PostGIS 3.5.3 Manual

Contents

1 Introduction **1**

1.1 Project Steering Committee . . . 1
1.2 Core Contributors Present . . . 1
1.3 Core Contributors Past . . . 2
1.4 Other Contributors . . . 2

2 PostGIS Installation **6**

2.1 Short Version . . . 6
2.2 Compiling and Install from Source . . . 6
2.2.1 Getting the Source . . . 7
2.2.2 Install Requirements . . . 7
2.2.3 Build configuration . . . 8
2.2.4 Building . . . 10
2.2.5 Building PostGIS Extensions and Deploying them . . . 10
2.2.6 Testing . . . 12
2.2.7 Installation . . . 15
2.3 Installing and Using the address standardizer . . . 15
2.4 Installing, Upgrading Tiger Geocoder, and loading data . . . 16
2.4.1 Tiger Geocoder Enabling your PostGIS database . . . 16
2.4.2 Using Address Standardizer Extension with Tiger geocoder . . . 19
2.4.3 Required tools for tiger data loading . . . 19
2.4.4 Upgrading your Tiger Geocoder Install and Data . . . 19
2.5 Common Problems during installation . . . 20

3 PostGIS Administration **21**

3.1 Performance Tuning . . . 21
3.1.1 Startup . . . 21
3.1.2 Runtime . . . 22
3.2 Configuring raster support . . . 22
3.3 Creating spatial databases . . . 23
3.3.1 Spatially enable database using EXTENSION . . . 23

3.3.2 Spatially enable database without using EXTENSION (discouraged) . . . 23
3.4 Upgrading spatial databases . . . 24
3.4.1 Soft upgrade . . . 24
3.4.1.1 Soft Upgrade 9.1+ using extensions . . . 24
3.4.1.2 Soft Upgrade Pre 9.1+ or without extensions . . . 25
3.4.2 Hard upgrade . . . 26

4 Data Management **28**
4.1 Spatial Data Model . . . 28
4.1.1 OGC Geometry . . . 28
4.1.1.1 Point . . . 29
4.1.1.2 LineString . . . 29
4.1.1.3 LinearRing . . . 29
4.1.1.4 Polygon . . . 29
4.1.1.5 MultiPoint . . . 29
4.1.1.6 MultiLineString . . . 29
4.1.1.7 MultiPolygon . . . 30
4.1.1.8 GeometryCollection . . . 30
4.1.1.9 PolyhedralSurface . . . 30
4.1.1.10 Triangle . . . 30
4.1.1.11 TIN . . . 30
4.1.2 SQL/MM Part 3 - Curves . . . 30
4.1.2.1 CircularString . . . 31
4.1.2.2 CompoundCurve . . . 31
4.1.2.3 CurvePolygon . . . 31
4.1.2.4 MultiCurve . . . 31
4.1.2.5 MultiSurface . . . 31
4.1.3 WKT and WKB . . . 32
4.2 Geometry Data Type . . . 33
4.2.1 PostGIS EWKB and EWKT . . . 33
4.3 Geography Data Type . . . 35
4.3.1 Creating Geography Tables . . . 35
4.3.2 Using Geography Tables . . . 36
4.3.3 When to use the Geography data type . . . 37
4.3.4 Geography Advanced FAQ . . . 37
4.4 Geometry Validation . . . 38
4.4.1 Simple Geometry . . . 38
4.4.2 Valid Geometry . . . 40
4.4.3 Managing Validity . . . 42

4.5 Spatial Reference Systems . . . 43
4.5.1 SPATIAL_REF_SYS Table . . . 44
4.5.2 User-Defined Spatial Reference Systems . . . 45
4.6 Spatial Tables . . . 45
4.6.1 Creating a Spatial Table . . . 45
4.6.2 GEOMETRY_COLUMNS View . . . 46
4.6.3 Manually Registering Geometry Columns . . . 47
4.7 Loading Spatial Data . . . 49
4.7.1 Using SQL to Load Data . . . 49
4.7.2 Using the Shapefile Loader . . . 49
4.8 Extracting Spatial Data . . . 51
4.8.1 Using SQL to Extract Data . . . 51
4.8.2 Using the Shapefile Dumper . . . 52
4.9 Spatial Indexes . . . 52
4.9.1 GiST Indexes . . . 53
4.9.2 BRIN Indexes . . . 53
4.9.3 SP-GiST Indexes . . . 55
4.9.4 Tuning Index Usage . . . 56

5 Spatial Queries **57**
5.1 Determining Spatial Relationships . . . 57
5.1.1 Dimensionally Extended 9-Intersection Model . . . 57
5.1.2 Named Spatial Relationships . . . 59
5.1.3 General Spatial Relationships . . . 60
5.2 Using Spatial Indexes . . . 62
5.3 Examples of Spatial SQL . . . 62

6 Performance Tips **65**
6.1 Small tables of large geometries . . . 65
6.1.1 Problem description . . . 65
6.1.2 Workarounds . . . 65
6.2 CLUSTERing on geometry indices . . . 66
6.3 Avoiding dimension conversion . . . 66

7 PostGIS Reference **67**
7.1 PostGIS Geometry/Geography/Box Data Types . . . 67
7.1.1 box2d . . . 67
7.1.2 box3d . . . 68
7.1.3 geometry . . . 68
7.1.4 geometry_dump . . . 69

7.1.5 geography . . . 69
7.2 Table Management Functions . . . 69
7.2.1 AddGeometryColumn . . . 69
7.2.2 DropGeometryColumn . . . 71
7.2.3 DropGeometryTable . . . 72
7.2.4 Find_SRID . . . 73
7.2.5 Populate_Geometry_Columns . . . 73
7.2.6 UpdateGeometrySRID . . . 75
7.3 Geometry Constructors . . . 76
7.3.1 ST_Collect . . . 76
7.3.2 ST_LineFromMultiPoint . . . 78
7.3.3 ST_MakeEnvelope . . . 78
7.3.4 ST_MakeLine . . . 79
7.3.5 ST_MakePoint . . . 80
7.3.6 ST_MakePointM . . . 81
7.3.7 ST_MakePolygon . . . 82
7.3.8 ST_Point . . . 84
7.3.9 ST_PointZ . . . 85
7.3.10 ST_PointM . . . 86
7.3.11 ST_PointZM . . . 87
7.3.12 ST_Polygon . . . 87
7.3.13 ST_TileEnvelope . . . 88
7.3.14 ST_HexagonGrid . . . 89
7.3.15 ST_Hexagon . . . 92
7.3.16 ST_SquareGrid . . . 93
7.3.17 ST_Square . . . 94
7.3.18 ST_Letters . . . 95
7.4 Geometry Accessors . . . 96
7.4.1 GeometryType . . . 96
7.4.2 ST_Boundary . . . 97
7.4.3 ST_BoundingDiagonal . . . 99
7.4.4 ST_CoordDim . . . 100
7.4.5 ST_Dimension . . . 101
7.4.6 ST_Dump . . . 101
7.4.7 ST_DumpPoints . . . 103
7.4.8 ST_DumpSegments . . . 107
7.4.9 ST_DumpRings . . . 109
7.4.10 ST_EndPoint . . . 110
7.4.11 ST_Envelope . . . 111

7.4.12 ST_ExteriorRing . . . 113
7.4.13 ST_GeometryN . . . 114
7.4.14 ST_GeometryType . . . 116
7.4.15 ST_HasArc . . . 117
7.4.16 ST_InteriorRingN . . . 117
7.4.17 ST_NumCurves . . . 118
7.4.18 ST_CurveN . . . 119
7.4.19 ST_IsClosed . . . 119
7.4.20 ST_IsCollection . . . 121
7.4.21 ST_IsEmpty . . . 122
7.4.22 ST_IsPolygonCCW . . . 123
7.4.23 ST_IsPolygonCW . . . 124
7.4.24 ST_IsRing . . . 125
7.4.25 ST_IsSimple . . . 126
7.4.26 ST_M . . . 126
7.4.27 ST_MemSize . . . 127
7.4.28 ST_NDims . . . 128
7.4.29 ST_NPoints . . . 129
7.4.30 ST_NRings . . . 130
7.4.31 ST_NumGeometries . . . 130
7.4.32 ST_NumInteriorRings . . . 131
7.4.33 ST_NumInteriorRing . . . 132
7.4.34 ST_NumPatches . . . 132
7.4.35 ST_NumPoints . . . 133
7.4.36 ST_PatchN . . . 133
7.4.37 ST_PointN . . . 134
7.4.38 ST_Points . . . 136
7.4.39 ST_StartPoint . . . 136
7.4.40 ST_Summary . . . 137
7.4.41 ST_X . . . 139
7.4.42 ST_Y . . . 139
7.4.43 ST_Z . . . 140
7.4.44 ST_Zmflag . . . 141
7.4.45 ST_HasZ . . . 142
7.4.46 ST_HasM . . . 143
7.5 Geometry Editors . . . 143
7.5.1 ST_AddPoint . . . 143
7.5.2 ST_CollectionExtract . . . 144
7.5.3 ST_CollectionHomogenize . . . 145

7.5.4 ST_CurveToLine . . . 147
7.5.5 ST_Scroll . . . 149
7.5.6 ST_FlipCoordinates . . . 150
7.5.7 ST_Force2D . . . 151
7.5.8 ST_Force3D . . . 151
7.5.9 ST_Force3DZ . . . 152
7.5.10 ST_Force3DM . . . 153
7.5.11 ST_Force4D . . . 154
7.5.12 ST_ForceCollection . . . 154
7.5.13 ST_ForceCurve . . . 156
7.5.14 ST_ForcePolygonCCW . . . 156
7.5.15 ST_ForcePolygonCW . . . 157
7.5.16 ST_ForceSFS . . . 157
7.5.17 ST_ForceRHR . . . 157
7.5.18 ST_LineExtend . . . 158
7.5.19 ST_LineToCurve . . . 159
7.5.20 ST_Multi . . . 160
7.5.21 ST_Normalize . . . 161
7.5.22 ST_Project . . . 161
7.5.23 ST_QuantizeCoordinates . . . 162
7.5.24 ST_RemovePoint . . . 164
7.5.25 ST_RemoveRepeatedPoints . . . 165
7.5.26 ST_RemoveIrrelevantPointsForView . . . 166
7.5.27 ST_RemoveSmallParts . . . 168
7.5.28 ST_Reverse . . . 170
7.5.29 ST_Segmentize . . . 170
7.5.30 ST_SetPoint . . . 172
7.5.31 ST_ShiftLongitude . . . 173
7.5.32 ST_WrapX . . . 174
7.5.33 ST_SnapToGrid . . . 175
7.5.34 ST_Snap . . . 176
7.5.35 ST_SwapOrdinates . . . 179
7.6 Geometry Validation . . . 180
7.6.1 ST_IsValid . . . 180
7.6.2 ST_IsValidDetail . . . 181
7.6.3 ST_IsValidReason . . . 183
7.6.4 ST_MakeValid . . . 184
7.7 Spatial Reference System Functions . . . 189
7.7.1 ST_InverseTransformPipeline . . . 189

7.7.2 ST_SetSRID . . . 190
7.7.3 ST_SRID . . . 191
7.7.4 ST_Transform . . . 192
7.7.5 ST_TransformPipeline . . . 194
7.7.6 postgis_srs_codes . . . 196
7.7.7 postgis_srs . . . 196
7.7.8 postgis_srs_all . . . 197
7.7.9 postgis_srs_search . . . 198
7.8 Geometry Input . . . 199
7.8.1 Well-Known Text (WKT) . . . 199
7.8.1.1 ST_BdPolyFromText . . . 199
7.8.1.2 ST_BdMPolyFromText . . . 199
7.8.1.3 ST_GeogFromText . . . 200
7.8.1.4 ST_GeographyFromText . . . 200
7.8.1.5 ST_GeomCollFromText . . . 201
7.8.1.6 ST_GeomFromEWKT . . . 201
7.8.1.7 ST_GeomFromMARC21 . . . 203
7.8.1.8 ST_GeometryFromText . . . 205
7.8.1.9 ST_GeomFromText . . . 205
7.8.1.10 ST_LineFromText . . . 207
7.8.1.11 ST_MLineFromText . . . 208
7.8.1.12 ST_MPointFromText . . . 208
7.8.1.13 ST_MPolyFromText . . . 209
7.8.1.14 ST_PointFromText . . . 210
7.8.1.15 ST_PolygonFromText . . . 211
7.8.1.16 ST_WKTToSQL . . . 212
7.8.2 Well-Known Binary (WKB) . . . 212
7.8.2.1 ST_GeogFromWKB . . . 212
7.8.2.2 ST_GeomFromEWKB . . . 213
7.8.2.3 ST_GeomFromWKB . . . 214
7.8.2.4 ST_LineFromWKB . . . 215
7.8.2.5 ST_LinestringFromWKB . . . 216
7.8.2.6 ST_PointFromWKB . . . 216
7.8.2.7 ST_WKBToSQL . . . 217
7.8.3 Other Formats . . . 218
7.8.3.1 ST_Box2dFromGeoHash . . . 218
7.8.3.2 ST_GeomFromGeoHash . . . 218
7.8.3.3 ST_GeomFromGML . . . 219
7.8.3.4 ST_GeomFromGeoJSON . . . 222

7.8.3.5 ST_GeomFromKML . . . 223
7.8.3.6 ST_GeomFromTWKB . . . 223
7.8.3.7 ST_GMLToSQL . . . 224
7.8.3.8 ST_LineFromEncodedPolyline . . . 224
7.8.3.9 ST_PointFromGeoHash . . . 225
7.8.3.10 ST_FromFlatGeobufToTable . . . 226
7.8.3.11 ST_FromFlatGeobuf . . . 226
7.9 Geometry Output . . . 227
7.9.1 Well-Known Text (WKT) . . . 227
7.9.1.1 ST_AsEWKT . . . 227
7.9.1.2 ST_AsText . . . 228
7.9.2 Well-Known Binary (WKB) . . . 229
7.9.2.1 ST_AsBinary . . . 229
7.9.2.2 ST_AsEWKB . . . 231
7.9.2.3 ST_AsHEXEWKB . . . 232
7.9.3 Other Formats . . . 233
7.9.3.1 ST_AsEncodedPolyline . . . 233
7.9.3.2 ST_AsFlatGeobuf . . . 234
7.9.3.3 ST_AsGeobuf . . . 234
7.9.3.4 ST_AsGeoJSON . . . 234
7.9.3.5 ST_AsGML . . . 237
7.9.3.6 ST_AsKML . . . 240
7.9.3.7 ST_AsLatLonText . . . 241
7.9.3.8 ST_AsMARC21 . . . 242
7.9.3.9 ST_AsMVTGeom . . . 245
7.9.3.10 ST_AsMVT . . . 246
7.9.3.11 ST_AsSVG . . . 247
7.9.3.12 ST_AsTWKB . . . 248
7.9.3.13 ST_AsX3D . . . 249
7.9.3.14 ST_GeoHash . . . 252
7.10 Operators . . . 254
7.10.1 Bounding Box Operators . . . 254
7.10.1.1 && . . . 254
7.10.1.2 &&(geometry,box2df) . . . 254
7.10.1.3 &&(box2df,geometry) . . . 255
7.10.1.4 &&(box2df,box2df) . . . 256
7.10.1.5 &&& . . . 257
7.10.1.6 &&&(geometry,gidx) . . . 258
7.10.1.7 &&&(gidx,geometry) . . . 259

7.10.1.8 &&&(gidx,gidx) . . . 260
7.10.1.9 &< . . . 260
7.10.1.10 &<| . . . 261
7.10.1.11 &> . . . 262
7.10.1.12 << . . . 263
7.10.1.13 <<| . . . 263
7.10.1.14 = . . . 264
7.10.1.15 >> . . . 265
7.10.1.16 @ . . . 266
7.10.1.17 @(geometry,box2df) . . . 267
7.10.1.18 @(box2df,geometry) . . . 268
7.10.1.19 @(box2df,box2df) . . . 268
7.10.1.20 |&> . . . 269
7.10.1.21 |>> . . . 270
7.10.1.22 ~ . . . 271
7.10.1.23 ~(geometry,box2df) . . . 271
7.10.1.24 ~(box2df,geometry) . . . 272
7.10.1.25 ~(box2df,box2df) . . . 273
7.10.1.26 ~= . . . 274
7.10.2 Distance Operators . . . 274
7.10.2.1 <-> . . . 274
7.10.2.2 |=| . . . 276
7.10.2.3 <#> . . . 277
7.10.2.4 <<->> . . . 278
7.11 Spatial Relationships . . . 279
7.11.1 Topological Relationships . . . 279
7.11.1.1 ST_3DIntersects . . . 279
7.11.1.2 ST_Contains . . . 280
7.11.1.3 ST_ContainsProperly . . . 284
7.11.1.4 ST_CoveredBy . . . 285
7.11.1.5 ST_Covers . . . 286
7.11.1.6 ST_Crosses . . . 288
7.11.1.7 ST_Disjoint . . . 290
7.11.1.8 ST_Equals . . . 291
7.11.1.9 ST_Intersects . . . 292
7.11.1.10 ST_LineCrossingDirection . . . 294
7.11.1.11 ST_OrderingEquals . . . 296
7.11.1.12 ST_Overlaps . . . 297
7.11.1.13 ST_Relate . . . 300

7.11.1.14 ST_RelateMatch . . . 303
7.11.1.15 ST_Touches . . . 304
7.11.1.16 ST_Within . . . 305
7.11.2 Distance Relationships . . . 307
7.11.2.1 ST_3DDWithin . . . 307
7.11.2.2 ST_3DDFullyWithin . . . 308
7.11.2.3 ST_DFullyWithin . . . 309
7.11.2.4 ST_DWithin . . . 310
7.11.2.5 ST_PointInsideCircle . . . 311
7.12 Measurement Functions . . . 312
7.12.1 ST_Area . . . 312
7.12.2 ST_Azimuth . . . 313
7.12.3 ST_Angle . . . 315
7.12.4 ST_ClosestPoint . . . 316
7.12.5 ST_3DClosestPoint . . . 317
7.12.6 ST_Distance . . . 318
7.12.7 ST_3DDistance . . . 320
7.12.8 ST_DistanceSphere . . . 321
7.12.9 ST_DistanceSpheroid . . . 322
7.12.10 ST_FrechetDistance . . . 323
7.12.11 ST_HausdorffDistance . . . 324
7.12.12 ST_Length . . . 325
7.12.13 ST_Length2D . . . 327
7.12.14 ST_3DLength . . . 327
7.12.15 ST_LengthSpheroid . . . 328
7.12.16 ST_LongestLine . . . 329
7.12.17 ST_3DLongestLine . . . 331
7.12.18 ST_MaxDistance . . . 332
7.12.19 ST_3DMaxDistance . . . 333
7.12.20 ST_MinimumClearance . . . 334
7.12.21 ST_MinimumClearanceLine . . . 335
7.12.22 ST_Perimeter . . . 335
7.12.23 ST_Perimeter2D . . . 337
7.12.24 ST_3DPerimeter . . . 337
7.12.25 ST_ShortestLine . . . 338
7.12.26 ST_3DShortestLine . . . 340
7.13 Overlay Functions . . . 341
7.13.1 ST_ClipByBox2D . . . 341
7.13.2 ST_Difference . . . 342

7.13.3 ST_Intersection . . . 343
7.13.4 ST_MemUnion . . . 346
7.13.5 ST_Node . . . 346
7.13.6 ST_Split . . . 347
7.13.7 ST_Subdivide . . . 350
7.13.8 ST_SymDifference . . . 352
7.13.9 ST_UnaryUnion . . . 353
7.13.10 ST_Union . . . 354
7.14 Geometry Processing . . . 356
7.14.1 ST_Buffer . . . 356
7.14.2 ST_BuildArea . . . 361
7.14.3 ST_Centroid . . . 363
7.14.4 ST_ChaikinSmoothing . . . 365
7.14.5 ST_ConcaveHull . . . 366
7.14.6 ST_ConvexHull . . . 370
7.14.7 ST_DelaunayTriangles . . . 371
7.14.8 ST_FilterByM . . . 376
7.14.9 ST_GeneratePoints . . . 377
7.14.10 ST_GeometricMedian . . . 378
7.14.11 ST_LineMerge . . . 379
7.14.12 ST_MaximumInscribedCircle . . . 382
7.14.13 ST_LargestEmptyCircle . . . 384
7.14.14 ST_MinimumBoundingCircle . . . 386
7.14.15 ST_MinimumBoundingRadius . . . 387
7.14.16 ST_OrientedEnvelope . . . 388
7.14.17 ST_OffsetCurve . . . 389
7.14.18 ST_PointOnSurface . . . 393
7.14.19 ST_Polygonize . . . 395
7.14.20 ST_ReducePrecision . . . 397
7.14.21 ST_SharedPaths . . . 398
7.14.22 ST_Simplify . . . 401
7.14.23 ST_SimplifyPreserveTopology . . . 402
7.14.24 ST_SimplifyPolygonHull . . . 404
7.14.25 ST_SimplifyVW . . . 407
7.14.26 ST_SetEffectiveArea . . . 408
7.14.27 ST_TriangulatePolygon . . . 410
7.14.28 ST_VoronoiLines . . . 412
7.14.29 ST_VoronoiPolygons . . . 413
7.15 Coverages . . . 414

7.15.1 ST_CoverageInvalidEdges . . . 414
7.15.2 ST_CoverageSimplify . . . 416
7.15.3 ST_CoverageUnion . . . 417
7.16 Affine Transformations . . . 419
7.16.1 ST_Affine . . . 419
7.16.2 ST_Rotate . . . 420
7.16.3 ST_RotateX . . . 421
7.16.4 ST_RotateY . . . 422
7.16.5 ST_RotateZ . . . 423
7.16.6 ST_Scale . . . 424
7.16.7 ST_Translate . . . 425
7.16.8 ST_TransScale . . . 427
7.17 Clustering Functions . . . 428
7.17.1 ST_ClusterDBSCAN . . . 428
7.17.2 ST_ClusterIntersecting . . . 430
7.17.3 ST_ClusterIntersectingWin . . . 430
7.17.4 ST_ClusterKMeans . . . 431
7.17.5 ST_ClusterWithin . . . 433
7.17.6 ST_ClusterWithinWin . . . 434
7.18 Bounding Box Functions . . . 435
7.18.1 Box2D . . . 435
7.18.2 Box3D . . . 436
7.18.3 ST_EstimatedExtent . . . 436
7.18.4 ST_Expand . . . 437
7.18.5 ST_Extent . . . 439
7.18.6 ST_3DExtent . . . 440
7.18.7 ST_MakeBox2D . . . 441
7.18.8 ST_3DMakeBox . . . 442
7.18.9 ST_XMax . . . 442
7.18.10 ST_XMin . . . 443
7.18.11 ST_YMax . . . 444
7.18.12 ST_YMin . . . 445
7.18.13 ST_ZMax . . . 446
7.18.14 ST_ZMin . . . 447
7.19 Linear Referencing . . . 448
7.19.1 ST_LineInterpolatePoint . . . 448
7.19.2 ST_3DLineInterpolatePoint . . . 450
7.19.3 ST_LineInterpolatePoints . . . 450
7.19.4 ST_LineLocatePoint . . . 451

7.19.5 ST_LineSubstring . . . 452
7.19.6 ST_LocateAlong . . . 454
7.19.7 ST_LocateBetween . . . 455
7.19.8 ST_LocateBetweenElevations . . . 457
7.19.9 ST_InterpolatePoint . . . 457
7.19.10 ST_AddMeasure . . . 458
7.20 Trajectory Functions . . . 459
7.20.1 ST_IsValidTrajectory . . . 459
7.20.2 ST_ClosestPointOfApproach . . . 460
7.20.3 ST_DistanceCPA . . . 461
7.20.4 ST_CPAWithin . . . 461
7.21 Version Functions . . . 462
7.21.1 PostGIS_Extensions_Upgrade . . . 462
7.21.2 PostGIS_Full_Version . . . 463
7.21.3 PostGIS_GEOS_Version . . . 464
7.21.4 PostGIS_GEOS_Compiled_Version . . . 464
7.21.5 PostGIS_Liblwgeom_Version . . . 465
7.21.6 PostGIS_LibXML_Version . . . 465
7.21.7 PostGIS_LibJSON_Version . . . 466
7.21.8 PostGIS_Lib_Build_Date . . . 466
7.21.9 PostGIS_Lib_Version . . . 466
7.21.10 PostGIS_PROJ_Version . . . 467
7.21.11 PostGIS_PROJ_Compiled_Version . . . 467
7.21.12 PostGIS_Wagyu_Version . . . 468
7.21.13 PostGIS_Scripts_Build_Date . . . 468
7.21.14 PostGIS_Scripts_Installed . . . 469
7.21.15 PostGIS_Scripts_Released . . . 470
7.21.16 PostGIS_Version . . . 470
7.22 Grand Unified Custom Variables (GUCs) . . . 471
7.22.1 postgis.gdal_datapath . . . 471
7.22.2 postgis.gdal_enabled_drivers . . . 472
7.22.3 postgis.enable_outdb_rasters . . . 473
7.22.4 postgis.gdal_vsi_options . . . 474
7.23 Troubleshooting Functions . . . 474
7.23.1 PostGIS_AddBBox . . . 474
7.23.2 PostGIS_DropBBox . . . 475
7.23.3 PostGIS_HasBBox . . . 476

8 SFCGAL Functions Reference 477

8.1 SFCGAL Management Functions . . . 477

8.1.1 postgis_sfcgal_version . . . 477

8.1.2 postgis_sfcgal_full_version . . . 477

8.2 SFCGAL Accessors and Setters . . . 478

8.2.1 CG_ForceLHR . . . 478

8.2.2 CG_IsPlanar . . . 478

8.2.3 CG_IsSolid . . . 479

8.2.4 CG_MakeSolid . . . 479

8.2.5 CG_Orientation . . . 479

8.2.6 CG_Area . . . 480

8.2.7 CG_3DArea . . . 480

8.2.8 CG_Volume . . . 481

8.2.9 ST_ForceLHR . . . 482

8.2.10 ST_IsPlanar . . . 483

8.2.11 ST_IsSolid . . . 483

8.2.12 ST_MakeSolid . . . 484

8.2.13 ST_Orientation . . . 484

8.2.14 ST_3DArea . . . 485

8.2.15 ST_Volume . . . 486

8.3 SFCGAL Processing and Relationship Functions . . . 487

8.3.1 CG_Intersection . . . 487

8.3.2 CG_Intersects . . . 487

8.3.3 CG_3DIntersects . . . 488

8.3.4 CG_Difference . . . 489

8.3.5 ST_3DDifference . . . 490

8.3.6 CG_3DDifference . . . 490

8.3.7 CG_Distance . . . 491

8.3.8 CG_3DDistance . . . 492

8.3.9 ST_3DConvexHull . . . 493

8.3.10 CG_3DConvexHull . . . 493

8.3.11 ST_3DIntersection . . . 494

8.3.12 CG_3DIntersection . . . 495

8.3.13 CG_Union . . . 497

8.3.14 ST_3DUnion . . . 498

8.3.15 CG_3DUnion . . . 498

8.3.16 ST_AlphaShape . . . 499

8.3.17 CG_AlphaShape . . . 500

8.3.18 CG_ApproxConvexPartition . . . 503

8.3.19 ST_ApproximateMedialAxis . . . 504
8.3.20 CG_ApproximateMedialAxis . . . 505
8.3.21 ST_ConstrainedDelaunayTriangles . . . 506
8.3.22 CG_ConstrainedDelaunayTriangles . . . 506
8.3.23 ST_Extrude . . . 508
8.3.24 CG_Extrude . . . 508
8.3.25 CG_ExtrudeStraightSkeleton . . . 510
8.3.26 CG_GreeneApproxConvexPartition . . . 511
8.3.27 ST_MinkowskiSum . . . 512
8.3.28 CG_MinkowskiSum . . . 512
8.3.29 ST_OptimalAlphaShape . . . 514
8.3.30 CG_OptimalAlphaShape . . . 515
8.3.31 CG_OptimalConvexPartition . . . 517
8.3.32 CG_StraightSkeleton . . . 518
8.3.33 ST_StraightSkeleton . . . 519
8.3.34 ST_Tesselate . . . 521
8.3.35 CG_Tesselate . . . 521
8.3.36 CG_Triangulate . . . 523
8.3.37 CG_Visibility . . . 524
8.3.38 CG_YMonotonePartition . . . 525

9 Topology **527**

9.1 Topology Types . . . 527
9.1.1 getfaceedges_returntype . . . 527
9.1.2 TopoGeometry . . . 528
9.1.3 validatetopology_returntype . . . 528
9.2 Topology Domains . . . 529
9.2.1 TopoElement . . . 529
9.2.2 TopoElementArray . . . 529
9.3 Topology and TopoGeometry Management . . . 530
9.3.1 AddTopoGeometryColumn . . . 530
9.3.2 RenameTopoGeometryColumn . . . 531
9.3.3 DropTopology . . . 531
9.3.4 RenameTopology . . . 532
9.3.5 DropTopoGeometryColumn . . . 532
9.3.6 Populate_Topology_Layer . . . 533
9.3.7 TopologySummary . . . 534
9.3.8 ValidateTopology . . . 535
9.3.9 ValidateTopologyRelation . . . 537

9.3.10 FindTopology . . . 537
9.3.11 FindLayer . . . 537
9.4 Topology Statistics Management . . . 538
9.5 Topology Constructors . . . 538
9.5.1 CreateTopology . . . 538
9.5.2 CopyTopology . . . 539
9.5.3 ST_InitTopoGeo . . . 540
9.5.4 ST_CreateTopoGeo . . . 540
9.5.5 TopoGeo_AddPoint . . . 541
9.5.6 TopoGeo_AddLineString . . . 541
9.5.7 TopoGeo_AddPolygon . . . 542
9.5.8 TopoGeo_LoadGeometry . . . 543
9.6 Topology Editors . . . 543
9.6.1 ST_AddIsoNode . . . 543
9.6.2 ST_AddIsoEdge . . . 544
9.6.3 ST_AddEdgeNewFaces . . . 544
9.6.4 ST_AddEdgeModFace . . . 545
9.6.5 ST_RemEdgeNewFace . . . 546
9.6.6 ST_RemEdgeModFace . . . 546
9.6.7 ST_ChangeEdgeGeom . . . 547
9.6.8 ST_ModEdgeSplit . . . 548
9.6.9 ST_ModEdgeHeal . . . 548
9.6.10 ST_NewEdgeHeal . . . 549
9.6.11 ST_MoveIsoNode . . . 549
9.6.12 ST_NewEdgesSplit . . . 550
9.6.13 ST_RemoveIsoNode . . . 551
9.6.14 ST_RemoveIsoEdge . . . 551
9.7 Topology Accessors . . . 552
9.7.1 GetEdgeByPoint . . . 552
9.7.2 GetFaceByPoint . . . 553
9.7.3 GetFaceContainingPoint . . . 554
9.7.4 GetNodeByPoint . . . 554
9.7.5 GetTopologyID . . . 555
9.7.6 GetTopologySRID . . . 555
9.7.7 GetTopologyName . . . 556
9.7.8 ST_GetFaceEdges . . . 556
9.7.9 ST_GetFaceGeometry . . . 557
9.7.10 GetRingEdges . . . 558
9.7.11 GetNodeEdges . . . 558

9.8 Topology Processing . . . 559
9.8.1 Polygonize . . . 559
9.8.2 AddNode . . . 560
9.8.3 AddEdge . . . 560
9.8.4 AddFace . . . 562
9.8.5 ST_Simplify . . . 563
9.8.6 RemoveUnusedPrimitives . . . 564
9.9 TopoGeometry Constructors . . . 564
9.9.1 CreateTopoGeom . . . 564
9.9.2 toTopoGeom . . . 566
9.9.3 TopoElementArray_Agg . . . 567
9.9.4 TopoElement . . . 567
9.10 TopoGeometry Editors . . . 568
9.10.1 clearTopoGeom . . . 568
9.10.2 TopoGeom_addElement . . . 569
9.10.3 TopoGeom_remElement . . . 569
9.10.4 TopoGeom_addTopoGeom . . . 570
9.10.5 toTopoGeom . . . 570
9.11 TopoGeometry Accessors . . . 570
9.11.1 GetTopoGeomElementArray . . . 570
9.11.2 GetTopoGeomElements . . . 571
9.11.3 ST_SRID . . . 571
9.12 TopoGeometry Outputs . . . 572
9.12.1 AsGML . . . 572
9.12.2 AsTopoJSON . . . 574
9.13 Topology Spatial Relationships . . . 576
9.13.1 Equals . . . 576
9.13.2 Intersects . . . 577
9.14 Importing and exporting Topologies . . . 577
9.14.1 Using the Topology exporter . . . 578
9.14.2 Using the Topology importer . . . 578

10 Raster Data Management, Queries, and Applications 579

10.1 Loading and Creating Rasters . . . 579
10.1.1 Using raster2pgsql to load rasters . . . 579
10.1.1.1 Example Usage . . . 579
10.1.1.2 raster2pgsql options . . . 580
10.1.2 Creating rasters using PostGIS raster functions . . . 581
10.1.3 Using "out db" cloud rasters . . . 582

10.2 Raster Catalogs . . . 583
10.2.1 Raster Columns Catalog . . . 583
10.2.2 Raster Overviews . . . 584
10.3 Building Custom Applications with PostGIS Raster . . . 585
10.3.1 PHP Example Outputting using ST_AsPNG in concert with other raster functions . . . 585
10.3.2 ASP.NET C# Example Outputting using ST_AsPNG in concert with other raster functions . . . 586
10.3.3 Java console app that outputs raster query as Image file . . . 587
10.3.4 Use PLPython to dump out images via SQL . . . 588
10.3.5 Outputting Rasters with PSQL . . . 589

11 Raster Reference **590**
11.1 Raster Support Data types . . . 591
11.1.1 geomval . . . 591
11.1.2 addbandarg . . . 591
11.1.3 rastbandarg . . . 591
11.1.4 raster . . . 592
11.1.5 reclassarg . . . 592
11.1.6 summarystats . . . 593
11.1.7 unionarg . . . 593
11.2 Raster Management . . . 594
11.2.1 AddRasterConstraints . . . 594
11.2.2 DropRasterConstraints . . . 596
11.2.3 AddOverviewConstraints . . . 597
11.2.4 DropOverviewConstraints . . . 597
11.2.5 PostGIS_GDAL_Version . . . 598
11.2.6 PostGIS_Raster_Lib_Build_Date . . . 598
11.2.7 PostGIS_Raster_Lib_Version . . . 599
11.2.8 ST_GDALDrivers . . . 599
11.2.9 ST_Contour . . . 603
11.2.10 ST_InterpolateRaster . . . 604
11.2.11 UpdateRasterSRID . . . 605
11.2.12 ST_CreateOverview . . . 606
11.3 Raster Constructors . . . 606
11.3.1 ST_AddBand . . . 606
11.3.2 ST_AsRaster . . . 609
11.3.3 ST_Band . . . 611
11.3.4 ST_MakeEmptyCoverage . . . 612
11.3.5 ST_MakeEmptyRaster . . . 613
11.3.6 ST_Tile . . . 614

11.3.7 ST_Retile . . . 617
11.3.8 ST_FromGDALRaster . . . 617
11.4 Raster Accessors . . . 618
11.4.1 ST_GeoReference . . . 618
11.4.2 ST_Height . . . 619
11.4.3 ST_IsEmpty . . . 619
11.4.4 ST_MemSize . . . 620
11.4.5 ST_MetaData . . . 621
11.4.6 ST_NumBands . . . 621
11.4.7 ST_PixelHeight . . . 622
11.4.8 ST_PixelWidth . . . 623
11.4.9 ST_ScaleX . . . 624
11.4.10 ST_ScaleY . . . 624
11.4.11 ST_RasterToWorldCoord . . . 625
11.4.12 ST_RasterToWorldCoordX . . . 626
11.4.13 ST_RasterToWorldCoordY . . . 627
11.4.14 ST_Rotation . . . 628
11.4.15 ST_SkewX . . . 628
11.4.16 ST_SkewY . . . 629
11.4.17 ST_SRID . . . 630
11.4.18 ST_Summary . . . 630
11.4.19 ST_UpperLeftX . . . 631
11.4.20 ST_UpperLeftY . . . 632
11.4.21 ST_Width . . . 632
11.4.22 ST_WorldToRasterCoord . . . 633
11.4.23 ST_WorldToRasterCoordX . . . 633
11.4.24 ST_WorldToRasterCoordY . . . 634
11.5 Raster Band Accessors . . . 635
11.5.1 ST_BandMetaData . . . 635
11.5.2 ST_BandNoDataValue . . . 636
11.5.3 ST_BandIsNoData . . . 637
11.5.4 ST_BandPath . . . 638
11.5.5 ST_BandFileSize . . . 639
11.5.6 ST_BandFileTimestamp . . . 639
11.5.7 ST_BandPixelType . . . 640
11.5.8 ST_MinPossibleValue . . . 641
11.5.9 ST_HasNoBand . . . 641
11.6 Raster Pixel Accessors and Setters . . . 642
11.6.1 ST_PixelAsPolygon . . . 642

11.6.2 ST_PixelAsPolygons . . . 643
11.6.3 ST_PixelAsPoint . . . 644
11.6.4 ST_PixelAsPoints . . . 644
11.6.5 ST_PixelAsCentroid . . . 645
11.6.6 ST_PixelAsCentroids . . . 646
11.6.7 ST_Value . . . 647
11.6.8 ST_NearestValue . . . 650
11.6.9 ST_SetZ . . . 652
11.6.10 ST_SetM . . . 653
11.6.11 ST_Neighborhood . . . 654
11.6.12 ST_SetValue . . . 656
11.6.13 ST_SetValues . . . 657
11.6.14 ST_DumpValues . . . 665
11.6.15 ST_PixelOfValue . . . 666
11.7 Raster Editors . . . 667
11.7.1 ST_SetGeoReference . . . 667
11.7.2 ST_SetRotation . . . 669
11.7.3 ST_SetScale . . . 669
11.7.4 ST_SetSkew . . . 670
11.7.5 ST_SetSRID . . . 671
11.7.6 ST_SetUpperLeft . . . 672
11.7.7 ST_Resample . . . 672
11.7.8 ST_Rescale . . . 674
11.7.9 ST_Reskew . . . 675
11.7.10 ST_SnapToGrid . . . 676
11.7.11 ST_Resize . . . 677
11.7.12 ST_Transform . . . 679
11.8 Raster Band Editors . . . 681
11.8.1 ST_SetBandNoDataValue . . . 681
11.8.2 ST_SetBandIsNoData . . . 682
11.8.3 ST_SetBandPath . . . 684
11.8.4 ST_SetBandIndex . . . 685
11.9 Raster Band Statistics and Analytics . . . 687
11.9.1 ST_Count . . . 687
11.9.2 ST_CountAgg . . . 687
11.9.3 ST_Histogram . . . 688
11.9.4 ST_Quantile . . . 690
11.9.5 ST_SummaryStats . . . 692
11.9.6 ST_SummaryStatsAgg . . . 694

11.9.7 ST_ValueCount . . . 695
11.10 Raster Inputs . . . 697
11.10.1 ST_RastFromWKB . . . 697
11.10.2 ST_RastFromHexWKB . . . 698
11.11 Raster Outputs . . . 699
11.11.1 ST_AsBinary/ST_AsWKB . . . 699
11.11.2 ST_AsHexWKB . . . 700
11.11.3 ST_AsGDALRaster . . . 700
11.11.4 ST_AsJPEG . . . 701
11.11.5 ST_AsPNG . . . 702
11.11.6 ST_AsTIFF . . . 703
11.12 Raster Processing: Map Algebra . . . 704
11.12.1 ST_Clip . . . 704
11.12.2 ST_ColorMap . . . 707
11.12.3 ST_Grayscale . . . 710
11.12.4 ST_Intersection . . . 712
11.12.5 ST_MapAlgebra (callback function version) . . . 714
11.12.6 ST_MapAlgebra (expression version) . . . 720
11.12.7 ST_MapAlgebraExpr . . . 722
11.12.8 ST_MapAlgebraExpr . . . 725
11.12.9 ST_MapAlgebraFct . . . 729
11.12.10 ST_MapAlgebraFct . . . 733
11.12.11 ST_MapAlgebraFctNgb . . . 737
11.12.12 ST_Reclass . . . 739
11.12.13 ST_Union . . . 741
11.13 Built-in Map Algebra Callback Functions . . . 742
11.13.1 ST_Distinct4ma . . . 742
11.13.2 ST_InvDistWeight4ma . . . 743
11.13.3 ST_Max4ma . . . 744
11.13.4 ST_Mean4ma . . . 745
11.13.5 ST_Min4ma . . . 746
11.13.6 ST_MinDist4ma . . . 747
11.13.7 ST_Range4ma . . . 748
11.13.8 ST_StdDev4ma . . . 749
11.13.9 ST_Sum4ma . . . 750
11.14 Raster Processing: DEM (Elevation) . . . 751
11.14.1 ST_Aspect . . . 751
11.14.2 ST_HillShade . . . 753
11.14.3 ST_Roughness . . . 755

11.14.4 ST_Slope . . . 755
11.14.5 ST_TPI . . . 757
11.14.6 ST_TRI . . . 757
11.15 Raster Processing: Raster to Geometry . . . 758
11.15.1 Box3D . . . 758
11.15.2 ST_ConvexHull . . . 759
11.15.3 ST_DumpAsPolygons . . . 760
11.15.4 ST_Envelope . . . 761
11.15.5 ST_MinConvexHull . . . 761
11.15.6 ST_Polygon . . . 762
11.16 Raster Operators . . . 764
11.16.1 && . . . 764
11.16.2 &< . . . 764
11.16.3 &> . . . 765
11.16.4 = . . . 766
11.16.5 @ . . . 766
11.16.6 ~= . . . 767
11.16.7 ~ . . . 767
11.17 Raster and Raster Band Spatial Relationships . . . 768
11.17.1 ST_Contains . . . 768
11.17.2 ST_ContainsProperly . . . 769
11.17.3 ST_Covers . . . 770
11.17.4 ST_CoveredBy . . . 771
11.17.5 ST_Disjoint . . . 772
11.17.6 ST_Intersects . . . 773
11.17.7 ST_Overlaps . . . 773
11.17.8 ST_Touches . . . 774
11.17.9 ST_SameAlignment . . . 775
11.17.10 ST_NotSameAlignmentReason . . . 776
11.17.11 ST_Within . . . 777
11.17.12 ST_DWithin . . . 778
11.17.13 ST_DFullyWithin . . . 779
11.18 Raster Tips . . . 780
11.18.1 Out-DB Rasters . . . 780
11.18.1.1 Directory containing many files . . . 780
11.18.1.2 Maximum Number of Open Files . . . 780
11.18.1.2.1 Maximum number of open files for the entire system . . . 781
11.18.1.2.2 Maximum number of open files per process . . . 781

12 PostGIS Extras **783**

12.1 Address Standardizer . . . 783

12.1.1 How the Parser Works . . . 783

12.1.2 Address Standardizer Types . . . 783

12.1.2.1 stdaddr . . . 783

12.1.3 Address Standardizer Tables . . . 784

12.1.3.1 rules table . . . 784

12.1.3.2 lex table . . . 787

12.1.3.3 gaz table . . . 787

12.1.4 Address Standardizer Functions . . . 788

12.1.4.1 debug_standardize_address . . . 788

12.1.4.2 parse_address . . . 789

12.1.4.3 standardize_address . . . 790

12.2 Tiger Geocoder . . . 792

12.2.1 Drop_Indexes_Generate_Script . . . 792

12.2.2 Drop_Nation_Tables_Generate_Script . . . 793

12.2.3 Drop_State_Tables_Generate_Script . . . 794

12.2.4 Geocode . . . 795

12.2.5 Geocode_Intersection . . . 797

12.2.6 Get_Geocode_Setting . . . 798

12.2.7 Get_Tract . . . 799

12.2.8 Install_Missing_Indexes . . . 800

12.2.9 Loader_Generate_Census_Script . . . 800

12.2.10 Loader_Generate_Script . . . 802

12.2.11 Loader_Generate_Nation_Script . . . 804

12.2.12 Missing_Indexes_Generate_Script . . . 805

12.2.13 Normalize_Address . . . 806

12.2.14 Pagc_Normalize_Address . . . 807

12.2.15 Pprint_Addy . . . 809

12.2.16 Reverse_Geocode . . . 810

12.2.17 Topology_Load_Tiger . . . 812

12.2.18 Set_Geocode_Setting . . . 814

13 PostGIS Special Functions Index **815**

13.1 PostGIS Aggregate Functions . . . 815

13.2 PostGIS Window Functions . . . 816

13.3 PostGIS SQL-MM Compliant Functions . . . 816

13.4 PostGIS Geography Support Functions . . . 820

13.5 PostGIS Raster Support Functions . . . 821

13.6 PostGIS Geometry / Geography / Raster Dump Functions . . . 826
13.7 PostGIS Box Functions . . . 827
13.8 PostGIS Functions that support 3D . . . 828
13.9 PostGIS Curved Geometry Support Functions . . . 833
13.10 PostGIS Polyhedral Surface Support Functions . . . 836
13.11 PostGIS Function Support Matrix . . . 839
13.12 New, Enhanced or changed PostGIS Functions . . . 857
13.12.1 PostGIS Functions new or enhanced in 3.5 . . . 857
13.12.2 PostGIS Functions new or enhanced in 3.4 . . . 859
13.12.3 PostGIS Functions new or enhanced in 3.3 . . . 860
13.12.4 PostGIS Functions new or enhanced in 3.2 . . . 861
13.12.5 PostGIS Functions new or enhanced in 3.1 . . . 863
13.12.6 PostGIS Functions new or enhanced in 3.0 . . . 864
13.12.7 PostGIS Functions new or enhanced in 2.5 . . . 866
13.12.8 PostGIS Functions new or enhanced in 2.4 . . . 867
13.12.9 PostGIS Functions new or enhanced in 2.3 . . . 868
13.12.10 PostGIS Functions new or enhanced in 2.2 . . . 870
13.12.11 PostGIS Functions new or enhanced in 2.1 . . . 873
13.12.12 PostGIS Functions new or enhanced in 2.0 . . . 878
13.12.13 PostGIS Functions new or enhanced in 1.5 . . . 888
13.12.14 PostGIS Functions new or enhanced in 1.4 . . . 889
13.12.15 PostGIS Functions new or enhanced in 1.3 . . . 890

14 Reporting Problems **891**
14.1 Reporting Software Bugs . . . 891
14.2 Reporting Documentation Issues . . . 891

A Appendix **892**
A.1 PostGIS 3.5.3 . . . 892
A.1.1 Bug Fixes . . . 892
A.2 PostGIS 3.5.2 . . . 892
A.2.1 Bug Fixes . . . 892
A.3 PostGIS 3.5.1 . . . 893
A.3.1 Breaking Changes . . . 893
A.3.2 Enhancements . . . 893
A.3.3 Breaking Changes . . . 893
A.4 PostGIS 3.5.0 . . . 893
A.4.1 Breaking Changes . . . 894
A.4.2 Deprecated signatures . . . 894
A.4.3 New features . . . 894
A.4.4 Enhancements . . . 895

Abstract

PostGIS is an extension to the PostgreSQL object-relational database system which allows GIS (Geographic Information Systems) objects to be stored in the database. PostGIS includes support for GiST-based R-Tree spatial indexes, and functions for analysis and processing of GIS objects.

This is the manual for version 3.5.3

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

Introduction

PostGIS is a spatial extension for the PostgreSQL relational database that was created by Refractions Research Inc, as a spatial database technology research project. Refractions is a GIS and database consulting company in Victoria, British Columbia, Canada, specializing in data integration and custom software development.

PostGIS is now a project of the OSGeo Foundation and is developed and funded by many FOSS4G developers and organizations all over the world that gain great benefit from its functionality and versatility.

The PostGIS project development group plans on supporting and enhancing PostGIS to better support a range of important GIS functionality in the areas of OGC and SQL/MM spatial standards, advanced topological constructs (coverages, surfaces, networks), data source for desktop user interface tools for viewing and editing GIS data, and web-based access tools.

1.1 Project Steering Committee

The PostGIS Project Steering Committee (PSC) coordinates the general direction, release cycles, documentation, and outreach efforts for the PostGIS project. In addition the PSC provides general user support, accepts and approves patches from the general PostGIS community and votes on miscellaneous issues involving PostGIS such as developer commit access, new PSC members or significant API changes.

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

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

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

Paul Ramsey (Chair) Co-founder of PostGIS project. General bug fixing, geography support, geography and geometry index support (2D, 3D, nD index and anything spatial index), underlying geometry internal structures, GEOS functionality integration and alignment with GEOS releases, alignment of PostGIS with PostgreSQL releases, loader/dumper, and Shapefile GUI loader.

Sandro Santilli Bug fixes and maintenance, ci maintenance, git mirror management, management functions, integration of new GEOS functionality and alignment with GEOS releases, topology support, and raster framework and low level API functions.

1.2 Core Contributors Present

Nicklas Avén Distance function enhancements (including 3D distance and relationship functions) and additions, Tiny WKB (TWKB) output format and general user support

Loïc Bartoletti SFCGAL enhancements and maintenance and ci support

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

Martin Davis GEOS enhancements and documentation

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

Aliaksandr Kalenik Geometry Processing, PostgreSQL gist, general bug fixing

1.3 Core Contributors Past

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

Mark Cave-Ayland Prior PSC Member. Coordinated bug fixing and maintenance effort, spatial index selectivity and binding, loader/dumper, and Shapefile GUI Loader, integration of new and new function enhancements.

Jorge Arévalo Raster development, GDAL driver support, loader

Olivier Courtin (Emeritus) Input/output XML (KML,GML)/GeoJSON functions, 3D support and bug fixes.

Chris Hodgson Prior PSC Member. General development, site and buildbot maintenance, OSGeo incubation management

Mateusz Loskot CMake support for PostGIS, built original raster loader in python and low level raster API functions

Kevin Neufeld Prior PSC Member. Documentation and documentation support tools, buildbot maintenance, advanced user support on PostGIS newsgroup, and PostGIS maintenance function enhancements.

Dave Blasby The original developer/Co-founder of PostGIS. Dave wrote the server side objects, index bindings, and many of the server side analytical functions.

Jeff Lounsbury Original development of the Shapefile loader/dumper.

Mark Leslie Ongoing maintenance and development of core functions. Enhanced curve support. Shapefile GUI loader.

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

David Zwarg Raster development (mostly map algebra analytic functions)

1.4 Other Contributors

Individual Contributors

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

Corporate Sponsors These are corporate entities that have contributed developer time, hosting, or direct monetary funding to the PostGIS project. In alphabetical order:

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

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

Crowd Funding Campaigns Crowd funding campaigns are campaigns we run to get badly wanted features funded that can service a large number of people. Each campaign is specifically focused on a particular feature or set of features. Each sponsor chips in a small fraction of the needed funding and with enough people/organizations contributing, we have the funds to pay for the work that will help many. If you have an idea for a feature you think many others would be willing to co-fund, please post to the PostGIS newsgroup your thoughts and together we can make it happen.

PostGIS 2.0.0 was the first release we tried this strategy. We used PledgeBank and we got two successful campaigns out of it.

postgistopology - 10 plus sponsors each contributed $250 USD to build toTopoGeometry function and beef up topology support in 2.0.0. It happened.

postgis64windows - 20 someodd sponsors each contributed $100 USD to pay for the work needed to work out PostGIS 64-bit issues on windows. It happened.

Important Support Libraries 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

Last but not least, PostgreSQL, the giant that PostGIS stands on. Much of the speed and flexibility of PostGIS would not be possible without the extensibility, great query planner, GIST index, and plethora of SQL features provided by PostgreSQL.

Chapter 2

PostGIS Installation

This chapter details the steps required to install PostGIS.

2.1 Short Version

To compile assuming you have all the dependencies in your search path:

```
tar -xvzf postgis-3.5.3.tar.gz
cd postgis-3.5.3
./configure
make
make install
```

Once PostGIS is installed, it needs to be enabled (Section 3.3) or upgraded (Section 3.4) in each individual database you want to use it in.

2.2 Compiling and Install from Source

Note

Many OS systems now include pre-built packages for PostgreSQL/PostGIS. In many cases compilation is only necessary if you want the most bleeding edge versions or you are a package maintainer.

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

If you are a windows user, you can get stable builds via Stackbuilder or PostGIS Windows download site We also have very bleeding-edge windows experimental builds that are built usually once or twice a week or whenever anything exciting happens. You can use these to experiment with the in progress releases of PostGIS

The PostGIS module is an extension to the PostgreSQL backend server. As such, PostGIS 3.5.3 *requires* full PostgreSQL server headers access in order to compile. It can be built against PostgreSQL versions 12 - 17. 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

For GEOS functionality, when you install PostgreSQL you may need to explicitly link PostgreSQL against the standard C++ library:

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

This is a workaround for bogus C++ exceptions interaction with older development tools. If you experience weird problems (backend unexpectedly closed or similar things) try this trick. This will require recompiling your PostgreSQL from scratch, of course.

The following steps outline the configuration and compilation of the PostGIS source. They are written for Linux users and will not work on Windows or Mac.

2.2.1 Getting the Source

Retrieve the PostGIS source archive from the downloads website https://download.osgeo.org/postgis/source/postgis-3.5.3.tar.gz

```
wget https://download.osgeo.org/postgis/source/postgis-3.5.3.tar.gz
tar -xvzf postgis-3.5.3.tar.gz
cd postgis-3.5.3
```

This will create a directory called `postgis-3.5.3` in the current working directory.

Alternatively, checkout the source from the git repository https://git.osgeo.org/gitea/postgis/postgis/ .

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

Change into the newly created `postgis` directory to continue the installation.

```
./configure
```

2.2.2 Install Requirements

PostGIS has the following requirements for building and usage:

Required

- PostgreSQL 12 - 17. 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`). Some other ANSI C compilers can be used to compile PostGIS, but we find far fewer problems when compiling with `gcc`.

- GNU Make (`gmake` or `make`). For many systems, GNU `make` is the default version of make. Check the version by invoking `make -v`. Other versions of `make` may not process the PostGIS `Makefile` properly.

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

- GEOS geometry library, version 3.8.0 or greater, but GEOS 3.12+ 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 or higher. JSON-C is currently used to import GeoJSON via the function ST_GeomFromGeoJson. JSON-C is available for download from https://github.com/json-c/json-c/releases/.
- GDAL, version 3+ is preferred. This is required for raster support. https://gdal.org/download.html.
- If compiling with PostgreSQL+JIT, LLVM version >=6 is required https://trac.osgeo.org/postgis/ticket/4125.

Optional

- GDAL (pseudo optional) only if you don't want raster you can leave it out. Also make sure to enable the drivers you want to use as described in Section 3.2.
- GTK (requires GTK+2.0, 2.8+) to compile the shp2pgsql-gui shape file loader. http://www.gtk.org/ .
- SFCGAL, 1.4.1 or higher is required and 1.5.0+ is needed to be able to use all functionality. SFCGAL can be used to provide additional 2D and 3D advanced analysis functions to PostGIS cf Chapter 8. And also allow to use SFCGAL rather than GEOS for some 2D functions provided by both backends (like ST_Intersection or ST_Area, for instance). A PostgreSQL configuration variable `postgis.backend` allow end user to control which backend he want to use if SFCGAL is installed (GEOS by default). Nota: SFCGAL 1.2 require at least CGAL 4.3 and Boost 1.54 (cf: https://sfcgal.org) https://gitlab.com/-sfcgal/SFCGAL/.
- In order to build the Section 12.1 you will also need PCRE http://www.pcre.org (which generally is already installed on nix systems). Section 12.1 will automatically be built if it detects a PCRE library, or you pass in a valid `--with-pcre-dir=/path/to/p` during configure.
- To enable ST_AsMVT protobuf-c library 1.1.0 or higher (for usage) and the protoc-c compiler (for building) are required. Also, pkg-config is required to verify the correct minimum version of protobuf-c. See protobuf-c. By default, Postgis will use Wagyu to validate MVT polygons faster which requires a c++11 compiler. It will use CXXFLAGS and the same compiler as the PostgreSQL installation. To disable this and use GEOS instead use the `--without-wagyu` during the configure step.
- CUnit (`CUnit`). This is needed for regression testing. http://cunit.sourceforge.net/
- DocBook (`xsltproc`) is required for building the documentation. Docbook is available from http://www.docbook.org/ .
- DBLatex (`dblatex`) is required for building the documentation in PDF format. DBLatex is available from http://dblatex.sourceforge.n .
- ImageMagick (`convert`) is required to generate the images used in the documentation. ImageMagick is available from http://www.imagemagick.org/ .

2.2.3 Build configuration

As with most linux installations, the first step is to generate the Makefile that will be used to build the source code. This is done by running the shell script

./configure

With no additional parameters, this command will attempt to automatically locate the required components and libraries needed to build the PostGIS source code on your system. Although this is the most common usage of **./configure**, the script accepts several parameters for those who have the required libraries and programs in non-standard locations.

The following list shows only the most commonly used parameters. For a complete list, use the **--help** or **--help=short** parameters.

`--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` This is the location the PostGIS loader executables and shared libs will be installed. By default, this location is the same as the detected PostgreSQL installation.

Caution
This parameter is currently broken, as the package will only install into the PostgreSQL installation directory. Visit http://trac.osgeo.org/postgis/ticket/635 to track this bug.

--with-pgconfig=FILE PostgreSQL provides a utility called **pg_config** to enable extensions like PostGIS to locate the PostgreSQL installation directory. Use this parameter (**--with-pgconfig=/path/to/pg_config**) to manually specify a particular PostgreSQL installation that PostGIS will build against.

--with-gdalconfig=FILE GDAL, a required library, provides functionality needed for raster support **gdal-config** to enable software installations to locate the GDAL installation directory. Use this parameter (**--with-gdalconfig=/path/to/gdal-config**) to manually specify a particular GDAL installation that PostGIS will build against.

--with-geosconfig=FILE GEOS, a required geometry library, provides a utility called **geos-config** to enable software installations to locate the GEOS installation directory. Use this parameter (**--with-geosconfig=/path/to/geos-config**) to manually specify a particular GEOS installation that PostGIS will build against.

--with-xml2config=FILE LibXML is the library required for doing GeomFromKML/GML processes. It normally is found if you have libxml installed, but if not or you want a specific version used, you'll need to point PostGIS at a specific `xml2-config` confi file to enable software installations to locate the LibXML installation directory. Use this parameter (**>--with-xml2config=/path/to/xml2-config**) to manually specify a particular LibXML installation that PostGIS will build against.

--with-projdir=DIR Proj is a reprojection library required by PostGIS. Use this parameter (**--with-projdir=/path/to/projdir**) to manually specify a particular Proj installation directory that PostGIS will build against.

--with-libiconv=DIR Directory where iconv is installed.

--with-jsondir=DIR JSON-C is an MIT-licensed JSON library required by PostGIS ST_GeomFromJSON support. Use this parameter (**--with-jsondir=/path/to/jsondir**) to manually specify a particular JSON-C installation directory that PostGIS will build against.

--with-pcredir=DIR PCRE is an BSD-licensed Perl Compatible Regular Expression library required by address_standardizer extension. Use this parameter (**--with-pcredir=/path/to/pcredir**) to manually specify a particular PCRE installation directory that PostGIS will build against.

--with-gui Compile the data import GUI (requires GTK+2.0). This will create shp2pgsql-gui graphical interface to shp2pgsql.

--without-raster Compile without raster support.

--without-topology Disable topology support. There is no corresponding library as all logic needed for topology is in postgis-3.5.3 library.

--with-gettext=no By default PostGIS will try to detect gettext support and compile with it, however if you run into incompatibility issues that cause breakage of loader, you can disable it entirely with this command. Refer to ticket http://trac.osgeo.org/postgis/ticket/748 for an example issue solved by configuring with this. NOTE: that you aren't missing much by turning this off. This is used for international help/label support for the GUI loader which is not yet documented and still experimental.

--with-sfcgal=PATH By default PostGIS will not install with sfcgal support without this switch. `PATH` is an optional argument that allows to specify an alternate PATH to sfcgal-config.

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

Note
If you obtained PostGIS from the code repository , the first step is really to run the script
./autogen.sh
This script will generate the **configure** script that in turn is used to customize the installation of PostGIS.
If you instead obtained PostGIS as a tarball, running **./autogen.sh** is not necessary as **configure** has already been generated.

2.2.4 Building

Once the Makefile has been generated, building PostGIS is as simple as running

make

The last line of the output should be "`PostGIS was built successfully. Ready to install.`"

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

make comments

Introduced in PostGIS 2.0. This generates html cheat sheets suitable for quick reference or for student handouts. This requires xsltproc to build and will generate 4 files in doc folder `topology_cheatsheet.html`, `tiger_geocoder_cheatsheet.html`, `raster_cheatsheet.html`, `postgis_cheatsheet.html`

You can download some pre-built ones available in html and pdf from PostGIS / PostgreSQL Study Guides

make cheatsheets

2.2.5 Building PostGIS Extensions and Deploying them

The PostGIS extensions are built and installed automatically if you are using PostgreSQL 9.1+.

If you are building from source repository, you need to build the function descriptions first. These get built if you have docbook installed. You can also manually build with the statement:

make comments

Building the comments is not necessary if you are building from a release tar ball since these are packaged pre-built with the tar ball already.

The extensions should automatically build as part of the make install process. You can if needed build from the extensions folders or copy files if you need them on a different server.

```
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

The extension files will always be the same for the same version of PostGIS and PostgreSQL regardless of OS, so it is fine to copy over the extension files from one OS to another as long as you have the PostGIS binaries already installed on your servers.

If you want to install the extensions manually on a separate server different from your development, You need to copy the following files from the extensions folder into the `PostgreSQL / share / extension` folder of your PostgreSQL install as well as the needed binaries for regular PostGIS if you don't have them already on the server.

- These are the control files that denote information such as the version of the extension to install if not specified. `postgis.control, postgis_topology.control`.

- All the files in the /sql folder of each extension. Note that these need to be copied to the root of the PostgreSQL share/extension folder `extensions/postgis/sql/*.sql`, `extensions/postgis_topology/sql/*.sql`

Once you do that, you should see `postgis`, `postgis_topology` as available extensions in PgAdmin -> extensions.

If you are using psql, you can verify that the extensions are installed by running this query:

```
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          | 3.5.3           | 3.5.3
 address_standardizer_data_us  | 3.5.3           | 3.5.3
 postgis                       | 3.5.3           | 3.5.3
 postgis_raster                | 3.5.3           | 3.5.3
 postgis_sfcgal                | 3.5.3           |
 postgis_tiger_geocoder        | 3.5.3           | 3.5.3
 postgis_topology              | 3.5.3           |
(6 rows)
```

If you have the extension installed in the database you are querying, you'll see mention in the `installed_version` column. If you get no records back, it means you don't have postgis extensions installed on the server at all. PgAdmin III 1.14+ will also provide this information in the `extensions` section of the database browser tree and will even allow upgrade or uninstall by right-clicking.

If you have the extensions available, you can install postgis extension in your database of choice by either using pgAdmin extension interface or running these sql commands:

```
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;
```

In psql you can use to see what versions you have installed and also what schema they are installed.

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

```
List of installed extensions
-[ RECORD 1 ]-------------------------------------------------
Name        | postgis
Version     | 3.5.3
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     | 3.5.3
Schema      | tiger
Description | PostGIS tiger geocoder and reverse geocoder
-[ RECORD 4 ]-------------------------------------------------
```

```
Name        | postgis_topology
Version     | 3.5.3
Schema      | topology
Description | PostGIS topology spatial types and functions
```

Warning

Extension tables `spatial_ref_sys`, `layer`, `topology` can not be explicitly backed up. They can only be backed up when the respective `postgis` or `postgis_topology` extension is backed up, which only seems to happen when you backup the whole database. As of PostGIS 2.0.1, only srid records not packaged with PostGIS are backed up when the database is backed up so don't go around changing srids we package and expect your changes to be there. Put in a ticket if you find an issue. The structures of extension tables are never backed up since they are created with `CREATE EXTENSION` and assumed to be the same for a given version of an extension. These behaviors are built into the current PostgreSQL extension model, so nothing we can do about it.

If you installed 3.5.3, without using our wonderful extension system, you can change it to be extension based by running the below commands to package the functions in their respective extension. Installing using `unpackaged` was removed in PostgreSQL 13, so you are advised to switch to an extension build before upgrading to PostgreSQL 13.

```
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 Testing

If you wish to test the PostGIS build, run

make check

The above command will run through various checks and regression tests using the generated library against an actual PostgreSQL database.

Note

If you configured PostGIS using non-standard PostgreSQL, GEOS, or Proj locations, you may need to add their library locations to the `LD_LIBRARY_PATH` environment variable.

Caution

Currently, the **make check** relies on the `PATH` and `PGPORT` environment variables when performing the checks - it does *not* use the PostgreSQL version that may have been specified using the configuration parameter **--with-pgconfig**. So make sure to modify your PATH to match the detected PostgreSQL installation during configuration or be prepared to deal with the impending headaches.

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
```

The `postgis_tiger_geocoder` and `address_standardizer` extensions, currently only support the standard PostgreSQL installcheck. To test these use the below. Note: the make install is not necessary if you already did make install at root of PostGIS code folder.

For address_standardizer:

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

Output should look like:

```
============== 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.
======================
```

For tiger geocoder, make sure you have postgis and fuzzystrmatch extensions available in your PostgreSQL instance. The address_standardizer tests will also kick in if you built postgis with address_standardizer support:

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

output should look like:

```
============== 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

To install PostGIS, type

make install

This will copy the PostGIS installation files into their appropriate subdirectory specified by the **--prefix** configuration parameter. In particular:

- The loader and dumper binaries are installed in `[prefix]/bin`.
- The SQL files, such as `postgis.sql`, are installed in `[prefix]/share/contrib`.
- The PostGIS libraries are installed in `[prefix]/lib`.

If you previously ran the **make comments** command to generate the `postgis_comments.sql`, `raster_comments.sql` file, install the sql file by running

make comments-install

Note
`postgis_comments.sql`, `raster_comments.sql`, `topology_comments.sql` was separated from the typical build and installation targets since with it comes the extra dependency of **xsltproc**.

2.3 Installing and Using the address standardizer

The `address_standardizer` extension used to be a separate package that required separate download. From PostGIS 2.2 on, it is now bundled in. For more information about the address_standardize, what it does, and how to configure it for your needs, refer to Section 12.1.

This standardizer can be used in conjunction with the PostGIS packaged tiger geocoder extension as a replacement for the Normalize_Address discussed. To use as replacement refer to Section 2.4.2. You can also use it as a building block for your own geocoder or use it to standardize your addresses for easier compare of addresses.

The address standardizer relies on PCRE which is usually already installed on many Nix systems, but you can download the latest at: http://www.pcre.org. If during Section 2.2.3, PCRE is found, then the address standardizer extension will automatically be built. If you have a custom pcre install you want to use instead, pass to configure `--with-pcredir=/path/to/pcre` where `/path/to/pcre` is the root folder for your pcre include and lib directories.

For Windows users, the PostGIS 2.1+ bundle is packaged with the address_standardizer already so no need to compile and can move straight to `CREATE EXTENSION` step.

Once you have installed, you can connect to your database and run the SQL:

```
CREATE EXTENSION address_standardizer;
```

The following test requires no rules, gaz, or lex tables

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

Output should be

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

2.4 Installing, Upgrading Tiger Geocoder, and loading data

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

2.4.1 Tiger Geocoder Enabling your PostGIS database

1. These directions assume your PostgreSQL installation already has the postgis_tiger_geocoder extension installed.
2. Connect to your database via psql or pgAdmin or some other tool and run the following SQL commands. Note that if you are installing in a database that already has postgis, you don't need to do the first step. If you have `fuzzystrmatch` extension already installed, you don't need to do the second step either.

   ```
   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;
   ```

   If you already have postgis_tiger_geocoder extension installed, and just want to update to the latest run:

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

   If you made custom entries or changes to `tiger.loader_platform` and `tiger.loader_variables` you may need to update these.
3. To confirm your install is working correctly, run this sql in your database:

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

   Which should output

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

4. Create a new record in `tiger.loader_platform` table with the paths of your executables and server.

   So for example to create a profile called debbie that follows `sh` convention. You would do:

```
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';
```

   And then edit the paths in the *declare_sect* column to those that fit Debbie's pg, unzip,shp2pgsql, psql, etc path locations.

   If you don't edit this `loader_platform` table, it will just contain common case locations of items and you'll have to edit the generated script after the script is generated.

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

   To enable it, do the following:

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

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

6. Create a folder called `gisdata` on root of server or your local pc if you have a fast network connection to the server. This folder is where the tiger files will be downloaded to and processed. If you are not happy with having the folder on the root of the server, or simply want to change to a different folder for staging, then edit the field `staging_fold` in the `tiger.loader_variables` table.

7. Create a folder called temp in the `gisdata` folder or wherever you designated the `staging_fold` to be. This will be the folder where the loader extracts the downloaded tiger data.

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

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

9. Run the generated nation load commandline scripts.

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

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

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

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

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

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

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

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

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

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

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

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

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

Warning
DO NOT Generate the state script until you have already loaded the nation data, because the state script utilizes county list loaded by nation script.

13.

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

14. Run the generated commandline scripts.

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

15. After you are done loading all data or at a stopping point, it's a good idea to analyze all the tiger tables to update the stats (include inherited stats)

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

2.4.2 Using Address Standardizer Extension with Tiger geocoder

One of the many complaints of folks is the address normalizer function Normalize_Address function that normalizes an address for prepping before geocoding. The normalizer is far from perfect and trying to patch its imperfectness takes a vast amount of resources. As such we have integrated with another project that has a much better address standardizer engine. To use this new address_standardizer, you compile the extension as described in Section 2.3 and install as an extension in your database.

Once you install this extension in the same database as you have installed `postgis_tiger_geocoder`, then the Pagc_Normalize_Ad can be used instead of Normalize_Address. This extension is tiger agnostic, so can be used with other data sources such as international addresses. The tiger geocoder extension does come packaged with its own custom versions of rules table (`tiger.pagc_rules`) , gaz table (`tiger.pagc_gaz`), and lex table (`tiger.pagc_lex`). These you can add and update to improve your standardizing experience for your own needs.

2.4.3 Required tools for tiger data loading

The load process downloads data from the census website for the respective nation files, states requested, extracts the files, and then loads each state into its own separate set of state tables. Each state table inherits from the tables defined in `tiger` schema so that its sufficient to just query those tables to access all the data and drop a set of state tables at any time using the Drop_State_Tables_Generate_Script if you need to reload a state or just don't need a state anymore.

In order to be able to load data you'll need the following tools:

- A tool to unzip the zip files from census website.

  For Unix like systems: `unzip` executable which is usually already installed on most Unix like platforms.

  For Windows, 7-zip which is a free compress/uncompress tool you can download from http://www.7-zip.org/

- `shp2pgsql` commandline which is installed by default when you install PostGIS.

- `wget` which is a web grabber tool usually installed on most Unix/Linux systems.

  If you are on windows, you can get pre-compiled binaries from http://gnuwin32.sourceforge.net/packages/wget.htm

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

To load state data refer to Loader_Generate_Script to generate a data load script for your platform for the states you desire. Note that you can install these piecemeal. You don't have to load all the states you want all at once. You can load them as you need them.

After the states you desire have been loaded, make sure to run the:

```
SELECT install_missing_indexes();
```

as described in Install_Missing_Indexes.

To test that things are working as they should, try to run a geocode on an address in your state using Geocode

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;
```

Next drop all nation tables and load up the new ones. Generate a drop script with this SQL statement as detailed in Drop_Nation_Tables_G

```
SELECT drop_nation_tables_generate_script();
```

Run the generated drop SQL statements.

Generate a nation load script with this SELECT statement as detailed in Loader_Generate_Nation_Script

For windows

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

For 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 Common Problems during installation

There are several things to check when your installation or upgrade doesn't go as you expected.

1. Check that you have installed PostgreSQL 12 or newer, and that you are compiling against the same version of the PostgreSQL source as the version of PostgreSQL that is running. Mix-ups can occur when your (Linux) distribution has already installed PostgreSQL, or you have otherwise installed PostgreSQL before and forgotten about it. PostGIS will only work with PostgreSQL 12 or newer, and strange, unexpected error messages will result if you use an older version. To check the version of PostgreSQL which is running, connect to the database using psql and run this query:

   ```
   SELECT version();
   ```

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

2. If your upgrade fails, make sure you are restoring into a database that already has PostGIS installed.

   ```
   SELECT postgis_full_version();
   ```

Also check that configure has correctly detected the location and version of PostgreSQL, the Proj library and the GEOS library.

1. The output from configure is used to generate the `postgis_config.h` file. Check that the `POSTGIS_PGSQL_VERSION`, `POSTGIS_PROJ_VERSION` and `POSTGIS_GEOS_VERSION` variables have been set correctly.

Chapter 3

PostGIS Administration

3.1 Performance Tuning

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

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

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

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

In addition to the Postgres settings, PostGIS has some custom settings which are listed in Section 7.22.

3.1.1 Startup

These settings are configured in `postgresql.conf`:

constraint_exclusion

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

shared_buffers

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

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

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

3.1.2 Runtime

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

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

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

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

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

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

max_parallel_workers_per_gather

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

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

3.2 Configuring raster support

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

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

If you want to enable offline raster:

```
POSTGIS_ENABLE_OUTDB_RASTERS=1
```

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

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

```
POSTGIS_GDAL_ENABLED_DRIVERS=ENABLE_ALL
```

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

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

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

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

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

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

3.3 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-3.4/

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

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

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

3.4 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 "3.5.3";
ALTER EXTENSION postgis_topology UPDATE TO "3.5.3";
```

If you get an error notice something like:

```
No migration path defined for ... to 3.5.3
```

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 "3.5.3" 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 "3.5.3next";
ALTER EXTENSION postgis_topology UPDATE TO "3.5.3next";
```

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-3.5.3/*_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
All floating point comparisons within the SQL-MM implementation are performed to a specified tolerance, currently 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.

As of PostGIS 1.4 PostGIS supports compound curves in a curve polygon.

```
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('\x01010000000000000000f03f000000000000f03f', 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 with SRID
- POINTM(0 0 0) -- XYM
- POINT(0 0 0 0) -- XYZM
- SRID=4326;MULTIPOINTM(0 0 0,1 2 1) -- XYM with 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.
```

Query and measurement functions use units of meters. So distance parameters should be expressed in meters, and return values should be expected in meters (or square meters for areas).

```
-- 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?

- If your data is contained in a small area, you might find that choosing an appropriate projection and using GEOMETRY is the best solution, in terms of performance and functionality available.
- If your data is global or covers a continental region, you may find that GEOGRAPHY allows you to build a system without having to worry about projection details. You store your data in longitude/latitude, and use the functions that have been defined on GEOGRAPHY.
- If you don't understand projections, and you don't want to learn about them, and you're prepared to accept the limitations in functionality available in GEOGRAPHY, then it might be easier for you to use GEOGRAPHY than GEOMETRY. Simply load your data up as longitude/latitude and go from there.

Refer to Section 13.11 for compare between what is supported for Geography vs. Geometry. For a brief listing and description of Geography functions, refer to Section 13.4

4.3.4 Geography Advanced FAQ

1. *Do you calculate on the sphere or the spheroid?*

   By default, all distance and area calculations are done on the spheroid. You should find that the results of calculations in local areas match up will with local planar results in good local projections. Over larger areas, the spheroidal calculations will be more accurate than any calculation done on a projected plane. All the geography functions have the option of using a sphere calculation, by setting a final boolean parameter to 'FALSE'. This will somewhat speed up calculations, particularly for cases where the geometries are very simple.

2. *What about the date-line and the poles?*

   All the calculations have no conception of date-line or poles, the coordinates are spherical (longitude/latitude) so a shape that crosses the dateline is, from a calculation point of view, no different from any other shape.

3. *What is the longest arc you can process?*

   We use great circle arcs as the "interpolation line" between two points. That means any two points are actually joined up two ways, depending on which direction you travel along the great circle. All our code assumes that the points are joined by the *shorter* of the two paths along the great circle. As a consequence, shapes that have arcs of more than 180 degrees will not be correctly modelled.

4. *Why is it so slow to calculate the area of Europe / Russia / insert big geographic region here ?*

   Because the polygon is so darned huge! Big areas are bad for two reasons: their bounds are huge, so the index tends to pull the feature no matter what query you run; the number of vertices is huge, and tests (distance, containment) have to traverse the vertex list at least once and sometimes N times (with N being the number of vertices in the other candidate feature). As with GEOMETRY, we recommend that when you have very large polygons, but are doing queries in small areas, you "denormalize" your geometric data into smaller chunks so that the index can effectively subquery parts of the object and so queries don't have to pull out the whole object every time. Please consult ST_Subdivide function documentation. Just because you *can* store all of Europe in one polygon doesn't mean you *should*.

4.4 Geometry Validation

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

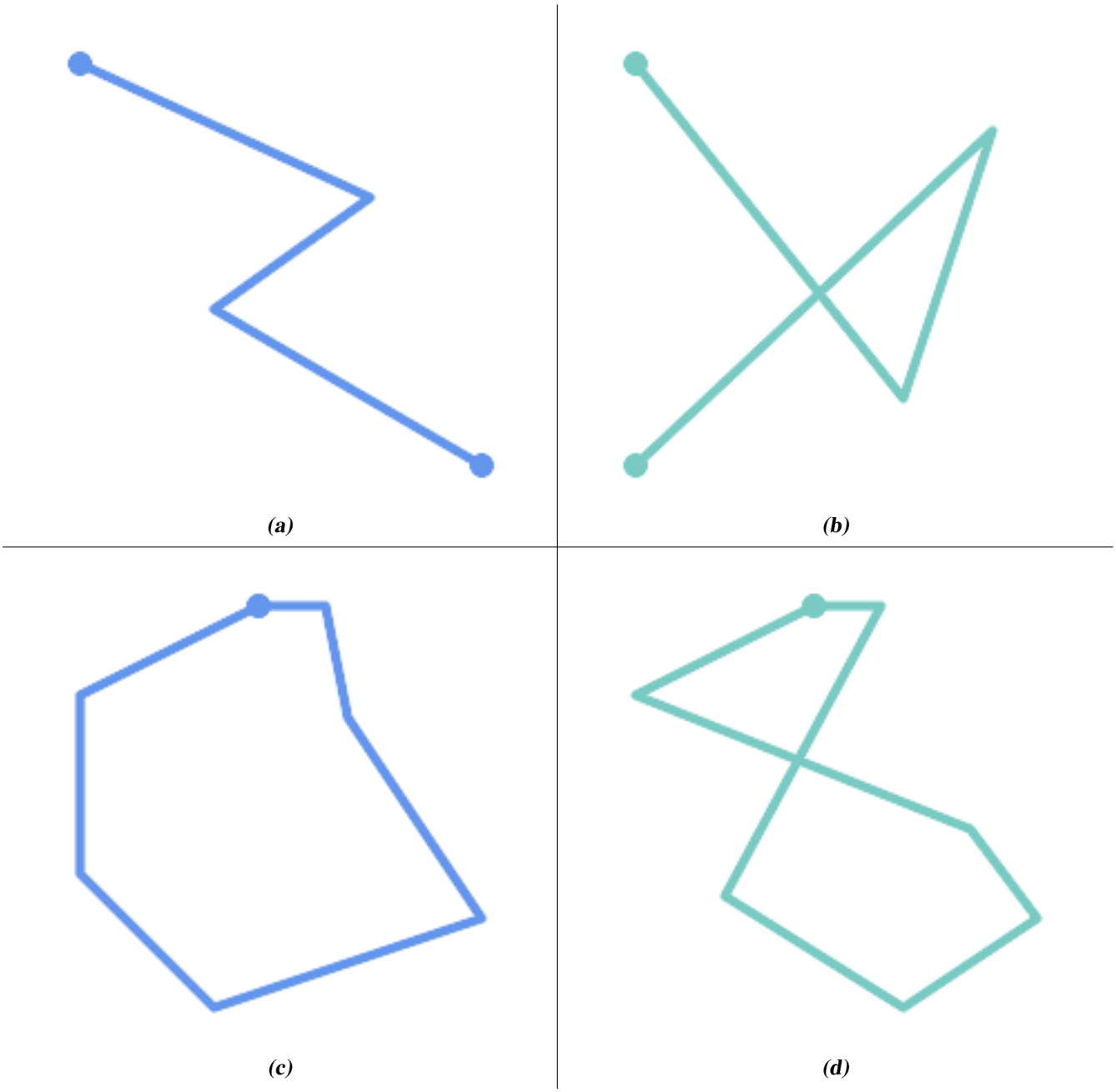
A *simple* geometry is one that has no anomalous geometric points, such as self intersection or self tangency.

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

`MULTIPOINT`s are *simple* if no two coordinates (`POINT`s) are equal (have identical coordinate values).

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.

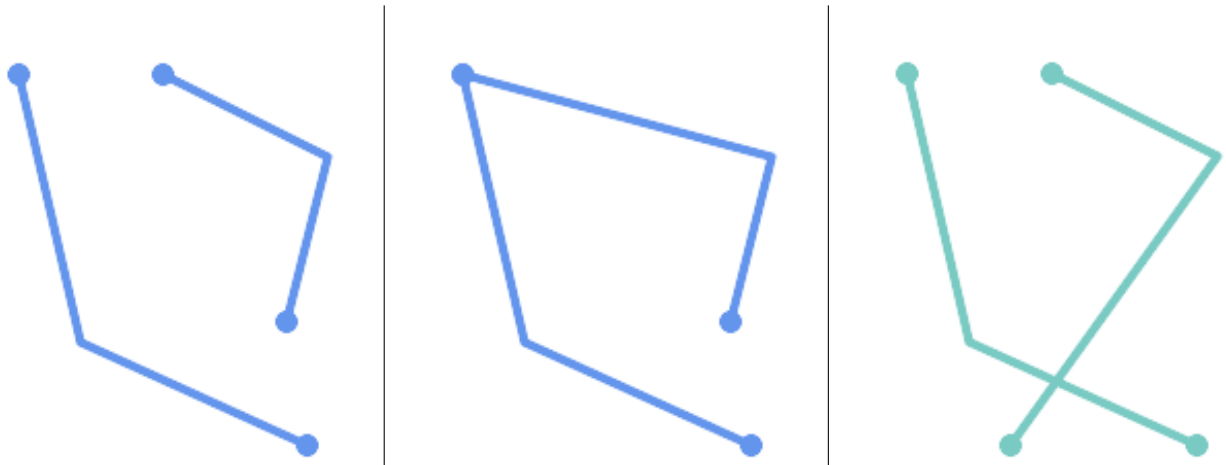
***(a)** and **(c)** are simple `LINESTRING`s. **(b)** and **(d)** are not simple. **(c)** is a closed Linear Ring.*



(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* MULTILINESTRINGs. ***(g)*** *is not simple.*



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

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

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

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

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

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

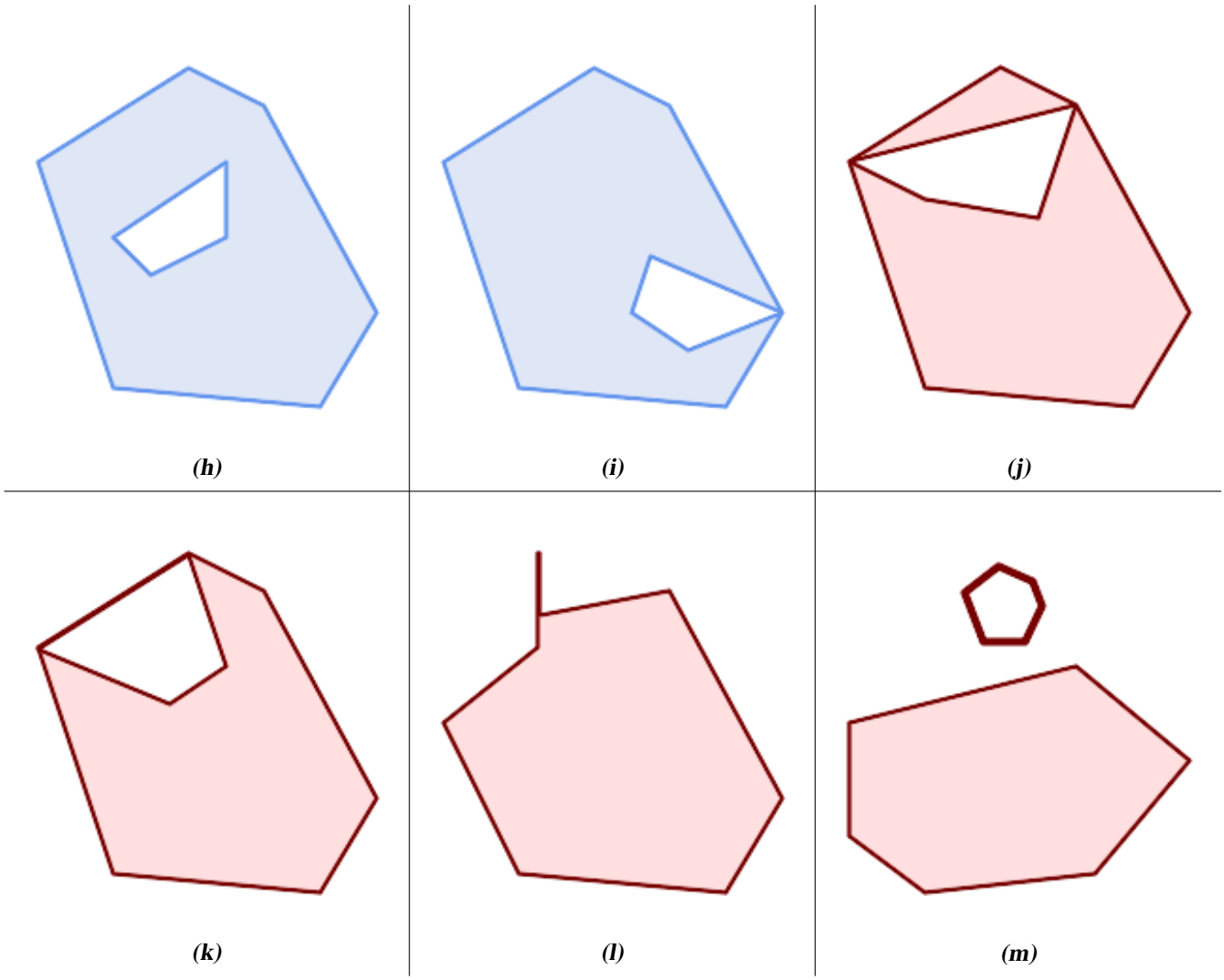
4.4.2 Valid Geometry

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

A POLYGON is *valid* if:

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

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



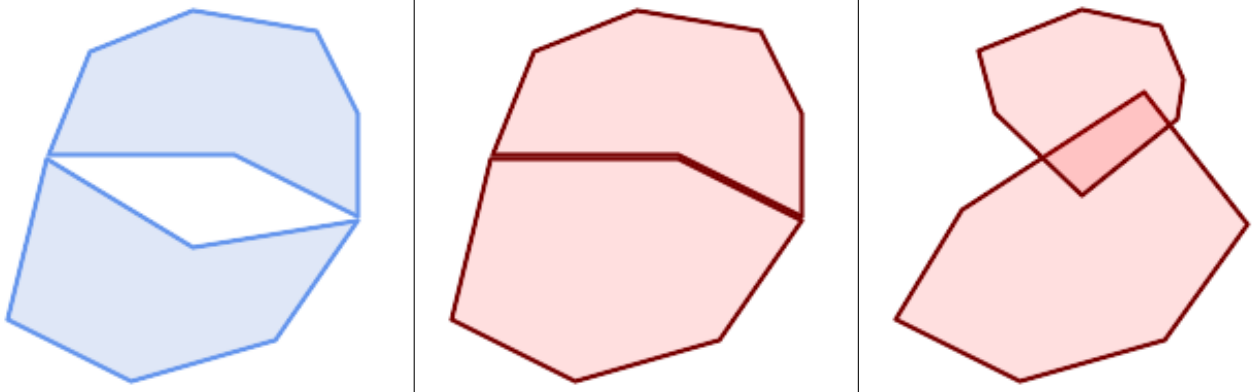
(h) ***(i)*** ***(j)***

(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 The Well-Known Text representation of the Spatial Reference System. An example of a WKT SRS representation is:

```
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 Creating a Spatial Table

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` The name of the geometry column in the feature table.

`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` The type of the spatial object. To restrict the spatial column to a single type, use one of: POINT, LINESTRING, POLYGON, MULTIPOINT, MULTILINESTRING, MULTIPOLYGON, GEOMETRYCOLLECTION or corresponding XYM versions POINTM, LINESTRINGM, POLYGONM, MULTIPOINTM, MULTILINESTRINGM, MULTIPOLYGONM, GEOMETRYCOLLECTIONM. For heterogeneous (mixed-type) collections, you can use "GEOMETRY" as the type.

4.6.3 Manually Registering Geometry Columns

Two of the cases where you may need this are the case of SQL Views and bulk inserts. For bulk insert case, you can correct the registration in the geometry_columns table by constraining the column or doing an alter table. For views, you could expose using a CAST operation. Note, if your column is typmod based, the creation process would register it correctly, so no need to do anything. Also views that have no spatial function applied to the geometry will register the same as the underlying table geometry column.

```
-- 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);
```

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

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

If we run in psql

```
\d pois_ny;
```

We observe they are defined differently -- one is typmod, one is 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)
```

In geometry_columns, they both register correctly

```
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
```

However -- if we were to create a view like this

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

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

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

```
   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) These are mutually exclusive options:

- **`-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 Only produces the table creation SQL code, without adding any actual data. This can be used if you need to completely separate the table creation and data loading steps.

-? Display help screen.

-D Use the PostgreSQL "dump" format for the output data. This can be combined with -a, -c and -d. It is much faster to load than the default "insert" SQL format. Use this for very large data sets.

-s [<FROM_SRID>:]<SRID> Creates and populates the geometry tables with the specified SRID. Optionally specifies that the input shapefile uses the given FROM_SRID, in which case the geometries will be reprojected to the target SRID.

-k Keep identifiers' case (column, schema and attributes). Note that attributes in Shapefile are all UPPERCASE.

-i Coerce all integers to standard 32-bit integers, do not create 64-bit bigints, even if the DBF header signature appears to warrant it.

-I Create a GiST index on the geometry column.

-m -m `a_file_name` Specify a file containing a set of mappings of (long) column names to 10 character DBF column names. The content of the file is one or more lines of two names separated by white space and no trailing or leading space. For example:

```
COLUMNNAME DBFFIELD1
AVERYLONGCOLUMNNAME DBFFIELD2
```

-S Generate simple geometries instead of MULTI geometries. Will only succeed if all the geometries are actually single (I.E. a MULTIPOLYGON with a single shell, or or a MULTIPOINT with a single vertex).

-t <dimensionality> Force the output geometry to have the specified dimensionality. Use the following strings to indicate the dimensionality: 2D, 3DZ, 3DM, 4D.

If the input has fewer dimensions that specified, the output will have those dimensions filled in with zeroes. If the input has more dimensions that specified, the unwanted dimensions will be stripped.

-w Output WKT format, instead of WKB. Note that this can introduce coordinate drifts due to loss of precision.

-e Execute each statement on its own, without using a transaction. This allows loading of the majority of good data when there are some bad geometries that generate errors. Note that this cannot be used with the -D flag as the "dump" format always uses a transaction.

-W <encoding> Specify encoding of the input data (dbf file). When used, all attributes of the dbf are converted from the specified encoding to UTF8. The resulting SQL output will contain a `SET CLIENT_ENCODING to UTF8` command, so that the backend will be able to reconvert from UTF8 to whatever encoding the database is configured to use internally.

-N <policy> NULL geometries handling policy (insert*,skip,abort)

-n -n Only import DBF file. If your data has no corresponding shapefile, it will automatically switch to this mode and load just the dbf. So setting this flag is only needed if you have a full shapefile set, and you only want the attribute data and no geometry.

-G Use geography type instead of geometry (requires lon/lat data) in WGS84 long lat (SRID=4326)

-T <tablespace> Specify the tablespace for the new table. Indexes will still use the default tablespace unless the -X parameter is also used. The PostgreSQL documentation has a good description on when to use custom tablespaces.

-X <tablespace> Specify the tablespace for the new table's indexes. This applies to the primary key index, and the GIST spatial index if -I is also used.

-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` This function tells whether two geometries share any space.

`=` This tests whether two geometries are geometrically identical. For example, if 'POLYGON((0 0,1 1,1 0,0 0))' is the same as 'POLYGON((0 0,1 1,1 0,0 0))' (it is).

Next, you can use these operators in queries. Note that when specifying geometries and boxes on the SQL command line, you must explicitly turn the string representations into geometries function. The 312 is a fictitious spatial reference system that matches our data. So, for example:

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

The above query would return the single record from the "ROADS_GEOM" table in which the geometry was equal to that value.

To check whether some of the roads passes in the area defined by a polygon:

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

The most common spatial query will probably be a "frame-based" query, used by client software, like data browsers and web mappers, to grab a "map frame" worth of data for display.

When using the "&&" operator, you can specify either a BOX3D as the comparison feature or a GEOMETRY. When you specify a GEOMETRY, however, its bounding box will be used for the comparison.

Using a "BOX3D" object for the frame, such a query looks like this:

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

Note the use of the SRID 312, to specify the projection of the envelope.

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>
```

The commandline options are:

`-f <filename>` Write the output to a particular filename.

`-h <host>` The database host to connect to.

`-p <port>` The port to connect to on the database host.

`-P <password>` The password to use when connecting to the database.

`-u <user>` The username to use when connecting to the database.

`-g <geometry column>` In the case of tables with multiple geometry columns, the geometry column to use when writing the shape file.

`-b` Use a binary cursor. This will make the operation faster, but will not work if any NON-geometry attribute in the table lacks a cast to text.

`-r` Raw mode. Do not drop the `gid` field, or escape column names.

`-m filename` Remap identifiers to ten character names. The content of the file is lines of two symbols separated by a single white space and no trailing or leading space: 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 Indexes

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).

The syntax for building a GiST index on a "geometry" column is as follows:

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

The above syntax will always build a 2D-index. To get the an n-dimensional index for the geometry type, you can create one using this syntax:

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

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

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

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

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

4.9.2 BRIN Indexes

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 Indexes

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.

Once a GIS data table exceeds a few thousand rows, an SP-GiST index may be used to speed up spatial searches of the data. The syntax for building an SP-GiST index on a "geometry" column is as follows:

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

The above syntax will build a 2-dimensional index. A 3-dimensional index for the geometry type can be created using the 3D operator class:

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

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

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

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

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

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

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

There is no support for kNN searches at the moment.

4.9.4 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

Spatial Queries

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

5.1 Determining Spatial Relationships

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

5.1.1 Dimensionally Extended 9-Intersection Model

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

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

Boundary

The boundary of a geometry is the set of geometries of the next lower dimension. For `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**.

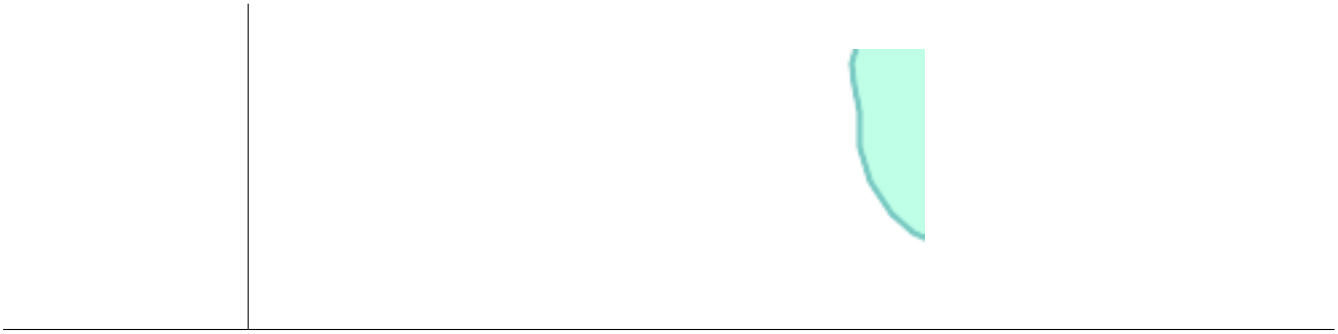
For a geometry *g* the *Interior*, *Boundary*, and *Exterior* are denoted using the notation *I(g)*, *B(g)*, and *E(g)*. Also, *dim(s)* denotes the dimension of a set *s* with the domain of `{0,1,2,F}`:

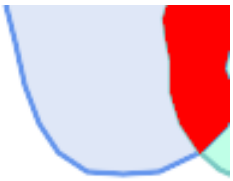
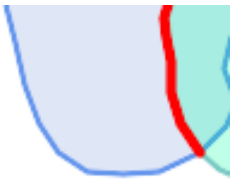
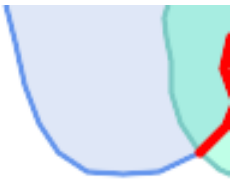
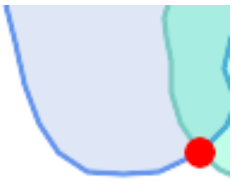
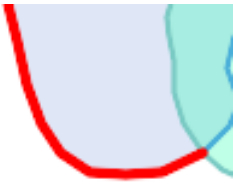
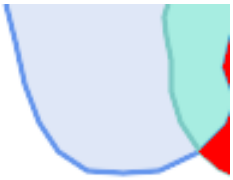
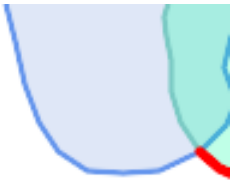
- `0` => point
- `1` => line
- `2` => area
- `F` => empty set

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:



	Interior	Boundary	Exterior
Interior	 *dim(I(a) ∩ I(b))* = **2**	 *dim(I(a) ∩ B(b)* = **1**	*dim(I(a) ∩ E(b))* = **2**
Boundary	 *dim(B(a) ∩ I(b))* = **1**	 *dim(B(a) ∩ B(b))* = **0**	 *dim(B(a) ∩ E(b))* = **1**
Exterior	 *dim(E(a) ∩ I(b))* = **2**	 *dim(E(a) ∩ B(b))* = **1**	*dim(E(a) ∩ E(b)* = **2**

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

For more information, refer to:

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

5.1.2 Named Spatial Relationships

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

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

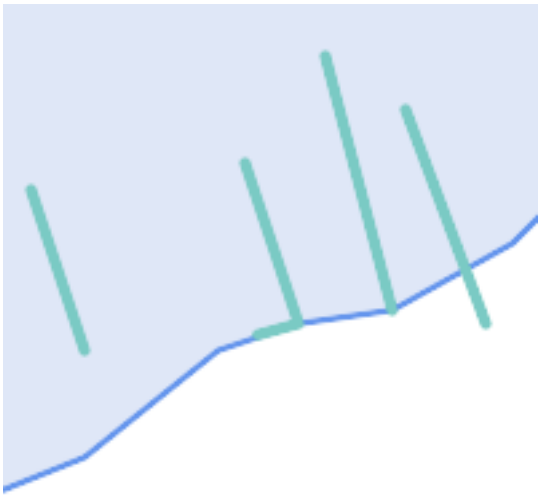
```
SELECT city.name, state.name, city.geom
FROM city JOIN state ON ST_Intersects(city.geom, state.geom);
```

For more details and illustrations, see the PostGIS Workshop.

5.1.3 General Spatial Relationships

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

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

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

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

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

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

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

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

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

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

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

Chapter 6

Performance Tips

6.1 Small tables of large geometries

6.1.1 Problem description

Current PostgreSQL versions (including 9.6) suffer from a query optimizer weakness regarding TOAST tables. TOAST tables are a kind of "extension room" used to store large (in the sense of data size) values that do not fit into normal data pages (like long texts, images or complex geometries with lots of vertices), see the PostgreSQL Documentation for TOAST for more information).

The problem appears if you happen to have a table with rather large geometries, but not too many rows of them (like a table containing the boundaries of all European countries in high resolution). Then the table itself is small, but it uses lots of TOAST space. In our example case, the table itself had about 80 rows and used only 3 data pages, but the TOAST table used 8225 pages.

Now issue a query where you use the geometry operator && to search for a bounding box that matches only very few of those rows. Now the query optimizer sees that the table has only 3 pages and 80 rows. It estimates that a sequential scan on such a small table is much faster than using an index. And so it decides to ignore the GIST index. Usually, this estimation is correct. But in our case, the && operator has to fetch every geometry from disk to compare the bounding boxes, thus reading all TOAST pages, too.

To see whether your suffer from this issue, use the "EXPLAIN ANALYZE" postgresql command. For more information and the technical details, you can read the thread on the PostgreSQL performance mailing list: http://archives.postgresql.org/pgsql-performance/2005-02/msg00030.php

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

6.1.2 Workarounds

The PostgreSQL people are trying to solve this issue by making the query estimation TOAST-aware. For now, here are two workarounds:

The first workaround is to force the query planner to use the index. Send "SET enable_seqscan TO off;" to the server before issuing the query. This basically forces the query planner to avoid sequential scans whenever possible. So it uses the GIST index as usual. But this flag has to be set on every connection, and it causes the query planner to make misestimations in other cases, so you should "SET enable_seqscan TO on;" after the query.

The second workaround is to make the sequential scan as fast as the query planner thinks. This can be achieved by creating an additional column that "caches" the bbox, and matching against this. In our example, the commands are like:

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

Now change your query to use the && operator against bbox instead of geom_column, like:

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

Of course, if you change or add rows to mytable, you have to keep the bbox "in sync". The most transparent way to do this would be triggers, but you also can modify your application to keep the bbox column current or run the UPDATE query above after every modification.

6.2 CLUSTERing on geometry indices

For tables that are mostly read-only, and where a single index is used for the majority of queries, PostgreSQL offers the CLUSTER command. This command physically reorders all the data rows in the same order as the index criteria, yielding two performance advantages: First, for index range scans, the number of seeks on the data table is drastically reduced. Second, if your working set concentrates to some small intervals on the indices, you have a more efficient caching because the data rows are spread along fewer data pages. (Feel invited to read the CLUSTER command documentation from the PostgreSQL manual at this point.)

However, currently PostgreSQL does not allow clustering on PostGIS GIST indices because GIST indices simply ignores NULL values, you get an error message like:

```
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.
```

As the HINT message tells you, one can work around this deficiency by adding a "not null" constraint to the table:

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

Of course, this will not work if you in fact need NULL values in your geometry column. Additionally, you must use the above method to add the constraint, using a CHECK constraint like "ALTER TABLE blubb ADD CHECK (geometry is not null);" will not work.

6.3 Avoiding dimension conversion

Sometimes, you happen to have 3D or 4D data in your table, but always access it using OpenGIS compliant ST_AsText() or ST_AsBinary() functions that only output 2D geometries. They do this by internally calling the ST_Force2D() function, which introduces a significant overhead for large geometries. To avoid this overhead, it may be feasible to pre-drop those additional dimensions once and forever:

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

Note that if you added your geometry column using AddGeometryColumn() there'll be a constraint on geometry dimension. To bypass it you will need to drop the constraint. Remember to update the entry in the geometry_columns table and recreate the constraint afterwards.

In case of large tables, it may be wise to divide this UPDATE into smaller portions by constraining the UPDATE to a part of the table via a WHERE clause and your primary key or another feasible criteria, and running a simple "VACUUM;" between your UPDATEs. This drastically reduces the need for temporary disk space. Additionally, if you have mixed dimension geometries, restricting the UPDATE by "WHERE dimension(geom)>2" skips re-writing of geometries that already are in 2D.

Chapter 7

PostGIS Reference

The functions given below are the ones which a user of PostGIS is likely to need. There are other functions which are required support functions to the PostGIS objects which are not of use to a general user.

Note
PostGIS has begun a transition from the existing naming convention to an SQL-MM-centric convention. As a result, most of the functions that you know and love have been renamed using the standard spatial type (ST) prefix. Previous functions are still available, though are not listed in this document where updated functions are equivalent. The non ST_ functions not listed in this documentation are deprecated and will be removed in a future release so STOP USING THEM.

7.1 PostGIS Geometry/Geography/Box Data Types

7.1.1 box2d

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

Description

`box2d` is a spatial data type used to represent the two-dimensional bounding box enclosing a geometry or collection of geometries. For example, the ST_Extent aggregate function returns a `box2d` object.

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)`.

Casting Behavior

This table lists the automatic and explicit casts allowed for this data type:

Cast To	Behavior
box3d	automatic
geometry	automatic

See Also

Section 13.7

7.1.2 box3d

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

Description

`box3d` is a PostGIS spatial data type used to represent the three-dimensional bounding box enclosing a geometry or collection of geometries. For example, the ST_3DExtent aggregate function returns a `box3d` object.

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

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

Casting Behavior

This table lists the automatic and explicit casts allowed for this data type:

Cast To	Behavior
box	automatic
box2d	automatic
geometry	automatic

See Also

Section 13.7

7.1.3 geometry

geometry — The type representing spatial features with planar coordinate systems.

Description

`geometry` is a fundamental PostGIS spatial data type used to represent a feature in planar (Euclidean) coordinate systems.

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

Casting Behavior

This table lists the automatic and explicit casts allowed for this data type:

Cast To	Behavior
box	automatic
box2d	automatic
box3d	automatic
bytea	automatic
geography	automatic
text	automatic

See Also

Section 4.1, Section 13.3

7.1.4 geometry_dump

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

Description

`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.

See Also

Section 13.6

7.1.5 geography

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

Description

`geography` is a spatial data type used to represent a feature in geodetic coordinate systems. Geodetic coordinate systems model the earth using an ellipsoid.

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

Casting Behavior

This table lists the automatic and explicit casts allowed for this data type:

Cast To	Behavior
geometry	explicit

See Also

Section 4.3, Section 13.4

7.2 Table Management Functions

7.2.1 AddGeometryColumn

AddGeometryColumn — Adds a geometry column to an existing table.

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);

Description

Adds a geometry column to an existing table of attributes. The `schema_name` is the name of the table schema. The `srid` must be an integer value reference to an entry in the SPATIAL_REF_SYS table. The `type` must be a string corresponding to the geometry type, eg, 'POLYGON' or 'MULTILINESTRING' . An error is thrown if the schemaname doesn't exist (or not visible in the current search_path) or the specified SRID, geometry type, or dimension is invalid.

Note
Changed: 2.0.0 This function no longer updates geometry_columns since geometry_columns is a view that reads from system catalogs. It by default also does not create constraints, but instead uses the built in type modifier behavior of PostgreSQL. So for example building a wgs84 POINT column with this function is now equivalent to: `ALTER TABLE some_table ADD COLUMN geom geometry(Point,4326);`
Changed: 2.0.0 If you require the old behavior of constraints use the default `use_typmod`, but set it to false.

Note
Changed: 2.0.0 Views can no longer be manually registered in geometry_columns, however views built against geometry typmod tables geometries and used without wrapper functions will register themselves correctly because they inherit the typmod behavior of their parent table column. Views that use geometry functions that output other geometries will need to be cast to typmod geometries for these view geometry columns to be registered correctly in geometry_columns. Refer to 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.

Enhanced: 2.0.0 use_typmod argument introduced. Defaults to creating typmod geometry column instead of constraint-based.

Examples

```
-- 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
```

See Also

DropGeometryColumn, DropGeometryTable, Section 4.6.2, Section 4.6.3

7.2.2 DropGeometryColumn

DropGeometryColumn — Removes a geometry column from a spatial table.

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);

Description

Removes a geometry column from a spatial table. Note that schema_name will need to match the f_table_schema field of the table's row in the geometry_columns table.

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
Changed: 2.0.0 This function is provided for backward compatibility. Now that since geometry_columns is now a view against the system catalogs, you can drop a geometry column like any other table column using `ALTER TABLE`

Examples

```
      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;
```

See Also

AddGeometryColumn, DropGeometryTable, Section 4.6.2

7.2.3 DropGeometryTable

DropGeometryTable — Drops a table and all its references in geometry_columns.

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);

Description

Drops a table and all its references in geometry_columns. Note: uses current_schema() on schema-aware pgsql installations if schema is not provided.

Note
Changed: 2.0.0 This function is provided for backward compatibility. Now that since geometry_columns is now a view against the system catalogs, you can drop a table with geometry columns like any other table using `DROP TABLE`

Examples

```
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;
```

See Also

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);

Description

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.

Examples

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

See Also

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);

Description

Ensures geometry columns have appropriate type modifiers or spatial constraints to ensure they are registered correctly in the geometry_columns view. By default will convert all geometry columns with no type modifier to ones with type modifiers.

For backwards compatibility and for spatial needs such as table inheritance where each child table may have different geometry type, the old check constraint behavior is still supported. If you need the old behavior, you need to pass in the new optional argument as false use_typmod=false. When this is done geometry columns will be created with no type modifiers but will have 3 constraints defined. In particular, this means that every geometry column belonging to a table has at least three constraints:

- enforce_dims_geom - ensures every geometry has the same dimension (see ST_NDims)
- enforce_geotype_geom - ensures every geometry is of the same type (see GeometryType)
- enforce_srid_geom - ensures every geometry is in the same projection (see ST_SRID)

If a table oid is provided, this function tries to determine the srid, dimension, and geometry type of all geometry columns in the table, adding constraints as necessary. If successful, an appropriate row is inserted into the geometry_columns table, otherwise, the exception is caught and an error notice is raised describing the problem.

If the oid of a view is provided, as with a table oid, this function tries to determine the srid, dimension, and type of all the geometries in the view, inserting appropriate entries into the geometry_columns table, but nothing is done to enforce constraints.

