichtigt. Wenn keine Band angegeben ist, wird die Bandnummer standardmäßig auf 1 gesetzt.
- ST_Quantile - Berechnet die Quantile eines Rasters oder einer Rastercoverage Tabelle im Kontext von Stichproben oder Bevölkerung. Dadurch kann untersucht werden, ob ein Wert bei 25%, 50% oder 75% Perzentil des Rasters liegt.
- ST_RastFromHexWKB - Gibt einen Rasterwert von einer Well-known-Binary (WKB) Hex-Darstellung eines Rasters zurück.
- ST_RastFromWKB - Gibt einen Rasterwert von einer Well-known-Binary (WKB) Darstellung eines Rasters zurück.
- ST_RasterToWorldCoord - Gibt die obere linke Ecke des Rasters in geodätischem X und Y (Länge und Breite) für eine gegebene Spalte und Zeile aus. Spalte und Zeile wird von 1 aufwärts gezählt.
- ST_RasterToWorldCoordX - Gibt die geodätische X Koordinate links oberhalb des Rasters, der Spalte und der Zeile aus. Die Nummerierung der Spalten und Zeilen beginnt mit 1.
- ST_RasterToWorldCoordY - Gibt die geodätische Y Koordinate links oberhalb des Rasters, der Spalte und der Zeile aus. Die Nummerierung der Spalten und Zeilen beginnt mit 1.
- ST_Reclass - Erstellt einen neuen Raster, der aus neu klassifizierten Bändern des Originalraster besteht. Das Band "nband" ist jenes das verändert werden soll. Wenn "nband" nicht angegeben ist, wird "Band 1" angenommen. Alle anderen Bänder bleiben unverändert. Anwendungsfall: zwecks einfacherer Visualisierung ein 16BUI-Band in ein 8BUI-Band konvertieren und so weiter.
- ST_Resample - Skaliert einen Raster mit einem bestimmten Algorithmus, neuen Dimensionen, einer beliebigen Gitterecke und über Parameter zur Georeferenzierung des Rasters, die angegeben oder von einem anderen Raster übernommen werden können.
- ST_Rescale - Resample a raster by adjusting only its scale (or pixel size). New pixel values are computed using the NearestNeighbor (english or american spelling), Bilinear, Cubic, CubicSpline, Lanczos, Max or Min resampling algorithm. Default is NearestNeighbor.
- ST_Resize - Ändert die Zellgröße - width/height - eines Rasters
- ST_Reskew - Skaliert einen Raster, indem lediglich der Versatz (oder Rotationsparameter) angepasst wird. Neue Pixelwerte werden über NearestNeighbor, bilinear, kubisch, CubicSpline oder mit dem Lanczos-Filter errechnet. Die Standardeinstellung ist NearestNeighbor.
- ST_Rotation - Gibt die Rotation des Rasters im Bogenmaß aus.
- ST_Roughness - Gibt einen Raster mit der berechneten "Rauhigkeit" des DHM zurück.
- ST_SRID - Gibt den Identifikator des Koordinatenreferenzsystems des Rasters aus, das in der Tabelle "spatial_ref_sys" definiert ist.
- ST_SameAlignment - Gibt TRUE zurück, wenn die Raster die selbe Rotation, Skalierung, Koordinatenreferenzsystem und Versatz (Pixel können auf dasselbe Gitter gelegt werden, ohne dass die Gitterlinien durch die Pixel schneiden) aufweisen. Wenn nicht, wird FALSE und eine Bechreibung des Problems ausgegeben.
- ST_ScaleX - Gibt die X-Komponente der Pixelbreite in den Einheiten des Koordinatenreferenzsystems aus.
- ST_ScaleY - Gibt die Y-Komponente der Pixelhöhe in den Einheiten des Koordinatenreferenzsystems aus.

- ST_SetBandIndex - Aktualisiert die externe Bandnummer eines out-db Bandes.
- ST_SetBandIsNoData - Setzt die Flag "isnodata" für das Band auf TRUE.
- ST_SetBandNoDataValue - Setzt den NODATA Wert eines Bandes. Wenn kein Band angegeben ist, wird Band 1 angenommen. Falls ein Band keinen NODATA Wert aufweisen soll, übergeben Sie bitte für den Parameter "nodatavalue" NULL.
- ST_SetBandPath - Aktualisiert den externen Dateipfad und die Bandnummer eines out-db Bandes.
- ST_SetGeoReference - Georeferenziert einen Raster über 6 Parameter in einem einzigen Aufruf. Die Zahlen müssen durch Leerzeichen getrennt sein. Die Funktion akzeptiert die Eingabe im Format von 'GDAL' und von 'ESRI'. Der Standardwert ist GDAL.
- ST_SetM - Returns a geometry with the same X/Y coordinates as the input geometry, and values from the raster copied into the M dimension using the requested resample algorithm.
- ST_SetRotation - Bestimmt die Rotation des Rasters in Radiant.
- ST_SetSRID - Setzt die SRID eines Rasters auf einen bestimmten Ganzzahlwert. Die SRID wird in der Tabelle "spatial_ref_sys" definiert.
- ST_SetScale - Setzt die X- und Y-Größe der Pixel in den Einheiten des Koordinatenreferenzsystems. Entweder eine Zahl pro Pixel oder Breite und Höhe.
- ST_SetSkew - Setzt den georeferenzierten X- und Y-Versatz (oder den Rotationsparameter). Wenn nur ein Wert übergeben wird, werden X und Y auf den selben Wert gesetzt.
- ST_SetUpperLeft - Setzt den Wert der oberen linke Ecke des Rasters auf die projizierten X- und Y-Koordinaten.
- ST_SetValue - Setzt den Wert für ein Pixel eines Bandes, das über columnx und rowy festgelegt wird, oder für die Pixel die eine bestimmte Geometrie schneiden, und gibt den veränderten Raster zurück. Die Bandnummerierung beginnt mit 1; wenn die Bandnummer nicht angegeben ist, wird 1 angenommen.
- ST_SetValues - Gibt einen Raster zurück, der durch das Setzen der Werte eines bestimmten Bandes verändert wurde.
- ST_SetZ - Returns a geometry with the same X/Y coordinates as the input geometry, and values from the raster copied into the Z dimension using the requested resample algorithm.
- ST_SkewX - Gibt den georeferenzierten Versatz in X-Richtung (oder den Rotationsparameter) aus.
- ST_SkewY - Gibt den georeferenzierten Versatz in Y-Richtung (oder den Rotationsparameter) aus.
- ST_Slope - Gibt die Neigung (standardmäßig in Grad) eines Höhenrasterbandes zurück. Nützlich für Terrain-Analysen.
- ST_SnapToGrid - Skaliert einen Raster durch Fangen an einem Führungsgitter. Neue Pixelwerte werden über NearestNeighbor, bilinear, kubisch, CubicSpline oder mit dem Lanczos-Filter errechnet. Die Standardeinstellung ist NearestNeighbor.
- ST_Summary - Gibt eine textliche Zusammenfassung des Rasterinhalts zurück.
- ST_SummaryStats - Gibt eine zusammenfassende Statistik aus, bestehend aus der Anzahl, der Summe, dem arithmetischen Mittel, der Standardabweichung, dem Minimum und dem Maximum der Werte eines Rasterbandes oder eines Rastercoverage. Wenn kein Band angegeben ist, wird Band 1 angenommen.
- ST_SummaryStatsAgg - Aggregatfunktion. Gibt eine zusammenfassende Statistik aus, die aus der Anzahl, der Summe, dem arithmetischen Mittel, dem Minimum und dem Maximum der Werte eines bestimmten Bandes eines Rastersatzes besteht. Wenn kein Band angegeben ist, wird Band 1 angenommen.
- ST_TPI - Berechnet den "Topographic Position Index" eines Raster.
- ST_TRI - Gibt einen Raster mit errechneten Geländerauheitsindex aus.
- ST_Tile - Gibt Raster, die aus einer Teilungsoperation des Eingaberasters resultieren, mit den gewünschten Dimensionen aus.
- ST_Touches - Gibt TRUE zurück, wenn rastA und rastB zumindest einen Punkt gemeinsam haben sich aber nicht überschneiden.

- ST_Transform - Projiziert einen Raster von einem bekannten Koordinatenreferenzsystem in ein anderes bekanntes Koordinatenreferenzsystem um. Die Optionen für die Skalierung sind NearestNeighbor, Bilinear, Cubisch, CubicSpline und der Lanczos-Filter, die Standardeinstellung ist NearestNeighbor.
- ST_Union - Gibt die Vereinigung mehrerer Rasterkacheln in einem einzelnen Raster mit mehreren Bändern zurück.
- ST_UpperLeftX - Gibt die obere linke X-Koordinate des Rasters im Koordinatenprojektionssystem aus.
- ST_UpperLeftY - Gibt die obere linke Y-Koordinate des Rasters im Koordinatenprojektionssystem aus.
- ST_Value - Gibt den Zellwert eines Pixels aus, das über columnx und rowy oder durch einen bestimmten geometrischen Punkt angegeben wird. Die Bandnummern beginnen mit 1 und wenn keine Bandnummer angegeben ist, dann wird Band 1 angenommen. Wenn exclude_nodata_value auf FALSE gesetzt ist, werden auch die Pixel mit einem nodata Wert mit einbezogen. Wenn exclude_nodata_value nicht übergeben wird, dann wird er über die Metadaten des Rasters ausgelesen.
- ST_ValueCount - Gibt Datensätze aus, die den Zellwert und die Anzahl der Pixel eines Rasterbandes (oder Rastercoverageandes) für gegebene Werte enthalten. Wenn kein Band angegeben ist, wird Band 1 angenommen. Pixel mit dem Wert NODATA werden standardmäßig nicht gezählt; alle anderen Pixelwerte des Bandes werden ausgegeben und auf die nächste Ganzzahl gerundet.
- ST_Width - Gibt die Breite des Rasters in Pixel aus.
- ST_Within - Gibt TRUE zurück, wenn kein Punkt des Rasters "rastA" außerhalb des Rasters "rastB" liegt und zumindest ein Punkt im Inneren von "rastA" auch im Inneren von "rastB" liegt.
- ST_WorldToRasterCoord - Gibt für ein geometrisches X und Y (geographische Länge und Breite) oder für eine Punktgeometrie im Koordinatenreferenzsystem des Rasters, die obere linke Ecke als Spalte und Zeile aus.
- ST_WorldToRasterCoordX - Gibt für eine Punktgeometrie (pt) oder eine globale X- und Y-Koordinate (xw, yw) die Rasterspalte im globalen Koordinatenreferenzsystem des Rasters aus.
- ST_WorldToRasterCoordY - Gibt für eine Punktgeometrie (pt) oder eine globale X- und Y-Koordinate (xw, yw) die Rasterzeile im globalen Koordinatenreferenzsystem des Rasters aus.
- UpdateRasterSRID - Änderung der SRID aller Raster in der vom Anwender angegebenen Spalte und Tabelle.

12.6 PostGIS Geometry / Geography / Raster Dump Functions

The functions given below are PostGIS functions that take as input or return as output a set of or single geometry_dump or geomval data type object.

- ST_DumpAsPolygons - Gibt geomval (geom,val) Zeilen eines Rasterbandes zurück. Wenn kein Band angegeben ist, wird die Bandnummer standardmäßig auf 1 gesetzt.
- ST_Intersection - Gibt Geometry-PixelValue Paare, oder einen Raster aus, der durch die Schnittmenge der beiden Raster bestimmt wird, oder durch die geometrische Verschneidung einer Vektorisierung des Rasters mit einem geometrischen Datentyp.

12.7 PostGIS Box Functions

The functions given below are PostGIS functions that take as input or return as output the box* family of PostGIS spatial types. The box family of types consists of box2d, and box3d

- Box2D - Returns a BOX2D representing the 2D extent of a geometry.
- Box3D - Returns a BOX3D representing the 3D extent of a geometry.
- Box3D - Stellt das umschreibende Rechteck eines Raster als Box3D dar.
- ST_3DExtent - Aggregate function that returns the 3D bounding box of geometries.

- ST_3DMakeBox - Creates a BOX3D defined by two 3D point geometries.
- ST_AsMVTGeom - Transforms a geometry into the coordinate space of a MVT tile.
- ST_AsTWKB - Gibt die Geometrie als TWKB, aka "Tiny Well-known Binary" zurück
- ST_Box2dFromGeoHash - Gibt die BOX2D einer GeoHash Zeichenkette zurück.
- ST_ClipByBox2D - Computes the portion of a geometry falling within a rectangle.
- ST_EstimatedExtent - Returns the estimated extent of a spatial table.
- ST_Expand - Returns a bounding box expanded from another bounding box or a geometry.
- ST_Extent - Aggregate function that returns the bounding box of geometries.
- ST_MakeBox2D - Creates a BOX2D defined by two 2D point geometries.
- ST_XMax - Returns the X maxima of a 2D or 3D bounding box or a geometry.
- ST_XMin - Returns the X minima of a 2D or 3D bounding box or a geometry.
- ST_YMax - Returns the Y maxima of a 2D or 3D bounding box or a geometry.
- ST_YMin - Returns the Y minima of a 2D or 3D bounding box or a geometry.
- ST_ZMax - Returns the Z maxima of a 2D or 3D bounding box or a geometry.
- ST_ZMin - Returns the Z minima of a 2D or 3D bounding box or a geometry.
- RemoveUnusedPrimitives - Removes topology primitives which not needed to define existing TopoGeometry objects.
- ValidateTopology - Liefert eine Menge validatetopology_returntype Objekte, die Probleme mit der Topologie beschreiben.
- ~(box2df,box2df) - Gibt TRUE zurück, wenn eine 2D float precision bounding box (BOX2DF) eine andere 2D float precision bounding box (BOX2DF) enthält.
- ~(box2df,geometry) - Gibt TRUE zurück, wenn eine 2D float precision bounding box (BOX2DF) die 2D Bounding Box einer Geometrie enthält.
- ~(geometry,box2df) - Gibt TRUE zurück, wenn die 2D bounding box einer Geometrie eine 2D float precision bounding box (GIDX) enthält.
- @(box2df,box2df) - Gibt TRUE zurück, wenn eine 2D float precision bounding box (BOX2DF) innerhalb einer anderen 2D float precision bounding box enthalten ist.
- @(box2df,geometry) - Gibt TRUE zurück, wenn eine 2D float precision bounding box (BOX2DF) in der 2D Bounding Box einer Geometrie enthalten ist..
- @(geometry,box2df) - Gibt TRUE zurück, wenn die 2D Bounding Box einer Geometrie in einer 2D float precision Bbounding Box (BOX2DF) enthalten ist.
- &&(box2df,box2df) - Gibt TRUE zurück, wenn sich zwei 2D float precision Bounding Boxes (BOX2DF) überschneiden.
- &&(box2df,geometry) - Gibt TRUE zurück, wenn eine 2D float precision bounding box (BOX2DF) eine Geometrie (cached) 2D bounding box schneidet.
- &&(geometry,box2df) - Gibt TRUE zurück, wenn sich die 2D Bounding Box (cached) einer Geometrie mit einer 2D Bounding Box mit Gleitpunktgenauigkeit (BOX2DF) überschneidet.

12.8 PostGIS Functions that support 3D

The functions given below are PostGIS functions that do not throw away the Z-Index.

- AddGeometryColumn - Entfernt eine Geometriespalte aus einer räumlichen Tabelle.
- Box3D - Returns a BOX3D representing the 3D extent of a geometry.
- DropGeometryColumn - Entfernt eine Geometriespalte aus einer räumlichen Tabelle.
- GeometryType - Gibt den Geometrietyp des ST_Geometry Wertes zurück.
- ST_3DArea - Computes area of 3D surface geometries. Will return 0 for solids.
- ST_3DClosestPoint - Gibt den 3-dimensionalen Punkt auf g1 zurück, der den kürzesten Abstand zu g2 hat. Dies ist der Anfangspunkt des kürzesten Abstands in 3D.
- ST_3DConvexHull - Berechnet die konvexe Hülle einer Geometrie.
- ST_3DDFullyWithin - Tests if two 3D geometries are entirely within a given 3D distance
- ST_3DDWithin - Tests if two 3D geometries are within a given 3D distance
- ST_3DDifference - Perform 3D difference
- ST_3DDistance - Für den geometrischen Datentyp. Es wird der geringste 3-dimensionale kartesische Abstand (basierend auf dem Koordinatenreferenzsystem) zwischen zwei geometrischen Objekten in projizierten Einheiten zurückgegeben.
- ST_3DExtent - Aggregate function that returns the 3D bounding box of geometries.
- ST_3DIntersection - Perform 3D intersection
- ST_3DIntersects - Tests if two geometries spatially intersect in 3D - only for points, linestrings, polygons, polyhedral surface (area)
- ST_3DLength - Gibt den geometrischen Schwerpunkt einer Geometrie zurück.
- ST_3DLineInterpolatePoint - Gibt einen oder mehrere, entlang einer Linie interpolierte Punkte zurück.
- ST_3DLongestLine - Gibt die größte 3-dimensionale Distanz zwischen zwei geometrischen Objekten als Linie zurück
- ST_3DMaxDistance - Für den geometrischen Datentyp. Gibt die maximale 3-dimensionale kartesische Distanz (basierend auf dem Koordinatenreferenzsystem) zwischen zwei geometrischen Objekten in projizierten Einheiten zurück.
- ST_3DPerimeter - Gibt den geometrischen Schwerpunkt einer Geometrie zurück.
- ST_3DShortestLine - Gibt den kürzesten 3-dimensionalen Abstand zwischen zwei geometrischen Objekten als Linie zurück
- ST_3DUnion - Perform 3D union.
- ST_AddMeasure - Interpolates measures along a linear geometry.
- ST_AddPoint - Fügt einem Linienzug einen Punkt hinzu.
- ST_Affine - Apply a 3D affine transformation to a geometry.
- ST_ApproximateMedialAxis - Berechnet die konvexe Hülle einer Geometrie.
- ST_AsBinary - Return the OGC/ISO Well-Known Binary (WKB) representation of the geometry/geography without SRID meta data.
- ST_AsEWKB - Return the Extended Well-Known Binary (EWKB) representation of the geometry with SRID meta data.
- ST_AsEWKT - Gibt die Well-known-Text(WKT) Darstellung der Geometrie mit den SRID-Metadaten zurück.
- ST_AsGML - Gibt die Geometrie als GML-Element - Version 2 oder 3 - zurück.

- ST_AsGeoJSON - Return a geometry as a GeoJSON element.
- ST_AsHEXEWKB - Gibt eine Geometrie im HEXEWKB Format (als Text) aus; verwendet entweder die Little-Endian (NDR) oder die Big-Endian (XDR) Zeichenkodierung.
- ST_AsKML - Gibt die Geometrie als GML-Element - Version 2 oder 3 - zurück.
- ST_AsX3D - Gibt eine Geometrie im X3D XML Knotenelement-Format zurück: ISO-IEC-19776-1.2-X3DEncodings-XML
- ST_Boundary - Gibt die abgeschlossene Hülle aus der kombinierten Begrenzung der Geometrie zurück.
- ST_BoundingDiagonal - Gibt die Diagonale des Umgebungsdreiecks der angegebenen Geometrie zurück.
- ST_CPAWithin - Tests if the closest point of approach of two trajectories is within the specified distance.
- ST_ChaikinSmoothing - Returns a smoothed version of a geometry, using the Chaikin algorithm
- ST_ClosestPointOfApproach - Returns a measure at the closest point of approach of two trajectories.
- ST_Collect - Creates a GeometryCollection or Multi* geometry from a set of geometries.
- ST_ConstrainedDelaunayTriangles - Return a constrained Delaunay triangulation around the given input geometry.
- ST_ConvexHull - Berechnet die konvexe Hülle einer Geometrie.
- ST_CoordDim - Gibt die Dimension der Koordinaten für den Wert von ST_Geometry zurück.
- ST_CurveToLine - Converts a geometry containing curves to a linear geometry.
- ST_DelaunayTriangles - Returns the Delaunay triangulation of the vertices of a geometry.
- ST_Difference - Computes a geometry representing the part of geometry A that does not intersect geometry B.
- ST_DistanceCPA - Returns the distance between the closest point of approach of two trajectories.
- ST_Dump - Returns a set of geometry_dump rows for the components of a geometry.
- ST_DumpPoints - Gibt eine Zusammenfassung des Inhalts einer Geometrie wieder.
- ST_DumpRings - Returns a set of geometry_dump rows for the exterior and interior rings of a Polygon.
- ST_DumpSegments - Gibt eine Zusammenfassung des Inhalts einer Geometrie wieder.
- ST_EndPoint - Gibt die Anzahl der Stützpunkte eines ST_LineString oder eines ST_CircularString zurück.
- ST_ExteriorRing - Gibt die Anzahl der inneren Ringe einer Polygongeometrie aus.
- ST_Extrude - Extrude a surface to a related volume
- ST_FlipCoordinates - Returns a version of a geometry with X and Y axis flipped.
- ST_Force2D - Die Geometrien in einen "2-dimensionalen Modus" zwingen.
- ST_ForceCurve - Wandelt einen geometrischen in einen Kurven Datentyp um, soweit anwendbar.
- ST_ForceLHR - Force LHR orientation
- ST_ForcePolygonCCW - Richtet alle äußeren Ringe gegen den Uhrzeigersinn und alle inneren Ringe mit dem Uhrzeigersinn aus.
- ST_ForcePolygonCW - Richtet alle äußeren Ringe im Uhrzeigersinn und alle inneren Ringe gegen den Uhrzeigersinn aus.
- ST_ForceRHR - Orientiert die Knoten in einem Polygon so, dass sie der Drei-Finger-Regel folgen.
- ST_ForceSFS - Erzwingt, dass Geometrien nur vom Typ SFS 1.1 sind.
- ST_Force_3D - Zwingt die Geometrien in einen XYZ Modus. Dies ist ein Alias für ST_Force3DZ.
- ST_Force_3DZ - Zwingt die Geometrien in einen XYZ Modus.

