PostGIS 3.8.0dev Handbuch

DEV (Sun Nov 15 09:26:18 AM UTC 2026 rev. 58fb588)

Inhaltsverzeichnis

1 Einführung **1**

1.1 What Is PostGIS? . . . 1
1.2 What Is in the PostGIS Distribution? . . . 1
1.3 The PostGIS Project Family . . . 2
1.4 How to Use This Manual . . . 2
1.5 Standards Support . . . 2
1.6 Project and Community . . . 3

2 PostGIS Installation **4**

2.1 Kurzfassung . . . 4
2.2 Kompilierung und Installation des Quellcodes: Detaillierte Beschreibung . . . 4
2.2.1 Nutzung des Quellcodes . . . 5
2.2.2 Systemvoraussetzungen . . . 5
2.2.3 Konfiguration . . . 6
2.2.4 Build-Prozess . . . 8
2.2.5 Build-Prozess für die PostGIS Extensions und deren Bereitstellung . . . 9
2.2.6 Softwaretest . . . 11
2.2.7 Installation . . . 13
2.3 Übliche Probleme bei der Installation . . . 14

3 PostGIS Verwaltung **15**

3.1 Leistungsoptimierung . . . 15
3.1.1 Startup . . . 15
3.1.2 Laufzeit . . . 16
3.2 Konfigurieren der Rasterunterstützung . . . 16
3.3 Erstellung räumlicher Datenbanken . . . 17
3.3.1 Die spatiale Datenbankerweiterung mittels EXTENSION aktivieren . . . 17
3.3.2 Räumliche Aktivierung der Datenbank ohne Verwendung von EXTENSION (nicht empfohlen) . . . 18
3.4 Aktualisierung von Geodatenbanken . . . 18
3.4.1 Sanfte Aktualisierung . . . 18
3.4.1.1 Soft upgrade using extensions . . . 19
3.4.1.2 Soft upgrade without extensions . . . 20
3.4.1.3 Hartes Upgrade . . . 20

4 Datenverwaltung **23**
4.1 Räumliches Datenmodell . . . 23
4.1.1 OGC-Geometrie . . . 23
4.1.1.1 Punkt . . . 24
4.1.1.2 Linie . . . 24
4.1.1.3 LinearRing . . . 24
4.1.1.4 Polygon . . . 24
4.1.1.5 MultiPoint . . . 24
4.1.1.6 MultiLineString . . . 25
4.1.1.7 MultiPolygon . . . 25
4.1.1.8 GeometryCollection . . . 25
4.1.1.9 PolyhedralSurface . . . 25
4.1.1.10 Dreieck . . . 25
4.1.1.11 TIN . . . 26
4.1.2 SQL/MM Teil 3 - Kurven . . . 26
4.1.2.1 CircularString . . . 26
4.1.2.2 CompoundCurve . . . 26
4.1.2.3 KurvenPolygon . . . 26
4.1.2.4 MultiCurve . . . 27
4.1.2.5 MultiSurface . . . 27
4.1.3 WKT und WKB . . . 27
4.2 Geometrie Datentyp . . . 28
4.2.1 PostGIS EWKB und EWKT . . . 29
4.3 Geographie Datentyp . . . 30
4.3.1 Erstellen von Grafietabellen . . . 31
4.3.2 Verwendung von Grafietabellen . . . 32
4.3.3 Wann wird der Datentyp Grafie verwendet? . . . 34
4.3.4 Fortgeschrittene FAQ's zum geographischen Datentyp . . . 35
4.4 Geometrievalidierung . . . 35
4.4.1 Einfache Geometrie . . . 35
4.4.2 Gültige Geometrie . . . 37
4.4.3 Verwaltung der Gültigkeit . . . 42
4.5 Räumliche Bezugssysteme . . . 44
4.5.1 SPATIAL_REF_SYS Tabelle . . . 45
4.5.2 Benutzerdefinierte räumliche Bezugssysteme . . . 46
4.6 Räumliche Tabellen . . . 46
4.6.1 Erstellung einer räumlichen Tabelle . . . 46
4.6.2 GEOMETRY_COLUMNS Ansicht . . . 47
4.6.3 Manuelles Registrieren von Geometriespalten . . . 48

4.7 Laden von Geodaten . . . 51
4.7.1 SQL zum Laden von Daten verwenden . . . 51
4.7.2 Verwendung des Shapefile Loaders . . . 51
4.8 Extrahieren von Geodaten . . . 53
4.8.1 SQL zum Extrahieren von Daten verwenden . . . 53
4.8.2 Verwendung des Shapefile-Dumpers . . . 54
4.9 Räumliche Indizes . . . 55
4.9.1 GiST-Indizes . . . 56
4.9.2 BRIN Indizes . . . 56
4.9.3 SP-GiST Indizes . . . 58
4.9.4 Abstimmung der Indexverwendung . . . 59

5 Räumliche Abfrage **61**
5.1 Räumliche Beziehungen feststellen . . . 61
5.1.1 Dimensionell erweitertes 9-Schnitte-Modell . . . 61
5.1.2 Benannte räumliche Beziehungen . . . 65
5.1.3 Allgemeine räumliche Beziehungen . . . 65
5.2 Räumliche Indizes verwenden . . . 67
5.3 Beispiele für Spatial SQL . . . 68

6 Performance Tipps **71**
6.1 Kleine Tabellen mit großen Geometrien . . . 71
6.1.1 Problembeschreibung . . . 71
6.1.2 Umgehungslösung . . . 71
6.2 CLUSTER auf die geometrischen Indizes . . . 72
6.3 Vermeidung von Dimensionsumrechnungen . . . 72

7 Referenz PostGIS **73**
7.1 PostgreSQL und PostGIS Datentypen - Geometry/Geography/Box . . . 73
7.1.1 box2d . . . 73
7.1.2 box3d . . . 74
7.1.3 geometry . . . 74
7.1.4 geometry_dump . . . 75
7.1.5 geography . . . 75
7.2 Geometrische Managementfunktionen . . . 76
7.2.1 AddGeometryColumn . . . 76
7.2.2 DropGeometryColumn . . . 78
7.2.3 DropGeometryTable . . . 79
7.2.4 Find_SRID . . . 80
7.2.5 Populate_Geometry_Columns . . . 80

7.2.6 UpdateGeometrySRID . . . 82
7.3 Geometrische Konstruktoren . . . 83
7.3.1 ST_Collect . . . 83
7.3.2 ST_LineFromMultiPoint . . . 87
7.3.3 ST_MakeEnvelope . . . 88
7.3.4 ST_MakeLine . . . 89
7.3.5 ST_MakePoint . . . 93
7.3.6 ST_MakePointM . . . 95
7.3.7 ST_MakePolygon . . . 97
7.3.8 ST_Point . . . 101
7.3.9 ST_PointZ . . . 102
7.3.10 ST_PointM . . . 103
7.3.11 ST_PointZM . . . 103
7.3.12 ST_Polygon . . . 104
7.3.13 ST_TileEnvelope . . . 106
7.3.14 ST_HexagonGrid . . . 108
7.3.15 ST_Hexagon . . . 113
7.3.16 ST_SquareGrid . . . 115
7.3.17 ST_Square . . . 116
7.3.18 ST_Letters . . . 117
7.3.19 ST_MakeNurbsCurve . . . 119
7.4 Geometrische Zugriffsfunktionen . . . 122
7.4.1 GeometryType . . . 122
7.4.2 ST_Boundary . . . 124
7.4.3 ST_BoundingDiagonal . . . 130
7.4.4 ST_CoordDim . . . 131
7.4.5 ST_Dimension . . . 132
7.4.6 ST_Dump . . . 134
7.4.7 ST_DumpPoints . . . 137
7.4.8 ST_DumpSegments . . . 142
7.4.9 ST_DumpRings . . . 145
7.4.10 ST_EndPoint . . . 147
7.4.11 ST_Envelope . . . 151
7.4.12 ST_ExteriorRing . . . 155
7.4.13 ST_GeometryN . . . 158
7.4.14 ST_GeometryType . . . 162
7.4.15 ST_HasArc . . . 163
7.4.16 ST_InteriorRingN . . . 164
7.4.17 ST_NumCurves . . . 166

7.4.18 ST_CurveN . . . 167
7.4.19 ST_IsClosed . . . 168
7.4.20 ST_IsCollection . . . 173
7.4.21 ST_IsEmpty . . . 175
7.4.22 ST_IsPolygonCCW . . . 177
7.4.23 ST_IsPolygonCW . . . 178
7.4.24 ST_IsRing . . . 179
7.4.25 ST_IsSimple . . . 181
7.4.26 ST_IsSimpleDetail . . . 183
7.4.27 ST_M . . . 185
7.4.28 ST_MemSize . . . 186
7.4.29 ST_NDims . . . 187
7.4.30 ST_NPoints . . . 188
7.4.31 ST_NRings . . . 189
7.4.32 ST_NumGeometries . . . 190
7.4.33 ST_NumInteriorRings . . . 191
7.4.34 ST_NumInteriorRing . . . 192
7.4.35 ST_NumPatches . . . 192
7.4.36 ST_NumPoints . . . 193
7.4.37 ST_PatchN . . . 194
7.4.38 ST_PointN . . . 195
7.4.39 ST_Points . . . 198
7.4.40 ST_StartPoint . . . 199
7.4.41 ST_StartM . . . 204
7.4.42 ST_EndM . . . 207
7.4.43 ST_SetStartM . . . 210
7.4.44 ST_SetEndM . . . 213
7.4.45 ST_Summary . . . 216
7.4.46 ST_X . . . 217
7.4.47 ST_Y . . . 218
7.4.48 ST_Z . . . 219
7.4.49 ST_ZMFlag . . . 220
7.4.50 ST_HasZ . . . 221
7.4.51 ST_HasM . . . 222
7.4.52 ST_ControlPoints . . . 224
7.4.53 ST_Degree . . . 225
7.4.54 ST_Weights . . . 227
7.4.55 ST_Knots . . . 228
7.4.56 ST_NumControlPoints . . . 230

7.4.57 ST_NurbsCurveIsRational . . . 232
7.4.58 ST_IsNurbsCurve . . . 234
7.4.59 ST_Evaluate . . . 236
7.4.60 ST_NurbsToLineString . . . 237
7.5 Geometrische Editoren . . . 239
7.5.1 ST_AddPoint . . . 239
7.5.2 ST_CollectionExtract . . . 240
7.5.3 ST_CollectionHomogenize . . . 243
7.5.4 ST_CurveToLine . . . 247
7.5.5 ST_Scroll . . . 251
7.5.6 ST_FlipCoordinates . . . 252
7.5.7 ST_Force2D . . . 253
7.5.8 ST_Force3D . . . 255
7.5.9 ST_Force3DZ . . . 257
7.5.10 ST_Force3DM . . . 259
7.5.11 ST_Force4D . . . 261
7.5.12 ST_ForceCollection . . . 263
7.5.13 ST_ForceCurve . . . 266
7.5.14 ST_ForcePolygonCCW . . . 267
7.5.15 ST_ForcePolygonCW . . . 268
7.5.16 ST_ForceSFS . . . 270
7.5.17 ST_ForceRHR . . . 271
7.5.18 ST_LineExtend . . . 272
7.5.19 ST_LineToCurve . . . 273
7.5.20 ST_Multi . . . 275
7.5.21 ST_Normalize . . . 276
7.5.22 ST_Project . . . 277
7.5.23 ST_QuantizeCoordinates . . . 279
7.5.24 ST_RemovePoint . . . 281
7.5.25 ST_RemoveRepeatedPoints . . . 281
7.5.26 ST_RemoveIrrelevantPointsForView . . . 285
7.5.27 ST_RemoveSmallParts . . . 289
7.5.28 ST_Reverse . . . 291
7.5.29 ST_Segmentize . . . 292
7.5.30 ST_SetPoint . . . 295
7.5.31 ST_ShiftLongitude . . . 298
7.5.32 ST_WrapX . . . 300
7.5.33 ST_SnapToGrid . . . 301
7.5.34 ST_Snap . . . 304

7.5.35 ST_SwapOrdinates . . . 308
7.6 Geometrievalidierung . . . 309
7.6.1 ST_IsValid . . . 309
7.6.2 ST_IsValidDetail . . . 312
7.6.3 ST_IsValidReason . . . 314
7.6.4 ST_MakeValid . . . 316
7.7 Funktionen des räumlichen Bezugssystems . . . 319
7.7.1 ST_InverseTransformPipeline . . . 319
7.7.2 ST_SetSRID . . . 322
7.7.3 ST_SRID . . . 324
7.7.4 ST_Transform . . . 324
7.7.5 ST_TransformPipeline . . . 329
7.7.6 postgis_srs_codes . . . 332
7.7.7 postgis_srs . . . 333
7.7.8 postgis_srs_all . . . 334
7.7.9 postgis_srs_search . . . 334
7.8 Geometrische Eingabe . . . 336
7.8.1 Well-known-Text (WKT) Repräsentation . . . 336
7.8.1.1 ST_BdPolyFromText . . . 336
7.8.1.2 ST_BdMPolyFromText . . . 337
7.8.1.3 ST_GeogFromText . . . 337
7.8.1.4 ST_GeographyFromText . . . 338
7.8.1.5 ST_GeomCollFromText . . . 339
7.8.1.6 ST_GeomFromEWKT . . . 339
7.8.1.7 ST_GeomFromMARC21 . . . 342
7.8.1.8 ST_GeometryFromText . . . 346
7.8.1.9 ST_GeomFromText . . . 347
7.8.1.10 ST_LineFromText . . . 352
7.8.1.11 ST_MLineFromText . . . 353
7.8.1.12 ST_MPointFromText . . . 354
7.8.1.13 ST_MPolyFromText . . . 354
7.8.1.14 ST_PointFromText . . . 357
7.8.1.15 ST_PolygonFromText . . . 359
7.8.1.16 ST_WKTToSQL . . . 361
7.8.2 Well-known-Binary (WKB) Repräsentation . . . 361
7.8.2.1 ST_GeogFromWKB . . . 361
7.8.2.2 ST_GeomFromEWKB . . . 362
7.8.2.3 ST_GeomFromWKB . . . 364
7.8.2.4 ST_LineFromWKB . . . 366

7.8.2.5 ST_LinestringFromWKB . . . 368
7.8.2.6 ST_PointFromWKB . . . 369
7.8.2.7 ST_WKBToSQL . . . 370
7.8.3 Weitere Formate . . . 371
7.8.3.1 ST_Box2dFromGeoHash . . . 371
7.8.3.2 ST_GeomFromGeoHash . . . 372
7.8.3.3 ST_GeomFromGML . . . 373
7.8.3.4 ST_GeomFromGeoJSON . . . 378
7.8.3.5 ST_GeomFromKML . . . 380
7.8.3.6 ST_GeomFromTWKB . . . 381
7.8.3.7 ST_GMLToSQL . . . 383
7.8.3.8 ST_LineFromEncodedPolyline . . . 383
7.8.3.9 ST_PointFromGeoHash . . . 385
7.8.3.10 ST_FromFlatGeobufToTable . . . 387
7.8.3.11 ST_FromFlatGeobuf . . . 388
7.9 Geometrieausgabe . . . 388
7.9.1 Well-known-Text (WKT) Repräsentation . . . 388
7.9.1.1 ST_AsEWKT . . . 388
7.9.1.2 ST_AsText . . . 390
7.9.2 Well-known-Binary (WKB) Repräsentation . . . 392
7.9.2.1 ST_AsBinary . . . 392
7.9.2.2 ST_AsEWKB . . . 394
7.9.2.3 ST_AsHEXEWKB . . . 395
7.9.3 Weitere Formate . . . 396
7.9.3.1 ST_AsEncodedPolyline . . . 396
7.9.3.2 ST_AsFlatGeobuf . . . 398
7.9.3.3 ST_AsGeobuf . . . 398
7.9.3.4 ST_AsGeoJSON . . . 399
7.9.3.5 ST_AsGML . . . 401
7.9.3.6 ST_AsKML . . . 405
7.9.3.7 ST_AsLatLonText . . . 406
7.9.3.8 ST_AsMARC21 . . . 408
7.9.3.9 ST_AsMVTGeom . . . 410
7.9.3.10 ST_AsMVT . . . 412
7.9.3.11 ST_AsSVG . . . 413
7.9.3.12 ST_AsTWKB . . . 416
7.9.3.13 ST_AsX3D . . . 417
7.9.3.14 ST_GeoHash . . . 421
7.10 Operatoren . . . 423

7.10.1 Bounding-Box-Operatoren . . . 423
7.10.1.1 && . . . 423
7.10.1.2 &&(geometry,box2df) . . . 424
7.10.1.3 &&(box2df,geometry) . . . 425
7.10.1.4 &&(box2df,box2df) . . . 426
7.10.1.5 &&& . . . 426
7.10.1.6 &&&(geometry,gidx) . . . 429
7.10.1.7 &&&(gidx,geometry) . . . 430
7.10.1.8 &&&(gidx,gidx) . . . 431
7.10.1.9 &< . . . 432
7.10.1.10 &<| . . . 433
7.10.1.11 &> . . . 434
7.10.1.12 << . . . 436
7.10.1.13 <<| . . . 437
7.10.1.14 = . . . 438
7.10.1.15 >> . . . 441
7.10.1.16 @ . . . 443
7.10.1.17 @(geometry,box2df) . . . 444
7.10.1.18 @(box2df,geometry) . . . 445
7.10.1.19 @(box2df,box2df) . . . 446
7.10.1.20 |&> . . . 446
7.10.1.21 |>> . . . 448
7.10.1.22 ~ . . . 449
7.10.1.23 ~(geometry,box2df) . . . 451
7.10.1.24 ~(box2df,geometry) . . . 451
7.10.1.25 ~(box2df,box2df) . . . 452
7.10.1.26 ~= . . . 453
7.10.2 Operatoren . . . 454
7.10.2.1 <-> . . . 454
7.10.2.2 |=| . . . 456
7.10.2.3 <#> . . . 457
7.10.2.4 <<->> . . . 459
7.11 Lagevergleiche . . . 459
7.11.1 Topologische Beziehungen . . . 459
7.11.1.1 ST_3DIntersects . . . 459
7.11.1.2 ST_Contains . . . 462
7.11.1.3 ST_ContainsProperly . . . 473
7.11.1.4 ST_CoveredBy . . . 476
7.11.1.5 ST_Covers . . . 478

7.11.1.6 ST_Crosses . . . 480
7.11.1.7 ST_Disjoint . . . 484
7.11.1.8 ST_Equals . . . 486
7.11.1.9 ST_Intersects . . . 487
7.11.1.10 ST_LineCrossingDirection . . . 490
7.11.1.11 ST_OrderingEquals . . . 493
7.11.1.12 ST_Overlaps . . . 496
7.11.1.13 ST_Relate . . . 502
7.11.1.14 ST_RelateMatch . . . 506
7.11.1.15 ST_Touches . . . 507
7.11.1.16 ST_Within . . . 514
7.11.2 Fernbeziehungen . . . 517
7.11.2.1 ST_3DDWithin . . . 517
7.11.2.2 ST_3DDFullyWithin . . . 518
7.11.2.3 ST_DFullyWithin . . . 519
7.11.2.4 ST_DWithin . . . 521
7.11.2.5 ST_PointInsideCircle . . . 524
7.12 Messfunktionen . . . 525
7.12.1 ST_Area . . . 525
7.12.2 ST_Azimuth . . . 528
7.12.3 ST_Angle . . . 531
7.12.4 ST_ClosestPoint . . . 533
7.12.5 ST_3DClosestPoint . . . 535
7.12.6 ST_Distance . . . 538
7.12.7 ST_3DDistance . . . 540
7.12.8 ST_DistanceSphere . . . 542
7.12.9 ST_DistanceSpheroid . . . 543
7.12.10 ST_FrechetDistance . . . 545
7.12.11 ST_HausdorffDistance . . . 547
7.12.12 ST_Length . . . 549
7.12.13 ST_Length2D . . . 552
7.12.14 ST_3DLength . . . 552
7.12.15 ST_LengthSpheroid . . . 553
7.12.16 ST_LongestLine . . . 556
7.12.17 ST_3DLongestLine . . . 559
7.12.18 ST_MaxDistance . . . 562
7.12.19 ST_3DMaxDistance . . . 565
7.12.20 ST_MinimumClearance . . . 566
7.12.21 ST_MinimumClearanceLine . . . 567

7.12.22 ST_Perimeter . . . 568
7.12.23 ST_Perimeter2D . . . 572
7.12.24 ST_3DPerimeter . . . 572
7.12.25 ST_ShortestLine . . . 574
7.12.26 ST_3DShortestLine . . . 575
7.13 Overlay-Funktionen . . . 578
7.13.1 ST_ClipByBox2D . . . 578
7.13.2 ST_Difference . . . 579
7.13.3 ST_Intersection . . . 581
7.13.4 ST_MemUnion . . . 585
7.13.5 ST_Node . . . 585
7.13.6 ST_Split . . . 587
7.13.7 ST_Subdivide . . . 590
7.13.8 ST_SymDifference . . . 594
7.13.9 ST_UnaryUnion . . . 596
7.13.10 ST_Union . . . 596
7.14 Geometrieverarbeitung . . . 602
7.14.1 ST_Buffer . . . 602
7.14.2 ST_BuildArea . . . 611
7.14.3 ST_Centroid . . . 614
7.14.4 ST_ChaikinSmoothing . . . 619
7.14.5 ST_CatmullRomSmoothing . . . 623
7.14.6 ST_ConcaveHull . . . 625
7.14.7 ST_ConvexHull . . . 629
7.14.8 ST_DelaunayTriangles . . . 631
7.14.9 ST_FilterByM . . . 634
7.14.10 ST_GeneratePoints . . . 636
7.14.11 ST_GeometricMedian . . . 638
7.14.12 ST_LineMerge . . . 639
7.14.13 ST_MaximumInscribedCircle . . . 644
7.14.14 ST_LargestEmptyCircle . . . 646
7.14.15 ST_MinimumBoundingCircle . . . 648
7.14.16 ST_MinimumBoundingRadius . . . 650
7.14.17 ST_OrientedEnvelope . . . 651
7.14.18 ST_OffsetCurve . . . 653
7.14.19 ST_PointOnSurface . . . 660
7.14.20 ST_Polygonize . . . 665
7.14.21 ST_ReducePrecision . . . 668
7.14.22 ST_SharedPaths . . . 672

7.14.23 ST_Simplify . . . 674
7.14.24 ST_SimplifyPreserveTopology . . . 677
7.14.25 ST_SimplifyPolygonHull . . . 680
7.14.26 ST_SimplifyVW . . . 683
7.14.27 ST_SetEffectiveArea . . . 687
7.14.28 ST_TriangulatePolygon . . . 688
7.14.29 ST_VoronoiLines . . . 691
7.14.30 ST_VoronoiPolygons . . . 692
7.15 Deckungen . . . 694
7.15.1 ST_CoverageInvalidEdges . . . 694
7.15.2 ST_CoverageClean . . . 696
7.15.3 ST_CoverageSimplify . . . 697
7.15.4 ST_CoverageEdges . . . 699
7.15.5 ST_CoverageUnion . . . 700
7.16 Affine Transformationen . . . 701
7.16.1 ST_Affine . . . 701
7.16.2 ST_Rotate . . . 704
7.16.3 ST_RotateX . . . 707
7.16.4 ST_RotateY . . . 708
7.16.5 ST_RotateZ . . . 709
7.16.6 ST_Scale . . . 711
7.16.7 ST_Translate . . . 715
7.16.8 ST_TransScale . . . 718
7.17 Clustering-Funktionen . . . 720
7.17.1 ST_ClusterDBSCAN . . . 720
7.17.2 ST_ClusterIntersecting . . . 722
7.17.3 ST_ClusterIntersectingWin . . . 723
7.17.4 ST_ClusterRelateWin . . . 725
7.17.5 ST_MinimumSpanningTree . . . 726
7.17.6 ST_ClusterKMeans . . . 727
7.17.7 ST_ClusterWithin . . . 732
7.17.8 ST_ClusterWithinWin . . . 733
7.18 Bounding Box-Funktionen . . . 735
7.18.1 Box2D . . . 735
7.18.2 Box3D . . . 737
7.18.3 ST_EstimatedExtent . . . 739
7.18.4 ST_Expand . . . 740
7.18.5 ST_Extent . . . 742
7.18.6 ST_3DExtent . . . 744

7.18.7 ST_MakeBox2D . . . 745
7.18.8 ST_3DMakeBox . . . 746
7.18.9 ST_XMax . . . 746
7.18.10 ST_XMin . . . 748
7.18.11 ST_YMax . . . 750
7.18.12 ST_YMin . . . 752
7.18.13 ST_ZMax . . . 754
7.18.14 ST_ZMin . . . 756
7.18.15 ST_XSize . . . 758
7.18.16 ST_YSize . . . 760
7.18.17 ST_ZSize . . . 761
7.18.18 ST_MMin . . . 763
7.18.19 ST_MMax . . . 763
7.18.20 ST_MSize . . . 764
7.19 Kilometrierung . . . 765
7.19.1 ST_LineInterpolatePoint . . . 765
7.19.2 ST_3DLineInterpolatePoint . . . 769
7.19.3 ST_LineInterpolatePoints . . . 770
7.19.4 ST_LineLocatePoint . . . 771
7.19.5 ST_LineSubstring . . . 774
7.19.6 ST_LocateAlong . . . 780
7.19.7 ST_LocateBetween . . . 781
7.19.8 ST_LocateBetweenElevations . . . 783
7.19.9 ST_InterpolatePoint . . . 785
7.19.10 ST_3DInterpolatePoint . . . 786
7.19.11 ST_AddMeasure . . . 788
7.20 Trajektorie-Funktionen . . . 791
7.20.1 ST_IsValidTrajectory . . . 791
7.20.2 ST_ClosestPointOfApproach . . . 792
7.20.3 ST_DistanceCPA . . . 794
7.20.4 ST_CPAWithin . . . 795
7.21 Version Funktionen . . . 796
7.21.1 PostGIS_Extensions_Upgrade . . . 796
7.21.2 PostGIS_Full_Version . . . 797
7.21.3 PostGIS_GEOS_Version . . . 798
7.21.4 PostGIS_GEOS_Compiled_Version . . . 798
7.21.5 PostGIS_Liblwgeom_Version . . . 799
7.21.6 PostGIS_LibXML_Version . . . 799
7.21.7 PostGIS_LibJSON_Version . . . 800

7.21.8 PostGIS_Lib_Build_Date . . . 801
7.21.9 PostGIS_Lib_Version . . . 801
7.21.10 PostGIS_PROJ_Version . . . 801
7.21.11 PostGIS_PROJ_Compiled_Version . . . 802
7.21.12 PostGIS_Wagyu_Version . . . 803
7.21.13 PostGIS_Scripts_Build_Date . . . 803
7.21.14 PostGIS_Scripts_Installed . . . 804
7.21.15 PostGIS_Scripts_Released . . . 804
7.21.16 PostGIS_Version . . . 805
7.22 PostGIS Grand Unified Custom Variables (GUCs) . . . 806
7.22.1 postgis.gdal_datapath . . . 806
7.22.2 postgis.gdal_enabled_drivers . . . 806
7.22.3 postgis.enable_outdb_rasters . . . 808
7.22.4 postgis.gdal_vsi_options . . . 809
7.22.5 postgis.gdal_cpl_debug . . . 810
7.23 Funktionen zur Fehlersuche . . . 810
7.23.1 PostGIS_AddBBox . . . 810
7.23.2 PostGIS_DropBBox . . . 811
7.23.3 PostGIS_HasBBox . . . 812

8 Referenz der SFCGAL-Funktionen 813
8.1 Verwaltungsfunktionen der SFCGAL . . . 813
8.1.1 postgis_sfcgal_version . . . 813
8.1.2 postgis_sfcgal_full_version . . . 813
8.2 SFCGAL-Accessoren und -Setzer . . . 814
8.2.1 CG_ForceLHR . . . 814
8.2.2 CG_IsPlanar . . . 815
8.2.3 CG_IsValid . . . 815
8.2.4 CG_IsValidDetail . . . 817
8.2.5 CG_IsSimple . . . 817
8.2.6 CG_IsSimpleDetail . . . 818
8.2.7 CG_IsSolid . . . 820
8.2.8 CG_MakeSolid . . . 821
8.2.9 CG_Orientation . . . 821
8.2.10 CG_Area . . . 822
8.2.11 CG_3DArea . . . 823
8.2.12 CG_Volume . . . 824
8.3 SFCGAL-Verarbeitung und Beziehungsfunktionen . . . 825
8.3.1 CG_Intersection . . . 825

8.3.2 CG_Intersects . . . 826
8.3.3 CG_3DIntersects . . . 828
8.3.4 CG_Difference . . . 829
8.3.5 CG_3DDifference . . . 831
8.3.6 CG_Distance . . . 832
8.3.7 CG_3DDistance . . . 833
8.3.8 CG_3DConvexHull . . . 834
8.3.9 CG_3DIntersection . . . 837
8.3.10 CG_Union . . . 840
8.3.11 CG_3DUnion . . . 841
8.3.12 CG_AlphaShape . . . 843
8.3.13 CG_ApproxConvexPartition . . . 846
8.3.14 CG_ApproximateMedialAxis . . . 847
8.3.15 CG_GenerateFlatRoof . . . 850
8.3.16 CG_GenerateHippedRoof . . . 851
8.3.17 CG_GenerateGableRoof . . . 852
8.3.18 CG_GenerateSkillionRoof . . . 853
8.3.19 CG_GenerateRoof . . . 855
8.3.20 CG_PolygonRepair . . . 856
8.3.21 CG_ConstrainedDelaunayTriangles . . . 858
8.3.22 CG_Extrude . . . 860
8.3.23 CG_ExtrudeStraightSkeleton . . . 862
8.3.24 CG_GreeneApproxConvexPartition . . . 863
8.3.25 CG_MinkowskiSum . . . 865
8.3.26 CG_OptimalAlphaShape . . . 867
8.3.27 CG_OptimalConvexPartition . . . 869
8.3.28 CG_StraightSkeleton . . . 870
8.3.29 CG_Tesselate . . . 872
8.3.30 CG_Triangulate . . . 875
8.3.31 CG_Visibility . . . 876
8.3.32 CG_YMonotonePartition . . . 878
8.3.33 CG_StraightSkeletonPartition . . . 879
8.3.34 CG_3DBuffer . . . 881
8.3.35 CG_Rotate . . . 884
8.3.36 CG_2DRotate . . . 885
8.3.37 CG_3DRotate . . . 886
8.3.38 CG_RotateX . . . 887
8.3.39 CG_RotateY . . . 888
8.3.40 CG_RotateZ . . . 889

8.3.41 CG_Scale . . . 890
8.3.42 CG_3DScale . . . 891
8.3.43 CG_3DScaleAroundCenter . . . 892
8.3.44 CG_Translate . . . 893
8.3.45 CG_3DTranslate . . . 894
8.3.46 CG_Simplify . . . 895
8.3.47 CG_3DAlphaWrapping . . . 899
8.3.48 CG_NurbsCurveInterpolate . . . 900
8.3.49 CG_NurbsCurveApproximate . . . 902
8.3.50 CG_NurbsCurveDerivative . . . 904
8.4 Deprecated SFCGAL Function Aliases . . . 908
8.4.1 ST_3DIntersection . . . 908
8.4.2 ST_3DDifference . . . 909
8.4.3 ST_3DUnion . . . 909
8.4.4 ST_Tesselate . . . 909
8.4.5 ST_3DArea . . . 910
8.4.6 ST_Extrude . . . 910
8.4.7 ST_ForceLHR . . . 911
8.4.8 ST_Orientation . . . 911
8.4.9 ST_MinkowskiSum . . . 911
8.4.10 ST_StraightSkeleton . . . 912
8.4.11 ST_ApproximateMedialAxis . . . 912
8.4.12 ST_IsPlanar . . . 913
8.4.13 ST_Volume . . . 913
8.4.14 ST_MakeSolid . . . 914
8.4.15 ST_IsSolid . . . 914
8.4.16 ST_ConstrainedDelaunayTriangles . . . 915
8.4.17 ST_3DConvexHull . . . 915
8.4.18 ST_AlphaShape . . . 915
8.4.19 ST_OptimalAlphaShape . . . 916

9 Topologie **917**
9.1 Building Faces from Linework . . . 917
9.2 Topology Primitive Tables . . . 919
9.3 Topologische Datentypen . . . 919
9.3.1 getfaceedges_returntype . . . 919
9.3.2 TopoGeometry . . . 920
9.3.3 validatetopology_returntype . . . 920
9.4 Topologische Domänen . . . 921

9.4.1 TopoElement . . . 921
9.4.2 TopoElementArray . . . 922
9.5 Verwaltung von Topologie und TopoGeometry . . . 923
9.5.1 AddTopoGeometryColumn . . . 923
9.5.2 RenameTopoGeometryColumn . . . 924
9.5.3 DropTopology . . . 924
9.5.4 RenameTopology . . . 925
9.5.5 DropTopoGeometryColumn . . . 925
9.5.6 FixCorruptTopoGeometryColumn . . . 926
9.5.7 Populate_Topology_Layer . . . 927
9.5.8 TopologySummary . . . 928
9.5.9 ValidateTopology . . . 929
9.5.10 ValidateTopologyRelation . . . 931
9.5.11 ValidateTopologyPrecision . . . 932
9.5.12 MakeTopologyPrecise . . . 933
9.5.13 FindTopology . . . 934
9.5.14 FindLayer . . . 934
9.5.15 TotalTopologySize . . . 935
9.5.16 UpgradeTopology . . . 936
9.6 Verwaltung der Topologie-Statistiken . . . 936
9.7 Topologie Konstruktoren . . . 937
9.7.1 CreateTopology . . . 937
9.7.2 CopyTopology . . . 938
9.7.3 ST_InitTopoGeo . . . 938
9.7.4 ST_CreateTopoGeo . . . 939
9.7.5 TopoGeo_AddPoint . . . 940
9.7.6 TopoGeo_AddLineString . . . 940
9.7.7 TopoGeo_AddPolygon . . . 941
9.7.8 TopoGeo_LoadGeometry . . . 942
9.8 Topologie Editoren . . . 942
9.8.1 ST_AddIsoNode . . . 942
9.8.2 ST_AddIsoEdge . . . 943
9.8.3 ST_AddEdgeNewFaces . . . 943
9.8.4 ST_AddEdgeModFace . . . 944
9.8.5 ST_RemEdgeNewFace . . . 945
9.8.6 ST_RemEdgeModFace . . . 945
9.8.7 ST_ChangeEdgeGeom . . . 946
9.8.8 ST_ModEdgeSplit . . . 947
9.8.9 ST_ModEdgeHeal . . . 948

9.8.10 ST_NewEdgeHeal . . . 948
9.8.11 ST_MoveIsoNode . . . 949
9.8.12 ST_NewEdgesSplit . . . 950
9.8.13 ST_RemoveIsoNode . . . 951
9.8.14 ST_RemoveIsoEdge . . . 951
9.9 Zugriffsfunktionen zur Topologie . . . 952
9.9.1 GetEdgeByPoint . . . 952
9.9.2 GetFaceByPoint . . . 953
9.9.3 GetFaceContainingPoint . . . 954
9.9.4 GetNodeByPoint . . . 955
9.9.5 GetTopologyID . . . 956
9.9.6 GetTopologySRID . . . 956
9.9.7 GetTopologyName . . . 957
9.9.8 ST_GetFaceEdges . . . 957
9.9.9 ST_GetFaceGeometry . . . 958
9.9.10 GetRingEdges . . . 959
9.9.11 GetNodeEdges . . . 960
9.9.12 FindVertexSegmentPairsBelowDistance . . . 960
9.10 Topologie Verarbeitung . . . 961
9.10.1 Polygonize . . . 961
9.10.2 AddNode . . . 961
9.10.3 AddEdge . . . 962
9.10.4 AddFace . . . 964
9.10.5 ST_Simplify . . . 965
9.10.6 RemoveUnusedPrimitives . . . 966
9.11 TopoGeometry Konstruktoren . . . 966
9.11.1 CreateTopoGeom . . . 966
9.11.2 toTopoGeom . . . 969
9.11.3 TopoElementArray_Agg . . . 971
9.11.4 TopoElement . . . 971
9.12 TopoGeometry Editoren . . . 972
9.12.1 clearTopoGeom . . . 972
9.12.2 TopoGeom_addElement . . . 972
9.12.3 TopoGeom_remElement . . . 973
9.12.4 TopoGeom_addTopoGeom . . . 973
9.12.5 toTopoGeom . . . 974
9.13 TopoGeometry Accessors . . . 974
9.13.1 GetTopoGeomElementArray . . . 974
9.13.2 GetTopoGeomElements . . . 975

9.13.3 ST_SRID . . . 975
9.14 TopoGeometry Ausgabe . . . 976
9.14.1 AsGML . . . 976
9.14.2 AsTopoJSON . . . 978
9.15 Räumliche Beziehungen einer Topologie . . . 980
9.15.1 Equals . . . 980
9.15.2 Intersects . . . 981
9.16 Importieren und Exportieren von Topologien . . . 982
9.16.1 Verwendung des Topologie-Exporters . . . 982
9.16.2 Verwendung des Topologie-Importers . . . 982

10 Rasterdatenverwaltung, -abfrage und Anwendungen 984
10.1 Laden und Erstellen von Rastertabellen . . . 984
10.1.1 Verwendung von raster2pgsql zum Laden von Rastern . . . 984
10.1.1.1 Beispiel für die Verwendung . . . 984
10.1.1.2 raster2pgsql-Optionen . . . 985
10.1.2 Erzeugung von Rastern mit den PostGIS Rasterfunktionen . . . 987
10.1.3 Verwendung von "out db"-Wolkenrastern . . . 988
10.2 Raster Coverages and Raster/Vector Analysis . . . 989
10.3 Raster Katalog . . . 990
10.3.1 Rasterspalten Katalog . . . 990
10.3.2 Raster Übersicht/Raster Overviews . . . 991
10.4 Eigene Anwendungen mit PostGIS Raster erstellen . . . 992
10.4.1 Rendering Map Algebra and Reclass Results . . . 992
10.4.2 PL/pgSQL Map Algebra Callback Example . . . 992
10.4.3 PHP Beispiel: Ausgabe mittels ST_AsPNG in Verbindung mit anderen Rasterfunktionen . . . 993
10.4.4 ASP.NET C# Beispiel: Ausgabe mittels ST_AsPNG in Verbindung mit anderen Rasterfunktionen . . . 994
10.4.5 Applikation für die Java-Konsole, welche eine Rasterabfrage als Bilddatei ausgibt . . . 996
10.4.6 Verwenden Sie PLPython um Bilder via SQL herauszuschreiben . . . 997
10.4.7 Faster mit PSQL ausgeben . . . 998

11 Referenz Raster 999
11.1 Datentypen zur Unterstützung von Rastern. . . . 1000
11.1.1 geomval . . . 1000
11.1.2 addbandarg . . . 1000
11.1.3 rastbandarg . . . 1000
11.1.4 raster . . . 1001
11.1.5 reclassarg . . . 1001
11.1.6 summarystats . . . 1002

11.1.7 unionarg . . . 1003
11.2 Rastermanagement . . . 1003
11.2.1 AddRasterConstraints . . . 1003
11.2.2 DropRasterConstraints . . . 1005
11.2.3 AddOverviewConstraints . . . 1006
11.2.4 DropOverviewConstraints . . . 1007
11.2.5 PostGIS_GDAL_Version . . . 1008
11.2.6 PostGIS_Raster_Lib_Build_Date . . . 1008
11.2.7 PostGIS_Raster_Lib_Version . . . 1009
11.2.8 ST_GDALDrivers . . . 1009
11.2.9 UpdateRasterSRID . . . 1012
11.2.10 ST_CreateOverview . . . 1013
11.3 Raster Constructors . . . 1014
11.3.1 ST_AddBand . . . 1014
11.3.2 ST_AsRaster . . . 1016
11.3.3 ST_AsRasterAgg . . . 1019
11.3.4 ST_Band . . . 1020
11.3.5 ST_MakeEmptyCoverage . . . 1024
11.3.6 ST_MakeEmptyRaster . . . 1025
11.3.7 ST_Tile . . . 1027
11.3.8 ST_Retile . . . 1029
11.3.9 ST_FromGDALRaster . . . 1029
11.4 Zugriffsfunktionen auf Raster . . . 1030
11.4.1 ST_GeoReference . . . 1030
11.4.2 ST_Height . . . 1031
11.4.3 ST_IsEmpty . . . 1032
11.4.4 ST_MemSize . . . 1033
11.4.5 ST_MetaData . . . 1033
11.4.6 ST_NumBands . . . 1034
11.4.7 ST_PixelHeight . . . 1035
11.4.8 ST_PixelWidth . . . 1036
11.4.9 ST_ScaleX . . . 1038
11.4.10 ST_ScaleY . . . 1038
11.4.11 ST_RasterToWorldCoord . . . 1039
11.4.12 ST_RasterToWorldCoordX . . . 1040
11.4.13 ST_RasterToWorldCoordY . . . 1041
11.4.14 ST_Rotation . . . 1042
11.4.15 ST_SkewX . . . 1043
11.4.16 ST_SkewY . . . 1044

11.4.17 ST_SRID . . . 1045
11.4.18 ST_Summary . . . 1046
11.4.19 ST_UpperLeftX . . . 1046
11.4.20 ST_UpperLeftY . . . 1047
11.4.21 ST_Width . . . 1047
11.4.22 ST_WorldToRasterCoord . . . 1048
11.4.23 ST_WorldToRasterCoordX . . . 1049
11.4.24 ST_WorldToRasterCoordY . . . 1049
11.5 Zugriffsfunktionen auf Rasterbänder . . . 1050
11.5.1 ST_BandMetaData . . . 1050
11.5.2 ST_BandNoDataValue . . . 1052
11.5.3 ST_BandIsNoData . . . 1052
11.5.4 ST_BandPath . . . 1054
11.5.5 ST_BandFileSize . . . 1055
11.5.6 ST_BandFileTimestamp . . . 1055
11.5.7 ST_BandPixelType . . . 1056
11.5.8 ST_MinPossibleValue . . . 1057
11.5.9 ST_HasNoBand . . . 1057
11.6 Zugriffsfunktionen und Änderungsmethoden für Rasterpixel . . . 1058
11.6.1 ST_PixelAsPolygon . . . 1058
11.6.2 ST_PixelAsPolygons . . . 1059
11.6.3 ST_PixelAsPoint . . . 1060
11.6.4 ST_PixelAsPoints . . . 1061
11.6.5 ST_PixelAsCentroid . . . 1063
11.6.6 ST_PixelAsCentroids . . . 1064
11.6.7 ST_Value . . . 1066
11.6.8 ST_NearestValue . . . 1069
11.6.9 ST_SetZ . . . 1071
11.6.10 ST_SetM . . . 1072
11.6.11 ST_Neighborhood . . . 1074
11.6.12 ST_SetValue . . . 1076
11.6.13 ST_SetValues . . . 1077
11.6.14 ST_DumpValues . . . 1085
11.6.15 ST_PixelOfValue . . . 1086
11.7 Raster Editoren . . . 1087
11.7.1 ST_SetGeoReference . . . 1087
11.7.2 ST_SetRotation . . . 1089
11.7.3 ST_SetScale . . . 1089
11.7.4 ST_SetSkew . . . 1090

11.7.5 ST_SetSRID . . . 1092
11.7.6 ST_SetUpperLeft . . . 1093
11.7.7 ST_Resample . . . 1093
11.7.8 ST_Rescale . . . 1094
11.7.9 ST_Reskew . . . 1096
11.7.10 ST_SnapToGrid . . . 1097
11.7.11 ST_Resize . . . 1099
11.7.12 ST_Transform . . . 1100
11.8 Editoren für Rasterbänder . . . 1104
11.8.1 ST_SetBandNoDataValue . . . 1104
11.8.2 ST_SetBandIsNoData . . . 1105
11.8.3 ST_SetBandPath . . . 1107
11.8.4 ST_SetBandIndex . . . 1108
11.9 Rasterband Statistik und Analytik . . . 1109
11.9.1 ST_Count . . . 1109
11.9.2 ST_CountAgg . . . 1110
11.9.3 ST_Histogram . . . 1111
11.9.4 ST_Quantile . . . 1113
11.9.5 ST_SummaryStats . . . 1115
11.9.6 ST_SummaryStatsAgg . . . 1117
11.9.7 ST_ValueCount . . . 1119
11.10 Rastereingabe . . . 1121
11.10.1 ST_RastFromWKB . . . 1121
11.10.2 ST_RastFromHexWKB . . . 1122
11.11 Ausgabe von Rastern . . . 1123
11.11.1 ST_AsBinary/ST_AsWKB . . . 1123
11.11.2 ST_AsHexWKB . . . 1124
11.11.3 ST_AsGDALRaster . . . 1124
11.11.4 ST_AsJPEG . . . 1126
11.11.5 ST_AsPNG . . . 1127
11.11.6 ST_AsTIFF . . . 1128
11.12 Rasterverarbeitung: Kartenalgebra . . . 1129
11.12.1 ST_Clip . . . 1129
11.12.2 ST_ColorMap . . . 1134
11.12.3 ST_Grayscale . . . 1137
11.12.4 ST_Intersection . . . 1141
11.12.5 ST_MapAlgebra (callback function version) . . . 1144
11.12.6 ST_MapAlgebra (expression version) . . . 1151
11.12.7 ST_MapAlgebraExpr . . . 1153

11.12.8 ST_MapAlgebraExpr . . . 1157
11.12.9 ST_MapAlgebraFct . . . 1163
11.12.10 ST_MapAlgebraFct . . . 1168
11.12.11 ST_MapAlgebraFctNgb . . . 1174
11.12.12 ST_Reclass . . . 1178
11.12.13 ST_ReclassExact . . . 1180
11.12.14 ST_Union . . . 1181
11.13 Integrierte Map Algebra Callback Funktionen . . . 1183
11.13.1 ST_Distinct4ma . . . 1183
11.13.2 ST_InvDistWeight4ma . . . 1184
11.13.3 ST_Max4ma . . . 1185
11.13.4 ST_Mean4ma . . . 1186
11.13.5 ST_Min4ma . . . 1187
11.13.6 ST_MinDist4ma . . . 1188
11.13.7 ST_Range4ma . . . 1190
11.13.8 ST_StdDev4ma . . . 1191
11.13.9 ST_Sum4ma . . . 1192
11.14 Rasterverarbeitung: DEM (Elevation) . . . 1193
11.14.1 ST_Aspect . . . 1193
11.14.2 ST_HillShade . . . 1196
11.14.3 ST_Roughness . . . 1197
11.14.4 ST_Slope . . . 1198
11.14.5 ST_TPI . . . 1200
11.14.6 ST_TRI . . . 1201
11.14.7 ST_InterpolateRaster . . . 1202
11.14.8 ST_Contour . . . 1203
11.15 Rasterverarbeitung: Raster zu Geometrie . . . 1204
11.15.1 Box3D . . . 1204
11.15.2 ST_ConvexHull . . . 1205
11.15.3 ST_DumpAsPolygons . . . 1206
11.15.4 ST_Envelope . . . 1207
11.15.5 ST_MinConvexHull . . . 1207
11.15.6 ST_Polygon . . . 1209
11.15.7 ST_IntersectionFractions . . . 1211
11.16 Rasteroperatoren . . . 1212
11.16.1 && . . . 1212
11.16.2 &< . . . 1212
11.16.3 &> . . . 1213
11.16.4 = . . . 1214

11.16.5 @ . . . 1214
11.16.6 ~= . . . 1215
11.16.7 ~ . . . 1215
11.17 Räumliche Beziehungen von Rastern und Rasterbändern . . . 1216
11.17.1 ST_Contains . . . 1216
11.17.2 ST_ContainsProperly . . . 1217
11.17.3 ST_Covers . . . 1218
11.17.4 ST_CoveredBy . . . 1219
11.17.5 ST_Disjoint . . . 1220
11.17.6 ST_Intersects . . . 1221
11.17.7 ST_Overlaps . . . 1222
11.17.8 ST_Touches . . . 1223
11.17.9 ST_SameAlignment . . . 1224
11.17.10 ST_NotSameAlignmentReason . . . 1225
11.17.11 ST_Within . . . 1225
11.17.12 ST_DWithin . . . 1226
11.17.13 ST_DFullyWithin . . . 1227
11.18 Raster Tipps . . . 1228
11.18.1 Out-DB Raster . . . 1228
11.18.1.1 Das Verzeichnis enthält eine Vielzahl an Dateien . . . 1228
11.18.1.2 Die maximale Anzahl geöffneter Dateien . . . 1229
11.18.1.2.1 Die maximale Anzahl geöffneter Dateien für das ganze System . . . 1229
11.18.1.2.2 Die maximale Anzahl geöffneter Dateien pro Prozess . . . 1230

12 PostGIS Extras **1232**

13 PostGIS Spezialfunktionen Index **1233**
13.1 PostGIS-Aggregat-Funktionen . . . 1233
13.2 PostGIS-Fenster-Funktionen . . . 1234
13.3 PostGIS SQL-MM-kompatible Funktionen . . . 1234
13.4 PostGIS-Funktionen zur Unterstützung der Geografie . . . 1238
13.5 PostGIS-Funktionen zur Unterstützung von Rastern . . . 1240
13.6 PostGIS Geometrie / Geographie / Raster Dump Funktionen . . . 1246
13.7 PostGIS-Box-Funktionen . . . 1246
13.8 PostGIS-Funktionen, die 3D unterstützen . . . 1247
13.9 PostGIS Funktionen zur Unterstützung gekrümmter Geometrien . . . 1253
13.10 PostGIS-Funktionen zur Unterstützung polyedrischer Flächen . . . 1257
13.11 PostGIS Funktionsunterstützungsmatrix . . . 1260
13.12 Neue, erweiterte oder geänderte PostGIS-Funktionen . . . 1279

13.12.1 PostGIS-Funktionen neu oder erweitert in 3.7 . . . 1279
13.12.2 PostGIS-Funktionen neu oder erweitert in 3.6 . . . 1281
13.12.3 PostGIS-Funktionen neu oder erweitert in 3.5 . . . 1282
13.12.4 PostGIS-Funktionen neu oder erweitert in 3.4 . . . 1284
13.12.5 PostGIS-Funktionen neu oder erweitert in 3.3 . . . 1285
13.12.6 PostGIS-Funktionen neu oder erweitert in 3.2 . . . 1286
13.12.7 PostGIS-Funktionen neu oder erweitert in 3.1 . . . 1288
13.12.8 PostGIS-Funktionen neu oder erweitert in 3.0 . . . 1289
13.12.9 PostGIS-Funktionen neu oder erweitert in 2.5 . . . 1291
13.12.10 PostGIS-Funktionen neu oder erweitert in 2.4 . . . 1292
13.12.11 PostGIS-Funktionen neu oder erweitert in 2.3 . . . 1293
13.12.12 PostGIS-Funktionen neu oder erweitert in 2.2 . . . 1295
13.12.13 PostGIS-Funktionen neu oder erweitert in 2.1 . . . 1298
13.12.14 PostGIS-Funktionen neu oder erweitert in 2.0 . . . 1303
13.12.15 PostGIS-Funktionen neu oder erweitert in 1.5 . . . 1313
13.12.16 PostGIS-Funktionen neu oder erweitert in 1.4 . . . 1315
13.12.17 PostGIS-Funktionen neu oder erweitert in 1.3 . . . 1316

14 Meldung von Problemen **1317**

14.1 Software Bugs melden . . . 1317
14.2 Probleme mit der Dokumentation melden . . . 1317

15 Credits and Acknowledgments **1319**

15.1 Project Steering Committee . . . 1319
15.2 Core Contributors Present . . . 1319
15.3 Core Contributors Past . . . 1320
15.4 PostGIS Raster Origins . . . 1320
15.5 Other Contributors . . . 1321

A Anmerkungen zur Release **1325**

A.1 PostGIS 3.7.0 . . . 1325
A.1.1 Wechselnde Änderungen . . . 1325
A.1.2 New Features . . . 1325
A.1.3 Erweiterungen . . . 1326
A.1.4 Bug Fixes . . . 1327

Zusammenfassung

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





Dieses Handbuch beschreibt die Version 3.8.0dev



Diese Arbeit ist unter der Creative Commons Attribution-Share Alike 3.0 License lizensiert. Sie können den Inhalt ungeniert nutzen, aber wir ersuchen Sie das PostGIS Projekt namentlich aufzuführen und wenn möglich einen Verweis auf https://postgis.net zu setzen.