The parameterless variant is a simple wrapper for the parameterized variant that first truncates and repopulates the geometry_columns table for every spatial table and view in the database, adding spatial constraints to tables where appropriate. It returns a summary of the number of geometry columns detected in the database and the number that were inserted into the geometry_columns table. The parameterized version simply returns the number of rows inserted into the geometry_columns table.

Availability: 1.4.0

Changed: 2.0.0 By default, now uses type modifiers instead of check constraints to constrain geometry types. You can still use check constraint behavior instead by using the new use_typmod and setting it to false.

Enhanced: 2.0.0 use_typmod optional argument was introduced that allows controlling if columns are created with typmodifiers or with check constraints.

Examples

```
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);

Description

Updates the SRID of all features in a geometry column, updating constraints and reference in geometry_columns. If the column was enforced by a type definition, the type definition will be changed. Note: uses current_schema() on schema-aware pgsql installations if schema is not provided.

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

This method supports Circular Strings and Curves.

Examples

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)
\.
```

This will change the srid of the roads table to 4326 from whatever it was before:

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

The prior example is equivalent to this DDL statement:

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

If you got the projection wrong (or brought it in as unknown) in load and you wanted to transform to web mercator all in one shot you can do this with DDL but there is no equivalent PostGIS management function to do so in one go.

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

See Also

UpdateRasterSRID, ST_SetSRID, ST_Transform, ST_GeomFromEWKT

7.3 Geometry Constructors

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);

Description

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.

Availability: 1.4.0 - ST_Collect(geomarray) was introduced. ST_Collect was enhanced to handle more geometries faster.

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

This method supports Circular Strings and Curves.

Examples - Two-input variant

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))
```

Examples - Array variant

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))
```

Examples - Aggregate variant

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
```

See Also

ST_Dump, ST_Union

7.3.2 ST_LineFromMultiPoint

ST_LineFromMultiPoint — Creates a LineString from a MultiPoint geometry.

Synopsis

geometry **ST_LineFromMultiPoint**(geometry aMultiPoint);

Description

Creates a LineString from a MultiPoint geometry.

Use ST_MakeLine to create lines from Point or LineString inputs.

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

Examples

Create a 3D line string from a 3D MultiPoint

```
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)
```

See Also

ST_AsEWKT, ST_MakeLine

7.3.3 ST_MakeEnvelope

ST_MakeEnvelope — Creates a rectangular Polygon from minimum and maximum coordinates.

Synopsis

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

Description

Creates a rectangular Polygon from the minimum and maximum values for X and Y. Input values must be in the spatial reference system specified by the SRID. If no SRID is specified the unknown spatial reference system (SRID 0) is used.

Availability: 1.5

Enhanced: 2.0: Ability to specify an envelope without specifying an SRID was introduced.

Example: Building a bounding box polygon

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

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

See Also

ST_MakePoint, ST_MakeLine, ST_MakePolygon, ST_TileEnvelope

7.3.4 ST_MakeLine

ST_MakeLine — Creates a LineString from Point, MultiPoint, or LineString geometries.

Synopsis

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

Description

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.

Availability: 2.3.0 - Support for MultiPoint input elements was introduced

Availability: 2.0.0 - Support for LineString input elements was introduced

Availability: 1.4.0 - ST_MakeLine(geomarray) was introduced. ST_MakeLine aggregate functions was enhanced to handle more points faster.

Examples: Two-input variant

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)
```

Create a 3D line from two 3D points.

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

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

Create a line from two disjoint LineStrings.

```
 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)
```

Examples: Array variant

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)
```

Examples: Aggregate variant

This example queries time-based sequences of GPS points from a set of tracks and creates one record for each track. The result geometries are LineStrings composed of the GPS track points in the order of travel.

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;
```

See Also

ST_RemoveRepeatedPoints, ST_AsEWKT, ST_AsText, ST_GeomFromText, ST_MakePoint, ST_Point

7.3.5 ST_MakePoint

ST_MakePoint — Creates a 2D, 3DZ or 4D Point.

Synopsis

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

geometry **ST_MakePoint**(float x, float y, float z);

geometry **ST_MakePoint**(float x, float y, float z, float m);

Description

Creates a 2D XY, 3D XYZ or 4D XYZM Point geometry. Use ST_MakePointM to make points with XYM coordinates.

Use ST_SetSRID to specify a SRID for the created point.

While not OGC-compliant, `ST_MakePoint` is faster and more precise than ST_GeomFromText and ST_PointFromText. It is also easier to use for numeric coordinate values.

Note
For geodetic coordinates, `X` is longitude and `Y` is latitude

Note
The functions ST_Point, ST_PointZ, ST_PointM, and ST_PointZM can be used to create points with a given SRID.

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

Examples

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

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

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

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

See Also

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

7.3.6 ST_MakePointM

ST_MakePointM — Creates a Point from X, Y and M values.

Synopsis

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

Description

Creates a point with X, Y and M (measure) ordinates. Use ST_MakePoint to make points with XY, XYZ, or XYZM coordinates. Use ST_SetSRID to specify a SRID for the created point.

Note
For geodetic coordinates, `X` is longitude and `Y` is latitude

Note
The functions ST_PointM, and ST_PointZM can be used to create points with an M value and a given SRID.

Examples

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)
```

Create point with a measure in the WGS 84 geodetic coordinate system.

```
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
```

See Also

ST_MakePoint, ST_SetSRID, ST_PointM, ST_PointZM

7.3.7 ST_MakePolygon

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

Synopsis

geometry **ST_MakePolygon**(geometry linestring);

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

Description

Creates a Polygon formed by the given shell and optional array of holes. Input geometries must be closed LineStrings (rings).

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
This function does not accept MultiLineStrings. Use ST_LineMerge to generate a LineString, or ST_Dump to extract LineStrings.

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

Examples: Single input variant

Create a Polygon from a 2D LineString.

```
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;
```

Create a Polygon from a 3D LineString

```
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))
```

Examples: Outer shell with inner holes variant

Create a donut Polygon with an extra hole

```
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

The CASE construct is used because passing a null array into ST_MakePolygon results in a NULL return value.

```
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;
```

See Also

ST_BuildArea ST_Polygon

7.3.8 ST_Point

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

Synopsis

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

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

Description

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

Examples: Geometry

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

Creating a point with SRID specified:

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

Alternative way of specifying SRID:

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

Examples: Geography

Create geography points using the `::` cast syntax:

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

Pre-PostGIS 3.2 code, using `CAST`:

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

If the point coordinates are not in a geodetic coordinate system (such as WGS84), then they must be reprojected before casting to a geography. In this example a point in Pennsylvania State Plane feet (SRID 2273) is projected to WGS84 (SRID 4326).

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

See Also

ST_MakePoint, ST_PointZ, ST_PointM, ST_PointZM, ST_SetSRID, ST_Transform

7.3.9 ST_PointZ

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

Synopsis

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

Description

Returns an Point with the given X, Y and Z coordinate values, and optionally an SRID number.

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.

Examples

```
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)
```

See Also

ST_MakePoint, ST_Point, ST_PointM, ST_PointZM

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);

Description

Returns an Point with the given X, Y and M coordinate values, and optionally an SRID number.

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.

Examples

```
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)
```

See Also

ST_MakePoint, ST_Point, ST_PointZ, ST_PointZM

7.3.11 ST_PointZM

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

Synopsis

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

Description

Returns an Point with the given X, Y, Z and M coordinate values, and optionally an SRID number.

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.

Examples

```
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)
```

See Also

ST_MakePoint, ST_Point, ST_PointM, ST_PointZ, ST_SetSRID

7.3.12 ST_Polygon

ST_Polygon — Creates a Polygon from a LineString with a specified SRID.

Synopsis

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

Description

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.

To create polygons with holes use ST_MakePolygon Variant 2 and then ST_SetSRID.

Note
This function does not accept MultiLineStrings. Use ST_LineMerge to generate a LineString, or ST_Dump to extract LineStrings.

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.

Examples

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))
```

See Also

ST_AsEWKT, ST_AsText, ST_GeomFromEWKT, ST_GeomFromText, ST_LineMerge, ST_MakePolygon

7.3.13 ST_TileEnvelope

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

Synopsis

geometry **ST_TileEnvelope**(integer tileZoom, integer tileX, integer tileY, geometry bounds=SRID=3857;LINESTRING(-20037508.342 -20037508.342789,20037508.342789 20037508.342789), float margin=0.0);

Description

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

By default, the tile envelope is in the Web Mercator coordinate system (SRID:3857) using the standard range of the Web Mercator system (-20037508.342789, 20037508.342789). This is the most common coordinate system used for MVT tiles. The optional `bounds` parameter can be used to generate tiles in any coordinate system. It is a geometry that has the SRID and extent of the "Zoom Level zero" square within which the XYZ tile system is inscribed.

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

Enhanced: 3.1.0 Added margin parameter.

Availability: 3.0.0

Example: Building a tile envelope

```
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))
```

See Also

ST_MakeEnvelope

7.3.14 ST_HexagonGrid

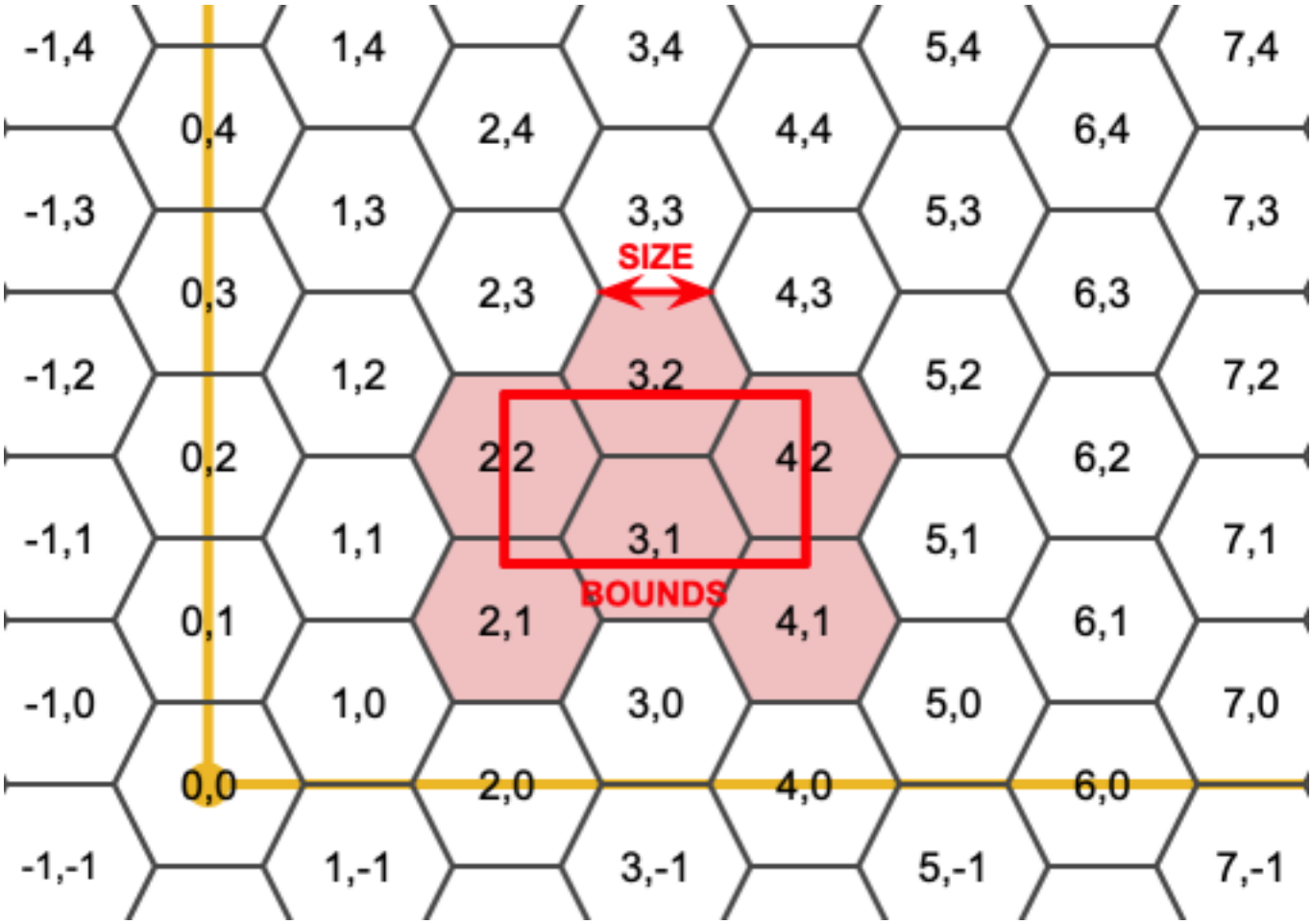
ST_HexagonGrid — Returns a set of hexagons and cell indices that completely cover the bounds of the geometry argument.

Synopsis

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

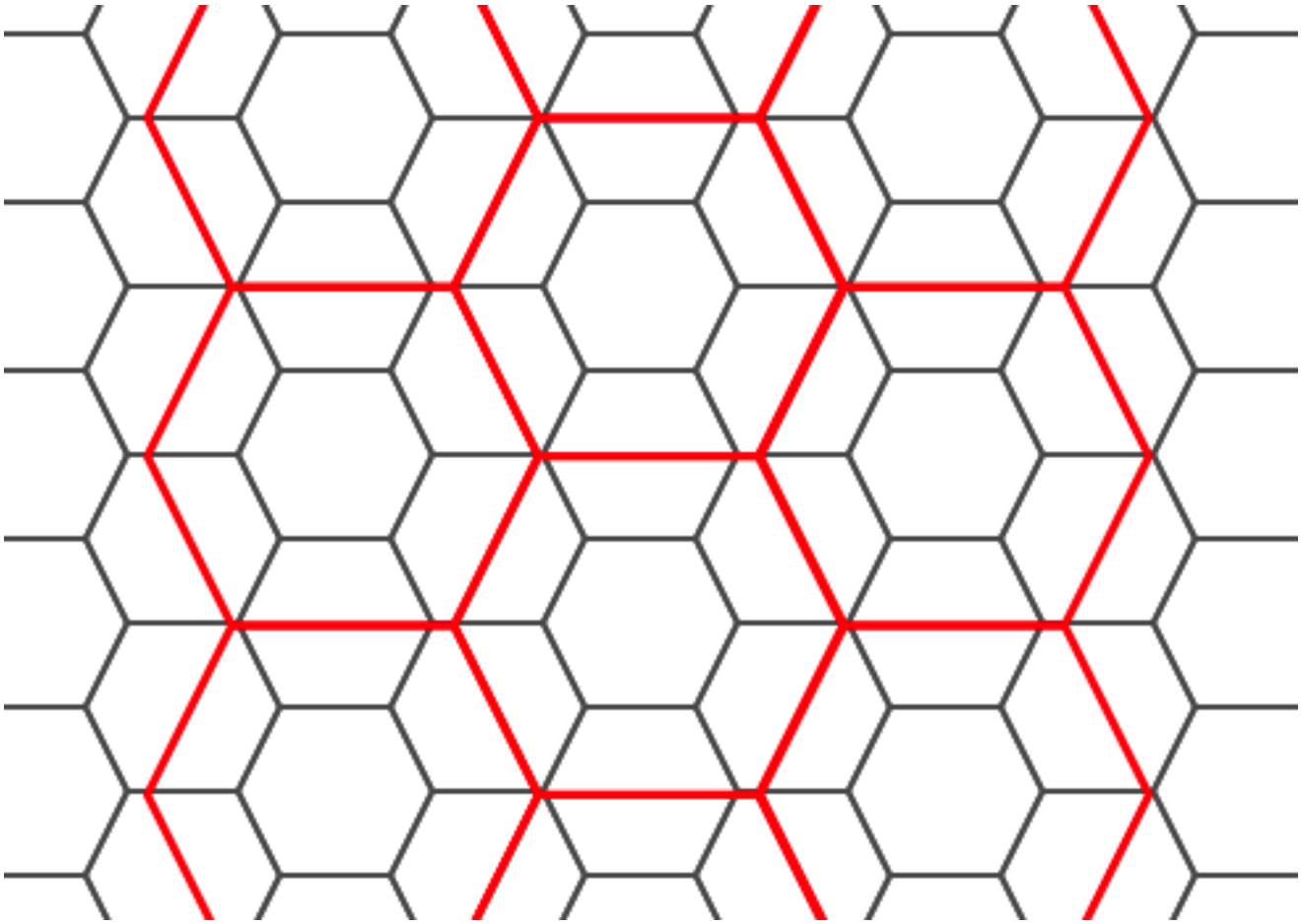
Description

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.



Availability: 3.1.0

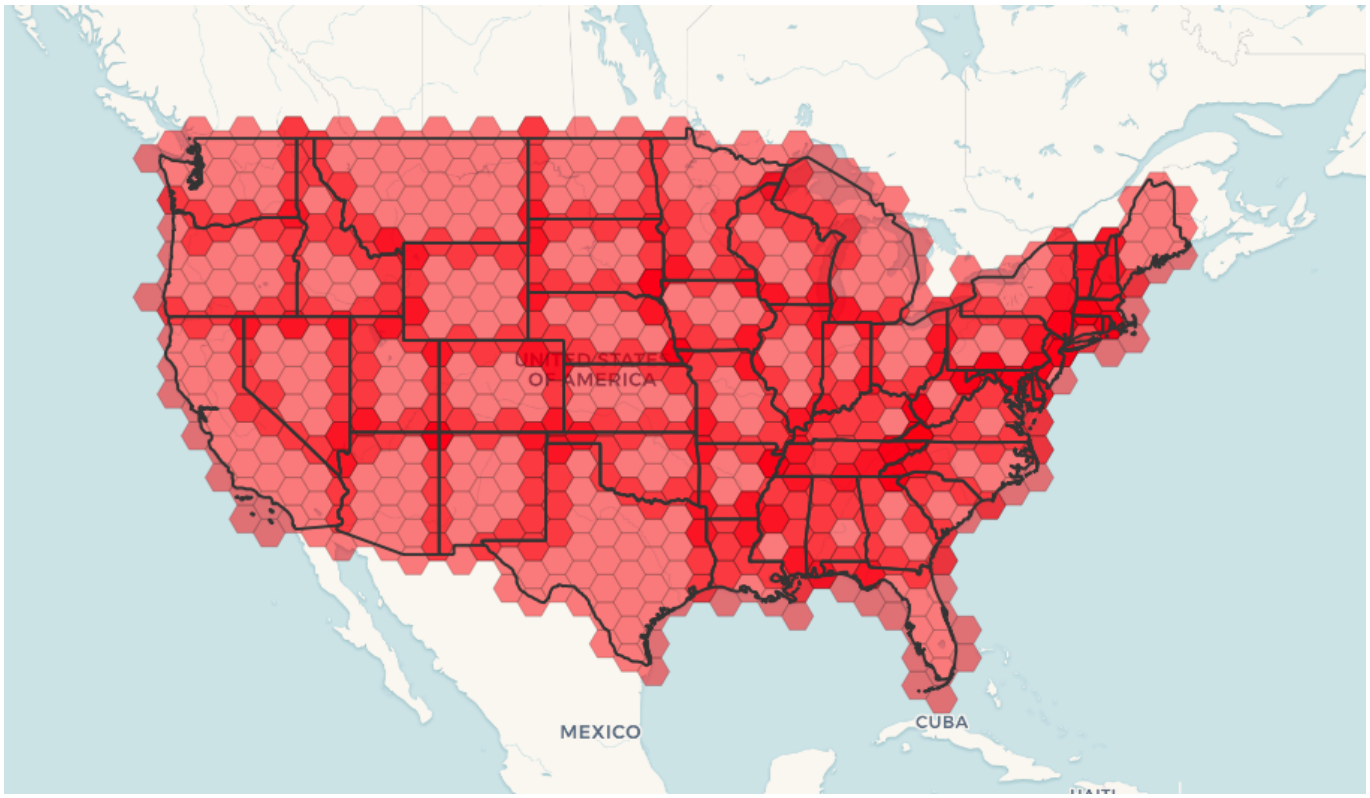
Example: Counting points in hexagons

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;
```

Example: Generating hex coverage of polygons

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)
```

See Also

ST_EstimatedExtent, ST_SetSRID, ST_SquareGrid, ST_TileEnvelope

7.3.15 ST_Hexagon

ST_Hexagon — Returns a single hexagon, using the provided edge size and cell coordinate within the hexagon grid space.

Synopsis

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

Description

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.

Availability: 3.1.0

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))
```

See Also

ST_TileEnvelope, ST_HexagonGrid, ST_Square

7.3.16 ST_SquareGrid

ST_SquareGrid — Returns a set of grid squares and cell indices that completely cover the bounds of the geometry argument.

Synopsis

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

Description

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.

Availability: 3.1.0

Example: Generating a 1 degree grid for a country

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
```

See Also

ST_TileEnvelope, ST_HexagonGrid , ST_EstimatedExtent , ST_SetSRID

7.3.17 ST_Square

ST_Square — Returns a single square, using the provided edge size and cell coordinate within the square grid space.

Synopsis

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

Description

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.

Availability: 3.1.0

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))
```

See Also

ST_TileEnvelope, ST_SquareGrid, ST_Hexagon

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);

Description

Uses a built-in font to render out a string as a multipolygon geometry. The default text height is 100.0, the distance from the bottom of a descender to the top of a capital. The default start position places the start of the baseline at the origin. Over-riding the font involves passing in a json map, with a character as the key, and base64 encoded TWKB for the font shape, with the fonts having a height of 1000 units from the bottom of the descenders to the tops of the capitals.

The text is generated at the origin by default, so to reposition and resize the text, first apply the `ST_Scale` function and then apply the `ST_Translate` function.

Availability: 3.3.0

Example: Generating the word 'Yo'

```
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);
```

See Also

ST_AsTWKB, ST_Scale, ST_Translate

7.4 Geometry Accessors

7.4.1 GeometryType

GeometryType — Returns the type of a geometry as text.

Synopsis

text **GeometryType**(geometry geomA);

Description

Returns the type of the geometry as a string. Eg: 'LINESTRING', 'POLYGON', 'MULTIPOINT', etc.

OGC SPEC s2.1.1.1 - Returns the name of the instantiable subtype of Geometry of which this Geometry instance is a member. The name of the instantiable subtype of Geometry is returned as a string.

Note
This function also indicates if the geometry is measured, by returning a string of the form 'POINTM'.

Enhanced: 2.0.0 support for Polyhedral surfaces, Triangles and TIN was introduced.

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).

Examples

```
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
```

See Also

ST_GeometryType

7.4.2 ST_Boundary

ST_Boundary — Returns the boundary of a geometry.

Synopsis

geometry **ST_Boundary**(geometry geomA);

Description

Returns the closure of the combinatorial boundary of this Geometry. The combinatorial boundary is defined as described in section 3.12.3.2 of the OGC SPEC. Because the result of this function is a closure, and hence topologically closed, the resulting boundary can be represented using representational geometry primitives as discussed in the OGC SPEC, section 3.12.2.

Performed by the GEOS module

Note
Prior to 2.0.0, this function throws an exception if used with `GEOMETRYCOLLECTION`. From 2.0.0 up it will return NULL instead (unsupported input).

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.

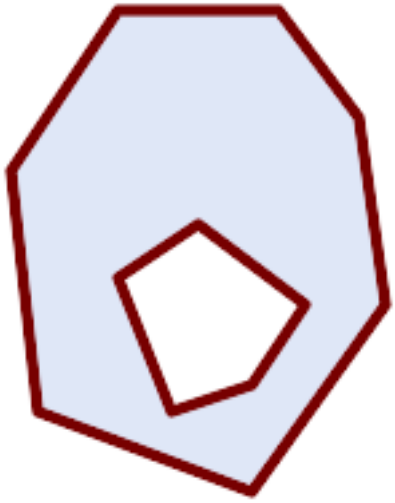
Enhanced: 2.1.0 support for Triangle was introduced

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

Examples

Linestring with boundary points overlaid

```
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 holes with boundary 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))
```

See Also

ST_AsText, ST_ExteriorRing, ST_MakePolygon

7.4.3 ST_BoundingDiagonal

ST_BoundingDiagonal — Returns the diagonal of a geometry's bounding box.

Synopsis

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

Description

Returns the diagonal of the supplied geometry's bounding box as a LineString. The diagonal is a 2-point LineString with the minimum values of each dimension in its start point and the maximum values in its end point. If the input geometry is empty, the diagonal line is a LINESTRING EMPTY.

The optional `fits` parameter specifies if the best fit is needed. If false, the diagonal of a somewhat larger bounding box can be accepted (which is faster to compute for geometries with many vertices). In either case, the bounding box of the returned diagonal line always covers the input geometry.

The returned geometry retains the SRID and dimensionality (Z and M presence) of the input geometry.

Note
In degenerate cases (i.e. a single vertex in input) the returned linestring will be formally invalid (no interior). The result is still topologically valid.

Availability: 2.2.0

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

This function supports M coordinates.

Examples

```
-- 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
```

See Also

ST_StartPoint, ST_EndPoint, ST_X, ST_Y, ST_Z, ST_M, ST_Envelope

7.4.4 ST_CoordDim

ST_CoordDim — Return the coordinate dimension of a geometry.

Synopsis

integer **ST_CoordDim**(geometry geomA);

Description

Return the coordinate dimension of the ST_Geometry value.

This is the MM compliant alias name for 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).

Examples

```
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
```

See Also

ST_NDims

7.4.5 ST_Dimension

ST_Dimension — Returns the topological dimension of a geometry.

Synopsis

integer **ST_Dimension**(geometry g);

Description

Return the topological dimension of this Geometry object, which must be less than or equal to the coordinate dimension. OGC SPEC s2.1.1.1 - returns 0 for `POINT`, 1 for `LINESTRING`, 2 for `POLYGON`, and the largest dimension of the components of a `GEOMETRYCOLLECTION`. If the dimension is unknown (e.g. for an empty `GEOMETRYCOLLECTION`) 0 is returned.

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

Enhanced: 2.0.0 support for Polyhedral surfaces and TINs was introduced. No longer throws an exception if given empty geometry.

Note
Prior to 2.0.0, this function throws an exception if used with empty geometry.

This function supports Polyhedral surfaces.

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

Examples

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

See Also

ST_NDims

7.4.6 ST_Dump

ST_Dump — Returns a set of `geometry_dump` rows for the components of a geometry.

Synopsis

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

Description

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.

Enhanced: 2.0.0 support for Polyhedral surfaces, Triangles and TIN was introduced.

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

Note

Prior to 1.3.4, this function crashes if used with geometries that contain CURVES. This is fixed in 1.3.4+

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 Examples

```
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)
```

Polyhedral Surfaces, TIN and Triangle Examples

```
-- 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))
```

See Also

geometry_dump, Section 13.6, ST_Collect, ST_GeometryN

7.4.7 ST_DumpPoints

ST_DumpPoints — Returns a set of `geometry_dump` rows for the coordinates in a geometry.

Synopsis

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

Description

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.

Enhanced: 2.0.0 support for Polyhedral surfaces, Triangles and TIN was introduced.

Availability: 1.5.0

This method supports Circular Strings and Curves.

This function supports Polyhedral surfaces.

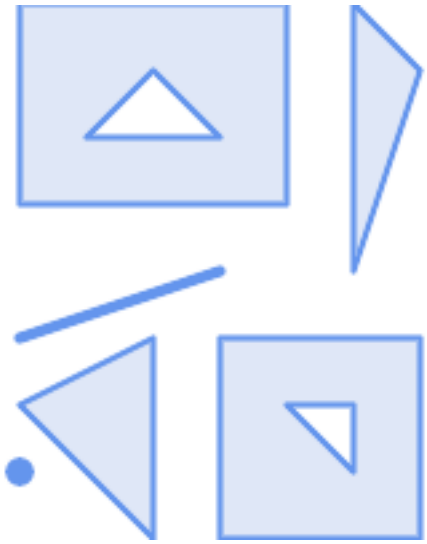
This function supports Triangles and Triangulated Irregular Network Surfaces (TIN).

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

Classic Explode a Table of LineStrings into nodes

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

Standard Geometry Examples



```
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)
```

Polyhedral Surfaces, TIN and Triangle Examples

```
-- 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)
```

See Also

geometry_dump, Section 13.6, ST_Dump, ST_DumpRings, ST_Points

7.4.8 ST_DumpSegments

ST_DumpSegments — Returns a set of `geometry_dump` rows for the segments in a geometry.

Synopsis

geometry_dump[] **ST_DumpSegments**(geometry geom);

Description

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

- the *geom* field `LINESTRING`s represent the linear segments of the supplied geometry, while the `CIRCULARSTRING`s represent the arc segments.
- the *path* field (an `integer[]`) is an index enumerating the segment start point positions in the elements of the supplied geometry. The indices are 1-based. For example, for a `LINESTRING` the paths are `{i}` where `i` is the `nth` segment start point in the `LINESTRING`. For a `POLYGON` the paths are `{i,j}` where `i` is the ring number (1 is outer; inner rings follow) and `j` is the segment start point position in the ring.

Availability: 3.2.0

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

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

Standard Geometry Examples

```
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)
```

TIN and Triangle Examples

```
-- 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)
```

See Also

geometry_dump, Section 13.6, ST_Dump, ST_DumpRings

7.4.9 ST_DumpRings

ST_DumpRings — Returns a set of `geometry_dump` rows for the exterior and interior rings of a Polygon.

Synopsis

geometry_dump[] **ST_DumpRings**(geometry a_polygon);

Description

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
This only works for POLYGON geometries. It does not work for MULTIPOLYGONS

Availability: PostGIS 1.1.3. Requires PostgreSQL 7.3 or higher.

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

Examples

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))
```

See Also

geometry_dump, Section 13.6, ST_Dump, ST_ExteriorRing, ST_InteriorRingN

7.4.10 ST_EndPoint

ST_EndPoint — Returns the last point of a LineString or CircularLineString.

Synopsis

geometry **ST_EndPoint**(geometry g);

Description

Returns the last point of a `LINESTRING` or `CIRCULARLINESTRING` geometry as a `POINT`. Returns `NULL` if the input is not a `LINESTRING` or `CIRCULARLINESTRING`.

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
Changed: 2.0.0 no longer works with single geometry MultiLineStrings. In older versions of PostGIS a single-line MultiLineString would work with this function and return the end point. In 2.0.0 it returns NULL like any other MultiLineString. The old behavior was an undocumented feature, but people who assumed they had their data stored as LINESTRING may experience these returning NULL in 2.0.0.

Examples

End point of a LineString

```
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
```

End point of a 3D LineString

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

End point of a CircularString

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

See Also

ST_PointN, ST_StartPoint

7.4.11 ST_Envelope

ST_Envelope — Returns a geometry representing the bounding box of a geometry.

Synopsis

geometry **ST_Envelope**(geometry g1);

Description

Returns the double-precision (float8) minimum bounding box for the supplied geometry, as a geometry. The polygon is defined by the corner points of the bounding box ((MINX, MINY), (MINX, MAXY), (MAXX, MAXY), (MAXX, MINY), (MINX, MINY)). (PostGIS will add a ZMIN/ZMAX coordinate as well).

Degenerate cases (vertical lines, points) will return a geometry of lower dimension than POLYGON, ie. POINT or LINESTRING.

Availability: 1.5.0 behavior changed to output double precision instead of float4

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

Examples

```
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;
```

Envelope of a point and linestring.

```
SELECT ST_AsText(ST_Envelope(
    ST_Collect(
      ST_GeomFromText('LINESTRING(55 75,125 150)'),
        ST_Point(20, 80))
```

```
        )) As wktenv;
wktenv
-----------
POLYGON((20 75,20 150,125 150,125 75,20 75))
```

See Also

Box2D, Box3D, ST_OrientedEnvelope

7.4.12 ST_ExteriorRing

ST_ExteriorRing — Returns a LineString representing the exterior ring of a Polygon.

Synopsis

geometry **ST_ExteriorRing**(geometry a_polygon);

Description

Returns a LINESTRING representing the exterior ring (shell) of a POLYGON. Returns NULL if the geometry is not a polygon.

Note
This function does not support MULTIPOLYGONs. For MULTIPOLYGONs use in conjunction with ST_GeometryN or ST_Dump

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.

Examples

```
--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)
```

See Also

ST_InteriorRingN, ST_Boundary, ST_NumInteriorRings

7.4.13 ST_GeometryN

ST_GeometryN — Return an element of a geometry collection.

Synopsis

geometry **ST_GeometryN**(geometry geomA, integer n);

Description

Return the 1-based Nth element geometry of an input geometry which is a GEOMETRYCOLLECTION, MULTIPOINT, MULTILINESTRING, MULTICURVE, MULTI)POLYGON, or POLYHEDRALSURFACE. Otherwise, returns NULL.

Note
Index is 1-based as for OGC specs since version 0.8.0. Previous versions implemented this as 0-based instead.

Note
To extract all elements of a geometry, ST_Dump is more efficient and works for atomic geometries.

Enhanced: 2.0.0 support for Polyhedral surfaces, Triangles and TIN was introduced.

Changed: 2.0.0 Prior versions would return NULL for singular geometries. This was changed to return the geometry for ST_GeometryN(..,1) case.

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 Examples

```
--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);
```

Polyhedral Surfaces, TIN and Triangle Examples

```
-- 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))
```

See Also

ST_Dump, ST_NumGeometries

7.4.14 ST_GeometryType

ST_GeometryType — Returns the SQL-MM type of a geometry as text.

Synopsis

text **ST_GeometryType**(geometry g1);

Description

Returns the type of the geometry as a string. EG: 'ST_LineString', 'ST_Polygon','ST_MultiPolygon' etc. This function differs from GeometryType(geometry) in the case of the string and ST in front that is returned, as well as the fact that it will not indicate whether the geometry is measured.

Enhanced: 2.0.0 support for Polyhedral surfaces was introduced.

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.

Examples

```
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
```

See Also

GeometryType

7.4.15 ST_HasArc

ST_HasArc — Tests if a geometry contains a circular arc

Synopsis

boolean **ST_HasArc**(geometry geomA);

Description

Returns true if a geometry or geometry collection contains a circular string

Availability: 1.2.3?

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

This method supports Circular Strings and Curves.

Examples

```
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
```

See Also

ST_CurveToLine, ST_LineToCurve

7.4.16 ST_InteriorRingN

ST_InteriorRingN — Returns the Nth interior ring (hole) of a Polygon.

Synopsis

geometry **ST_InteriorRingN**(geometry a_polygon, integer n);

Description

Returns the Nth interior ring (hole) of a POLYGON geometry as a LINESTRING. The index starts at 1. Returns NULL if the geometry is not a polygon or the index is out of range.

Note
This function does not support MULTIPOLYGONs. For MULTIPOLYGONs use in conjunction with ST_GeometryN or 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: 8.2.6, 8.3.5

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

Examples

```
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;
```

See Also

ST_ExteriorRing, ST_BuildArea, ST_Collect, ST_Dump, ST_NumInteriorRing, ST_NumInteriorRings

7.4.17 ST_NumCurves

ST_NumCurves — Return the number of component curves in a CompoundCurve.

Synopsis

integer **ST_NumCurves**(geometry a_compoundcurve);

Description

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

This method implements the SQL/MM specification. SQL-MM 3: 8.2.6, 8.3.5

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

Examples

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

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

See Also

ST_CurveN, ST_Dump, ST_ExteriorRing, ST_NumInteriorRings, ST_NumGeometries

7.4.18 ST_CurveN

ST_CurveN — Returns the Nth component curve geometry of a CompoundCurve.

Synopsis

geometry **ST_CurveN**(geometry a_compoundcurve, integer index);

Description

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

This method implements the SQL/MM specification. SQL-MM 3: 8.2.6, 8.3.5

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

Examples

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

See Also

ST_NumCurves, ST_Dump, ST_ExteriorRing, ST_NumInteriorRings, ST_NumGeometries

7.4.19 ST_IsClosed

ST_IsClosed — Tests if a LineStrings's start and end points are coincident. For a PolyhedralSurface tests if it is closed (volumetric).

Synopsis

boolean **ST_IsClosed**(geometry g);

Description

Returns `TRUE` if the `LINESTRING`'s start and end points are coincident. For Polyhedral Surfaces, reports if the surface is areal (open) or volumetric (closed).

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

This method implements the SQL/MM specification. SQL-MM 3: 7.1.5, 9.3.3

Note

SQL-MM defines the result of `ST_IsClosed(NULL)` to be 0, while PostGIS returns `NULL`.

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

This method supports Circular Strings and Curves.

Enhanced: 2.0.0 support for Polyhedral surfaces was introduced.

This function supports Polyhedral surfaces.

Line String and Point Examples

```
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)
```

Polyhedral Surface Examples

```
    -- 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
```

See Also

ST_IsRing

7.4.20 ST_IsCollection

ST_IsCollection — Tests if a geometry is a geometry collection type.

Synopsis

boolean **ST_IsCollection**(geometry g);

Description

Returns `TRUE` if the geometry type of the argument a geometry collection type. Collection types are the following:

- GEOMETRYCOLLECTION
- MULTI{POINT,POLYGON,LINESTRING,CURVE,SURFACE}
- COMPOUNDCURVE

Note
This function analyzes the type of the geometry. This means that it will return `TRUE` on collections that are empty or that contain a single element.

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

This method supports Circular Strings and Curves.

Examples

```
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)
```

See Also

ST_NumGeometries

7.4.21 ST_IsEmpty

ST_IsEmpty — Tests if a geometry is empty.

Synopsis

boolean **ST_IsEmpty**(geometry geomA);

Description

Returns true if this Geometry is an empty geometry. If true, then this Geometry represents an empty geometry collection, polygon, point etc.

Note
SQL-MM defines the result of ST_IsEmpty(NULL) to be 0, while PostGIS returns NULL.

This method implements the OGC Simple Features Implementation Specification for SQL 1.1. s2.1.1.1

This method implements the SQL/MM specification. SQL-MM 3: 5.1.7

This method supports Circular Strings and Curves.

Warning
Changed: 2.0.0 In prior versions of PostGIS ST_GeomFromText('GEOMETRYCOLLECTION(EMPTY)') was allowed. This is now illegal in PostGIS 2.0.0 to better conform with SQL/MM standards

Examples

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

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

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

 st_isempty
------------
 f
(1 row)

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

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

7.4.22 ST_IsPolygonCCW

ST_IsPolygonCCW — Tests if Polygons have exterior rings oriented counter-clockwise and interior rings oriented clockwise.

Synopsis

boolean **ST_IsPolygonCCW** (geometry geom);

Description

Returns true if all polygonal components of the input geometry use a counter-clockwise orientation for their exterior ring, and a clockwise direction for all interior rings.

Returns true if the geometry has no polygonal components.

Note
Closed linestrings are not considered polygonal components, so you would still get a true return by passing a single closed linestring no matter its orientation.

Note
If a polygonal geometry does not use reversed orientation for interior rings (i.e., if one or more interior rings are oriented in the same direction as an exterior ring) then both ST_IsPolygonCW and ST_IsPolygonCCW will return false.

Availability: 2.4.0

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

This function supports M coordinates.

See Also

ST_ForcePolygonCW , ST_ForcePolygonCCW , ST_IsPolygonCW

7.4.23 ST_IsPolygonCW

ST_IsPolygonCW — Tests if Polygons have exterior rings oriented clockwise and interior rings oriented counter-clockwise.

Synopsis

boolean **ST_IsPolygonCW** (geometry geom);

Description

Returns true if all polygonal components of the input geometry use a clockwise orientation for their exterior ring, and a counter-clockwise direction for all interior rings.

Returns true if the geometry has no polygonal components.

Note
Closed linestrings are not considered polygonal components, so you would still get a true return by passing a single closed linestring no matter its orientation.

Note
If a polygonal geometry does not use reversed orientation for interior rings (i.e., if one or more interior rings are oriented in the same direction as an exterior ring) then both ST_IsPolygonCW and ST_IsPolygonCCW will return false.

Availability: 2.4.0

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

This function supports M coordinates.

See Also

ST_ForcePolygonCW , ST_ForcePolygonCCW , ST_IsPolygonCW

7.4.24 ST_IsRing

ST_IsRing — Tests if a LineString is closed and simple.

Synopsis

boolean **ST_IsRing**(geometry g);

Description

Returns `TRUE` if this `LINESTRING` is both ST_IsClosed (`ST_StartPoint(g) ~= ST_Endpoint(g)`) and ST_IsSimple (does not self intersect).

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 defines the result of `ST_IsRing(NULL)` to be 0, while PostGIS returns `NULL`.

Examples

```
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)
```

See Also

ST_IsClosed, ST_IsSimple, ST_StartPoint, ST_EndPoint

7.4.25 ST_IsSimple

ST_IsSimple — Tests if a geometry has no points of self-intersection or self-tangency.

Synopsis

boolean **ST_IsSimple**(geometry geomA);

Description

Returns true if this Geometry has no anomalous geometric points, such as self-intersection or self-tangency. For more information on the OGC's definition of geometry simplicity and validity, refer to "Ensuring OpenGIS compliance of geometries"

Note
SQL-MM defines the result of ST_IsSimple(NULL) to be 0, while PostGIS returns NULL.

This method implements the OGC Simple Features Implementation Specification for SQL 1.1. s2.1.1.1

This method implements the SQL/MM specification. SQL-MM 3: 5.1.8

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

Examples

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

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

See Also

ST_IsValid

7.4.26 ST_M

ST_M — Returns the M coordinate of a Point.

Synopsis

float **ST_M**(geometry a_point);

Description

Return the M coordinate of a Point, or NULL if not available. Input must be a Point.

Note
This is not (yet) part of the OGC spec, but is listed here to complete the point coordinate extractor function list.

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.

Examples

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

See Also

ST_GeomFromEWKT, ST_X, ST_Y, ST_Z

7.4.27 ST_MemSize

ST_MemSize — Returns the amount of memory space a geometry takes.

Synopsis

integer **ST_MemSize**(geometry geomA);

Description

Returns the amount of memory space (in bytes) the geometry takes.

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.

Examples

```
--Return how much byte space Boston takes up  in our Mass data set
SELECT pg_size_pretty(SUM(ST_MemSize(geom))) as totgeomsum,
pg_size_pretty(SUM(CASE WHEN town = 'BOSTON' THEN ST_MemSize(geom) ELSE 0 END)) As bossum,
CAST(SUM(CASE WHEN town = 'BOSTON' THEN ST_MemSize(geom) ELSE 0 END)*1.00 /
    SUM(ST_MemSize(geom))*100 As numeric(10,2)) As perbos
FROM towns;

totgeomsum  bossum  perbos
----------  ------  ------
1522 kB   30 kB 1.99


SELECT ST_MemSize(ST_GeomFromText('CIRCULARSTRING(220268 150415,220227 150505,220227  ↩
    150406)'));

---
73

--What percentage of our table is taken up by just the geometry
SELECT pg_total_relation_size('public.neighborhoods') As fulltable_size, sum(ST_MemSize( ↩
    geom)) As geomsize,
sum(ST_MemSize(geom))*1.00/pg_total_relation_size('public.neighborhoods')*100 As pergeom
FROM neighborhoods;
fulltable_size geomsize  pergeom
-----------------------------------------------
262144         96238   36.71188354492187500000
```

7.4.28 ST_NDims

ST_NDims — Returns the coordinate dimension of a geometry.

Synopsis

integer **ST_NDims**(geometry g1);

Description

Returns the coordinate dimension of the geometry. PostGIS supports 2 - (x,y) , 3 - (x,y,z) or 2D with measure - x,y,m, and 4 - 3D with measure space x,y,z,m

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

Examples

```
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
```

See Also

ST_CoordDim, ST_Dimension, ST_GeomFromEWKT

7.4.29 ST_NPoints

ST_NPoints — Returns the number of points (vertices) in a geometry.

Synopsis

integer **ST_NPoints**(geometry g1);

Description

Return the number of points in a geometry. Works for all geometries.

Enhanced: 2.0.0 support for Polyhedral surfaces was introduced.

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.

Examples

```
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
```

See Also

ST_NumPoints

7.4.30 ST_NRings

ST_NRings — Returns the number of rings in a polygonal geometry.

Synopsis

integer **ST_NRings**(geometry geomA);

Description

If the geometry is a polygon or multi-polygon returns the number of rings. Unlike NumInteriorRings, it counts the outer rings as well.

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

This method supports Circular Strings and Curves.

Examples

```
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)
```

See Also

ST_NumInteriorRings

7.4.31 ST_NumGeometries

ST_NumGeometries — Returns the number of elements in a geometry collection.

Synopsis

integer **ST_NumGeometries**(geometry geom);

Description

Returns the number of elements in a geometry collection (GEOMETRYCOLLECTION or MULTI*). For non-empty atomic geometries returns 1. For empty geometries returns 0.

Enhanced: 2.0.0 support for Polyhedral surfaces, Triangles and TIN was introduced.

Changed: 2.0.0 In prior versions this would return NULL if the geometry was not a collection/MULTI type. 2.0.0+ now returns 1 for single geometries e.g POLYGON, LINESTRING, POINT.

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).

Examples

```
--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
```

See Also

ST_GeometryN, ST_Multi

7.4.32 ST_NumInteriorRings

ST_NumInteriorRings — Returns the number of interior rings (holes) of a Polygon.

Synopsis

integer **ST_NumInteriorRings**(geometry a_polygon);

Description

Return the number of interior rings of a polygon geometry. Return NULL if the geometry is not a polygon.

This method implements the SQL/MM specification. SQL-MM 3: 8.2.5

Changed: 2.0.0 - in prior versions it would allow passing a MULTIPOLYGON, returning the number of interior rings of first POLYGON.

Examples

```
--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;
```

See Also

ST_NumInteriorRing, ST_InteriorRingN

7.4.33 ST_NumInteriorRing

ST_NumInteriorRing — Returns the number of interior rings (holes) of a Polygon. Aias for ST_NumInteriorRings

Synopsis

integer **ST_NumInteriorRing**(geometry a_polygon);

See Also

ST_NumInteriorRings, ST_InteriorRingN

7.4.34 ST_NumPatches

ST_NumPatches — Return the number of faces on a Polyhedral Surface. Will return null for non-polyhedral geometries.

Synopsis

integer **ST_NumPatches**(geometry g1);

Description

Return the number of faces on a Polyhedral Surface. Will return null for non-polyhedral geometries. This is an alias for ST_NumGeometries to support MM naming. Faster to use ST_NumGeometries if you don't care about MM convention.

Availability: 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.

Examples

```
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
```

See Also

ST_GeomFromEWKT, ST_NumGeometries

7.4.35 ST_NumPoints

ST_NumPoints — Returns the number of points in a LineString or CircularString.

Synopsis

integer **ST_NumPoints**(geometry g1);

Description

Return the number of points in an ST_LineString or ST_CircularString value. Prior to 1.4 only works with linestrings as the specs state. From 1.4 forward this is an alias for ST_NPoints which returns number of vertices for not just linestrings. Consider using ST_NPoints instead which is multi-purpose and works with many geometry types.

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

Examples

```
SELECT ST_NumPoints(ST_GeomFromText('LINESTRING(77.29 29.07,77.42 29.26,77.27 29.31,77.29 ↩
    29.07)'));
    --result
    4
```

See Also

ST_NPoints

7.4.36 ST_PatchN

ST_PatchN — Returns the Nth geometry (face) of a PolyhedralSurface.

Synopsis

geometry **ST_PatchN**(geometry geomA, integer n);

Description

Returns the 1-based Nth geometry (face) if the geometry is a POLYHEDRALSURFACE or POLYHEDRALSURFACEM. Otherwise, returns NULL. This returns the same answer as ST_GeometryN for PolyhedralSurfaces. Using ST_GeometryN is faster.

Note
Index is 1-based.

Note
If you want to extract all elements of a geometry ST_Dump is more efficient.

Availability: 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.

Examples

```
--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))
```

See Also

ST_AsEWKT, ST_GeomFromEWKT, ST_Dump, ST_GeometryN, ST_NumGeometries

7.4.37 ST_PointN

ST_PointN — Returns the Nth point in the first LineString or circular LineString in a geometry.

Synopsis

geometry **ST_PointN**(geometry a_linestring, integer n);

Description

Return the Nth point in a single linestring or circular linestring in the geometry. Negative values are counted backwards from the end of the LineString, so that -1 is the last point. Returns NULL if there is no linestring in the geometry.

Note
Index is 1-based as for OGC specs since version 0.8.0. Backward indexing (negative index) is not in OGC Previous versions implemented this as 0-based instead.

Note
If you want to get the Nth point of each LineString in a MultiLineString, use in conjunction with 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
Changed: 2.0.0 no longer works with single geometry multilinestrings. In older versions of PostGIS -- a single line multilinestring would work happily with this function and return the start point. In 2.0.0 it just returns NULL like any other multilinestring.
Changed: 2.3.0 : negative indexing available (-1 is last point)

Examples

```
-- 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)
```

See Also

ST_NPoints

7.4.38 ST_Points

ST_Points — Returns a MultiPoint containing the coordinates of a geometry.

Synopsis

geometry **ST_Points**(geometry geom);

Description

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.

Availability: 2.3.0

Examples

```
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))
```

See Also

ST_RemoveRepeatedPoints, ST_DumpPoints

7.4.39 ST_StartPoint

ST_StartPoint — Returns the first point of a LineString.

Synopsis

geometry **ST_StartPoint**(geometry geomA);

Description

Returns the first point of a LINESTRING or CIRCULARLINESTRING geometry as a POINT. Returns NULL if the input is not a LINESTRING or CIRCULARLINESTRING.

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.
Changed: 2.0.0 no longer works with single geometry MultiLineStrings. In older versions of PostGIS a single-line MultiLineString would work happily with this function and return the start point. In 2.0.0 it just returns NULL like any other MultiLineString. The old behavior was an undocumented feature, but people who assumed they had their data stored as LINESTRING may experience these returning NULL in 2.0.0.

Examples

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)
```

Start point of a CircularString

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

See Also

ST_EndPoint, ST_PointN

7.4.40 ST_Summary

ST_Summary — Returns a text summary of the contents of a geometry.

Synopsis

text **ST_Summary**(geometry g);
text **ST_Summary**(geography g);

Description

Returns a text summary of the contents of the geometry.

Flags shown square brackets after the geometry type have the following meaning:

- M: has M coordinate
- Z: has Z coordinate
- B: has a cached bounding box
- G: is geodetic (geography)
- S: has spatial reference system

This method supports Circular Strings and Curves.

This function supports Polyhedral surfaces.

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

Availability: 1.2.2

Enhanced: 2.0.0 added support for geography

Enhanced: 2.1.0 S flag to denote if has a known spatial reference system

Enhanced: 2.2.0 Added support for TIN and Curves

Examples

```
=# 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)
```

See Also

PostGIS_DropBBox, PostGIS_AddBBox, ST_Force3DM, ST_Force3DZ, ST_Force2D, geography

ST_IsValid, ST_IsValid, ST_IsValidReason, ST_IsValidDetail

7.4.41 ST_X

ST_X — Returns the X coordinate of a Point.

Synopsis

float **ST_X**(geometry a_point);

Description

Return the X coordinate of the point, or NULL if not available. Input must be a point.

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.

Examples

```
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)
```

See Also

ST_Centroid, ST_GeomFromEWKT, ST_M, ST_XMax, ST_XMin, ST_Y, ST_Z

7.4.42 ST_Y

ST_Y — Returns the Y coordinate of a Point.

Synopsis

float **ST_Y**(geometry a_point);

Description

Return the Y coordinate of the point, or NULL if not available. Input must be a point.

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.

Examples

```
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)
```

See Also

ST_Centroid, ST_GeomFromEWKT, ST_M, ST_X, ST_YMax, ST_YMin, ST_Z

7.4.43 ST_Z

ST_Z — Returns the Z coordinate of a Point.

Synopsis

float **ST_Z**(geometry a_point);

Description

Return the Z coordinate of the point, or NULL if not available. Input must be a point.

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.

Examples

```
SELECT ST_Z(ST_GeomFromEWKT('POINT(1 2 3 4)'));
 st_z
------
  3
(1 row)
```

See Also

ST_GeomFromEWKT, ST_M, ST_X, ST_Y, ST_ZMax, ST_ZMin

7.4.44 ST_Zmflag

ST_Zmflag — Returns a code indicating the ZM coordinate dimension of a geometry.

Synopsis

smallint **ST_Zmflag**(geometry geomA);

Description

Returns a code indicating the ZM coordinate dimension of a geometry.

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.

Examples

```
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
```

See Also

ST_CoordDim, ST_NDims, ST_Dimension

7.4.45 ST_HasZ

ST_HasZ — Checks if a geometry has a Z dimension.

Synopsis

boolean **ST_HasZ**(geometry geom);

Description

Checks if the input geometry has a Z dimension and returns a boolean value. If the geometry has a Z dimension, it returns true; otherwise, it returns false.

Geometry objects with a Z dimension typically represent three-dimensional (3D) geometries, while those without it are two-dimensional (2D) geometries.

This function is useful for determining if a geometry has elevation or height information.

Availability: 3.5.0

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

This function supports M coordinates.

Examples

```
SELECT ST_HasZ(ST_GeomFromText('POINT(1 2 3)'));
 --result
 true
```

```
SELECT ST_HasZ(ST_GeomFromText('LINESTRING(0 0, 1 1)'));
 --result
 false
```

See Also

ST_Zmflag

ST_HasM

7.4.46 ST_HasM

ST_HasM — Checks if a geometry has an M (measure) dimension.

Synopsis

boolean **ST_HasM**(geometry geom);

Description

Checks if the input geometry has an M (measure) dimension and returns a boolean value. If the geometry has an M dimension, it returns true; otherwise, it returns false.

Geometry objects with an M dimension typically represent measurements or additional data associated with spatial features.

This function is useful for determining if a geometry includes measure information.

Availability: 3.5.0

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

This function supports M coordinates.

Examples

```
SELECT ST_HasM(ST_GeomFromText('POINTM(1 2 3)'));
 --result
 true
```

```
SELECT ST_HasM(ST_GeomFromText('LINESTRING(0 0, 1 1)'));
 --result
 false
```

See Also

ST_Zmflag

ST_HasZ

7.5 Geometry Editors

7.5.1 ST_AddPoint

ST_AddPoint — Add a point to a LineString.

Synopsis

geometry **ST_AddPoint**(geometry linestring, geometry point);

geometry **ST_AddPoint**(geometry linestring, geometry point, integer position = -1);

Description

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.

Availability: 1.1.0

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

Examples

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;
```

See Also

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);

Description

Given a geometry collection, returns a homogeneous multi-geometry.

If the *type* is not specified, returns a multi-geometry containing only geometries of the highest dimension. So polygons are preferred over lines, which are preferred over points.

If the *type* is specified, returns a multi-geometry containing only that type. If there are no sub-geometries of the right type, an EMPTY geometry is returned. Only points, lines and polygons are supported. The type numbers are:

- 1 == POINT
- 2 == LINESTRING
- 3 == POLYGON

For atomic geometry inputs, the geometry is returned unchanged if the input type matches the requested type. Otherwise, the result is an EMPTY geometry of the specified type. If required, these can be converted to multi-geometries using ST_Multi.

Warning
MultiPolygon results are not checked for validity. If the polygon components are adjacent or overlapping the result will be invalid. (For example, this can occur when applying this function to an ST_Split result.) This situation can be checked with ST_IsValid and repaired with ST_MakeValid.

Availability: 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.

Examples

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))
```

See Also

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);

Description

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.

Availability: 2.0.0

Examples

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)))
```

See Also

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);

Description

Converts a CIRCULAR STRING to regular LINESTRING or CURVEPOLYGON to POLYGON or MULTISURFACE to MULTIPOLYGON. Useful for outputting to devices that can't support CIRCULARSTRING geometry types

Converts a given geometry to a linear geometry. Each curved geometry or segment is converted into a linear approximation using the given `tolerance` and options (32 segments per quadrant and no options by default).

The 'tolerance_type' argument determines interpretation of the `tolerance` argument. It can take the following values:

- 0 (default): Tolerance is max segments per quadrant.
- 1: Tolerance is max-deviation of line from curve, in source units.
- 2: Tolerance is max-angle, in radians, between generating radii.

The 'flags' argument is a bitfield. 0 by default. Supported bits are:

- 1: Symmetric (orientation idependent) output.
- 2: Retain angle, avoids reducing angles (segment lengths) when producing symmetric output. Has no effect when Symmetric flag is off.

Availability: 1.3.0

Enhanced: 2.4.0 added support for max-deviation and max-angle tolerance, and for symmetric output.

Enhanced: 3.0.0 implemented a minimum number of segments per linearized arc to prevent topological collapse.

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

This method implements the SQL/MM specification. SQL-MM 3: 7.1.7

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

This method supports Circular Strings and Curves.

Examples

```
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)
```

See Also

ST_LineToCurve

7.5.5 ST_Scroll

ST_Scroll — Change start point of a closed LineString.

Synopsis

geometry **ST_Scroll**(geometry linestring, geometry point);

Description

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.

Examples

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)
```

See Also

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);

Description

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).

Availability: 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).

Example

```
SELECT ST_AsEWKT(ST_FlipCoordinates(GeomFromEWKT('POINT(1 2)')));
 st_asewkt
------------
POINT(2 1)
```

See Also

ST_SwapOrdinates

7.5.7 ST_Force2D

ST_Force2D — Force the geometries into a "2-dimensional mode".

Synopsis

geometry **ST_Force2D**(geometry geomA);

Description

Forces the geometries into a "2-dimensional mode" so that all output representations will only have the X and Y coordinates. This is useful for force OGC-compliant output (since OGC only specifies 2-D geometries).

Enhanced: 2.0.0 support for Polyhedral surfaces was introduced.

Changed: 2.1.0. Up to 2.0.x this was called ST_Force_2D.

This method supports Circular Strings and Curves.

This function supports Polyhedral surfaces.

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

Examples

```
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))
```

See Also

ST_Force3D

7.5.8 ST_Force3D

ST_Force3D — Force the geometries into XYZ mode. This is an alias for ST_Force3DZ.

Synopsis

geometry **ST_Force3D**(geometry geomA, float Zvalue = 0.0);

Description

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.

Enhanced: 2.0.0 support for Polyhedral surfaces was introduced.

Changed: 2.1.0. Up to 2.0.x this was called ST_Force_3D.

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.

Examples

```
    --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))
```

See Also

ST_AsEWKT, ST_Force2D, ST_Force3DM, ST_Force3DZ

7.5.9 ST_Force3DZ

ST_Force3DZ — Force the geometries into XYZ mode.

Synopsis

geometry **ST_Force3DZ**(geometry geomA, float Zvalue = 0.0);

Description

Forces the geometries into XYZ mode. If a geometry has no Z component, then a *Zvalue* Z coordinate is tacked on.

Enhanced: 2.0.0 support for Polyhedral surfaces was introduced.

Changed: 2.1.0. Up to 2.0.x this was called ST_Force_3DZ.

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.

Examples

```
--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))
```

See Also

ST_AsEWKT, ST_Force2D, ST_Force3DM, ST_Force3D

7.5.10 ST_Force3DM

ST_Force3DM — Force the geometries into XYM mode.

Synopsis

geometry **ST_Force3DM**(geometry geomA, float Mvalue = 0.0);

Description

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

Changed: 2.1.0. Up to 2.0.x this was called ST_Force_3DM.

Changed: 3.1.0. Added support for supplying a non-zero M value.

This method supports Circular Strings and Curves.

Examples

```
--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))
```

See Also

ST_AsEWKT, ST_Force2D, ST_Force3DM, ST_Force3D, ST_GeomFromEWKT

7.5.11 ST_Force4D

ST_Force4D — Force the geometries into XYZM mode.

Synopsis

geometry **ST_Force4D**(geometry geomA, float Zvalue = 0.0, float Mvalue = 0.0);

Description

Forces the geometries into XYZM mode. *Zvalue* and *Mvalue* is tacked on for missing Z and M dimensions, respectively.

Changed: 2.1.0. Up to 2.0.x this was called ST_Force_4D.

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.

Examples

```
--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))
```

See Also

ST_AsEWKT, ST_Force2D, ST_Force3DM, ST_Force3D

7.5.12 ST_ForceCollection

ST_ForceCollection — Convert the geometry into a GEOMETRYCOLLECTION.

Synopsis

geometry **ST_ForceCollection**(geometry geomA);

Description

Converts the geometry into a GEOMETRYCOLLECTION. This is useful for simplifying the WKB representation.

Enhanced: 2.0.0 support for Polyhedral surfaces was introduced.

Availability: 1.2.2, prior to 1.3.4 this function will crash with Curves. This is fixed in 1.3.4+

Changed: 2.1.0. Up to 2.0.x this was called ST_Force_Collection.

This function supports Polyhedral surfaces.

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

This method supports Circular Strings and Curves.

Examples

```
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))
)
```

See Also

ST_AsEWKT, ST_Force2D, ST_Force3DM, ST_Force3D, ST_GeomFromEWKT

7.5.13 ST_ForceCurve

ST_ForceCurve — Upcast a geometry into its curved type, if applicable.

Synopsis

geometry **ST_ForceCurve**(geometry g);

Description

Turns a geometry into its curved representation, if applicable: lines become compoundcurves, multilines become multicurves polygons become curvepolygons multipolygons become multisurfaces. If the geometry input is already a curved representation returns back same as input.

Availability: 2.2.0

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

This method supports Circular Strings and Curves.

Examples

```
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)
```

See Also

ST_LineToCurve

7.5.14 ST_ForcePolygonCCW

ST_ForcePolygonCCW — Orients all exterior rings counter-clockwise and all interior rings clockwise.

Synopsis

geometry **ST_ForcePolygonCCW** (geometry geom);

Description

Forces (Multi)Polygons to use a counter-clockwise orientation for their exterior ring, and a clockwise orientation for their interior rings. Non-polygonal geometries are returned unchanged.

Availability: 2.4.0

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

This function supports M coordinates.

See Also

ST_ForcePolygonCW , ST_IsPolygonCCW , ST_IsPolygonCW

7.5.15 ST_ForcePolygonCW

ST_ForcePolygonCW — Orients all exterior rings clockwise and all interior rings counter-clockwise.

Synopsis

geometry **ST_ForcePolygonCW** (geometry geom);

Description

Forces (Multi)Polygons to use a clockwise orientation for their exterior ring, and a counter-clockwise orientation for their interior rings. Non-polygonal geometries are returned unchanged.

Availability: 2.4.0

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

This function supports M coordinates.

See Also

ST_ForcePolygonCCW , ST_IsPolygonCCW , ST_IsPolygonCW

7.5.16 ST_ForceSFS

ST_ForceSFS — Force the geometries to use SFS 1.1 geometry types only.

Synopsis

geometry **ST_ForceSFS**(geometry geomA);
geometry **ST_ForceSFS**(geometry geomA, text version);

Description

This function supports Polyhedral surfaces.

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

This method supports Circular Strings and Curves.

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

7.5.17 ST_ForceRHR

ST_ForceRHR — Force the orientation of the vertices in a polygon to follow the Right-Hand-Rule.

Synopsis

geometry **ST_ForceRHR**(geometry g);

Description

Forces the orientation of the vertices in a polygon to follow a Right-Hand-Rule, in which the area that is bounded by the polygon is to the right of the boundary. In particular, the exterior ring is orientated in a clockwise direction and the interior rings in a counter-clockwise direction. This function is a synonym for ST_ForcePolygonCW

Note
The above definition of the Right-Hand-Rule conflicts with definitions used in other contexts. To avoid confusion, it is recommended to use ST_ForcePolygonCW.

Enhanced: 2.0.0 support for Polyhedral surfaces was introduced.

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

This function supports Polyhedral surfaces.

Examples

```
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)
```

See Also

ST_ForcePolygonCCW , ST_ForcePolygonCW , ST_IsPolygonCCW , ST_IsPolygonCW , ST_BuildArea, ST_Polygonize, ST_Reverse

7.5.18 ST_LineExtend

ST_LineExtend — Returns a line extended forwards and backwards by specified distances.

Synopsis

geometry **ST_LineExtend**(geometry line, float distance_forward, float distance_backward=0.0);

Description

Returns a line extended forwards and backwards by adding new start (and end) points at the given distance(s). A distance of zero does not add a point. Only non-negative distances are allowed. The direction(s) of the added point(s) is determined by the first (and last) two distinct points of the line. Duplicate points are ignored.

Availability: 3.4.0

Example: Extends a line 5 units forward and 6 units backward

```
SELECT ST_AsText(ST_LineExtend('LINESTRING(0 0, 0 10)'::geometry, 5, 6));
--------------------------------------------
LINESTRING(0 -6,0 0,0 10,0 15)
```

See Also

ST_LineSubstring, ST_LocateAlong, ST_Project

7.5.19 ST_LineToCurve

ST_LineToCurve — Converts a linear geometry to a curved geometry.

Synopsis

geometry **ST_LineToCurve**(geometry geomANoncircular);

Description

Converts plain LINESTRING/POLYGON to CIRCULAR STRINGs and Curved Polygons. Note much fewer points are needed to describe the curved equivalent.

> **Note**
> If the input LINESTRING/POLYGON is not curved enough to clearly represent a curve, the function will return the same input geometry.

Availability: 1.3.0

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

This method supports Circular Strings and Curves.

Examples

```
 -- 2D Example
SELECT ST_AsText(ST_LineToCurve(foo.geom)) As curvedastext,ST_AsText(foo.geom) As  ↩
    non_curvedastext
    FROM (SELECT ST_Buffer('POINT(1 3)'::geometry, 3) As geom) As foo;

curvedatext                                                          non_curvedastext
-------------------------------------------------------------------|-------------------------------

CURVEPOLYGON(CIRCULARSTRING(4 3,3.12132034355964 0.878679656440359, | POLYGON((4  ↩
    3,3.94235584120969 2.41472903395162,3.77163859753386 1.85194970290473,
1 0,-1.12132034355965 5.12132034355963,4 3))                        |  3.49440883690764  ↩
    1.33328930094119,3.12132034355964 0.878679656440359,
                                                                    |  2.66671069905881  ↩
                                                                       0.505591163092366,2.14805029
                                                                        0.228361402466141,
```

```
                                                                    |  1.58527096604839  ↩
                                                                       0.0576441587903094,1 ↩
                                                                        0,
                                                                    |  0.414729033951621  ↩
                                                                       0.0576441587903077,-0.1480502
                                                                        0.228361402466137,
                                                                    |  -0.666710699058802  ↩
                                                                       0.505591163092361,-1.1213203
                                                                        0.878679656440353,
                                                                    |  -1.49440883690763  ↩
                                                                       1.33328930094119,-1.7716385
                                                                        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)
```

See Also

ST_CurveToLine

7.5.20 ST_Multi

ST_Multi — Return the geometry as a MULTI* geometry.

Synopsis

geometry **ST_Multi**(geometry geom);

Description

Returns the geometry as a MULTI* geometry collection. If the geometry is already a collection, it is returned unchanged.

Examples

```
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)))
```

See Also

ST_AsText

7.5.21 ST_Normalize

ST_Normalize — Return the geometry in its canonical form.

Synopsis

geometry **ST_Normalize**(geometry geom);

Description

Returns the geometry in its normalized/canonical form. May reorder vertices in polygon rings, rings in a polygon, elements in a multi-geometry complex.

Mostly only useful for testing purposes (comparing expected and obtained results).

Availability: 2.3.0

Examples

```
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)
```

See Also

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);

Description

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.

Availability: 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)
```

See Also

ST_Azimuth, ST_Distance, PostgreSQL function radians()

7.5.23 ST_QuantizeCoordinates

ST_QuantizeCoordinates — Sets least significant bits of coordinates to zero

Synopsis

geometry **ST_QuantizeCoordinates** (geometry g , int prec_x , int prec_y , int prec_z , int prec_m);

Description

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.

The coordinates produced by ST_QuantizeCoordinates are independent of the geometry that contains those coordinates and the relative position of those coordinates within the geometry. As a result, existing topological relationships between geometries are unaffected by use of this function. The function may produce invalid geometry when it is called with a number of digits lower than the intrinsic precision of the geometry.

Availability: 2.5.0

Technical Background

PostGIS stores all coordinate values as double-precision floating point integers, which can reliably represent 15 significant digits. However, PostGIS may be used to manage data that intrinsically has fewer than 15 significant digits. An example is TIGER data, which is provided as geographic coordinates with six digits of precision after the decimal point (thus requiring only nine significant digits of longitude and eight significant digits of latitude.)

When 15 significant digits are available, there are many possible representations of a number with 9 significant digits. A double precision floating point number uses 52 explicit bits to represent the significand (mantissa) of the coordinate. Only 30 bits are needed to represent a mantissa with 9 significant digits, leaving 22 insignificant bits; we can set their value to anything we like and still end up with a number that rounds to our input value. For example, the value 100.123456 can be represented by the floating point numbers closest to 100.123456000000, 100.123456000001, and 100.123456432199. All are equally valid, in that ST_AsText(geom, 6) will return the same result with any of these inputs. As we can set these bits to any value, ST_QuantizeCoordinates sets the 22 insignificant bits to zero. For a long coordinate sequence this creates a pattern of blocks of consecutive zeros that is compressed by PostgreSQL more efficiently.

Note
Only the on-disk size of the geometry is potentially affected by ST_QuantizeCoordinates. ST_MemSize, which reports the in-memory usage of the geometry, will return the the same value regardless of the disk space used by a geometry.

Examples

```
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      | 010100000000000000000005e400000000000005e40 | POINT(120 120)
-3      | 01010000000000000000005840000000000005840 | POINT(96 96)
-4      | 01010000000000000000005840000000000005840 | POINT(96 96)
-5      | 01010000000000000000005840000000000005840 | POINT(96 96)
-6      | 01010000000000000000005840000000000005840 | POINT(96 96)
-7      | 01010000000000000000005840000000000005840 | POINT(96 96)
-8      | 01010000000000000000005840000000000005840 | POINT(96 96)
-9      | 01010000000000000000005840000000000005840 | POINT(96 96)
-10     | 01010000000000000000005840000000000005840 | POINT(96 96)
-11     | 01010000000000000000005840000000000005840 | POINT(96 96)
-12     | 01010000000000000000005840000000000005840 | POINT(96 96)
-13     | 01010000000000000000005840000000000005840 | POINT(96 96)
-14     | 01010000000000000000005840000000000005840 | POINT(96 96)
-15     | 01010000000000000000005840000000000005840 | POINT(96 96)
```

See Also

ST_SnapToGrid

7.5.24 ST_RemovePoint

ST_RemovePoint — Remove a point from a linestring.

Synopsis

geometry **ST_RemovePoint**(geometry linestring, integer offset);

Description

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

Availability: 1.1.0

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

Examples

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);
```

See Also

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);

Description

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

Availability: 2.2.0

This function supports Polyhedral surfaces.

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

Examples

```
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)
```

See Also

ST_Simplify

7.5.26 ST_RemoveIrrelevantPointsForView

ST_RemoveIrrelevantPointsForView — Removes points that are irrelevant for rendering a specific rectangluar view of a geometry.

Synopsis

geometry **ST_RemoveIrrelevantPointsForView**(geometry geom, box2d bounds, boolean cartesian_hint = false);

Description

Returns a geometry without points being irrelevant for rendering the geometry within a given rectangluar view.

This function can be used to quickly preprocess geometries that should be rendered only within certain bounds.

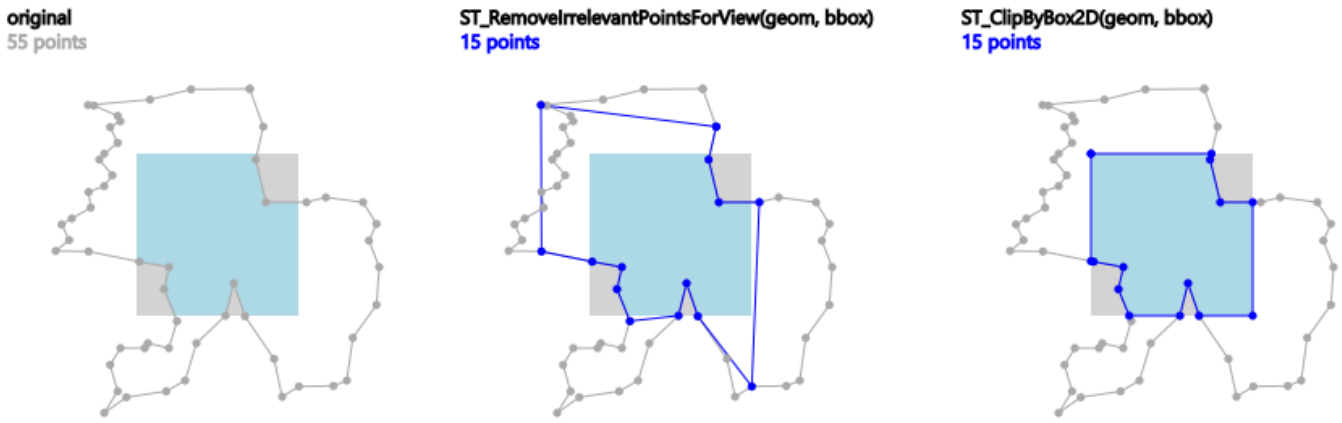
Only geometries of type (MULTI)POLYGON and (MULTI)LINESTRING are evaluated. Other geometries keep unchanged.

In contrast to `ST_ClipByBox2D()` this function

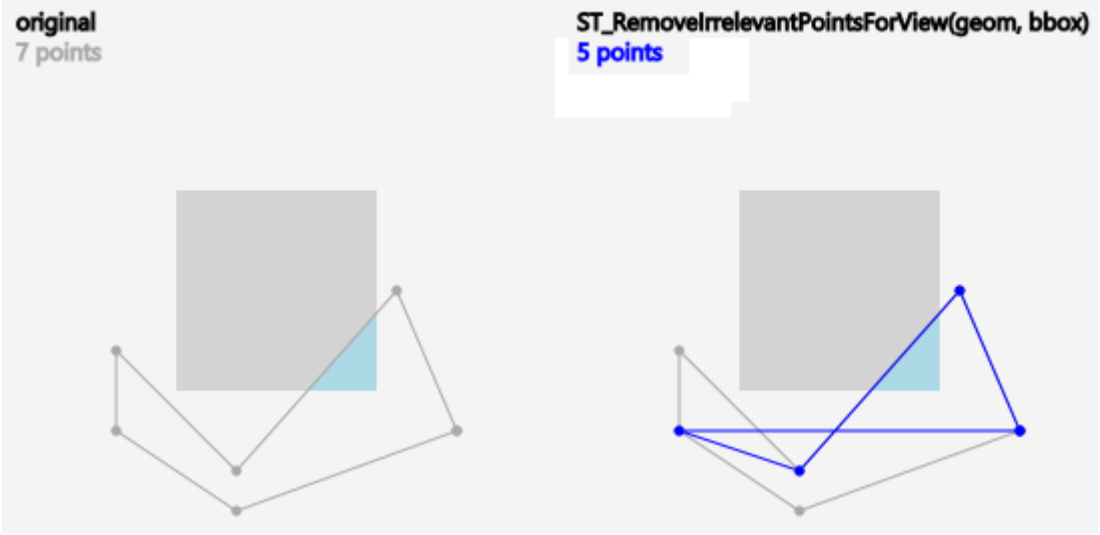
- sorts out points without computing new intersection points which avoids rounding errors and usually increases performance,
- returns a geometry with equal or similar point number,
- leads to the same rendering result within the specified view, and
- may introduce self-intersections which would make the resulting geometry invalid (see example below).

If `cartesian_hint` is set to `true`, the algorithm applies additional optimizations involving cartesian math to further reduce the resulting point number. Please note that using this option might introduce rendering artifacts if the resulting coordinates are projected into another (non-cartesian) coordinate system before rendering.

Warning
For polygons, this function does currently not ensure that the result is valid. This situation can be checked with ST_IsValid and repaired with ST_MakeValid.



Example: ST_RemoveIrrelevantPointsForView() applied to a polygon. Blue points remain, the rendering result (light-blue area) within the grey view box remains as well.



Example: Due to the fact that points are just sorted out and no new points are computed, the result of ST_RemoveIrrelevantPointsForView() may contain self-intersections.

Availability: 3.5.0

Examples

```
  SELECT ST_AsText(
  ST_RemoveIrrelevantPointsForView(
  ST_GeomFromText('MULTIPOLYGON(((10 10, 20 10, 30 10, 40 10, 20 20, 10 20, 10 10)) ↩
      ,((10 10, 20 10, 20 20, 10 20, 10 10)))'),
  ST_MakeEnvelope(12,12,18,18), true));

st_astext
---------
    MULTIPOLYGON(((10 10,40 10,20 20,10 20,10 10)),((10 10,20 10,20 20,10 20,10 10)))
```

```
  SELECT ST_AsText(
  ST_RemoveIrrelevantPointsForView(
  ST_GeomFromText('MULTILINESTRING((0 0, 10 0,20 0,30 0), (0 15, 5 15, 10 15, 15 15, 20 ↩
       15, 25 15, 30 15, 40 15), (13 13,15 15,17 17))'),
```

```
  ST_MakeEnvelope(12,12,18,18), true));

st_astext
---------
  MULTILINESTRING((10 15,15 15,20 15),(13 13,15 15,17 17))
```

```
  SELECT ST_AsText(
  ST_RemoveIrrelevantPointsForView(
  ST_GeomFromText('LINESTRING(0 0, 10 0,20 0,30 0)'),
  ST_MakeEnvelope(12,12,18,18), true));

st_astext
---------
    LINESTRING EMPTY
```

```
  SELECT ST_AsText(
  ST_RemoveIrrelevantPointsForView(
  ST_GeomFromText('POLYGON((0 30, 15 30, 30 30, 30 0, 0 0, 0 30))'),
  ST_MakeEnvelope(12,12,18,18), true));

st_astext
---------
    POLYGON((15 30,30 0,0 0,15 30))
```

```
  SELECT ST_AsText(
  ST_RemoveIrrelevantPointsForView(
  ST_GeomFromText('POLYGON((0 30, 15 30, 30 30, 30 0, 0 0, 0 30))'),
  ST_MakeEnvelope(12,12,18,18)));

st_astext
---------
    POLYGON((0 30,30 30,30 0,0 0,0 30))
```

See Also

ST_ClipByBox2D, ST_Intersection

7.5.27 ST_RemoveSmallParts

ST_RemoveSmallParts — Removes small parts (polygon rings or linestrings) of a geometry.

Synopsis

geometry **ST_RemoveSmallParts**(geometry geom, double precision minSizeX, double precision minSizeY);

Description

Returns a geometry without small parts (exterior or interior polygon rings, or linestrings).

This function can be used as preprocessing step for creating simplified maps, e. g. to remove small islands or holes.

It evaluates only geometries of type (MULTI)POLYGON and (MULTI)LINESTRING. Other geometries remain unchanged.

If *minSizeX* is greater than 0, parts are sorted out if their width is smaller than *minSizeX*.

If *minSizeY* is greater than 0, parts are sorted out if their height is smaller than *minSizeY*.

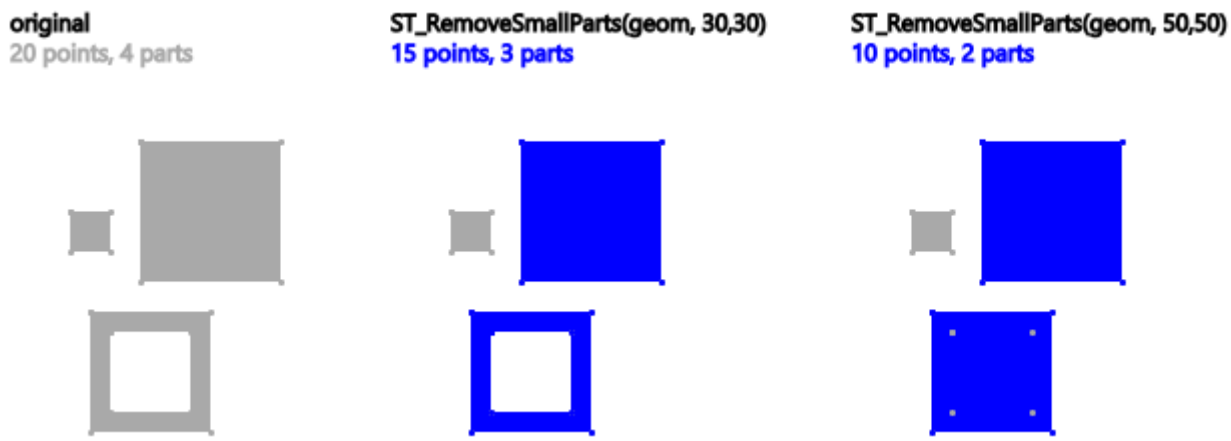
Both *minSizeX* and *minSizeY* are measured in coordinate system units of the geometry.

For polygon types, evaluation is done separately for each ring which can lead to one of the following results:

- the original geometry,
- a POLYGON with all rings with less vertices,
- a POLYGON with a reduced number of interior rings (having possibly less vertices),
- a POLYGON EMPTY, or
- a MULTIPOLYGON with a reduced number of polygons (having possibly less interior rings or vertices), or
- a MULTIPOLYGON EMPTY.

For linestring types, evaluation is done for each linestring which can lead to one of the following results:

- the original geometry,
- a LINESTRING with a reduced number of vertices,
- a LINESTRING EMPTY,
- a MULTILINESTRING with a reduced number of linestrings (having possibly less vertices), or
- a MULTILINESTRING EMPTY.



Example: ST_RemoveSmallParts() applied to a multi-polygon. Blue parts remain.

Availability: 3.5.0

Examples

```
    SELECT ST_AsText(
    ST_RemoveSmallParts(
    ST_GeomFromText('MULTIPOLYGON(
      ((60 160, 120 160, 120 220, 60 220, 60 160), (70 170, 70 210, 110 210, 110 170, 70 ↩
         170)),
      ((85 75, 155 75, 155 145, 85 145, 85 75)),
      ((50 110, 70 110, 70 130, 50 130, 50 110)))'),
      50, 50));

  st_astext
  ---------
    MULTIPOLYGON(((60 160,120 160,120 220,60 220,60 160)),((85 75,155 75,155 145,85 ↩
       145,85 75)))
```

```
      SELECT ST_AsText(
      ST_RemoveSmallParts(
      ST_GeomFromText('LINESTRING(10 10, 20 20)'),
        50, 50));

    st_astext
    ---------
      LINESTRING EMPTY
```

7.5.28 ST_Reverse

ST_Reverse — Return the geometry with vertex order reversed.

Synopsis

geometry **ST_Reverse**(geometry g1);

Description

Can be used on any geometry and reverses the order of the vertices.

Enhanced: 2.4.0 support for curves was introduced.

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

This function supports Polyhedral surfaces.

Examples

```
SELECT ST_AsText(geom) as line, ST_AsText(ST_Reverse(geom)) As reverseline
FROM
(SELECT ST_MakeLine(ST_Point(1,2),
    ST_Point(1,10)) As geom) as foo;
--result
    line         |     reverseline
--------------------+---------------------
LINESTRING(1 2,1 10) | LINESTRING(1 10,1 2)
```

7.5.29 ST_Segmentize

ST_Segmentize — Returns a modified geometry/geography having no segment longer than a given distance.

Synopsis

geometry **ST_Segmentize**(geometry geom, float max_segment_length);
geography **ST_Segmentize**(geography geog, float max_segment_length);

Description

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.

Availability: 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

Enhanced: 2.1.0 support for geography was introduced.

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)`)

Examples

Segmentizing a line. Long segments are split evenly, and short segments are not split.

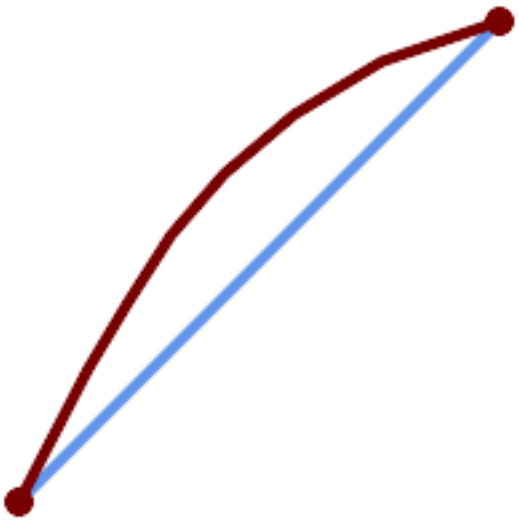
```
SELECT ST_AsText(ST_Segmentize(
    'MULTILINESTRING((0 0, 0 1, 0 9),(1 10, 1 18))'::geometry,
  5 ) );
-----------------------------------------------------
MULTILINESTRING((0 0,0 1,0 5,0 9),(1 10,1 14,1 18))
```

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

See Also

ST_LineSubstring

7.5.30 ST_SetPoint

ST_SetPoint — Replace point of a linestring with a given point.

Synopsis

geometry **ST_SetPoint**(geometry linestring, integer zerobasedposition, geometry point);

Description

Replace point N of linestring with given point. Index is 0-based.Negative index are counted backwards, so that -1 is last point. This is especially useful in triggers when trying to maintain relationship of joints when one vertex moves.

Availability: 1.1.0

Updated 2.3.0 : negative indexing

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

Examples

```
--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)
```

See Also

ST_AddPoint, ST_NPoints, ST_NumPoints, ST_PointN, ST_RemovePoint

7.5.31 ST_ShiftLongitude

ST_ShiftLongitude — Shifts the longitude coordinates of a geometry between -180..180 and 0..360.

Synopsis

geometry **ST_ShiftLongitude**(geometry geom);

Description

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 bug prevented this from working for MULTIPOINT. 1.3.4+ works with MULTIPOINT as well.

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

Enhanced: 2.0.0 support for Polyhedral surfaces and TIN was introduced.

NOTE: this function was renamed from "ST_Shift_Longitude" in 2.2.0

This function supports Polyhedral surfaces.

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

Examples

```
--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)
```

See Also

ST_WrapX

7.5.32 ST_WrapX

ST_WrapX — Wrap a geometry around an X value.

Synopsis

geometry **ST_WrapX**(geometry geom, float8 wrap, float8 move);

Description

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
This is useful to "recenter" long-lat input to have features of interest not spawned from one side to the other.

Availability: 2.3.0 requires GEOS

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

Examples

```
-- 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);
```

See Also

ST_ShiftLongitude

7.5.33 ST_SnapToGrid

ST_SnapToGrid — Snap all points of the input geometry to a regular grid.

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);

Description

Variant 1,2,3: Snap all points of the input geometry to the grid defined by its origin and cell size. Remove consecutive points falling on the same cell, eventually returning NULL if output points are not enough to define a geometry of the given type. Collapsed geometries in a collection are stripped from it. Useful for reducing precision.

Variant 4: Introduced 1.1.0 - Snap all points of the input geometry to the grid defined by its origin (the second argument, must be a point) and cell sizes. Specify 0 as size for any dimension you don't want to snap to a grid.

Note
The returned geometry might lose its simplicity (see ST_IsSimple).

Note
Before release 1.1.0 this function always returned a 2d geometry. Starting at 1.1.0 the returned geometry will have same dimensionality as the input one with higher dimension values untouched. Use the version taking a second geometry argument to define all grid dimensions.

Availability: 1.0.0RC1

Availability: 1.1.0 - Z and M support

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

Examples

```
--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)
```

See Also

ST_Snap, ST_AsEWKT, ST_AsText, ST_GeomFromText, ST_GeomFromEWKT, ST_Simplify

7.5.34 ST_Snap

ST_Snap — Snap segments and vertices of input geometry to vertices of a reference geometry.

Synopsis

geometry **ST_Snap**(geometry input, geometry reference, float tolerance);

Description

Snaps the vertices and segments of a geometry to another Geometry's vertices. A snap distance tolerance is used to control where snapping is performed. The result geometry is the input geometry with the vertices snapped. If no snapping occurs then the input geometry is returned unchanged.

Snapping one geometry to another can improve robustness for overlay operations by eliminating nearly-coincident edges (which cause problems during noding and intersection calculation).

Too much snapping can result in invalid topology being created, so the number and location of snapped vertices is decided using heuristics to determine when it is safe to snap. This can result in some potential snaps being omitted, however.

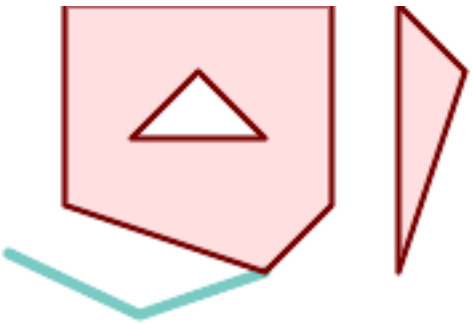
Note
The returned geometry might lose its simplicity (see ST_IsSimple) and validity (see ST_IsValid).

Performed by the GEOS module.

Availability: 2.0.0

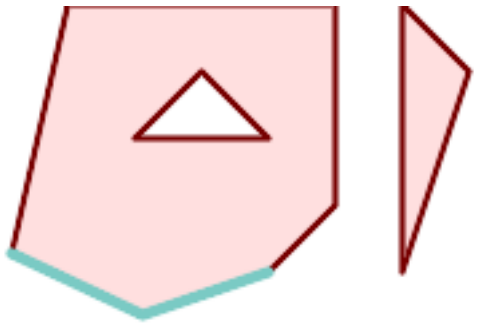
Examples

A multipolygon shown with a linestring (before any snapping)



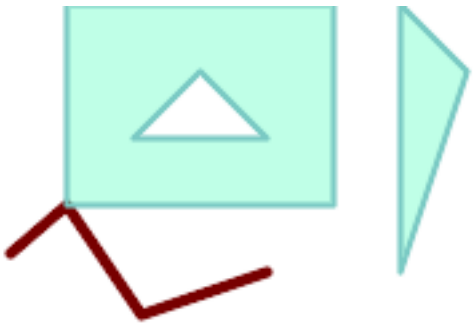
A multipolygon snapped to linestring to tolerance: 1.01 of distance. The new multipolygon is shown with reference linestring

```
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)))
```



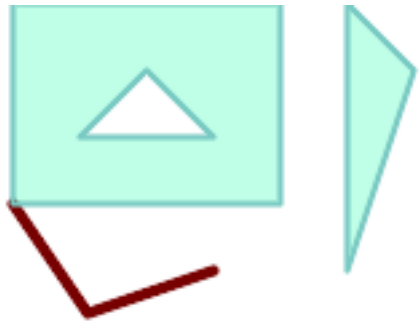
A multipolygon snapped to linestring to tolerance: 1.25 of distance. The new multipolygon is shown with reference linestring

```
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)))
```



The linestring snapped to the original multipolygon at tolerance 1.01 of distance. The new linestring is shown with reference multipolygon

```
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)
```



The linestring snapped to the original multipolygon at tolerance 1.25 of distance. The new linestring is shown with reference multipolygon

```
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)
```

See Also

ST_SnapToGrid

7.5.35 ST_SwapOrdinates

ST_SwapOrdinates — Returns a version of the given geometry with given ordinate values swapped.

Synopsis

geometry **ST_SwapOrdinates**(geometry geom, cstring ords);

Description

Returns a version of the given geometry with given ordinates swapped.

The `ords` parameter is a 2-characters string naming the ordinates to swap. Valid names are: x,y,z and m.

Availability: 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).

Example

```
-- 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)
```

See Also

ST_FlipCoordinates

7.6 Geometry Validation

7.6.1 ST_IsValid

ST_IsValid — Tests if a geometry is well-formed in 2D.

Synopsis

boolean **ST_IsValid**(geometry g);
boolean **ST_IsValid**(geometry g, integer flags);

Description

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
```

See Also

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);

Description

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.

Availability: 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     |        |
```

See Also

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);

Description

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
```

See Also

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);

Description

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.

Availability: 2.0.0

Enhanced: 2.0.1, speed improvements

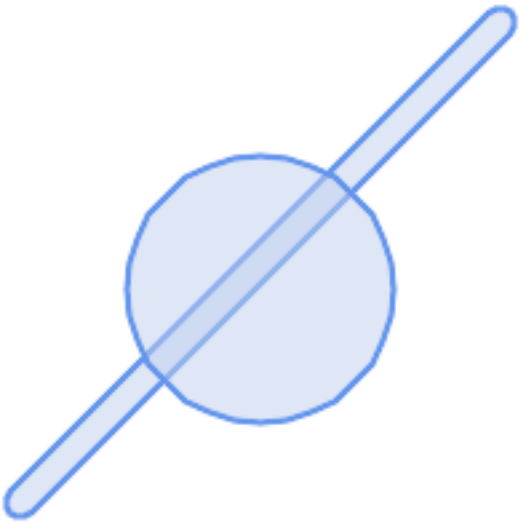
Enhanced: 2.1.0, added support for GEOMETRYCOLLECTION and MULTIPOINT.

Enhanced: 3.1.0, added removal of Coordinates with NaN values.

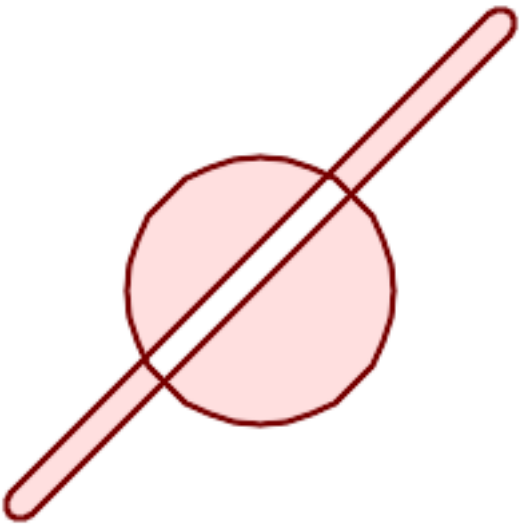
Enhanced: 3.2.0, added algorithm options, 'linework' and 'structure' which requires GEOS >= 3.10.0.

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

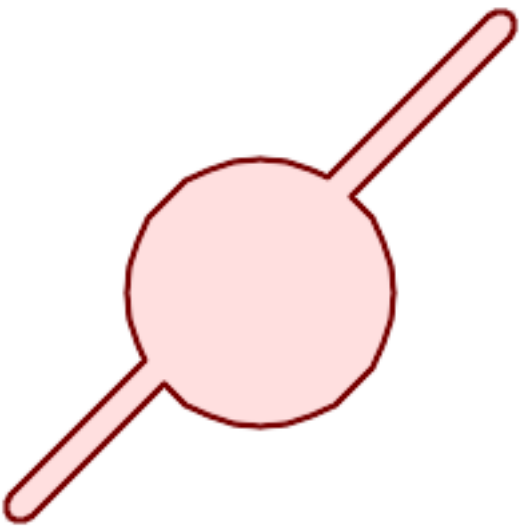
Examples



before_geom: MULTIPOLYGON of 2 overlapping polygons

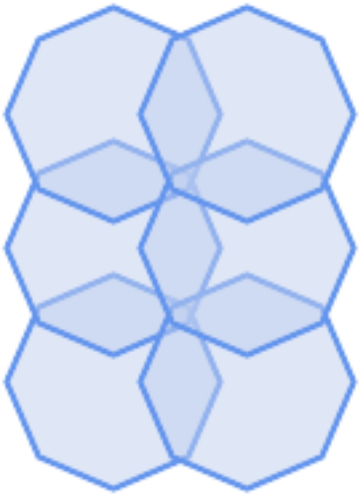


after_geom: MULTIPOLYGON of 4 non-overlapping polygons

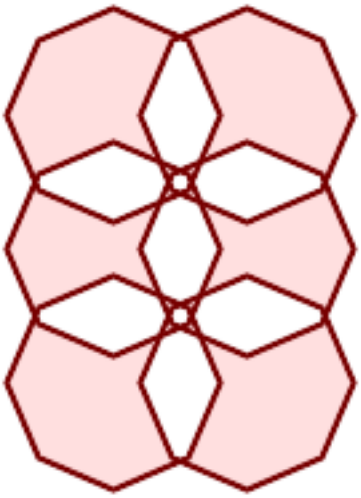


after_geom_structure: MULTIPOLYGON of 1 non-overlapping polygon

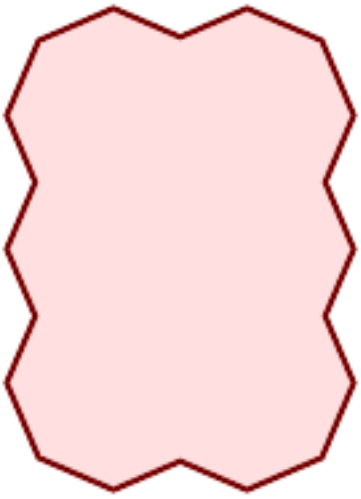
```
SELECT f.geom AS before_geom, ST_MakeValid(f.geom) AS after_geom, ST_MakeValid(f.geom,  ↩
    'method=structure') AS after_geom_structure
FROM (SELECT 'MULTIPOLYGON(((186 194,187 194,188 195,189 195,190 195,
191 195,192 195,193 194,194 194,194 194,193 195,192 195,191
```



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
```

See Also

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);

Description

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)
```

See Also

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);

Description

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)
```

See Also

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);

Description

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
```

See Also

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);

Description

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;
```

See Also

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);

Description

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)
```

See Also

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);

Description

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
```

See Also

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);

Description

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
```

See Also

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);

Description

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
```

See Also

postgis_srs_codes, postgis_srs, postgis_srs_search

7.7.9 postgis_srs_search

postgis_srs_search — Return metadata records for projected coordinate systems that have areas of usage that fully contain the bounds parameter.

Synopsis

setof record **postgis_srs_search**(geometry bounds, text auth_name=EPSG);

Description

Return a set of metadata records for projected coordinate systems that have areas of usage that fully contain the bounds parameter. Each record will have the `auth_name`, `auth_srid`, `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;
```

See Also

postgis_srs_codes, postgis_srs_all, postgis_srs

7.8 Geometry Input

7.8.1 Well-Known Text (WKT)

7.8.1.1 ST_BdPolyFromText

ST_BdPolyFromText — Construct a Polygon given an arbitrary collection of closed linestrings as a MultiLineString Well-Known text representation.

Synopsis

geometry **ST_BdPolyFromText**(text WKT, integer srid);

Description

Construct a Polygon given an arbitrary collection of closed linestrings as a MultiLineString Well-Known text representation.

Note
Throws an error if WKT is not a MULTILINESTRING. Throws an error if output is a MULTIPOLYGON; use ST_BdMPolyFromText in that case, or see ST_BuildArea() for a postgis-specific approach.

This method implements the OGC Simple Features Implementation Specification for SQL 1.1. s3.2.6.2

Performed by the GEOS module.

Availability: 1.1.0

See Also

ST_BuildArea, ST_BdMPolyFromText

7.8.1.2 ST_BdMPolyFromText

ST_BdMPolyFromText — Construct a MultiPolygon given an arbitrary collection of closed linestrings as a MultiLineString text representation Well-Known text representation.

Synopsis

geometry **ST_BdMPolyFromText**(text WKT, integer srid);

Description

Construct a Polygon given an arbitrary collection of closed linestrings, polygons, MultiLineStrings as Well-Known text representation.

Note
Throws an error if WKT is not a MULTILINESTRING. Forces MULTIPOLYGON output even when result is really only composed by a single POLYGON; use ST_BdPolyFromText if you're sure a single POLYGON will result from operation, or see ST_BuildArea() for a postgis-specific approach.

This method implements the OGC Simple Features Implementation Specification for SQL 1.1. s3.2.6.2

Performed by the GEOS module.

Availability: 1.1.0

See Also

ST_BuildArea, ST_BdPolyFromText

7.8.1.3 ST_GeogFromText

ST_GeogFromText — Return a specified geography value from Well-Known Text representation or extended (WKT).

Synopsis

geography **ST_GeogFromText**(text EWKT);

Description

Returns a geography object from the well-known text or extended well-known representation. SRID 4326 is assumed if unspecified. This is an alias for ST_GeographyFromText. Points are always expressed in long lat form.

Examples

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

See Also

ST_AsText, ST_GeographyFromText

7.8.1.4 ST_GeographyFromText

ST_GeographyFromText — Return a specified geography value from Well-Known Text representation or extended (WKT).

Synopsis

geography **ST_GeographyFromText**(text EWKT);

Description

Returns a geography object from the well-known text representation. SRID 4326 is assumed if unspecified.

See Also

ST_GeogFromText, ST_AsText

7.8.1.5 ST_GeomCollFromText

ST_GeomCollFromText — Makes a collection Geometry from collection WKT with the given SRID. If SRID is not given, it defaults to 0.

Synopsis

geometry **ST_GeomCollFromText**(text WKT, integer srid);
geometry **ST_GeomCollFromText**(text WKT);

Description

Makes a collection Geometry from the Well-Known-Text (WKT) representation with the given SRID. If SRID is not given, it defaults to 0.

OGC SPEC 3.2.6.2 - option SRID is from the conformance suite

Returns null if the WKT is not a GEOMETRYCOLLECTION

Note
If you are absolutely sure all your WKT geometries are collections, don't use this function. It is slower than ST_GeomFromText since it adds an additional validation step.

This method implements the OGC Simple Features Implementation Specification for SQL 1.1. s3.2.6.2

This method implements the SQL/MM specification.

Examples

```
SELECT ST_GeomCollFromText('GEOMETRYCOLLECTION(POINT(1 2),LINESTRING(1 2, 3 4))');
```

See Also

ST_GeomFromText, ST_SRID

7.8.1.6 ST_GeomFromEWKT

ST_GeomFromEWKT — Return a specified ST_Geometry value from Extended Well-Known Text representation (EWKT).

Synopsis

geometry **ST_GeomFromEWKT**(text EWKT);

Description

Constructs a PostGIS ST_Geometry object from the OGC Extended Well-Known text (EWKT) representation.

Note
The EWKT format is not an OGC standard, but an PostGIS specific format that includes the spatial reference system (SRID) identifier

Enhanced: 2.0.0 support for Polyhedral surfaces and TIN was introduced.

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_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))
)');
```

See Also

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);

Description

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.

Examples

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)
```

See Also

ST_AsMARC21

7.8.1.8 ST_GeometryFromText

ST_GeometryFromText — Return a specified ST_Geometry value from Well-Known Text representation (WKT). This is an alias name for ST_GeomFromText

Synopsis

geometry **ST_GeometryFromText**(text WKT);
geometry **ST_GeometryFromText**(text WKT, integer srid);

Description

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

See Also

ST_GeomFromText

7.8.1.9 ST_GeomFromText

ST_GeomFromText — Return a specified ST_Geometry value from Well-Known Text representation (WKT).

Synopsis

geometry **ST_GeomFromText**(text WKT);
geometry **ST_GeomFromText**(text WKT, integer srid);

Description

Constructs a PostGIS ST_Geometry object from the OGC Well-Known text representation.

Note
There are two variants of ST_GeomFromText function. The first takes no SRID and returns a geometry with no defined spatial reference system (SRID=0). The second takes a SRID as the second argument and returns a geometry that includes this SRID as part of its metadata.

This method implements the OGC Simple Features Implementation Specification for SQL 1.1. s3.2.6.2 - option SRID is from the conformance suite.

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
Changed: 2.0.0 In prior versions of PostGIS ST_GeomFromText('GEOMETRYCOLLECTION(EMPTY)') was allowed. This is now illegal in PostGIS 2.0.0 to better conform with SQL/MM standards. This should now be written as ST_GeomFromText('GEOMETRYCOLLECTION EMPTY')

Examples

```
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)');
```

See Also

ST_GeomFromEWKT, ST_GeomFromWKB, ST_SRID

7.8.1.10 ST_LineFromText

ST_LineFromText — Makes a Geometry from WKT representation with the given SRID. If SRID is not given, it defaults to 0.

Synopsis

geometry **ST_LineFromText**(text WKT);
geometry **ST_LineFromText**(text WKT, integer srid);

Description

Makes a Geometry from WKT with the given SRID. If SRID is not given, it defaults to 0. If WKT passed in is not a LINESTRING, then null is returned.

Note
OGC SPEC 3.2.6.2 - option SRID is from the conformance suite.

Note
If you know all your geometries are LINESTRINGS, its more efficient to just use ST_GeomFromText. This just calls ST_GeomFromText and adds additional validation that it returns a linestring.

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

Examples

```
SELECT ST_LineFromText('LINESTRING(1 2, 3 4)') AS aline, ST_LineFromText('POINT(1 2)') AS  ↩
    null_return;
aline                              | null_return
-----------------------------------------------
0102000000020000000000000000F ... | t
```

See Also

ST_GeomFromText

7.8.1.11 ST_MLineFromText

ST_MLineFromText — Return a specified ST_MultiLineString value from WKT representation.

Synopsis

geometry **ST_MLineFromText**(text WKT, integer srid);
geometry **ST_MLineFromText**(text WKT);

Description

Makes a Geometry from Well-Known-Text (WKT) with the given SRID. If SRID is not given, it defaults to 0.

OGC SPEC 3.2.6.2 - option SRID is from the conformance suite

Returns null if the WKT is not a MULTILINESTRING

Note
If you are absolutely sure all your WKT geometries are points, don't use this function. It is slower than ST_GeomFromText since it adds an additional validation step.

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

Examples

```
SELECT ST_MLineFromText('MULTILINESTRING((1 2, 3 4), (4 5, 6 7))');
```

See Also

ST_GeomFromText

7.8.1.12 ST_MPointFromText

ST_MPointFromText — Makes a Geometry from WKT with the given SRID. If SRID is not given, it defaults to 0.

Synopsis

geometry **ST_MPointFromText**(text WKT, integer srid);
geometry **ST_MPointFromText**(text WKT);

Description

Makes a Geometry from WKT with the given SRID. If SRID is not given, it defaults to 0.

OGC SPEC 3.2.6.2 - option SRID is from the conformance suite

Returns null if the WKT is not a MULTIPOINT

Note
If you are absolutely sure all your WKT geometries are points, don't use this function. It is slower than ST_GeomFromText since it adds an additional validation step.

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

Examples

```
SELECT ST_MPointFromText('MULTIPOINT((1 2),(3 4))');
SELECT ST_MPointFromText('MULTIPOINT((-70.9590 42.1180),(-70.9611 42.1223))', 4326);
```

See Also

ST_GeomFromText

7.8.1.13 ST_MPolyFromText

ST_MPolyFromText — Makes a MultiPolygon Geometry from WKT with the given SRID. If SRID is not given, it defaults to 0.

Synopsis

geometry **ST_MPolyFromText**(text WKT, integer srid);
geometry **ST_MPolyFromText**(text WKT);

Description

Makes a MultiPolygon from WKT with the given SRID. If SRID is not given, it defaults to 0.

OGC SPEC 3.2.6.2 - option SRID is from the conformance suite

Throws an error if the WKT is not a MULTIPOLYGON

Note
If you are absolutely sure all your WKT geometries are multipolygons, don't use this function. It is slower than ST_GeomFromText since it adds an additional validation step.

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

Examples

```
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);
```

See Also

ST_GeomFromText, ST_SRID

7.8.1.14 ST_PointFromText

ST_PointFromText — Makes a point Geometry from WKT with the given SRID. If SRID is not given, it defaults to unknown.

Synopsis

geometry **ST_PointFromText**(text WKT);
geometry **ST_PointFromText**(text WKT, integer srid);

Description

Constructs a PostGIS ST_Geometry point object from the OGC Well-Known text representation. If SRID is not given, it defaults to unknown (currently 0). If geometry is not a WKT point representation, returns null. If completely invalid WKT, then throws an error.

Note
There are 2 variants of ST_PointFromText function, the first takes no SRID and returns a geometry with no defined spatial reference system. The second takes a spatial reference id as the second argument and returns an ST_Geometry that includes this srid as part of its meta-data. The srid must be defined in the spatial_ref_sys table.

Note
If you are absolutely sure all your WKT geometries are points, don't use this function. It is slower than ST_GeomFromText since it adds an additional validation step. If you are building points from long lat coordinates and care more about performance and accuracy than OGC compliance, use ST_MakePoint or OGC compliant alias ST_Point.

This method implements the OGC Simple Features Implementation Specification for SQL 1.1. s3.2.6.2 - option SRID is from the conformance suite.

This method implements the SQL/MM specification. SQL-MM 3: 6.1.8

Examples

```
SELECT ST_PointFromText('POINT(-71.064544 42.28787)');
SELECT ST_PointFromText('POINT(-71.064544 42.28787)', 4326);
```

See Also

ST_GeomFromText, ST_MakePoint, ST_Point, ST_SRID

7.8.1.15 ST_PolygonFromText

ST_PolygonFromText — Makes a Geometry from WKT with the given SRID. If SRID is not given, it defaults to 0.

Synopsis

geometry **ST_PolygonFromText**(text WKT);
geometry **ST_PolygonFromText**(text WKT, integer srid);

Description

Makes a Geometry from WKT with the given SRID. If SRID is not given, it defaults to 0. Returns null if WKT is not a polygon.

OGC SPEC 3.2.6.2 - option SRID is from the conformance suite

Note
If you are absolutely sure all your WKT geometries are polygons, don't use this function. It is slower than ST_GeomFromText since it adds an additional validation step.

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

Examples

```
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
------------------
0103000000010000000500000006...