- ST_Force_4D - Zwingt die Geometrien in einen XYZM Modus.
- ST_Force_Collection - Wandelt eine Geometrie in eine GEOMETRYCOLLECTION um.
- ST_GeomFromEWKB - Gibt einen geometrischen Datentyp (ST_Geometry) aus einer Well-known-Binary (WKB) Darstellung zurück.
- ST_GeomFromEWKT - Gibt einen spezifizierten ST_Geometry-Wert von einer erweiterten Well-known-Text Darstellung (EWKT) zurück.
- ST_GeomFromGML - Nimmt als Eingabe eine GML-Darstellung der Geometrie und gibt ein geometrisches PostGIS-Objekt aus.
- ST_GeomFromGeoJSON - Nimmt als Eingabe eine GeoJSON-Darstellung der Geometrie und gibt ein geometrisches PostGIS-Objekt aus.
- ST_GeomFromKML - Nimmt als Eingabe eine KML-Darstellung der Geometrie und gibt ein geometrisches PostGIS-Objekt aus.
- ST_GeometricMedian - Gibt den geometrischen Median eines Mehrfachpunktes zurück.
- ST_GeometryN - Gibt den Geometrietyp des ST_Geometry Wertes zurück.
- ST_GeometryType - Gibt den Geometrietyp des ST_Geometry Wertes zurück.
- ST_HasArc - Tests if a geometry contains a circular arc
- ST_InteriorRingN - Gibt die Anzahl der inneren Ringe einer Polygongeometrie aus.
- ST_InterpolatePoint - Für einen gegebenen Punkt wird die Kilometrierung auf dem nächstliegenden Punkt einer Geometrie zurück.
- ST_Intersection - Computes a geometry representing the shared portion of geometries A and B.
- ST_IsClosed - Gibt den Wert TRUE zurück, wenn die Anfangs- und Endpunkte des LINESTRING's zusammenfallen. Bei polyedrischen Oberflächen, wenn sie geschlossen (volumetrisch) sind.
- ST_IsCollection - Gibt den Wert TRUE zurück, falls es sich bei der Geometrie um eine leere GeometryCollection, Polygon, Point etc. handelt.
- ST_IsPlanar - Check if a surface is or not planar
- ST_IsPolygonCCW - Gibt TRUE zurück, wenn alle äußeren Ringe gegen den Uhrzeigersinn orientiert sind und alle inneren Ringe im Uhrzeigersinn ausgerichtet sind.
- ST_IsPolygonCW - Gibt den Wert TRUE zurück, wenn alle äußeren Ringe im Uhrzeigersinn und alle inneren Ringe gegen den Uhrzeigersinn ausgerichtet sind.
- ST_IsSimple - Gibt den Wert (TRUE) zurück, wenn die Geometrie keine irregulären Stellen, wie Selbstüberschneidungen oder Selbstberührungen, aufweist.
- ST_IsSolid - Test if the geometry is a solid. No validity check is performed.
- ST_IsValidTrajectory - Tests if the geometry is a valid trajectory.
- ST_Length_Spheroid - Gibt den geometrischen Schwerpunkt einer Geometrie zurück.
- ST_LineFromMultiPoint - Erzeugt einen LineString aus einer MultiPoint Geometrie.
- ST_LineInterpolatePoint - Gibt einen oder mehrere, entlang einer Linie interpolierte Punkte zurück.
- ST_LineInterpolatePoints - Gibt einen oder mehrere, entlang einer Linie interpolierte Punkte zurück.
- ST_LineSubstring - Returns the part of a line between two fractional locations.
- ST_LineToCurve - Converts a linear geometry to a curved geometry.

- ST_LocateBetweenElevations - Returns the portions of a geometry that lie in an elevation (Z) range.
- ST_M - Returns the M coordinate of a Point.
- ST_MakeLine - Erzeugt einen Linienzug aus einer Punkt-, Mehrfachpunkt- oder Liniengeometrie.
- ST_MakePoint - Erzeugt eine 2D-, 3DZ- oder 4D-Punktgeometrie.
- ST_MakePolygon - Creates a Polygon from a shell and optional list of holes.
- ST_MakeSolid - Cast the geometry into a solid. No check is performed. To obtain a valid solid, the input geometry must be a closed Polyhedral Surface or a closed TIN.
- ST_MakeValid - Attempts to make an invalid geometry valid without losing vertices.
- ST_MemSize - Gibt den Geometrietyp des ST_Geometry Wertes zurück.
- ST_MemUnion - Aggregate function which unions geometries in a memory-efficent but slower way
- ST_NDims - Gibt die Dimension der Koordinaten für den Wert von ST_Geometry zurück.
- ST_NPoints - Gibt die Anzahl der Punkte (Knoten) einer Geometrie zurück.
- ST_NRings - Gibt die Anzahl der inneren Ringe einer Polygongeometrie aus.
- ST_Node - Nodes a collection of lines.
- ST_NumGeometries - Gibt die Anzahl der Punkte einer Geometrie zurück. Funktioniert für alle Geometrien.
- ST_NumPatches - Gibt die Anzahl der Maschen einer polyedrischen Oberfläche aus. Gibt NULL zurück, wenn es sich nicht um polyedrische Geometrien handelt.
- ST_Orientation - Determine surface orientation
- ST_PatchN - Gibt den Geometrietyp des ST_Geometry Wertes zurück.
- ST_PointFromWKB - Erzeugt eine Geometrie mit gegebener SRID von WKB.
- ST_PointN - Gibt die Anzahl der Stützpunkte eines ST_LineString oder eines ST_CircularString zurück.
- ST_PointOnSurface - Computes a point guaranteed to lie in a polygon, or on a geometry.
- ST_Points - Gibt einen MultiPoint zurück, welcher alle Koordinaten einer Geometrie enthält.
- ST_Polygon - Creates a Polygon from a LineString with a specified SRID.
- ST_RemovePoint - Remove a point from a linestring.
- ST_RemoveRepeatedPoints - Returns a version of a geometry with duplicate points removed.
- ST_Reverse - Gibt die Geometrie in umgekehrter Knotenreihenfolge zurück.
- ST_Rotate - Rotates a geometry about an origin point.
- ST_RotateX - Rotates a geometry about the X axis.
- ST_RotateY - Rotates a geometry about the Y axis.
- ST_RotateZ - Rotates a geometry about the Z axis.
- ST_Scale - Scales a geometry by given factors.
- ST_Scroll - Change start point of a closed LineString.
- ST_SetPoint - Einen Punkt eines Linienzuges durch einen gegebenen Punkt ersetzen.
- ST_ShiftLongitude - Shifts the longitude coordinates of a geometry between -180..180 and 0..360.
- ST_SnapToGrid - Fängt alle Punkte der Eingabegeometrie auf einem regelmäßigen Gitter.

- ST_StartPoint - Returns the first point of a LineString.
- ST_StraightSkeleton - Berechnet die konvexe Hülle einer Geometrie.
- ST_SwapOrdinates - Gibt eine Version der Ausgangsgeometrie zurück, in der die angegebenen Ordinatenwerte ausgetauscht werden.
- ST_SymDifference - Computes a geometry representing the portions of geometries A and B that do not intersect.
- ST_Tesselate - Perform surface Tesselation of a polygon or polyhedralsurface and returns as a TIN or collection of TINS
- ST_TransScale - Translates and scales a geometry by given offsets and factors.
- ST_Translate - Translates a geometry by given offsets.
- ST_UnaryUnion - Computes the union of the components of a single geometry.
- ST_Union - Computes a geometry representing the point-set union of the input geometries.
- ST_Volume - Computes the volume of a 3D solid. If applied to surface (even closed) geometries will return 0.
- ST_WrapX - Versammelt eine Geometrie um einen X-Wert
- ST_X - Returns the X coordinate of a Point.
- ST_XMax - Returns the X maxima of a 2D or 3D bounding box or a geometry.
- ST_XMin - Returns the X minima of a 2D or 3D bounding box or a geometry.
- ST_Y - Returns the Y coordinate of a Point.
- ST_YMax - Returns the Y maxima of a 2D or 3D bounding box or a geometry.
- ST_YMin - Returns the Y minima of a 2D or 3D bounding box or a geometry.
- ST_Z - Returns the Z coordinate of a Point.
- ST_ZMax - Returns the Z maxima of a 2D or 3D bounding box or a geometry.
- ST_ZMin - Returns the Z minima of a 2D or 3D bounding box or a geometry.
- ST_Zmflag - Gibt die Dimension der Koordinaten von ST_Geometry zurück.
- TG_Equals - Gibt TRUE zurück, wenn zwei TopoGeometry Objekte aus denselben topologischen Elementarstrukturen bestehen.
- TG_Intersects - Gibt TRUE zurück, wenn sich kein beliebiges Paar von Elemtarstrukturen zweier TopoGeometry Objekte überschneidet.
- UpdateGeometrySRID - Updates the SRID of all features in a geometry column, and the table metadata.
- geometry_overlaps_nd - Gibt TRUE zurück, wenn A's n-D bounding box B's n-D bounding box schneidet.
- overlaps_nd_geometry_gidx - Gibt TRUE zurück, wenn die (cached) n-D bounding box einer Geometrie eine n-D float precision bounding box (GIDX) schneidet.
- overlaps_nd_gidx_geometry - Gibt TRUE zurück, wenn eine n-D float precision bounding box (GIDX) eine (cached) n-D bounding box einer Geometrie schneidet.
- overlaps_nd_gidx_gidx - Gibt TRUE zurück, wenn sich zwei n-D float precision bounding boxes (GIDX) gegenseitig überschneiden.
- postgis_sfcgal_full_version - Returns the full version of SFCGAL in use including CGAL and Boost versions
- postgis_sfcgal_version - Returns the version of SFCGAL in use

12.9 PostGIS Curved Geometry Support Functions

The functions given below are PostGIS functions that can use CIRCULARSTRING, CURVEPOLYGON, and other curved geometry types

- AddGeometryColumn - Entfernt eine Geometriespalte aus einer räumlichen Tabelle.
- Box2D - Returns a BOX2D representing the 2D extent of a geometry.
- Box3D - Returns a BOX3D representing the 3D extent of a geometry.
- DropGeometryColumn - Entfernt eine Geometriespalte aus einer räumlichen Tabelle.
- GeometryType - Gibt den Geometrietyp des ST_Geometry Wertes zurück.
- PostGIS_AddBBox - Add bounding box to the geometry.
- PostGIS_DropBBox - Drop the bounding box cache from the geometry.
- PostGIS_HasBBox - Returns TRUE if the bbox of this geometry is cached, FALSE otherwise.
- ST_3DExtent - Aggregate function that returns the 3D bounding box of geometries.
- ST_Affine - Apply a 3D affine transformation to a geometry.
- ST_AsBinary - Return the OGC/ISO Well-Known Binary (WKB) representation of the geometry/geography without SRID meta data.
- ST_AsEWKB - Return the Extended Well-Known Binary (EWKB) representation of the geometry with SRID meta data.
- ST_AsEWKT - Gibt die Well-known-Text(WKT) Darstellung der Geometrie mit den SRID-Metadaten zurück.
- ST_AsHEXEWKB - Gibt eine Geometrie im HEXEWKB Format (als Text) aus; verwendet entweder die Little-Endian (NDR) oder die Big-Endian (XDR) Zeichenkodierung.
- ST_AsSVG - Gibt eine Geometrie als SVG-Pfad aus.
- ST_AsText - Gibt die Well-known-Text(WKT) Darstellung der Geometrie/Geographie ohne die SRID Metadaten zurück.
- ST_ClusterDBSCAN - Window function that returns a cluster id for each input geometry using the DBSCAN algorithm.
- ST_ClusterWithin - Aggregate function that clusters geometries by separation distance.
- ST_ClusterWithinWin - Window function that returns a cluster id for each input geometry, clustering using separation distance.
- ST_Collect - Creates a GeometryCollection or Multi* geometry from a set of geometries.
- ST_CoordDim - Gibt die Dimension der Koordinaten für den Wert von ST_Geometry zurück.
- ST_CurveToLine - Converts a geometry containing curves to a linear geometry.
- ST_Distance - Gibt die größte 3-dimensionale Distanz zwischen zwei geometrischen Objekten als Linie zurück
- ST_Dump - Returns a set of geometry_dump rows for the components of a geometry.
- ST_DumpPoints - Gibt eine Zusammenfassung des Inhalts einer Geometrie wieder.
- ST_EndPoint - Gibt die Anzahl der Stützpunkte eines ST_LineString oder eines ST_CircularString zurück.
- ST_EstimatedExtent - Returns the estimated extent of a spatial table.
- ST_FlipCoordinates - Returns a version of a geometry with X and Y axis flipped.
- ST_Force2D - Die Geometrien in einen "2-dimensionalen Modus" zwingen.
- ST_ForceCurve - Wandelt einen geometrischen in einen Kurven Datentyp um, soweit anwendbar.

- ST_ForceSFS - Erzwingt, dass Geometrien nur vom Typ SFS 1.1 sind.
- ST_Force3D - Zwingt die Geometrien in einen XYZ Modus. Dies ist ein Alias für ST_Force3DZ.
- ST_Force3DM - Zwingt die Geometrien in einen XYM Modus.
- ST_Force3DZ - Zwingt die Geometrien in einen XYZ Modus.
- ST_Force4D - Zwingt die Geometrien in einen XYZM Modus.
- ST_ForceCollection - Wandelt eine Geometrie in eine GEOMETRYCOLLECTION um.
- ST_GeoHash - Gibt die Geometrie in der GeoHash Darstellung aus.
- ST_GeogFromWKB - Erzeugt ein geographisches Objekt aus der Well-known-Binary (WKB) oder der erweiterten Well-known-Binary (EWKB) Darstellung.
- ST_GeomFromEWKB - Gibt einen geometrischen Datentyp (ST_Geometry) aus einer Well-known-Binary (WKB) Darstellung zurück.
- ST_GeomFromEWKT - Gibt einen spezifizierten ST_Geometry-Wert von einer erweiterten Well-known-Text Darstellung (EWKT) zurück.
- ST_GeomFromText - Gibt einen spezifizierten ST_Geometry Wert aus einer Well-known-Text Darstellung (WKT) zurück.
- ST_GeomFromWKB - Erzeugt ein geometrisches Objekt aus der Well-known-Binary (WKB) Darstellung und einer optionalen SRID.
- ST_GeometryN - Gibt den Geometrietyp des ST_Geometry Wertes zurück.
- = - Gibt TRUE zurück, wenn die Koordinaten und die Reihenfolge der Koordinaten der Geometrie/Geographie A und der Geometrie/Geographie B ident sind.
- &<| - Gibt TRUE zurück, wenn die bounding box von A jene von B überlagert oder unterhalb liegt.
- ST_HasArc - Tests if a geometry contains a circular arc
- ST_Intersects - Tests if two geometries intersect (they have at least one point in common)
- ST_IsClosed - Gibt den Wert TRUE zurück, wenn die Anfangs- und Endpunkte des LINESTRING's zusammenfallen. Bei polyedrischen Oberflächen, wenn sie geschlossen (volumetrisch) sind.
- ST_IsCollection - Gibt den Wert TRUE zurück, falls es sich bei der Geometrie um eine leere GeometryCollection, Polygon, Point etc. handelt.
- ST_IsEmpty - Tests if a geometry is empty.
- ST_LineToCurve - Converts a linear geometry to a curved geometry.
- ST_MemSize - Gibt den Geometrietyp des ST_Geometry Wertes zurück.
- ST_NPoints - Gibt die Anzahl der Punkte (Knoten) einer Geometrie zurück.
- ST_NRings - Gibt die Anzahl der inneren Ringe einer Polygongeometrie aus.
- ST_PointFromWKB - Erzeugt eine Geometrie mit gegebener SRID von WKB.
- ST_PointN - Gibt die Anzahl der Stützpunkte eines ST_LineString oder eines ST_CircularString zurück.
- ST_Points - Gibt einen MultiPoint zurück, welcher alle Koordinaten einer Geometrie enthält.
- ST_Rotate - Rotates a geometry about an origin point.
- ST_RotateZ - Rotates a geometry about the Z axis.
- ST_SRID - Returns the spatial reference identifier for a geometry.

- ST_Scale - Scales a geometry by given factors.
- ST_SetSRID - Set the SRID on a geometry.
- ST_StartPoint - Returns the first point of a LineString.
- ST_Summary - Gibt eine Zusammenfassung des Inhalts einer Geometrie wieder.
- ST_SwapOrdinates - Gibt eine Version der Ausgangsgeometrie zurück, in der die angegebenen Ordinatenwerte ausgetauscht werden.
- ST_TransScale - Translates and scales a geometry by given offsets and factors.
- ST_Transform - Return a new geometry with coordinates transformed to a different spatial reference system.
- ST_Translate - Translates a geometry by given offsets.
- ST_XMax - Returns the X maxima of a 2D or 3D bounding box or a geometry.
- ST_XMin - Returns the X minima of a 2D or 3D bounding box or a geometry.
- ST_YMax - Returns the Y maxima of a 2D or 3D bounding box or a geometry.
- ST_YMin - Returns the Y minima of a 2D or 3D bounding box or a geometry.
- ST_ZMax - Returns the Z maxima of a 2D or 3D bounding box or a geometry.
- ST_ZMin - Returns the Z minima of a 2D or 3D bounding box or a geometry.
- ST_Zmflag - Gibt die Dimension der Koordinaten von ST_Geometry zurück.
- UpdateGeometrySRID - Updates the SRID of all features in a geometry column, and the table metadata.
- ~(box2df,box2df) - Gibt TRUE zurück, wenn eine 2D float precision bounding box (BOX2DF) eine andere 2D float precision bounding box (BOX2DF) enthält.
- ~(box2df,geometry) - Gibt TRUE zurück, wenn eine 2D float precision bounding box (BOX2DF) die 2D Bounding Box einer Geometrie enthält.
- ~(geometry,box2df) - Gibt TRUE zurück, wenn die 2D bounding box einer Geometrie eine 2D float precision bounding box (GIDX) enthält.
- && - Gibt TRUE zurück, wenn die 2D Bounding Box von A die 2D Bounding Box von B schneidet.
- &&& - Gibt TRUE zurück, wenn A's n-D bounding box B's n-D bounding box schneidet.
- @(box2df,box2df) - Gibt TRUE zurück, wenn eine 2D float precision bounding box (BOX2DF) innerhalb einer anderen 2D float precision bounding box enthalten ist.
- @(box2df,geometry) - Gibt TRUE zurück, wenn eine 2D float precision bounding box (BOX2DF) in der 2D Bounding Box einer Geometrie enthalten ist..
- @(geometry,box2df) - Gibt TRUE zurück, wenn die 2D Bounding Box einer Geometrie in einer 2D float precision Bbounding Box (BOX2DF) enthalten ist.
- &&(box2df,box2df) - Gibt TRUE zurück, wenn sich zwei 2D float precision Bounding Boxes (BOX2DF) überschneiden.
- &&(box2df,geometry) - Gibt TRUE zurück, wenn eine 2D float precision bounding box (BOX2DF) eine Geometrie (cached) 2D bounding box schneidet.
- &&(geometry,box2df) - Gibt TRUE zurück, wenn sich die 2D Bounding Box (cached) einer Geometrie mit einer 2D Bounding Box mit Gleitpunktgenauigkeit (BOX2DF) überschneidet.
- &&&(geometry,gidx) - Gibt TRUE zurück, wenn die (cached) n-D bounding box einer Geometrie eine n-D float precision bounding box (GIDX) schneidet.
- &&&(gidx,geometry) - Gibt TRUE zurück, wenn eine n-D float precision bounding box (GIDX) eine (cached) n-D bounding box einer Geometrie schneidet.
- &&&(gidx,gidx) - Gibt TRUE zurück, wenn sich zwei n-D float precision bounding boxes (GIDX) gegenseitig überschneiden.