Kapitel 1

Einführung

1.1 What Is PostGIS?

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

The main `postgis` extension adds:

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

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

1.2 What Is in the PostGIS Distribution?

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

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

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

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

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

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

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

1.3 The PostGIS Project Family

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

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

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

1.4 How to Use This Manual

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

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

If you are installing or upgrading PostGIS Read Kapitel 2 and Kapitel 3. Package-specific instructions and supported release information are maintained on the PostGIS website.

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

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

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

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

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

1.5 Standards Support

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

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

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

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

1.6 Project and Community

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

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

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

Kapitel 2

PostGIS Installation

Dieses Kapitel erläutert die notwendigen Schritte zur Installation von PostGIS.

2.1 Kurzfassung

Zum Kompilieren müssen die Abhängigkeiten im Suchpfad eingetragen sein:

Code

```
tar -xvzf postgis-3.8.0dev.tar.gz
cd postgis-3.8.0dev
./configure
make
make install
```

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

2.2 Kompilierung und Installation des Quellcodes: Detaillierte Beschreibung



Anmerkung
Viele Betriebssysteme stellen heute bereits vorkompilierte Pakete für PostgreSQL/PostGIS zur Verfügung. Somit ist eine Kompilation nur notwendig, wenn man die aktuellsten Versionen benötigt oder für die Paketverwaltung zuständig ist.
This section includes general compilation instructions. If you are compiling for Windows or another operating system, see the PostGIS getting started guides and the development environment guides.
Pre-Built Packages for various OS are listed in the PostGIS getting started guides
Wenn Sie ein Windowsbenutzer sind, können Sie stabile Kompilationen mittels Stackbuilder oder die PostGIS Windows download site erhalten. Es gibt auch very bleeding-edge windows experimental builds, die ein oder zweimal pro Woche, bzw. anlassweise kompiliert werden. Damit können Sie mit im Aufbau befindlichen PostGIS Releases experimentieren.

Das PostGIS-Modul ist eine Erweiterung für den PostgreSQL-Backend-Server. Als solches benötigt PostGIS 3.8.0dev *vollen Zugriff auf die Header des PostgreSQL-Servers*, um kompiliert werden zu können. Es kann mit den PostgreSQL-Versionen 14 - 19 kompiliert werden. Frühere Versionen von PostgreSQL werden *nicht unterstützt*.

Lesen Sie die PostgreSQL-Installationsanleitung, wenn Sie PostgreSQL noch nicht installiert haben. https://www.postgresql.org .



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

Code

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

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

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

2.2.1 Nutzung des Quellcodes

Das PostGIS Quellarchiv kann von der Download Webseite https://postgis.net/stuff/postgis-3.8.0dev.tar.gz bezogen werden.

Code

```
wget https://postgis.net/stuff/postgis-3.8.0dev.tar.gz
tar -xvzf postgis-3.8.0dev.tar.gz
cd postgis-3.8.0dev
```

Dadurch wird das Verzeichnis `postgis-3.8.0dev` im aktuellen Arbeitsverzeichnis erzeugt.

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

Code

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

Um die Installation fortzusetzen ist in das neu erstellte Verzeichnis `postgis-3.8.0dev` zu wechseln.

Code

```
./configure
```

2.2.2 Systemvoraussetzungen

Zur Kompilation und Anwendung stellt PostGIS die folgenden Systemanforderungen:

Notwendige Systemvoraussetzungen

- PostgreSQL 14 - 19. A complete installation of PostgreSQL (including server headers) is required. PostgreSQL is available from https://www.postgresql.org 18 .

  For current release-line support and end-of-life policy, see https://postgis.net/development/versions_eol/

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

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

- Proj Reprojektion Bibliothek. Proj 6.1 oder höher ist erforderlich. Die Proj-Bibliothek wird zur Unterstützung der Koordinatenreprojektion in PostGIS verwendet. Proj steht zum Download zur Verfügung unter https://proj.org/ .
- GEOS geometry library, version 3.10.0 or greater, but GEOS 3.15+ is required to take full advantage of all the new functions and features. GEOS is available for download from https://libgeos.org .
- LibXML2. LibXML2 is currently used in some imports functions (ST_GeomFromGML and ST_GeomFromKML). LibXML2 is available for download from https://gitlab.gnome.org/GNOME/libxml2/-/releases.
- Perl is required to build PostGIS. Perl is available for download from https://www.perl.org/.
- The iconv library is required by the shp2pgsql loader for encoding conversion. On GNU/Linux systems it is typically provided by libc; on other platforms you may need a separate libiconv package.
- The GNU Multiple Precision Arithmetic Library (libgmp). Used to control precision of math operations in PostGIS Topology. GMP is available for download from https://gmplib.org/. It is detected via **pkg-config** and has no fallback, so **pkg-config** must be available for a default build.
- GDAL, version 2.4 or higher, is required for raster support. GDAL 3+ is preferred. https://gdal.org/download.html.
- Wenn mit PostgreSQL+JIT kompiliert wird, ist die LLVM-Version >=6 erforderlich https://trac.osgeo.org/postgis/ticket/4125.

Optionale Systemanforderungen

- GDAL (pseudo optional) nur wenn Sie kein Rasterunterstützung möchten, können Sie es weglassen. Sorgen Sie außerdem dafür das Treiber, die Sie brauchen wie in Abschnitt 3.2 beschrieben, aktiviert sind.
- JSON-C. JSON-C is currently used to import GeoJSON via the function ST_GeomFromGeoJson. JSON-C is available for download from https://github.com/json-c/json-c/releases/. It is enabled by default but optional; if it is not found configure only warns and the build continues. Disable with `--without-json`.
- GNU gettext for loader and documentation translation support. It may be disabled with `--with-gettext=no`. https://www.gnu.org/software/gettext/.
- GTK (benötigt GTK+2.0, 2.8+) um den "shp2pgsql-gui shape file loader" zu kompilieren. http://www.gtk.org/ .
- SFCGAL, 1.3.1 or higher is required, 2.3+ is needed to be able to use all functionality. SFCGAL can be used to provide additional 2D and 3D advanced analysis functions to PostGIS cf Kapitel 8. https://sfcgal.gitlab.io/sfcgal/.
- Um ST_AsMVT verwenden zu können, wird die protobuf-c Bibliothek (für die Anwendung) und der protoc-c Kompiler (für die Kompilation) benötigt. Weiters ist pgk-config erforderlich um die korrekte Minimumversion von protobuf-c zu bestimmen. Siehe protobuf-c.
- CUnit (`CUnit`). Wird für Regressionstest benötigt. http://cunit.sourceforge.net/
- DocBook (`xsltproc`) ist für die Kompilation der Dokumentation notwendig. Docbook steht unter http://www.docbook.org/ zur Verfügung.
- Python 3 is required, when configured with **--with-visual-examples** (off by default), to run the manual example tester that generates the visual examples which the HTML and PDF documentation embed. The examples are executed against a live PostgreSQL server running the staged PostGIS build via **make -C regress visual-examples**; no Python database driver is required, the tooling talks to the server through the **psql** client. Python 3 is available from https://www.python.org/ .
- DBLatex (`dblatex`) ist zur Kompilation der Dokumentation im PDF-Format nötig. DBLatex liegt unter http://dblatex.sourceforge.net vor.
- PDF documentation needs an SVG rasterizer for its high-resolution figure fallbacks. The build prefers `rsvg-convert`, and can also use GraphicsMagick (`gm convert`) or ImageMagick (`magick` or `convert`).

2.2.3 Konfiguration

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

Code

```
./configure
```

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

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

--with-library-minor-version Ab PostGIS 3.0 werden die standardmäßig erzeugten Bibliotheksdateien nicht mehr die Nebenversion als Teil des Dateinamens haben. Das bedeutet, dass alle PostGIS 3 Bibliotheken auf `postgis-3` enden. Dies wurde gemacht, um pg_upgrade einfacher zu machen, mit dem Nachteil, dass Sie nur eine Version der PostGIS 3 Serie auf Ihrem Server installieren können. Um das alte Verhalten der Datei einschließlich der Nebenversion zu erhalten: z.B. `postgis-3.0` fügen Sie diesen Schalter zu Ihrer configure-Anweisung hinzu.

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



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

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

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

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

--with-xml2config=FILE LibXML ist die Bibliothek, die für die Durchführung von GeomFromKML/GML-Prozessen erforderlich ist. Normalerweise wird sie gefunden, wenn Sie libxml installiert haben, aber wenn nicht oder Sie eine bestimmte Version verwenden wollen, müssen Sie PostGIS auf eine bestimmte `xml2-config` confi-Datei verweisen, damit Softwareinstallationen das LibXML-Installationsverzeichnis finden können. Verwenden Sie diesen Parameter (**>--with-xml2config=/path/to/xml2-config**), um manuell eine bestimmte LibXML-Installation anzugeben, gegen die PostGIS gebaut werden soll.

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

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

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

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

--without-raster Ohne Rasterunterstützung kompilieren.

--without-topology Compile without topology support.

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

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

--without-phony-revision Deaktivieren Sie die Aktualisierung von postgis_revision.h, um es an den aktuellen HEAD des Git-Repository anzupassen.



Anmerkung
If you obtained PostGIS from the code repository , the first step is really to run the script

Code

```
./autogen.sh
```

Dieses Skript erzeugt das **configure** Skript, welches seinerseits zur Anpassung der Installation von PostGIS eingesetzt wird.

Falls Sie stattdessen PostGIS als Tarball vorliegen haben, dann ist es nicht notwendig **./autogen.sh** auszuführen, da **configure** bereits erzeugt wurde.

2.2.4 Build-Prozess

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

Code

```
make
```

Output

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

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 comment files 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.

Code

```
make comments
```

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

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

Code

```
make cheatsheets
```

2.2.5 Build-Prozess für die PostGIS Extensions und deren Bereitstellung

The PostGIS extensions are built and installed automatically when PostgreSQL extension support is available.

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

Code

```
make comments
```

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

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

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

Code

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



Anmerkung
`make check` benutzt psql um Tests auszuführen und kann daher psql Umgebungsvariablen benutzen. Gängige Variablen, die man überschreiben kann, sind `PGUSER`,`PGPORT` und `PGHOST`. Siehe psql Umgebungsvariablen

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

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

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

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

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

Code

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

Output

```
+-----------------+-----------------+-------------------+
| name            | default_version | installed_version |
+-----------------+-----------------+-------------------+
| postgis         | 3.8.0dev        | 3.8.0dev          |
| postgis_raster  | 3.8.0dev        | 3.8.0dev          |
| postgis_sfcgal  | 3.8.0dev        |                   |
| postgis_topology | 3.8.0dev        |                   |
+-----------------+-----------------+-------------------+
```

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

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

Code

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

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

Code

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

Output

```
List of installed extensions
+-[ RECORD 1 ]-------------------------------------------+
| Name        | postgis                                  |
| Version     | 3.8.0dev                                   |
| Schema      | public                                   |
| Description | PostGIS geometry, geography, and raster...  |
+-[ RECORD 2 ]-------------------------------------------+
| Name        | postgis_raster                           |
| Version     | 3.8.0dev                                   |
| Schema      | public                                   |
| Description | PostGIS raster types and functions       |
+-[ RECORD 3 ]-------------------------------------------+
| Name        | postgis_topology                         |
| Version     | 3.8.0dev                                   |
| Schema      | topology                                 |
| Description | PostGIS topology spatial types and functions |
+-------------------------------------------------------+
```



Warnung

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 happens when you back up the whole database. Only srid records not packaged with PostGIS are captured in backups, so don't change the entries we ship and expect the modifications to persist. 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.

Wenn Sie 3.8.0dev ohne unser wunderbares Extension System installiert haben, können Sie auf erweiterungsbasiert wechseln, indem Sie folgende Befehle ausführen, welche die Funktionen in ihre entsprechenden Erweiterungen paketieren.

Code

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

2.2.6 Softwaretest

Wenn Sie die Kompilation von PostGIS überprüfen wollen:

Code

```
make check
```

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



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



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



Anmerkung
Sandboxed build accounts that are not PostgreSQL superusers can delegate database ownership during the regression cycle by exporting `POSTGIS_REGRESS_DB_OWNER`. The harness will create the temporary regression database owned by the nominated role while continuing to connect using the less privileged account. Combine this with `POSTGIS_REGRESS_ROLE_EXT_CREATOR` when the extension creation role must differ from the database owner. These variables allow automated environments to exercise the full upgrade and extension install paths without promoting the calling account to superuser, provided the target PostgreSQL instance permits extension installation by those delegate roles.

Wenn make check erfolgreich ist, wird die Ausgabe von fast 500 Tests erzeugt. Die Ergebnisse sehen ähnlich aus wie die folgenden (zahlreiche Zeilen wurden weggelassen):

Code

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

  .
  .
  .

Run Summary:    Type  Total    Ran Passed Failed Inactive
              suites     44     44    n/a      0        0
```

```
               tests    300    300    300      0        0
             asserts   4215   4215   4215      0      n/a
Elapsed time =    0.229 seconds

  .
  .
  .

Running tests

  .
  .
  .

Run tests: 134
Failed: 0


-- if you build with SFCGAL

  .
  .
  .

Running tests

  .
  .
  .

Run tests: 13
Failed: 0

-- if you built with raster support

  .
  .
  .

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



  .
  .
  .

Running tests

  .
  .
  .

Run tests: 101
Failed: 0

-- topology regress

.
.
```

```
.

Running tests

  .
  .
  .

Run tests: 51
Failed: 0

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

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

  .
  .
  .

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

Die Ausgabe sollte folgendermaßen aussehen:

Output

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

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

2.2.7 Installation

Um PostGIS zu installieren geben Sie bitte folgendes ein

Code

```
make install
```

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

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

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

Code

```
make comments-install
```



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

2.3 Übliche Probleme bei der Installation

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

PostgreSQL version mismatch.

Überprüfen Sie, ob Sie PostgreSQL 14 oder neuer installiert haben und dass die Version des PostgreSQL Quellcodes, gegen den Sie kompilieren, mit der Version der laufenden PostgreSQL Datenbank übereinstimmt. Ein Wirrwarr kann dann entstehen, wenn die Linux Distribution bereits PostgreSQL installiert hat, oder wenn Sie PostgreSQL in einem anderen Zusammenhang installiert und darauf vergessen haben. PostGIS funktioniert nur mit PostgreSQL 14 oder jünger und es kommt zu merkwürdigen, unerwarteten Fehlermeldungen, wenn Sie eine ältere Version verwenden. Um die Version Ihrer laufenden PostgreSQL Datenbank zu überprüfen, können Sie sich mittels psql zur Datenbank verbinden und folgende Anfrage ausführen:

Code

```
SELECT version();
```

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

Code

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

Restoring an upgrade into a database without PostGIS.

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

Code

```
SELECT PostGIS_Full_Version();
```

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

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

Kapitel 3

PostGIS Verwaltung

3.1 Leistungsoptimierung

Das Tuning für die PostGIS-Leistung ist ähnlich wie das Tuning für jede PostgreSQL-Arbeitslast. Die einzige zusätzliche Überlegung ist, dass Geometrien und Raster in der Regel groß sind, so dass speicherbezogene Optimierungen im Allgemeinen einen größeren Einfluss auf PostGIS haben als andere Arten von PostgreSQL-Abfragen.

Allgemeine Informationen zur Optimierung von PostgreSQL finden Sie unter Tuning your PostgreSQL Server.

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

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

Code

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

Zusätzlich zu den Postgres-Einstellungen verfügt PostGIS über einige benutzerdefinierte Einstellungen, die unter Abschnitt 7.22 aufgeführt sind.

3.1.1 Startup

Diese Einstellungen werden in `postgresql.conf` konfiguriert:

constraint_exclusion

- Standard: Partition
- This is generally used for table partitioning. The default of "partition" forces the planner to only analyze tables for constraint consideration if they are in an inherited hierarchy and avoids paying the planner penalty otherwise.

shared_buffers

- Default: ~128MB
- Setzen Sie den Wert auf etwa 25 % bis 40 % des verfügbaren RAM. Unter Windows können Sie diesen Wert möglicherweise nicht so hoch einstellen.

max_worker_processes controls the maximum number of background processes available to the server and therefore also bounds how many workers can participate in parallel queries.

- Standardwert: 8
- Setzt die maximale Anzahl an Hintergrundprozessen, die von dem System unterstützt werden. Dieser Parameter kann nur durch einen Serverstart gesetzt werden.

3.1.2 Laufzeit

work_mem - setzt die Speichergröße für Sortiervorgänge und komplexe Abfragen

- Standardwert: 1-4MB
- Bitte passen Sie bei großen Datenbanken und für komplexe Abfragen mit entsprechend viel RAM an
- Bitte verringern Sie den Wert bei vielen gleichzeitigen Benutzern oder wenn nur wenig RAM vorhanden ist.
- Wenn Sie viel RAM und wenig Entwickler haben:

Code

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

maintenance_work_mem - die Speichergröße, welche für VACUUM, CREATE INDEX, etc. verwendet wird.

- Standardwert: 16-64MB
- Im Allgemeinen zu niedrig - bindet E/A, sperrt Objekte beim Auslagern von Speicher
- Wir empfehlen 32 MB bis 1 GB auf Produktionsservern mit viel RAM, aber das hängt von der Anzahl der gleichzeitigen Benutzer ab. Wenn Sie viel RAM und wenige Entwickler haben:

Code

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

max_parallel_Arbeiter_pro_Gruppe

Parallel query support in modern PostgreSQL and PostGIS uses this setting to decide how many workers a plan may request. 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.

- Voreinstellung: 0
- Legt die maximale Anzahl von Workern fest, die von einem einzelnen `Gather` Knoten gestartet werden können. Parallele Worker werden aus dem durch `max_worker_processes` festgelegten Pool von Prozessen genommen. Beachten Sie, dass die angeforderte Anzahl von Workern zur Laufzeit möglicherweise nicht verfügbar ist. Wenn dies der Fall ist, wird der Plan mit weniger Arbeitern als erwartet ausgeführt, was ineffizient sein kann. Wenn Sie diesen Wert auf 0 setzen, was der Standardwert ist, wird die parallele Abfrageausführung deaktiviert.

3.2 Konfigurieren der Rasterunterstützung

Wenn die Rasterunterstützung aktiviert ist, sollte diese wie folgt konfiguriert werden.

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. You can also use the more cross-platform approach of setting the corresponding Abschnitt 7.22.

Wenn Sie Offline-Raster aktivieren möchten:

Code

```
POSTGIS_ENABLE_OUTDB_RASTERS=1
```

Jede andere und auch keine Einstellung deaktiviert die out-of-db Raster.

Um alle installierten GDAL-Treiber zu aktivieren, muss diese Umgebungsvariable wie folgt gesetzt werden

Code

```
POSTGIS_GDAL_ENABLED_DRIVERS=ENABLE_ALL
```

Wenn nur bestimmte Treiber aktiviert werden sollen, muss diese Umgebungsvariable wie folgt gesetzt werden:

Code

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



Anmerkung
Unter Windows darf die Treiberliste nicht unter Hochkomma gestellt werden

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/MAJOR/CLUSTER/environment` where the placeholders refer to the PostgreSQL major version and cluster name.

Wenn Sie PostgreSQL unter Windows als Dienst ausführen, können sie die Systemvariablen unter Windows 7 mit einem Rechtsklick auf Computer->roperties Advanced System Settings oder im Explorer unter `Control Panel\All Control Panel Items\System`. setzen. Klicken Sie anschließend auf *Advanced System Settings ->Advanced->Environment Variables* um neue Systemvariablen hinzuzufügen.

Nachdem die Umgebungsvariablen gesetzt wurden, muss der PostgreSQL-Dienst neu gestartet werden, damit die Änderungen wirksam werden.

3.3 Erstellung räumlicher Datenbanken

3.3.1 Die spatiale Datenbankerweiterung mittels EXTENSION aktivieren

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

Die Postgis-Erweiterung selbst enthält die Module geometry, geography, spatial_ref_sys inklusive aller Funktionen und Kommentare. Die Module raster und topology sind in separate Erweiterungen verpackt.

Um die spatiale Erweiterung in einer Datenbank zu aktivieren, wird der folgende SQL Befehl ausgeführt:

Code

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

3.3.2 Räumliche Aktivierung der Datenbank ohne Verwendung von EXTENSION (nicht empfohlen)



Anmerkung
Dies ist in der Regel nur dann notwendig, wenn PostGIS nicht in das Erweiterungsverzeichnis von PostgreSQL installiert werden kann oder soll (wie in einer Test-, Entwicklungs- oder eingeschränkten Umgebung).

Das Hinzufügen von PostGIS-Objekten und Funktionsdefinitionen in Ihre Datenbank erfolgt durch das Laden unterschiedlicher SQL-Dateien unter `[prefix]/share/contrib`, wie während der Erstellungsphase angezeigt.

Die zentralen PostGIS-Objekte (Geometrie- und Geografietypen sowie deren Unterstützungsfunktionen) befinden sich im Skript `postgis.sql`. Rasterobjekte sind im Skript `rtpostgis.sql`. Topologieobjekte sind im Skript `topology.sql`.

Für einen vollständigen Satz an EPSG-Koordinatensystemdefinitionen können Sie auch die Definitionsdatei `spatial_ref_sys.sql` laden um die Tabelle `spatial_ref_sys` zu befüllen. Dies ermöglicht Ihnen ST_Transform()-Operationen an Geometrien durchzuführen.

Wenn Sie Kommentare zu den PostGIS-Funktionen hinzufügen möchten, finden Sie diese im Skript `postgis_comments.sql`. Kommentare können durch die Eingabe von **\dd [function_name]** in einem **psql** Terminalfenster angezeigt werden.

Führen Sie die folgenden Shell-Befehle in Ihrem Terminal aus:

Code

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

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

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

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

3.4 Aktualisierung von Geodatenbanken

Die Aktualisierung bestehender räumlicher Datenbanken kann schwierig sein, da sie den Austausch oder die Einführung neuer PostGIS Objektdefinitionen erfordert.

Da leider nicht alle Definitionen in einer aktiven Datenbank einfach ersetzt werden können, ist manchmal ein Dump/Reload-Prozess die beste Lösung.

PostGIS bietet ein SOFT UPGRADE-Verfahren für kleinere oder Bugfix-Releases sowie ein HARD-UPGRADE-Verfahren für Haupt-Releases.

Bevor PostGIS aktualisiert wird, sollten die Daten unbedingt gesichert werden. Wenn die Option -Fc beim Befehl pg_dump verwendet wird, kann der Dump bei einem HARD UPGRADE immer wieder hergestellt werden.

3.4.1 Sanfte Aktualisierung

Wenn Sie Ihre Datenbank mithilfe von EXTENSION installiert haben, müssen Sie das Upgrade ebenfalls über das EXTENSION Modell durchführen. Falls Sie die Installation über die alte SQL-Skript-Methode vorgenommen haben, wird empfohlen, auf die Nutzung von EXTENSION umzusteigen, da die Skript-Methode nicht länger unterstützt wird.

3.4.1.1 Soft upgrade 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.

Use the PostGIS_Extensions_Upgrade function to upgrade to the latest version you have installed.

Code

```
SELECT postgis_extensions_upgrade();
```

If you are upgrading a database that still runs PostGIS 2.5, issue an **ALTER EXTENSION** followed by `postgis_extensions_upg` to make sure the legacy `postgis_raster` objects are folded back into the main extension.

Code

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

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

Code

```
SELECT postgis_extensions_upgrade();
```

Wenn Sie mehrere Versionen von PostGIS installiert haben und nicht auf die neueste Version aktualisieren möchten, können Sie die gewünschte Version wie folgt explizit angeben:

Code

```
ALTER EXTENSION postgis UPDATE TO "3.8.0dev";
ALTER EXTENSION postgis_topology UPDATE TO "3.8.0dev";
```

Falls Sie eine Fehlermeldung erhalten, die in etwa wie folgt aussieht:

Output

```
No migration path defined for ... to 3.8.0dev
```

Dann müssen Sie eine Sicherungskopie Ihrer Datenbank anlegen, eine neue Datenbank erstellen, wie unter Abschnitt 3.3.1 beschrieben, und dann Ihre Sicherungskopie in dieser neuen Datenbank wiederherstellen.

Falls Sie eine Hinweisnachricht erhalten, die in etwa wie folgt aussieht:

Output

```
Version "3.8.0dev" of extension "postgis" is already installed
```

Dann ist bereits alles auf dem neuesten Stand und Sie können das getrost ignorieren. **Es sei denn,** Sie versuchen, von einer Entwicklungsversion auf die nächste zu aktualisieren (die keine neue Versionsnummer erhält); in diesem Fall können Sie "next" an die Versionszeichenfolge anhängen, und beim nächsten Mal müssen Sie das Suffix "next" wieder weglassen:

Code

```
ALTER EXTENSION postgis UPDATE TO "3.8.0devnext";
        ALTER EXTENSION postgis_topology UPDATE TO "3.8.0devnext";
```



Anmerkung
Wenn Sie PostGIS ursprünglich installiert haben ohne eine Version anzugeben , können Sie die Neuinstallation der postgis-Erweiterung vor der Wiederherstellung oftmals überspringen, da im Backup nur `CREATE EXTENSION postgis` steht und daher die aktuellste Version für die Wiederherstellung herangezogen wird.



Anmerkung
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.

Code

```
DROP EXTENSION postgis_raster;
```

3.4.1.2 Soft upgrade without extensions

Extensions have been the supported installation method for many releases. If you still rely on an unpackaged installation, perform a HARD UPGRADE into a database created with extensions and migrate your data there.

3.4.1.3 Hartes Upgrade

Unter HARD UPGRADE verstehen wir ein vollständiges Dump/Reload von PostGIS-aktivierten Datenbanken. Ein HARD UPGRADE ist erforderlich, wenn sich der interne Speicher von PostGIS-Objekten ändert oder wenn ein SOFT UPGRADE nicht möglich ist. Der Anhang Release Notes gibt für jede Version an, ob ein Dump/Reload (HARD UPGRADE) für ein Upgrade erforderlich ist.

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

Supplementary instructions for windows users are available in the PostGIS Windows installation guide.

Das Verfahren ist wie folgt:

1. Erstellen Sie einen Dump im "benutzerdefinierten Format" der Datenbank, die Sie aktualisieren möchten (nennen wir ihn `olddb`), einschließlich binärer Blobs (-b) und ausführlicher (-v) Ausgabe. Der Benutzer kann der Eigentümer der Datenbank sein, er muss nicht das Postgres-Superkonto sein.

   Code

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

2. Führen Sie eine Neuinstallation von PostGIS in einer neuen Datenbank durch - wir bezeichnen diese Datenbank als `newdb`. Eine Anleitung dazu finden Sie unter Abschnitt 3.3.2 und Abschnitt 3.3.1.

   Die spatial_ref_sys-Einträge, die Sie in Ihrem Dump gefunden haben, werden wiederhergestellt, überschreiben aber nicht die vorhandenen Einträge in spatial_ref_sys. Damit soll sichergestellt werden, dass die Korrekturen im offiziellen Satz ordnungsgemäß in die wiederhergestellten Datenbanken übertragen werden. Wenn Sie aus irgendeinem Grund eigene Überschreibungen von Standardeinträgen wünschen, laden Sie die Datei spatial_ref_sys.sql beim Erstellen der neuen Datenbank einfach nicht.

   Wenn Ihre Datenbank wirklich alt ist oder Sie wissen, dass Sie lange veraltete Funktionen in Ihren Ansichten und Funktionen verwendet haben, müssen Sie möglicherweise `legacy.sql` laden, damit alle Ihre Funktionen und Ansichten usw. wieder richtig funktionieren. Tun Sie dies nur, wenn es _wirklich_ notwendig ist. Erwägen Sie stattdessen, Ihre Ansichten und Funktionen vor dem Dumping zu aktualisieren, falls möglich. Die veralteten Funktionen können später durch Laden von `uninstall_legacy.sql` entfernt werden.

3. Stellen Sie Ihre Sicherung mit postgis_restore in Ihre neue Datenbank `newdb` wieder her. Unerwartete Fehler werden, falls vorhanden, von psql in den Standardfehlerstrom ausgegeben. Führen Sie ein Protokoll über diese Fehler.

Code

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

In den folgenden Fällen kann es zu Fehlern kommen:

1. Einige Ihrer Ansichten oder Funktionen verwenden veraltete PostGIS-Objekte. Um dies zu beheben, können Sie versuchen, das Skript `legacy.sql` vor der Wiederherstellung zu laden, oder Sie müssen eine Version von PostGIS wiederherstellen, die diese Objekte noch enthält, und nach der Portierung Ihres Codes erneut eine Migration versuchen. Wenn der `legacy.sql` Weg für Sie funktioniert, vergessen Sie nicht, Ihren Code so zu korrigieren, dass er keine veralteten Funktionen mehr verwendet und sie durch Laden von `uninstall_legacy.sql` fallen lässt.

2. Einige benutzerdefinierte Datensätze von spatial_ref_sys in der Dump-Datei haben einen ungültigen SRID-Wert. Gültige SRID-Werte sind größer als 0 und kleiner als 999000. Werte im Bereich 999000.999999 sind für den internen Gebrauch reserviert, während Werte > 999999 überhaupt nicht verwendet werden können. Alle benutzerdefinierten Datensätze mit ungültigen SRIDs werden beibehalten, wobei die Werte > 999999 in den reservierten Bereich verschoben werden, aber die Tabelle spatial_ref_sys würde eine Prüfbeschränkung verlieren, die dafür sorgt, dass diese Invariante beibehalten wird, und möglicherweise auch ihren Primärschlüssel (wenn mehrere ungültige SRIDS in denselben reservierten SRID-Wert umgewandelt werden).

   Um das Problem zu beheben, sollten Sie Ihre benutzerdefinierte SRS auf eine SRID mit einem gültigen Wert (vielleicht im Bereich 910000..910999) kopieren, alle Ihre Tabellen in die neue SRID konvertieren (siehe UpdateGeometrySRID), den ungültigen Eintrag aus spatial_ref_sys löschen und die Prüfung(en) mit neu konstruieren:

   Code

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

   Code

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

   Wenn Sie eine alte Datenbank aktualisieren, die französische IGN -Kartografie enthält, werden Sie wahrscheinlich SRIDs außerhalb des Bereichs haben und beim Importieren Ihrer Datenbank Probleme wie diese sehen:

   Code

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

   In diesem Fall können Sie die folgenden Schritte versuchen: Zuerst werfen Sie die IGN komplett aus der Sql, die aus postgis_restore resultiert. Also, nachdem Sie :

   Code

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

   führen Sie diesen Befehl aus:

   Code

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

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

Code

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

Kapitel 4

Datenverwaltung

4.1 Räumliches Datenmodell

4.1.1 OGC-Geometrie

Das Open Geospatial Consortium (OGC) hat den Standard *Simple Features Access* (SFA) entwickelt, um ein Modell für Geodaten bereitzustellen. Er definiert den grundlegenden Raumtyp **Geometrie** sowie Operationen, die Geometriewerte manipulieren und transformieren, um räumliche Analyseaufgaben durchzuführen. PostGIS implementiert das OGC Geometry Modell als die PostgreSQL Datentypen Geometry und Geography.

Geometrie ist ein *abstrakter* Typ. Geometriewerte gehören zu einem seiner *konkreten* Untertypen, die verschiedene Arten und Dimensionen geometrischer Formen darstellen. Dazu gehören die **atomaren** Typen Point, LineString, LinearRing und Polygon, und die **Sammlung** Typen MultiPoint, MultiLineString, MultiPolygon und GeometryCollection. Der *Simple Features Access - Part 1: Gemeinsame Architektur v1.2.1* fügt Subtypen für die Strukturen PolyhedralSurface, Triangle und TIN hinzu.

Geometry modelliert Formen in der 2-dimensionalen kartesischen Ebene. Die Typen PolyhedralSurface, Triangle und TIN können auch Formen im 3-dimensionalen Raum darstellen. Die Größe und Lage von Formen wird durch ihre **Koordinaten** angegeben. Jede Koordinate hat einen X- und Y **Ordinatenwert**, der ihre Lage in der Ebene bestimmt. Formen werden aus Punkten oder Liniensegmenten konstruiert, wobei Punkte durch eine einzelne Koordinate und Liniensegmente durch zwei Koordinaten angegeben werden.

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

Geometriewerte sind mit einem **räumlichen Bezugssystem** verbunden, das das Koordinatensystem angibt, in das sie eingebettet sind. Das räumliche Bezugssystem wird durch die SRID-Nummer der Geometrie identifiziert. Die Einheiten der X- und Y-Achsen werden durch das räumliche Bezugssystem bestimmt. In **planaren** Bezugssystemen stellen die X- und Y-Koordinaten typischerweise Ost- und Nordrichtung dar, während sie in **geodätischen** Systemen Längen- und Breitengrad darstellen. SRID 0 steht für eine unendliche kartesische Ebene, deren Achsen keine Einheiten zugewiesen sind. Siehe Abschnitt 4.5.

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

Ein Geometriewert kann **leer** sein. Leere Werte enthalten keine Scheitelpunkte (bei atomaren Geometrietypen) oder keine Elemente (bei Sammlungen).

Eine wichtige Eigenschaft von Geometriewerten ist ihre räumliche **Ausdehnung** oder **bounding box**, die im OGC-Modell **envelope** genannt wird. Dies ist der 2- oder 3-dimensionale Rahmen, der die Koordinaten einer Geometrie umschließt. Es ist ein effizientes Mittel, um die Ausdehnung einer Geometrie im Koordinatenraum darzustellen und zu prüfen, ob zwei Geometrien interagieren.

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

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

4.1.1.1 Punkt

Ein Punkt ist eine 0-dimensionale Geometrie, die einen einzelnen Ort im Koordinatenraum darstellt.

Code

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

4.1.1.2 Linie

Ein LineString ist eine eindimensionale Linie, die aus einer zusammenhängenden Folge von Liniensegmenten besteht. Jedes Liniensegment wird durch zwei Punkte definiert, wobei der Endpunkt eines Segments den Startpunkt des nächsten Segments bildet. Ein OGC-konformer LineString hat entweder null oder zwei oder mehr Punkte, aber PostGIS erlaubt auch Ein-Punkt-LineStrings. LineStrings können sich selbst kreuzen (self-intersect). Ein LineString ist **geschlossen** wenn der Start- und Endpunkt gleich sind. Ein LineString ist **einfach**, wenn er sich nicht selbst schneidet.

Code

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

4.1.1.3 LinearRing

Ein LinearRing ist ein LineString, der sowohl geschlossen als auch einfach ist. Der erste und der letzte Punkt müssen gleich sein, und die Linie darf sich nicht selbst schneiden.

Code

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

4.1.1.4 Polygon

Ein Polygon ist ein 2-dimensionaler ebener Bereich, der durch eine äußere Begrenzung (die Schale) und null oder mehr innere Begrenzungen (Löcher) begrenzt wird. Jede Begrenzung ist ein LinearRing.

Code

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

4.1.1.5 MultiPoint

Ein MultiPoint ist eine Sammlung von Punkten.

Code

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

4.1.1.6 MultiLineString

Ein MultiLineString ist eine Sammlung von LineStrings. Ein MultiLineString ist geschlossen, wenn jedes seiner Elemente geschlossen ist.

Code

```
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. (Two polygons are adjacent if they share an edge. They touch if they share only points or edges on their boundaries). For more details on MultiPolygon validity, see Abschnitt 4.4.

Code

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

4.1.1.8 GeometryCollection

Eine GeometryCollection ist eine heterogene (gemischte) Sammlung von Geometrien.

Code

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

4.1.1.9 PolyhedralSurface

Eine PolyhedralSurface ist eine zusammenhängende Sammlung von Flächen oder Facetten, die einige Kanten gemeinsam haben. Jedes Feld ist ein planares Polygon. Wenn die Polygonkoordinaten Z-Ordinaten haben, ist die Fläche 3-dimensional.

Code

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

4.1.1.10 Dreieck

Ein Dreieck ist ein Polygon, das durch drei verschiedene, nicht kollineare Scheitelpunkte definiert ist. Da ein Dreieck ein Polygon ist, wird es durch vier Koordinaten definiert, wobei die erste und die vierte gleich sind.

Code

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

4.1.1.11 TIN

Ein TIN ist eine Sammlung von nicht überlappenden Dreiecken, die ein Trianguliertes unregelmäßiges Netz darstellen.

Code

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

4.1.2 SQL/MM Teil 3 - Kurven

Der *ISO/IEC 13249-3 SQL Multimedia - Spatial* Standard (SQL/MM) erweitert den OGC SFA um die Definition von Geometrie-Subtypen, die Kurven mit Kreisbögen enthalten. Die SQL/MM-Typen unterstützen 3DM-, 3DZ- und 4D-Koordinaten.



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

4.1.2.1 CircularString

CircularString ist der grundlegende Kurventyp, ähnlich einem LineString in der linearen Welt. Ein einzelnes Bogensegment wird durch drei Punkte spezifiziert: den Anfangs- und Endpunkt (erster und dritter Punkt) und einen weiteren Punkt auf dem Bogen. Zur Angabe eines geschlossenen Kreises sind der Anfangs- und der Endpunkt identisch, und der mittlere Punkt ist der gegenüberliegende Punkt auf dem Kreisdurchmesser (der den Mittelpunkt des Bogens bildet). In einer Folge von Bögen ist der Endpunkt des vorherigen Bogens der Startpunkt des nächsten Bogens, genau wie die Segmente eines LineString. Dies bedeutet, dass ein CircularString eine ungerade Anzahl von Punkten größer als 1 haben muss.

Code

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

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

4.1.2.2 CompoundCurve

Eine CompoundCurve ist eine einzelne kontinuierliche Kurve, die sowohl Kreisbogensegmente als auch lineare Segmente enthalten kann. Das bedeutet, dass nicht nur wohlgeformte Komponenten vorhanden sein müssen, sondern auch der Endpunkt jeder Komponente (außer der letzten) mit dem Anfangspunkt der folgenden Komponente übereinstimmen muss.

Code

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

4.1.2.3 KurvenPolygon

Ein CurvePolygon ist wie ein Polygon, mit einem äußeren Ring und null oder mehr inneren Ringen. Der Unterschied besteht darin, dass ein Ring sowohl ein CircularString oder CompoundCurve als auch ein LineString sein kann.

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

Code

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

Beispiel: Ein CurvePolygon mit einer Hülle, die durch eine CompoundCurve definiert ist, die einen CircularString und einen LineString enthält, und einem Loch, das durch einen CircularString definiert ist

Code

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

4.1.2.4 MultiCurve

Eine MultiCurve ist eine Sammlung von Kurven, die LineStrings, CircularStrings oder CompoundCurves enthalten kann.

Code

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

4.1.2.5 MultiSurface

Eine MultiSurface ist eine Sammlung von Flächen, die (lineare) Polygone oder CurvePolygone sein können.

Code

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

4.1.3 WKT und WKB

Die OGC SFA-Spezifikation definiert zwei Formate für die Darstellung von Geometriewerten zur externen Verwendung: Well-Known Text (WKT) und Well-Known Binary (WKB). Sowohl WKT als auch WKB enthalten Informationen über den Typ des Objekts und die Koordinaten, die es definieren.

Well-Known Text (WKT) bietet eine standardisierte textuelle Darstellung von räumlichen Daten. Beispiele für WKT-Darstellungen von räumlichen Objekten sind:

- POINT(0 0)
- PUNKT Z (0 0 0)
- PUNKT ZM (0 0 0 0)
- PUNKT LEER
- LINESTRING(0 0,1 1,1 2)
- ZEILENSTRING LEER
- 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))
- MEHRPUNKT Z ((0 0 0),(1 2 3))

- MULTIPOINT LEER
- 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 LEER

Die Ein- und Ausgabe des WKT erfolgt über die Funktionen ST_AsText und ST_GeomFromText:

Code

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

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

Code

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

Well-Known Binary (WKB) bietet eine tragbare, hochpräzise Darstellung von Geodaten als Binärdaten (Arrays von Bytes). Beispiele für die WKB-Darstellung von Geo-Objekten sind:

- WKT: PUNKT(1 1)

  WKB: 010100000000000000000F03F000000000000F03
- WKT: LINIENSTRING (2 2, 9 9)

  WKB: 0102000000020000000000000000000040000000000000004000000000000022400000000000002240

Die Ein- und Ausgabe von WKB erfolgt über die Funktionen ST_AsBinary und ST_GeomFromWKB:

Code

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

Eine Anweisung zum Erstellen und Einfügen eines Geo-Objekts aus WKB lautet zum Beispiel so:

Code

```
INSERT INTO geotable ( geom, name )
VALUES ( ST_GeomFromWKB('\x010100000000000000000f03f000000000000f03f', 312), 'A Place');
```

4.2 Geometrie Datentyp

PostGIS implementiert das OGC Simple Features Modell durch die Definition eines PostgreSQL Datentyps namens `geometry`. Er repräsentiert alle Geometrie-Subtypen durch Verwendung eines internen Typcodes (siehe GeometryType und ST_GeometryType). Dies ermöglicht die Modellierung von räumlichen Merkmalen als Zeilen von Tabellen, die mit einer Spalte vom Typ `geometry` definiert sind.

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



Anmerkung
PostGIS folgt dem SFA-Standard, indem es den räumlichen Funktionen das Kürzel "ST_" voranstellt. Dies sollte für "Spatial and Temporal" (räumlich und zeitlich) stehen, aber der zeitliche Teil des Standards wurde nie entwickelt. Stattdessen kann es als "Spatial Type" interpretiert werden.

Der SFA-Standard sieht vor, dass Geo-Objekte einen Spatial Reference System Identifier (SRID) enthalten. Der SRID ist erforderlich, wenn Geo-Objekte zum Einfügen in die Datenbank erstellt werden (er kann auf 0 voreingestellt sein). Siehe ST_SRID und Abschnitt 4.5

Um die Abfrage von Geometrien effizient zu gestalten, definiert PostGIS verschiedene Arten von räumlichen Indizes und räumliche Operatoren, um diese zu verwenden. Siehe Abschnitt 4.9 und Abschnitt 5.2 für Details.

4.2.1 PostGIS EWKB und EWKT

Die OGC SFA-Spezifikationen unterstützten ursprünglich nur 2D-Geometrien, und die Geometrie-SRID ist nicht in den Eingabe-/Ausgabedarstellungen enthalten. Die OGC-SFA-Spezifikation 1.2.1 (die mit der ISO-Norm 19125 übereinstimmt) unterstützt nun auch 3D- (ZYZ) und gemessene (XYM und XYZM) Koordinaten, enthält aber immer noch keinen SRID-Wert.

Aufgrund dieser Einschränkungen hat PostGIS erweiterte EWKB- und EWKT-Formate definiert. Sie bieten Unterstützung für 3D- (XYZ und XYM) und 4D-Koordinaten (XYZM) und enthalten SRID-Informationen. Durch die Einbeziehung aller Geometrieinformationen kann PostGIS EWKB als Datensatzformat verwenden (z. B. in DUMP-Dateien).

EWKB und EWKT werden für die "kanonischen Formen" von PostGIS-Datenobjekten verwendet. Für die Eingabe ist die kanonische Form für binäre Daten EWKB, und für Textdaten wird entweder EWKB oder EWKT akzeptiert. Dies ermöglicht die Erstellung von Geometriewerten durch Umwandlung eines Textwerts in HEXEWKB oder EWKT in einen Geometriewert unter Verwendung von `::geometry`. Für die Ausgabe ist die kanonische Form für Binärdaten EWKB und für Textdaten HEXEWKB (hexkodiertes EWKB).

Diese Anweisung erzeugt zum Beispiel eine Geometrie durch Casting aus einem EWKT-Textwert und gibt sie in der kanonischen Form HEXEWKB aus:

Code

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

Output