SELECT ST_PolygonFromText('POINT(1 2)') IS NULL as point_is_notpoly;

point_is_not_poly
----------
t
```

See Also

ST_GeomFromText

7.8.1.16 ST_WKTToSQL

ST_WKTToSQL — Return a specified ST_Geometry value from Well-Known Text representation (WKT). This is an alias name for ST_GeomFromText

Synopsis

geometry **ST_WKTToSQL**(text WKT);

Description

This method implements the SQL/MM specification. SQL-MM 3: 5.1.34

See Also

ST_GeomFromText

7.8.2 Well-Known Binary (WKB)

7.8.2.1 ST_GeogFromWKB

ST_GeogFromWKB — Creates a geography instance from a Well-Known Binary geometry representation (WKB) or extended Well Known Binary (EWKB).

Synopsis

geography **ST_GeogFromWKB**(bytea wkb);

Description

The `ST_GeogFromWKB` function, takes a well-known binary representation (WKB) of a geometry or PostGIS Extended WKB and creates an instance of the appropriate geography type. This function plays the role of the Geometry Factory in SQL.

If SRID is not specified, it defaults to 4326 (WGS 84 long lat).

This method supports Circular Strings and Curves.

Examples

```
--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)
```

See Also

ST_GeogFromText, ST_AsBinary

7.8.2.2 ST_GeomFromEWKB

ST_GeomFromEWKB — Return a specified ST_Geometry value from Extended Well-Known Binary representation (EWKB).

Synopsis

geometry **ST_GeomFromEWKB**(bytea EWKB);

Description

Constructs a PostGIS ST_Geometry object from the OGC Extended Well-Known binary (EWKT) representation.

Note
The EWKB format is not an OGC standard, but a PostGIS specific format that includes the spatial reference system (SRID) identifier

Enhanced: 2.0.0 support for Polyhedral surfaces and TIN was introduced.

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

line string binary rep 0f LINESTRING(-71.160281 42.258729,-71.160837 42.259113,-71.161144 42.25932) in NAD 83 long lat (4269).

Note
NOTE: Even though byte arrays are delimited with \ and may have ', we need to escape both out with \ and '' if standard_conforming_strings is off. So it does not look exactly like its AsEWKB representation.

```
SELECT ST_GeomFromEWKB(E'\\001\\002\\000\\000 \\255\\020\\000\\000\\003\\000\\000\\000\\344 ↩
    J=
\\013B\\312Q\\300n\\303(\\010\\036!E@''\\277E''K
\\312Q\\300\\366{b\\235*!E@\\225|\\354.P\\312Q
\\300p\\231\\323e1!E@');
```

Note
In PostgreSQL 9.1+ - standard_conforming_strings is set to on by default, where as in past versions it was set to off. You can change defaults as needed for a single query or at the database or server level. Below is how you would do it with standard_conforming_strings = on. In this case we escape the ' with standard ansi ', but slashes are not escaped

```
      set standard_conforming_strings = on;
SELECT ST_GeomFromEWKB('\001\002\000\000 \255\020\000\000\003\000\000\000\344J=\012\013B
    \312Q\300n\303(\010\036!E@''\277E''K\012\312Q\300\366{b\235*!E@\225|\354.P\312Q\012\300 ↩
       p\231\323e1')
```

See Also

ST_AsBinary, ST_AsEWKB, ST_GeomFromWKB

7.8.2.3 ST_GeomFromWKB

ST_GeomFromWKB — Creates a geometry instance from a Well-Known Binary geometry representation (WKB) and optional SRID.

Synopsis

geometry **ST_GeomFromWKB**(bytea geom);
geometry **ST_GeomFromWKB**(bytea geom, integer srid);

Description

The `ST_GeomFromWKB` function, takes a well-known binary representation of a geometry and a Spatial Reference System ID (`SRID`) and creates an instance of the appropriate geometry type. This function plays the role of the Geometry Factory in SQL. This is an alternate name for ST_WKBToSQL.

If SRID is not specified, it defaults to 0 (Unknown).

This method implements the OGC Simple Features Implementation Specification for SQL 1.1. s3.2.7.2 - the optional SRID is from the conformance suite

This method implements the SQL/MM specification. SQL-MM 3: 5.1.41

This method supports Circular Strings and Curves.

Examples

```
--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)
```

See Also

ST_WKBToSQL, ST_AsBinary, ST_GeomFromEWKB

7.8.2.4 ST_LineFromWKB

ST_LineFromWKB — Makes a LINESTRING from WKB with the given SRID

Synopsis

geometry **ST_LineFromWKB**(bytea WKB);
geometry **ST_LineFromWKB**(bytea WKB, integer srid);

Description

The ST_LineFromWKB function, takes a well-known binary representation of geometry and a Spatial Reference System ID (SRID) and creates an instance of the appropriate geometry type - in this case, a LINESTRING geometry. This function plays the role of the Geometry Factory in SQL.

If an SRID is not specified, it defaults to 0. NULL is returned if the input bytea does not represent a LINESTRING.

Note
OGC SPEC 3.2.6.2 - option SRID is from the conformance suite.

Note
If you know all your geometries are LINESTRINGs, its more efficient to just use ST_GeomFromWKB. This function just calls ST_GeomFromWKB and adds additional validation that it returns a linestring.

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

Examples

```
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
```

See Also

ST_GeomFromWKB, ST_LinestringFromWKB

7.8.2.5 ST_LinestringFromWKB

ST_LinestringFromWKB — Makes a geometry from WKB with the given SRID.

Synopsis

geometry **ST_LinestringFromWKB**(bytea WKB);
geometry **ST_LinestringFromWKB**(bytea WKB, integer srid);

Description

The `ST_LinestringFromWKB` function, takes a well-known binary representation of geometry and a Spatial Reference System ID (`SRID`) and creates an instance of the appropriate geometry type - in this case, a `LINESTRING` geometry. This function plays the role of the Geometry Factory in SQL.

If an SRID is not specified, it defaults to 0. `NULL` is returned if the input `bytea` does not represent a `LINESTRING` geometry. This an alias for ST_LineFromWKB.

Note
OGC SPEC 3.2.6.2 - optional SRID is from the conformance suite.

Note
If you know all your geometries are `LINESTRING`s, it's more efficient to just use ST_GeomFromWKB. This function just calls ST_GeomFromWKB and adds additional validation that it returns a `LINESTRING`.

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

Examples

```
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
```

See Also

ST_GeomFromWKB, ST_LineFromWKB

7.8.2.6 ST_PointFromWKB

ST_PointFromWKB — Makes a geometry from WKB with the given SRID

Synopsis

geometry **ST_GeomFromWKB**(bytea geom);
geometry **ST_GeomFromWKB**(bytea geom, integer srid);

Description

The `ST_PointFromWKB` function, takes a well-known binary representation of geometry and a Spatial Reference System ID (`SRID`) and creates an instance of the appropriate geometry type - in this case, a `POINT` geometry. This function plays the role of the Geometry Factory in SQL.

If an SRID is not specified, it defaults to 0. `NULL` is returned if the input `bytea` does not represent a `POINT` geometry.

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.

Examples

```
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)
```

See Also

ST_GeomFromWKB, ST_LineFromWKB

7.8.2.7 ST_WKBToSQL

ST_WKBToSQL — Return a specified ST_Geometry value from Well-Known Binary representation (WKB). This is an alias name for ST_GeomFromWKB that takes no srid

Synopsis

geometry **ST_WKBToSQL**(bytea WKB);

Description

This method implements the SQL/MM specification. SQL-MM 3: 5.1.36

See Also

ST_GeomFromWKB

7.8.3 Other Formats

7.8.3.1 ST_Box2dFromGeoHash

ST_Box2dFromGeoHash — Return a BOX2D from a GeoHash string.

Synopsis

box2d **ST_Box2dFromGeoHash**(text geohash, integer precision=full_precision_of_geohash);

Description

Return a BOX2D from a GeoHash string.

If no `precision` is specified ST_Box2dFromGeoHash returns a BOX2D based on full precision of the input GeoHash string.

If `precision` is specified ST_Box2dFromGeoHash will use that many characters from the GeoHash to create the BOX2D. Lower precision values results in larger BOX2Ds and larger values increase the precision.

Availability: 2.1.0

Examples

```
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)
```

See Also

ST_GeoHash, ST_GeomFromGeoHash, ST_PointFromGeoHash

7.8.3.2 ST_GeomFromGeoHash

ST_GeomFromGeoHash — Return a geometry from a GeoHash string.

Synopsis

geometry **ST_GeomFromGeoHash**(text geohash, integer precision=full_precision_of_geohash);

Description

Return a geometry from a GeoHash string. The geometry will be a polygon representing the GeoHash bounds.

If no `precision` is specified ST_GeomFromGeoHash returns a polygon based on full precision of the input GeoHash string.

If `precision` is specified ST_GeomFromGeoHash will use that many characters from the GeoHash to create the polygon.

Availability: 2.1.0

Examples

```
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))
```

See Also

ST_GeoHash,ST_Box2dFromGeoHash, ST_PointFromGeoHash

7.8.3.3 ST_GeomFromGML

ST_GeomFromGML — Takes as input GML representation of geometry and outputs a PostGIS geometry object

Synopsis

geometry **ST_GeomFromGML**(text geomgml);
geometry **ST_GeomFromGML**(text geomgml, integer srid);

Description

Constructs a PostGIS ST_Geometry object from the OGC GML representation.

ST_GeomFromGML works only for GML Geometry fragments. It throws an error if you try to use it on a whole GML document.

OGC GML versions supported:

- GML 3.2.1 Namespace
- GML 3.1.1 Simple Features profile SF-2 (with GML 3.1.0 and 3.0.0 backward compatibility)
- GML 2.1.2

OGC GML standards, cf: http://www.opengeospatial.org/standards/gml:

Availability: 1.5, requires libxml2 1.6+

Enhanced: 2.0.0 support for Polyhedral surfaces and TIN was introduced.

Enhanced: 2.0.0 default srid optional parameter added.

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 allow mixed dimensions (2D and 3D inside the same MultiGeometry for instance). As PostGIS geometries don't, ST_GeomFromGML convert the whole geometry to 2D if a missing Z dimension is found once.

GML support mixed SRS inside the same MultiGeometry. As PostGIS geometries don't, ST_GeomFromGML, in this case, reproject all subgeometries to the SRS root node. If no srsName attribute available for the GML root node, the function throw an error.

ST_GeomFromGML function is not pedantic about an explicit GML namespace. You could avoid to mention it explicitly for common usages. But you need it if you want to use XLink feature inside GML.

Note
ST_GeomFromGML function not support SQL/MM curves geometries.

Examples - A single geometry with srsName

```
SELECT ST_GeomFromGML($$
    <gml:LineString xmlns:gml="http://www.opengis.net/gml"
      srsName="EPSG:4269">
        <gml:coordinates>
            -71.16028,42.258729 -71.160837,42.259112 -71.161143,42.25932
        </gml:coordinates>
    </gml:LineString>
$$);
```

Examples - XLink usage

```
SELECT ST_GeomFromGML($$
    <gml:LineString xmlns:gml="http://www.opengis.net/gml"
            xmlns:xlink="http://www.w3.org/1999/xlink"
            srsName="urn:ogc:def:crs:EPSG::4269">
        <gml:pointProperty>
            <gml:Point gml:id="p1"><gml:pos>42.258729 -71.16028</gml:pos></gml:Point>
        </gml:pointProperty>
        <gml:pos>42.259112 -71.160837</gml:pos>
        <gml:pointProperty>
            <gml:Point xlink:type="simple" xlink:href="#p1"/>
        </gml:pointProperty>
    </gml:LineString>
$$);
```

Examples - Polyhedral Surface

```
SELECT ST_AsEWKT(ST_GeomFromGML('
<gml:PolyhedralSurface xmlns:gml="http://www.opengis.net/gml">
<gml:polygonPatches>
  <gml:PolygonPatch>
    <gml:exterior>
      <gml:LinearRing><gml:posList srsDimension="3">0 0 0 0 0 1 0 1 1 0 1 0 0 0 0</gml: ↩
          posList></gml:LinearRing>
    </gml:exterior>
  </gml:PolygonPatch>
  <gml:PolygonPatch>
    <gml:exterior>
  <gml:LinearRing><gml:posList srsDimension="3">0 0 0 0 1 0 1 1 0 1 0 0 0 0 0</gml:posList ↩
     ></gml:LinearRing>
    </gml:exterior>
  </gml:PolygonPatch>
  <gml:PolygonPatch>
    <gml:exterior>
  <gml:LinearRing><gml:posList srsDimension="3">0 0 0 1 0 0 1 0 1 0 0 1 0 0 0</gml:posList ↩
     ></gml:LinearRing>
    </gml:exterior>
  </gml:PolygonPatch>
  <gml:PolygonPatch>
    <gml:exterior>
  <gml:LinearRing><gml:posList srsDimension="3">1 1 0 1 1 1 1 0 1 1 0 0 1 1 0</gml:posList ↩
     ></gml:LinearRing>
    </gml:exterior>
  </gml:PolygonPatch>
  <gml:PolygonPatch>
    <gml:exterior>
  <gml:LinearRing><gml:posList srsDimension="3">0 1 0 0 1 1 1 1 1 1 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)))
```

See Also

Section 2.2.3, ST_AsGML, ST_GMLToSQL

7.8.3.4 ST_GeomFromGeoJSON

ST_GeomFromGeoJSON — Takes as input a geojson representation of a geometry and outputs a PostGIS geometry object

Synopsis

geometry **ST_GeomFromGeoJSON**(text geomjson);
geometry **ST_GeomFromGeoJSON**(json geomjson);
geometry **ST_GeomFromGeoJSON**(jsonb geomjson);

Description

Constructs a PostGIS geometry object from the GeoJSON representation.

ST_GeomFromGeoJSON works only for JSON Geometry fragments. It throws an error if you try to use it on a whole JSON document.

Enhanced: 3.0.0 parsed geometry defaults to SRID=4326 if not specified otherwise.

Enhanced: 2.5.0 can now accept json and jsonb as inputs.

Availability: 2.0.0 requires - JSON-C >= 0.9

Note
If you do not have JSON-C enabled, support you will get an error notice instead of seeing an output. To enable JSON-C, run configure --with-jsondir=/path/to/json-c. See Section 2.2.3 for details.

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

Examples

```
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)
```

See Also

ST_AsText, ST_AsGeoJSON, Section 2.2.3

7.8.3.5 ST_GeomFromKML

ST_GeomFromKML — Takes as input KML representation of geometry and outputs a PostGIS geometry object

Synopsis

geometry **ST_GeomFromKML**(text geomkml);

Description

Constructs a PostGIS ST_Geometry object from the OGC KML representation.

ST_GeomFromKML works only for KML Geometry fragments. It throws an error if you try to use it on a whole KML document.

OGC KML versions supported:

- KML 2.2.0 Namespace

OGC KML standards, cf: http://www.opengeospatial.org/standards/kml:

Availability: 1.5, requires libxml2 2.6+

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

Note
ST_GeomFromKML function not support SQL/MM curves geometries.

Examples - A single geometry with srsName

```
SELECT ST_GeomFromKML($$
    <LineString>
        <coordinates>-71.1663,42.2614
            -71.1667,42.2616</coordinates>
    </LineString>
$$);
```

See Also

Section 2.2.3, ST_AsKML

7.8.3.6 ST_GeomFromTWKB

ST_GeomFromTWKB — Creates a geometry instance from a TWKB ("Tiny Well-Known Binary") geometry representation.

Synopsis

geometry **ST_GeomFromTWKB**(bytea twkb);

Description

The ST_GeomFromTWKB function, takes a a TWKB ("Tiny Well-Known Binary") geometry representation (WKB) and creates an instance of the appropriate geometry type.

Examples

```
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)
```

See Also

ST_AsTWKB

7.8.3.7 ST_GMLToSQL

ST_GMLToSQL — Return a specified ST_Geometry value from GML representation. This is an alias name for ST_GeomFromGML

Synopsis

geometry **ST_GMLToSQL**(text geomgml);
geometry **ST_GMLToSQL**(text geomgml, integer srid);

Description

This method implements the SQL/MM specification. SQL-MM 3: 5.1.50 (except for curves support).

Availability: 1.5, requires libxml2 1.6+

Enhanced: 2.0.0 support for Polyhedral surfaces and TIN was introduced.

Enhanced: 2.0.0 default srid optional parameter added.

See Also

Section 2.2.3, ST_GeomFromGML, ST_AsGML

7.8.3.8 ST_LineFromEncodedPolyline

ST_LineFromEncodedPolyline — Creates a LineString from an Encoded Polyline.

Synopsis

geometry **ST_LineFromEncodedPolyline**(text polyline, integer precision=5);

Description

Creates a LineString from an Encoded Polyline string.

Optional `precision` specifies how many decimal places will be preserved in Encoded Polyline. Value should be the same on encoding and decoding, or coordinates will be incorrect.

See http://developers.google.com/maps/documentation/utilities/polylinealgorithm

Availability: 2.2.0

Examples

```
-- 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)
```

See Also

ST_AsEncodedPolyline

7.8.3.9 ST_PointFromGeoHash

ST_PointFromGeoHash — Return a point from a GeoHash string.

Synopsis

point **ST_PointFromGeoHash**(text geohash, integer precision=full_precision_of_geohash);

Description

Return a point from a GeoHash string. The point represents the center point of the GeoHash.

If no `precision` is specified ST_PointFromGeoHash returns a point based on full precision of the input GeoHash string.

If `precision` is specified ST_PointFromGeoHash will use that many characters from the GeoHash to create the point.

Availability: 2.1.0

Examples

```
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)
```

See Also

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);

Description

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);

Description

Reads FlatGeobuf data (http://flatgeobuf.org). NOTE: PostgreSQL bytea cannot exceed 1GB.

`tabletype` reference to a table type.

`data` input FlatGeobuf data.

Availability: 3.2.0

7.9 Geometry Output

7.9.1 Well-Known Text (WKT)

7.9.1.1 ST_AsEWKT

ST_AsEWKT — Return the Well-Known Text (WKT) representation of the geometry with SRID meta data.

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);

Description

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.

Enhanced: 2.0.0 support for Geography, Polyhedral surfaces, Triangles and TIN was introduced.

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_AsEWKT('0103000020E6100000010000000500000000000000
      0000000000000000000000000000000000000000000000000000000000
      F03F000000000000F03F000000000000F03F000000000000F03
      F0000000000000000000000000000000000000000000000000'::geometry);

       st_asewkt
---------------------------------
SRID=4326;POLYGON((0 0,0 1,1 1,1 0,0 0))
(1 row)

SELECT ST_AsEWKT('0108000080030000000000000060 ↩
    E30A4100000000785C0241000000000000F03F0000000018
E20A4100000000485F024100000000000004000000000018
E20A4100000000305C0241000000000000840')

--st_asewkt---
CIRCULARSTRING(220268 150415 1,220227 150505 2,220227 150406 3)
```

See Also

ST_AsBinary, ST_AsEWKB, ST_AsText, ST_GeomFromEWKT

7.9.1.2 ST_AsText

ST_AsText — Return the Well-Known Text (WKT) representation of the geometry/geography without SRID metadata.

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);

Description

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.

Availability: 1.5 - support for geography was introduced.

Enhanced: 2.5 - optional parameter precision introduced.

This method implements the OGC Simple Features Implementation Specification for SQL 1.1. s2.1.1.1

This method implements the SQL/MM specification. SQL-MM 3: 5.1.25

This method supports Circular Strings and Curves.

Examples

```
SELECT ST_AsText('0103000000010000000500000000000000000000
0000000000000000000000000000000000000000000000000000
F03F000000000000F03F000000000000F03F000000000000F03
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)
```

See Also

ST_AsBinary, ST_AsEWKB, ST_AsEWKT, ST_GeomFromText

7.9.2 Well-Known Binary (WKB)

7.9.2.1 ST_AsBinary

ST_AsBinary — Return the OGC/ISO Well-Known Binary (WKB) representation of the geometry/geography without SRID meta data.

Synopsis

bytea **ST_AsBinary**(geometry g1);
bytea **ST_AsBinary**(geometry g1, text NDR_or_XDR);
bytea **ST_AsBinary**(geography g1);
bytea **ST_AsBinary**(geography g1, text NDR_or_XDR);

Description

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.

Enhanced: 2.0.0 support for Polyhedral surfaces, Triangles and TIN was introduced.

Enhanced: 2.0.0 support for higher coordinate dimensions was introduced.

Enhanced: 2.0.0 support for specifying endian with geography was introduced.

Availability: 1.5.0 geography support was introduced.

Changed: 2.0.0 Inputs to this function can not be unknown -- must be geometry. Constructs such as `ST_AsBinary('POINT(1 2)')` are no longer valid and you will get an `n st_asbinary(unknown) is not unique error`. Code like that needs to be changed to `ST_AsBinary('POINT(1 2)'::geometry);`. If that is not possible, then install `legacy.sql`.

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.

Examples

```
SELECT ST_AsBinary(ST_GeomFromText('POLYGON((0 0,0 1,1 1,1 0,0 0))',4326));

       st_asbinary
--------------------------------
\x0103000000010000000500000000000000000000000000000000000000000000000000000000000000000000000000000000
000000f03f000000000000f03f000000000000f03f000000000000f03f000000000000000000000000
0000000000000000000000000000
```

```
SELECT ST_AsBinary(ST_GeomFromText('POLYGON((0 0,0 1,1 1,1 0,0 0))',4326), 'XDR');
       st_asbinary
--------------------------------
\x0000000003000000010000000500000000000000000000000000000000000000000000000003ff000
00000000003ff00000000000003ff00000000000003ff0000000000000000000000000000000000
0000000000000000000000000
```

See Also

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);

Description

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.

Enhanced: 2.0.0 support for Polyhedral surfaces, Triangles and TIN was introduced.

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_AsEWKB(ST_GeomFromText('POLYGON((0 0,0 1,1 1,1 0,0 0))',4326));

       st_asewkb
--------------------------------
\x0103000020e6100000010000000500000000000000000000000000000000000000000000000000000
00000000000000f03f000000000000f03f000000000000f03f000000000000f03f00000000000000
0000000000000000000000000000000000
```

```
      SELECT ST_AsEWKB(ST_GeomFromText('POLYGON((0 0,0 1,1 1,1 0,0 0))',4326), 'XDR');
       st_asewkb
---------------------------------
\x00200000003000010e600000001000000050000000000000000000000000000000000000000000000000000000000000000
003ff00000000000003ff00000000000003ff00000000000003ff00000000000000000000000000000
00000000000000000000000000000000000000
```

See Also

ST_AsBinary, ST_GeomFromEWKB, ST_SRID

7.9.2.3 ST_AsHEXEWKB

ST_AsHEXEWKB — Returns a Geometry in HEXEWKB format (as text) using either little-endian (NDR) or big-endian (XDR) encoding.

Synopsis

text **ST_AsHEXEWKB**(geometry g1, text NDRorXDR);
text **ST_AsHEXEWKB**(geometry g1);

Description

Returns a Geometry in HEXEWKB format (as text) using either little-endian (NDR) or big-endian (XDR) encoding. If no encoding is specified, then NDR is used.

Note
Availability: 1.2.2

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

This method supports Circular Strings and Curves.

Examples

```
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
    F00000000000000000000000000000000000000000000000
```

7.9.3 Other Formats

7.9.3.1 ST_AsEncodedPolyline

ST_AsEncodedPolyline — Returns an Encoded Polyline from a LineString geometry.

Synopsis

text **ST_AsEncodedPolyline**(geometry geom, integer precision=5);

Description

Returns the geometry as an Encoded Polyline. This format is used by Google Maps with precision=5 and by Open Source Routing Machine with precision=5 and 6.

Optional `precision` specifies how many decimal places will be preserved in Encoded Polyline. Value should be the same on encoding and decoding, or coordinates will be incorrect.

Availability: 2.2.0

Examples

Basic

```
  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`@
```

Use in conjunction with geography linestring and geography segmentize, and put on google maps

```
-- the SQL for Boston to San Francisco, segments every 100 KM
  SELECT ST_AsEncodedPolyline(
    ST_Segmentize(
      ST_GeogFromText('LINESTRING(-71.0519 42.4935,-122.4483 37.64)'),
        100000)::geometry) As encodedFlightPath;
```

javascript will look something like this where $ variable you replace with query result

```
<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>
```

See Also

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);

Description

Return a FlatGeobuf representation (http://flatgeobuf.org) of a set of rows corresponding to a FeatureCollection. NOTE: PostgreSQL bytea cannot exceed 1GB.

`row` row data with at least a geometry column.

`index` toggle spatial index creation. Default is false.

`geom_name` is the name of the geometry column in the row data. If NULL it will default to the first found geometry column.

Availability: 3.2.0

7.9.3.3 ST_AsGeobuf

ST_AsGeobuf — Return a Geobuf representation of a set of rows.

Synopsis

bytea **ST_AsGeobuf**(anyelement set row);
bytea **ST_AsGeobuf**(anyelement row, text geom_name);

Description

Return a Geobuf representation (https://github.com/mapbox/geobuf) of a set of rows corresponding to a FeatureCollection. Every input geometry is analyzed to determine maximum precision for optimal storage. Note that Geobuf in its current form cannot be streamed so the full output will be assembled in memory.

`row` row data with at least a geometry column.

`geom_name` is the name of the geometry column in the row data. If NULL it will default to the first found geometry column.

Availability: 2.4.0

Examples

```
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 or feature in GeoJSON format.

Synopsis

text **ST_AsGeoJSON**(record feature, text geom_column="", integer maxdecimaldigits=9, boolean pretty_bool=false, text id_column='')
text **ST_AsGeoJSON**(geometry geom, integer maxdecimaldigits=9, integer options=8);
text **ST_AsGeoJSON**(geography geog, integer maxdecimaldigits=9, integer options=0);

Description

Returns a geometry as a GeoJSON "geometry" object, or a row as a GeoJSON "feature" object.

The resulting GeoJSON geometry and feature representations conform with the GeoJSON specifications RFC 7946, except when the parsed geometries are referenced with a CRS other than WGS84 longitude and latitude (EPSG:4326, urn:ogc:def:crs:OGC::CRS84); the GeoJSON geometry object will then have a short CRS SRID identifier attached by default. 2D and 3D Geometries are both supported. GeoJSON only supports SFS 1.1 geometry types (no curve support for example).

The `geom_column` parameter is used to distinguish between multiple geometry columns. If omitted, the first geometry column in the record will be determined. Conversely, passing the parameter will save column type lookups.

The `maxdecimaldigits` argument may be used to reduce the maximum number of decimal places used in output (defaults to 9). If you are using EPSG:4326 and are outputting the geometry only for display, `maxdecimaldigits=6` can be a good choice for many maps.

Warning
Using the *`maxdecimaldigits`* parameter can cause output geometry to become invalid. To avoid this use ST_ReducePrecision with a suitable gridsize first.

The `options` argument can be used to add BBOX or CRS in GeoJSON output:

- 0: means no option
- 1: GeoJSON BBOX
- 2: GeoJSON Short CRS (e.g EPSG:4326)
- 4: GeoJSON Long CRS (e.g urn:ogc:def:crs:EPSG::4326)
- 8: GeoJSON Short CRS if not EPSG:4326 (default)

The `id_column` parameter is used to set the "id" member of the returned GeoJSON features. As per GeoJSON RFC, this SHOULD be used whenever a feature has a commonly used identifier, such as a primary key. When not specified, the produced features will not get an "id" member and any columns other than the geometry, including any potential keys, will just end up inside the feature's "properties" member.

The GeoJSON specification states that polygons are oriented using the Right-Hand Rule, and some clients require this orientation. This can be ensured by using ST_ForcePolygonCCW . The specification also requires that geometry be in the WGS84 coordinate system (SRID = 4326). If necessary geometry can be projected into WGS84 using ST_Transform: `ST_Transform( geom, 4326 )`.

GeoJSON can be tested and viewed online at geojson.io and geojsonlint.com. It is widely supported by web mapping frameworks:

- OpenLayers GeoJSON Example
- Leaflet GeoJSON Example
- Mapbox GL GeoJSON Example

Availability: 1.3.4

Availability: 1.5.0 geography support was introduced.

Changed: 2.0.0 support default args and named args.

Changed: 3.0.0 support records as input

Changed: 3.0.0 output SRID if not EPSG:4326.

Changed: 3.5.0 allow specifying the column containing the feature id

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

Examples

Generate a FeatureCollection:

```
SELECT json_build_object(
    'type', 'FeatureCollection',
    'features', json_agg(ST_AsGeoJSON(t.*, id_column => 'id')::json)
    )
FROM ( VALUES (1, 'one', 'POINT(1 1)'::geometry),
              (2, 'two', 'POINT(2 2)'),
              (3, 'three', 'POINT(3 3)')
     ) as t(id, name, geom);
```

```
{"type" : "FeatureCollection", "features" : [{"type": "Feature", "geometry": {"type":"Point ↩
    ","coordinates":[1,1]}, "id": 1, "properties": {"name": "one"}}, {"type": "Feature", " ↩
    geometry": {"type":"Point","coordinates":[2,2]}, "id": 2, "properties": {"name": "two ↩
    "}}, {"type": "Feature", "geometry": {"type":"Point","coordinates":[3,3]}, "id": 3, " ↩
    properties": {"name": "three"}}]}
```

Generate a Feature:

```
SELECT ST_AsGeoJSON(t.*, id_column => 'id')
FROM (VALUES (1, 'one', 'POINT(1 1)'::geometry)) AS t(id, name, geom);
```

```
                                                st_asgeojson
-----------------------------------------------------------------------------------------------------

 {"type": "Feature", "geometry": {"type":"Point","coordinates":[1,1]}, "id": 1, "properties ↩
     ": {"name": "one"}}
```

Don’t forget to transform your data to WGS84 longitude, latitude to conform with the GeoJSON specification:

```
SELECT ST_AsGeoJSON(ST_Transform(geom,4326)) from fe_edges limit 1;
```

```
             st_asgeojson
-----------------------------------------------------------------------------------------------------



{"type":"MultiLineString","coordinates":[[[-89.734634999999997,31.492072000000000],
[-89.734955999999997,31.492237999999997]]]}
```

3D geometries are supported:

```
SELECT ST_AsGeoJSON('LINESTRING(1 2 3, 4 5 6)');
```

```
{"type":"LineString","coordinates":[[1,2,3],[4,5,6]]}
```

Options argument can be used to add BBOX and CRS in GeoJSON output:

```
SELECT ST_AsGeoJSON(ST_SetSRID('POINT(1 1)'::geometry, 4326), 9, 4|1);
```

```
{"type":"Point","crs":{"type":"name","properties":{"name":"urn:ogc:def:crs:EPSG ↩
    ::4326"}},"bbox":[1.000000000,1.000000000,1.000000000,1.000000000],"coordinates ↩
    ":[1,1]}
```

See Also

ST_GeomFromGeoJSON, ST_ForcePolygonCCW , ST_Transform

7.9.3.5 ST_AsGML

ST_AsGML — Return the geometry as a GML version 2 or 3 element.

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);

Description

Return the geometry as a Geography Markup Language (GML) element. The version parameter, if specified, may be either 2 or 3. If no version parameter is specified then the default is assumed to be 2. The `maxdecimaldigits` argument may be used to reduce the maximum number of decimal places used in output (defaults to 15).

Warning
Using the *`maxdecimaldigits`* parameter can cause output geometry to become invalid. To avoid this use ST_ReducePrecision with a suitable gridsize first.

GML 2 refer to 2.1.2 version, GML 3 to 3.1.1 version

The 'options' argument is a bitfield. It could be used to define CRS output type in GML output, and to declare data as lat/lon:

- 0: GML Short CRS (e.g EPSG:4326), default value
- 1: GML Long CRS (e.g urn:ogc:def:crs:EPSG::4326)
- 2: For GML 3 only, remove srsDimension attribute from output.
- 4: For GML 3 only, use <LineString> rather than <Curve> tag for lines.
- 16: Declare that data are lat/lon (e.g srid=4326). Default is to assume that data are planars. This option is useful for GML 3.1.1 output only, related to axis order. So if you set it, it will swap the coordinates so order is lat lon instead of database lon lat.
- 32: Output the box of the geometry (envelope).

The 'namespace prefix' argument may be used to specify a custom namespace prefix or no prefix (if empty). If null or omitted 'gml' prefix is used

Availability: 1.3.2

Availability: 1.5.0 geography support was introduced.

Enhanced: 2.0.0 prefix support was introduced. Option 4 for GML3 was introduced to allow using LineString instead of Curve tag for lines. GML3 Support for Polyhedral surfaces and TINS was introduced. Option 32 was introduced to output the box.

Changed: 2.0.0 use default named args

Enhanced: 2.1.0 id support was introduced, for GML 3.

Note
Only version 3+ of ST_AsGML supports Polyhedral Surfaces and 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).

Examples: 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>
```

Examples: 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>
```

See Also

ST_GeomFromGML

7.9.3.6 ST_AsKML

ST_AsKML — Return the geometry as a KML element.

Synopsis

text **ST_AsKML**(geometry geom, integer maxdecimaldigits=15, text nprefix=NULL);
text **ST_AsKML**(geography geog, integer maxdecimaldigits=15, text nprefix=NULL);

Description

Return the geometry as a Keyhole Markup Language (KML) element. default maximum number of decimal places is 15, default namespace is no prefix.

Warning
Using the *maxdecimaldigits* parameter can cause output geometry to become invalid. To avoid this use ST_ReducePrecision with a suitable gridsize first.

Note
Requires PostGIS be compiled with Proj support. Use PostGIS_Full_Version to confirm you have proj support compiled in.

Note
Availability: 1.2.2 - later variants that include version param came in 1.3.2

Note
Enhanced: 2.0.0 - Add prefix namespace, use default and named args

Note
Changed: 3.0.0 - Removed the "versioned" variant signature

Note
AsKML output will not work with geometries that do not have an SRID

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

Examples

```
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>
```

See Also

ST_AsSVG, ST_AsGML

7.9.3.7 ST_AsLatLonText

ST_AsLatLonText — Return the Degrees, Minutes, Seconds representation of the given point.

Synopsis

text **ST_AsLatLonText**(geometry pt, text format='');

Description

Returns the Degrees, Minutes, Seconds representation of the point.

Note
It is assumed the point is in a lat/lon projection. The X (lon) and Y (lat) coordinates are normalized in the output to the "normal" range (-180 to +180 for lon, -90 to +90 for lat).

The text parameter is a format string containing the format for the resulting text, similar to a date format string. Valid tokens are "D" for degrees, "M" for minutes, "S" for seconds, and "C" for cardinal direction (NSEW). DMS tokens may be repeated to indicate desired width and precision ("SSS.SSSS" means " 1.0023").

"M", "S", and "C" are optional. If "C" is omitted, degrees are shown with a "-" sign if south or west. If "S" is omitted, minutes will be shown as decimal with as many digits of precision as you specify. If "M" is also omitted, degrees are shown as decimal with as many digits precision as you specify.

If the format string is omitted (or zero-length) a default format will be used.

Availability: 2.0

Examples

Default format.

```
SELECT (ST_AsLatLonText('POINT (-3.2342342 -2.32498)'));
      st_aslatlontext
---------------------------
 2\textdegree{}19'29.928"S 3\textdegree{}14'3.243"W
```

Providing a format (same as the default).

```
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
```

Characters other than D, M, S, C and . are just passed through.

```
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
```

Signed degrees instead of cardinal directions.

```
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"
```

Decimal degrees.

```
SELECT (ST_AsLatLonText('POINT (-3.2342342 -2.32498)', 'D.DDDD degrees C'));
          st_aslatlontext
-----------------------------------
 2.3250 degrees S 3.2342 degrees W
```

Excessively large values are normalized.

```
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');

Description

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.

Examples

Converting a `POINT` to MARC21/XML formatted 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 formatted in decimal degrees

```
SELECT ST_AsMARC21('SRID=4326;POLYGON((-4.5792388916015625  ↩
    54.18172660239091,-4.56756591796875  ↩
    54.196993557130355,-4.546623229980469  ↩
    54.18313300502024,-4.5792388916015625 54.18172660239091))'::geometry,' ↩
    hddd.dddd');

<record xmlns="http://www.loc.gov/MARC21/slim">
    <datafield tag="034" ind1="1" ind2=" ">
        <subfield code="a">a</subfield>
        <subfield code="d">W004.5792</subfield>
        <subfield code="e">W004.5466</subfield>
        <subfield code="f">N054.1970</subfield>
        <subfield code="g">N054.1817</subfield>
    </datafield>
</record>
```

Converting a GEOMETRYCOLLECTION to MARC21/XML formatted in decimal minutes. The geometries order in the MARC21/XML output correspond to their order in the collection.

```
SELECT ST_AsMARC21('SRID=4326;GEOMETRYCOLLECTION(POLYGON((13.1  ↩
    52.65,13.516666666666667 52.65,13.516666666666667 52.38333333333333,13.1 ↩
     52.38333333333333,13.1 52.65)),POINT(-4.5 54.25))'::geometry,'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>
```

See Also

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);

Description

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.

Availability: 2.4.0

Note
From 3.0, Wagyu can be chosen at configure time to clip and validate MVT polygons. This library is faster and produces more correct results than the GEOS default, but it might drop small polygons.

Examples

```
SELECT ST_AsText(ST_AsMVTGeom(
  ST_GeomFromText('POLYGON ((0 0, 10 0, 10 5, 0 -5, 0 0))'),
  ST_MakeBox2D(ST_Point(0, 0), ST_Point(4096, 4096)),
  4096, 0, false));
                              st_astext
-----------------------------------------------------------------
 MULTIPOLYGON(((5 4096,10 4091,10 4096,5 4096)),((5 4096,0 4101,0 4096,5 4096)))
```

Canonical example for a Web Mercator tile using a computed tile bounds to query and clip geometry. This assumes the data.geom column has srid of 4326.

```
SELECT ST_AsMVTGeom(
            ST_Transform( geom, 3857 ),
            ST_TileEnvelope(12, 513, 412), extent => 4096, buffer => 64) AS geom
  FROM data
  WHERE geom && ST_Transform(ST_TileEnvelope(12, 513, 412, margin => (64.0 / 4096)),4326)
```

See Also

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);

Description

An aggregate function which returns a binary Mapbox Vector Tile representation of a set of rows corresponding to a tile layer. The rows must contain a geometry column which will be encoded as a feature geometry. The geometry must be in tile coordinate space and valid as per the MVT specification. ST_AsMVTGeom can be used to transform geometry into tile coordinate space. Other row columns are encoded as feature attributes.

The Mapbox Vector Tile format can store features with varying sets of attributes. To use this capability supply a JSONB column in the row data containing Json objects one level deep. The keys and values in the JSONB values will be encoded as feature attributes.

Tiles with multiple layers can be created by concatenating multiple calls to this function using `||` or `STRING_AGG`.

Important
Do not call with a `GEOMETRYCOLLECTION` as an element in the row. However you can use ST_AsMVTGeom to prepare a geometry collection for inclusion.

`row` row data with at least a geometry column.

`name` is the name of the layer. Default is the string "default".

`extent` is the tile extent in screen space as defined by the specification. Default is 4096.

`geom_name` is the name of the geometry column in the row data. Default is the first geometry column. Note that PostgreSQL by default automatically folds unquoted identifiers to lower case, which means that unless the geometry column is quoted, e.g. `"MyMVTGeom"`, this parameter must be provided as lowercase.

`feature_id_name` is the name of the Feature ID column in the row data. If NULL or negative the Feature ID is not set. The first column matching name and valid type (smallint, integer, bigint) will be used as Feature ID, and any subsequent column will be added as a property. JSON properties are not supported.

Enhanced: 3.0 - added support for Feature ID.

Enhanced: 2.5.0 - added support parallel query.

Availability: 2.4.0

Examples

```
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;
```

See Also

ST_AsMVTGeom, ST_TileEnvelope

7.9.3.11 ST_AsSVG

ST_AsSVG — Returns SVG path data for a geometry.

Synopsis

text **ST_AsSVG**(geometry geom, integer rel=0, integer maxdecimaldigits=15);
text **ST_AsSVG**(geography geog, integer rel=0, integer maxdecimaldigits=15);

Description

Return the geometry as Scalar Vector Graphics (SVG) path data. Use 1 as second argument to have the path data implemented in terms of relative moves, the default (or 0) uses absolute moves. Third argument may be used to reduce the maximum number of decimal digits used in output (defaults to 15). Point geometries will be rendered as cx/cy when 'rel' arg is 0, x/y when 'rel' is 1. Multipoint geometries are delimited by commas (","), GeometryCollection geometries are delimited by semicolons (";").

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

Changed: 2.0.0 to use default args and support named args

Note
Availability: 1.2.2. Availability: 1.4.0 Changed in PostGIS 1.4.0 to include L command in absolute path to conform to http://www.w3.org/TR/SVG/paths.html#PathDataBNF

This method supports Circular Strings and Curves.

Examples

```
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 — Returns the geometry as TWKB, aka "Tiny Well-Known Binary"

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);

Description

Returns the geometry in TWKB (Tiny Well-Known Binary) format. TWKB is a compressed binary format with a focus on minimizing the size of the output.

The decimal digits parameters control how much precision is stored in the output. By default, values are rounded to the nearest unit before encoding. If you want to transfer more precision, increase the number. For example, a value of 1 implies that the first digit to the right of the decimal point will be preserved.

The sizes and bounding boxes parameters control whether optional information about the encoded length of the object and the bounds of the object are included in the output. By default they are not. Do not turn them on unless your client software has a use for them, as they just use up space (and saving space is the point of TWKB).

The array-input form of the function is used to convert a collection of geometries and unique identifiers into a TWKB collection that preserves the identifiers. This is useful for clients that expect to unpack a collection and then access further information about the objects inside. You can create the arrays using the array_agg function. The other parameters operate the same as for the simple form of the function.

Note
The format specification is available online at https://github.com/TWKB/Specification, and code for building a JavaScript client can be found at https://github.com/TWKB/twkb.js.

Enhanced: 2.4.0 memory and speed improvements.

Availability: 2.2.0

Examples

```
SELECT ST_AsTWKB('LINESTRING(1 1,5 5)'::geometry);
                 st_astwkb
-------------------------------------------
\x02000202020808
```

To create an aggregate TWKB object including identifiers aggregate the desired geometries and objects first, using "array_agg()", then call the appropriate TWKB function.

```
SELECT ST_AsTWKB(array_agg(geom), array_agg(gid)) FROM mytable;
                 st_astwkb
-------------------------------------------
\x040402020400000202
```

See Also

ST_GeomFromTWKB, ST_AsBinary, ST_AsEWKB, ST_AsEWKT, ST_GeomFromText

7.9.3.13 ST_AsX3D

ST_AsX3D — Returns a Geometry in X3D xml node element format: ISO-IEC-19776-1.2-X3DEncodings-XML

Synopsis

text **ST_AsX3D**(geometry g1, integer maxdecimaldigits=15, integer options=0);

Description

Returns a geometry as an X3D xml formatted node element http://www.web3d.org/standards/number/19776-1. If `maxdecimaldigits` (precision) is not specified then defaults to 15.

Note
There are various options for translating PostGIS geometries to X3D since X3D geometry types don't map directly to PostGIS geometry types and some newer X3D types that might be better mappings we have avoided since most rendering tools don't currently support them. These are the mappings we have settled on. Feel free to post a bug ticket if you have thoughts on the idea or ways we can allow people to denote their preferred mappings.
Below is how we currently map PostGIS 2D/3D types to X3D types

The 'options' argument is a bitfield. For PostGIS 2.2+, this is used to denote whether to represent coordinates with X3D GeoCoordinates Geospatial node and also whether to flip the x/y axis. By default, `ST_AsX3D` outputs in database form (long,lat or X,Y), but X3D default of lat/lon, y/x may be preferred.

- 0: X/Y in database order (e.g. long/lat = X,Y is standard database order), default value, and non-spatial coordinates (just regular old Coordinate tag).
- 1: Flip X and Y. If used in conjunction with the GeoCoordinate option switch, then output will be default "latitude_first" and coordinates will be flipped as well.
- 2: Output coordinates in GeoSpatial GeoCoordinates. This option will throw an error if geometries are not in WGS 84 long lat (srid: 4326). This is currently the only GeoCoordinate type supported. Refer to X3D specs specifying a spatial reference system.. Default output will be `GeoCoordinate geoSystem='"GD" "WE" "longitude_first"'`. If you prefer the X3D default of `GeoCoordinate geoSystem='"GD" "WE" "latitude_first"'` use `(2 + 1) = 3`

PostGIS Type	2D X3D Type	3D X3D Type
LINESTRING	not yet implemented - will be PolyLine2D	LineSet
MULTILINESTRING	not yet implemented - will be PolyLine2D	IndexedLineSet
MULTIPOINT	Polypoint2D	PointSet
POINT	outputs the space delimited coordinates	outputs the space delimited coordinates
(MULTI) POLYGON, POLYHEDRALSURFACE	Invalid X3D markup	IndexedFaceSet (inner rings currently output as another faceset)
TIN	TriangleSet2D (Not Yet Implemented)	IndexedTriangleSet

Note
2D geometry support not yet complete. Inner rings currently just drawn as separate polygons. We are working on these.

Lots of advancements happening in 3D space particularly with X3D Integration with HTML5

There is also a nice open source X3D viewer you can use to view rendered geometries. Free Wrl http://freewrl.sourceforge.net/ binaries available for Mac, Linux, and Windows. Use the FreeWRL_Launcher packaged to view the geometries.

Also check out PostGIS minimalist X3D viewer that utilizes this function and x3dDom html/js open source toolkit.

Availability: 2.0.0: ISO-IEC-19776-1.2-X3DEncodings-XML

Enhanced: 2.2.0: Support for GeoCoordinates and axis (x/y, long/lat) flipping. Look at options for details.

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).

Example: Create a fully functional X3D document - This will generate a cube that is viewable in FreeWrl and other X3D viewers.

```
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

Example: An Octagon elevated 3 Units and decimal precision of 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>
```

Example: 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>
```

Example: Closed multilinestring (the boundary of a polygon with holes)

```
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 — Return a GeoHash representation of the geometry.

Synopsis

text **ST_GeoHash**(geometry geom, integer maxchars=full_precision_of_point);

Description

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.

Availability: 1.4.0

Note
ST_GeoHash requires input geometry to be in geographic (lon/lat) coordinates.

This method supports Circular Strings and Curves.

Examples

```
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
```

See Also

ST_GeomFromGeoHash, ST_PointFromGeoHash, ST_Box2dFromGeoHash

7.10 Operators

7.10.1 Bounding Box Operators

7.10.1.1 &&

&& — Returns TRUE if A's 2D bounding box intersects B's 2D bounding box.

Synopsis

boolean **&&**(geometry A , geometry B);
boolean **&&**(geography A , geography B);

Description

The && operator returns TRUE if the 2D bounding box of geometry A intersects the 2D bounding box of geometry B.

Note
This operand will make use of any indexes that may be available on the geometries.

Enhanced: 2.0.0 support for Polyhedral surfaces was introduced.

Availability: 1.5.0 support for geography was introduced.

This method supports Circular Strings and Curves.

This function supports Polyhedral surfaces.

Examples

```
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)
```

See Also

ST_Intersects, ST_Extent, |&>, &>, &<|, &<, ~, @

7.10.1.2 &&(geometry,box2df)

&&(geometry,box2df) — Returns TRUE if a geometry's (cached) 2D bounding box intersects a 2D float precision bounding box (BOX2DF).

Synopsis

boolean **&&**(geometry A , box2df B);

Description

The `&&` operator returns `TRUE` if the cached 2D bounding box of geometry A intersects the 2D bounding box B, using float precision. This means that if B is a (double precision) box2d, it will be internally converted to a float precision 2D bounding box (BOX2DF)

Note
This operand is intended to be used internally by BRIN indexes, more than by users.

Availability: 2.3.0 support for Block Range INdexes (BRIN) was introduced. Requires PostgreSQL 9.5+.

This method supports Circular Strings and Curves.

This function supports Polyhedral surfaces.

Examples

```
SELECT ST_Point(1,1) && ST_MakeBox2D(ST_Point(0,0), ST_Point(2,2)) AS overlaps;

 overlaps
----------
 t
(1 row)
```

See Also

&&(box2df,geometry), &&(box2df,box2df), ~(geometry,box2df), ~(box2df,geometry), ~(box2df,box2df), @(geometry,box2df), @(box2df,geometry), @(box2df,box2df)

7.10.1.3 &&(box2df,geometry)

&&(box2df,geometry) — Returns `TRUE` if a 2D float precision bounding box (BOX2DF) intersects a geometry's (cached) 2D bounding box.

Synopsis

boolean **&&**(box2df A , geometry B);

Description

The `&&` operator returns `TRUE` if the 2D bounding box A intersects the cached 2D bounding box of geometry B, using float precision. This means that if A is a (double precision) box2d, it will be internally converted to a float precision 2D bounding box (BOX2DF)

Note
This operand is intended to be used internally by BRIN indexes, more than by users.

Availability: 2.3.0 support for Block Range INdexes (BRIN) was introduced. Requires PostgreSQL 9.5+.

This method supports Circular Strings and Curves.

This function supports Polyhedral surfaces.

Examples

```
SELECT ST_MakeBox2D(ST_Point(0,0), ST_Point(2,2)) && ST_Point(1,1) AS overlaps;

 overlaps
----------
 t
(1 row)
```

See Also

&&(geometry,box2df), &&(box2df,box2df), ~(geometry,box2df), ~(box2df,geometry), ~(box2df,box2df), @(geometry,box2df), @(box2df,geometry), @(box2df,box2df)

7.10.1.4 &&(box2df,box2df)

&&(box2df,box2df) — Returns TRUE if two 2D float precision bounding boxes (BOX2DF) intersect each other.

Synopsis

boolean **&&**(box2df A , box2df B);

Description

The && operator returns TRUE if two 2D bounding boxes A and B intersect each other, using float precision. This means that if A (or B) is a (double precision) box2d, it will be internally converted to a float precision 2D bounding box (BOX2DF)

Note
This operator is intended to be used internally by BRIN indexes, more than by users.

Availability: 2.3.0 support for Block Range INdexes (BRIN) was introduced. Requires PostgreSQL 9.5+.

This method supports Circular Strings and Curves.

This function supports Polyhedral surfaces.

Examples

```
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)
```

See Also

&&(geometry,box2df), &&(box2df,geometry), ~(geometry,box2df), ~(box2df,geometry), ~(box2df,box2df), @(geometry,box2df), @(box2df,geometry), @(box2df,box2df)

7.10.1.5 &&&

&&& — Returns TRUE if A's n-D bounding box intersects B's n-D bounding box.

Synopsis

boolean **&&&**(geometry A , geometry B);

Description

The &&& operator returns TRUE if the n-D bounding box of geometry A intersects the n-D bounding box of geometry B.

Note
This operand will make use of any indexes that may be available on the geometries.

Availability: 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.

Examples: 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
```

Examples: 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
```

See Also

&&

7.10.1.6 &&&(geometry,gidx)

&&&(geometry,gidx) — Returns TRUE if a geometry's (cached) n-D bounding box intersects a n-D float precision bounding box (GIDX).

Synopsis

boolean **&&&**(geometry A , gidx B);

Description

The &&& operator returns TRUE if the cached n-D bounding box of geometry A intersects the n-D bounding box B, using float precision. This means that if B is a (double precision) box3d, it will be internally converted to a float precision 3D bounding box (GIDX)

Note
This operator is intended to be used internally by BRIN indexes, more than by users.

Availability: 2.3.0 support for Block Range INdexes (BRIN) was introduced. Requires PostgreSQL 9.5+.

This method supports Circular Strings and Curves.

This function supports Polyhedral surfaces.

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

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

Examples

```
SELECT ST_MakePoint(1,1,1) &&& ST_3DMakeBox(ST_MakePoint(0,0,0), ST_MakePoint(2,2,2)) AS  ↩
    overlaps;

 overlaps
----------
 t
(1 row)
```

See Also

&&&(gidx,geometry), &&&(gidx,gidx)

7.10.1.7 &&&(gidx,geometry)

&&&(gidx,geometry) — Returns TRUE if a n-D float precision bounding box (GIDX) intersects a geometry's (cached) n-D bounding box.

Synopsis

boolean **&&&**(gidx A , geometry B);

Description

The &&& operator returns TRUE if the n-D bounding box A intersects the cached n-D bounding box of geometry B, using float precision. This means that if A is a (double precision) box3d, it will be internally converted to a float precision 3D bounding box (GIDX)

Note
This operator is intended to be used internally by BRIN indexes, more than by users.

Availability: 2.3.0 support for Block Range INdexes (BRIN) was introduced. Requires PostgreSQL 9.5+.

This method supports Circular Strings and Curves.

This function supports Polyhedral surfaces.

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

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

Examples

```
SELECT ST_3DMakeBox(ST_MakePoint(0,0,0), ST_MakePoint(2,2,2)) &&& ST_MakePoint(1,1,1) AS  ↩
    overlaps;

 overlaps
----------
 t
(1 row)
```

See Also

&&&(geometry,gidx), &&&(gidx,gidx)

7.10.1.8 &&&(gidx,gidx)

&&&(gidx,gidx) — Returns TRUE if two n-D float precision bounding boxes (GIDX) intersect each other.

Synopsis

boolean **&&&**(gidx A , gidx B);

Description

The &&& operator returns TRUE if two n-D bounding boxes A and B intersect each other, using float precision. This means that if A (or B) is a (double precision) box3d, it will be internally converted to a float precision 3D bounding box (GIDX)

Note
This operator is intended to be used internally by BRIN indexes, more than by users.

Availability: 2.3.0 support for Block Range INdexes (BRIN) was introduced. Requires PostgreSQL 9.5+.

This method supports Circular Strings and Curves.

This function supports Polyhedral surfaces.

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

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

Examples

```
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)
```

See Also

&&&(geometry,gidx), &&&(gidx,geometry)

7.10.1.9 &<

&< — Returns TRUE if A's bounding box overlaps or is to the left of B's.

Synopsis

boolean **&<**(geometry A , geometry B);

Description

The &< operator returns TRUE if the bounding box of geometry A overlaps or is to the left of the bounding box of geometry B, or more accurately, overlaps or is NOT to the right of the bounding box of geometry B.

Note
This operand will make use of any indexes that may be available on the geometries.

Examples

```
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)
```

See Also

&&, |&>, &>, &<|

7.10.1.10 &<|

&<| — Returns TRUE if A's bounding box overlaps or is below B's.

Synopsis

boolean **&<|**(geometry A , geometry B);

Description

The &<| operator returns TRUE if the bounding box of geometry A overlaps or is below of the bounding box of geometry B, or more accurately, overlaps or is NOT above the bounding box of geometry B.

This method supports Circular Strings and Curves.

This function supports Polyhedral surfaces.

Note
This operand will make use of any indexes that may be available on the geometries.

Examples

```
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)
```

See Also

&&, |&>, &>, &<

7.10.1.11 &>

&> — Returns TRUE if A' bounding box overlaps or is to the right of B's.

Synopsis

boolean **&>**(geometry A , geometry B);

Description

The &> operator returns TRUE if the bounding box of geometry A overlaps or is to the right of the bounding box of geometry B, or more accurately, overlaps or is NOT to the left of the bounding box of geometry B.

Note
This operand will make use of any indexes that may be available on the geometries.

Examples

```
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)
```

See Also

&&, |&>, &<|, &<

7.10.1.12 <<

<< — Returns TRUE if A's bounding box is strictly to the left of B's.

Synopsis

boolean **<<**(geometry A , geometry B);

Description

The << operator returns TRUE if the bounding box of geometry A is strictly to the left of the bounding box of geometry B.

Note
This operand will make use of any indexes that may be available on the geometries.

Examples

```
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)
```

See Also

>>, |>>, <<|

7.10.1.13 <<|

<<| — Returns TRUE if A's bounding box is strictly below B's.

Synopsis

boolean **<<|**(geometry A , geometry B);

Description

The <<| operator returns TRUE if the bounding box of geometry A is strictly below the bounding box of geometry B.

Note
This operand will make use of any indexes that may be available on the geometries.

Examples

```
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)
```

See Also

<<, >>, |>>

7.10.1.14 =

= — Returns TRUE if the coordinates and coordinate order geometry/geography A are the same as the coordinates and coordinate order of geometry/geography B.

Synopsis

boolean =(geometry A , geometry B);
boolean =(geography A , geography B);

Description

The = operator returns TRUE if the coordinates and coordinate order geometry/geography A are the same as the coordinates and coordinate order of geometry/geography B. PostgreSQL uses the =, <, and > operators defined for geometries to perform internal orderings and comparison of geometries (ie. in a GROUP BY or ORDER BY clause).

Note
Only geometry/geography that are exactly equal in all respects, with the same coordinates, in the same order, are considered equal by this operator. For "spatial equality", that ignores things like coordinate order, and can detect features that cover the same spatial area with different representations, use ST_OrderingEquals or ST_Equals

Caution
This operand will NOT make use of any indexes that may be available on the geometries. For an index assisted exact equality test, combine = with &&.

Changed: 2.4.0, in prior versions this was bounding box equality not a geometric equality. If you need bounding box equality, use ~= instead.

This method supports Circular Strings and Curves.

This function supports Polyhedral surfaces.

Examples

```
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
```

See Also

ST_Equals, ST_OrderingEquals, ~=

7.10.1.15 >>

>> — Returns TRUE if A's bounding box is strictly to the right of B's.

Synopsis

boolean **>>**(geometry A , geometry B);

Description

The >> operator returns `TRUE` if the bounding box of geometry A is strictly to the right of the bounding box of geometry B.

Note
This operand will make use of any indexes that may be available on the geometries.

Examples

```
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)
```

See Also

<<, |>>, <<|

7.10.1.16 @

@ — Returns `TRUE` if A's bounding box is contained by B's.

Synopsis

boolean @(geometry A , geometry B);

Description

The `@` operator returns `TRUE` if the bounding box of geometry A is completely contained by the bounding box of geometry B.

Note
This operand will make use of any indexes that may be available on the geometries.

Examples

```
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)
```

See Also

~, &&

7.10.1.17 @(geometry,box2df)

@(geometry,box2df) — Returns `TRUE` if a geometry's 2D bounding box is contained into a 2D float precision bounding box (BOX2DF).

Synopsis

boolean @(geometry A , box2df B);

Description

The `@` operator returns `TRUE` if the A geometry's 2D bounding box is contained the 2D bounding box B, using float precision. This means that if B is a (double precision) box2d, it will be internally converted to a float precision 2D bounding box (BOX2DF)

Note
This operand is intended to be used internally by BRIN indexes, more than by users.

Availability: 2.3.0 support for Block Range INdexes (BRIN) was introduced. Requires PostgreSQL 9.5+.

This method supports Circular Strings and Curves.

This function supports Polyhedral surfaces.

Examples

```
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)
```

See Also

&&(geometry,box2df), &&(box2df,geometry), &&(box2df,box2df), ~(geometry,box2df), ~(box2df,geometry), ~(box2df,box2df), @(box2df,geometry), @(box2df,box2df)

7.10.1.18 @(box2df,geometry)

@(box2df,geometry) — Returns TRUE if a 2D float precision bounding box (BOX2DF) is contained into a geometry's 2D bounding box.

Synopsis

boolean @(box2df A , geometry B);

Description

The @ operator returns TRUE if the 2D bounding box A is contained into the B geometry's 2D bounding box, using float precision. This means that if B is a (double precision) box2d, it will be internally converted to a float precision 2D bounding box (BOX2DF)

Note
This operand is intended to be used internally by BRIN indexes, more than by users.

Availability: 2.3.0 support for Block Range INdexes (BRIN) was introduced. Requires PostgreSQL 9.5+.

This method supports Circular Strings and Curves.

This function supports Polyhedral surfaces.

Examples

```
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)
```

See Also

&&(geometry,box2df), &&(box2df,geometry), &&(box2df,box2df), ~(geometry,box2df), ~(box2df,geometry), ~(box2df,box2df), @(geometry,box2df), @(box2df,box2df)

7.10.1.19 @(box2df,box2df)

@(box2df,box2df) — Returns TRUE if a 2D float precision bounding box (BOX2DF) is contained into another 2D float precision bounding box.

Synopsis

boolean @(box2df A , box2df B);

Description

The @ operator returns TRUE if the 2D bounding box A is contained into the 2D bounding box B, using float precision. This means that if A (or B) is a (double precision) box2d, it will be internally converted to a float precision 2D bounding box (BOX2DF)

Note
This operand is intended to be used internally by BRIN indexes, more than by users.

Availability: 2.3.0 support for Block Range INdexes (BRIN) was introduced. Requires PostgreSQL 9.5+.

This method supports Circular Strings and Curves.

This function supports Polyhedral surfaces.

Examples

```
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)
```

See Also

&&(geometry,box2df), &&(box2df,geometry), &&(box2df,box2df), ~(geometry,box2df), ~(box2df,geometry), ~(box2df,box2df), @(geometry,box2df), @(box2df,geometry)

7.10.1.20 |&>

|&> — Returns TRUE if A's bounding box overlaps or is above B's.

Synopsis

boolean **|&>**(geometry A , geometry B);

Description

The |&> operator returns TRUE if the bounding box of geometry A overlaps or is above the bounding box of geometry B, or more accurately, overlaps or is NOT below the bounding box of geometry B.

Note
This operand will make use of any indexes that may be available on the geometries.

Examples

```
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)
```

See Also

&&, &>, &<|, &<

7.10.1.21 |>>

|>> — Returns TRUE if A's bounding box is strictly above B's.

Synopsis

boolean **|>>**(geometry A , geometry B);

Description

The |>> operator returns TRUE if the bounding box of geometry A is strictly above the bounding box of geometry B.

Note
This operand will make use of any indexes that may be available on the geometries.

Examples

```
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)
```

See Also

<<, >>, <<|

7.10.1.22 ~

~ — Returns TRUE if A's bounding box contains B's.

Synopsis

boolean ~(geometry A , geometry B);

Description

The ~ operator returns TRUE if the bounding box of geometry A completely contains the bounding box of geometry B.

Note
This operand will make use of any indexes that may be available on the geometries.

Examples

```
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)
```

See Also

@, &&

7.10.1.23 ~(geometry,box2df)

~(geometry,box2df) — Returns TRUE if a geometry's 2D bonding box contains a 2D float precision bounding box (GIDX).

Synopsis

boolean ~(geometry A , box2df B);

Description

The ~ operator returns TRUE if the 2D bounding box of a geometry A contains the 2D bounding box B, using float precision. This means that if B is a (double precision) box2d, it will be internally converted to a float precision 2D bounding box (BOX2DF)

Note
This operand is intended to be used internally by BRIN indexes, more than by users.

Availability: 2.3.0 support for Block Range INdexes (BRIN) was introduced. Requires PostgreSQL 9.5+.

This method supports Circular Strings and Curves.

This function supports Polyhedral surfaces.

Examples

```
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)
```

See Also

&&(geometry,box2df), &&(box2df,geometry), &&(box2df,box2df), ~(box2df,geometry), ~(box2df,box2df), @(geometry,box2df), @(box2df,geometry), @(box2df,box2df)

7.10.1.24 ~(box2df,geometry)

~(box2df,geometry) — Returns TRUE if a 2D float precision bounding box (BOX2DF) contains a geometry's 2D bonding box.

Synopsis

boolean ~(box2df A , geometry B);

Description

The ~ operator returns TRUE if the 2D bounding box A contains the B geometry's bounding box, using float precision. This means that if A is a (double precision) box2d, it will be internally converted to a float precision 2D bounding box (BOX2DF)

Note
This operand is intended to be used internally by BRIN indexes, more than by users.

Availability: 2.3.0 support for Block Range INdexes (BRIN) was introduced. Requires PostgreSQL 9.5+.

This method supports Circular Strings and Curves.

This function supports Polyhedral surfaces.

Examples

```
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)
```

See Also

&&(geometry,box2df), &&(box2df,geometry), &&(box2df,box2df), ~(geometry,box2df), ~(box2df,box2df), @(geometry,box2df), @(box2df,geometry), @(box2df,box2df)

7.10.1.25 ~(box2df,box2df)

~(box2df,box2df) — Returns TRUE if a 2D float precision bounding box (BOX2DF) contains another 2D float precision bounding box (BOX2DF).

Synopsis

boolean ~(box2df A , box2df B);

Description

The ~ operator returns TRUE if the 2D bounding box A contains the 2D bounding box B, using float precision. This means that if A is a (double precision) box2d, it will be internally converted to a float precision 2D bounding box (BOX2DF)

Note
This operand is intended to be used internally by BRIN indexes, more than by users.

Availability: 2.3.0 support for Block Range INdexes (BRIN) was introduced. Requires PostgreSQL 9.5+.

This method supports Circular Strings and Curves.

This function supports Polyhedral surfaces.

Examples

```
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)
```

See Also

&&(geometry,box2df), &&(box2df,geometry), &&(box2df,box2df), ~(geometry,box2df), ~(box2df,geometry), @(geometry,box2df), @(box2df,geometry), @(box2df,box2df)

7.10.1.26 ~=

~= — Returns TRUE if A's bounding box is the same as B's.

Synopsis

boolean **~=**(geometry A , geometry B);

Description

The ~= operator returns TRUE if the bounding box of geometry/geography A is the same as the bounding box of geometry/geography B.

Note
This operand will make use of any indexes that may be available on the geometries.

Availability: 1.5.0 changed behavior

This function supports Polyhedral surfaces.

Warning
This operator has changed behavior in PostGIS 1.5 from testing for actual geometric equality to only checking for bounding box equality. To complicate things it also depends on if you have done a hard or soft upgrade which behavior your database has. To find out which behavior your database has you can run the query below. To check for true equality use ST_OrderingEquals or ST_Equals.

Examples

```
select 'LINESTRING(0 0, 1 1)'::geometry ~= 'LINESTRING(0 1, 1 0)'::geometry as equality;
 equality   |
------------------+
          t     |
```

See Also

ST_Equals, ST_OrderingEquals, =

7.10.2 Distance Operators

7.10.2.1 <->

<-> — Returns the 2D distance between A and B.

Synopsis

double precision **<->**(geometry A , geometry B);
double precision **<->**(geography A , geography B);

Description

The <-> operator returns the 2D distance between two geometries. Used in the "ORDER BY" clause provides index-assisted nearest-neighbor result sets. For PostgreSQL below 9.5 only gives centroid distance of bounding boxes and for PostgreSQL 9.5+, does true KNN distance search giving true distance between geometries, and distance sphere for geographies.

Note
This operand will make use of 2D GiST indexes that may be available on the geometries. It is different from other operators that use spatial indexes in that the spatial index is only used when the operator is in the ORDER BY clause.

Note
Index only kicks in if one of the geometries is a constant (not in a subquery/cte). e.g. 'SRID=3005;POINT(1011102 450541)'::geometry instead of a.geom

Refer to PostGIS workshop: Nearest-Neighbor Searching for a detailed example.

Enhanced: 2.2.0 -- True KNN ("K nearest neighbor") behavior for geometry and geography for PostgreSQL 9.5+. Note for geography KNN is based on sphere rather than spheroid. For PostgreSQL 9.4 and below, geography support is new but only supports centroid box.

Changed: 2.2.0 -- For PostgreSQL 9.5 users, old Hybrid syntax may be slower, so you'll want to get rid of that hack if you are running your code only on PostGIS 2.2+ 9.5+. See examples below.

Availability: 2.0.0 -- Weak KNN provides nearest neighbors based on geometry centroid distances instead of true distances. Exact results for points, inexact for all other types. Available for PostgreSQL 9.1+

Examples

```
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)
```

If you run "EXPLAIN ANALYZE" on the two queries you would see a performance improvement for the second.

For users running with PostgreSQL < 9.5, use a hybrid query to find the true nearest neighbors. First a CTE query using the index-assisted KNN, then an exact query to get correct ordering:

```
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)
```

See Also

ST_DWithin, ST_Distance, <#>

7.10.2.2 |=|

|=| — Returns the distance between A and B trajectories at their closest point of approach.

Synopsis

double precision **|=|**(geometry A , geometry B);

Description

The | = | operator returns the 3D distance between two trajectories (See ST_IsValidTrajectory). This is the same as ST_DistanceCPA but as an operator it can be used for doing nearest neighbor searches using an N-dimensional index (requires PostgreSQL 9.5.0 or higher).

Note
This operand will make use of ND GiST indexes that may be available on the geometries. It is different from other operators that use spatial indexes in that the spatial index is only used when the operator is in the ORDER BY clause.

Note
Index only kicks in if one of the geometries is a constant (not in a subquery/cte). e.g. 'SRID=3005;LINESTRINGM(0 0 0,0 0 1)'::geometry instead of a.geom

Availability: 2.2.0. Index-supported only available for PostgreSQL 9.5+

Examples

```
-- 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)
```

See Also

ST_DistanceCPA, ST_ClosestPointOfApproach, ST_IsValidTrajectory

7.10.2.3 <#>

<#> — Returns the 2D distance between A and B bounding boxes.

Synopsis

double precision **<#>**(geometry A , geometry B);

Description

The <#> operator returns distance between two floating point bounding boxes, possibly reading them from a spatial index (PostgreSQL 9.1+ required). Useful for doing nearest neighbor **approximate** distance ordering.

Note
This operand will make use of any indexes that may be available on the geometries. It is different from other operators that use spatial indexes in that the spatial index is only used when the operator is in the ORDER BY clause.

Note
Index only kicks in if one of the geometries is a constant e.g. ORDER BY (ST_GeomFromText('POINT(1 2)') <#> geom) instead of g1.geom <#>.

Availability: 2.0.0 -- KNN only available for PostgreSQL 9.1+

Examples

```
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)
```

See Also

ST_DWithin, ST_Distance, <->

7.10.2.4 <<->>

<<->> — Returns the n-D distance between the A and B geometries or bounding boxes

Synopsis

double precision **<<->>**(geometry A , geometry B);

Description

The <<->> operator returns the n-D (euclidean) distance between the centroids of the bounding boxes of two geometries. Useful for doing nearest neighbor **approximate** distance ordering.

Note
This operand will make use of n-D GiST indexes that may be available on the geometries. It is different from other operators that use spatial indexes in that the spatial index is only used when the operator is in the ORDER BY clause.

Note
Index only kicks in if one of the geometries is a constant (not in a subquery/cte). e.g. 'SRID=3005;POINT(1011102 450541)'::geometry instead of a.geom

Availability: 2.2.0 -- KNN only available for PostgreSQL 9.1+

See Also

<->

7.11 Spatial Relationships

7.11.1 Topological Relationships

7.11.1.1 ST_3DIntersects

ST_3DIntersects — Tests if two geometries spatially intersect in 3D - only for points, linestrings, polygons, polyhedral surface (area)

Synopsis

boolean **ST_3DIntersects**(geometry geomA , geometry geomB);

Description

Overlaps, Touches, Within all imply spatial intersection. If any of the aforementioned returns true, then the geometries also spatially intersect. Disjoint implies false for spatial intersection.

Note
This function automatically includes a bounding box comparison that makes use of any spatial indexes that are available on the geometries.

Note
Because of floating robustness failures, geometries don't always intersect as you'd expect them to after geometric processing. For example the closest point on a linestring to a geometry may not lie on the linestring. For these kind of issues where a distance of a centimeter you want to just consider as intersecting, use ST_3DDWithin.

Changed: 3.0.0 SFCGAL backend removed, GEOS backend supports TINs.

Availability: 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

Geometry Examples

```
SELECT ST_3DIntersects(pt, line), ST_Intersects(pt, line)
  FROM (SELECT 'POINT(0 0 2)'::geometry As pt, 'LINESTRING (0 0 1, 0 2 3)'::geometry As ↩
      line) As foo;
 st_3dintersects | st_intersects
-----------------+---------------
 f               | t
(1 row)
```

TIN Examples

```
SELECT ST_3DIntersects('TIN(((0 0 0,1 0 0,0 1 0,0 0 0)))'::geometry, 'POINT(.1 .1 0)':: ↩
    geometry);
 st_3dintersects
-----------------
 t
```

See Also

ST_3DDWithin, ST_Intersects

7.11.1.2 ST_Contains

ST_Contains — Tests if every point of B lies in A, and their interiors have a point in common

Synopsis

boolean **ST_Contains**(geometry geomA, geometry geomB);

Description

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. To avoid index use, use the function `_ST_Contains`.

Performed by the GEOS module

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.

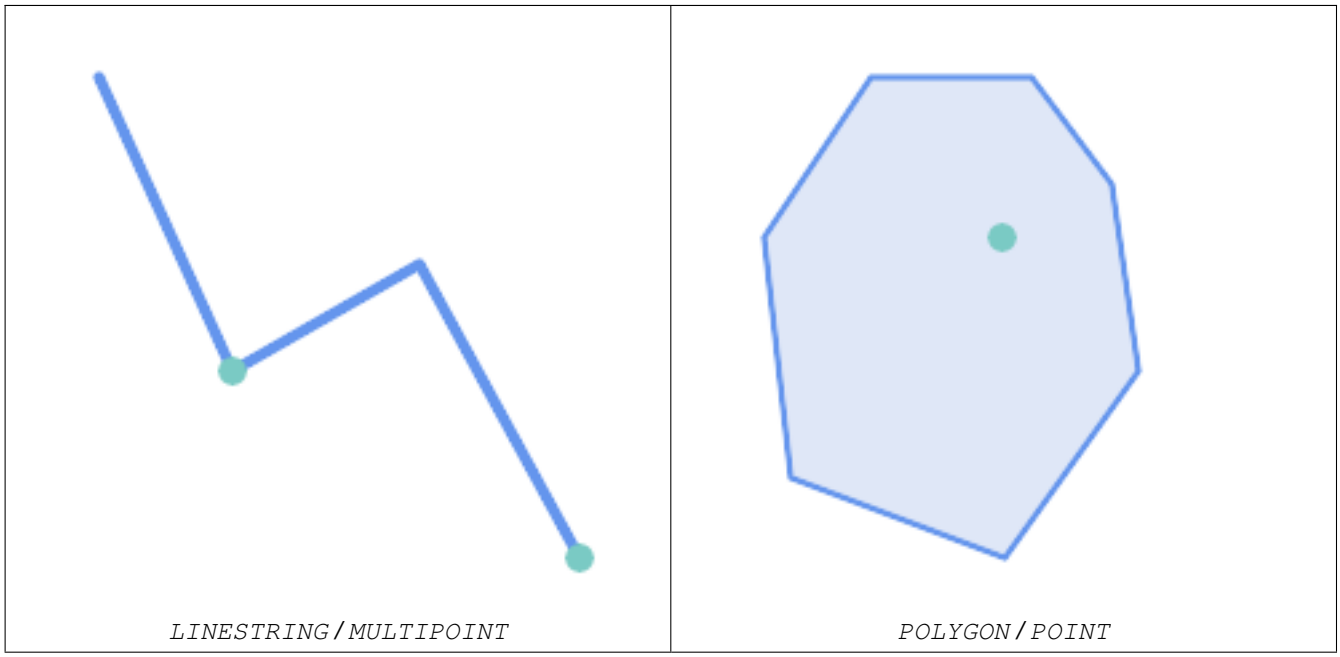
NOTE: this is the "allowable" version that returns a boolean, not an integer.

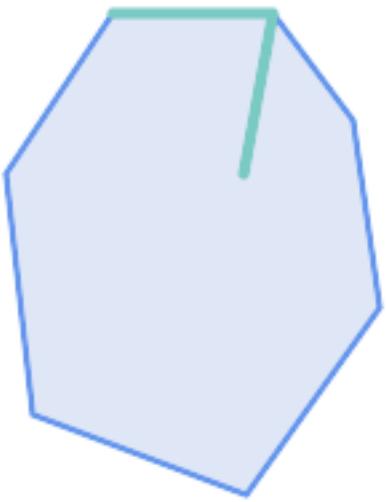
This method implements the OGC Simple Features Implementation Specification for SQL 1.1. s2.1.1.2 // s2.1.13.3 - same as within(geometry B, geometry A)

This method implements the SQL/MM specification. SQL-MM 3: 5.1.31

Examples

`ST_Contains` returns `TRUE` in the following situations:

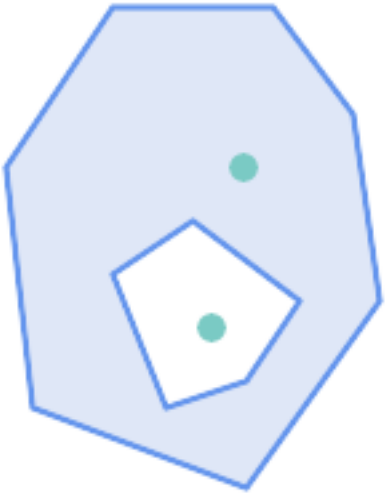




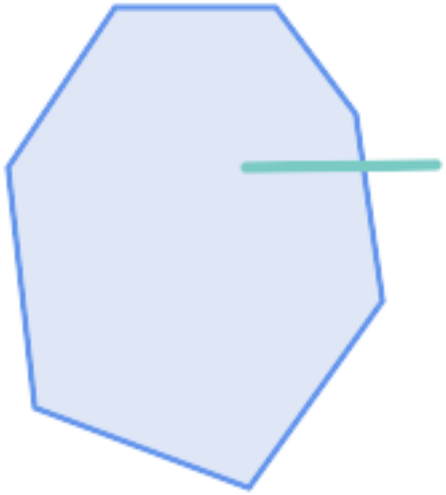
POLYGON / LINESTRING

POLYGON / POLYGON

`ST_Contains` returns `FALSE` in the following situations:

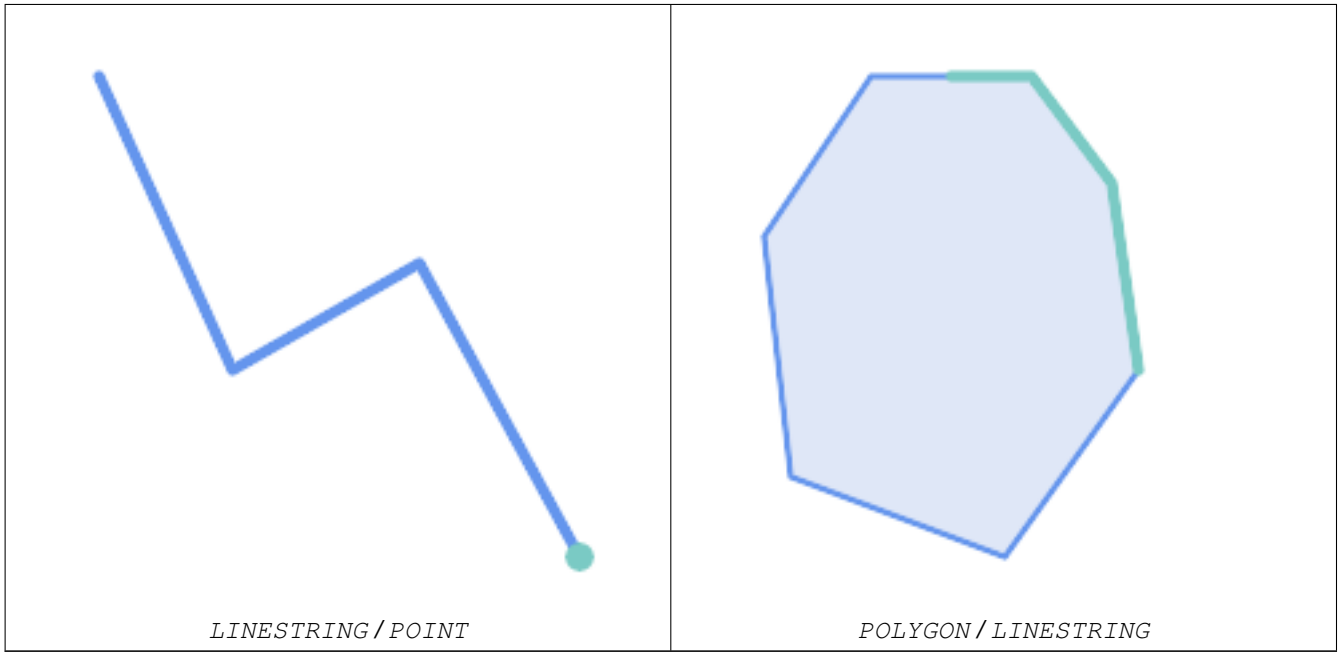


POLYGON / MULTIPOINT



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
```

See Also

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);

Description

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)* $\Leftrightarrow$ $Int(A) \bigcap B = B$

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
```

See Also

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);

Description

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.

Performed by the GEOS module

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)
```

See Also

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);

Description

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.

Performed by the GEOS module

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
```

See Also

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);

Description

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.

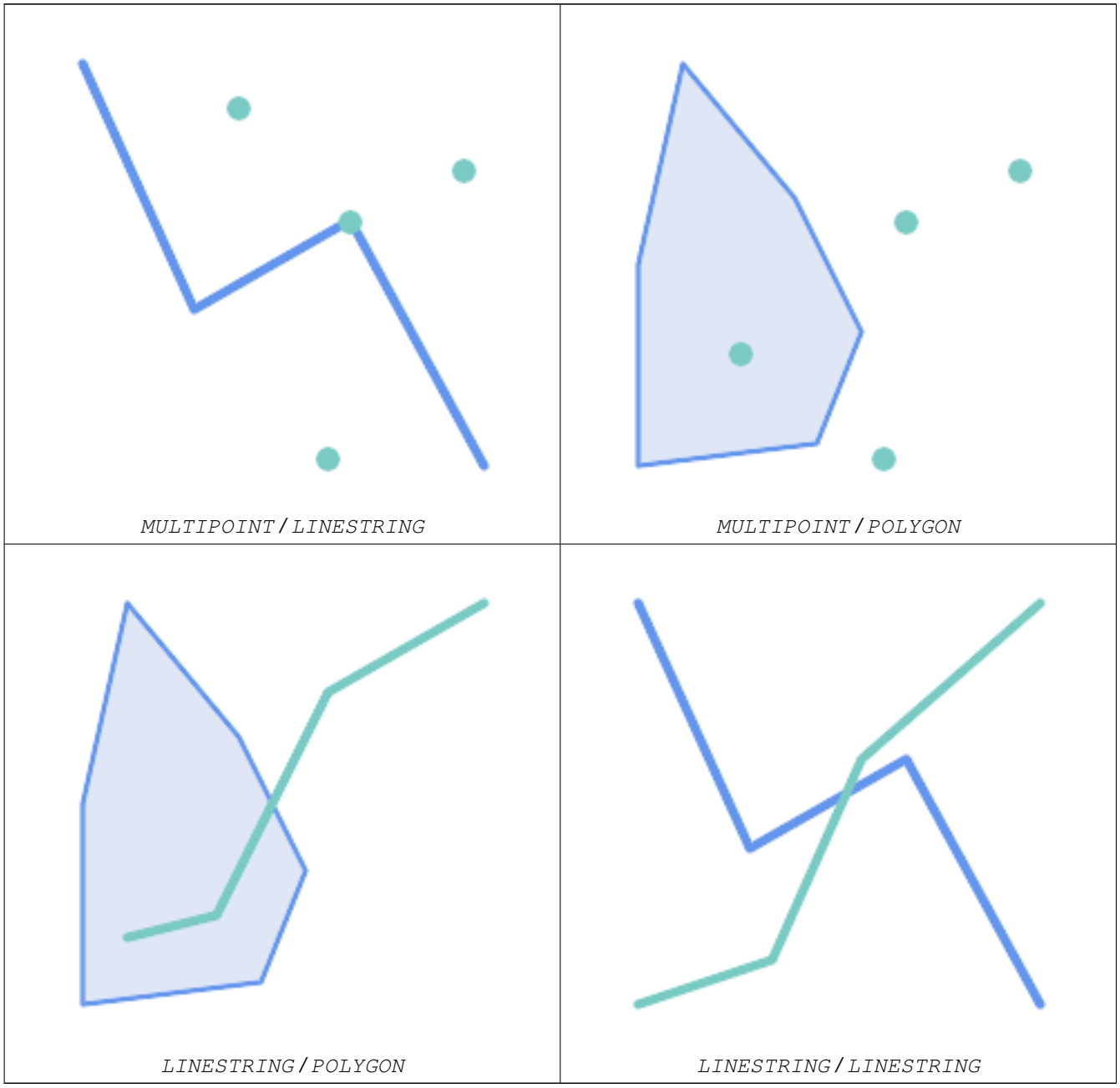
Important
Enhanced: 3.0.0 enabled support for `GEOMETRYCOLLECTION`

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`.



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 similar to:

```
SELECT roads.id
FROM roads, highways
WHERE ST_Crosses(roads.geom, highways.geom);
```

See Also

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);

Description

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`

Performed by the GEOS module

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)
```

See Also

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);

Description

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)
```

See Also

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);

Description

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).

Geometry Examples

```
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
```

See Also

&&, 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);

Description

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.

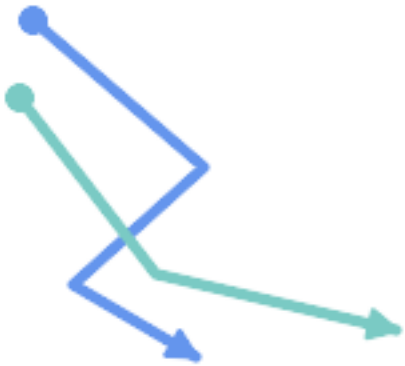
The crossing number has the following meaning:

- 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

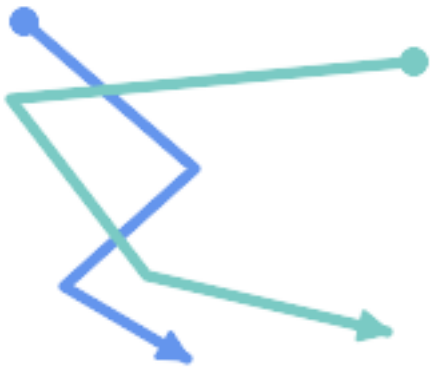
Example: LINE CROSS LEFT and LINE CROSS RIGHT



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

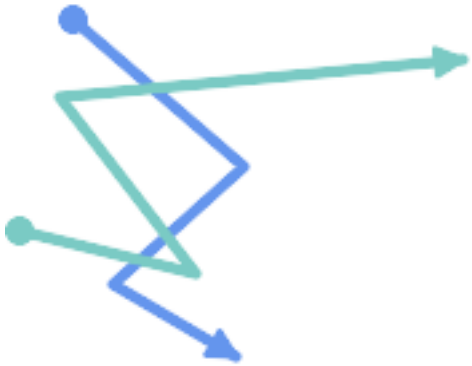


Blue: Line A; Green: Line B

```
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;
```

See Also

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);

Description

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)
```

See Also

&&, 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);

Description

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`.

Performed by the GEOS module

Important
Enhanced: 3.0.0 enabled support for `GEOMETRYCOLLECTION`

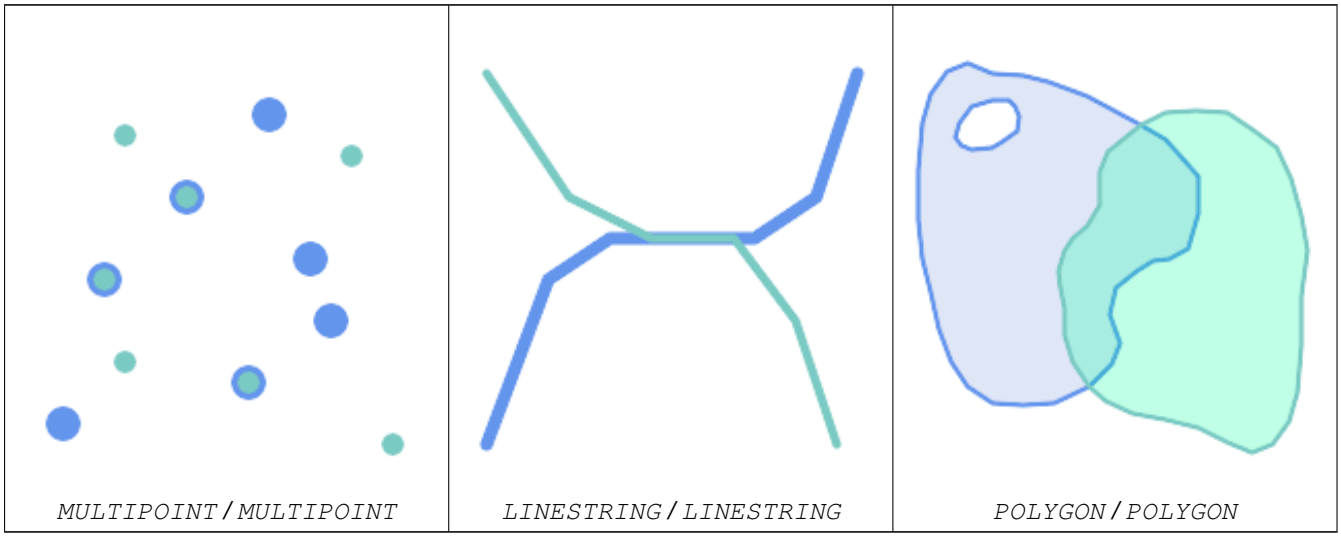
NOTE: this is the "allowable" version that returns a boolean, not an integer.

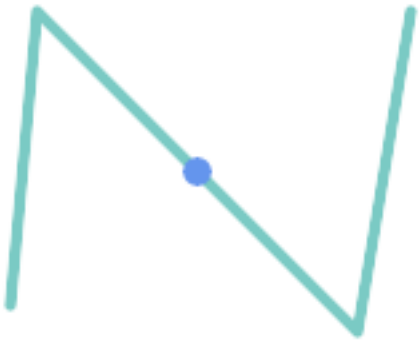
This method implements the OGC Simple Features Implementation Specification for SQL 1.1. s2.1.1.2 // s2.1.13.3

This method implements the SQL/MM specification. SQL-MM 3: 5.1.32

Examples

`ST_Overlaps` returns `TRUE` in the following situations:



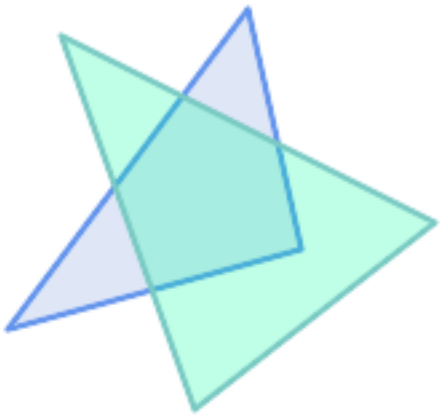


A Point on a LineString is contained, but since it has lower dimension it does not overlap or cross.

```
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
```

See Also

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);

Description

These functions allow testing and evaluating the spatial (topological) relationship between two geometries, as defined by the Dimensionally Extended 9-Intersection Model (DE-9IM).

The DE-9IM is specified as a 9-element matrix indicating the dimension of the intersections between the Interior, Boundary and Exterior of two geometries. It is represented by a 9-character text string using the symbols 'F', '0', '1', '2' (e.g. `'FF1FF0102'`).

A specific kind of spatial relationship can be tested by matching the intersection matrix to an *intersection matrix pattern*. Patterns can include the additional symbols 'T' (meaning "intersection is non-empty") and '*' (meaning "any value"). Common spatial

relationships are provided by the named functions ST_Contains, ST_ContainsProperly, ST_Covers, ST_CoveredBy, ST_Crosses, ST_Disjoint, ST_Equals, ST_Intersects, ST_Overlaps, ST_Touches, and ST_Within. Using an explicit pattern allows testing multiple conditions of intersects, crosses, etc in one step. It also allows testing spatial relationships which do not have a named spatial relationship function. For example, the relationship "Interior-Intersects" has the DE-9IM pattern `T********`, which is not evaluated by any named predicate.

For more information refer to Section 5.1.

Variant 1: Tests if two geometries are spatially related according to the given `intersectionMatrixPattern`.

Note
Unlike most of the named spatial relationship predicates, this does NOT automatically include an index call. The reason is that some relationships are true for geometries which do NOT intersect (e.g. Disjoint). If you are using a relationship pattern that requires intersection, then include the && index call.

Note
It is better to use a named relationship function if available, since they automatically use a spatial index where one exists. Also, they may implement performance optimizations which are not available with full relate evaluation.

Variant 2: Returns the DE-9IM matrix string for the spatial relationship between the two input geometries. The matrix string can be tested for matching a DE-9IM pattern using ST_RelateMatch.

Variant 3: Like variant 2, but allows specifying a **Boundary Node Rule**. A boundary node rule allows finer control over whether the endpoints of MultiLineStrings are considered to lie in the DE-9IM Interior or Boundary. The `boundaryNodeRule` values are:

- `1`: **OGC-Mod2** - line endpoints are in the Boundary if they occur an odd number of times. This is the rule defined by the OGC SFS standard, and is the default for `ST_Relate`.
- `2`: **Endpoint** - all endpoints are in the Boundary.
- `3`: **MultivalentEndpoint** - endpoints are in the Boundary if they occur more than once. In other words, the boundary is all the "attached" or "inner" endpoints (but not the "unattached/outer" ones).
- `4`: **MonovalentEndpoint** - endpoints are in the Boundary if they occur only once. In other words, the boundary is all the "unattached" or "outer" endpoints.

This function is not in the OGC spec, but is implied. see s2.1.13.2

This method implements the OGC Simple Features Implementation Specification for SQL 1.1. s2.1.1.2 // s2.1.13.3

This method implements the SQL/MM specification. SQL-MM 3: 5.1.25

Performed by the GEOS module

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
```

See Also

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);

Description

Tests if a Dimensionally Extended 9-Intersection Model (DE-9IM) `intersectionMatrix` value satisfies an `intersectionMatri`
Intersection matrix values can be computed by ST_Relate.

For more information refer to Section 5.1.

Performed by the GEOS module

Availability: 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
```

See Also

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);

Description

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: $ST\_Touches(A, B) \Leftrightarrow (Int(A) \bigcap Int(B) = \varnothing) \wedge (A \bigcap B \neq \varnothing)$

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.

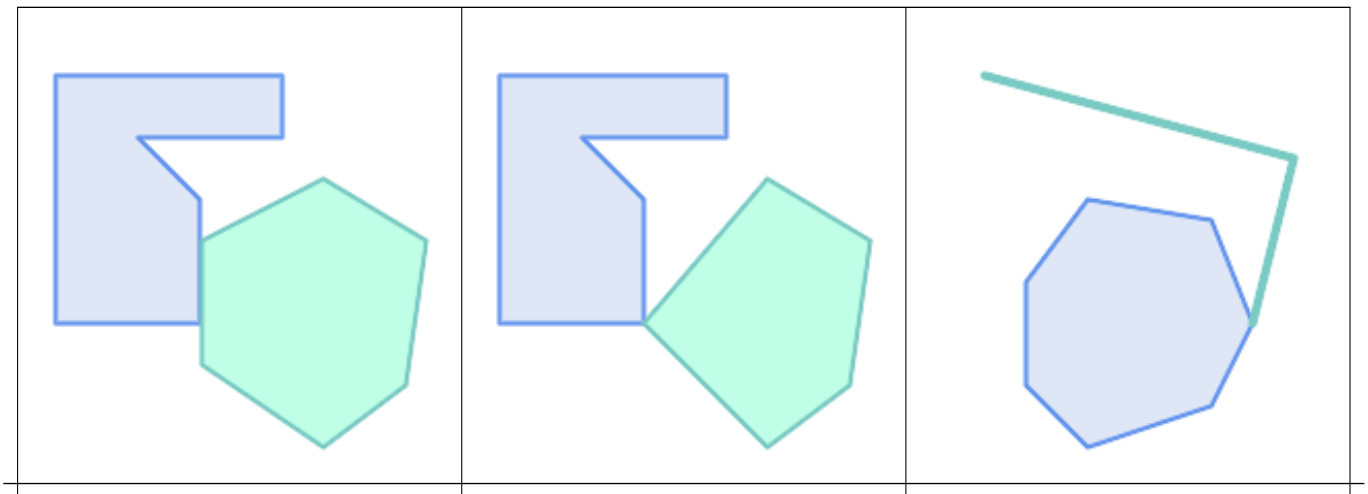
Important
Enhanced: 3.0.0 enabled support for `GEOMETRYCOLLECTION`

This method implements the OGC Simple Features Implementation Specification for SQL 1.1. s2.1.1.2 // s2.1.13.3

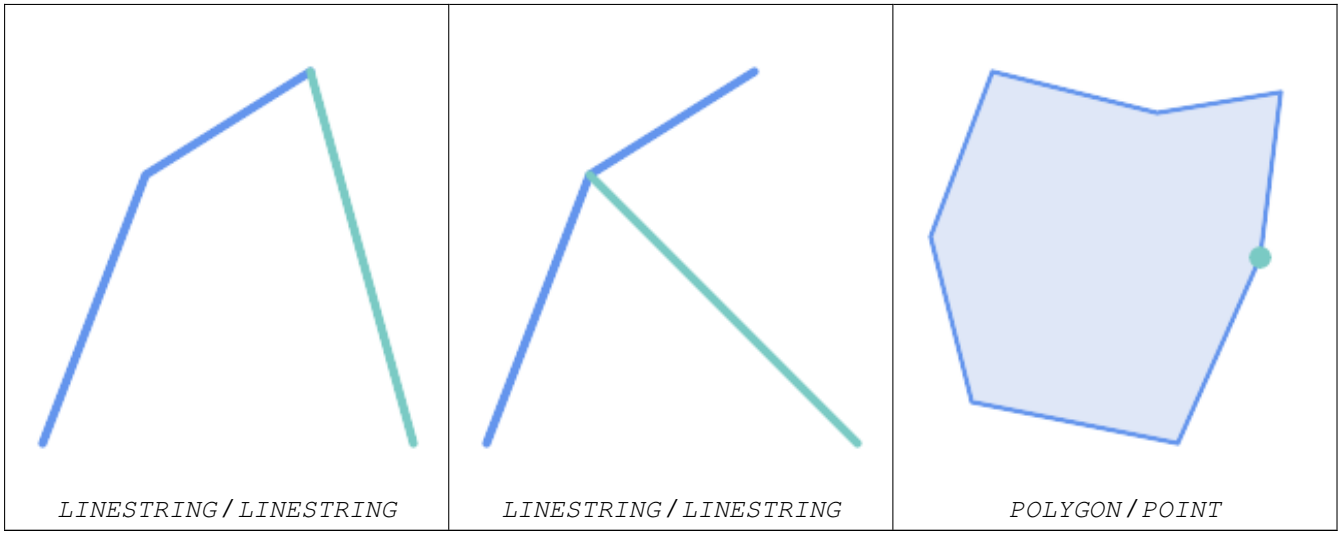
This method implements the SQL/MM specification. SQL-MM 3: 5.1.28

Examples

The `ST_Touches` predicate returns `TRUE` in the following examples.



POLYGON / POLYGON | *POLYGON / POLYGON* | *POLYGON / LINESTRING*




```
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);

Description

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`.

Performed by the GEOS module

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)
```

See Also

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);

Description

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 ?

Availability: 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
```

See Also

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);

Description

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.

Availability: 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
```

See Also

ST_3DDWithin, ST_DWithin, ST_DFullyWithin, ST_3DMaxDistance

7.11.2.3 ST_DFullyWithin

ST_DFullyWithin — Tests if a geometry is entirely inside a distance of another

Synopsis

boolean **ST_DFullyWithin**(geometry g1, geometry g2, double precision distance);

Description

Returns true if `g2` is entirely within `distance` of `g1`. Visually, the condition is true if `g2` is contained within a `distance` buffer of `g1`. The distance is specified in units defined by the spatial reference system of the geometries.

Note
This function automatically includes a bounding box comparison that makes use of any spatial indexes that are available on the geometries.

Availability: 1.5.0

Changed: 3.5.0 : the logic behind the function now uses a test of containment within a buffer, rather than the ST_MaxDistance algorithm. Results will differ from prior versions, but should be closer to user expectations.

Examples

```
SELECT
    ST_DFullyWithin(geom_a, geom_b, 10) AS DFullyWithin10,
    ST_DWithin(geom_a, geom_b, 10) AS DWithin10,
    ST_DFullyWithin(geom_a, geom_b, 20) AS DFullyWithin20
FROM (VALUES
    ('POINT(1 1)', 'LINESTRING(1 5, 2 7, 1 9, 14 12)')
    ) AS v(geom_a, geom_b)

 dfullywithin10 | dwithin10 | dfullywithin20
----------------+-----------+----------------
 f              | t         | t
```

See Also

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);

Description

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);
```

See Also

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);

Description

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
```

See Also

ST_DWithin

7.12 Measurement Functions

7.12.1 ST_Area

ST_Area — Returns the area of a polygonal geometry.

Synopsis

float **ST_Area**(geometry g1);
float **ST_Area**(geography geog, boolean use_spheroid = true);

Description

Returns the area of a polygonal geometry. For geometry types a 2D Cartesian (planar) area is computed, with units specified by the SRID. For geography types by default area is determined on a spheroid with units in square meters. To compute the area using the faster but less accurate spherical model use `ST_Area(geog,false)`.

Enhanced: 2.0.0 - support for 2D polyhedral surfaces was introduced.

Enhanced: 2.2.0 - measurement on spheroid performed with GeographicLib for improved accuracy and robustness. Requires PROJ >= 4.9.0 to take advantage of the new feature.

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
For polyhedral surfaces, only supports 2D polyhedral surfaces (not 2.5D). For 2.5D, may give a non-zero answer, but only for the faces that sit completely in XY plane.

Examples

Return area in square feet for a plot of Massachusetts land and multiply by conversion to get square meters. Note this is in square feet because EPSG:2249 is Massachusetts State Plane Feet

```
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
```

Return area square feet and transform to Massachusetts state plane meters (EPSG:26986) to get square meters. Note this is in square feet because 2249 is Massachusetts State Plane Feet and transformed area is in square meters since EPSG:26986 is state plane Massachusetts meters

```
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;&#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
```

Return area square feet and square meters using geography data type. Note that we transform to our geometry to geography (before you can do that make sure your geometry is in WGS 84 long lat 4326). Geography always measures in meters. This is just for demonstration to compare. Normally your table will be stored in geography data type already.

```
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;&#x2
&#x2502;  sqft_spheroid   &#x2502;   sqft_sphere    &#x2502;   sqm_spheroid   &#x2502;
&#x251c;&#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;&#x2
```

If your data is in geography already:

```
select ST_Area(geog) / 0.3048 ^ 2 sqft,
    ST_Area(the_geog) sqm
from somegeogtable;
```

See Also

ST_3DArea, ST_GeomFromText, ST_GeographyFromText, ST_SetSRID, ST_Transform

7.12.2 ST_Azimuth

ST_Azimuth — Returns the north-based azimuth of a line between two points.

Synopsis

float **ST_Azimuth**(geometry origin, geometry target);
float **ST_Azimuth**(geography origin, geography target);

Description

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()`.

Azimuth can be used in conjunction with ST_Translate to shift an object along its perpendicular axis. See the `upgis_lineshift()` function in the PostGIS wiki for an implementation of this.

Availability: 1.1.0

Enhanced: 2.0.0 support for geography was introduced.

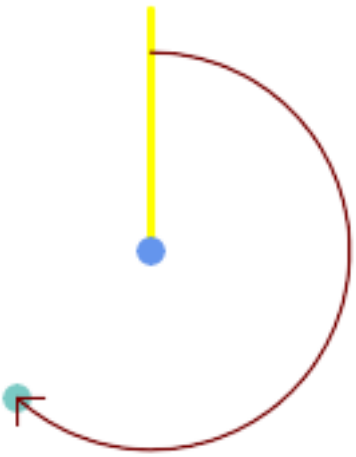
Enhanced: 2.2.0 measurement on spheroid performed with GeographicLib for improved accuracy and robustness. Requires PROJ >= 4.9.0 to take advantage of the new feature.

Examples

Geometry Azimuth in degrees

```
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.

See Also

ST_Angle, ST_Point, ST_Translate, ST_Project, PostgreSQL Math Functions

7.12.3 ST_Angle

ST_Angle — Returns the angle between two vectors defined by 3 or 4 points, or 2 lines.

Synopsis

float **ST_Angle**(geometry point1, geometry point2, geometry point3, geometry point4);
float **ST_Angle**(geometry line1, geometry line2);

Description

Computes the clockwise angle between two vectors.

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

The result is a positive angle between 0 and 2π radians. The radian result can be converted to degrees using the PostgreSQL function `degrees()`.

Note that `ST_Angle(P1,P2,P3) = ST_Angle(P2,P1,P2,P3)`.

Availability: 2.5.0

Examples

Angle between three points

```
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
```

See Also

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);

Description

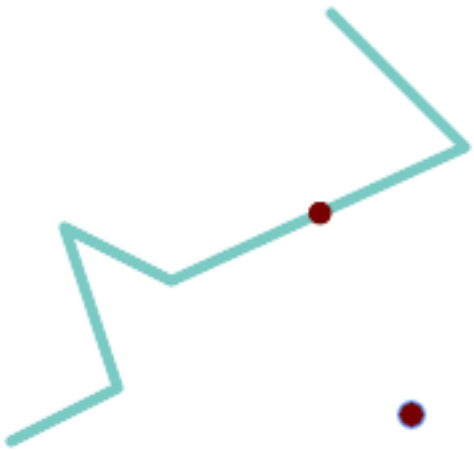
Returns the 2-dimensional point on `geom1` that is closest to `geom2`. This is the first point of the shortest line between the geometries (as computed by ST_ShortestLine).

Note
If you have a 3D Geometry, you may prefer to use ST_3DClosestPoint.

Enhanced: 3.4.0 - Support for geography.

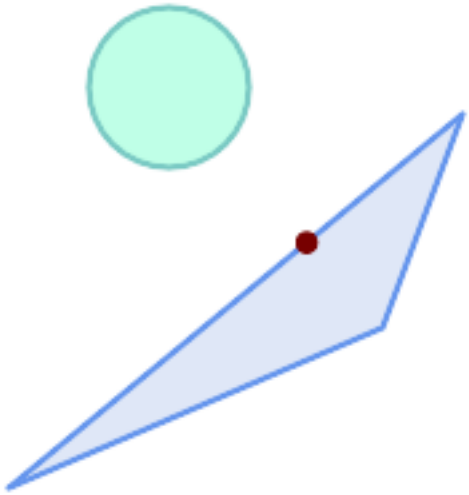
Availability: 1.5.0

Examples



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)
```

See Also

ST_3DClosestPoint, ST_Distance, ST_LongestLine, ST_ShortestLine, ST_MaxDistance

7.12.5 ST_3DClosestPoint

ST_3DClosestPoint — Returns the 3D point on g1 that is closest to g2. This is the first point of the 3D shortest line.

Synopsis

geometry **ST_3DClosestPoint**(geometry g1, geometry g2);

Description

Returns the 3-dimensional point on g1 that is closest to g2. This is the first point of the 3D shortest line. The 3D length of the 3D shortest line is the 3D distance.

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

This function supports Polyhedral surfaces.

Availability: 2.0.0

Changed: 2.2.0 - if 2 2D geometries are input, a 2D point is returned (instead of old behavior assuming 0 for missing Z). In case of 2D and 3D, Z is no longer assumed to be 0 for missing Z.

Examples

linestring and point -- both 3d and 2d closest point

```
SELECT ST_AsEWKT(ST_3DClosestPoint(line,pt)) AS cp3d_line_pt,
                ST_AsEWKT(ST_ClosestPoint(line,pt)) As cp2d_line_pt
        FROM (SELECT 'POINT(100 100 30)'::geometry As pt,
                        'LINESTRING (20 80 20, 98 190 1, 110 180 3, 50 75 1000)':: ↩
    geometry As line
                ) As foo;


 cp3d_line_pt                                                    |                ↩
    cp2d_line_pt
-----------------------------------------------------------------+-----------------------------------
 POINT(54.6993798867619 128.935022917228 11.5475869506606) | POINT(73.0769230769231  ↩
    115.384615384615)
```

linestring and multipoint -- both 3d and 2d closest point

```
SELECT ST_AsEWKT(ST_3DClosestPoint(line,pt)) AS cp3d_line_pt,
                ST_AsEWKT(ST_ClosestPoint(line,pt)) As cp2d_line_pt
        FROM (SELECT 'MULTIPOINT(100 100 30, 50 74 1000)'::geometry As pt,
                        'LINESTRING (20 80 20, 98 190 1, 110 180 3, 50 75 900)':: ↩
    geometry As line
                ) As foo;


                       cp3d_line_pt                          | cp2d_line_pt
-------------------------------------------------------------+--------------
 POINT(54.6993798867619 128.935022917228 11.5475869506606) | POINT(50 75)
```

Multilinestring and polygon both 3d and 2d closest point

```
SELECT ST_AsEWKT(ST_3DClosestPoint(poly, mline)) As cp3d,
    ST_AsEWKT(ST_ClosestPoint(poly, mline)) As cp2d
        FROM (SELECT  ST_GeomFromEWKT('POLYGON((175 150 5, 20 40 5, 35 45 5, 50 60 5,  ↩
     100 100 5, 175 150 5))') As poly,
                ST_GeomFromEWKT('MULTILINESTRING((175 155 2, 20 40 20, 50 60 -2, 125  ↩
     100 1, 175 155 1),
                (1 10 2, 5 20 1))') As mline ) As foo;
                   cp3d                    |     cp2d
-------------------------------------------+--------------
 POINT(39.993580415989 54.1889925532825 5) | POINT(20 40)
```

See Also

ST_AsEWKT, ST_ClosestPoint, ST_3DDistance, ST_3DShortestLine

7.12.6 ST_Distance

ST_Distance — Returns the distance between two geometry or geography values.

Synopsis

float **ST_Distance**(geometry g1, geometry g2);
float **ST_Distance**(geography geog1, geography geog2, boolean use_spheroid = true);

Description

For geometry types returns the minimum 2D Cartesian (planar) distance between two geometries, in projected units (spatial ref units).

For geography types defaults to return the minimum geodesic distance between two geographies in meters, compute on the spheroid determined by the SRID. If `use_spheroid` is false, a faster spherical calculation is used.

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

This method implements the SQL/MM specification. SQL-MM 3: 5.1.23

This method supports Circular Strings and Curves.

Availability: 1.5.0 geography support was introduced in 1.5. Speed improvements for planar to better handle large or many vertex geometries

Enhanced: 2.1.0 improved speed for geography. See Making Geography faster for details.

Enhanced: 2.1.0 - support for curved geometries was introduced.

Enhanced: 2.2.0 - measurement on spheroid performed with GeographicLib for improved accuracy and robustness. Requires PROJ >= 4.9.0 to take advantage of the new feature.

Changed: 3.0.0 - does not depend on SFCGAL anymore.

Geometry Examples

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
```

Geography Examples

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
```

See Also

ST_3DDistance, ST_DWithin, ST_DistanceSphere, ST_DistanceSpheroid, ST_MaxDistance, ST_HausdorffDistance, ST_FrechetDistan
ST_Transform

7.12.7 ST_3DDistance

ST_3DDistance — Returns the 3D cartesian minimum distance (based on spatial ref) between two geometries in projected units.

Synopsis

float **ST_3DDistance**(geometry g1, geometry g2);

Description

Returns the 3-dimensional minimum cartesian distance between two geometries in projected units (spatial ref units).

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

Availability: 2.0.0

Changed: 2.2.0 - In case of 2D and 3D, Z is no longer assumed to be 0 for missing Z.

Changed: 3.0.0 - SFCGAL version removed

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_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
```

See Also

ST_Distance, ST_3DClosestPoint, ST_3DDWithin, ST_3DMaxDistance, ST_3DShortestLine, ST_Transform

7.12.8 ST_DistanceSphere

ST_DistanceSphere — Returns minimum distance in meters between two lon/lat geometries using a spherical earth model.