12.10 PostGIS Polyhedral Surface Support Functions

The functions given below are PostGIS functions that can use POLYHEDRALSURFACE, POLYHEDRALSURFACEM geometries

- AddGeometryColumn - Entfernt eine Geometriespalte aus einer räumlichen Tabelle.
- Box2D - Returns a BOX2D representing the 2D extent of a geometry.
- Box3D - Returns a BOX3D representing the 3D extent of a geometry.
- DropGeometryColumn - Entfernt eine Geometriespalte aus einer räumlichen Tabelle.
- GeometryType - Gibt den Geometrietyp des ST_Geometry Wertes zurück.
- PostGIS_AddBBox - Add bounding box to the geometry.
- PostGIS_DropBBox - Drop the bounding box cache from the geometry.
- PostGIS_HasBBox - Returns TRUE if the bbox of this geometry is cached, FALSE otherwise.
- ST_3DExtent - Aggregate function that returns the 3D bounding box of geometries.
- ST_Affine - Apply a 3D affine transformation to a geometry.
- ST_AsBinary - Return the OGC/ISO Well-Known Binary (WKB) representation of the geometry/geography without SRID meta data.
- ST_AsEWKB - Return the Extended Well-Known Binary (EWKB) representation of the geometry with SRID meta data.
- ST_AsEWKT - Gibt die Well-known-Text(WKT) Darstellung der Geometrie mit den SRID-Metadaten zurück.
- ST_AsHEXEWKB - Gibt eine Geometrie im HEXEWKB Format (als Text) aus; verwendet entweder die Little-Endian (NDR) oder die Big-Endian (XDR) Zeichenkodierung.
- ST_AsSVG - Gibt eine Geometrie als SVG-Pfad aus.
- ST_AsText - Gibt die Well-known-Text(WKT) Darstellung der Geometrie/Geographie ohne die SRID Metadaten zurück.
- ST_ClusterDBSCAN - Window function that returns a cluster id for each input geometry using the DBSCAN algorithm.
- ST_ClusterWithin - Aggregate function that clusters geometries by separation distance.
- ST_ClusterWithinWin - Window function that returns a cluster id for each input geometry, clustering using separation distance.
- ST_Collect - Creates a GeometryCollection or Multi* geometry from a set of geometries.
- ST_CoordDim - Gibt die Dimension der Koordinaten für den Wert von ST_Geometry zurück.
- ST_CurveToLine - Converts a geometry containing curves to a linear geometry.
- ST_Distance - Gibt die größte 3-dimensionale Distanz zwischen zwei geometrischen Objekten als Linie zurück
- ST_Dump - Returns a set of geometry_dump rows for the components of a geometry.
- ST_DumpPoints - Gibt eine Zusammenfassung des Inhalts einer Geometrie wieder.
- ST_EndPoint - Gibt die Anzahl der Stützpunkte eines ST_LineString oder eines ST_CircularString zurück.
- ST_EstimatedExtent - Returns the estimated extent of a spatial table.
- ST_FlipCoordinates - Returns a version of a geometry with X and Y axis flipped.
- ST_Force2D - Die Geometrien in einen "2-dimensionalen Modus" zwingen.
- ST_ForceCurve - Wandelt einen geometrischen in einen Kurven Datentyp um, soweit anwendbar.

- ST_ForceSFS - Erzwingt, dass Geometrien nur vom Typ SFS 1.1 sind.
- ST_Force3D - Zwingt die Geometrien in einen XYZ Modus. Dies ist ein Alias für ST_Force3DZ.
- ST_Force3DM - Zwingt die Geometrien in einen XYM Modus.
- ST_Force3DZ - Zwingt die Geometrien in einen XYZ Modus.
- ST_Force4D - Zwingt die Geometrien in einen XYZM Modus.
- ST_ForceCollection - Wandelt eine Geometrie in eine GEOMETRYCOLLECTION um.
- ST_GeoHash - Gibt die Geometrie in der GeoHash Darstellung aus.
- ST_GeogFromWKB - Erzeugt ein geographisches Objekt aus der Well-known-Binary (WKB) oder der erweiterten Well-known-Binary (EWKB) Darstellung.
- ST_GeomFromEWKB - Gibt einen geometrischen Datentyp (ST_Geometry) aus einer Well-known-Binary (WKB) Darstellung zurück.
- ST_GeomFromEWKT - Gibt einen spezifizierten ST_Geometry-Wert von einer erweiterten Well-known-Text Darstellung (EWKT) zurück.
- ST_GeomFromText - Gibt einen spezifizierten ST_Geometry Wert aus einer Well-known-Text Darstellung (WKT) zurück.
- ST_GeomFromWKB - Erzeugt ein geometrisches Objekt aus der Well-known-Binary (WKB) Darstellung und einer optionalen SRID.
- ST_GeometryN - Gibt den Geometrietyp des ST_Geometry Wertes zurück.
- = - Gibt TRUE zurück, wenn die Koordinaten und die Reihenfolge der Koordinaten der Geometrie/Geographie A und der Geometrie/Geographie B ident sind.
- &<| - Gibt TRUE zurück, wenn die bounding box von A jene von B überlagert oder unterhalb liegt.
- ST_HasArc - Tests if a geometry contains a circular arc
- ST_Intersects - Tests if two geometries intersect (they have at least one point in common)
- ST_IsClosed - Gibt den Wert TRUE zurück, wenn die Anfangs- und Endpunkte des LINESTRING's zusammenfallen. Bei polyedrischen Oberflächen, wenn sie geschlossen (volumetrisch) sind.
- ST_IsCollection - Gibt den Wert TRUE zurück, falls es sich bei der Geometrie um eine leere GeometryCollection, Polygon, Point etc. handelt.
- ST_IsEmpty - Tests if a geometry is empty.
- ST_LineToCurve - Converts a linear geometry to a curved geometry.
- ST_MemSize - Gibt den Geometrietyp des ST_Geometry Wertes zurück.
- ST_NPoints - Gibt die Anzahl der Punkte (Knoten) einer Geometrie zurück.
- ST_NRings - Gibt die Anzahl der inneren Ringe einer Polygongeometrie aus.
- ST_PointFromWKB - Erzeugt eine Geometrie mit gegebener SRID von WKB.
- ST_PointN - Gibt die Anzahl der Stützpunkte eines ST_LineString oder eines ST_CircularString zurück.
- ST_Points - Gibt einen MultiPoint zurück, welcher alle Koordinaten einer Geometrie enthält.
- ST_Rotate - Rotates a geometry about an origin point.
- ST_RotateZ - Rotates a geometry about the Z axis.
- ST_SRID - Returns the spatial reference identifier for a geometry.

- ST_Scale - Scales a geometry by given factors.
- ST_SetSRID - Set the SRID on a geometry.
- ST_StartPoint - Returns the first point of a LineString.
- ST_Summary - Gibt eine Zusammenfassung des Inhalts einer Geometrie wieder.
- ST_SwapOrdinates - Gibt eine Version der Ausgangsgeometrie zurück, in der die angegebenen Ordinatenwerte ausgetauscht werden.
- ST_TransScale - Translates and scales a geometry by given offsets and factors.
- ST_Transform - Return a new geometry with coordinates transformed to a different spatial reference system.
- ST_Translate - Translates a geometry by given offsets.
- ST_XMax - Returns the X maxima of a 2D or 3D bounding box or a geometry.
- ST_XMin - Returns the X minima of a 2D or 3D bounding box or a geometry.
- ST_YMax - Returns the Y maxima of a 2D or 3D bounding box or a geometry.
- ST_YMin - Returns the Y minima of a 2D or 3D bounding box or a geometry.
- ST_ZMax - Returns the Z maxima of a 2D or 3D bounding box or a geometry.
- ST_ZMin - Returns the Z minima of a 2D or 3D bounding box or a geometry.
- ST_Zmflag - Gibt die Dimension der Koordinaten von ST_Geometry zurück.
- UpdateGeometrySRID - Updates the SRID of all features in a geometry column, and the table metadata.
- ~(box2df,box2df) - Gibt TRUE zurück, wenn eine 2D float precision bounding box (BOX2DF) eine andere 2D float precision bounding box (BOX2DF) enthält.
- ~(box2df,geometry) - Gibt TRUE zurück, wenn eine 2D float precision bounding box (BOX2DF) die 2D Bounding Box einer Geometrie enthält.
- ~(geometry,box2df) - Gibt TRUE zurück, wenn die 2D bounding box einer Geometrie eine 2D float precision bounding box (GIDX) enthält.
- && - Gibt TRUE zurück, wenn die 2D Bounding Box von A die 2D Bounding Box von B schneidet.
- &&& - Gibt TRUE zurück, wenn A's n-D bounding box B's n-D bounding box schneidet.
- @(box2df,box2df) - Gibt TRUE zurück, wenn eine 2D float precision bounding box (BOX2DF) innerhalb einer anderen 2D float precision bounding box enthalten ist.
- @(box2df,geometry) - Gibt TRUE zurück, wenn eine 2D float precision bounding box (BOX2DF) in der 2D Bounding Box einer Geometrie enthalten ist..
- @(geometry,box2df) - Gibt TRUE zurück, wenn die 2D Bounding Box einer Geometrie in einer 2D float precision Bbounding Box (BOX2DF) enthalten ist.
- &&(box2df,box2df) - Gibt TRUE zurück, wenn sich zwei 2D float precision Bounding Boxes (BOX2DF) überschneiden.
- &&(box2df,geometry) - Gibt TRUE zurück, wenn eine 2D float precision bounding box (BOX2DF) eine Geometrie (cached) 2D bounding box schneidet.
- &&(geometry,box2df) - Gibt TRUE zurück, wenn sich die 2D Bounding Box (cached) einer Geometrie mit einer 2D Bounding Box mit Gleitpunktgenauigkeit (BOX2DF) überschneidet.
- &&&(geometry,gidx) - Gibt TRUE zurück, wenn die (cached) n-D bounding box einer Geometrie eine n-D float precision bounding box (GIDX) schneidet.
- &&&(gidx,geometry) - Gibt TRUE zurück, wenn eine n-D float precision bounding box (GIDX) eine (cached) n-D bounding box einer Geometrie schneidet.
- &&&(gidx,gidx) - Gibt TRUE zurück, wenn sich zwei n-D float precision bounding boxes (GIDX) gegenseitig überschneiden.

12.11 PostGIS Function Support Matrix

Below is an alphabetical listing of spatial specific functions in PostGIS and the kinds of spatial types they work with or OGC/SQL compliance they try to conform to.

- A ✓ means the function works with the type or subtype natively.
- A means it works but with a transform cast built-in using cast to geometry, transform to a "best srid" spatial ref and then cast back. Results may not be as expected for large areas or areas at poles and may accumulate floating point junk.
- A means the function works with the type because of a auto-cast to another such as to box3d rather than direct type support.
- A means the function only available if PostGIS compiled with SFCGAL support.
- A means the function support is provided by SFCGAL if PostGIS compiled with SFCGAL support, otherwise GEOS/built-in support.
- geom - Basic 2D geometry support (x,y).
- geog - Basic 2D geography support (x,y).
- 2.5D - basic 2D geometries in 3 D/4D space (has Z or M coord).
- PS - Polyhedral surfaces
- T - Triangles and Triangulated Irregular Network surfaces (TIN)

Function	geom	geog	2.5D	Curves	SQL MM	PS	T
Box2D	✓			✓		✓	✓
Box3D	✓		✓	✓		✓	✓
GeometryType	✓		✓	✓		✓	✓
PostGIS_AddBBox	✓			✓			
PostGIS_DropBBox	✓			✓			
PostGIS_HasBBox	✓			✓			
ST_3DArea							
ST_3DClosestPoint	✓		✓			✓	
ST_3DConvexHull							
ST_3DDFullyWithi	✓		✓			✓	
ST_3DDWithin	✓		✓		✓	✓	
ST_3DDifference							
ST_3DDistance	✓		✓		✓	✓	
ST_3DExtent	✓		✓	✓		✓	✓

Function	geom	geog	2.5D	Curves	SQL MM	PS	T
ST_3DIntersection							
ST_3DIntersects	✓		✓		✓	✓	✓
ST_3DLength	✓		✓		✓		
ST_3DLineInterpol oint	✓		✓				
ST_3DLongestLine	✓		✓			✓	
ST_3DMakeBox	✓						
ST_3DMaxDistance	✓		✓			✓	
ST_3DPerimeter	✓		✓		✓		
ST_3DShortestLine	✓		✓			✓	
ST_3DUnion							
ST_AddMeasure	✓		✓				
ST_AddPoint	✓		✓				
ST_Affine	✓		✓	✓		✓	✓
ST_AlphaShape							
ST_Angle	✓						
ST_ApproximateM Axis							
ST_Area	✓	✓			✓	✓	
ST_AsBinary	✓	✓	✓	✓	✓	✓	✓
ST_AsEWKB	✓		✓	✓		✓	✓
ST_AsEWKT	✓	✓	✓	✓		✓	✓
ST_AsEncodedPoly	✓						
ST_AsFlatGeobuf							
ST_AsGML	✓	✓	✓		✓	✓	✓
ST_AsGeoJSON	✓	✓	✓				
ST_AsGeobuf							
ST_AsHEXEWKB	✓		✓	✓			
ST_AsKML	✓	✓	✓				
ST_AsLatLonText	✓						
ST_AsMARC21	✓						
ST_AsMVT							
ST_AsMVTGeom	✓						
ST_AsSVG	✓	✓		✓			

Function	geom	geog	2.5D	Curves	SQL MM	PS	T
ST_AsTWKB	✓						
ST_AsText	✓	✓		✓	✓		
ST_AsX3D	✓		✓			✓	✓
ST_Azimuth	✓	✓					
ST_BdMPolyFromT	✓						
ST_BdPolyFromTex	✓						
ST_Boundary	✓		✓		✓		
ST_BoundingDiago	✓		✓				
ST_Box2dFromGeo h							
ST_Buffer	✓				✓		
ST_BuildArea	✓						
ST_CPAWithin	✓		✓				
ST_Centroid	✓	✓			✓		
ST_ChaikinSmooth	✓		✓				
ST_ClipByBox2D	✓						
ST_ClosestPoint	✓	✓					
ST_ClosestPointOf / oach	✓		✓				
ST_ClusterDBSCA	✓			✓			
ST_ClusterIntersect	✓						
ST_ClusterIntersect Vin	✓						
ST_ClusterKMeans	✓						
ST_ClusterWithin	✓			✓			
ST_ClusterWithinW	✓			✓			
ST_Collect	✓		✓	✓			
ST_CollectionExtra	✓						
ST_CollectionHom ize	✓						
ST_ConcaveHull	✓						
ST_ConstrainedDe yTriangles							
ST_Contains	✓				✓		
ST_ContainsProper	✓						
ST_ConvexHull	✓		✓		✓		
ST_CoordDim	✓		✓	✓	✓	✓	✓
ST_CoverageInvalic es	✓						

Function	geom	geog	2.5D	Curves	SQL MM	PS	T
ST_CoverageSimpli	✓						
ST_CoverageUnion	✓						
ST_CoveredBy	✓	✓					
ST_Covers	✓	✓					
ST_Crosses	✓				✓		
ST_CurveToLine	✓		✓	✓	✓		
ST_DFullyWithin	✓						
ST_DWithin	✓	✓					
ST_DelaunayTriang	✓		✓				✓
ST_Difference	✓		✓		✓		
ST_Dimension	✓				✓	✓	✓
ST_Disjoint	✓				✓		
ST_Distance	✓	✓		✓	✓		
ST_DistanceCPA	✓		✓				
ST_DistanceSphere	✓						
ST_DistanceSphero	✓						
ST_Dump	✓		✓	✓		✓	✓
ST_DumpPoints	✓		✓	✓		✓	✓
ST_DumpRings	✓		✓				
ST_DumpSegments	✓		✓				✓
ST_EndPoint	✓		✓	✓	✓		
ST_Envelope	✓				✓		
ST_Equals	✓				✓		
ST_EstimatedExten	✓			✓			
ST_Expand	✓					✓	✓
ST_Extent	✓					✓	✓
ST_ExteriorRing	✓		✓		✓		
ST_Extrude							
ST_FilterByM	✓						
ST_FlipCoordinates	✓		✓	✓		✓	✓
ST_Force2D	✓		✓	✓		✓	
ST_ForceCurve	✓		✓	✓			
ST_ForceLHR							

Function	geom	geog	2.5D	Curves	SQL MM	PS	T
ST_ForcePolygonC	✓		✓				
ST_ForcePolygonC	✓		✓				
ST_ForceRHR	✓		✓			✓	
ST_ForceSFS	✓		✓	✓		✓	✓
ST_Force3D	✓		✓	✓		✓	
ST_Force3DM	✓			✓			
ST_Force3DZ	✓		✓	✓		✓	
ST_Force4D	✓		✓	✓			
ST_ForceCollection	✓		✓	✓		✓	
ST_FrechetDistance	✓						
ST_FromFlatGeobuf							
ST_FromFlatGeobufToTable							
ST_GMLToSQL	✓				✓		
ST_GeneratePoints	✓						
ST_GeoHash	✓			✓			
ST_GeogFromText		✓					
ST_GeogFromWKB		✓		✓			
ST_GeographyFromText		✓					
ST_GeomCollFrom	✓				✓		
ST_GeomFromEWI	✓		✓	✓		✓	✓
ST_GeomFromEWI	✓		✓	✓		✓	✓
ST_GeomFromGM	✓		✓			✓	✓
ST_GeomFromGeo	✓						
ST_GeomFromGeo	✓		✓				
ST_GeomFromKM	✓		✓				
ST_GeomFromMA	✓						
ST_GeomFromTWI	✓						
ST_GeomFromText	✓			✓	✓		
ST_GeomFromWK	✓			✓	✓		
ST_GeometricMedi	✓		✓				
ST_GeometryFrom	✓				✓		
ST_GeometryN	✓		✓	✓	✓	✓	✓
ST_GeometryType	✓		✓		✓	✓	
|>>	✓						
<<|	✓						

Function	geom	geog	2.5D	Curves	SQL MM	PS	T
~	✓						
@	✓						
=	✓	✓		✓		✓	
<<	✓						
|&>	✓						
&<|	✓			✓		✓	
&<	✓						
&>	✓						
>>	✓						
~=	✓					✓	
ST_HasArc	✓		✓	✓			
ST_HausdorffDistan	✓						
ST_Hexagon	✓						
ST_HexagonGrid	✓						
ST_InteriorRingN	✓		✓		✓		
ST_InterpolatePoint	✓		✓				
ST_Intersection	✓		✓		✓		
ST_Intersects	✓	✓		✓	✓		✓
ST_InverseTransfor	✓ peline						
ST_IsClosed	✓		✓	✓	✓	✓	
ST_IsCollection	✓		✓	✓			
ST_IsEmpty	✓			✓	✓		
ST_IsPlanar							
ST_IsPolygonCCW	✓		✓				
ST_IsPolygonCW	✓		✓				
ST_IsRing	✓				✓		
ST_IsSimple	✓		✓		✓		
ST_IsSolid							
ST_IsValid	✓				✓		
ST_IsValidDetail	✓						
ST_IsValidReason	✓						
ST_IsValidTrajecto	✓		✓				
ST_LargestEmptyC	✓						