```
geometry
----------------------------------------------------
0101000020040000000000000000000000000000000000000000
```

Die PostGIS EWKT-Ausgabe weist einige Unterschiede zur OGC WKT auf:

- Bei 3DZ-Geometrien entfällt der Qualifier Z:
  OGC: PUNKT Z (1 2 3)
  EWKT: PUNKT (1 2 3)
- Für 3DM-Geometrien ist der Qualifier M enthalten:
  OGC: PUNKT M (1 2 3)
  EWKT: POINTM (1 2 3)
- Bei 4D-Geometrien entfällt der ZM-Bezeichner:
  OGC: PUNKT ZM (1 2 3 4)
  EWKT: PUNKT (1 2 3 4)

EWKT vermeidet eine übermäßige Spezifizierung der Dimensionalität und die Inkonsistenzen, die beim OGC/ISO-Format auftreten können, wie z. B.:

- PUNKT ZM (1 1)
- PUNKT ZM (1 1 1)
- PUNKT (1 1 1 1)



Achtung
Die erweiterten PostGIS-Formate sind derzeit eine Obermenge der OGC-Formate, so dass jedes gültige OGC-WKB/WKT auch ein gültiges EWKB/EWKT ist. Dies könnte sich jedoch in Zukunft ändern, wenn das OGC ein Format in einer Weise erweitert, die mit der PosGIS-Definition in Konflikt steht. Sie sollten sich also NICHT auf diese Kompatibilität verlassen!

Beispiele für die EWKT-Textdarstellung von räumlichen Objekten sind:

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

Die Ein- und Ausgabe in diesen Formaten ist über die folgenden Funktionen möglich:

Code

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

Eine Anweisung zum Erstellen und Einfügen eines PostGIS-Gebietsobjekts unter Verwendung von EWKT lautet zum Beispiel so:

Code

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

4.3 Geographie Datentyp

Der Datentyp PostGIS `geography` bietet native Unterstützung für räumliche Merkmale, die in "geografischen" Koordinaten (manchmal auch "geodätische" Koordinaten oder "lat/lon" oder "lon/lat" genannt) dargestellt werden. Geografische Koordinaten sind sphärische Koordinaten, die in Winkeleinheiten (Grad) ausgedrückt werden.

Die Grundlage für den PostGIS-Geometriedatentyp ist eine Ebene. Der kürzeste Weg zwischen zwei Punkten in der Ebene ist eine gerade Linie. Das bedeutet, dass Funktionen für Geometrien (Flächen, Abstände, Längen, Schnittpunkte usw.) mit Geradenvektoren und kartesischer Mathematik berechnet werden. Dadurch sind sie einfacher zu implementieren und schneller auszuführen, aber auch ungenau für Daten auf der sphäroidischen Oberfläche der Erde.

Der PostGIS-Geodatentyp basiert auf einem Kugelmodell. Der kürzeste Weg zwischen zwei Punkten auf der Kugel ist ein Großkreisbogen. Funktionen auf Geografien (Flächen, Entfernungen, Längen, Schnittpunkte usw.) werden mit Hilfe von Bögen auf der Kugel berechnet. Da die Funktionen die Kugelform der Welt berücksichtigen, liefern sie genauere Ergebnisse.

Da die zugrunde liegende Mathematik komplizierter ist, sind für den Typ Geografie weniger Funktionen definiert als für den Typ Geometrie. Im Laufe der Zeit, wenn neue Algorithmen hinzukommen, werden sich die Möglichkeiten des Typs Geografie erweitern. Als Abhilfe kann man zwischen den Typen Geometrie und Geografie hin- und herwechseln.

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. You can add your own custom geodetic spatial reference system as described in Abschnitt 4.5.2.

Für alle räumlichen Bezugssysteme sind die Einheiten, die von Messfunktionen (z. B. ST_Distance, ST_Length, ST_Perimeter, ST_Area) und für das Entfernungsargument von ST_DWithin zurückgegeben werden, in Metern.

4.3.1 Erstellen von Geografietabellen

Sie können eine Tabelle zum Speichern von geografischen Daten mit der SQL-Anweisung CREATE TABLE mit einer Spalte vom Typ `Geografie` erstellen. Das folgende Beispiel erstellt eine Tabelle mit einer Geografiespalte, die 2D LineStrings im geodätischen Koordinatensystem WGS84 (SRID 4326) speichert:

Code

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

Der Geografietyp unterstützt zwei optionale Typmodifikatoren:

- Der Modifikator für die Raumart schränkt die Art der in der Spalte zulässigen Formen und Abmessungen ein. Für die Raumart sind folgende Werte zulässig: POINT, LINESTRING, POLYGON, MULTIPOINT, MULTILINESTRING, MULTIPOLYGON, GEOMETRYCOLLECTION. Der Geografietyp unterstützt keine Kurven, TINS oder POLYHEDRALSURFACEs. Der Modifikator unterstützt Einschränkungen der Koordinatendimensionalität durch Hinzufügen von Suffixen: Z, M und ZM. Ein Modifikator "LINESTRINGM" lässt beispielsweise nur Linienzüge mit drei Dimensionen zu und behandelt die dritte Dimension als Maß. In ähnlicher Weise erfordert 'POINTZM' vierdimensionale (XYZM) Daten.
- Der Modifikator SRID schränkt das räumliche Bezugssystem SRID auf eine bestimmte Nummer ein. Wird dieser Modifikator weggelassen, so ist das SRID standardmäßig 4326 (WGS84 geodätisch), und alle Berechnungen werden mit WGS84 durchgeführt.

Beispiele für die Erstellung von Tabellen mit geografischen Spalten:

Erstellen Sie eine Tabelle mit 2D-Punktgeografie mit dem Standard-SRID 4326 (WGS84 long/lat):

Code

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

Erstellen Sie eine Tabelle mit 2D-Punktgeografie in NAD83 longlat:

Code

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

Erstellen Sie eine Tabelle mit 3D (XYZ) POINTs und einer expliziten SRID von 4326:

Code

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

Erstellen Sie eine Tabelle mit der Geografie 2D LINESTRING mit dem Standard-SRID 4326:

Code

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

Erstellen Sie eine Tabelle mit einer 2D POLYGON-Geografie mit dem SRID 4267 (NAD 1927 long lat):

Code

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

Geografische Felder werden in der Systemansicht `geography_columns` registriert. Sie können die Ansicht `geography_columns` abfragen und sehen, dass die Tabelle aufgeführt ist:

Code

```
SELECT * FROM geography_columns;
```

Das Erstellen eines räumlichen Indexes funktioniert genauso wie bei Geometriespalten. PostGIS stellt fest, dass der Spaltentyp GEOGRAPHIE ist und erstellt einen entsprechenden kugelbasierten Index anstelle des üblichen planaren Index für GEOMETRI.

Code

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

4.3.2 Verwendung von Geografietabellen

Sie können Daten in Geografietabellen auf dieselbe Weise wie Geometrie einfügen. Geometriedaten werden automatisch in den Geographietyp übertragen, wenn sie SRID 4326 haben. Die Formate EWKT und EWKB können auch zur Angabe von Geografiewerten verwendet werden.

Code

```
INSERT INTO global_points (name, location) VALUES ('Town', 'SRID=4326;POINT(-110 30)');
INSERT INTO global_points (name, location) VALUES ('Forest', 'SRID=4326;POINT(-109 29)');
INSERT INTO global_points (name, location) VALUES ('London', 'SRID=4326;POINT(0 49)');
```

Jedes in der Tabelle `spatial_ref_sys` aufgeführte geodätische (long/lat) Raumbezugssystem kann als Geografie-SRID angegeben werden. Nicht-geodätische Koordinatensysteme führen zu einem Fehler, wenn sie verwendet werden.

A NAD83 longitude/latitude geography value:

Code

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

Output

```
geography
----------------------------------------------------
 0101000020AD10000000000000000C05EC00000000000004140
```

A NAD27 longitude/latitude geography value:

Code

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

Output

```
geography
----------------------------------------------------
 0101000020AB10000000000000000C05EC00000000000004140
```

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

Code

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

Output

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

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

This query finds rows within a 1000 km tolerance:

Code

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

Sie können die Macht der Geografie in Aktion sehen, indem Sie berechnen, wie nahe ein Flugzeug, das eine Großkreisroute von Seattle nach London (LINESTRING(-122.33 47.606, 0.0 51.5)) fliegt, an Reykjavik (POINT(-21.96 64.15)) herankommt (Karte der Route).

Der Geographie-Typ berechnet die wahre kürzeste Entfernung von 122,235 km über die Kugel zwischen Reykjavik und der Großkreisfluglinie zwischen Seattle und London.

Code

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

Output

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

Der Geometrietyp berechnet eine bedeutungslose kartesische Entfernung zwischen Reykjavik und der geraden Strecke von Seattle nach London, die auf einer flachen Weltkarte eingezeichnet ist. Die nominale Einheit des Ergebnisses ist "Grad", aber das Ergebnis entspricht keiner echten Winkeldifferenz zwischen den Punkten, so dass selbst die Bezeichnung "Grad" ungenau ist.

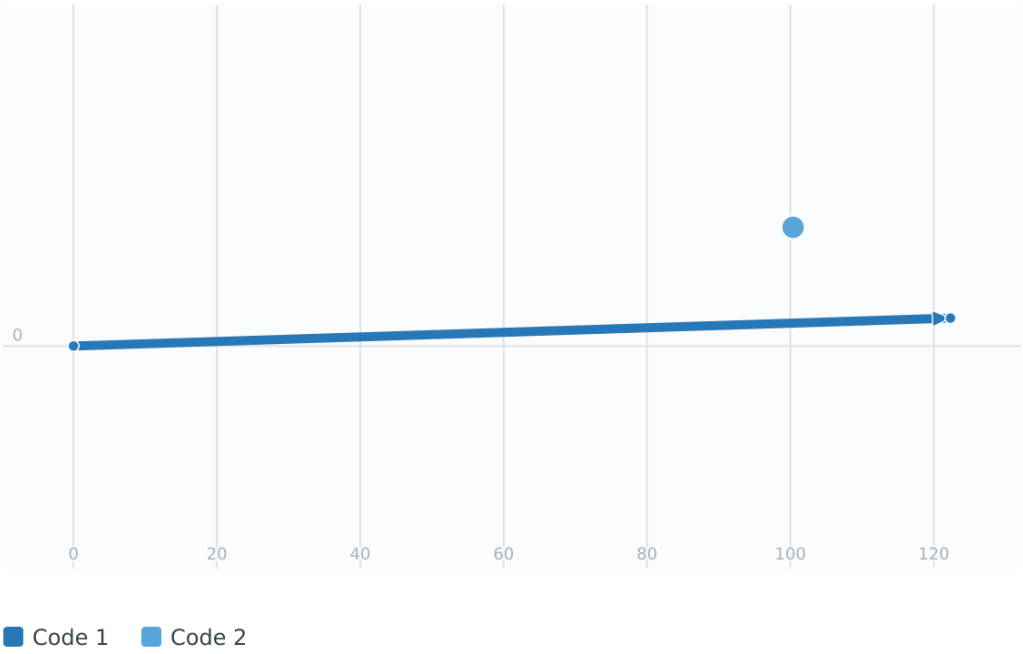
Code

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

Output

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

Figure



4.3.3 Wann wird der Datentyp Geografie verwendet?

Der Datentyp GEOGRAPHY ermöglicht die Speicherung von Daten in Längen- und Breitenkoordinaten, allerdings zu einem gewissen Preis: Für GEOGRAPHY sind weniger Funktionen definiert als für GEOMETRY; die definierten Funktionen benötigen mehr CPU-Zeit zur Ausführung.

Der von Ihnen gewählte Datentyp sollte sich nach dem voraussichtlichen Arbeitsbereich der Anwendung richten, die Sie erstellen. Erstrecken sich Ihre Daten über den gesamten Globus oder ein großes kontinentales Gebiet, oder sind sie auf ein Bundesland, einen Bezirk oder eine Gemeinde beschränkt?

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

Für einen Vergleich, welche Funktionalität von Geography vs. Geometry unterstützt wird, siehe Abschnitt 13.11. Für eine kurze Liste mit der Beschreibung der geographischen Funktionen, siehe Abschnitt 13.4

4.3.4 Fortgeschrittene FAQ's zum geographischen Datentyp

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

   By default, all distance and area calculations are done on the spheroid. You should find that the results of calculations in local areas match up well with local planar results in good local projections. Over larger areas, the spheroidal calculations will be more accurate than any calculation done on a projected plane. The difference is that the spheroid takes into account the curvature and non-spherical shape of the Earth, whereas planar calculations assume a flat surface. Alle geographischen Funktionen verfügen über eine Option um die Berechnung auf einer Kugel durchzuführen. Dies erreicht man, indem der letzte boolesche Eingabewert auf 'FALSE' gesetzt wird. Dies beschleunigt die Berechnung einigermaßen, insbesondere wenn die Geometrie sehr einfach gestaltet ist.

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

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

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

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

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

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

4.4 Geometrievalidierung

PostGIS ist mit der Spezifikation Simple Features des Open Geospatial Consortium (OGC) konform. Dieser Standard definiert die Konzepte der Geometrie *simple* und *valid*. Diese Definitionen ermöglichen es dem Simple-Features-Geometriemodell, räumliche Objekte in einer konsistenten und eindeutigen Weise darzustellen, die effiziente Berechnungen unterstützt. (Anmerkung: OGC SF und SQL/MM haben die gleichen Definitionen für einfach und gültig).

4.4.1 Einfache Geometrie

Eine *einfache* Geometrie ist eine Geometrie, die keine anomalen geometrischen Punkte wie Selbstschnittpunkte oder Selbsttangenten aufweist.

Ein `POINT` ist von Natur aus *einfach* als ein 0-dimensionales Geometrieobjekt.

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

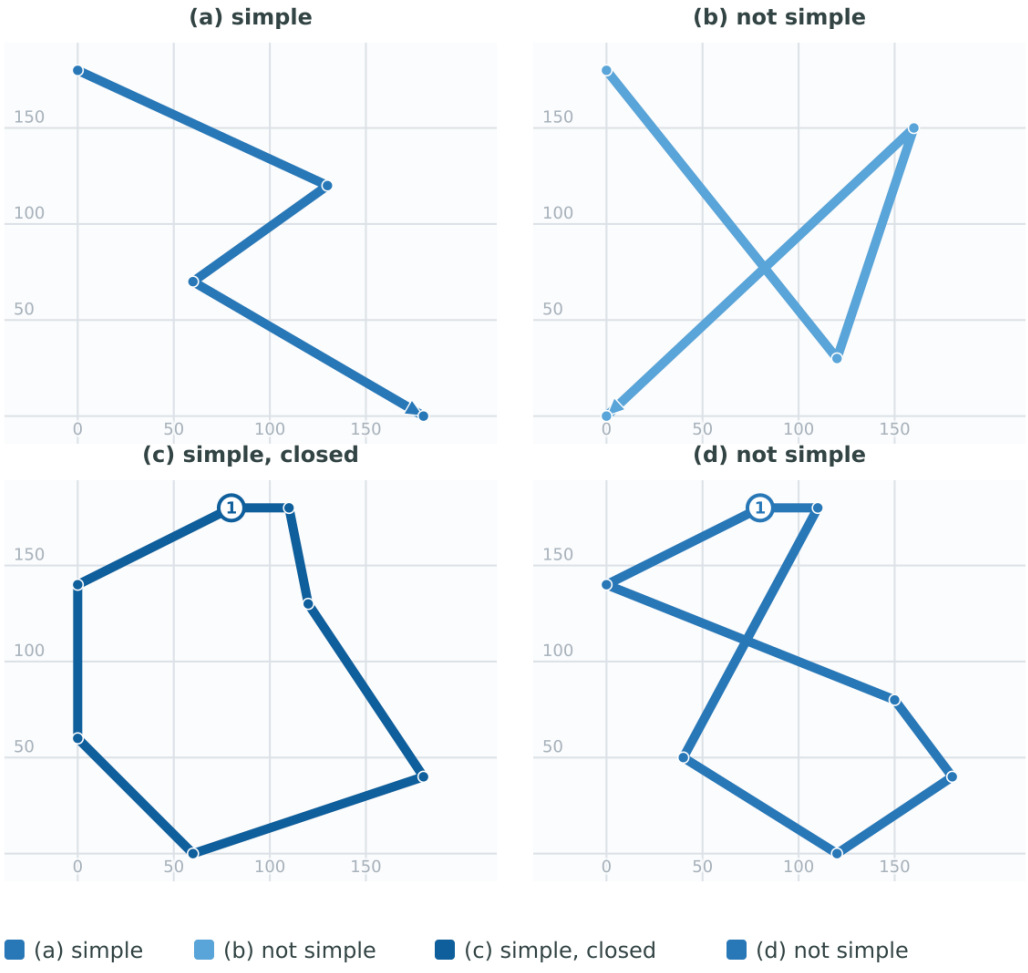
Ein `LINESTRING` ist *einfach*, wenn er, abgesehen von den Endpunkten, nicht zweimal durch denselben Punkt verläuft. Wenn die Endpunkte eines einfachen Linienstrangs identisch sind, wird er *geschlossen* und als linearer Ring bezeichnet.

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

Code

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

Figure



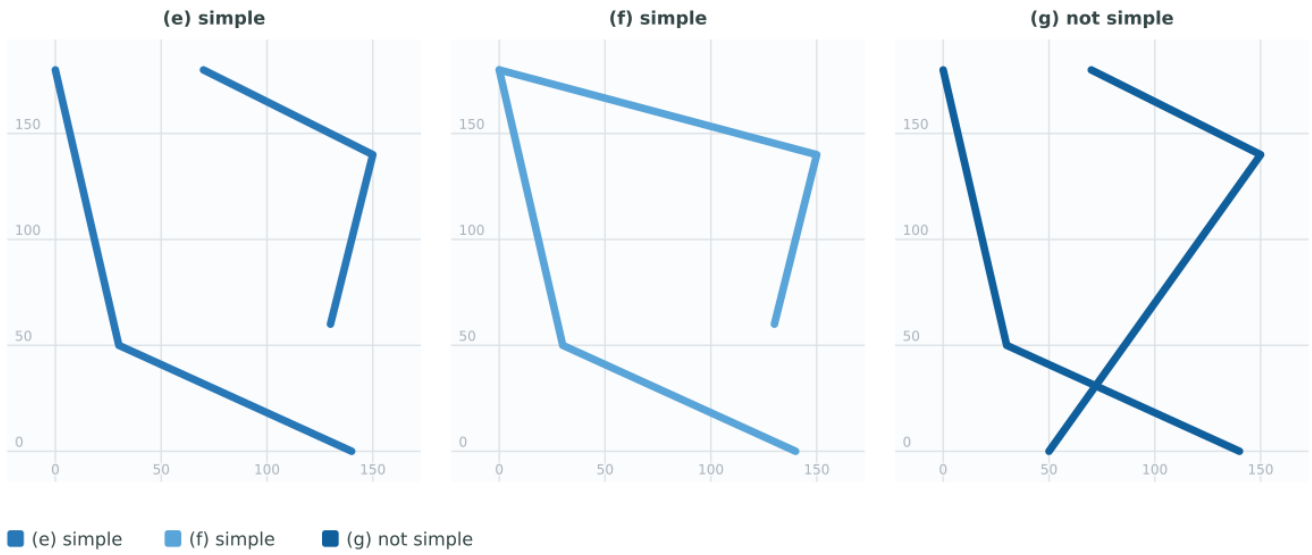
Ein `MULTILINESTRING` ist nur dann *einfach*, wenn alle seine Elemente einfach sind und der einzige Schnittpunkt zwischen zwei beliebigen Elementen an Punkten auftritt, die auf den Grenzen der beiden Elemente liegen.

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

Code

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

Figure



`POLYGON`s werden aus linearen Ringen gebildet, daher ist eine gültige polygonale Geometrie immer *einfach*.

Im Allgemeinen verlangen PostGIS-Funktionen nicht, dass geometrische Argumente einfach sind. Die Einfachheit wird in erster Linie als Grundlage für die Definition der geometrischen Gültigkeit verwendet. Sie ist auch eine Voraussetzung für einige Arten von Geodatenmodellen (z. B. lassen lineare Netze oft keine Linien zu, die sich kreuzen). Mehrpunkt- und lineare Geometrie kann mit ST_UnaryUnion vereinfacht werden.

4.4.2 Gültige Geometrie

Die Gültigkeit der Geometrie gilt in erster Linie für 2-dimensionale Geometrien (`POLYGON`s und `MULTIPOLYGON`s) . Die Gültigkeit wird durch Regeln definiert, die es der polygonalen Geometrie ermöglichen, ebene Flächen eindeutig zu modellieren.

Ein `POLYGON` ist *gültig* wenn:

1. Die Begrenzungsringe des Polygons (der äußere Schalenring und die inneren Lochringe) sind *einfach* (kreuzen sich nicht und berühren sich nicht). Aus diesem Grund kann ein Polygon keine Schnittlinien, Zacken oder Schleifen haben. Dies bedeutet, dass Polygonlöcher als innere Ringe dargestellt werden müssen, anstatt dass der äußere Ring sich selbst berührt (ein sogenanntes "umgekehrtes Loch").
2. Grenzringe kreuzen sich nicht
3. Begrenzungsringe können sich in Punkten berühren, aber nur als Tangente (d. h. nicht in einer Linie)

4. innere Ringe sind im äußeren Ring enthalten
5. das Innere des Polygons ist einfach verbunden (d. h. die Ringe dürfen sich nicht so berühren, dass das Polygon in mehr als einen Teil zerfällt)

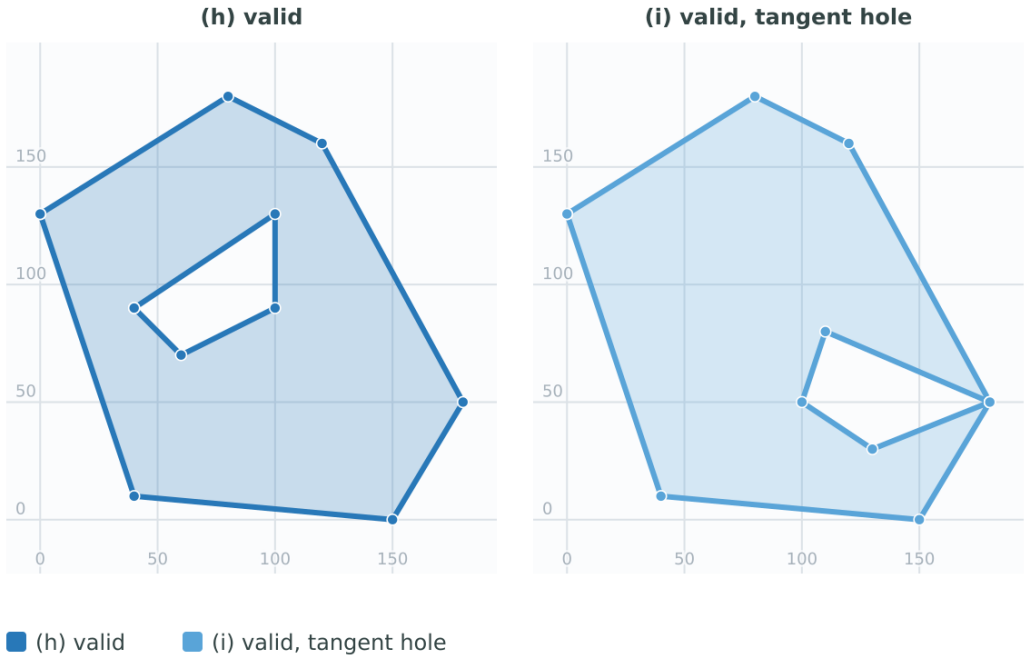
Consecutive repeated points in a polygon boundary ring do not make the ring non-simple. Under the SFS curve model, a curve is interpreted by a continuous parameterization; consecutive equal coordinates are topologically indistinguishable from a single vertex. PostGIS therefore treats them as redundant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

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

Code

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

Figure



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

Code

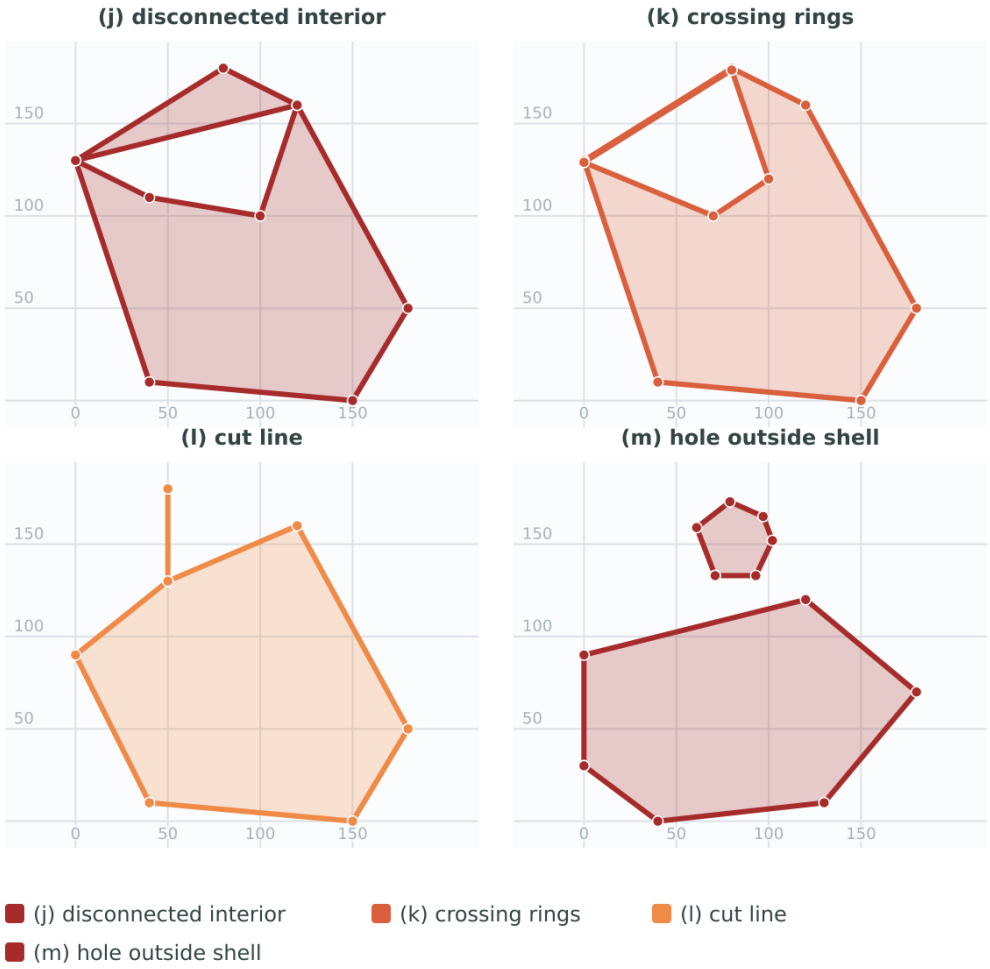
```
WITH cases(label, geom, reason_pattern) AS (VALUES
  ('(j) disconnected interior',
   'POLYGON ((10 140, 90 190, 130 170, 190 60, 160 10, 50 20, 10 140),
             (130 170, 10 140, 50 120, 110 110, 130 170))'::geometry,
```

```
   'Interior is disconnected%'),
  ('(k) crossing rings',
   'POLYGON ((10 140, 90 190, 130 170, 190 60, 160 10, 50 20, 10 140),
             (90 189, 10 139, 80 110, 110 130, 90 189))'::geometry,
   'Self-intersection%'),
  ('(l) cut line',
   'POLYGON ((10 100, 60 140, 60 190, 60 140, 130 170,
              190 60, 160 10, 50 20, 10 100))'::geometry,
   'Ring Self-intersection%'),
  ('(m) hole outside shell',
   'POLYGON ((10 40, 10 100, 130 130, 190 80, 140 20, 50 10, 10 40),
             (81 143, 103 143, 112 162, 107 175, 89 183, 71 169, 81 143))'::geometry,
   'Hole lies outside shell%')
)
SELECT label, geom
FROM cases
WHERE NOT ST_IsValid(geom)
  AND ST_IsValidReason(geom) LIKE reason_pattern
ORDER BY label;
```

Output

```
label          | geom
-----------------------+-----------------------
```

Figure



Ein `MULTIPOLYGON` ist *gültig* wenn:

1. sein Element `POLYGON`s gültig sind
2. die Elemente dürfen sich nicht überschneiden (d. h. ihre Innenräume dürfen sich nicht überschneiden)
3. Elemente berühren sich nur an Punkten (d. h. nicht entlang einer Linie)

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

Code

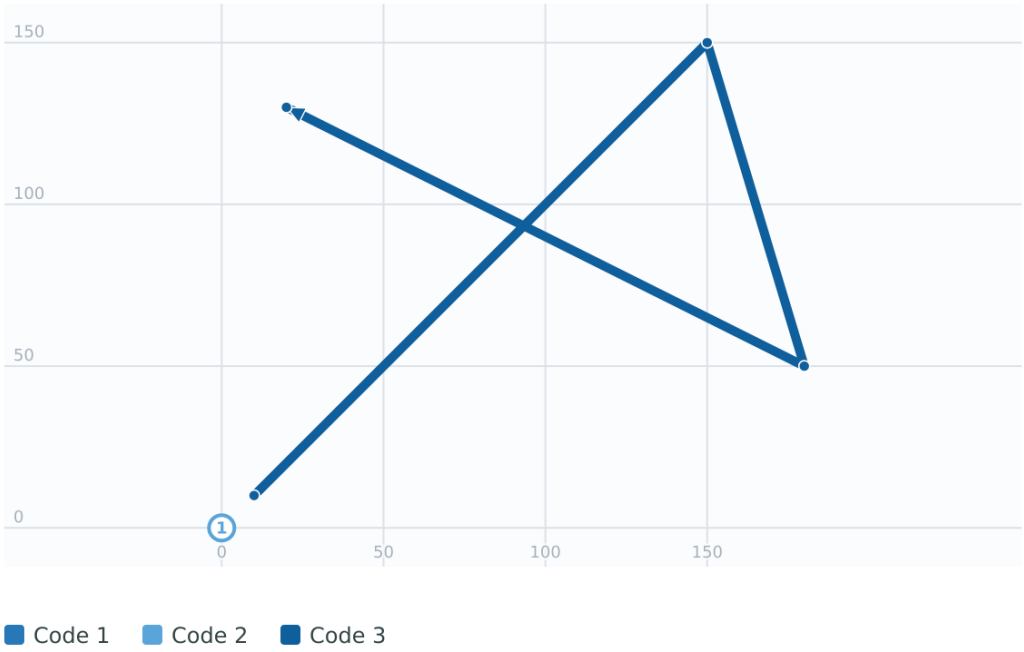
```
WITH cases(label, geom, valid, reason_pattern) AS (VALUES
  ('(n) point touch',
   'MULTIPOLYGON (((10 40, 29 118, 100 75, 180 88, 140 30, 70 10, 10 40)),
                  ((50 170, 100 190, 160 180, 180 140, 180 90, 120 120,
                    30 120, 50 170)))'::geometry,
   true, NULL),
  ('(o) shared edge',
   'MULTIPOLYGON (((10 40, 30 120, 120 120, 180 88, 140 30, 70 10, 10 40)),
                  ((50 170, 100 190, 160 180, 180 140, 180 90, 120 120,
                    30 120, 50 170)))'::geometry,
   false, 'Self-intersection%'),
  ('(p) overlap',
   'MULTIPOLYGON (((10 40, 42 93, 130 150, 180 86, 140 30, 70 10, 10 40)),
```

```
                  ((50 170, 100 190, 138 182, 149 156, 146 137, 100 100,
                    58 140, 50 170)))'::geometry,
   false, 'Self-intersection%')
)
SELECT label, geom
FROM cases
WHERE ST_IsValid(geom) = valid
  AND (valid OR ST_IsValidReason(geom) LIKE reason_pattern)
ORDER BY label;
```

Output

```
label       | geom
------------------+-----------------------
```

Figure



`Die Geometrien POINT` und `MULTIPOINT` haben keine Gültigkeitsregeln.

4.4.3 Verwaltung der Gültigkeit

PostGIS ermöglicht die Erstellung und Speicherung von gültiger und ungültiger Geometrie. Dies ermöglicht es, ungültige Geometrie zu erkennen und zu markieren oder zu korrigieren. Es gibt auch Situationen, in denen die OGC-Gültigkeitsregeln strenger sind als erwünscht (Beispiele hierfür sind Linienstränge mit Nulllänge und Polygone mit invertierten Löchern).

Viele der von PostGIS bereitgestellten Funktionen beruhen auf der Annahme, dass die Geometrieargumente gültig sind. So ist es beispielsweise nicht sinnvoll, die Fläche eines Polygons zu berechnen, in dem ein Loch außerhalb des Polygons definiert ist, oder ein Polygon aus einer nicht einfachen Begrenzungslinie zu konstruieren. Durch die Annahme gültiger geometrischer Eingaben können die Funktionen effizienter arbeiten, da sie nicht auf topologische Korrektheit geprüft werden müssen. (Bemerkenswerte Ausnahmen sind Linien der Länge Null und Polygone mit Invertierungen, die im Allgemeinen korrekt behandelt werden). Außerdem erzeugen die meisten PostGIS-Funktionen eine gültige Geometrieausgabe, wenn die Eingaben gültig sind. Dadurch können PostGIS-Funktionen sicher miteinander verkettet werden.

Wenn Sie beim Aufruf von PostGIS-Funktionen unerwartete Fehlermeldungen erhalten (z. B. "GEOS Intersection() hat einen Fehler ausgelöst!"), sollten Sie sich zunächst vergewissern, dass die Argumente der Funktion gültig sind. Wenn dies nicht der Fall ist, sollten Sie eine der folgenden Techniken anwenden, um sicherzustellen, dass die zu verarbeitenden Daten gültig sind.



Anmerkung
Wenn eine Funktion bei gültigen Eingaben einen Fehler meldet, dann haben Sie möglicherweise einen Fehler in PostGIS oder in einer der verwendeten Bibliotheken gefunden und sollten dies dem PostGIS-Projekt melden. Dasselbe gilt, wenn eine PostGIS-Funktion bei gültiger Eingabe eine ungültige Geometrie zurückgibt.

Um zu prüfen, ob eine Geometrie gültig ist, verwenden Sie die Funktion ST_IsValid:

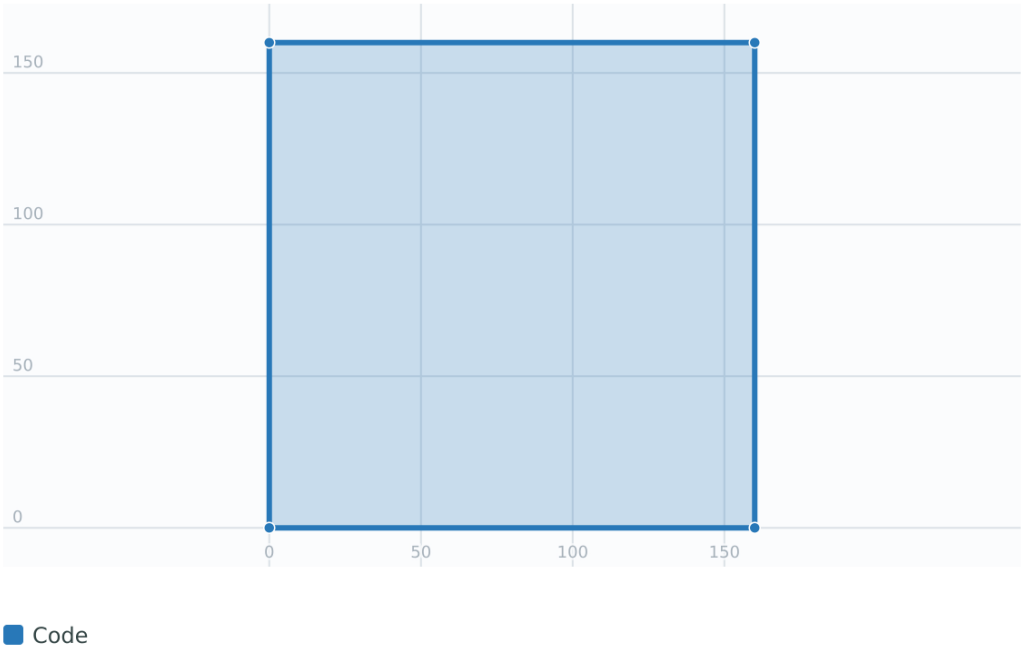
Code

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

Output

```
t
```

Figure



Informationen über die Art und den Ort einer geometrischen Ungültigkeit werden von der Funktion ST_IsValidDetail geliefert:

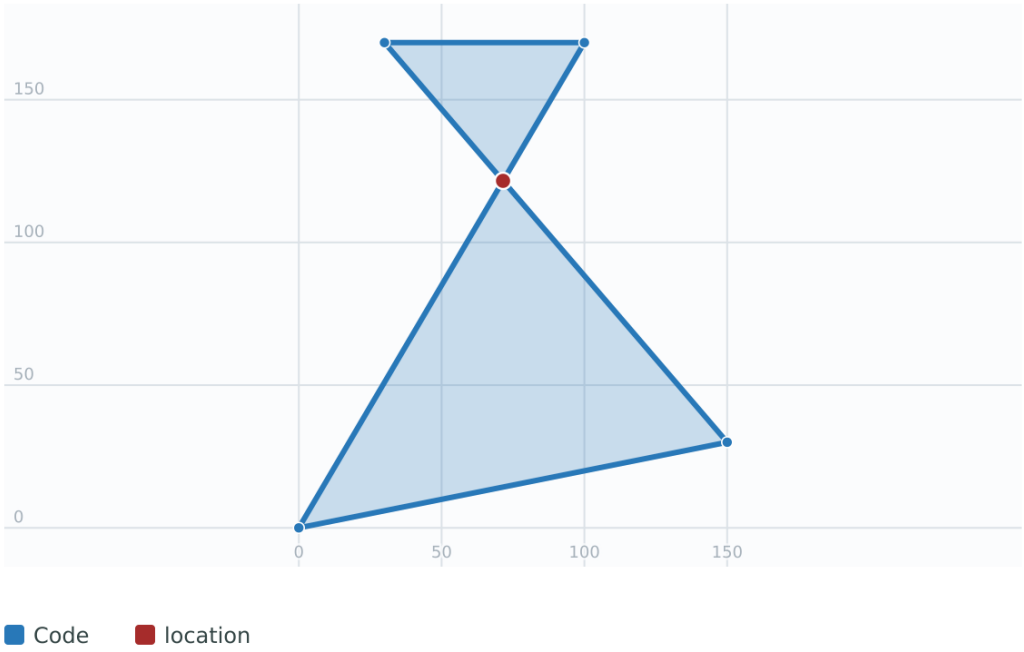
Code

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

Output

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

Figure



In manchen Situationen ist es wünschenswert, ungültige Geometrien automatisch zu korrigieren. Verwenden Sie dazu die Funktion ST_MakeValid. (`ST_MakeValid` ist ein Fall einer räumlichen Funktion, die ungültige Eingaben zulässt!)

Standardmäßig prüft PostGIS beim Laden von Geometrien nicht auf Gültigkeit, da die Gültigkeitsprüfung bei komplexen Geometrien viel CPU-Zeit in Anspruch nehmen kann. Wenn Sie Ihren Datenquellen nicht trauen, können Sie eine Gültigkeitsprüfung für Ihre Tabellen erzwingen, indem Sie eine Prüfbeschränkung hinzufügen:

Code

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

4.5 Räumliche Bezugssysteme

Ein Raumbezugssystem (SRS) (auch Koordinatenreferenzsystem (CRS) genannt) definiert, wie die Geometrie auf Orte auf der Erdoberfläche bezogen wird. Es gibt drei Arten von SRS:

- Ein **geodätisches** SRS verwendet Winkelkoordinaten (Längen- und Breitengrad), die direkt auf der Erdoberfläche abgebildet werden.
- Eine **projizierte** SRS verwendet eine mathematische Projektionstransformation, um die Oberfläche der sphäroidischen Erde auf eine Ebene zu "glätten". Dabei werden Ortskoordinaten so zugewiesen, dass eine direkte Messung von Größen wie Entfernung, Fläche und Winkel möglich ist. Das Koordinatensystem ist kartesisch, d. h. es hat einen definierten Ursprungspunkt und zwei senkrecht zueinander stehende Achsen (in der Regel nach Norden und Osten ausgerichtet). Jede projizierte SRS verwendet eine bestimmte Längeneinheit (in der Regel Meter oder Fuß). Ein projiziertes SRS kann in seinem Anwendungsbereich begrenzt sein, um Verzerrungen zu vermeiden und in die definierten Koordinatengrenzen zu passen.
- Ein **local** SRS ist ein kartesisches Koordinatensystem, das nicht auf die Erdoberfläche referenziert ist. In PostGIS wird dies durch einen SRID-Wert von 0 angegeben.

Es gibt viele verschiedene räumliche Bezugssysteme, die verwendet werden. Die gängigen SRS sind in der European Petroleum Survey Group EPSG-Datenbank standardisiert. Der Einfachheit halber bezieht sich PostGIS (und viele andere raumbezogene Systeme) auf SRS-Definitionen unter Verwendung eines ganzzahligen Bezeichners, der SRID genannt wird.

Eine Geometrie ist mit einem räumlichen Bezugssystem durch ihren SRID-Wert verbunden, auf den über ST_SRID zugegriffen wird. Der SRID für eine Geometrie kann mit ST_SetSRID zugewiesen werden. Einige Geometriekonstruktorfunktionen ermöglichen die Angabe eines SRID (z. B. ST_Point und ST_MakeEnvelope). Das Format EWKT unterstützt SRIDs mit dem Präfix `SRID=n;`.

Räumliche Funktionen, die Paare von Geometrien verarbeiten (z. B. die Funktionen overlay und relationship), setzen voraus, dass die eingegebenen Geometrien im selben räumlichen Bezugssystem (mit demselben SRID) vorliegen. Geometriedaten können mit ST_Transform und ST_TransformPipeline in ein anderes räumliches Bezugssystem transformiert werden. Die von den Funktionen zurückgegebenen Geometrien haben dasselbe SRS wie die Eingabegeometrien.

4.5.1 SPATIAL_REF_SYS Tabelle

Die von PostGIS verwendete Tabelle `SPATIAL_REF_SYS` ist eine OGC-konforme Datenbanktabelle, die die verfügbaren räumlichen Bezugssysteme definiert. Sie enthält die numerischen SRIDs und textuelle Beschreibungen der Koordinatensysteme.

Die Definition der Tabelle `spatial_ref_sys` lautet:

Code

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

Die Spalten sind:

`srid` Ein ganzzahliger Code, der das Raumbezugssystem (SRS) innerhalb der Datenbank eindeutig identifiziert.

`auth_name` Der Name der Norm oder des Normungsgremiums, das für dieses Referenzsystem zitiert wird. Zum Beispiel ist "EPSG" ein gültiger `auth_name`.

`auth_srid` Die ID des räumlichen Bezugssystems, wie von der in `auth_name` genannten Behörde definiert. Im Falle der EPSG ist dies der EPSG-Code.

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

Code

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

Eine Diskussion über SRS WKT findet sich im OGC-Standard Well-known text representation of coordinate reference systems.

`proj4text` PostGIS verwendet die PROJ-Bibliothek, um Koordinatentransformationen zu ermöglichen. Die Spalte `proj4text` enthält die PROJ-Koordinatendefinitionszeichenfolge für eine bestimmte SRID. Zum Beispiel:

Code

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

Weitere Informationen finden Sie auf der PROJ-Website. Die Datei `spatial_ref_sys.sql` enthält sowohl `srtext` als auch `proj4text` Definitionen für alle EPSG-Projektionen.

Beim Abrufen von Definitionen für räumliche Bezugssysteme zur Verwendung in Transformationen verwendet PostGIS die folgende Strategie:

- Wenn `auth_name` und `auth_srid` vorhanden sind (nicht NULL), verwenden Sie den PROJ SRS, der auf diesen Einträgen basiert (falls einer existiert).
- Wenn `srtext` vorhanden ist, erstellen Sie, wenn möglich, ein SRS mit diesem Text.
- Wenn `proj4text` vorhanden ist, erstellen Sie, wenn möglich, eine SRS mit diesem Text.

4.5.2 Benutzerdefinierte räumliche Bezugssysteme

Die PostGIS-Tabelle `spatial_ref_sys` enthält über 3000 der gebräuchlichsten Definitionen für räumliche Bezugssysteme, die von der Projektionsbibliothek PROJ verarbeitet werden. Es gibt jedoch viele Koordinatensysteme, die darin nicht enthalten sind. Sie können der Tabelle SRS-Definitionen hinzufügen, wenn Sie die erforderlichen Informationen über das räumliche Bezugssystem haben. Sie können aber auch Ihr eigenes räumliches Bezugssystem definieren, wenn Sie mit den PROJ-Konstruktionen vertraut sind. Denken Sie daran, dass die meisten räumlichen Bezugssysteme regional sind und keine Bedeutung haben, wenn sie außerhalb der Grenzen verwendet werden, für die sie bestimmt sind.

Eine Ressource zum Auffinden von nicht im Kernsatz definierten räumlichen Bezugssystemen ist 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. For WGS84 longitude/latitude data, the UTM zone is `least(floor((longitude + 180) / 6) + 1, 60)`; the EPSG code is `32600 + zone` in the northern hemisphere or `32700 + zone` in the southern hemisphere.

Die US-Bundesstaaten verwenden State Plane-Raumbezugssysteme (meter- oder feet-basiert) - in der Regel gibt es ein oder zwei pro Staat. Die meisten der Meter-basierten Systeme sind im Kernsatz enthalten, aber viele der Fuß-basierten oder von ESRI erstellten Systeme müssen von spatialreference.org kopiert werden.

Sie können sogar Koordinatensysteme definieren, die nicht auf der Erde basieren, wie z.B. Mars 2000 Dieses Marskoordinatensystem ist nicht planar (es ist in Grad sphäroidisch), aber Sie können es mit dem Typ `Geographie` verwenden, um Längen- und Entfernungsmessungen in Metern statt in Grad zu erhalten.

Hier ein Beispiel für das Laden eines benutzerdefinierten Koordinatensystems unter Verwendung eines nicht zugewiesenen SRID und der PROJ-Definition für eine US-zentrische Lambert-konforme Projektion:

Code

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

4.6 Räumliche Tabellen

4.6.1 Erstellung einer räumlichen Tabelle

Sie können eine Tabelle zur Speicherung von Geometriedaten mit der SQL-Anweisung CREATE TABLE mit einer Spalte vom Typ `Geometrie` erstellen. Das folgende Beispiel erstellt eine Tabelle mit einer Geometriespalte, die 2D (XY) LineStrings im BC-Albers-Koordinatensystem (SRID 3005) speichert:

Code

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

Der Geometrietyp unterstützt zwei optionale **Typmodifikatoren**:

- der **spatial type modifier** schränkt die Art der in der Spalte zulässigen Formen und Abmessungen ein. Der Wert kann einer der unterstützten Geometrie-Subtypen sein (z. B. POINT, LINESTRING, POLYGON, MULTIPOINT, MULTILINESTRING, MULTIPOLYGON, GEOMETRYCOLLECTION, usw.). Der Modifikator unterstützt Einschränkungen der Koordinatendimensionalität durch Hinzufügen von Suffixen: Z, M und ZM. Ein Modifikator 'LINESTRINGM' lässt beispielsweise nur Linienzüge mit drei Dimensionen zu und behandelt die dritte Dimension als Maß. In ähnlicher Weise erfordert 'POINTZM' vierdimensionale (XYZM) Daten.
- Der **SRID-Modifikator** schränkt das Raumbezugssystem SRID auf eine bestimmte Zahl ein. Wird der Modifikator weggelassen, ist das SRID standardmäßig auf 0 gesetzt.

Beispiele für die Erstellung von Tabellen mit Geometriespalten:

Erstellen Sie eine Tabelle, die jede Art von Geometrie mit dem Standard-SRID enthält:

Code

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

Erstellen Sie eine Tabelle mit 2D-Punktgeometrie mit dem Standard-SRID:

Code

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

Erstellen Sie eine Tabelle mit 3D (XYZ) POINTs und einer expliziten SRID von 3005:

Code

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

Erstellen Sie eine Tabelle mit 4D (XYZM) LINESTRING-Geometrie mit dem Standard-SRID:

Code

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

Erstellen Sie eine Tabelle mit 2D POLYGON Geometrie mit dem SRID 4267 (NAD 1927 long lat):

Code

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

Es ist möglich, mehr als eine Geometriespalte in einer Tabelle zu haben. Dies kann bei der Erstellung der Tabelle angegeben werden, oder eine Spalte kann mit der SQL-Anweisung ALTER TABLE hinzugefügt werden. In diesem Beispiel wird eine Spalte hinzugefügt, die 3D LineStrings enthalten kann:

Code

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

4.6.2 GEOMETRY_COLUMNS Ansicht

Die OGC *Simple Features Specification for SQL* definiert die `GEOMETRY_COLUMNS` Metadatentabelle zur Beschreibung der Geometrietabellenstruktur. In PostGIS `ist geometry_columns` ein View, der aus Katalogtabellen des Datenbanksystems gelesen wird. Dadurch wird sichergestellt, dass die räumlichen Metadateninformationen immer mit den aktuell definierten Tabellen und Views konsistent sind. Die Struktur des Views ist:

Code