Synopsis

float **ST_DistanceSphere**(geometry geomlonlatA, geometry geomlonlatB, float8 radius=6371008);

Description

Returns minimum distance in meters between two lon/lat points. Uses a spherical earth and radius derived from the spheroid defined by the SRID. Faster than ST_DistanceSpheroid, but less accurate. PostGIS Versions prior to 1.5 only implemented for points.

Availability: 1.5 - support for other geometry types besides points was introduced. Prior versions only work with points.

Changed: 2.2.0 In prior versions this used to be called ST_Distance_Sphere

Examples

```
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
```

See Also

ST_Distance, ST_DistanceSpheroid

7.12.9 ST_DistanceSpheroid

ST_DistanceSpheroid — Returns the minimum distance between two lon/lat geometries using a spheroidal earth model.

Synopsis

float **ST_DistanceSpheroid**(geometry geomlonlatA, geometry geomlonlatB, spheroid measurement_spheroid=WGS84);

Description

Returns minimum distance in meters between two lon/lat geometries given a particular spheroid. See the explanation of spheroids given for ST_LengthSpheroid.

Note
This function does not look at the SRID of the geometry. It assumes the geometry coordinates are based on the provided spheroid.

Availability: 1.5 - support for other geometry types besides points was introduced. Prior versions only work with points.

Changed: 2.2.0 In prior versions this was called ST_Distance_Spheroid

Examples

```
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
```

See Also

ST_Distance, ST_DistanceSphere

7.12.10 ST_FrechetDistance

ST_FrechetDistance — Returns the Fréchet distance between two geometries.

Synopsis

float **ST_FrechetDistance**(geometry g1, geometry g2, float densifyFrac = -1);

Description

Implements algorithm for computing the Fréchet distance restricted to discrete points for both geometries, based on Computing Discrete Fréchet Distance. The Fréchet distance is a measure of similarity between curves that takes into account the location and ordering of the points along the curves. Therefore it is often better than the Hausdorff distance.

When the optional densifyFrac is specified, this function performs a segment densification before computing the discrete Fréchet distance. The densifyFrac parameter sets the fraction by which to densify each segment. Each segment will be split into a number of equal-length subsegments, whose fraction of the total length is closest to the given fraction.

Units are in the units of the spatial reference system of the geometries.

Note
The current implementation supports only vertices as the discrete locations. This could be extended to allow an arbitrary density of points to be used.

Note
The smaller densifyFrac we specify, the more accurate Fréchet distance we get. But, the computation time and the memory usage increase with the square of the number of subsegments.

Performed by the GEOS module.

Availability: 2.4.0 - requires GEOS >= 3.7.0

Examples

```
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)
```

See Also

ST_HausdorffDistance

7.12.11 ST_HausdorffDistance

ST_HausdorffDistance — Returns the Hausdorff distance between two geometries.

Synopsis

float **ST_HausdorffDistance**(geometry g1, geometry g2);
float **ST_HausdorffDistance**(geometry g1, geometry g2, float densifyFrac);

Description

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.

Availability: 1.5.0

Examples

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);
```

See Also

ST_FrechetDistance

7.12.12 ST_Length

ST_Length — Returns the 2D length of a linear geometry.

Synopsis

float **ST_Length**(geometry a_2dlinestring);
float **ST_Length**(geography geog, boolean use_spheroid = true);

Description

For geometry types: returns the 2D Cartesian length of the geometry if it is a LineString, MultiLineString, ST_Curve, ST_MultiCurve. For areal geometries 0 is returned; use ST_Perimeter instead. The units of length is determined by the spatial reference system of the geometry.

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.

Currently for geometry this is an alias for ST_Length2D, but this may change to support higher dimensions.

Warning
Changed: 2.0.0 Breaking change -- in prior versions applying this to a MULTI/POLYGON of type geography would give you the perimeter of the POLYGON/MULTIPOLYGON. In 2.0.0 this was changed to return 0 to be in line with geometry behavior. Please use ST_Perimeter if you want the perimeter of a polygon

Note
For geography the calculation defaults to using a spheroidal model. To use the faster but less accurate spherical calculation use ST_Length(gg,false);

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

Availability: 1.5.0 geography support was introduced in 1.5.

Geometry Examples

Return length in feet for line string. Note this is in feet because EPSG:2249 is Massachusetts State Plane Feet

```
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
```

Geography Examples

Return length of WGS 84 geography line

```
-- 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
```

See Also

ST_GeographyFromText, ST_GeomFromEWKT, ST_LengthSpheroid, ST_Perimeter, ST_Transform

7.12.13 ST_Length2D

ST_Length2D — Returns the 2D length of a linear geometry. Alias for `ST_Length`

Synopsis

float **ST_Length2D**(geometry a_2dlinestring);

Description

Returns the 2D length of the geometry if it is a linestring or multi-linestring. This is an alias for `ST_Length`

See Also

ST_Length, ST_3DLength

7.12.14 ST_3DLength

ST_3DLength — Returns the 3D length of a linear geometry.

Synopsis

float **ST_3DLength**(geometry a_3dlinestring);

Description

Returns the 3-dimensional or 2-dimensional length of the geometry if it is a LineString or MultiLineString. For 2-d lines it will just return the 2-d length (same as ST_Length and 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

Changed: 2.0.0 In prior versions this used to be called ST_Length3D

Examples

Return length in feet for a 3D cable. Note this is in feet because EPSG:2249 is Massachusetts State Plane Feet

```
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
```

See Also

ST_Length, ST_Length2D

7.12.15 ST_LengthSpheroid

ST_LengthSpheroid — Returns the 2D or 3D length/perimeter of a lon/lat geometry on a spheroid.

Synopsis

float **ST_LengthSpheroid**(geometry a_geometry, spheroid a_spheroid);

Description

Calculates the length or perimeter of a geometry on an ellipsoid. This is useful if the coordinates of the geometry are in longitude/latitude and a length is desired without reprojection. The spheroid is specified by a text value as follows:

```
SPHEROID[<NAME>,<SEMI-MAJOR AXIS>,<INVERSE FLATTENING>]
```

For example:

```
SPHEROID["GRS_1980",6378137,298.257222101]
```

Availability: 1.2.2

Changed: 2.2.0 In prior versions this was called ST_Length_Spheroid and had the alias ST_3DLength_Spheroid

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

Examples

```
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
```

See Also

ST_GeometryN, ST_Length

7.12.16 ST_LongestLine

ST_LongestLine — Returns the 2D longest line between two geometries.

Synopsis

geometry **ST_LongestLine**(geometry g1, geometry g2);

Description

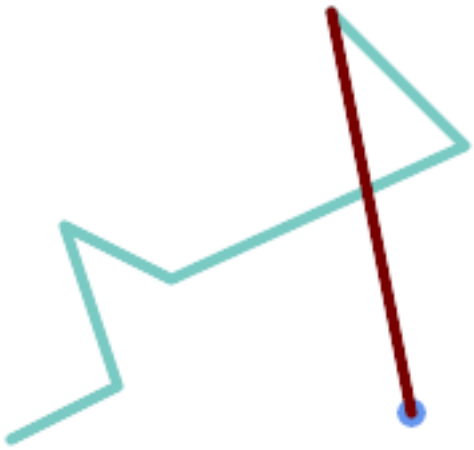
Returns the 2-dimensional longest line between the points of two geometries. The line returned starts on `g1` and ends on `g2`.

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.

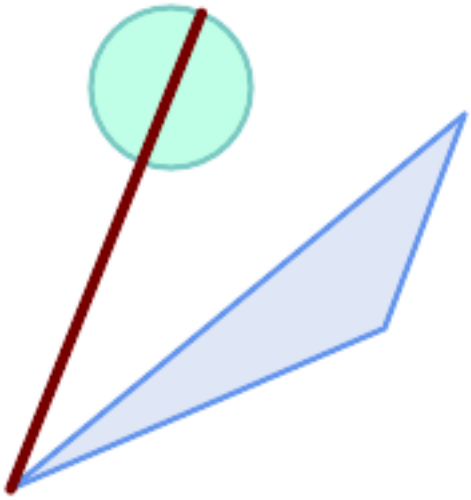
Availability: 1.5.0

Examples



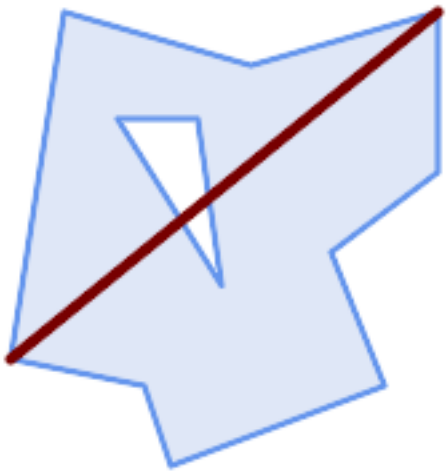
Longest line between a point and a line

```
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)
```



Longest line between two polygons

```
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
```

See Also

ST_MaxDistance, ST_ShortestLine, ST_3DLongestLine, ST_MinimumBoundingCircle

7.12.17 ST_3DLongestLine

ST_3DLongestLine — Returns the 3D longest line between two geometries

Synopsis

geometry **ST_3DLongestLine**(geometry g1, geometry g2);

Description

Returns the 3-dimensional longest line between two geometries. The function returns the first longest line if more than one. The line returned starts in g1 and ends in g2. The 3D length of the line is equal to the distance returned by ST_3DMaxDistance.

Availability: 2.0.0

Changed: 2.2.0 - if 2 2D geometries are input, a 2D point is returned (instead of old behavior assuming 0 for missing Z). In case of 2D and 3D, Z is no longer assumed to be 0 for missing Z.

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

This function supports Polyhedral surfaces.

Examples

linestring and point -- both 3d and 2d longest line

```
SELECT ST_AsEWKT(ST_3DLongestLine(line,pt)) AS lol3d_line_pt,
                ST_AsEWKT(ST_LongestLine(line,pt)) As lol2d_line_pt
        FROM (SELECT 'POINT(100 100 30)'::geometry As pt,
                        'LINESTRING (20 80 20, 98 190 1, 110 180 3, 50 75 1000)':: ↩
    geometry As line
                ) As foo;


           lol3d_line_pt            |       lol2d_line_pt
------------------------------------+---------------------------
 LINESTRING(50 75 1000,100 100 30) | LINESTRING(98 190,100 100)
```

linestring and multipoint -- both 3d and 2d longest line

```
SELECT ST_AsEWKT(ST_3DLongestLine(line,pt)) AS lol3d_line_pt,
                ST_AsEWKT(ST_LongestLine(line,pt)) As lol2d_line_pt
        FROM (SELECT 'MULTIPOINT(100 100 30, 50 74 1000)'::geometry As pt,
                        'LINESTRING (20 80 20, 98 190 1, 110 180 3, 50 75 900)':: ↩
    geometry As line
                ) As foo;


          lol3d_line_pt           |      lol2d_line_pt
----------------------------------+-------------------------
 LINESTRING(98 190 1,50 74 1000) | LINESTRING(98 190,50 74)
```

MultiLineString and Polygon both 3d and 2d longest line

```
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)
```

See Also

ST_3DClosestPoint, ST_3DDistance, ST_LongestLine, ST_3DShortestLine, ST_3DMaxDistance

7.12.18 ST_MaxDistance

ST_MaxDistance — Returns the 2D largest distance between two geometries in projected units.

Synopsis

float **ST_MaxDistance**(geometry g1, geometry g2);

Description

Returns the 2-dimensional maximum distance between two geometries, in projected units. The maximum distance always occurs between two vertices. This is the length of the line returned by ST_LongestLine.

If g1 and g2 are the same geometry, returns the distance between the two vertices farthest apart in that geometry.

Availability: 1.5.0

Examples

Maximum distance between a point and lines.

```
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
```

See Also

ST_Distance, ST_LongestLine, ST_DFullyWithin

7.12.19 ST_3DMaxDistance

ST_3DMaxDistance — Returns the 3D cartesian maximum distance (based on spatial ref) between two geometries in projected units.

Synopsis

float **ST_3DMaxDistance**(geometry g1, geometry g2);

Description

Returns the 3-dimensional maximum cartesian distance between two geometries in projected units (spatial ref units).

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

This function supports Polyhedral surfaces.

Availability: 2.0.0

Changed: 2.2.0 - In case of 2D and 3D, Z is no longer assumed to be 0 for missing Z.

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_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
```

See Also

ST_Distance, ST_3DDWithin, ST_3DMaxDistance, ST_Transform

7.12.20 ST_MinimumClearance

ST_MinimumClearance — Returns the minimum clearance of a geometry, a measure of a geometry's robustness.

Synopsis

float **ST_MinimumClearance**(geometry g);

Description

It is possible for a geometry to meet the criteria for validity according to ST_IsValid (polygons) or ST_IsSimple (lines), but to become invalid if one of its vertices is moved by a small distance. This can happen due to loss of precision during conversion to text formats (such as WKT, KML, GML, GeoJSON), or binary formats that do not use double-precision floating point coordinates (e.g. 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.

If a geometry has a minimum clearance of `e`, then:

- No two distinct vertices in the geometry are closer than the distance `e`.
- No vertex is closer than `e` to a line segment of which it is not an endpoint.

If no minimum clearance exists for a geometry (e.g. a single point, or a MultiPoint whose points are identical), the return value is `Infinity`.

To avoid validity issues caused by precision loss, ST_ReducePrecision can reduce coordinate precision while ensuring that polygonal geometry remains valid.

Availability: 2.3.0

Examples

```
SELECT ST_MinimumClearance('POLYGON ((0 0, 1 0, 1 1, 0.5 3.2e-4, 0 0))');
 st_minimumclearance
---------------------
             0.00032
```

See Also

ST_MinimumClearanceLine, ST_IsSimple, ST_IsValid, ST_ReducePrecision

7.12.21 ST_MinimumClearanceLine

ST_MinimumClearanceLine — Returns the two-point LineString spanning a geometry's minimum clearance.

Synopsis

Geometry **ST_MinimumClearanceLine**(geometry g);

Description

Returns the two-point LineString spanning a geometry's minimum clearance. If the geometry does not have a minimum clearance, `LINESTRING EMPTY` is returned.

Performed by the GEOS module.

Availability: 2.3.0 - requires GEOS >= 3.6.0

Examples

```
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)
```

See Also

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);

Description

Returns the 2D perimeter of the geometry/geography if it is a ST_Surface, ST_MultiSurface (Polygon, MultiPolygon). 0 is returned for non-areal geometries. For linear geometries use ST_Length. For geometry types, units for perimeter measures are specified by the spatial reference system of the geometry.

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.

Currently this is an alias for ST_Perimeter2D, but this may change to support higher dimensions.

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

Availability 2.0.0: Support for geography was introduced

Examples: Geometry

Return perimeter in feet for Polygon and MultiPolygon. Note this is in feet because EPSG:2249 is Massachusetts State Plane Feet

```
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)
```

Examples: Geography

Return perimeter in meters and feet for Polygon and MultiPolygon. Note this is geography (WGS 84 long lat)

```
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
```

See Also

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);

Description

Returns the 2-dimensional perimeter of a polygonal geometry.

Note
This is currently an alias for ST_Perimeter. In future versions ST_Perimeter may return the highest dimension perimeter for a geometry. This is still under consideration

See Also

ST_Perimeter

7.12.24 ST_3DPerimeter

ST_3DPerimeter — Returns the 3D perimeter of a polygonal geometry.

Synopsis

float **ST_3DPerimeter**(geometry geomA);

Description

Returns the 3-dimensional perimeter of the geometry, if it is a polygon or multi-polygon. If the geometry is 2-dimensional, then the 2-dimensional perimeter is returned.

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

Changed: 2.0.0 In prior versions this used to be called ST_Perimeter3D

Examples

Perimeter of a slightly elevated polygon in the air 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
```

See Also

ST_GeomFromEWKT, ST_Perimeter, ST_Perimeter2D

7.12.25 ST_ShortestLine

ST_ShortestLine — Returns the 2D shortest line between two geometries

Synopsis

geometry **ST_ShortestLine**(geometry geom1, geometry geom2);
geography **ST_ShortestLine**(geography geom1, geography geom2, boolean use_spheroid = true);

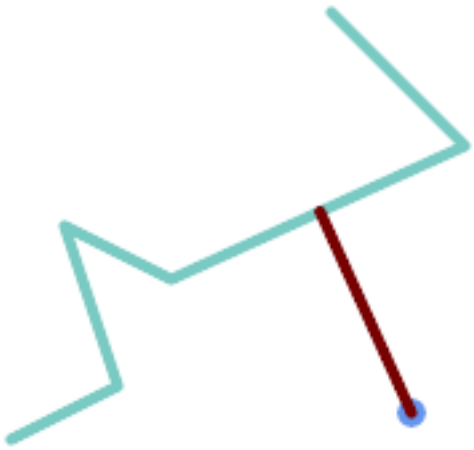
Description

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.

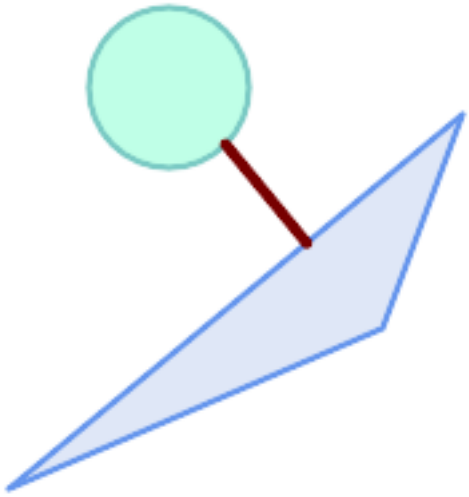
Availability: 1.5.0

Examples



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)
```

See Also

ST_ClosestPoint, ST_Distance, ST_LongestLine, ST_MaxDistance

7.12.26 ST_3DShortestLine

ST_3DShortestLine — Returns the 3D shortest line between two geometries

Synopsis

geometry **ST_3DShortestLine**(geometry g1, geometry g2);

Description

Returns the 3-dimensional shortest line between two geometries. The function will only return the first shortest line if more than one, that the function finds. If g1 and g2 intersects in just one point the function will return a line with both start and end in that intersection-point. If g1 and g2 are intersecting with more than one point the function will return a line with start and end in the same point but it can be any of the intersecting points. The line returned will always start in g1 and end in g2. The 3D length of the line this function returns will always be the same as ST_3DDistance returns for g1 and g2.

Availability: 2.0.0

Changed: 2.2.0 - if 2 2D geometries are input, a 2D point is returned (instead of old behavior assuming 0 for missing Z). In case of 2D and 3D, Z is no longer assumed to be 0 for missing Z.

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

This function supports Polyhedral surfaces.

Examples

linestring and point -- both 3d and 2d shortest line

```
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)
```

linestring and multipoint -- both 3d and 2d shortest line

```
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)
```

MultiLineString and polygon both 3d and 2d shortest line

```
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)
```

See Also

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);

Description

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;
```

See Also

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);

Description

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` parameter is given (GEOS-3.9.0 or higher required), all result vertices are guaranteed to fall on a grid of the specified size. For the operation to give predictable results all the input vertices must fall already on the specified grid, see ST_ReducePrecision.

Performed by the GEOS module

Enhanced: 3.1.0 accept a gridSize parameter.

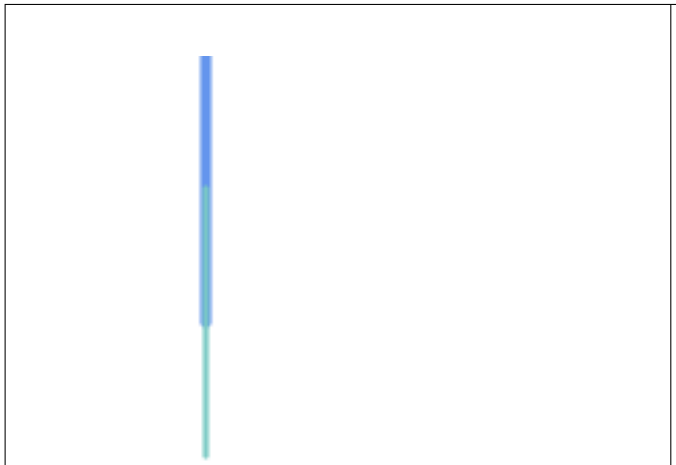
Requires GEOS >= 3.9.0 to use the gridSize parameter.

This method implements the OGC Simple Features Implementation Specification for SQL 1.1. s2.1.1.3

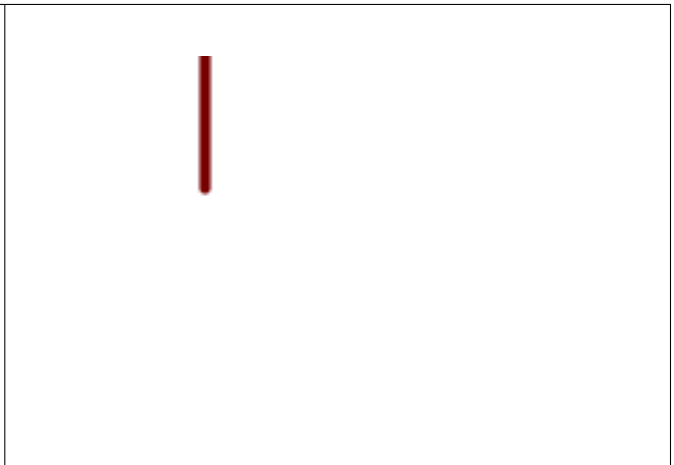
This method implements the SQL/MM specification. SQL-MM 3: 5.1.20

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)
```

See Also

ST_SymDifference, ST_Intersection, ST_Union, ST_ReducePrecision

7.13.3 ST_Intersection

ST_Intersection — Computes a geometry representing the shared portion of geometries A and B.

Synopsis

geometry **ST_Intersection**(geometry geomA , geometry geomB , float8 gridSize = -1);
geography **ST_Intersection**(geography geogA , geography geogB);

Description

Returns a geometry representing the point-set intersection of two geometries. In other words, that portion of geometry A and geometry B that is shared between the two geometries.

If the geometries have no points in common (i.e. are disjoint) then an empty atomic geometry of appropriate type is returned.

If the optional `gridSize` parameter is given (GEOS-3.9.0 or higher required), all result vertices are guaranteed to fall on a grid of the specified size. For the operation to give predictable results all the input vertices must fall already on the specified grid, see ST_ReducePrecision.

ST_Intersection in conjunction with ST_Intersects is useful for clipping geometries such as in bounding box, buffer, or region queries where you only require the portion of a geometry that is inside a country or region of interest.

Note

For geography this is a thin wrapper around the geometry implementation. It first determines the best SRID that fits the bounding box of the 2 geography objects (if geography objects are within one half zone UTM but not same UTM will pick one of those) (favoring UTM or Lambert Azimuthal Equal Area (LAEA) north/south pole, and falling back on mercator in worst case scenario) and then intersection in that best fit planar spatial ref and retransforms back to WGS84 geography.

Warning

This function will drop the M coordinate values if present.

Warning

If working with 3D geometries, you may want to use 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.

Performed by the GEOS module

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)
```

See Also

ST_3DIntersection, ST_Difference, ST_Union, ST_ClipByBox2D, ST_Dimension, ST_Dump, ST_Force2D, ST_SymDifference, ST_Intersects, ST_Multi, ST_ReducePrecision

7.13.4 ST_MemUnion

ST_MemUnion — Aggregate function which unions geometries in a memory-efficent but slower way

Synopsis

geometry **ST_MemUnion**(geometry set geomfield);

Description

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;
```

See Also

ST_Union

7.13.5 ST_Node

ST_Node — Nodes a collection of lines.

Synopsis

geometry **ST_Node**(geometry geom);

Description

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.

Availability: 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))
```

See Also

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);

Description

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.

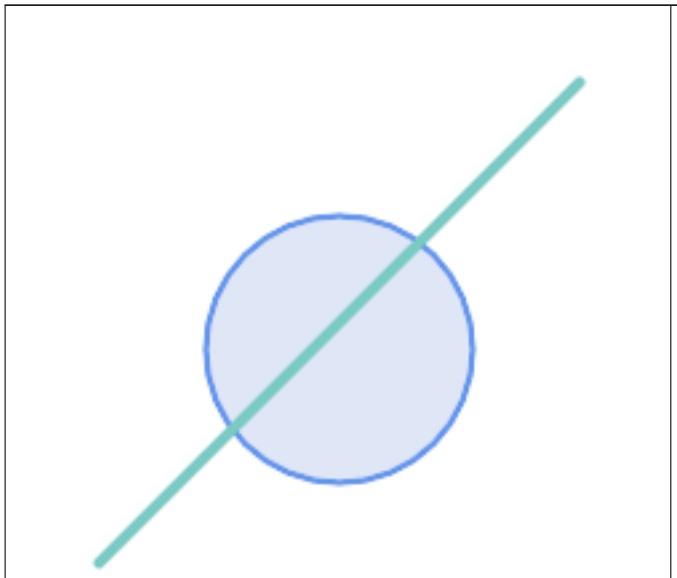
Availability: 2.0.0 requires GEOS

Enhanced: 2.2.0 support for splitting a line by a multiline, a multipoint or (multi)polygon boundary was introduced.

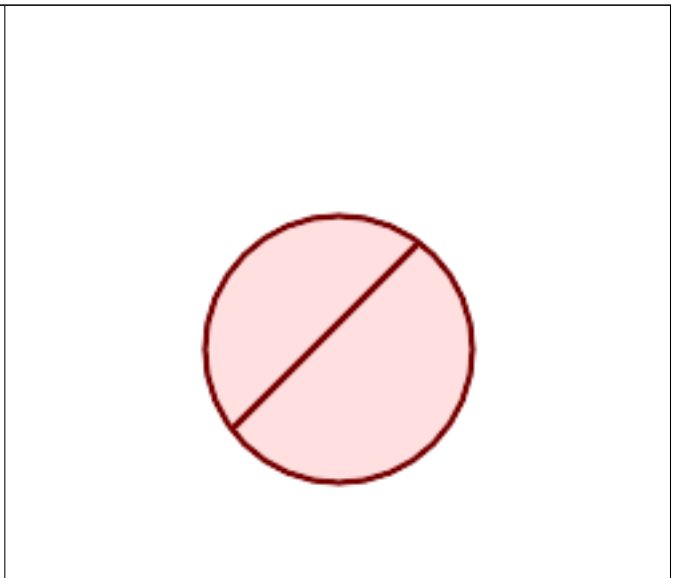
Enhanced: 2.5.0 support for splitting a polygon by a multiline was introduced.

Examples

Split a Polygon by a Line.



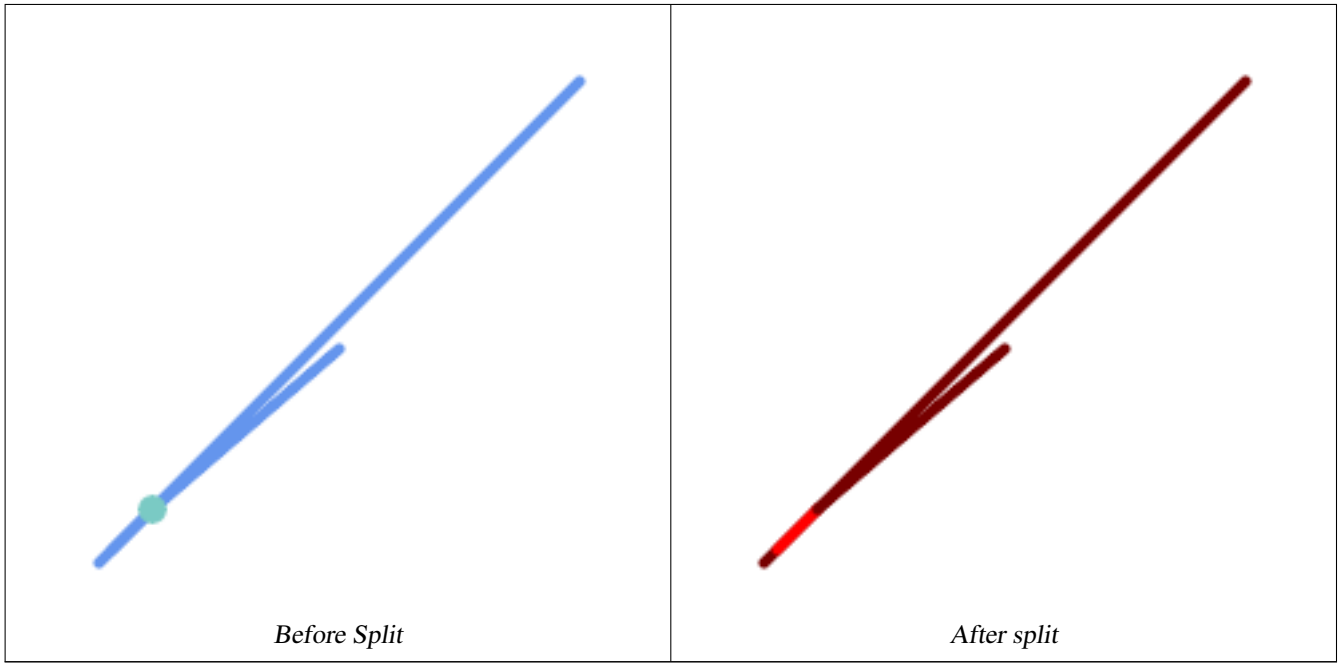
Before Split



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 by a Point, where the point lies exactly on both LineStrings elements.



Before Split | *After split*

```
SELECT ST_AsText(ST_Split(
    'MULTILINESTRING((10 10, 190 190), (15 15, 30 30, 100 90))',
    ST_Point(30,30))) As split;

split
------
GEOMETRYCOLLECTION(
    LINESTRING(10 10,30 30),
    LINESTRING(30 30,190 190),
    LINESTRING(15 15,30 30),
    LINESTRING(30 30,100 90)
)
```

Split a LineString by a Point, where the point does not lie exactly on the line. Shows using ST_Snap to snap the line to the point to allow it to be split.

```
WITH data AS (SELECT
  'LINESTRING(0 0, 100 100)'::geometry AS line,
  'POINT(51 50)':: geometry AS point
)
SELECT ST_AsText( ST_Split( ST_Snap(line, point, 1), point)) AS snapped_split,
       ST_AsText( ST_Split(line, point)) AS not_snapped_not_split
       FROM data;

                            snapped_split                                |             ↩
                                not_snapped_not_split
-------------------------------------------------------------------------+-------------------------------

 GEOMETRYCOLLECTION(LINESTRING(0 0,51 50),LINESTRING(51 50,100 100)) | GEOMETRYCOLLECTION( ↩
     LINESTRING(0 0,100 100))
```

See Also

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);

Description

Returns a set of geometries that are the result of dividing `geom` into parts using rectilinear lines, with each part containing no more than `max_vertices`.

`max_vertices` must be 5 or more, as 5 points are needed to represent a closed box.

If the optional `gridSize` parameter is given (GEOS-3.9.0 or higher required), all result vertices are guaranteed to fall on a grid of the specified size. For the operation to give predictable results all the input vertices must fall already on the specified grid, see ST_ReducePrecision.

Point-in-polygon and other spatial operations are normally faster for indexed subdivided datasets. Since the bounding boxes for the parts usually cover a smaller area than the original geometry bbox, index queries produce fewer "hit" cases. The "hit" cases are faster because the spatial operations executed by the index recheck process fewer points.

Note
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

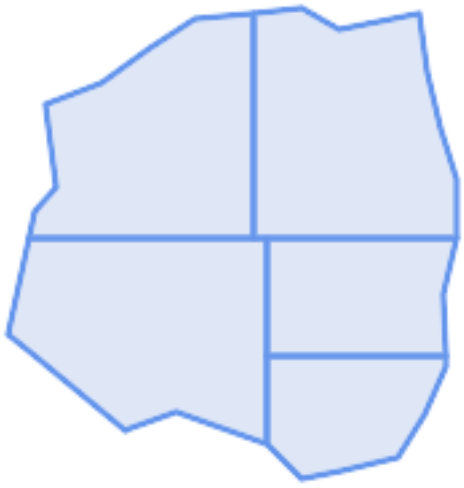
Enhanced: 2.5.0 reuses existing points on polygon split, vertex count is lowered from 8 to 5.

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;&#x2500;&#x2500;&#x2500;&#x2500;&#x2500;&#x2
  1 &#x2502; POLYGON((119 23,85 35,68 29,66 28,32 56,22 64,29.8260869565217 100,119 100,119 ↩
      23))
  2 &#x2502; POLYGON((132 10,119 23,119 56,186 56,186 52,178 34,168 18,147 13,132 10))
  3 &#x2502; POLYGON((119 56,119 100,190 100,185 79,186 56,119 56))
  4 &#x2502; POLYGON((29.8260869565217 100,32 110,40 119,36 150,57 158,75 171,92 182,114  ↩
     184,114 100,29.8260869565217 100))
  5 &#x2502; POLYGON((114 184,132 186,146 178,176 184,179 162,184 141,190 122,190 100,114  ↩
     100,114 184))
```

Example: Densify a long geography line using ST_Segmentize(geography, distance), and use ST_Subdivide to split the resulting line into sublines of 8 vertices.

The densified and split lines.

```
SELECT ST_AsText( ST_Subdivide(
            ST_Segmentize('LINESTRING(0 0, 85 85)'::geography,
                        1200000)::geometry,    8));
```

```
LINESTRING(0 0,0.487578359029357 5.57659056746196,0.984542144675897  ↩
    11.1527721155093,1.50101059639722 16.7281035483571,1.94532113630331 21.25)
LINESTRING(1.94532113630331 21.25,2.04869538062779 22.3020741387339,2.64204641967673  ↩
    27.8740533545155,3.29994062412787 33.443216802941,4.04836719489742  ↩
    39.0084282520239,4.59890468420694 42.5)
LINESTRING(4.59890468420694 42.5,4.92498503922732 44.5680389206321,5.98737409390639  ↩
    50.1195229244701,7.3290919767674 55.6587646879025,8.79638749938413 60.1969505994924)
LINESTRING(8.79638749938413 60.1969505994924,9.11375579533779  ↩
    61.1785363177625,11.6558166691368 66.6648504160202,15.642041247655  ↩
    72.0867690601745,22.8716627200212 77.3609628116894,24.6991785131552 77.8939011989848)
LINESTRING(24.6991785131552 77.8939011989848,39.4046096622744  ↩
    82.1822848017636,44.7994523421035 82.5156766227011)
LINESTRING(44.7994523421035 82.5156766227011,85 85)
```

Example: Subdivide the complex geometries of a table in-place. The original geometry records are deleted from the source table, and new records for each subdivided result geometry are inserted.

```
WITH complex_areas_to_subdivide AS (
    DELETE from polygons_table
    WHERE ST_NPoints(geom) > 255
    RETURNING id, column1, column2, column3, geom
)
INSERT INTO polygons_table (fid, column1, column2, column3, geom)
    SELECT fid, column1, column2, column3,
           ST_Subdivide(geom, 255) as geom
    FROM complex_areas_to_subdivide;
```

Example: Create a new table containing subdivided geometries, retaining the key of the original geometry so that the new table can be joined to the source table. Since ST_Subdivide is a set-returning (table) function that returns a set of single-value rows, this syntax automatically produces a table with one row for each result part.

```
CREATE TABLE subdivided_geoms AS
    SELECT pkey, ST_Subdivide(geom) AS geom
    FROM original_geoms;
```

See Also

ST_ClipByBox2D, ST_Segmentize, ST_Split, ST_NPoints, ST_ReducePrecision

7.13.8 ST_SymDifference

ST_SymDifference — Computes a geometry representing the portions of geometries A and B that do not intersect.

Synopsis

geometry **ST_SymDifference**(geometry geomA, geometry geomB, float8 gridSize = -1);

Description

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` parameter is given (GEOS-3.9.0 or higher required), all result vertices are guaranteed to fall on a grid of the specified size. For the operation to give predictable results all the input vertices must fall already on the specified grid, see ST_ReducePrecision.

Performed by the GEOS module

Enhanced: 3.1.0 accept a gridSize parameter.

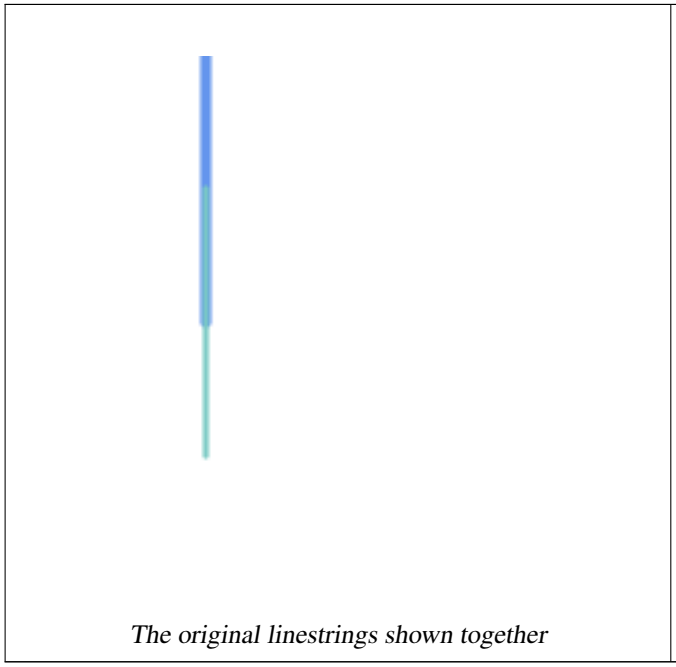
Requires GEOS >= 3.9.0 to use the gridSize parameter

This method implements the OGC Simple Features Implementation Specification for SQL 1.1. s2.1.1.3

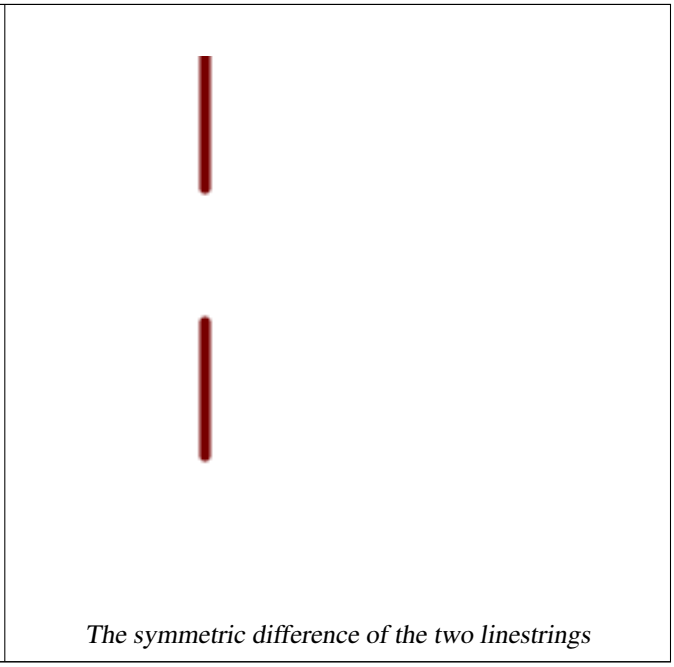
This method implements the SQL/MM specification. SQL-MM 3: 5.1.21

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))
```

See Also

ST_Difference, ST_Intersection, ST_Union, ST_ReducePrecision

7.13.9 ST_UnaryUnion

ST_UnaryUnion — Computes the union of the components of a single geometry.

Synopsis

geometry **ST_UnaryUnion**(geometry geom, float8 gridSize = -1);

Description

A single-input variant of ST_Union. The input may be a single geometry, a MultiGeometry, or a GeometryCollection. The union is applied to the individual elements of the input.

This function can be used to fix MultiPolygons which are invalid due to overlapping components. However, the input components must each be valid. An invalid input component such as a bow-tie polygon may cause an error. For this reason it may be better to use ST_MakeValid.

Another use of this function is to node and dissolve a collection of linestrings which cross or overlap to make them simple. (ST_Node also does this, but it does not provide the `gridSize` option.)

It is possible to combine ST_UnaryUnion with ST_Collect to fine-tune how many geometries are be unioned at once. This allows trading off between memory usage and compute time, striking a balance between ST_Union and ST_MemUnion.

If the optional `gridSize` parameter is given (GEOS-3.9.0 or higher required), all result vertices are guaranteed to fall on a grid of the specified size. For the operation to give predictable results all the input vertices must fall already on the specified grid, see ST_ReducePrecision.

This function supports 3d and will not drop the z-index. However, the result is computed using XY only. The result Z values are copied, averaged or interpolated.

Enhanced: 3.1.0 accept a gridSize parameter.

Requires GEOS >= 3.9.0 to use the gridSize parameter

Availability: 2.0.0

See Also

ST_Union, ST_MemUnion, ST_MakeValid, ST_Collect, ST_Node, ST_ReducePrecision

7.13.10 ST_Union

ST_Union — Computes a geometry representing the point-set union of the input geometries.

Synopsis

geometry **ST_Union**(geometry g1, geometry g2);
geometry **ST_Union**(geometry g1, geometry g2, float8 gridSize);
geometry **ST_Union**(geometry[] g1_array);
geometry **ST_Union**(geometry set g1field);
geometry **ST_Union**(geometry set g1field, float8 gridSize);

Description

Unions the input geometries, merging geometry to produce a result geometry with no overlaps. The output may be an atomic geometry, a MultiGeometry, or a Geometry Collection. Comes in several variants:

Two-input variant: returns a geometry that is the union of two input geometries. If either input is NULL, then NULL is returned.

Array variant: returns a geometry that is the union of an array of geometries.

Aggregate variant: returns a geometry that is the union of a rowset of geometries. The ST_Union() function is an "aggregate" function in the terminology of PostgreSQL. That means that it operates on rows of data, in the same way the SUM() and AVG() functions do and like most aggregates, it also ignores NULL geometries.

See ST_UnaryUnion for a non-aggregate, single-input variant.

The ST_Union array and set variants use the fast Cascaded Union algorithm described in http://blog.cleverelephant.ca/2009/01/-must-faster-unions-in-postgis-14.html

If the optional `gridSize` parameter is given (GEOS-3.9.0 or higher required), all result vertices are guaranteed to fall on a grid of the specified size. For the operation to give predictable results all the input vertices must fall already on the specified grid, see ST_ReducePrecision.

Note
ST_Collect may sometimes be used in place of ST_Union, if the result is not required to be non-overlapping. ST_Collect is usually faster than ST_Union because it performs no processing on the collected geometries.

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))
```

See Also

ST_Collect, ST_UnaryUnion, ST_MemUnion, ST_Intersection, ST_Difference, ST_SymDifference, ST_ReducePrecision

7.14 Geometry Processing

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);

Description

Computes a POLYGON or MULTIPOLYGON that represents all points whose distance from a geometry/geography is less than or equal to a given distance. A negative distance shrinks the geometry rather than expanding it. A negative distance may shrink a polygon completely, in which case POLYGON EMPTY is returned. For points and lines negative distances always return empty results.

For geometry, the distance is specified in the units of the Spatial Reference System of the geometry. For geography, the distance is specified in meters.

The optional third parameter controls the buffer accuracy and style. The accuracy of circular arcs in the buffer is specified as the number of line segments used to approximate a quarter circle (default is 8). The buffer style can be specified by providing a list of blank-separated key=value pairs as follows:

- 'quad_segs=#' : number of line segments used to approximate a quarter circle (default is 8).
- 'endcap=round|flat|square' : endcap style (defaults to "round"). 'butt' is accepted as a synonym for 'flat'.
- 'join=round|mitre|bevel' : join style (defaults to "round"). 'miter' is accepted as a synonym for 'mitre'.
- 'mitre_limit=#.#' : mitre ratio limit (only affects mitered join style). 'miter_limit' is accepted as a synonym for 'mitre_limit'.
- 'side=both|left|right' : 'left' or 'right' performs a single-sided buffer on the geometry, with the buffered side relative to the direction of the line. This is only applicable to LINESTRING geometry and does not affect POINT or POLYGON geometries. By default end caps are square.

Note

For geography this is a thin wrapper around the geometry implementation. It determines a planar spatial reference system that best fits the bounding box of the geography object (trying UTM, Lambert Azimuthal Equal Area (LAEA) North/South pole, and finally Mercator). The buffer is computed in the planar space, and then transformed back to WGS84. This may not produce the desired behavior if the input object is much larger than a UTM zone or crosses the dateline

Note

Buffer 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.

Enhanced: 2.5.0 - ST_Buffer geometry support was enhanced to allow for side buffering specification `side=both|left|right`.

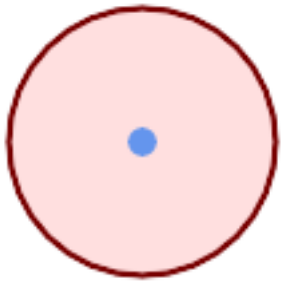
Availability: 1.5 - ST_Buffer was enhanced to support different endcaps and join types. These are useful for example to convert road linestrings into polygon roads with flat or square edges instead of rounded edges. Thin wrapper for geography was added.

Performed by the GEOS module.

This method implements the OGC Simple Features Implementation Specification for SQL 1.1. s2.1.1.3

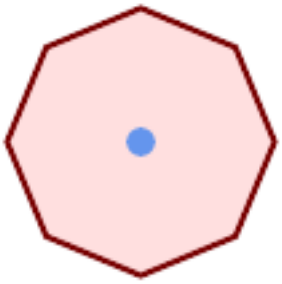
This method implements the SQL/MM specification. SQL-MM IEC 13249-3: 5.1.30

Examples



quad_segs=8 (default)

```
SELECT ST_Buffer(
 ST_GeomFromText('POINT(100 90)'),
 50, 'quad_segs=8');
```



quad_segs=2 (lame)

```
SELECT ST_Buffer(
 ST_GeomFromText('POINT(100 90)'),
 50, 'quad_segs=2');
```

endcap=round join=round (default)

```
SELECT ST_Buffer(
 ST_GeomFromText(
  'LINESTRING(50 50,150 150,150 50)'
 ), 10, 'endcap=round join=round');
```

endcap=square

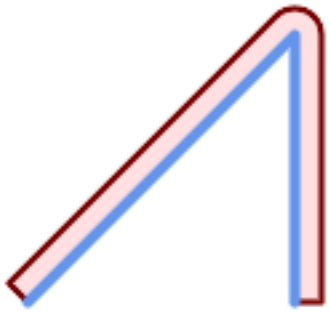
```
SELECT ST_Buffer(
 ST_GeomFromText(
  'LINESTRING(50 50,150 150,150 50)'
 ), 10, 'endcap=square join=round');
```

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');
```

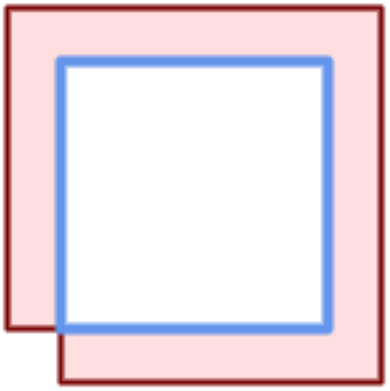


side=left

```
SELECT ST_Buffer(
 ST_GeomFromText(
  'LINESTRING(50 50,150 150,150 50)'
 ), 10, 'side=left');
```

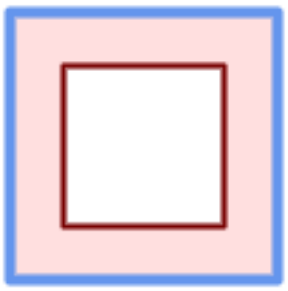
side=right

```
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))
```

See Also

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);

Description

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.

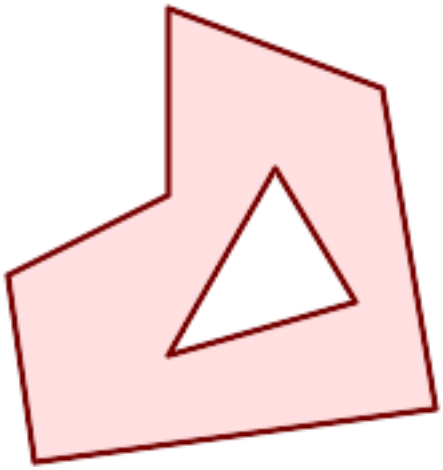
This function converts inner rings into holes. To turn inner rings into polygons as well, use ST_Polygonize.

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.

Availability: 1.1.0

Examples

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;
```

See Also

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 — Returns the geometric center of a geometry.

Synopsis

geometry **ST_Centroid**(geometry g1);
geography **ST_Centroid**(geography g1, boolean use_spheroid = true);

Description

Computes a point which is the geometric center of mass of a geometry. For [MULTI]POINTs, the centroid is the arithmetic mean of the input coordinates. For [MULTI]LINESTRINGs, the centroid is computed using the weighted length of each line segment. For [MULTI]POLYGONs, the centroid is computed in terms of area. If an empty geometry is supplied, an empty GEOMETRYCOLLECTION is returned. If NULL is supplied, NULL is returned. If CIRCULARSTRING or COMPOUNDCURVE are supplied, they are converted to linestring with CurveToLine first, then same than for LINESTRING

For mixed-dimension input, the result is equal to the centroid of the component Geometries of highest dimension (since the lower-dimension geometries contribute zero "weight" to the centroid).

Note that for polygonal geometries the centroid does not necessarily lie in the interior of the polygon. For example, see the diagram below of the centroid of a C-shaped polygon. To construct a point guaranteed to lie in the interior of a polygon use ST_PointOnSurface.

New in 2.3.0 : supports CIRCULARSTRING and COMPOUNDCURVE (using CurveToLine)

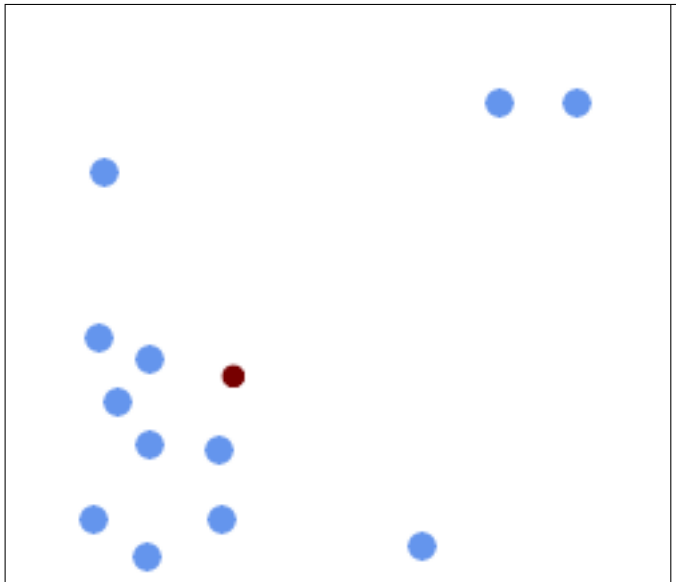
Availability: 2.4.0 support for geography was introduced.

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

This method implements the SQL/MM specification. SQL-MM 3: 8.1.4, 9.5.5

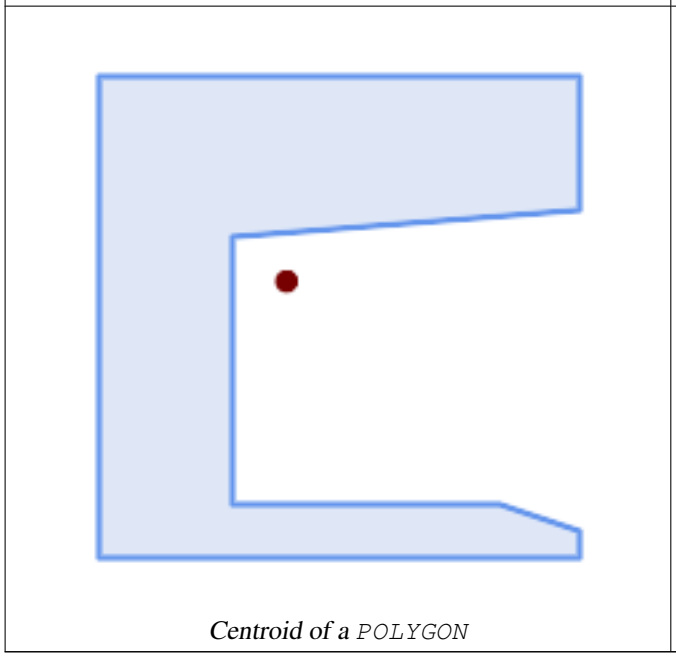
Examples

In the following illustrations the red dot is the centroid of the source geometry.

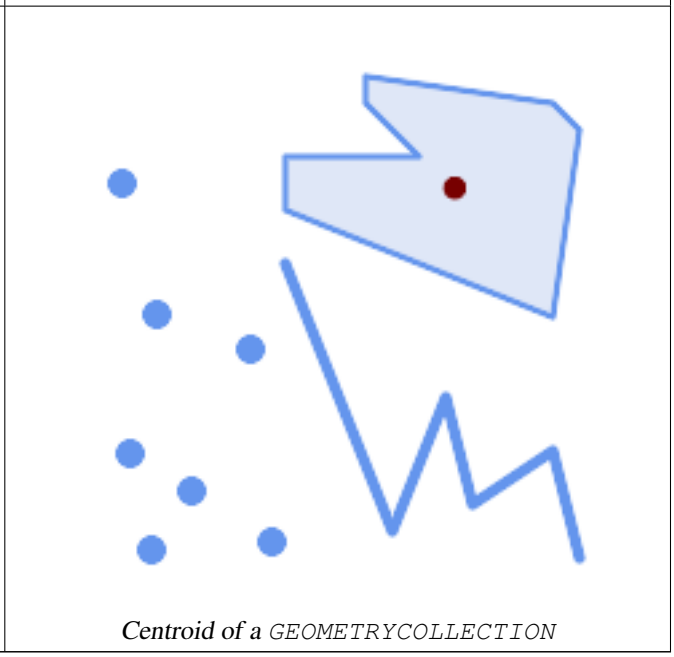


Centroid of a MULTIPOINT

Centroid of a LINESTRING



Centroid of a POLYGON



Centroid of a 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)
```

See Also

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);

Description

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 dimensions when present.

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

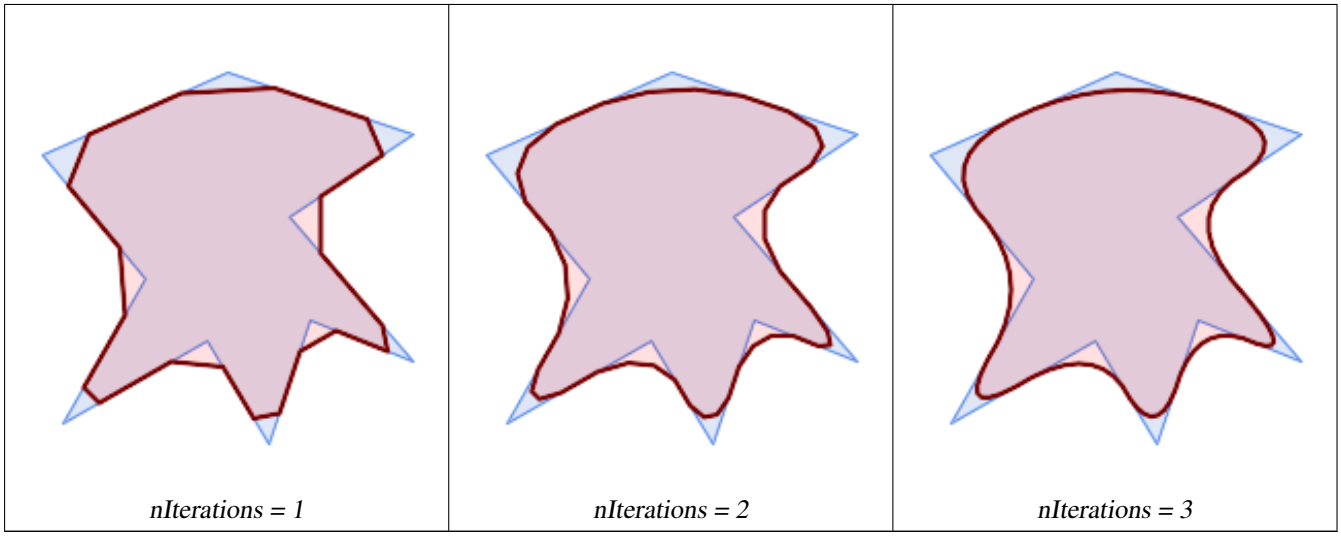
Availability: 2.5.0

Examples

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))
```

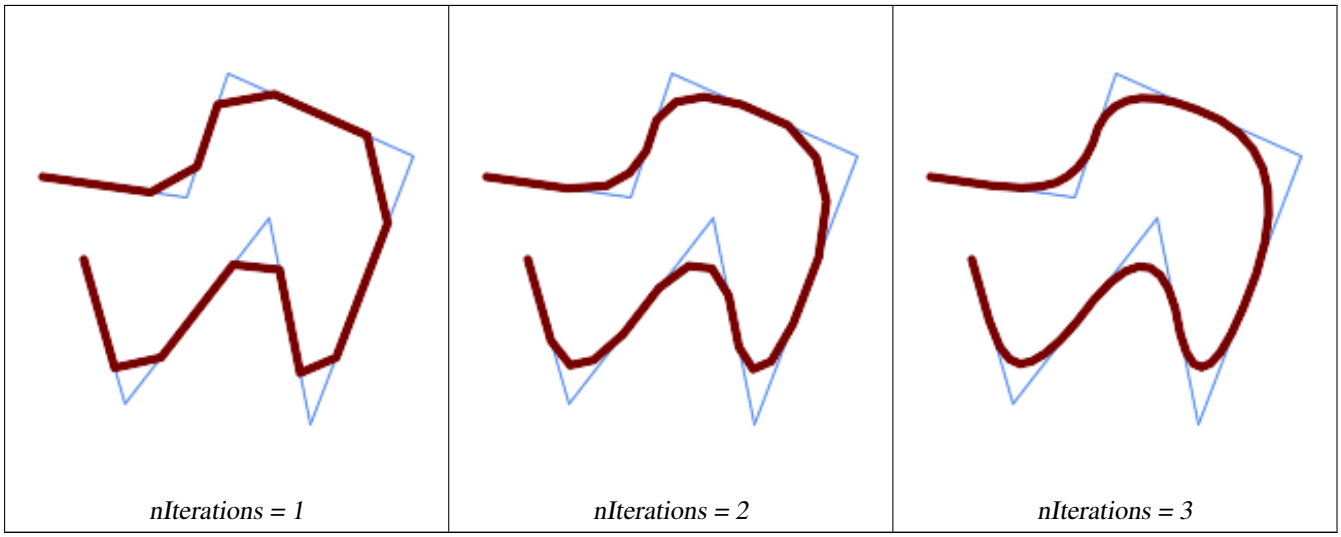
Smoothing a Polygon using 1, 2 and 3 iterations:



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) );
```

See Also

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);

Description

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.

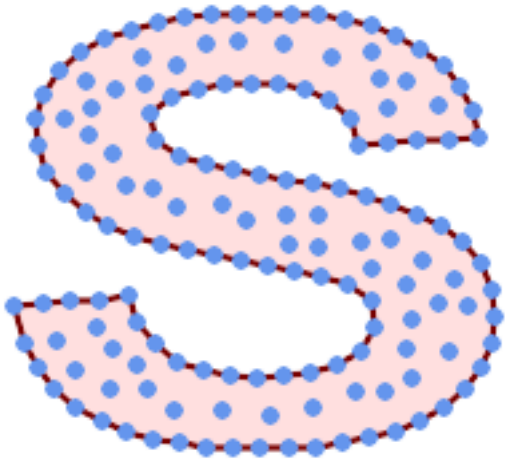
For point and linear inputs, the hull will enclose all the points of the inputs. For polygonal inputs, the hull will enclose all the points of the input *and also* all the areas covered by the input. If you want a point-wise hull of a polygonal input, convert it to points first using ST_Points.

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)`.

Availability: 2.0.0

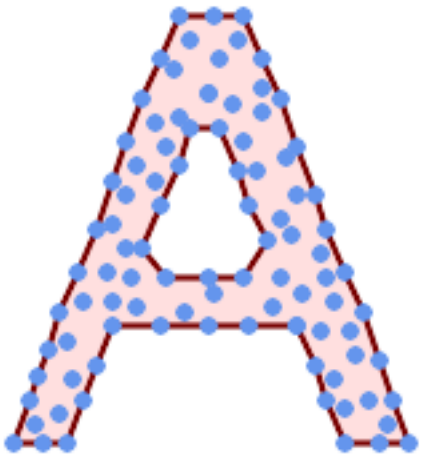
Enhanced: 3.3.0, GEOS native implementation enabled for GEOS 3.11+

Examples



Concave Hull of a MultiPoint

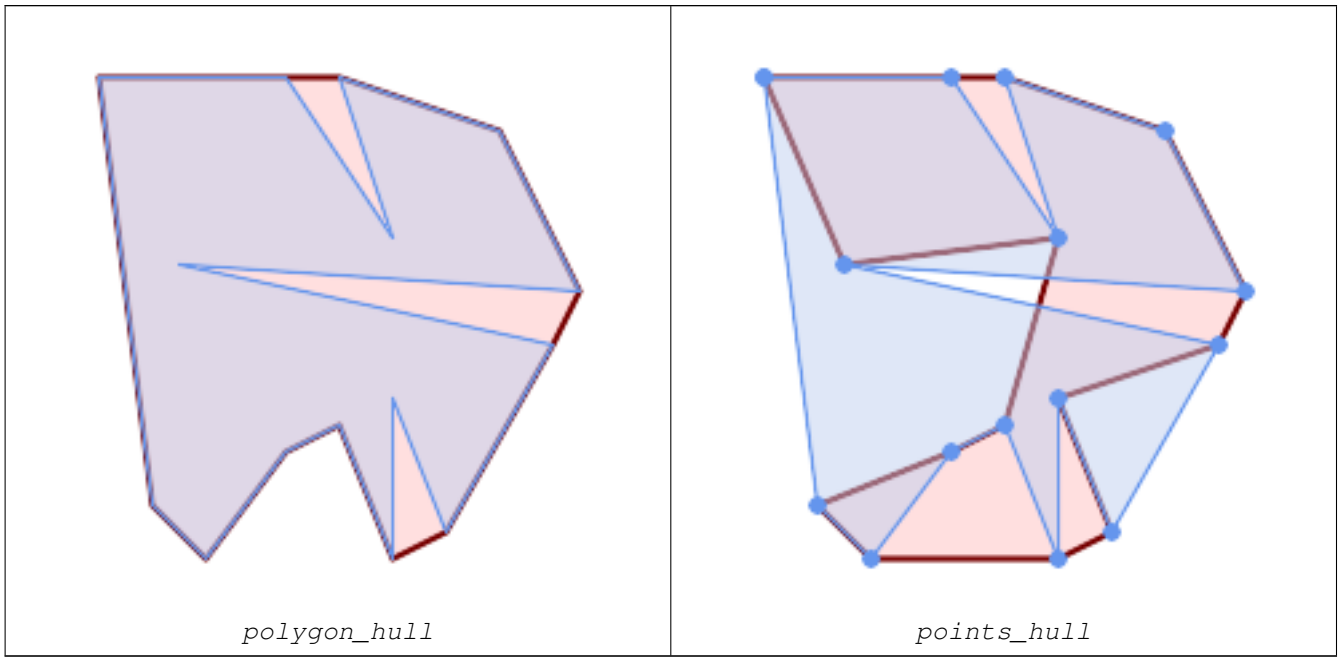
```
SELECT ST_AsText( ST_ConcaveHull(
        'MULTIPOINT ((10 72), (53 76), (56 66), (63 58), (71 51), (81 48), (91 46), (101 ↩
            45), (111 46), (121 47), (131 50), (140 55), (145 64), (144 74), (135 80), (125 ↩
            83), (115 85), (105 87), (95 89), (85 91), (75 93), (65 95), (55 98), (45 102), ↩
            (37 107), (29 114), (22 122), (19 132), (18 142), (21 151), (27 160), (35 167), ↩
            (44 172), (54 175), (64 178), (74 180), (84 181), (94 181), (104 181), (114 181) ↩
            , (124 181), (134 179), (144 177), (153 173), (162 168), (171 162), (177 154), ↩
            (182 145), (184 135), (139 132), (136 142), (128 149), (119 153), (109 155), (99 ↩
             155), (89 155), (79 153), (69 150), (61 144), (63 134), (72 128), (82 125), (92 ↩
             123), (102 121), (112 119), (122 118), (132 116), (142 113), (151 110), (161 ↩
            106), (170 102), (178 96), (185 88), (189 78), (190 68), (189 58), (185 49), ↩
            (179 41), (171 34), (162 29), (153 25), (143 23), (133 21), (123 19), (113 19), ↩
            (102 19), (92 19), (82 19), (72 21), (62 22), (52 25), (43 29), (33 34), (25 41) ↩
            , (19 49), (14 58), (21 73), (31 74), (42 74), (173 134), (161 134), (150 133), ↩
            (97 104), (52 117), (157 156), (94 171), (112 106), (169 73), (58 165), (149 40) ↩
            , (70 33), (147 157), (48 153), (140 96), (47 129), (173 55), (144 86), (159 67) ↩
            , (150 146), (38 136), (111 170), (124 94), (26 59), (60 41), (71 162), (41 64), ↩
             (88 110), (122 34), (151 97), (157 56), (39 146), (88 33), (159 45), (47 56), ↩
            (138 40), (129 165), (33 48), (106 31), (169 147), (37 122), (71 109), (163 89), ↩
             (37 156), (82 170), (180 72), (29 142), (46 41), (59 155), (124 106), (157 80), ↩
             (175 82), (56 50), (62 116), (113 95), (144 167))',
         0.1 ) );
---st_astext--
POLYGON ((18 142, 21 151, 27 160, 35 167, 44 172, 54 175, 64 178, 74 180, 84 181, 94 181, ↩
    104 181, 114 181, 124 181, 134 179, 144 177, 153 173, 162 168, 171 162, 177 154, 182 ↩
    145, 184 135, 173 134, 161 134, 150 133, 139 132, 136 142, 128 149, 119 153, 109 155, 99 ↩
     155, 89 155, 79 153, 69 150, 61 144, 63 134, 72 128, 82 125, 92 123, 102 121, 112 119, ↩
    122 118, 132 116, 142 113, 151 110, 161 106, 170 102, 178 96, 185 88, 189 78, 190 68, ↩
    189 58, 185 49, 179 41, 171 34, 162 29, 153 25, 143 23, 133 21, 123 19, 113 19, 102 19, ↩
    92 19, 82 19, 72 21, 62 22, 52 25, 43 29, 33 34, 25 41, 19 49, 14 58, 10 72, 21 73, 31 ↩
    74, 42 74, 53 76, 56 66, 63 58, 71 51, 81 48, 91 46, 101 45, 111 46, 121 47, 131 50, 140 ↩
     55, 145 64, 144 74, 135 80, 125 83, 115 85, 105 87, 95 89, 85 91, 75 93, 65 95, 55 98, ↩
    45 102, 37 107, 29 114, 22 122, 19 132, 18 142))
```



Concave Hull of a MultiPoint, allowing holes

```
SELECT ST_AsText( ST_ConcaveHull(
        'MULTIPOINT ((132 64), (114 64), (99 64), (81 64), (63 64), (57 49), (52 36), (46 ↩
            20), (37 20), (26 20), (32 36), (39 55), (43 69), (50 84), (57 100), (63 118), ↩
            (68 133), (74 149), (81 164), (88 180), (101 180), (112 180), (119 164), (126 ↩
            149), (132 131), (139 113), (143 100), (150 84), (157 69), (163 51), (168 36), ↩
```

```
            (174 20), (163 20), (150 20), (143 36), (139 49), (132 64), (99 151), (92 138),  ↩
            (88 124), (81 109), (74 93), (70 82), (83 82), (99 82), (112 82), (126 82), (121 ↩
             96), (114 109), (110 122), (103 138), (99 151), (34 27), (43 31), (48 44), (46  ↩
            58), (52 73), (63 73), (61 84), (72 71), (90 69), (101 76), (123 71), (141 62),  ↩
            (166 27), (150 33), (159 36), (146 44), (154 53), (152 62), (146 73), (134 76),  ↩
            (143 82), (141 91), (130 98), (126 104), (132 113), (128 127), (117 122), (112  ↩
            133), (119 144), (108 147), (119 153), (110 171), (103 164), (92 171), (86 160), ↩
             (88 142), (79 140), (72 124), (83 131), (79 118), (68 113), (63 102), (68 93),  ↩
            (35 45))',
          0.15, true ) );
---st_astext--
POLYGON ((43 69, 50 84, 57 100, 63 118, 68 133, 74 149, 81 164, 88 180, 101 180, 112 180,  ↩
    119 164, 126 149, 132 131, 139 113, 143 100, 150 84, 157 69, 163 51, 168 36, 174 20, 163 ↩
     20, 150 20, 143 36, 139 49, 132 64, 114 64, 99 64, 81 64, 63 64, 57 49, 52 36, 46 20,  ↩
    37 20, 26 20, 32 36, 35 45, 39 55, 43 69), (88 124, 81 109, 74 93, 83 82, 99 82, 112 82, ↩
     121 96, 114 109, 110 122, 103 138, 92 138, 88 124))
```



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;
```

Using with ST_Collect to compute the concave hull of a geometry set.

```
-- Compute estimate of infected area based on point observations
SELECT disease_type,
    ST_ConcaveHull( ST_Collect(obs_pnt), 0.3 ) AS geom
  FROM disease_obs
  GROUP BY disease_type;
```

See Also

ST_ConvexHull, ST_Collect, ST_AlphaShape, ST_OptimalAlphaShape

7.14.6 ST_ConvexHull

ST_ConvexHull — Computes the convex hull of a geometry.

Synopsis

geometry ST_ConvexHull(geometry geomA);

Description

Computes the convex hull of a geometry. The convex hull is the smallest convex geometry that encloses all geometries in input.

One can think of the convex hull as the geometry obtained by wrapping an rubber band around a set of geometries. Th different from a concave hull which is analogous to "shrink-wrapping" the geometries. A convex hull is often used to determi 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. T 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 in geometry collection (e.g. ST_ConvexHull(ST_Collect(geom))).

Performed by the GEOS module

This method implements the OGC Simple Features Implementation Speci cation for SQL 1.1. s2.1.1.3

This method implements the SQL/MM speci cation. SQL-MM IEC 13249-3: 5.1.16

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

Examples

Convex Hull of a MultiLinestring and a MultiPoint

Note
Only use with MultiLineString/LineStrings. Other geometry types return an empty GeometryCollection

Performed by the GEOS module.

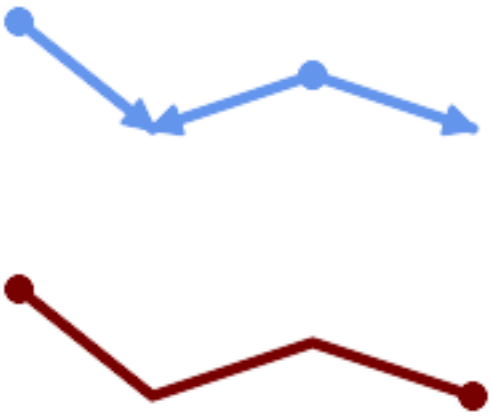
Enhanced: 3.3.0 accept a directed parameter.

Requires GEOS >= 3.11.0 to use the directed parameter.

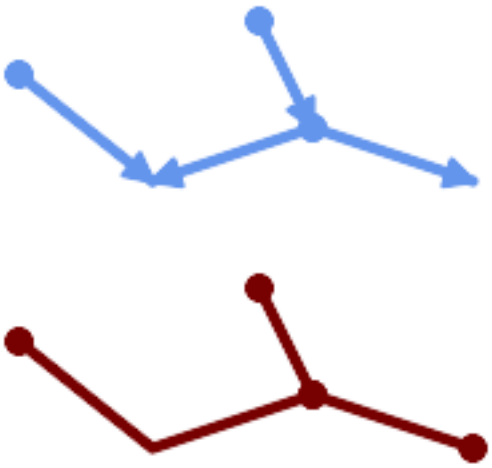
Availability: 1.1.0

Warning
This function strips the M dimension.

Examples



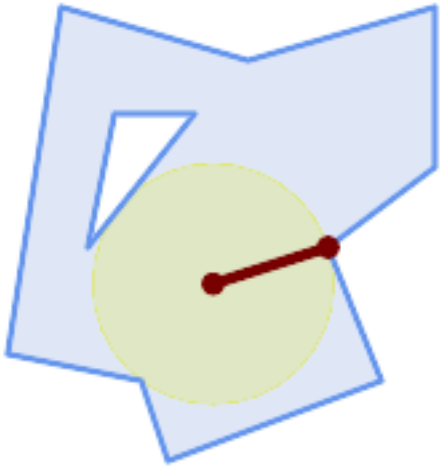
Merging lines with different orientation.



Lines are not merged across intersections with degree > 2.

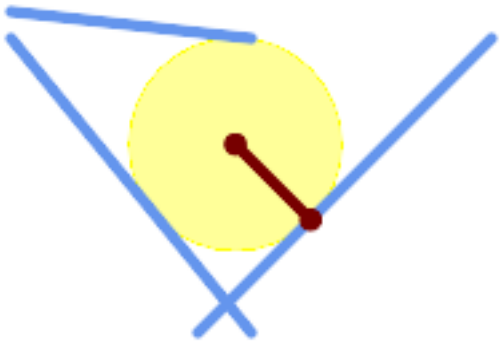
```
SELECT ST_AsText(ST_LineMerge(
'MULTILINESTRING
```

Examples



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.

See Also

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);

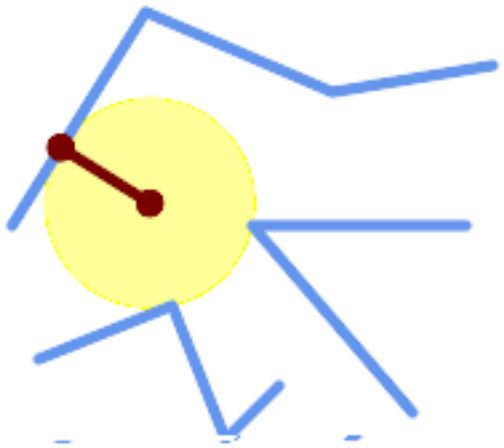
Description

Finds the largest circle which does not overlap a set of point and line obstacles. (Polygonal geometries may be included as obstacles, but only their boundary lines are used.) The center of the circle is constrained to lie inside a polygonal boundary, which by default is the convex hull of the input geometry. The circle center is the point in the interior of the boundary which has the farthest distance from the obstacles. The circle itself is provided by the center point and a nearest point lying on an obstacle determining the circle radius.

The circle center is determined to a given accuracy specified by a distance tolerance, using an iterative algorithm. If the accuracy distance is not specified a reasonable default is used.

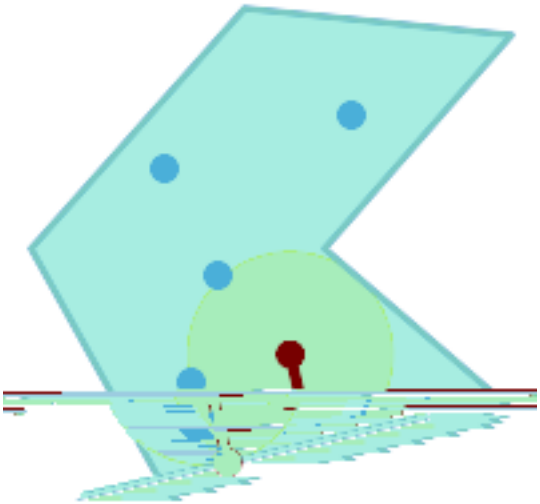
Returns a record with fields:

- `center` - center point of the circle
- `nearest`center



Largest Empty Circle within a set of lines.

```
SELECT radius,
       center,
       nearest
  FROM ST_LargestEmptyCircle(
         ST_Collect(
           'MULTIPOINT ((70 50), (60 130), (130 150), (80 90))'::geometry,
           'POLYGON ((90 190, 10 100, 60 10, 190 40, 120 100, 190 180, 90 190))'::geometry) ↩
               ,
           0,
         'POLYGON ((90 190, 10 100, 60 10, 190 40, 120 100, 190 180, 90 190))'::geometry
       );
```



Largest Empty Circle within a set of points, constrained to lie in a polygon. The constraint polygon boundary must be included as an obstacle, as well as specified as the constraint for the circle center.

See Also

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);

Description

Returns the smallest circle polygon that contains a geometry.

> Note
> The bounding circle is approximated by a polygon with a default of 48 segments per quarter circle. Because the polygon is an approximation of the minimum bounding circle, some points in the input geometry may not be contained within the polygon. The approximation can be improved by increasing the number of segments. For applications where an approximation is not suitable ST_MinimumBoundingRadius may be used.

Use with ST_Collect to get the minimum bounding circle of a set of geometries.

To compute two points lying on the minimum circle (the "maximum diameter") use ST_LongestLine.

The ratio of the area of a polygon divided by the area of its Minimum Bounding Circle is referred to as the Reock compactness score

Performed by the GEOS module.

Availability: 1.4.0

Examples

```
SELECT  d.disease_type,
      ST_MinimumBoundingCircle(ST_Collect(d.geom))  As  geom
      FROM  disease_obs  As  d
      GROUP  BY  d.disease_type;
```

Minimum bounding circle of a point and linestring. Using 8 segs to approximate a quarter circle

Examples

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
```

See Also

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);

Description

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
```

See Also

ST_GeomFromEWKT, ST_GeomFromText, ST_XMin, ST_XMax, ST_YMax, ST_YMin, ST_ZMax

7.19 Linear Referencing

7.19.1 ST_LineInterpolatePoint

ST_LineInterpolatePoint — Returns a point interpolated along a line at a fractional location.

Synopsis

geometry **ST_LineInterpolatePoint**(geometry a_linestring, float8 a_fraction);
geography **ST_LineInterpolatePoint**(geography a_linestring, float8 a_fraction, boolean use_spheroid = true);

Description

Returns a point interpolated along a line at a fractional location. First argument must be a LINESTRING. Second argument is a float between 0 and 1 representing the fraction of line length where the point is to be located. The Z and M values are interpolated if present.

See ST_LineLocatePoint for computing the line location nearest to a Point.

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.

Description

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.

If `startfraction` and `endfraction` have the same value this is equivalent to ST_LineInterpolatePoint.

Note
This only works with LINESTRINGs. To use on contiguous MULTILINESTRINGs first join them with ST_LineMerge.

Note
Since release 1.1.1 this function interpolates M and Z values. Prior releases set Z and M to unspecified values.

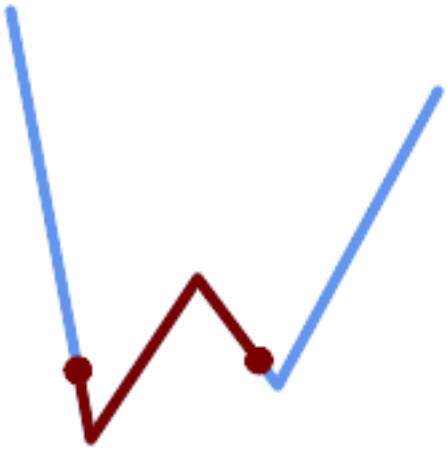
Enhanced: 3.4.0 - Support for geography was introduced.

Changed: 2.1.0. Up to 2.0.x this was called ST_Line_Substring.

Availability: 1.1.0, Z and M supported added in 1.1.1

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

Examples



A LineString seen with 1/3 midrange overlaid (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)
```

See Also

ST_Length, ST_LineExtend, ST_LineInterpolatePoint, ST_LineMerge

7.19.6 ST_LocateAlong

ST_LocateAlong — Returns the point(s) on a geometry that match a measure value.

Synopsis

geometry **ST_LocateAlong**(geometry geom_with_measure, float8 measure, float8 offset = 0);

```
---------------------------------------
 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);

Description

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
```

See Also

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);

Description

Returns the smallest measure at which points interpolated along the given trajectories are the least distance apart.

Inputs must be valid trajectories as checked by ST_IsValidTrajectory. Null is returned if the trajectories do not overlap in their M ranges.

To obtain the actual points at the computed measure use ST_LocateAlong .

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_GeometryN(ST_LocateAlong(a,m),1) pa,
         ST_GeometryN(ST_LocateAlong(b,m),1) pb
  FROM inp, cpa
)
SELECT to_timestamp(m) t,
       ST_3DDistance(pa,pb) distance,
       ST_AsText(pa, 2) AS pa, ST_AsText(pb, 2) AS pb
FROM points, cpa;

               t               |      distance      |                  pa                  ↩
                   |                  pb
-------------------------------+--------------------+--------------------------------------+--------

 2015-05-26 10:45:31.034483-07 | 1.9652147377620688 | POINT ZM (7.59 0 3.79 1432662331.03)  ↩
     | POINT ZM (9.1 1.24 3.93 1432662331.03)
```

Description

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
```

See Also

ST_IsValidTrajectory, ST_ClosestPointOfApproach, ST_DistanceCPA, |=|

7.21 Version Functions

7.21.1 PostGIS_Extensions_Upgrade

PostGIS_Extensions_Upgrade — Packages and upgrades PostGIS extensions (e.g. postgis_raster, postgis_topology, postgis_sfcgal) to given or latest version.

Synopsis

text **PostGIS_Extensions_Upgrade**(text target_version=null);

Description

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)
```

See Also

Section 3.4, PostGIS_GEOS_Version, PostGIS_Lib_Version, PostGIS_LibXML_Version, PostGIS_PROJ_Version, PostGIS_Version

7.21.2 PostGIS_Full_Version

PostGIS_Full_Version — Reports full PostGIS version and build configuration infos.

Synopsis

text **PostGIS_Full_Version**();

Description

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)
```

Synopsis

text **PostGIS_Scripts_Build_Date**();

Description

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)
```

See Also

PostGIS_Full_Version, PostGIS_GEOS_Version, PostGIS_Lib_Version, PostGIS_LibXML_Version, PostGIS_Version

7.21.14 PostGIS_Scripts_Installed

PostGIS_Scripts_Installed — Returns version of the PostGIS scripts installed in this database.

Synopsis

text **PostGIS_Scripts_Installed**();

Description

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)
```

See Also

PostGIS_Full_Version, PostGIS_Scripts_Released, PostGIS_Version

7.21.15 PostGIS_Scripts_Released

PostGIS_Scripts_Released — Returns the version number of the postgis.sql script released with the installed PostGIS lib.

Synopsis

text **PostGIS_Scripts_Released**();

Description

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)
```

See Also

PostGIS_Full_Version, PostGIS_Scripts_Installed, PostGIS_Lib_Version

7.21.16 PostGIS_Version

PostGIS_Version — Returns PostGIS version number and compile-time options.

Synopsis

text **PostGIS_Version**();

Description

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)
```

See Also

PostGIS_Full_Version, PostGIS_GEOS_Version, PostGIS_Lib_Version, PostGIS_LibXML_Version, PostGIS_PROJ_Version

7.22 Grand Unified Custom Variables (GUCs)

7.22.1 postgis.gdal_datapath

postgis.gdal_datapath — A configuration option to assign the value of GDAL's GDAL_DATA option. If not set, the environmentally set GDAL_DATA variable is used.

Description

A PostgreSQL GUC variable for setting the value of GDAL's GDAL_DATA option. The `postgis.gdal_datapath` value should be the complete physical path to GDAL's data files.

This configuration option is of most use for Windows platforms where GDAL's data files path is not hard-coded. This option should also be set when GDAL's data files are not located in GDAL's expected path.

Note
This option can be set in PostgreSQL's configuration file postgresql.conf. It can also be set by connection or transaction.

Availability: 2.2.0

Note
Additional information about GDAL_DATA is available at GDAL's Configuration Options.

Examples

Set and reset `postgis.gdal_datapath`

```
SET postgis.gdal_datapath TO '/usr/local/share/gdal.hidden';
SET postgis.gdal_datapath TO default;
```

Setting on windows for a particular database

```
ALTER DATABASE gisdb
SET postgis.gdal_datapath = 'C:/Program Files/PostgreSQL/9.3/gdal-data';
```

See Also

PostGIS_GDAL_Version, ST_Transform

7.22.2 postgis.gdal_enabled_drivers

postgis.gdal_enabled_drivers — A configuration option to set the enabled GDAL drivers in the PostGIS environment. Affects the GDAL configuration variable GDAL_SKIP.

Description

A configuration option to set the enabled GDAL drivers in the PostGIS environment. Affects the GDAL configuration variable GDAL_SKIP. This option can be set in PostgreSQL's configuration file: postgresql.conf. It can also be set by connection or transaction.

The initial value of `postgis.gdal_enabled_drivers` may also be set by passing the environment variable `POSTGIS_GDAL_EN` with the list of enabled drivers to the process starting PostgreSQL.

Enabled GDAL specified drivers can be specified by the driver's short-name or code. Driver short-names or codes can be found at GDAL Raster Formats. Multiple drivers can be specified by putting a space between each driver.

Note
There are three special codes available for `postgis.gdal_enabled_drivers`. The codes are case-sensitive.

- `DISABLE_ALL` disables all GDAL drivers. If present, `DISABLE_ALL` overrides all other values in `postgis.gdal_enabled_drivers`.
- `ENABLE_ALL` enables all GDAL drivers.
- `VSICURL` enables GDAL's `/vsicurl/` virtual file system.

When `postgis.gdal_enabled_drivers` is set to DISABLE_ALL, attempts to use out-db rasters, ST_FromGDALRaster(), ST_AsGDALRaster(), ST_AsTIFF(), ST_AsJPEG() and ST_AsPNG() will result in error messages.

Note
In the standard PostGIS installation, `postgis.gdal_enabled_drivers` is set to DISABLE_ALL.

Note
Additional information about GDAL_SKIP is available at GDAL's Configuration Options.

Availability: 2.2.0

Examples

To set and reset `postgis.gdal_enabled_drivers` for current session

```
SET postgis.gdal_enabled_drivers = 'ENABLE_ALL';
SET postgis.gdal_enabled_drivers = default;
```

Set for all new connections to a specific database to specific drivers

ve DATABASEs mygisdbs SETs pPosgiss

```
 --writes to postgres.auto.conf
ALTER SYSTEM SET postgis.enable_outdb_rasters = true;
 --Reloads postgres conf
SELECT pg_reload_conf();
```

See Also

postgis.gdal_enabled_drivers postgis.gdal_vsi_options

7.22.4 postgis.gdal_vsi_options

postgis.gdal_vsi_options — A string configuration to set options used when working with an out-db raster.

Description

A string configuration to set options used when working with an out-db raster. Configuration options control things like how much space GDAL allocates to local data cache, whether to read overviews, and what access keys to use for remote out-db data sources.

Availability: 3.2.0

Examples

Set `postgis.gdal_vsi_options` for current session:

```
SET postgis.gdal_vsi_options = 'AWS_ACCESS_KEY_ID=xxxxxxxxxxxxxxx AWS_SECRET_ACCESS_KEY= ↩
    yyyyyyyyyyyyyyyyyyyyyyyyyyy';
```

Set `postgis.gdal_vsi_options` just for the *current transaction* using the `LOCAL` keyword:

```
SET LOCAL postgis.gdal_vsi_options = 'AWS_ACCESS_KEY_ID=xxxxxxxxxxxxxxx  ↩
    AWS_SECRET_ACCESS_KEY=yyyyyyyyyyyyyyyyyyyyyyyyyyy';
```

See Also

postgis.enable_outdb_rasters postgis.gdal_enabled_drivers

7.23 Troubleshooting Functions

7.23.1 PostGIS_AddBBox

PostGIS_AddBBox — Add bounding box to the geometry.

Synopsis

geometry **PostGIS_AddBBox**(geometry geomA);

Description

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 re-add.

This method supports Circular Strings and Curves.

Examples

```
UPDATE sometable
 SET geom =  PostGIS_AddBBox(geom)
 WHERE PostGIS_HasBBox(geom) = false;
```

See Also

PostGIS_DropBBox, PostGIS_HasBBox

7.23.2 PostGIS_DropBBox

PostGIS_DropBBox — Drop the bounding box cache from the geometry.

Synopsis

geometry **PostGIS_DropBBox**(geometry geomA);

Description

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 re-add. 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 re-add the bbox or do a dump reload.

This method supports Circular Strings and Curves.

Examples

PostGI_AddBBoxs

=s

Chapter 8

SFCGAL Functions Reference

SFCGAL is a C++ wrapper library around CGAL that provides advanced 2D and 3D spatial functions. For robustness, geometry coordinates have an exact rational number representation.

Installation instructions for the library can be found on the SFCGAL home page (http://www.sfcgal.org). To enable the functions use `create extension postgis_sfcgal`.

8.1 SFCGAL Management Functions

8.1.1 postgis_sfcgal_version

postgis_sfcgal_version — Returns the version of SFCGAL in use

Synopsis

text **postgis_sfcgal_version**(void);

Description

Returns the version of SFCGAL in use

Availability: 2.1.0

This method needs SFCGAL backend.

See Also

postgis_sfcgal_full_version

8.1.2 postgis_sfcgal_full_version

postgis_sfcgal_full_version — Returns the full version of SFCGAL in use including CGAL and Boost versions

Synopsis

text **postgis_sfcgal_full_version**(void);

Description

Returns the full version of SFCGAL in use including CGAL and Boost versions

Availability: 3.3.0

This method needs SFCGAL backend.

See Also

postgis_sfcgal_version

8.2 SFCGAL Accessors and Setters

8.2.1 CG_ForceLHR

CG_ForceLHR — Force LHR orientation

Synopsis

geometry **CG_ForceLHR**(geometry geom);

Description

Availability: 3.5.0

This method needs SFCGAL backend.

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

This function supports Polyhedral surfaces.

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

8.2.2 CG_IsPlanar

CG_IsPlanar — Check if a surface is or not planar

Synopsis

boolean **CG_IsPlanar**(geometry geom);

Description

Availability: 3.5.0

This method needs SFCGAL backend.

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

This function supports Polyhedral surfaces.

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

Description

The function only applies to polygons. It returns -1 if the polygon is counterclockwise oriented and 1 if the polygon is clockwise oriented.

Availability: 3.5.0

This method needs SFCGAL backend.

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

8.2.6 CG_Area

CG_Area — Calculates the area of a geometry

Synopsis

double precision **CG_Area**(geometry geom);

Description

Calculates the area of a geometry.

Performed by the SFCGAL module

Note
NOTE: this function returns a double precision value representing the area.

Availability: 3.5.0

This method needs SFCGAL backend.

Geometry Examples

```
SELECT CG_Area('Polygon ((0 0, 0 5, 5 5, 5 0, 0 0), (1 1, 2 1, 2 2, 1 2, 1 1), (3 3, 4 3, 4 ↩
    4, 3 4, 3 3))');
               cg_area
               --------
               25
               (1 row)
```

See Also

ST_3DArea, ST_Area

8.2.7 CG_3DArea

CG_3DArea — Computes area of 3D surface geometries. Will return 0 for solids.

Description

Availability: 3.5.0

This method needs SFCGAL backend.

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

This function supports Polyhedral surfaces.

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

This method implements the SQL/MM specification. SQL-MM IEC 13249-3: 9.1 (same as CG_3DVolume)

Example

When closed surfaces are created with WKT, they are treated as areal rather than solid. To make them solid, you need to use CG_MakeSolid. Areal geometries have no volume. Here is an example to demonstrate.

```
SELECT CG_Volume(geom) As cube_surface_vol,
    CG_Volume(CG_MakeSolid(geom)) As solid_surface_vol
    FROM (SELECT 'POLYHEDRALSURFACE( ((0 0 0, 0 0 1, 0 1 1, 0 1 0, 0 0 0)),
    ((0 0 0, 0 1 0, 1 1 0, 1 0 0, 0 0 0)),
    ((0 0 0, 1 0 0, 1 0 1, 0 0 1, 0 0 0)),
    ((1 1 0, 1 1 1, 1 0 1, 1 0 0, 1 1 0)),
    ((0 1 0, 0 1 1, 1 1 1, 1 1 0, 0 1 0)),
    ((0 0 1, 1 0 1, 1 1 1, 0 1 1, 0 0 1)) )'::geometry) As f(geom);

    cube_surface_vol | solid_surface_vol
    -----------------+------------------
    0 |                1
```

See Also

CG_3DArea, CG_MakeSolid, CG_IsSolid

8.2.9 ST_ForceLHR

ST_ForceLHR — Force LHR orientation

Synopsis

geometry **ST_ForceLHR**(geometry geom);

Description

Warning
ST_ForceLHR is deprecated as of 3.5.0. Use CG_ForceLHR instead.

The function only applies to polygons. It returns -1 if the polygon is counterclockwise oriented and 1 if the polygon is clockwis
oriented.

Availability: 2.1.0

This method needs SFCGAL backend.

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

8.2.14 ST_3DArea

ST_3DArea — Computes area of 3D surface geometries. Will return 0 for solids.

Synopsis

oat ST_3DArea(geometry geom1);

Description

Warning
ST_3DArea is deprecated as of 3.5.0. Use CG_3DArea instead.

Availability: 2.1.0

This method needs SFCGAL backend.

This method implements the SQL/MM speci cation. 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).

Examples

Note: By default a PolyhedralSurface built from WKT is a surface geometry, not solid. It therefore has surface area. Or
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
```

Description

Warning
ST_AlphaShape is deprecated as of 3.5.0. Use CG_AlphaShape instead.

Computes the Alpha-Shape of the points in a geometry. An alpha-shape is a (usually) concave polygonal geometry which contains all the vertices of the input, and whose vertices are a subset of the input vertices. An alpha-shape provides a closer fit to the shape of the input than the shape produced by the convex hull.

8.3.17 CG_AlphaShape

CG_AlphaShape — Computes an Alpha-shape enclosing a geometry

Synopsis

geometry **CG_AlphaShape**(geometry geom, float alpha, boolean allow_holes = false);

Description

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.

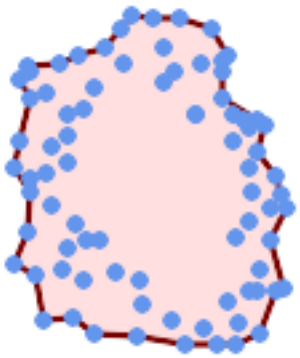
The "closeness of fit" is controlled by the `alpha` parameter, which can have values from 0 to infinity. Smaller alpha values produce more concave results. Alpha values greater than some data-dependent value produce the convex hull of the input.

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 alg5(it[(.)]TJ 0 -17.933 Td A)7334(v)25imibil2ni:s. 3.5nte quirolesCGAL=

Examples



Alpha-shape of a MultiPoint (same example As CG_OptimalAlphaShape)

```
SELECT ST_AsText(CG_AlphaShape('MULTIPOINT((63 84),(76 88),(68 73),(53 18),(91 50),(81 70),
            (88 29),(24 82),(32 51),(37 23),(27 54),(84 19),(75 87),(44 42),(77 67),(90 30) ↩
               ,(36 61),(32 65),
            (81 47),(88 58),(68 73),(49 95),(81 60),(87 50),
            (78 16),(79 21),(30 22),(78 43),(26 85),(48 34),(35 35),(36 40),(31 79),(83 29) ↩
               ,(27 84),(52 98),(72 95),(85 71),
            (75 84),(75 77),(81 29),(77 73),(41 42),(83 72),(23 36),(89 53),(27 57),(57 97) ↩
               ,(27 77),(39 88),(60 81),
            (80 72),(54 32),(55 26),(62 22),(70 20),(76 27),(84 35),(87 42),(82 54),(83 64) ↩
               ,(69 86),(60 90),(50 86),(43 80),(36 73),
            (36 68),(40 75),(24 67),(23 60),(26 44),(28 33),(40 32),(43 19),(65 16),(73 16) ↩
               ,(38 46),(31 59),(34 86),(45 90),(64 97))'::geometry,80.2));
```

```
POLYGON((89 53,91 50,87 42,90 30,88 29,84 19,78 16,73 16,65 16,53 18,43 19,
        37 23,30 22,28 33,23 36,26 44,27 54,23 60,24 67,27 77,
        24 82,26 85,34 86,39 88,45 90,49 95,52 98,57 97,
        64 97,72 95,76 88,75 84,83 72,85 71,88 58,89 53))
```

Alpha-shape of a MultiPoint, allowing holes (same example as CG_OptimalAlphaShape)

```
            (74 93,81 109,88 124,92 138,103 138,110 122,114 109,121 96,112 82,99 82,83  ↩
               82,74 93))
```

See Also

ST_ConcaveHull, CG_OptimalAlphaShape

8.3.18 CG_ApproxConvexPartition

CG_ApproxConvexPartition — Computes approximal convex partition of the polygon geometry

Synopsis

geometry **CG_ApproxConvexPartition**(geometry geom);

Description

Computes approximal convex partition of the polygon geometry (using a triangulation).

Note
A partition of a polygon P is a set of polygons such that the interiors of the polygons do not intersect and the union of the polygons is equal to the interior of the original polygon P. CG_ApproxConvexPartition and CG_GreeneApproxConvexPartition functions produce approximately optimal convex partitions. Both these functions produce convex decompositions by first decomposing the polygon into simpler polygons; CG_ApproxConvexPartition uses a triangulation and CG_GreeneApproxConvexPartition a monotone partition. These two functions both guarantee that they will produce no more than four times the optimal number of convex pieces but they differ in their runtime complexities. Though the triangulation-based approximation algorithm often results in fewer convex pieces, this is not always the case.

Availability: 3.5.0 - requires SFCGAL >= 1.5.0.

Requires SFCGAL >= 1.5.0

This method needs SFCGAL backend.

Return an approximate medial axis for the areal input based on its straight skeleton. Uses an SFCGAL specific API when built against a capable version (1.2.0+). Otherwise the function is just a wrapper around CG_StraightSkeleton (slower case).

Availability: 2.2.0

This method needs SFCGAL backend.

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

This function supports Polyhedral surfaces.

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

8.3.20 CG_ApproximateMedialAxis

CG_ApproximateMedialAxis — Compute the approximate medial axis of an areal geometry.

Synopsis

geometry **CG_ApproximateMedialAxis**(geometry geom);

Description

Return an approximate medial axis for the areal input based on its straight skeleton. Uses an SFCGAL specific API when built against a capable version (1.2.0+). Otherwise the function is just a wrapper around CG_StraightSkeleton (slower case).

Availability: 3.5.0

This method needs SFCGAL backend.

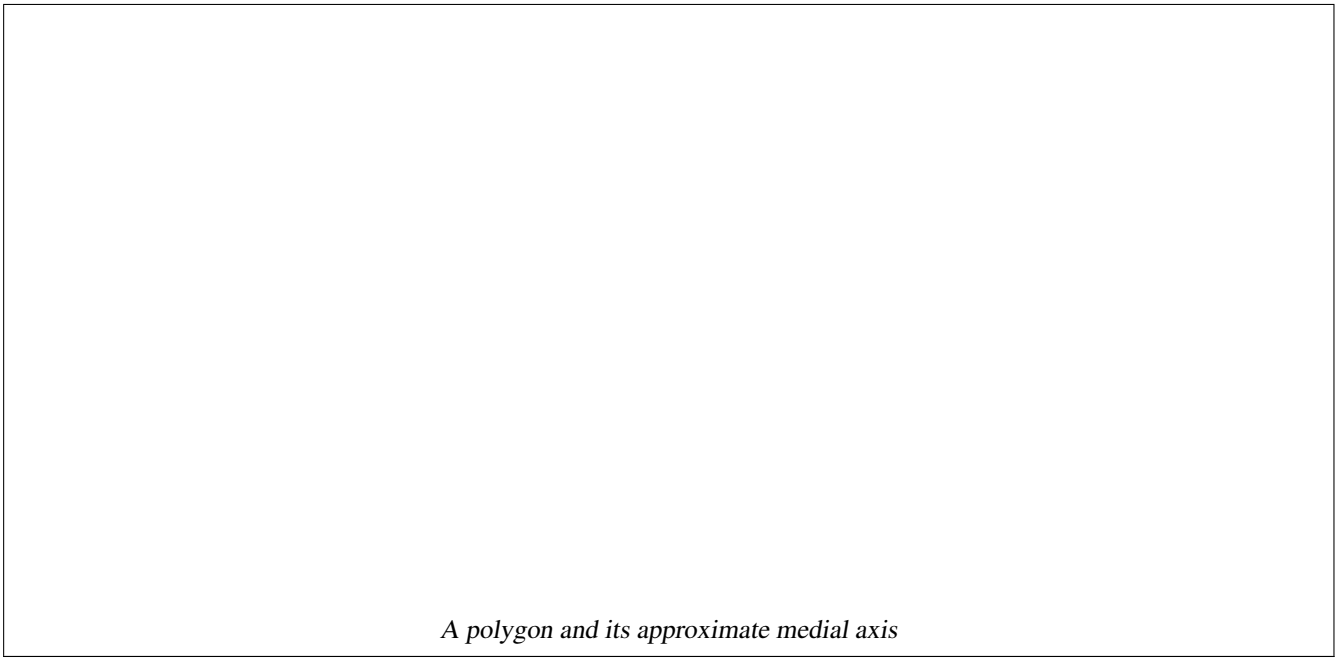
This function supports 3d and will not drop the z-index.

This function supports Polyhedral surfaces.

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

Examples

```
SELECT CG_ApproximateMedialAxis(ST_GeomFromText('POLYGON (( 190 190, 10 190, 10 10, 190 10, ↩
     190 20, 160 30, 60 30, 60 130, 190 140, 190 190 ))'));
```



A polygon and its approximate medial axis

See Also

CG_StraightSkeleton

8.3.21 ST_ConstrainedDelaunayTriangles

ST_ConstrainedDelaunayTriangles — Return a constrained Delaunay triangulation around the given input geometry.

Synopsis

geometry **ST_ConstrainedDelaunayTriangles**(geometry g1);

Description

Warning
ST_ConstrainedDelaunayTriangles is deprecated as of 3.5.0. Use CG_ConstrainedDelaunayTriangles instead.

Return a Constrained Delaunay triangulation around the [-108ices of the input geometry. Output is a TIN.

This method needs SFCGAL backend.

Availability: 3.0.0

This function supports 3d and will not drop the z-inde [-1x.

8.3.22 CG_ConstrainedDelaunayTriangles

CG_ConstrainedDelaunayTriangles — Return a constrained Delaunay triangulation around the given input geometry.

Synopsis

geometry **CG_ConstrainedDelaunayTriangles**(geometry g1);

Description

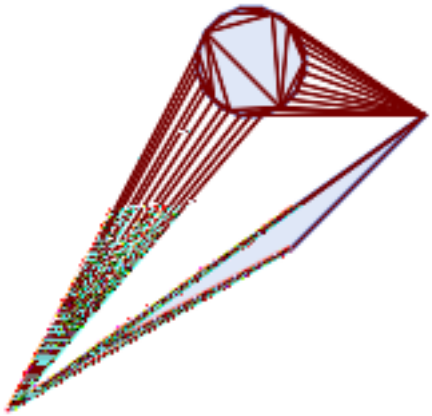
Return a Constrained Delaunay triangulation around the vertices of the input geometry. Output is a TIN.

This method needs SFCGAL backend.

Availability: 3.0.0

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

Examples



CG_ConstrainedDelaunayTriangles of 2 polygons

```
select CG_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
ST_Buffer('POINT(110 170)'::geometry, 20)
)
) ↩
;
```

See Also

ST_DelaunayTriangles, ST_TriangulatePolygon, CG_Tesselate, ST_ConcaveHull, ST_Dump

8.3.23 ST_Extrude

ST_Extrude — Extrude a surface to a related volume

Synopsis

geometry **ST_Extrude**(geometry geom, float x, float y, float z);

Description

Warning
ST_Extrude is deprecated as of 3.5.0. Use CG_Extrude instead.

Availability: 2.1.0

This method needs SFCGAL backend.

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

This function supports Polyhedral surfaces.

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

8.3.24 CG_Extrude

CG_Extrude — Extrude a surface to a related volume

Synopsis

geometry **CG_Extrude**(geometry geom, float x, float y, float z);

Description

Availability: 3.5.0

This method needs SFCGAL backend.

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

This function supports Polyhedral surfaces.

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

Examples

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

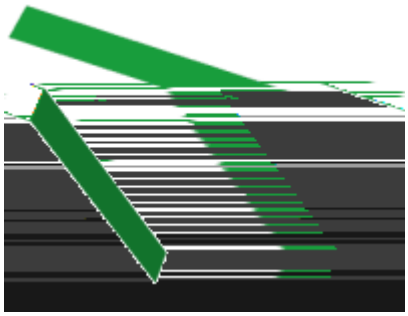
```
CG_Extrude(ST_Buffer(ST_GeomFromText(' ↩
    POINT(100 90)'),
                                    50, ' ↩
    quad_segs=2'),0,0,30);
```

Hexagon extruded 30 units along Z produces a PolyhedralSurfaceZ

```
SELECT ST_GeomFromText('LINESTRING(50 50, ↩
     100 90, 95 150)')
```

Original linestring

```
SELECT CG_Extrude(
                                    ↩
    ST_GeomFromText('LINESTRING(50 50, 100 90, 95 15
```



LineString Extruded along Z produces a PolyhedralSurfaceZ

See Also

ST_AsX3D, CG_ExtrudeStraightSkeleton

8.3.25 CG_ExtrudeStraightSkeleton

CG_ExtrudeStraightSkeleton — Straight Skeleton Extrusion

Synopsis

geometry **CG_ExtrudeStraightSkeleton**(geometry geom, float roof_height, float body_height = 0);

Description

Computes an extrusion with a maximal height of the polygon geometry.

Note

Perhaps the first (historically) use-case of straight skeletons: given a polygonal roof, the straight skeleton directly gives the layout of each tent. If each skeleton edge is lifted from the plane a height equal to its offset distance, the resulting roof is "correct" in that water will always fall down to the contour edges (the roof's border), regardless of where it falls on the roof. The function computes this extrusion aka "roof" on a polygon. If the argument body_height > 0, so the polygon is extruded like with CG_Extrude(polygon, 0, 0, body_height). The result is an union of these polyhedralsurfaces.

Availability: 3.5.0 - requires SFCGAL >= 1.5.0.

Requires SFCGAL >= 1.5.0

This method needs SFCGAL backend.