Function	geom	geog	2.5D	Curves	SQL MM	PS	T
ST_Length	✓	✓			✓		
ST_Length2D	✓						
ST_LengthSpheroid	✓		✓				
ST_Letters	✓						
ST_LineCrossingDi	✓ ion						
ST_LineExtend	✓						
ST_LineFromEncoc	✓ olyline						
ST_LineFromMulti	✓ t		✓				
ST_LineFromText	✓				✓		
ST_LineFromWKB	✓				✓		
ST_LineInterpolatel	✓ t	✓	✓				
ST_LineInterpolatel	✓ ts	✓	✓				
ST_LineLocatePoin	✓	✓					
ST_LineMerge	✓						
ST_LineSubstring	✓	✓	✓				
ST_LineToCurve	✓		✓	✓			
ST_LinestringFrom	✓ B				✓		
ST_LocateAlong	✓				✓		
ST_LocateBetween	✓				✓		
ST_LocateBetweenI	✓ ations		✓				
ST_LongestLine	✓						
ST_M	✓		✓		✓		
ST_MLineFromTex	✓				✓		
ST_MPointFromTe	✓				✓		
ST_MPolyFromTex	✓				✓		
ST_MakeBox2D	✓						
ST_MakeEnvelope	✓						
ST_MakeLine	✓		✓				
ST_MakePoint	✓		✓				
ST_MakePointM	✓						
ST_MakePolygon	✓		✓				
ST_MakeSolid							
ST_MakeValid	✓		✓				

Function	geom	geog	2.5D	Curves	SQL MM	PS	T
ST_MaxDistance	✓						
ST_MaximumInscri	✓ Circle						
ST_MemSize	✓		✓	✓		✓	✓
ST_MemUnion	✓		✓				
ST_MinimumBoun	✓ Circle						
ST_MinimumBoun	✓ Radius						
ST_MinimumClear	✓						
ST_MinimumClear	✓ Line						
ST_MinkowskiSun							
ST_Multi	✓						
ST_NDims	✓		✓				
ST_NPoints	✓		✓	✓		✓	
ST_NRings	✓		✓	✓			
ST_Node	✓		✓				
ST_Normalize	✓						
ST_NumGeometrie	✓		✓		✓	✓	✓
ST_NumInteriorRin	✓						
ST_NumInteriorRin	✓				✓		
ST_NumPatches	✓		✓		✓	✓	
ST_NumPoints	✓				✓		
ST_OffsetCurve	✓						
ST_OptimalAlphaS							
ST_OrderingEquals	✓				✓		
ST_Orientation							
ST_OrientedEnvelo	✓						
ST_Overlaps	✓				✓		
ST_PatchN	✓		✓		✓	✓	
ST_Perimeter	✓	✓			✓		
ST_Perimeter2D	✓						
ST_Point	✓				✓		
ST_PointFromGeoHash							
ST_PointFromText	✓				✓		
ST_PointFromWKB	✓		✓	✓	✓		

Function	geom	geog	2.5D	Curves	SQL MM	PS	T
ST_PointInsideCirc	✓						
ST_PointM	✓						
ST_PointN	✓		✓	✓	✓		
ST_PointOnSurface	✓		✓		✓		
ST_PointZ	✓						
ST_PointZM	✓						
ST_Points	✓		✓	✓			
ST_Polygon	✓		✓		✓		
ST_PolygonFromTe	✓				✓		
ST_Polygonize	✓						
ST_Project	✓	✓					
ST_QuantizeCoordi	✓ s						
ST_ReducePrecisio	✓						
ST_Relate	✓				✓		
ST_RelateMatch							
ST_RemovePoint	✓		✓				
ST_RemoveRepeate	✓ ints		✓			✓	
ST_Reverse	✓		✓			✓	
ST_Rotate	✓		✓	✓		✓	✓
ST_RotateX	✓		✓			✓	✓
ST_RotateY	✓		✓			✓	✓
ST_RotateZ	✓		✓	✓		✓	✓
ST_SRID	✓			✓	✓		
ST_Scale	✓		✓	✓		✓	✓
ST_Scroll	✓		✓				
ST_Segmentize	✓	✓					
ST_SetEffectiveAre	✓						
ST_SetPoint	✓		✓				
ST_SetSRID	✓			✓			
ST_SharedPaths	✓						
ST_ShiftLongitude	✓		✓			✓	✓
ST_ShortestLine	✓	✓					
ST_Simplify	✓						
ST_SimplifyPolygo	✓ ll						

Function	geom	geog	2.5D	Curves	SQL MM	PS	T
ST_SimplifyPreserv	✓ pology						
ST_SimplifyVW	✓						
ST_Snap	✓						
ST_SnapToGrid	✓		✓				
ST_Split	✓						
ST_Square	✓						
ST_SquareGrid	✓						
ST_StartPoint	✓		✓	✓	✓		
ST_StraightSkeleto							
ST_Subdivide	✓						
ST_Summary	✓	✓		✓		✓	✓
ST_SwapOrdinates	✓		✓	✓		✓	✓
ST_SymDifference	✓		✓		✓		
ST_Tesselate							
ST_TileEnvelope	✓						
ST_Touches	✓				✓		
ST_TransScale	✓		✓	✓			
ST_Transform	✓			✓	✓	✓	
ST_TransformPipel	✓						
ST_Translate	✓		✓	✓			
ST_TriangulatePoly	✓						
ST_UnaryUnion	✓		✓				
ST_Union	✓		✓		✓		
ST_Volume							
ST_VoronoiLines	✓						
ST_VoronoiPolygor	✓						
ST_WKBToSQL	✓				✓		
ST_WKTToSQL	✓				✓		
ST_Within	✓				✓		
ST_WrapX	✓		✓				
ST_X	✓		✓		✓		
ST_XMax			✓	✓			
ST_XMin			✓	✓			

Function	geom	geog	2.5D	Curves	SQL MM	PS	T
ST_Y	✓		✓		✓		
ST_YMax	✓		✓	✓			
ST_YMin	✓		✓	✓			
ST_Z	✓		✓		✓		
ST_ZMax	✓		✓	✓			
ST_ZMin	✓		✓	✓			
ST_Zmflag	✓		✓	✓			
~(box2df,box2df)	✓			✓		✓	
~(box2df,geometry)	✓			✓		✓	
~(geometry,box2df)	✓			✓		✓	
<#>	✓						
<<#>>	✓						
<<->>	✓						
|=|	✓						
<->	✓	✓					
&&	✓	✓		✓		✓	
&&&	✓		✓	✓		✓	✓
@(box2df,box2df)	✓			✓		✓	
@(box2df,geometry	✓			✓		✓	
@(geometry,box2df	✓			✓		✓	
&&(box2df,box2df)	✓			✓		✓	
&&(box2df,geomet	✓			✓		✓	
&&(geometry,box2	✓			✓		✓	
&&&(geometry,gid	✓		✓	✓		✓	✓
&&&(gidx,geometr	✓		✓	✓		✓	✓
&&&(gidx,gidx)			✓	✓		✓	✓
postgis.backend							
postgis.enable_outdb_rasters							
postgis.gdal_datapath							
postgis.gdal_enabled_drivers							
postgis.gdal_config_options							
postgis_sfcgal_full_version							
postgis_sfcgal_version							
postgis_srs							
postgis_srs_all							
postgis_srs_codes							

Function	geom	geog	2.5D	Curves	SQL MM	PS	T
postgis_srs_search	✓						

12.12 New, Enhanced or changed PostGIS Functions

12.12.1 PostGIS Functions new or enhanced in 3.4

The functions given below are PostGIS functions that were added or enhanced.

Functions new in PostGIS 3.4

- PostGIS_GEOS_Compiled_Version - Availability: 3.4.0 Returns the version number of the GEOS library against which PostGIS was built.
- ST_ClusterIntersectingWin - Availability: 3.4.0 Window function that returns a cluster id for each input geometry, clustering input geometries into connected sets.
- ST_ClusterWithinWin - Availability: 3.4.0 Window function that returns a cluster id for each input geometry, clustering using separation distance.
- ST_CoverageInvalidEdges - Availability: 3.4.0 Window function that finds locations where polygons fail to form a valid coverage.
- ST_CoverageSimplify - Availability: 3.4.0 Window function that simplifies the edges of a polygonal coverage.
- ST_CoverageUnion - Availability: 3.4.0 - requires GEOS >= 3.8.0 Computes the union of a set of polygons forming a coverage by removing shared edges.
- ST_InverseTransformPipeline - Availability: 3.4.0 Return a new geometry with coordinates transformed to a different spatial reference system using the inverse of a defined coordinate transformation pipeline.
- ST_LargestEmptyCircle - Availability: 3.4.0. Computes the largest circle not overlapping a geometry.
- ST_LineExtend - Availability: 3.4.0 Returns a line with the last and first segments extended the specified distance(s).
- ST_TransformPipeline - Availability: 3.4.0 Return a new geometry with coordinates transformed to a different spatial reference system using a defined coordinate transformation pipeline.
- postgis_srs - Availability: 3.4.0 Return a metadata record for the requested authority and srid.
- postgis_srs_all - Availability: 3.4.0 Return metadata records for every spatial reference system in the underlying Proj database.
- postgis_srs_codes - Availability: 3.4.0 Return the list of SRS codes associated with the given authority.
- postgis_srs_search - Availability: 3.4.0 Return metadata records for projected coordinate systems that have areas of useage that fully contain the bounds parameter.

Functions enhanced in PostGIS 3.4

- PostGIS_Full_Version - Enhanced: 3.4.0 now includes extra PROJ configurations NETWORK_ENABLED, URL_ENDPOINT and DATABASE_PATH of proj.db location Reports full PostGIS version and build configuration infos.
- PostGIS_PROJ_Version - Enhanced: 3.4.0 now includes NETWORK_ENABLED, URL_ENDPOINT and DATABASE_PATH of proj.db location Returns the version number of the PROJ4 library.
- ST_AsSVG - Enhanced: 3.4.0 to support all curve types Gibt eine Geometrie als SVG-Pfad aus.
- ST_ClosestPoint - Enhanced: 3.4.0 - Support for geography. Returns the 2D point on g1 that is closest to g2. This is the first point of the shortest line from one geometry to the other.

- ST_LineSubstring - Enhanced: 3.4.0 - Support for geography was introduced. Returns the part of a line between two fractional locations.
- ST_Project - Enhanced: 3.4.0 Allow geometry arguments and two-point form omitting azimuth. Returns a point projected from a start point by a distance and bearing (azimuth).
- ST_ShortestLine - Enhanced: 3.4.0 - support for geography. Gibt die 2-dimenionale kürzeste Strecke zwischen zwei Geometrien als Linie zurück

Functions changed in PostGIS 3.4

- PostGIS_Extensions_Upgrade - Changed: 3.4.0 to add target_version argument. Packages and upgrades PostGIS extensions (e.g. postgis_raster, postgis_topology, postgis_sfcgal) to given or latest version.

12.12.2 PostGIS Functions new or enhanced in 3.3

The functions given below are PostGIS functions that were added or enhanced.

Functions new in PostGIS 3.3

- ST_3DConvexHull - Verfügbarkeit: 2.3.0 Berechnet die konvexe Hülle einer Geometrie.
- ST_3DUnion - Availability: 3.3.0 aggregate variant was added Perform 3D union.
- ST_AlphaShape - Availability: 3.3.0 - requires SFCGAL >= 1.4.1. Computes an Alpha-shape enclosing a geometry
- ST_AsMARC21 - Availability: 3.3.0 Returns geometry as a MARC21/XML record with a geographic datafield (034).
- ST_GeomFromMARC21 - Availability: 3.3.0, requires libxml2 2.6+ Takes MARC21/XML geographic data as input and returns a PostGIS geometry object.
- ST_Letters - Verfügbarkeit: 2.1.0 Returns the input letters rendered as geometry with a default start position at the origin and default text height of 100.
- ST_OptimalAlphaShape - Availability: 3.3.0 - requires SFCGAL >= 1.4.1. Computes an Alpha-shape enclosing a geometry using an "optimal" alpha value.
- ST_SimplifyPolygonHull - Availability: 3.3.0. Computes a simplifed topology-preserving outer or inner hull of a polygonal geometry.
- ST_TriangulatePolygon - Availability: 3.3.0. Computes the constrained Delaunay triangulation of polygons
- postgis_sfcgal_full_version - Verfügbarkeit: 2.3.0 Returns the full version of SFCGAL in use including CGAL and Boost versions

Functions enhanced in PostGIS 3.3

- ST_ConcaveHull - Enhanced: 3.3.0, GEOS native implementation enabled for GEOS 3.11+ Computes a possibly concave geometry that contains all input geometry vertices
- ST_LineMerge - Enhanced: 3.3.0 accept a directed parameter. Return the lines formed by sewing together a MultiLineString.

Functions changed in PostGIS 3.3

- PostGIS_Extensions_Upgrade - Changed: 3.3.0 support for upgrades from any PostGIS version. Does not work on all systems. Packages and upgrades PostGIS extensions (e.g. postgis_raster, postgis_topology, postgis_sfcgal) to given or latest version.

12.12.3 PostGIS Functions new or enhanced in 3.2

The functions given below are PostGIS functions that were added or enhanced.

Functions new in PostGIS 3.2

- ST_AsFlatGeobuf - Availability: 3.2.0 Return a FlatGeobuf representation of a set of rows.
- ST_DumpSegments - Verfügbarkeit: 2.2.0 Gibt eine Zusammenfassung des Inhalts einer Geometrie wieder.
- ST_FromFlatGeobuf - Availability: 3.2.0 Reads FlatGeobuf data.
- ST_FromFlatGeobufToTable - Availability: 3.2.0 Creates a table based on the structure of FlatGeobuf data.
- ST_Scroll - Availability: 3.2.0 Change start point of a closed LineString.
- postgis.gdal_config_options - Verfügbarkeit: 2.2.0 Eine boolesche Konfigurationsmöglichkeit um den Zugriff auf out-db Rasterbänder zu ermöglichen

Functions enhanced in PostGIS 3.2

- ST_ClusterKMeans - Enhanced: 3.2.0 Support for max_radius Window function that returns a cluster id for each input geometry using the K-means algorithm.
- ST_MakeValid - Enhanced: 3.2.0, added algorithm options, 'linework' and 'structure' which requires GEOS >= 3.10.0. Attempts to make an invalid geometry valid without losing vertices.
- ST_Point - Enhanced: 3.2.0 srid as an extra optional argument was added. Older installs require combining with ST_SetSRID to mark the srid on the geometry. Creates a Point with X, Y and SRID values.
- ST_PointM - Enhanced: 3.2.0 srid as an extra optional argument was added. Older installs require combining with ST_SetSRID to mark the srid on the geometry. Creates a Point with X, Y, M and SRID values.
- ST_PointZ - Enhanced: 3.2.0 srid as an extra optional argument was added. Older installs require combining with ST_SetSRID to mark the srid on the geometry. Creates a Point with X, Y, Z and SRID values.
- ST_PointZM - Enhanced: 3.2.0 srid as an extra optional argument was added. Older installs require combining with ST_SetSRID to mark the srid on the geometry. Creates a Point with X, Y, Z, M and SRID values.
- ST_RemovePoint - Enhanced: 3.2.0 Remove a point from a linestring.
- ST_RemoveRepeatedPoints - Enhanced: 3.2.0 Returns a version of a geometry with duplicate points removed.
- ST_StartPoint - Enhanced: 3.2.0 returns a point for all geometries. Prior behavior returns NULLs if input was not a LineString. Returns the first point of a LineString.

Functions changed in PostGIS 3.2

- ST_Boundary - Changed: 3.2.0 support for TIN, does not use geos, does not linearize curves Gibt die abgeschlossene Hülle aus der kombinierten Begrenzung der Geometrie zurück.

12.12.4 PostGIS Functions new or enhanced in 3.1

The functions given below are PostGIS functions that were added or enhanced.

Functions new in PostGIS 3.1

- ST_Hexagon - Verfügbarkeit: 2.1.0 Returns a single hexagon, using the provided edge size and cell coordinate within the hexagon grid space.
- ST_HexagonGrid - Verfügbarkeit: 2.1.0 Returns a set of hexagons and cell indices that completely cover the bounds of the geometry argument.

- ST_MaximumInscribedCircle - Availability: 3.1.0. Berechnet die konvexe Hülle einer Geometrie.
- ST_ReducePrecision - Availability: 3.1.0. Returns a valid geometry with points rounded to a grid tolerance.
- ST_Square - Verfügbarkeit: 2.1.0 Returns a single square, using the provided edge size and cell coordinate within the square grid space.
- ST_SquareGrid - Verfügbarkeit: 2.1.0 Returns a set of grid squares and cell indices that completely cover the bounds of the geometry argument.

Functions enhanced in PostGIS 3.1

- ST_AsEWKT - Enhanced: 3.1.0 support for optional precision parameter. Gibt die Well-known-Text(WKT) Darstellung der Geometrie mit den SRID-Metadaten zurück.
- ST_ClusterKMeans - Enhanced: 3.1.0 Support for 3D geometries and weights Window function that returns a cluster id for each input geometry using the K-means algorithm.
- ST_Difference - Enhanced: 3.1.0 accept a gridSize parameter. Computes a geometry representing the part of geometry A that does not intersect geometry B.
- ST_Intersection - Enhanced: 3.1.0 accept a gridSize parameter Computes a geometry representing the shared portion of geometries A and B.
- ST_MakeValid - Enhanced: 3.1.0, added removal of Coordinates with NaN values. Attempts to make an invalid geometry valid without losing vertices.
- ST_Subdivide - Enhanced: 3.1.0 accept a gridSize parameter. Computes a rectilinear subdivision of a geometry.
- ST_SymDifference - Enhanced: 3.1.0 accept a gridSize parameter. Computes a geometry representing the portions of geometries A and B that do not intersect.
- ST_TileEnvelope - Erweiterung: 2.0.0 Standardwert für den optionalen Parameter SRID eingefügt. Creates a rectangular Polygon in Web Mercator (SRID:3857) using the XYZ tile system.
- ST_UnaryUnion - Enhanced: 3.1.0 accept a gridSize parameter. Computes the union of the components of a single geometry.
- ST_Union - Enhanced: 3.1.0 accept a gridSize parameter. Computes a geometry representing the point-set union of the input geometries.

Functions changed in PostGIS 3.1

- ST_Force3D - Changed: 3.1.0. Added support for supplying a non-zero Z value. Zwingt die Geometrien in einen XYZ Modus. Dies ist ein Alias für ST_Force3DZ.
- ST_Force3DM - Changed: 3.1.0. Added support for supplying a non-zero M value. Zwingt die Geometrien in einen XYM Modus.
- ST_Force3DZ - Changed: 3.1.0. Added support for supplying a non-zero Z value. Zwingt die Geometrien in einen XYZ Modus.
- ST_Force4D - Changed: 3.1.0. Added support for supplying non-zero Z and M values. Zwingt die Geometrien in einen XYZM Modus.

12.12.5 PostGIS Functions new or enhanced in 3.0

The functions given below are PostGIS functions that were added or enhanced.

Functions new in PostGIS 3.0

- ST_3DLineInterpolatePoint - Verfügbarkeit: 2.0.0 Gibt einen oder mehrere, entlang einer Linie interpolierte Punkte zurück.
- ST_ConstrainedDelaunayTriangles - Verfügbarkeit: 2.0.0 Return a constrained Delaunay triangulation around the given input geometry.
- ST_TileEnvelope - Verfügbarkeit: 2.1.0 Creates a rectangular Polygon in Web Mercator (SRID:3857) using the XYZ tile system.