```
\d geometry_columns
```

Output

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

Die Spalten sind:

f_table_catalog, f_table_schema, f_table_name Der voll qualifizierte Name der Merkmalstabelle, die die Geometriespalte enthält. Es gibt kein PostgreSQL-Analogon für "catalog", daher wird diese Spalte leer gelassen. Für "schema" wird der Name des PostgreSQL-Schemas verwendet (`public` ist der Standard).

f_geometry_column Der Name der Geometriespalte in der Feature-Tabelle.

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

srid Die ID des räumlichen Bezugssystems, das für die Koordinatengeometrie in dieser Tabelle verwendet wird. Es handelt sich um einen Fremdschlüsselverweis auf die Tabelle `spatial_ref_sys` (siehe Abschnitt 4.5.1).

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

4.6.3 Manuelles Registrieren von Geometriespalten

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

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

Code

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

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

Code

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

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

Code

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

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

Code

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

Create a 2D spatial index on the new table.

Code

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

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

Code

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

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

Code

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

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

Code

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

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

Code

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

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

Code

```
\d pois_ny
```

Output

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

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

Beide registrieren sich korrekt in "geometry_columns"

Code

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

Output

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

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

Code

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

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

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

Output

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

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

Code

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

Output

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

4.7 Laden von Geodaten

Sobald Sie eine Geodatentabelle erstellt haben, können Sie Geodaten in die Datenbank hochladen. Es gibt zwei eingebaute Möglichkeiten, Geodaten in eine PostGIS/PostgreSQL-Datenbank zu übertragen: mit formatierten SQL-Anweisungen oder mit dem Shapefile-Loader.

4.7.1 SQL zum Laden von Daten verwenden

Wenn raumbezogene Daten in eine Textdarstellung konvertiert werden können (entweder als WKT oder WKB), dann ist die Verwendung von SQL möglicherweise der einfachste Weg, um Daten in PostGIS zu erhalten. Daten können in großem Umfang in PostGIS/PostgreSQL geladen werden, indem eine Textdatei mit SQL `INSERT` Anweisungen mit dem `psql` SQL-Dienstprogramm geladen wird.

Eine SQL-Ladedatei (z. B.`roads.sql`) könnte wie folgt aussehen:

Code

```
BEGIN;
INSERT INTO roads (road_id, roads_geom, road_name)
  VALUES (1, 'LINESTRING(191232 243118,191108 243242)', 'Jeff Rd');
INSERT INTO roads (road_id, roads_geom, road_name)
  VALUES (2, 'LINESTRING(189141 244158,189265 244817)', 'Geordie Rd');
INSERT INTO roads (road_id, roads_geom, road_name)
  VALUES (3, 'LINESTRING(192783 228138,192612 229814)', 'Paul St');
INSERT INTO roads (road_id, roads_geom, road_name)
  VALUES (4, 'LINESTRING(189412 252431,189631 259122)', 'Graeme Ave');
INSERT INTO roads (road_id, roads_geom, road_name)
  VALUES (5, 'LINESTRING(190131 224148,190871 228134)', 'Phil Tce');
INSERT INTO roads (road_id, roads_geom, road_name)
  VALUES (6, 'LINESTRING(198231 263418,198213 268322)', 'Dave Cres');
COMMIT;
```

Die SQL-Datei kann mit `psql` in PostgreSQL geladen werden:

Code

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

4.7.2 Verwendung des Shapefile Loaders

Der `shp2pgsql` Datenlader konvertiert Shapefiles in SQL, das zum Einfügen in eine PostGIS/PostgreSQL-Datenbank geeignet ist, entweder im Geometrie- oder im Geografieformat. Der Lader hat mehrere Betriebsmodi, die über Kommandozeilenflags ausgewählt werden können.

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

Es gibt auch eine grafische Schnittstelle `shp2pgsql-gui` mit den meisten Optionen des Befehlszeilen-Laders. Dies kann für einmaliges, nicht skriptgesteuertes Laden oder für PostGIS-Neulinge einfacher zu verwenden sein. Es kann auch als Plugin für PgAdminIII konfiguriert werden.

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

-c Erzeugt eine neue Tabelle und füllt sie aus dem Shapefile. *Dies ist der Standardmodus.*

-a Fügt Daten aus dem Shapefile in die Datenbanktabelle ein. Beachten Sie, dass die Dateien die gleichen Attribute und Datentypen haben müssen, um mit dieser Option mehrere Dateien zu laden.

-d Löscht die Datenbanktabelle, bevor eine neue Tabelle mit den Daten im Shapefile erstellt wird.

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

-? Zeigt die Hilfe an.

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

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

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

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

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

-I Einen GIST Index auf die Geometriespalte anlegen.

-u Create the target table as `UNLOGGED`. This can speed up loading transient staging data, but PostgreSQL does not preserve unlogged table contents after a crash or unclean shutdown.

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

```
COLUMNNAME DBFFIELD1
AVERYLONGCOLUMNNAME DBFFIELD2
```

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

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

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

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

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

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

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

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

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

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

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

-Z Wenn dieses Flag verwendet wird, verhindert es die Erzeugung von `ANALYZE` Anweisungen. Ohne das Flag -Z (Standardverhalten) werden die Anweisungen `ANALYZE` erzeugt.

Eine Beispielsitzung, bei der der Loader eine Eingabedatei erstellt und diese lädt, könnte folgendermaßen aussehen:

Code

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

Eine Konvertierung und das Laden können in einem Schritt über UNIX-Pipes durchgeführt werden:

Code

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

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

Code

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

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

4.8 Extrahieren von Geodaten

Geodaten können entweder mit SQL oder mit dem Shapefile-Dumper aus der Datenbank extrahiert werden. Im Abschnitt über SQL werden einige der Funktionen vorgestellt, die für Vergleiche und Abfragen von räumlichen Tabellen zur Verfügung stehen.

4.8.1 SQL zum Extrahieren von Daten verwenden

Die einfachste Möglichkeit, Geodaten aus der Datenbank zu extrahieren, ist die Verwendung einer SQL `SELECT` Abfrage, um den zu extrahierenden Datensatz zu definieren und die resultierenden Spalten in eine parsbare Textdatei zu übertragen:

Code

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

Output

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

Es kann vorkommen, dass eine Art von Einschränkung erforderlich ist, um die Anzahl der zurückgegebenen Datensätze zu verringern. Im Falle von attributbasierten Einschränkungen verwenden Sie dieselbe SQL-Syntax wie bei einer nicht räumlichen Tabelle. Bei räumlichen Einschränkungen sind die folgenden Funktionen nützlich:

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

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

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

Code

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

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

Überprüfung ob einige der Strassen in die Polygonfläche hineinreichen:

Code

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

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

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

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

Code

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

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

4.8.2 Verwendung des Shapefile-Dumpers

Der `pgsql2shp` Tabellendumper verbindet sich mit der Datenbank und konvertiert eine Tabelle (möglicherweise durch eine Abfrage definiert) in eine Shape-Datei. Die grundlegende Syntax lautet:

Code

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

Code

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

Optionen auf der Befehlszeile:

-f <filename> Ausgabe in eine bestimmte Datei.

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

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

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

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

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

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

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

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

4.9 Räumliche Indizes

Räumliche Indizes ermöglichen die Verwendung einer räumlichen Datenbank für große Datensätze. Ohne Indizierung erfordert die Suche nach Merkmalen ein sequentielles Durchsuchen aller Datensätze in der Datenbank. Die Indizierung beschleunigt die Suche, indem die Daten in einer Struktur organisiert werden, die schnell durchlaufen werden kann, um passende Datensätze zu finden.

Die B-Baum-Index-Methode, die üblicherweise für Attributdaten verwendet wird, ist für räumliche Daten nicht sehr nützlich, da sie nur die Speicherung und Abfrage von Daten in einer einzigen Dimension unterstützt. Daten wie Geometrie (die 2 oder mehr Dimensionen haben) erfordern eine Indexmethode, die Bereichsabfragen über alle Datendimensionen unterstützt. Einer der Hauptvorteile von PostgreSQL für den Umgang mit räumlichen Daten ist, dass es mehrere Arten von Indexmethoden bietet, die gut für mehrdimensionale Daten funktionieren: GiST-, BRIN- und SP-GiST-Indizes.

- **GiST (Generalized Search Tree)** Indizes unterteilen Daten in "Dinge, die auf einer Seite liegen", "Dinge, die sich überschneiden", "Dinge, die im Inneren liegen" und können für eine Vielzahl von Datentypen verwendet werden, einschließlich GIS-Daten. PostGIS verwendet einen R-Tree-Index, der auf GiST aufbaut, um räumliche Daten zu indizieren. GiST ist die am weitesten verbreitete und vielseitigste räumliche Indexmethode und bietet eine sehr gute Abfrageleistung.
- **BRIN (Block Range Index)** Indizes fassen die räumliche Ausdehnung von Bereichen von Tabellendatensätzen zusammen. Die Suche erfolgt über einen Scan der Bereiche. BRIN ist nur für einige Arten von Daten geeignet (räumlich sortiert, mit seltenen oder keinen Aktualisierungen). Es ermöglicht jedoch eine wesentlich schnellere Indexerstellung und eine wesentlich geringere Indexgröße.

- **SP-GiST (Space-Partitioned Generalized Search Tree)** ist eine generische Indexmethode, die partitionierte Suchbäume wie Quad-Bäume, k-d-Bäume und Radix-Bäume (Tries) unterstützt.

Räumliche Indizes speichern nur die Bounding Box von Geometrien. Räumliche Abfragen verwenden den Index als **primären Filter**, um schnell einen Satz von Geometrien zu ermitteln, die möglicherweise der Abfragebedingung entsprechen. Die meisten räumlichen Abfragen erfordern einen **sekundären Filter**, der eine räumliche Prädikatsfunktion verwendet, um eine spezifischere räumliche Bedingung zu testen. Weitere Informationen über Abfragen mit räumlichen Prädikaten finden Sie unter Abschnitt 5.2.

Siehe auch den PostGIS Workshop Abschnitt über räumliche Indizes, und das PostgreSQL Handbuch.

4.9.1 GiST-Indizes

GiST steht für "Generalized Search Tree" (verallgemeinerter Suchbaum) und ist eine generische Form der Indexierung für mehrdimensionale Daten. PostGIS verwendet einen R-Tree-Index, der auf GiST aufbaut, um räumliche Daten zu indizieren. GiST ist die am häufigsten verwendete und vielseitigste räumliche Indexmethode und bietet eine sehr gute Abfrageleistung. Andere GiST-Implementierungen werden verwendet, um die Suche in allen Arten von unregelmäßigen Datenstrukturen (Integer-Arrays, Spektraldaten usw.) zu beschleunigen, die für eine normale B-Tree-Indexierung nicht geeignet sind. Weitere Informationen finden Sie im PostgreSQL-Handbuch.

Sobald eine Geodatentabelle einige tausend Zeilen überschreitet, sollten Sie einen Index erstellen, um die räumliche Suche in den Daten zu beschleunigen (es sei denn, alle Ihre Suchvorgänge basieren auf Attributen; in diesem Fall sollten Sie einen normalen Index für die Attributfelder erstellen).

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

Code

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

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

Code

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

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

Code

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

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

Code

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

4.9.2 BRIN Indizes

BRIN stands for "Block Range Index". It is a general-purpose index method provided by PostgreSQL. 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.

Sobald eine räumliche Tabelle einige tausend Zeilen überschreitet, sollten Sie einen Index erstellen, um die räumliche Suche in den Daten zu beschleunigen. GiST-Indizes sind sehr leistungsfähig, solange ihre Größe den für die Datenbank verfügbaren Arbeitsspeicher nicht übersteigt und Sie sich die Größe des Indexspeichers und die Kosten der Indexaktualisierung beim Schreiben leisten können. Andernfalls kann für sehr große Tabellen der BRIN-Index als Alternative in Betracht gezogen werden.

Ein BRIN-Index speichert die Bounding Box, die alle in den Zeilen enthaltenen Geometrien in einem zusammenhängenden Satz von Tabellenblöcken einschließt, genannt *block range*. Bei der Ausführung einer Abfrage unter Verwendung des Indexes werden die Blockbereiche gescannt, um diejenigen zu finden, die den Abfragebereich überschneiden. Dies ist nur dann effizient, wenn die Daten physisch so geordnet sind, dass sich die Begrenzungsrahmen für die Blockbereiche minimal überschneiden (und sich im Idealfall gegenseitig ausschließen). Der daraus resultierende Index ist sehr klein, aber in der Regel weniger performant beim Lesen als ein GiST-Index über dieselben Daten.

Die Erstellung eines BRIN-Index ist wesentlich weniger rechenintensiv als die Erstellung eines GiST-Index. In der Regel ist ein BRIN-Index zehnmal schneller zu erstellen als ein GiST-Index für dieselben Daten. Und da ein BRIN-Index nur eine Bounding Box für jeden Bereich von Tabellenblöcken speichert, benötigt er in der Regel bis zu tausendmal weniger Plattenplatz als ein GiST-Index.

Sie können die Anzahl der Blöcke wählen, die in einem Bereich zusammengefasst werden sollen. Wenn Sie diese Zahl verringern, wird der Index größer, bietet aber wahrscheinlich eine bessere Leistung.

Damit BRIN effektiv ist, sollten die Tabellendaten in einer physischen Reihenfolge gespeichert werden, die die Überlappung der Blöcke minimiert. Es kann sein, daß die Daten bereits entsprechend sortiert sind (z.B. wenn sie aus einem anderen Datensatz geladen wurden, der bereits räumlich sortiert ist). Andernfalls kann dies durch Sortieren der Daten nach einem eindimensionalen räumlichen Schlüssel erreicht werden. Eine Möglichkeit, dies zu tun, besteht darin, eine neue Tabelle zu erstellen, die nach den Geometriewerten sortiert ist (was in neueren PostGIS-Versionen eine effiziente Hilbert-Kurvenordnung verwendet):

Code

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

Alternativ können die Daten an Ort und Stelle sortiert werden, indem ein GeoHash als (temporärer) Index verwendet und anhand dieses Indexes eine Gruppierung vorgenommen wird:

Code

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

Die Syntax für die Erstellung eines BRIN-Index für eine Spalte `geometry` lautet:

Code

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

Mit der obigen Syntax wird ein 2D-Index erstellt. Um einen 3D-dimensionalen Index zu erstellen, verwenden Sie diese Syntax:

Code

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

Sie können auch einen 4D-dimensionalen Index erhalten, indem Sie die 4D-Operatorklasse verwenden:

Code

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

Die obigen Befehle verwenden die Standardanzahl von Blöcken in einem Bereich, die 128 beträgt. Um die Anzahl der zusammenzufassenden Blöcke in einem Bereich anzugeben, verwenden Sie folgende Syntax

Code

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

Beachten Sie, dass ein BRIN-Index nur einen Indexeintrag für eine große Anzahl von Zeilen speichert. Wenn Ihre Tabelle Geometrien mit einer gemischten Anzahl von Dimensionen speichert, ist es wahrscheinlich, dass der resultierende Index eine schlechte Leistung hat. Sie können diese Leistungseinbußen vermeiden, indem Sie die Operatorklasse mit der geringsten Anzahl von Dimensionen der gespeicherten Geometrien wählen

Der Datentyp `geography` wird für die BRIN-Indizierung unterstützt. Die Syntax für die Erstellung eines BRIN-Index für eine Geografiespalte lautet:

Code

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

Mit der obigen Syntax wird ein 2D-Index für geografische Objekte auf dem Sphäroid erstellt.

Derzeit wird nur "Einschlussunterstützung" geboten, d. h. nur die Operatoren `&&` , ~ und `@` können für die 2D-Fälle verwendet werden (sowohl für die Geometrie als auch für die Geographie), und nur der Operator `&&&` für 3D-Geometrien. Derzeit gibt es keine Unterstützung für kNN-Suchen.

Ein wichtiger Unterschied zwischen BRIN und anderen Indexarten ist, daß die Datenbank den Index nicht dynamisch pflegt. Änderungen an räumlichen Daten in der Tabelle werden einfach an das Ende des Indexes angehängt. Dies führt dazu, daß die Leistung der Indexsuche mit der Zeit abnimmt. Der Index kann durch Ausführen eines `VACUUM` oder durch Verwendung einer speziellen Funktion `brin_summarize_new_values(regclass)` aktualisiert werden. Aus diesem Grund eignet sich BRIN vor allem für Daten, die nur gelesen werden oder sich nur selten ändern. Weitere Informationen finden Sie im Handbuch.

Zusammenfassung der Verwendung von BRIN für räumliche Daten:

- Der Indexaufbau ist sehr schnell und die Indexgröße ist sehr klein.
- Die Indexabfragezeit ist langsamer als bei GiST, kann aber immer noch sehr akzeptabel sein.
- Erfordert, dass die Tabellendaten in einer räumlichen Reihenfolge sortiert werden.
- Erfordert manuelle Indexpflege.
- Am besten geeignet für sehr große Tabellen mit geringer oder gar keiner Überschneidung (z. B. Punkte), die statisch sind oder sich nur selten ändern.
- Effektiver für Abfragen, die eine relativ große Anzahl von Datensätzen zurückgeben.

4.9.3 SP-GiST Indizes

SP-GiST steht für "Space-Partitioned Generalized Search Tree" und ist eine generische Form der Indizierung für mehrdimensionale Datentypen, die partitionierte Suchbäume wie Quad-Trees, K-D-Trees und Radix-Trees unterstützt (Tries). Das gemeinsame Merkmal dieser Datenstrukturen ist, dass sie den Suchraum wiederholt in Partitionen unterteilen, die nicht gleich groß sein müssen. Neben der räumlichen Indizierung wird SP-GiST zur Beschleunigung von Suchvorgängen bei vielen Arten von Daten verwendet, wie z. B. Telefon-Routing, IP-Routing, Teilstringsuche usw. Weitere Informationen finden Sie im PostgreSQL-Handbuch.

Wie GiST-Indizes sind auch SP-GiST-Indizes verlustbehaftet, da sie die Bounding Box speichern, die räumliche Objekte umschließt. SP-GiST-Indizes können als eine Alternative zu GiST-Indizes betrachtet werden.

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

Code

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

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

Code

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

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

Code

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

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

Code

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

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

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

kNN Suche wird zurzeit nicht unterstützt.

4.9.4 Abstimmung der Indexverwendung

Normalerweise beschleunigen Indizes unsichtbar den Datenzugriff: Sobald ein Index erstellt ist, entscheidet der PostgreSQL-Abfrageplaner automatisch, wann er verwendet wird, um die Abfrageleistung zu verbessern. Es gibt jedoch einige Situationen, in denen der Planer nicht entscheidet, vorhandene Indizes zu verwenden, so dass Abfragen am Ende langsame sequenzielle Scans anstelle eines räumlichen Indexes verwenden.

Wenn Sie feststellen, dass Ihre räumlichen Indizes nicht verwendet werden, können Sie einige Dinge tun:

- Prüfen Sie den Abfrageplan und stellen Sie sicher, dass Ihre Abfrage tatsächlich das berechnet, was Sie brauchen. Ein fehlerhafter JOIN, der entweder vergessen wurde oder sich auf die falsche Tabelle bezieht, kann unerwartet mehrfach Tabellendatensätze abrufen. Um den Abfrageplan zu erhalten, führen Sie ihn mit `EXPLAIN` vor der Abfrage aus.
- Stellen Sie sicher, dass Statistiken über die Anzahl und die Verteilungen der Werte in einer Tabelle gesammelt werden, um dem Abfrageplaner bessere Informationen für Entscheidungen über die Indexnutzung zu liefern. **VACUUM ANALYZE** berechnet beides.

  Sie sollten Ihre Datenbanken auf jeden Fall regelmäßig leeren. Viele PostgreSQL-DBAs lassen **VACUUM** regelmäßig als Cron-Job außerhalb der Spitzenzeiten laufen.
- Wenn das Vakuumieren nicht hilft, können Sie den Planer vorübergehend zwingen, die Indexinformationen zu verwenden, indem Sie den Befehl **SET ENABLE_SEQSCAN TO OFF;** verwenden. Auf diese Weise können Sie überprüfen, ob der Planer überhaupt in der Lage ist, einen indexbeschleunigten Abfrageplan für Ihre Abfrage zu erstellen. Sie sollten diesen Befehl nur zur Fehlersuche verwenden; im Allgemeinen weiß der Planer besser als Sie, wann Indizes verwendet werden sollten. Nachdem Sie Ihre Abfrage ausgeführt haben, vergessen Sie nicht, **SET ENABLE_SEQSCAN TO ON;** auszuführen, damit der Planer bei anderen Abfragen normal arbeitet.

- Wenn **SET ENABLE_SEQSCAN TO OFF;** Ihre Abfrage schneller laufen lässt, ist Ihr Postgres wahrscheinlich nicht auf Ihre Hardware abgestimmt. Wenn Sie feststellen, dass der Planer die Kosten für sequentielle und Index-Scans falsch einschätzt, versuchen Sie, den Wert von `RANDOM_PAGE_COST` in `postgresql.conf` zu reduzieren, oder verwenden Sie **SET RANDOM_PAGE_COST TO 1.1;**. Der Standardwert für `RANDOM_PAGE_COST` ist 4.0. Versuchen Sie, ihn auf 1,1 (für SSD) oder 2,0 (für schnelle Magnetplatten) zu setzen. Wenn Sie den Wert verringern, wird der Planer mit größerer Wahrscheinlichkeit Index-Scans verwenden.
- Wenn **SET ENABLE_SEQSCAN TO OFF;** Ihrer Abfrage nicht hilft, verwendet die Abfrage möglicherweise ein SQL-Konstrukt, das der Postgres-Planer noch nicht optimieren kann. Es kann möglich sein, die Abfrage so umzuschreiben, dass der Planner sie verarbeiten kann. Zum Beispiel kann eine Subquery mit einem Inline-SELECT keinen effizienten Plan erzeugen, kann aber möglicherweise mit einem LATERAL JOIN umgeschrieben werden.

Weitere Informationen finden Sie im Postgres-Handbuch im Abschnitt Query Planning.

Kapitel 5

Räumliche Abfrage

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

5.1 Räumliche Beziehungen feststellen

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

5.1.1 Dimensionell erweitertes 9-Schnitte-Modell

Laut OpenGIS Simple Features Implementation Specification for SQL besteht der grundlegende Ansatz für den Vergleich zweier Geometrien darin, paarweise Tests der Schnittpunkte zwischen den Innen-, Rand- und Außenbereichen der beiden Geometrien durchzuführen und die Beziehung zwischen den beiden Geometrien auf der Grundlage der Einträge in der resultierenden 'Schnittpunkt'-Matrix zu klassifizieren".

In der Theorie der Punktmengentopologie werden die Punkte in einer Geometrie, die in den 2-dimensionalen Raum eingebettet ist, in drei Gruppen eingeteilt:

Grenze

Die Begrenzung einer Geometrie ist die Menge der Geometrien der nächstniedrigeren Dimension. Für `POINT`s, die eine Dimension von 0 haben, ist die Begrenzung die leere Menge. Die Begrenzung eines `LINESTRING` sind die beiden Endpunkte. Für `POLYGON`s ist die Begrenzung das Liniennetz der äußeren und inneren Ringe.

Innenbereich

Das Innere einer Geometrie sind die Punkte einer Geometrie, die nicht in der Begrenzung liegen. Für `POINT`s ist das Innere der Punkt selbst. Das Innere eines `LINESTRING` ist die Menge der Punkte zwischen den Endpunkten. Für `POLYGON`s ist das Innere die Fläche innerhalb des Polygons.

Äußeres

Das Äußere einer Geometrie ist der Rest des Raums, in den die Geometrie eingebettet ist; mit anderen Worten, alle Punkte, die sich nicht im Inneren oder auf dem Rand der Geometrie befinden. Es handelt sich um eine 2-dimensionale, nicht geschlossene Fläche.

Das Dimensionally Extended 9-Intersection Model (DE-9IM) beschreibt die räumliche Beziehung zwischen zwei Geometrien durch Angabe der Dimensionen der 9 Schnittpunkte zwischen den oben genannten Mengen für jede Geometrie. Die Schnittpunktdimensionen können formal in einer 3x3 **Schnittpunktmatrix** dargestellt werden.

Für eine Geometrie *g* werden *Interior*, *Boundary*, und *Exterior* mit den Bezeichnungen *I(g)*, *B(g)*, und *E(g)* bezeichnet. Außerdem bezeichnet *dim(s)* die Dimension einer Menge *s* mit dem Bereich `{0,1,2,F}`:

- `0` => Punkt
- `1` => Zeile
- `2` => Fläche
- `F` => leere Menge

Unter Verwendung dieser Notation lautet die Schnittpunktmatrix für zwei Geometrien *a* und *b*:

	Innenbereich	**Grenze**	**Äußeres**
Innenbereich	*dim(I(a) ∩ I(b))*	*dim(I(a) ∩ B(b))*	*dim(I(a) ∩ E(b))*
Grenze	*dim(B(a) ∩ I(b))*	*dim(B(a) ∩ B(b))*	*dim(B(a) ∩ E(b))*
Äußeres	*dim(E(a) ∩ I(b))*	*dim(E(a) ∩ B(b))*	*dim(E(a) ∩ E(b))*

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

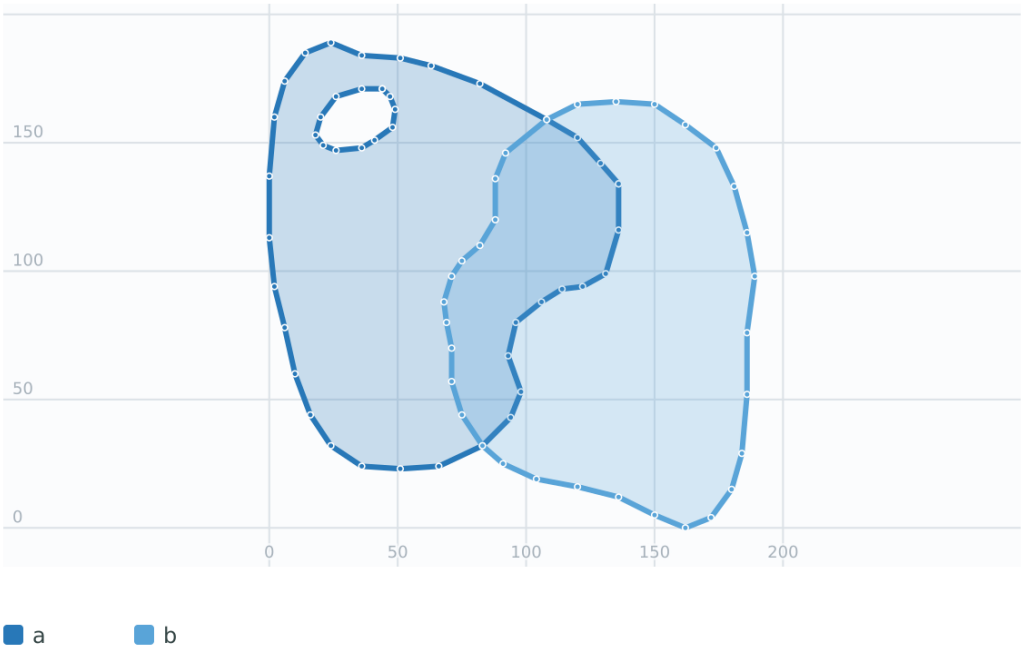
Code

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

Output

```
matrix   | matches_overlap_pattern | overlaps
-----------+-------------------------+----------
 212101212 | t                       | t
(1 row)
```

Figure



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

Code

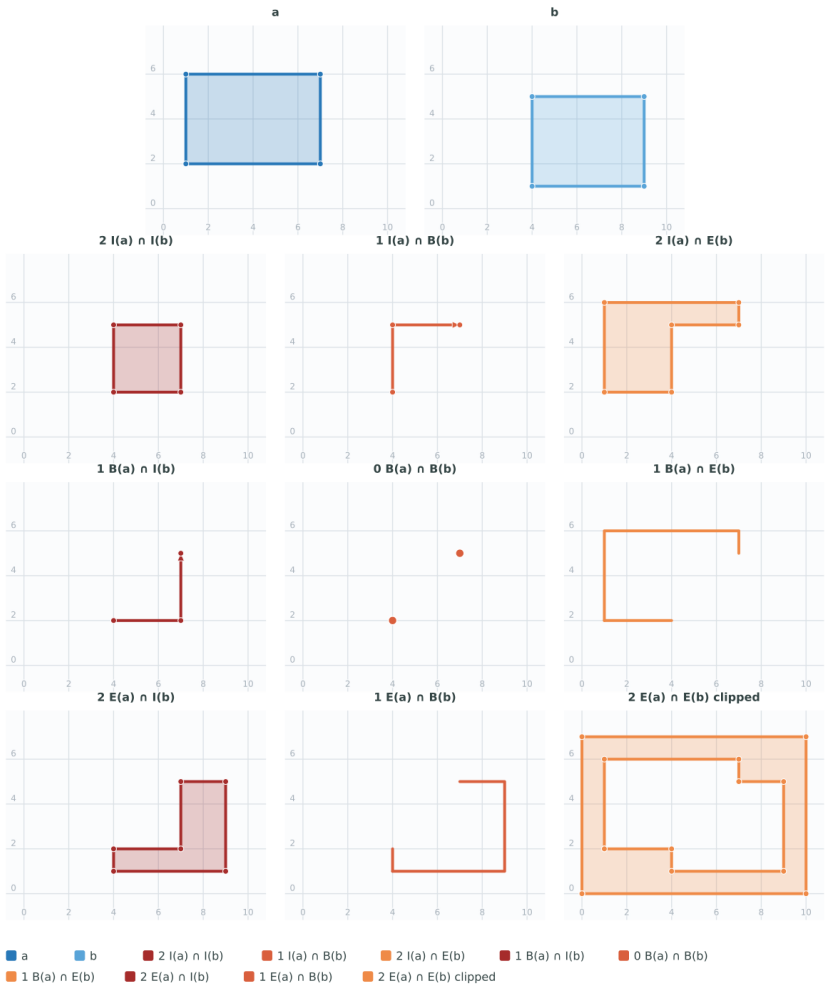
```
WITH example AS (
  SELECT
    ST_MakeEnvelope(0, 0, 6, 4) AS a,
    ST_MakeEnvelope(3, -1, 8, 3) AS b,
    ST_MakeEnvelope(-1, -2, 9, 5) AS viewport
), parts AS (
  SELECT
    a,
    b,
    ST_Boundary(a) AS ba,
    ST_Boundary(b) AS bb,
    viewport
  FROM example
)
SELECT
  ST_Relate(a, b) AS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

```
  ))) AS "1 E(a) ∩ B(b)",
  ST_AsText(ST_Normalize(ST_Difference(viewport, ST_Union(a, b))))
    AS "2 E(a) ∩ E(b) clipped"
FROM parts;
```

Output

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

Figure



Von links nach rechts und von oben nach unten gelesen, wird die Kreuzungsmatrix als Textstring '**212101212**' dargestellt.

Weitere Informationen finden Sie unter:

- OpenGIS Simple Features Implementation Specification for SQL (Version 1.1, Abschnitt 2.1.13.2)
- Wikipedia: Dimensionserweitertes Neuner-Intersektions-Modell (DE-9IM)
- GeoTools: Punktmengentheorie und die DE-9IM-Matrix

5.1.2 Benannte räumliche Beziehungen

Um die Bestimmung allgemeiner räumlicher Beziehungen zu erleichtern, definiert das OGC SFS eine Reihe von *genannten räumlichen Beziehungsprädikaten*. PostGIS stellt diese als die Funktionen ST_Contains, ST_Crosses, ST_Disjoint, ST_Equals, ST_Intersects, ST_Overlaps, ST_Touches, ST_Within zur Verfügung. Es definiert auch die Nicht-Standard-Beziehungsprädikate ST_Covers, ST_CoveredBy und ST_ContainsProperly.

Räumliche Prädikate werden normalerweise als Bedingungen in SQL `WHERE` oder `JOIN` Klauseln verwendet. Die benannten räumlichen Prädikate verwenden automatisch einen räumlichen Index, wenn einer vorhanden ist, so dass es nicht notwendig ist, auch den Bounding-Box-Operator `&&` zu verwenden. Zum Beispiel:

Code

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

Weitere Einzelheiten und Abbildungen finden Sie im PostGIS Workshop.

5.1.3 Allgemeine räumliche Beziehungen

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

Um eine bestimmte räumliche Beziehung zu testen, wird ein **Kreuzungsmatrixmuster** verwendet. Dabei handelt es sich um die Matrixdarstellung, die um die zusätzlichen Symbole `{T,*}` erweitert wurde:

- `T` => Schnittmenge ist nicht leer, d. h. sie liegt in `{0,1,2}`
- `*` => ist mir egal

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

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

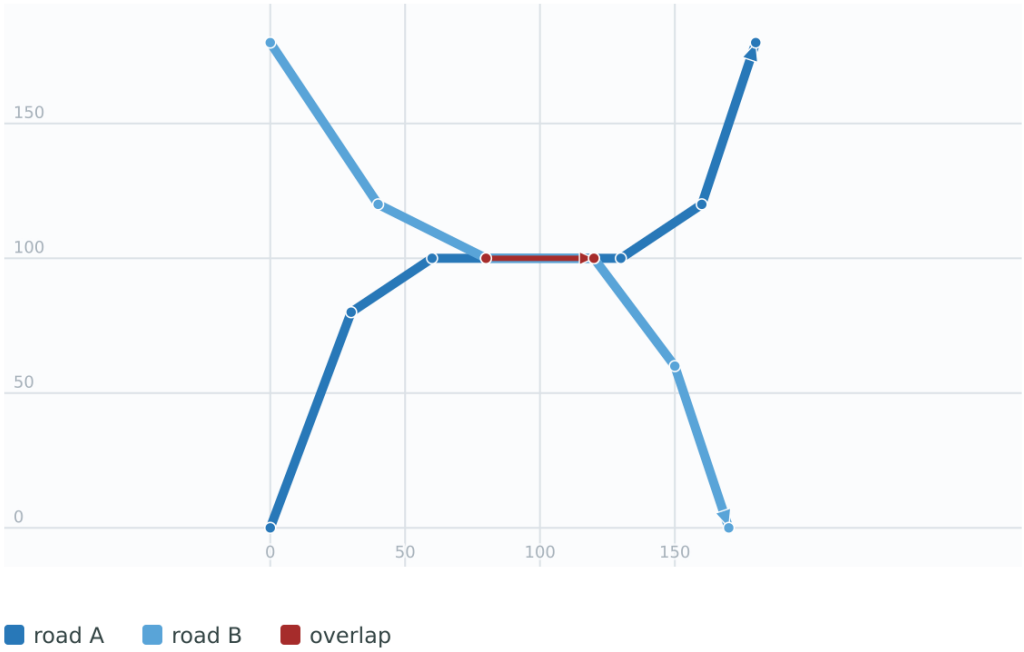
Code

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

Output

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

Figure



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

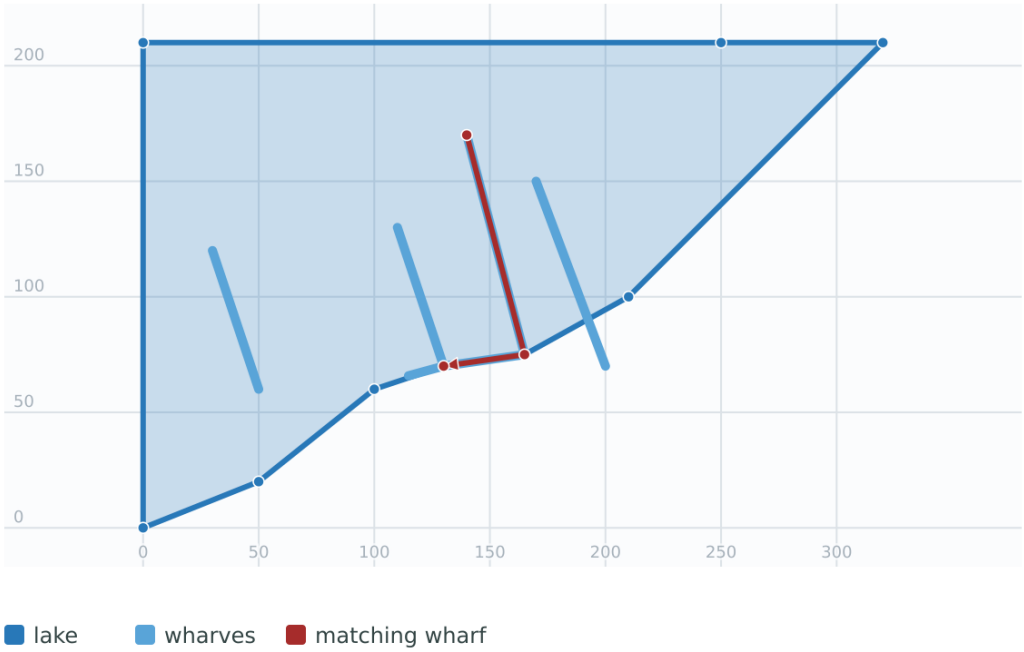
Code

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

Output

```
wharf |           matching wharf           |  matrix
-------+------------------------------------+-----------
     1 | LINESTRING(120 180,145 85,110 80) | 102101FF2
(1 row)
```

Figure



5.2 Räumliche Indizes verwenden

Bei der Erstellung von Abfragen mit räumlichen Bedingungen ist es für eine optimale Leistung wichtig, dass ein räumlicher Index verwendet wird, sofern ein solcher existiert (siehe Abschnitt 4.9). Zu diesem Zweck muss ein räumlicher Operator oder eine indexfähige Funktion in einer `WHERE` oder `ON` Klausel der Abfrage verwendet werden.

Zu den räumlichen Operatoren gehören die Bounding-Box-Operatoren (von denen der am häufigsten verwendete && ist; eine vollständige Liste finden Sie unter Abschnitt 7.10.1) und die Abstandsoperatoren, die bei Abfragen nach dem nächsten Nachbarn verwendet werden (der am häufigsten verwendete ist <->; eine vollständige Liste finden Sie unter Abschnitt 7.10.2).

Indexgestützte Funktionen fügen der räumlichen Bedingung automatisch einen Begrenzungsrahmenoperator hinzu. Zu den indexbasierten Funktionen gehören die benannten räumlichen Beziehungsprädikate ST_Contains, ST_ContainsProperly, ST_CoveredBy, ST_Covers, ST_Crosses, ST_Intersects, ST_Overlaps, ST_Touches, ST_Within, ST_Within und ST_3DIntersects sowie die Entfernungsprädikate ST_DWithin, ST_DFullyWithin, ST_3DDFullyWithin und ST_3DDWithin).

Funktionen wie ST_Distance *und nicht* verwenden Indizes, um ihren Betrieb zu optimieren. Die folgende Abfrage wäre zum Beispiel bei einer großen Tabelle ziemlich langsam:

Code

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

Diese Abfrage wählt alle Geometrien in `geom_table` aus, die innerhalb von 100 Einheiten des Punktes (100000, 200000) liegen. Sie ist langsam, weil sie den Abstand zwischen jedem Punkt in der Tabelle und dem angegebenen Punkt berechnet, d. h. eine `ST_Distance()` Berechnung wird für **jede** Zeile in der Tabelle berechnet.

Die Anzahl der zu verarbeitenden Zeilen kann durch die Verwendung der indexbasierten Funktion ST_DWithin erheblich reduziert werden:

Code

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

Diese Abfrage wählt dieselben Geometrien aus, allerdings auf effizientere Weise. Dies wird durch `ST_DWithin()` ermöglicht, die den `&&` Operator intern auf einem erweiterten Begrenzungsrahmen der Abfragegeometrie verwendet. Wenn es einen räumlichen Index auf `geom` gibt, erkennt der Abfrageplaner, dass er den Index verwenden kann, um die Anzahl der gescannten Zeilen zu reduzieren, bevor die Entfernung berechnet wird. Der räumliche Index ermöglicht es, nur Datensätze mit Geometrien abzurufen, deren Begrenzungsrahmen die erweiterte Ausdehnung überlappen und die daher innerhalb der erforderlichen Entfernung liegen könnten. Der tatsächliche Abstand wird dann berechnet, um zu bestätigen, ob der Datensatz in die Ergebnismenge aufgenommen werden soll.

Weitere Informationen und Beispiele finden Sie im PostGIS Workshop.

5.3 Beispiele für Spatial SQL

Die Beispiele in diesem Abschnitt verwenden eine Tabelle mit linearen Straßen und eine Tabelle mit polygonalen Gemeindegrenzen. Die Definition der Tabelle `bc_roads` lautet:

Output

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

Die Definition der Tabelle `bc_municipality` lautet:

Output

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

1. *Wie lang ist die Gesamtlänge aller Straßen, ausgedrückt in Kilometern?*

   Sie können diese Frage mit einem sehr einfachen SQL-Programm beantworten:

   Code

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

   Output

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

2. *Wie groß ist die Stadt Prince George (in Hektar)?*

   Diese Abfrage kombiniert eine Attributbedingung (für den Gemeindenamen) mit einer räumlichen Berechnung (der Polygonfläche):

Code

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

Output

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

3. *Welche ist die flächenmäßig größte Gemeinde der Provinz?*

   Diese Abfrage verwendet ein räumliches Maß als Ordnungswert. Es gibt mehrere Möglichkeiten, dieses Problem anzugehen, aber die effizienteste ist die folgende:

   Code

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

   Output

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

   Beachten Sie, dass wir zur Beantwortung dieser Abfrage die Fläche jedes Polygons berechnen müssen. Wenn wir dies häufig tun würden, wäre es sinnvoll, der Tabelle eine Flächenspalte hinzuzufügen, die aus Leistungsgründen indiziert werden könnte. Indem wir die Ergebnisse absteigend sortieren und den PostgreSQL-Befehl "LIMIT" verwenden, können wir einfach nur den größten Wert auswählen, ohne eine Aggregatfunktion wie MAX() zu verwenden.

4. *Wie lang ist die Länge der Straßen, die vollständig in jeder Gemeinde liegen?*

   Dies ist ein Beispiel für eine "räumliche Verknüpfung", bei der Daten aus zwei Tabellen (mit einer Verknüpfung) unter Verwendung einer räumlichen Interaktion ("enthalten") als Verknüpfungsbedingung zusammengeführt werden (anstelle des üblichen relationalen Ansatzes der Verknüpfung über einen gemeinsamen Schlüssel):

   Code

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

   Output

   ```
   name                        | roads_km
   ----------------------------+----------
   SURREY                      | 1539.476
   VANCOUVER                   | 1450.331
   LANGLEY DISTRICT            |  833.793
   BURNABY                     |  773.769
   PRINCE GEORGE               |  694.376
   ...
   ```

Diese Abfrage dauert eine Weile, da jede Straße in der Tabelle im Endergebnis zusammengefasst wird (etwa 250K Straßen für die Beispieltabelle). Bei kleineren Datensätzen (mehrere tausend Datensätze auf mehrere hundert) kann die Antwort sehr schnell sein.

5. *Erstellen Sie eine neue Tabelle mit allen Straßen innerhalb der Stadt Prince George.*

   Dies ist ein Beispiel für eine "Überlagerung", die zwei Tabellen aufnimmt und eine neue Tabelle aus räumlich beschnittenen oder geschnittenen Ergebnisgrößen ausgibt. Anders als bei der oben gezeigten "räumlichen Verknüpfung" werden bei dieser Abfrage neue Geometrien erstellt. Ein Overlay ist eine Art "Turbo" für räumliche Verknüpfungen und eignet sich für genauere Analysen:

   Code

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

6. *Wie lang ist die "Douglas St" in Victoria in Kilometern?*

   Code

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

   Output

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

7. *Welches ist das größte Gemeindepolygon, das ein Loch hat?*

   Code

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

   Output

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

Kapitel 6

Performance Tipps

6.1 Kleine Tabellen mit großen Geometrien

6.1.1 Problembeschreibung

Current PostgreSQL versions 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).

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

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

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

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

6.1.2 Umgehungslösung

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

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

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

Code

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

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

Code

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

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

6.2 CLUSTER auf die geometrischen Indizes

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

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

Code

```
CLUSTER my_table USING my_geom_index;
```

Output

```
CLUSTER
```

6.3 Vermeidung von Dimensionsumrechnungen

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

Code

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

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

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

Kapitel 7

Referenz PostGIS

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



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

7.1 PostgreSQL und PostGIS Datentypen - Geometry/Geography/Box

7.1.1 box2d

box2d — Der Typ, der einen 2-dimensionalen Begrenzungsrahmen darstellt.

Beschreibung

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

Die Darstellung enthält die Werte `xmin, ymin, xmax, ymax`. Dies sind die Mindest- und Höchstwerte der X- und Y-Ausdehnung.

`box2d` Objekte haben eine Textdarstellung, die wie folgt aussieht `BOX(1 2,5 6)`.

Verhaltensweise bei der Typumwandlung

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

Typumwandlung nach	Verhaltensweise
box3d	implizit
geometry	implizit

Siehe auch

Abschnitt 13.7

7.1.2 box3d

box3d — Der Typ, der einen 3-dimensionalen Begrenzungsrahmen darstellt.

Beschreibung

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

Die Darstellung enthält die Werte `xmin, ymin, zmin, xmax, ymax, zmax`. Dies sind die Minimal- und Maximalwerte der X-, Y- und Z-Ausdehnung.

`box3d` Objekte haben eine Textdarstellung, die wie folgt aussieht `BOX3D(1 2 3,5 6 5)`.

Verhaltensweise bei der Typumwandlung

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

Typumwandlung nach	Verhaltensweise
box	implizit
box2d	implizit
geometry	implizit

Siehe auch

Abschnitt 13.7

7.1.3 geometry

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

Beschreibung

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

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

Verhaltensweise bei der Typumwandlung

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

Typumwandlung nach	Verhaltensweise
box	implizit
box2d	implizit

Typumwandlung nach	Verhaltensweise
box3d	implizit
Bytea	implizit
geography	implizit
Text	implizit

Siehe auch

Abschnitt 4.1, Abschnitt 4.3

7.1.4 geometry_dump

geometry_dump — Ein zusammengesetzter Typ, der zur Beschreibung der Teile einer komplexen Geometrie verwendet wird.

Beschreibung

`geometry_dump` ist ein zusammengesetzter Datentyp, der die Felder enthält:

- `geom` - eine Geometrie, die eine Komponente der ausgelagerten Geometrie darstellt. Der Geometrietyp hängt von der Ursprungsfunktion ab.
- `path[]` - ein ganzzahliges Array, das den Navigationspfad innerhalb der ausgelagerten Geometrie zur Komponente `geom` definiert. Das Pfad-Array ist 1-basiert (d.h. `path[1]` ist das erste Element).