Examples

```
SELECT ST_AsText(CG_ExtrudeStraightSkeleton('POLYGON (( 0 0, 5 0, 5 5, 4 5, 4 4, 0 4, 0 0 ) ↩
    , (1 1, 1 2,2 2, 2 1, 1 1))', 3.0, 2.0));
```

```
POLYHEDRALSURFACE Z (((0 0 0,0 4 0,4 4 0,4 5 0,5 5 0,5 0 0,0 0 0),(1 1 0,2 1 0,2 2 0,1 2  ↩
    0,1 1 0)),((0 0 0,0 0 2,0 4 2,0 4 0,0 0 0)),((0 4 0,0 4 2,4 4 2,4 4 0,0 4 0)),((4 4 0,4  ↩
    4 2,4 5 2,4 5 0,4 4 0)),((4 5 0,4 5 2,5 5 2,5 5 0,4 5 0)),((5 5 0,5 5 2,5 0 2,5 0 0,5 5  ↩
    0)),((5 0 0,5 0 2,0 0 2,0 0 0,5 0 0)),((1 1 0,1 1 2,2 1 2,2 1 0,1 1 0)),((2 1 0,2 1 2,2  ↩
    2 2,2 2 0,2 1 0)),((2 2 0,2 2 2,1 2 2,1 2 0,2 2 0)),((1 2 0,1 2 2,1 1 2,1 1 0,1 2 0)) ↩
    ,((4 5 2,5 5 2,4 4 2,4 5 2)),((2 1 2,5 0 2,0 0 2,2 1 2)),((5 5 2,5 0 2,4 4 2,5 5 2)),((2 ↩
     1 2,0 0 2,1 1 2,2 1 2)),((1 2 2,1 1 2,0 0 2,1 2 2)),((0 4 2,2 2 2,1 2 2,0 4 2)),((0 4  ↩
    2,1 2 2,0 0 2,0 4 2)),((4 4 2,5 0 2,2 2 2,4 4 2)),((4 4 2,2 2 2,0 4 2,4 4 2)),((2 2 2,5  ↩
    0 2,2 1 2,2 2 2)),((0.5 2.5 2.5,0 0 2,0.5 0.5 2.5,0.5 2.5 2.5)),((1 3 3,0 4 2,0.5 2.5  ↩
    2.5,1 3 3)),((0.5 2.5 2.5,0 4 2,0 0 2,0.5 2.5 2.5)),((2.5 0.5 2.5,5 0 2,3.5 1.5 3.5,2.5  ↩
    0.5 2.5)),((0 0 2,5 0 2,2.5 0.5 2.5,0 0 2)),((0.5 0.5 2.5,0 0 2,2.5 0.5 2.5,0.5 0.5 2.5) ↩
    ),((4.5 3.5 2.5,5 5 2,4.5 4.5 2.5,4.5 3.5 2.5)),((3.5 2.5 3.5,3.5 1.5 3.5,4.5 3.5  ↩
    2.5,3.5 2.5 3.5)),((4.5 3.5 2.5,5 0 2,5 5 2,4.5 3.5 2.5)),((3.5 1.5 3.5,5 0 2,4.5 3.5  ↩
    2.5,3.5 1.5 3.5)),((5 5 2,4 5 2,4.5 4.5 2.5,5 5 2)),((4.5 4.5 2.5,4 4 2,4.5 3.5 2.5,4.5  ↩
    4.5 2.5)),((4.5 4.5 2.5,4 5 2,4 4 2,4.5 4.5 2.5)),((3 3 3,0 4 2,1 3 3,3 3 3)),((3.5 2.5  ↩
    3.5,4.5 3.5 2.5,3 3 3,3.5 2.5 3.5)),((3 3 3,4 4 2,0 4 2,3 3 3)),((4.5 3.5 2.5,4 4 2,3 3  ↩
    3,4.5 3.5 2.5)),((2 1 2,1 1 2,0.5 0.5 2.5,2 1 2)),((2.5 0.5 2.5,2 1 2,0.5 0.5 2.5,2.5  ↩
    0.5 2.5)),((1 1 2,1 2 2,0.5 2.5 2.5,1 1 2)),((0.5 0.5 2.5,1 1 2,0.5 2.5 2.5,0.5 0.5 2.5) ↩
    ),((1 3 3,2 2 2,3 3 3,1 3 3)),((0.5 2.5 2.5,1 2 2,1 3 3,0.5 2.5 2.5)),((1 3 3,1 2 2,2 2  ↩
    2,1 3 3)),((2 2 2,2 1 2,2.5 0.5 2.5,2 2 2)),((3.5 2.5 3.5,3 3 3,3.5 1.5 3.5,3.5 2.5 3.5) ↩
    ),((3.5 1.5 3.5,2 2 2,2.5 0.5 2.5,3.5 1.5 3.5)),((3 3 3,2 2 2,3.5 1.5 3.5,3 3 3)))
```

See Also

ST_Extrude, CG_StraightSkeleton

8.3.26 CG_GreeneApproxConvexPartition

CG_GreeneApproxConvexPartition — Computes approximal convex partition of the polygon geometry

Synopsis

geometry **CG_GreeneApproxConvexPartition**(geometry geom);

Description

Computes approximal monotone convex partition of the polygon geometry.

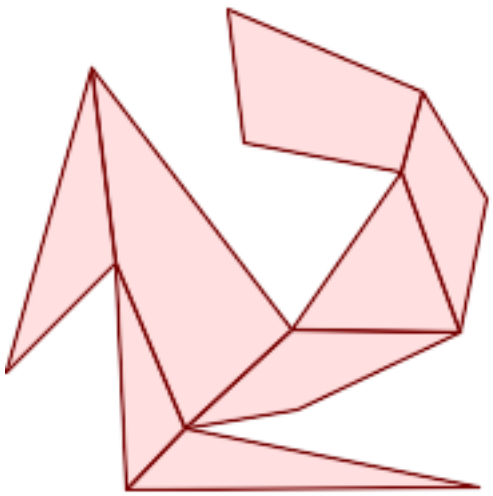
Note
A partition of a polygon P is a set of polygons such that the interiors of the polygons do not intersect and the union of the polygons is equal to the interior of the original polygon P. CG_ApproxConvexPartition and CG_GreeneApproxConvexPartition functions produce approximately optimal convex partitions. Both these functions produce convex decompositions by first decomposing the polygon into simpler polygons; CG_ApproxConvexPartition uses a triangulation and CG_GreeneApproxConvexPartition a monotone partition. These two functions both guarantee that they will produce no more than four times the optimal number of convex pieces but they differ in their runtime complexities. Though the triangulation-based approximation algorithm often results in fewer convex pieces, this is not always the case.

Availability: 3.5.0 - requires SFCGAL >= 1.5.0.

Requires SFCGAL >= 1.5.0

This method needs SFCGAL backend.

Examples



Greene Approximal Convex Partition (same example As CG_YMonotonePartition, CG_ApproxConvexPartition and CG_OptimalConvexPartition)

```
SELECT ST_AsText(CG_GreeneApproxConvexPartition('POLYGON((156 150,83 181,89 131,148 120,107 ↩
    61,32 159,0 45,41 86,45 1,177 2,67 24,109 31,170 60,180 110,156 150))'::geometry));
```

```
GEOMETRYCOLLECTION(POLYGON((32 159,0 45,41 86,32 159)),POLYGON((45 1,177 2,67 24,45 1)), ↩
    POLYGON((67 24,109 31,170 60,107 61,67 24)),POLYGON((41 86,45 1,67 24,41 86)),POLYGON ↩
    ((107 61,32 159,41 86,67 24,107 61)),POLYGON((148 120,107 61,170 60,148 120)),POLYGON ↩
    ((148 120,170 60,180 110,156 150,148 120)),POLYGON((156 150,83 181,89 131,148 120,156  ↩
    150)))
```

See Also

CG_Y7.146 cm0 0 2.989 10.959 re fQ0 g 0 G0.95 g 0.95 G0 g 0 G0ry870 g 0 G15 cm0 0 2.98Ptition56v 2.9e 2.9x0 2.98Ptition56v 2.9

Description

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.

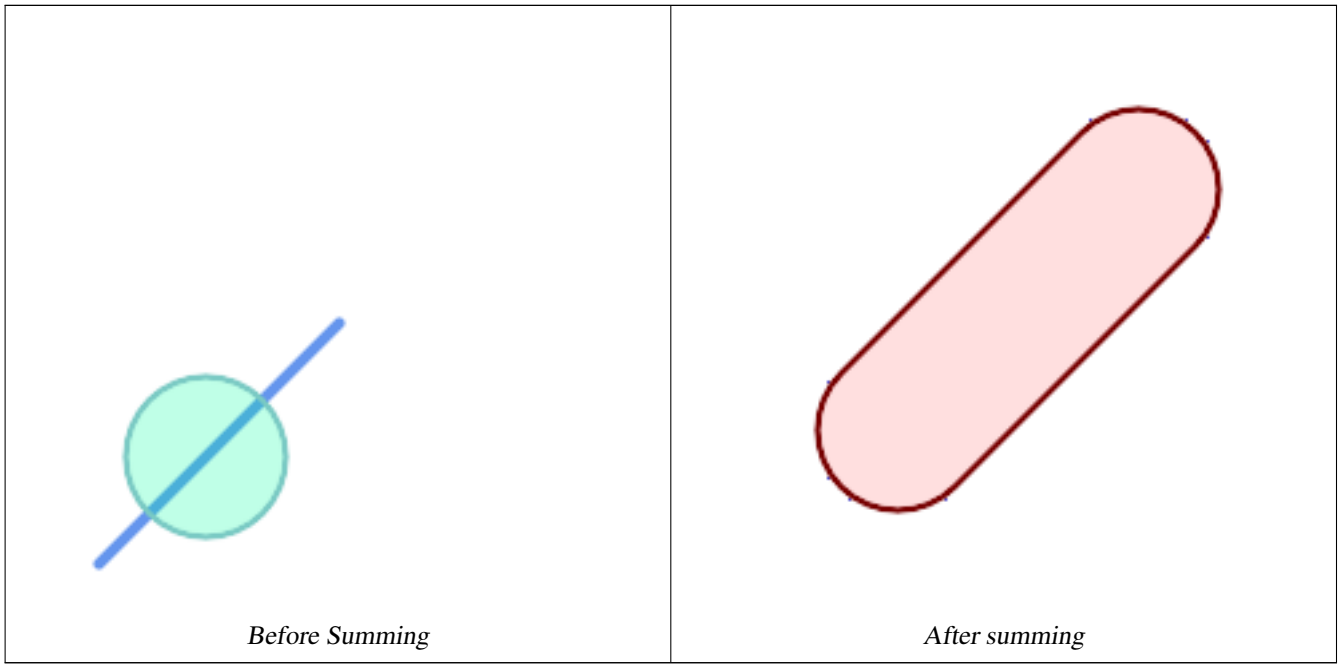
Implementation utilizes CGAL 2D Minkowskisum.

Availability: 3.5.0

This method needs SFCGAL backend.

Examples

Minkowski Sum of Linestring and circle polygon where Linestring cuts thru the circle

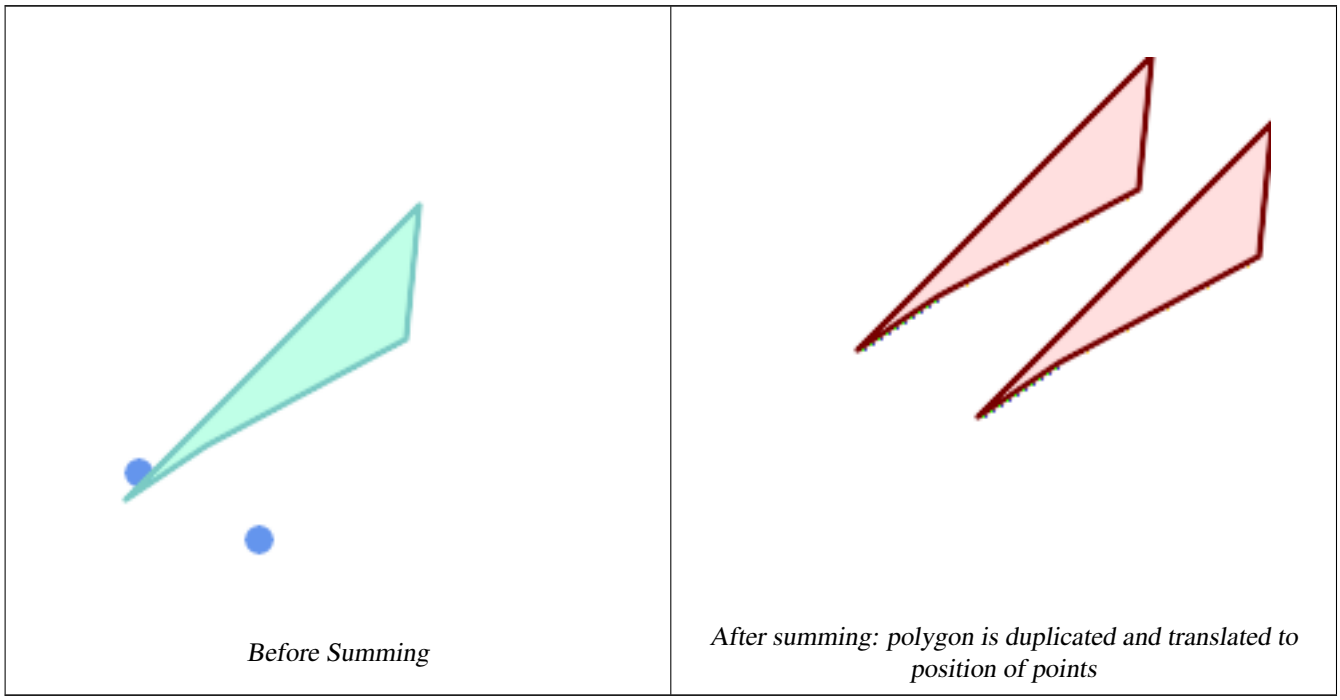


Before Summing | *After summing*

```
SELECT CG_MinkowskiSum(line, circle))
FROM (SELECT
ST_MakeLine(ST_Point(10, 10),ST_Point(100, 100)) As line,
ST_Buffer(ST_GeomFromText('POINT(50 50)'), 30) As circle) As foo;

-- wkt --
MULTIPOLYGON(((30 59.9999999999999,30.5764415879031  ↩
    54.1472903395161,32.2836140246614 48.5194970290472,35.0559116309237  ↩
    43.3328930094119,38.7867965644036 38.7867965644035,43.332893009412  ↩
    35.0559116309236,48.5194970290474 32.2836140246614,54.1472903395162  ↩
    30.5764415879031,60.0000000000001 30,65.8527096604839  ↩
    30.5764415879031,71.4805029709527 32.2836140246614,76.6671069905881  ↩
    35.0559116309237,81.2132034355964 38.7867965644036,171.213203435596  ↩
    128.786796564404,174.944088369076 133.332893009412,177.716385975339  ↩
    138.519497029047,179.423558412097 144.147290339516,180 150,179.423558412097  ↩
    155.852709660484,177.716385975339 161.480502970953,174.944088369076  ↩
    166.667106990588,171.213203435596 171.213203435596,166.667106990588  ↩
    174.944088369076,
```

```
            161.480502970953 177.716385975339,155.852709660484 179.423558412097,150  ↩
                180,144.147290339516 179.423558412097,138.519497029047  ↩
                177.716385975339,133.332893009412 174.944088369076,128.786796564403  ↩
                171.213203435596,38.7867965644035 81.2132034355963,35.0559116309236  ↩
                76.667106990588,32.2836140246614 71.4805029709526,30.5764415879031  ↩
                65.8527096604838,30 59.9999999999999)))
```

Minkowski Sum of a polygon and multipoint



Before Summing

After summing: polygon is duplicated and translated to position of points

```
SELECT CG_MinkowskiSum(mp, poly)
        FROM (SELECT 'MULTIPOINT(25 50,70 25)'::gQ5.8 0 G0.95 g 0.95/:
```

Description

Warning
ST_OptimalAlphaShape is deprecated as of 3.5.0. Use CG_OptimalAlphaShape instead.

Computes the "optimal" alpha-shape of the points in a geometry. The alpha-shape is computed using a value of α chosen so that:

1. the number of polygon elements is equal to or smaller than `nb_components` (which defaults to 1)
2. all input points are contained in the shape

The result will not contain holes unless the optional `allow_holes` argument is specified as true.

Availability: 3.3.0 - requires SFCGAL >= 1.4.1.

This method needs SFCGAL backend.

8.3.30 CG_OptimalAlphaShape

CG_OptimalAlphaShape — Computes an Alpha-shape enclosing a geometry using an "optimal" alpha value.

Synopsis

geometry **CG_OptimalAlphaShape**(geometry geom, boolean allow_holes = false, integer nb_components = 1);

Description

Computes the "optimal" alpha-shape of the points in a geometry. The alpha-shape is computed using a value of α chosen so that:

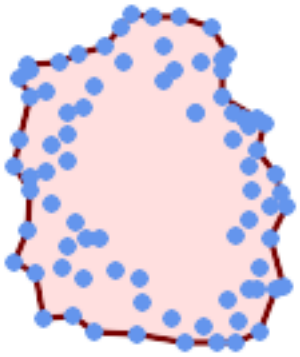
1. the number of polygon elements is equal to or smaller than `nb_components` (which defaults to 1)
2. all input points are contained in the shape

The result will not contain holes unless the optional `allow_holes` argument is specified as true.

Availability: 3.5.0 - requires SFCGAL >= 1.4.1.

This method needs SFCGAL backend.

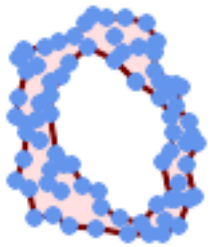
Examples



Optimal alpha-shape of a MultiPoint (same example as CG_AlphaShape)

```
SELECT ST_AsText(CG_OptimalAlphaShape('MULTIPOINT((63 84),(76 88),(68 73),(53 18),(91 50) ↩
    ,(81 70),
                (88 29),(24 82),(32 51),(37 23),(27 54),(84 19),(75 87),(44 42),(77 67),(90 ↩
                    30),(36 61),(32 65),
                (81 47),(88 58),(68 73),(49 95),(81 60),(87 50),
                (78 16),(79 21),(30 22),(78 43),(26 85),(48 34),(35 35),(36 40),(31 79),(83 ↩
                    29),(27 84),(52 98),(72 95),(85 71),
                (75 84),(75 77),(81 29),(77 73),(41 42),(83 72),(23 36),(89 53),(27 57),(57 ↩
                    97),(27 77),(39 88),(60 81),
                (80 72),(54 32),(55 26),(62 22),(70 20),(76 27),(84 35),(87 42),(82 54),(83 ↩
                    64),(69 86),(60 90),(50 86),(43 80),(36 73),
                (36 68),(40 75),(24 67),(23 60),(26 44),(28 33),(40 32),(43 19),(65 16),(73 ↩
                    16),(38 46),(31 59),(34 86),(45 90),(64 97))'::geometry));
```

```
POLYGON((89 53,91 50,87 42,90 30,88 29,84 19,78 16,73 16,65 16,53 18,43 19,37 23,30 22,28  ↩
    33,23 36,
            26 44,27 54,23 60,24 67,27 77,24 82,26 85,34 86,39 88,45 90,49 95,52 98,57  ↩
                97,64 97,72 95,76 88,75 84,75 77,83 72,85 71,83 64,88 58,89 53))
```



Optimal alpha-shape of a MultiPoint, allowing holes (same example as CG_AlphaShape)

```
SELECT ST_AsText(CG_OptimalAlphaShape('MULTIPOINT((63 84),(76 88),(68 73),(53 18),(91 50) ↩
    ,(81 70),(88 29),(24 82),(32 51),(37 23),(27 54),(84 19),(75 87),(44 42),(77 67),(90 30) ↩
    ,(36 61),(32 65),(81 47),(88 58),(68 73),(49 95),(81 60),(87 50),
        (78 16),(79 21),(30 22),(78 43),(26 85),(48 34),(35 35),(36 40),(31 79),(83 29),(27 ↩
            84),(52 98),(72 95),(85 71),
        (75 84),(75 77),(81 29),(77 73),(41 42),(83 72),(23 36),(89 53),(27 57),(57 97),(27 ↩
            77),(39 88),(60 81),
        (80 72),(54 32),(55 26),(62 22),(70 20),(76 27),(84 35),(87 42),(82 54),(83 64),(69 ↩
            86),(60 90),(50 86),(43 80),(36 73),
        (36 68),(40 75),(24 67),(23 60),(26 44),(28 33),(40 32),(43 19),(65 16),(73 16),(38 ↩
            46),(31 59),(34 86),(45 90),(64 97))'::geometry, allow_holes => true));
```

```
POLYGON((89 53,91 50,87 42,90 30,88 29,84 19,78 16,73 16,65 16,53 18,43 19,37 23,30 22,28  ↩
    33,23 36,26 44,27 54,23 60,24 67,27 77,24 82,26 85,34 86,39 88,45 90,49 95,52 98,57  ↩
    97,64 97,72 95,76 88,75 84,75 77,83 72,85 71,83 64,88 58,89 53),(36 61,36 68,40 75,43  ↩
    80,50 86,60 81,68 73,77 67,81 60,82 54,81 47,78 43,81 29,76 27,70 20,62 22,55 26,54  ↩
    32,48 34,44 42,38 46,36 61))
```

See Also

ST_ConcaveHull, CG_AlphaShape

8.3.31 CG_OptimalConvexPartition

CG_OptimalConvexPartition — Computes an optimal convex partition of the polygon geometry

Synopsis

geometry **CG_OptimalConvexPartition**(geometry geom);

Description

Computes an optimal convex partition of the polygon geometry.

Synopsis

geometry **ST_StraightSkeleton**(geometry geom);

Description

Warning
ST_StraightSkeleton is deprecated as of 3.5.0. Use CG_StraightSkeleton instead.

Availability: 2.1.0

This method needs SFCGAL backend.

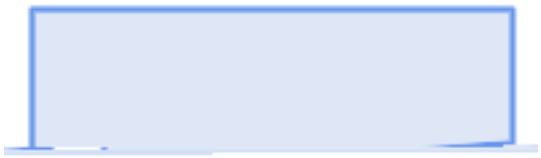
This function supports 3d and will not drop the z-index.

This function supports Polyhedral surfaces.

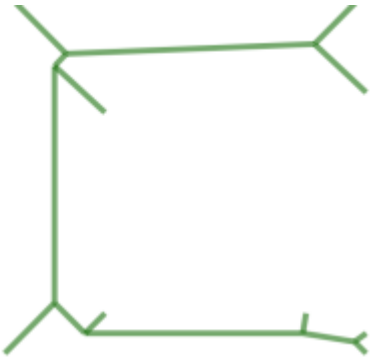
This function supports Triangles and Triangulated Irregular Network Surfaces (TIN).

Examples

```
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 ))'));
```



Original polygon



Straight Skeleton of polygon

See Also

CG_ExtrudeStraightSkeleton

8.3.34 ST_Tesselate

ST_Tesselate — Perform surface Tessellation of a polygon or polyhedralsurface and returns as a TIN or collection of TINS

Synopsis

geometry **ST_Tesselate**(geometry geom);

Description

Warning
ST_Tesselate is deprecated as of 3.5.0. Use CG_Tesselate instead.

Takes as input a surface such a MULTI(POLYGON) or POLYHEDRALSURFACE and returns a TIN representation via the process of tessellation using triangles.

Note
ST_TriangulatePolygon does similar to this function except that it returns a geometry collection of polygons instead of a TIN and also only works with 2D geometries.

Availability: 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).

8.3.35 CG_Tesselate

CG_Tesselate — Perform surface Tessellation of a polygon or polyhedralsurface and returns as a TIN or collection of TINS

Synopsis

geometry **CG_Tesselate**(geometry geom);

Description

Takes as input a surface such a MULTI(POLYGON) or POLYHEDRALSURFACE and returns a TIN representation via the process of tessellation using triangles.

Note
ST_TriangulatePolygon does similar to this function except that it returns a geometry collection of polygons instead of a TIN and also only works with 2D geometries.

Availability: 3.5.0

This method needs SFCGAL backend.

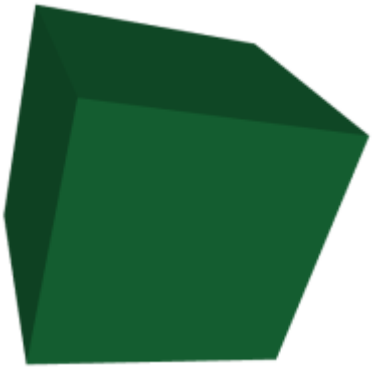
This function supports 3d and will not drop the z-index.

This function supports Polyhedral surfaces.

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

Examples

```
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
                                             ((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
```



Original Cube

```
SELECT CG_Tesselate(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
                                             ((1 1 0, 1 1 1, 1 0 1, 1 0 0, 1 1 0)),
                                             ((0 1 0, 0 1 1, 1 1 1, 1 1 0, 0 1 0)), ((0 0 1, 1 0 1, 1
```

ST_AsText output:

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

Tessellated Cube with triangles colored

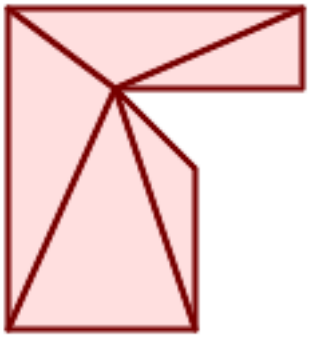
```
SELECT 'POLYGON (( 10 190, 10 70, 80 70,  ↩
    80 130, 50 160, 120 160, 120 190, 10 190 ))'::geometry;
```

Original polygon

```
SELECT
                        CG_Tesselate(' ↩
    POLYGON (( 10 190, 10 70, 80 70, 80 130, 50 160,
    ;
```

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)))
```



Tessellated Polygon

See Also

CG_ConstrainedDelaunayTriangles, ST_DelaunayTriangles, ST_TriangulatePolygon

8.3.36 CG_Triangulate

CG_Triangulate — Triangulates a polygonal geometry

Synopsis

geometry **CG_Triangulate**(geometry geom);

Description

Triangulates a polygonal geometry.

Performed by the SFCGAL module

Note

NOTE: this function returns a geometry representing the triangulated result.

Availability: 3.5.0

This method needs SFCGAL backend.

Geometry Examples

```
SELECT CG_Triangulate('POLYGON((0.0 0.0,1.0 0.0,1.0 1.0,0.0 1.0,0.0 0.0),(0.2 0.2,0.2 ↩
    0.8,0.8 0.8,0.8 0.2,0.2 0.2))');
                cg_triangulate
                ----------------
                TIN(((0.8 0.2,0.2 0.2,1 0,0.8 0.2)),((0.2 0.2,0 0,1 0,0.2 0.2)),((1 1,0.8 ↩
                    0.8,0.8 0.2,1 1)),((0 1,0 0,0.2 0.2,0 1)),((0 1,0.2 0.8,1 1,0 1)),((0 ↩
                    1,0.2 0.2,0.2 0.8,0 1)),((0.2 0.8,0.8 0.8,1 1,0.2 0.8)),((0.2 0.8,0.2 ↩
                    0.2,0.8 0.2,0.2 0.8)),((1 1,0.8 0.2,1 0,1 1)),((0.8 0.8,0.2 0.8,0.8 ↩
                    0.2,0.8 0.8)))

                (1 row)
```

See Also

CG_ConstrainedDelaunayTriangles, ST_DelaunayTriangles, ST_TriangulatePolygon

8.3.37 CG_Visibility

CG_Visibility — Compute a visibility polygon from a point or a segment in a polygon geometry

Synopsis

geometry **CG_Visibility**(geometry polygon, geometry point);
geometry **CG_Visibility**(geometry polygon, geometry pointA, geometry pointB);

Description

Availability: 3.5.0 - requires SFCGAL >= 1.5.0.

Requires SFCGAL >= 1.5.0

This method needs SFCGAL backend.

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

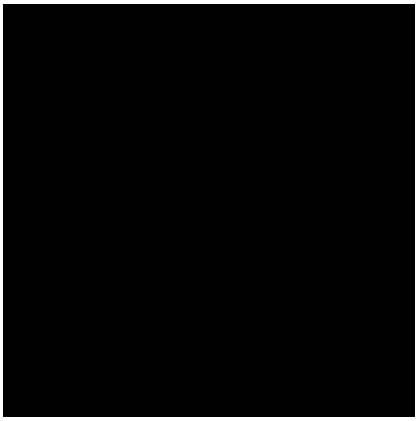
This function supports Polyhedral surfaces.

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

Examples

```
SELECT CG_Visibility('POLYGON((23.5 23.5,23.5 173.5,173.5 173.5,173.5 23.5,23.5 23.5),(108 ↩
    98,108 36,156 37,155 99,108 98),(107 157.5,107 106.5,135 107.5,133 127.5,143.5 ↩
    127.5,143.5 108.5,153.5 109.5,151.5 166,107 157.5),(41 95.5,41 35,100.5 36,98.5 68,78.5 ↩
    68,77.5 96.5,41 95.5),(39 150,40 104,97.5 106.5,95.5 152,39 150))'::geometry, 'POINT(91 ↩
    87)'::geometry);
```

```
SELECT CG_Visibility('POLYGON((23.5 23.5,23.5 173.5,173.5 173.5,173.5 23.5,23.5 23.5),(108 ↩
    98,108 36,156 37,155 99,108 98),(107 157.5,107 106.5,135 107.5,133 127.5,143.5 ↩
    127.5,143.5 108.5,153.5 109.5,151.5 166,107 157.5),(41 95.5,41 35,100.5 36,98.5 68,78.5 ↩
    68,77.5 96.5,41 95.5),(39 150,40 104,97.5 106.5,95.5 152,39 150))'::geometry,'POINT(78.5 ↩
     68)'::geometry, 'POINT(98.5 68)'::geometry);
```



Original polygon

Visibility from the point

Visibility from the segment

8.3.38 CG_YMonotonePartition

CG_YMonotonePartition — Computes y-monotone partition of the polygon geometry

Synopsis

geometry **CG_YMonotonePartition**(geometry geom);

Description

Computes y-monotone partition of the polygon geometry.

Note

A partition of a polygon P is a set of polygons such that the interiors of the polygons do not intersect and the union of the polygons is equal to the interior of the original polygon P. A y-monotone polygon is a polygon whose vertices v1,. . . ,vn can be divided into two chains v1,. . . ,vk and vk,. . . ,vn,v1, such that any horizontal line intersects either chain at most once. This algorithm does not guarantee a bound on the number of polygons produced with respect to the optimal number.

Availability: 3.5.0 - requires SFCGAL >= 1.5.0.

Requires SFCGAL >= 1.5.0

This method needs SFCGAL backend.

Examples

9.1.2 TopoGeometry

TopoGeometry — A composite type representing a topologically de ned geometry.

Description

A composite type that refers to a topology geometry in a speci c topology layer, having a speci c type and a speci c id. Th elements of a TopoGeometry are the properties: topology_id, layer_id, id integer, type integer.

1. topology_id is an integer: Refers to a topology de ned in the topology.topology table which de nes the topology schema and srid.
2. layer_id is an integer: The layer_id in the layers table that the TopoGeometry belongs to. The combination of topo ogy_id, layer_id provides a unique reference in the topology.layers table.
3. id is an integer: The id is the autogenerated sequence number that uniquely de nes the topogeometry in the respec topology layer.
4. type integer between 1 - 4 that de nes the geometry type: 1:[multi]point, 2:[multi]line, 3:[multi]poly, 4:collection

Casting Behavior

This section lists the automatic as well as explicit casts allowed for this data type

Cast To	Behavior
geometry	automatic

See Also

CreateTopoGeom

9.1.3 validatetopology_returntype

validatetopology_returntype — A composite type that consists of an error message and id1 and id2 to denote location of e This is the return type for ValidateTopology .

Description

A composite type that consists of an error message and two integers. The ValidateTopology function returns a set of thes denote validation errors and the id1 and id2 to denote the ids of the topology objects involved in the error.

1. error is varchar: Denotes type of error.

   Current error descriptors are: coincident nodes, edge crosses node, edge not simple, edge end node geometry mism edge start node geometry mismatch, face overlaps face,face within face,
2. id1 is an integer: Denotes identi er of edge / face / nodes in error.
3. id2 is an integer: For errors that involve 2 objects denotes the secondary edge / or node

See Also

ValidateTopology

Description

Drops the topogeometry column from the table named `table_name` in schema `schema_name` and unregisters the columns from topology.layer table. Returns summary of drop status. NOTE: it first sets all values to NULL before dropping to bypass referential integrity checks.

Availability: 1.1

Examples

```
SELECT topology.DropTopoGeometryColumn('ma_topo', 'parcel_topo', 'topo');
```

See Also

AddTopoGeometryColumn

9.3.6 Populate_Topology_Layer

Populate_Topology_Layer — Adds missing entries to topology.layer table by reading metadata from topo tables.

Synopsis

setof record **Populate_Topology_Layer**();

Description

Adds missing entries to the `topology.layer` table by inspecting topology constraints on tables. This function is useful for fixing up entries in topology catalog after restores of schemas with topo data.

It returns the list of entries created. Returned columns are `schema_name`, `table_name`, `feature_column`.

Availability: 2.3.0

Examples

```
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;
```

9.3.8 ValidateTopology

ValidateTopology — Returns a set of validatetopology_returntype objects detailing issues with topology.

Synopsis

setof validatetopology_returntype **ValidateTopology**(varchar toponame, geometry bbox);

Description

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 mismatch	Identifier of the edge.	Identifier of the indicated start node.	The geometry of the node indicated as the starting node for an edge does not match the first point of the edge geometry. See ST_StartPoint.
edge end node geometry mismatch	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.

9.3.9 ValidateTopologyRelation

ValidateTopologyRelation — Returns info about invalid topology relation records

Synopsis

setof record **ValidateTopologyRelation**(varchar toponame);

Description

Returns a set records giving information about invalidities in the relation table of the topology.

Availability: 3.2.0

See Also

ValidateTopology

9.3.10 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);

Description

Takes a topology identifier or the identifier of a topology-related object and returns a topology.topology record.

Availability: 3.2.0

Examples

```
SELECT name(findTopology('features.land_parcels', 'feature'));
   name
-----------
 city_data
(1 row)
```

See Also

FindLayer

9.3.11 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);

Description

Takes a layer identifier or the identifier of a topology-related object and returns a topology.layer record.

Availability: 3.2.0

Examples

```
SELECT layer_id(findLayer('features.land_parcels', 'feature'));
 layer_id
----------
        1
(1 row)
```

See Also

FindTopology

9.4 Topology Statistics Management

Adding elements to a topology triggers many database queries for finding existing edges that will be split, adding nodes and updating edges that will node with the new linework. For this reason it is useful that statistics about the data in the topology tables are up-to-date.

PostGIS Topology population and editing functions do not automatically update the statistics because a updating stats after each and every change in a topology would be overkill, so it is the caller's duty to take care of that.

Note
That the statistics updated by autovacuum will NOT be visible to transactions which started before autovacuum process completed, so long-running transactions will need to run ANALYZE themselves, to use updated statistics.

9.5 Topology Constructors

9.5.1 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);

Description

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

Examples

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
```

See Also

Section 4.5, ST_InitTopoGeo, Topology_Load_Tiger

9.5.2 CopyTopology

CopyTopology — Makes a copy of a topology (nodes, edges, faces, layers and TopoGeometries) into a new schema

Synopsis

integer **CopyTopology**(varchar existing_topology_name, varchar new_name);

Description

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.

Availability: 2.0.0

Examples

Make a backup of a topology called ma_topo.

```
SELECT topology.CopyTopology('ma_topo', 'ma_topo_backup');
```

See Also

Section 4.5, CreateTopology, RenameTopology

9.5.3 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);

Description

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

Examples

```
SELECT topology.ST_InitTopoGeo('topo_schema_to_create') AS topocreation;
                      astopocreation
---------------------------------------------------------------
 Topology-Geometry 'topo_schema_to_create' (id:7) created.
```

See Also

CreateTopology

9.5.4 ST_CreateTopoGeo

ST_CreateTopoGeo — Adds a collection of geometries to a given empty topology and returns a message detailing success.

Synopsis

text **ST_CreateTopoGeo**(varchar atopology, geometry acollection);

Description

Adds a collection of geometries to a given empty topology and returns a message detailing success.

Useful for populating an empty topology.

Availability: 2.0

This method implements the SQL/MM specification. SQL-MM: Topo-Geo and Topo-Net 3: Routine Details -- X.3.18

Synopsis

SETOF integer **TopoGeo_AddLineString**(varchar atopology, geometry aline, float8 tolerance);

Description

Adds a linestring to an existing topology and returns a set of signed edge identifiers forming it up (negative identifies mean the edge goes in the opposite direction of the input linestring). The given line will snap to existing nodes or edges within given tolerance. Existing edges and faces may be split by the line. New nodes and faces may be added.

Note
Updating statistics about topologies being loaded via this function is up to caller, see maintaining statistics during topology editing and population.

Availability: 2.0.0

Enhanced: 3.2.0 added support for returning signed identifier.

See Also

TopoGeo_AddPoint, TopoGeo_AddPolygon, TopoGeo_LoadGeometry, AddEdge, CreateTopology

9.5.7 TopoGeo_AddPolygon

TopoGeo_AddPolygon — Adds a polygon to an existing topology using a tolerance and possibly splitting existing edges/faces. Returns face identifiers.

Synopsis

SETOF integer **TopoGeo_AddPolygon**(varchar atopology, geometry apoly, float8 tolerance);

Description

Adds a polygon to an existing topology and returns a set of face identifiers forming it up. The boundary of the given polygon will snap to existing nodes or edges within given tolerance. Existing edges and faces may be split by the boundary of the new polygon.

Note
Updating statistics about topologies being loaded via this function is up to caller, see maintaining statistics during topology editing and population.

Availability: 2.0.0

See Also

TopoGeo_AddPoint, TopoGeo_AddLineString, TopoGeo_LoadGeometry, AddFace, CreateTopology

9.5.8 TopoGeo_LoadGeometry

TopoGeo_LoadGeometry — Load a geometry into an existing topology, snapping and splitting as needed.

Synopsis

void **TopoGeo_LoadGeometry**(varchar atopology, geometry ageom, float8 tolerance);

Description

Loads a geometry into an existing topology. The given geometry will snap to existing nodes or edges within given tolerance. Existing edges and faces may be split as a consequence of the load.

Note
Updating statistics about topologies being loaded via this function is up to caller, see maintaining statistics during topology editing and population.

Availability: 3.5.0

See Also

TopoGeo_AddPoint, TopoGeo_AddLineString, TopoGeo_AddPolygon, CreateTopology

9.6 Topology Editors

9.6.1 ST_AddIsoNode

ST_AddIsoNode — Adds an isolated node to a face in a topology and returns the nodeid of the new node. If face is null, the node is still created.

Synopsis

integer **ST_AddIsoNode**(varchar atopology, integer aface, geometry apoint);

Description

Adds an isolated node with point location `apoint` to an existing face with faceid `aface` to a topology `atopology` and returns the nodeid of the new node.

If the spatial reference system (srid) of the point geometry is not the same as the topology, the `apoint` is not a point geometry, the point is null, or the point intersects an existing edge (even at the boundaries) then an exception is thrown. If the point already exists as a node, an exception is thrown.

If `aface` is not null and the `apoint` is not within the face, then an exception is thrown.

Availability: 1.1

This method implements the SQL/MM specification. SQL-MM: Topo-Net Routines: X+1.3.1

Examples

See Also

AddNode, CreateTopology, DropTopology, ST_Intersects

9.6.2 ST_AddIsoEdge

ST_AddIsoEdge — Adds an isolated edge defined by geometry `alinestring` to a topology connecting two existing isolated nodes `anode` and `anothernode` and returns the edge id of the new edge.

Synopsis

integer **ST_AddIsoEdge**(varchar atopology, integer anode, integer anothernode, geometry alinestring);

Description

Adds an isolated edge defined by geometry `alinestring` to a topology connecting two existing isolated nodes `anode` and `anothernode` and returns the edge id of the new edge.

If the spatial reference system (srid) of the `alinestring` geometry is not the same as the topology, any of the input arguments are null, or the nodes are contained in more than one face, or the nodes are start or end nodes of an existing edge, then an exception is thrown.

If the `alinestring` is not within the face of the face the `anode` and `anothernode` belong to, then an exception is thrown.

If the `anode` and `anothernode` are not the start and end points of the `alinestring` then an exception is thrown.

Availability: 1.1

This method implements the SQL/MM specification. SQL-MM: Topo-Geo and Topo-Net 3: Routine Details: X.3.4

Examples

See Also

ST_AddIsoNode, ST_IsSimple, ST_Within

9.6.3 ST_AddEdgeNewFaces

ST_AddEdgeNewFaces — Add a new edge and, if in doing so it splits a face, delete the original face and replace it with two new faces.

Synopsis

integer **ST_AddEdgeNewFaces**(varchar atopology, integer anode, integer anothernode, geometry acurve);

Description

Add a new edge and, if in doing so it splits a face, delete the original face and replace it with two new faces. Returns the id of the newly added edge.

Updates all existing joined edges and relationships accordingly.

If any arguments are null, the given nodes are unknown (must already exist in the `node` table of the topology schema) , the `acurve` is not a `LINESTRING`, the `anode` and `anothernode` are not the start and endpoints of `acurve` then an error is thrown.

If the spatial reference system (srid) of the `acurve` geometry is not the same as the topology an exception is thrown.

Availability: 2.0

This method implements the SQL/MM specification. SQL-MM: Topo-Geo and Topo-Net 3: Routine Details: X.3.12

Examples

See Also

ST_RemEdgeNewFace

ST_AddEdgeModFace

9.6.4 ST_AddEdgeModFace

ST_AddEdgeModFace — Add a new edge and, if in doing so it splits a face, modify the original face and add a new face.

Synopsis

integer **ST_AddEdgeModFace**(varchar atopology, integer anode, integer anothernode, geometry acurve);

Description

Add a new edge and, if doing so splits a face, modify the original face and add a new one.

Note
If possible, the new face will be created on left side of the new edge. This will not be possible if the face on the left side will need to be the Universe face (unbounded).

Returns the id of the newly added edge.

Updates all existing joined edges and relationships accordingly.

If any arguments are null, the given nodes are unknown (must already exist in the `node` table of the topology schema) , the `acurve` is not a `LINESTRING`, the `anode` and `anothernode` are not the start and endpoints of `acurve` then an error is thrown.

If the spatial reference system (srid) of the `acurve` geometry is not the same as the topology an exception is thrown.

Availability: 2.0

This method implements the SQL/MM specification. SQL-MM: Topo-Geo and Topo-Net 3: Routine Details: X.3.13

Description

Removes an edge, and if the removed edge separates two faces deletes one face and modifies the other face to cover the space of

See Also

ST_AddEdgeModFace

ST_RemEdgeModFace

ST_ModEdgeSplit

9.6.8 ST_ModEdgeSplit

ST_ModEdgeSplit — Split an edge by creating a new node along an existing edge, modifying the original edge and adding a new edge.

Synopsis

integer **ST_ModEdgeSplit**(varchar atopology, integer anedge, geometry apoint);

Description

Split an edge by creating a new node along an existing edge, modifying the original edge and adding a new edge. Updates all existing joined edges and relationships accordingly. Returns the identifier of the newly added node.

Availability: 1.1

Changed: 2.0 - In prior versions, this was misnamed ST_ModEdgesSplit

This method implements the SQL/MM specification. SQL-MM: Topo-Geo and Topo-Net 3: Routine Details: X.3.9

Examples

```
-- 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
```

See Also

ST_NewEdgesSplit, ST_ModEdgeHeal, ST_NewEdgeHeal, AddEdge

9.6.9 ST_ModEdgeHeal

ST_ModEdgeHeal — Heals two edges by deleting the node connecting them, modifying the first edge and deleting the second edge. Returns the id of the deleted node.

Synopsis

int **ST_ModEdgeHeal**(varchar atopology, integer anedge, integer anotheredge);

Description

Heals two edges by deleting the node connecting them, modifying the first edge and deleting the second edge. Returns the id of the deleted node. Updates all existing joined edges and relationships accordingly.

Description

Moves an isolated node in a topology from one point to another. If new `apoint` geometry exists as a node an error is thrown.

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.

If the spatial reference system (srid) of the point geometry is not the same as the topology an exception is thrown.

Availability: 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

Examples

```
-- 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
```

See Also

ST_AddIsoNode

9.6.12 ST_NewEdgesSplit

ST_NewEdgesSplit — Split an edge by creating a new node along an existing edge, deleting the original edge and replacing it with two new edges. Returns the id of the new node created that joins the new edges.

Synopsis

integer **ST_NewEdgesSplit**(varchar atopology, integer anedge, geometry apoint);

Description

Split an edge with edge id `anedge` by creating a new node with point location `apoint` along current edge, deleting the original edge and replacing it with two new edges. Returns the id of the new node created that joins the new edges. Updates all existing joined edges and relationships accordingly.

If the spatial reference system (srid) of the point geometry is not the same as the topology, the `apoint` is not a point geometry, the point is null, the point already exists as a node, the edge does not correspond to an existing edge or the point is not within the edge then an exception is thrown.

Availability: 1.1

This method implements the SQL/MM specification. SQL-MM: Topo-Net Routines: X.3.8

Synopsis

text **ST_RemoveIsoEdge**(varchar atopology, integer anedge);

Description

Removes an isolated edge and returns description of action. If the edge is not isolated, then an exception is thrown.

Availability: 1.1

This method implements the SQL/MM specification. SQL-MM: Topo-Geo and Topo-Net 3: Routine Details: X+1.3.3

Examples

```
-- Remove an isolated node with no face  --
SELECT topology.ST_RemoveIsoNode('ma_topo',  7 ) As result;
         result
-------------------------
 Isolated node 7 removed
```

See Also

ST_AddIsoNode

9.7 Topology Accessors

9.7.1 GetEdgeByPoint

GetEdgeByPoint — Finds the edge-id of an edge that intersects a given point.

Synopsis

integer **GetEdgeByPoint**(varchar atopology, geometry apoint, float8 tol1);

Description

Retrieves the id of an edge that intersects a Point.

The function returns an integer (id-edge) given a topology, a POINT and a tolerance. If tolerance = 0 then the point has to intersect the edge.

If `apoint` doesn't intersect an edge, returns 0 (zero).

If use tolerance > 0 and there is more than one edge near the point then an exception is thrown.

Note
If tolerance = 0, the function uses ST_Intersects otherwise uses ST_DWithin.

Performed by the GEOS module.

Availability: 2.0.0

Examples

These examples use edges we created in AddEdge

See Also

GetFaceContainingPoint, AddFace, GetNodeByPoint, GetEdgeByPoint

9.7.3 GetFaceContainingPoint

GetFaceContainingPoint — Finds the face containing a point.

Synopsis

integer **GetFaceContainingPoint**(text atopology, geometry apoint);

Description

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

See Also

ST_GetFaceGeometry

9.7.4 GetNodeByPoint

GetNodeByPoint — Finds the node-id of a node at a point location.

Synopsis

integer **GetNodeByPoint**(varchar atopology, geometry apoint, float8 tol1);

Description

Retrieves the id of a node at a point location.

The function returns an integer (id-node) given a topology, a POINT and a tolerance. If tolerance = 0 means exact intersection, otherwise retrieves the node from an interval.

If `apoint` doesn't intersect a node, returns 0 (zero).

If use tolerance > 0 and there is more than one node near the point then an exception is thrown.

Note
If tolerance = 0, the function uses ST_Intersects otherwise uses ST_DWithin.

Performed by the GEOS module.

Availability: 2.0.0

Description

Returns the spatial reference id of a topology in the topology.topology table given the name of the topology.

Availability: 2.0.0

Examples

```
SELECT topology.GetTopologySRID('ma_topo') As SRID;
 SRID
-------
  4326
```

See Also

CreateTopology, DropTopology, GetTopologyName, GetTopologyID

9.7.7 GetTopologyName

GetTopologyName — Returns the name of a topology (schema) given the id of the topology.

Synopsis

varchar **GetTopologyName**(integer topology_id);

Description

Returns the topology name (schema) of a topology from the topology.topology table given the topology id of the topology.

Availability: 1.1

Examples

```
SELECT topology.GetTopologyName(1) As topo_name;
 topo_name
-----------
 ma_topo
```

See Also

CreateTopology, DropTopology, GetTopologyID, GetTopologySRID

9.7.8 ST_GetFaceEdges

ST_GetFaceEdges — Returns a set of ordered edges that bound `aface`.

Synopsis

getfaceedges_returntype **ST_GetFaceEdges**(varchar atopology, integer aface);

See Also

Geometry ST_Simplify, ST_IsSimple, ST_IsValid, ST_ModEdgeSplit

9.8.6 RemoveUnusedPrimitives

RemoveUnusedPrimitives — Removes topology primitives which not needed to define existing TopoGeometry objects.

Synopsis

int **RemoveUnusedPrimitives**(text topology_name, geometry bbox);

Description

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

See Also

ST_ModEdgeHeal, ST_RemEdgeModFace

9.9 TopoGeometry Constructors

9.9.1 CreateTopoGeom

CreateTopoGeom — Creates a new topo geometry object from topo element array - 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);

Description

Creates a topogeometry object for layer denoted by `layer_id` and registers it in the relations table in the `toponame` schema.

`tg_type` is an integer: 1:[multi]point (punctal), 2:[multi]line (lineal), 3:[multi]poly (areal), 4:collection. `layer_id` is the layer id in the topology.layer table.

punctal layers are formed from set of nodes, lineal layers are formed from a set of edges, areal layers are formed from a set of faces, and collections can be formed from a mixture of nodes, edges, and faces.

Omitting the array of components generates an empty TopoGeometry object.

Availability: 1.1

Examples: Form from existing edges

Create a topogeom in ri_topo schema for layer 2 (our ri_roads), of type (2) LINE, for the first edge (we loaded in `ST_CreateTopoGeo`)

```
INSERT INTO ri.ri_roads(road_name, topo) VALUES('Unknown', topology.CreateTopoGeom('ri_topo ↩
    ',2,2,'{{1,2}}'::topology.topoelementarray);
```

Examples: Convert an areal geometry to best guess topogeometry

Lets say we have geometries that should be formed from a collection of faces. We have for example blockgroups table and want to know the topo geometry of each block group. If our data was perfectly aligned, we could do this:

```
-- 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;
```

See Also

AddTopoGeometryColumn, toTopoGeom ST_CreateTopoGeo, ST_GetFaceGeometry, TopoElementArray, TopoElementArray_Agg

9.9.2 toTopoGeom

toTopoGeom — Converts a simple Geometry into a topo geometry.

Synopsis

topogeometry **toTopoGeom**(geometry geom, varchar toponame, integer layer_id, float8 tolerance);
topogeometry **toTopoGeom**(geometry geom, topogeometry topogeom, float8 tolerance);

Description

Converts a simple Geometry into a TopoGeometry.

Topological primitives required to represent the input geometry will be added to the underlying topology, possibly splitting existing ones, and they will be associated with the output TopoGeometry in the `relation` table.

Existing TopoGeometry objects (with the possible exception of `topogeom`, if given) will retain their shapes.

When `tolerance` is given it will be used to snap the input geometry to existing primitives.

In the first form a new TopoGeometry will be created for the given layer (`layer_id`) of the given topology (`toponame`).

In the second form the primitives resulting from the conversion will be added to the pre-existing TopoGeometry (`topogeom`), possibly adding space to its final shape. To have the new shape completely replace the old one see clearTopoGeom.

Availability: 2.0

Enhanced: 2.1.0 adds the version taking an existing TopoGeometry.

Examples

This is a full self-contained workflow

```
 -- 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
  );
```

See Also

CreateTopology, AddTopoGeometryColumn, CreateTopoGeom, TopologySummary, clearTopoGeom

9.9.3 TopoElementArray_Agg

TopoElementArray_Agg — Returns a `topoelementarray` for a set of element_id, type arrays (topoelements).

Synopsis

topoelementarray **TopoElementArray_Agg**(topoelement set tefield);

Description

Used to create a TopoElementArray from a set of TopoElement.

Availability: 2.0.0

Examples

```
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}}
```

See Also

TopoElement, TopoElementArray

9.9.4 TopoElement

TopoElement — Converts a topogeometry to a topoelement.

Synopsis

topoelement **TopoElement**(topogeometry topo);

Description

Converts a TopoGeometry to a TopoElement.

Availability: 3.4.0

Examples

This is a full self-contained workflow

```
-- 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;
```

See Also

TopoElementArray_Agg, TopoGeometry, TopoElement

9.10 TopoGeometry Editors

9.10.1 clearTopoGeom

clearTopoGeom — Clears the content of a topo geometry.

Synopsis

topogeometry **clearTopoGeom**(topogeometry topogeom);

Description

Clears the content a TopoGeometry turning it into an empty one. Mostly useful in conjunction with toTopoGeom to replace the shape of existing objects and any dependent object in higher hierarchical levels.

Availability: 2.1

Examples

```
-- Shrink all TopoGeometry polygons by 10 meters
UPDATE nei_topo SET topo = ST_Buffer(clearTopoGeom(topo), -10);
```

See Also

toTopoGeom

9.10.2 TopoGeom_addElement

TopoGeom_addElement — Adds an element to the definition of a TopoGeometry.

Synopsis

topogeometry **TopoGeom_addElement**(topogeometry tg, topoelement el);

Description

Adds a TopoElement to the definition of a TopoGeometry object. Does not error out if the element is already part of the definition.

Availability: 2.3

Examples

```
-- Add edge 5 to TopoGeometry tg
UPDATE mylayer SET tg = TopoGeom_addElement(tg, '{5,2}');
```

See Also

TopoGeom_remElement, CreateTopoGeom

9.10.3 TopoGeom_remElement

TopoGeom_remElement — Removes an element from the definition of a TopoGeometry.

Synopsis

topogeometry **TopoGeom_remElement**(topogeometry tg, topoelement el);

Description

Removes a TopoElement from the definition of a TopoGeometry object.

Availability: 2.3

Examples

```
-- Remove face 43 from TopoGeometry tg
UPDATE mylayer SET tg = TopoGeom_remElement(tg, '{43,3}');
```

See Also

TopoGeom_addElement, CreateTopoGeom

9.10.4 TopoGeom_addTopoGeom

TopoGeom_addTopoGeom — Adds element of a TopoGeometry to the definition of another TopoGeometry.

Synopsis

topogeometry **TopoGeom_addTopoGeom**(topogeometry tgt, topogeometry src);

Description

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

Examples

```
-- 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
);
```

See Also

TopoGeom_addElement, clearTopoGeom, CreateTopoGeom

9.10.5 toTopoGeom

toTopoGeom — Adds a geometry shape to an existing topo geometry.

Description

Refer to toTopoGeom.

9.11 TopoGeometry Accessors

9.11.1 GetTopoGeomElementArray

GetTopoGeomElementArray — Returns a `topoelementarray` (an array of topoelements) containing the topological elements and type of the given TopoGeometry (primitive elements).

Synopsis

topoelementarray **GetTopoGeomElementArray**(varchar toponame, integer layer_id, integer tg_id);

topoelementarray **GetTopoGeomElementArray**(topogeometry tg);

Description

Returns a TopoElementArray containing the topological elements and type of the given TopoGeometry (primitive elements). This is similar to GetTopoGeomElements except it returns the elements as an array rather than as a dataset.

tg_id is the topogeometry id of the topogeometry object in the topology in the layer denoted by `layer_id` in the topology.layer table.

Availability: 1.1

Examples

See Also

GetTopoGeomElements, TopoElementArray

9.11.2 GetTopoGeomElements

GetTopoGeomElements — Returns a set of `topoelement` objects containing the topological element_id,element_type of the given TopoGeometry (primitive elements).

Synopsis

setof topoelement **GetTopoGeomElements**(varchar toponame, integer layer_id, integer tg_id);

setof topoelement **GetTopoGeomElements**(topogeometry tg);

Description

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 is the topogeometry id of the topogeometry object in the topology in the layer denoted by `layer_id` in the topology.layer table.

Availability: 2.0.0

Examples

See Also

GetTopoGeomElementArray, TopoElement, TopoGeom_addElement, TopoGeom_remElement

9.11.3 ST_SRID

ST_SRID — Returns the spatial reference identifier for a topogeometry.

Description

Returns the GML representation of a topogeometry in version GML3 format. If no `nsprefix_in` is specified then `gml` is used. Pass in an empty string for nsprefix to get a non-qualified name space. The precision (default: 15) and options (default 1) parameters, if given, are passed untouched to the underlying call to ST_AsGML.

The `visitedTable` parameter, if given, is used for keeping track of the visited Node and Edge elements so to use cross-references (xlink:xref) rather than duplicating definitions. The table is expected to have (at least) two integer fields: 'element_type' and 'element_id'. The calling user must have both read and write privileges on the given table. For best performance, an index should be defined on `element_type` and `element_id`, in that order. Such index would be created automatically by adding a unique constraint to the fields. Example:

```
CREATE TABLE visited (
  element_type integer, element_id integer,
  unique(element_type, element_id)
);
```

The `idprefix` parameter, if given, will be prepended to Edge and Node tag identifiers.

The `gmlver` parameter, if given, will be passed to the underlying ST_AsGML. Defaults to 3.

Availability: 2.0.0

Examples

This uses the topo geometry we created in CreateTopoGeom

```
SELECT topology.AsGML(topo) As rdgml
  FROM ri.roads
  WHERE road_name = 'Unknown';

-- rdgml--
<gml:TopoCurve>
    <gml:directedEdge>
        <gml:Edge gml:id="E1">
            <gml:directedNode orientation="-">
                <gml:Node gml:id="N1"/>
            </gml:directedNode>
            <gml: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>
```

Same exercise as previous without namespace

```
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>
```

See Also

CreateTopoGeom, ST_CreateTopoGeo

9.12.2 AsTopoJSON

AsTopoJSON — Returns the TopoJSON representation of a topogeometry.

Synopsis

text AsTopoJSON(topogeometry tg, regclass edgeMapTable);

Description

Returns the TopoJSON representation of a topogeometry. If edgeMapTable is not null, it will be used as a lookup/storage mapping of edge identifiers to arc indices. This is to be able to allow for a compact "arcs" array in the final document.

The table, if given, is expected to have an "arc_id" field of type "serial" and an "edge_id" of type integer; the code will query th table for "edge_id" so it is recommended to add an index on that field.

Note
Arc indices in the TopoJSON output are 0-based but they are 1-based in the "edgeMapTable" table.

A full TopoJSON document will be need to contain, in addition to the snippets returned by this function, the actual arcs pl some headers. See the TopoJSON specification.

Availability: 2.1.0

Enhanced: 2.2.1 added support for puntal inputs

See Also

ST_AsGeoJSON

Examples

```
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
```

Raster processing: Optional parameters used to manipulate input raster dataset

-s <SRID> Assign output raster with specified SRID. If not provided or is zero, raster's metadata will be checked to determine an appropriate SRID.

-b BAND Index (1-based) of band to extract from raster. For more than one band index, separate with comma (,). If unspecified, all bands of raster will be extracted.

-t TILE_SIZE Cut raster into tiles to be inserted one per table row. `TILE_SIZE` is expressed as WIDTHxHEIGHT or set to the value "auto" to allow the loader to compute an appropriate tile size using the first raster and applied to all rasters.

-P Pad right-most and bottom-most tiles to guarantee that all tiles have the same width and height.

-R, --register Register the raster as a filesystem (out-db) raster.
Only the metadata of the raster and path location to the raster is stored in the database (not the pixels).

-l OVERVIEW_FACTOR Create overview of the raster. For more than one factor, separate with comma(,). Overview table name follows the pattern o_`overview factor_table`, where `overview factor` is a placeholder for numerical overview factor and `table` is replaced with the base table name. Created overview is stored in the database and is not affected by -R. Note that your generated sql file will contain both the main table and overview tables.

-N NODATA NODATA value to use on bands without a NODATA value.

Optional parameters used to manipulate database objects

-f COLUMN Specify name of destination raster column, default is 'rast'

-F Add a column with the name of the file

-n COLUMN Specify the name of the filename column. Implies -F.

-q Wrap PostgreSQL identifiers in quotes.

-I Create a GiST index on the raster column.

-M Vacuum analyze the raster table.

-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 Specify the tablespace for the new table. Note that indices (including the primary key) will still use the default tablespace unless the -X flag is also used.

-X tablespace Specify the tablespace for the table's new index. This applies to the primary key and the spatial index if the -I flag is used.

-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 Execute each statement individually, do not use a transaction.

-E ENDIAN Control endianness of generated binary output of raster; specify 0 for XDR and 1 for NDR (default); only NDR output is supported now

-V version Specify version of output format. Default is 0. Only 0 is supported at this time.

10.1.2 Creating rasters using PostGIS raster functions

On many occasions, you'll want to create rasters and raster tables right in the database. There are a plethora of functions to do that. The general steps to follow.

1. Create a table with a raster column to hold the new raster records which can be accomplished with:

```
CREATE TABLE myrasters(rid serial primary key, rast raster);
```

2. There are many functions to help with that goal. If you are creating rasters not as a derivative of other rasters, you will want to start with: ST_MakeEmptyRaster, followed by ST_AddBand

   You can also create rasters from geometries. To achieve that you'll want to use ST_AsRaster perhaps accompanied with other functions such as ST_Union or ST_MapAlgebraFct or any of the family of other map algebra functions.

   There are even many more options for creating new raster tables from existing tables. For example you can create a raster table in a different projection from an existing one using ST_Transform

3. Once you are done populating your table initially, you'll want to create a spatial index on the raster column with something like:

```
CREATE INDEX myrasters_rast_st_convexhull_idx ON myrasters USING gist( ST_ConvexHull( ↩
    rast) );
```

   Note the use of ST_ConvexHull since most raster operators are based on the convex hull of the rasters.

   Note
   Pre-2.0 versions of PostGIS raster were based on the envelop rather than the convex hull. For the spatial indexes to work properly you'll need to drop those and replace with convex hull based index.

4. Apply raster constraints using AddRasterConstraints

10.1.3 Using "out db" cloud rasters

The `raster2pgsql` tool uses GDAL to access raster data, and can take advantage of a key GDAL feature: the ability to read from rasters that are stored remotely in cloud "object stores" (e.g. AWS S3, Google Cloud Storage).

Efficient use of cloud stored rasters requires the use of a "cloud optimized" format. The most well-known and widely used is the "cloud optimized GeoTIFF" format. Using a non-cloud format, like a JPEG, or an un-tiled TIFF will result in very poor performance, as the system will have to download the entire raster each time it needs to access a subset.

First, load your raster into the cloud storage of your choice. Once it is loaded, you will have a URI to access it with, either an "http" URI, or sometimes a URI specific to the service. (e.g., "s3://bucket/object"). To access non-public buckets, you will need to supply GDAL config options to authenticate your connection. Note that this command is *reading* from the cloud raster and *writing* to the database.

```
AWS_ACCESS_KEY_ID=xxxxxxxxxxxxxxxxxxx \
```

For non-public rasters, you may have to provide access keys to read from the cloud rasters. The same keys you used to write the `raster2pgsql` call can be set for use inside the database, with the postgis.gdal_vsi_options configuration. Note that multiple options can be set by space-separating the `key=value` pairs.

```
SET postgis.gdal_vsi_options = 'AWS_ACCESS_KEY_ID=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.

10.2 Raster Catalogs

There are two raster catalog views that come packaged with PostGIS. Both views utilize information embedded in the constraints of the raster tables. As a result the catalog views are always consistent with the raster data in the tables since the constraints are enforced.

1. `raster_columns` this view catalogs all the raster table columns in your database.
2. `raster_overviews` this view catalogs all the raster table columns in your database that serve as overviews for a finer grained table. Tables of this type are generated when you use the `-l` switch during load.

10.2.1 Raster Columns Catalog

The `raster_columns` is a catalog of all raster table columns in your database that are of type raster. It is a view utilizing the constraints on the tables so the information is always consistent even if you restore one raster table from a backup of another database. The following columns exist in the `raster_columns` catalog.

If you created your tables not with the loader or forgot to specify the `-C` flag during load, you can enforce the constraints after the fact using AddRasterConstraints so that the `raster_columns` catalog registers the common information about your raster tiles.

- `r_table_catalog` The database the table is in. This will always read the current database.
- `r_table_schema` The database schema the raster table belongs to.
- `r_table_name` raster table
- `r_raster_column` the column in the `r_table_name` table that is of type raster. There is nothing in PostGIS preventing you from having multiple raster columns per table so its possible to have a raster table listed multiple times with a different raster column for each.
- `srid` The spatial reference identifier of the raster. Should be an entry in the Section 4.5.
- `scale_x` The scaling between geometric spatial coordinates and pixel. This is only available if all tiles in the raster column have the same `scale_x` and this constraint is applied. Refer to ST_ScaleX for more details.
- `scale_y` The scaling between geometric spatial coordinates and pixel. This is only available if all tiles in the raster column have the same `scale_y` and the `scale_y` constraint is applied. Refer to ST_ScaleY for more details.
- `blocksize_x` The width (number of pixels across) of each raster tile . Refer to ST_Width for more details.
- `blocksize_y` The width (number of pixels down) of each raster tile . Refer to ST_Height for more details.
- `same_alignment` A boolean that is true if all the raster tiles have the same alignment . Refer to ST_SameAlignment for more details.
- `regular_blocking` If the raster column has the spatially unique and coverage tile constraints, the value with be TRUE. Otherwise, it will be FALSE.

- num_bands The number of bands in each tile of your raster set. This is the same information as what is provided by ST_NumBands
- pixel_types An array defining the pixel type for each band. You will have the same number of elements in this array as you have number of bands. The pixel_types are one of the following defined in ST_BandPixelType.
- nodata_values An array of double precision numbers denoting the nodata_value for each band. You will have the same number of elements in this array as you have number of bands. These numbers define the pixel value for each band that should be ignored for most operations. This is similar information provided by ST_BandNoDataValue.
- out_db An array of boolean flags indicating if the raster bands data is maintained outside the database. You will have the same number of elements in this array as you have number of bands.
- extent This is the extent of all the raster rows in your raster set. If you plan to load more data that will change the extent of the set, you'll want to run the DropRasterConstraints function before load and then reapply constraints with AddRasterConstraints after load.
- spatial_index A boolean that is true if raster column has a spatial index.

10.2.2 Raster Overviews

raster_overviews catalogs information about raster table columns used for overviews and additional information about them that is useful to know when utilizing overviews. Overview tables are cataloged in both raster_columns and raster_overvi because they are rasters in their own right but also serve an additional special purpose of being a lower resolution caricature of a higher resolution table. These are generated along-side the main raster table when you use the -l switch in raster loading or can be generated manually using AddOverviewConstraints.

Overview tables contain the same constraints as other raster tables as well as additional informational only constraints specific to overviews.

Note
The information in raster_overviews does not duplicate the information in raster_columns. If you need the information about an overview table present in raster_columns you can join the raster_overviews and raster_columns together to get the full set of information you need.

Two main reasons for overviews are:

1. Low resolution representation of the core tables commonly used for fast mapping zoom-out.
2. Computations are generally faster to do on them than their higher resolution parents because there are fewer records and each pixel covers more territory. Though the computations are not as accurate as the high-res tables they support, they can be sufficient in many rule-of-thumb computations.

The raster_overviews catalog contains the following columns of information.

- o_table_catalog The database the overview table is in. This will always read the current database.
- o_table_schema The database schema the overview raster table belongs to.
- o_table_name raster overview table name
- o_raster_column the raster column in the overview table.
- r_table_catalog The database the raster table that this overview services is in. This will always read the current database.
- r_table_schema The database schema the raster table that this overview services belongs to.
- r_table_name raster table that this overview services.

- `r_raster_column` the raster column that this overview column services.
- `overview_factor` - this is the pyramid level of the overview table. The higher the number the lower the resolution of the table. raster2pgsql if given a folder of images, will compute overview of each image file and load separately. Level 1 is assumed and always the original file. Level 2 is will have each tile represent 4 of the original. So for example if you have a folder of 5000x5000 pixel image files that you chose to chunk 125x125, for each image file your base table will have

10.3.2 ASP.NET C# Example Outputting using ST_AsPNG in concert with other raster functions

In this section, we'll demonstrate how to use Npgsql PostgreSQL .NET driver and the ST_AsGDALRaster family of functions to output band 1,2,3 of a raster to a PHP request stream that can then be embedded in an img src html tag.

You will need the npgsql .NET PostgreSQL driver for this exercise which you can get the latest of from http://npgsql.projects.postgresql.o . Just download the latest and drop into your ASP.NET bin folder and you'll be good to go.

The sample query demonstrates how to combine a whole bunch of raster functions together to grab all tiles that intersect a particular wgs 84 bounding box and then unions with ST_Union the intersecting tiles together returning all bands, transforms to user specified projection using ST_Transform, and then outputs the results as a png using ST_AsPNG.

This is same example as Section 10.3.1 except implemented in C#.

You would call the below using

```
http://mywebserver/TestRaster.ashx?srid=2249
```

to get the raster image in Massachusetts state plane feet.

```
 -- web.config connection string section --
<connectionStrings>
    <add name="DSN"
        connectionString="server=localhost;database=mydb;Port=5432;User Id=myuser;password= ↩
            mypwd"/>
</connectionStrings>
```

```
// Code for TestRaster.ashx
<%@ WebHandler Language="C#" Class="TestRaster" %>
using System;
using System.Data;
using System.Web;
using Npgsql;

public class TestRaster : IHttpHandler
{
  public void ProcessRequest(HttpContext context)
  {

    context.Response.ContentType = "image/png";
    context.Response.BinaryWrite(GetResults(context));

  }

  public bool IsReusable {
    get { return false; }
  }

  public byte[] GetResults(HttpContext context)
  {
    byte[] result = null;
    NpgsqlCommand command;
    string sql = null;
    int input_srid = 26986;
        try {
        using (NpgsqlConnection conn = new NpgsqlConnection(System.Configuration. ↩
            ConfigurationManager.ConnectionStrings["DSN"].ConnectionString)) {
          conn.Open();

                if (context.Request["srid"] != null)
                {
                    input_srid = Convert.ToInt32(context.Request["srid"]);
                }
```

```
                sql = @"SELECT ST_AsPNG(
                            ST_Transform(
                          ST_AddBand(
                                    ST_Union(rast,1), ARRAY[ST_Union(rast,2),ST_Union(rast,3)])
                                ,:input_srid) ) As new_rast
                            FROM aerials.boston
                              WHERE
                                  ST_Intersects(rast,
                                        ST_Transform(ST_MakeEnvelope(-71.1217, 42.227,  ↩
                                            -71.1210, 42.218,4326),26986) )";
          command = new NpgsqlCommand(sql, conn);
                command.Parameters.Add(new NpgsqlParameter("input_srid", input_srid));


          result = (byte[]) command.ExecuteScalar();
                conn.Close();
      }

    }
        catch (Exception ex)
        {
            result = null;
            context.Response.Write(ex.Message.Trim());
        }
    return result;
  }
}
```

10.3.3 Java console app that outputs raster query as Image file

This is a simple java console app that takes a query that returns one image and outputs to specified file.

You can download the latest PostgreSQL JDBC drivers from http://jdbc.postgresql.org/download.html

You can compile the following code using a command something like:

```
set env CLASSPATH .:..\postgresql-9.0-801.jdbc4.jar
javac SaveQueryImage.java
jar cfm SaveQueryImage.jar Manifest.txt *.class
```

And call it from the command-line with something like

```
java -jar SaveQueryImage.jar "SELECT ST_AsPNG(ST_AsRaster(ST_Buffer(ST_Point(1,5),10, ' ↩
    quad_segs=2'),150, 150, '8BUI',100));" "test.png"
```

```
 -- Manifest.txt --
Class-Path: postgresql-9.0-801.jdbc4.jar
Main-Class: SaveQueryImage
```

```
// Code for SaveQueryImage.java
import java.sql.Connection;
import java.sql.SQLException;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.io.*;

public class SaveQueryImage {
  public static void main(String[] argv) {
      System.out.println("Checking if Driver is registered with DriverManager.");
```

```
      try {
        //java.sql.DriverManager.registerDriver (new org.postgresql.Driver());
        Class.forName("org.postgresql.Driver");
      }
      catch (ClassNotFoundException cnfe) {
        System.out.println("Couldn't find the driver!");
        cnfe.printStackTrace();
        System.exit(1);
      }

      Connection conn = null;

      try {
        conn = DriverManager.getConnection("jdbc:postgresql://localhost:5432/mydb","myuser ↩
            ", "mypwd");
        conn.setAutoCommit(false);

        PreparedStatement sGetImg = conn.prepareStatement(argv[0]);

        ResultSet rs = sGetImg.executeQuery();

    FileOutputStream fout;
    try
    {
      rs.next();
      /** Output to file name requested by user **/
      fout = new FileOutputStream(new File(argv[1]) );
      fout.write(rs.getBytes(1));
      fout.close();
    }
    catch(Exception e)
    {
      System.out.println("Can't create file");
      e.printStackTrace();
    }

        rs.close();
    sGetImg.close();
        conn.close();
      }
      catch (SQLException se) {
        System.out.println("Couldn't connect: print out a stack trace and exit.");
        se.printStackTrace();
        System.exit(1);
      }
  }
}
```

10.3.4 Use PLPython to dump out images via SQL

This is a plpython stored function that creates a file in the server directory for each record. Requires you have plpython installed. Should work fine with both plpythonu and 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
```

10.3.5 Outputting Rasters with PSQL

Sadly PSQL doesn't have easy to use built-in functionality for outputting binaries. This is a bit of a hack that piggy backs on PostgreSQL somewhat legacy large object support. To use first launch your psql commandline connected to your database.

Unlike the python approach, this approach creates the file on your local computer.

```
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);
```

See Also

ST_MapAlgebra (callback function version)

11.1.4 raster

raster — raster spatial data type.

Description

raster is a spatial data type used to represent raster data such as those imported from JPEGs, TIFFs, PNGs, digital elevation models. Each raster has 1 or more bands each having a set of pixel values. Rasters can be georeferenced.

---

pixeltype text One of defined pixel types as described in ST_BandPixelType

nodataval double precision Value to treat as no data. For image outputs that support transparency, these will be blank.

Example: Reclassify band 2 as an 8BUI where 255 is nodata value

```
SELECT ROW(2, '0-100:1-10, 101-500:11-150,501 - 10000: 151-254', '8BUI', 255)::reclassarg;
```

Example: Reclassify band 1 as an 1BB and no nodata value defined

```
SELECT ROW(1, '0-100]:0, (100-255:1', '1BB', NULL)::reclassarg;
```

See Also

ST_Reclass

11.1.6 summarystats

summarystats — A composite type returned by the ST_SummaryStats and ST_SummaryStatsAgg functions.

Description

A composite type returned by the ST_SummaryStats and ST_SummaryStatsAgg functions.

count integer Number of pixels counted for the summary statistics.

sum double precision Sum of all counted pixel values.

mean double precision Arithmetic mean of all counted pixel values.

stddev double precision Standard deviation of all counted pixel values.

min double precision Minimum value of counted pixel values.

max double precision Maximum value of counted pixel values.

See Also

ST_SummaryStats, ST_SummaryStatsAgg

11.1.7 unionarg

unionarg — A composite type used as input into the ST_Union function defining the bands to be processed and behavior of the UNION operation.

Description

A composite type used as input into the ST_Union function defining the bands to be processed and behavior of the UNION operation.

nband integer 1-based value indicating the band of each input raster to be processed.

uniontype text Type of UNION operation. One of defined types as described in ST_Union.

See Also

ST_Union

11.2 Raster Management

11.2.1 AddRasterConstraints

AddRasterConstraints — Adds raster constraints to a loaded raster table for a specific column that constrains spatial ref, scaling, blocksize, alignment, bands, band type and a flag to denote if raster column is regularly blocked. The table must be loaded with data for the constraints to be inferred. Returns true if the constraint setting was accomplished and issues a notice otherwise.

Synopsis

boolean **AddRasterConstraints**(name rasttable, name rastcolumn, boolean srid=true, boolean scale_x=true, boolean scale_y=true, boolean blocksize_x=true, boolean blocksize_y=true, boolean same_alignment=true, boolean regular_blocking=false, boolean num_bands=true , boolean pixel_types=true , boolean 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);

Description

Generates constraints on a raster column that are used to display information in the `raster_columns` raster catalog. The `rastschema` is the name of the table schema the table resides in. The `srid` must be an integer value reference to an entry in the SPATIAL_REF_SYS table.

`raster2pgsql` loader uses this function to register raster tables

Valid constraint names to pass in: refer to Section 10.2.1 for more details.

- `blocksize` sets both X and Y blocksize
- `blocksize_x` sets X tile (width in pixels of each tile)
- `blocksize_y` sets Y tile (height in pixels of each tile)
- `extent` computes extent of whole table and applies constraint all rasters must be within that extent
- `num_bands` number of bands
- `pixel_types` reads array of pixel types for each band ensure all band n have same pixel type
- `regular_blocking` sets spatially unique (no two rasters can be spatially the same) and coverage tile (raster is aligned to a coverage) constraints
- `same_alignment` ensures they all have same alignment meaning any two tiles you compare will return true for. Refer to ST_SameAlignment.
- `srid` ensures all have same srid
- More -- any listed as inputs into the above functions

11.2.2 DropRasterConstraints

DropRasterConstraints — Drops PostGIS raster constraints that refer to a raster table column. Useful if you need to reload data or update your raster column data.

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);

Description

Drops PostGIS raster constraints that refer to a raster table column that were added by AddRasterConstraints. Useful if you need to load more data or update your raster column data. You do not need to do this if you want to get rid of a raster table or a raster column.

To drop a raster table use the standard

```
DROP TABLE mytable
```

To drop just a raster column and leave the rest of the table, use standard SQL

```
ALTER TABLE mytable DROP COLUMN rast
```

the table will disappear from the `raster_columns` catalog if the column or table is dropped. However if only the constraints are dropped, the raster column will still be listed in the `raster_columns` catalog, but there will be no other information about it aside from the column name and table.

Availability: 2.0.0

Examples

```
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 |         |         |             |             |           |             |
```

See Also

AddRasterConstraints

11.2.3 AddOverviewConstraints

AddOverviewConstraints — Tag a raster column as being an overview of another.

Synopsis

boolean **AddOverviewConstraints**(name ovschema, name ovtable, name ovcolumn, name refschema, name reftable, name refcolumn, int ovfactor);
boolean **AddOverviewConstraints**(name ovtable, name ovcolumn, name reftable, name refcolumn, int ovfactor);

Description

Adds constraints on a raster column that are used to display information in the `raster_overviews` raster catalog.

The `ovfactor` parameter represents the scale multiplier in the overview column: higher overview factors have lower resolution.

When the `ovschema` and `refschema` parameters are omitted, the first table found scanning the `search_path` will be used.

Availability: 2.0.0

Examples

```
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)
```

See Also

Section 10.2.2, DropOverviewConstraints, ST_CreateOverview, AddRasterConstraints

11.2.4 DropOverviewConstraints

DropOverviewConstraints — Untag a raster column from being an overview of another.

Synopsis

boolean **DropOverviewConstraints**(name ovschema, name ovtable, name ovcolumn);
boolean **DropOverviewConstraints**(name ovtable, name ovcolumn);

Description

Remove from a raster column the constraints used to show it as being an overview of another in the `raster_overviews` raster catalog.

When the `ovschema` parameter is omitted, the first table found scanning the `search_path` will be used.

Availability: 2.0.0

See Also

Section 10.2.2, AddOverviewConstraints, DropRasterConstraints

11.2.5 PostGIS_GDAL_Version

PostGIS_GDAL_Version — Reports the version of the GDAL library in use by PostGIS.

Synopsis

text **PostGIS_GDAL_Version**();

Description

Reports the version of the GDAL library in use by PostGIS. Will also check and report if GDAL can find its data files.

Examples

```
SELECT PostGIS_GDAL_Version();
       postgis_gdal_version
---------------------------------
 GDAL 1.11dev, released 2013/04/13
```

See Also

postgis.gdal_datapath

11.2.6 PostGIS_Raster_Lib_Build_Date

PostGIS_Raster_Lib_Build_Date — Reports full raster library build date.

Synopsis

text **PostGIS_Raster_Lib_Build_Date**();

Description

Reports raster build date

Examples

```
SELECT PostGIS_Raster_Lib_Build_Date();
postgis_raster_lib_build_date
-----------------------------
2010-04-28 21:15:10
```

See Also

PostGIS_Raster_Lib_Version

11.2.7 PostGIS_Raster_Lib_Version

PostGIS_Raster_Lib_Version — Reports full raster version and build configuration infos.

Synopsis

text **PostGIS_Raster_Lib_Version**();

Description

Reports full raster version and build configuration infos.

Examples

```
SELECT PostGIS_Raster_Lib_Version();
postgis_raster_lib_version
-----------------------------
 2.0.0
```

See Also

PostGIS_Lib_Version

11.2.8 ST_GDALDrivers

ST_GDALDrivers — Returns a list of raster formats supported by PostGIS through GDAL. Only those formats with can_write=True can be used by ST_AsGDALRaster

Synopsis

setof record **ST_GDALDrivers**(integer OUT idx, text OUT short_name, text OUT long_name, text OUT can_read, text OUT can_write, text OUT create_options);

```
 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
```

Example: List of options for each driver

```
-- Output the create options XML column of JPEG as a table  --
-- Note you can use these creator options in ST_AsGDALRaster options argument
SELECT (xpath('@name', g.opt))[1]::text As oname,
       (xpath('@type', g.opt))[1]::text As otype,
       (xpath('@description', g.opt))[1]::text As descrip
FROM (SELECT unnest(xpath('/CreationOptionList/Option', create_options::xml)) As opt
FROM  st_gdaldrivers()
WHERE short_name = 'JPEG') As g;

       oname        |  otype  |      descrip
--------------------+---------+---------------------------------------------------
 PROGRESSIVE        | boolean | whether to generate a progressive JPEG
 QUALITY            | int     | good=100, bad=0, default=75
 WORLDFILE          | boolean | whether to generate a worldfile
 INTERNAL_MASK      | boolean | whether to generate a validity mask
 COMMENT            | string  | Comment
 SOURCE_ICC_PROFILE | string  | ICC profile encoded in Base64
 EXIF_THUMBNAIL     | boolean | whether to generate an EXIF thumbnail(overview).
                                By default its max dimension will be 128
 THUMBNAIL_WIDTH    | int     | Forced thumbnail width
 THUMBNAIL_HEIGHT   | int     | Forced thumbnail height
(9 rows)
```

```
-- raw xml output for creator options for GeoTiff --
SELECT create_options
FROM st_gdaldrivers()
WHERE short_name = 'GTiff';

<CreationOptionList>
    <Option name="COMPRESS" type="string-select">
        <Value>NONE</Value>
        <Value>LZW</Value>
        <Value>PACKBITS</Value>
        <Value>JPEG</Value>
        <Value>CCITTRLE</Value>
        <Value>CCITTFAX3</Value>
        <Value>CCITTFAX4</Value>
        <Value>DEFLATE</Value>
    </Option>
    <Option name="PREDICTOR" type="int" description="Predictor Type"/>
    <Option name="JPEG_QUALITY" type="int" description="JPEG quality 1-100" default="75"/>
    <Option name="ZLEVEL" type="int" description="DEFLATE compression level 1-9" default ↩
        ="6"/>
```

```
    <Option name="NBITS" type="int" description="BITS for sub-byte files (1-7), sub-uint16 ↩
        (9-15), sub-uint32 (17-31)"/>
    <Option name="INTERLEAVE" type="string-select" default="PIXEL">
        <Value>BAND</Value>
        <Value>PIXEL</Value>
    </Option>
    <Option name="TILED" type="boolean" description="Switch to tiled format"/>
    <Option name="TFW" type="boolean" description="Write out world file"/>
    <Option name="RPB" type="boolean" description="Write out .RPB (RPC) file"/>
    <Option name="BLOCKXSIZE" type="int" description="Tile Width"/>
    <Option name="BLOCKYSIZE" type="int" description="Tile/Strip Height"/>
    <Option name="PHOTOMETRIC" type="string-select">
        <Value>MINISBLACK</Value>
        <Value>MINISWHITE</Value>
        <Value>PALETTE</Value>
        <Value>RGB</Value>
        <Value>CMYK</Value>
        <Value>YCBCR</Value>
        <Value>CIELAB</Value>
        <Value>ICCLAB</Value>
        <Value>ITULAB</Value>
    </Option>
    <Option name="SPARSE_OK" type="boolean" description="Can newly created files have ↩
        missing blocks?" default="FALSE"/>
    <Option name="ALPHA" type="boolean" description="Mark first extrasample as being alpha ↩
        "/>
    <Option name="PROFILE" type="string-select" default="GDALGeoTIFF">
        <Value>GDALGeoTIFF</Value>
        <Value>GeoTIFF</Value>
        <Value>BASELINE</Value>
    </Option>
    <Option name="PIXELTYPE" type="string-select">
        <Value>DEFAULT</Value>
        <Value>SIGNEDBYTE</Value>
    </Option>
    <Option name="BIGTIFF" type="string-select" description="Force creation of BigTIFF file ↩
        ">
        <Value>YES</Value>
        <Value>NO</Value>
        <Value>IF_NEEDED</Value>
        <Value>IF_SAFER</Value>
    </Option>
    <Option name="ENDIANNESS" type="string-select" default="NATIVE" description="Force ↩
        endianness of created file. For DEBUG purpose mostly">
        <Value>NATIVE</Value>
        <Value>INVERTED</Value>
        <Value>LITTLE</Value>
        <Value>BIG</Value>
    </Option>
    <Option name="COPY_SRC_OVERVIEWS" type="boolean" default="NO" description="Force copy ↩
        of overviews of source dataset (CreateCopy())"/>
</CreationOptionList>

-- Output the create options XML column for GTiff as a table  --
SELECT (xpath('@name', g.opt))[1]::text As oname,
       (xpath('@type', g.opt))[1]::text As otype,
       (xpath('@description', g.opt))[1]::text As descrip,
       array_to_string(xpath('Value/text()', g.opt),', ') As vals
FROM (SELECT unnest(xpath('/CreationOptionList/Option', create_options::xml)) As opt
FROM  st_gdaldrivers()
WHERE short_name = 'GTiff') As g;
```

```
       oname         |     otype     |                              descrip  ↩
                                          |                                   vals
--------------------+---------------+-------------------------------------------------------------

 COMPRESS            | string-select |  ↩
                                                                          | NONE, LZW,  ↩
    PACKBITS, JPEG, CCITTRLE, CCITTFAX3, CCITTFAX4, DEFLATE
 PREDICTOR           | int           | Predictor Type  ↩
                                                          |
 JPEG_QUALITY        | int           | JPEG quality 1-100  ↩
                                                      |
 ZLEVEL              | int           | DEFLATE compression level 1-9  ↩
                                            |
 NBITS               | int           | BITS for sub-byte files (1-7), sub-uint16 (9-15), sub ↩
    -uint32 (17-31) |
 INTERLEAVE          | string-select |  ↩
                                                                          | BAND, PIXEL
 TILED               | boolean       | Switch to tiled format  ↩
                                                  |
 TFW                 | boolean       | Write out world file  ↩
                                                    |
 RPB                 | boolean       | Write out .RPB (RPC) file  ↩
                                                |
 BLOCKXSIZE          | int           | Tile Width  ↩
                                                              |
 BLOCKYSIZE          | int           | Tile/Strip Height  ↩
                                                      |
 PHOTOMETRIC         | string-select |  ↩
                                                                          | MINISBLACK,  ↩
    MINISWHITE, PALETTE, RGB, CMYK, YCBCR, CIELAB, ICCLAB, ITULAB
 SPARSE_OK           | boolean       | Can newly created files have missing blocks?  ↩
                              |
 ALPHA               | boolean       | Mark first extrasample as being alpha  ↩
                                    |
 PROFILE             | string-select |  ↩
                                                                          | GDALGeoTIFF,  ↩
    GeoTIFF, BASELINE
 PIXELTYPE           | string-select |  ↩
                                                                          | DEFAULT,  ↩
    SIGNEDBYTE
 BIGTIFF             | string-select | Force creation of BigTIFF file  ↩
                                          | YES, NO, IF_NEEDED, IF_SAFER
 ENDIANNESS          | string-select | Force endianness of created file. For DEBUG purpose  ↩
    mostly            | NATIVE, INVERTED, LITTLE, BIG
 COPY_SRC_OVERVIEWS | boolean       | Force copy of overviews of source dataset (CreateCopy ↩
    ())              |
(19 rows)
```

See Also

ST_AsGDALRaster, ST_SRID, postgis.gdal_enabled_drivers

11.2.9 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=1, double precision level_interval=100.0, double precision level_base=0.0, double precision[] fixed_levels=ARRAY[], boolean polygonize=false);

Description

Generates a set of vector contours from the provided raster band, using the GDAL contouring algorithm.

When the `fixed_levels` parameter is a non-empty array, the `level_interval` and `level_base` parameters are ignored.

Input parameters are:

`rast` The raster to generate the contour of

`bandnumber` The band to generate the contour of

`level_interval` The elevation interval between contours generated

`level_base` The "base" relative to which contour intervals are applied, this is normally zero, but could be different. To generate 10m contours at 5, 15, 25, ... the LEVEL_BASE would be 5.

`fixed_levels` The elevation interval between contours generated

`polygonize` If `true`, contour polygons will be created, rather than polygon lines.

Return values are a set of records with the following attributes:

`geom` The geometry of the contour line.

`id` A unique identifier given to the contour line by GDAL.

`value` The raster value the line represents. For an elevation DEM input, this would be the elevation of the output contour.

Availability: 3.2.0

Example

```
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;
```

See Also

ST_InterpolateRaster

11.2.10 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);

Description

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,

See Also

UpdateGeometrySRID

11.2.12 ST_CreateOverview

ST_CreateOverview — Create an reduced resolution version of a given raster coverage.

Synopsis

regclass **ST_CreateOverview**(regclass tab, name col, int factor, text algo='NearestNeighbor');

Description

Create an overview table with resampled tiles from the source table. Output tiles will have the same size of input tiles and cover the same spatial extent with a lower resolution (pixel size will be 1/`factor` of the original in both directions).

The overview table will be made available in the `raster_overviews` catalog and will have raster constraints enforced.

Algorithm options are: 'NearestNeighbor', 'Bilinear', 'Cubic', 'CubicSpline', and 'Lanczos'. Refer to: GDAL Warp resampling methods for more details.

Availability: 2.2.0

Example

Output to generally better quality but slower to product format

```
SELECT ST_CreateOverview('mydata.mytable'::regclass, 'rast', 2, 'Lanczos');
```

Output to faster to process default nearest neighbor

```
SELECT ST_CreateOverview('mydata.mytable'::regclass, 'rast', 2);
```

See Also

ST_Retile, AddOverviewConstraints, AddRasterConstraints, Section 10.2.2

11.3 Raster Constructors

11.3.1 ST_AddBand

ST_AddBand — Returns a raster with the new band(s) of given type added with given initial value in the given index location. If no index is specified, the band is added to the end.

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);

Examples: Multiple New Bands

```
SELECT
    *
FROM  ST_BandMetadata(
    ST_AddBand(
        ST_MakeEmptyRaster(10,  10,  0,  0,  1,  -1,  0,  0,  0),
        ARRAY[
            ROW( 663.685 Td [(ROW)]TJ 0 g 0 G  [(.0)]TJ 0 0 0 rg 0 0 0 RG  [( 663.6Ted0)]3.0 Tf 4d0 50
```

Description

Returns an empty raster (having no band) of given dimensions (width & height) and georeferenced in spatial (or world) coordinates with upper left X (upperleftx), upper left Y (upperlefty), pixel size and rotation (scalex, scaley, skewx & skewy) and reference system (srid).

The last version use a single parameter to specify the pixel size (pixelsize). scalex is set to this argument and scaley is set to the negative value of this argument. skewx and skewy are set to 0.

If an existing raster is passed in, it returns a new raster with the same meta data settings (without the bands).

If no srid is specified it defaults to 0. After you create an empty raster you probably want to add bands to it and maybe edit it. Refer to ST_AddBand to define bands and ST_SetValue to set initial pixel values.

Examples

```
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
```

See Also

ST_AddBand, ST_MetaData, ST_ScaleX, ST_ScaleY, ST_SetValue, ST_SkewX, , ST_SkewY

11.3.6 ST_Tile

ST_Tile — Returns a set of rasters resulting from the split of the input raster based upon the desired dimensions of the output rasters.

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);

Description

Returns a set of rasters resulting from the split of the input raster based upon the desired dimensions of the output rasters.

If `padwithnodata` = FALSE, edge tiles on the right and bottom sides of the raster may have different dimensions than the rest of the tiles. If `padwithnodata` = TRUE, all tiles will have the same dimensions with the possibility that edge tiles being padded with NODATA values. If raster band(s) do not have NODATA value(s) specified, one can be specified by setting `nodataval`.

Note
If a specified band of the input raster is out-of-db, the corresponding band in the output rasters will also be out-of-db.

Availability: 2.1.0

Examples

```
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)
```

See Also

ST_Union, ST_Retile

11.3.7 ST_Retile

ST_Retile — Return a set of configured tiles from an arbitrarily tiled raster coverage.

Synopsis

SETOF raster **ST_Retile**(regclass tab, name col, geometry ext, float8 sfx, float8 sfy, int tw, int th, text algo='NearestNeighbor');

Description

Return a set of tiles having the specified scale (`sfx`, `sfy`) and max size (`tw`, `th`) and covering the specified extent (`ext`) with data coming from the specified raster coverage (`tab`, `col`).

Algorithm options are: 'NearestNeighbor', 'Bilinear', 'Cubic', 'CubicSpline', and 'Lanczos'. Refer to: GDAL Warp resampling methods for more details.

Availability: 2.2.0

See Also

ST_CreateOverview

11.3.8 ST_FromGDALRaster

ST_FromGDALRaster — Returns a raster from a supported GDAL raster file.

Synopsis

raster **ST_FromGDALRaster**(bytea gdaldata, integer srid=NULL);

Description

Returns a raster from a supported GDAL raster file. `gdaldata` is of type bytea and should be the contents of the GDAL raster file.

If `srid` is NULL, the function will try to automatically assign the SRID from the GDAL raster. If `srid` is provided, the value provided will override any automatically assigned SRID.

Availability: 2.1.0

Examples

```
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)
```

See Also

ST_AsGDALRaster

11.4 Raster Accessors

11.4.1 ST_GeoReference

ST_GeoReference — Returns the georeference meta data in GDAL or ESRI format as commonly seen in a world file. Default is GDAL.

Synopsis

text **ST_GeoReference**(raster rast, text format=GDAL);

Description

Returns the georeference meta data including carriage return in GDAL or ESRI format as commonly seen in a world file. Default is GDAL if no type specified. type is string 'GDAL' or 'ESRI'.

Difference between format representations is as follows:

GDAL:

```
scalex
skewy
skewx
scaley
upperleftx
upperlefty
```

ESRI:

```
scalex
skewy
skewx
scaley
upperleftx + scalex*0.5
upperlefty + scaley*0.5
```

Examples

```
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
```

See Also

ST_SetGeoReference, ST_ScaleX, ST_ScaleY

11.4.2 ST_Height

ST_Height — Returns the height of the raster in pixels.

Synopsis

integerST_Height(raster rast);

Description

Returns the height of the raster.

Examples

```
SELECT rid, ST_Height(rast) As rastheight
FROM dummy_rast;

 rid | rastheight
-----+------------
   1 |         20
   2 |          5
```

See Also

ST_Width

11.4.3 ST_IsEmpty

ST_IsEmpty — Returns true if the raster is empty (width = 0 and height = 0). Otherwise, returns false.

Synopsis

booleanST_IsEmpty(raster rast);

Description

Returns the Y component of the pixel height in units of coordinate reference system. May be negative. Refer to World File for more details.

Changed: 2.0.0. In WKTRaster versions this was called ST_PixelSizeY.

Examples

```
SELECT rid, ST_ScaleY(rast) As rastpixheight
FROM dummy_rast;

 rid | rastpixheight
-----+---------------
   1 |             3
   2 |         -0.05
```

See Also

ST_Height

11.4.11 ST_RasterToWorldCoord

ST_RasterToWorldCoord — Returns the raster's upper left corner as geometric X and Y (longitude and latitude) given a column and row. Column and row starts at 1.

Synopsis

record **ST_RasterToWorldCoord**(raster rast, integer xcolumn, integer yrow);

Description

Returns the upper left corner as geometric X and Y (longitude and latitude) given a column and row. Returned X and Y are in geometric units of the georeferenced raster. Numbering of column and row starts at 1 but if either parameter is passed a zero, a negative number or a number greater than the respective dimension of the raster, it will return coordinates outside of the raster assuming the raster's grid is applicable outside the raster's bounds.

Availability: 2.1.0

Examples

```
-- 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
```

See Also

ST_RasterToWorldCoordX, ST_RasterToWorldCoordY, ST_SetSkew

11.4.12 ST_RasterToWorldCoordX

ST_RasterToWorldCoordX — Returns the geometric X coordinate upper left of a raster, column and row. Numbering of columns and rows starts at 1.

Synopsis

float8 **ST_RasterToWorldCoordX**(raster rast, integer xcolumn);
float8 **ST_RasterToWorldCoordX**(raster rast, integer xcolumn, integer yrow);

Description

Returns the upper left X coordinate of a raster column row in geometric units of the georeferenced raster. Numbering of columns and rows starts at 1 but if you pass in a negative number or number higher than number of columns in raster, it will give you coordinates outside of the raster file to left or right with the assumption that the skew and pixel sizes are same as selected raster.

Note
For non-skewed rasters, providing the X column is sufficient. For skewed rasters, the georeferenced coordinate is a function of the ST_ScaleX and ST_SkewX and row and column. An error will be raised if you give just the X column for a skewed raster.

Changed: 2.1.0 In prior versions, this was called ST_Raster2WorldCoordX

Examples

```
-- 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
```

See Also

ST_ScaleX, ST_RasterToWorldCoordY, ST_SetSkew, ST_SkewX

11.4.13 ST_RasterToWorldCoordY

ST_RasterToWorldCoordY — Returns the geometric Y coordinate upper left corner of a raster, column and row. Numbering of columns and rows starts at 1.

Synopsis

float8 **ST_RasterToWorldCoordY**(raster rast, integer yrow);
float8 **ST_RasterToWorldCoordY**(raster rast, integer xcolumn, integer yrow);

Description

Returns the upper left Y coordinate of a raster column row in geometric units of the georeferenced raster. Numbering of columns and rows starts at 1 but if you pass in a negative number or number higher than number of columns/rows in raster, it will give you coordinates outside of the raster file to left or right with the assumption that the skew and pixel sizes are same as selected raster tile.

Note
For non-skewed rasters, providing the Y column is sufficient. For skewed rasters, the georeferenced coordinate is a function of the ST_ScaleY and ST_SkewY and row and column. An error will be raised if you give just the Y row for a skewed raster.

Changed: 2.1.0 In prior versions, this was called ST_Raster2WorldCoordY

Examples

```
-- 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;
```

Synopsis

float8 **ST_SkewX**(raster rast);

Description

Returns the georeference X skew (or rotation parameter). Refer to World File for more details.

Examples

```
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
```

See Also

ST_GeoReference, ST_SkewY, ST_SetSkew

11.4.16 ST_SkewY

ST_SkewY — Returns the georeference Y skew (or rotation parameter).

Synopsis

float8 **ST_SkewY**(raster rast);

Description

Returns the georeference Y skew (or rotation parameter). Refer to World File for more details.

Examples

```
SELECT rid, ST_SkewX(rast) As skewx, ST_SkewY(rast) As skewy,
    ST_GeoReference(rast) as georef
FROM dummy_rast;

 rid | skewx | skewy |       georef
```

Examples

```
SELECt rid, ST_UpperLeftX(rast) As ulx
FROM dummy_rast;

 rid |    ulx
-----+------------
   1 |        0.5
   2 | 3427927.75
```

See Also

ST_UpperLeftY, ST_GeoReference, Box3D

11.4.20 ST_UpperLeftY

ST_UpperLeftY — Returns the upper left Y coordinate of raster in projected spatial ref.

Synopsis

float8 **ST_UpperLeftY**(raster rast);

Description

Returns the upper left Y coordinate of raster in projected spatial ref.

Examples

```
SELECT rid, ST_UpperLeftY(rast) As uly
FROM dummy_rast;

 rid |   uly
-----+---------
   1 |     0.5
   2 | 5793244
```

See Also

ST_UpperLeftX, ST_GeoReference, Box3D

11.4.21 ST_Width

ST_Width — Returns the width of the raster in pixels.

Synopsis

integer **ST_Width**(raster rast);

Description

Returns the width of the raster in pixels.

Synopsis

integer **ST_WorldToRasterCoordX**(raster rast, geometry pt);
integer **ST_WorldToRasterCoordX**(raster rast, double precision xw);
integer **ST_WorldToRasterCoordX**(raster rast, double precision xw, double precision yw);

Description

Returns the column in the raster of the point geometry (pt) or a X and Y world coordinate (xw, yw). A point, or (both xw and yw world coordinates are required if a raster is skewed). If a raster is not skewed then xw is sufficient. World coordinates are in the spatial reference coordinate system of the raster.

Changed: 2.1.0 In prior versions, this was called ST_World2RasterCoordX

Examples

```
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
```

See Also

ST_RasterToWorldCoordX, ST_RasterToWorldCoordY, ST_SRID

11.4.24 ST_WorldToRasterCoordY

ST_WorldToRasterCoordY — Returns the row in the raster of the point geometry (pt) or a X and Y world coordinate (xw, yw) represented in world spatial reference system of raster.

Synopsis

integer **ST_WorldToRasterCoordY**(raster rast, geometry pt);
integer **ST_WorldToRasterCoordY**(raster rast, double precision xw);
integer **ST_WorldToRasterCoordY**(raster rast, double precision xw, double precision yw);

Description

Returns the row in the raster of the point geometry (pt) or a X and Y world coordinate (xw, yw). A point, or (both xw and yw world coordinates are required if a raster is skewed). If a raster is not skewed then xw is sufficient. World coordinates are in the spatial reference coordinate system of the raster.

Changed: 2.1.0 In prior versions, this was called ST_World2RasterCoordY

Examples

```
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
```

See Also

ST_RasterToWorldCoordX, ST_RasterToWorldCoordY, ST_SRID

11.5 Raster Band Accessors

11.5.1 ST_BandMetaData

ST_BandMetaData — Returns basic meta data for a specific raster band. band num 1 is assumed if none-specified.

Synopsis

(1) record **ST_BandMetaData**(raster rast, integer band=1);
(2) record **ST_BandMetaData**(raster rast, integer[] band);

Description

Returns basic meta data about a raster band. Columns returned: pixeltype, nodatavalue, isoutdb, path, outdbbandnum, filesize, filetimestamp.

Note
If raster contains no bands then an error is thrown.

Note
If band has no NODATA value, nodatavalue are NULL.

Note
If isoutdb is False, path, outdbbandnum, filesize and filetimestamp are NULL. If outdb access is disabled, filesize and filetimestamp will also be NULL.

Enhanced: 2.5.0 to include *outdbbandnum*, *filesize* and *filetimestamp* for outdb rasters.

Description

Returns the value that represents no data for the band

Examples

```
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
```

See Also

ST_NumBands

11.5.3 ST_BandIsNoData

ST_BandIsNoData — Returns true if the band is filled with only nodata values.

Synopsis

boolean **ST_BandIsNoData**(raster rast, integer band, boolean forceChecking=true);
boolean **ST_BandIsNoData**(raster rast, boolean forceChecking=true);

Description

Returns true if the band is filled with only nodata values. Band 1 is assumed if not specified. If the last argument is TRUE, the entire band is checked pixel by pixel. Otherwise, the function simply returns the value of the isnodata flag for the band. The default value for this parameter is FALSE, if not specified.

Availability: 2.0.0

Note
If the flag is dirty (this is, the result is different using TRUE as last parameter and not using it) you should update the raster to set this flag to true, by using ST_SetBandIsNodata(), or ST_SetBandNodataValue() with TRUE as last argument. See ST_SetBandIsNoData.

Examples

```
-- 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
```

See Also

ST_BandNoDataValue, ST_NumBands, ST_SetBandNoDataValue, ST_SetBandIsNoData

11.5.4 ST_BandPath

ST_BandPath — Returns system file path to a band stored in file system. If no bandnum specified, 1 is assumed.

Synopsis

text **ST_BandPath**(raster rast, integer bandnum=1);

Description

Returns system file path to a band. Throws an error if called with an in db band.

Examples

See Also

11.5.5 ST_BandFileSize

ST_BandFileSize — Returns the file size of a band stored in file system. If no bandnum specified, 1 is assumed.

Synopsis

bigint **ST_BandFileSize**(raster rast, integer bandnum=1);

Description

Returns the file size of a band stored in file system. Throws an error if called with an in db band, or if outdb access is not enabled.

This function is typically used in conjunction with ST_BandPath() and ST_BandFileTimestamp() so a client can determine if the filename of a outdb raster as seen by it is the same as the one seen by the server.

Availability: 2.5.0

Examples

```
SELECT ST_BandFileSize(rast,1) FROM dummy_rast WHERE rid = 1;

 st_bandfilesize
-----------------
          240574
```

11.5.6 ST_BandFileTimestamp

ST_BandFileTimestamp — Returns the file timestamp of a band stored in file system. If no bandnum specified, 1 is assumed.

Synopsis

bigint **ST_BandFileTimestamp**(raster rast, integer bandnum=1);

Description

Returns the file timestamp (number of seconds since Jan 1st 1970 00:00:00 UTC) of a band stored in file system. Throws an error if called with an in db band, or if outdb access is not enabled.

This function is typically used in conjunction with ST_BandPath() and ST_BandFileSize() so a client can determine if the filename of a outdb raster as seen by it is the same as the one seen by the server.

Availability: 2.5.0

Examples

```
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
```

See Also

ST_NumBands

11.6 Raster Pixel Accessors and Setters

11.6.1 ST_PixelAsPolygon

ST_PixelAsPolygon — Returns the polygon geometry that bounds the pixel for a particular row and column.

Synopsis

geometry **ST_PixelAsPolygon**(raster rast, integer columnx, integer rowy);

Description

Returns the polygon geometry that bounds the pixel for a particular row and column.

Availability: 2.0.0

Examples

```
-- 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, ..
```

See Also

ST_DumpAsPolygons, ST_PixelAsPolygons, ST_PixelAsPoint, ST_PixelAsPoints, ST_PixelAsCentroid, ST_PixelAsCentroids, ST_Intersection, ST_AsText

11.6.2

11.6.3 ST_PixelAsPoint

ST_PixelAsPoint — Returns a point geometry of the pixel's upper-left corner.

Synopsis

geometry **ST_PixelAsPoint**(raster rast, integer columnx, integer rowy);

Description

Returns a point geometry of the pixel's upper-left corner.

Availability: 2.1.0

Examples

```
SELECT ST_AsText(ST_PixelAsPoint(rast, 1, 1)) FROM dummy_rast WHERE rid = 1;

   st_astext
----------------
 POINT(0.5 0.5)
```

See Also

ST_DumpAsPolygons, ST_PixelAsPolygon, ST_PixelAsPolygons, ST_PixelAsPoints, ST_PixelAsCentroid, ST_PixelAsCentroids

11.6.4 ST_PixelAsPoints

ST_PixelAsPoints — Returns a point geometry for each pixel of a raster band along with the value, the X and the Y raster coordinates of each pixel. The coordinates of the point geometry are of the pixel's upper-left corner.

Synopsis

setof record **ST_PixelAsPoints**(raster rast, integer band=1, boolean exclude_nodata_value=TRUE);

Description

Returns a point geometry for each pixel of a raster band along with the value, the X and the Y raster coordinates of each pixel. The coordinates of the point geometry are of the pixel's upper-left corner.

Return record format: *geom* geometry, *val* double precision, *x* integer, *y* integers.

Note

When *exclude_nodata_value* = TRUE, only those pixels whose values are not NODATA are returned as points.

Availability: 2.1.0

Changed: 2.1.1 Changed behavior of exclude_nodata_value.

```
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)
```

See Also

ST_DumpAsPolygons, ST_PixelAsPolygon, ST_PixelAsPolygons, ST_PixelAsPoint, ST_PixelAsPoints, ST_PixelAsCentroid

11.6.7 ST_Value

ST_Value — Returns the value of a given band in a given columnx, rowy pixel or at a particular geometric point. Band numbers start at 1 and assumed to be 1 if not specified. If `exclude_nodata_value` is set to false, then all pixels include `nodata` pixels are considered to intersect and return value. If `exclude_nodata_value` is not passed in then reads it from metadata of raster.

Synopsis

double precision **ST_Value**

```
-- 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)))
```

See Also

ST_SetValue, ST_DumpAsPolygons, ST_NumBands, ST_PixelAsPolygon, ST_ScaleX, ST_ScaleY, ST_UpperLeftX, ST_UpperLeftY, ST_SRID, ST_AsText, ST_Point, ST_MakeEnvelope, ST_Intersects, ST_Intersection

11.6.8 ST_NearestValue

ST_NearestValue — Returns the nearest non-NODATA value of a given band's pixel specified by a columnx and rowy or a geometric point expressed in the same spatial reference coordinate system as the raster.

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);

Description

Returns the nearest non-NODATA value of a given band in a given columnx, rowy pixel or at a specific geometric point. If the columnx, rowy pixel or the pixel at the specified geometric point is NODATA, the function will find the nearest pixel to the columnx, rowy pixel or geometric point whose value is not NODATA.

Band numbers start at 1 and bandnum is assumed to be 1 if not specified. If exclude_nodata_value is set to false, then all pixels include nodata pixels are considered to intersect and return value. If exclude_nodata_value is not passed in then reads it from metadata of raster.

Availability: 2.1.0

Note
ST_NearestValue is a drop-in replacement for ST_Value.

Examples

```
-- 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
```

See Also

ST_Neighborhood, ST_Value

11.6.9 ST_SetZ

ST_SetZ — Returns a geometry with the same X/Y coordinates as the input geometry, and values from the raster copied into the Z dimension using the requested resample algorithm.

Synopsis

geometry **ST_SetZ**(raster rast, geometry geom, text resample=nearest, integer band=1);

Description

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

Examples

```
--
-- 2x2 test raster with values
--
-- 10 50
-- 40 20
--
```

```
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)
```

See Also

ST_Value, ST_SetM

11.6.10 ST_SetM

ST_SetM — Returns a geometry with the same X/Y coordinates as the input geometry, and values from the raster copied into the M dimension using the requested resample algorithm.

Synopsis

geometry **ST_SetM**(raster rast, geometry geom, text resample=nearest, integer band=1);

Description

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

Examples

```
--
-- 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)
```

See Also

ST_Value, ST_SetZ

11.6.11 ST_Neighborhood

ST_Neighborhood — Returns a 2-D double precision array of the non-NODATA values around a given band's pixel specified by either a columnX and rowY or a geometric point expressed in the same spatial reference coordinate system as the raster.

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

Description

Returns a 2-D double precision array of the non-NODATA values around a given band's pixel specified by either a columnX and rowY or a geometric point expressed in the same spatial reference coordinate system as the raster. The distanceX and distanceY parameters define the number of pixels around the specified pixel in the X and Y axes, e.g. I want all values within 3 pixel distance along the X axis and 2 pixel distance along the Y axis around my pixel of interest. The center value of the 2-D array will be the value at the pixel specified by the columnX and rowY or the geometric point.

Band numbers start at 1 and bandnum is assumed to be 1 if not specified. If exclude_nodata_value is set to false, then all pixels include nodata pixels are considered to intersect and return value. If exclude_nodata_value is not passed in then reads it from metadata of raster.