Functions enhanced in PostGIS 3.0

- ST_AsMVT - Erweiterung: 3.0 - Unterstützung für eine Feature-ID. Aggregate function returning a MVT representation of a set of rows.
- ST_Contains - Enhanced: 3.0.0 enabled support for GEOMETRYCOLLECTION Tests if every point of B lies in A, and their interiors have a point in common
- ST_ContainsProperly - Enhanced: 3.0.0 enabled support for GEOMETRYCOLLECTION Tests if every point of B lies in the interior of A
- ST_CoveredBy - Enhanced: 3.0.0 enabled support for GEOMETRYCOLLECTION Tests if every point of A lies in B
- ST_Covers - Enhanced: 3.0.0 enabled support for GEOMETRYCOLLECTION Tests if every point of B lies in A
- ST_Crosses - Enhanced: 3.0.0 enabled support for GEOMETRYCOLLECTION Tests if two geometries have some, but not all, interior points in common
- ST_CurveToLine - Erweiterung: 3.0.0 führte eine minimale Anzahl an Segmenten pro linearisierten Bogen ein, um einem topologischen Kollaps vorzubeugen. Converts a geometry containing curves to a linear geometry.
- ST_Disjoint - Enhanced: 3.0.0 enabled support for GEOMETRYCOLLECTION Tests if two geometries have no points in common
- ST_Equals - Enhanced: 3.0.0 enabled support for GEOMETRYCOLLECTION Tests if two geometries include the same set of points
- ST_GeneratePoints - Erweiterung: mit 3.0.0 wurde das Argument "seed" hinzugefügt Generates random points contained in a Polygon or MultiPolygon.
- ST_GeomFromGeoJSON - Enhanced: 3.0.0 parsed geometry defaults to SRID=4326 if not specified otherwise. Nimmt als Eingabe eine GeoJSON-Darstellung der Geometrie und gibt ein geometrisches PostGIS-Objekt aus.
- ST_LocateBetween - Enhanced: 3.0.0 - added support for POLYGON, TIN, TRIANGLE. Returns the portions of a geometry that match a measure range.
- ST_LocateBetweenElevations - Enhanced: 3.0.0 - added support for POLYGON, TIN, TRIANGLE. Returns the portions of a geometry that lie in an elevation (Z) range.
- ST_Overlaps - Enhanced: 3.0.0 enabled support for GEOMETRYCOLLECTION Tests if two geometries have the same dimension and intersect, but each has at least one point not in the other
- ST_Relate - Enhanced: 3.0.0 enabled support for GEOMETRYCOLLECTION Tests if two geometries have a topological relationship matching an Intersection Matrix pattern, or computes their Intersection Matrix
- ST_Segmentize - Enhanced: 3.0.0 Segmentize geometry now produces equal-length subsegments Returns a modified geometry/geography having no segment longer than a given distance.
- ST_Touches - Enhanced: 3.0.0 enabled support for GEOMETRYCOLLECTION Tests if two geometries have at least one point in common, but their interiors do not intersect

- ST_Within - Enhanced: 3.0.0 enabled support for GEOMETRYCOLLECTION Tests if every point of A lies in B, and their interiors have a point in common

Functions changed in PostGIS 3.0

- PostGIS_Extensions_Upgrade - Changed: 3.0.0 to repackage loose extensions and support postgis_raster. Packages and upgrades PostGIS extensions (e.g. postgis_raster, postgis_topology, postgis_sfcgal) to given or latest version.
- ST_3DDistance - Changed: 3.0.0 - SFCGAL version removed Für den geometrischen Datentyp. Es wird der geringste 3-dimensionale kartesische Abstand (basierend auf dem Koordinatenreferenzsystem) zwischen zwei geometrischen Objekten in projizierten Einheiten zurückgegeben.
- ST_3DIntersects - Änderung: 3.0.0 das SFCGAL Back-end wurde entfernt, das GEOS Back-end unterstützt TIN. Tests if two geometries spatially intersect in 3D - only for points, linestrings, polygons, polyhedral surface (area)
- ST_Area - Changed: 3.0.0 - does not depend on SFCGAL anymore. Gibt den geometrischen Schwerpunkt einer Geometrie zurück.
- ST_AsGeoJSON - Änderung: 3.0.0 Unterstützung von Datensätzen bei der Eingabe Return a geometry as a GeoJSON element.
- ST_AsGeoJSON - Änderung: 3.0.0 Ausgabe der SRID wenn nicht EPSG:4326 Return a geometry as a GeoJSON element.
- ST_AsKML - Changed: 3.0.0 - Removed the "versioned" variant signature Gibt die Geometrie als GML-Element - Version 2 oder 3 - zurück.
- ST_Distance - Changed: 3.0.0 - does not depend on SFCGAL anymore. Gibt die größte 3-dimensionale Distanz zwischen zwei geometrischen Objekten als Linie zurück
- ST_Intersection - Changed: 3.0.0 does not depend on SFCGAL. Computes a geometry representing the shared portion of geometries A and B.
- ST_Intersects - Changed: 3.0.0 SFCGAL version removed and native support for 2D TINS added. Tests if two geometries intersect (they have at least one point in common)
- ST_Union - Changed: 3.0.0 does not depend on SFCGAL. Computes a geometry representing the point-set union of the input geometries.

12.12.6 PostGIS Functions new or enhanced in 2.5

The functions given below are PostGIS functions that were added or enhanced.

Functions new in PostGIS 2.5

- PostGIS_Extensions_Upgrade - Availability: 2.5.0 Packages and upgrades PostGIS extensions (e.g. postgis_raster, postgis_topology, postgis_sfcgal) to given or latest version.
- ST_Angle - Verfügbarkeit: 2.5.0 Gibt den Winkel zwischen 3 Punkten oder zwischen 2 Vektoren (4 Punkte oder 2 Linien) zurück.
- ST_ChaikinSmoothing - Verfügbarkeit: 2.5.0 Returns a smoothed version of a geometry, using the Chaikin algorithm
- ST_FilterByM - Verfügbarkeit: 2.5.0 Removes vertices based on their M value
- ST_LineInterpolatePoints - Verfügbarkeit: 2.5.0 Gibt einen oder mehrere, entlang einer Linie interpolierte Punkte zurück.
- ST_OrientedEnvelope - Availability: 2.5.0. Returns a minimum-area rectangle containing a geometry.
- ST_QuantizeCoordinates - Verfügbarkeit: 2.5.0 Setzt die niedrigwertigsten Bits der Koordinaten auf Null

Functions enhanced in PostGIS 2.5

- ST_AsMVT - Erweiterung: 2.5.0 - Unterstützung von nebenläufigen Abfragen. Aggregate function returning a MVT representation of a set of rows.
- ST_AsText - Erweiterung: 2.5 - der optionale Parameter "precision" wurde eingeführt. Gibt die Well-known-Text(WKT) Darstellung der Geometrie/Geographie ohne die SRID Metadaten zurück.
- ST_Buffer - Erweiterung: 2.5.0 - ST_Buffer ermöglicht jetzt auch eine seitliche Pufferzonenberechnung über side=both|left|right. Computes a geometry covering all points within a given distance from a geometry.
- ST_GeomFromGeoJSON - Erweiterung: 2.5.0 unterstützt nun auch die Eingabe von json und jsonb. Nimmt als Eingabe eine GeoJSON-Darstellung der Geometrie und gibt ein geometrisches PostGIS-Objekt aus.
- ST_GeometricMedian - Erweiterung: 2.5.0 Unterstützung für M zur Gewichtung nach Punkten. Gibt den geometrischen Median eines Mehrfachpunktes zurück.
- ST_Intersects - Enhanced: 2.5.0 Supports GEOMETRYCOLLECTION. Tests if two geometries intersect (they have at least one point in common)
- ST_OffsetCurve - Erweiterung: ab 2.5 wird auch GEOMETRYCOLLECTION und MULTILINESTRING unterstützt. Returns an offset line at a given distance and side from an input line.
- ST_Scale - Enhanced: 2.5.0 support for scaling relative to a local origin (origin parameter) was introduced. Scales a geometry by given factors.
- ST_Split - Enhanced: 2.5.0 support for splitting a polygon by a multiline was introduced. Returns a collection of geometries created by splitting a geometry by another geometry.
- ST_Subdivide - Enhanced: 2.5.0 reuses existing points on polygon split, vertex count is lowered from 8 to 5. Computes a rectilinear subdivision of a geometry.

12.12.7 PostGIS Functions new or enhanced in 2.4

The functions given below are PostGIS functions that were added or enhanced.

Functions new in PostGIS 2.4

- ST_AsGeobuf - Verfügbarkeit: 2.4.0 Gibt eine Menge an Zeilen in der Geobuf Darstellung aus.
- ST_AsMVT - Verfügbarkeit: 2.4.0 Aggregate function returning a MVT representation of a set of rows.
- ST_AsMVTGeom - Verfügbarkeit: 2.4.0 Transforms a geometry into the coordinate space of a MVT tile.
- ST_Centroid - Verfügbarkeit: Mit 2.4.0 wurde die Unterstützung für den geograpischen Datentyp eingeführt. Gibt den geometrischen Schwerpunkt einer Geometrie zurück.
- ST_ForcePolygonCCW - Verfügbarkeit: 2.4.0 Richtet alle äußeren Ringe gegen den Uhrzeigersinn und alle inneren Ringe mit dem Uhrzeigersinn aus.
- ST_ForcePolygonCW - Verfügbarkeit: 2.4.0 Richtet alle äußeren Ringe im Uhrzeigersinn und alle inneren Ringe gegen den Uhrzeigersinn aus.
- ST_FrechetDistance - Verfügbarkeit: 2.4.0 - benötigt GEOS >= 3.7.0 Gibt den kürzesten 3-dimensionalen Abstand zwischen zwei geometrischen Objekten als Linie zurück
- ST_IsPolygonCCW - Verfügbarkeit: 2.2.0 Gibt TRUE zurück, wenn alle äußeren Ringe gegen den Uhrzeigersinn orientiert sind und alle inneren Ringe im Uhrzeigersinn ausgerichtet sind.
- ST_IsPolygonCW - Verfügbarkeit: 2.2.0 Gibt den Wert TRUE zurück, wenn alle äußeren Ringe im Uhrzeigersinn und alle inneren Ringe gegen den Uhrzeigersinn ausgerichtet sind.

Functions enhanced in PostGIS 2.4

- ST_AsTWKB - Erweiterung: 2.4.0 Hauptspeicher- und Geschwindigkeitsverbesserungen. Gibt die Geometrie als TWKB, aka "Tiny Well-known Binary" zurück
- ST_Covers - Enhanced: 2.4.0 Support for polygon in polygon and line in polygon added for geography type Tests if every point of B lies in A
- ST_CurveToLine - Erweiterung: ab 2.4.0 kann die Toleranz über die 'maximale Abweichung' und den 'maximalen Winkel' angegeben werden. Die symmetrische Ausgabe wurde hinzugefügt. Converts a geometry containing curves to a linear geometry.
- ST_Project - Enhanced: 2.4.0 Allow negative distance and non-normalized azimuth. Returns a point projected from a start point by a distance and bearing (azimuth).
- ST_Reverse - Erweiterung: mit 2.4.0 wurde die Unterstützung für Kurven eingeführt. Gibt die Geometrie in umgekehrter Knotenreihenfolge zurück.

Functions changed in PostGIS 2.4

- = - Änderung: 2.4.0, in Vorgängerversionen war dies die Gleichheit der umschreibenden Rechtecke, nicht die geometrische Gleichheit. Falls Sie auf Gleichheit der umschreibenden Rechtecke prüfen wollen, verwenden Sie stattdesse bitte . Gibt TRUE zurück, wenn die Koordinaten und die Reihenfolge der Koordinaten der Geometrie/Geographie A und der Geometrie/Geographie B ident sind.
- ST_Node - Changed: 2.4.0 this function uses GEOSNode internally instead of GEOSUnaryUnion. This may cause the resulting linestrings to have a different order and direction compared to PostGIS < 2.4. Nodes a collection of lines.

12.12.8 PostGIS Functions new or enhanced in 2.3

The functions given below are PostGIS functions that were added or enhanced.

Functions new in PostGIS 2.3

- &&&(geometry,gidx) - Verfügbarkeit: Mit 2.3.0 wurde die Unterstützung von Block Range INdexes (BRIN) eingeführt. Erfordert PostgreSQL 9.5+. Gibt TRUE zurück, wenn die (cached) n-D bounding box einer Geometrie eine n-D float precision bounding box (GIDX) schneidet.
- &&&(gidx,geometry) - Verfügbarkeit: Mit 2.3.0 wurde die Unterstützung von Block Range INdexes (BRIN) eingeführt. Erfordert PostgreSQL 9.5+. Gibt TRUE zurück, wenn eine n-D float precision bounding box (GIDX) eine (cached) n-D bounding box einer Geometrie schneidet.
- &&&(gidx,gidx) - Verfügbarkeit: Mit 2.3.0 wurde die Unterstützung von Block Range INdexes (BRIN) eingeführt. Erfordert PostgreSQL 9.5+. Gibt TRUE zurück, wenn sich zwei n-D float precision bounding boxes (GIDX) gegenseitig überschneiden.
- &&(box2df,box2df) - Verfügbarkeit: Mit 2.3.0 wurde die Unterstützung von Block Range INdexes (BRIN) eingeführt. Erfordert PostgreSQL 9.5+. Gibt TRUE zurück, wenn sich zwei 2D float precision Bounding Boxes (BOX2DF) überschneiden.
- &&(box2df,geometry) - Verfügbarkeit: Mit 2.3.0 wurde die Unterstützung von Block Range INdexes (BRIN) eingeführt. Erfordert PostgreSQL 9.5+. Gibt TRUE zurück, wenn eine 2D float precision bounding box (BOX2DF) eine Geometrie (cached) 2D bounding box schneidet.
- &&(geometry,box2df) - Verfügbarkeit: Mit 2.3.0 wurde die Unterstützung von Block Range INdexes (BRIN) eingeführt. Erfordert PostgreSQL 9.5+. Gibt TRUE zurück, wenn sich die 2D Bounding Box (cached) einer Geometrie mit einer 2D Bounding Box mit Gleitpunktgenauigkeit (BOX2DF) überschneidet.
- @(box2df,box2df) - Verfügbarkeit: Mit 2.3.0 wurde die Unterstützung von Block Range INdexes (BRIN) eingeführt. Erfordert PostgreSQL 9.5+. Gibt TRUE zurück, wenn eine 2D float precision bounding box (BOX2DF) innerhalb einer anderen 2D float precision bounding box enthalten ist.
- @(box2df,geometry) - Verfügbarkeit: Mit 2.3.0 wurde die Unterstützung von Block Range INdexes (BRIN) eingeführt. Erfordert PostgreSQL 9.5+. Gibt TRUE zurück, wenn eine 2D float precision bounding box (BOX2DF) in der 2D Bounding Box einer Geometrie enthalten ist..

- @(geometry,box2df) - Verfügbarkeit: Mit 2.3.0 wurde die Unterstützung von Block Range INdexes (BRIN) eingeführt. Erfordert PostgreSQL 9.5+. Gibt TRUE zurück, wenn die 2D Bounding Box einer Geometrie in einer 2D float precision Bbounding Box (BOX2DF) enthalten ist.
- ST_ClusterDBSCAN - Availability: 2.3.0 Window function that returns a cluster id for each input geometry using the DBSCAN algorithm.
- ST_ClusterKMeans - Availability: 2.3.0 Window function that returns a cluster id for each input geometry using the K-means algorithm.
- ST_GeneratePoints - Verfügbarkeit: 2.3.0 Generates random points contained in a Polygon or MultiPolygon.
- ST_GeometricMedian - Verfügbarkeit: 2.3.0 Gibt den geometrischen Median eines Mehrfachpunktes zurück.
- ST_MakeLine - Verfügbarkeit: Mit 2.3.0 wurde die Unterstützung zur Eingabe von MultiPoint Elementen eingeführt Erzeugt einen Linienzug aus einer Punkt-, Mehrfachpunkt- oder Liniengeometrie.
- ST_MinimumBoundingRadius - Verfügbarkeit: 2.3.0 Returns the center point and radius of the smallest circle that contains a geometry.
- ST_MinimumClearance - Verfügbarkeit: 2.3.0 Gibt das Mindestabstandsmaß für eine Geometrie zurück; ein Maß für die Robustheit einer Geometrie.
- ST_MinimumClearanceLine - Verfügbarkeit: 2.3.0 - benötigt GEOS >= 3.6.0 Gibt ein Linienstück mit zwei Punkten zurück, welche sich über das Mindestabstandsmaß erstreckt.
- ST_Normalize - Verfügbarkeit: 2.3.0 Gibt die Geometrie in Normalform zurück.
- ST_Points - Verfügbarkeit: 2.3.0 Gibt einen MultiPoint zurück, welcher alle Koordinaten einer Geometrie enthält.
- ST_VoronoiLines - Verfügbarkeit: 2.3.0 Returns the boundaries of the Voronoi diagram of the vertices of a geometry.
- ST_VoronoiPolygons - Verfügbarkeit: 2.3.0 Returns the cells of the Voronoi diagram of the vertices of a geometry.
- ST_WrapX - Availability: 2.3.0 requires GEOS Versammelt eine Geometrie um einen X-Wert
- ~(box2df,box2df) - Verfügbarkeit: Mit 2.3.0 wurde die Unterstützung von Block Range INdexes (BRIN) eingeführt. Erfordert PostgreSQL 9.5+. Gibt TRUE zurück, wenn eine 2D float precision bounding box (BOX2DF) eine andere 2D float precision bounding box (BOX2DF) enthält.
- ~(box2df,geometry) - Verfügbarkeit: Mit 2.3.0 wurde die Unterstützung von Block Range INdexes (BRIN) eingeführt. Erfordert PostgreSQL 9.5+. Gibt TRUE zurück, wenn eine 2D float precision bounding box (BOX2DF) die 2D Bounding Box einer Geometrie enthält.
- ~(geometry,box2df) - Verfügbarkeit: Mit 2.3.0 wurde die Unterstützung von Block Range INdexes (BRIN) eingeführt. Erfordert PostgreSQL 9.5+. Gibt TRUE zurück, wenn die 2D bounding box einer Geometrie eine 2D float precision bounding box (GIDX) enthält.

Functions enhanced in PostGIS 2.3

- ST_Contains - Enhanced: 2.3.0 Enhancement to PIP short-circuit extended to support MultiPoints with few points. Prior versions only supported point in polygon. Tests if every point of B lies in A, and their interiors have a point in common
- ST_Covers - Enhanced: 2.3.0 Enhancement to PIP short-circuit for geometry extended to support MultiPoints with few points. Prior versions only supported point in polygon. Tests if every point of B lies in A
- ST_Expand - Enhanced: 2.3.0 support was added to expand a box by different amounts in different dimensions. Returns a bounding box expanded from another bounding box or a geometry.
- ST_Intersects - Enhanced: 2.3.0 Enhancement to PIP short-circuit extended to support MultiPoints with few points. Prior versions only supported point in polygon. Tests if two geometries intersect (they have at least one point in common)
- ST_Segmentize - Enhanced: 2.3.0 Segmentize geography now produces equal-length subsegments Returns a modified geometry/geography having no segment longer than a given distance.

- ST_Transform - Enhanced: 2.3.0 support for direct PROJ.4 text was introduced. Return a new geometry with coordinates transformed to a different spatial reference system.
- ST_Within - Enhanced: 2.3.0 Enhancement to PIP short-circuit for geometry extended to support MultiPoints with few points. Prior versions only supported point in polygon. Tests if every point of A lies in B, and their interiors have a point in common

Functions changed in PostGIS 2.3

- ST_PointN - Änderung: 2.3.0 : negatives Indizieren verfügbar (-1 entspricht dem Endpunkt) Gibt die Anzahl der Stützpunkte eines ST_LineString oder eines ST_CircularString zurück.

12.12.9 PostGIS Functions new or enhanced in 2.2

The functions given below are PostGIS functions that were added or enhanced.