Er wird von der Funktionsfamilie `ST_Dump*` als Ausgabetyp verwendet, um eine komplexe Geometrie in ihre Bestandteile zu zerlegen.

Siehe auch

Abschnitt 13.6

7.1.5 geography

geography — Der Typ, der räumliche Merkmale mit geodätischen (ellipsoidischen) Koordinatensystemen darstellt.

Beschreibung

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

Räumliche Operationen am Geographietyp liefern genauere Ergebnisse, da das Ellipsoidmodell berücksichtigt wird.

Verhaltensweise bei der Typumwandlung

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

Typumwandlung nach	Verhaltensweise
geometry	explizit

Siehe auch

Abschnitt 4.3, Abschnitt 4.3

7.2 Geometrische Managementfunktionen

7.2.1 AddGeometryColumn

AddGeometryColumn — Entfernt eine Geometriespalte aus einer räumlichen Tabelle.

Synopsis

text **AddGeometryColumn**(varchar table_name, varchar column_name, integer srid, varchar type, integer dimension, boolean use_typmod=true);
text **AddGeometryColumn**(varchar schema_name, varchar table_name, varchar column_name, integer srid, varchar type, integer dimension, boolean use_typmod=true);
text **AddGeometryColumn**(varchar catalog_name, varchar schema_name, varchar table_name, varchar column_name, integer srid, varchar type, integer dimension, boolean use_typmod=true);

Beschreibung

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

Änderung: 2.0.0 Diese Funktion aktualisiert die geometry_columns Tabelle nicht mehr, da geometry_columns jetzt ein View ist, welcher den Systemkatalog ausliest. Standardmäßig werden auch keine Bedingungen/constraints erzeugt, sondern es wird der in PostgreSQL integrierte Typmodifikaor verwendet. So entspricht zum Beispiel die Erzeugung einer wgs84 POINT Spalte mit dieser Funktion: `ALTER TABLE some_table ADD COLUMN geom geometry(Point,4326);`

Änderung: 2.0.0 Falls Sie das alte Verhalten mit Constraints wünschen, setzen Sie bitte `use_typmod` vom standardmäßigen true auf false.

Änderung: 2.0.0 Views können nicht mehr händisch in "geometry_columns" registriert werden. Views auf eine Geometrie in Typmod-Tabellen, bei denen keine Adapterfunktion verwendet wird, registrieren sich selbst auf korrekte Weise, da sie die Typmod-Verhaltensweise von der Spalte der Stammtabelle erben. Views die ein geometrische Funktion ausführen die eine andere Geometrie ausgibt, benötigen die Umwandlung in eine Typmod-Geometrie, damit die Geometrie des Views korrekt in "geometry_columns" registriert wird. Siehe Abschnitt 4.6.3.

 Diese Methode implementiert die OGC Simple Features Implementation Specification for SQL 1.1.

 Diese Funktion unterstützt 3d und lässt den Z-Index nicht fallen.

 Diese Methode unterstützt kreisförmige Strings und Kurven.

Verbesserung: 2.0.0 use_typmod Argument eingeführt. Standardmäßig wird eine typmod Geometrie anstelle einer Constraint-basierten Geometrie erzeugt.

Beispiele

This example creates a schema to hold data.

Code

```
CREATE SCHEMA my_schema;
```

Create a simple PostgreSQL table in the schema.

Code

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

Describing the table shows a single `id` column.

Code

```
\d my_schema.my_spatial_table
```

Output

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

Add a spatial column to the table.

Code

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

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

Code

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

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

Code

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

Output

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

Describe the table again to see the new geometry columns.

Code

```
\d my_schema.my_spatial_table
```

Output

```
Table "my_schema.my_spatial_table"

Columns:
  id       integer              not null default nextval('my_schema.my_spatial_table_id_seq'::regclass)
  geom     geometry(Point,4326)
  geom_c   geometry
  geomcp_c geometry

Check constraints:
    "enforce_dims_geom_c" CHECK (st_ndims(geom_c) = 2)
    "enforce_dims_geomcp_c" CHECK (st_ndims(geomcp_c) = 2)
    "enforce_geotype_geom_c" CHECK (geometrytype(geom_c) = 'POINT'::text OR geom_c IS NULL)
    "enforce_geotype_geomcp_c" CHECK (geometrytype(geomcp_c) = 'CURVEPOLYGON'::text OR geomcp_c IS NULL)
    "enforce_srid_geom_c" CHECK (st_srid(geom_c) = 4326)
    "enforce_srid_geomcp_c" CHECK (st_srid(geomcp_c) = 4326)
```

The `geometry_columns` view also registers the new columns.

Code

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

Output

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

Siehe auch

DropGeometryColumn, DropGeometryTable, Abschnitt 4.6.2, Abschnitt 4.6.3

7.2.2 DropGeometryColumn

DropGeometryColumn — Entfernt eine Geometriespalte aus einer räumlichen Tabelle.

Synopsis

text **DropGeometryColumn**(varchar table_name, varchar column_name);
text **DropGeometryColumn**(varchar schema_name, varchar table_name, varchar column_name);
text **DropGeometryColumn**(varchar catalog_name, varchar schema_name, varchar table_name, varchar column_name);

Beschreibung

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

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

Diese Methode implementiert die OGC Simple Features Implementation Specification for SQL 1.1.

Diese Funktion unterstützt 3d und lässt den Z-Index nicht fallen.

Diese Methode unterstützt kreisförmige Strings und Kurven.

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

Beispiele

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

Code

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

The legacy compatibility function performs the equivalent operation.

Code

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

Output

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

Siehe auch

AddGeometryColumn, DropGeometryTable, Abschnitt 4.6.2

7.2.3 DropGeometryTable

DropGeometryTable — Löscht eine Tabelle und alle Referenzen in dem geometry_columns View.

Synopsis

boolean **DropGeometryTable**(varchar table_name);
boolean **DropGeometryTable**(varchar schema_name, varchar table_name);
boolean **DropGeometryTable**(varchar catalog_name, varchar schema_name, varchar table_name);

Beschreibung

Löscht eine Tabelle und deren Verweise in "geometry_columns". Anmerkung: verwendet current_schema() wenn kein Schema angegeben wird, eine Schema erkennende pgsql Installation vorausgesetzt.



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

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

Beispiele

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

Code

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

The legacy compatibility function performs the equivalent operation.

Code

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

Output

```
my_schema.my_spatial_table dropped.
```

Siehe auch

AddGeometryColumn, DropGeometryColumn, Abschnitt 4.6.2

7.2.4 Find_SRID

Find_SRID — Gibt die für eine Geometriespalte definierte SRID zurück.

Synopsis

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

Beschreibung

Liefert die Integer-SRID der angegebenen Geometriespalte durch Suche in der Tabelle GEOMETRY_COLUMNS. Wenn die Geometriespalte nicht ordnungsgemäß hinzugefügt wurde (z. B. mit der Funktion AddGeometryColumn), funktioniert diese Funktion nicht.

Beispiele

Code

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

Output

```
4269
```

Siehe auch

ST_SRID

7.2.5 Populate_Geometry_Columns

Populate_Geometry_Columns — Stellt sicher, dass Geometriespalten mit Typmodifikatoren definiert sind oder geeignete räumliche Beschränkungen haben.

Synopsis

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

Beschreibung

Sorgt dafür, dass die Geometriespalten mit Typmodifikatoren oder mit passenden räumlichen Constraints versehen sind. Dadurch wird die korrekte Registrierung im View `geometry_columns` sichergestellt. Standardmäßig werden alle Geometriespalten, die keinen Typmodifikator aufweisen, mit Typmodifikatoren versehen. Für die alte Verhaltensweise setzen Sie bitte `use_typmod=false`

Aus Gründen der Abwärtskompatibilität und für räumliche Anwendungen, wie eine Tabellenvererbung bei denen jede Kindtabelle einen anderen geometrischen Datentyp aufweist, wird die alte Verhaltensweise mit Check-Constraints weiter unterstützt. Wenn Sie diese alte Verhaltensweise benötigen, können Sie den neuen Übergabewert auf FALSE setzen - `use_typmod=false`. Wenn Sie dies tun, so werden die Geometriespalten anstelle von Typmodifikatoren mit 3 Constraints erstellt. Insbesondere bedeutet dies, dass jede Geometriespalte, die zu einer Tabelle gehört, mindestens drei Constraints aufweist:

- `enforce_dims_the_geom` - stellt sicher, dass jede Geometrie dieselbe Dimension hat (siehe ST_NDims)
- `enforce_geotype_the_geom` - stellt sicher, dass jede Geometrie vom selben Datentyp ist (siehe GeometryType)
- `enforce_srid_the_geom` - stellt sicher, dass jede Geometrie die selbe Projektion hat (siehe ST_SRID)

Wenn die `oid` einer Tabelle übergeben wird, so versucht diese Funktion, die SRID, die Dimension und den Datentyp der Geometrie in der Tabelle zu bestimmen und fügt, falls notwendig, Constraints hinzu. Bei Erfolg wird eine entsprechende Spalte in die Tabelle "geometry_columns" eingefügt, andernfalls wird der Fehler abgefangen und eine Fehlermeldung ausgegeben, die das Problem beschreibt.

Wenn die `oid` eines Views übergeben wird, so versucht diese Funktion, die SRID, die Dimension und den Datentyp der Geometrie in dem View zu bestimmen und die entsprechenden Einträge in die Tabelle `geometry_columns` vorzunehmen. Constraints werden allerdings nicht erzwungen.

Die parameterlose Variante ist ein einfacher Adapter für die parametrisierte Variante, welche die Tabelle "geometry_columns" zuerst entleert und dann für jede räumliche Tabelle oder View in der Datenbank wiederbefüllt. Wo es passend ist, werden räumliche Constraints auf die Tabellen gelegt. Es wird die Anzahl der in der Datenbank gefundenen Geometriespalten und die Anzahl der in die Tabelle `geometry_columns` eingefügten Zeilen ausgegeben. Die parametrisierte Version gibt lediglich die Anzahl der Zeilen aus, die in die Tabelle `geometry_columns` eingefügt wurden.

Verfügbarkeit: 1.4.0

Änderung: 2.0.0 Standardmäßig werden nun Typmodifikatoren anstelle von Check-Constraints für die Beschränkung des Geometrietyps verwendet. Sie können nach wie vor stattdessen die Verhaltensweise mit Check-Constraints verwenden, indem Sie die neu eingeführte Variable `use_typmod` auf FALSE setzen.

Erweiterung: 2.0.0 Der optionale Übergabewert `use_typmod` wurde eingeführt, um bestimmen zu können, ob die Spalten mit Typmodifikatoren oder mit Check-Constraints erstellt werden sollen.

Beispiele

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

Code

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

Output

```
1
```

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

Code

```
\d myspatial_table
```

Output

```
Table "public.myspatial_table"

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

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

Code

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

Output

```
1
```

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

Code

```
\d myspatial_table_cs
```

Output

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

7.2.6 UpdateGeometrySRID

UpdateGeometrySRID — Aktualisiert die SRID aller Features in einer Geometriespalte und die Metadaten der Tabelle.

Synopsis

text **UpdateGeometrySRID**(varchar table_name, varchar column_name, integer srid);
text **UpdateGeometrySRID**(varchar schema_name, varchar table_name, varchar column_name, integer srid);
text **UpdateGeometrySRID**(varchar catalog_name, varchar schema_name, varchar table_name, varchar column_name, integer srid);

Beschreibung

Erneuert die SRID aller Features in einer Geometriespalte; erneuert die Constraints und die Referenz in "geometry_columns". Anmerkung: verwendet current_schema() wenn kein Schema angegeben wird, eine Schema erkennende pgsql Installation vorausgesetzt.

Diese Funktion unterstützt 3d und lässt den Z-Index nicht fallen.

Diese Methode unterstützt kreisförmige Strings und Kurven.

Beispiele

Einfügen von Geometrien in eine Straßentabelle mit einem SRID-Satz, der bereits das EWKT-Format verwendet:

Code

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

Ändert die SRID der Straßentabelle auf 4326

Code

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

Das vorhergegangene Beispiel ist gleichbedeutend mit diesr DDL Anweisung

Code

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

Falls Sie sich in der Projektion geirrt haben (oder sie als "unknown" importiert haben) und sie in einem Aufwaschen in die Web Mercator Projektion transformieren wollen, so können Sie dies mit DDL bewerkstelligen. Es gibt jedoch keine äquivalente PostGIS Managementfunktion, die dies in einem Schritt bewerkstelligen könnte.

Code

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

Siehe auch

UpdateRasterSRID, ST_SetSRID, ST_Transform

7.3 Geometrische Konstruktoren

7.3.1 ST_Collect

ST_Collect — Erzeugt eine GeometryCollection oder Multi*-Geometrie aus einer Reihe von Geometrien.

Synopsis

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

Beschreibung

Sammelt Geometrien in einer Geometriesammlung. Das Ergebnis ist entweder ein Multi* oder eine GeometryCollection, je nachdem, ob die Eingabegeometrien denselben oder einen unterschiedlichen Typ haben (homogen oder heterogen). Die Eingangsgeometrien bleiben in der Sammlung unverändert.

Variante 1: akzeptiert zwei Eingabegeometrien

Variante 2: akzeptiert eine Reihe von Geometrien

Variante 3: Aggregatfunktion, die ein Rowset von Geometrien akzeptiert.



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

If any input geometry is a collection (Multi* or GeometryCollection), `ST_Collect` returns a GeometryCollection, since that is the only type which can contain nested collections. To prevent this, use ST_Dump in a subquery to expand input collections to their atomic elements (see example below).

Verfügbarkeit: 1.4.0 - ST_MakeLine(geomarray) wurde eingeführt. ST_MakeLine Aggregatfunktion wurde verbessert, um mehr Punkte schneller handhaben zu können.

 Diese Funktion unterstützt 3d und lässt den Z-Index nicht fallen.

 Diese Methode unterstützt kreisförmige Strings und Kurven.

Beispiele

Two-input variant.

Sammeln Sie 2D-Punkte.

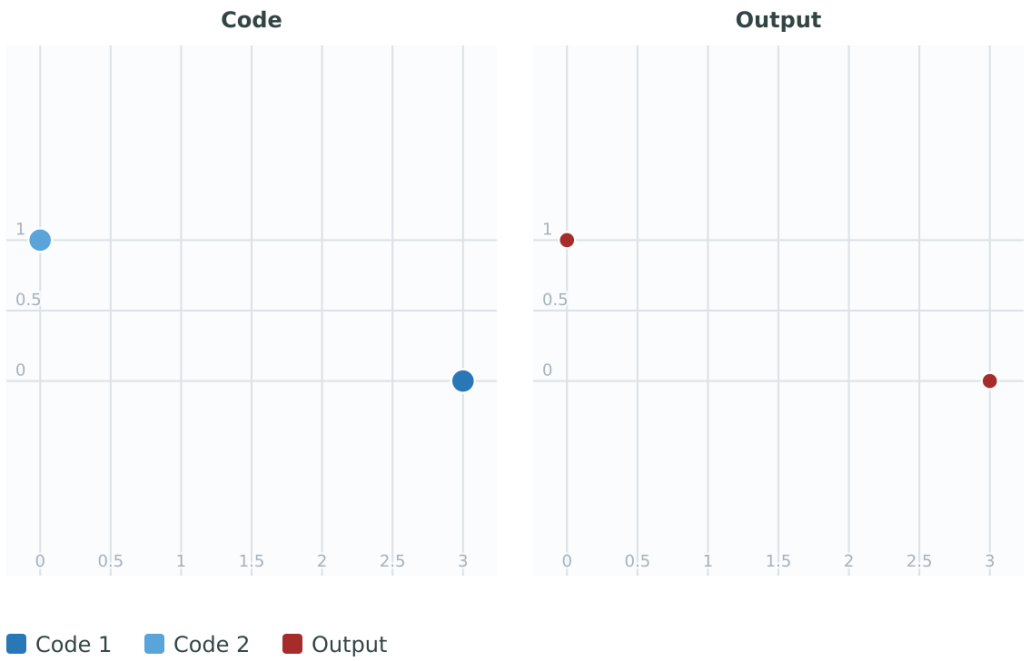
Code

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

Output

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

Figure



Sammeln Sie 3D-Punkte.

Code

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

Output

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

Figure

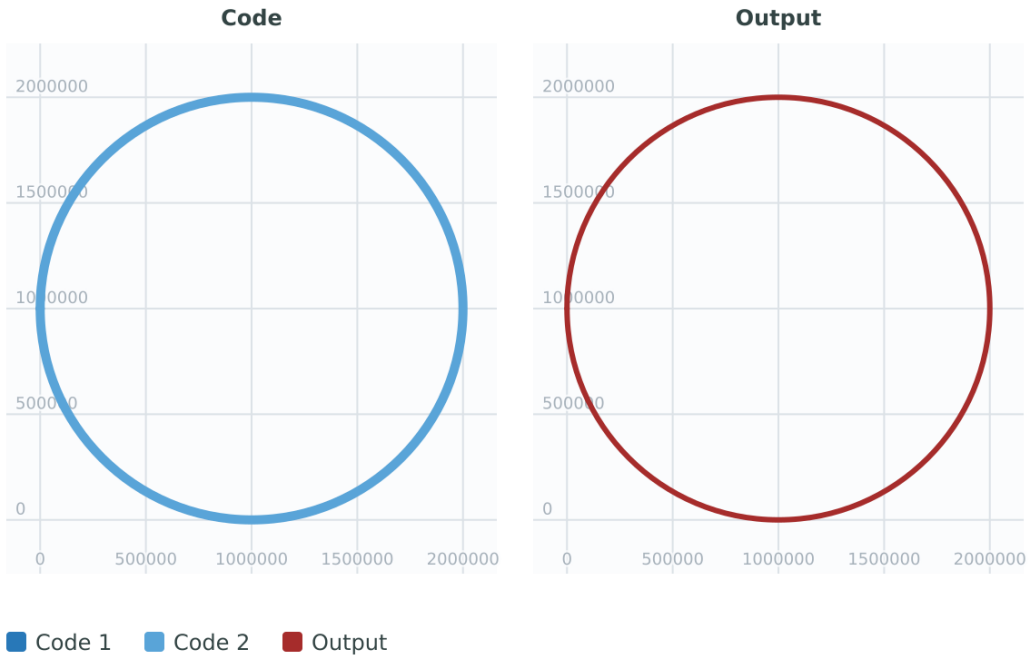


Kurven sammeln.

Code

```
SELECT ST_Collect(
        'CIRCULARSTRING(220268 150415,220227 150505,220227 150406)',
        'CIRCULARSTRING(220227 150406,2220227 150407,220227 150406)');
```

Figure



Array variant.

Verwendung eines Array-Konstruktors für eine Subquery.

Code

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

Verwendung eines Array-Konstruktors für Werte.

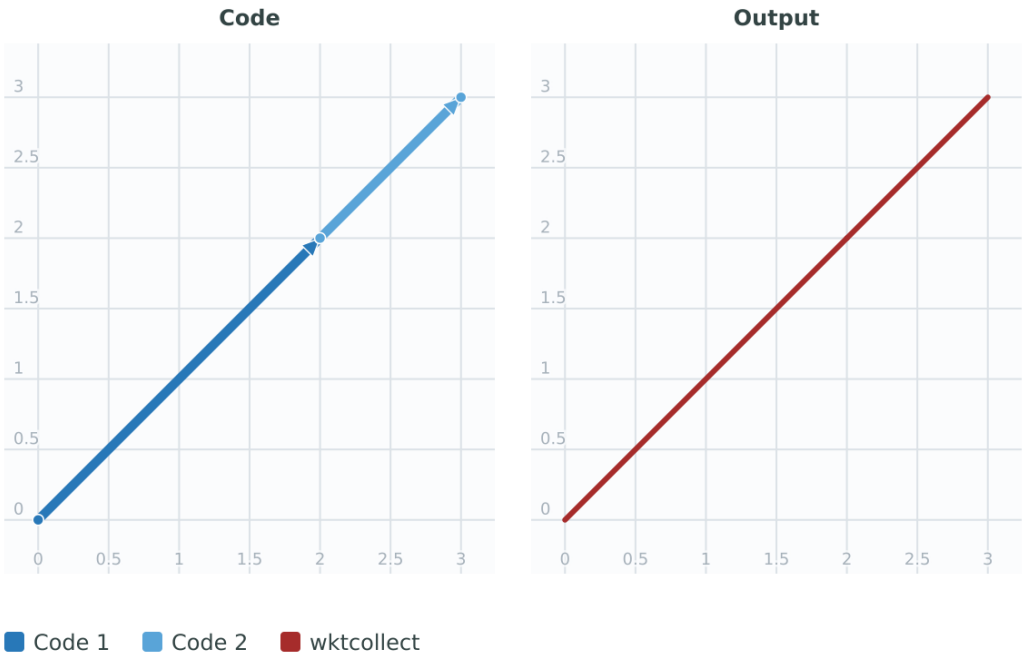
Code

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

Output

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

Figure



Aggregate variant.

Erstellen mehrerer Sammlungen durch Gruppierung von Geometrien in einer Tabelle.

Code

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

Siehe auch

ST_Dump, ST_AsBinary

7.3.2 ST_LineFromMultiPoint

ST_LineFromMultiPoint — Erzeugt einen LineString aus einer MultiPoint Geometrie.

Synopsis

geometry **ST_LineFromMultiPoint**(geometry aMultiPoint);

Beschreibung

Erzeugt einen LineString aus einer MultiPoint Geometrie.

Für Punkt mit X-, Y- und M-Koordinaten verwenden Sie bitte ST_MakePointM .

Diese Funktion unterstützt 3d und lässt den Z-Index nicht fallen.

Beispiele

Erzeugt einen LineString aus einer MultiPoint Geometrie.

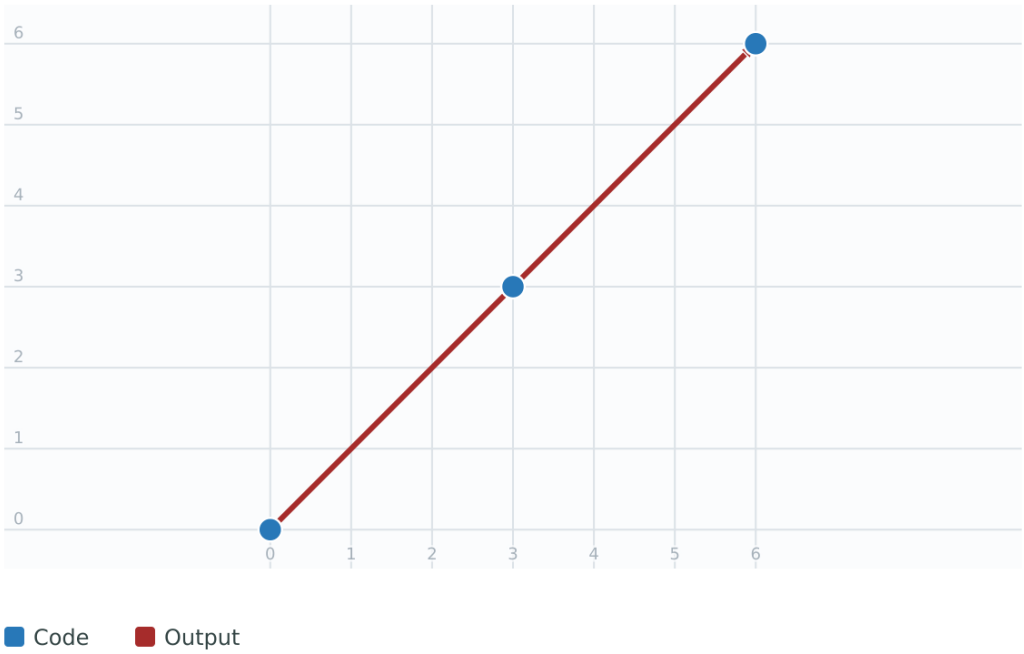
Code

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

Output

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

Figure



Siehe auch

ST_AsEWKT, ST_AsKML

7.3.3 ST_MakeEnvelope

ST_MakeEnvelope — Erzeugt ein rechteckiges Polygon aus den gegebenen Minimum- und Maximumwerten. Die Eingabewerte müssen in dem Koordinatenreferenzsystem sein, welches durch die SRID angegeben wird.

Synopsis

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

Beschreibung

Erzeugt ein rechteckiges Polygon das durch die Minima und Maxima angegeben wird. durch die gegebene Hülle. Die Eingabewerte müssen in dem Koordinatenreferenzsystem sein, welches durch die SRID angegeben wird. Wenn keine SRID angegeben ist, so wird das Koordinatenreferenzsystem "unknown" angenommen

Verfügbarkeit: 1.5

Erweiterung: 2.0: es wurde die Möglichkeit eingeführt, eine Einhüllende/Envelope festzulegen, ohne dass die SRID spezifiziert ist.

Beispiele

Building a bounding box polygon.

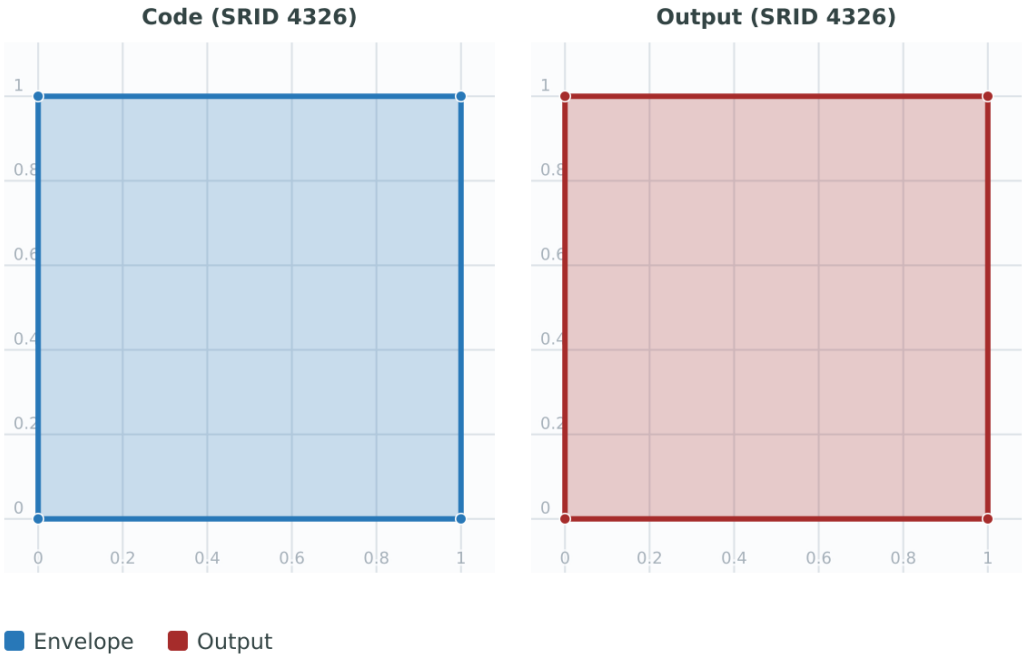
Code

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

Output

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

Figure



Siehe auch

ST_MakePoint, ST_MakePoint, ST_Point, ST_SRID

7.3.4 ST_MakeLine

ST_MakeLine — Erzeugt einen Linienzug aus einer Punkt-, Mehrfachpunkt- oder Liniengeometrie.

Synopsis

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

Beschreibung

Erzeugt einen LineString, der die Punkte von Point-, MultiPoint- oder LineString-Geometrien enthält. Andere Geometrietypen verursachen einen Fehler.

Variante 1: akzeptiert zwei Eingabegeometrien

Variante 2: akzeptiert eine Reihe von Geometrien

Variante 3: Aggregatfunktion, die ein Rowset von Geometrien akzeptiert. Um die Reihenfolge der Eingabegeometrien sicherzustellen, verwenden Sie `ORDER BY` im Funktionsaufruf oder eine Unterabfrage mit einer `ORDER BY` Klausel.

Repeated nodes at the beginning of input LineStrings are collapsed to a single point. Repeated points in Point and MultiPoint inputs are not collapsed. Components of MultiLineString are handled in the order they appear in the collection. ST_RemoveRepeatedPoints can be used to collapse repeated points from the output LineString.

Diese Funktion unterstützt 3d und lässt den Z-Index nicht fallen.

Availability: 3.7.0 - Support for MultiLineString input elements was introduced

Verfügbarkeit: Mit 2.3.0 wurde die Unterstützung zur Eingabe von MultiPoint Elementen eingeführt

Verfügbarkeit: 2.0.0 - Unterstützung zur Eingabe von LineString Elementen eingeführt

Verfügbarkeit: 1.4.0 - ST_MakeLine(geomarray) wurde eingeführt. ST_MakeLine Aggregatfunktion wurde verbessert, um mehr Punkte schneller handhaben zu können.

Beispiele

Two-input variant.

Erstellen Sie eine Linie, die aus zwei Punkten besteht.

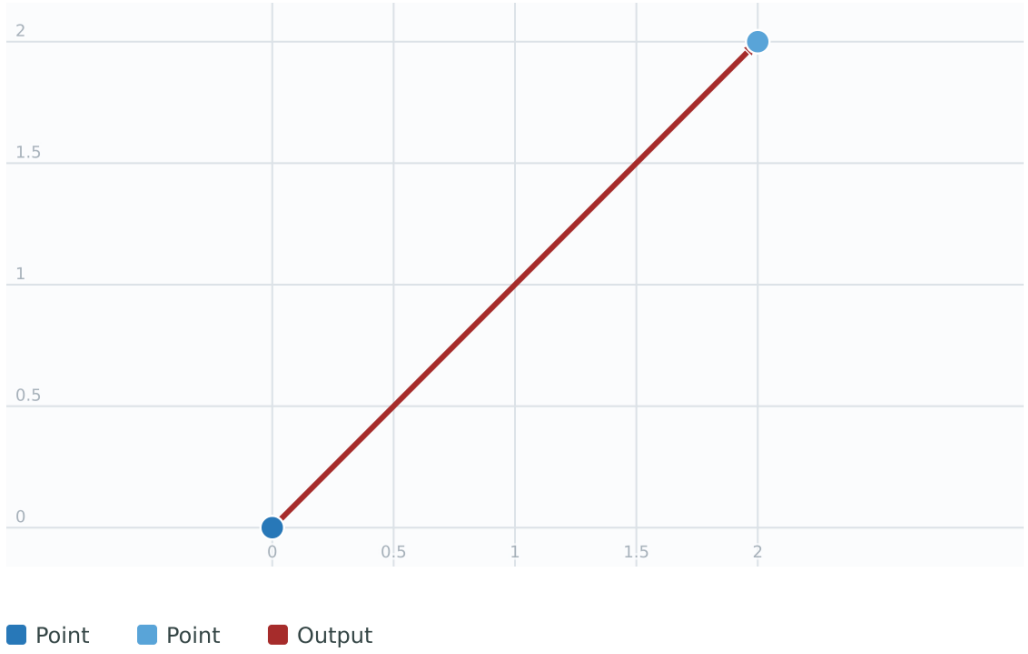
Code

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

Output

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

Figure



Erzeugt eine BOX3D, die durch 2 geometrische 3D-Punkte definiert wird.

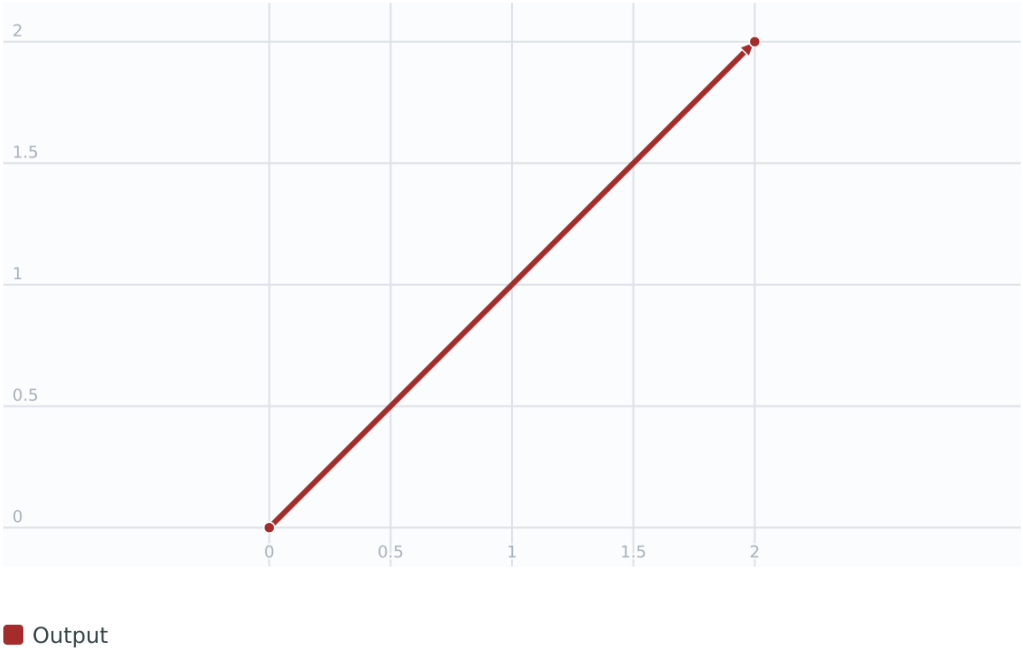
Code

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

Output

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

Figure



Erzeugt einen Linienzug aus einer Punkt-, Mehrfachpunkt- oder Liniengeometrie.

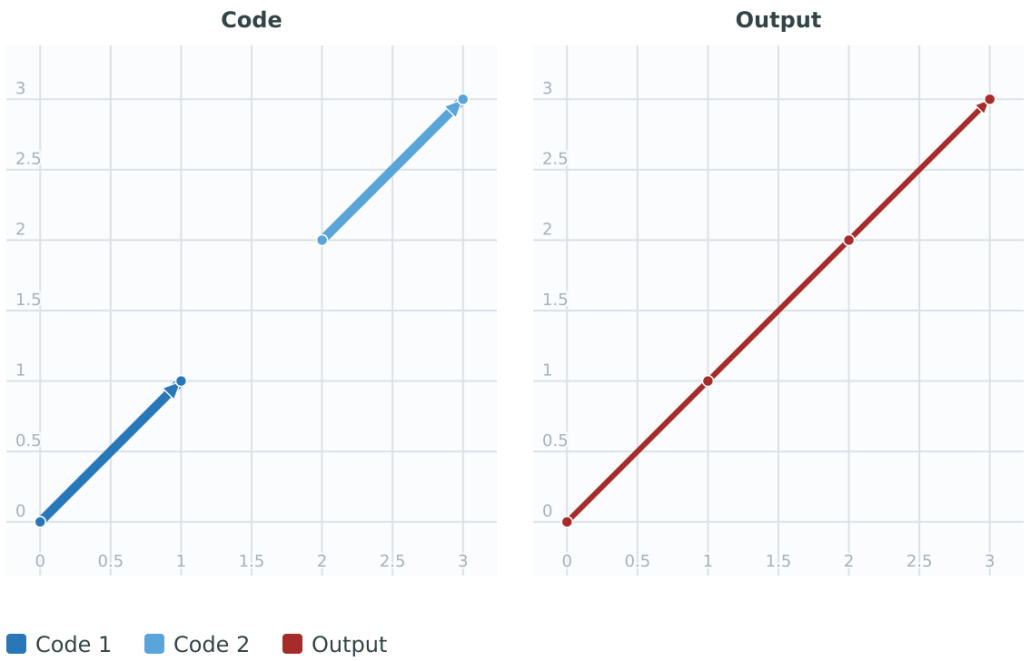
Code

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

Output

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

Figure



Array variant.

Erstellen Sie eine Zeile aus einem Array, das durch eine Unterabfrage mit Bestellung gebildet wird.

Code

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

Erstellen einer 3D-Linie aus einem Array von 3D-Punkten

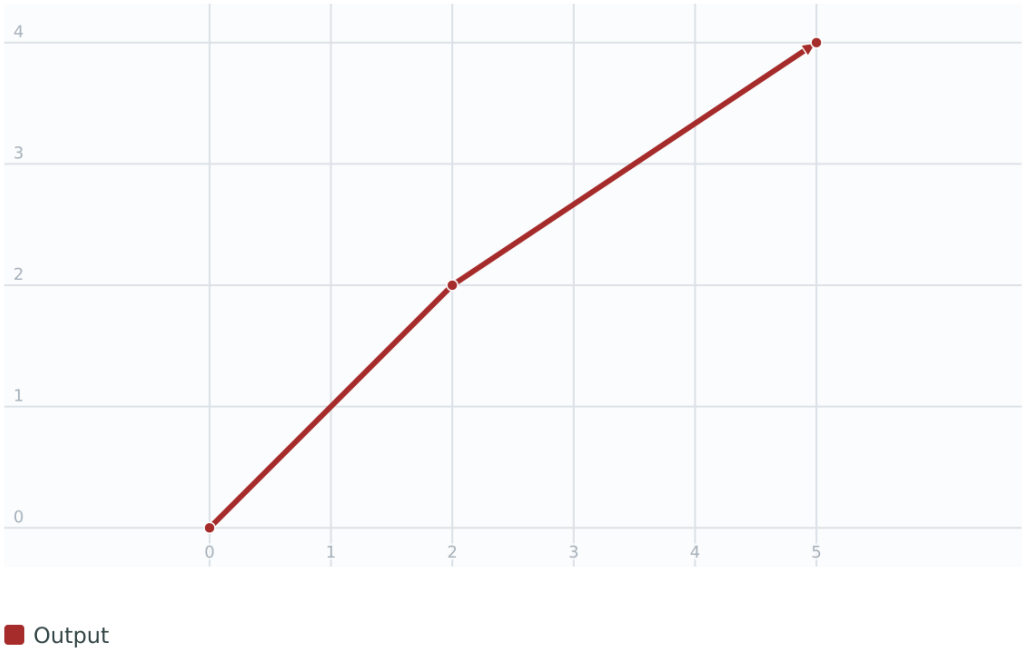
Code

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

Output

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

Figure



Aggregate variant.

Diese Beispiel nimmt eine Abfolge von GPS Punkten entgegen und erzeugt einen Datensatz für jeden GPS Pfad, wobei das Geometriefeld ein Linienzug ist, welcher in der Reihenfolge der Aufnahmeroute aus den GPS Punkten zusammengesetzt wird.

Die Verwendung des Aggregats `ORDER BY` liefert einen korrekt geordneten LineString.

Code

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

When ordering within the aggregate is not possible, ordering in a subquery can be used. However, sometimes the query plan may not respect the order of the subquery.

Code

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

Siehe auch

ST_RemoveRepeatedPoints, ST_AsText, ST_GeomFromText, ST_MakePoint

7.3.5 ST_MakePoint

ST_MakePoint — Erzeugt eine 2D-, 3DZ- oder 4D-Punktgeometrie.

Synopsis

geometry **ST_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);

Beschreibung

Erzeugt eine 2D XY, 3D XYZ oder 4D XYZM Punktgeometrie. Verwenden Sie ST_MakePointM, um Punkte mit XYM-Koordinaten zu erstellen.

Verwenden Sie ST_SetSRID, um eine SRID für den erstellten Punkt anzugeben.

`ST_MakePoint` ist zwar nicht OGC-konform, aber schneller als ST_GeomFromText und ST_PointFromText. Auch ist die Verwendung von numerischen Koordinatenwerten einfacher.



Anmerkung
Bei geodätischen Koordinaten ist `X` der Längengrad und `Y` der Breitengrad.



Anmerkung
Die Funktionen ST_Point, ST_PointZ, ST_PointM und ST_PointZM können verwendet werden, um Punkte mit einer bestimmten SRID zu erstellen.



Diese Funktion unterstützt 3d und lässt den Z-Index nicht fallen.

Beispiele

This example creates a point with an unknown SRID.

Code

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

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

Code

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

This example creates a 3D point with altitude.

Code

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

This example returns the Z coordinate of a point.

Code

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

Output

```
1.5
```

Siehe auch

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

7.3.6 ST_MakePointM

ST_MakePointM — Erzeugt einen Punkt mit x, y und measure/Kilometrierungs Koordinaten.

Synopsis

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

Beschreibung

Erstellt einen Punkt mit X-, Y- und M-Koordinaten (Maß). Verwenden Sie ST_MakePoint, um Punkte mit XY-, XYZ- oder XYZM-Koordinaten zu erstellen.

Verwenden Sie ST_SetSRID, um eine SRID für den erstellten Punkt anzugeben.



Anmerkung
Bei geodätischen Koordinaten ist `X` der Längengrad und `Y` der Breitengrad.



Anmerkung
Die Funktionen ST_PointM und ST_PointZM können verwendet werden, um Punkte mit einem M-Wert und einem bestimmten SRID zu erstellen.

Beispiele



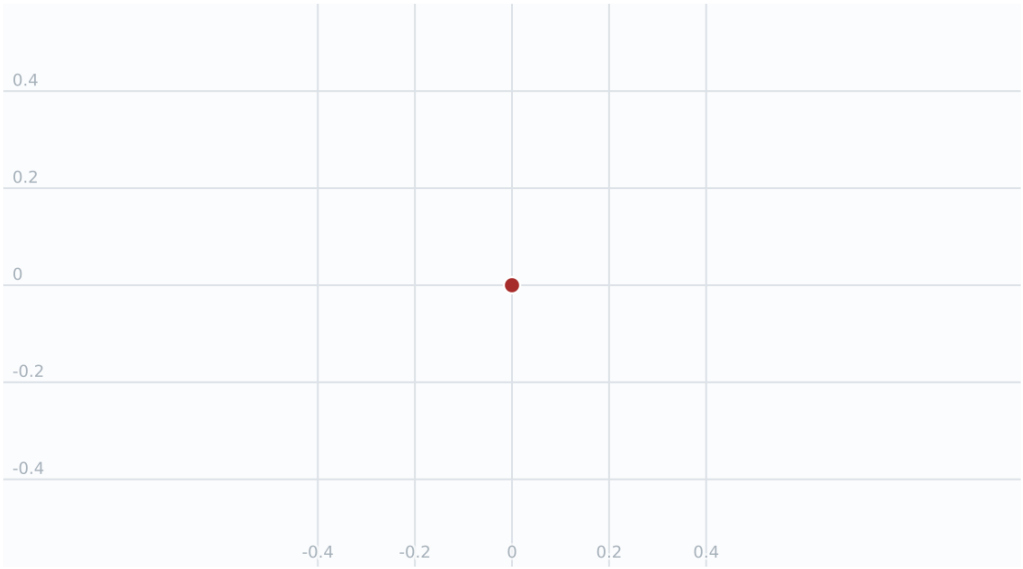
Anmerkung
ST_AsEWKT wird für die Textausgabe verwendet, da ST_AsText keine M-Werte unterstützt.

Punkt mit unbekanntem SRID erstellen.

Code

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

Figure

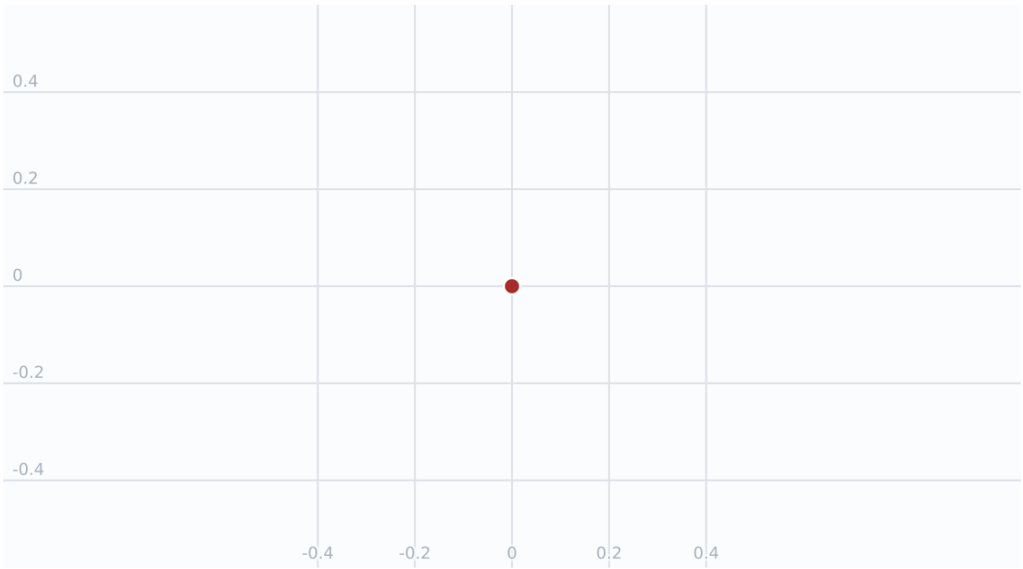


Erzeugt einen Punkt mit x, y und measure/Kilometrierungs Koordinaten.

Code

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

Figure



Ermittelt das Maß des erstellten Punktes.

Code

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

Output