Note
The number of elements along each axis of the returning 2-D array is 2 * (distanceX|distanceY) + 1. So for a distanceX and distanceY of 1, the returning array will be 3x3.

Note
The 2-D array output can be passed to any of the raster processing builtin functions, e.g. ST_Min4ma, ST_Sum4ma, ST_Mean4ma.

Availability: 2.1.0

Examples

```
-- 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}}
```

See Also

ST_NearestValue, ST_Min4ma, ST_Max4ma, ST_Sum4ma, ST_Mean4ma, ST_Range4ma, ST_Distinct4ma, ST_StdDev4ma

11.6.12 ST_SetValue

ST_SetValue — Returns modified raster resulting from setting the value of a given band in a given columnx, rowy pixel or the pixels that intersect a particular geometry. Band numbers start at 1 and assumed to be 1 if not specified.

Synopsis

raster **ST_SetValue**(raster rast, integer bandnum, geometry geom, double precision newvalue);
raster **ST_SetValue**(raster rast, geometry geom, double precision newvalue);
raster **ST_SetValue**(raster rast, integer bandnum, integer columnx, integer rowy, double precision newvalue);
raster **ST_SetValue**(raster rast, integer columnx, integer rowy, double precision newvalue);

Description

Returns modified raster resulting from setting the specified pixels' values to new value for the designated band given the raster's row and column or a geometry. If no band is specified, then band 1 is assumed.

Enhanced: 2.1.0 Geometry variant of ST_SetValue() now supports any geometry type, not just point. The geometry variant is a wrapper around the geomval[] variant of ST_SetValues()

Examples

```
                -- 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   ;
```

See Also

ST_Value, ST_DumpAsPolygons

11.6.13 ST_SetValues

ST_SetValues — Returns modified raster resulting from setting the values of a given band.

Synopsis

raster **ST_SetValues**(raster rast, integer nband, integer columnx, integer rowy, double precision[][] newvalueset, boolean[][] noset=NULL, boolean keepnodata=FALSE);
raster **ST_SetValues**(raster rast, integer nband, integer columnx, integer rowy, double precision[][] newvalueset, double precision nosetvalue, boolean keepnodata=FALSE);
raster **ST_SetValues**(raster rast, integer nband, integer columnx, integer rowy, integer width, integer height, double precision newvalue, boolean keepnodata=FALSE);
raster **ST_SetValues**(raster rast, integer columnx, integer rowy, integer width, integer height, double precision newvalue, boolean keepnodata=FALSE);
raster **ST_SetValues**(raster rast, integer nband, geomval[] geomvalset, boolean keepnodata=FALSE);

Description

Returns modified raster resulting from setting specified pixels to new value(s) for the designated band. `columnx` and `rowy` are 1-indexed.

If `keepnodata` is TRUE, those pixels whose values are NODATA will not be set with the corresponding value in `newvalueset`.

For Variant 1, the specific pixels to be set are determined by the `columnx`, `rowy` pixel coordinates and the dimensions of the `newvalueset` array. `noset` can be used to prevent pixels with values present in `newvalueset` from being set (due to PostgreSQL not permitting ragged/jagged arrays). See example Variant 1.

Variant 2 is like Variant 1 but with a simple double precision `nosetvalue` instead of a boolean `noset` array. Elements in `newvalueset` with the `nosetvalue` value with be skipped. See example Variant 2.

For Variant 3, the specific pixels to be set are determined by the `columnx`, `rowy` pixel coordinates, `width` and `height`. See example Variant 3.

Variant 4 is the same as Variant 3 with the exception that it assumes that the first band's pixels of `rast` will be set.

For Variant 5, an array of geomval is used to determine the speci c pixels to be set. If all the geometries in the array are type POINT or MULTIPOINT, the function uses a shortcut where the longitude and latitude of each point is used to set a pix directly. Otherwise, the geometries are converted to rasters and then iterated through in one pass. See example Variant 5.

Availability: 2.1.0

Examples: Variant 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 |
```

```
ORDER BY 1, 2;

 x | y | val
---+---+-----
 1 | 1 |   9
 1 | 2 |   1
 1 | 3 |   9
 2 | 1 |   9
 2 | 2 |
 2 | 3 |   9
 3 | 1 |   9
 3 | 2 |   9
 3 | 3 |   9
```

```
/*
The ST_SetValues() does the following...

+ - + - + - +          + - + - + - +
|   | 1 | 1 |          |   | 9 | 9 |
+ - + - + - +          + - + - + - +
| 1 | 1 | 1 |    =>    | 1 |   | 9 |
+ - + - + - +          + - + - + - +
| 1 | 1 | 1 |          | 9 | 9 | 9 |
+ - + - + - +          + - + - + - +
*/
SELECT
    (poly).x,
    (poly).y,
    (poly).val
FROM (
SELECT
    ST_PixelAsPolygons(
        ST_SetValues(
            ST_SetValue(
                ST_AddBand(
                    ST_MakeEmptyRaster(3, 3, 0, 0, 1, -1, 0, 0, 0),
                    1, '8BUI', 1, 0
                ),
                1, 1, 1, NULL
            ),
            1, 1, 1,
                ARRAY[[9, 9, 9], [9, NULL, 9], [9, 9, 9]]::double precision[][],
                ARRAY[[false], [true]]::boolean[][],
                TRUE
        )
    ) AS poly
) foo
ORDER BY 1, 2;

 x | y | val
---+---+-----
 1 | 1 |
 1 | 2 |   1
 1 | 3 |   9
 2 | 1 |   9
 2 | 2 |
 2 | 3 |   9
 3 | 1 |   9
 3 | 2 |   9
 3 | 3 |   9
```

Examples: Variant 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
```

Examples: Variant 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
```

Examples: Variant 5

```
WITH foo AS (
    SELECT 1 AS rid, ST_AddBand(ST_MakeEmptyRaster(5, 5, 0, 0, 1, -1, 0, 0, 0), 1, '8BUI',  ↩
        0, 0) AS rast
), bar AS (
    SELECT 1 AS gid, 'SRID=0;POINT(2.5 -2.5)'::geometry geom UNION ALL
```

```
    SELECT 2 AS gid, 'SRID=0;POLYGON((1 -1, 4 -1, 4 -4, 1 -4, 1 -1))'::geometry geom UNION ↩
        ALL
    SELECT 3 AS gid, 'SRID=0;POLYGON((0 0, 5 0, 5 -1, 1 -1, 1 -4, 0 -4, 0 0))'::geometry ↩
        geom UNION ALL
    SELECT 4 AS gid, 'SRID=0;MULTIPOINT(0 0, 4 4, 4 -4)'::geometry
)
SELECT
    rid, gid, ST_DumpValues(ST_SetValue(rast, 1, geom, gid))
FROM foo t1
CROSS JOIN bar t2
ORDER BY rid, gid;

 rid | gid |                                                                st_dumpvalues
-----+-----+-----------------------------------------------------------------------------------------

   1 |   1 | (1,"{{NULL,NULL,NULL,NULL,NULL},{NULL,NULL,NULL,NULL,NULL},{NULL,NULL,1,NULL, ↩
       NULL},{NULL,NULL,NULL,NULL,NULL},{NULL,NULL,NULL,NULL,NULL}}")
   1 |   2 | (1,"{{NULL,NULL,NULL,NULL,NULL},{NULL,2,2,2,NULL},{NULL,2,2,2,NULL},{NULL ↩
       ,2,2,2,NULL},{NULL,NULL,NULL,NULL,NULL}}")
   1 |   3 | (1,"{{3,3,3,3,3},{3,NULL,NULL,NULL,NULL},{3,NULL,NULL,NULL,NULL},{3,NULL,NULL, ↩
       NULL,NULL},{NULL,NULL,NULL,NULL,NULL}}")
   1 |   4 | (1,"{{4,NULL,NULL,NULL,NULL},{NULL,NULL,NULL,NULL,NULL},{NULL,NULL,NULL,NULL, ↩
       NULL},{NULL,NULL,NULL,NULL,NULL},{NULL,NULL,NULL,NULL,4}}")
(4 rows)
```

The following shows that geomvals later in the array can overwrite prior geomvals

```
WITH foo AS (
    SELECT 1 AS rid, ST_AddBand(ST_MakeEmptyRaster(5, 5, 0, 0, 1, -1, 0, 0, 0), 1, '8BUI', ↩
        0, 0) AS rast
), bar AS (
    SELECT 1 AS gid, 'SRID=0;POINT(2.5 -2.5)'::geometry geom UNION ALL
    SELECT 2 AS gid, 'SRID=0;POLYGON((1 -1, 4 -1, 4 -4, 1 -4, 1 -1))'::geometry geom UNION ↩
        ALL
    SELECT 3 AS gid, 'SRID=0;POLYGON((0 0, 5 0, 5 -1, 1 -1, 1 -4, 0 -4, 0 0))'::geometry ↩
        geom UNION ALL
    SELECT 4 AS gid, 'SRID=0;MULTIPOINT(0 0, 4 4, 4 -4)'::geometry
)
SELECT
    t1.rid, t2.gid, t3.gid, ST_DumpValues(ST_SetValues(rast, 1, ARRAY[ROW(t2.geom, t2.gid), ↩
         ROW(t3.geom, t3.gid)]::geomval[]))
FROM foo t1
CROSS JOIN bar t2
CROSS JOIN bar t3
WHERE t2.gid = 1
    AND t3.gid = 2
ORDER BY t1.rid, t2.gid, t3.gid;

 rid | gid | gid |                                                    st_dumpvalues
-----+-----+-----+-----------------------------------------------------------------------------------

   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)
```

This example is the opposite of the prior example

```
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)
```

See Also

ST_Value, ST_SetValue, ST_PixelAsPolygons

11.6.14 ST_DumpValues

ST_DumpValues — Get the values of the specified band as a 2-dimension array.

Synopsis

setof record **ST_DumpValues**(raster rast , integer[] nband=NULL , boolean exclude_nodata_value=true);
double precision[][] **ST_DumpValues**(raster rast , integer nband , boolean exclude_nodata_value=true);

Description

Get the values of the specified band as a 2-dimension array (first index is row, second is column). If `nband` is NULL or not provided, all raster bands are processed.

Availability: 2.1.0

Examples

```
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)
```

See Also

ST_Value, ST_SetValue, ST_SetValues

11.6.15 ST_PixelOfValue

ST_PixelOfValue — Get the columnx, rowy coordinates of the pixel whose value equals the search value.

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);

Description

Get the columnx, rowy coordinates of the pixel whose value equals the search value. If no band is specified, then band 1 is assumed.

Availability: 2.1.0

Synopsis

raster **ST_SetGeoReference**(raster rast, text georefcoords, text format=GDAL);
raster **ST_SetGeoReference**(raster rast, double precision upperleftx, double precision upperlefty, double precision scalex, double precision scaley, double precision skewx, double precision skewy);

Description

Set Georeference 6 georeference parameters in a single call. Accepts inputs in 'GDAL' or 'ESRI' format. Default is GDAL. If 6 coordinates are not provided will return null.

Difference between format representations is as follows:

GDAL:

```
scalex skewy skewx scaley upperleftx upperlefty
```

ESRI:

```
scalex skewy skewx scaley upperleftx + scalex*0.5 upperlefty + scaley*0.5
```

Note
If the raster has out-db bands, changing the georeference may result in incorrect access of the band's externally stored data.

Enhanced: 2.1.0 Addition of ST_SetGeoReference(raster, double precision, ...) variant

Examples

```
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
```

Description

Sets the georeference X and Y skew (or rotation parameter). If only one is passed in, sets X and Y to the same value. Refer to World File for more details.

Examples

```
-- 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
```

See Also

ST_GeoReference, ST_SetGeoReference, ST_SkewX, ST_SkewY

11.7.5 ST_SetSRID

ST_SetSRID — Sets the SRID of a raster to a particular integer srid defined in the spatial_ref_sys table.

Synopsis

raster **ST_SetSRID**(raster rast, integer srid);

Description

Sets the SRID on a raster to a particular integer value.

Note
This function does not transform the raster in any way - it simply sets meta data defining the spatial ref of the coordinate reference system that it's currently in. Useful for transformations later.

See Also

Section 4.5, ST_SRID

11.7.6 ST_SetUpperLeft

ST_SetUpperLeft — Sets the value of the upper left corner of the pixel of the raster to projected X and Y coordinates.

Synopsis

raster **ST_SetUpperLeft**(raster rast, double precision x, double precision y);

Description

Set the value of the upper left corner of raster to the projected X and Y coordinates

Examples

```
SELECT ST_SetUpperLeft(rast,-71.01,42.37)
FROM dummy_rast
WHERE rid = 2;
```

See Also

ST_UpperLeftX, ST_UpperLeftY

11.7.7 ST_Resample

ST_Resample — Resample a raster using a specified resampling algorithm, new dimensions, an arbitrary grid corner and a set of raster georeferencing attributes defined or borrowed from another raster.

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);

11.7.8 ST_Rescale

ST_Rescale — Resample a raster by adjusting only its scale (or pixel size). New pixel values are computed using the NearestNeighbor (english or american spelling), Bilinear, Cubic, CubicSpline, Lanczos, Max or Min resampling algorithm. Default is NearestNeighbor.

Synopsis

raster **ST_Rescale**(raster rast, double precision scalexy, text algorithm=NearestNeighbor, double precision maxerr=0.125);
raster **ST_Rescale**(raster rast, double precision scalex, double precision scaley, text algorithm=NearestNeighbor, double precision maxerr=0.125);

Description

Resample a raster by adjusting only its scale (or pixel size). New pixel values are computed using one of the following resampling algorithms:

- NearestNeighbor (english or american spelling)
- Bilinear
- Cubic
- CubicSpline
- Lanczos
- Max
- Min

The default is NearestNeighbor which is the fastest but results in the worst interpolation.

`scalex` and `scaley` define the new pixel size. scaley must often be negative to get well oriented raster.

When the new scalex or scaley is not a divisor of the raster width or height, the extent of the resulting raster is expanded to encompass the extent of the provided raster. If you want to be sure to retain exact input extent see ST_Resize

`maxerr` is the threshold for transformation approximation by the resampling algorithm (in pixel units). A default of 0.125 is used if no `maxerr` is specified, which is the same value used in GDAL gdalwarp utility. If set to zero, no approximation takes place.

Note
Refer to: GDAL Warp resampling methods for more details.

Note
ST_Rescale is different from ST_SetScale in that ST_SetScale do not resample the raster to match the raster extent. ST_SetScale only changes the metadata (or georeference) of the raster to correct an originally mis-specified scaling. ST_Rescale results in a raster having different width and height computed to fit the geographic extent of the input raster. ST_SetScale do not modify the width, nor the height of the raster.

Availability: 2.0.0 Requires GDAL 1.6.1+

Enhanced: 3.4.0 max and min resampling options added

Changed: 2.1.0 Works on rasters with no SRID

Examples

A simple example rescaling a raster from a pixel size of 0.001 degree to a pixel size of 0.0015 degree.

```
-- 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
```

See Also

ST_Resize, ST_Resample, ST_SetScale, ST_ScaleX, ST_ScaleY, ST_Transform

11.7.9 ST_Reskew

ST_Reskew — Resample a raster by adjusting only its skew (or rotation parameters). New pixel values are computed using the NearestNeighbor (english or american spelling), Bilinear, Cubic, CubicSpline or Lanczos resampling algorithm. Default is NearestNeighbor.

Synopsis

raster **ST_Reskew**(raster rast, double precision skewxy, text algorithm=NearestNeighbor, double precision maxerr=0.125);
raster **ST_Reskew**(raster rast, double precision skewx, double precision skewy, text algorithm=NearestNeighbor, double precision maxerr=0.125);

Description

Resample a raster by adjusting only its skew (or rotation parameters). New pixel values are computed using the NearestNeighbor (english or american spelling), Bilinear, Cubic, CubicSpline or Lanczos resampling algorithm. The default is NearestNeighbor which is the fastest but results in the worst interpolation.

`skewx` and `skewy` define the new skew.

The extent of the new raster will encompass the extent of the provided raster.

A maxerror percent of 0.125 if no `maxerr` is specified.

Note
Refer to: GDAL Warp resampling methods for more details.

Note
ST_Reskew is different from ST_SetSkew in that ST_SetSkew do not resample the raster to match the raster extent. ST_SetSkew only changes the metadata (or georeference) of the raster to correct an originally mis-specified skew. ST_Reskew results in a raster having different width and height computed to fit the geographic extent of the input raster. ST_SetSkew do not modify the width, nor the height of the raster.

Availability: 2.0.0 Requires GDAL 1.6.1+

Changed: 2.1.0 Works on rasters with no SRID

Examples

A simple example reskewing a raster from a skew of 0.0 to a skew of 0.0015.

```
-- 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
```

See Also

ST_Resample, ST_Rescale, ST_SetSkew, ST_SetRotation, ST_SkewX, ST_SkewY, ST_Transform

11.7.10 ST_SnapToGrid

ST_SnapToGrid — Resample a raster by snapping it to a grid. New pixel values are computed using the NearestNeighbor (english or american spelling), Bilinear, Cubic, CubicSpline or Lanczos resampling algorithm. Default is 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);

Description

Resample a raster by snapping it to a grid defined by an arbitrary pixel corner (gridx & gridy) and optionally a pixel size (scalex & scaley). New pixel values are computed using the NearestNeighbor (english or american spelling), Bilinear, Cubic, CubicSpline or Lanczos resampling algorithm. The default is NearestNeighbor which is the fastest but results in the worst interpolation.

`gridx` and `gridy` define any arbitrary pixel corner of the new grid. This is not necessarily the upper left corner of the new raster and it does not have to be inside or on the edge of the new raster extent.

You can optionally define the pixel size of the new grid with `scalex` and `scaley`.

The extent of the new raster will encompass the extent of the provided raster.

A maxerror percent of 0.125 if no `maxerr` is specified.

Note
Refer to: GDAL Warp resampling methods for more details.

Note
Use ST_Resample if you need more control over the grid parameters.

Availability: 2.0.0 Requires GDAL 1.6.1+

Changed: 2.1.0 Works on rasters with no SRID

Examples

A simple example snapping a raster to a slightly different grid.

```
-- 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
```

See Also

ST_Resample, ST_Rescale, ST_UpperLeftX, ST_UpperLeftY

11.7.11 ST_Resize

ST_Resize — Resize a raster to a new width/height

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);

Description

Resize a raster to a new width/height. The new width/height can be specified in exact number of pixels or a percentage of the raster's width/height. The extent of the the new raster will be the same as the extent of the provided raster.

New pixel values are computed using the NearestNeighbor (english or american spelling), Bilinear, Cubic, CubicSpline or Lanczos resampling algorithm. The default is NearestNeighbor which is the fastest but results in the worst interpolation.

Variant 1 expects the actual width/height of the output raster.

Variant 2 expects decimal values between zero (0) and one (1) indicating the percentage of the input raster's width/height.

Variant 3 takes either the actual width/height of the output raster or a textual percentage ("20%") indicating the percentage of the input raster's width/height.

Availability: 2.1.0 Requires GDAL 1.6.1+

Examples

```
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)
```

See Also

ST_Resample, ST_Rescale, ST_Reskew, ST_SnapToGrid

11.7.12 ST_Transform

ST_Transform — Reprojects a raster in a known spatial reference system to another known spatial reference system using specified resampling algorithm. Options are NearestNeighbor, Bilinear, Cubic, CubicSpline, Lanczos defaulting to NearestNeighbor.

Synopsis

raster **ST_Transform**(raster rast, integer srid, text algorithm=NearestNeighbor, double precision maxerr=0.125, double precision scalex, double precision scaley);
raster **ST_Transform**(raster rast, integer srid, double precision scalex, double precision scaley, text algorithm=NearestNeighbor, double precision maxerr=0.125);
raster **ST_Transform**(raster rast, raster alignto, text algorithm=NearestNeighbor, double precision maxerr=0.125);

Description

Reprojects a raster in a known spatial reference system to another known spatial reference system using specified pixel warping algorithm. Uses 'NearestNeighbor' if no algorithm is specified and maxerror percent of 0.125 if no maxerr is specified.

Algorithm options are: 'NearestNeighbor', 'Bilinear', 'Cubic', 'CubicSpline', and 'Lanczos'. Refer to: GDAL Warp resampling methods for more details.

ST_Transform is often confused with ST_SetSRID(). ST_Transform actually changes the coordinates of a raster (and resamples the pixel values) from one spatial reference system to another, while ST_SetSRID() simply changes the SRID identifier of the raster.

Unlike the other variants, Variant 3 requires a reference raster as `alignto`. The transformed raster will be transformed to the spatial reference system (SRID) of the reference raster and be aligned (ST_SameAlignment = TRUE) to the reference raster.

Note
If you find your transformation support is not working right, you may need to set the environment variable PROJSO to the .so or .dll projection library your PostGIS is using. This just needs to have the name of the file. So for example on windows, you would in Control Panel -> System -> Environment Variables add a system variable called `PROJSO` and set it to `libproj.dll` (if you are using proj 4.6.1). You'll have to restart your PostgreSQL service/daemon after this change.

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.

Availability: 2.0.0 Requires GDAL 1.6.1+

Enhanced: 2.1.0 Addition of ST_Transform(rast, alignto) variant

Examples

```
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
```

original mass state plane meters (mass_stm)

After transform to wgs 84 long lat (wgs_84)

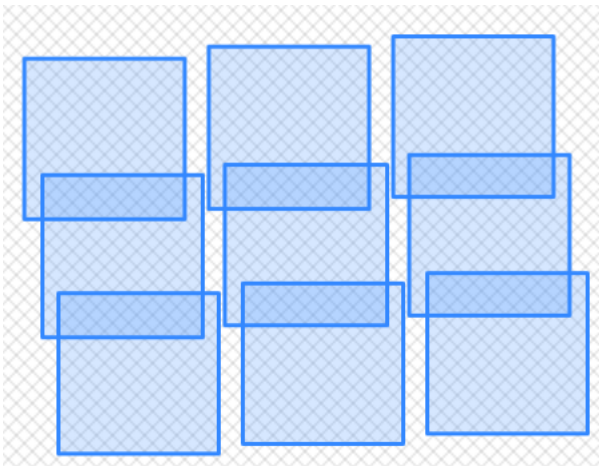
After transform to wgs 84 long lat with bilinear algorithm instead of NN default (wgs_84_bilin)

Examples: Variant 3

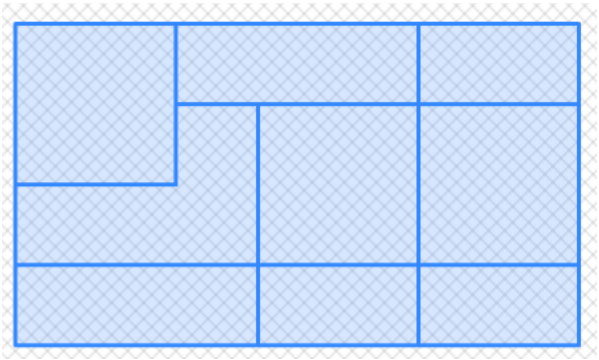
The following shows the difference between using ST_Transform(raster, srid) and ST_Transform(raster, alignto)

```
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

See Also

ST_Transform, ST_SetSRID

11.8 Raster Band Editors

11.8.1 ST_SetBandNoDataValue

ST_SetBandNoDataValue — Sets the value for the given band that represents no data. Band 1 is assumed if no band is specified. To mark a band as having no nodata value, set the nodata value = NULL.

Synopsis

raster **ST_SetBandNoDataValue**(raster rast, double precision nodatavalue);
raster **ST_SetBandNoDataValue**(raster rast, integer band, double precision nodatavalue, boolean forcechecking=false);

Description

Sets the value that represents no data for the band. Band 1 is assumed if not specified. This will affect results from ST_Polygon, ST_DumpAsPolygons, and the ST_PixelAs...() functions.

Examples

```
-- 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;
```

See Also

ST_BandNoDataValue, ST_NumBands

11.8.2 ST_SetBandIsNoData

ST_SetBandIsNoData — Sets the isnodata flag of the band to TRUE.

Synopsis

raster **ST_SetBandIsNoData**(raster rast, integer band=1);

Description

Sets the isnodata flag for the band to true. Band 1 is assumed if not specified. This function should be called only when the flag is considered dirty. That is, when the result calling ST_BandIsNoData is different using TRUE as last argument and without using it

Availability: 2.0.0

Examples

```
-- 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
```

See Also

ST_BandNoDataValue, ST_NumBands, ST_SetBandNoDataValue, ST_BandIsNoData

11.8.3 ST_SetBandPath

ST_SetBandPath — Update the external path and band number of an out-db band

Synopsis

raster **ST_SetBandPath**(raster rast, integer band, text outdbpath, integer outdbindex, boolean force=false);

Description

Updates an out-db band's external raster file path and external band number.

```
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
```

See Also

Examples

```
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
```

See Also

ST_BandMetaData, ST_SetBandPath

11.9 Raster Band Statistics and Analytics

11.9.1 ST_Count

ST_Count — Returns the number of pixels in a given band of a raster or raster coverage. If no band is specified defaults to band 1. If exclude_nodata_value is set to true, will only count pixels that are not equal to the nodata value.

Synopsis

bigint **ST_Count**(raster rast, integer nband=1, boolean exclude_nodata_value=true);
bigint **ST_Count**(raster rast, boolean exclude_nodata_value);

Description

Returns the number of pixels in a given band of a raster or raster coverage. If no band is specified `nband` defaults to 1.

Note
If `exclude_nodata_value` is set to true, will only count pixels with value not equal to the `nodata` value of the raster. Set `exclude_nodata_value` to false to get count all pixels

Changed: 3.1.0 - The ST_Count(rastertable, rastercolumn, ...) variants removed. Use ST_CountAgg instead.

Availability: 2.0.0

Examples

```
--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
```

See Also

ST_CountAgg, ST_SummaryStats, ST_SetBandNoDataValue

11.9.2 ST_CountAgg

ST_CountAgg — Aggregate. Returns the number of pixels in a given band of a set of rasters. If no band is specified defaults to band 1. If exclude_nodata_value is set to true, will only count pixels that are not equal to the NODATA value.

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);

SETOF record **ST_Histogram**(raster rast, integer nband, integer bins, double precision[] width=NULL, boolean right=false);
SETOF record **ST_Histogram**(raster rast, integer nband, boolean exclude_nodata_value, integer bins, boolean right);
SETOF record **ST_Histogram**(raster rast, integer nband, integer bins, boolean right);

Description

Returns set of records consisting of min, max, count, percent for a given raster band for each bin. If no band is specified `nband` defaults to 1.

Note
By default only considers pixel values not equal to the `nodata` value . Set `exclude_nodata_value` to false to get count all pixels.

`width` width: an array indicating the width of each category/bin. If the number of bins is greater than the number of widths, the widths are repeated.

Example: 9 bins, widths are [a, b, c] will have the output be [a, b, c, a, b, c, a, b, c]

`bins` Number of breakouts -- this is the number of records you'll get back from the function if specified. If not specified then the number of breakouts is autocomputed.

`right` compute the histogram from the right rather than from the left (default). This changes the criteria for evaluating a value x from [a, b) to (a, b]

Changed: 3.1.0 Removed ST_Histogram(table_name, column_name) variant.

Availability: 2.0.0

Example: Single raster tile - compute histograms for bands 1, 2, 3 and autocompute bins

```
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
```

Example: Just band 2 but for 6 bins

```
SELECT (stats).*
FROM (SELECT rid, ST_Histogram(rast, 2,6) As stats
    FROM dummy_rast
     WHERE rid=2) As foo;

    min     |    max     | count | percent
------------+------------+-------+---------
         78 | 107.333333 |     9 |    0.36
 107.333333 | 136.666667 |     6 |    0.24
 136.666667 |        166 |     0 |       0
        166 | 195.333333 |     4 |    0.16
 195.333333 | 224.666667 |     1 |    0.04
 224.666667 |        254 |     5 |     0.2
(6 rows)

-- Same as previous but we explicitly control the pixel value range of each bin.
SELECT (stats).*
FROM (SELECT rid, ST_Histogram(rast, 2,6,ARRAY[0.5,1,4,100,5]) As stats
    FROM dummy_rast
     WHERE rid=2) As foo;

  min   |  max   | count | percent
-------+-------+-------+----------
    78 |  78.5 |     1 |     0.08
  78.5 |  79.5 |     1 |     0.04
  79.5 |  83.5 |     0 |        0
  83.5 | 183.5 |    17 |   0.0068
 183.5 | 188.5 |     0 |        0
 188.5 |   254 |     6 | 0.003664
(6 rows)
```

See Also

ST_Count, ST_SummaryStats, ST_SummaryStatsAgg

11.9.4 ST_Quantile

ST_Quantile — Compute quantiles for a raster or raster table coverage in the context of the sample or population. Thus, a value could be examined to be at the raster's 25%, 50%, 75% percentile.

Synopsis

SETOF record **ST_Quantile**(raster rast, integer nband=1, boolean exclude_nodata_value=true, double precision[] quantiles=NULL);
SETOF record **ST_Quantile**(raster rast, double precision[] quantiles);
SETOF record **ST_Quantile**(raster rast, integer nband, double precision[] quantiles);
double precision **ST_Quantile**(raster rast, double precision quantile);
double precision **ST_Quantile**(raster rast, boolean exclude_nodata_value, double precision quantile=NULL);
double precision **ST_Quantile**(raster rast, integer nband, double precision quantile);
double precision **ST_Quantile**(raster rast, integer nband, boolean exclude_nodata_value, double precision quantile);
double precision **ST_Quantile**(raster rast, integer nband, double precision quantile);

Description

Compute quantiles for a raster or raster table coverage in the context of the sample or population. Thus, a value could be examined to be at the raster's 25%, 50%, 75% percentile.

Note
If `exclude_nodata_value` is set to false, will also count pixels with no data.

Changed: 3.1.0 Removed ST_Quantile(table_name, column_name) variant.

Availability: 2.0.0

Examples

```
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
```

Example: Summarize pixels that intersect buildings of interest

This example took 574ms on PostGIS windows 64-bit with all of Boston Buildings and aerial Tiles (tiles each 150x150 pixels ~ 134,000 tiles), ~102,000 building records

```
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
```

Example: Raster coverage

```
-- 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
```

See Also

summarystats, ST_SummaryStatsAgg, ST_Count, ST_Clip

11.9.6 ST_SummaryStatsAgg

ST_SummaryStatsAgg — Aggregate. Returns summarystats consisting of count, sum, mean, stddev, min, max for a given raster band of a set of raster. Band 1 is assumed is no band is specified.

Synopsis

summarystats **ST_SummaryStatsAgg**(raster rast, integer nband, boolean exclude_nodata_value, double precision sample_percent);
summarystats **ST_SummaryStatsAgg**(raster rast, boolean exclude_nodata_value, double precision sample_percent);
summarystats **ST_SummaryStatsAgg**(raster rast, integer nband, boolean exclude_nodata_value);

Description

Returns summarystats consisting of count, sum, mean, stddev, min, max for a given raster band of a raster or raster coverage. If no band is specified `nband` defaults to 1.

Note
By default only considers pixel values not equal to the `NODATA` value. Set `exclude_nodata_value` to False to get count of all pixels.

Note
By default will sample all pixels. To get faster response, set `sample_percent` to value between 0 and 1

Availability: 2.2.0

Examples

```
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)
```

See Also

summarystats, ST_SummaryStats, ST_Count, ST_Clip

11.9.7 ST_ValueCount

ST_ValueCount — Returns a set of records containing a pixel band value and count of the number of pixels in a given band of a raster (or a raster coverage) that have a given set of values. If no band is specified defaults to band 1. By default nodata value pixels are not counted. and all other values in the pixel are output and pixel band values are rounded to the nearest integer.

Synopsis

SETOF record **ST_ValueCount**(raster rast, integer nband=1, boolean exclude_nodata_value=true, double precision[] searchvalues=NULL, double precision roundto=0, double precision OUT value, integer OUT count);
SETOF record **ST_ValueCount**(raster rast, integer nband, double precision[] searchvalues, double precision roundto=0, double precision OUT value, integer OUT count);
SETOF record **ST_ValueCount**(raster rast, double precision[] searchvalues, double precision roundto=0, double precision OUT value, integer OUT count);
bigint **ST_ValueCount**(raster rast, double precision searchvalue, double precision roundto=0);
bigint **ST_ValueCount**(raster rast, integer nband, boolean exclude_nodata_value, double precision searchvalue, double precision roundto=0);
bigint **ST_ValueCount**(raster rast, integer nband, double precision searchvalue, double precision roundto=0);
SETOF record **ST_ValueCount**(text rastertable, text rastercolumn, integer nband=1, boolean exclude_nodata_value=true, double precision[] searchvalues=NULL, double precision roundto=0, double precision OUT value, integer OUT count);
SETOF record **ST_ValueCount**(text rastertable, text rastercolumn, double precision[] searchvalues, double precision roundto=0, double precision OUT value, integer OUT count);
SETOF record **ST_ValueCount**(text rastertable, text rastercolumn, integer nband, double precision[] searchvalues, double precision roundto=0, double precision OUT value, integer OUT count);
bigint**ST_ValueCount**(text rastertable, text rastercolumn, integer nband, boolean exclude_nodata_value, double precision searchvalue, double precision roundto=0);
bigint **ST_ValueCount**(text rastertable, text rastercolumn, double precision searchvalue, double precision roundto=0);
bigint **ST_ValueCount**(text rastertable, text rastercolumn, integer nband, double precision searchvalue, double precision roundto=0);

Description

Returns a set of records with columns `value count` which contain the pixel band value and count of pixels in the raster tile or raster coverage of selected band.

If no band is specified `nband` defaults to 1. If no `searchvalues` are specified, will return all pixel values found in the raster or raster coverage. If one searchvalue is given, will return an integer instead of records denoting the count of pixels having that pixel band value

Note
If `exclude_nodata_value` is set to false, will also count pixels with no data.

Availability: 2.0.0

Examples

```
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
```

Examples: Variant 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;
```

See Also

ST_AsPNG, ST_Reclass, ST_ColorMap

11.12.4 ST_Intersection

ST_Intersection — Returns a raster or a set of geometry-pixelvalue pairs representing the shared portion of two rasters or the geometrical intersection of a vectorization of the raster and a geometry.

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);

Description

Returns a raster or a set of geometry-pixelvalue pairs representing the shared portion of two rasters or the geometrical intersection of a vectorization of the raster and a geometry.

The first three variants, returning a setof geomval, works in vector space. The raster is first vectorized (using ST_DumpAsPolygons) into a set of geomval rows and those rows are then intersected with the geometry using the ST_Intersection (geometry, geometry) PostGIS function. Geometries intersecting only with a nodata value area of a raster returns an empty geometry. They are normally excluded from the results by the proper usage of ST_Intersects in the WHERE clause.

You can access the geometry and the value parts of the resulting set of geomval by surrounding them with parenthesis and adding '.geom' or '.val' at the end of the expression. e.g. (ST_Intersection(rast, geom)).geom

The other variants, returning a raster, works in raster space. They are using the two rasters version of ST_MapAlgebraExpr to perform the intersection.

The extent of the resulting raster corresponds to the geometrical intersection of the two raster extents. The resulting ra includes 'BAND1', 'BAND2' or 'BOTH' bands, following what is passed as the returnband parameter. Nodata value areas present in any band results in nodata value areas in every bands of the result. In other words, any pixel intersecting with a no value pixel becomes a nodata value pixel in the result.

Rasters resulting from ST_Intersection must have a nodata value assigned for areas not intersecting. You can de ne or replac nodata value for any resulting band by providing a nodataval[] array of one or two nodata values depending if you request 'BAND1', 'BAND2' or 'BOTH' bands. The rst value in the array replace the nodata value in the rst band and the second value replace the nodata value in the second band. If one input band do not have a nodata value de ned and none are provided array, one is chosen using the ST_MinPossibleValue function. All variant accepting an array of nodata value can also acce single value which will be assigned to each requested band.

In all variants, if no band number is speci ed band 1 is assumed. If you need an intersection between a raster and geometry returns a raster, refer to ST_Clip.

Note
To get more control on the resulting extent or on what to return when encountering a nodata value, use the two rasters version of ST_MapAlgebraExpr.

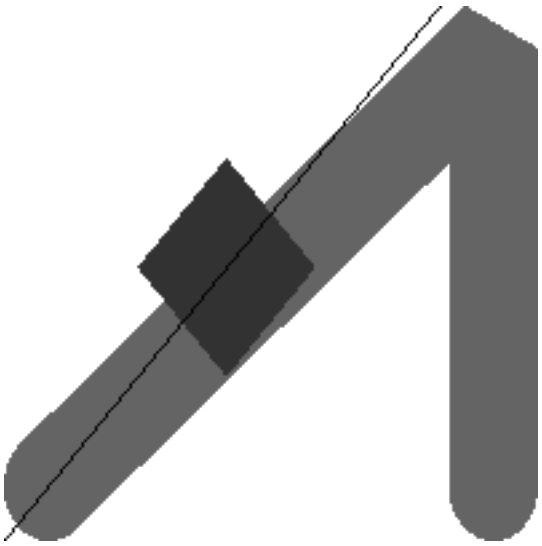
Note
To compute the intersection of a raster band with a geometry in raster space, use ST_Clip. ST_Clip works on multiple bands rasters and does not return a band corresponding to the rasterized geometry.

Note
ST_Intersection should be used in conjunction with ST_Intersects and an index on the raster column and/or the geometry column.

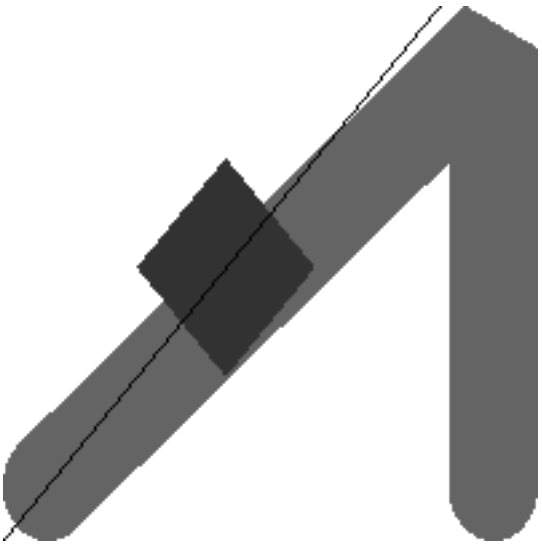
Enhanced: 2.0.0 - Intersection in the raster space was introduced. In earlier pre-2.0.0 versions, only intersection performe vector space were supported.

Examples: Geometry, Raster -- resulting in geometry vals

```
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;
```

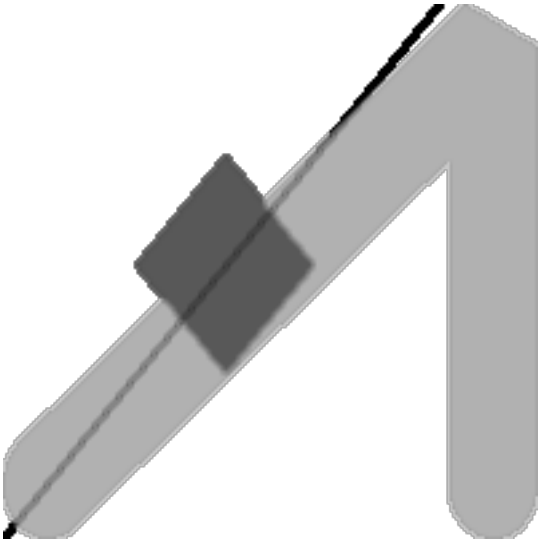


original



no mask mean value (same as having all 1s in mask matrix)

mask only consider neighbors, exclude center



mask weighted only consider neighbors, exclude center multi other pixel values by 2

See Also

rastbandarg, ST_Union, ST_MapAlgebra (expression version)

11.12.6 ST_MapAlgebra (expression version)

ST_MapAlgebra (expression version) — Expression version - Returns a one-band raster given one or two input rasters, band indexes and one or more user-specified SQL expressions.

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);

Description

Expression version - Returns a one-band raster given one or two input rasters, band indexes and one or more user-specified SQL expressions.

Availability: 2.1.0

Description: Variants 1 and 2 (one raster)

Creates a new one band raster formed by applying a valid PostgreSQL algebraic operation defined by the `expression` on the input raster (`rast`). If `nband` is not provided, band 1 is assumed. The new raster will have the same georeference, width, and height as the original raster but will only have one band.

If `pixeltype` is passed in, then the new raster will have a band of that pixeltype. If pixeltype is passed NULL, then the new raster band will have the same pixeltype as the input `rast` band.

- Keywords permitted for `expression`
  1. `[rast]` - Pixel value of the pixel of interest
  2. `[rast.val]` - Pixel value of the pixel of interest
  3. `[rast.x]` - 1-based pixel column of the pixel of interest
  4. `[rast.y]` - 1-based pixel row of the pixel of interest

Description: Variants 3 and 4 (two raster)

Creates a new one band raster formed by applying a valid PostgreSQL algebraic operation to the two bands defined by the `expression` on the two input raster bands `rast1`, (`rast2`). If no `band1`, `band2` is specified band 1 is assumed. The resulting raster will be aligned (scale, skew and pixel corners) on the grid defined by the first raster. The resulting raster will have the extent defined by the `extenttype` parameter.

`expression` A PostgreSQL algebraic expression involving the two rasters and PostgreSQL defined functions/operators that will define the pixel value when pixels intersect. e.g. (([rast1] + [rast2])/2.0)::integer

`pixeltype` The resulting pixel type of the output raster. Must be one listed in ST_BandPixelType, left out or set to NULL. If not passed in or set to NULL, will default to the pixeltype of the first raster.

`extenttype` Controls the extent of resulting raster

1. `INTERSECTION` - The extent of the new raster is the intersection of the two rasters. This is the default.
2. `UNION` - The extent of the new raster is the union of the two rasters.
3. `FIRST` - The extent of the new raster is the same as the one of the first raster.
4. `SECOND` - The extent of the new raster is the same as the one of the second raster.

`nodata1expr` An algebraic expression involving only `rast2` or a constant that defines what to return when pixels of `rast1` are nodata values and spatially corresponding rast2 pixels have values.

nodata2expr An algebraic expression involving only rast1 or a constant that defines what to return when pixels of rast2 are nodata values and spatially corresponding rast1 pixels have values.

nodatanodataval A numeric constant to return when spatially corresponding rast1 and rast2 pixels are both nodata values.

- Keywords permitted in expression, nodata1expr and nodata2expr
  1. [rast1] - Pixel value of the pixel of interest from rast1
  2. [rast1.val] - Pixel value of the pixel of interest from rast1
  3. [rast1.x] - 1-based pixel column of the pixel of interest from rast1
  4. [rast1.y] - 1-based pixel row of the pixel of interest from rast1
  5. [rast2] - Pixel value of the pixel of interest from rast2
  6. [rast2.val] - Pixel value of the pixel of interest from rast2
  7. [rast2.x] - 1-based pixel column of the pixel of interest from rast2
  8. [rast2.y] - 1-based pixel row of the pixel of interest from rast2

Examples: Variants 1 and 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;
```

Examples: Variant 3 and 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;
```

See Also

rastbandarg, ST_Union, ST_MapAlgebra (callback function version)

11.12.7 ST_MapAlgebraExpr

ST_MapAlgebraExpr — 1 raster band version: Creates a new one band raster formed by applying a valid PostgreSQL algebraic operation on the input raster band and of pixeltype provided. Band 1 is assumed if no band is specified.

Synopsis

raster **ST_MapAlgebraExpr**(raster rast, integer band, text pixeltype, text expression, double precision nodataval=NULL);
raster **ST_MapAlgebraExpr**(raster rast, text pixeltype, text expression, double precision nodataval=NULL);

Description

Warning
ST_MapAlgebraExpr is deprecated as of 2.1.0. Use ST_MapAlgebra (expression version) instead.

Creates a new one band raster formed by applying a valid PostgreSQL algebraic operation defined by the `expression` on the input raster (`rast`). If no `band` is specified band 1 is assumed. The new raster will have the same georeference, width, and height as the original raster but will only have one band.

If `pixeltype` is passed in, then the new raster will have a band of that pixeltype. If pixeltype is passed NULL, then the new raster band will have the same pixeltype as the input `rast` band.

In the expression you can use the term `[rast]` to refer to the pixel value of the original band, `[rast.x]` to refer to the 1-based pixel column index, `[rast.y]` to refer to the 1-based pixel row index.

Availability: 2.0.0

Examples

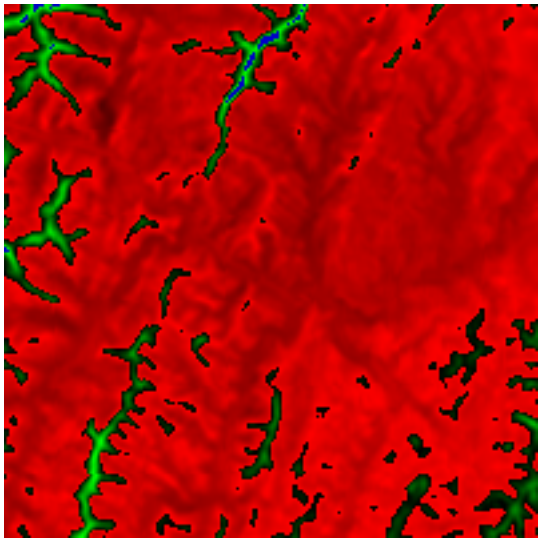
Create a new 1 band raster from our original that is a function of modulo 2 of the original raster band.

```
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
```

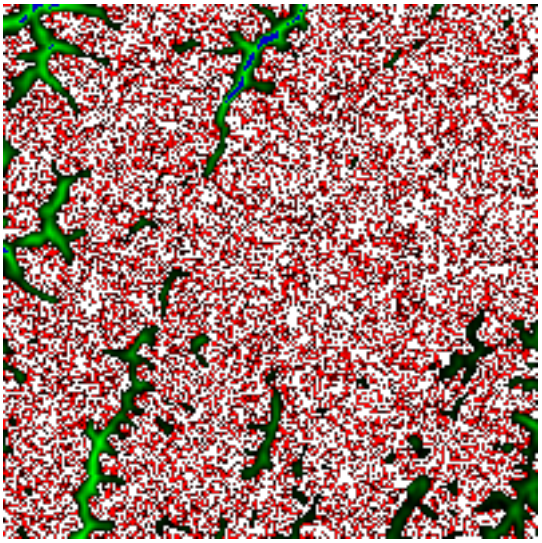
Create a new 1 band raster of pixel-type 2BUI from our original that is reclassified and set the nodata value to be 0.

```
ALTER TABLE dummy_rast ADD COLUMN map_rast2 raster;
UPDATE dummy_rast SET
    map_rast2 = ST_MapAlgebraExpr(rast,'2BUI'::text,'CASE WHEN [rast] BETWEEN 100 and 250  ↩
        THEN 1 WHEN [rast] = 252 THEN 2 WHEN [rast] BETWEEN 253 and 254 THEN 3 ELSE 0 END':: ↩
        text, '0')
```

```
WHERE rid = 2;

SELECT DISTINCT
    ST_Value(rast,1,i,j) As origval,
    ST_Value(map_rast2, 1, i, j) As mapval
FROM dummy_rast
CROSS JOIN generate_series(1, 5) AS i
CROSS JOIN generate_series(1,5) AS j
WHERE rid = 2;

 origval | mapval
---------+--------
     249 |      1
     250 |      1
     251 |
     252 |      2
     253 |      3
     254 |      3

SELECT
    ST_BandPixelType(map_rast2) As b1pixtyp
FROM dummy_rast
WHERE rid = 2;

 b1pixtyp
----------
 2BUI
```



original (column rast_view)



rast_view_ma

Create a new 3 band raster same pixel type from our original 3 band raster with first band altered by map algebra and remaining 2 bands unaltered.

```
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;
```

See Also

ST_MapAlgebraExpr, ST_MapAlgebraFct, ST_BandPixelType, ST_GeoReference, ST_Value

11.12.8 ST_MapAlgebraExpr

ST_MapAlgebraExpr — 2 raster band version: Creates a new one band raster formed by applying a valid PostgreSQL algebraic operation on the two input raster bands and of pixeltype provided. band 1 of each raster is assumed if no band numbers are specified. The resulting raster will be aligned (scale, skew and pixel corners) on the grid defined by the first raster and have its extent defined by the "extenttype" parameter. Values for "extenttype" can be: INTERSECTION, UNION, FIRST, SECOND.

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);

Description

Warning
ST_MapAlgebraExpr is deprecated as of 2.1.0. Use ST_MapAlgebra (expression version) instead.

Creates a new one band raster formed by applying a valid PostgreSQL algebraic operation to the two bands defined by the `expression` on the two input raster bands `rast1`, (`rast2`). If no `band1`, `band2` is specified band 1 is assumed. The resulting raster will be aligned (scale, skew and pixel corners) on the grid defined by the first raster. The resulting raster will have the extent defined by the `extenttype` parameter.

`expression` A PostgreSQL algebraic expression involving the two rasters and PostgreSQL defined functions/operators that will define the pixel value when pixels intersect. e.g. (([rast1] + [rast2])/2.0)::integer

`pixeltype` The resulting pixel type of the output raster. Must be one listed in ST_BandPixelType, left out or set to NULL. If not passed in or set to NULL, will default to the pixeltype of the first raster.

`extenttype` Controls the extent of resulting raster

1. `INTERSECTION` - The extent of the new raster is the intersection of the two rasters. This is the default.
2. `UNION` - The extent of the new raster is the union of the two rasters.
3. `FIRST` - The extent of the new raster is the same as the one of the first raster.
4. `SECOND` - The extent of the new raster is the same as the one of the second raster.

`nodata1expr` An algebraic expression involving only `rast2` or a constant that defines what to return when pixels of `rast1` are nodata values and spatially corresponding rast2 pixels have values.

nodata2expr An algebraic expression involving only `rast1` or a constant that defines what to return when pixels of `rast2` are nodata values and spatially corresponding rast1 pixels have values.

nodatanodataval A numeric constant to return when spatially corresponding rast1 and rast2 pixels are both nodata values.

If `pixeltype` is passed in, then the new raster will have a band of that pixeltype. If pixeltype is passed NULL or no pixel type specified, then the new raster band will have the same pixeltype as the input `rast1` band.

Use the term `[rast1.val] [rast2.val]` to refer to the pixel value of the original raster bands and `[rast1.x]`, `[rast1.y]` etc. to refer to the column / row positions of the pixels.

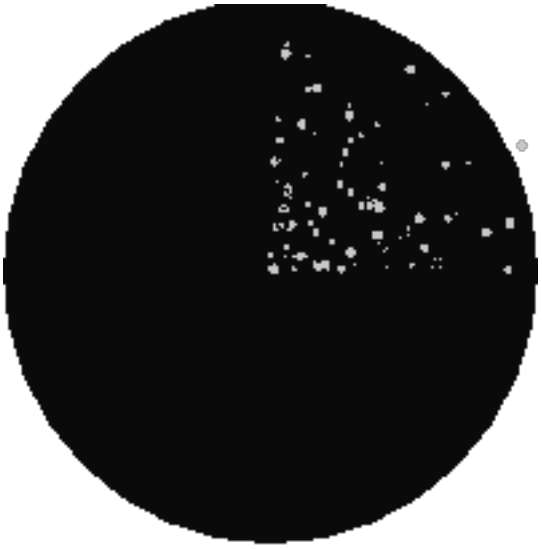
Availability: 2.0.0

Example: 2 Band Intersection and Union

Create a new 1 band raster from our original that is a function of modulo 2 of the original raster band.

```
--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
```

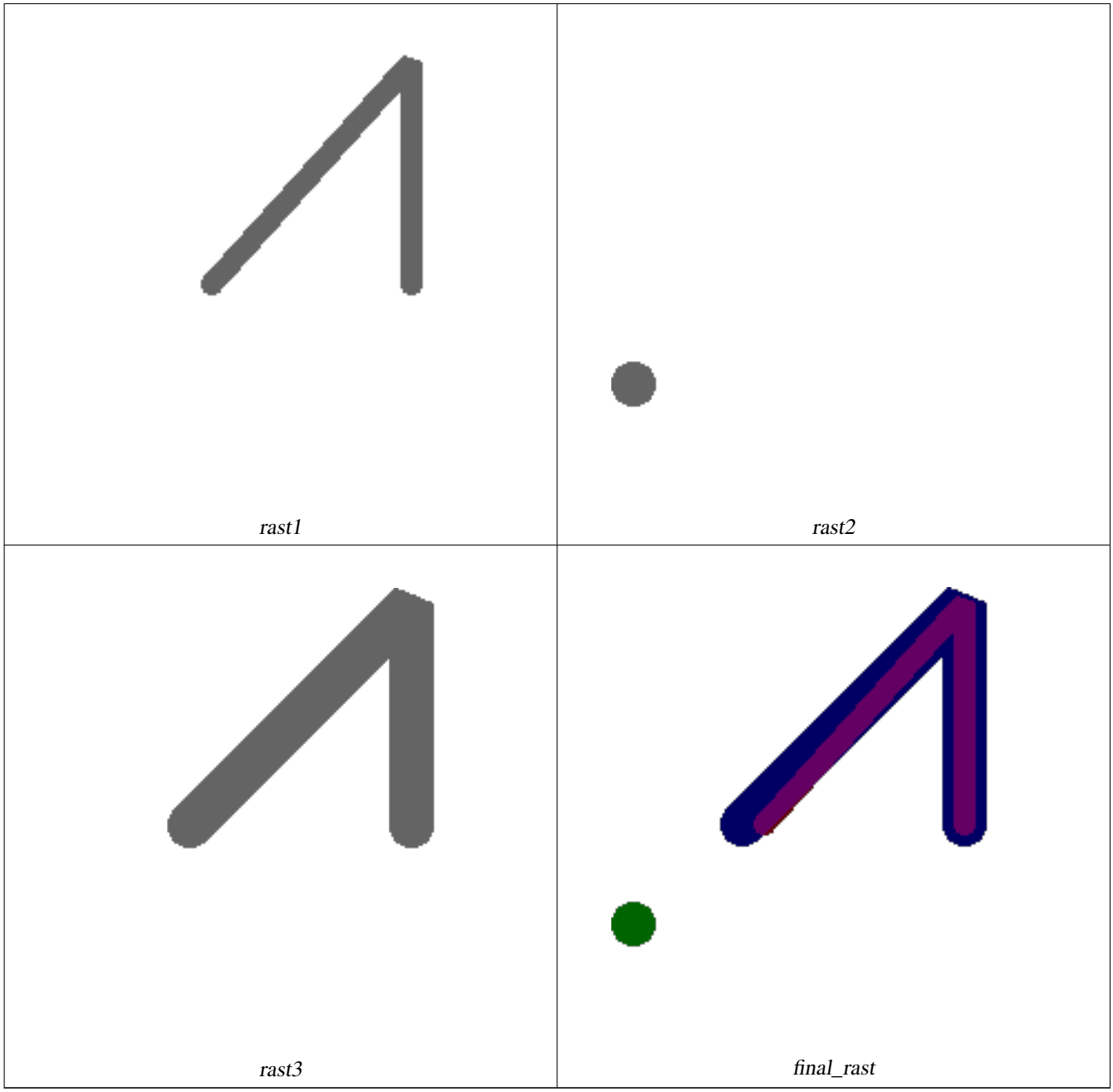
mapalgebra intersection



map algebra union

Example: Overlaying rasters on a canvas as separate bands

```
-- 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;
```



Example: Overlay 2 meter boundary of select parcels over an aerial imagery

```
-- 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
```

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);

Description

Warning
ST_MapAlgebraFct is deprecated as of 2.1.0. Use ST_MapAlgebra (callback function version) instead.

Creates a new one band raster formed by applying a valid PostgreSQL function specified by the `onerasteruserfunc` on the input raster (`rast`). If no `band` is specified, band 1 is assumed. The new raster will have the same georeference, width, and height as the original raster but will only have one band.

If `pixeltype` is passed in, then the new raster will have a band of that pixeltype. If pixeltype is passed NULL, then the new raster band will have the same pixeltype as the input `rast` band.

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.

The third argument to the `userfunction` is a variadic text array. All trailing text arguments to any ST_MapAlgebraFct call are passed through to the specified `userfunction`, and are contained in the `args` argument.

Note
For more information about the VARIADIC keyword, please refer to the PostgreSQL documentation and the "SQL Functions with Variable Numbers of Arguments" section of Query Language (SQL) Functions.

Note
The text[] argument to the `userfunction` is required, regardless of whether you choose to pass any arguments to your user function for processing or not.

Availability: 2.0.0

Examples

Create a new 1 band raster from our original that is a function of modulo 2 of the original raster band.

```
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
```

Create a new 1 band raster of pixel-type 2BUI from our original that is reclassi ed and set the nodata value to a passed param
to the user function (0).

```
ALTER TABLE dummy_rast ADD COLUMN map_rast2 raster;
CREATE FUNCTION classify_fct(pixel float, pos integer[], variadic args text[])
RETURNS float
AS
$$
DECLARE
      nodata float := 0;
BEGIN
      IF NOT args[1] IS NULL THEN
            nodata := args[1];
      END IF;
      IF pixel < 251 THEN
            RETURN 1;
      ELSIF pixel = 252 THEN
            RETURN 2;
      ELSIF pixel > 252 THEN
            RETURN 3;
      ELSE
            RETURN nodata;
      END IF;
END;
$$
LANGUAGE plpgsql ;
UPDATE dummy_rast SET map_rast2 = ST_MapAlgebraFct(rast, 2BUI , classify_fct(float,integer -
     [],text[]) ::regprocedure, 0 ) WHERE rid = 2;
```

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.

The fourth argument to the `tworastuserfunc` is a variadic text array. All trailing text arguments to any ST_MapAlgebraFct call are passed through to the specified `tworastuserfunc`, and are contained in the `userargs` argument.

Note
For more information about the VARIADIC keyword, please refer to the PostgreSQL documentation and the "SQL Functions with Variable Numbers of Arguments" section of Query Language (SQL) Functions.

Note
The text[] argument to the `tworastuserfunc` is required, regardless of whether you choose to pass any arguments to your user function for processing or not.

Availability: 2.0.0

Example: Overlaying rasters on a canvas as separate bands

```
-- 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;
```

map bands overlay (canvas) (R: small road, G: circle, B: big road)

User Defined function that takes extra args

```
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)';
```

user defined with extra args and different bands from same raster

See Also

ST_MapAlgebraExpr, ST_BandPixelType, ST_GeoReference, ST_SetValue

11.12.11 ST_MapAlgebraFctNgb

ST_MapAlgebraFctNgb — 1-band version: Map Algebra Nearest Neighbor using user-defined PostgreSQL function. Return a raster which values are the result of a PLPGSQL user function involving a neighborhood of values from the input raster band.

Synopsis

raster **ST_MapAlgebraFctNgb**(raster rast, integer band, text pixeltype, integer ngbwidth, integer ngbheight, regprocedure onerastngbuserfunc, text nodatamode, text[] VARIADIC args);

Description

Warning
ST_MapAlgebraFctNgb is deprecated as of 2.1.0. Use ST_MapAlgebra (callback function version) instead.

(one raster version) Return a raster which values are the result of a PLPGSQL user function involving a neighborhood of values from the input raster band. The user function takes the neighborhood of pixel values as an array of numbers, for each pixel, returns the result from the user function, replacing pixel value of currently inspected pixel with the function result.

`rast` Raster on which the user function is evaluated.

band Band number of the raster to be evaluated. Default to 1.

pixeltype The resulting pixel type of the output raster. Must be one listed in ST_BandPixelType or left out or set to NULL. If not passed in or set to NULL, will default to the pixeltype of the `rast`. Results are truncated if they are larger than what is allowed for the pixeltype.

ngbwidth The width of the neighborhood, in cells.

ngbheight The height of the neighborhood, in cells.

onerastngbuserfunc PLPGSQL/psql user function to apply to neighborhood pixels of a single band of a raster. The first element is a 2-dimensional array of numbers representing the rectangular pixel neighborhood

nodatamode Defines what value to pass to the function for a neighborhood pixel that is nodata or NULL

'ignore': any NODATA values encountered in the neighborhood are ignored by the computation -- this flag must be sent to the user callback function, and the user function decides how to ignore it.

'NULL': any NODATA values encountered in the neighborhood will cause the resulting pixel to be NULL -- the user callback function is skipped in this case.

'value': any NODATA values encountered in the neighborhood are replaced by the reference pixel (the one in the center of the neighborhood). Note that if this value is NODATA, the behavior is the same as 'NULL' (for the affected neighborhood)

args Arguments to pass into the user function.

Availability: 2.0.0

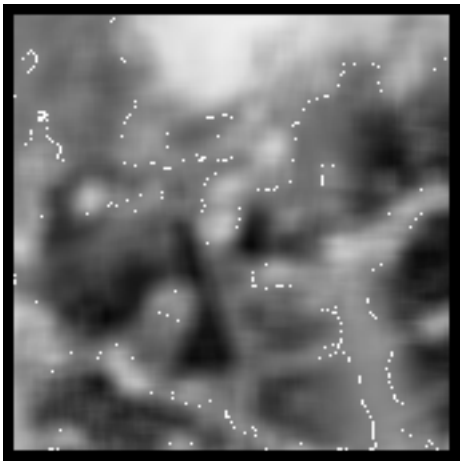
Examples

Examples utilize the katrina raster loaded as a single tile described in http://trac.osgeo.org/gdal/wiki/frmts_wtkraster.html and then prepared in the ST_Rescale examples

```
--
-- 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;
```

First band of our raster



new raster after averaging pixels within 4x4 pixels of each other

See Also

ST_MapAlgebraFct, ST_MapAlgebraExpr, ST_Rescale

11.12.12 ST_Reclass

ST_Reclass — Creates a new raster composed of band types reclassified from original. The nband is the band to be changed. If nband is not specified assumed to be 1. All other bands are returned unchanged. Use case: convert a 16BUI band to a 8BUI and so forth for simpler rendering as viewable formats.

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);

Description

Creates a new raster formed by applying a valid PostgreSQL algebraic operation defined by the `reclassexpr` on the input raster (`rast`). If no `band` is specified band 1 is assumed. The new raster will have the same georeference, width, and height as the original raster. Bands not designated will come back unchanged. Refer to reclassarg for description of valid reclassification expressions.

The bands of the new raster will have pixel type of `pixeltype`. If `reclassargset` is passed in then each reclassarg defines behavior of each band generated.

Availability: 2.0.0

Examples Basic

Create a new raster from the original where band 2 is converted from 8BUI to 4BUI and all values from 101-254 are set to nodata value.

```
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
```

Example: Advanced using multiple reclassargs

Create a new raster from the original where band 1,2,3 is converted to 1BB,4BUI, 4BUI respectively and reclassified. Note this uses the variadic `reclassarg` argument which can take as input an indefinite number of reclassargs (theoretically as many bands as you have)

```
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
```

Example: Advanced Map a single band 32BF raster to multiple viewable bands

Create a new 3 band (8BUI,8BUI,8BUI viewable raster) from a raster that has only one 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)
    ]
    );
```

See Also

ST_AddBand, ST_Band, ST_BandPixelType, ST_MakeEmptyRaster, reclassarg, ST_Value

11.12.13 ST_Union

ST_Union — Returns the union of a set of raster tiles into a single raster composed of 1 or more bands.

Synopsis

raster **ST_Union**(setof raster rast);
raster **ST_Union**(setof raster rast, unionarg[] unionargset);
raster **ST_Union**(setof raster rast, integer nband);
raster **ST_Union**(setof raster rast, text uniontype);
raster **ST_Union**(setof raster rast, integer nband, text uniontype);

Description

Returns the union of a set of raster tiles into a single raster composed of at least one band. The resulting raster's extent is the extent of the whole set. In the case of intersection, the resulting value is defined by `uniontype` which is one of the following: LAST (default), FIRST, MIN, MAX, COUNT, SUM, MEAN, RANGE.

Note
In order for rasters to be unioned, they must all have the same alignment. Use ST_SameAlignment and ST_NotSameAlignmentReason for more details and help. One way to fix alignment issues is to use ST_Resample and use the same reference raster for alignment.

Availability: 2.0.0

Enhanced: 2.1.0 Improved Speed (fully C-Based).

Availability: 2.1.0 ST_Union(rast, unionarg) variant was introduced.

Enhanced: 2.1.0 ST_Union(rast) (variant 1) unions all bands of all input rasters. Prior versions of PostGIS assumed the first band.

Enhanced: 2.1.0 ST_Union(rast, uniontype) (variant 4) unions all bands of all input rasters.

Examples: Reconstitute a single band chunked raster tile

```
-- this creates a single band from first band of raster tiles
-- that form the original file system tile
SELECT filename, ST_Union(rast,1) As file_rast
FROM sometable WHERE filename IN('dem01', 'dem02') GROUP BY filename;
```

Examples: Return a multi-band raster that is the union of tiles intersecting geometry

```
-- this creates a multi band raster collecting all the tiles that intersect a line
-- Note: 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) ↩
    );
```

Examples: Return a multi-band raster that is the union of tiles intersecting geometry

Here we use the longer syntax if we only wanted a subset of bands or we want to change order of bands

```
-- 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) ↩
    );
```

See Also

unionarg, ST_Envelope, ST_ConvexHull, ST_Clip, ST_Union

11.13 Built-in Map Algebra Callback Functions

11.13.1 ST_Distinct4ma

ST_Distinct4ma — Raster processing function that calculates the number of unique pixel values in a neighborhood.

Synopsis

float8 **ST_Distinct4ma**(float8[][] matrix, text nodatamode, text[] VARIADIC args);
double precision **ST_Distinct4ma**(double precision[][][] value, integer[][] pos, text[] VARIADIC userargs);

Description

Calculate the number of unique pixel values in a neighborhood of pixels.

Note
Variant 1 is a specialized callback function for use as a callback parameter to ST_MapAlgebraFctNgb.

Note
Variant 2 is a specialized callback function for use as a callback parameter to ST_MapAlgebra (callback function version).

Warning
Use of Variant 1 is discouraged since ST_MapAlgebraFctNgb has been deprecated as of 2.1.0.

Availability: 2.0.0

Enhanced: 2.1.0 Addition of Variant 2

Examples

```
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)
```

See Also

ST_MapAlgebraFctNgb, ST_MapAlgebra (callback function version), ST_Min4ma, ST_Max4ma, ST_Sum4ma, ST_Mean4ma, ST_Distinct4ma, ST_StdDev4ma

11.13.2 ST_InvDistWeight4ma

ST_InvDistWeight4ma — Raster processing function that interpolates a pixel's value from the pixel's neighborhood.

Synopsis

double precision **ST_InvDistWeight4ma**(double precision[][][] value, integer[][] pos, text[] VARIADIC userargs);

Description

Calculate an interpolated value for a pixel using the Inverse Distance Weighted method.

There are two optional parameters that can be passed through `userargs`. The first parameter is the power factor (variable k in the equation below) between 0 and 1 used in the Inverse Distance Weighted equation. If not specified, default value is 1. The second parameter is the weight percentage applied only when the value of the pixel of interest is included with the interpolated value from the neighborhood. If not specified and the pixel of interest has a value, that value is returned.

The basic inverse distance weight equation is:

$$\hat{z}(x_o) = \frac{\sum_{j=1}^{m} z(x_j) d_{ij}^{-k}}{\sum_{j=1}^{m} d_{ij}^{-k}}$$

k = power factor, a real number between 0 and 1

Note
This function is a specialized callback function for use as a callback parameter to ST_MapAlgebra (callback function version).

Availability: 2.1.0

Examples

```
-- NEEDS EXAMPLE
```

See Also

ST_MapAlgebra (callback function version), ST_MinDist4ma

11.13.3 ST_Max4ma

ST_Max4ma — Raster processing function that calculates the maximum pixel value in a neighborhood.

Synopsis

float8 **ST_Max4ma**(float8[][] matrix, text nodatamode, text[] VARIADIC args);
double precision **ST_Max4ma**(double precision[][][] value, integer[][] pos, text[] VARIADIC userargs);

Description

Calculate the maximum pixel value in a neighborhood of pixels.

For Variant 2, a substitution value for NODATA pixels can be specified by passing that value to userargs.

Note
Variant 1 is a specialized callback function for use as a callback parameter to ST_MapAlgebraFctNgb.

Note
Variant 2 is a specialized callback function for use as a callback parameter to ST_MapAlgebra (callback function version).

Warning
Use of Variant 1 is discouraged since ST_MapAlgebraFctNgb has been deprecated as of 2.1.0.

Availability: 2.0.0

Enhanced: 2.1.0 Addition of Variant 2

Examples

```
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)
```

See Also

ST_MapAlgebraFctNgb, ST_MapAlgebra (callback function version), ST_Min4ma, ST_Sum4ma, ST_Mean4ma, ST_Range4ma, ST_Distinct4ma, ST_StdDev4ma

11.13.4 ST_Mean4ma

ST_Mean4ma — Raster processing function that calculates the mean pixel value in a neighborhood.

Synopsis

float8 **ST_Mean4ma**(float8[][] matrix, text nodatamode, text[] VARIADIC args);
double precision **ST_Mean4ma**(double precision[][][] value, integer[][] pos, text[] VARIADIC userargs);

Description

Calculate the mean pixel value in a neighborhood of pixels.

For Variant 2, a substitution value for NODATA pixels can be specified by passing that value to userargs.

Note
Variant 1 is a specialized callback function for use as a callback parameter to ST_MapAlgebraFctNgb.

Note
Variant 2 is a specialized callback function for use as a callback parameter to ST_MapAlgebra (callback function version).

Description

Calculate the minimum pixel value in a neighborhood of pixels.

For Variant 2, a substitution value for NODATA pixels can be specified by passing that value to userargs.

Note
Variant 1 is a specialized callback function for use as a callback parameter to ST_MapAlgebraFctNgb.

Note
Variant 2 is a specialized callback function for use as a callback parameter to ST_MapAlgebra (callback function version).

Warning
Use of Variant 1 is discouraged since ST_MapAlgebraFctNgb has been deprecated as of 2.1.0.

Availability: 2.0.0

Enhanced: 2.1.0 Addition of Variant 2

Examples

```
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)
```

See Also

ST_MapAlgebraFctNgb, ST_MapAlgebra (callback function version), ST_Max4ma, ST_Sum4ma, ST_Mean4ma, ST_Range4ma, ST_Distinct4ma, ST_StdDev4ma

11.13.6 ST_MinDist4ma

ST_MinDist4ma — Raster processing function that returns the minimum distance (in number of pixels) between the pixel of interest and a neighboring pixel with value.

Synopsis

double precision **ST_MinDist4ma**(double precision[][][] value, integer[][] pos, text[] VARIADIC userargs);

Description

Return the shortest distance (in number of pixels) between the pixel of interest and the closest pixel with value in the neighborhood.

Note
The intent of this function is to provide an informative data point that helps infer the usefulness of the pixel of interest's interpolated value from ST_InvDistWeight4ma. This function is particularly useful when the neighborhood is sparsely populated.

Note
This function is a specialized callback function for use as a callback parameter to ST_MapAlgebra (callback function version).

Availability: 2.1.0

Examples

```
-- NEEDS EXAMPLE
```

See Also

ST_MapAlgebra (callback function version), ST_InvDistWeight4ma

11.13.7 ST_Range4ma

ST_Range4ma — Raster processing function that calculates the range of pixel values in a neighborhood.

Synopsis

float8 **ST_Range4ma**(float8[][] matrix, text nodatamode, text[] VARIADIC args);
double precision **ST_Range4ma**(double precision[][][] value, integer[][] pos, text[] VARIADIC userargs);

Description

Calculate the range of pixel values in a neighborhood of pixels.

For Variant 2, a substitution value for NODATA pixels can be specified by passing that value to userargs.

Note
Variant 1 is a specialized callback function for use as a callback parameter to ST_MapAlgebraFctNgb.

Note
Variant 2 is a specialized callback function for use as a callback parameter to ST_MapAlgebra (callback function version).

Warning
Use of Variant 1 is discouraged since ST_MapAlgebraFctNgb has been deprecated as of 2.1.0.

Availability: 2.0.0

Enhanced: 2.1.0 Addition of Variant 2

Examples

```
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)
```

See Also

ST_MapAlgebraFctNgb, ST_MapAlgebra (callback function version), ST_Min4ma, ST_Max4ma, ST_Sum4ma, ST_Mean4ma, ST_Distinct4ma, ST_StdDev4ma

11.13.8 ST_StdDev4ma

ST_StdDev4ma — Raster processing function that calculates the standard deviation of pixel values in a neighborhood.

Synopsis

float8 **ST_StdDev4ma**(float8[][] matrix, text nodatamode, text[] VARIADIC args);
double precision **ST_StdDev4ma**(double precision[][][] value, integer[][] pos, text[] VARIADIC userargs);

Description

Calculate the standard deviation of pixel values in a neighborhood of pixels.

Note
Variant 1 is a specialized callback function for use as a callback parameter to ST_MapAlgebraFctNgb.

Note
Variant 2 is a specialized callback function for use as a callback parameter to ST_MapAlgebra (callback function version).

Warning
Use of Variant 1 is discouraged since ST_MapAlgebraFctNgb has been deprecated as of 2.1.0.

Availability: 2.0.0

Enhanced: 2.1.0 Addition of Variant 2

Examples

```
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)
```

See Also

ST_MapAlgebraFctNgb, ST_MapAlgebra (callback function version), ST_Min4ma, ST_Max4ma, ST_Mean4ma, ST_Range4ma, ST_Distinct4ma, ST_StdDev4ma

11.14 Raster Processing: DEM (Elevation)

11.14.1 ST_Aspect

ST_Aspect — Returns the aspect (in degrees by default) of an elevation raster band. Useful for analyzing terrain.

Synopsis

raster **ST_Aspect**(raster rast, integer band=1, text pixeltype=32BF, text units=DEGREES, boolean interpolate_nodata=FALSE);
raster **ST_Aspect**(raster rast, integer band, raster customextent, text pixeltype=32BF, text units=DEGREES, boolean interpolate_nodata=FALSE);

Description

Returns the aspect (in degrees by default) of an elevation raster band. Utilizes map algebra and applies the aspect equation to neighboring pixels.

`units` indicates the units of the aspect. Possible values are: RADIANS, DEGREES (default).

When `units` = RADIANS, values are between 0 and 2 * pi radians measured clockwise from North.

When `units` = DEGREES, values are between 0 and 360 degrees measured clockwise from North.

If slope of pixel is zero, aspect of pixel is -1.

Note
For more information about Slope, Aspect and Hillshade, please refer to ESRI - How hillshade works and ERDAS Field Guide - Aspect Images.

Availability: 2.0.0

Enhanced: 2.1.0 Uses ST_MapAlgebra() and added optional `interpolate_nodata` function parameter

Changed: 2.1.0 In prior versions, return values were in radians. Now, return values default to degrees

Examples: Variant 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)
```

Examples: Variant 2

Complete example of tiles of a coverage. This query only works with PostgreSQL 9.1 or higher.

```
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;
```

See Also

ST_MapAlgebra (callback function version), ST_TRI, ST_TPI, ST_Roughness, ST_HillShade, ST_Slope

11.14.2 ST_HillShade

ST_HillShade — Returns the hypothetical illumination of an elevation raster band using provided azimuth, altitude, brightness and scale inputs.

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);

Description

Returns the hypothetical illumination of an elevation raster band using the azimuth, altitude, brightness, and scale inputs. Utilizes map algebra and applies the hill shade equation to neighboring pixels. Return pixel values are between 0 and 255.

`azimuth` is a value between 0 and 360 degrees measured clockwise from North.

`altitude` is a value between 0 and 90 degrees where 0 degrees is at the horizon and 90 degrees is directly overhead.

`max_bright` is a value between 0 and 255 with 0 as no brightness and 255 as max brightness.

`scale` is the ratio of vertical units to horizontal. For Feet:LatLon use scale=370400, for Meters:LatLon use scale=111120.

If `interpolate_nodata` is TRUE, values for NODATA pixels from the input raster will be interpolated using ST_InvDistWeight4ma before computing the hillshade illumination.

Note
For more information about Hillshade, please refer to How hillshade works.

Availability: 2.0.0

Enhanced: 2.1.0 Uses ST_MapAlgebra() and added optional `interpolate_nodata` function parameter

Changed: 2.1.0 In prior versions, azimuth and altitude were expressed in radians. Now, azimuth and altitude are expressed in degrees

See Also

ST_MapAlgebra (callback function version), ST_TRI, ST_TPI, ST_Roughness, ST_Aspect, ST_Slope

11.14.3 ST_Roughness

ST_Roughness — Returns a raster with the calculated "roughness" of a DEM.

Synopsis

raster **ST_Roughness**(raster rast, integer nband, raster customextent, text pixeltype="32BF" , boolean interpolate_nodata=FALSE);

Description

Calculates the "roughness" of a DEM, by subtracting the maximum from the minimum for a given area.

Availability: 2.1.0

Examples

```
-- needs examples
```

See Also

ST_MapAlgebra (callback function version), ST_TRI, ST_TPI, ST_Slope, ST_HillShade, ST_Aspect

11.14.4 ST_Slope

ST_Slope — Returns the slope (in degrees by default) of an elevation raster band. Useful for analyzing terrain.

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);

Description

Returns the slope (in degrees by default) of an elevation raster band. Utilizes map algebra and applies the slope equation to neighboring pixels.

`units` indicates the units of the slope. Possible values are: RADIANS, DEGREES (default), PERCENT.

`scale` is the ratio of vertical units to horizontal. For Feet:LatLon use scale=370400, for Meters:LatLon use scale=111120.

If `interpolate_nodata` is TRUE, values for NODATA pixels from the input raster will be interpolated using ST_InvDistWeight4ma before computing the surface slope.

Note
For more information about Slope, Aspect and Hillshade, please refer to ESRI - How hillshade works and ERDAS Field Guide - Slope Images.

Availability: 2.0.0

Enhanced: 2.1.0 Uses ST_MapAlgebra() and added optional `units`, `scale`, `interpolate_nodata` function parameters

Changed: 2.1.0 In prior versions, return values were in radians. Now, return values default to degrees

Examples: Variant 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)
```

Examples: Variant 2

Complete example of tiles of a coverage. This query only works with PostgreSQL 9.1 or higher.

```
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;
```

See Also

ST_MapAlgebra (callback function version), ST_TRI, ST_TPI, ST_Roughness, ST_HillShade, ST_Aspect

11.14.5 ST_TPI

ST_TPI — Returns a raster with the calculated Topographic Position Index.

Synopsis

raster **ST_TPI**(raster rast, integer nband, raster customextent, text pixeltype="32BF" , boolean interpolate_nodata=FALSE);

Description

Calculates the Topographic Position Index, which is defined as the focal mean with radius of one minus the center cell.

Note
This function only supports a focalmean radius of one.

Availability: 2.1.0

Examples

```
-- needs examples
```

See Also

ST_MapAlgebra (callback function version), ST_TRI, ST_Roughness, ST_Slope, ST_HillShade, ST_Aspect

11.14.6 ST_TRI

ST_TRI — Returns a raster with the calculated Terrain Ruggedness Index.

See Also

ST_Envelope

11.15.2 ST_ConvexHull

ST_ConvexHull — Return the convex hull geometry of the raster including pixel values equal to BandNoDataValue. For regu shaped and non-skewed rasters, this gives the same result as ST_Envelope so only useful for irregularly shaped or skewed ra

Synopsis

geometry ST_ConvexHull(raster rast);

Description

Return the convex hull geometry of the raster including the NoDataBandValue band pixels. For regular shaped and non-ske rasters, this gives more or less the same result as ST_Envelope so only useful for irregularly shaped or skewed rasters.

> Note
> ST_Envelope oors the coordinates and hence add a little buffer around the raster so the answer is subtly different from ST_ConvexHull which does not oor.

Examples

Refer to PostGIS Raster Speci cation for a diagram of this.

```
-- 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)
    )
```

See Also

ST_Envelope, ST_MinConvexHull, ST_ConvexHull, ST_AsText

Synopsis

geometry **ST_Polygon**(raster rast, integer band_num=1);

Description

Changed 3.3.0, validation and fixing is disabled to improve performance. May result invalid geometries.

Availability: 0.1.6 Requires GDAL 1.7 or higher.

Enhanced: 2.1.0 Improved Speed (fully C-Based) and the returning multipolygon is ensured to be valid.

Changed: 2.1.0 In prior versions would sometimes return a polygon, changed to always return multipolygon.

Examples

```
-- 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)))
```

See Also

ST_Value, ST_DumpAsPolygons

11.16 Raster Operators

11.16.1 &&

&& — Returns TRUE if A's bounding box intersects B's bounding box.

Synopsis

boolean **&&**(raster A , raster B);
boolean **&&**(raster A , geometry B);
boolean **&&**(geometry B , raster A);

Description

The && operator returns TRUE if the bounding box of raster/geometr A intersects the bounding box of raster/geometr B.

Note
This operand will make use of any indexes that may be available on the rasters.

Availability: 2.0.0

Examples

```
SELECT A.rid As a_rid, B.rid As b_rid, A.rast && B.rast As intersect
 FROM dummy_rast AS A CROSS JOIN dummy_rast AS B LIMIT 3;

 a_rid | b_rid | intersect
-------+-------+---------
     2 |     2 | t
     2 |     3 | f
     2 |     1 | f
```

11.16.2 &<

&< — Returns TRUE if A's bounding box is to the left of B's.

Synopsis

boolean **&<**(raster A , raster B);

Description

The `&<` operator returns `TRUE` if the bounding box of raster A overlaps or is to the left of the bounding box of raster B, or more accurately, overlaps or is NOT to the right of the bounding box of raster B.

Note
This operand will make use of any indexes that may be available on the rasters.

Examples

```
SELECT A.rid As a_rid, B.rid As b_rid, A.rast &< B.rast As overleft
 FROM dummy_rast AS A CROSS JOIN dummy_rast AS B;

a_rid | b_rid | overleft
------+-------+----------
    2 |     2 | t
    2 |     3 | f
    2 |     1 | f
    3 |     2 | t
    3 |     3 | t
    3 |     1 | f
    1 |     2 | t
    1 |     3 | t
    1 |     1 | t
```

11.16.3 &>

&> — Returns `TRUE` if A's bounding box is to the right of B's.

Synopsis

boolean **&>**(raster A , raster B);

Description

The `&>` operator returns `TRUE` if the bounding box of raster A overlaps or is to the right of the bounding box of raster B, or more accurately, overlaps or is NOT to the left of the bounding box of raster B.

Note
This operand will make use of any indexes that may be available on the geometries.

Examples

```
SELECT A.rid As a_rid, B.rid As b_rid, A.rast &> B.rast As overright
 FROM dummy_rast AS A CROSS JOIN dummy_rast AS B;

 a_rid | b_rid | overright
-------+-------+-----------
     2 |     2 | t
     2 |     3 | t
     2 |     1 | t
     3 |     2 | f
     3 |     3 | t
     3 |     1 | f
     1 |     2 | f
     1 |     3 | t
     1 |     1 | t
```

11.16.4 =

= — Returns TRUE if A's bounding box is the same as B's. Uses double precision bounding box.

Synopsis

boolean =(raster A , raster B);

Description

The = operator returns TRUE if the bounding box of raster A is the same as the bounding box of raster B. PostgreSQL uses the =, <, and > operators defined for rasters to perform internal orderings and comparison of rasters (ie. in a GROUP BY or ORDER BY clause).

Caution
This operand will NOT make use of any indexes that may be available on the rasters. Use ~= instead. This operator exists mostly so one can group by the raster column.

Availability: 2.1.0

See Also

~=

11.16.5 @

@ — Returns TRUE if A's bounding box is contained by B's. Uses double precision bounding box.

Synopsis

boolean @(raster A , raster B);
boolean @(geometry A , raster B);
boolean @(raster B , geometry A);

Description

The @ operator returns TRUE if the bounding box of raster/geometry A is contained by bounding box of raster/geometr B.

Note
This operand will use spatial indexes on the rasters.

Availability: 2.0.0 raster @ raster, raster @ geometry introduced

Availability: 2.0.5 geometry @ raster introduced

See Also

~

11.16.6 ~=

~= — Returns TRUE if A's bounding box is the same as B's.

Synopsis

boolean **~=**(raster A , raster B);

Description

The ~= operator returns TRUE if the bounding box of raster A is the same as the bounding box of raster B.

Note
This operand will make use of any indexes that may be available on the rasters.

Availability: 2.0.0

Examples

Very useful usecase is for taking two sets of single band rasters that are of the same chunk but represent different themes and creating a multi-band raster

```
SELECT ST_AddBand(prec.rast, alt.rast) As new_rast
    FROM prec INNER JOIN alt ON (prec.rast ~= alt.rast);
```

See Also

ST_AddBand, =

11.16.7 ~

~ — Returns TRUE if A's bounding box is contains B's. Uses double precision bounding box.

Synopsis

boolean ~(raster A , raster B);
boolean ~(geometry A , raster B);
boolean ~(raster B , geometry A);

Description

The ~ operator returns `TRUE` if the bounding box of raster/geometry A is contains bounding box of raster/geometr B.

Note
This operand will use spatial indexes on the rasters.

Availability: 2.0.0

See Also

@

11.17 Raster and Raster Band Spatial Relationships

11.17.1 ST_Contains

ST_Contains — Return true if no points of raster rastB lie in the exterior of raster rastA and at least one point of the interior of rastB lies in the interior of rastA.

Synopsis

boolean **ST_Contains**(raster rastA , integer nbandA , raster rastB , integer nbandB);
boolean **ST_Contains**(raster rastA , raster rastB);

Description

Raster rastA contains rastB if and only if no points of rastB lie in the exterior of rastA and at least one point of the interior of rastB lies in the interior of rastA. If the band number is not provided (or set to NULL), only the convex hull of the raster is considered in the test. If the band number is provided, only those pixels with value (not NODATA) are considered in the test.

Note
This function will make use of any indexes that may be available on the rasters.

Note
To test the spatial relationship of a raster and a geometry, use ST_Polygon on the raster, e.g. ST_Contains(ST_Polygon(raster), geometry) or ST_Contains(geometry, ST_Polygon(raster)).

Note
To test the spatial relationship of a raster and a geometry, use ST_Polygon on the raster, e.g. ST_ContainsProperly(ST_Polygon(raster), geometry) or ST_ContainsProperly(geometry, ST_Polygon(raster)).

Availability: 2.1.0

Examples

```
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
```

See Also

ST_Intersects, ST_Contains

11.17.3 ST_Covers

ST_Covers — Return true if no points of raster rastB lie outside raster rastA.

Synopsis

boolean **ST_Covers**(raster rastA , integer nbandA , raster rastB , integer nbandB);
boolean **ST_Covers**(raster rastA , raster rastB);

Description

Raster rastA covers rastB if and only if no points of rastB lie in the exterior of rastA. If the band number is not provided (or set to NULL), only the convex hull of the raster is considered in the test. If the band number is provided, only those pixels with value (not NODATA) are considered in the test.

Note
This function will make use of any indexes that may be available on the rasters.

Note
To test the spatial relationship of a raster and a geometry, use ST_Polygon on the raster, e.g. ST_Covers(ST_Polygon(raster), geometry) or ST_Covers(geometry, ST_Polygon(raster)).

Availability: 2.1.0

Examples

```
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
```

See Also

ST_Intersects, ST_CoveredBy

11.17.4 ST_CoveredBy

ST_CoveredBy — Return true if no points of raster rastA lie outside raster rastB.

Synopsis

boolean **ST_CoveredBy**(raster rastA , integer nbandA , raster rastB , integer nbandB);
boolean **ST_CoveredBy**(raster rastA , raster rastB);

Description

Raster rastA is covered by rastB if and only if no points of rastA lie in the exterior of rastB. If the band number is not provided (or set to NULL), only the convex hull of the raster is considered in the test. If the band number is provided, only those pixels with value (not NODATA) are considered in the test.

Note
This function will make use of any indexes that may be available on the rasters.

Note
To test the spatial relationship of a raster and a geometry, use ST_Polygon on the raster, e.g. ST_CoveredBy(ST_Polygon(raster), geometry) or ST_CoveredBy(geometry, ST_Polygon(raster)).

Availability: 2.1.0

Examples

```
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
```

See Also

ST_Intersects, ST_Covers

11.17.5 ST_Disjoint

ST_Disjoint — Return true if raster rastA does not spatially intersect rastB.

Synopsis

boolean **ST_Disjoint**(raster rastA , integer nbandA , raster rastB , integer nbandB);
boolean **ST_Disjoint**(raster rastA , raster rastB);

Description

Raster rastA and rastB are disjointed if they do not share any space together. If the band number is not provided (or set to NULL), only the convex hull of the raster is considered in the test. If the band number is provided, only those pixels with value (not NODATA) are considered in the test.

Note
This function does NOT use any indexes.

Note
To test the spatial relationship of a raster and a geometry, use ST_Polygon on the raster, e.g. ST_Disjoint(ST_Polygon(raster), geometry).

Availability: 2.1.0

Examples

```
-- 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
```

See Also

ST_Intersects

11.17.6 ST_Intersects

ST_Intersects — Return true if raster rastA spatially intersects raster rastB.

Synopsis

boolean **ST_Intersects**(raster rastA , integer nbandA , raster rastB , integer nbandB);
boolean **ST_Intersects**(raster rastA , raster rastB);
boolean **ST_Intersects**(raster rast , integer nband , geometry geommin);
boolean **ST_Intersects**(raster rast , geometry geommin , integer nband=NULL);
boolean **ST_Intersects**(geometry geommin , raster rast , integer nband=NULL);

Description

Return true if raster rastA spatially intersects raster rastB. If the band number is not provided (or set to NULL), only the convex hull of the raster is considered in the test. If the band number is provided, only those pixels with value (not NODATA) are considered in the test.

Note
This function will make use of any indexes that may be available on the rasters.

Enhanced: 2.0.0 support raster/raster intersects was introduced.

Warning
Changed: 2.1.0 The behavior of the ST_Intersects(raster, geometry) variants changed to match that of ST_Intersects(geometry, raster).

Examples

```
-- different bands of same raster
SELECT ST_Intersects(rast, 2, rast, 3) FROM dummy_rast WHERE rid = 2;

 st_intersects
---------------
 t
```

See Also

ST_Intersection, ST_Disjoint

11.17.7 ST_Overlaps

ST_Overlaps — Return true if raster rastA and rastB intersect but one does not completely contain the other.

Synopsis

boolean **ST_Overlaps**(raster rastA , integer nbandA , raster rastB , integer nbandB);
boolean **ST_Overlaps**(raster rastA , raster rastB);

Description

Return true if raster rastA spatially overlaps raster rastB. This means that rastA and rastB intersect but one does not completely contain the other. If the band number is not provided (or set to NULL), only the convex hull of the raster is considered in the test. If the band number is provided, only those pixels with value (not NODATA) are considered in the test.

Note
This function will make use of any indexes that may be available on the rasters.

Note
To test the spatial relationship of a raster and a geometry, use ST_Polygon on the raster, e.g. ST_Overlaps(ST_Polygon(raster), geometry).

Availability: 2.1.0

Examples

```
-- comparing different bands of same raster
SELECT ST_Overlaps(rast, 1, rast, 2) FROM dummy_rast WHERE rid = 2;

 st_overlaps
-------------
 f
```

See Also

ST_Intersects

11.17.8 ST_Touches

ST_Touches — Return true if raster rastA and rastB have at least one point in common but their interiors do not intersect.

Synopsis

boolean **ST_Touches**(raster rastA , integer nbandA , raster rastB , integer nbandB);
boolean **ST_Touches**(raster rastA , raster rastB);

Description

Return true if raster rastA spatially touches raster rastB. This means that rastA and rastB have at least one point in common but their interiors do not intersect. If the band number is not provided (or set to NULL), only the convex hull of the raster is considered in the test. If the band number is provided, only those pixels with value (not NODATA) are considered in the test.

Note
This function will make use of any indexes that may be available on the rasters.

Note
To test the spatial relationship of a raster and a geometry, use ST_Polygon on the raster, e.g. ST_Touches(ST_Polygon(raster), geometry).

Availability: 2.1.0

Examples

```
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
```

See Also

ST_Intersects

11.17.9 ST_SameAlignment

ST_SameAlignment — Returns true if rasters have same skew, scale, spatial ref, and offset (pixels can be put on same grid without cutting into pixels) and false if they don't with notice detailing issue.

Synopsis

boolean **ST_SameAlignment**(raster rastA , raster rastB);
boolean **ST_SameAlignment**(double precision ulx1 , double precision uly1 , double precision scalex1 , double precision scaley1 , double precision skewx1 , double precision skewy1 , double precision ulx2 , double precision uly2 , double precision scalex2 , double precision scaley2 , double precision skewx2 , double precision skewy2);
boolean **ST_SameAlignment**(raster set rastfield);

Description

Non-Aggregate version (Variants 1 and 2): Returns true if the two rasters (either provided directly or made using the values for upperleft, scale, skew and srid) have the same scale, skew, srid and at least one of any of the four corners of any pixel of one raster falls on any corner of the grid of the other raster. Returns false if they don't and a NOTICE detailing the alignment issue.

Aggregate version (Variant 3): From a set of rasters, returns true if all rasters in the set are aligned. The ST_SameAlignment() 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.

Availability: 2.0.0

Enhanced: 2.1.0 addition of Aggegrate variant

Examples: Rasters

```
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
```

See Also

Section 10.1, ST_NotSameAlignmentReason, ST_MakeEmptyRaster

11.17.10 ST_NotSameAlignmentReason

ST_NotSameAlignmentReason — Returns text stating if rasters are aligned and if not aligned, a reason why.

Synopsis

text **ST_NotSameAlignmentReason**(raster rastA, raster rastB);

Description

Returns text stating if rasters are aligned and if not aligned, a reason why.

Note
If there are several reasons why the rasters are not aligned, only one reason (the first test to fail) will be returned.

Availability: 2.1.0

Examples

```
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
```

See Also

ST_Intersects, ST_Contains, ST_DWithin, ST_DFullyWithin

11.17.12 ST_DWithin

ST_DWithin — Return true if rasters rastA and rastB are within the specified distance of each other.

Synopsis

boolean **ST_DWithin**(raster rastA , integer nbandA , raster rastB , integer nbandB , double precision distance_of_srid);
boolean **ST_DWithin**(raster rastA , raster rastB , double precision distance_of_srid);

Description

Return true if rasters rastA and rastB are within the specified distance of each other. If the band number is not provided (or set to NULL), only the convex hull of the raster is considered in the test. If the band number is provided, only those pixels with value (not NODATA) are considered in the test.

The distance is specified in units defined by the spatial reference system of the rasters. For this function to make sense, the source rasters must both be of the same coordinate projection, having the same SRID.

Note
This operand will make use of any indexes that may be available on the rasters.

Note
To test the spatial relationship of a raster and a geometry, use ST_Polygon on the raster, e.g. ST_DWithin(ST_Polygon(raster), geometry).

Availability: 2.1.0

Examples

```
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
```

See Also

ST_Within, ST_DFullyWithin

11.17.13 ST_DFullyWithin

ST_DFullyWithin — Return true if rasters rastA and rastB are fully within the specified distance of each other.

Synopsis

boolean **ST_DFullyWithin**(raster rastA , integer nbandA , raster rastB , integer nbandB , double precision distance_of_srid);
boolean **ST_DFullyWithin**(raster rastA , raster rastB , double precision distance_of_srid);

Description

Return true if rasters rastA and rastB are fully within the specified distance of each other. If the band number is not provided (or set to NULL), only the convex hull of the raster is considered in the test. If the band number is provided, only those pixels with value (not NODATA) are considered in the test.

The distance is specified in units defined by the spatial reference system of the rasters. For this function to make sense, the source rasters must both be of the same coordinate projection, having the same SRID.

Note
This operand will make use of any indexes that may be available on the rasters.

Note
To test the spatial relationship of a raster and a geometry, use ST_Polygon on the raster, e.g. ST_DFullyWithin(ST_Polygon(raster), geometry).

Availability: 2.1.0

Examples

```
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
```

See Also

ST_Within, ST_DWithin

11.18 Raster Tips

11.18.1 Out-DB Rasters

11.18.1.1 Directory containing many files

When GDAL opens a file, GDAL eagerly scans the directory of that file to build a catalog of other files. If this directory contains many files (e.g. thousands, millions), opening that file becomes extremely slow (especially if that file happens to be on a network drive such as NFS).

To control this behavior, GDAL provides the following environment variable: GDAL_DISABLE_READDIR_ON_OPEN. Set `GDAL_DISABLE_READDIR_ON_OPEN` to `TRUE` to disable directory scanning.

In Ubuntu (and assuming you are using PostgreSQL's packages for Ubuntu), `GDAL_DISABLE_READDIR_ON_OPEN` can be set in */etc/postgresql/POSTGRESQL_VERSION/CLUSTER_NAME/environment* (where POSTGRESQL_VERSION is the version of PostgreSQL, e.g. 9.6 and CLUSTER_NAME is the name of the cluster, e.g. maindb). You can also set PostGIS environment variables here as well.

```
# environment variables for postmaster process
# This file has the same syntax as postgresql.conf:
#  VARIABLE = simple_value
#  VARIABLE2 = 'any value!'
# I. e. you need to enclose any value which does not only consist of letters,
# numbers, and '-', '_', '.' in single quotes. Shell commands are not
# evaluated.
POSTGIS_GDAL_ENABLED_DRIVERS = 'ENABLE_ALL'

POSTGIS_ENABLE_OUTDB_RASTERS = 1

GDAL_DISABLE_READDIR_ON_OPEN = 'TRUE'
```

11.18.1.2 Maximum Number of Open Files

The maximum number of open files permitted by Linux and PostgreSQL are typically conservative (typically 1024 open files per process) given the assumption that the system is consumed by human users. For Out-DB Rasters, a single valid query can easily exceed this limit (e.g. a dataset of 10 year's worth of rasters with one raster for each day containing minimum and maximum temperatures and we want to know the absolute min and max value for a pixel in that dataset).

The easiest change to make is the following PostgreSQL setting: max_files_per_process. The default is set to 1000, which is far too low for Out-DB Rasters. A safe starting value could be 65536 but this really depends on your datasets and the queries run against those datasets. This setting can only be made on server start and probably only in the PostgreSQL configuration file (e.g. */etc/postgresql/POSTGRESQL_VERSION/CLUSTER_NAME/postgresql.conf* in Ubuntu environments).

```
...
# - Kernel Resource Usage -

max_files_per_process = 65536           # min 25
                                        # (change requires restart)
...
```

The major change to make is the Linux kernel's open files limits. There are two parts to this:

- Maximum number of open files for the entire system
- Maximum number of open files per process

11.18.1.2.1 Maximum number of open files for the entire system

You can inspect the current maximum number of open files for the entire system with the following example:

```
$ sysctl -a | grep fs.file-max
fs.file-max = 131072
```

If the value returned is not large enough, add a file in /etc/sysctl.d/ as per the following example:

```
$ echo "fs.file-max = 6145324" >> /etc/sysctl.d/fs.conf

$ cat /etc/sysctl.d/fs.conf
fs.file-max = 6145324

$ sysctl -p --system
* Applying /etc/sysctl.d/fs.conf ...
fs.file-max = 2097152
* Applying /etc/sysctl.conf ...

$ sysctl -a | grep fs.file-max
fs.file-max = 6145324
```

11.18.1.2.2 Maximum number of open files per process

We need to increase the maximum number of open files per process for the PostgreSQL server processes.

To see what the current PostgreSQL service processes are using for maximum number of open files, do as per the follow example (make sure to have PostgreSQL running):

```
$ ps aux | grep postgres
postgres 31713  0.0  0.4 179012 17564 pts/0    S    Dec26   0:03 /home/dustymugs/devel/ -
    postgresql/sandbox/10/usr/local/bin/postgres -D /home/dustymugs/devel/postgresql/sandbox -
    /10/pgdata
postgres 31716  0.0  0.8 179776 33632 ?        Ss   Dec26   0:01 postgres: checkpointer -
    process
postgres 31717  0.0  0.2 179144  9416 ?        Ss   Dec26   0:05 postgres: writer process
postgres 31718  0.0  0.2 179012  8708 ?        Ss   Dec26   0:06 postgres: wal writer -
    process
postgres 31719  0.0  0.1 179568  7252 ?        Ss   Dec26   0:03 postgres: autovacuum -
    launcher process
postgres 31720  0.0  0.1  34228  4124 ?        Ss   Dec26   0:09 postgres: stats collector -
    process
postgres 31721  0.0  0.1 179308  6052 ?        Ss   Dec26   0:00 postgres: bgworker: -
    logical replication launcher

$ cat /proc/31718/limits
Limit                     Soft Limit           Hard Limit           Units
Max cpu time              unlimited            unlimited            seconds
Max file size             unlimited            unlimited            bytes
Max data size             unlimited            unlimited            bytes
Max stack size            8388608              unlimited            bytes
Max core file size        0                    unlimited            bytes
Max resident set          unlimited            unlimited            bytes
Max processes             15738                15738                processes
Max open files            1024                 4096                 files
Max locked memory         65536                65536                bytes
Max address space         unlimited            unlimited            bytes
Max file locks            unlimited            unlimited            locks
Max pending signals       15738                15738                signals
Max msgqueue size         819200               819200               bytes
Max nice priority         0                    0
```

rule text field denoting the rule. Details at PAGC Address Standardizer Rule records.

A rule consists of a set of non-negative integers representing input tokens, terminated by a -1, followed by an equal number of non-negative integers representing postal attributes, terminated by a -1, followed by an integer representing a rule type, followed by an integer representing the rank of the rule. The rules are ranked from 0 (lowest) to 17 (highest).

So for example the rule `2 0 2 22 3 -1 5 5 6 7 3 -1 2 6` maps to sequence of output tokens *TYPE NUMBER TYPE DIRECT QUALIF* to the output sequence *STREET STREET SUFTYP SUFDIR QUALIF*. The rule is an ARC_C rule of rank 6.

Numbers for corresponding output tokens are listed in stdaddr.

Input Tokens

Each rule starts with a set of input tokens followed by a terminator `-1`. Valid input tokens excerpted from PAGC Input Tokens are as follows:

Form-Based Input Tokens

AMPERS (13). The ampersand (&) is frequently used to abbreviate the word "and".

DASH (9). A punctuation character.

DOUBLE (21). A sequence of two letters. Often used as identifiers.

FRACT (25). Fractions are sometimes used in civic numbers or unit numbers.

MIXED (23). An alphanumeric string that contains both letters and digits. Used for identifiers.

NUMBER (0). A string of digits.

ORD (15). Representations such as First or 1st. Often used in street names.

ORD (18). A single letter.

WORD (1). A word is a string of letters of arbitrary length. A single letter can be both a SINGLE and a WORD.

Function-based Input Tokens

BOXH (14). Words used to denote post office boxes. For example *Box* or *PO Box*.

BUILDH (19). Words used to denote buildings or building complexes, usually as a prefix. For example: *Tower* in *Tower 7A*.

BUILDT (24). Words and abbreviations used to denote buildings or building complexes, usually as a suffix. For example: *Shopping Centre*.

DIRECT (22). Words used to denote directions, for example *North*.

MILE (20). Words used to denote milepost addresses.

ROAD (6). Words and abbreviations used to denote highways and roads. For example: the *Interstate* in *Interstate 5*

RR (8). Words and abbreviations used to denote rural routes. *RR*.

TYPE (2). Words and abbreviation used to denote street typess. For example: *ST* or *AVE*.

UNITH (16). Words and abbreviation used to denote internal subaddresses. For example, *APT* or *UNIT*.

Postal Type Input Tokens

QUINT (28). A 5 digit number. Identifies a Zip Code

QUAD (29). A 4 digit number. Identifies ZIP4.

PCH (27). A 3 character sequence of letter number letter. Identifies an FSA, the first 3 characters of a Canadian postal code.

PCT (26). A 3 character sequence of number letter number. Identifies an LDU, the last 3 characters of a Canadian postal code.

Stopwords

STOPWORDS combine with WORDS. In rules a string of multiple WORDs and STOPWORDs will be represented by a single WORD token.

STOPWORD (7). A word with low lexical significance, that can be omitted in parsing. For example: *THE*.

Output Tokens

After the first -1 (terminator), follows the output tokens and their order, followed by a terminator `-1`. Numbers for corresponding output tokens are listed in stdaddr. What are allowed is dependent on kind of rule. Output tokens valid for each rule type are listed in the section called "Rule Types and Rank".

Rule Types and Rank

The final part of the rule is the rule type which is denoted by one of the following, followed by a rule rank. The rules are ranked from 0 (lowest) to 17 (highest).

MACRO_C

(token number = "**0**"). The class of rules for parsing MACRO clauses such as *PLACE STATE ZIP*

MACRO_C output tokens (excerpted from http://www.pagcgeo.org/docs/html/pagc-12.html#--r-typ--.

CITY (token number "10"). Example "Albany"

STATE (token number "11"). Example "NY"

NATION (token number "12"). This attribute is not used in most reference files. Example "USA"

POSTAL (token number "13"). (SADS elements "ZIP CODE" , "PLUS 4"). This attribute is used for both the US Zip and the Canadian Postal Codes.

MICRO_C

(token number = "**1**"). The class of rules for parsing full MICRO clauses (such as House, street, sufdir, predir, pretyp, suftype, qualif) (ie ARC_C plus CIVIC_C). These rules are not used in the build phase.

MICRO_C output tokens (excerpted from http://www.pagcgeo.org/docs/html/pagc-12.html#--r-typ--.

HOUSE is a text (token number `1`): This is the street number on a street. Example *75* in `75 State Street`.

predir is text (token number `2`): STREET NAME PRE-DIRECTIONAL such as North, South, East, West etc.

qual is text (token number `3`): STREET NAME PRE-MODIFIER Example *OLD* in `3715 OLD HIGHWAY 99`.

pretype is text (token number `4`): STREET PREFIX TYPE

street is text (token number `5`): STREET NAME

suftype is text (token number `6`): STREET POST TYPE e.g. St, Ave, Cir. A street type following the root street name. Example *STREET* in `75 State Street`.

sufdir is text (token number `7`): STREET POST-DIRECTIONAL A directional modifier that follows the street name.. Example *WEST* in `3715 TENTH AVENUE WEST`.

ARC_C

(token number = "**2**"). The class of rules for parsing MICRO clauses, excluding the HOUSE attribute. As such uses same set of output tokens as MICRO_C minus the HOUSE token.

CIVIC_C

(token number = "**3**"). The class of rules for parsing the HOUSE attribute.

EXTRA_C

(token number = "**4**"). The class of rules for parsing EXTRA attributes - attributes excluded from geocoding. These rules are not used in the build phase.

EXTRA_C output tokens (excerpted from http://www.pagcgeo.org/docs/html/pagc-12.html#--r-typ--.

BLDNG (token number `0`): Unparsed building identifiers and types.

BOXH (token number `14`): The **BOX** in `BOX 3B`

BOXT (token number `15`): The **3B** in `BOX 3B`

RR (token number `8`): The **RR** in `RR 7`

UNITH (token number `16`): The **APT** in `APT 3B`

UNITT (token number `17`): The **3B** in `APT 3B`

UNKNWN (token number `9`): An otherwise unclassified output.

12.1.3.2 lex table

lex table — A lex table is used to classify alphanumeric input and associate that input with (a) input tokens (See the section called "Input Tokens") and (b) standardized representations.

Description

A lex (short for lexicon) table is used to classify alphanumeric input and associate that input with the section called "Input Tokens" and (b) standardized representations. Things you will find in these tables are `ONE` mapped to stdword: `1`.

A lex has at least the following columns in the table. You may add

id Primary key of table

seq integer: definition number?

word text: the input word

stdword text: the standardized replacement word

token integer: the kind of word it is. Only if it is used in this context will it be replaced. Refer to PAGC Tokens.

12.1.3.3 gaz table

gaz table — A gaz table is used to standardize place names and associate that input with (a) input tokens (See the section called "Input Tokens") and (b) standardized representations.

Description

A gaz (short for gazeteer) table is used to standardize place names and associate that input with the section called "Input Tokens" and (b) standardized representations. For example if you are in US, you may load these with State Names and associated abbreviations.

A gaz table has at least the following columns in the table. You may add more columns if you wish for your own purposes.

id Primary key of table

seq integer: definition number? - identifier used for that instance of the word

word text: the input word

stdword text: the standardized replacement word

token integer: the kind of word it is. Only if it is used in this context will it be replaced. Refer to PAGC Tokens.

12.1.4 Address Standardizer Functions

12.1.4.1 debug_standardize_address

debug_standardize_address — Returns a json formatted text listing the parse tokens and standardizations

Synopsis

text **debug_standardize_address**(text lextab, text gaztab, text rultab, text micro, text macro=NULL);

Description

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.

Examples

Using address_standardizer_data_us extension

```
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)
```

See Also

stdaddr, rules table, lex table, gaz table, Pagc_Normalize_Address

12.1.4.2 parse_address

parse_address — Takes a 1 line address and breaks into parts

Synopsis

record **parse_address**(text address);

Description

Returns takes an address as input, and returns a record output consisting of fields *num*, *street*, *street2*, *address1*, *city*, *state*, *zip*, *zipplus*, *country*.

Availability: 2.2.0

This method needs address_standardizer extension.

Examples

Single Address

```
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
```

Table of addresses

```
-- 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)
```

See Also

12.1.4.3 standardize_address

standardize_address — Returns an stdaddr form of an input address utilizing lex, gaz, and rule tables.

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);

Description

Returns an stdaddr form of an input address utilizing lex table table name, gaz table, and rules table table names and an address.

Variant 1: Takes an address as a single line.