Functions new in PostGIS 2.2

- <<#>> - Verfügbarkeit: 2.2.0 -- KNN steht erst ab PostgreSQL 9.1+ zur Verfügung. Gibt die n-D Entfernung zwischen den Bounding Boxes von A und B zurück.
- <<->> - Verfügbarkeit: 2.2.0 -- KNN steht erst ab PostgreSQL 9.1+ zur Verfügung. Gibt die n-D Entfernung zwischen den geometrischen Schwerpunkten der Begrenzungsrechtecke/Bounding Boxes von A und B zurück.
- ST_3DDifference - Verfügbarkeit: 2.2.0 Perform 3D difference
- ST_3DUnion - Verfügbarkeit: 2.2.0 Perform 3D union.
- ST_ApproximateMedialAxis - Verfügbarkeit: 2.2.0 Berechnet die konvexe Hülle einer Geometrie.
- ST_AsEncodedPolyline - Verfügbarkeit: 2.2.0 Erzeugt eine codierte Polylinie aus einer LineString Geometrie.
- ST_AsTWKB - Verfügbarkeit: 2.2.0 Gibt die Geometrie als TWKB, aka "Tiny Well-known Binary" zurück
- ST_BoundingDiagonal - Verfügbarkeit: 2.2.0 Gibt die Diagonale des Umgebungsdreiecks der angegebenen Geometrie zurück.
- ST_CPAWithin - Availability: 2.2.0 Tests if the closest point of approach of two trajectories is within the specified distance.
- ST_ClipByBox2D - Availability: 2.2.0 Computes the portion of a geometry falling within a rectangle.
- ST_ClosestPointOfApproach - Availability: 2.2.0 Returns a measure at the closest point of approach of two trajectories.
- ST_ClusterIntersecting - Availability: 2.2.0 Aggregate function that clusters input geometries into connected sets.
- ST_ClusterWithin - Availability: 2.2.0 Aggregate function that clusters geometries by separation distance.
- ST_DistanceCPA - Availability: 2.2.0 Returns the distance between the closest point of approach of two trajectories.
- ST_ForceCurve - Verfügbarkeit: 2.2.0 Wandelt einen geometrischen in einen Kurven Datentyp um, soweit anwendbar.
- ST_IsPlanar - Availability: 2.2.0: This was documented in 2.1.0 but got accidentally left out in 2.1 release. Check if a surface is or not planar
- ST_IsSolid - Verfügbarkeit: 2.2.0 Test if the geometry is a solid. No validity check is performed.
- ST_IsValidTrajectory - Availability: 2.2.0 Tests if the geometry is a valid trajectory.
- ST_LineFromEncodedPolyline - Verfügbarkeit: 2.2.0 Erzeugt einen LineString aus einem codierten Linienzug.
- ST_MakeSolid - Verfügbarkeit: 2.2.0 Cast the geometry into a solid. No check is performed. To obtain a valid solid, the input geometry must be a closed Polyhedral Surface or a closed TIN.
- ST_RemoveRepeatedPoints - Verfügbarkeit: 2.2.0 Returns a version of a geometry with duplicate points removed.
- ST_SetEffectiveArea - Verfügbarkeit: 2.2.0 Sets the effective area for each vertex, using the Visvalingam-Whyatt algorithm.

- ST_SimplifyVW - Verfügbarkeit: 2.2.0 Returns a simplified version of a geometry, using the Visvalingam-Whyatt algorithm
- ST_Subdivide - Availability: 2.2.0 Computes a rectilinear subdivision of a geometry.
- ST_SwapOrdinates - Verfügbarkeit: 2.2.0 Gibt eine Version der Ausgangsgeometrie zurück, in der die angegebenen Ordinatenwerte ausgetauscht werden.
- ST_Volume - Verfügbarkeit: 2.2.0 Computes the volume of a 3D solid. If applied to surface (even closed) geometries will return 0.
- postgis.enable_outdb_rasters - Verfügbarkeit: 2.2.0 Eine boolesche Konfigurationsmöglichkeit um den Zugriff auf out-db Rasterbänder zu ermöglichen
- postgis.gdal_datapath - Verfügbarkeit: 2.2.0 Eine Konfigurationsmöglichkeit um den Wert von GDAL's GDAL_DATA Option zu setzen. Wenn sie nicht gesetzt ist, wird die Umgebungsvariable GDAL_DATA verwendet.
- postgis.gdal_enabled_drivers - Verfügbarkeit: 2.2.0 Eine Konfigurationsmöglichkeit um einen GDAL Treiber in der PostGIS Umgebung zu aktivieren. Beeinflusst die Konfigurationsvariable GDAL_SKIP von GDAL.
- |=| - Verfügbarkeit: 2.2.0. Index-unterstützt steht erst ab PostgreSQL 9.5+ zur Verfügung. Gibt die Entfernung zwischen den Trajektorien A und B, am Ort der dichtesten Annäherung, an.

Functions enhanced in PostGIS 2.2

- <-> - Verbesserung: 2.2.0 -- Echtes KNN ("K nearest neighbor") Verhalten für Geometrie und Geographie ab PostgreSQL 9.5+. Beachten Sie bitte, das KNN für Geographie auf der Späre und nicht auf dem Sphäroid beruht. Für PostgreSQL 9.4 und darunter, wird die Berechnung nur auf Basis des Centroids der Box unterstützt. Gibt die 2D Entfernung zwischen A und B zurück.
- ST_Area - Erweiterung: 2.2.0 - die Messung auf dem Referenzellipsoid wird mit der Bibliothek "GeographicLib" durchgeführt. Dadurch wurde die Genauigkeit und die Robustheit erhöht. Um die Vorteile dieser neuen Funktionalität zu nutzen, benötigen Sie Proj >= 4.9.0. Gibt den geometrischen Schwerpunkt einer Geometrie zurück.
- ST_AsX3D - Erweiterung: 2.2.0: Unterstützung für geographische Koordinaten und Vertauschen der Achsen (x/y, Länge/Breite). Für nähere Details siehe Optionen. Gibt eine Geometrie im X3D XML Knotenelement-Format zurück: ISO-IEC-19776-1.2-X3DEncodings-XML
- ST_Azimuth - Erweiterung: 2.2.0 die Messungen auf dem Referenzellipsoid werden mit der Bibliothek "GeographicLib" durchgeführt. Dadurch wurde die Genauigkeit und die Robustheit erhöht. Um die Vorteile dieser neuen Funktionalität zu nutzen, benötigen Sie Proj >= 4.9.0. Gibt die 2-dimenionale kürzeste Strecke zwischen zwei Geometrien als Linie zurück
- ST_Distance - Erweiterung: 2.2.0 - die Messung auf dem Referenzellipsoid wird mit der Bibliothek "GeographicLib" durchgeführt. Dadurch wurde die Genauigkeit und die Robustheit erhöht. Um die Vorteile dieser neuen Funktionalität zu nutzen, benötigen Sie Proj >= 4.9.0. Gibt die größte 3-dimensionale Distanz zwischen zwei geometrischen Objekten als Linie zurück
- ST_Scale - Enhanced: 2.2.0 support for scaling all dimension (factor parameter) was introduced. Scales a geometry by given factors.
- ST_Split - Enhanced: 2.2.0 support for splitting a line by a multiline, a multipoint or (multi)polygon boundary was introduced. Returns a collection of geometries created by splitting a geometry by another geometry.
- ST_Summary - Erweiterung: 2.2.0 Unterstützung für TIN und Kurven Gibt eine Zusammenfassung des Inhalts einer Geometrie wieder.

Functions changed in PostGIS 2.2

- <-> - Änderung: 2.2.0 -- Da für Anwender von PostgreSQL 9.5 der alte hybride Syntax langsamer sein kann, möchten sie diesen Hack eventuell loswerden, falls der Code nur auf PostGIS 2.2+ 9.5+ läuft. Siehe die unteren Beispiele. Gibt die 2D Entfernung zwischen A und B zurück.

- ST_3DClosestPoint - Änderung: 2.2.0 - Wenn 2 geometrische Objekte in 2D übergegeben werden, wird ein 2D-Punkt zurückgegeben (anstelle wie früher 0 für ein fehlendes Z). Im Falle von 2D und 3D, wird für fehlende Z nicht länger 0 angenommen. Gibt den 3-dimensionalen Punkt auf g1 zurück, der den kürzesten Abstand zu g2 hat. Dies ist der Anfangspunkt des kürzesten Abstands in 3D.
- ST_3DDistance - Änderung: 2.2.0 - Im Falle von 2D und 3D wird für ein fehlendes Z nicht mehr 0 angenommen. Für den geometrischen Datentyp. Es wird der geringste 3-dimensionale kartesische Abstand (basierend auf dem Koordinatenreferenzsystem) zwischen zwei geometrischen Objekten in projizierten Einheiten zurückgegeben.
- ST_3DLongestLine - Änderung: 2.2.0 - Wenn 2 geometrische Objekte in 2D übergegeben werden, wird ein 2D-Punkt zurückgegeben (anstelle wie früher 0 für ein fehlendes Z). Im Falle von 2D und 3D, wird für fehlende Z nicht länger 0 angenommen. Gibt die größte 3-dimensionale Distanz zwischen zwei geometrischen Objekten als Linie zurück
- ST_3DMaxDistance - Änderung: 2.2.0 - Im Falle von 2D und 3D wird für ein fehlendes Z nicht mehr 0 angenommen. Für den geometrischen Datentyp. Gibt die maximale 3-dimensionale kartesische Distanz (basierend auf dem Koordinatenreferenzsystem) zwischen zwei geometrischen Objekten in projizierten Einheiten zurück.
- ST_3DShortestLine - Änderung: 2.2.0 - Wenn 2 geometrische Objekte in 2D übergegeben werden, wird ein 2D-Punkt zurückgegeben (anstelle wie früher 0 für ein fehlendes Z). Im Falle von 2D und 3D, wird für fehlende Z nicht länger 0 angenommen. Gibt den kürzesten 3-dimensionalen Abstand zwischen zwei geometrischen Objekten als Linie zurück
- ST_DistanceSphere - Änderung: 2.2.0 In Vorgängerversionen als ST_Distance_Sphere bezeichet. Gibt die kürzeste Distanz zwischen zwei geometrischen Objekten zurück, die über Länge, Breite und ein bestimmtes Referenzellipsoid gegeben sind. Vorgängerversionen von PostGIS 1.5 unterstützten nur Punkte.
- ST_DistanceSpheroid - Änderung: 2.2.0 In Vorgängerversionen als ST_Distance_Spheroid bezeichet. Gibt die kürzeste Distanz zwischen zwei geometrischen Objekten zurück, die über Länge, Breite und ein bestimmtes Referenzellipsoid gegeben sind. Vorgängerversionen von PostGIS 1.5 unterstützten nur Punkte.
- ST_Equals - Changed: 2.2.0 Returns true even for invalid geometries if they are binary equal Tests if two geometries include the same set of points
- ST_LengthSpheroid - Änderung: 2.2.0 In Vorgängerversionen als ST_Length_Spheroid bezeichet.und mit dem Alias "ST_3DLength_S versehen Gibt den geometrischen Schwerpunkt einer Geometrie zurück.
- ST_MemSize - Changed: 2.2.0 name changed to ST_MemSize to follow naming convention. Gibt den Geometrietyp des ST_Geometry Wertes zurück.
- ST_PointInsideCircle - Changed: 2.2.0 In prior versions this was called ST_Point_Inside_Circle Tests if a point geometry is inside a circle defined by a center and radius

12.12.10 PostGIS Functions new or enhanced in 2.1

The functions given below are PostGIS functions that were added or enhanced.

Functions new in PostGIS 2.1

- ST_3DArea - Verfügbarkeit: 2.1.0 Computes area of 3D surface geometries. Will return 0 for solids.
- ST_3DIntersection - Verfügbarkeit: 2.1.0 Perform 3D intersection
- ST_Box2dFromGeoHash - Verfügbarkeit: 2.1.0 Gibt die BOX2D einer GeoHash Zeichenkette zurück.
- ST_DelaunayTriangles - Verfügbarkeit: 2.1.0 Returns the Delaunay triangulation of the vertices of a geometry.
- ST_Extrude - Verfügbarkeit: 2.1.0 Extrude a surface to a related volume
- ST_ForceLHR - Verfügbarkeit: 2.1.0 Force LHR orientation
- ST_GeomFromGeoHash - Verfügbarkeit: 2.1.0 Gibt die Geometrie einer GeoHash Zeichenfolge zurück.
- ST_MinkowskiSum - Verfügbarkeit: 2.1.0 Performs Minkowski sum

- ST_Orientation - Verfügbarkeit: 2.1.0 Determine surface orientation
- ST_PointFromGeoHash - Verfügbarkeit: 2.1.0 Gibt einen Punkt von einer GeoHash Zeichenfolge zurück.
- ST_StraightSkeleton - Verfügbarkeit: 2.1.0 Berechnet die konvexe Hülle einer Geometrie.
- ST_Tesselate - Verfügbarkeit: 2.1.0 Perform surface Tesselation of a polygon or polyhedralsurface and returns as a TIN or collection of TINS
- postgis.backend - Verfügbarkeit: 2.1.0 Dieses Backend stellt eine Funktion zur Auswahl zwischen GEOS und SFCGAL zur Verfügung.
- postgis_sfcgal_version - Verfügbarkeit: 2.1.0 Returns the version of SFCGAL in use

Functions enhanced in PostGIS 2.1

- ST_AsGML - Erweiterung: 2.1.0 Für GML 3 wurde die Unterstützung einer ID eingeführt. Gibt die Geometrie als GML-Element - Version 2 oder 3 - zurück.
- ST_Boundary - Erweiterung: mit 2.1.0 wurde die Unterstützung von Dreiecken eingeführt Gibt die abgeschlossene Hülle aus der kombinierten Begrenzung der Geometrie zurück.
- ST_DWithin - Enhanced: 2.1.0 improved speed for geography. See Making Geography faster for details. Tests if two geometries are within a given distance
- ST_DWithin - Enhanced: 2.1.0 support for curved geometries was introduced. Tests if two geometries are within a given distance
- ST_Distance - Enhanced: 2.1.0 Geschwindigkeitsverbesserung beim geographischen Datentyp. Siehe Making Geography faster für Details. Gibt die größte 3-dimensionale Distanz zwischen zwei geometrischen Objekten als Linie zurück
- ST_Distance - Erweiterung: 2.1.0 - Unterstützung für Kurven beim geometrischen Datentyp eingeführt. Gibt die größte 3-dimensionale Distanz zwischen zwei geometrischen Objekten als Linie zurück
- ST_DumpPoints - Enhanced: 2.1.0 Faster speed. Reimplemented as native-C. Gibt eine Zusammenfassung des Inhalts einer Geometrie wieder.
- ST_MakeValid - Enhanced: 2.1.0, added support for GEOMETRYCOLLECTION and MULTIPOINT. Attempts to make an invalid geometry valid without losing vertices.
- ST_Segmentize - Erweiterung: mit 2.1.0 wurde die Unterstützung des geographischen Datentyps eingeführt. Returns a modified geometry/geography having no segment longer than a given distance.
- ST_Summary - Erweiterung: 2.1.0 S-Flag, diese zeigt an ob das Koordinatenreferenzsystem bekannt ist Gibt eine Zusammenfassung des Inhalts einer Geometrie wieder.

Functions changed in PostGIS 2.1

- ST_EstimatedExtent - Changed: 2.1.0. Up to 2.0.x this was called ST_Estimated_Extent. Returns the estimated extent of a spatial table.
- ST_Force2D - Änderung: 2.1.0. Bis zu 2.0.x wurde diese Funktion mit ST_Force_2D bezeichnet. Die Geometrien in einen "2-dimensionalen Modus" zwingen.
- ST_Force3D - Änderung: 2.1.0. Bis zu 2.0.x wurde diese Funktion mit ST_Force_3D bezeichnet. Zwingt die Geometrien in einen XYZ Modus. Dies ist ein Alias für ST_Force3DZ.
- ST_Force3DM - Änderung: 2.1.0. Bis zu 2.0.x wurde diese Funktion mit ST_Force_3DM bezeichnet. Zwingt die Geometrien in einen XYM Modus.
- ST_Force3DZ - Änderung: 2.1.0. Bis zu 2.0.x wurde diese Funktion mit ST_Force_3DZ bezeichnet. Zwingt die Geometrien in einen XYZ Modus.

- ST_Force4D - Änderung: 2.1.0. Bis zu 2.0.x wurde diese Funktion mit ST_Force_4D bezeichnet. Zwingt die Geometrien in einen XYZM Modus.
- ST_ForceCollection - Änderung: 2.1.0. Bis zu 2.0.x wurde diese Funktion mit ST_Force_Collection bezeichnet. Wandelt eine Geometrie in eine GEOMETRYCOLLECTION um.
- ST_LineInterpolatePoint - Änderung: 2.1.0. Bis zu 2.0.x wurde diese Funktion mit ST_Line_Interpolate_Point bezeichnet. Gibt einen oder mehrere, entlang einer Linie interpolierte Punkte zurück.
- ST_LineLocatePoint - Änderung: 2.1.0. Bis zu 2.0.x wurde diese Funktion mit ST_Line_Locate_Point bezeichnet. Returns the fractional location of the closest point on a line to a point.
- ST_LineSubstring - Änderung: 2.1.0. Bis zu 2.0.x wurde diese Funktion mit ST_Line_Substring bezeichnet. Returns the part of a line between two fractional locations.
- ST_Segmentize - Changed: 2.1.0 As a result of the introduction of geography support, the usage ST_Segmentize('LINESTRING(1 2, 3 4)', 0.5) causes an ambiguous function error. The input needs to be properly typed as a geometry or geography. Use ST_GeomFromText, ST_GeogFromText or a cast to the required type (e.g. ST_Segmentize('LINESTRING(1 2, 3 4)'::geometry, 0.5)) Returns a modified geometry/geography having no segment longer than a given distance.

12.12.11 PostGIS Functions new or enhanced in 2.0

The functions given below are PostGIS functions that were added or enhanced.