```
10
```

Siehe auch

ST_MakePoint, ST_SetSRID, ST_PointM, ST_PointZM

7.3.7 ST_MakePolygon

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

Synopsis

geometry **ST_MakePolygon**(geometry linestring);

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

Beschreibung

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

Variant 1: Accepts one shell ring.

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



Anmerkung
Diese Funktion akzeptiert keine MULTILINESTRINGs. Verwenden Sie bitte ST_LineMerge oder ST_Dump um Linienzüge zu erzeugen.

 Diese Funktion unterstützt 3d und lässt den Z-Index nicht fallen.

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

Beispiele

Single input variant.

Erzeugt einen LineString aus einem codierten Linienzug.

Code

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

Erstellen Sie ein Polygon aus einem offenen LineString, indem Sie ST_StartPoint und ST_AddPoint verwenden, um es zu schließen.

Code

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

Erzeugt einen LineString aus einem codierten Linienzug.

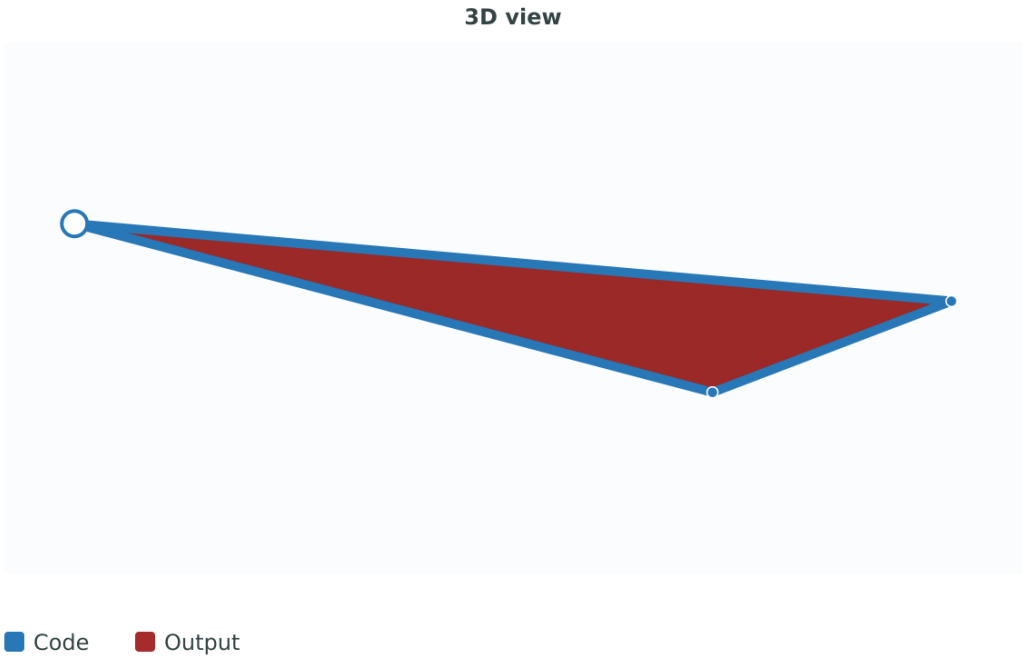
Code

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

Output

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

Figure

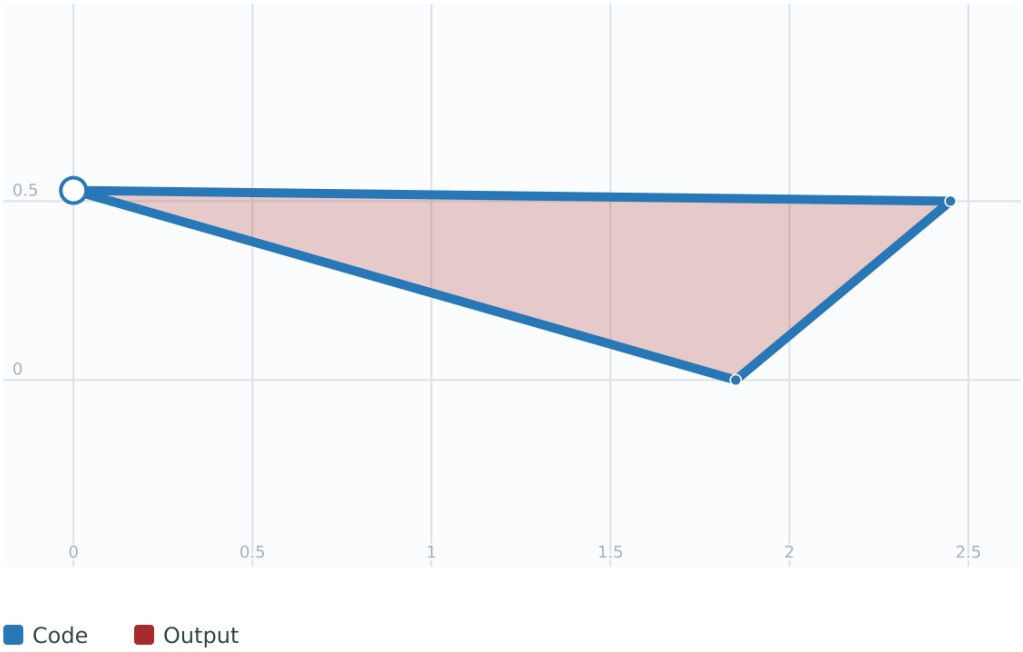


Erstellen eines Polygons aus einem LineString mit Maßen

Code

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

Figure



Create a CurvePolygon from a CircularString shell.

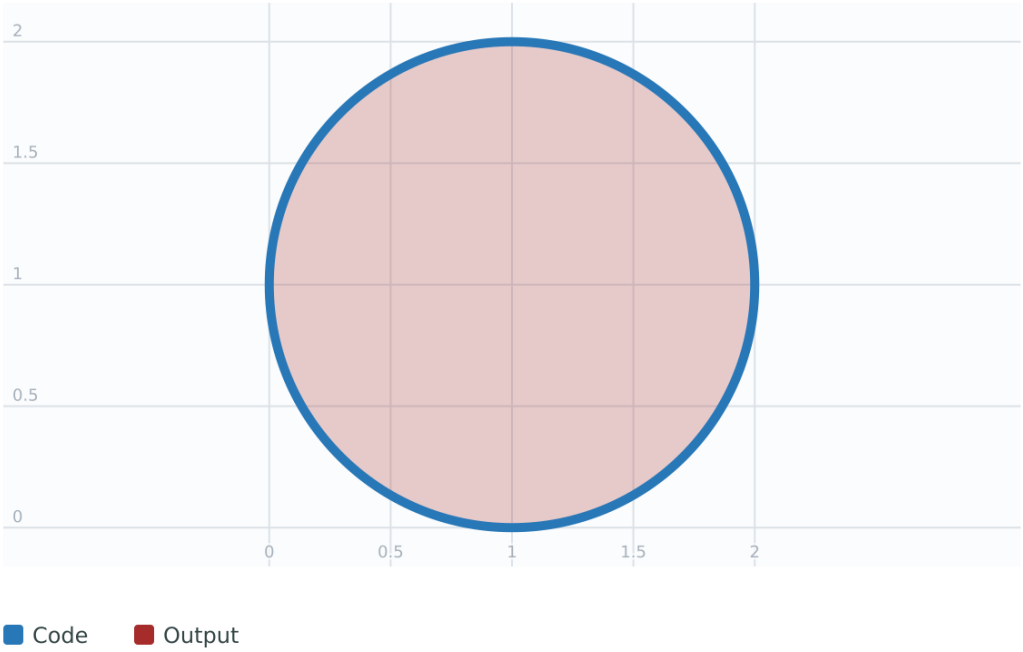
Code

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

Output

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

Figure



Outer shell with inner holes variant.

Erzeugung eines Donuts mit einem Ameisenloch

Code

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

Erstellen Sie einen Satz von Provinzgrenzen mit Löchern, die Seen darstellen. Die Eingabe ist eine Tabelle mit Provinz-Polygonen/MultiP und eine Tabelle mit Wasserlinien. Die Linien, die Seen bilden, werden mit ST_IsClosed ermittelt. Das Provinzlinienwerk wird mit ST_Boundary extrahiert. Wie von `ST_MakePolygon` gefordert, wird die Begrenzung mit ST_LineMerge zu einem einzigen LineString gezwungen. (Beachten Sie jedoch, dass dies ein ungültiges Polygon ergibt, wenn eine Provinz mehr als eine Region oder Inseln hat). Die Verwendung eines LEFT JOIN stellt sicher, dass alle Provinzen einbezogen werden, auch wenn sie keine Seen haben.



Anmerkung
Das Konstrukt mit CASE wird verwendet, da die Übergabe eines NULL-Feldes an ST_MakePolygon NULL ergibt.

Code

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

Eine andere Technik besteht darin, eine korrelierte Unterabfrage und den ARRAY()-Konstruktor zu verwenden, der einen Zeilensatz in ein Array umwandelt.

Code

```
SELECT p.gid, p.province_name,
  CASE
    WHEN EXISTS (
      SELECT w.geom
      FROM waterlines w
      WHERE ST_Within(w.geom, p.geom)
      AND ST_IsClosed(w.geom)
    )
    THEN ST_MakePolygon(
      ST_LineMerge(ST_Boundary(p.geom)),
      ARRAY(
        SELECT w.geom
        FROM waterlines w
        WHERE ST_Within(w.geom, p.geom)
        AND ST_IsClosed(w.geom)
```

```
      )
    )
    ELSE p.geom
  END AS geom
FROM provinces p;
```

Siehe auch

ST_BuildArea ST_Polygon

7.3.8 ST_Point

ST_Point — Erzeugt einen Punkt mit X-, Y- und SRID-Werten.

Synopsis

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

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

Beschreibung

Gibt einen Punkt mit den angegebenen X- und Y-Koordinatenwerten zurück. Dies ist das SQL-MM-Äquivalent für ST_MakePoint, das nur X und Y benötigt.



Anmerkung
Bei geodätischen Koordinaten ist `X` der Längengrad und `Y` der Breitengrad.

Verbessert: 3.2.0 srid wurde als zusätzliches optionales Argument hinzugefügt. Ältere Installationen erfordern die Kombination mit ST_SetSRID, um das Raster auf der Geometrie zu markieren.

 Diese Methode setzt die SQL/MM-Spezifikation um. SQL-MM 3: 6.1.2

Beispiele

Geometry.

Code

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

Erstellen eines Punktes mit angegebener SRID:

Code

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

Alternative Möglichkeit der Angabe von SRID:

Code

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

Erstellen Sie die Punkte geography mit der Syntax `::` cast:

Code

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

Pre-PostGIS 3.2 Code, mit `CAST`:

Code

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

Wenn die Punktkoordinaten nicht in einem geodätischen Koordinatensystem (z. B. WGS84) vorliegen, müssen sie vor der Übertragung in eine Geografie neu projiziert werden. In diesem Beispiel wird ein Punkt in Pennsylvania State Plane feet (SRID 2273) auf WGS84 (SRID 4326) projiziert.

Code

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

Siehe auch

ST_MakePoint, ST_PointZ, ST_PointM, ST_PointZM, ST_SetSRID, ST_Transform

7.3.9 ST_PointZ

ST_PointZ — Erzeugt einen Punkt mit X-, Y-, Z- und SRID-Werten.

Synopsis

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

Beschreibung

Gibt einen ST_Point mit den gegebenen Koordinatenwerten aus. Ein OGC-Alias für ST_MakePoint.

Verbessert: 3.2.0 srid wurde als zusätzliches optionales Argument hinzugefügt. Ältere Installationen erfordern die Kombination mit ST_SetSRID, um das Raster auf der Geometrie zu markieren.

Beispiele

Code

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

Code

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

Code

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

Siehe auch

ST_MakePoint, ST_PointFromText, ST_SetSRID, ST_MakePointM

7.3.10 ST_PointM

ST_PointM — Erzeugt einen Punkt mit den Werten X, Y, M und SRID.

Synopsis

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

Beschreibung

Gibt einen ST_Point mit den gegebenen Koordinatenwerten aus. Ein OGC-Alias für ST_MakePoint.

Verbessert: 3.2.0 srid wurde als zusätzliches optionales Argument hinzugefügt. Ältere Installationen erfordern die Kombination mit ST_SetSRID, um das Raster auf der Geometrie zu markieren.

Beispiele

Code

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

Code

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

Code

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

Siehe auch

ST_MakePoint, ST_PointFromText, ST_SetSRID, ST_MakePointM

7.3.11 ST_PointZM

ST_PointZM — Erzeugt einen Punkt mit den Werten X, Y, Z, M und SRID.

Synopsis

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

Beschreibung

Gibt einen ST_Point mit den gegebenen Koordinatenwerten aus. Ein OGC-Alias für ST_MakePoint.

Verbessert: 3.2.0 srid wurde als zusätzliches optionales Argument hinzugefügt. Ältere Installationen erfordern die Kombination mit ST_SetSRID, um das Raster auf der Geometrie zu markieren.

Beispiele

Code

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

Code

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

Code

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

Siehe auch

ST_MakePoint, ST_Point, ST_PointM, ST_PointZ, ST_SetSRID

7.3.12 ST_Polygon

ST_Polygon — Erzeugt ein Polygon aus einem LineString mit einem angegebenen SRID.

Synopsis

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

Beschreibung

Gibt ein Polygon aus dem angegebenen LineString zurück und setzt das räumliche Bezugssystem aus dem `srid`.

ST_Polygon ist ähnlich wie ST_MakePolygon Variante 1 mit dem Zusatz, dass der SRID gesetzt wird.

, ST_MakePoint, ST_SetSRID



Anmerkung
Diese Funktion akzeptiert keine MULTILINESTRINGs. Verwenden Sie bitte ST_LineMerge oder ST_Dump um Linienzüge zu erzeugen.

 Diese Methode implementiert die OGC Simple Features Implementation Specification for SQL 1.1.

 Diese Methode setzt die SQL/MM-Spezifikation um. SQL-MM 3: 8.3.2

 Diese Funktion unterstützt 3d und lässt den Z-Index nicht fallen.

Beispiele

Erstellen Sie ein 2D-Polygon.

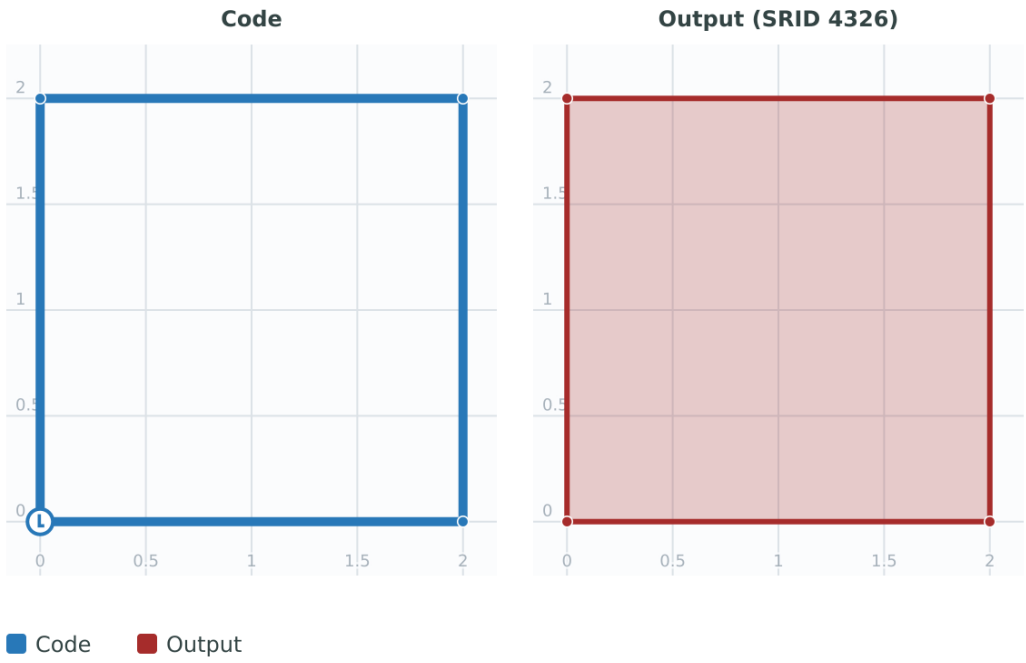
Code

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

Output

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

Figure



Erstellen Sie ein 3D-Polygon.

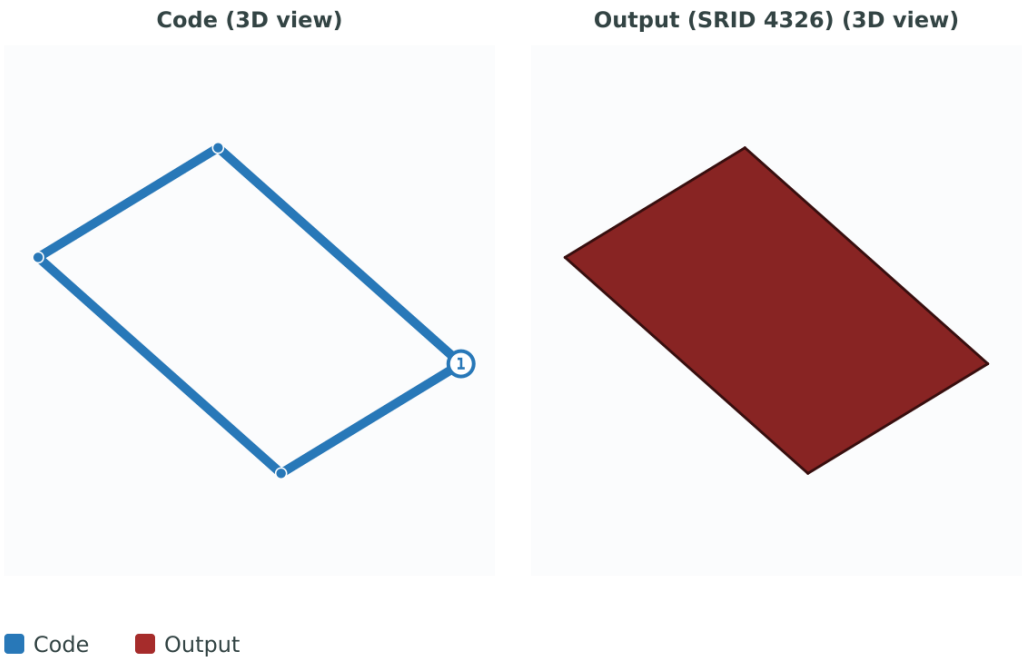
Code

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

Output

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

Figure



Siehe auch

ST_AsEWKT, ST_AsText, ST_GeomFromEWKT, ST_GeomFromText, ST_LineMerge, ST_MakePolygon

7.3.13 ST_TileEnvelope

ST_TileEnvelope — Erzeugt ein rechteckiges Polygon in Web Mercator (SRID:3857) unter Verwendung des XYZ-Kachelsystems.

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);

Beschreibung

Erzeugt ein rechteckiges Polygon, das die Ausdehnung einer Kachel im XYZ-Kachelsystem angibt. Die Kachel wird durch die Zoomstufe Z und den XY-Index der Kachel im Raster auf dieser Stufe angegeben. Kann verwendet werden, um die Kachelgrenzen zu definieren, die von ST_AsMVTGeom benötigt werden, um Geometrie in den MVT-Kachelkoordinatenraum zu konvertieren.

Standardmäßig ist die Kachelhülle im Web Mercator-Koordinatensystem (SRID:3857) unter Verwendung des Standardbereichs des Web Mercator-Systems (-20037508.342789, 20037508.342789). Dies ist das am häufigsten verwendete Koordinatensystem für MVT-Kacheln. Der optionale Parameter `bounds` kann verwendet werden, um Kacheln in einem beliebigen Koordinatensystem zu erzeugen. Es handelt sich um eine Geometrie, die den SRID und die Ausdehnung des Quadrats "Zoomstufe Null" hat, in das das XYZ-Kachelsystem eingeschrieben ist.

Der optionale Parameter `margin` kann verwendet werden, um eine Kachel um den angegebenen Prozentsatz zu vergrößern. Z.B. `margin=0.125` vergrößert die Kachel um 12,5%, was bei einer Kachelgröße von 4096, wie sie in ST_AsMVTGeom verwendet wird, einem Puffer=512 entspricht. Dies ist nützlich, um einen Kachelpuffer zu erstellen, der Daten enthält, die außerhalb des sichtbaren Bereichs der Kachel liegen, deren Vorhandensein sich aber auf die Darstellung der Kachel auswirkt. Beispielsweise könnte ein Städtename (ein Punkt) in der Nähe eines Kachelrandes liegen, so dass seine Beschriftung auf zwei Kacheln gerendert

werden sollte, obwohl sich der Punkt im sichtbaren Bereich nur einer Kachel befindet. Wenn Sie erweiterte Kacheln in einer Abfrage verwenden, wird der Stadtpunkt in beide Kacheln aufgenommen. Verwenden Sie stattdessen einen negativen Wert, um die Kachel zu verkleinern. Werte unter -0,5 sind unzulässig, da die Kachel dann vollständig verschwinden würde. Geben Sie keinen Rand an, wenn Sie `ST_AsMVTGeom` verwenden. Siehe das Beispiel für ST_AsMVT.

Erweiterung: 2.0.0 Standardwert für den optionalen Parameter SRID eingefügt.

Verfügbarkeit: 2.1.0

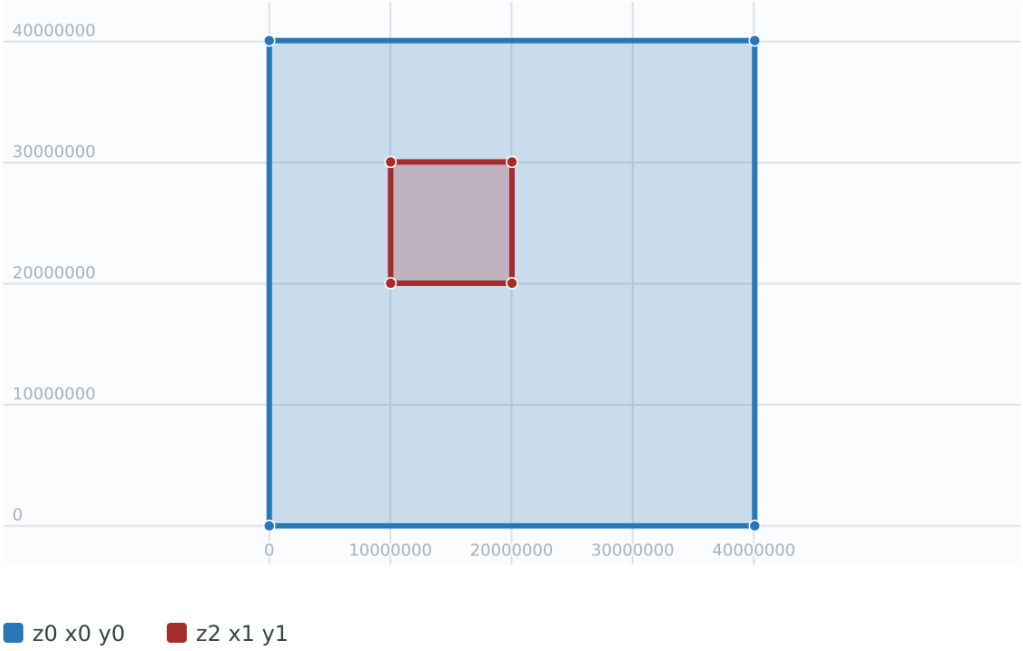
Beispiele

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

Code

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

Figure



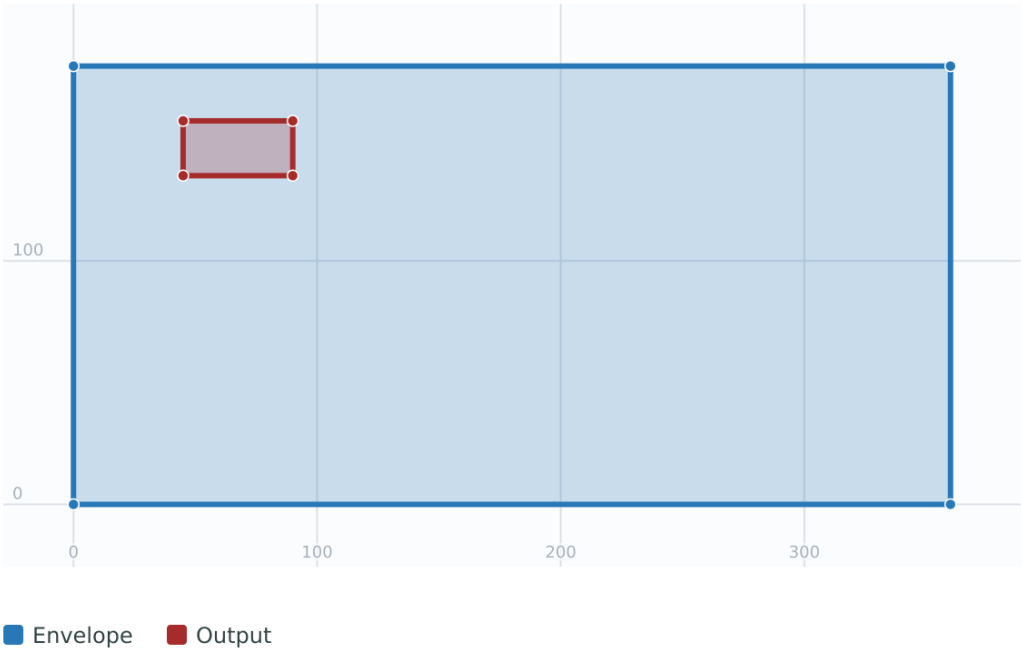
Code

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

Output

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

Figure



Siehe auch

ST_MakeEnvelope

7.3.14 ST_HexagonGrid

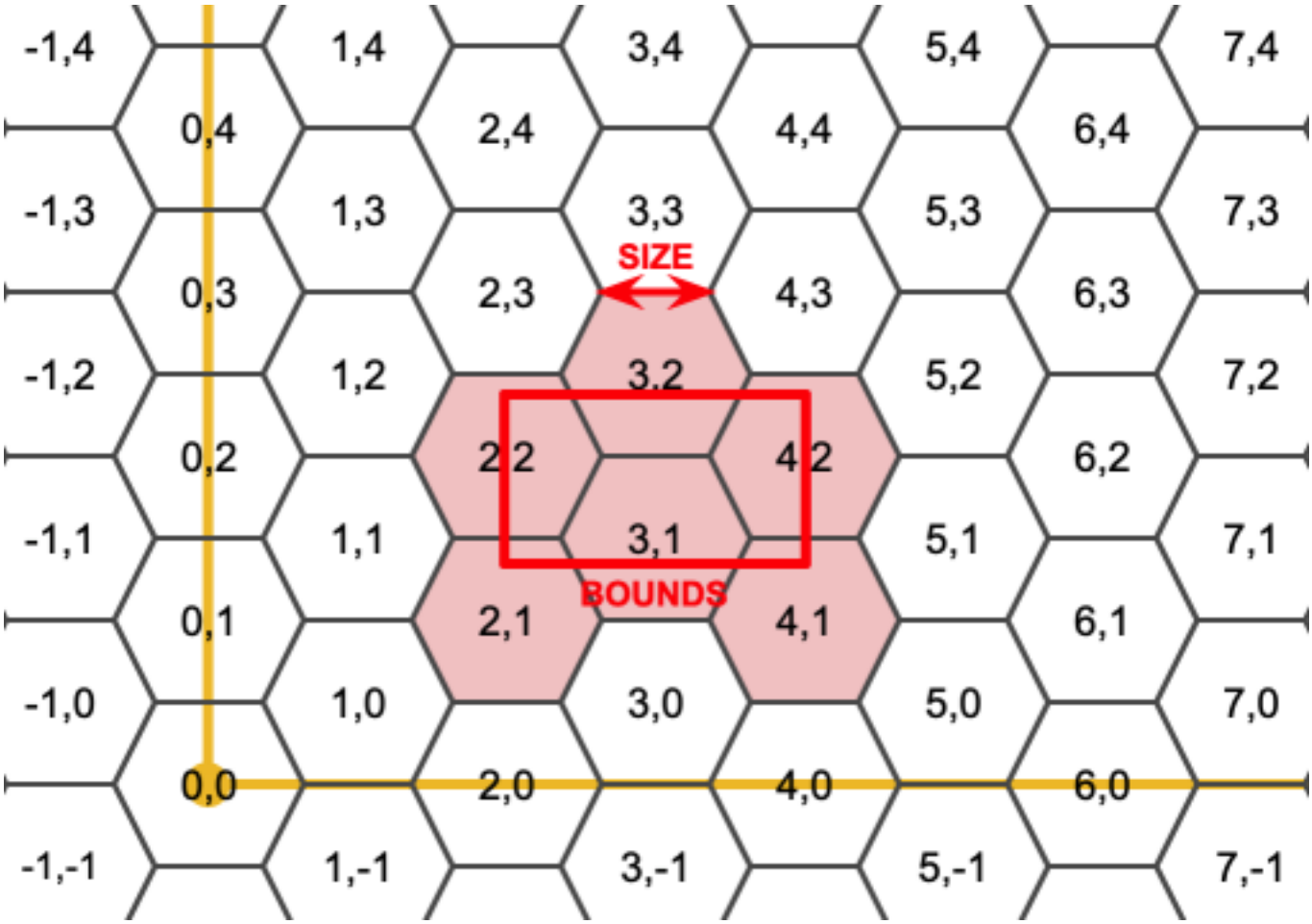
ST_HexagonGrid — Gibt eine Menge von Sechsecken und Zellindizes zurück, die die Grenzen des Arguments Geometrie vollständig abdecken.

Synopsis

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

Beschreibung

Starts with the concept of a hexagon tiling of the plane. (Not a hexagon tiling of the globe; for H3 use the h3-pg extension.) For a given planar SRS, and a given edge size, starting at the origin of the SRS, there is one unique hexagonal tiling of the plane, Tiling(SRS, Size). This function answers the question: what hexagons in a given Tiling(SRS, Size) overlap with a given bounds.



Die SRS für die Ausgangssechsecke ist die SRS, die durch die Begrenzungsgeometrie bereitgestellt wird.

Durch Verdoppelung oder Verdreifachung der Kantengröße des Sechsecks wird eine neue übergeordnete Kachel erzeugt, die zur ursprünglichen Kachel passt. Leider ist es nicht möglich, übergeordnete Sechseckkacheln zu erzeugen, in die die untergeordneten Kacheln perfekt hineinpassen.

Verfügbarkeit: 2.1.0

Beispiele

Unit hexagons covering rectangular bounds.

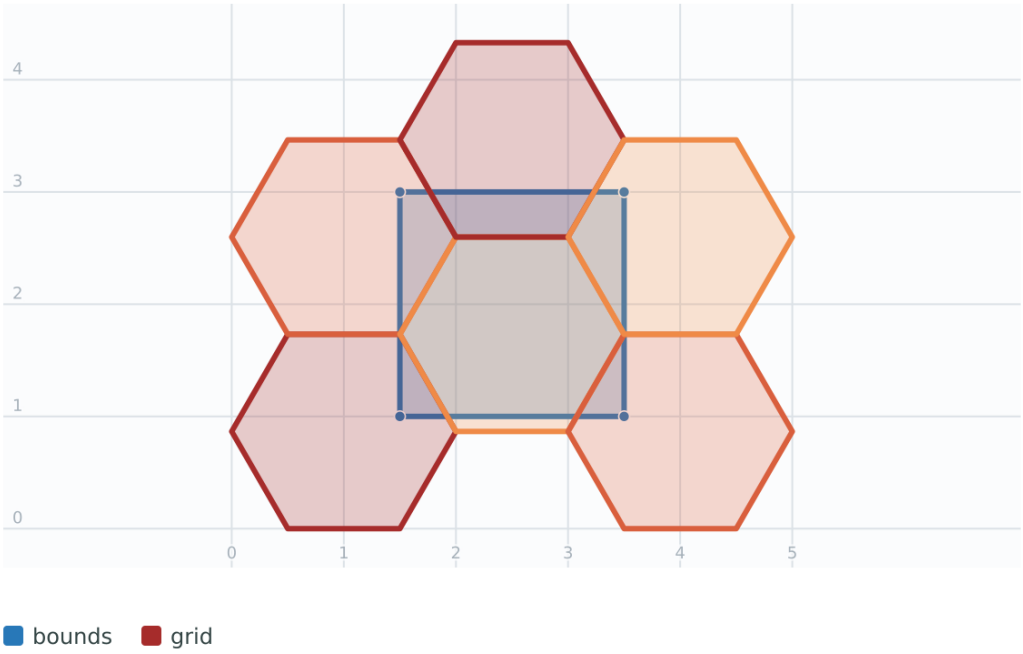
Code

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

Output

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

Figure

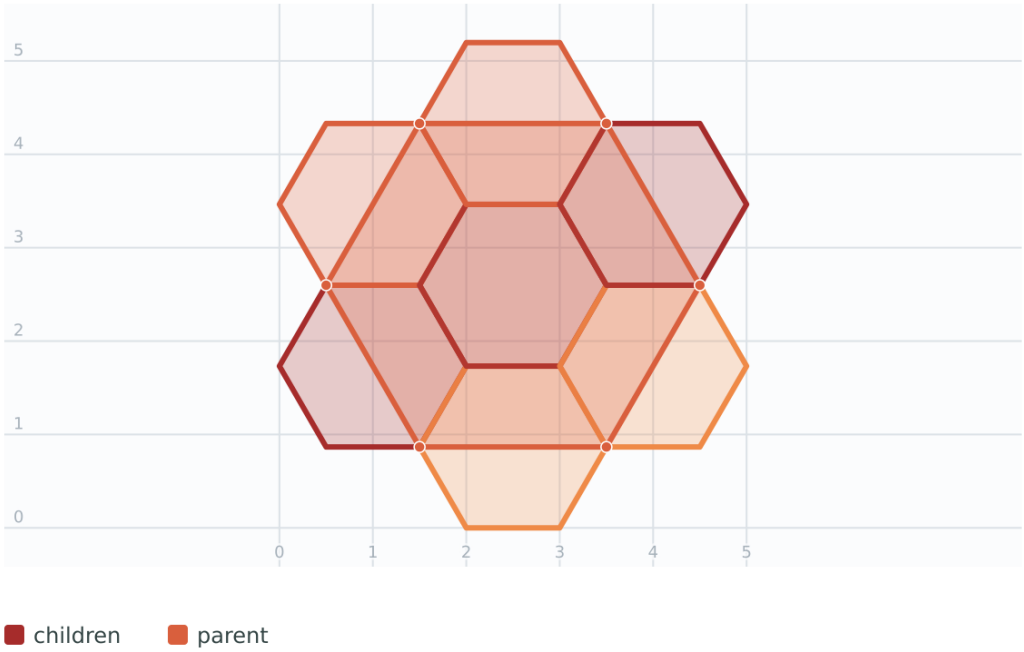


Child tiles that intersect a doubled-size parent hexagon share the tiling origin, but do not nest perfectly inside the parent.

Code

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

Figure



Counting points in hexagons.

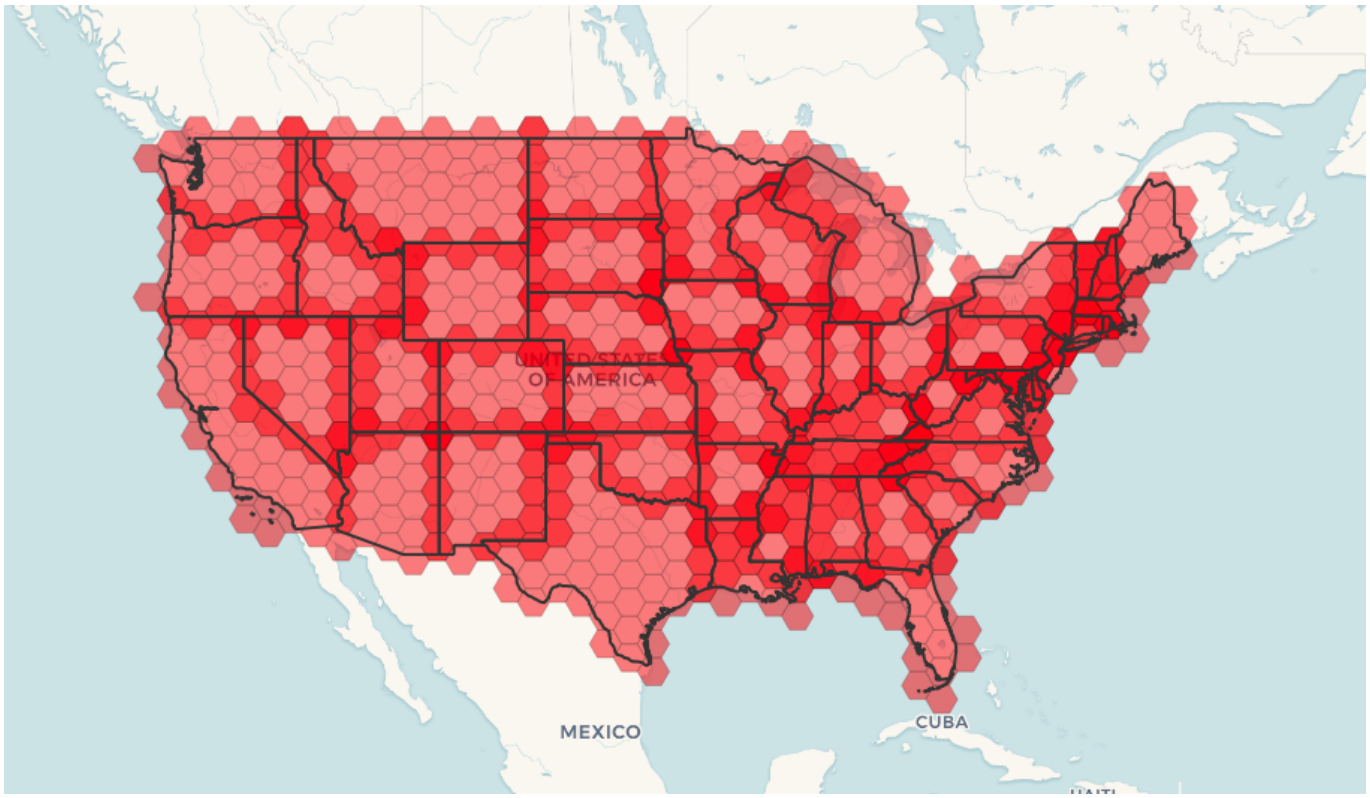
Um eine Punktzusammenfassung anhand eines sechseckigen Kachels vorzunehmen, erstellen Sie ein Sechseckgitter, wobei Sie die Ausdehnung der Punkte als Grenzen verwenden, und verbinden Sie es dann räumlich mit diesem Gitter.

Code

```
SELECT COUNT(*), hexes.geom
FROM ST_HexagonGrid(
  10000,
  ST_SetSRID(ST_EstimatedExtent('pointtable', 'geom'), 3857)
) AS hexes
INNER JOIN pointtable AS pts
  ON ST_Intersects(pts.geom, hexes.geom)
GROUP BY hexes.geom;
```

Generating hex coverage of polygons.

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



Code

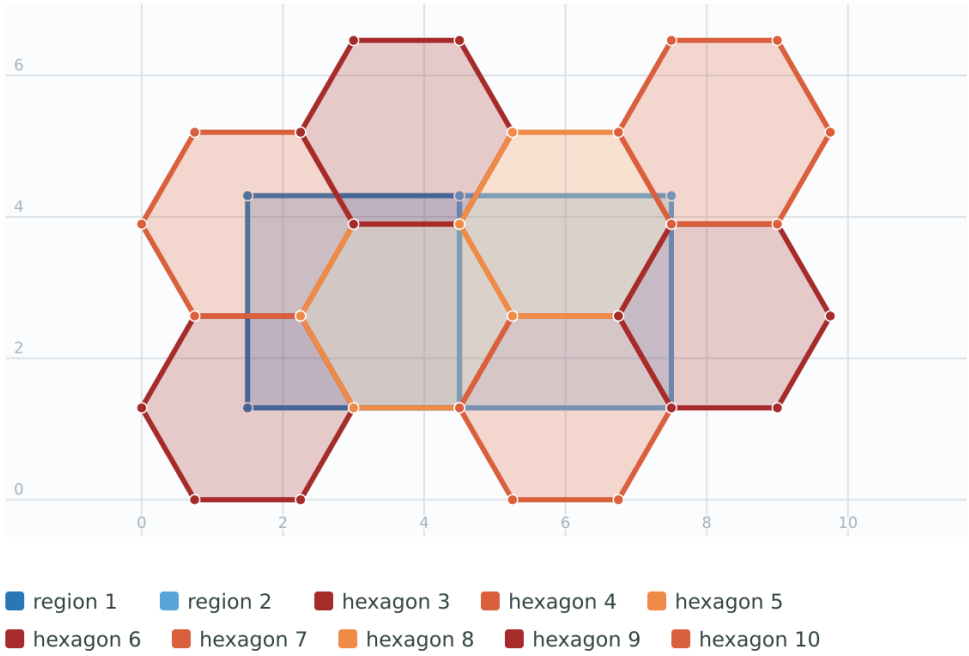
```
WITH admin1(gid, geom) AS (
    VALUES
        (1, ST_MakeEnvelope(0, 0, 3, 3, 3857)),
        (2, ST_MakeEnvelope(3, 0, 6, 3, 3857))
), coverage AS (
    SELECT hex.i, hex.j, hex.geom
    FROM admin1
    CROSS JOIN LATERAL ST_HexagonGrid(1.5, admin1.geom) AS hex
    WHERE ST_Intersects(admin1.geom,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

Output

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

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

Figure



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



Anmerkung
Das Schlüsselwort LATERAL wird für Funktionen mit Mengenrückgabe impliziert, wenn auf eine vorherige Tabelle in der FROM-Liste verwiesen wird. CROSS JOIN LATERAL, CROSS JOIN oder einfach nur , sind also gleichwertige Konstrukte für dieses Beispiel.

Code

```
SELECT admin1.gid, hex.geom
FROM
    admin1
    CROSS JOIN
    ST_HexagonGrid(100000, admin1.geom) AS hex
WHERE
    adm0_a3 = 'USA'
    AND
    ST_Intersects(admin1.geom, hex.geom)
```

Siehe auch

ST_EstimatedExtent, ST_MakePoint, ST_Point, ST_SRID

7.3.15 ST_Hexagon

ST_Hexagon — Liefert ein einzelnes Sechseck unter Verwendung der angegebenen Kantengröße und Zellkoordinate innerhalb des Sechseck-Gitterraums.

Synopsis

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

Beschreibung

Verwendet dasselbe Sechseck-Kacheln-Konzept wie ST_HexagonGrid, erzeugt aber nur ein Sechseck an der gewünschten Zellkoordinate. Optional kann die Ursprungskoordinate der Kacheln angepasst werden, der Standardursprung liegt bei 0,0.

Sechsecke werden ohne SRID generiert. Verwenden Sie daher ST_SetSRID, um den SRID auf den von Ihnen erwarteten Wert zu setzen.

Verfügbarkeit: 2.1.0

Beispiele

Creating a hexagon at the origin.

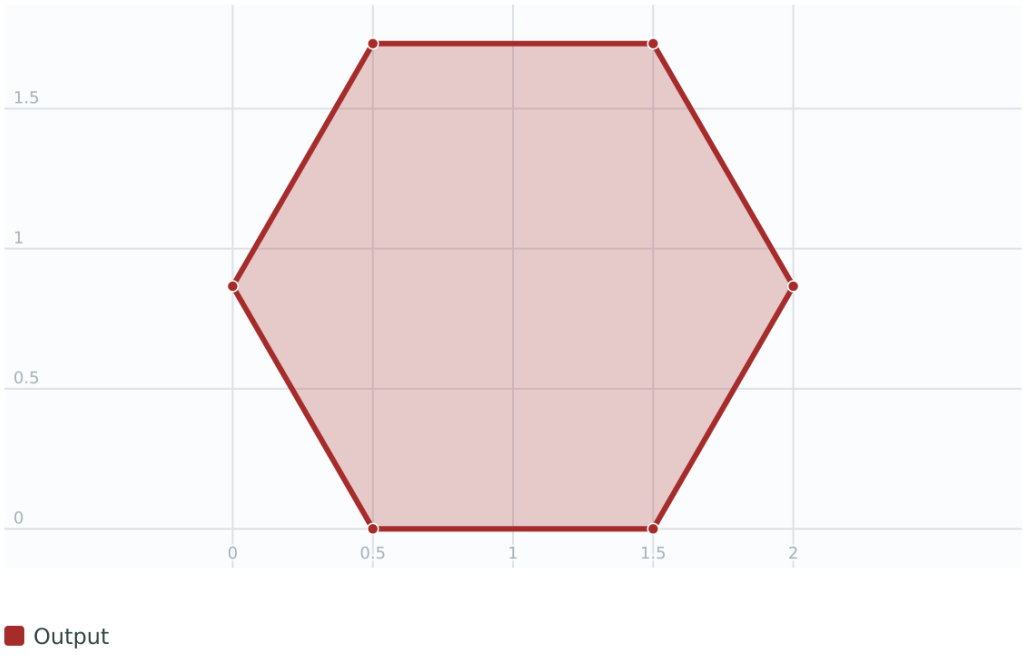
Code

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

Output

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

Figure



Siehe auch

ST_TileEnvelope, ST_MakePoint, ST_SetSRID

7.3.16 ST_SquareGrid

ST_SquareGrid — Gibt eine Menge von Gitterquadraten und Zellindizes zurück, die die Grenzen des Arguments Geometrie vollständig abdecken.

Synopsis

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

Beschreibung

Beginnt mit dem Konzept einer quadratischen Kachelung der Ebene. Für eine gegebene planare SRS und eine gegebene Kantengröße gibt es, ausgehend vom Ursprung der SRS, ein einziges quadratisches Tiling der Ebene, Tiling(SRS, Size). Diese Funktion beantwortet die Frage, welche Gitter in einem gegebenen Tiling(SRS, Size) sich mit einer gegebenen Grenze überlappen.

Die SRS für die Ausgangsquadrate ist die SRS, die durch die Begrenzungsgeometrie bereitgestellt wird.

Durch Verdoppelung der Größe des Quadrats oder der Kanten wird eine neue übergeordnete Kachel erzeugt, die perfekt zur ursprünglichen Kachel passt. Standard-Webmap-Kacheln in Mercator sind einfach Zweierpotenzen von quadratischen Gittern in der Mercator-Ebene.

Verfügbarkeit: 2.1.0

Beispiele

Generating a 1 degree grid for a country.

Das Raster füllt die gesamten Grenzen des Landes aus. Wenn Sie also nur Quadrate haben wollen, die das Land berühren, müssen Sie anschließend mit ST_Intersects filtern.

Code

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

Counting points in squares (using single chopped grid).

Um eine Punktzusammenfassung anhand einer quadratischen Kachel zu erstellen, erzeugen Sie ein quadratisches Gitter, das die Ausdehnung der Punkte als Grenzen verwendet, und verbinden Sie es dann räumlich mit diesem Gitter. Beachten Sie, dass die geschätzte Ausdehnung von der tatsächlichen Ausdehnung abweichen kann, seien Sie also vorsichtig und stellen Sie zumindest sicher, dass Sie Ihre Tabelle analysiert haben.

Code

```
SELECT COUNT(*), squares.geom
FROM pointtable AS pts
INNER JOIN ST_SquareGrid(
  1000,
  ST_SetSRID(ST_EstimatedExtent('pointtable', 'geom'), 3857)
) AS squares
  ON ST_Intersects(pts.geom, squares.geom)
GROUP BY squares.geom;
```

Counting points in squares using set of grid per point.

Dies führt zum gleichen Ergebnis wie das erste Beispiel, ist aber bei einer großen Anzahl von Punkten langsamer

Code

```
SELECT COUNT(*), squares.geom
FROM pointtable AS pts
INNER JOIN ST_SquareGrid(
  1000,
  pts.geom
) AS squares
  ON ST_Intersects(pts.geom, squares.geom)
GROUP BY squares.geom;
```

Siehe auch

ST_TileEnvelope, ST_Point, ST_SetSRID, ST_SRID

7.3.17 ST_Square

ST_Square — Gibt ein einzelnes Quadrat mit der angegebenen Kantengröße und Zellkoordinate innerhalb des quadratischen Gitterraums zurück.

Synopsis

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

Beschreibung

Verwendet dasselbe Konzept der quadratischen Kachelung wie ST_SquareGrid, erzeugt aber nur ein Quadrat an der gewünschten Zellkoordinate. Optional kann die Ursprungskoordinate der Kacheln angepasst werden, der Standardursprung liegt bei 0,0.

Squares are generated with the SRID of the given origin. Use ST_SetSRID to set the SRID if the given origin has an unknown SRID (as is the case by default).

Verfügbarkeit: 2.1.0

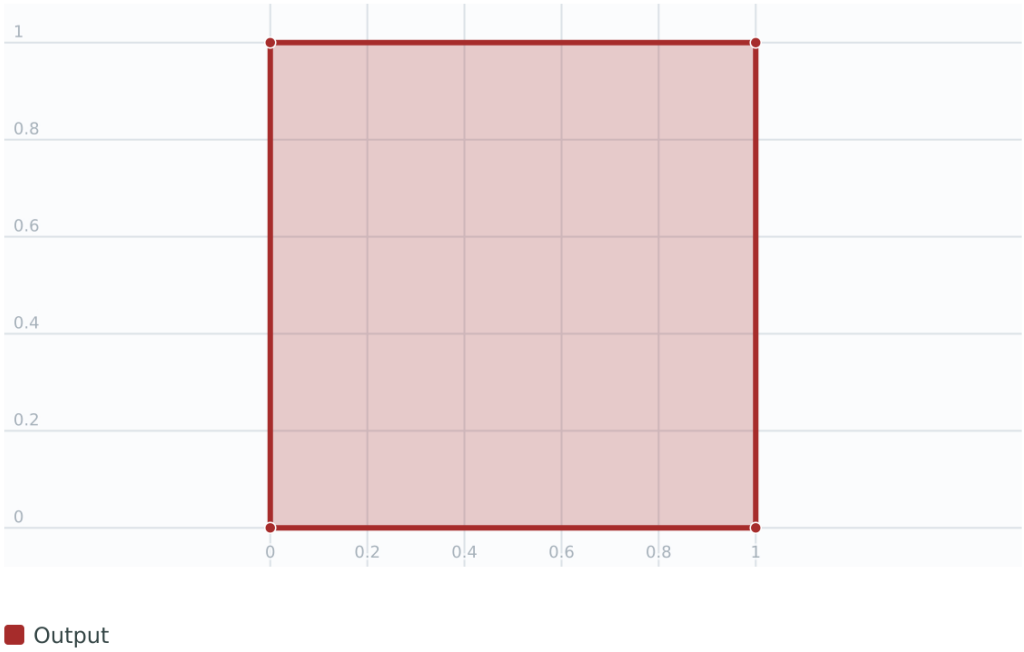
Beispiele

Creating a square at the origin.

Code

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

Figure



Siehe auch

ST_TileEnvelope, ST_MakeLine, ST_MakePolygon

7.3.18 ST_Letters

ST_Letters — Gibt die eingegebenen Buchstaben als Geometrie mit einer Standardstartposition am Ursprung und einer Standardtexthöhe von 100 zurück.