Variant 2: Takes an address as 2 parts. 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`.

Availability: 2.2.0

This method needs address_standardizer extension.

Examples

Using address_standardizer_data_us extension

```
CREATE EXTENSION address_standardizer_data_us; -- only needs to be done once
```

Variant 1: Single line address. This doesn't work well with non-US addresses

```
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
```

Using tables packaged with tiger geocoder. This example only works if you installed `postgis_tiger_geocoder`.

```
SELECT *  FROM standardize_address('tiger.pagc_lex',
         'tiger.pagc_gaz', 'tiger.pagc_rules', 'One Devonshire Place, PH 301, Boston, MA  ↩
            02109-1234');
```

Make easier to read we'll dump output using hstore extension CREATE EXTENSION hstore; you need to install

```
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)
```

Variant 2: As a two part Address

```
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)
```

See Also

stdaddr, rules table, lex table, gaz table, Pagc_Normalize_Address

12.2 Tiger Geocoder

There are a couple other open source geocoders for PostGIS, that unlike tiger geocoder have the advantage of multi-country geocoding support

- Nominatim uses OpenStreetMap gazeteer formatted data. It requires osm2pgsql for loading the data, PostgreSQL 8.4+ and PostGIS 1.5+ to function. It is packaged as a webservice interface and seems designed to be called as a webservice. Just like the tiger geocoder, it has both a geocoder and a reverse geocoder component. From the documentation, it is unclear if it has a pure SQL interface like the tiger geocoder, or if a good deal of the logic is implemented in the web interface.
- GIS Graphy also utilizes PostGIS and like Nominatim works with OpenStreetMap (OSM) data. It comes with a loader to load OSM data and similar to Nominatim is capable of geocoding not just US. Much like Nominatim, it runs as a webservice and relies on Java 1.5, Servlet apps, Solr. GisGraphy is cross-platform and also has a reverse geocoder among some other neat features.

12.2.1 Drop_Indexes_Generate_Script

Drop_Indexes_Generate_Script — Generates a script that drops all non-primary key and non-unique indexes on tiger schema and user specified schema. Defaults schema to `tiger_data` if no schema is specified.

Synopsis

text **Drop_Indexes_Generate_Script**(text param_schema=tiger_data);

Description

Generates a script that drops all non-primary key and non-unique indexes on tiger schema and user specified schema. Defaults schema to `tiger_data` if no schema is specified.

This is useful for minimizing index bloat that may confuse the query planner or take up unnecessary space. Use in combination with Install_Missing_Indexes to add just the indexes used by the geocoder.

Availability: 2.0.0

Examples

```
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;
:
:
```

See Also

Install_Missing_Indexes, Missing_Indexes_Generate_Script

12.2.2 Drop_Nation_Tables_Generate_Script

Drop_Nation_Tables_Generate_Script — Generates a script that drops all tables in the specified schema that start with `county_all`, `state_all` or state code followed by `county` or `state`.

Synopsis

text **Drop_Nation_Tables_Generate_Script**(text param_schema=tiger_data);

See Also

Loader_Generate_Script

12.2.4 Geocode

Geocode — Takes in an address as a string (or other normalized address) and outputs a set of possible locations which include a point geometry in NAD 83 long lat, a normalized address for each, and the rating. The lower the rating the more likely the match. Results are sorted by lowest rating first. Can optionally pass in maximum results, defaults to 10, and restrict_region (defaults to 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);

Description

Takes in an address as a string (or already normalized address) and outputs a set of possible locations which include a point geometry in NAD 83 long lat, a `normalized_address` (addy) for each, and the rating. The lower the rating the more likely the match. Results are sorted by lowest rating first. Uses Tiger data (edges,faces,addr), PostgreSQL fuzzy string matching (soundex,levenshtein) and PostGIS line interpolation functions to interpolate address along the Tiger edges. The higher the rating the less likely the geocode is right. The geocoded point is defaulted to offset 10 meters from center-line off to side (L/R) of street address is located on.

Enhanced: 2.0.0 to support Tiger 2010 structured data and revised some logic to improve speed, accuracy of geocoding, and to offset point from centerline to side of street address is located on. The new parameter `max_results` useful for specifying number of best results or just returning the best result.

Examples: Basic

The below examples timings are on a 3.0 GHZ single processor Windows 7 machine with 2GB ram running PostgreSQL 9.1rc1/PostGIS 2.0 loaded with all of MA,MN,CA, RI state Tiger data loaded.

Exact matches are faster to compute (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
```

Even if zip is not passed in the geocoder can guess (took about 122-150 ms)

```
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
```

Can handle misspellings and provides more than one possible solution with ratings and takes longer (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
```

Using to do a batch geocode of addresses. Easiest is to set `max_results=1`. Only process those not yet geocoded (have no rating).

```
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)
```

Examples: Using Geometry filter

```
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
```

See Also

Normalize_Address, Pprint_Addy, ST_AsText, ST_SnapToGrid, ST_X, ST_Y

12.2.5 Geocode_Intersection

Geocode_Intersection — Takes in 2 streets that intersect and a state, city, zip, and outputs a set of possible locations on the first cross street that is at the intersection, also includes a geomout as the point location in NAD 83 long lat, a `normalized_address` (addy) for each location, and the rating. The lower the rating the more likely the match. Results are sorted by lowest rating first. Can optionally pass in maximum results, defaults to 10. Uses Tiger data (edges, faces, addr), PostgreSQL fuzzy string matching (soundex, levenshtein).

Synopsis

setof record **geocode_intersection**(text roadway1, text roadway2, text in_state, text in_city, text in_zip, integer max_results=10, norm_addy OUT addy, geometry OUT geomout, integer OUT rating);

Description

Takes in 2 streets that intersect and a state, city, zip, and outputs a set of possible locations on the first cross street that is at the intersection, also includes a point geometry in NAD 83 long lat, a normalized address for each location, and the rating. The lower the rating the more likely the match. Results are sorted by lowest rating first. Can optionally pass in maximum results, defaults to 10. Returns `normalized_address` (addy) for each, geomout as the point location in nad 83 long lat, and the rating. The lower the rating the more likely the match. Results are sorted by lowest rating first. Uses Tiger data (edges,faces,addr), PostgreSQL fuzzy string matching (soundex,levenshtein)

Availability: 2.0.0

Examples: Basic

The below examples timings are on a 3.0 GHZ single processor Windows 7 machine with 2GB ram running PostgreSQL 9.0/PostGIS 1.5 loaded with all of MA state Tiger data loaded. Currently a bit slow (3000 ms)

Testing on Windows 2003 64-bit 8GB on PostGIS 2.0 PostgreSQL 64-bit Tiger 2011 data loaded -- (41ms)

```
                                                                    instead of tiger  ↩
                                                                       normalize_address built ↩
                                                                        one
```

Changed: 2.2.0 : default settings are now kept in a table called geocode_settings_default. Use customized settingsa are in geocode_settings and only contain those that have been set by user.

Availability: 2.1.0

Example return debugging setting

```
SELECT get_geocode_setting('debug_geocode_address') As result;
result
---------
false
```

See Also

Set_Geocode_Setting

12.2.7 Get_Tract

Get_Tract — Returns census tract or field from tract table of where the geometry is located. Default to returning short name of tract.

Synopsis

text **get_tract**(geometry loc_geom, text output_field=name);

Description

Given a geometry will return the census tract location of that geometry. NAD 83 long lat is assumed if no spatial ref sys is specified.

Note

This function uses the census `tract` which is not loaded by default. If you have already loaded your state table, you can load tract as well as bg, and tabblock using the Loader_Generate_Census_Script script.
If you have not loaded your state data yet and want these additional tables loaded, do the following

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

then they will be included by the Loader_Generate_Script.

Availability: 2.0.0

Examples: Basic

```
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
```

See Also

Geocode>

12.2.8 Install_Missing_Indexes

Install_Missing_Indexes — Finds all tables with key columns used in geocoder joins and filter conditions that are missing used indexes on those columns and will add them.

Synopsis

boolean **Install_Missing_Indexes**();

Description

Finds all tables in `tiger` and `tiger_data` schemas with key columns used in geocoder joins and filters that are missing indexes on those columns and will output the SQL DDL to define the index for those tables and then execute the generated script. This is a helper function that adds new indexes needed to make queries faster that may have been missing during the load process. This function is a companion to Missing_Indexes_Generate_Script that in addition to generating the create index script, also executes it. It is called as part of the `update_geocode.sql` upgrade script.

Availability: 2.0.0

Examples

```
SELECT install_missing_indexes();
         install_missing_indexes
-------------------------
 t
```

See Also

Loader_Generate_Script, Missing_Indexes_Generate_Script

12.2.9 Loader_Generate_Census_Script

Loader_Generate_Census_Script — Generates a shell script for the specified platform for the specified states that will download Tiger census state tract, bg, and tabblocks data tables, stage and load into `tiger_data` schema. Each state script is returned as a separate record.

Synopsis

setof text **loader_generate_census_script**(text[] param_states, text os);

Description

Generates a shell script for the specified platform for the specified states that will download Tiger data census state `tract`, block groups `bg`, and `tabblocks` data tables, stage and load into `tiger_data` schema. Each state script is returned as a separate record.

It uses unzip on Linux (7-zip on Windows by default) and wget to do the downloading. It uses Section 4.7.2 to load in the data. Note the smallest unit it does is a whole state. It will only process the files in the staging and temp folders.

It uses the following control tables to control the process and different OS shell syntax variations.

1. `loader_variables` keeps track of various variables such as census site, year, data and staging schemas
2. `loader_platform` profiles of various platforms and where the various executables are located. Comes with windows and linux. More can be added.
3. `loader_lookuptables` each record defines a kind of table (state, county), whether to process records in it and how to load them in. Defines the steps to import data, stage data, add, removes columns, indexes, and constraints for each. Each table is prefixed with the state and inherits from a table in the tiger schema. e.g. creates `tiger_data.ma_faces` which inherits from `tiger.faces`

Availability: 2.0.0

Note
Loader_Generate_Script includes this logic, but if you installed tiger geocoder prior to PostGIS 2.0.0 alpha5, you'll need to run this on the states you have already done to get these additional tables.

Examples

Generate script to load up data for select states in Windows shell script format.

```
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');"
:
```

Generate sh script

```
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
:
:
```

See Also

Loader_Generate_Script

12.2.10 Loader_Generate_Script

Loader_Generate_Script — Generates a shell script for the specified platform for the specified states that will download Tiger data, stage and load into `tiger_data` schema. Each state script is returned as a separate record. Latest version supports Tiger 2010 structural changes and also loads census tract, block groups, and blocks tables.

Synopsis

setof text **loader_generate_script**(text[] param_states, text os);

Description

Generates a shell script for the specified platform for the specified states that will download Tiger data, stage and load into `tiger_data` schema. Each state script is returned as a separate record.

It uses unzip on Linux (7-zip on Windows by default) and wget to do the downloading. It uses Section 4.7.2 to load in the data. Note the smallest unit it does is a whole state, but you can overwrite this by downloading the files yourself. It will only process the files in the staging and temp folders.

It uses the following control tables to control the process and different OS shell syntax variations.

1. `loader_variables` keeps track of various variables such as census site, year, data and staging schemas
2. `loader_platform` profiles of various platforms and where the various executables are located. Comes with windows and linux. More can be added.
3. `loader_lookuptables` each record defines a kind of table (state, county), whether to process records in it and how to load them in. Defines the steps to import data, stage data, add, removes columns, indexes, and constraints for each. Each table is prefixed with the state and inherits from a table in the tiger schema. e.g. creates `tiger_data.ma_faces` which inherits from `tiger.faces`

Availability: 2.0.0 to support Tiger 2010 structured data and load census tract (tract), block groups (bg), and blocks (tabblocks) tables .

Note
If you are using pgAdmin 3, be warned that by default pgAdmin 3 truncates long text. To fix, change *File -> Options -> Query Tool -> Query Editor - > Max. characters per column* to larger than 50000 characters.

Examples

Using psql where gistest is your database and `/gisdata/data_load.sh` is the file to create with the shell commands to run.

```
psql -U postgres -h localhost -d gistest -A -t \
 -c "SELECT Loader_Generate_Script(ARRAY['MA'], 'gistest')" > /gisdata/data_load.sh;
```

Generate script to load up data for 2 states in Windows shell script format.

```
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
:
:
```

Generate sh script

```
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}/*.*
:
:
```

See Also

Section 2.4.1, Loader_Generate_Nation_Script, Drop_State_Tables_Generate_Script

12.2.11 Loader_Generate_Nation_Script

Loader_Generate_Nation_Script — Generates a shell script for the specified platform that loads in the county and state lookup tables.

Synopsis

text **loader_generate_nation_script**(text os);

Description

Generates a shell script for the specified platform that loads in the `county_all`, `county_all_lookup`, `state_all` tables into `tiger_data` schema. These inherit respectively from the `county`, `county_lookup`, `state` tables in `tiger` schema.

It uses unzip on Linux (7-zip on Windows by default) and wget to do the downloading. It uses Section 4.7.2 to load in the data.

It uses the following control tables `tiger.loader_platform`, `tiger.loader_variables`, and `tiger.loader_lookupt` to control the process and different OS shell syntax variations.

1. `loader_variables` keeps track of various variables such as census site, year, data and staging schemas
2. `loader_platform` profiles of various platforms and where the various executables are located. Comes with windows and linux/unix. More can be added.
3. `loader_lookuptables` each record defines a kind of table (state, county), whether to process records in it and how to load them in. Defines the steps to import data, stage data, add, removes columns, indexes, and constraints for each. Each table is prefixed with the state and inherits from a table in the tiger schema. e.g. creates `tiger_data.ma_faces` which inherits from `tiger.faces`

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.

Availability: 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
If you were running `tiger_2010` version and you want to reload as state with newer tiger data, you'll need to for the very first load generate and run drop statements Drop_Nation_Tables_Generate_Script before you run this script.

Examples

Generate script script to load nation data Windows.

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

Generate script to load up data for Linux/Unix systems.

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

See Also

Loader_Generate_Script, Drop_Nation_Tables_Generate_Script

12.2.12 Missing_Indexes_Generate_Script

Missing_Indexes_Generate_Script — Finds all tables with key columns used in geocoder joins that are missing indexes on those columns and will output the SQL DDL to define the index for those tables.

Synopsis

text **Missing_Indexes_Generate_Script**();

Description

Finds all tables in `tiger` and `tiger_data` schemas with key columns used in geocoder joins that are missing indexes on those columns and will output the SQL DDL to define the index for those tables. This is a helper function that adds new indexes needed to make queries faster that may have been missing during the load process. As the geocoder is improved, this function will be updated to accommodate new indexes being used. If this function outputs nothing, it means all your tables have what we think are the key indexes already in place.

Availability: 2.0.0

Examples

```
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);
```

See Also

Loader_Generate_Script, Install_Missing_Indexes

12.2.13 Normalize_Address

Normalize_Address — Given a textual street address, returns a composite `norm_addy` type that has road suffix, prefix and type standardized, street, streetname etc. broken into separate fields. This function will work with just the lookup data packaged with the tiger_geocoder (no need for tiger census data).

Synopsis

norm_addy **normalize_address**(varchar in_address);

Description

Given a textual street address, returns a composite `norm_addy` type that has road suffix, prefix and type standardized, street, streetname etc. broken into separate fields. This is the first step in the geocoding process to get all addresses into normalized postal form. No other data is required aside from what is packaged with the geocoder.

This function just uses the various direction/state/suffix lookup tables preloaded with the tiger_geocoder and located in the `tiger` schema, so it doesn't need you to download tiger census data or any other additional data to make use of it. You may find the need to add more abbreviations or alternative namings to the various lookup tables in the `tiger` schema.

It uses various control lookup tables located in `tiger` schema to normalize the input address.

Fields in the `norm_addy` type object returned by this function in this order where () indicates a field required by the geocoder, [] indicates an optional field:

(address) [predirAbbrev] (streetName) [streetTypeAbbrev] [postdirAbbrev] [internal] [location] [stateAbbrev] [zip] [parsed] [zip4] [address_alphanumeric]

Enhanced: 2.4.0 norm_addy object includes additional fields zip4 and address_alphanumeric.

1. `address` is an integer: The street number
2. `predirAbbrev` is varchar: Directional prefix of road such as N, S, E, W etc. These are controlled using the `direction_look` table.
3. `streetName` varchar
4. `streetTypeAbbrev` varchar abbreviated version of street type: e.g. St, Ave, Cir. These are controlled using the `street_type_lookup` table.
5. `postdirAbbrev` varchar abbreviated directional suffice of road N, S, E, W etc. These are controlled using the `direction_lo` table.
6. `internal` varchar internal address such as an apartment or suite number.

7. `location` varchar usually a city or governing province.
8. `stateAbbrev` varchar two character US State. e.g MA, NY, MI. These are controlled by the `state_lookup` table.
9. `zip` varchar 5-digit zipcode. e.g. 02109.
10. `parsed` boolean - denotes if address was formed from normalize process. The normalize_address function sets this to true before returning the address.
11. `zip4` last 4 digits of a 9 digit zip code. Availability: PostGIS 2.4.0.
12. `address_alphanumeric` Full street number even if it has alpha characters like 17R. Parsing of this is better using Pagc_Normalize_Address function. Availability: PostGIS 2.4.0.

Examples

Output select fields. Use Pprint_Addy if you want a pretty textual output.

```
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  |
```

See Also

Geocode, Pprint_Addy

12.2.14 Pagc_Normalize_Address

Pagc_Normalize_Address — Given a textual street address, returns a composite `norm_addy` type that has road suffix, prefix and type standardized, street, streetname etc. broken into separate fields. This function will work with just the lookup data packaged with the tiger_geocoder (no need for tiger census data). Requires address_standardizer extension.

Synopsis

norm_addy **pagc_normalize_address**(varchar in_address);

Description

Given a textual street address, returns a composite `norm_addy` type that has road suffix, prefix and type standardized, street, streetname etc. broken into separate fields. This is the first step in the geocoding process to get all addresses into normalized postal form. No other data is required aside from what is packaged with the geocoder.

This function just uses the various pagc_* lookup tables preloaded with the tiger_geocoder and located in the `tiger` schema, so it doesn't need you to download tiger census data or any other additional data to make use of it. You may find the need to add more abbreviations or alternative namings to the various lookup tables in the `tiger` schema.

It uses various control lookup tables located in `tiger` schema to normalize the input address.

Fields in the norm_addy type object returned by this function in this order where () indicates a field required by the geocoder, [] indicates an optional field:

There are slight variations in casing and formatting over the Normalize_Address.

Availability: 2.1.0

This method needs address_standardizer extension.

(address) [predirAbbrev] (streetName) [streetTypeAbbrev] [postdirAbbrev] [internal] [location] [stateAbbrev] [zip]

The native standardaddr of address_standardizer extension is at this time a bit richer than norm_addy since its designed to support international addresses (including country). standardaddr equivalent fields are:

house_num,predir, name, suftype, sufdir, unit, city, state, postcode

Enhanced: 2.4.0 norm_addy object includes additional fields zip4 and address_alphanumeric.

1. address is an integer: The street number
2. predirAbbrev is varchar: Directional prefix of road such as N, S, E, W etc. These are controlled using the direction_look table.
3. streetName varchar
4. streetTypeAbbrev varchar abbreviated version of street type: e.g. St, Ave, Cir. These are controlled using the street_type_lookup table.
5. postdirAbbrev varchar abbreviated directional suffice of road N, S, E, W etc. These are controlled using the direction_lo table.
6. internal varchar internal address such as an apartment or suite number.
7. location varchar usually a city or governing province.
8. stateAbbrev varchar two character US State. e.g MA, NY, MI. These are controlled by the state_lookup table.
9. zip varchar 5-digit zipcode. e.g. 02109.
10. parsed boolean - denotes if address was formed from normalize process. The normalize_address function sets this to true before returning the address.
11. zip4 last 4 digits of a 9 digit zip code. Availability: PostGIS 2.4.0.
12. address_alphanumeric Full street number even if it has alpha characters like 17R. Parsing of this is better using Pagc_Normalize_Address function. Availability: PostGIS 2.4.0.

Examples

Single call example

```
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. There are currently speed issues with the way postgis_tiger_geocoder wraps the address_standardizer. These will hopefully be resolved in later editions. To work around them, if you need speed for batch geocoding to call generate a normaddy in batch mode, you are encouraged to directly call the address_standardizer standardize_address function as shown below which is similar exercise to what we did in Normalize_Address that uses data created in Geocode.

```
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
```

See Also

Normalize_Address, Geocode

12.2.15 Pprint_Addy

Pprint_Addy — Given a `norm_addy` composite type object, returns a pretty print representation of it. Usually used in conjunction with normalize_address.

Synopsis

varchar **pprint_addy**(norm_addy in_addy);

Description

Given a `norm_addy` composite type object, returns a pretty print representation of it. No other data is required aside from what is packaged with the geocoder.

Usually used in conjunction with Normalize_Address.

Examples

Pretty print a single address

```
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
```

Pretty print address a table of addresses

```
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
```

See Also

Normalize_Address

12.2.16 Reverse_Geocode

Reverse_Geocode — Takes a geometry point in a known spatial ref sys and returns a record containing an array of theoretically possible addresses and an array of cross streets. If include_strnum_range = true, includes the street range in the cross streets.

Synopsis

record **Reverse_Geocode**(geometry pt, boolean include_strnum_range=false, geometry[] OUT intpt, norm_addy[] OUT addy, varchar[] OUT street);

Description

Takes a geometry point in a known spatial ref and returns a record containing an array of theoretically possible addresses and an array of cross streets. If include_strnum_range = true, includes the street range in the cross streets. include_strnum_range defaults to false if not passed in. Addresses are sorted according to which road a point is closest to so first address is most likely the right one.

Why do we say theoretical instead of actual addresses. The Tiger data doesn't have real addresses, but just street ranges. As such the theoretical address is an interpolated address based on the street ranges. Like for example interpolating one of my addresses returns a 26 Court St. and 26 Court Sq., though there is no such place as 26 Court Sq. This is because a point may be at a corner of 2 streets and thus the logic interpolates along both streets. The logic also assumes addresses are equally spaced along a street, which of course is wrong since you can have a municipal building taking up a good chunk of the street range and the rest of the buildings are clustered at the end.

Note: Hmm this function relies on Tiger data. If you have not loaded data covering the region of this point, then hmm you will get a record filled with NULLS.

Returned elements of the record are as follows:

1. `intpt` is an array of points: These are the center line points on the street closest to the input point. There are as many points as there are addresses.
2. `addy` is an array of norm_addy (normalized addresses): These are an array of possible addresses that fit the input point. The first one in the array is most likely. Generally there should be only one, except in the case when a point is at the corner of 2 or 3 streets, or the point is somewhere on the road and not off to the side.
3. `street` an array of varchar: These are cross streets (or the street) (streets that intersect or are the street the point is projected to be on).

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.

Availability: 2.0.0

Examples

Example of a point at the corner of two streets, but closest to one. This is approximate location of MIT: 77 Massachusetts Ave, Cambridge, MA 02139 Note that although we don't have 3 streets, PostgreSQL will just return null for entries above our upper bound so safe to use. This includes street ranges

```
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
```

Here we choose not to include the address ranges for the cross streets and picked a location really really close to a corner of 2 streets thus could be known by two different addresses.

```
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
```

For this one we reuse our geocoded example from Geocode and we only want the primary address and at most 2 cross streets.

```
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
```

See Also

Pprint_Addy, Geocode, Loader_Generate_Nation_Script

12.2.17 Topology_Load_Tiger

Topology_Load_Tiger — Loads a defined region of tiger data into a PostGIS Topology and transforming the tiger data to spatial reference of the topology and snapping to the precision tolerance of the topology.

Synopsis

text **Topology_Load_Tiger**(varchar topo_name, varchar region_type, varchar region_id);

Description

Loads a defined region of tiger data into a PostGIS Topology. The faces, nodes and edges are transformed to the spatial reference system of the target topology and points are snapped to the tolerance of the target topology. The created faces, nodes, edges maintain the same ids as the original Tiger data faces, nodes, edges so that datasets can be in the future be more easily reconciled with tiger data. Returns summary details about the process.

This would be useful for example for redistricting data where you require the newly formed polygons to follow the center lines of streets and for the resulting polygons not to overlap.

Note
This function relies on Tiger data as well as the installation of the PostGIS topology module. For more information, refer to Chapter 9 and Section 2.2.3. If you have not loaded data covering the region of interest, then no topology records will be created. This function will also fail if you have not created a topology using the topology functions.

Note
Most topology validation errors are a result of tolerance issues where after transformation the edges points don't quite line up or overlap. To remedy the situation you may want to increase or lower the precision if you get topology validation failures.

Required arguments:

1. `topo_name` The name of an existing PostGIS topology to load data into.
2. `region_type` The type of bounding region. Currently only `place` and `county` are supported. Plan is to have several more. This is the table to look into to define the region bounds. e.g `tiger.place`, `tiger.county`
3. `region_id` This is what TIGER calls the geoid. It is the unique identifier of the region in the table. For place it is the `plcidfp` column in `tiger.place`. For county it is the `cntyidfp` column in `tiger.county`

Availability: 2.0.0

Example: Boston, Massachusetts Topology

Create a topology for Boston, Massachusetts in Mass State Plane Feet (2249) with tolerance 0.25 feet and then load in Boston city tiger faces, edges, nodes.

```
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
-------------------+----------+-----------
```

Example: Suffolk, Massachusetts Topology

Create a topology for Suffolk, Massachusetts in Mass State Plane Meters (26986) with tolerance 0.25 meters and then load in Suffolk county tiger faces, edges, nodes.

```
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
```

See Also

CreateTopology, CreateTopoGeom, TopologySummary, ValidateTopology

12.2.18 Set_Geocode_Setting

Set_Geocode_Setting — Sets a setting that affects behavior of geocoder functions.

Synopsis

text **Set_Geocode_Setting**(text setting_name, text setting_value);

Description

Sets value of specific setting stored in `tiger.geocode_settings` table. Settings allow you to toggle debugging of functions. Later plans will be to control rating with settings. Current list of settings are listed in Get_Geocode_Setting.

Availability: 2.1.0

Example return debugging setting

If you run Geocode when this function is true, the NOTICE log will output timing and queries.