Functions new in PostGIS 2.0

- &&& - Verfügbarkeit: 2.0.0 Gibt TRUE zurück, wenn A's n-D bounding box B's n-D bounding box schneidet.
- <#> - Verfügbarkeit: 2.0.0 -- KNN steht erst ab PostgreSQL 9.1+ zur Verfügung Gibt die 2D Entfernung zwischen den Bounding Boxes von A und B zurück
- <-> - Verfügbarkeit: 2.0.0 -- Weak KNN liefert nearest neighbors, welche sich auf die Entfernung der Centroide der Geometrien, anstatt auf den tatsächlichen Entfernungen, stützen. Genaue Ergebnisse für Punkte, ungenau für alle anderen Geometrietypen. Verfügbar ab PostgreSQL 9.1+. Gibt die 2D Entfernung zwischen A und B zurück.
- ST_3DClosestPoint - Verfügbarkeit: 2.0.0 Gibt den 3-dimensionalen Punkt auf g1 zurück, der den kürzesten Abstand zu g2 hat. Dies ist der Anfangspunkt des kürzesten Abstands in 3D.
- ST_3DDFullyWithin - Verfügbarkeit: 2.0.0 Tests if two 3D geometries are entirely within a given 3D distance
- ST_3DDWithin - Verfügbarkeit: 2.0.0 Tests if two 3D geometries are within a given 3D distance
- ST_3DDistance - Verfügbarkeit: 2.0.0 Für den geometrischen Datentyp. Es wird der geringste 3-dimensionale kartesische Abstand (basierend auf dem Koordinatenreferenzsystem) zwischen zwei geometrischen Objekten in projizierten Einheiten zurückgegeben.
- ST_3DIntersects - Verfügbarkeit: 2.0.0 Tests if two geometries spatially intersect in 3D - only for points, linestrings, polygons, polyhedral surface (area)
- ST_3DLongestLine - Verfügbarkeit: 2.0.0 Gibt die größte 3-dimensionale Distanz zwischen zwei geometrischen Objekten als Linie zurück
- ST_3DMaxDistance - Verfügbarkeit: 2.0.0 Für den geometrischen Datentyp. Gibt die maximale 3-dimensionale kartesische Distanz (basierend auf dem Koordinatenreferenzsystem) zwischen zwei geometrischen Objekten in projizierten Einheiten zurück.
- ST_3DShortestLine - Verfügbarkeit: 2.0.0 Gibt den kürzesten 3-dimensionalen Abstand zwischen zwei geometrischen Objekten als Linie zurück
- ST_AsLatLonText - Verfügbarkeit: 2.0 Gibt die "Grad, Minuten, Sekunden"-Darstellung für den angegebenen Punkt aus.
- ST_AsX3D - Verfügbarkeit: 2.0.0: ISO-IEC-19776-1.2-X3DEncodings-XML Gibt eine Geometrie im X3D XML Knotenelement-Format zurück: ISO-IEC-19776-1.2-X3DEncodings-XML

- ST_CollectionHomogenize - Verfügbarkeit: 2.0.0 Returns the simplest representation of a geometry collection.
- ST_ConcaveHull - Verfügbarkeit: 2.0.0 Computes a possibly concave geometry that contains all input geometry vertices
- ST_FlipCoordinates - Verfügbarkeit: 2.0.0 Returns a version of a geometry with X and Y axis flipped.
- ST_GeomFromGeoJSON - Verfügbarkeit: 2.0.0 benötigt - JSON-C >= 0.9 Nimmt als Eingabe eine GeoJSON-Darstellung der Geometrie und gibt ein geometrisches PostGIS-Objekt aus.
- ST_InterpolatePoint - Verfügbarkeit: 2.0.0 Für einen gegebenen Punkt wird die Kilometrierung auf dem nächstliegenden Punkt einer Geometrie zurück.
- ST_IsValidDetail - Verfügbarkeit: 2.0.0 Returns a valid_detail row stating if a geometry is valid or if not a reason and a location.
- ST_IsValidReason - Availability: 2.0 version taking flags. Returns text stating if a geometry is valid, or a reason for invalidity.
- ST_MakeLine - Verfügbarkeit: 2.0.0 - Unterstützung zur Eingabe von LineString Elementen eingeführt Erzeugt einen Linienzug aus einer Punkt-, Mehrfachpunkt- oder Liniengeometrie.
- ST_MakeValid - Verfügbarkeit: 2.0.0 Attempts to make an invalid geometry valid without losing vertices.
- ST_Node - Verfügbarkeit: 2.0.0 Nodes a collection of lines.
- ST_NumPatches - Verfügbarkeit: 2.0.0 Gibt die Anzahl der Maschen einer polyedrischen Oberfläche aus. Gibt NULL zurück, wenn es sich nicht um polyedrische Geometrien handelt.
- ST_OffsetCurve - Verfügbarkeit: 2.0 Returns an offset line at a given distance and side from an input line.
- ST_PatchN - Verfügbarkeit: 2.0.0 Gibt den Geometrietyp des ST_Geometry Wertes zurück.
- ST_Perimeter - Verfügbarkeit: Mit 2.0.0 wurde die Unterstützung für geograpischen Koordinaten eingeführt Returns the length of the boundary of a polygonal geometry or geography.
- ST_Project - Verfügbarkeit: 2.0.0 Returns a point projected from a start point by a distance and bearing (azimuth).
- ST_RelateMatch - Verfügbarkeit: 2.0.0 Tests if a DE-9IM Intersection Matrix matches an Intersection Matrix pattern
- ST_SharedPaths - Verfügbarkeit: 2.0.0 Gibt eine Sammelgeometrie zurück, welche die gemeinsamen Strecken der beiden eingegebenen LineStrings/MultiLinestrings enthält.
- ST_Snap - Verfügbarkeit: 2.0.0 Fängt die Segmente und Knoten einer Eingabegeometrie an den Knoten einer Referenzgeometrie.
- ST_Split - Availability: 2.0.0 requires GEOS Returns a collection of geometries created by splitting a geometry by another geometry.
- ST_UnaryUnion - Verfügbarkeit: 2.0.0 Computes the union of the components of a single geometry.

Functions enhanced in PostGIS 2.0

- && - Erweiterung: Mit 2.0.0 wurde die Unterstützung für polyedrische Oberflächen eingeführt. Gibt TRUE zurück, wenn die 2D Bounding Box von A die 2D Bounding Box von B schneidet.
- AddGeometryColumn - Verbesserung: 2.0.0 use_typmod Argument eingeführt. Standardmäßig wird eine typmod Geometrie anstelle einer Constraint-basierten Geometrie erzeugt. Entfernt eine Geometriespalte aus einer räumlichen Tabelle.
- Box2D - Enhanced: 2.0.0 support for Polyhedral surfaces, Triangles and TIN was introduced. Returns a BOX2D representing the 2D extent of a geometry.
- Box3D - Enhanced: 2.0.0 support for Polyhedral surfaces, Triangles and TIN was introduced. Returns a BOX3D representing the 3D extent of a geometry.
- GeometryType - Erweiterung: Mit 2.0.0 wurde die Unterstützung für polyedrische Oberflächen, Dreiecke und TIN eingeführt. Gibt den Geometrietyp des ST_Geometry Wertes zurück.

- Populate_Geometry_Columns - Erweiterung: 2.0.0 Der optionale Übergabewert use_typmod wurde eingeführt, um bestimmen zu können, ob die Spalten mit Typmodifikatoren oder mit Check-Constraints erstellt werden sollen. Ensures geometry columns are defined with type modifiers or have appropriate spatial constraints.
- ST_3DExtent - Enhanced: 2.0.0 support for Polyhedral surfaces, Triangles and TIN was introduced. Aggregate function that returns the 3D bounding box of geometries.
- ST_Affine - Enhanced: 2.0.0 support for Polyhedral surfaces, Triangles and TIN was introduced. Apply a 3D affine transformation to a geometry.
- ST_Area - Erweiterung: Mit 2.0.0 wurde 2D-Unterstützung für polyedrische Oberflächen eingeführt. Gibt den geometrischen Schwerpunkt einer Geometrie zurück.
- ST_AsBinary - Erweiterung: 2.0.0 - Unterstützung für polyedrische Oberflächen, Dreiecke und TIN eingeführt. Return the OGC/ISO Well-Known Binary (WKB) representation of the geometry/geography without SRID meta data.
- ST_AsBinary - Erweiterung: 2.0.0 - Unterstützung für höherdimensionale Koordinatensysteme eingeführt. Return the OGC/ISO Well-Known Binary (WKB) representation of the geometry/geography without SRID meta data.
- ST_AsBinary - Erweiterung: 2.0.0 Unterstützung zum Festlegen des Endian beim geographischen Datentyp eingeführt. Return the OGC/ISO Well-Known Binary (WKB) representation of the geometry/geography without SRID meta data.
- ST_AsEWKB - Erweiterung: 2.0.0 - Unterstützung für polyedrische Oberflächen, Dreiecke und TIN eingeführt. Return the Extended Well-Known Binary (EWKB) representation of the geometry with SRID meta data.
- ST_AsEWKT - Erweiterung: Mit 2.0.0 wurde die Unterstützung für den geographischen Datentyp, polyedrische Oberflächen, Dreiecke und TIN eingeführt. Gibt die Well-known-Text(WKT) Darstellung der Geometrie mit den SRID-Metadaten zurück.
- ST_AsGML - Erweiterung: 2.0.0 Unterstützung durch Präfix eingeführt. Für GML3 wurde die Option 4 eingeführt, um die Verwendung von LineString anstatt von Kurven für Linien zu erlauben. Ebenfalls wurde die GML3 Unterstützung für polyedrische Oberflächen und TINS eingeführt, sowie die Option 32 zur Ausgabe der BBox. Gibt die Geometrie als GML-Element - Version 2 oder 3 - zurück.
- ST_AsKML - Erweiterung: 2.0.0 - Präfix Namensraum hinzugefügt. Standardmäßig kein Präfix Gibt die Geometrie als GML-Element - Version 2 oder 3 - zurück.
- ST_Azimuth - Erweiterung: mit 2.0.0 wurde die Unterstützung des geographischen Datentyps eingeführt. Gibt die 2-dimenionale kürzeste Strecke zwischen zwei Geometrien als Linie zurück
- ST_Dimension - Erweiterung: 2.0.0 - Unterstützung für polyedrische Oberflächen und TIN eingeführt. Gibt die Dimension der Koordinaten für den Wert von ST_Geometry zurück.
- ST_Dump - Erweiterung: Mit 2.0.0 wurde die Unterstützung für polyedrische Oberflächen, Dreiecke und TIN eingeführt. Returns a set of geometry_dump rows for the components of a geometry.
- ST_DumpPoints - Erweiterung: Mit 2.0.0 wurde die Unterstützung für polyedrische Oberflächen, Dreiecke und TIN eingeführt. Gibt eine Zusammenfassung des Inhalts einer Geometrie wieder.
- ST_Expand - Enhanced: 2.0.0 support for Polyhedral surfaces, Triangles and TIN was introduced. Returns a bounding box expanded from another bounding box or a geometry.
- ST_Extent - Enhanced: 2.0.0 support for Polyhedral surfaces, Triangles and TIN was introduced. Aggregate function that returns the bounding box of geometries.
- ST_Force2D - Erweiterung: Mit 2.0.0 wurde die Unterstützung für polyedrische Oberflächen eingeführt. Die Geometrien in einen "2-dimensionalen Modus" zwingen.
- ST_Force3D - Erweiterung: Mit 2.0.0 wurde die Unterstützung für polyedrische Oberflächen eingeführt. Zwingt die Geometrien in einen XYZ Modus. Dies ist ein Alias für ST_Force3DZ.
- ST_Force3DZ - Erweiterung: Mit 2.0.0 wurde die Unterstützung für polyedrische Oberflächen eingeführt. Zwingt die Geometrien in einen XYZ Modus.

- ST_ForceCollection - Erweiterung: Mit 2.0.0 wurde die Unterstützung für polyedrische Oberflächen eingeführt. Wandelt eine Geometrie in eine GEOMETRYCOLLECTION um.
- ST_ForceRHR - Erweiterung: Mit 2.0.0 wurde die Unterstützung für polyedrische Oberflächen eingeführt. Orientiert die Knoten in einem Polygon so, dass sie der Drei-Finger-Regel folgen.
- ST_GMLToSQL - Erweiterung: Mit 2.0.0 wurde die Unterstützung für polyedrische Oberflächen und TIN eingeführt. Gibt einen spezifizierten ST_Geometry Wert aus einer GML-Darstellung zurück. Dies ist ein Aliasname für ST_GeomFromGML
- ST_GMLToSQL - Erweiterung: 2.0.0 Standardwert für den optionalen Parameter SRID eingefügt. Gibt einen spezifizierten ST_Geometry Wert aus einer GML-Darstellung zurück. Dies ist ein Aliasname für ST_GeomFromGML
- ST_GeomFromEWKB - Erweiterung: Mit 2.0.0 wurde die Unterstützung für polyedrische Oberflächen und TIN eingeführt. Gibt einen geometrischen Datentyp (ST_Geometry) aus einer Well-known-Binary (WKB) Darstellung zurück.
- ST_GeomFromEWKT - Erweiterung: Mit 2.0.0 wurde die Unterstützung für polyedrische Oberflächen und TIN eingeführt. Gibt einen spezifizierten ST_Geometry-Wert von einer erweiterten Well-known-Text Darstellung (EWKT) zurück.
- ST_GeomFromGML - Erweiterung: Mit 2.0.0 wurde die Unterstützung für polyedrische Oberflächen und TIN eingeführt. Nimmt als Eingabe eine GML-Darstellung der Geometrie und gibt ein geometrisches PostGIS-Objekt aus.
- ST_GeomFromGML - Erweiterung: 2.0.0 Standardwert für den optionalen Parameter SRID eingefügt. Nimmt als Eingabe eine GML-Darstellung der Geometrie und gibt ein geometrisches PostGIS-Objekt aus.
- ST_GeometryN - Erweiterung: Mit 2.0.0 wurde die Unterstützung für polyedrische Oberflächen, Dreiecke und TIN eingeführt. Gibt den Geometrietyp des ST_Geometry Wertes zurück.
- ST_GeometryType - Erweiterung: Mit 2.0.0 wurde die Unterstützung für polyedrische Oberflächen eingeführt. Gibt den Geometrietyp des ST_Geometry Wertes zurück.
- ST_IsClosed - Erweiterung: Mit 2.0.0 wurde die Unterstützung für polyedrische Oberflächen eingeführt. Gibt den Wert TRUE zurück, wenn die Anfangs- und Endpunkte des LINESTRING's zusammenfallen. Bei polyedrischen Oberflächen, wenn sie geschlossen (volumetrisch) sind.
- ST_MakeEnvelope - Erweiterung: 2.0: es wurde die Möglichkeit eingeführt, eine Einhüllende/Envelope festzulegen, ohne dass die SRID spezifiziert ist. Erzeugt ein rechteckiges Polygon aus den gegebenen Minimum- und Maximumwerten. Die Eingabewerte müssen in dem Koordinatenreferenzsystem sein, welches durch die SRID angegeben wird.
- ST_MakeValid - Enhanced: 2.0.1, speed improvements Attempts to make an invalid geometry valid without losing vertices.
- ST_NPoints - Erweiterung: Mit 2.0.0 wurde die Unterstützung für polyedrische Oberflächen eingeführt. Gibt die Anzahl der Punkte (Knoten) einer Geometrie zurück.
- ST_NumGeometries - Erweiterung: Mit 2.0.0 wurde die Unterstützung für polyedrische Oberflächen, Dreiecke und TIN eingeführt. Gibt die Anzahl der Punkte einer Geometrie zurück. Funktioniert für alle Geometrien.
- ST_Relate - Enhanced: 2.0.0 - added support for specifying boundary node rule. Tests if two geometries have a topological relationship matching an Intersection Matrix pattern, or computes their Intersection Matrix
- ST_Rotate - Enhanced: 2.0.0 support for Polyhedral surfaces, Triangles and TIN was introduced. Rotates a geometry about an origin point.
- ST_Rotate - Enhanced: 2.0.0 additional parameters for specifying the origin of rotation were added. Rotates a geometry about an origin point.
- ST_RotateX - Enhanced: 2.0.0 support for Polyhedral surfaces, Triangles and TIN was introduced. Rotates a geometry about the X axis.
- ST_RotateY - Enhanced: 2.0.0 support for Polyhedral surfaces, Triangles and TIN was introduced. Rotates a geometry about the Y axis.
- ST_RotateZ - Enhanced: 2.0.0 support for Polyhedral surfaces, Triangles and TIN was introduced. Rotates a geometry about the Z axis.

- ST_Scale - Enhanced: 2.0.0 support for Polyhedral surfaces, Triangles and TIN was introduced. Scales a geometry by given factors.
- ST_ShiftLongitude - Erweiterung: Mit 2.0.0 wurde die Unterstützung für polyedrische Oberflächen und TIN eingeführt. Shifts the longitude coordinates of a geometry between -180..180 and 0..360.
- ST_Summary - Erweiterung: 2.0.0 Unterstützung für geographische Koordinaten hinzugefügt Gibt eine Zusammenfassung des Inhalts einer Geometrie wieder.
- ST_Transform - Enhanced: 2.0.0 support for Polyhedral surfaces was introduced. Return a new geometry with coordinates transformed to a different spatial reference system.

Functions changed in PostGIS 2.0

- AddGeometryColumn - Änderung: 2.0.0 Diese Funktion aktualisiert die geometry_columns Tabelle nicht mehr, da geometry_columns jetzt ein View ist, welcher den Systemkatalog ausliest. Standardmäßig werden auch keine Bedingungen/constraints erzeugt, sondern es wird der in PostgreSQL integrierte Typmodifikaor verwendet. So entspricht zum Beispiel die Erzeugung einer wgs84 POINT Spalte mit dieser Funktion: ALTER TABLE some_table ADD COLUMN geom geometry(Point,4326); Entfernt eine Geometriespalte aus einer räumlichen Tabelle.
- AddGeometryColumn - Änderung: 2.0.0 Falls Sie das alte Verhalten mit Constraints wünschen, setzen Sie bitte use_typmod vom standardmäßigen true auf false. Entfernt eine Geometriespalte aus einer räumlichen Tabelle.
- AddGeometryColumn - Änderung: 2.0.0 Views können nicht mehr händisch in "geometry_columns" registriert werden. Views auf eine Geometrie in Typmod-Tabellen, bei denen keine Adapterfunktion verwendet wird, registrieren sich selbst auf korrekte Weise, da sie die Typmod-Verhaltensweise von der Spalte der Stammtabelle erben. Views die ein geometrische Funktion ausführen die eine andere Geometrie ausgibt, benötigen die Umwandlung in eine Typmod-Geometrie, damit die Geometrie des Views korrekt in "geometry_columns" registriert wird. Siehe . Entfernt eine Geometriespalte aus einer räumlichen Tabelle.
- DropGeometryColumn - Änderung: 2.0.0 Diese Funktion wurde zwecks Abwärtskompatibilität eingeführt. Seit geometry_columns ein View auf den Systemkatalog ist, können Sie die Geometriespalte, so wie jede andere Tabellenspalte, mit ALTER TABLE löschen. Entfernt eine Geometriespalte aus einer räumlichen Tabelle.
- DropGeometryTable - Änderung: 2.0.0 Diese Funktion wurde zwecks Abwärtskompatibilität eingeführt. Seit geometry_columns ein View auf den Systemkatalog ist, können Sie eine Tabelle mit einer Geometriespalte, so wie jede andere Tabelle, mit DROP TABLE löschen. Löscht eine Tabelle und alle Referenzen in dem geometry_columns View.
- Populate_Geometry_Columns - Änderung: 2.0.0 Standardmäßig werden nun Typmodifikatoren anstelle von Check-Constraints für die Beschränkung des Geometrietyps verwendet. Sie können nach wie vor stattdessen die Verhaltensweise mit Check-Constraints verwenden, indem Sie die neu eingeführte Variable use_typmod auf FALSE setzen. Ensures geometry columns are defined with type modifiers or have appropriate spatial constraints.
- ST_3DExtent - Changed: 2.0.0 In prior versions this used to be called ST_Extent3D Aggregate function that returns the 3D bounding box of geometries.
- ST_3DLength - Änderung: 2.0.0 In Vorgängerversionen als ST_Length3D bezeichet. Gibt den geometrischen Schwerpunkt einer Geometrie zurück.
- ST_3DMakeBox - Changed: 2.0.0 In prior versions this used to be called ST_MakeBox3D Creates a BOX3D defined by two 3D point geometries.
- ST_3DPerimeter - Änderung: 2.0.0 In Vorgängerversionen als ST_Perimeter3D bezeichet. Gibt den geometrischen Schwerpunkt einer Geometrie zurück.
- ST_AsBinary - Änderung: 2.0.0 - Eingabewerte für diese Funktion dürfen nicht "unknown" sein -- es muss sich um eine Geometrie handeln. Konstrukte, wie ST_AsBinary('POINT(1 2)'), sind nicht länger gültig und geben folgende Fehlermeldung aus: n st_asbinary(unknown) is not unique error. Dieser Code muss in ST_AsBinary('POINT(1 2)'::geometry); geändert werden. Falls dies nicht möglich ist, so installieren Sie bitte legacy.sql. Return the OGC/ISO Well-Known Binary (WKB) representation of the geometry/geography without SRID meta data.
- ST_AsGML - Änderung: 2.0.0 verwendet standardmäßig benannte Argumente. Gibt die Geometrie als GML-Element - Version 2 oder 3 - zurück.