Synopsis

geometry **ST_Letters**(text letters, json font);

Beschreibung

Verwendet eine eingebaute Schriftart, um eine Zeichenkette als Multipolygongeometrie darzustellen. Die Standard Texthöhe ist 100.0, der Abstand von der Unterkante einer Unterlänge bis zur Oberkante eines Großbuchstabens. Die Standard-Startposition setzt den Beginn der Grundlinie auf den Ursprung. Um die Schriftart zu überschreiben, muss eine json-Map mit einem Zeichen als Schlüssel und base64-kodierten TWKB für die Schriftform übergeben werden, wobei die Schriftarten eine Höhe von 1000 Einheiten von der Unterkante der Unterlängen bis zur Oberkante der Großbuchstaben haben.

Der Text wird standardmäßig am Ursprung erzeugt. Um den Text neu zu positionieren und in der Größe zu verändern, wenden Sie zuerst die Funktion `ST_Scale` und dann die Funktion `ST_Translate` an.

Verfügbarkeit: 3.3.0

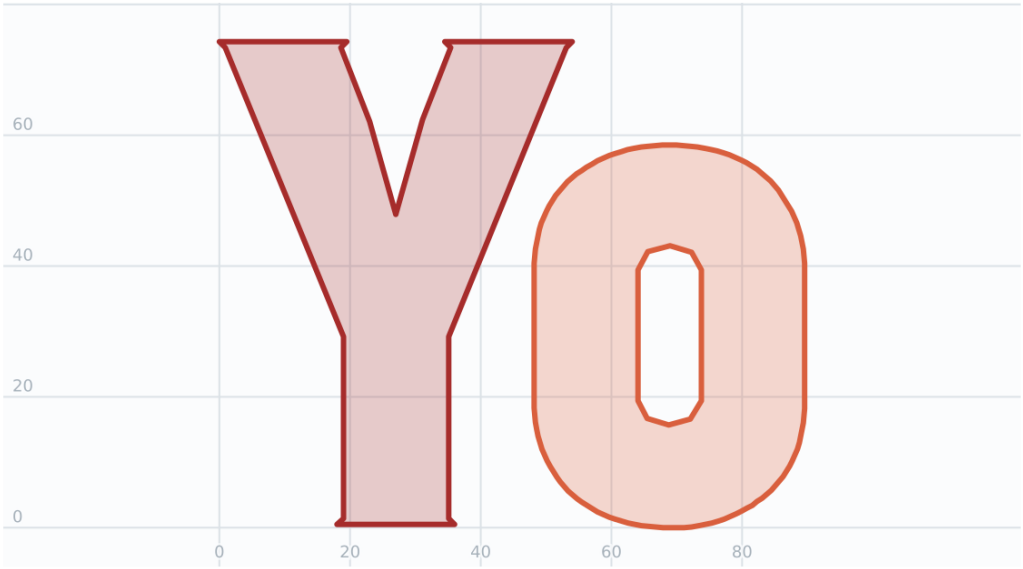
Beispiele

Generating the word 'Yo'.

Code

```
SELECT ST_Letters('Yo');
```

Figure



Output

Generating the word 'PostGIS'.

Code

```
SELECT ST_Letters('PostGIS') AS letters;
```

Figure



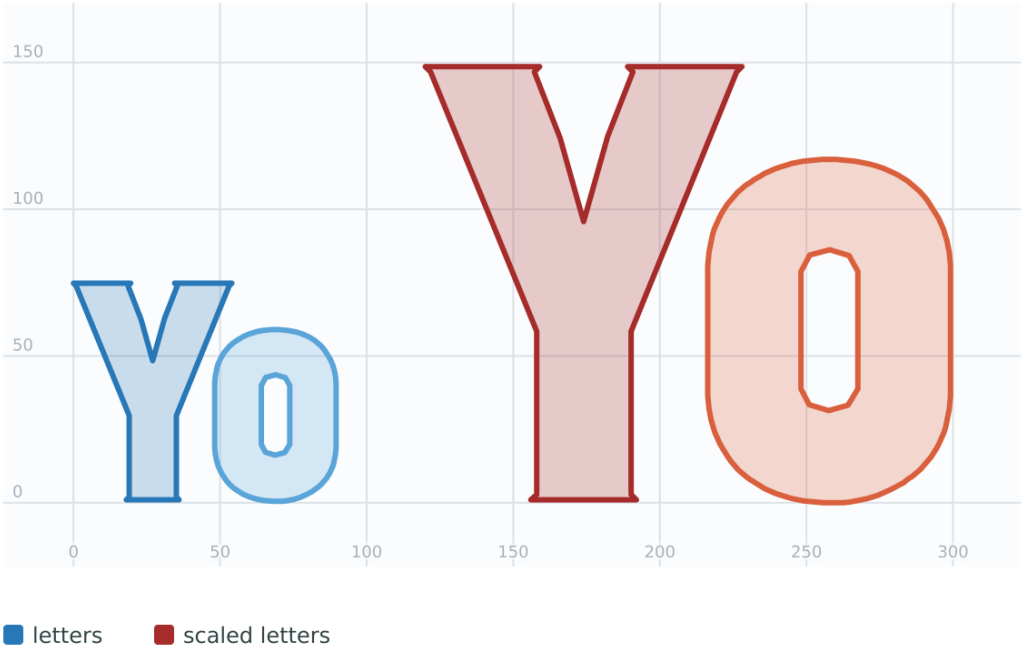
Output

Scaling and moving words.

Code

```
WITH letters AS (
    SELECT ST_Letters('Yo') AS geom
)
SELECT geom AS input_letters,
       ST_Translate(ST_Scale(geom, 2, 2), 120, 0)
           AS scaled_letters
FROM letters;
```

Figure



Siehe auch

ST_AsTWKB, ST_Scale, ST_Translate

7.3.19 ST_MakeNurbsCurve

ST_MakeNurbsCurve — Creates a NURBS (Non-Uniform Rational B-Spline) curve from control points, optional weights, and optional knot vector.

Synopsis

geometry **ST_MakeNurbsCurve**(integer degree, geometry control_points, float8[] weights, float8[] knots);

Beschreibung

Creates a NURBS (Non-Uniform Rational B-Spline) curve geometry from the specified parameters. NURBS curves are smooth parametric curves that provide precise control over curve shape through control points, weights, and knot values.

The `degree` parameter specifies the polynomial degree of the curve (must be between 1 and 10). Higher degrees produce smoother curves.

The `control_points` must be a LINESTRING geometry defining the control polygon. The number of control points must be at least `degree` + 1.

The optional `weights` array assigns a weight to each control point. All weights must be positive. If NULL or omitted, uniform weights (1.0) are used, resulting in a non-rational B-spline.

The optional `knots` array defines the parameter space. If NULL or omitted, a uniform clamped knot vector is automatically generated. If provided, the knot vector must have exactly (`npoints` + `degree` + 1) elements and must be non-decreasing.

Availability: 3.7.0

Beispiele

This example creates a degree-2 NURBS curve with uniform weights.

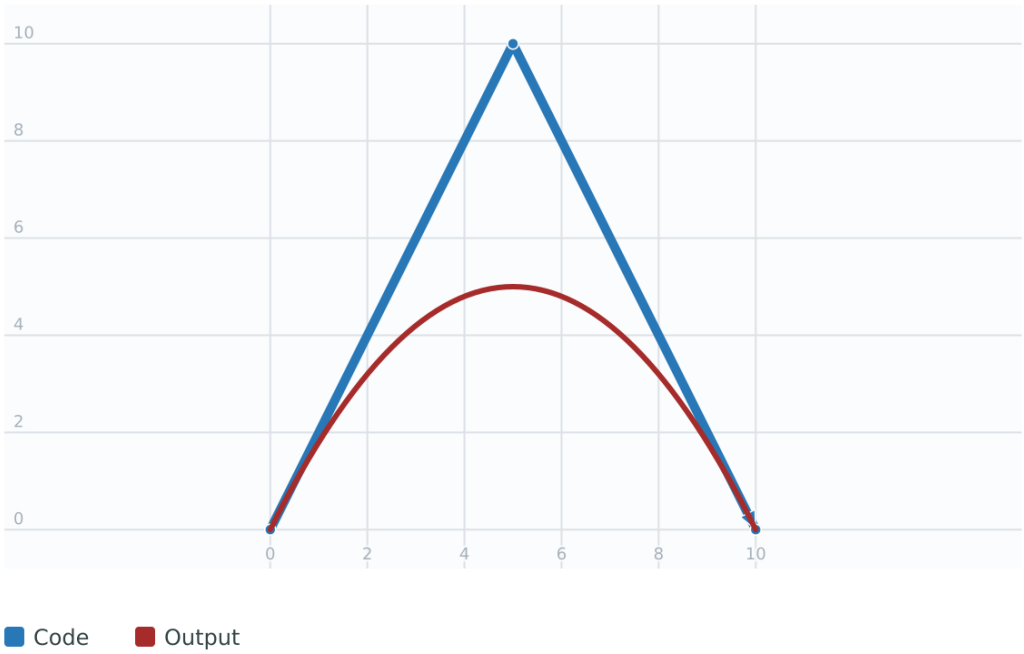
Code

```
SELECT ST_MakeNurbsCurve(
    2,
    'LINESTRING(0 0,5 10,10 0)'::geometry
);
```

Output

```
NURBSCURVE(DEGREE 2, CONTROLPOINTS(NURBSPOINT(WEIGHTEDPOINT(0 0),WEIGHT 1), NURBSPOINT(WEIGHTEDPOINT(5 10),WEIGHT 1),
    NURBSPOINT(WEIGHTEDPOINT(10 0),WEIGHT 1)),KNOTS (KNOT(0,3), KNOT(1,3)))
```

Figure



This example creates a rational NURBS curve with custom weights.

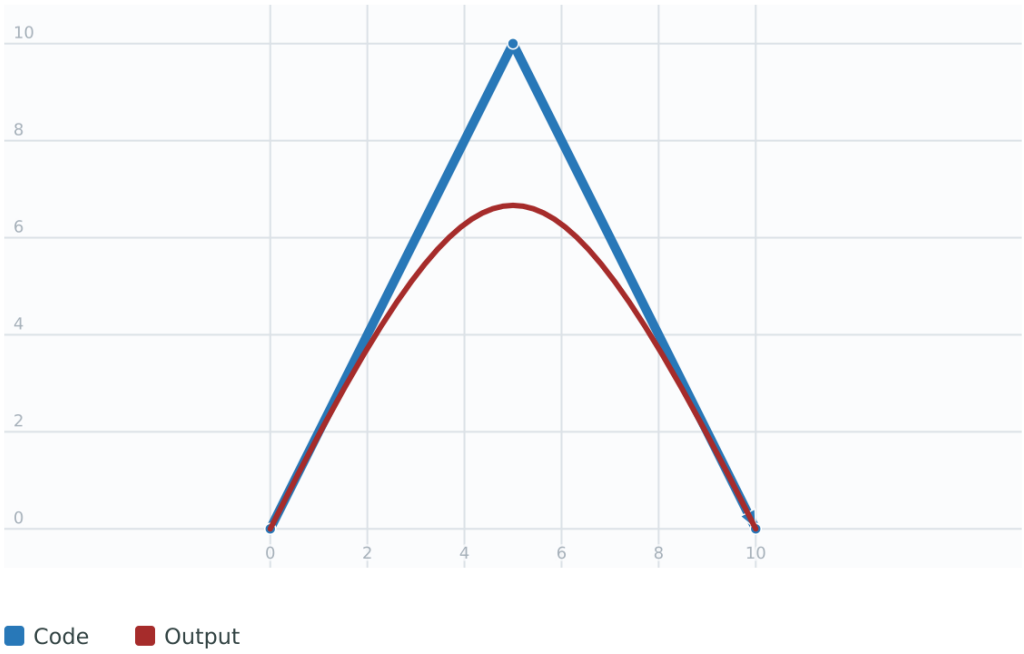
Code

```
SELECT ST_MakeNurbsCurve(
    2,
    'LINESTRING(0 0,5 10,10 0)'::geometry,
    ARRAY[1.0, 2.0, 1.0]
);
```

Output

```
NURBSCURVE(DEGREE 2, CONTROLPOINTS(NURBSPOINT(WEIGHTEDPOINT(0 0),WEIGHT 1), NURBSPOINT(WEIGHTEDPOINT(5 10),WEIGHT 2),
    NURBSPOINT(WEIGHTEDPOINT(10 0),WEIGHT 1)),KNOTS (KNOT(0,3), KNOT(1,3)))
```

Figure



This example creates a NURBS curve with a non-uniform knot vector.

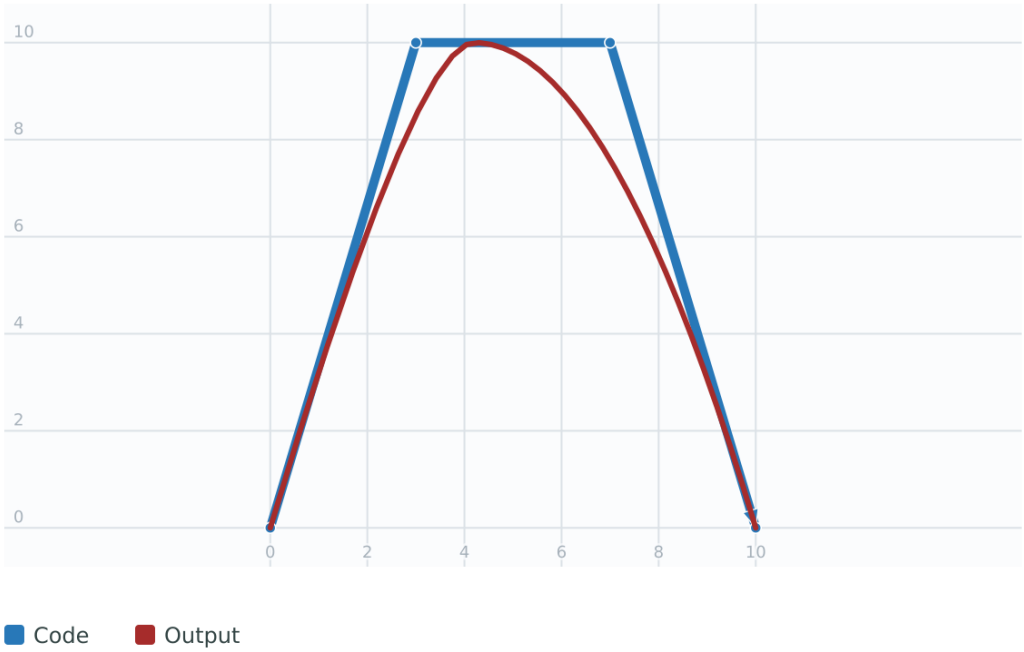
Code

```
SELECT ST_MakeNurbsCurve(
    2,
    'LINESTRING(0 0,3 10,7 10,10 0)'::geometry,
    ARRAY[1.0, 1.0, 1.0, 1.0],
    ARRAY[0.0, 0.0, 0.0, 0.3, 1.0, 1.0, 1.0]
);
```

Output

```
NURBSCURVE(DEGREE 2, CONTROLPOINTS(NURBSPOINT(WEIGHTEDPOINT(0 0),WEIGHT 1), NURBSPOINT(WEIGHTEDPOINT(3 10),WEIGHT 1),
    NURBSPOINT(WEIGHTEDPOINT(7 10),WEIGHT 1), NURBSPOINT(WEIGHTEDPOINT(10 0),WEIGHT 1)),KNOTS (KNOT(0,3), KNOT(0.3,1),
     KNOT(1,3)))
```

Figure



Siehe auch

ST_NurbsToLineString, ST_Evaluate, ST_ControlPoints

7.4 Geometrische Zugriffsfunktionen

7.4.1 GeometryType

GeometryType — Gibt den Geometrietyp des ST_Geometry Wertes zurück.

Synopsis

text **GeometryType**(geometry geomA);

Beschreibung

Gibt den Geometrietyp als Zeichenkette zurück. z.B.: 'LINESTRING', 'POLYGON', 'MULTIPOINT', etc.

OGC SPEC s2.1.1.1 - Gibt den Namen des instanziierbaren Subtyps der Geometrie zurück, von dem die geometrische Instanz ein Mitglied ist. Der Name des instanziierbaren Subtyps der Geometrie wird als Zeichenkette ausgegeben.



Anmerkung
Die Funktion zeigt auch an ob die Geometrie eine Maßzahl aufweist, indem eine Zeichenkette wie 'POINTM' zurückgegeben wird.

Erweiterung: Mit 2.0.0 wurde die Unterstützung für polyedrische Oberflächen, Dreiecke und TIN eingeführt.

Diese Methode implementiert die OGC Simple Features Implementation Specification for SQL 1.1.

Diese Methode unterstützt kreisförmige Strings und Kurven.

Diese Funktion unterstützt 3d und lässt den Z-Index nicht fallen.

Diese Funktion unterstützt polyedrische Flächen.

Diese Funktion unterstützt Dreiecke und dreieckige unregelmäßige Netzoberflächen (TIN).

Beispiele

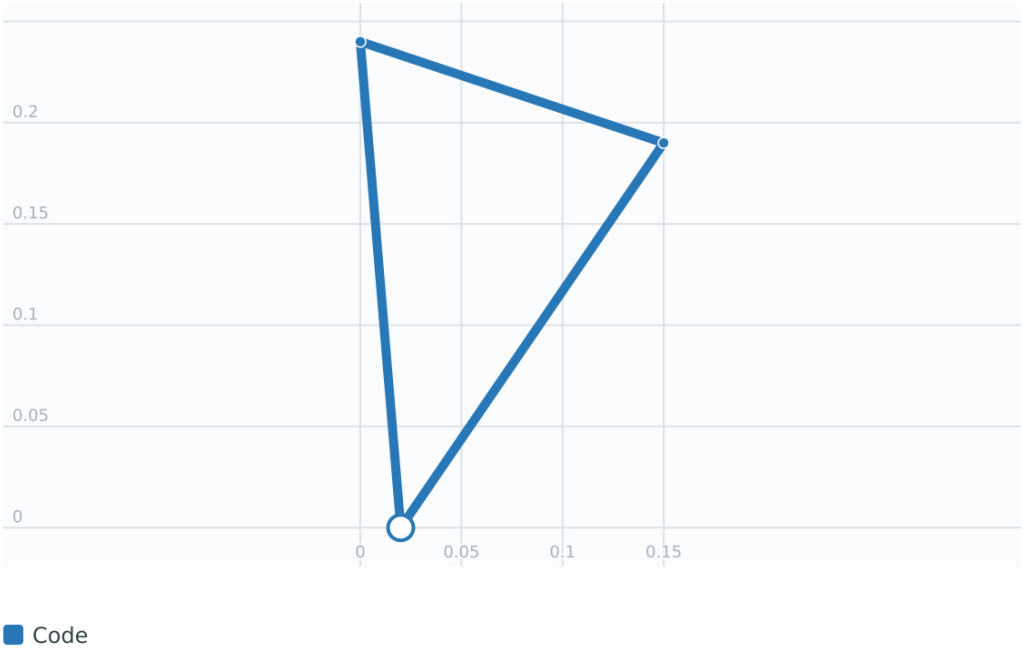
Code

```
SELECT GeometryType('LINESTRING(77.29 29.07,77.42 29.26,77.27 29.31,77.29 29.07)'::geometry) ↩
    ;
```

Output

```
LINESTRING
```

Figure



Code

```
SELECT ST_GeometryType('POLYHEDRALSURFACE( ((0 0 0,0 0 1,0 1 1,0 1 0,0 0 0)),
((0 0 0,0 1 0,1 1 0,1 0 0,0 0 0)),((0 0 0,1 0 0,1 0 1,0 0 1,0 0 0)),
((1 1 0,1 1 1,1 0 1,1 0 0,1 1 0)),
((0 1 0,0 1 1,1 1 1,1 1 0,0 1 0)),((0 0 1,1 0 1,1 1 1,0 1 1,0 0 1)) )');
    POLYHEDRALSURFACE
```

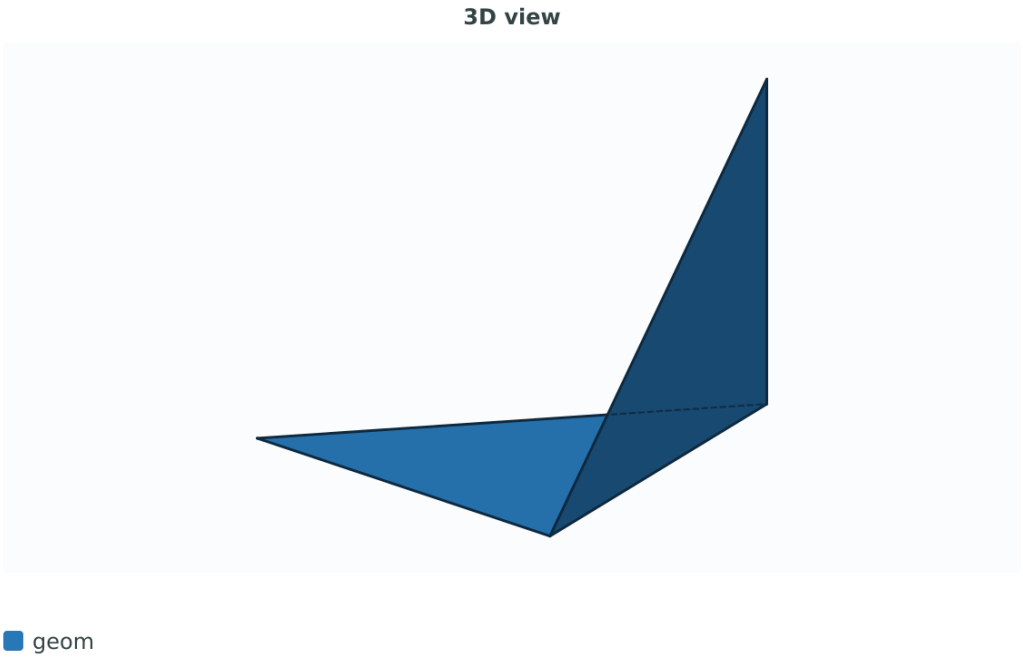
Code

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

Output

```
TIN
```

Figure



Siehe auch

ST_GeometryType

7.4.2 ST_Boundary

ST_Boundary — Gibt die abgeschlossene Hülle aus der kombinierten Begrenzung der Geometrie zurück.

Synopsis

geometry **ST_Boundary**(geometry geomA);

Beschreibung

Gibt die abgeschlossene Hülle aus der kombinierten Begrenzung der Geometrie zurück. Die Definition der kombinierte Begrenzung ist in Abschnitt 3.12.3.2 der OGC SPEC beschrieben. Da das Ergebnis dieser Funktion eine abgeschlossene Hülle und daher topologisch geschlossen ist, kann die resultierende Begrenzung durch geometrische Primitive, wie in Abschnitt 3.12.2. der OGC SPEC erörtert, dargestellt werden.

Wird durch das GEOS Modul ausgeführt



Anmerkung
Vor 2.0.0 meldete diese Funktion einen Fehler, falls sie auf eine `GEOMETRYCOLLECTION` angewandt wurde. Ab 2.0.0 wird stattdessen NULL (nicht unterstützte Eingabe) zurückgegeben.

 Diese Methode implementiert die OGC Simple Features Implementation Specification for SQL 1.1. OGC SPEC s2.1.1.1

 Diese Methode setzt die SQL/MM-Spezifikation um. SQL-MM IEC 13249-3: 5.1.17

 Diese Funktion unterstützt 3d und lässt den Z-Index nicht fallen.

Erweiterung: mit 2.1.0 wurde die Unterstützung von Dreiecken eingeführt

Geändert: 3.2.0 Unterstützung für TIN, verwendet keine Geos, linearisiert keine Kurven

Beispiele

LineString with boundary points overlaid.

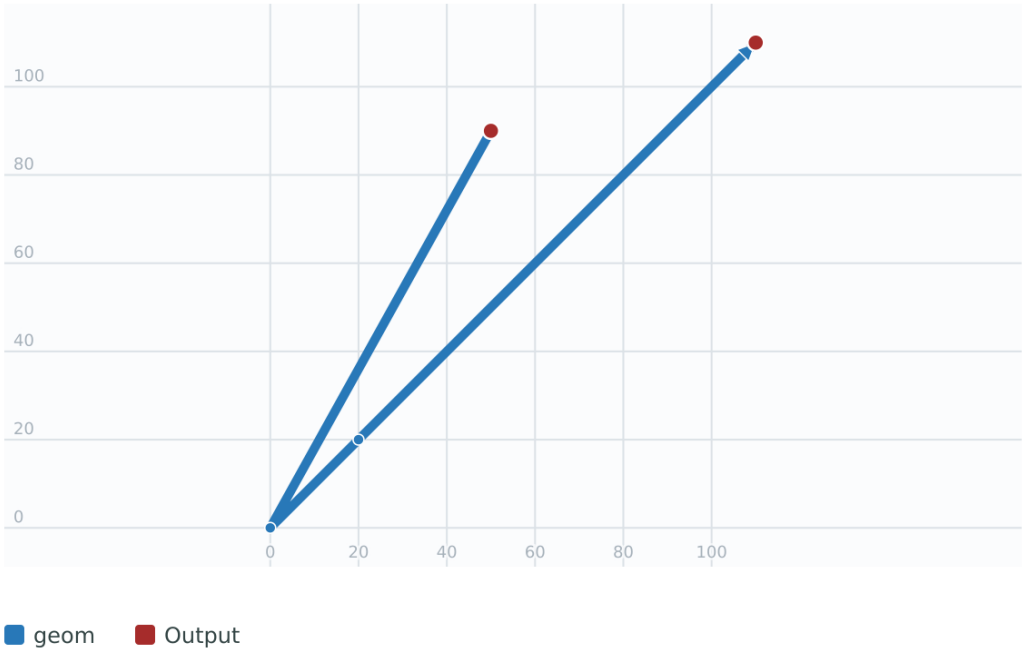
Code

```
SELECT ST_Boundary(geom)
FROM (SELECT 'LINESTRING(100 150,50 60,70 80,160 170)'::geometry As geom) As f;
```

Output

```
MULTIPOINT((100 150),(160 170))
```

Figure



Polygon holes with the boundary MultiLineString overlaid.

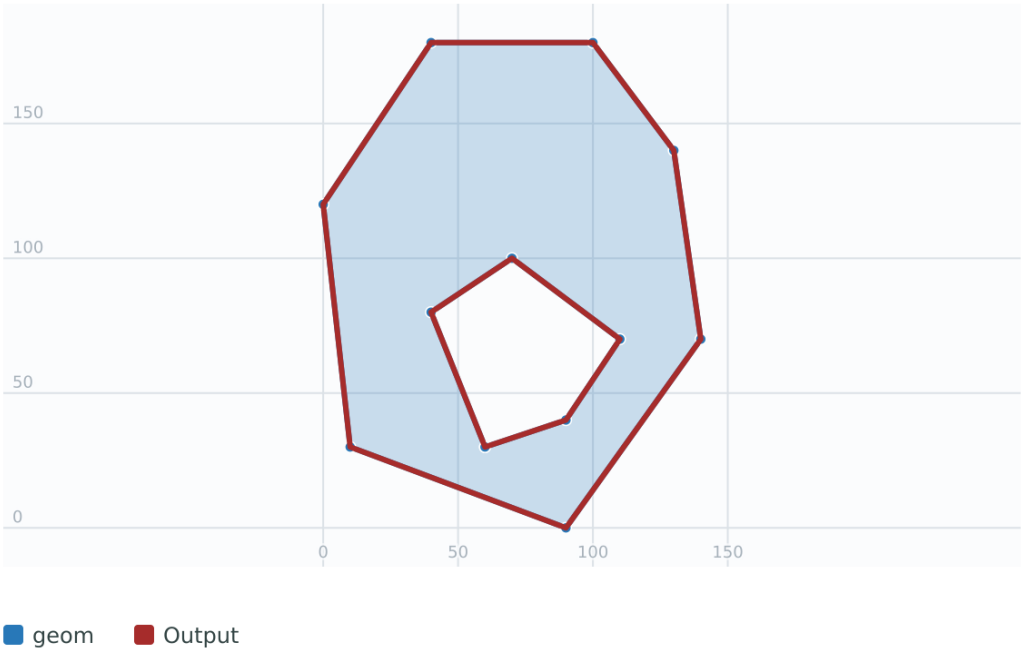
Code

```
SELECT ST_Boundary(geom)
FROM (SELECT
'POLYGON (( 10 130,50 190,110 190,140 150,150 80,100 10,20 40,10 130 ),
    ( 70 40,100 50,120 80,80 110,50 90,70 40 ))'::geometry As geom) As f;
```

Output

```
MULTILINESTRING((10 130,50 190,110 190,140 150,150 80,100 10,20 40,10 130),
        (70 40,100 50,120 80,80 110,50 90,70 40))
```

Figure



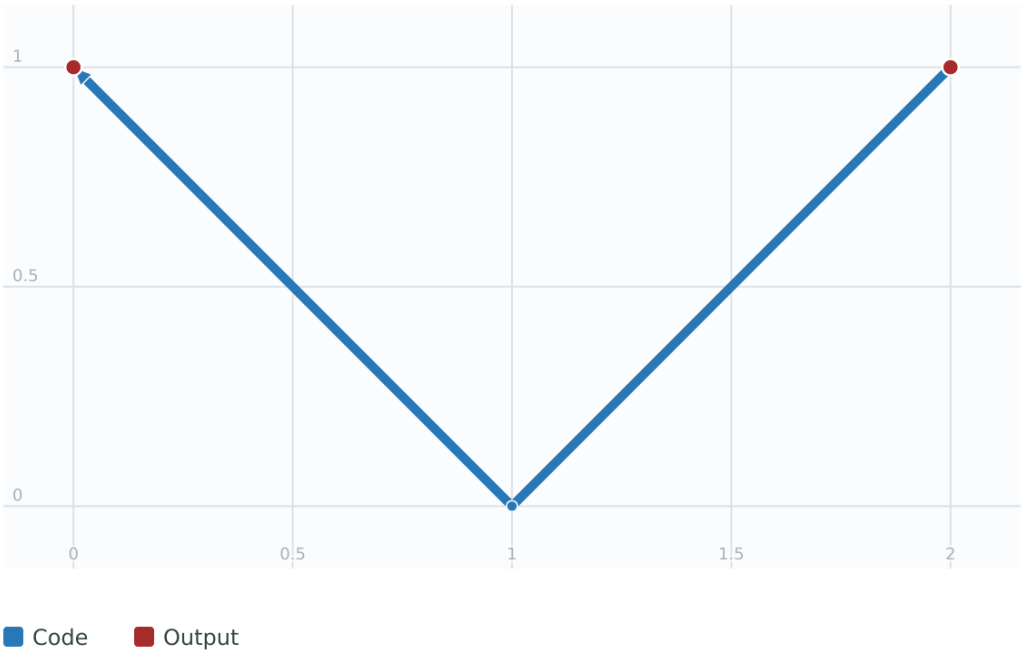
Code

```
SELECT ST_Boundary('LINESTRING(1 1,0 0,-1 1)');
```

Output

```
MULTIPOINT((1 1),(-1 1))
```

Figure



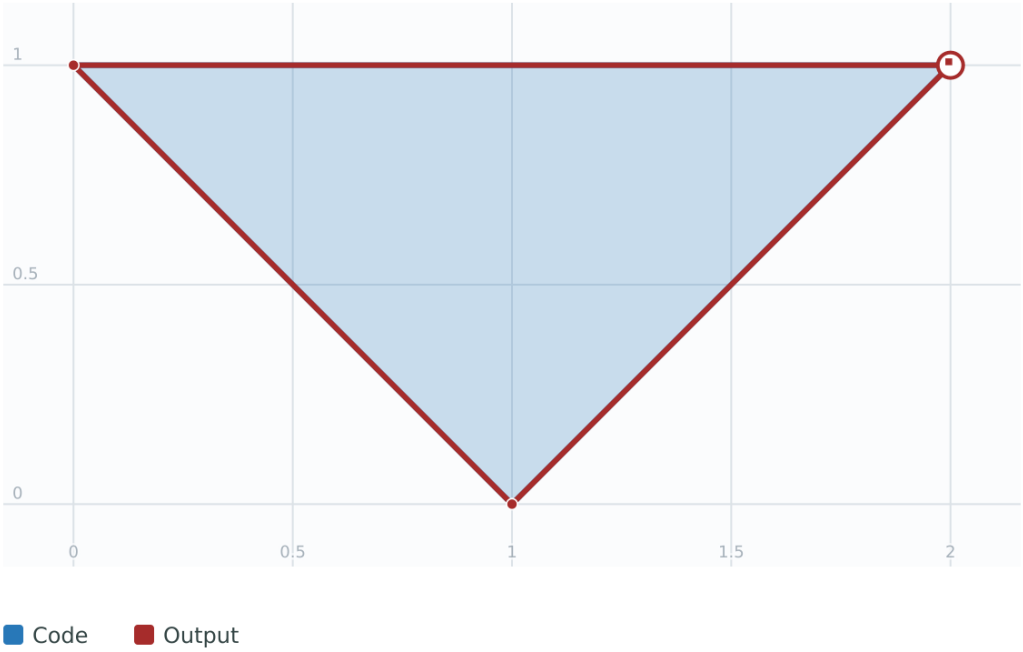
Code

```
SELECT ST_Boundary('POLYGON((1 1,0 0,-1 1,1 1))');
```

Output

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

Figure



This example uses a 3D polygon.

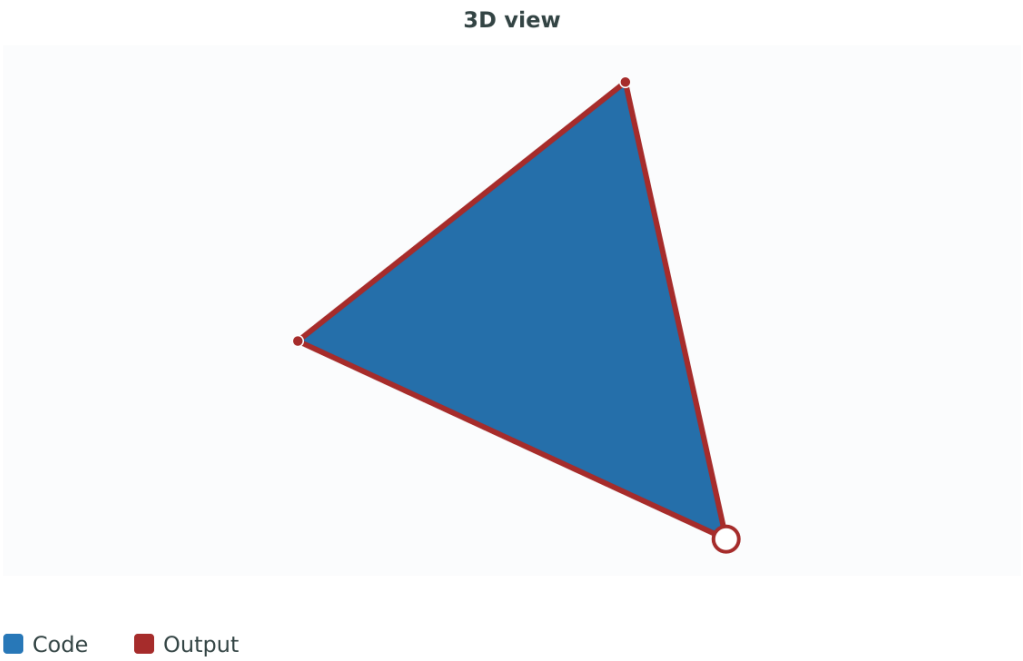
Code

```
SELECT ST_Boundary('POLYGON((1 1 1,0 0 1,-1 1 1,1 1 1))');
```

Output

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

Figure



This example uses a 3D MultiLineString.

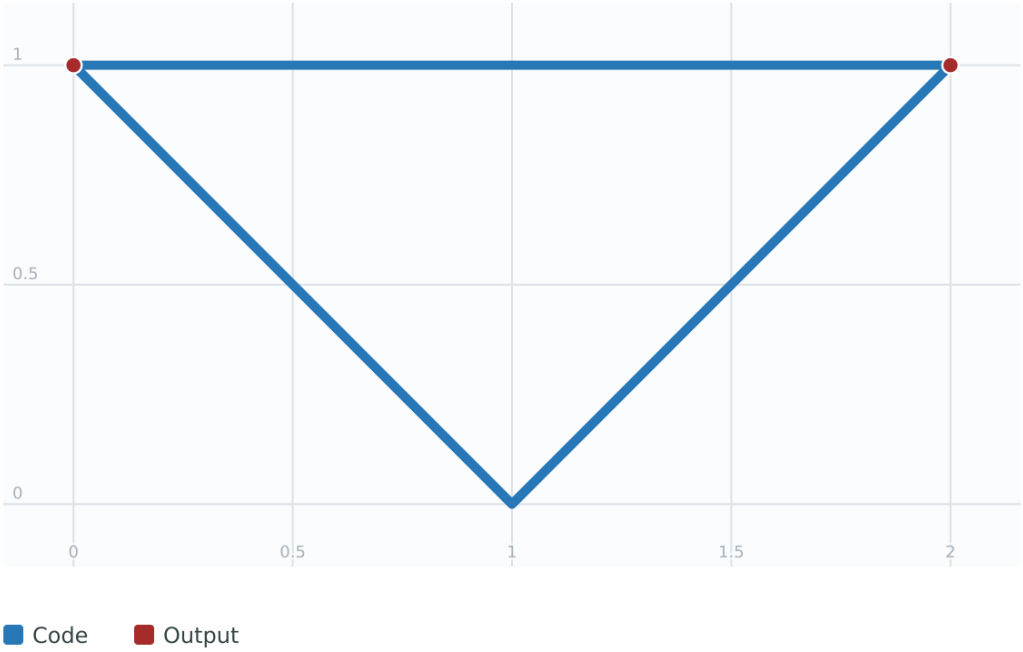
Code

```
SELECT ST_Boundary('MULTILINESTRING((1 1 1,0 0 0.5,-1 1 1),(1 1 0.5,0 0 0.5,-1 1 0.5,1 1 ↩
    0.5) )');
```

Output

```
MULTIPOINT(1 1 1,-1 1 1)
```

Figure



Siehe auch

ST_AsText, ST_ExteriorRing, ST_MakePolygon

7.4.3 ST_BoundingDiagonal

ST_BoundingDiagonal — Gibt die Diagonale des Umgebungsdreiecks der angegebenen Geometrie zurück.

Synopsis

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

Beschreibung

Gibt für eine angegebenen Geometrie die Diagonale des Umgebungsrechtecks als Linienzug zurück. Wenn die Geometrie leer ist, so ist auch die Diagonale Linie leer. Anderenfalls wird ein Linienzug aus 2 Punkten mit den kleinsten xy-Werten am Anfangspunkt und den größten xy-Werten am Endpunkt ausgegeben.

Der `fits` Parameter bestimmt ob die bestmögliche Anpassung notwendig ist. Wenn er FALSE ist, so kann auch die Diagonale eines etwas größeren Umgebungsrechtecks akzeptiert werden (dies ist für Geometrien mit vielen Knoten schneller). Auf jeden Fall wird immer die gesamte Eingabegeometrie durch das von der Diagonale bestimmten Umgebungsrechtecks abgedeckt.

Die zurückgegebene Linienzug-Geometrie beinhaltet immer die SRID und die Dimensionalität (Anwesenheit von Z und M) der eingegebenen Geometrie.



Anmerkung
Bei Spezialfällen (ein einzelner Knoten als Eingabewert) ist der zurückgegebene Linienzug topologisch ungültig (kein Inneres/Interior). Das Ergebnis ist dadurch jedoch nicht semantisch ungültig.

Verfügbarkeit: 2.2.0

 Diese Funktion unterstützt 3d und lässt den Z-Index nicht fallen.

 Diese Funktion unterstützt M-Koordinaten.

Beispiele

This example shows a polygon together with the diagonal of its bounding box.

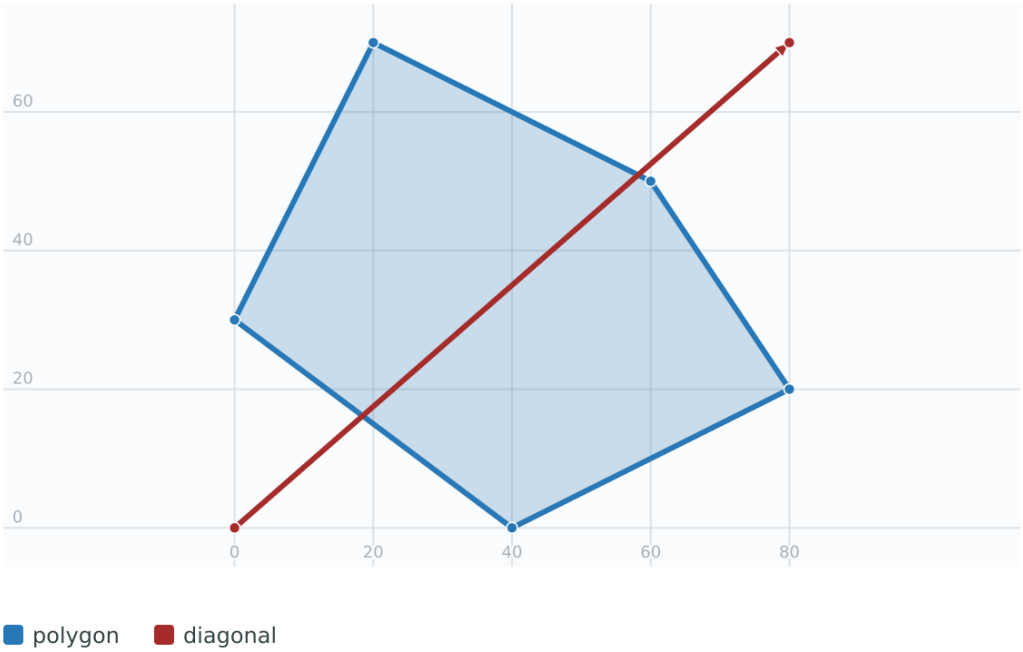
Code

```
WITH input AS (
  SELECT 'POLYGON((10 20,30 60,70 40,90 10,50 -10,10 20))'::geometry AS geom
)
SELECT geom AS input_polygon,
       ST_BoundingDiagonal(geom) AS diagonal
FROM input;
```

Output

```
POLYGON((10 20,30 60,70 40,90 10,50 -10,10 20)) | LINESTRING(10 -10,90 60)
```

Figure



This example returns the minimum X coordinate of a buffer around a point.

Code

```
SELECT ST_X(ST_StartPoint(ST_BoundingDiagonal(
  ST_Buffer(ST_Point(0, 0), 10)
)));
```

Output

```
-10
```

Siehe auch

ST_StartPoint, ST_EndPoint, ST_X, ST_Y, ST_Z, ST_M, &&&

7.4.4 ST_CoordDim

ST_CoordDim — Gibt die Dimension der Koordinaten für den Wert von ST_Geometry zurück.

Synopsis

integer **ST_CoordDim**(geometry geomA);

Beschreibung

Gibt die Dimension der Koordinaten für den Wert von ST_Geometry zurück.

Dies ist der MM konforme Alias für ST_NDims

Diese Methode implementiert die OGC Simple Features Implementation Specification for SQL 1.1.

Diese Methode setzt die SQL/MM-Spezifikation um. SQL-MM 3: 5.1.3

Diese Methode unterstützt kreisförmige Strings und Kurven.

Diese Funktion unterstützt 3d und lässt den Z-Index nicht fallen.

Diese Funktion unterstützt polyedrische Flächen.

Diese Funktion unterstützt Dreiecke und dreieckige unregelmäßige Netzoberflächen (TIN).

Beispiele

Code

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

Output

```
3
```

Code

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

Output

```
2
```

Siehe auch

ST_NDims

7.4.5 ST_Dimension

ST_Dimension — Gibt die Dimension der Koordinaten für den Wert von ST_Geometry zurück.

Synopsis

integer **ST_Dimension**(geometry g);

Beschreibung

Die inhärente Dimension eines geometrischen Objektes, welche kleiner oder gleich der Dimension der Koordinaten sein muss. Nach OGC SPEC s2.1.1.1 wird 0 für `POINT`, 1 für `LINESTRING`, 2 for `POLYGON`, und die größte Dimension der Teile einer `GEOMETRYCOLLECTION` zurückgegeben. Wenn die Dimension nicht bekannt ist (leere`GEOMETRYCOLLECTION`) wird 0 zurückgegeben.

Diese Methode setzt die SQL/MM-Spezifikation um. SQL-MM 3: 5.1.2



Anmerkung
A `TIN`, including a closed TIN surface, is a surface geometry and has dimension 2. Use CG_MakeSolid to make the solid representation explicit; the resulting 3D solid has dimension 3. Solid geometry semantics, such as volume, are handled separately by the SQL/MM 3D solid functions (SQL-MM IEC 13249-3: 9.1).

Erweiterung: 2.0.0 - Unterstützung für polyedrische Oberflächen und TIN eingeführt.



Anmerkung
Vor 2.0.0 meldete diese Funktion einen Fehler, falls sie auf eine leere Geometrie angewandt wurde.

Diese Funktion unterstützt polyedrische Flächen.

Diese Funktion unterstützt Dreiecke und dreieckige unregelmäßige Netzoberflächen (TIN).

Beispiele

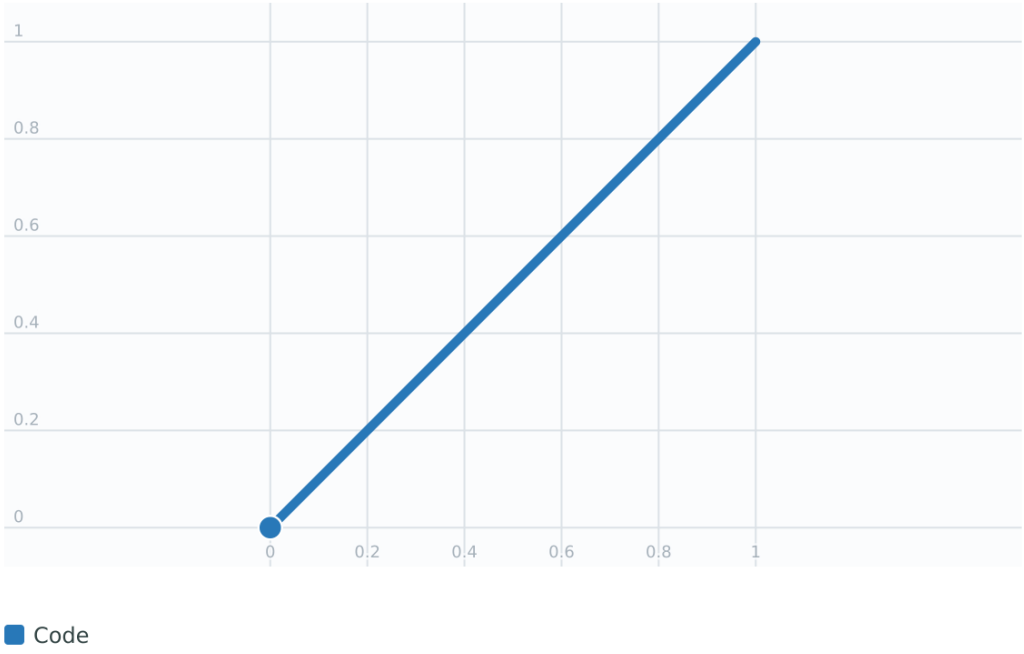
Code

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

Output

```
1
```

Figure



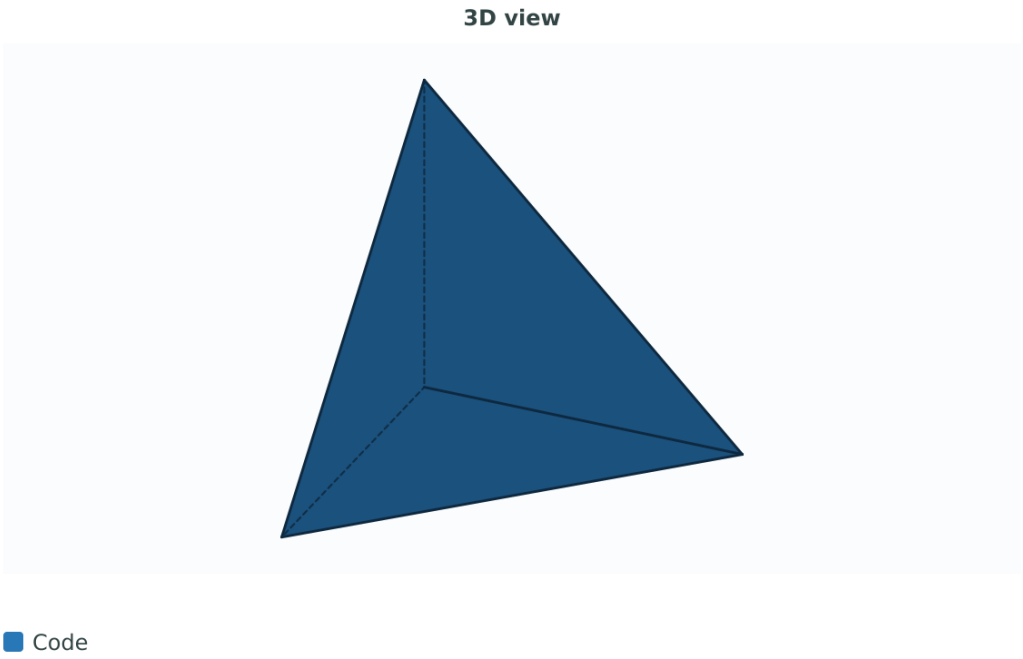
Code

```
SELECT ST_Dimension('TIN Z (((0 0 0,0 0 1,0 1 0,0 0 0)),((0 0 0,0 1 0,1 0 0,0 0 0)),((0 0 ↩
    0,1 0 0,0 0 1,0 0 0)),((1 0 0,0 1 0,0 0 1,1 0 0)))');
```

Output

```
2
```

Figure



Siehe auch

ST_NDims, CG_MakeSolid, CG_Volume

7.4.6 ST_Dump

ST_Dump — Gibt einen Satz von `geometry_dump` Zeilen für die Komponenten einer Geometrie zurück.