```
SELECT set_geocode_setting('debug_geocode_address', 'true') As result;
result
---------
true
```

See Also

Get_Geocode_Setting

Chapter 13

PostGIS Special Functions Index

13.1 PostGIS Aggregate Functions

The functions below are spatial aggregate functions that are used in the same way as SQL aggregate function such as `sum` and `average`.

- CG_3DUnion - Perform 3D union using postgis_sfcgal.
- ST_3DExtent - Aggregate function that returns the 3D bounding box of geometries.
- ST_3DUnion - Perform 3D union.
- ST_AsFlatGeobuf - Return a FlatGeobuf representation of a set of rows.
- ST_AsGeobuf - Return a Geobuf representation of a set of rows.
- ST_AsMVT - Aggregate function returning a MVT representation of a set of rows.
- ST_ClusterDBSCAN - Window function that returns a cluster id for each input geometry using the DBSCAN algorithm.
- ST_ClusterIntersecting - Aggregate function that clusters input geometries into connected sets.
- ST_ClusterIntersectingWin - Window function that returns a cluster id for each input geometry, clustering input geometries into connected sets.
- ST_ClusterKMeans - Window function that returns a cluster id for each input geometry using the K-means algorithm.
- ST_ClusterWithin - Aggregate function that clusters geometries by separation distance.
- ST_ClusterWithinWin - Window function that returns a cluster id for each input geometry, clustering using separation distance.
- ST_Collect - Creates a GeometryCollection or Multi* geometry from a set of geometries.
- ST_CoverageInvalidEdges - Window function that finds locations where polygons fail to form a valid coverage.
- ST_CoverageSimplify - Window function that simplifies the edges of a polygonal coverage.
- ST_CoverageUnion - Computes the union of a set of polygons forming a coverage by removing shared edges.
- ST_Extent - Aggregate function that returns the bounding box of geometries.
- ST_MakeLine - Creates a LineString from Point, MultiPoint, or LineString geometries.
- 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 - Returns true if rasters have same skew, scale, spatial ref, and offset (pixels can be put on same grid without cutting into pixels) and false if they don't with notice detailing issue.
- ST_Union - Computes a geometry representing the point-set union of the input geometries.
- ST_Union - Returns the union of a set of raster tiles into a single raster composed of 1 or more bands.
- TopoElementArray_Agg - Returns a topoelementarray for a set of element_id, type arrays (topoelements).

13.2 PostGIS Window Functions

The functions below are spatial window functions that are used in the same way as SQL window functions such as `row_number()`, `lead()`, and `lag()`. They must be followed by an `OVER()` clause.

- ST_ClusterDBSCAN - Window function that returns a cluster id for each input geometry using the DBSCAN algorithm.
- ST_ClusterIntersectingWin - Window function that returns a cluster id for each input geometry, clustering input geometries into connected sets.
- ST_ClusterKMeans - Window function that returns a cluster id for each input geometry using the K-means algorithm.
- ST_ClusterWithinWin - Window function that returns a cluster id for each input geometry, clustering using separation distance.
- ST_CoverageInvalidEdges - Window function that finds locations where polygons fail to form a valid coverage.
- ST_CoverageSimplify - Window function that simplifies the edges of a polygonal coverage.

13.3 PostGIS SQL-MM Compliant Functions

The functions given below are PostGIS functions that conform to the SQL/MM 3 standard

- CG_3DArea - Computes area of 3D surface geometries. Will return 0 for solids.
- CG_3DDifference - Perform 3D difference
- CG_3DIntersection - Perform 3D intersection
- CG_3DUnion - Perform 3D union using postgis_sfcgal.
- CG_Volume - Computes the volume of a 3D solid. If applied to surface (even closed) geometries will return 0.
- ST_3DArea - Computes area of 3D surface geometries. Will return 0 for solids.
- ST_3DDWithin - Tests if two 3D geometries are within a given 3D distance
- ST_3DDifference - Perform 3D difference
- ST_3DDistance - Returns the 3D cartesian minimum distance (based on spatial ref) between two geometries in projected units.
- 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 - Returns the 3D length of a linear geometry.
- ST_3DPerimeter - Returns the 3D perimeter of a polygonal geometry.
- ST_3DUnion - Perform 3D union.
- ST_AddEdgeModFace - Add a new edge and, if in doing so it splits a face, modify the original face and add a new face.

- ST_GeometryN - Return an element of a geometry collection.
- ST_GeometryType - Returns the SQL-MM type of a geometry as text.
- ST_GetFaceEdges - Returns a set of ordered edges that bound aface.
- ST_GetFaceGeometry - Returns the polygon in the given topology with the specified face id.
- ST_InitTopoGeo - Creates a new topology schema and registers it in the topology.topology table.
- ST_InteriorRingN - Returns the Nth interior ring (hole) of a Polygon.
- ST_Intersection - Computes a geometry representing the shared portion of geometries A and B.
- ST_Intersects - Tests if two geometries intersect (they have at least one point in common)
- ST_IsClosed - Tests if a LineStrings's start and end points are coincident. For a PolyhedralSurface tests if it is closed (volumetric).
- ST_IsEmpty - Tests if a geometry is empty.
- ST_IsRing - Tests if a LineString is closed and simple.
- ST_IsSimple - Tests if a geometry has no points of self-intersection or self-tangency.
- ST_IsValid - Tests if a geometry is well-formed in 2D.
- ST_Length - Returns the 2D length of a linear geometry.
- ST_LineFromText - Makes a Geometry from WKT representation with the given SRID. If SRID is not given, it defaults to 0.
- ST_LineFromWKB - Makes a LINESTRING from WKB with the given SRID
- ST_LinestringFromWKB - Makes a geometry from WKB with the given SRID.
- ST_LocateAlong - Returns the point(s) on a geometry that match a measure value.
- ST_LocateBetween - Returns the portions of a geometry that match a measure range.
- ST_M - Returns the M coordinate of a Point.
- ST_MLineFromText - Return a specified ST_MultiLineString value from WKT representation.
- ST_MPointFromText - Makes a Geometry from WKT with the given SRID. If SRID is not given, it defaults to 0.
- ST_MPolyFromText - Makes a MultiPolygon Geometry from WKT with the given SRID. If SRID is not given, it defaults to 0.
- ST_ModEdgeHeal - Heals two edges by deleting the node connecting them, modifying the first edgeand deleting the second edge. Returns the id of the deleted node.
- ST_ModEdgeSplit - Split an edge by creating a new node along an existing edge, modifying the original edge and adding a new edge.
- ST_MoveIsoNode - Moves an isolated node in a topology from one point to another. If new apoint geometry exists as a node an error is thrown. Returns description of move.
- ST_NewEdgeHeal - Heals two edges by deleting the node connecting them, deleting both edges,and replacing them with an edge whose direction is the same as the firstedge provided.
- ST_NewEdgesSplit - Split an edge by creating a new node along an existing edge, deleting the original edge and replacing it with two new edges. Returns the id of the new node created that joins the new edges.
- ST_NumCurves - Return the number of component curves in a CompoundCurve.
- ST_NumGeometries - Returns the number of elements in a geometry collection.
- ST_NumInteriorRings - Returns the number of interior rings (holes) of a Polygon.

- ST_NumPatches - Return the number of faces on a Polyhedral Surface. Will return null for non-polyhedral geometries.
- ST_NumPoints - Returns the number of points in a LineString or CircularString.
- ST_OrderingEquals - Tests if two geometries represent the same geometry and have points in the same directional order
- ST_Overlaps - Tests if two geometries have the same dimension and intersect, but each has at least one point not in the other
- ST_PatchN - Returns the Nth geometry (face) of a PolyhedralSurface.
- ST_Perimeter - Returns the length of the boundary of a polygonal geometry or geography.
- ST_Point - Creates a Point with X, Y and SRID values.
- ST_PointFromText - Makes a point Geometry from WKT with the given SRID. If SRID is not given, it defaults to unknown.
- ST_PointFromWKB - Makes a geometry from WKB with the given SRID
- ST_PointN - Returns the Nth point in the first LineString or circular LineString in a geometry.
- ST_PointOnSurface - Computes a point guaranteed to lie in a polygon, or on a geometry.
- ST_Polygon - Creates a Polygon from a LineString with a specified SRID.
- ST_PolygonFromText - Makes a Geometry from WKT with the given SRID. If SRID is not given, it defaults to 0.
- ST_Relate - Tests if two geometries have a topological relationship matching an Intersection Matrix pattern, or computes their Intersection Matrix
- ST_RemEdgeModFace - Removes an edge, and if the edge separates two facesdeletes one face and modifies the other face to cover the space of both.
- ST_RemEdgeNewFace - Removes an edge and, if the removed edge separated two faces,delete the original faces and replace them with a new face.
- ST_RemoveIsoEdge - Removes an isolated edge and returns description of action. If the edge is not isolated, then an exception is thrown.
- ST_RemoveIsoNode - Removes an isolated node and returns description of action. If the node is not isolated (is start or end of an edge), then an exception is thrown.
- ST_SRID - Returns the spatial reference identifier for a geometry.
- ST_StartPoint - Returns the first point of a LineString.
- ST_SymDifference - Computes a geometry representing the portions of geometries A and B that do not intersect.
- ST_Touches - Tests if two geometries have at least one point in common, but their interiors do not intersect
- ST_Transform - Return a new geometry with coordinates transformed to a different spatial reference system.
- ST_Union - Computes a geometry representing the point-set union of the input geometries.
- ST_Volume - Computes the volume of a 3D solid. If applied to surface (even closed) geometries will return 0.
- ST_WKBToSQL - Return a specified ST_Geometry value from Well-Known Binary representation (WKB). This is an alias name for ST_GeomFromWKB that takes no srid
- ST_WKTToSQL - Return a specified ST_Geometry value from Well-Known Text representation (WKT). This is an alias name for ST_GeomFromText
- ST_Within - Tests if every point of A lies in B, and their interiors have a point in common
- ST_X - Returns the X coordinate of a Point.
- ST_Y - Returns the Y coordinate of a Point.
- ST_Z - Returns the Z coordinate of a Point.
- ST_SRID - Returns the spatial reference identifier for a topogeometry.

13.4 PostGIS Geography Support Functions

The functions and operators given below are PostGIS functions/operators that take as input or return as output a geography data type object.

Note
Functions with a (T) are not native geodetic functions, and use a ST_Transform call to and from geometry to do the operation. As a result, they may not behave as expected when going over dateline, poles, and for large geometries or geometry pairs that cover more than one UTM zone. Basic transform - (favoring UTM, Lambert Azimuthal (North/South), and falling back on mercator in worst case scenario)

- ST_Area - Returns the area of a polygonal geometry.
- ST_AsBinary - Return the OGC/ISO Well-Known Binary (WKB) representation of the geometry/geography without SRID meta data.
- ST_AsEWKT - Return the Well-Known Text (WKT) representation of the geometry with SRID meta data.
- ST_AsGML - Return the geometry as a GML version 2 or 3 element.
- ST_AsGeoJSON - Return a geometry or feature in GeoJSON format.
- ST_AsKML - Return the geometry as a KML element.
- ST_AsSVG - Returns SVG path data for a geometry.
- ST_AsText - Return the Well-Known Text (WKT) representation of the geometry/geography without SRID metadata.
- ST_Azimuth - Returns the north-based azimuth of a line between two points.
- ST_Buffer - Computes a geometry covering all points within a given distance from a geometry.
- ST_Centroid - Returns the geometric center of a geometry.
- 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 - Returns the distance between two geometry or geography values.
- ST_GeogFromText - Return a specified geography value from Well-Known Text representation or extended (WKT).
- ST_GeogFromWKB - Creates a geography instance from a Well-Known Binary geometry representation (WKB) or extended Well Known Binary (EWKB).
- ST_GeographyFromText - Return a specified geography value from Well-Known Text representation or extended (WKT).
- = - Returns TRUE if the coordinates and coordinate order geometry/geography A are the same as the coordinates and coordinate order of geometry/geography B.
- ST_Intersection - Computes a geometry representing the shared portion of geometries A and B.
- ST_Intersects - Tests if two geometries intersect (they have at least one point in common)
- ST_Length - Returns the 2D length of a linear geometry.
- ST_LineInterpolatePoint - Returns a point interpolated along a line at a fractional location.

- ST_LineInterpolatePoints - Returns points interpolated along a line at a fractional interval.
- ST_LineLocatePoint - Returns the fractional location of the closest point on a line to a point.
- ST_LineSubstring - Returns the part of a line between two fractional locations.
- ST_Perimeter - Returns the length of the boundary of a polygonal geometry or geography.
- ST_Project - Returns a point projected from a start point by a distance and bearing (azimuth).
- ST_Segmentize - Returns a modified geometry/geography having no segment longer than a given distance.
- ST_ShortestLine - Returns the 2D shortest line between two geometries
- ST_Summary - Returns a text summary of the contents of a geometry.
- <-> - Returns the 2D distance between A and B.
- && - Returns TRUE if A's 2D bounding box intersects B's 2D bounding box.

13.5 PostGIS Raster Support Functions

The functions and operators given below are PostGIS functions/operators that take as input or return as output a raster data type object. Listed in alphabetical order.

- Box3D - Returns the box 3d representation of the enclosing box of the raster.
- @ - Returns TRUE if A's bounding box is contained by B's. Uses double precision bounding box.
- ~ - Returns TRUE if A's bounding box is contains B's. Uses double precision bounding box.
- = - Returns TRUE if A's bounding box is the same as B's. Uses double precision bounding box.
- && - Returns TRUE if A's bounding box intersects B's bounding box.
- &< - Returns TRUE if A's bounding box is to the left of B's.
- &> - Returns TRUE if A's bounding box is to the right of B's.
- ~= - Returns TRUE if A's bounding box is the same as B's.
- ST_Retile - Return a set of configured tiles from an arbitrarily tiled raster coverage.
- ST_AddBand - Returns a raster with the new band(s) of given type added with given initial value in the given index location. If no index is specified, the band is added to the end.
- ST_AsBinary/ST_AsWKB - Return the Well-Known Binary (WKB) representation of the raster.
- ST_AsGDALRaster - Return the raster tile in the designated GDAL Raster format. Raster formats are one of those supported by your compiled library. Use ST_GDALDrivers() to get a list of formats supported by your library.
- ST_AsHexWKB - Return the Well-Known Binary (WKB) in Hex representation of the raster.
- ST_AsJPEG - Return the raster tile selected bands as a single Joint Photographic Exports Group (JPEG) image (byte array). If no band is specified and 1 or more than 3 bands, then only the first band is used. If only 3 bands then all 3 bands are used and mapped to RGB.
- ST_AsPNG - Return the raster tile selected bands as a single portable network graphics (PNG) image (byte array). If 1, 3, or 4 bands in raster and no bands are specified, then all bands are used. If more 2 or more than 4 bands and no bands specified, then only band 1 is used. Bands are mapped to RGB or RGBA space.
- ST_AsRaster - Converts a PostGIS geometry to a PostGIS raster.

- ST_AsTIFF - Return the raster selected bands as a single TIFF image (byte array). If no band is specified or any of specified bands does not exist in the raster, then will try to use all bands.
- ST_Aspect - Returns the aspect (in degrees by default) of an elevation raster band. Useful for analyzing terrain.
- ST_Band - Returns one or more bands of an existing raster as a new raster. Useful for building new rasters from existing rasters.
- ST_BandFileSize - Returns the file size of a band stored in file system. If no bandnum specified, 1 is assumed.
- ST_BandFileTimestamp - Returns the file timestamp of a band stored in file system. If no bandnum specified, 1 is assumed.
- ST_BandIsNoData - Returns true if the band is filled with only nodata values.
- ST_BandMetaData - Returns basic meta data for a specific raster band. band num 1 is assumed if none-specified.
- ST_BandNoDataValue - Returns the value in a given band that represents no data. If no band num 1 is assumed.
- ST_BandPath - Returns system file path to a band stored in file system. If no bandnum specified, 1 is assumed.
- ST_BandPixelType - Returns the type of pixel for given band. If no bandnum specified, 1 is assumed.
- ST_Clip - Returns the raster clipped by the input geometry. If band number is not specified, all bands are processed. If crop is not specified or TRUE, the output raster is cropped. If touched is set to TRUE, then touched pixels are included, otherwise only if the center of the pixel is in the geometry it is included.
- ST_ColorMap - Creates a new raster of up to four 8BUI bands (grayscale, RGB, RGBA) from the source raster and a specified band. Band 1 is assumed if not specified.
- ST_Contains - Return true if no points of raster rastB lie in the exterior of raster rastA and at least one point of the interior of rastB lies in the interior of rastA.
- ST_ContainsProperly - Return true if rastB intersects the interior of rastA but not the boundary or exterior of rastA.
- ST_Contour - Generates a set of vector contours from the provided raster band, using the GDAL contouring algorithm.
- ST_ConvexHull - Return the convex hull geometry of the raster including pixel values equal to BandNoDataValue. For regular shaped and non-skewed rasters, this gives the same result as ST_Envelope so only useful for irregularly shaped or skewed rasters.
- ST_Count - Returns the number of pixels in a given band of a raster or raster coverage. If no band is specified defaults to band 1. If exclude_nodata_value is set to true, will only count pixels that are not equal to the nodata value.
- ST_CountAgg - Aggregate. Returns the number of pixels in a given band of a set of rasters. If no band is specified defaults to band 1. If exclude_nodata_value is set to true, will only count pixels that are not equal to the NODATA value.
- ST_CoveredBy - Return true if no points of raster rastA lie outside raster rastB.
- ST_Covers - Return true if no points of raster rastB lie outside raster rastA.
- ST_DFullyWithin - Return true if rasters rastA and rastB are fully within the specified distance of each other.
- ST_DWithin - Return true if rasters rastA and rastB are within the specified distance of each other.
- ST_Disjoint - Return true if raster rastA does not spatially intersect rastB.
- ST_DumpAsPolygons - Returns a set of geomval (geom,val) rows, from a given raster band. If no band number is specified, band num defaults to 1.
- ST_DumpValues - Get the values of the specified band as a 2-dimension array.
- ST_Envelope - Returns the polygon representation of the extent of the raster.
- ST_FromGDALRaster - Returns a raster from a supported GDAL raster file.
- ST_GeoReference - Returns the georeference meta data in GDAL or ESRI format as commonly seen in a world file. Default is GDAL.

- ST_Grayscale - Creates a new one-8BUI band raster from the source raster and specified bands representing Red, Green and Blue
- ST_HasNoBand - Returns true if there is no band with given band number. If no band number is specified, then band number 1 is assumed.
- ST_Height - Returns the height of the raster in pixels.
- ST_HillShade - Returns the hypothetical illumination of an elevation raster band using provided azimuth, altitude, brightness and scale inputs.
- ST_Histogram - Returns a set of record summarizing a raster or raster coverage data distribution separate bin ranges. Number of bins are autocomputed if not specified.
- 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 - Returns a raster or a set of geometry-pixelvalue pairs representing the shared portion of two rasters or the geometrical intersection of a vectorization of the raster and a geometry.
- ST_Intersects - Return true if raster rastA spatially intersects raster rastB.
- ST_IsEmpty - Returns true if the raster is empty (width = 0 and height = 0). Otherwise, returns false.
- ST_MakeEmptyCoverage - Cover georeferenced area with a grid of empty raster tiles.
- ST_MakeEmptyRaster - Returns an empty raster (having no bands) of given dimensions (width & height), upperleft X and Y, pixel size and rotation (scalex, scaley, skewx & skewy) and reference system (srid). If a raster is passed in, returns a new raster with the same size, alignment and SRID. If srid is left out, the spatial ref is set to unknown (0).
- ST_MapAlgebra (callback function version) - Callback function version - Returns a one-band raster given one or more input rasters, band indexes and one user-specified callback function.
- ST_MapAlgebraExpr - 1 raster band version: Creates a new one band raster formed by applying a valid PostgreSQL algebraic operation on the input raster band and of pixeltype provided. Band 1 is assumed if no band is specified.
- ST_MapAlgebraExpr - 2 raster band version: Creates a new one band raster formed by applying a valid PostgreSQL algebraic operation on the two input raster bands and of pixeltype provided. band 1 of each raster is assumed if no band numbers are specified. The resulting raster will be aligned (scale, skew and pixel corners) on the grid defined by the first raster and have its extent defined by the "extenttype" parameter. Values for "extenttype" can be: INTERSECTION, UNION, FIRST, SECOND.
- ST_MapAlgebraFct - 1 band version - Creates a new one band raster formed by applying a valid PostgreSQL function on the input raster band and of pixeltype prodived. Band 1 is assumed if no band is specified.
- ST_MapAlgebraFct - 2 band version - Creates a new one band raster formed by applying a valid PostgreSQL function on the 2 input raster bands and of pixeltype prodived. Band 1 is assumed if no band is specified. Extent type defaults to INTERSECTION if not specified.
- ST_MapAlgebraFctNgb - 1-band version: Map Algebra Nearest Neighbor using user-defined PostgreSQL function. Return a raster which values are the result of a PLPGSQL user function involving a neighborhood of values from the input raster band.
- ST_MapAlgebra (expression version) - Expression version - Returns a one-band raster given one or two input rasters, band indexes and one or more user-specified SQL expressions.
- ST_MemSize - Returns the amount of space (in bytes) the raster takes.
- ST_MetaData - Returns basic meta data about a raster object such as pixel size, rotation (skew), upper, lower left, etc.
- ST_MinConvexHull - Return the convex hull geometry of the raster excluding NODATA pixels.
- ST_NearestValue - Returns the nearest non-NODATA value of a given band's pixel specified by a columnx and rowy or a geometric point expressed in the same spatial reference coordinate system as the raster.
- ST_Neighborhood - Returns a 2-D double precision array of the non-NODATA values around a given band's pixel specified by either a columnX and rowY or a geometric point expressed in the same spatial reference coordinate system as the raster.

- ST_NotSameAlignmentReason - Returns text stating if rasters are aligned and if not aligned, a reason why.
- ST_NumBands - Returns the number of bands in the raster object.
- ST_Overlaps - Return true if raster rastA and rastB intersect but one does not completely contain the other.
- ST_PixelAsCentroid - Returns the centroid (point geometry) of the area represented by a pixel.
- ST_PixelAsCentroids - Returns the centroid (point geometry) for each pixel of a raster band along with the value, the X and the Y raster coordinates of each pixel. The point geometry is the centroid of the area represented by a pixel.
- ST_PixelAsPoint - Returns a point geometry of the pixel's upper-left corner.
- ST_PixelAsPoints - Returns a point geometry for each pixel of a raster band along with the value, the X and the Y raster coordinates of each pixel. The coordinates of the point geometry are of the pixel's upper-left corner.
- ST_PixelAsPolygon - Returns the polygon geometry that bounds the pixel for a particular row and column.
- ST_PixelAsPolygons - Returns the polygon geometry that bounds every pixel of a raster band along with the value, the X and the Y raster coordinates of each pixel.
- ST_PixelHeight - Returns the pixel height in geometric units of the spatial reference system.
- ST_PixelOfValue - Get the columnx, rowy coordinates of the pixel whose value equals the search value.
- ST_PixelWidth - Returns the pixel width in geometric units of the spatial reference system.
- ST_Polygon - Returns a multipolygon geometry formed by the union of pixels that have a pixel value that is not no data value. If no band number is specified, band num defaults to 1.
- ST_Quantile - Compute quantiles for a raster or raster table coverage in the context of the sample or population. Thus, a value could be examined to be at the raster's 25%, 50%, 75% percentile.
- ST_RastFromHexWKB - Return a raster value from a Hex representation of Well-Known Binary (WKB) raster.
- ST_RastFromWKB - Return a raster value from a Well-Known Binary (WKB) raster.
- ST_RasterToWorldCoord - Returns the raster's upper left corner as geometric X and Y (longitude and latitude) given a column and row. Column and row starts at 1.
- ST_RasterToWorldCoordX - Returns the geometric X coordinate upper left of a raster, column and row. Numbering of columns and rows starts at 1.
- ST_RasterToWorldCoordY - Returns the geometric Y coordinate upper left corner of a raster, column and row. Numbering of columns and rows starts at 1.
- ST_Reclass - Creates a new raster composed of band types reclassified from original. The nband is the band to be changed. If nband is not specified assumed to be 1. All other bands are returned unchanged. Use case: convert a 16BUI band to a 8BUI and so forth for simpler rendering as viewable formats.
- ST_Resample - Resample a raster using a specified resampling algorithm, new dimensions, an arbitrary grid corner and a set of raster georeferencing attributes defined or borrowed from another raster.
- 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 - Resize a raster to a new width/height
- ST_Reskew - Resample a raster by adjusting only its skew (or rotation parameters). New pixel values are computed using the NearestNeighbor (english or american spelling), Bilinear, Cubic, CubicSpline or Lanczos resampling algorithm. Default is NearestNeighbor.
- ST_Rotation - Returns the rotation of the raster in radian.
- ST_Roughness - Returns a raster with the calculated "roughness" of a DEM.

- ST_SRID - Returns the spatial reference identifier of the raster as defined in spatial_ref_sys table.
- ST_SameAlignment - Returns true if rasters have same skew, scale, spatial ref, and offset (pixels can be put on same grid without cutting into pixels) and false if they don't with notice detailing issue.
- ST_ScaleX - Returns the X component of the pixel width in units of coordinate reference system.
- ST_ScaleY - Returns the Y component of the pixel height in units of coordinate reference system.
- ST_SetBandIndex - Update the external band number of an out-db band
- ST_SetBandIsNoData - Sets the isnodata flag of the band to TRUE.
- ST_SetBandNoDataValue - Sets the value for the given band that represents no data. Band 1 is assumed if no band is specified. To mark a band as having no nodata value, set the nodata value = NULL.
- ST_SetBandPath - Update the external path and band number of an out-db band
- ST_SetGeoReference - Set Georeference 6 georeference parameters in a single call. Numbers should be separated by white space. Accepts inputs in GDAL or ESRI format. Default is 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 - Set the rotation of the raster in radian.
- ST_SetSRID - Sets the SRID of a raster to a particular integer srid defined in the spatial_ref_sys table.
- ST_SetScale - Sets the X and Y size of pixels in units of coordinate reference system. Number units/pixel width/height.
- ST_SetSkew - Sets the georeference X and Y skew (or rotation parameter). If only one is passed in, sets X and Y to the same value.
- ST_SetUpperLeft - Sets the value of the upper left corner of the pixel of the raster to projected X and Y coordinates.
- ST_SetValue - Returns modified raster resulting from setting the value of a given band in a given columnx, rowy pixel or the pixels that intersect a particular geometry. Band numbers start at 1 and assumed to be 1 if not specified.
- ST_SetValues - Returns modified raster resulting from setting the values of a given band.
- 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 - Returns the georeference X skew (or rotation parameter).
- ST_SkewY - Returns the georeference Y skew (or rotation parameter).
- ST_Slope - Returns the slope (in degrees by default) of an elevation raster band. Useful for analyzing terrain.
- ST_SnapToGrid - Resample a raster by snapping it to a grid. New pixel values are computed using the NearestNeighbor (english or american spelling), Bilinear, Cubic, CubicSpline or Lanczos resampling algorithm. Default is NearestNeighbor.
- ST_Summary - Returns a text summary of the contents of the raster.
- ST_SummaryStats - Returns summarystats consisting of count, sum, mean, stddev, min, max for a given raster band of a raster or raster coverage. Band 1 is assumed is no band is specified.
- ST_SummaryStatsAgg - Aggregate. Returns summarystats consisting of count, sum, mean, stddev, min, max for a given raster band of a set of raster. Band 1 is assumed is no band is specified.
- ST_TPI - Returns a raster with the calculated Topographic Position Index.
- ST_TRI - Returns a raster with the calculated Terrain Ruggedness Index.
- ST_Tile - Returns a set of rasters resulting from the split of the input raster based upon the desired dimensions of the output rasters.

- ST_Touches - Return true if raster rastA and rastB have at least one point in common but their interiors do not intersect.
- ST_Transform - Reprojects a raster in a known spatial reference system to another known spatial reference system using specified resampling algorithm. Options are NearestNeighbor, Bilinear, Cubic, CubicSpline, Lanczos defaulting to NearestNeighbor.
- ST_Union - Returns the union of a set of raster tiles into a single raster composed of 1 or more bands.
- ST_UpperLeftX - Returns the upper left X coordinate of raster in projected spatial ref.
- ST_UpperLeftY - Returns the upper left Y coordinate of raster in projected spatial ref.
- ST_Value - Returns the value of a given band in a given columnx, rowy pixel or at a particular geometric point. Band numbers start at 1 and assumed to be 1 if not specified. If exclude_nodata_value is set to false, then all pixels include nodata pixels are considered to intersect and return value. If exclude_nodata_value is not passed in then reads it from metadata of raster.
- ST_ValueCount - Returns a set of records containing a pixel band value and count of the number of pixels in a given band of a raster (or a raster coverage) that have a given set of values. If no band is specified defaults to band 1. By default nodata value pixels are not counted. and all other values in the pixel are output and pixel band values are rounded to the nearest integer.
- ST_Width - Returns the width of the raster in pixels.
- ST_Within - Return true if no points of raster rastA lie in the exterior of raster rastB and at least one point of the interior of rastA lies in the interior of rastB.
- ST_WorldToRasterCoord - Returns the upper left corner as column and row given geometric X and Y (longitude and latitude) or a point geometry expressed in the spatial reference coordinate system of the raster.
- ST_WorldToRasterCoordX - Returns the column in the raster of the point geometry (pt) or a X and Y world coordinate (xw, yw) represented in world spatial reference system of raster.
- ST_WorldToRasterCoordY - Returns the row in the raster of the point geometry (pt) or a X and Y world coordinate (xw, yw) represented in world spatial reference system of raster.
- UpdateRasterSRID - Change the SRID of all rasters in the user-specified column and table.

13.6 PostGIS Geometry / Geography / Raster Dump Functions

The functions given below are PostGIS functions that take as input or return as output a set of or single geometry_dump or geomval data type object.

- ST_DumpAsPolygons - Returns a set of geomval (geom,val) rows, from a given raster band. If no band number is specified, band num defaults to 1.
- ST_Intersection - Returns a raster or a set of geometry-pixelvalue pairs representing the shared portion of two rasters or the geometrical intersection of a vectorization of the raster and a geometry.
- ST_Dump - Returns a set of geometry_dump rows for the components of a geometry.
- ST_DumpPoints - Returns a set of geometry_dump rows for the coordinates in a geometry.
- ST_DumpRings - Returns a set of geometry_dump rows for the exterior and interior rings of a Polygon.
- ST_DumpSegments - Returns a set of geometry_dump rows for the segments in a geometry.

13.7 PostGIS Box Functions

The functions given below are PostGIS functions that take as input or return as output the box* family of PostGIS spatial types. The box family of types consists of box2d, and box3d

- Box2D - Returns a BOX2D representing the 2D extent of a geometry.
- Box3D - Returns a BOX3D representing the 3D extent of a geometry.
- Box3D - Returns the box 3d representation of the enclosing box of the raster.
- 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 - Returns the geometry as TWKB, aka "Tiny Well-Known Binary"
- ST_Box2dFromGeoHash - Return a BOX2D from a GeoHash string.
- ST_ClipByBox2D - Computes the portion of a geometry falling within a rectangle.
- ST_EstimatedExtent - Returns the estimated extent of a spatial table.
- ST_Expand - Returns a bounding box expanded from another bounding box or a geometry.
- ST_Extent - Aggregate function that returns the bounding box of geometries.
- ST_MakeBox2D - Creates a BOX2D defined by two 2D point geometries.
- ST_RemoveIrrelevantPointsForView - Removes points that are irrelevant for rendering a specific rectangluar view of a geometry.
- ST_XMax - Returns the X maxima of a 2D or 3D bounding box or a geometry.
- ST_XMin - Returns the X minima of a 2D or 3D bounding box or a geometry.
- ST_YMax - Returns the Y maxima of a 2D or 3D bounding box or a geometry.
- ST_YMin - Returns the Y minima of a 2D or 3D bounding box or a geometry.
- ST_ZMax - Returns the Z maxima of a 2D or 3D bounding box or a geometry.
- ST_ZMin - Returns the Z minima of a 2D or 3D bounding box or a geometry.
- RemoveUnusedPrimitives - Removes topology primitives which not needed to define existing TopoGeometry objects.
- ValidateTopology - Returns a set of validatetopology_returntype objects detailing issues with topology.
- ~(box2df,box2df) - Returns TRUE if a 2D float precision bounding box (BOX2DF) contains another 2D float precision bounding box (BOX2DF).
- ~(box2df,geometry) - Returns TRUE if a 2D float precision bounding box (BOX2DF) contains a geometry's 2D bonding box.
- ~(geometry,box2df) - Returns TRUE if a geometry's 2D bonding box contains a 2D float precision bounding box (GIDX).
- @(box2df,box2df) - Returns TRUE if a 2D float precision bounding box (BOX2DF) is contained into another 2D float precision bounding box.
- @(box2df,geometry) - Returns TRUE if a 2D float precision bounding box (BOX2DF) is contained into a geometry's 2D bounding box.
- @(geometry,box2df) - Returns TRUE if a geometry's 2D bounding box is contained into a 2D float precision bounding box (BOX2DF).

- &&(box2df,box2df) - Returns TRUE if two 2D float precision bounding boxes (BOX2DF) intersect each other.
- &&(box2df,geometry) - Returns TRUE if a 2D float precision bounding box (BOX2DF) intersects a geometry's (cached) 2D bounding box.
- &&(geometry,box2df) - Returns TRUE if a geometry's (cached) 2D bounding box intersects a 2D float precision bounding box (BOX2DF).

13.8 PostGIS Functions that support 3D

The functions given below are PostGIS functions that do not throw away the Z-Index.

- AddGeometryColumn - Adds a geometry column to an existing table.
- Box3D - Returns a BOX3D representing the 3D extent of a geometry.
- CG_3DArea - Computes area of 3D surface geometries. Will return 0 for solids.
- CG_3DConvexHull - Computes the 3D convex hull of a geometry.
- CG_3DDifference - Perform 3D difference
- CG_3DIntersection - Perform 3D intersection
- CG_3DUnion - Perform 3D union using postgis_sfcgal.
- CG_ApproximateMedialAxis - Compute the approximate medial axis of an areal geometry.
- CG_ConstrainedDelaunayTriangles - Return a constrained Delaunay triangulation around the given input geometry.
- CG_Extrude - Extrude a surface to a related volume
- CG_ForceLHR - Force LHR orientation
- CG_IsPlanar - Check if a surface is or not planar
- CG_IsSolid - Test if the geometry is a solid. No validity check is performed.
- CG_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.
- CG_Orientation - Determine surface orientation
- CG_StraightSkeleton - Compute a straight skeleton from a geometry
- CG_Tesselate - Perform surface Tessellation of a polygon or polyhedralsurface and returns as a TIN or collection of TINS
- CG_Visibility - Compute a visibility polygon from a point or a segment in a polygon geometry
- CG_Volume - Computes the volume of a 3D solid. If applied to surface (even closed) geometries will return 0.
- DropGeometryColumn - Removes a geometry column from a spatial table.
- GeometryType - Returns the type of a geometry as text.
- ST_3DArea - Computes area of 3D surface geometries. Will return 0 for solids.
- ST_3DClosestPoint - Returns the 3D point on g1 that is closest to g2. This is the first point of the 3D shortest line.
- ST_3DConvexHull - Computes the 3D convex hull of a geometry.
- 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 - Returns the 3D cartesian minimum distance (based on spatial ref) between two geometries in projected units.
- 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 - Returns the 3D length of a linear geometry.
- ST_3DLineInterpolatePoint - Returns a point interpolated along a 3D line at a fractional location.
- ST_3DLongestLine - Returns the 3D longest line between two geometries
- ST_3DMaxDistance - Returns the 3D cartesian maximum distance (based on spatial ref) between two geometries in projected units.
- ST_3DPerimeter - Returns the 3D perimeter of a polygonal geometry.
- ST_3DShortestLine - Returns the 3D shortest line between two geometries
- ST_3DUnion - Perform 3D union.
- ST_AddMeasure - Interpolates measures along a linear geometry.
- ST_AddPoint - Add a point to a LineString.
- ST_Affine - Apply a 3D affine transformation to a geometry.
- ST_ApproximateMedialAxis - Compute the approximate medial axis of an areal 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 - Return the Well-Known Text (WKT) representation of the geometry with SRID meta data.
- ST_AsGML - Return the geometry as a GML version 2 or 3 element.
- ST_AsGeoJSON - Return a geometry or feature in GeoJSON format.
- ST_AsHEXEWKB - Returns a Geometry in HEXEWKB format (as text) using either little-endian (NDR) or big-endian (XDR) encoding.
- ST_AsKML - Return the geometry as a KML element.
- ST_AsX3D - Returns a Geometry in X3D xml node element format: ISO-IEC-19776-1.2-X3DEncodings-XML
- ST_Boundary - Returns the boundary of a geometry.
- ST_BoundingDiagonal - Returns the diagonal of a geometry's bounding box.
- ST_CPAWithin - Tests if the closest point of approach of two trajectoriesis within the specified distance.
- ST_ChaikinSmoothing - Returns a smoothed version of a geometry, using the Chaikin algorithm
- ST_ClosestPointOfApproach - Returns a measure at the closest point of approach of two trajectories.
- ST_Collect - Creates a GeometryCollection or Multi* geometry from a set of geometries.
- ST_ConstrainedDelaunayTriangles - Return a constrained Delaunay triangulation around the given input geometry.
- ST_ConvexHull - Computes the convex hull of a geometry.

- ST_CoordDim - Return the coordinate dimension of a geometry.
- ST_CurveN - Returns the Nth component curve geometry of a CompoundCurve.
- ST_CurveToLine - Converts a geometry containing curves to a linear geometry.
- ST_DelaunayTriangles - Returns the Delaunay triangulation of the vertices of a geometry.
- ST_Difference - Computes a geometry representing the part of geometry A that does not intersect geometry B.
- ST_DistanceCPA - Returns the distance between the closest point of approach of two trajectories.
- ST_Dump - Returns a set of geometry_dump rows for the components of a geometry.
- ST_DumpPoints - Returns a set of geometry_dump rows for the coordinates in a geometry.
- ST_DumpRings - Returns a set of geometry_dump rows for the exterior and interior rings of a Polygon.
- ST_DumpSegments - Returns a set of geometry_dump rows for the segments in a geometry.
- ST_EndPoint - Returns the last point of a LineString or CircularLineString.
- ST_ExteriorRing - Returns a LineString representing the exterior ring of a Polygon.
- 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 - Force the geometries into a "2-dimensional mode".
- ST_ForceCurve - Upcast a geometry into its curved type, if applicable.
- ST_ForceLHR - Force LHR orientation
- ST_ForcePolygonCCW - Orients all exterior rings counter-clockwise and all interior rings clockwise.
- ST_ForcePolygonCW - Orients all exterior rings clockwise and all interior rings counter-clockwise.
- ST_ForceRHR - Force the orientation of the vertices in a polygon to follow the Right-Hand-Rule.
- ST_ForceSFS - Force the geometries to use SFS 1.1 geometry types only.
- ST_Force3D - Force the geometries into XYZ mode. This is an alias for ST_Force3DZ.
- ST_Force3DZ - Force the geometries into XYZ mode.
- ST_Force4D - Force the geometries into XYZM mode.
- ST_ForceCollection - Convert the geometry into a GEOMETRYCOLLECTION.
- ST_GeomFromEWKB - Return a specified ST_Geometry value from Extended Well-Known Binary representation (EWKB).
- ST_GeomFromEWKT - Return a specified ST_Geometry value from Extended Well-Known Text representation (EWKT).
- ST_GeomFromGML - Takes as input GML representation of geometry and outputs a PostGIS geometry object
- ST_GeomFromGeoJSON - Takes as input a geojson representation of a geometry and outputs a PostGIS geometry object
- ST_GeomFromKML - Takes as input KML representation of geometry and outputs a PostGIS geometry object
- ST_GeometricMedian - Returns the geometric median of a MultiPoint.
- ST_GeometryN - Return an element of a geometry collection.
- ST_GeometryType - Returns the SQL-MM type of a geometry as text.
- ST_HasArc - Tests if a geometry contains a circular arc
- ST_HasM - Checks if a geometry has an M (measure) dimension.

- ST_HasZ - Checks if a geometry has a Z dimension.
- ST_InteriorRingN - Returns the Nth interior ring (hole) of a Polygon.
- ST_InterpolatePoint - Returns the interpolated measure of a geometry closest to a point.
- ST_Intersection - Computes a geometry representing the shared portion of geometries A and B.
- ST_IsClosed - Tests if a LineStrings's start and end points are coincident. For a PolyhedralSurface tests if it is closed (volumetric).
- ST_IsCollection - Tests if a geometry is a geometry collection type.
- ST_IsPlanar - Check if a surface is or not planar
- ST_IsPolygonCCW - Tests if Polygons have exterior rings oriented counter-clockwise and interior rings oriented clockwise.
- ST_IsPolygonCW - Tests if Polygons have exterior rings oriented clockwise and interior rings oriented counter-clockwise.
- ST_IsSimple - Tests if a geometry has no points of self-intersection or self-tangency.
- 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_LengthSpheroid - Returns the 2D or 3D length/perimeter of a lon/lat geometry on a spheroid.
- ST_LineFromMultiPoint - Creates a LineString from a MultiPoint geometry.
- ST_LineInterpolatePoint - Returns a point interpolated along a line at a fractional location.
- ST_LineInterpolatePoints - Returns points interpolated along a line at a fractional interval.
- ST_LineSubstring - Returns the part of a line between two fractional locations.
- ST_LineToCurve - Converts a linear geometry to a curved geometry.
- ST_LocateBetweenElevations - Returns the portions of a geometry that lie in an elevation (Z) range.
- ST_M - Returns the M coordinate of a Point.
- ST_MakeLine - Creates a LineString from Point, MultiPoint, or LineString geometries.
- ST_MakePoint - Creates a 2D, 3DZ or 4D Point.
- 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 - Returns the amount of memory space a geometry takes.
- ST_MemUnion - Aggregate function which unions geometries in a memory-efficent but slower way
- ST_NDims - Returns the coordinate dimension of a geometry.
- ST_NPoints - Returns the number of points (vertices) in a geometry.
- ST_NRings - Returns the number of rings in a polygonal geometry.
- ST_Node - Nodes a collection of lines.
- ST_NumCurves - Return the number of component curves in a CompoundCurve.
- ST_NumGeometries - Returns the number of elements in a geometry collection.
- ST_NumPatches - Return the number of faces on a Polyhedral Surface. Will return null for non-polyhedral geometries.

- ST_Orientation - Determine surface orientation
- ST_PatchN - Returns the Nth geometry (face) of a PolyhedralSurface.
- ST_PointFromWKB - Makes a geometry from WKB with the given SRID
- ST_PointN - Returns the Nth point in the first LineString or circular LineString in a geometry.
- ST_PointOnSurface - Computes a point guaranteed to lie in a polygon, or on a geometry.
- ST_Points - Returns a MultiPoint containing the coordinates of a geometry.
- 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 - Return the geometry with vertex order reversed.
- 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 - Replace point of a linestring with a given point.
- ST_ShiftLongitude - Shifts the longitude coordinates of a geometry between -180..180 and 0..360.
- ST_SnapToGrid - Snap all points of the input geometry to a regular grid.
- ST_StartPoint - Returns the first point of a LineString.
- ST_StraightSkeleton - Compute a straight skeleton from a geometry
- ST_SwapOrdinates - Returns a version of the given geometry with given ordinate values swapped.
- ST_SymDifference - Computes a geometry representing the portions of geometries A and B that do not intersect.
- ST_Tesselate - Perform surface Tessellation 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 - Wrap a geometry around an X value.
- 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 - Returns a code indicating the ZM coordinate dimension of a geometry.
- Equals - Returns true if two topogeometries are composed of the same topology primitives.
- Intersects - Returns true if any pair of primitives from the two topogeometries intersect.
- UpdateGeometrySRID - Updates the SRID of all features in a geometry column, and the table metadata.
- &&& - Returns TRUE if A's n-D bounding box intersects B's n-D bounding box.
- &&&(geometry,gidx) - Returns TRUE if a geometry's (cached) n-D bounding box intersects a n-D float precision bounding box (GIDX).
- &&&(gidx,geometry) - Returns TRUE if a n-D float precision bounding box (GIDX) intersects a geometry's (cached) n-D bounding box.
- &&&(gidx,gidx) - Returns TRUE if two n-D float precision bounding boxes (GIDX) intersect each other.

13.9 PostGIS Curved Geometry Support Functions

The functions given below are PostGIS functions that can use CIRCULARSTRING, CURVEPOLYGON, and other curved geometry types

- AddGeometryColumn - Adds a geometry column to an existing table.
- Box2D - Returns a BOX2D representing the 2D extent of a geometry.
- Box3D - Returns a BOX3D representing the 3D extent of a geometry.
- DropGeometryColumn - Removes a geometry column from a spatial table.
- GeometryType - Returns the type of a geometry as text.
- 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 - Return the Well-Known Text (WKT) representation of the geometry with SRID meta data.
- ST_AsHEXEWKB - Returns a Geometry in HEXEWKB format (as text) using either little-endian (NDR) or big-endian (XDR) encoding.
- ST_AsSVG - Returns SVG path data for a geometry.
- ST_AsText - Return the Well-Known Text (WKT) representation of the geometry/geography without SRID metadata.

- 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 - Return the coordinate dimension of a geometry.
- ST_CurveToLine - Converts a geometry containing curves to a linear geometry.
- ST_Distance - Returns the distance between two geometry or geography values.
- ST_Dump - Returns a set of geometry_dump rows for the components of a geometry.
- ST_DumpPoints - Returns a set of geometry_dump rows for the coordinates in a geometry.
- ST_EndPoint - Returns the last point of a LineString or CircularLineString.
- 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 - Force the geometries into a "2-dimensional mode".
- ST_ForceCurve - Upcast a geometry into its curved type, if applicable.
- ST_ForceSFS - Force the geometries to use SFS 1.1 geometry types only.
- ST_Force3D - Force the geometries into XYZ mode. This is an alias for ST_Force3DZ.
- ST_Force3DM - Force the geometries into XYM mode.
- ST_Force3DZ - Force the geometries into XYZ mode.
- ST_Force4D - Force the geometries into XYZM mode.
- ST_ForceCollection - Convert the geometry into a GEOMETRYCOLLECTION.
- ST_GeoHash - Return a GeoHash representation of the geometry.
- ST_GeogFromWKB - Creates a geography instance from a Well-Known Binary geometry representation (WKB) or extended Well Known Binary (EWKB).
- ST_GeomFromEWKB - Return a specified ST_Geometry value from Extended Well-Known Binary representation (EWKB).
- ST_GeomFromEWKT - Return a specified ST_Geometry value from Extended Well-Known Text representation (EWKT).
- ST_GeomFromText - Return a specified ST_Geometry value from Well-Known Text representation (WKT).
- ST_GeomFromWKB - Creates a geometry instance from a Well-Known Binary geometry representation (WKB) and optional SRID.
- ST_GeometryN - Return an element of a geometry collection.
- = - Returns TRUE if the coordinates and coordinate order geometry/geography A are the same as the coordinates and coordinate order of geometry/geography B.
- &<| - Returns TRUE if A's bounding box overlaps or is below B's.
- 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 - Tests if a LineStrings's start and end points are coincident. For a PolyhedralSurface tests if it is closed (volumetric).

- ST_IsCollection - Tests if a geometry is a geometry collection type.
- ST_IsEmpty - Tests if a geometry is empty.
- ST_LineToCurve - Converts a linear geometry to a curved geometry.
- ST_MemSize - Returns the amount of memory space a geometry takes.
- ST_NPoints - Returns the number of points (vertices) in a geometry.
- ST_NRings - Returns the number of rings in a polygonal geometry.
- ST_PointFromWKB - Makes a geometry from WKB with the given SRID
- ST_PointN - Returns the Nth point in the first LineString or circular LineString in a geometry.
- ST_Points - Returns a MultiPoint containing the coordinates of a geometry.
- 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 - Returns a text summary of the contents of a geometry.
- ST_SwapOrdinates - Returns a version of the given geometry with given ordinate values swapped.
- 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 - Returns a code indicating the ZM coordinate dimension of a geometry.
- UpdateGeometrySRID - Updates the SRID of all features in a geometry column, and the table metadata.
- ~(box2df,box2df) - Returns TRUE if a 2D float precision bounding box (BOX2DF) contains another 2D float precision bounding box (BOX2DF).
- ~(box2df,geometry) - Returns TRUE if a 2D float precision bounding box (BOX2DF) contains a geometry's 2D bonding box.
- ~(geometry,box2df) - Returns TRUE if a geometry's 2D bonding box contains a 2D float precision bounding box (GIDX).
- && - Returns TRUE if A's 2D bounding box intersects B's 2D bounding box.
- &&& - Returns TRUE if A's n-D bounding box intersects B's n-D bounding box.
- @(box2df,box2df) - Returns TRUE if a 2D float precision bounding box (BOX2DF) is contained into another 2D float precision bounding box.

- @(box2df,geometry) - Returns TRUE if a 2D float precision bounding box (BOX2DF) is contained into a geometry's 2D bounding box.
- @(geometry,box2df) - Returns TRUE if a geometry's 2D bounding box is contained into a 2D float precision bounding box (BOX2DF).
- &&(box2df,box2df) - Returns TRUE if two 2D float precision bounding boxes (BOX2DF) intersect each other.
- &&(box2df,geometry) - Returns TRUE if a 2D float precision bounding box (BOX2DF) intersects a geometry's (cached) 2D bounding box.
- &&(geometry,box2df) - Returns TRUE if a geometry's (cached) 2D bounding box intersects a 2D float precision bounding box (BOX2DF).
- &&&(geometry,gidx) - Returns TRUE if a geometry's (cached) n-D bounding box intersects a n-D float precision bounding box (GIDX).
- &&&(gidx,geometry) - Returns TRUE if a n-D float precision bounding box (GIDX) intersects a geometry's (cached) n-D bounding box.
- &&&(gidx,gidx) - Returns TRUE if two n-D float precision bounding boxes (GIDX) intersect each other.

13.10 PostGIS Polyhedral Surface Support Functions

The functions given below are PostGIS functions that can use POLYHEDRALSURFACE, POLYHEDRALSURFACEM geometries

- Box2D - Returns a BOX2D representing the 2D extent of a geometry.
- Box3D - Returns a BOX3D representing the 3D extent of a geometry.
- CG_3DArea - Computes area of 3D surface geometries. Will return 0 for solids.
- CG_3DConvexHull - Computes the 3D convex hull of a geometry.
- CG_3DDifference - Perform 3D difference
- CG_3DIntersection - Perform 3D intersection
- CG_3DUnion - Perform 3D union using postgis_sfcgal.
- CG_ApproximateMedialAxis - Compute the approximate medial axis of an areal geometry.
- CG_Extrude - Extrude a surface to a related volume
- CG_ForceLHR - Force LHR orientation
- CG_IsPlanar - Check if a surface is or not planar
- CG_IsSolid - Test if the geometry is a solid. No validity check is performed.
- CG_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.
- CG_StraightSkeleton - Compute a straight skeleton from a geometry
- CG_Tesselate - Perform surface Tessellation of a polygon or polyhedralsurface and returns as a TIN or collection of TINS
- CG_Visibility - Compute a visibility polygon from a point or a segment in a polygon geometry
- CG_Volume - Computes the volume of a 3D solid. If applied to surface (even closed) geometries will return 0.
- GeometryType - Returns the type of a geometry as text.

- ST_3DArea - Computes area of 3D surface geometries. Will return 0 for solids.
- ST_3DClosestPoint - Returns the 3D point on g1 that is closest to g2. This is the first point of the 3D shortest line.
- ST_3DConvexHull - Computes the 3D convex hull of a geometry.
- 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 - Returns the 3D cartesian minimum distance (based on spatial ref) between two geometries in projected units.
- 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_3DLongestLine - Returns the 3D longest line between two geometries
- ST_3DMaxDistance - Returns the 3D cartesian maximum distance (based on spatial ref) between two geometries in projected units.
- ST_3DShortestLine - Returns the 3D shortest line between two geometries
- ST_3DUnion - Perform 3D union.
- ST_Affine - Apply a 3D affine transformation to a geometry.
- ST_ApproximateMedialAxis - Compute the approximate medial axis of an areal geometry.
- ST_Area - Returns the area of a polygonal 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 - Return the Well-Known Text (WKT) representation of the geometry with SRID meta data.
- ST_AsGML - Return the geometry as a GML version 2 or 3 element.
- ST_AsX3D - Returns a Geometry in X3D xml node element format: ISO-IEC-19776-1.2-X3DEncodings-XML
- ST_CoordDim - Return the coordinate dimension of a geometry.
- ST_Dimension - Returns the topological dimension of a geometry.
- ST_Dump - Returns a set of geometry_dump rows for the components of a geometry.
- ST_DumpPoints - Returns a set of geometry_dump rows for the coordinates in a geometry.
- ST_Expand - Returns a bounding box expanded from another bounding box or a geometry.
- ST_Extent - Aggregate function that returns the bounding box of geometries.
- ST_Extrude - Extrude a surface to a related volume
- ST_FlipCoordinates - Returns a version of a geometry with X and Y axis flipped.
- ST_Force2D - Force the geometries into a "2-dimensional mode".
- ST_ForceLHR - Force LHR orientation
- ST_ForceRHR - Force the orientation of the vertices in a polygon to follow the Right-Hand-Rule.

- ST_ForceSFS - Force the geometries to use SFS 1.1 geometry types only.
- ST_Force3D - Force the geometries into XYZ mode. This is an alias for ST_Force3DZ.
- ST_Force3DZ - Force the geometries into XYZ mode.
- ST_ForceCollection - Convert the geometry into a GEOMETRYCOLLECTION.
- ST_GeomFromEWKB - Return a specified ST_Geometry value from Extended Well-Known Binary representation (EWKB).
- ST_GeomFromEWKT - Return a specified ST_Geometry value from Extended Well-Known Text representation (EWKT).
- ST_GeomFromGML - Takes as input GML representation of geometry and outputs a PostGIS geometry object
- ST_GeometryN - Return an element of a geometry collection.
- ST_GeometryType - Returns the SQL-MM type of a geometry as text.
- = - Returns TRUE if the coordinates and coordinate order geometry/geography A are the same as the coordinates and coordinate order of geometry/geography B.
- &<| - Returns TRUE if A's bounding box overlaps or is below B's.
- ~= - Returns TRUE if A's bounding box is the same as B's.
- ST_IsClosed - Tests if a LineStrings's start and end points are coincident. For a PolyhedralSurface tests if it is closed (volumetric).
- ST_IsPlanar - Check if a surface is or not planar
- ST_IsSolid - Test if the geometry is a solid. No validity check is performed.
- 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_MemSize - Returns the amount of memory space a geometry takes.
- ST_NPoints - Returns the number of points (vertices) in a geometry.
- ST_NumGeometries - Returns the number of elements in a geometry collection.
- ST_NumPatches - Return the number of faces on a Polyhedral Surface. Will return null for non-polyhedral geometries.
- ST_PatchN - Returns the Nth geometry (face) of a PolyhedralSurface.
- ST_RemoveRepeatedPoints - Returns a version of a geometry with duplicate points removed.
- ST_Reverse - Return the geometry with vertex order reversed.
- ST_Rotate - Rotates a geometry about an origin point.
- ST_RotateX - Rotates a geometry about the X axis.
- ST_RotateY - Rotates a geometry about the Y axis.
- ST_RotateZ - Rotates a geometry about the Z axis.
- ST_Scale - Scales a geometry by given factors.
- ST_ShiftLongitude - Shifts the longitude coordinates of a geometry between -180..180 and 0..360.
- ST_StraightSkeleton - Compute a straight skeleton from a geometry
- ST_Summary - Returns a text summary of the contents of a geometry.
- ST_SwapOrdinates - Returns a version of the given geometry with given ordinate values swapped.
- ST_Tesselate - Perform surface Tessellation of a polygon or polyhedralsurface and returns as a TIN or collection of TINS

- ST_Transform - Return a new geometry with coordinates transformed to a different spatial reference system.
- ST_Volume - Computes the volume of a 3D solid. If applied to surface (even closed) geometries will return 0.
- ~(box2df,box2df) - Returns TRUE if a 2D float precision bounding box (BOX2DF) contains another 2D float precision bounding box (BOX2DF).
- ~(box2df,geometry) - Returns TRUE if a 2D float precision bounding box (BOX2DF) contains a geometry's 2D bonding box.
- ~(geometry,box2df) - Returns TRUE if a geometry's 2D bonding box contains a 2D float precision bounding box (GIDX).
- && - Returns TRUE if A's 2D bounding box intersects B's 2D bounding box.
- &&& - Returns TRUE if A's n-D bounding box intersects B's n-D bounding box.
- @(box2df,box2df) - Returns TRUE if a 2D float precision bounding box (BOX2DF) is contained into another 2D float precision bounding box.
- @(box2df,geometry) - Returns TRUE if a 2D float precision bounding box (BOX2DF) is contained into a geometry's 2D bounding box.
- @(geometry,box2df) - Returns TRUE if a geometry's 2D bounding box is contained into a 2D float precision bounding box (BOX2DF).
- &&(box2df,box2df) - Returns TRUE if two 2D float precision bounding boxes (BOX2DF) intersect each other.
- &&(box2df,geometry) - Returns TRUE if a 2D float precision bounding box (BOX2DF) intersects a geometry's (cached) 2D bounding box.
- &&(geometry,box2df) - Returns TRUE if a geometry's (cached) 2D bounding box intersects a 2D float precision bounding box (BOX2DF).
- &&&(geometry,gidx) - Returns TRUE if a geometry's (cached) n-D bounding box intersects a n-D float precision bounding box (GIDX).
- &&&(gidx,geometry) - Returns TRUE if a n-D float precision bounding box (GIDX) intersects a geometry's (cached) n-D bounding box.
- &&&(gidx,gidx) - Returns TRUE if two n-D float precision bounding boxes (GIDX) intersect each other.

13.11 PostGIS Function Support Matrix

Below is an alphabetical listing of spatial specific functions in PostGIS and the kinds of spatial types they work with or OGC/SQL compliance they try to conform to.

- A means the function works with the type or subtype natively.
- A means it works but with a transform cast built-in using cast to geometry, transform to a "best srid" spatial ref and then cast back. Results may not be as expected for large areas or areas at poles and may accumulate floating point junk.
- A means the function works with the type because of a auto-cast to another such as to box3d rather than direct type support.
- A means the function only available if PostGIS compiled with SFCGAL support.
- geom - Basic 2D geometry support (x,y).
- geog - Basic 2D geography support (x,y).

- 2.5D - basic 2D geometries in 3 D/4D space (has Z or M coord).
- PS - Polyhedral surfaces
- T - Triangles and Triangulated Irregular Network surfaces (TIN)

Function	geom	geog	2.5D	Curves	SQL MM	PS	T
ST_Collect	✓		✓	✓			
ST_LineFromMulti	✓ t		✓				
ST_MakeEnvelope	✓						
ST_MakeLine	✓		✓				
ST_MakePoint	✓		✓				
ST_MakePointM	✓						
ST_MakePolygon	✓		✓				
ST_Point	✓				✓		
ST_PointZ	✓						
ST_PointM	✓						
ST_PointZM	✓						
ST_Polygon	✓		✓		✓		
ST_TileEnvelope	✓						
ST_HexagonGrid	✓						
ST_Hexagon	✓						
ST_SquareGrid	✓						
ST_Square	✓						
ST_Letters	✓						
GeometryType	✓		✓	✓		✓	✓
ST_Boundary	✓		✓		✓		
ST_BoundingDiago	✓		✓				
ST_CoordDim	✓		✓	✓	✓	✓	✓
ST_Dimension	✓				✓	✓	✓
ST_Dump	✓		✓	✓		✓	✓
ST_DumpPoints	✓		✓	✓		✓	✓
ST_DumpSegments	✓		✓				✓
ST_DumpRings	✓		✓				
ST_EndPoint	✓		✓	✓	✓		
ST_Envelope	✓				✓		
ST_ExteriorRing	✓		✓		✓		

Function	geom	geog	2.5D	Curves	SQL MM	PS	T
ST_GeometryN	✓		✓	✓	✓	✓	✓
ST_GeometryType	✓		✓		✓	✓	
ST_HasArc	✓		✓	✓			
ST_InteriorRingN	✓		✓		✓		
ST_NumCurves	✓		✓		✓		
ST_CurveN	✓		✓		✓		
ST_IsClosed	✓		✓	✓	✓	✓	
ST_IsCollection	✓		✓	✓			
ST_IsEmpty	✓			✓	✓		
ST_IsPolygonCCW	✓		✓				
ST_IsPolygonCW	✓		✓				
ST_IsRing	✓				✓		
ST_IsSimple	✓		✓		✓		
ST_M	✓		✓		✓		
ST_MemSize	✓		✓	✓		✓	✓
ST_NDims	✓		✓				
ST_NPoints	✓		✓	✓		✓	
ST_NRings	✓		✓	✓			
ST_NumGeometries	✓		✓		✓	✓	✓
ST_NumInteriorRing	✓				✓		
ST_NumInteriorRings	✓						
ST_NumPatches	✓		✓		✓	✓	
ST_NumPoints	✓				✓		
ST_PatchN	✓		✓		✓	✓	
ST_PointN	✓		✓	✓	✓		
ST_Points	✓		✓	✓			
ST_StartPoint	✓		✓	✓	✓		
ST_Summary	✓	✓		✓		✓	✓
ST_X	✓		✓		✓		
ST_Y	✓		✓		✓		
ST_Z	✓		✓		✓		
ST_Zmflag	✓		✓	✓			
ST_HasZ	✓		✓				
ST_HasM	✓		✓				

Function	geom	geog	2.5D	Curves	SQL MM	PS	T
ST_AddPoint	✓		✓				
ST_CollectionExtra	✓						
ST_CollectionHom	✓	ize					
ST_CurveToLine	✓		✓	✓	✓		
ST_Scroll	✓		✓				
ST_FlipCoordinates	✓		✓	✓		✓	✓
ST_Force2D	✓		✓	✓		✓	
ST_Force3D	✓		✓	✓		✓	
ST_Force3DZ	✓		✓	✓		✓	
ST_Force3DM	✓			✓			
ST_Force4D	✓		✓	✓			
ST_ForceCollection	✓		✓	✓		✓	
ST_ForceCurve	✓		✓	✓			
ST_ForcePolygonC	✓		✓				
ST_ForcePolygonC	✓		✓				
ST_ForceSFS	✓		✓	✓		✓	✓
ST_ForceRHR	✓		✓			✓	
ST_LineExtend	✓						
ST_LineToCurve	✓		✓	✓			
ST_Multi	✓						
ST_Normalize	✓						
ST_Project	✓	✓					
ST_QuantizeCoordi	✓	s					
ST_RemovePoint	✓		✓				
ST_RemoveRepeate	✓	ints	✓			✓	
ST_RemoveIrreleva	✓	intsForView					
ST_RemoveSmallP	✓						
ST_Reverse	✓		✓			✓	
ST_Segmentize	✓	✓					
ST_SetPoint	✓		✓				
ST_ShiftLongitude	✓		✓			✓	✓
ST_WrapX	✓		✓				
ST_SnapToGrid	✓		✓				
ST_Snap	✓						

Function	geom	geog	2.5D	Curves	SQL MM	PS	T
ST_SwapOrdinates	✓		✓	✓		✓	✓
ST_IsValid	✓				✓		
ST_IsValidDetail	✓						
ST_IsValidReason	✓						
ST_MakeValid	✓		✓				
ST_InverseTransfor	✓ peline						
ST_SetSRID	✓			✓			
ST_SRID	✓			✓	✓		
ST_Transform	✓			✓	✓	✓	
ST_TransformPipel	✓						
postgis_srs_codes							
postgis_srs							
postgis_srs_all							
postgis_srs_search	✓						
ST_BdPolyFromTe	✓						
ST_BdMPolyFromT	✓						
ST_GeogFromText		✓					
ST_GeographyFromText		✓					
ST_GeomCollFrom	✓				✓		
ST_GeomFromEWI	✓		✓	✓		✓	✓
ST_GeomFromMA	✓ 1						
ST_GeometryFrom	✓				✓		
ST_GeomFromText	✓			✓	✓		
ST_LineFromText	✓				✓		
ST_MLineFromTex	✓				✓		
ST_MPointFromTe	✓				✓		
ST_MPolyFromTex	✓				✓		
ST_PointFromText	✓				✓		
ST_PolygonFromTe	✓				✓		
ST_WKTToSQL	✓				✓		
ST_GeogFromWKB		✓		✓			
ST_GeomFromEWI	✓		✓	✓		✓	✓
ST_GeomFromWK	✓			✓	✓		
ST_LineFromWKB	✓				✓		
ST_LinestringFrom	✓ B				✓		

Function	geom	geog	2.5D	Curves	SQL MM	PS	T
ST_PointFromWKB	✓		✓	✓	✓		
ST_WKBToSQL	✓				✓		
ST_Box2dFromGeoHash	☑						
ST_GeomFromGeoHash	✓						
ST_GeomFromGML	✓		✓			✓	✓
ST_GeomFromGeoJSON	✓		✓				
ST_GeomFromKML	✓		✓				
ST_GeomFromTWKB	✓						
ST_GMLToSQL	✓				✓		
ST_LineFromEncodedPolyline	✓						
ST_PointFromGeoHash							
ST_FromFlatGeobufToTable							
ST_FromFlatGeobuf							
ST_AsEWKT	✓	✓	✓	✓		✓	✓
ST_AsText	✓	✓		✓	✓		
ST_AsBinary	✓	✓	✓	✓	✓	✓	✓
ST_AsEWKB	✓		✓	✓		✓	✓
ST_AsHEXEWKB	✓		✓	✓			
ST_AsEncodedPolyline	✓						
ST_AsFlatGeobuf	☑						
ST_AsGeobuf	☑						
ST_AsGeoJSON	✓	✓	✓				
ST_AsGML	✓	✓	✓		✓	✓	✓
ST_AsKML	✓	✓	✓				
ST_AsLatLonText	✓						
ST_AsMARC21	✓						
ST_AsMVTGeom	✓						
ST_AsMVT	☑						
ST_AsSVG	✓	✓		✓			
ST_AsTWKB	✓						
ST_AsX3D	✓		✓			✓	✓
ST_GeoHash	✓			✓			
&&	✓	✓		✓		✓	
&&(geometry,box2df)	✓			✓		✓	
&&(box2df,geometry)	✓			✓		✓	

Function	geom	geog	2.5D	Curves	SQL MM	PS	T
&&(box2df,box2df)	✓			✓		✓	
&&&	✓		✓	✓		✓	✓
&&&(geometry,gidx)	✓		✓	✓		✓	✓
&&&(gidx,geometry)	✓		✓	✓		✓	✓
&&&(gidx,gidx)			✓	✓		✓	✓
&<	✓						
&<|	✓			✓		✓	
&>	✓						
<<	✓						
<<|	✓						
=	✓	✓		✓		✓	
>>	✓						
@	✓						
@(geometry,box2df)	✓			✓		✓	
@(box2df,geometry)	✓			✓		✓	
@(box2df,box2df)	✓			✓		✓	
|&>	✓						
|>>	✓						
~	✓						
~(geometry,box2df)	✓			✓		✓	
~(box2df,geometry)	✓			✓		✓	
~(box2df,box2df)	✓			✓		✓	
~=	✓					✓	
<->	✓	✓					
|=|	✓						
<#>	✓						
<<->>	✓						
ST_3DIntersects	✓		✓		✓	✓	✓
ST_Contains	✓				✓		
ST_ContainsProperly	✓						
ST_CoveredBy	✓	✓					
ST_Covers	✓	✓					
ST_Crosses	✓				✓		
ST_Disjoint	✓				✓		

Function	geom	geog	2.5D	Curves	SQL MM	PS	T
ST_Equals	✓				✓		
ST_Intersects	✓	✓		✓	✓		✓
ST_LineCrossingDi ion	✓						
ST_OrderingEquals	✓				✓		
ST_Overlaps	✓				✓		
ST_Relate	✓				✓		
ST_RelateMatch							
ST_Touches	✓				✓		
ST_Within	✓				✓		
ST_3DDWithin	✓		✓		✓	✓	
ST_3DDFullyWithi	✓		✓			✓	
ST_DFullyWithin	✓						
ST_DWithin	✓	✓					
ST_PointInsideCirc	✓						
ST_Area	✓	✓			✓	✓	
ST_Azimuth	✓	✓					
ST_Angle	✓						
ST_ClosestPoint	✓	✓					
ST_3DClosestPoint	✓		✓			✓	
ST_Distance	✓	✓		✓	✓		
ST_3DDistance	✓		✓		✓	✓	
ST_DistanceSphere	✓						
ST_DistanceSphero	✓						
ST_FrechetDistance	✓						
ST_HausdorffDistar	✓						
ST_Length	✓	✓			✓		
ST_Length2D	✓						
ST_3DLength	✓		✓		✓		
ST_LengthSpheroid	✓		✓				
ST_LongestLine	✓						
ST_3DLongestLine	✓		✓			✓	
ST_MaxDistance	✓						
ST_3DMaxDistance	✓		✓			✓	
ST_MinimumClear	✓						

Function	geom	geog	2.5D	Curves	SQL MM	PS	T
ST_MinimumClear	✓ Line						
ST_Perimeter	✓	✓			✓		
ST_Perimeter2D	✓						
ST_3DPerimeter	✓		✓		✓		
ST_ShortestLine	✓	✓					
ST_3DShortestLine	✓		✓			✓	
ST_ClipByBox2D	✓						
ST_Difference	✓		✓		✓		
ST_Intersection	✓		✓		✓		
ST_MemUnion	✓		✓				
ST_Node	✓		✓				
ST_Split	✓						
ST_Subdivide	✓						
ST_SymDifference	✓		✓		✓		
ST_UnaryUnion	✓		✓				
ST_Union	✓		✓		✓		
ST_Buffer	✓				✓		
ST_BuildArea	✓						
ST_Centroid	✓	✓			✓		
ST_ChaikinSmooth	✓		✓				
ST_ConcaveHull	✓						
ST_ConvexHull	✓		✓		✓		
ST_DelaunayTriang	✓		✓				✓
ST_FilterByM	✓						
ST_GeneratePoints	✓						
ST_GeometricMedi	✓		✓				
ST_LineMerge	✓						
ST_MaximumInscri	✓ Circle						
ST_LargestEmptyC	✓						
ST_MinimumBoun	✓ Circle						
ST_MinimumBoun	✓ Radius						
ST_OrientedEnvelo	✓						
ST_OffsetCurve	✓						
ST_PointOnSurface	✓		✓		✓		

Function	geom	geog	2.5D	Curves	SQL MM	PS	T
ST_Polygonize	✓						
ST_ReducePrecisio	✓						
ST_SharedPaths	✓						
ST_Simplify	✓						
ST_SimplifyPreserv	✓ pology						
ST_SimplifyPolygo	✓ ll						
ST_SimplifyVW	✓						
ST_SetEffectiveAre	✓						
ST_TriangulatePoly	✓						
ST_VoronoiLines	✓						
ST_VoronoiPolygor	✓						
ST_CoverageInvalic	✓ es						
ST_CoverageSimpli	✓						
ST_CoverageUnion	✓						
ST_Affine	✓		✓	✓		✓	✓
ST_Rotate	✓		✓	✓		✓	✓
ST_RotateX	✓		✓			✓	✓
ST_RotateY	✓		✓			✓	✓
ST_RotateZ	✓		✓	✓		✓	✓
ST_Scale	✓		✓	✓		✓	✓
ST_Translate	✓		✓	✓			
ST_TransScale	✓		✓	✓			
ST_ClusterDBSCA	✓			✓			
ST_ClusterIntersect	✓						
ST_ClusterIntersect	✓ Vin						
ST_ClusterKMeans	✓						
ST_ClusterWithin	✓			✓			
ST_ClusterWithinW	✓			✓			
Box2D	✓			✓		✓	✓
Box3D	✓		✓	✓		✓	✓
ST_EstimatedExten	☑			✓			
ST_Expand	✓					✓	✓
ST_Extent	✓					✓	✓
ST_3DExtent	✓		✓	✓		✓	✓

Function	geom	geog	2.5D	Curves	SQL MM	PS	T
ST_MakeBox2D	✓						
ST_3DMakeBox	✓						
ST_XMax	☑		✓	✓			
ST_XMin	☑		✓	✓			
ST_YMax	☑		✓	✓			
ST_YMin	☑		✓	✓			
ST_ZMax	☑		✓	✓			
ST_ZMin	☑		✓	✓			
ST_LineInterpolatel t	✓	✓	✓				
ST_3DLineInterpol oint	✓		✓				
ST_LineInterpolatel ts	✓	✓	✓				
ST_LineLocatePoin	✓	✓					
ST_LineSubstring	✓	✓	✓				
ST_LocateAlong	✓				✓		
ST_LocateBetween	✓				✓		
ST_LocateBetween ations	✓		✓				
ST_InterpolatePoint	✓		✓				
ST_AddMeasure	✓		✓				
ST_IsValidTrajecto	✓		✓				
ST_ClosestPointOf oach	✓		✓				
ST_DistanceCPA	✓		✓				
ST_CPAWithin	✓		✓				
postgis.gdal_datapath							
postgis.gdal_enabled_drivers							
postgis.enable_outdb_rasters							
postgis.gdal_vsi_options							
PostGIS_AddBBox	✓			✓			
PostGIS_DropBBox	✓			✓			
PostGIS_HasBBox	✓			✓			
postgis_sfcgal_version							
postgis_sfcgal_full_version							
CG_ForceLHR	🧊		🧊			🧊	🧊
CG_IsPlanar	🧊		🧊			🧊	🧊
CG_IsSolid	🧊		🧊			🧊	🧊
CG_MakeSolid	🧊		🧊			🧊	🧊

Function	geom	geog	2.5D	Curves	SQL MM	PS	T
CG_Orientation							
CG_Area							
CG_3DArea							
CG_Volume							
ST_ForceLHR							
ST_IsPlanar							
ST_IsSolid							
ST_MakeSolid							
ST_Orientation							
ST_3DArea							
ST_Volume							
CG_Intersection							
CG_Intersects							
CG_3DIntersects							
CG_Difference							
ST_3DDifference							
CG_3DDifference							
CG_Distance							
CG_3DDistance							
ST_3DConvexHull							
CG_3DConvexHull							
ST_3DIntersection							
CG_3DIntersection							
CG_Union							

Function	geom	geog	2.5D	Curves	SQL MM	PS	T
ST_3DUnion							
CG_3DUnion							
ST_AlphaShape	✓						
CG_AlphaShape							
CG_ApproxConvex tion							
ST_ApproximateM Axis							
CG_ApproximateM lAxis							
ST_ConstrainedDe yTriangles							
CG_ConstrainedDe ayTriangles							
ST_Extrude							
CG_Extrude							
CG_ExtrudeStraigh leton							
CG_GreeneApprox exPartition							
ST_MinkowskiSun							
CG_MinkowskiSu							
ST_OptimalAlphaS							
CG_OptimalAlpha							
CG_OptimalConve ition							
CG_StraightSkelet							
ST_StraightSkeleto							
ST_Tesselate							
CG_Tesselate							
CG_Triangulate							
CG_Visibility							

Function	geom	geog	2.5D	Curves	SQL MM	PS	T
CG_YMonotonePa n							
getfaceedges_returntype							
TopoGeometry							
validatetopology_returntype							
TopoElement							
TopoElementArray							
AddTopoGeometryColumn							
RenameTopoGeometryColumn							
DropTopology							
RenameTopology							
DropTopoGeometryColumn							
Populate_Topology_Layer							
TopologySummary							
ValidateTopology ✓							
ValidateTopologyRelation							
FindTopology							
FindLayer							
CreateTopology							
CopyTopology							
ST_InitTopoGeo					✓		
ST_CreateTopoGeo ✓					✓		
TopoGeo_AddPoint ✓							
TopoGeo_AddLine ✓ g							
TopoGeo_AddPoly ✓							
TopoGeo_LoadGeo ✓ y							
ST_AddIsoNode ✓					✓		
ST_AddIsoEdge ✓					✓		
ST_AddEdgeNewF ✓					✓		
ST_AddEdgeModF ✓					✓		
ST_RemEdgeNewFace					✓		
ST_RemEdgeModFace					✓		
ST_ChangeEdgeGe ✓					✓		
ST_ModEdgeSplit ✓					✓		
ST_ModEdgeHeal					✓		
ST_NewEdgeHeal					✓		
ST_MoveIsoNode ✓					✓		
ST_NewEdgesSplit ✓					✓		
ST_RemoveIsoNode					✓		
ST_RemoveIsoEdge					✓		
GetEdgeByPoint ✓							

Function	geom	geog	2.5D	Curves	SQL MM	PS	T
GetFaceByPoint	✓						
GetFaceContainingPoint	✓						
GetNodeByPoint	✓						
GetTopologyID							
GetTopologySRID							
GetTopologyName							
ST_GetFaceEdges					✓		
ST_GetFaceGeometry	✓				✓		
GetRingEdges							
GetNodeEdges							
Polygonize							
AddNode	✓						
AddEdge	✓						
AddFace	✓						
ST_Simplify	✓						
RemoveUnusedPrimitives	✓						
CreateTopoGeom	✓						
toTopoGeom	✓						
TopoElementArray_Agg							
TopoElement	✓						
clearTopoGeom	✓						
TopoGeom_addElement	✓						
TopoGeom_remElement	✓						
TopoGeom_addTopoGeom	✓						
toTopoGeom							
GetTopoGeomElementArray							
GetTopoGeomElements							
ST_SRID	✓				✓		
AsGML	✓						
AsTopoJSON	✓						
Equals	✓		✓				
Intersects	✓		✓				
geomval							
addbandarg							
rastbandarg							
raster							
reclassarg							
summarystats							
unionarg							
AddRasterConstraints							
DropRasterConstraints							
AddOverviewConstraints							

Function	geom	geog	2.5D	Curves	SQL MM	PS	T
DropOverviewConstraints							
PostGIS_GDAL_Version							
PostGIS_Raster_Lib_Build_Date							
PostGIS_Raster_Lib_Version							
ST_GDALDrivers							
ST_Contour							
ST_InterpolateRaste	✓						
UpdateRasterSRID							
ST_CreateOverview							
ST_AddBand							
ST_AsRaster	✓						
ST_Band							
ST_MakeEmptyCoverage							
ST_MakeEmptyRaster							
ST_Tile							
ST_Retile	✓						
ST_FromGDALRaster							
ST_GeoReference							
ST_Height							
ST_IsEmpty							
ST_MemSize							
ST_MetaData							
ST_NumBands							
ST_PixelHeight							
ST_PixelWidth							
ST_ScaleX							
ST_ScaleY							
ST_RasterToWorldCoord							
ST_RasterToWorldCoordX							
ST_RasterToWorldCoordY							
ST_Rotation							
ST_SkewX							
ST_SkewY							
ST_SRID							
ST_Summary							
ST_UpperLeftX							
ST_UpperLeftY							
ST_Width							
ST_WorldToRaster(	✓	'd					
ST_WorldToRaster(	✓	'dX					
ST_WorldToRaster(	✓	'dY					
ST_BandMetaData							
ST_BandNoDataValue							
ST_BandIsNoData							
ST_BandPath							
ST_BandFileSize							
ST_BandFileTimestamp							
ST_BandPixelType							
ST_MinPossibleValue							
ST_HasNoBand							
ST_PixelAsPolygon	✓						

Function	geom	geog	2.5D	Curves	SQL MM	PS	T
ST_PixelAsPolygons							
ST_PixelAsPoint	✓						
ST_PixelAsPoints							
ST_PixelAsCentroid	✓						
ST_PixelAsCentroids							
ST_Value	✓						
ST_NearestValue	✓						
ST_SetZ	✓						
ST_SetM	✓						
ST_Neighborhood	✓						
ST_SetValue	✓						
ST_SetValues							
ST_DumpValues							
ST_PixelOfValue							
ST_SetGeoReference							
ST_SetRotation							
ST_SetScale							
ST_SetSkew							
ST_SetSRID							
ST_SetUpperLeft							
ST_Resample							
ST_Rescale							
ST_Reskew							
ST_SnapToGrid							
ST_Resize							
ST_Transform							
ST_SetBandNoDataValue							
ST_SetBandIsNoData							
ST_SetBandPath							
ST_SetBandIndex							
ST_Count							
ST_CountAgg							
ST_Histogram							
ST_Quantile							
ST_SummaryStats							
ST_SummaryStatsAgg							
ST_ValueCount							
ST_RastFromWKB							
ST_RastFromHexWKB							
ST_AsBinary/ST_AsWKB							
ST_AsHexWKB							
ST_AsGDALRaster							
ST_AsJPEG							
ST_AsPNG							
ST_AsTIFF							
ST_Clip	✓						
ST_ColorMap							
ST_Grayscale							
ST_Intersection	✓						

Function	geom	geog	2.5D	Curves	SQL MM	PS	T
ST_MapAlgebra (callback function version)							
ST_MapAlgebra (expression version)							
ST_MapAlgebraExpr							
ST_MapAlgebraExpr							
ST_MapAlgebraFct							
ST_MapAlgebraFct							
ST_MapAlgebraFctNgb							
ST_Reclass							
ST_Union							
ST_Distinct4ma							
ST_InvDistWeight4ma							
ST_Max4ma							
ST_Mean4ma							
ST_Min4ma							
ST_MinDist4ma							
ST_Range4ma							
ST_StdDev4ma							
ST_Sum4ma							
ST_Aspect							
ST_HillShade							
ST_Roughness							
ST_Slope							
ST_TPI							
ST_TRI							
Box3D							
ST_ConvexHull	✓						
ST_DumpAsPolygons							
ST_Envelope	✓						
ST_MinConvexHull	✓						
ST_Polygon	✓						
&&	✓						
&<							
&>							
=							
@	✓						
~=							
~	✓						
ST_Contains							
ST_ContainsProperly							
ST_Covers							
ST_CoveredBy							
ST_Disjoint							
ST_Intersects	✓						
ST_Overlaps							
ST_Touches							

Function	geom	geog	2.5D	Curves	SQL MM	PS	T
ST_SameAlignment							
ST_NotSameAlignmentReason							
ST_Within							
ST_DWithin							
ST_DFullyWithin							
stdaddr							
rules table							
lex table							
gaz table							
debug_standardize_address							
parse_address							
standardize_address							
Drop_Indexes_Generate_Script							
Drop_Nation_Tables_Generate_Script							
Drop_State_Tables_Generate_Script							
Geocode	✓						
Geocode_Intersecti	✓						
Get_Geocode_Setting							
Get_Tract	✓						
Install_Missing_Indexes							
Loader_Generate_Census_Script							
Loader_Generate_Script							
Loader_Generate_Nation_Script							
Missing_Indexes_Generate_Script							
Normalize_Address							
Pagc_Normalize_Address							
Pprint_Addy							
Reverse_Geocode	✓						
Topology_Load_Tiger							
Set_Geocode_Setting							

13.12 New, Enhanced or changed PostGIS Functions

13.12.1 PostGIS Functions new or enhanced in 3.5

The functions given below are PostGIS functions that were added or enhanced.

Functions new in PostGIS 3.5

- CG_3DArea - Availability: 3.5.0 Computes area of 3D surface geometries. Will return 0 for solids.
- CG_3DConvexHull - Availability: 3.5.0 Computes the 3D convex hull of a geometry.
- CG_3DDifference - Availability: 3.5.0 Perform 3D difference
- CG_3DDistance - Availability: 3.5.0 Computes the minimum 3D distance between two geometries
- CG_3DIntersection - Availability: 3.5.0 Perform 3D intersection
- CG_3DIntersects - Availability: 3.5.0 Tests if two 3D geometries intersect
- CG_3DUnion - Availability: 3.5.0 Perform 3D union using postgis_sfcgal.
- CG_AlphaShape - Availability: 3.5.0 - requires SFCGAL >= 1.4.1. Computes an Alpha-shape enclosing a geometry

- CG_ApproxConvexPartition - Availability: 3.5.0 - requires SFCGAL >= 1.5.0. Computes approximal convex partition of the polygon geometry
- CG_ApproximateMedialAxis - Availability: 3.5.0 Compute the approximate medial axis of an areal geometry.
- CG_Area - Availability: 3.5.0 Calculates the area of a geometry
- CG_Difference - Availability: 3.5.0 Computes the geometric difference between two geometries
- CG_Distance - Availability: 3.5.0 Computes the minimum distance between two geometries
- CG_Extrude - Availability: 3.5.0 Extrude a surface to a related volume
- CG_ExtrudeStraightSkeleton - Availability: 3.5.0 - requires SFCGAL >= 1.5.0. Straight Skeleton Extrusion
- CG_ForceLHR - Availability: 3.5.0 Force LHR orientation
- CG_GreeneApproxConvexPartition - Availability: 3.5.0 - requires SFCGAL >= 1.5.0. Computes approximal convex partition of the polygon geometry
- CG_Intersection - Availability: 3.5.0 Computes the intersection of two geometries
- CG_Intersects - Availability: 3.5.0 Tests if two geometries intersect (they have at least one point in common)
- CG_IsPlanar - Availability: 3.5.0 Check if a surface is or not planar
- CG_IsSolid - Availability: 3.5.0 Test if the geometry is a solid. No validity check is performed.
- CG_MakeSolid - Availability: 3.5.0 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.
- CG_MinkowskiSum - Availability: 3.5.0 Performs Minkowski sum
- CG_OptimalAlphaShape - Availability: 3.5.0 - requires SFCGAL >= 1.4.1. Computes an Alpha-shape enclosing a geometry using an "optimal" alpha value.
- CG_OptimalConvexPartition - Availability: 3.5.0 - requires SFCGAL >= 1.5.0. Computes an optimal convex partition of the polygon geometry
- CG_Orientation - Availability: 3.5.0 Determine surface orientation
- CG_StraightSkeleton - Availability: 3.5.0 Compute a straight skeleton from a geometry
- CG_Tesselate - Availability: 3.5.0 Perform surface Tessellation of a polygon or polyhedralsurface and returns as a TIN or collection of TINS
- CG_Triangulate - Availability: 3.5.0 Triangulates a polygonal geometry
- CG_Union - Availability: 3.5.0 Computes the union of two geometries
- CG_Visibility - Availability: 3.5.0 - requires SFCGAL >= 1.5.0. Compute a visibility polygon from a point or a segment in a polygon geometry
- CG_Volume - Availability: 3.5.0 Computes the volume of a 3D solid. If applied to surface (even closed) geometries will return 0.
- CG_YMonotonePartition - Availability: 3.5.0 - requires SFCGAL >= 1.5.0. Computes y-monotone partition of the polygon geometry
- ST_HasM - Availability: 3.5.0 Checks if a geometry has an M (measure) dimension.
- ST_HasZ - Availability: 3.5.0 Checks if a geometry has a Z dimension.
- ST_RemoveIrrelevantPointsForView - Availability: 3.5.0 Removes points that are irrelevant for rendering a specific rectangluar view of a geometry.
- ST_RemoveSmallParts - Availability: 3.5.0 Removes small parts (polygon rings or linestrings) of a geometry.

- TopoGeo_LoadGeometry - Availability: 3.5.0 Load a geometry into an existing topology, snapping and splitting as needed.

Functions enhanced in PostGIS 3.5

- ST_Clip - Enhanced: 3.5.0 - touched argument added. Returns the raster clipped by the input geometry. If band number is not specified, all bands are processed. If crop is not specified or TRUE, the output raster is cropped. If touched is set to TRUE, then touched pixels are included, otherwise only if the center of the pixel is in the geometry it is included.

Functions changed in PostGIS 3.5

- ST_AsGeoJSON - Changed: 3.5.0 allow specifying the column containing the feature id Return a geometry or feature in GeoJSON format.
- ST_DFullyWithin - Changed: 3.5.0 : the logic behind the function now uses a test of containment within a buffer, rather than the ST_MaxDistance algorithm. Results will differ from prior versions, but should be closer to user expectations. Tests if a geometry is entirely inside a distance of another

13.12.2 PostGIS Functions new or enhanced in 3.4

The functions given below are PostGIS functions that were added or enhanced.

Functions new in PostGIS 3.4

- PostGIS_GEOS_Compiled_Version - Availability: 3.4.0 Returns the version number of the GEOS library against which PostGIS was built.
- PostGIS_PROJ_Compiled_Version - Availability: 3.5.0 Returns the version number of the PROJ library against which PostGIS was built.
- RenameTopoGeometryColumn - Availability: 3.4.0 Renames a topogeometry column
- RenameTopology - Availability: 3.4.0 Renames a topology
- ST_ClusterIntersectingWin - Availability: 3.4.0 Window function that returns a cluster id for each input geometry, clustering input geometries into connected sets.
- ST_ClusterWithinWin - Availability: 3.4.0 Window function that returns a cluster id for each input geometry, clustering using separation distance.
- ST_CoverageInvalidEdges - Availability: 3.4.0 Window function that finds locations where polygons fail to form a valid coverage.
- ST_CoverageSimplify - Availability: 3.4.0 Window function that simplifies the edges of a polygonal coverage.
- ST_CoverageUnion - Availability: 3.4.0 - requires GEOS >= 3.8.0 Computes the union of a set of polygons forming a coverage by removing shared edges.
- ST_InverseTransformPipeline - Availability: 3.4.0 Return a new geometry with coordinates transformed to a different spatial reference system using the inverse of a defined coordinate transformation pipeline.
- ST_LargestEmptyCircle - Availability: 3.4.0. Computes the largest circle not overlapping a geometry.
- ST_LineExtend - Availability: 3.4.0 Returns a line extended forwards and backwards by specified distances.
- ST_TransformPipeline - Availability: 3.4.0 Return a new geometry with coordinates transformed to a different spatial reference system using a defined coordinate transformation pipeline.
- TopoElement - Availability: 3.4.0 Converts a topogeometry to a topoelement.
- debug_standardize_address - Availability: 3.4.0 Returns a json formatted text listing the parse tokens and standardizations
- postgis_srs - Availability: 3.4.0 Return a metadata record for the requested authority and srid.

- postgis_srs_all - Availability: 3.4.0 Return metadata records for every spatial reference system in the underlying Proj database.
- postgis_srs_codes - Availability: 3.4.0 Return the list of SRS codes associated with the given authority.
- postgis_srs_search - Availability: 3.4.0 Return metadata records for projected coordinate systems that have areas of usage that fully contain the bounds parameter.

Functions enhanced in PostGIS 3.4

- PostGIS_Full_Version - Enhanced: 3.4.0 now includes extra PROJ configurations NETWORK_ENABLED, URL_ENDPOINT and DATABASE_PATH of proj.db location Reports full PostGIS version and build configuration infos.
- PostGIS_PROJ_Version - Enhanced: 3.4.0 now includes NETWORK_ENABLED, URL_ENDPOINT and DATABASE_PATH of proj.db location Returns the version number of the PROJ4 library.
- ST_AsSVG - Enhanced: 3.4.0 to support all curve types Returns SVG path data for a geometry.
- ST_ClosestPoint - Enhanced: 3.4.0 - Support for geography. Returns the 2D point on g1 that is closest to g2. This is the first point of the shortest line from one geometry to the other.
- ST_LineSubstring - Enhanced: 3.4.0 - Support for geography was introduced. Returns the part of a line between two fractional locations.
- ST_Project - Enhanced: 3.4.0 Allow geometry arguments and two-point form omitting azimuth. Returns a point projected from a start point by a distance and bearing (azimuth).
- ST_Resample - Enhanced: 3.4.0 max and min resampling options added Resample a raster using a specified resampling algorithm, new dimensions, an arbitrary grid corner and a set of raster georeferencing attributes defined or borrowed from another raster.
- ST_Rescale - Enhanced: 3.4.0 max and min resampling options added Resample a raster by adjusting only its scale (or pixel size). New pixel values are computed using the NearestNeighbor (english or american spelling), Bilinear, Cubic, CubicSpline, Lanczos, Max or Min resampling algorithm. Default is NearestNeighbor.
- ST_ShortestLine - Enhanced: 3.4.0 - support for geography. Returns the 2D shortest line between two geometries

Functions changed in PostGIS 3.4

- PostGIS_Extensions_Upgrade - Changed: 3.4.0 to add target_version argument. Packages and upgrades PostGIS extensions (e.g. postgis_raster,postgis_topology, postgis_sfcgal) to given or latest version.

13.12.3 PostGIS Functions new or enhanced in 3.3

The functions given below are PostGIS functions that were added or enhanced.

Functions new in PostGIS 3.3

- RemoveUnusedPrimitives - Availability: 3.3.0 Removes topology primitives which not needed to define existing TopoGeometry objects.
- ST_3DConvexHull - Availability: 3.3.0 Computes the 3D convex hull of a geometry.
- ST_3DUnion - Availability: 3.3.0 aggregate variant was added Perform 3D union.
- ST_AsMARC21 - Availability: 3.3.0 Returns geometry as a MARC21/XML record with a geographic datafield (034).
- ST_GeomFromMARC21 - Availability: 3.3.0, requires libxml2 2.6+ Takes MARC21/XML geographic data as input and returns a PostGIS geometry object.
- ST_Letters - Availability: 3.3.0 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 - Availability: 3.3.0 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.

13.12.4 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

- FindLayer - Availability: 3.2.0 Returns a topology.layer record by different means.
- FindTopology - Availability: 3.2.0 Returns a topology record by different means.
- GetFaceContainingPoint - Availability: 3.2.0 Finds the face containing a point.
- ST_AsFlatGeobuf - Availability: 3.2.0 Return a FlatGeobuf representation of a set of rows.
- ST_Contour - Availability: 3.2.0 Generates a set of vector contours from the provided raster band, using the GDAL contouring algorithm.
- ST_DumpSegments - Availability: 3.2.0 Returns a set of geometry_dump rows for the segments in a geometry.
- ST_FromFlatGeobuf - Availability: 3.2.0 Reads FlatGeobuf data.
- ST_FromFlatGeobufToTable - Availability: 3.2.0 Creates a table based on the structure of FlatGeobuf data.
- ST_InterpolateRaster - Availability: 3.2.0 Interpolates a gridded surface based on an input set of 3-d points, using the X- and Y-values to position the points on the grid and the Z-value of the points as the surface elevation.
- ST_SRID - Availability: 3.2.0 Returns the spatial reference identifier for a topogeometry.
- ST_Scroll - Availability: 3.2.0 Change start point of a closed LineString.
- ST_SetM - Availability: 3.2.0 Returns a geometry with the same X/Y coordinates as the input geometry, and values from the raster copied into the M dimension using the requested resample algorithm.
- ST_SetZ - Availability: 3.2.0 Returns a geometry with the same X/Y coordinates as the input geometry, and values from the raster copied into the Z dimension using the requested resample algorithm.
- TopoGeom_addTopoGeom - Availability: 3.2 Adds element of a TopoGeometry to the definition of another TopoGeometry.
- ValidateTopologyRelation - Availability: 3.2.0 Returns info about invalid topology relation records

- postgis.gdal_vsi_options - Availability: 3.2.0 A string configuration to set options used when working with an out-db raster.

Functions enhanced in PostGIS 3.2

- GetFaceByPoint - Enhanced: 3.2.0 more efficient implementation and clearer contract, stops working with invalid topologies. Finds face intersecting a given point.
- ST_ClusterKMeans - Enhanced: 3.2.0 Support for max_radius Window function that returns a cluster id for each input geometry using the K-means algorithm.
- ST_MakeValid - Enhanced: 3.2.0, added algorithm options, 'linework' and 'structure' which requires GEOS >= 3.10.0. Attempts to make an invalid geometry valid without losing vertices.
- ST_PixelAsCentroid - Enhanced: 3.2.0 Faster now implemented in C. Returns the centroid (point geometry) of the area represented by a pixel.
- ST_PixelAsCentroids - Enhanced: 3.2.0 Faster now implemented in C. Returns the centroid (point geometry) for each pixel of a raster band along with the value, the X and the Y raster coordinates of each pixel. The point geometry is the centroid of the area represented by a pixel.
- ST_Point - Enhanced: 3.2.0 srid as an extra optional argument was added. Older installs require combining with ST_SetSRID to mark the srid on the geometry. Creates a Point with X, Y and SRID values.
- ST_PointM - Enhanced: 3.2.0 srid as an extra optional argument was added. Older installs require combining with ST_SetSRID to mark the srid on the geometry. Creates a Point with X, Y, M and SRID values.
- ST_PointZ - Enhanced: 3.2.0 srid as an extra optional argument was added. Older installs require combining with ST_SetSRID to mark the srid on the geometry. Creates a Point with X, Y, Z and SRID values.
- ST_PointZM - Enhanced: 3.2.0 srid as an extra optional argument was added. Older installs require combining with ST_SetSRID to mark the srid on the geometry. Creates a Point with X, Y, Z, M and SRID values.
- ST_RemovePoint - Enhanced: 3.2.0 Remove a point from a linestring.
- ST_RemoveRepeatedPoints - Enhanced: 3.2.0 Returns a version of a geometry with duplicate points removed.
- ST_StartPoint - Enhanced: 3.2.0 returns a point for all geometries. Prior behavior returns NULLs if input was not a LineString. Returns the first point of a LineString.
- ST_Value - Enhanced: 3.2.0 resample optional argument was added. Returns the value of a given band in a given columnx, rowy pixel or at a particular geometric point. Band numbers start at 1 and assumed to be 1 if not specified. If exclude_nodata_value is set to false, then all pixels include nodata pixels are considered to intersect and return value. If exclude_nodata_value is not passed in then reads it from metadata of raster.
- TopoGeo_AddLineString - Enhanced: 3.2.0 added support for returning signed identifier. Adds a linestring to an existing topology using a tolerance and possibly splitting existing edges/faces.

Functions changed in PostGIS 3.2

- ST_Boundary - Changed: 3.2.0 support for TIN, does not use geos, does not linearize curves Returns the boundary of a geometry.
- ValidateTopology - Changed: 3.2.0 added optional bbox parameter, perform face labeling and edge linking checks. Returns a set of validatetopology_returntype objects detailing issues with topology.

13.12.5 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 - Availability: 3.1.0 Returns a single hexagon, using the provided edge size and cell coordinate within the hexagon grid space.
- ST_HexagonGrid - Availability: 3.1.0 Returns a set of hexagons and cell indices that completely cover the bounds of the geometry argument.
- ST_MaximumInscribedCircle - Availability: 3.1.0. Computes the largest circle contained within a geometry.
- ST_ReducePrecision - Availability: 3.1.0. Returns a valid geometry with points rounded to a grid tolerance.
- ST_Square - Availability: 3.1.0 Returns a single square, using the provided edge size and cell coordinate within the square grid space.
- ST_SquareGrid - Availability: 3.1.0 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. Return the Well-Known Text (WKT) representation of the geometry with SRID meta data.
- 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 - Enhanced: 3.1.0 Added margin parameter. Creates a rectangular Polygon in Web Mercator (SRID:3857) using the XYZ tile system.
- ST_UnaryUnion - Enhanced: 3.1.0 accept a gridSize parameter. Computes the union of the components of a single geometry.
- ST_Union - Enhanced: 3.1.0 accept a gridSize parameter. Computes a geometry representing the point-set union of the input geometries.

Functions changed in PostGIS 3.1

- ST_Count - Changed: 3.1.0 - The ST_Count(rastertable, rastercolumn, ...) variants removed. Use instead. Returns the number of pixels in a given band of a raster or raster coverage. If no band is specified defaults to band 1. If exclude_nodata_value is set to true, will only count pixels that are not equal to the nodata value.
- ST_Force3D - Changed: 3.1.0. Added support for supplying a non-zero Z value. Force the geometries into XYZ mode. This is an alias for ST_Force3DZ.
- ST_Force3DM - Changed: 3.1.0. Added support for supplying a non-zero M value. Force the geometries into XYM mode.

- ST_Force3DZ - Changed: 3.1.0. Added support for supplying a non-zero Z value. Force the geometries into XYZ mode.
- ST_Force4D - Changed: 3.1.0. Added support for supplying non-zero Z and M values. Force the geometries into XYZM mode.
- ST_Histogram - Changed: 3.1.0 Removed ST_Histogram(table_name, column_name) variant. Returns a set of record summarizing a raster or raster coverage data distribution separate bin ranges. Number of bins are autocomputed if not specified.
- ST_Quantile - Changed: 3.1.0 Removed ST_Quantile(table_name, column_name) variant. Compute quantiles for a raster or raster table coverage in the context of the sample or population. Thus, a value could be examined to be at the raster's 25%, 50%, 75% percentile.
- ST_SummaryStats - Changed: 3.1.0 ST_SummaryStats(rastertable, rastercolumn, ...) variants are removed. Use instead. Returns summarystats consisting of count, sum, mean, stddev, min, max for a given raster band of a raster or raster coverage. Band 1 is assumed is no band is specified.

13.12.6 PostGIS Functions new or enhanced in 3.0

The functions given below are PostGIS functions that were added or enhanced.

Functions new in PostGIS 3.0

- CG_ConstrainedDelaunayTriangles - Availability: 3.0.0 Return a constrained Delaunay triangulation around the given input geometry.
- ST_3DLineInterpolatePoint - Availability: 3.0.0 Returns a point interpolated along a 3D line at a fractional location.
- ST_ConstrainedDelaunayTriangles - Availability: 3.0.0 Return a constrained Delaunay triangulation around the given input geometry.
- ST_TileEnvelope - Availability: 3.0.0 Creates a rectangular Polygon in Web Mercator (SRID:3857) using the XYZ tile system.

Functions enhanced in PostGIS 3.0

- ST_AsMVT - Enhanced: 3.0 - added support for Feature ID. Aggregate function returning a MVT representation of a set of rows.
- ST_Contains - Enhanced: 3.0.0 enabled support for GEOMETRYCOLLECTION Tests if every point of B lies in A, and their interiors have a point in common
- ST_ContainsProperly - Enhanced: 3.0.0 enabled support for GEOMETRYCOLLECTION Tests if every point of B lies in the interior of A
- ST_CoveredBy - Enhanced: 3.0.0 enabled support for GEOMETRYCOLLECTION Tests if every point of A lies in B
- ST_Covers - Enhanced: 3.0.0 enabled support for GEOMETRYCOLLECTION Tests if every point of B lies in A
- ST_Crosses - Enhanced: 3.0.0 enabled support for GEOMETRYCOLLECTION Tests if two geometries have some, but not all, interior points in common
- ST_CurveToLine - Enhanced: 3.0.0 implemented a minimum number of segments per linearized arc to prevent topological collapse. Converts a geometry containing curves to a linear geometry.
- ST_Disjoint - Enhanced: 3.0.0 enabled support for GEOMETRYCOLLECTION Tests if two geometries have no points in common
- ST_Equals - Enhanced: 3.0.0 enabled support for GEOMETRYCOLLECTION Tests if two geometries include the same set of points
- ST_GeneratePoints - Enhanced: 3.0.0, added seed parameter Generates a multipoint of random points contained in a Polygon or MultiPolygon.

- ST_GeomFromGeoJSON - Enhanced: 3.0.0 parsed geometry defaults to SRID=4326 if not specified otherwise. Takes as input a geojson representation of a geometry and outputs a PostGIS geometry object
- 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 Returns the 3D cartesian minimum distance (based on spatial ref) between two geometries in projected units.
- ST_3DIntersects - Changed: 3.0.0 SFCGAL backend removed, GEOS backend supports TINs. Tests if two geometries spatially intersect in 3D - only for points, linestrings, polygons, polyhedral surface (area)
- ST_Area - Changed: 3.0.0 - does not depend on SFCGAL anymore. Returns the area of a polygonal geometry.
- ST_AsGeoJSON - Changed: 3.0.0 support records as input Return a geometry or feature in GeoJSON format.
- ST_AsGeoJSON - Changed: 3.0.0 output SRID if not EPSG:4326. Return a geometry or feature in GeoJSON format.
- ST_AsKML - Changed: 3.0.0 - Removed the "versioned" variant signature Return the geometry as a KML element.
- ST_Distance - Changed: 3.0.0 - does not depend on SFCGAL anymore. Returns the distance between two geometry or geography values.
- ST_Intersection - Changed: 3.0.0 does not depend on SFCGAL. Computes a geometry representing the shared portion of geometries A and B.
- ST_Intersects - Changed: 3.0.0 SFCGAL version removed and native support for 2D TINS added. Tests if two geometries intersect (they have at least one point in common)
- ST_Union - Changed: 3.0.0 does not depend on SFCGAL. Computes a geometry representing the point-set union of the input geometries.

13.12.7 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

- ST_QuantizeCoordinates - Availability: 2.5.0 Sets least significant bits of coordinates to zero
- PostGIS_Extensions_Upgrade - Availability: 2.5.0 Packages and upgrades PostGIS extensions (e.g. postgis_raster,postgis_topology, postgis_sfcgal) to given or latest version.
- ST_Angle - Availability: 2.5.0 Returns the angle between two vectors defined by 3 or 4 points, or 2 lines.
- ST_AsHexWKB - Availability: 2.5.0 Return the Well-Known Binary (WKB) in Hex representation of the raster.
- ST_BandFileSize - Availability: 2.5.0 Returns the file size of a band stored in file system. If no bandnum specified, 1 is assumed.
- ST_BandFileTimestamp - Availability: 2.5.0 Returns the file timestamp of a band stored in file system. If no bandnum specified, 1 is assumed.
- ST_ChaikinSmoothing - Availability: 2.5.0 Returns a smoothed version of a geometry, using the Chaikin algorithm
- ST_FilterByM - Availability: 2.5.0 Removes vertices based on their M value
- ST_Grayscale - Availability: 2.5.0 Creates a new one-8BUI band raster from the source raster and specified bands representing Red, Green and Blue
- ST_LineInterpolatePoints - Availability: 2.5.0 Returns points interpolated along a line at a fractional interval.
- ST_OrientedEnvelope - Availability: 2.5.0. Returns a minimum-area rectangle containing a geometry.
- ST_RastFromHexWKB - Availability: 2.5.0 Return a raster value from a Hex representation of Well-Known Binary (WKB) raster.
- ST_RastFromWKB - Availability: 2.5.0 Return a raster value from a Well-Known Binary (WKB) raster.
- ST_SetBandIndex - Availability: 2.5.0 Update the external band number of an out-db band
- ST_SetBandPath - Availability: 2.5.0 Update the external path and band number of an out-db band

Functions enhanced in PostGIS 2.5

- ST_GeometricMedian - Enhanced: 2.5.0 Added support for M as weight of points. Returns the geometric median of a Multi-Point.
- ST_AsBinary/ST_AsWKB - Enhanced: 2.5.0 Addition of ST_AsWKB Return the Well-Known Binary (WKB) representation of the raster.
- ST_AsMVT - Enhanced: 2.5.0 - added support parallel query. Aggregate function returning a MVT representation of a set of rows.
- ST_AsText - Enhanced: 2.5 - optional parameter precision introduced. Return the Well-Known Text (WKT) representation of the geometry/geography without SRID metadata.
- ST_BandMetaData - Enhanced: 2.5.0 to include outdbbandnum, filesize and filetimestamp for outdb rasters. Returns basic meta data for a specific raster band. band num 1 is assumed if none-specified.
- ST_Buffer - Enhanced: 2.5.0 - ST_Buffer geometry support was enhanced to allow for side buffering specification side=both|left|right. Computes a geometry covering all points within a given distance from a geometry.
- ST_GeomFromGeoJSON - Enhanced: 2.5.0 can now accept json and jsonb as inputs. Takes as input a geojson representation of a geometry and outputs a PostGIS geometry object

- ST_Intersects - Enhanced: 2.5.0 Supports GEOMETRYCOLLECTION. Tests if two geometries intersect (they have at least one point in common)
- ST_OffsetCurve - Enhanced: 2.5 - added support for GEOMETRYCOLLECTION and MULTILINESTRING Returns an offset line at a given distance and side from an input line.
- ST_Scale - Enhanced: 2.5.0 support for scaling relative to a local origin (origin parameter) was introduced. Scales a geometry by given factors.
- ST_Split - Enhanced: 2.5.0 support for splitting a polygon by a multiline was introduced. Returns a collection of geometries created by splitting a geometry by another geometry.
- ST_Subdivide - Enhanced: 2.5.0 reuses existing points on polygon split, vertex count is lowered from 8 to 5. Computes a rectilinear subdivision of a geometry.

Functions changed in PostGIS 2.5

- ST_GDALDrivers - Changed: 2.5.0 - add can_read and can_write columns. Returns a list of raster formats supported by PostGIS through GDAL. Only those formats with can_write=True can be used by ST_AsGDALRaster

13.12.8 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_ForcePolygonCCW - Availability: 2.4.0 Orients all exterior rings counter-clockwise and all interior rings clockwise.
- ST_ForcePolygonCW - Availability: 2.4.0 Orients all exterior rings clockwise and all interior rings counter-clockwise.
- ST_IsPolygonCCW - Availability: 2.4.0 Tests if Polygons have exterior rings oriented counter-clockwise and interior rings oriented clockwise.
- ST_IsPolygonCW - Availability: 2.4.0 Tests if Polygons have exterior rings oriented clockwise and interior rings oriented counter-clockwise.
- ST_AsGeobuf - Availability: 2.4.0 Return a Geobuf representation of a set of rows.
- ST_AsMVT - Availability: 2.4.0 Aggregate function returning a MVT representation of a set of rows.
- ST_AsMVTGeom - Availability: 2.4.0 Transforms a geometry into the coordinate space of a MVT tile.
- ST_Centroid - Availability: 2.4.0 support for geography was introduced. Returns the geometric center of a geometry.
- ST_FrechetDistance - Availability: 2.4.0 - requires GEOS >= 3.7.0 Returns the Fréchet distance between two geometries.
- ST_MakeEmptyCoverage - Availability: 2.4.0 Cover georeferenced area with a grid of empty raster tiles.

Functions enhanced in PostGIS 2.4

- Loader_Generate_Nation_Script - Enhanced: 2.4.1 zip code 5 tabulation area (zcta5) load step was fixed and when enabled, zcta5 data is loaded as a single table called zcta5_all as part of the nation script load. Generates a shell script for the specified platform that loads in the county and state lookup tables.
- Normalize_Address - Enhanced: 2.4.0 norm_addy object includes additional fields zip4 and address_alphanumeric. Given a textual street address, returns a composite norm_addy type that has road suffix, prefix and type standardized, street, streetname etc. broken into separate fields. This function will work with just the lookup data packaged with the tiger_geocoder (no need for tiger census data).

- Pagc_Normalize_Address - Enhanced: 2.4.0 norm_addy object includes additional fields zip4 and address_alphanumeric. Given a textual street address, returns a composite norm_addy type that has road suffix, prefix and type standardized, street, streetname etc. broken into separate fields. This function will work with just the lookup data packaged with the tiger_geocoder (no need for tiger census data). Requires address_standardizer extension.
- Reverse_Geocode - Enhanced: 2.4.1 if optional zcta5 dataset is loaded, the reverse_geocode function can resolve to state and zip even if the specific state data is not loaded. Refer to for details on loading zcta5 data. Takes a geometry point in a known spatial ref sys and returns a record containing an array of theoretically possible addresses and an array of cross streets. If include_strnum_range = true, includes the street range in the cross streets.
- ST_AsTWKB - Enhanced: 2.4.0 memory and speed improvements. Returns the geometry as TWKB, aka "Tiny Well-Known Binary"
- ST_Covers - Enhanced: 2.4.0 Support for polygon in polygon and line in polygon added for geography type Tests if every point of B lies in A
- ST_CurveToLine - Enhanced: 2.4.0 added support for max-deviation and max-angle tolerance, and for symmetric output. Converts a geometry containing curves to a linear geometry.
- ST_Project - Enhanced: 2.4.0 Allow negative distance and non-normalized azimuth. Returns a point projected from a start point by a distance and bearing (azimuth).
- ST_Reverse - Enhanced: 2.4.0 support for curves was introduced. Return the geometry with vertex order reversed.

Functions changed in PostGIS 2.4

- = - Changed: 2.4.0, in prior versions this was bounding box equality not a geometric equality. If you need bounding box equality, use instead. Returns TRUE if the coordinates and coordinate order geometry/geography A are the same as the coordinates and coordinate order of geometry/geography B.
- ST_Node - Changed: 2.4.0 this function uses GEOSNode internally instead of GEOSUnaryUnion. This may cause the resulting linestrings to have a different order and direction compared to PostGIS < 2.4. Nodes a collection of lines.

13.12.9 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

- ST_GeometricMedian - Availability: 2.3.0 Returns the geometric median of a MultiPoint.
- &&&(geometry,gidx) - Availability: 2.3.0 support for Block Range INdexes (BRIN) was introduced. Requires PostgreSQL 9.5+. Returns TRUE if a geometry's (cached) n-D bounding box intersects a n-D float precision bounding box (GIDX).
- &&&(gidx,geometry) - Availability: 2.3.0 support for Block Range INdexes (BRIN) was introduced. Requires PostgreSQL 9.5+. Returns TRUE if a n-D float precision bounding box (GIDX) intersects a geometry's (cached) n-D bounding box.
- &&&(gidx,gidx) - Availability: 2.3.0 support for Block Range INdexes (BRIN) was introduced. Requires PostgreSQL 9.5+. Returns TRUE if two n-D float precision bounding boxes (GIDX) intersect each other.
- &&(box2df,box2df) - Availability: 2.3.0 support for Block Range INdexes (BRIN) was introduced. Requires PostgreSQL 9.5+. Returns TRUE if two 2D float precision bounding boxes (BOX2DF) intersect each other.
- &&(box2df,geometry) - Availability: 2.3.0 support for Block Range INdexes (BRIN) was introduced. Requires PostgreSQL 9.5+. Returns TRUE if a 2D float precision bounding box (BOX2DF) intersects a geometry's (cached) 2D bounding box.
- &&(geometry,box2df) - Availability: 2.3.0 support for Block Range INdexes (BRIN) was introduced. Requires PostgreSQL 9.5+. Returns TRUE if a geometry's (cached) 2D bounding box intersects a 2D float precision bounding box (BOX2DF).
- @(box2df,box2df) - Availability: 2.3.0 support for Block Range INdexes (BRIN) was introduced. Requires PostgreSQL 9.5+. Returns TRUE if a 2D float precision bounding box (BOX2DF) is contained into another 2D float precision bounding box.

- @(box2df,geometry) - Availability: 2.3.0 support for Block Range INdexes (BRIN) was introduced. Requires PostgreSQL 9.5+. Returns TRUE if a 2D float precision bounding box (BOX2DF) is contained into a geometry's 2D bounding box.
- @(geometry,box2df) - Availability: 2.3.0 support for Block Range INdexes (BRIN) was introduced. Requires PostgreSQL 9.5+. Returns TRUE if a geometry's 2D bounding box is contained into a 2D float precision bounding box (BOX2DF).
- Populate_Topology_Layer - Availability: 2.3.0 Adds missing entries to topology.layer table by reading metadata from topo tables.
- 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 - Availability: 2.3.0 Generates a multipoint of random points contained in a Polygon or MultiPolygon.
- ST_MakeLine - Availability: 2.3.0 - Support for MultiPoint input elements was introduced Creates a LineString from Point, MultiPoint, or LineString geometries.
- ST_MinimumBoundingRadius - Availability - 2.3.0 Returns the center point and radius of the smallest circle that contains a geometry.
- ST_MinimumClearance - Availability: 2.3.0 Returns the minimum clearance of a geometry, a measure of a geometry's robustness.
- ST_MinimumClearanceLine - Availability: 2.3.0 - requires GEOS >= 3.6.0 Returns the two-point LineString spanning a geometry's minimum clearance.
- ST_Normalize - Availability: 2.3.0 Return the geometry in its canonical form.
- ST_Points - Availability: 2.3.0 Returns a MultiPoint containing the coordinates of a geometry.
- ST_VoronoiLines - Availability: 2.3.0 Returns the boundaries of the Voronoi diagram of the vertices of a geometry.
- ST_VoronoiPolygons - Availability: 2.3.0 Returns the cells of the Voronoi diagram of the vertices of a geometry.
- ST_WrapX - Availability: 2.3.0 requires GEOS Wrap a geometry around an X value.
- TopoGeom_addElement - Availability: 2.3 Adds an element to the definition of a TopoGeometry.
- TopoGeom_remElement - Availability: 2.3 Removes an element from the definition of a TopoGeometry.
- ~(box2df,box2df) - Availability: 2.3.0 support for Block Range INdexes (BRIN) was introduced. Requires PostgreSQL 9.5+. Returns TRUE if a 2D float precision bounding box (BOX2DF) contains another 2D float precision bounding box (BOX2DF).
- ~(box2df,geometry) - Availability: 2.3.0 support for Block Range INdexes (BRIN) was introduced. Requires PostgreSQL 9.5+. Returns TRUE if a 2D float precision bounding box (BOX2DF) contains a geometry's 2D bonding box.
- ~(geometry,box2df) - Availability: 2.3.0 support for Block Range INdexes (BRIN) was introduced. Requires PostgreSQL 9.5+. Returns TRUE if a geometry's 2D bonding box contains a 2D float precision bounding box (GIDX).

Functions enhanced in PostGIS 2.3

- ST_Contains - Enhanced: 2.3.0 Enhancement to PIP short-circuit extended to support MultiPoints with few points. Prior versions only supported point in polygon. Tests if every point of B lies in A, and their interiors have a point in common
- ST_Covers - Enhanced: 2.3.0 Enhancement to PIP short-circuit for geometry extended to support MultiPoints with few points. Prior versions only supported point in polygon. Tests if every point of B lies in A
- ST_Expand - Enhanced: 2.3.0 support was added to expand a box by different amounts in different dimensions. Returns a bounding box expanded from another bounding box or a geometry.
- ST_Intersects - Enhanced: 2.3.0 Enhancement to PIP short-circuit extended to support MultiPoints with few points. Prior versions only supported point in polygon. Tests if two geometries intersect (they have at least one point in common)

- ST_Segmentize - Enhanced: 2.3.0 Segmentize geography now produces equal-length subsegments Returns a modified geometry/geography having no segment longer than a given distance.
- ST_Transform - Enhanced: 2.3.0 support for direct PROJ.4 text was introduced. Return a new geometry with coordinates transformed to a different spatial reference system.
- ST_Within - Enhanced: 2.3.0 Enhancement to PIP short-circuit for geometry extended to support MultiPoints with few points. Prior versions only supported point in polygon. Tests if every point of A lies in B, and their interiors have a point in common

Functions changed in PostGIS 2.3

- ST_PointN - Changed: 2.3.0 : negative indexing available (-1 is last point) Returns the Nth point in the first LineString or circular LineString in a geometry.

13.12.10 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

- <<->> - Availability: 2.2.0 -- KNN only available for PostgreSQL 9.1+ Returns the n-D distance between the A and B geometries or bounding boxes
- ST_3DDifference - Availability: 2.2.0 Perform 3D difference
- ST_3DUnion - Availability: 2.2.0 Perform 3D union.
- ST_ApproximateMedialAxis - Availability: 2.2.0 Compute the approximate medial axis of an areal geometry.
- ST_AsEncodedPolyline - Availability: 2.2.0 Returns an Encoded Polyline from a LineString geometry.
- ST_AsTWKB - Availability: 2.2.0 Returns the geometry as TWKB, aka "Tiny Well-Known Binary"
- ST_BoundingDiagonal - Availability: 2.2.0 Returns the diagonal of a geometry's bounding box.
- ST_CPAWithin - Availability: 2.2.0 Tests if the closest point of approach of two trajectoriesis 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_CountAgg - Availability: 2.2.0 Aggregate. Returns the number of pixels in a given band of a set of rasters. If no band is specified defaults to band 1. If exclude_nodata_value is set to true, will only count pixels that are not equal to the NODATA value.
- ST_CreateOverview - Availability: 2.2.0 Create an reduced resolution version of a given raster coverage.
- ST_DistanceCPA - Availability: 2.2.0 Returns the distance between the closest point of approach of two trajectories.
- ST_ForceCurve - Availability: 2.2.0 Upcast a geometry into its curved type, if applicable.
- 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 - Availability: 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 - Availability: 2.2.0 Creates a LineString from an Encoded Polyline.

- ST_MakeSolid - Availability: 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_MapAlgebra (callback function version) - Availability: 2.2.0: Ability to add a mask Callback function version - Returns a one-band raster given one or more input rasters, band indexes and one user-specified callback function.
- ST_MemSize - Availability: 2.2.0 Returns the amount of space (in bytes) the raster takes.
- ST_RemoveRepeatedPoints - Availability: 2.2.0 Returns a version of a geometry with duplicate points removed.
- ST_Retile - Availability: 2.2.0 Return a set of configured tiles from an arbitrarily tiled raster coverage.
- ST_SetEffectiveArea - Availability: 2.2.0 Sets the effective area for each vertex, using the Visvalingam-Whyatt algorithm.
- ST_SimplifyVW - Availability: 2.2.0 Returns a simplified representation of a geometry, using the Visvalingam-Whyatt algorithm
- ST_Subdivide - Availability: 2.2.0 Computes a rectilinear subdivision of a geometry.
- ST_SummaryStatsAgg - Availability: 2.2.0 Aggregate. Returns summarystats consisting of count, sum, mean, stddev, min, max for a given raster band of a set of raster. Band 1 is assumed is no band is specified.
- ST_SwapOrdinates - Availability: 2.2.0 Returns a version of the given geometry with given ordinate values swapped.
- ST_Volume - Availability: 2.2.0 Computes the volume of a 3D solid. If applied to surface (even closed) geometries will return 0.
- parse_address - Availability: 2.2.0 Takes a 1 line address and breaks into parts
- postgis.enable_outdb_rasters - Availability: 2.2.0 A boolean configuration option to enable access to out-db raster bands.
- postgis.gdal_datapath - Availability: 2.2.0 A configuration option to assign the value of GDAL's GDAL_DATA option. If not set, the environmentally set GDAL_DATA variable is used.
- postgis.gdal_enabled_drivers - Availability: 2.2.0 A configuration option to set the enabled GDAL drivers in the PostGIS environment. Affects the GDAL configuration variable GDAL_SKIP.
- standardize_address - Availability: 2.2.0 Returns an stdaddr form of an input address utilizing lex, gaz, and rule tables.
- |=| - Availability: 2.2.0. Index-supported only available for PostgreSQL 9.5+ Returns the distance between A and B trajectories at their closest point of approach.

Functions enhanced in PostGIS 2.2

- <-> - Enhanced: 2.2.0 -- True KNN ("K nearest neighbor") behavior for geometry and geography for PostgreSQL 9.5+. Note for geography KNN is based on sphere rather than spheroid. For PostgreSQL 9.4 and below, geography support is new but only supports centroid box. Returns the 2D distance between A and B.
- AsTopoJSON - Enhanced: 2.2.1 added support for puntal inputs Returns the TopoJSON representation of a topogeometry.
- ST_Area - Enhanced: 2.2.0 - measurement on spheroid performed with GeographicLib for improved accuracy and robustness. Requires PROJ >= 4.9.0 to take advantage of the new feature. Returns the area of a polygonal geometry.
- ST_AsX3D - Enhanced: 2.2.0: Support for GeoCoordinates and axis (x/y, long/lat) flipping. Look at options for details. Returns a Geometry in X3D xml node element format: ISO-IEC-19776-1.2-X3DEncodings-XML
- ST_Azimuth - Enhanced: 2.2.0 measurement on spheroid performed with GeographicLib for improved accuracy and robustness. Requires PROJ >= 4.9.0 to take advantage of the new feature. Returns the north-based azimuth of a line between two points.
- ST_Distance - Enhanced: 2.2.0 - measurement on spheroid performed with GeographicLib for improved accuracy and robustness. Requires PROJ >= 4.9.0 to take advantage of the new feature. Returns the distance between two geometry or geography values.

- 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 - Enhanced: 2.2.0 Added support for TIN and Curves Returns a text summary of the contents of a geometry.

Functions changed in PostGIS 2.2

- <-> - Changed: 2.2.0 -- For PostgreSQL 9.5 users, old Hybrid syntax may be slower, so you'll want to get rid of that hack if you are running your code only on PostGIS 2.2+ 9.5+. See examples below. Returns the 2D distance between A and B.
- Get_Geocode_Setting - Changed: 2.2.0 : default settings are now kept in a table called geocode_settings_default. Use customized settingsa are in geocode_settings and only contain those that have been set by user. Returns value of specific setting stored in tiger.geocode_settings table.
- ST_3DClosestPoint - Changed: 2.2.0 - if 2 2D geometries are input, a 2D point is returned (instead of old behavior assuming 0 for missing Z). In case of 2D and 3D, Z is no longer assumed to be 0 for missing Z. Returns the 3D point on g1 that is closest to g2. This is the first point of the 3D shortest line.
- ST_3DDistance - Changed: 2.2.0 - In case of 2D and 3D, Z is no longer assumed to be 0 for missing Z. Returns the 3D cartesian minimum distance (based on spatial ref) between two geometries in projected units.
- ST_3DLongestLine - Changed: 2.2.0 - if 2 2D geometries are input, a 2D point is returned (instead of old behavior assuming 0 for missing Z). In case of 2D and 3D, Z is no longer assumed to be 0 for missing Z. Returns the 3D longest line between two geometries
- ST_3DMaxDistance - Changed: 2.2.0 - In case of 2D and 3D, Z is no longer assumed to be 0 for missing Z. Returns the 3D cartesian maximum distance (based on spatial ref) between two geometries in projected units.
- ST_3DShortestLine - Changed: 2.2.0 - if 2 2D geometries are input, a 2D point is returned (instead of old behavior assuming 0 for missing Z). In case of 2D and 3D, Z is no longer assumed to be 0 for missing Z. Returns the 3D shortest line between two geometries
- ST_DistanceSphere - Changed: 2.2.0 In prior versions this used to be called ST_Distance_Sphere Returns minimum distance in meters between two lon/lat geometries using a spherical earth model.
- ST_DistanceSpheroid - Changed: 2.2.0 In prior versions this was called ST_Distance_Spheroid Returns the minimum distance between two lon/lat geometries using a spheroidal earth model.
- 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 - Changed: 2.2.0 In prior versions this was called ST_Length_Spheroid and had the alias ST_3DLength_Spheroid Returns the 2D or 3D length/perimeter of a lon/lat geometry on a spheroid.
- ST_MemSize - Changed: 2.2.0 name changed to ST_MemSize to follow naming convention. Returns the amount of memory space a geometry takes.
- 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
- ValidateTopology - Changed: 2.2.0 values for id1 and id2 were swapped for 'edge crosses node' to be consistent with error description. Returns a set of validatetopology_returntype objects detailing issues with topology.

13.12.11 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

- = - Availability: 2.1.0 Returns TRUE if A's bounding box is the same as B's. Uses double precision bounding box.
- AsTopoJSON - Availability: 2.1.0 Returns the TopoJSON representation of a topogeometry.
- Drop_Nation_Tables_Generate_Script - Availability: 2.1.0 Generates a script that drops all tables in the specified schema that start with county_all, state_all or state code followed by county or state.
- Get_Geocode_Setting - Availability: 2.1.0 Returns value of specific setting stored in tiger.geocode_settings table.
- Loader_Generate_Nation_Script - Availability: 2.1.0 Generates a shell script for the specified platform that loads in the county and state lookup tables.
- Pagc_Normalize_Address - Availability: 2.1.0 Given a textual street address, returns a composite norm_addy type that has road suffix, prefix and type standardized, street, streetname etc. broken into separate fields. This function will work with just the lookup data packaged with the tiger_geocoder (no need for tiger census data). Requires address_standardizer extension.
- ST_3DArea - Availability: 2.1.0 Computes area of 3D surface geometries. Will return 0 for solids.
- ST_3DIntersection - Availability: 2.1.0 Perform 3D intersection
- ST_Box2dFromGeoHash - Availability: 2.1.0 Return a BOX2D from a GeoHash string.
- ST_ColorMap - Availability: 2.1.0 Creates a new raster of up to four 8BUI bands (grayscale, RGB, RGBA) from the source raster and a specified band. Band 1 is assumed if not specified.
- ST_Contains - Availability: 2.1.0 Return true if no points of raster rastB lie in the exterior of raster rastA and at least one point of the interior of rastB lies in the interior of rastA.
- ST_ContainsProperly - Availability: 2.1.0 Return true if rastB intersects the interior of rastA but not the boundary or exterior of rastA.
- ST_CoveredBy - Availability: 2.1.0 Return true if no points of raster rastA lie outside raster rastB.
- ST_Covers - Availability: 2.1.0 Return true if no points of raster rastB lie outside raster rastA.
- ST_DFullyWithin - Availability: 2.1.0 Return true if rasters rastA and rastB are fully within the specified distance of each other.
- ST_DWithin - Availability: 2.1.0 Return true if rasters rastA and rastB are within the specified distance of each other.
- ST_DelaunayTriangles - Availability: 2.1.0 Returns the Delaunay triangulation of the vertices of a geometry.
- ST_Disjoint - Availability: 2.1.0 Return true if raster rastA does not spatially intersect rastB.
- ST_DumpValues - Availability: 2.1.0 Get the values of the specified band as a 2-dimension array.
- ST_Extrude - Availability: 2.1.0 Extrude a surface to a related volume
- ST_ForceLHR - Availability: 2.1.0 Force LHR orientation
- ST_FromGDALRaster - Availability: 2.1.0 Returns a raster from a supported GDAL raster file.
- ST_GeomFromGeoHash - Availability: 2.1.0 Return a geometry from a GeoHash string.
- ST_InvDistWeight4ma - Availability: 2.1.0 Raster processing function that interpolates a pixel's value from the pixel's neighborhood.
- ST_MapAlgebra (callback function version) - Availability: 2.1.0 Callback function version - Returns a one-band raster given one or more input rasters, band indexes and one user-specified callback function.

- ST_MapAlgebra (expression version) - Availability: 2.1.0 Expression version - Returns a one-band raster given one or two input rasters, band indexes and one or more user-specified SQL expressions.
- ST_MinConvexHull - Availability: 2.1.0 Return the convex hull geometry of the raster excluding NODATA pixels.
- ST_MinDist4ma - Availability: 2.1.0 Raster processing function that returns the minimum distance (in number of pixels) between the pixel of interest and a neighboring pixel with value.
- ST_MinkowskiSum - Availability: 2.1.0 Performs Minkowski sum
- ST_NearestValue - Availability: 2.1.0 Returns the nearest non-NODATA value of a given band's pixel specified by a columnx and rowy or a geometric point expressed in the same spatial reference coordinate system as the raster.
- ST_Neighborhood - Availability: 2.1.0 Returns a 2-D double precision array of the non-NODATA values around a given band's pixel specified by either a columnX and rowY or a geometric point expressed in the same spatial reference coordinate system as the raster.
- ST_NotSameAlignmentReason - Availability: 2.1.0 Returns text stating if rasters are aligned and if not aligned, a reason why.
- ST_Orientation - Availability: 2.1.0 Determine surface orientation
- ST_Overlaps - Availability: 2.1.0 Return true if raster rastA and rastB intersect but one does not completely contain the other.
- ST_PixelAsCentroid - Availability: 2.1.0 Returns the centroid (point geometry) of the area represented by a pixel.
- ST_PixelAsCentroids - Availability: 2.1.0 Returns the centroid (point geometry) for each pixel of a raster band along with the value, the X and the Y raster coordinates of each pixel. The point geometry is the centroid of the area represented by a pixel.
- ST_PixelAsPoint - Availability: 2.1.0 Returns a point geometry of the pixel's upper-left corner.
- ST_PixelAsPoints - Availability: 2.1.0 Returns a point geometry for each pixel of a raster band along with the value, the X and the Y raster coordinates of each pixel. The coordinates of the point geometry are of the pixel's upper-left corner.
- ST_PixelOfValue - Availability: 2.1.0 Get the columnx, rowy coordinates of the pixel whose value equals the search value.
- ST_PointFromGeoHash - Availability: 2.1.0 Return a point from a GeoHash string.
- ST_RasterToWorldCoord - Availability: 2.1.0 Returns the raster's upper left corner as geometric X and Y (longitude and latitude) given a column and row. Column and row starts at 1.
- ST_Resize - Availability: 2.1.0 Requires GDAL 1.6.1+ Resize a raster to a new width/height
- ST_Roughness - Availability: 2.1.0 Returns a raster with the calculated "roughness" of a DEM.
- ST_SetValues - Availability: 2.1.0 Returns modified raster resulting from setting the values of a given band.
- ST_Simplify - Availability: 2.1.0 Returns a "simplified" geometry version of the given TopoGeometry using the Douglas-Peucker algorithm.
- ST_StraightSkeleton - Availability: 2.1.0 Compute a straight skeleton from a geometry
- ST_Summary - Availability: 2.1.0 Returns a text summary of the contents of the raster.
- ST_TPI - Availability: 2.1.0 Returns a raster with the calculated Topographic Position Index.
- ST_TRI - Availability: 2.1.0 Returns a raster with the calculated Terrain Ruggedness Index.
- ST_Tesselate - Availability: 2.1.0 Perform surface Tessellation of a polygon or polyhedralsurface and returns as a TIN or collection of TINS
- ST_Tile - Availability: 2.1.0 Returns a set of rasters resulting from the split of the input raster based upon the desired dimensions of the output rasters.
- ST_Touches - Availability: 2.1.0 Return true if raster rastA and rastB have at least one point in common but their interiors do not intersect.

- ST_Union - Availability: 2.1.0 ST_Union(rast, unionarg) variant was introduced. Returns the union of a set of raster tiles into a single raster composed of 1 or more bands.
- ST_Within - Availability: 2.1.0 Return true if no points of raster rastA lie in the exterior of raster rastB and at least one point of the interior of rastA lies in the interior of rastB.
- ST_WorldToRasterCoord - Availability: 2.1.0 Returns the upper left corner as column and row given geometric X and Y (longitude and latitude) or a point geometry expressed in the spatial reference coordinate system of the raster.
- Set_Geocode_Setting - Availability: 2.1.0 Sets a setting that affects behavior of geocoder functions.
- UpdateRasterSRID - Availability: 2.1.0 Change the SRID of all rasters in the user-specified column and table.
- clearTopoGeom - Availability: 2.1 Clears the content of a topo geometry.
- postgis_sfcgal_version - Availability: 2.1.0 Returns the version of SFCGAL in use

Functions enhanced in PostGIS 2.1

- ST_AddBand - Enhanced: 2.1.0 support for addbandarg added. Returns a raster with the new band(s) of given type added with given initial value in the given index location. If no index is specified, the band is added to the end.
- ST_AddBand - Enhanced: 2.1.0 support for new out-db bands added. Returns a raster with the new band(s) of given type added with given initial value in the given index location. If no index is specified, the band is added to the end.
- ST_AsBinary/ST_AsWKB - Enhanced: 2.1.0 Addition of outasin Return the Well-Known Binary (WKB) representation of the raster.
- ST_AsGML - Enhanced: 2.1.0 id support was introduced, for GML 3. Return the geometry as a GML version 2 or 3 element.
- ST_Aspect - Enhanced: 2.1.0 Uses ST_MapAlgebra() and added optional interpolate_nodata function parameter Returns the aspect (in degrees by default) of an elevation raster band. Useful for analyzing terrain.
- ST_Boundary - Enhanced: 2.1.0 support for Triangle was introduced Returns the boundary of a geometry.
- ST_Clip - Enhanced: 2.1.0 Rewritten in C Returns the raster clipped by the input geometry. If band number is not specified, all bands are processed. If crop is not specified or TRUE, the output raster is cropped. If touched is set to TRUE, then touched pixels are included, otherwise only if the center of the pixel is in the geometry it is included.
- ST_DWithin - Enhanced: 2.1.0 improved speed for geography. See Making Geography faster for details. Tests if two geometries are within a given distance
- ST_DWithin - Enhanced: 2.1.0 support for curved geometries was introduced. Tests if two geometries are within a given distance
- ST_Distance - Enhanced: 2.1.0 improved speed for geography. See Making Geography faster for details. Returns the distance between two geometry or geography values.
- ST_Distance - Enhanced: 2.1.0 - support for curved geometries was introduced. Returns the distance between two geometry or geography values.
- ST_Distinct4ma - Enhanced: 2.1.0 Addition of Variant 2 Raster processing function that calculates the number of unique pixel values in a neighborhood.
- ST_DumpPoints - Enhanced: 2.1.0 Faster speed. Reimplemented as native-C. Returns a set of geometry_dump rows for the coordinates in a geometry.
- ST_HillShade - Enhanced: 2.1.0 Uses ST_MapAlgebra() and added optional interpolate_nodata function parameter Returns the hypothetical illumination of an elevation raster band using provided azimuth, altitude, brightness and scale inputs.
- ST_MakeValid - Enhanced: 2.1.0, added support for GEOMETRYCOLLECTION and MULTIPOINT. Attempts to make an invalid geometry valid without losing vertices.

- ST_Max4ma - Enhanced: 2.1.0 Addition of Variant 2 Raster processing function that calculates the maximum pixel value in a neighborhood.
- ST_Mean4ma - Enhanced: 2.1.0 Addition of Variant 2 Raster processing function that calculates the mean pixel value in a neighborhood.
- ST_Min4ma - Enhanced: 2.1.0 Addition of Variant 2 Raster processing function that calculates the minimum pixel value in a neighborhood.
- ST_PixelAsPolygons - Enhanced: 2.1.0 exclude_nodata_value optional argument was added. Returns the polygon geometry that bounds every pixel of a raster band along with the value, the X and the Y raster coordinates of each pixel.
- ST_Polygon - Enhanced: 2.1.0 Improved Speed (fully C-Based) and the returning multipolygon is ensured to be valid. Returns a multipolygon geometry formed by the union of pixels that have a pixel value that is not no data value. If no band number is specified, band num defaults to 1.
- ST_Range4ma - Enhanced: 2.1.0 Addition of Variant 2 Raster processing function that calculates the range of pixel values in a neighborhood.
- ST_SameAlignment - Enhanced: 2.1.0 addition of Aggegrate variant Returns true if rasters have same skew, scale, spatial ref, and offset (pixels can be put on same grid without cutting into pixels) and false if they don't with notice detailing issue.
- ST_Segmentize - Enhanced: 2.1.0 support for geography was introduced. Returns a modified geometry/geography having no segment longer than a given distance.
- ST_SetGeoReference - Enhanced: 2.1.0 Addition of ST_SetGeoReference(raster, double precision, ...) variant Set Georeference 6 georeference parameters in a single call. Numbers should be separated by white space. Accepts inputs in GDAL or ESRI format. Default is GDAL.
- ST_SetValue - Enhanced: 2.1.0 Geometry variant of ST_SetValue() now supports any geometry type, not just point. The geometry variant is a wrapper around the geomval[] variant of ST_SetValues() Returns modified raster resulting from setting the value of a given band in a given columnx, rowy pixel or the pixels that intersect a particular geometry. Band numbers start at 1 and assumed to be 1 if not specified.
- ST_Slope - Enhanced: 2.1.0 Uses ST_MapAlgebra() and added optional units, scale, interpolate_nodata function parameters Returns the slope (in degrees by default) of an elevation raster band. Useful for analyzing terrain.
- ST_StdDev4ma - Enhanced: 2.1.0 Addition of Variant 2 Raster processing function that calculates the standard deviation of pixel values in a neighborhood.
- ST_Sum4ma - Enhanced: 2.1.0 Addition of Variant 2 Raster processing function that calculates the sum of all pixel values in a neighborhood.
- ST_Summary - Enhanced: 2.1.0 S flag to denote if has a known spatial reference system Returns a text summary of the contents of a geometry.
- ST_Transform - Enhanced: 2.1.0 Addition of ST_Transform(rast, alignto) variant Reprojects a raster in a known spatial reference system to another known spatial reference system using specified resampling algorithm. Options are NearestNeighbor, Bilinear, Cubic, CubicSpline, Lanczos defaulting to NearestNeighbor.
- ST_Union - Enhanced: 2.1.0 Improved Speed (fully C-Based). Returns the union of a set of raster tiles into a single raster composed of 1 or more bands.
- ST_Union - Enhanced: 2.1.0 ST_Union(rast) (variant 1) unions all bands of all input rasters. Prior versions of PostGIS assumed the first band. Returns the union of a set of raster tiles into a single raster composed of 1 or more bands.
- ST_Union - Enhanced: 2.1.0 ST_Union(rast, uniontype) (variant 4) unions all bands of all input rasters. Returns the union of a set of raster tiles into a single raster composed of 1 or more bands.
- toTopoGeom - Enhanced: 2.1.0 adds the version taking an existing TopoGeometry. Converts a simple Geometry into a topo geometry.

Functions changed in PostGIS 2.1

- ST_Aspect - Changed: 2.1.0 In prior versions, return values were in radians. Now, return values default to degrees Returns the aspect (in degrees by default) of an elevation raster band. Useful for analyzing terrain.
- 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 - Changed: 2.1.0. Up to 2.0.x this was called ST_Force_2D. Force the geometries into a "2-dimensional mode".
- ST_Force3D - Changed: 2.1.0. Up to 2.0.x this was called ST_Force_3D. Force the geometries into XYZ mode. This is an alias for ST_Force3DZ.
- ST_Force3DM - Changed: 2.1.0. Up to 2.0.x this was called ST_Force_3DM. Force the geometries into XYM mode.
- ST_Force3DZ - Changed: 2.1.0. Up to 2.0.x this was called ST_Force_3DZ. Force the geometries into XYZ mode.
- ST_Force4D - Changed: 2.1.0. Up to 2.0.x this was called ST_Force_4D. Force the geometries into XYZM mode.
- ST_ForceCollection - Changed: 2.1.0. Up to 2.0.x this was called ST_Force_Collection. Convert the geometry into a GEOMETRYCOLLECTION.
- ST_HillShade - Changed: 2.1.0 In prior versions, azimuth and altitude were expressed in radians. Now, azimuth and altitude are expressed in degrees Returns the hypothetical illumination of an elevation raster band using provided azimuth, altitude, brightness and scale inputs.
- ST_LineInterpolatePoint - Changed: 2.1.0. Up to 2.0.x this was called ST_Line_Interpolate_Point. Returns a point interpolated along a line at a fractional location.
- ST_LineLocatePoint - Changed: 2.1.0. Up to 2.0.x this was called ST_Line_Locate_Point. Returns the fractional location of the closest point on a line to a point.
- ST_LineSubstring - Changed: 2.1.0. Up to 2.0.x this was called ST_Line_Substring. Returns the part of a line between two fractional locations.
- ST_PixelAsCentroids - Changed: 2.1.1 Changed behavior of exclude_nodata_value. Returns the centroid (point geometry) for each pixel of a raster band along with the value, the X and the Y raster coordinates of each pixel. The point geometry is the centroid of the area represented by a pixel.
- ST_PixelAsPoints - Changed: 2.1.1 Changed behavior of exclude_nodata_value. Returns a point geometry for each pixel of a raster band along with the value, the X and the Y raster coordinates of each pixel. The coordinates of the point geometry are of the pixel's upper-left corner.
- ST_PixelAsPolygons - Changed: 2.1.1 Changed behavior of exclude_nodata_value. Returns the polygon geometry that bounds every pixel of a raster band along with the value, the X and the Y raster coordinates of each pixel.
- ST_Polygon - Changed: 2.1.0 In prior versions would sometimes return a polygon, changed to always return multipolygon. Returns a multipolygon geometry formed by the union of pixels that have a pixel value that is not no data value. If no band number is specified, band num defaults to 1.
- ST_RasterToWorldCoordX - Changed: 2.1.0 In prior versions, this was called ST_Raster2WorldCoordX Returns the geometric X coordinate upper left of a raster, column and row. Numbering of columns and rows starts at 1.
- ST_RasterToWorldCoordY - Changed: 2.1.0 In prior versions, this was called ST_Raster2WorldCoordY Returns the geometric Y coordinate upper left corner of a raster, column and row. Numbering of columns and rows starts at 1.
- ST_Rescale - Changed: 2.1.0 Works on rasters with no SRID Resample a raster by adjusting only its scale (or pixel size). New pixel values are computed using the NearestNeighbor (english or american spelling), Bilinear, Cubic, CubicSpline, Lanczos, Max or Min resampling algorithm. Default is NearestNeighbor.
- ST_Reskew - Changed: 2.1.0 Works on rasters with no SRID Resample a raster by adjusting only its skew (or rotation parameters). New pixel values are computed using the NearestNeighbor (english or american spelling), Bilinear, Cubic, CubicSpline or Lanczos resampling algorithm. Default is NearestNeighbor.

- ST_Segmentize - Changed: 2.1.0 As a result of the introduction of geography support, the usage ST_Segmentize('LINESTRING(1 2, 3 4)', 0.5) causes an ambiguous function error. The input needs to be properly typed as a geometry or geography. Use ST_GeomFromText, ST_GeogFromText or a cast to the required type (e.g. ST_Segmentize('LINESTRING(1 2, 3 4)'::geometry, 0.5)) Returns a modified geometry/geography having no segment longer than a given distance.
- ST_Slope - Changed: 2.1.0 In prior versions, return values were in radians. Now, return values default to degrees Returns the slope (in degrees by default) of an elevation raster band. Useful for analyzing terrain.
- ST_SnapToGrid - Changed: 2.1.0 Works on rasters with no SRID Resample a raster by snapping it to a grid. New pixel values are computed using the NearestNeighbor (english or american spelling), Bilinear, Cubic, CubicSpline or Lanczos resampling algorithm. Default is NearestNeighbor.
- ST_WorldToRasterCoordX - Changed: 2.1.0 In prior versions, this was called ST_World2RasterCoordX Returns the column in the raster of the point geometry (pt) or a X and Y world coordinate (xw, yw) represented in world spatial reference system of raster.
- ST_WorldToRasterCoordY - Changed: 2.1.0 In prior versions, this was called ST_World2RasterCoordY Returns the row in the raster of the point geometry (pt) or a X and Y world coordinate (xw, yw) represented in world spatial reference system of raster.

13.12.12 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

- && - Availability: 2.0.0 Returns TRUE if A's bounding box intersects B's bounding box.
- &&& - Availability: 2.0.0 Returns TRUE if A's n-D bounding box intersects B's n-D bounding box.
- <#> - Availability: 2.0.0 -- KNN only available for PostgreSQL 9.1+ Returns the 2D distance between A and B bounding boxes.
- <-> - Availability: 2.0.0 -- Weak KNN provides nearest neighbors based on geometry centroid distances instead of true distances. Exact results for points, inexact for all other types. Available for PostgreSQL 9.1+ Returns the 2D distance between A and B.
- @ - Availability: 2.0.0 raster @ raster, raster @ geometry introduced Returns TRUE if A's bounding box is contained by B's. Uses double precision bounding box.
- @ - Availability: 2.0.5 geometry @ raster introduced Returns TRUE if A's bounding box is contained by B's. Uses double precision bounding box.
- AddEdge - Availability: 2.0.0 Adds a linestring edge to the edge table and associated start and end points to the point nodes table of the specified topology schema using the specified linestring geometry and returns the edgeid of the new (or existing) edge.
- AddFace - Availability: 2.0.0 Registers a face primitive to a topology and gets its identifier.
- AddNode - Availability: 2.0.0 Adds a point node to the node table in the specified topology schema and returns the nodeid of new node. If point already exists as node, the existing nodeid is returned.
- AddOverviewConstraints - Availability: 2.0.0 Tag a raster column as being an overview of another.
- AddRasterConstraints - Availability: 2.0.0 Adds raster constraints to a loaded raster table for a specific column that constrains spatial ref, scaling, blocksize, alignment, bands, band type and a flag to denote if raster column is regularly blocked. The table must be loaded with data for the constraints to be inferred. Returns true if the constraint setting was accomplished and issues a notice otherwise.
- AsGML - Availability: 2.0.0 Returns the GML representation of a topogeometry.

- CopyTopology - Availability: 2.0.0 Makes a copy of a topology (nodes, edges, faces, layers and TopoGeometries) into a new schema
- DropOverviewConstraints - Availability: 2.0.0 Untag a raster column from being an overview of another.
- DropRasterConstraints - Availability: 2.0.0 Drops PostGIS raster constraints that refer to a raster table column. Useful if you need to reload data or update your raster column data.
- Drop_Indexes_Generate_Script - Availability: 2.0.0 Generates a script that drops all non-primary key and non-unique indexes on tiger schema and user specified schema. Defaults schema to tiger_data if no schema is specified.
- Drop_State_Tables_Generate_Script - Availability: 2.0.0 Generates a script that drops all tables in the specified schema that are prefixed with the state abbreviation. Defaults schema to tiger_data if no schema is specified.
- Geocode_Intersection - Availability: 2.0.0 Takes in 2 streets that intersect and a state, city, zip, and outputs a set of possible locations on the first cross street that is at the intersection, also includes a geomout as the point location in NAD 83 long lat, a normalized_address (addy) for each location, and the rating. The lower the rating the more likely the match. Results are sorted by lowest rating first. Can optionally pass in maximum results, defaults to 10. Uses Tiger data (edges, faces, addr), PostgreSQL fuzzy string matching (soundex, levenshtein).
- GetEdgeByPoint - Availability: 2.0.0 Finds the edge-id of an edge that intersects a given point.
- GetFaceByPoint - Availability: 2.0.0 Finds face intersecting a given point.
- GetNodeByPoint - Availability: 2.0.0 Finds the node-id of a node at a point location.
- GetNodeEdges - Availability: 2.0 Returns an ordered set of edges incident to the given node.
- GetRingEdges - Availability: 2.0.0 Returns the ordered set of signed edge identifiers met by walking on ana given edge side.
- GetTopoGeomElements - Availability: 2.0.0 Returns a set of topoelement objects containing the topological element_id,element_type of the given TopoGeometry (primitive elements).
- GetTopologySRID - Availability: 2.0.0 Returns the SRID of a topology in the topology.topology table given the name of the topology.
- Get_Tract - Availability: 2.0.0 Returns census tract or field from tract table of where the geometry is located. Default to returning short name of tract.
- Install_Missing_Indexes - Availability: 2.0.0 Finds all tables with key columns used in geocoder joins and filter conditions that are missing used indexes on those columns and will add them.
- Loader_Generate_Census_Script - Availability: 2.0.0 Generates a shell script for the specified platform for the specified states that will download Tiger census state tract, bg, and tabblocks data tables, stage and load into tiger_data schema. Each state script is returned as a separate record.
- Loader_Generate_Script - Availability: 2.0.0 to support Tiger 2010 structured data and load census tract (tract), block groups (bg), and blocks (tabblocks) tables . Generates a shell script for the specified platform for the specified states that will download Tiger data, stage and load into tiger_data schema. Each state script is returned as a separate record. Latest version supports Tiger 2010 structural changes and also loads census tract, block groups, and blocks tables.
- Missing_Indexes_Generate_Script - Availability: 2.0.0 Finds all tables with key columns used in geocoder joins that are missing indexes on those columns and will output the SQL DDL to define the index for those tables.
- Polygonize - Availability: 2.0.0 Finds and registers all faces defined by topology edges.
- Reverse_Geocode - Availability: 2.0.0 Takes a geometry point in a known spatial ref sys and returns a record containing an array of theoretically possible addresses and an array of cross streets. If include_strnum_range = true, includes the street range in the cross streets.
- ST_3DClosestPoint - Availability: 2.0.0 Returns the 3D point on g1 that is closest to g2. This is the first point of the 3D shortest line.
- ST_3DDFullyWithin - Availability: 2.0.0 Tests if two 3D geometries are entirely within a given 3D distance

- ST_3DDWithin - Availability: 2.0.0 Tests if two 3D geometries are within a given 3D distance
- ST_3DDistance - Availability: 2.0.0 Returns the 3D cartesian minimum distance (based on spatial ref) between two geometries in projected units.
- ST_3DIntersects - Availability: 2.0.0 Tests if two geometries spatially intersect in 3D - only for points, linestrings, polygons, polyhedral surface (area)
- ST_3DLongestLine - Availability: 2.0.0 Returns the 3D longest line between two geometries
- ST_3DMaxDistance - Availability: 2.0.0 Returns the 3D cartesian maximum distance (based on spatial ref) between two geometries in projected units.
- ST_3DShortestLine - Availability: 2.0.0 Returns the 3D shortest line between two geometries
- ST_AddEdgeModFace - Availability: 2.0 Add a new edge and, if in doing so it splits a face, modify the original face and add a new face.
- ST_AddEdgeNewFaces - Availability: 2.0 Add a new edge and, if in doing so it splits a face, delete the original face and replace it with two new faces.
- ST_AsGDALRaster - Availability: 2.0.0 - requires GDAL >= 1.6.0. Return the raster tile in the designated GDAL Raster format. Raster formats are one of those supported by your compiled library. Use ST_GDALDrivers() to get a list of formats supported by your library.
- ST_AsJPEG - Availability: 2.0.0 - requires GDAL >= 1.6.0. Return the raster tile selected bands as a single Joint Photographic Exports Group (JPEG) image (byte array). If no band is specified and 1 or more than 3 bands, then only the first band is used. If only 3 bands then all 3 bands are used and mapped to RGB.
- ST_AsLatLonText - Availability: 2.0 Return the Degrees, Minutes, Seconds representation of the given point.
- ST_AsPNG - Availability: 2.0.0 - requires GDAL >= 1.6.0. Return the raster tile selected bands as a single portable network graphics (PNG) image (byte array). If 1, 3, or 4 bands in raster and no bands are specified, then all bands are used. If more 2 or more than 4 bands and no bands specified, then only band 1 is used. Bands are mapped to RGB or RGBA space.
- ST_AsRaster - Availability: 2.0.0 - requires GDAL >= 1.6.0. Converts a PostGIS geometry to a PostGIS raster.
- ST_AsTIFF - Availability: 2.0.0 - requires GDAL >= 1.6.0. Return the raster selected bands as a single TIFF image (byte array). If no band is specified or any of specified bands does not exist in the raster, then will try to use all bands.
- ST_AsX3D - Availability: 2.0.0: ISO-IEC-19776-1.2-X3DEncodings-XML Returns a Geometry in X3D xml node element format: ISO-IEC-19776-1.2-X3DEncodings-XML
- ST_Aspect - Availability: 2.0.0 Returns the aspect (in degrees by default) of an elevation raster band. Useful for analyzing terrain.
- ST_Band - Availability: 2.0.0 Returns one or more bands of an existing raster as a new raster. Useful for building new rasters from existing rasters.
- ST_BandIsNoData - Availability: 2.0.0 Returns true if the band is filled with only nodata values.
- ST_Clip - Availability: 2.0.0 Returns the raster clipped by the input geometry. If band number is not specified, all bands are processed. If crop is not specified or TRUE, the output raster is cropped. If touched is set to TRUE, then touched pixels are included, otherwise only if the center of the pixel is in the geometry it is included.
- ST_CollectionHomogenize - Availability: 2.0.0 Returns the simplest representation of a geometry collection.
- ST_ConcaveHull - Availability: 2.0.0 Computes a possibly concave geometry that contains all input geometry vertices
- ST_Count - Availability: 2.0.0 Returns the number of pixels in a given band of a raster or raster coverage. If no band is specified defaults to band 1. If exclude_nodata_value is set to true, will only count pixels that are not equal to the nodata value.
- ST_CreateTopoGeo - Availability: 2.0 Adds a collection of geometries to a given empty topology and returns a message detailing success.

- ST_Distinct4ma - Availability: 2.0.0 Raster processing function that calculates the number of unique pixel values in a neighborhood.
- ST_FlipCoordinates - Availability: 2.0.0 Returns a version of a geometry with X and Y axis flipped.
- ST_GDALDrivers - Availability: 2.0.0 - requires GDAL >= 1.6.0. Returns a list of raster formats supported by PostGIS through GDAL. Only those formats with can_write=True can be used by ST_AsGDALRaster
- ST_GeomFromGeoJSON - Availability: 2.0.0 requires - JSON-C >= 0.9 Takes as input a geojson representation of a geometry and outputs a PostGIS geometry object
- ST_GetFaceEdges - Availability: 2.0 Returns a set of ordered edges that bound aface.
- ST_HasNoBand - Availability: 2.0.0 Returns true if there is no band with given band number. If no band number is specified, then band number 1 is assumed.
- ST_HillShade - Availability: 2.0.0 Returns the hypothetical illumination of an elevation raster band using provided azimuth, altitude, brightness and scale inputs.
- ST_Histogram - Availability: 2.0.0 Returns a set of record summarizing a raster or raster coverage data distribution separate bin ranges. Number of bins are autocomputed if not specified.
- ST_InterpolatePoint - Availability: 2.0.0 Returns the interpolated measure of a geometry closest to a point.
- ST_IsEmpty - Availability: 2.0.0 Returns true if the raster is empty (width = 0 and height = 0). Otherwise, returns false.
- ST_IsValidDetail - Availability: 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 - Availability: 2.0.0 - Support for LineString input elements was introduced Creates a LineString from Point, MultiPoint, or LineString geometries.
- ST_MakeValid - Availability: 2.0.0 Attempts to make an invalid geometry valid without losing vertices.
- ST_MapAlgebraExpr - Availability: 2.0.0 1 raster band version: Creates a new one band raster formed by applying a valid PostgreSQL algebraic operation on the input raster band and of pixeltype provided. Band 1 is assumed if no band is specified.
- ST_MapAlgebraExpr - Availability: 2.0.0 2 raster band version: Creates a new one band raster formed by applying a valid PostgreSQL algebraic operation on the two input raster bands and of pixeltype provided. band 1 of each raster is assumed if no band numbers are specified. The resulting raster will be aligned (scale, skew and pixel corners) on the grid defined by the first raster and have its extent defined by the "extenttype" parameter. Values for "extenttype" can be: INTERSECTION, UNION, FIRST, SECOND.
- ST_MapAlgebraFct - Availability: 2.0.0 1 band version - Creates a new one band raster formed by applying a valid PostgreSQL function on the input raster band and of pixeltype prodived. Band 1 is assumed if no band is specified.
- ST_MapAlgebraFct - Availability: 2.0.0 2 band version - Creates a new one band raster formed by applying a valid PostgreSQL function on the 2 input raster bands and of pixeltype prodived. Band 1 is assumed if no band is specified. Extent type defaults to INTERSECTION if not specified.
- ST_MapAlgebraFctNgb - Availability: 2.0.0 1-band version: Map Algebra Nearest Neighbor using user-defined PostgreSQL function. Return a raster which values are the result of a PLPGSQL user function involving a neighborhood of values from the input raster band.
- ST_Max4ma - Availability: 2.0.0 Raster processing function that calculates the maximum pixel value in a neighborhood.
- ST_Mean4ma - Availability: 2.0.0 Raster processing function that calculates the mean pixel value in a neighborhood.
- ST_Min4ma - Availability: 2.0.0 Raster processing function that calculates the minimum pixel value in a neighborhood.
- ST_ModEdgeHeal - Availability: 2.0 Heals two edges by deleting the node connecting them, modifying the first edgeand deleting the second edge. Returns the id of the deleted node.

- ST_MoveIsoNode - Availability: 2.0.0 Moves an isolated node in a topology from one point to another. If new apoint geometry exists as a node an error is thrown. Returns description of move.
- ST_NewEdgeHeal - Availability: 2.0 Heals two edges by deleting the node connecting them, deleting both edges,and replacing them with an edge whose direction is the same as the firstedge provided.
- ST_Node - Availability: 2.0.0 Nodes a collection of lines.
- ST_NumPatches - Availability: 2.0.0 Return the number of faces on a Polyhedral Surface. Will return null for non-polyhedral geometries.
- ST_OffsetCurve - Availability: 2.0 Returns an offset line at a given distance and side from an input line.
- ST_PatchN - Availability: 2.0.0 Returns the Nth geometry (face) of a PolyhedralSurface.
- ST_Perimeter - Availability 2.0.0: Support for geography was introduced Returns the length of the boundary of a polygonal geometry or geography.
- ST_PixelAsPolygon - Availability: 2.0.0 Returns the polygon geometry that bounds the pixel for a particular row and column.
- ST_PixelAsPolygons - Availability: 2.0.0 Returns the polygon geometry that bounds every pixel of a raster band along with the value, the X and the Y raster coordinates of each pixel.
- ST_Project - Availability: 2.0.0 Returns a point projected from a start point by a distance and bearing (azimuth).
- ST_Quantile - Availability: 2.0.0 Compute quantiles for a raster or raster table coverage in the context of the sample or population. Thus, a value could be examined to be at the raster's 25%, 50%, 75% percentile.
- ST_Range4ma - Availability: 2.0.0 Raster processing function that calculates the range of pixel values in a neighborhood.
- ST_Reclass - Availability: 2.0.0 Creates a new raster composed of band types reclassified from original. The nband is the band to be changed. If nband is not specified assumed to be 1. All other bands are returned unchanged. Use case: convert a 16BUI band to a 8BUI and so forth for simpler rendering as viewable formats.
- ST_RelateMatch - Availability: 2.0.0 Tests if a DE-9IM Intersection Matrix matches an Intersection Matrix pattern
- ST_RemEdgeModFace - Availability: 2.0 Removes an edge, and if the edge separates two facesdeletes one face and modifies the other face to cover the space of both.
- ST_RemEdgeNewFace - Availability: 2.0 Removes an edge and, if the removed edge separated two faces,delete the original faces and replace them with a new face.
- ST_Resample - Availability: 2.0.0 Requires GDAL 1.6.1+ Resample a raster using a specified resampling algorithm, new dimensions, an arbitrary grid corner and a set of raster georeferencing attributes defined or borrowed from another raster.
- ST_Rescale - Availability: 2.0.0 Requires GDAL 1.6.1+ Resample a raster by adjusting only its scale (or pixel size). New pixel values are computed using the NearestNeighbor (english or american spelling), Bilinear, Cubic, CubicSpline, Lanczos, Max or Min resampling algorithm. Default is NearestNeighbor.
- ST_Reskew - Availability: 2.0.0 Requires GDAL 1.6.1+ Resample a raster by adjusting only its skew (or rotation parameters). New pixel values are computed using the NearestNeighbor (english or american spelling), Bilinear, Cubic, CubicSpline or Lanczos resampling algorithm. Default is NearestNeighbor.
- ST_SameAlignment - Availability: 2.0.0 Returns true if rasters have same skew, scale, spatial ref, and offset (pixels can be put on same grid without cutting into pixels) and false if they don't with notice detailing issue.
- ST_SetBandIsNoData - Availability: 2.0.0 Sets the isnodata flag of the band to TRUE.
- ST_SharedPaths - Availability: 2.0.0 Returns a collection containing paths shared by the two input linestrings/multilinestrings.
- ST_Slope - Availability: 2.0.0 Returns the slope (in degrees by default) of an elevation raster band. Useful for analyzing terrain.
- ST_Snap - Availability: 2.0.0 Snap segments and vertices of input geometry to vertices of a reference geometry.

- ST_SnapToGrid - Availability: 2.0.0 Requires GDAL 1.6.1+ Resample a raster by snapping it to a grid. New pixel values are computed using the NearestNeighbor (english or american spelling), Bilinear, Cubic, CubicSpline or Lanczos resampling algorithm. Default is NearestNeighbor.
- ST_Split - Availability: 2.0.0 requires GEOS Returns a collection of geometries created by splitting a geometry by another geometry.
- ST_StdDev4ma - Availability: 2.0.0 Raster processing function that calculates the standard deviation of pixel values in a neighborhood.
- ST_Sum4ma - Availability: 2.0.0 Raster processing function that calculates the sum of all pixel values in a neighborhood.
- ST_SummaryStats - Availability: 2.0.0 Returns summarystats consisting of count, sum, mean, stddev, min, max for a given raster band of a raster or raster coverage. Band 1 is assumed is no band is specified.
- ST_Transform - Availability: 2.0.0 Requires GDAL 1.6.1+ Reprojects a raster in a known spatial reference system to another known spatial reference system using specified resampling algorithm. Options are NearestNeighbor, Bilinear, Cubic, CubicSpline, Lanczos defaulting to NearestNeighbor.
- ST_UnaryUnion - Availability: 2.0.0 Computes the union of the components of a single geometry.
- ST_Union - Availability: 2.0.0 Returns the union of a set of raster tiles into a single raster composed of 1 or more bands.
- ST_ValueCount - Availability: 2.0.0 Returns a set of records containing a pixel band value and count of the number of pixels in a given band of a raster (or a raster coverage) that have a given set of values. If no band is specified defaults to band 1. By default nodata value pixels are not counted. and all other values in the pixel are output and pixel band values are rounded to the nearest integer.
- TopoElementArray_Agg - Availability: 2.0.0 Returns a topoelementarray for a set of element_id, type arrays (topoelements).
- TopoGeo_AddLineString - Availability: 2.0.0 Adds a linestring to an existing topology using a tolerance and possibly splitting existing edges/faces.
- TopoGeo_AddPoint - Availability: 2.0.0 Adds a point to an existing topology using a tolerance and possibly splitting an existing edge.
- TopoGeo_AddPolygon - Availability: 2.0.0 Adds a polygon to an existing topology using a tolerance and possibly splitting existing edges/faces. Returns face identifiers.
- TopologySummary - Availability: 2.0.0 Takes a topology name and provides summary totals of types of objects in topology.
- Topology_Load_Tiger - Availability: 2.0.0 Loads a defined region of tiger data into a PostGIS Topology and transforming the tiger data to spatial reference of the topology and snapping to the precision tolerance of the topology.
- toTopoGeom - Availability: 2.0 Converts a simple Geometry into a topo geometry.
- ~ - Availability: 2.0.0 Returns TRUE if A's bounding box is contains B's. Uses double precision bounding box.
- ~= - Availability: 2.0.0 Returns TRUE if A's bounding box is the same as B's.

Functions enhanced in PostGIS 2.0

- && - Enhanced: 2.0.0 support for Polyhedral surfaces was introduced. Returns TRUE if A's 2D bounding box intersects B's 2D bounding box.
- AddGeometryColumn - Enhanced: 2.0.0 use_typmod argument introduced. Defaults to creating typmod geometry column instead of constraint-based. Adds a geometry column to an existing table.
- 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.

- CreateTopology - Enhanced: 2.0 added the signature accepting hasZ Creates a new topology schema and registers it in the topology.topology table.
- Geocode - Enhanced: 2.0.0 to support Tiger 2010 structured data and revised some logic to improve speed, accuracy of geocoding, and to offset point from centerline to side of street address is located on. The new parameter max_results useful for specifying number of best results or just returning the best result. Takes in an address as a string (or other normalized address) and outputs a set of possible locations which include a point geometry in NAD 83 long lat, a normalized address for each, and the rating. The lower the rating the more likely the match. Results are sorted by lowest rating first. Can optionally pass in maximum results, defaults to 10, and restrict_region (defaults to NULL)
- GeometryType - Enhanced: 2.0.0 support for Polyhedral surfaces, Triangles and TIN was introduced. Returns the type of a geometry as text.
- Populate_Geometry_Columns - Enhanced: 2.0.0 use_typmod optional argument was introduced that allows controlling if columns are created with typmodifiers or with check constraints. 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 - Enhanced: 2.0.0 - support for 2D polyhedral surfaces was introduced. Returns the area of a polygonal geometry.
- ST_AsBinary - Enhanced: 2.0.0 support for Polyhedral surfaces, Triangles and TIN was introduced. Return the OGC/ISO Well-Known Binary (WKB) representation of the geometry/geography without SRID meta data.
- ST_AsBinary - Enhanced: 2.0.0 support for higher coordinate dimensions was introduced. Return the OGC/ISO Well-Known Binary (WKB) representation of the geometry/geography without SRID meta data.
- ST_AsBinary - Enhanced: 2.0.0 support for specifying endian with geography was introduced. Return the OGC/ISO Well-Known Binary (WKB) representation of the geometry/geography without SRID meta data.
- ST_AsEWKB - Enhanced: 2.0.0 support for Polyhedral surfaces, Triangles and TIN was introduced. Return the Extended Well-Known Binary (EWKB) representation of the geometry with SRID meta data.
- ST_AsEWKT - Enhanced: 2.0.0 support for Geography, Polyhedral surfaces, Triangles and TIN was introduced. Return the Well-Known Text (WKT) representation of the geometry with SRID meta data.
- ST_AsGML - Enhanced: 2.0.0 prefix support was introduced. Option 4 for GML3 was introduced to allow using LineString instead of Curve tag for lines. GML3 Support for Polyhedral surfaces and TINS was introduced. Option 32 was introduced to output the box. Return the geometry as a GML version 2 or 3 element.
- ST_AsKML - Enhanced: 2.0.0 - Add prefix namespace, use default and named args Return the geometry as a KML element.
- ST_Azimuth - Enhanced: 2.0.0 support for geography was introduced. Returns the north-based azimuth of a line between two points.
- ST_Dimension - Enhanced: 2.0.0 support for Polyhedral surfaces and TINs was introduced. No longer throws an exception if given empty geometry. Returns the topological dimension of a geometry.
- ST_Dump - Enhanced: 2.0.0 support for Polyhedral surfaces, Triangles and TIN was introduced. Returns a set of geometry_dump rows for the components of a geometry.
- ST_DumpPoints - Enhanced: 2.0.0 support for Polyhedral surfaces, Triangles and TIN was introduced. Returns a set of geometry_dump rows for the coordinates in a geometry.
- 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 - Enhanced: 2.0.0 support for Polyhedral surfaces was introduced. Force the geometries into a "2-dimensional mode".
- ST_Force3D - Enhanced: 2.0.0 support for Polyhedral surfaces was introduced. Force the geometries into XYZ mode. This is an alias for ST_Force3DZ.
- ST_Force3DZ - Enhanced: 2.0.0 support for Polyhedral surfaces was introduced. Force the geometries into XYZ mode.
- ST_ForceCollection - Enhanced: 2.0.0 support for Polyhedral surfaces was introduced. Convert the geometry into a GEOMETRYCOLLECTION.
- ST_ForceRHR - Enhanced: 2.0.0 support for Polyhedral surfaces was introduced. Force the orientation of the vertices in a polygon to follow the Right-Hand-Rule.
- ST_GMLToSQL - Enhanced: 2.0.0 support for Polyhedral surfaces and TIN was introduced. Return a specified ST_Geometry value from GML representation. This is an alias name for ST_GeomFromGML
- ST_GMLToSQL - Enhanced: 2.0.0 default srid optional parameter added. Return a specified ST_Geometry value from GML representation. This is an alias name for ST_GeomFromGML
- ST_GeomFromEWKB - Enhanced: 2.0.0 support for Polyhedral surfaces and TIN was introduced. Return a specified ST_Geometry value from Extended Well-Known Binary representation (EWKB).
- ST_GeomFromEWKT - Enhanced: 2.0.0 support for Polyhedral surfaces and TIN was introduced. Return a specified ST_Geometry value from Extended Well-Known Text representation (EWKT).
- ST_GeomFromGML - Enhanced: 2.0.0 support for Polyhedral surfaces and TIN was introduced. Takes as input GML representation of geometry and outputs a PostGIS geometry object
- ST_GeomFromGML - Enhanced: 2.0.0 default srid optional parameter added. Takes as input GML representation of geometry and outputs a PostGIS geometry object
- ST_GeometryN - Enhanced: 2.0.0 support for Polyhedral surfaces, Triangles and TIN was introduced. Return an element of a geometry collection.
- ST_GeometryType - Enhanced: 2.0.0 support for Polyhedral surfaces was introduced. Returns the SQL-MM type of a geometry as text.
- ST_IsClosed - Enhanced: 2.0.0 support for Polyhedral surfaces was introduced. Tests if a LineStrings's start and end points are coincident. For a PolyhedralSurface tests if it is closed (volumetric).
- ST_MakeEnvelope - Enhanced: 2.0: Ability to specify an envelope without specifying an SRID was introduced. Creates a rectangular Polygon from minimum and maximum coordinates.
- ST_MakeValid - Enhanced: 2.0.1, speed improvements Attempts to make an invalid geometry valid without losing vertices.
- ST_NPoints - Enhanced: 2.0.0 support for Polyhedral surfaces was introduced. Returns the number of points (vertices) in a geometry.
- ST_NumGeometries - Enhanced: 2.0.0 support for Polyhedral surfaces, Triangles and TIN was introduced. Returns the number of elements in a geometry collection.
- 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 - Enhanced: 2.0.0 support for Polyhedral surfaces and TIN was introduced. Shifts the longitude coordinates of a geometry between -180..180 and 0..360.
- ST_Summary - Enhanced: 2.0.0 added support for geography Returns a text summary of the contents of a geometry.
- 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.
- ST_Value - Enhanced: 2.0.0 exclude_nodata_value optional argument was added. Returns the value of a given band in a given columnx, rowy pixel or at a particular geometric point. Band numbers start at 1 and assumed to be 1 if not specified. If exclude_nodata_value is set to false, then all pixels include nodata pixels are considered to intersect and return value. If exclude_nodata_value is not passed in then reads it from metadata of raster.
- ValidateTopology - Enhanced: 2.0.0 more efficient edge crossing detection and fixes for false positives that were existent in prior versions. Returns a set of validatetopology_returntype objects detailing issues with topology.

Functions changed in PostGIS 2.0

- AddGeometryColumn - Changed: 2.0.0 This function no longer updates geometry_columns since geometry_columns is a view that reads from system catalogs. It by default also does not create constraints, but instead uses the built in type modifier behavior of PostgreSQL. So for example building a wgs84 POINT column with this function is now equivalent to: ALTER TABLE some_table ADD COLUMN geom geometry(Point,4326); Adds a geometry column to an existing table.
- AddGeometryColumn - Changed: 2.0.0 If you require the old behavior of constraints use the default use_typmod, but set it to false. Adds a geometry column to an existing table.
- AddGeometryColumn - Changed: 2.0.0 Views can no longer be manually registered in geometry_columns, however views built against geometry typmod tables geometries and used without wrapper functions will register themselves correctly because they inherit the typmod behavior of their parent table column. Views that use geometry functions that output other geometries will need to be cast to typmod geometries for these view geometry columns to be registered correctly in geometry_columns. Refer to . Adds a geometry column to an existing table.
- Box3D - Changed: 2.0.0 In pre-2.0 versions, there used to be a box2d instead of box3d. Since box2d is a deprecated type, this was changed to box3d. Returns the box 3d representation of the enclosing box of the raster.
- DropGeometryColumn - Changed: 2.0.0 This function is provided for backward compatibility. Now that since geometry_columns is now a view against the system catalogs, you can drop a geometry column like any other table column using ALTER TABLE Removes a geometry column from a spatial table.
- DropGeometryTable - Changed: 2.0.0 This function is provided for backward compatibility. Now that since geometry_columns is now a view against the system catalogs, you can drop a table with geometry columns like any other table using DROP TABLE Drops a table and all its references in geometry_columns.
- Populate_Geometry_Columns - Changed: 2.0.0 By default, now uses type modifiers instead of check constraints to constrain geometry types. You can still use check constraint behavior instead by using the new use_typmod and setting it to false. 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 - Changed: 2.0.0 In prior versions this used to be called ST_Length3D Returns the 3D length of a linear geometry.

- 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 - Changed: 2.0.0 In prior versions this used to be called ST_Perimeter3D Returns the 3D perimeter of a polygonal geometry.
- ST_AsBinary - Changed: 2.0.0 Inputs to this function can not be unknown -- must be geometry. Constructs such as ST_AsBinary('POI 2)') are no longer valid and you will get an n st_asbinary(unknown) is not unique error. Code like that needs to be changed to ST_AsBinary('POINT(1 2)'::geometry);. If that is not possible, then install legacy.sql. Return the OGC/ISO Well-Known Binary (WKB) representation of the geometry/geography without SRID meta data.
- ST_AsGML - Changed: 2.0.0 use default named args Return the geometry as a GML version 2 or 3 element.
- ST_AsGeoJSON - Changed: 2.0.0 support default args and named args. Return a geometry or feature in GeoJSON format.
- ST_AsSVG - Changed: 2.0.0 to use default args and support named args Returns SVG path data for a geometry.
- ST_EndPoint - Changed: 2.0.0 no longer works with single geometry MultiLineStrings. In older versions of PostGIS a single-line MultiLineString would work with this function and return the end point. In 2.0.0 it returns NULL like any other MultiLineString. The old behavior was an undocumented feature, but people who assumed they had their data stored as LINESTRING may experience these returning NULL in 2.0.0. Returns the last point of a LineString or CircularLineString.
- ST_GDALDrivers - Changed: 2.0.6, 2.1.3 - by default no drivers are enabled, unless GUC or Environment variable gdal_enabled_drive is set. Returns a list of raster formats supported by PostGIS through GDAL. Only those formats with can_write=True can be used by ST_AsGDALRaster
- ST_GeomFromText - Changed: 2.0.0 In prior versions of PostGIS ST_GeomFromText('GEOMETRYCOLLECTION(EMPTY)') was allowed. This is now illegal in PostGIS 2.0.0 to better conform with SQL/MM standards. This should now be written as ST_GeomFromText('GEOMETRYCOLLECTION EMPTY') Return a specified ST_Geometry value from Well-Known Text representation (WKT).
- ST_GeometryN - Changed: 2.0.0 Prior versions would return NULL for singular geometries. This was changed to return the geometry for ST_GeometryN(..,1) case. Return an element of a geometry collection.
- ST_IsEmpty - Changed: 2.0.0 In prior versions of PostGIS ST_GeomFromText('GEOMETRYCOLLECTION(EMPTY)') was allowed. This is now illegal in PostGIS 2.0.0 to better conform with SQL/MM standards Tests if a geometry is empty.
- ST_Length - Changed: 2.0.0 Breaking change -- in prior versions applying this to a MULTI/POLYGON of type geography would give you the perimeter of the POLYGON/MULTIPOLYGON. In 2.0.0 this was changed to return 0 to be in line with geometry behavior. Please use ST_Perimeter if you want the perimeter of a polygon Returns the 2D length of a linear geometry.
- ST_LocateAlong - Changed: 2.0.0 in prior versions this used to be called ST_Locate_Along_Measure. Returns the point(s) on a geometry that match a measure value.
- ST_LocateBetween - Changed: 2.0.0 - in prior versions this used to be called ST_Locate_Between_Measures. Returns the portions of a geometry that match a measure range.
- ST_ModEdgeSplit - Changed: 2.0 - In prior versions, this was misnamed ST_ModEdgesSplit Split an edge by creating a new node along an existing edge, modifying the original edge and adding a new edge.
- ST_NumGeometries - Changed: 2.0.0 In prior versions this would return NULL if the geometry was not a collection/MULTI type. 2.0.0+ now returns 1 for single geometries e.g POLYGON, LINESTRING, POINT. Returns the number of elements in a geometry collection.
- ST_NumInteriorRings - Changed: 2.0.0 - in prior versions it would allow passing a MULTIPOLYGON, returning the number of interior rings of first POLYGON. Returns the number of interior rings (holes) of a Polygon.
- ST_PointN - Changed: 2.0.0 no longer works with single geometry multilinestrings. In older versions of PostGIS -- a single line multilinestring would work happily with this function and return the start point. In 2.0.0 it just returns NULL like any other multilinestring. Returns the Nth point in the first LineString or circular LineString in a geometry.
- ST_ScaleX - Changed: 2.0.0. In WKTRaster versions this was called ST_PixelSizeX. Returns the X component of the pixel width in units of coordinate reference system.

- ST_ScaleY - Changed: 2.0.0. In WKTRaster versions this was called ST_PixelSizeY. Returns the Y component of the pixel height in units of coordinate reference system.
- ST_SetScale - Changed: 2.0.0 In WKTRaster versions this was called ST_SetPixelSize. This was changed in 2.0.0. Sets the X and Y size of pixels in units of coordinate reference system. Number units/pixel width/height.
- ST_StartPoint - Changed: 2.0.0 no longer works with single geometry MultiLineStrings. In older versions of PostGIS a single-line MultiLineString would work happily with this function and return the start point. In 2.0.0 it just returns NULL like any other MultiLineString. The old behavior was an undocumented feature, but people who assumed they had their data stored as LINESTRING may experience these returning NULL in 2.0.0. Returns the first point of a LineString.

13.12.13 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

- && - Availability: 1.5.0 support for geography was introduced. Returns TRUE if A's 2D bounding box intersects B's 2D bounding box.
- PostGIS_LibXML_Version - Availability: 1.5 Returns the version number of the libxml2 library.
- ST_AddMeasure - Availability: 1.5.0 Interpolates measures along a linear geometry.
- ST_AsBinary - Availability: 1.5.0 geography support was introduced. Return the OGC/ISO Well-Known Binary (WKB) representation of the geometry/geography without SRID meta data.
- ST_AsGML - Availability: 1.5.0 geography support was introduced. Return the geometry as a GML version 2 or 3 element.
- ST_AsGeoJSON - Availability: 1.5.0 geography support was introduced. Return a geometry or feature in GeoJSON format.
- ST_AsText - Availability: 1.5 - support for geography was introduced. Return the Well-Known Text (WKT) representation of the geometry/geography without SRID metadata.
- ST_Buffer - Availability: 1.5 - ST_Buffer was enhanced to support different endcaps and join types. These are useful for example to convert road linestrings into polygon roads with flat or square edges instead of rounded edges. Thin wrapper for geography was added. Computes a geometry covering all points within a given distance from a geometry.
- ST_ClosestPoint - Availability: 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 - Availability: 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 a geometry is entirely inside a distance of another
- ST_DWithin - Availability: 1.5.0 support for geography was introduced Tests if two geometries are within a given distance
- ST_Distance - Availability: 1.5.0 geography support was introduced in 1.5. Speed improvements for planar to better handle large or many vertex geometries Returns the distance between two geometry or geography values.
- ST_DistanceSphere - Availability: 1.5 - support for other geometry types besides points was introduced. Prior versions only work with points. Returns minimum distance in meters between two lon/lat geometries using a spherical earth model.
- ST_DistanceSpheroid - Availability: 1.5 - support for other geometry types besides points was introduced. Prior versions only work with points. Returns the minimum distance between two lon/lat geometries using a spheroidal earth model.
- ST_DumpPoints - Availability: 1.5.0 Returns a set of geometry_dump rows for the coordinates in a geometry.
- ST_Envelope - Availability: 1.5.0 behavior changed to output double precision instead of float4 Returns a geometry representing the bounding box of a geometry.

13.12.15 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 - Availability: 1.3.2 Return the geometry as a GML version 2 or 3 element.
- ST_AsGeoJSON - Availability: 1.3.4 Return a geometry or feature in GeoJSON format.
- ST_CurveToLine - Availability: 1.3.0 Converts a geometry containing curves to a linear geometry.
- ST_LineToCurve - Availability: 1.3.0 Converts a linear geometry to a curved geometry.
- ST_SimplifyPreserveTopology - Availability: 1.3.3 Returns a simplified and valid representation of a geometry, using the Douglas-Peucker algorithm.

Chapter 14

Reporting Problems

14.1 Reporting Software Bugs

Reporting bugs effectively is a fundamental way to help PostGIS development. The most effective bug report is that enabling PostGIS developers to reproduce it, so it would ideally contain a script triggering it and every information regarding the environment in which it was detected. Good enough info can be extracted running `SELECT postgis_full_version()` [for PostGIS] and `SELECT version()` [for postgresql].

If you aren't using the latest release, it's worth taking a look at its release changelog first, to find out if your bug has already been fixed.

Using the PostGIS bug tracker will ensure your reports are not discarded, and will keep you informed on its handling process. Before reporting a new bug please query the database to see if it is a known one, and if it is please add any new information you have about it.

You might want to read Simon Tatham's paper about How to Report Bugs Effectively before filing a new report.

14.2 Reporting Documentation Issues

The documentation should accurately reflect the features and behavior of the software. If it doesn't, it could be because of a software bug or because the documentation is in error or deficient.

Documentation issues can also be reported to the PostGIS bug tracker.

If your revision is trivial, just describe it in a new bug tracker issue, being specific about its location in the documentation.

If your changes are more extensive, a patch is definitely preferred. This is a four step process on Unix (assuming you already have git installed):

1. Clone the PostGIS' git repository. On Unix, type:

   git clone https://git.osgeo.org/gitea/postgis/postgis.git

   This will be stored in the directory postgis

2. Make your changes to the documentation with your favorite text editor. On Unix, type (for example):

   vim doc/postgis.xml

   Note that the documentation is written in DocBook XML rather than HTML, so if you are not familiar with it please follow the example of the rest of the documentation.

3. Make a patch file containing the differences from the master copy of the documentation. On Unix, type:

   git diff doc/postgis.xml > doc.patch

4. Attach the patch to a new issue in bug tracker.

Appendix A

Appendix

A.1 PostGIS 3.5.3

2025/05/17

This version requires PostgreSQL 12-18beta1, GEOS 3.8 or higher, and Proj 6.1+. To take advantage of all features, GEOS 3.12+ is needed. To take advantage of all SFCGAL features, SFCGAL 1.5.0+ is needed.

A.1.1 Bug Fixes

Do not complain about illegal option when calling shp2pgsql -? (Sandro Santilli, Giovanni Zezza)

5862, [topology] Prevent another topology corruption with TopoGeo_addPoint near almost collinear edges (Sandro Santilli)

5841, Change approach to interrupt handling to conform to PgSQL recommended practice (Paul Ramsey)

5855, Fix index binding in ST_DFullyWithin (Paul Ramsey)

5819, Support longer names in estimated extent (Paul Ramsey)

Fix misassignment of result in _lwt_HealEdges (Maxim Korotkov)

5876, ST_AddPoint with empty argument adds garbage (Paul Ramsey)

5874, Line substring returns wrong answer (Paul Ramsey)

5829, geometry_columns with non-standard constraints (Paul Ramsey)

5818, GT-244 Fix CG_IsSolid function (Loïc Bartoletti)

5885, Fix documentation about grid-based overlay operations (Sandro Santilli)

For SFCGAL 2.1.0+ prevent using deprecated functions (Regina Obe)

A.2 PostGIS 3.5.2

2025/01/18

This version requires PostgreSQL 12-17, GEOS 3.8 or higher, and Proj 6.1+. To take advantage of all features, GEOS 3.12+ is needed. To take advantage of all SFCGAL features, SFCGAL 1.5.0+ is needed.

A.2.1 Bug Fixes

#5677, Retain SRID during unary union (Paul Ramsey)

#5833, pg_upgrade fix for postgis_sfcgal (Regina Obe)

#5564, BRIN crash fix and support for parallel in PG17+ (Paul Ramsey, Regina Obe)

A.3 PostGIS 3.5.1

2024/12/22

This version requires PostgreSQL 12-17, GEOS 3.8 or higher, and Proj 6.1+. To take advantage of all features, GEOS 3.12+ is needed. To take advantage of all SFCGAL features, SFCGAL 1.5.0+ is needed.

A.3.1 Breaking Changes

#5677, Retain SRID during unary union (Paul Ramsey)

#5792, [topology] Prevent topology corruption with TopoGeo_addPoint near almost collinear edges (Sandro Santilli)

#5795, [topology] Fix ST_NewEdgesSplit can cause invalid topology (Björn Harrtell)

#5794, [topology] Fix crash in TopoGeo_addPoint (Sandro Santilli)

#5785, [raster] ST_MapAlgebra segfaults when expression references a supernumerary rast argument (Dian M Fay)

#5787, Check that ST_ChangeEdgeGeom doesn't change winding of rings (Sandro Santilli)

#5791, Add legacy stubs for old transaction functions to allow pg_upgrade (Regina Obe)

#5800, PROJ compiled version reading the wrong minor and micro (Regina Obe)

#5790, Non-schema qualified calls causing issue with materialized views (Regina Obe)

#5812, Performance regression in ST_Within (Paul Ramsey)

#5815, Remove hash/merge promise from <> operator (Paul Ramsey)

#5823, Build support for Pg18 (Paul Ramsey)

A.3.2 Enhancements

#5782, Improve robustness of min distance calculation (Sandro Santilli)

[topology] Speedup topology building when closing large rings with many holes (Björn Harrtell)

#5810, Update tiger geocoder to handle TIGER 2024 data (Regina Obe)

A.3.3 Breaking Changes

#5799, make ST_TileEnvelope clip envelopes to tile plane extent (Paul Ramsey)

A.4 PostGIS 3.5.0

2024/09/25

This version requires PostgreSQL 12-17, GEOS 3.8 or higher, and Proj 6.1+. To take advantage of all features, GEOS 3.12+ is needed. To take advantage of all SFCGAL features, SFCGAL 1.5.0+ is needed.

Many thanks to our translation teams, in particular:

Dapeng Wang, Zuo Chenwei from HighGo (Chinese Team)

Teramoto Ikuhiro (Japanese Team)

Vincent Bre (French Team)

A.4.1 Breaking Changes

#5546, TopoGeometry <> TopoGeometry is now ambiguous, to get the old behaviour, assuming your TopoGeometry objects are named tg1 and tg2, use: (id(tg1) <> id(tg2) OR topology_id(tg1) <> topology_id(tg2) OR layer_id(tg1) <> layer_id(tg2) OR type(tg1) <> type(tg2)) (Sandro Santilli)

#5536, comments are not anymore included in PostGIS extensions (Sandro Santilli)

xmllint is now required to build comments (Sandro Santilli)

DocBook5 XSL is now required to build html (Sandro Santilli)

#5602, Drop support for GEOS 3.6 and 3.7 (Regina Obe)

#5571, Improve ST_GeneratePoints performance, but old seeded pseudo random points will need to be regenerated.

#5596, GH-749, Allow promoting column as an id in ST_AsGeoJson(record,..). Views and materialized views that use the ST_AsGeoJSON(record ..) will need rebuilding to upgrade to new signature (Jan Tojnar)

#5496, ST_Clip all variants replaced, will require rebuilding of materialized views that use them (funding from The National Institute for Agricultural and Food Research and Technology (INIA-CSIC)), Regina Obe

#5659, ST_DFullyWithin behaviour has changed to be ST_Contains(ST_Buffer(A, R), B) (Paul Ramsey)

Remove the WFS_locks extra package. (Paul Ramsey)

5747, GH-776, ST_Length: Return 0 for CurvePolygon (Dan Baston)

5770, support for GEOS 3.13 and RelateNG. Most functionality remains the same, but new GEOS predicate implementation has a few small changes.

Boundary Node Rule relate matrices might be different when using the "multi-valent end point" rule.

Relate matrices for situations with invalid MultiPolygons with shared boundaries might be different. Run ST_MakeValid to get valid inputs to feed to the calculation.

Zero length LineStrings are treated as if they are the equivalent Point object.

A.4.2 Deprecated signatures

GH-761, ST_StraightSkeleton => CG_StraightSkeleton (Loïc Bartoletti)

GH-189, All SFCGAL functions now use the prefix CG_, with the old ones using ST_ being deprecated. (Loïc Bartoletti)

A.4.3 New features

Improvements in the 'postgis' script:

- new command list-enabled
- new command list-all
- command upgrade upgrades all databases that need to be
- command status reports status of all databases

(Sandro Santilli)

#5742, expose version of PROJ at compile time (Sandro Santilli)

#5721, postgis_topology: Allow sharing sequences between different topologies (Lars Opsahl)

#5667, postgis_topology: TopoGeo_LoadGeometry (Sandro Santilli)

#5055, add explicit <> geometry operator to prevent non-unique error with <> and != (Paul Ramsey)

Add ST_HasZ/ST_HasM (Loïc Bartoletti)

GT-123, postgis_sfcgal: CG_YMonotonePartition, CG_ApproxConvexPartition, CG_GreeneApproxConvexPartition and CG_OptimalC (Loïc Bartoletti)

GT-156, postgis_sfcgal: CG_Visibility (Loïc Bartoletti)

GT-157, postgis_sfcgal: Add ST_ExtrudeStraightSkeleton (Loïc Bartoletti)

#5496, postgis_raster: ST_Clip support for touched (Regina Obe)

GH-760, postgis_sfcgal: CG_Intersection, CG_3DIntersects, CG_Intersects, CG_Difference, CG_Union (and aggregate), CG_Triangula CG_Area, CG_3DDistance, CG_Distance (Loïc Bartoletti)

#5687, Don't rely on search_path to determine postgis schema Fix for PG17 security change (Regina Obe)

#5705, GH-767, ST_RemoveIrrelevantPointsForView (Sam Peters)

#5706, GH-768, ST_RemoveSmallParts (Sam Peters)

A.4.4 Enhancements

5550, Fix upgrades from 2.x in sandboxed systems (Sandro Santilli)

#3587, postgis_topology: faster load of big lines in topologies (Sandro Santilli)

#5670, postgis_topology: faster ST_CreateTopoGeo (Sandro Santilli)

#5531, documentation format upgraded to DocBook 5 (Sandro Santilli)

#5543, allow building without documentation (Sandro Santilli)

#5596, GH-749, Allow promoting column as an id in ST_AsGeoJson(record,..). (Jan Tojnar)

GH-744, Don't create docbook.css for the HTML manual, use style.css instead (Chris Mayo)

Faster implementation of point-in-poly cached index (Paul Ramsey)

Improve performance of ST_GeneratePoints (Paul Ramsey)

#5361, ST_CurveN, ST_NumCurves and consistency in accessors on curved geometry (Paul Ramsey)

GH-761, postgis_sfcgal: Add an optional parameter to CG_StraightSkeleton (was ST_StraightSkeleton) to use m as a distance in result (Hannes Janetzek, Loïc Bartoletti)