- ST_AsGeoJSON - Änderung: 2.0.0 Unterstützung für Standardargumente und benannte Argumente. Return a geometry as a GeoJSON element.
- ST_AsSVG - Änderung: 2.0.0 verwendet Standardargumente und unterstützt benannte Argumente. Gibt eine Geometrie als SVG-Pfad aus.
- ST_EndPoint - Änderung: 2.0.0 unterstützt die Verarbeitung von MultiLinestring's die nur aus einer einzelnen Geometrie bestehen, nicht mehr. In früheren Versionen von PostGIS gab die Funktion bei einem aus einer einzelnen Linie bestehender MultiLinestring den Anfangspunkt zurück. Ab 2.0.0 gibt sie nur NULL zurück, so wie bei jedem anderen MultiLinestring. Die alte Verhaltensweise war undokumentiert, aber Anwender, die annahmen, dass Sie Ihre Daten als LINESTRING vorliegen haben, könnten in 2.0 dieses zurückgegebene NULL bemerken. Gibt die Anzahl der Stützpunkte eines ST_LineString oder eines ST_CircularString zurück.
- ST_GeomFromText - Änderung: 2.0.0 - In Vorgängerversionen von PostGIS war ST_GeomFromText('GEOMETRYCOLLECTION(E erlaubt. Um eine bessere Übereinstimmung mit der SQL/MM Norm zu erreichen, ist dies in PostGIS 2.0.0 nun nicht mehr gestattet. Hier sollte nun ST_GeomFromText('GEOMETRYCOLLECTION EMPTY') geschrieben werden. Gibt einen spezifizierten ST_Geometry Wert aus einer Well-known-Text Darstellung (WKT) zurück.
- ST_GeometryN - Änderung: 2.0.0 Vorangegangene Versionen geben bei Einzelgeometrien NULL zurück. Dies wurde geändert um die Geometrie für den ST_GeometrieN(..,1) Fall zurückzugeben. Gibt den Geometrietyp des ST_Geometry Wertes zurück.
- ST_IsEmpty - Änderung: 2.0.0 - In Vorgängerversionen von PostGIS war ST_GeomFromText('GEOMETRYCOLLECTION(EMPTY erlaubt. Um eine bessere Übereinstimmung mit der SQL/MM Norm zu erreichen, ist dies nun nicht mehr gestattet. Tests if a geometry is empty.
- ST_Length - Änderung: 2.0.0 Wesentliche Änderung -- In früheren Versionen ergab die Anwendung auf ein MULTI/POLYGON vom geographischen Datentyp den Umfang des POLYGON/MULTIPOLYGON. In 2.0.0 wurde dies geändert und es wird jetzt 0 zurückgegeben, damit es mit der Verhaltensweise beim geometrischen Datentyp übereinstimmt. Verwenden Sie bitte ST_Perimeter, wenn Sie den Umfang eines Polygons wissen wollen Gibt den geometrischen Schwerpunkt einer Geometrie zurück.
- ST_LocateAlong - Änderung: 2.0.0 In Vorgängerversionen als ST_Locate_Along_Measure bezeichet. Der alte Name ist überholt und wird in der Zukunft entfernt ist aber noch verfügbar. Returns the point(s) on a geometry that match a measure value.
- ST_LocateBetween - Änderung: 2.0.0 In Vorgängerversionen als ST_Locate_Along_Measure bezeichet. Der alte Name ist überholt und wird in der Zukunft entfernt ist aber noch verfügbar. Returns the portions of a geometry that match a measure range.
- ST_NumGeometries - Änderung: 2.0.0 Bei früheren Versionen wurde NULL zurückgegeben, wenn die Geometrie nicht vom Typ GEOMETRYCOLLECTION/MULTI war. 2.0.0+ gibt nun 1 für Einzelgeometrien, wie POLYGON, LINESTRING, POINT zurück. Gibt die Anzahl der Punkte einer Geometrie zurück. Funktioniert für alle Geometrien.
- ST_NumInteriorRings - Änderung: 2.0.0 - In füheren Versionen war ein MULTIPOLYGON als Eingabe erlaubt, wobei die Anzahl der inneren Ringe des ersten Polygons ausgegeben wurde. Gibt die Anzahl der inneren Ringe einer Polygongeometrie aus.
- ST_PointN - Änderung: 2.0.0 arbeitet nicht mehr mit MultiLinestring's, die nur eine einzelne Geometrie enthalten. In früheren Versionen von PostGIS gab die Funktion bei einem, aus einer einzelnen Linie bestehender MultiLinestring, den Anfangspunkt zurück. Ab 2.0.0 wird, so wie bei jedem anderen MultiLinestring auch, NULL zurückgegeben. Gibt die Anzahl der Stützpunkte eines ST_LineString oder eines ST_CircularString zurück.
- ST_StartPoint - Änderung: 2.0.0 unterstützt die Verarbeitung von MultiLinestring's die nur aus einer einzelnen Geometrie bestehen, nicht mehr. In früheren Versionen von PostGIS gab die Funktion bei einem aus einer einzelnen Linie bestehender MultiLinestring den Anfangspunkt zurück. Ab 2.0.0 gibt sie nur NULL zurück, so wie bei jedem anderen MultiLinestring. Die alte Verhaltensweise war undokumentiert, aber Anwender, die annahmen, dass Sie Ihre Daten als LINESTRING vorliegen haben, könnten in 2.0 dieses zurückgegebene NULL bemerken. Returns the first point of a LineString.

12.12.12 PostGIS Functions new or enhanced in 1.5

The functions given below are PostGIS functions that were added or enhanced.

Functions new in PostGIS 1.5

- && - Verfügbarkeit: Mit 1.5.0 wurde die Unterstützung von geograpischen Koordinaten eingeführt Gibt TRUE zurück, wenn die 2D Bounding Box von A die 2D Bounding Box von B schneidet.
- PostGIS_LibXML_Version - Availability: 1.5 Returns the version number of the libxml2 library.
- ST_AddMeasure - Verfügbarkeit: 1.5.0 Interpolates measures along a linear geometry.
- ST_AsBinary - Verfügbarkeit: 1.5.0 Unterstützung von geograpischen Koordinaten. Return the OGC/ISO Well-Known Binary (WKB) representation of the geometry/geography without SRID meta data.
- ST_AsGML - Verfügbarkeit: 1.5.0 Unterstützung von geograpischen Koordinaten. Gibt die Geometrie als GML-Element - Version 2 oder 3 - zurück.
- ST_AsGeoJSON - Verfügbarkeit: 1.5.0 Unterstützung von geograpischen Koordinaten. Return a geometry as a GeoJSON element.
- ST_AsText - Verfügbarkeit: 1.5 - Unterstützung von geograpischen Koordinaten. Gibt die Well-known-Text(WKT) Darstellung der Geometrie/Geographie ohne die SRID Metadaten zurück.
- ST_Buffer - Verfügbarkeit: 1.5 - ST_Buffer wurde um die Unterstützung von Abschlusstücken/endcaps und Join-Typen erweitert. Diese können zum Beispiel dazu verwendet werden, um Linienzüge von Straßen in Straßenpolygone mit flachen oder rechtwinkeligen Abschlüssen anstatt mit runden Enden umzuwandeln. Ein schlanker Adapter für den geographischen Datentyp wurde hinzugefügt. Computes a geometry covering all points within a given distance from a geometry.
- ST_ClosestPoint - Verfügbarkeit: 1.5.0 Returns the 2D point on g1 that is closest to g2. This is the first point of the shortest line from one geometry to the other.
- ST_CollectionExtract - Verfügbarkeit: 1.5.0 Given a geometry collection, returns a multi-geometry containing only elements of a specified type.
- ST_Covers - Availability: 1.5 - support for geography was introduced. Tests if every point of B lies in A
- ST_DFullyWithin - Availability: 1.5.0 Tests if two geometries are entirely within a given distance
- ST_DWithin - Availability: 1.5.0 support for geography was introduced Tests if two geometries are within a given distance
- ST_Distance - Verfügbarkeit: 1.5.0 die Unterstützung des geograpischen Datentyps wurde eingeführt. Geschwindigkeitsverbesserungen bei einer umfangreichen Geometrie und bei einer Geometrie mit vielen Knoten Gibt die größte 3-dimensionale Distanz zwischen zwei geometrischen Objekten als Linie zurück
- ST_DistanceSphere - Verfügbarkeit: 1.5 die Unterstützung für weitere geometrische Datentypen neben Punkten eingeführt. Bei Vorgängerversionen wurden nur Punkte unterstützt. Gibt die kürzeste Distanz zwischen zwei geometrischen Objekten zurück, die über Länge, Breite und ein bestimmtes Referenzellipsoid gegeben sind. Vorgängerversionen von PostGIS 1.5 unterstützten nur Punkte.
- ST_DistanceSpheroid - Verfügbarkeit: 1.5 die Unterstützung für weitere geometrische Datentypen neben Punkten eingeführt. Bei Vorgängerversionen wurden nur Punkte unterstützt. Gibt die kürzeste Distanz zwischen zwei geometrischen Objekten zurück, die über Länge, Breite und ein bestimmtes Referenzellipsoid gegeben sind. Vorgängerversionen von PostGIS 1.5 unterstützten nur Punkte.
- ST_DumpPoints - Verfügbarkeit: 1.2.2 Gibt eine Zusammenfassung des Inhalts einer Geometrie wieder.
- ST_Envelope - Verfügbarkeit: 1.5.0 Änderung der Verhaltensweise insofern, das die Ausgabe in Double Precision anstelle von Float4 erfolgt Gibt eine Geometrie in doppelter Genauigkeit (float8) zurück, welche das Umgebungsrechteck der beigestellten Geometrie darstellt.
- ST_Expand - Availability: 1.5.0 behavior changed to output double precision instead of float4 coordinates. Returns a bounding box expanded from another bounding box or a geometry.

- ST_GMLToSQL - Verfügbarkeit: 1.5, benötigt libxml2 1.6+ Gibt einen spezifizierten ST_Geometry Wert aus einer GML-Darstellung zurück. Dies ist ein Aliasname für ST_GeomFromGML
- ST_GeomFromGML - Verfügbarkeit: 1.5, benötigt libxml2 1.6+ Nimmt als Eingabe eine GML-Darstellung der Geometrie und gibt ein geometrisches PostGIS-Objekt aus.
- ST_GeomFromKML - Verfügbarkeit: 1.5, benötigt libxml2 2.6+ Nimmt als Eingabe eine KML-Darstellung der Geometrie und gibt ein geometrisches PostGIS-Objekt aus.
- ST_HausdorffDistance - Verfügbarkeit: 1.5.0 Gibt den kürzesten 3-dimensionalen Abstand zwischen zwei geometrischen Objekten als Linie zurück
- ST_Intersection - Availability: 1.5 support for geography data type was introduced. Computes a geometry representing the shared portion of geometries A and B.
- ST_Intersects - Availability: 1.5 support for geography was introduced. Tests if two geometries intersect (they have at least one point in common)
- ST_Length - Verfügbarkeit: 1.5.0 Unterstützung von geograpischen Koordinaten. Gibt den geometrischen Schwerpunkt einer Geometrie zurück.
- ST_LongestLine - Verfügbarkeit: 1.5.0 Gibt die größte 3-dimensionale Distanz zwischen zwei geometrischen Objekten als Linie zurück
- ST_MakeEnvelope - Verfügbarkeit: 1.5 Erzeugt ein rechteckiges Polygon aus den gegebenen Minimum- und Maximumwerten. Die Eingabewerte müssen in dem Koordinatenreferenzsystem sein, welches durch die SRID angegeben wird.
- ST_MaxDistance - Verfügbarkeit: 1.5.0 Gibt die größte 2-dimensionale Distanz zwischen zwei geometrischen Objekten in projizierten Einheiten zurück.
- ST_ShortestLine - Verfügbarkeit: 1.5.0 Gibt die 2-dimenionale kürzeste Strecke zwischen zwei Geometrien als Linie zurück
- ~= - Verfügbarkeit: 1.5.0 "Verhaltensänderung" Gibt TRUE zurück, wenn die bounding box von A ident mit jener von B ist.

12.12.13 PostGIS Functions new or enhanced in 1.4

The functions given below are PostGIS functions that were added or enhanced.

Functions new in PostGIS 1.4

- Populate_Geometry_Columns - Verfügbarkeit: 1.4.0 Ensures geometry columns are defined with type modifiers or have appropriate spatial constraints.
- ST_Collect - Verfügbarkeit: 1.4.0 - ST_MakeLine(geomarray) wurde eingeführt. ST_MakeLine Aggregatfunktion wurde verbessert, um mehr Punkte schneller handhaben zu können. Creates a GeometryCollection or Multi* geometry from a set of geometries.
- ST_ContainsProperly - Availability: 1.4.0 Tests if every point of B lies in the interior of A
- ST_GeoHash - Verfügbarkeit: 1.4.0 Gibt die Geometrie in der GeoHash Darstellung aus.
- ST_IsValidReason - Availability: 1.4 Returns text stating if a geometry is valid, or a reason for invalidity.
- ST_LineCrossingDirection - Availability: 1.4 Returns a number indicating the crossing behavior of two LineStrings
- ST_LocateBetweenElevations - Verfügbarkeit: 1.4.0 Returns the portions of a geometry that lie in an elevation (Z) range.
- ST_MakeLine - Verfügbarkeit: 1.4.0 - ST_MakeLine(geomarray) wurde eingeführt. ST_MakeLine Aggregatfunktion wurde verbessert, um mehr Punkte schneller handhaben zu können. Erzeugt einen Linienzug aus einer Punkt-, Mehrfachpunkt- oder Liniengeometrie.
- ST_MinimumBoundingCircle - Verfügbarkeit: 1.4.0 Returns the smallest circle polygon that contains a geometry.
- ST_Union - Availability: 1.4.0 - ST_Union was enhanced. ST_Union(geomarray) was introduced and also faster aggregate collection in PostgreSQL. Computes a geometry representing the point-set union of the input geometries.

12.12.14 PostGIS Functions new or enhanced in 1.3

The functions given below are PostGIS functions that were added or enhanced.

Functions new in PostGIS 1.3

- ST_AsGML - Verfügbarkeit: 1.3.2 Gibt die Geometrie als GML-Element - Version 2 oder 3 - zurück.
- ST_AsGeoJSON - Verfügbarkeit: 1.3.4 Return a geometry as a GeoJSON element.
- ST_CurveToLine - Verfügbarkeit: 1.3.0 Converts a geometry containing curves to a linear geometry.
- ST_LineToCurve - Verfügbarkeit: 1.3.0 Converts a linear geometry to a curved geometry.
- ST_SimplifyPreserveTopology - Verfügbarkeit: 1.3.3 Returns a simplified and valid version of a geometry, using the Douglas-Peucker algorithm.

Chapter 13

Meldung von Problemen

13.1 Software Bugs melden

Effektive Fehlerberichte sind ein wesentlicher Beitrag zur Weiterentwicklung von PostGIS. Am wirksamsten ist ein Fehlerbericht dann, wenn er von den PostGIS-Entwicklern reproduziert werden kann. Idealerweise enthält er ein Skript das den Fehler auslöst und eine vollständige Beschreibung der Umgebung in der er aufgetreten ist. Ausreichend gute Information liefert `SELECT postgis_full_version()` [für PostGIS] und `SELECT version()` [für PostgreSQL].

Falls Sie nicht die aktuelle Version verwenden, sollten Sie zuerst unter release changelog nachsehen, ob Ihr Bug nicht bereits bereinigt wurde.

Die Verwendung des PostGIS bug tracker stellt sicher dass Ihre Berichte nicht verworfen werden, und dass Sie über die Prozessabwicklung am Laufenden gehalten werden. Bevor Sie einen neuen Fehler melden fragen Sie bitte die Datenbank ab ob der Fehler schon bekannt ist. Wenn es ein bekannter Fehler ist, so fügen Sie bitte jegliche neue Information die Sie herausgefunden haben hinzu.

Vielleicht möchten Sie zuvor Simon Tatham's Artikel über How to Report Bugs Effectively lesen, bevor Sie einen Fehlerbericht senden.

13.2 Probleme mit der Dokumentation melden

Die Dokumentation sollte die Eigenschaften und das Verhalten der Software exakt wiederspiegeln. Wenn das nicht der Fall ist, so kann entweder ein Softwarebug oder eine fehlerhafte bzw. unzulängliche Dokumentation daran Schuld sein.

Probleme in der Dokumentation können unter PostGIS bug tracker gemeldet werden.

Wenn die Überarbeitung trivial ist, können Sie diese in einem neuen Bug Tracker Issue beschreiben. Geben Sie bitte die exakte Stelle in der Dokumentation an.

Wenn es sich um umfangreichere Änderungen handelt, ist ein Subversion Patch zweifellos die bessere Wahl. Dabei handelt es sich um einen vierstufigen Vorgang unter Unix (angenommen, Sie haben Subversion bereits installiert):

1. Eine Kopie des PostGIS Subversion Trunks auschecken. Eingabe unter Unix:

   git clone https://git.osgeo.org/gitea/postgis/postgis.git

   Dies wird im Verzeichnis ./trunk gespeichert

2. Erledigen Sie die Änderungen an der Dokumentation mit Ihrem Lieblingstexteditor. Auf Unix, tippen Sie (zum Beispiel):

   vim trunk/doc/postgis.xml

   Bedenken Sie bitte, dass die Dokumentation in DocBook XML und nicht in HTML geschrieben ist. Falls Sie damit nicht vertraut sind, so folgen Sie bitte dem Beispiel in der restlichen Dokumentation.

3. Erzeugung einer Patchdatei, welche die Unterschiede zur Master-Kopie der Dokumentation enthält. Unter Unix tippen Sie bitte:

   svn diff trunk/doc/postgis.xml > doc.patch

4. Fügen Sie den Patch einem neuen Thema/Issue im Bug Tracker bei.

Appendix A

Anhang

A.1 PostGIS 3.4.0

2023/08/13

This version requires PostgreSQL 12 or higher, GEOS 3.6 or higher, and Proj 6.1+. To take advantage of all features, GEOS 3.12+ is needed. To take advantage of all SFCGAL featurs, SFCGAL 1.4.1+ is needed.

NOTE: GEOS 3.12.0 details at GEOS 3.12.0 release notes

Many thanks to our translation teams, in particular:

Teramoto Ikuhiro (Japanese Team)

Vincent Bre (French Team)

A.1.1 New features

5055, complete manual internationalization (Sandro Santilli)

5052, target version support in postgis_extensions_upgrade (Sandro Santilli)

5306, expose version of GEOS at compile time (Sandro Santilli)

New install-extension-upgrades command in postgis script (Sandro Santilli)

5257, 5261, 5277, Support changes for PostgreSQL 16 (Regina Obe)

5006, 705, ST_Transform: Support PROJ pipelines (Robert Coup, Koordinates)

5283, [postgis_topology] RenameTopology (Sandro Santilli)

5286, [postgis_topology] RenameTopoGeometryColumn (Sandro Santilli)

703, [postgis_raster] Add min/max resampling as options (Christian Schroeder)

5336, [postgis_topology] topogeometry cast to topoelement support (Regina Obe)

Allow singleton geometry to be inserted into Geometry(Multi*) columns (Paul Ramsey)

721, New window-based ST_ClusterWithinWin and ST_ClusterIntersectingWin (Paul Ramsey)

5397, [address_standardizer] debug_standardize_address function (Regina Obe)

5373ST_LargestEmptyCircle, exposes extra semantics on circle finding. Geos 3.9+ required(Martin Davis)

5267, ST_Project signature for geometry, and two-point signature (Paul Ramsey)

5267, ST_LineExtend for extending linestrings (Paul Ramsey)

New coverage functions ST_CoverageInvalidEdges, ST_CoverageSimplify, ST_CoverageUnion (Paul Ramsey)

A.1.2 Enhancements

5194, do not update system catalogs from postgis_extensions_upgrade (Sandro Santilli)

5092, reduce number of upgrade paths installed on system (Sandro Santilli)

635, honour --bindir (and --prefix) configure switch for executables (Sandro Santilli)

Honour --mandir (and --prefix) configure switch for man pages install path (Sandro Santilli)

Honour --htmldir (and --docdir and --prefix) configure switch for html pages install path (Sandro Santilli)

5447 Manual pages added for postgis and postgis_restore utilities (Sandro Santilli)

[postgis_topology] Speed up check of topology faces without edges (Sandro Santilli)

[postgis_topology] Speed up coincident nodes check in topology validation (Sandro Santilli)

718, ST_QuantizeCoordinates(): speed-up implementation (Even Rouault)

Repair spatial planner stats to use computed selectivity for contains/within queries (Paul Ramsey)

734, Additional metadata on Proj installation in postgis_proj_version (Paul Ramsey)

5177, Allow building tools without PostgreSQL server headers. Respect prefix/bin for tools install (Sandro Santilli)

ST_Project signature for geometry, and two-point signature (Paul Ramsey)

4913, ST_AsSVG support for curve types CircularString, CompoundCurve, MultiCurve, and MultiSurface (Regina Obe)

5266, ST_ClosestPoint, ST_ShortestLine, ST_LineSubString support for geography type (MobilityDB Esteban Zimanyi, Maxime Schoemans, Paul Ramsey)

A.1.3 Breaking Changes

5229, Drop support for Proj < 6.1 and PG 11 (Regina Obe)

5306, 734, postgis_full_version() and postgis_proj_version() now output more information about proj network configuration and data paths. GEOS compile-time version also shown if different from run-time (Paul Ramsey, Sandro Santilli)

5447, postgis_restore.pl renamed to postgis_restore (Sandro Santilli)

Utilities now installed in OS bin or user specified --bindir and --prefix instead of postgresql bin and extension stripped except on windows (postgis, postgis_restore, shp2pgsql, raster2pgsql, pgsql2shp, pgtopo_import, pgtopo_export)