Synopsis

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

Beschreibung

Eine Funktion, die eine Menge zurückgibt (SRF), die die Komponenten einer Geometrie extrahiert. Sie gibt einen Satz von geometry_dump Zeilen zurück, die jeweils eine Geometrie (*`geom`* Feld) und eine Reihe von Ganzzahlen (*`path`* Feld) enthalten.

Für einen atomaren Geometrietyp (POINT, LINESTRING, POLYGON) wird ein einzelner Datensatz mit einem leeren Array *`path`* und die Eingabegeometrie als *`geom`* zurückgegeben. Bei einer Sammlung oder Multi-Geometrie wird ein Datensatz für jede der Komponenten der Sammlung zurückgegeben, und der *`path`* bezeichnet die Position der Komponente innerhalb der Sammlung.

ST_Dump ist nützlich für die Erweiterung von Geometrien. Es ist die Umkehrung einer ST_Collect / GROUP BY, da es neue Zeilen erstellt. Zum Beispiel kann es verwendet werden, um MULTIPOLYGONS in POLYGONS zu erweitern.

Erweiterung: Mit 2.0.0 wurde die Unterstützung für polyedrische Oberflächen, Dreiecke und TIN eingeführt.

Verfügbarkeit: PostGIS 1.0.0RC1. Benötigt PostgreSQL 7.3 oder höher.



Anmerkung
Vor 1.3.4 ist diese Funktion abgestürzt, wenn die Geometrien CURVES enthalten. Dies wurde mit 1.3.4+ behoben

Diese Methode unterstützt kreisförmige Strings und Kurven.

Diese Funktion unterstützt polyedrische Flächen.

Diese Funktion unterstützt Dreiecke und dreieckige unregelmäßige Netzoberflächen (TIN).

Diese Funktion unterstützt 3d und lässt den Z-Index nicht fallen.

Beispiele

Dump the components of geometries stored in a table.

Code

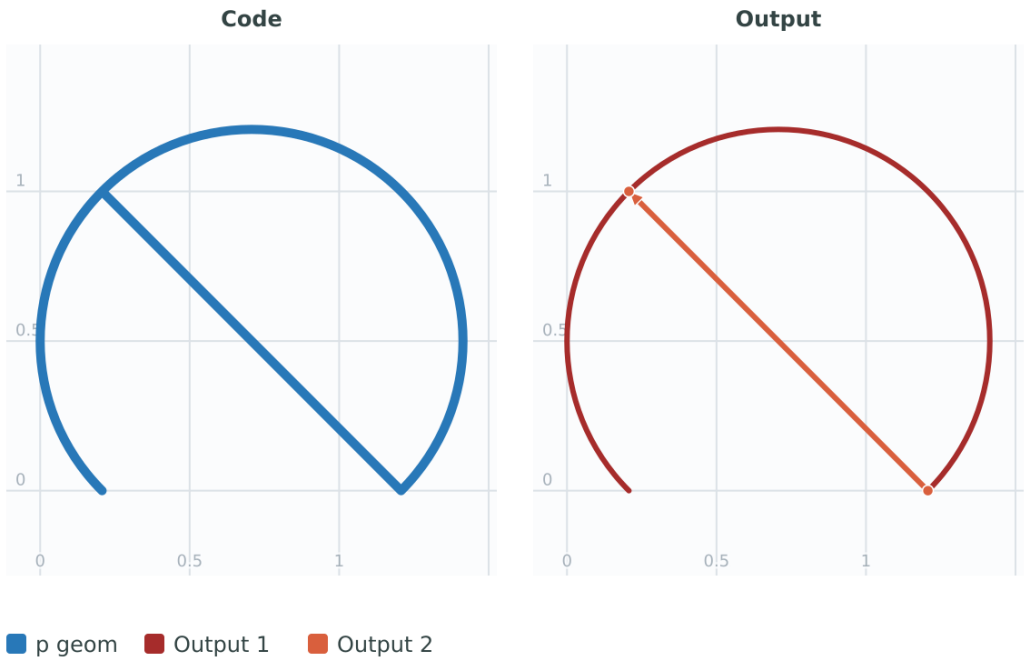
```
SELECT sometable.field1,
       (ST_Dump(sometable.geom)).geom AS geom
FROM sometable;
```

Break a CompoundCurve into its constituent LineStrings and CircularStrings.

Code

```
SELECT a.geom, ST_HasArc(a.geom)
  FROM ( SELECT (ST_Dump(p_geom)).geom AS geom
         FROM (SELECT 'COMPOUNDCURVE(CIRCULARSTRING(0 0,1 1,1 0),(1 0,0 1))'::geometry AS ↩
    p_geom) AS b
        ) AS a;
```

Figure

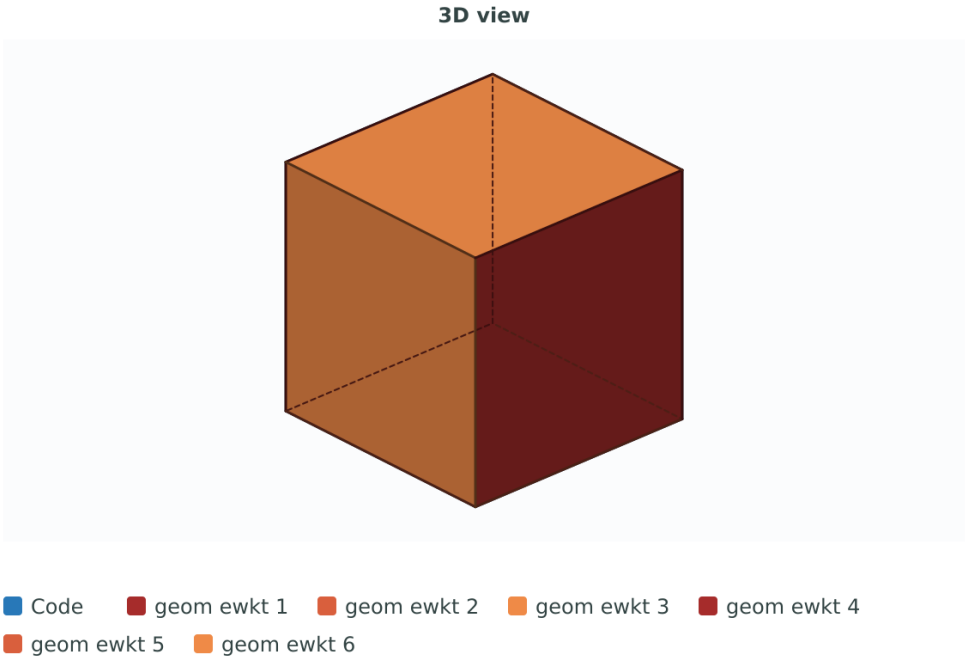


This example breaks a PolyhedralSurface into its faces.

Code

```
SELECT (a.p_geom).path[1] As path,(a.p_geom).geom As geom_ewkt
  FROM (SELECT ST_Dump('POLYHEDRALSURFACE(
((0 0 0,0 0 1,0 1 1,0 1 0,0 0 0)),
((0 0 0,0 1 0,1 1 0,1 0 0,0 0 0)),((0 0 0,1 0 0,1 0 1,0 0 1,0 0 0)),((1 1 0,1 1 1,1 0 1,1 0 0,1 1 0)),
((0 1 0,0 1 1,1 1 1,1 1 0,0 1 0)),((0 0 1,1 0 1,1 1 1,0 1 1,0 0 1))
)' ) AS p_geom )  AS a;
```

Figure

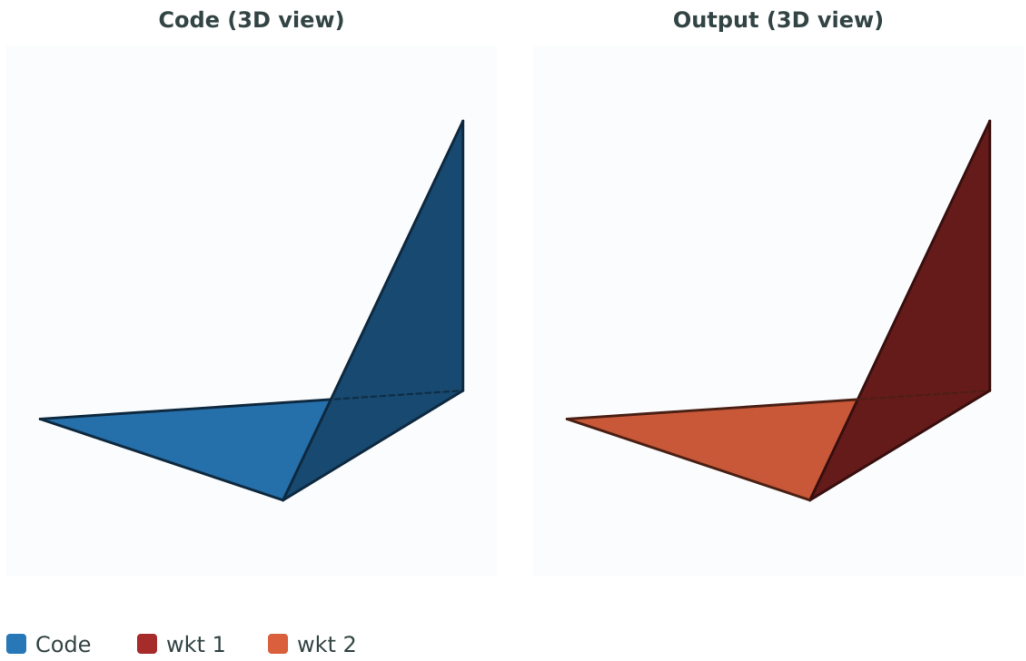


This example uses a TIN geometry.

Code

```
SELECT (g.gdump).path,(g.gdump).geom as wkt
  FROM
    (SELECT
       ST_Dump('TIN (((
                  0 0 0,
                  0 0 1,
                  0 1 0,
                  0 0 0
            )),((
                  0 0 0,
                  0 1 0,
                  1 1 0,
                  0 0 0
            ))
            )' ) AS gdump
    ) AS g;
```

Figure



Siehe auch

geometry_dump, ST_GeomFromEWKT, ST_Dump, ST_GeometryN, ST_NumGeometries

7.4.7 ST_DumpPoints

ST_DumpPoints — Gibt eine Zusammenfassung des Inhalts einer Geometrie wieder.

Synopsis

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

Beschreibung

Eine Funktion, die eine Menge zurückliefert (SRF), die die Koordinaten (Eckpunkte) einer Geometrie extrahiert. Sie gibt einen Satz von geometry_dump Zeilen zurück, die jeweils eine Geometrie (*geom* Feld) und eine Reihe von Ganzzahlen (*path* Feld) enthalten.

- das Feld *geom* `POINT`s stellt die Koordinaten der gelieferten Geometrie dar.
- das Feld *path* (ein `integer[]`) ist ein Index, der die Koordinatenpositionen in den Elementen der gelieferten Geometrie aufzählt. Die Indizes sind 1-basiert. Für einen `LINESTRING` sind die Pfade beispielsweise `{i}` wobei `i` die `n-te` Koordinate im `LINESTRING` ist. Für ein `POLYGON` lauten die Pfade `{i,j}`, wobei `i` die Ringnummer ist (1 ist der äußere, die inneren Ringe folgen) und `j` die Koordinatenposition im Ring.

Um eine einzelne Geometrie zu erhalten, die die Koordinaten enthält, verwenden Sie ST_Points.

Verbessert: 2.1.0 Höhere Geschwindigkeit. Reimplementiert als natives C.

Erweiterung: Mit 2.0.0 wurde die Unterstützung für polyedrische Oberflächen, Dreiecke und TIN eingeführt.

Verfügbarkeit: 1.5.0

Diese Methode unterstützt kreisförmige Strings und Kurven.

Diese Funktion unterstützt polyedrische Flächen.

Diese Funktion unterstützt Dreiecke und dreieckige unregelmäßige Netzoberflächen (TIN).

Diese Funktion unterstützt 3d und lässt den Z-Index nicht fallen.

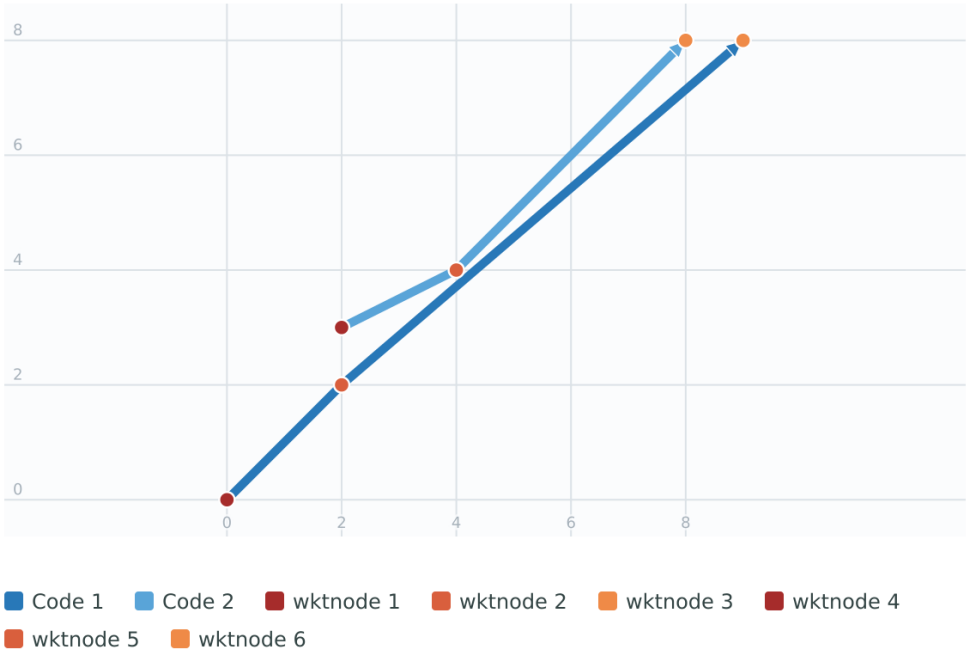
Beispiele

Explode a table of LineStrings into their vertices.

Code

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

Figure



Dump every vertex from a mixed GeometryCollection.

Code

```
SELECT path,geom
FROM (
  SELECT (ST_DumpPoints(g.geom)).*
  FROM
    (SELECT
       'GEOMETRYCOLLECTION(
```

```
          POINT ( 0 1 ),
          LINESTRING ( 0 3,3 4 ),
          POLYGON (( 2 0,2 3,0 2,2 0 )),
          POLYGON (( 3 0,3 3,6 3,6 0,3 0 ),
                   ( 5 1,4 2,5 2,5 1 )),
          MULTIPOLYGON (
                  (( 0 5,0 8,4 8,4 5,0 5 ),
                   ( 1 6,3 6,2 7,1 6 )),
                  (( 5 4,5 8,6 7,5 4 ))
          )
        )'::geometry AS geom
    ) AS g
  ) j;
```

Output

```
path    | st_astext
--------+------------
 {1,1}     | POINT(0 1)
 {2,1}     | POINT(0 3)
 {2,2}     | POINT(3 4)
 {3,1,1}   | POINT(2 0)
 {3,1,2}   | POINT(2 3)
 {3,1,3}   | POINT(0 2)
 {3,1,4}   | POINT(2 0)
 {4,1,1}   | POINT(3 0)
 {4,1,2}   | POINT(3 3)
 {4,1,3}   | POINT(6 3)
 {4,1,4}   | POINT(6 0)
 {4,1,5}   | POINT(3 0)
 {4,2,1}   | POINT(5 1)
 {4,2,2}   | POINT(4 2)
 {4,2,3}   | POINT(5 2)
 {4,2,4}   | POINT(5 1)
 {5,1,1,1} | POINT(0 5)
 {5,1,1,2} | POINT(0 8)
 {5,1,1,3} | POINT(4 8)
 {5,1,1,4} | POINT(4 5)
 {5,1,1,5} | POINT(0 5)
 {5,1,2,1} | POINT(1 6)
 {5,1,2,2} | POINT(3 6)
 {5,1,2,3} | POINT(2 7)
 {5,1,2,4} | POINT(1 6)
 {5,2,1,1} | POINT(5 4)
 {5,2,1,2} | POINT(5 8)
 {5,2,1,3} | POINT(6 7)
 {5,2,1,4} | POINT(5 4)
(29 rows)
```

Collect the dumped vertices to visualize them over the source geometry.

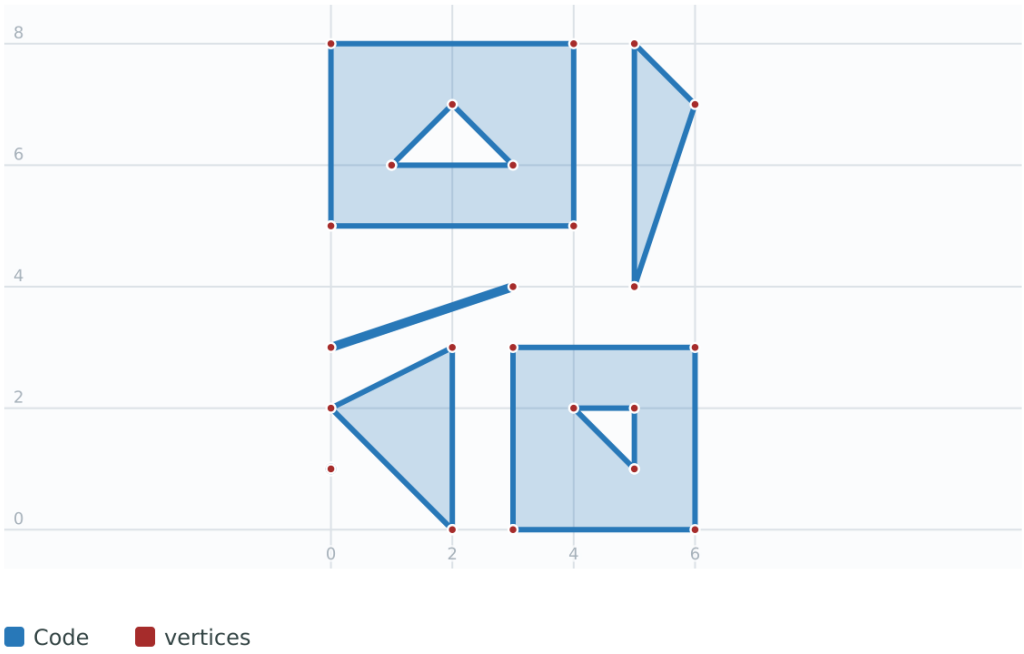
Code

```
WITH source(geom) AS (VALUES (
  'GEOMETRYCOLLECTION(
     POINT (0 1),
     LINESTRING (0 3,3 4),
     POLYGON ((2 0,2 3,0 2,2 0)),
     POLYGON ((3 0,3 3,6 3,6 0,3 0),(5 1,4 2,5 2,5 1)),
     MULTIPOLYGON (((0 5,0 8,4 8,4 5,0 5),(1 6,3 6,2 7,1 6)),
                   ((5 4,5 8,6 7,5 4))))'::geometry
))
SELECT ST_Collect((dp).geom ORDER BY (dp).path) AS vertices
FROM source
CROSS JOIN LATERAL ST_DumpPoints(source.geom) AS dp;
```

Output

```
MULTIPOINT((0 1),(0 3),(3 4),(2 0),(2 3),(0 2),(2 0),(3 0),(3 3),(6 3),(6 0),(3 0),(5 1),(4 2),(5 2),(5 1),(0 5),(0 8) ↩
    ,(4 8),(4 5),(0 5),(1 6),(3 6),(2 7),(1 6),(5 4),(5 8),(6 7),(5 4))
```

Figure



This example uses a cube represented as a PolyhedralSurface.

Code

```
SELECT (g.gdump).path,(g.gdump).geom as wkt
  FROM
    (SELECT
       ST_DumpPoints('POLYHEDRALSURFACE( ((0 0 0,0 0 1,0 1 1,0 1 0,0 0 0)),
((0 0 0,0 1 0,1 1 0,1 0 0,0 0 0)),((0 0 0,1 0 0,1 0 1,0 0 1,0 0 0)),
((1 1 0,1 1 1,1 0 1,1 0 0,1 1 0)),
((0 1 0,0 1 1,1 1 1,1 1 0,0 1 0)),((0 0 1,1 0 1,1 1 1,0 1 1,0 0 1)) )' ) AS gdump
    ) AS g;
```

Output

```
path   |     wkt
--------+-------------
 {1,1,1} | POINT(0 0 0)
 {1,1,2} | POINT(0 0 1)
 {1,1,3} | POINT(0 1 1)
 {1,1,4} | POINT(0 1 0)
 {1,1,5} | POINT(0 0 0)
 {2,1,1} | POINT(0 0 0)
 {2,1,2} | POINT(0 1 0)
 {2,1,3} | POINT(1 1 0)
 {2,1,4} | POINT(1 0 0)
 {2,1,5} | POINT(0 0 0)
 {3,1,1} | POINT(0 0 0)
 {3,1,2} | POINT(1 0 0)
 {3,1,3} | POINT(1 0 1)
 {3,1,4} | POINT(0 0 1)
 {3,1,5} | POINT(0 0 0)
 {4,1,1} | POINT(1 1 0)
 {4,1,2} | POINT(1 1 1)
 {4,1,3} | POINT(1 0 1)
 {4,1,4} | POINT(1 0 0)
 {4,1,5} | POINT(1 1 0)
 {5,1,1} | POINT(0 1 0)
```

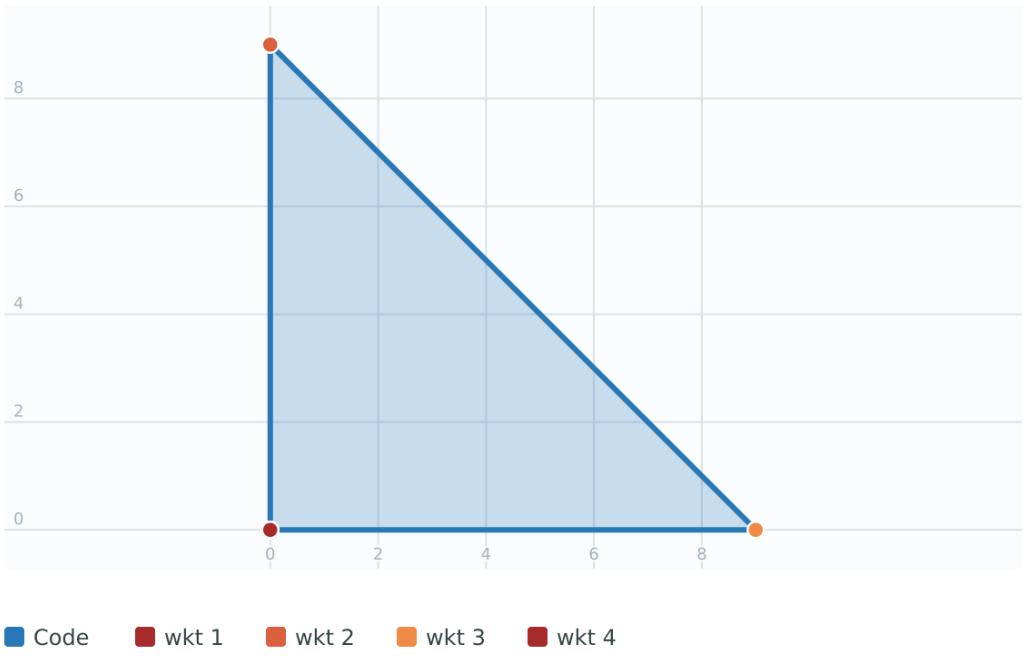
```
 {5,1,2} | POINT(0 1 1)
 {5,1,3} | POINT(1 1 1)
 {5,1,4} | POINT(1 1 0)
 {5,1,5} | POINT(0 1 0)
 {6,1,1} | POINT(0 0 1)
 {6,1,2} | POINT(1 0 1)
 {6,1,3} | POINT(1 1 1)
 {6,1,4} | POINT(0 1 1)
 {6,1,5} | POINT(0 0 1)
(30 rows)
```

This example uses a Triangle geometry.

Code

```
SELECT (g.gdump).path,(g.gdump).geom as wkt
  FROM
    (SELECT
       ST_DumpPoints('TRIANGLE ((
                  0 0,
                  0 9,
                  9 0,
                  0 0
              ))' ) AS gdump
    ) AS g;
```

Figure



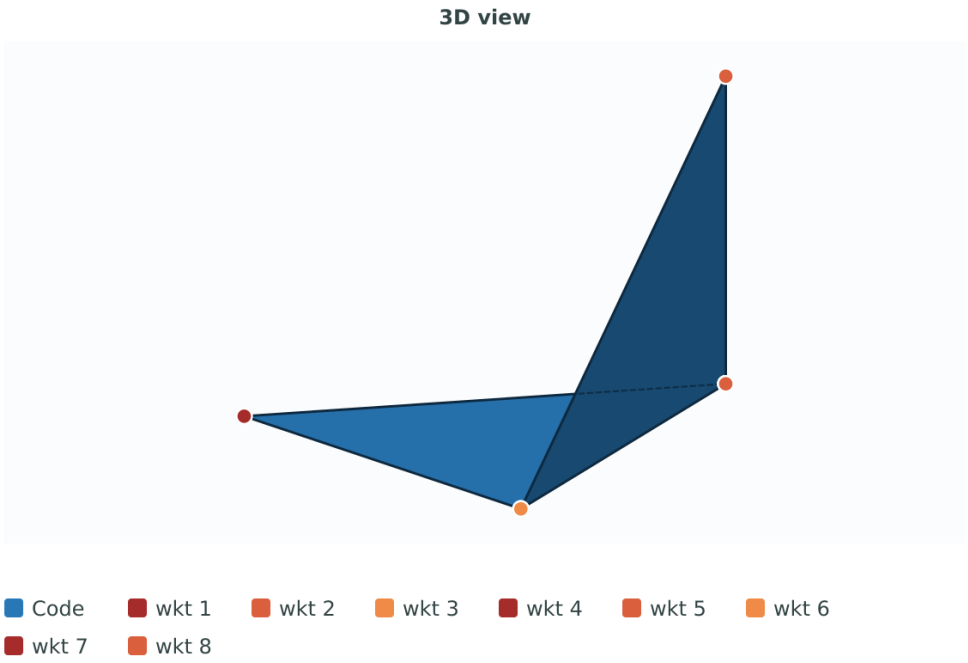
This example uses a TIN geometry.

Code

```
SELECT (g.gdump).path,(g.gdump).geom as wkt
  FROM
    (SELECT
       ST_DumpPoints('TIN (((
                  0 0 0,
                  0 0 1,
                  0 1 0,
                  0 0 0
              )),((
```

```
                0 0 0,
                0 1 0,
                1 1 0,
                0 0 0
            ))
            )' ) AS gdump
    ) AS g;
```

Figure



Siehe auch

geometry_dump, ST_GeomFromEWKT, ST_Dump, ST_GeometryN, ST_NumGeometries

7.4.8 ST_DumpSegments

ST_DumpSegments — Gibt eine Zusammenfassung des Inhalts einer Geometrie wieder.

Synopsis

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

Beschreibung

Eine Funktion, die eine Menge zurückgibt (SRF), die die Segmente einer Geometrie extrahiert. Sie gibt einen Satz von geometry_dump Zeilen zurück, die jeweils eine Geometrie (Feld*geom*) und eine Reihe von Ganzzahlen (Feld*path*) enthalten.

- Die Felder *geom* LINESTRINGs stellen die linearen Segmente der gelieferten Geometrie dar, während die Felder CIRCULARSTRIN die Bogensegmente darstellen.

- Das Feld *path* (ein integer[]) ist ein Index, der die Positionen der Segmentstartpunkte in den Elementen der gelieferten Geometrie auflistet. Die Indizes sind 1-basiert. Zum Beispiel sind für einen LINESTRING die Pfade {i} wobei i der n-te Segmentstartpunkt im LINESTRING ist. Für ein POLYGON lauten die Pfade {i, j}, wobei i die Ringnummer ist (1 ist der äußere, die inneren Ringe folgen) und j die Position des Segment-Startpunkts im Ring.

Verfügbarkeit: 3.2.0

Diese Funktion unterstützt Dreiecke und dreieckige unregelmäßige Netzoberflächen (TIN).

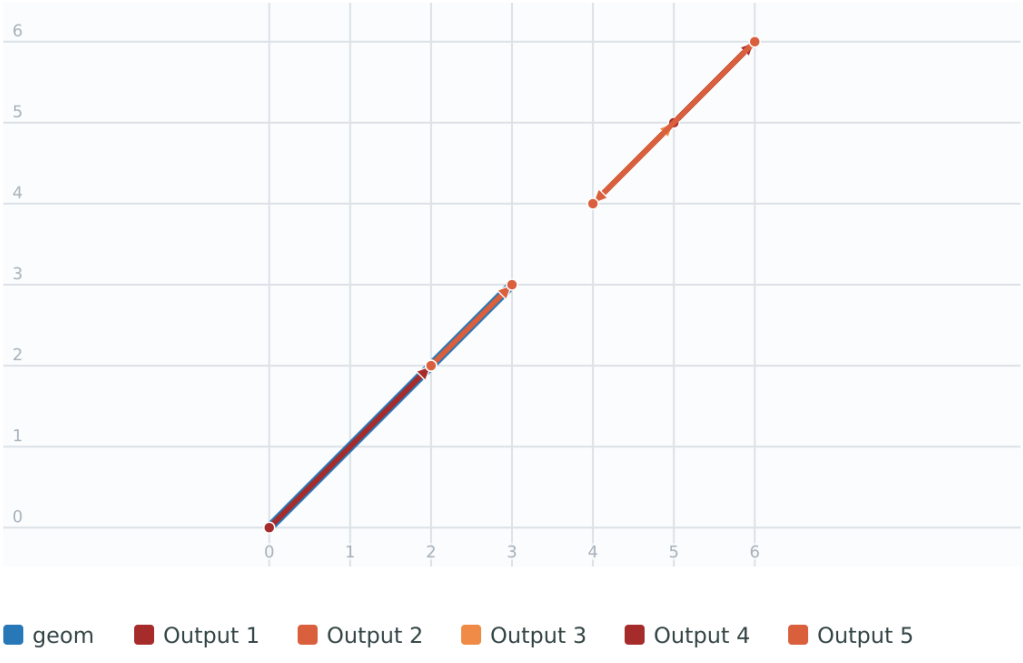
Diese Funktion unterstützt 3d und lässt den Z-Index nicht fallen.

Standard Beispiele

Code

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

Figure



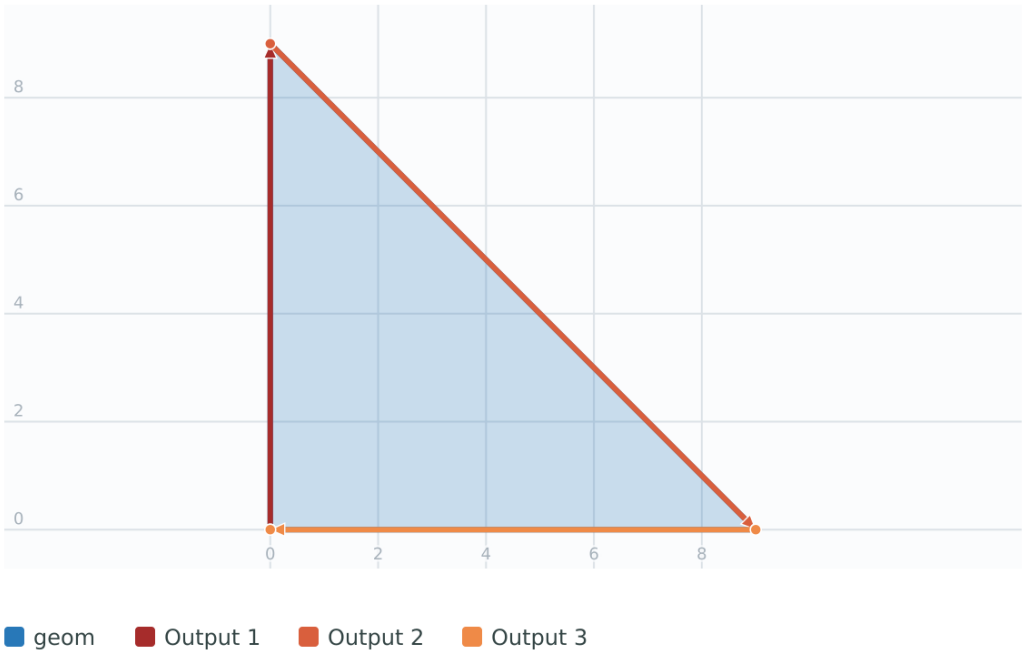
Beispiele für polyedrische Oberflächen, TIN und Dreieck

This example uses a Triangle geometry.

Code

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

Figure

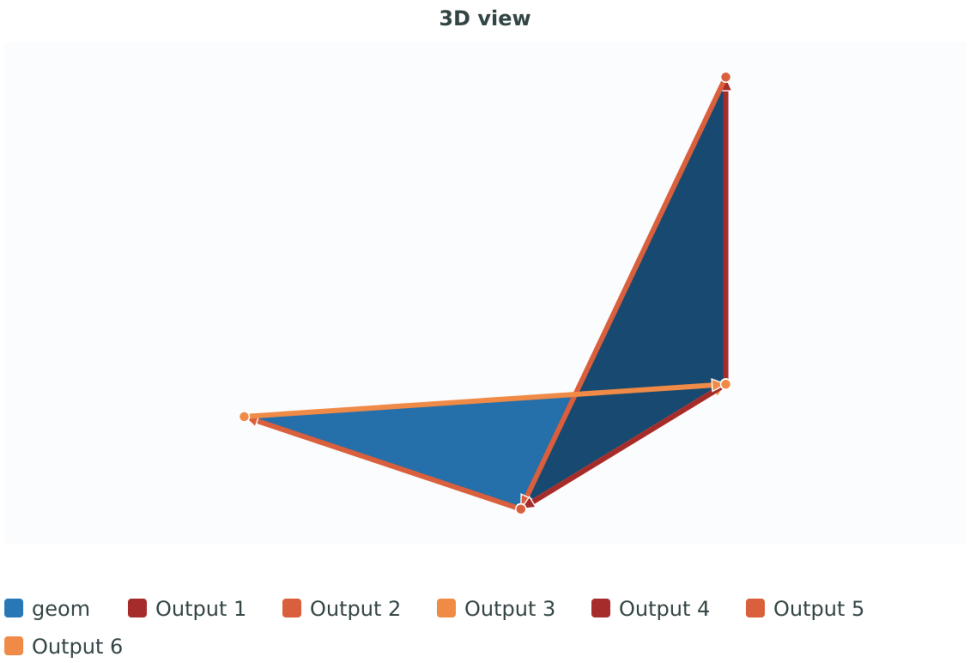


This example uses a TIN geometry.

Code

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

Figure



Siehe auch

geometry_dump, ST_Collect, ST_Dump, ST_NumInteriorRing,

7.4.9 ST_DumpRings

ST_DumpRings — Gibt einen Satz von `geometry_dump` Zeilen für die äußeren und inneren Ringe eines Polygons zurück.

Synopsis

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

Beschreibung

Eine Funktion, die eine Menge zurückgibt (SRF), die die Ringe eines Polygons extrahiert. Sie gibt einen Satz von geometry_dump Zeilen zurück, die jeweils eine Geometrie (*`geom`* Feld) und eine Reihe von Ganzzahlen (*`path`* Feld) enthalten.

Das Feld *`geom`* enthält jeden Ring als POLYGON. Das Feld *`path`* ist ein ganzzahliges Feld der Länge 1, das den Polygonringindex enthält. Der äußere Ring (Schale) hat den Index 0. Die inneren Ringe (Löcher) haben Indizes von 1 und höher.



Anmerkung
This only works for POLYGON geometries, not MULTIPOLYGONs. Use ST_Dump to extract polygon elements from polygonal geometries: `ST_DumpRings((ST_Dump(geom)).geom )`

Verfügbarkeit: PostGIS 1.1.3. Benötigt PostgreSQL 7.3 oder höher.


Diese Funktion unterstützt 3d und lässt den Z-Index nicht fallen.

Beispiele

Extracting all rings as polygons.

Code

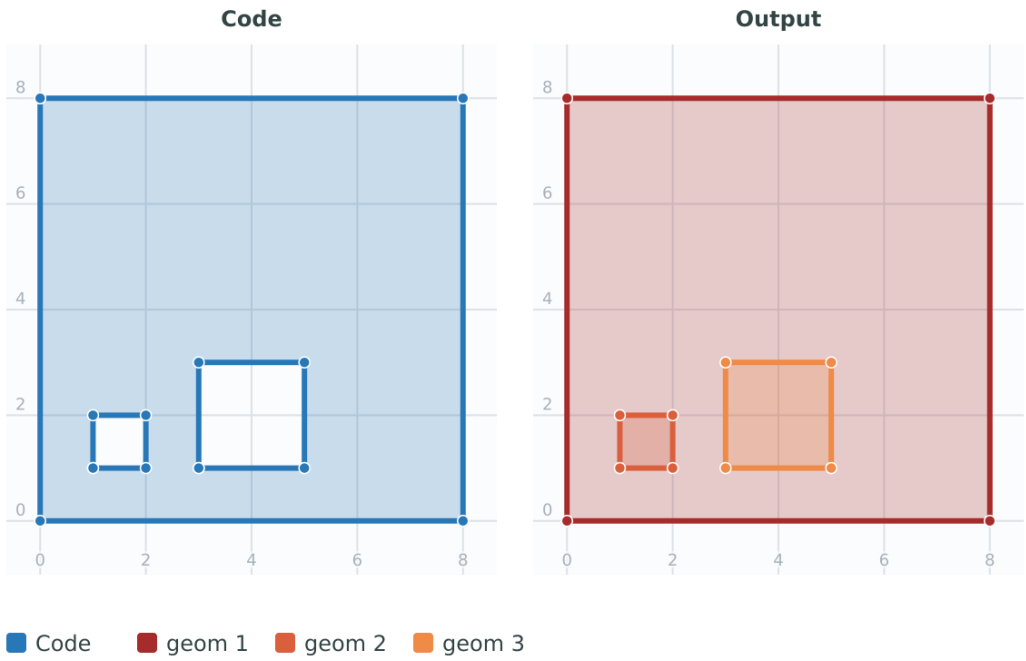
```
SELECT polyTable.field1, polyTable.field1,
      (ST_DumpRings(polyTable.geom)).geom As geom
FROM polyTable
```

Extracting shell and holes from a polygon.

Code

```
SELECT path,geom As geom
FROM ST_DumpRings('POLYGON ((1 9,9 9,9 1,1 1,1 9),(2 2,2 3,3 3,3 2,2 2),(4 2,4 4,6 4,6 2,4 ↩
    2))');
```

Figure



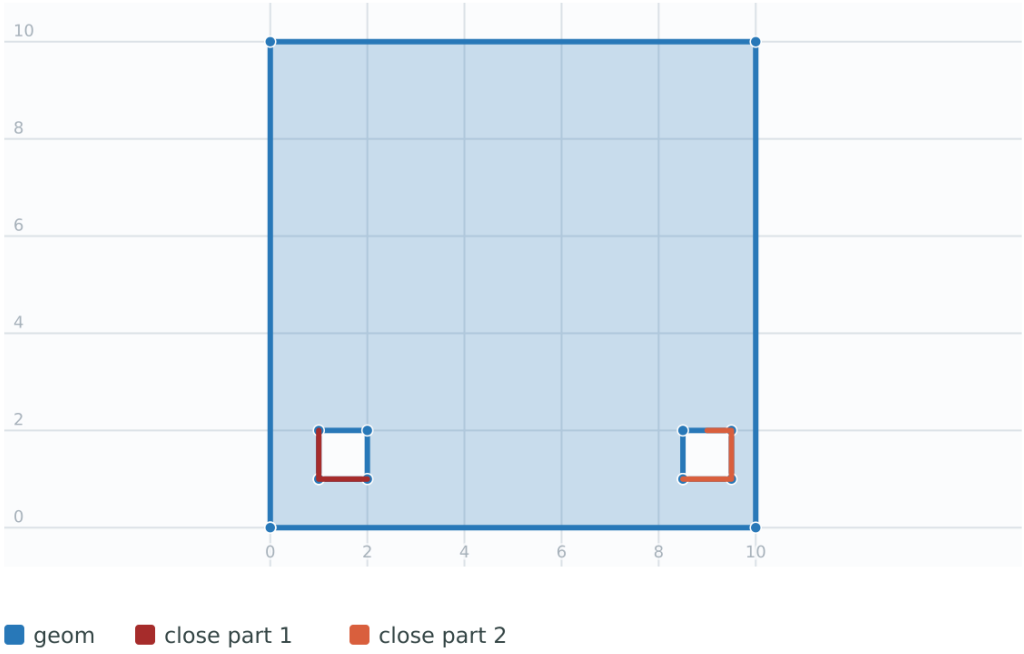
Finding interior rings which lie close to the exterior ring.

Code

```
WITH poly AS (
  SELECT 'POLYGON ((
      0 0, 10 0, 10 10, 0 10, 0 0
    ), (
      1 1, 1 2, 2 2, 2 1, 1 1
    ), (
      8.5 1, 8.5 2, 9.5 2, 9.5 1, 8.5 1
    ))'::geometry AS geom
),
rings AS (
  SELECT d.path[1] AS ring_no,
         ST_Boundary(d.geom) AS ring_geom
  FROM poly
  CROSS JOIN LATERAL ST_DumpRings(poly.geom) AS d
```

```
),
shell AS (
  SELECT ring_geom
  FROM rings
  WHERE ring_no = 0
),
holes AS (
  SELECT ring_no, ring_geom
  FROM rings
  WHERE ring_no > 0
)
SELECT h.ring_no,
       ST_Normalize(ST_Multi(ST_LineMerge(
         ST_Intersection(ST_Buffer(s.ring_geom, 1.0), h.ring_geom)
       ))) AS close_part
FROM shell AS s
JOIN holes AS h
  ON ST_DWithin(s.ring_geom, h.ring_geom, 1.0);
```

Figure



Siehe auch

geometry_dump, Abschnitt 13.6, ST_Boundary, ST_Buffer, ST_Dump, ST_DWithin, ST_ExteriorRing, ST_InteriorRingN, ST_Intersec

7.4.10 ST_EndPoint

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

Synopsis

geometry **ST_EndPoint**(geometry g);

Beschreibung

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

Diese Methode setzt die SQL/MM-Spezifikation um. SQL-MM 3: 7.1.4

Diese Funktion unterstützt 3d und lässt den Z-Index nicht fallen.

Diese Methode unterstützt kreisförmige Strings und Kurven.

Änderung: 2.0.0 unterstützt die Verarbeitung von MultiLinestring's die nur aus einer einzelnen Geometrie bestehen, nicht mehr. In früheren Versionen von PostGIS gab die Funktion bei einem aus einer einzelnen Linie bestehender MultiLinestring den Anfangspunkt zurück. Ab 2.0.0 gibt sie nur NULL zurück, so wie bei jedem anderen MultiLinestring. Die alte Verhaltensweise war undokumentiert, aber Anwender, die annahmen, dass Sie Ihre Daten als LINESTRING vorliegen haben, könnten in 2.0 dieses zurückgegebene NULL bemerken.

Beispiele

Einhüllende von Punkt und Linienzug.

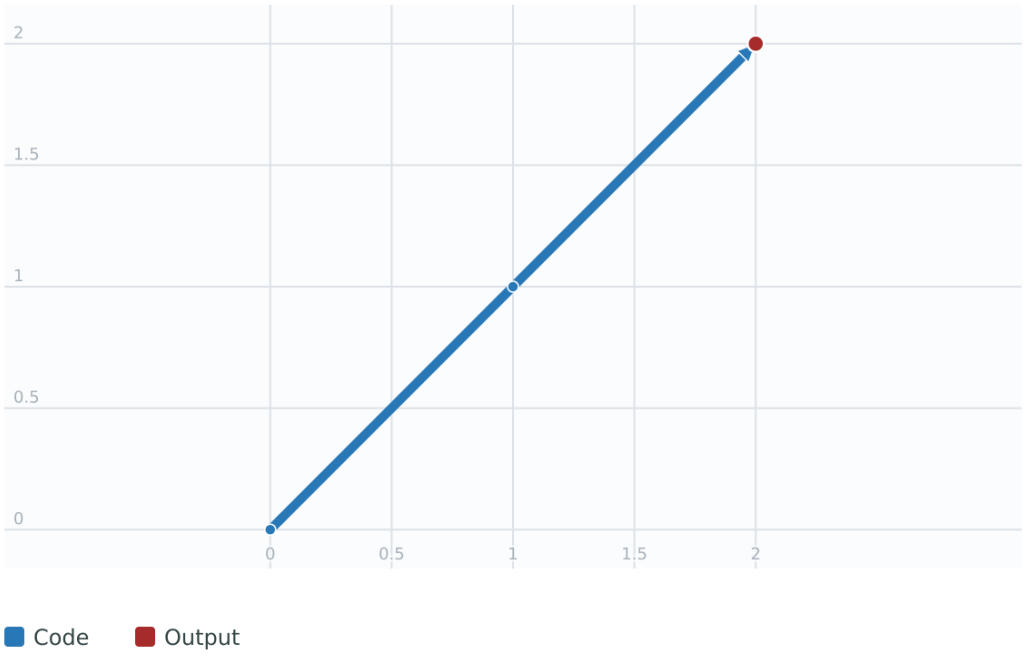
Code

```
SELECT ST_EndPoint('LINESTRING(1 1,2 2,3 3)'::geometry);
```

Output

```
POINT(3 3)
```

Figure



Endpunkt eines Nicht-LineString ist NULL

Code

```
SELECT ST_EndPoint('POINT(1 1)'::geometry) IS NULL AS is_null;
```

Output

```
t
```

Einhüllende von Punkt und Linienzug.

3D endpoint.

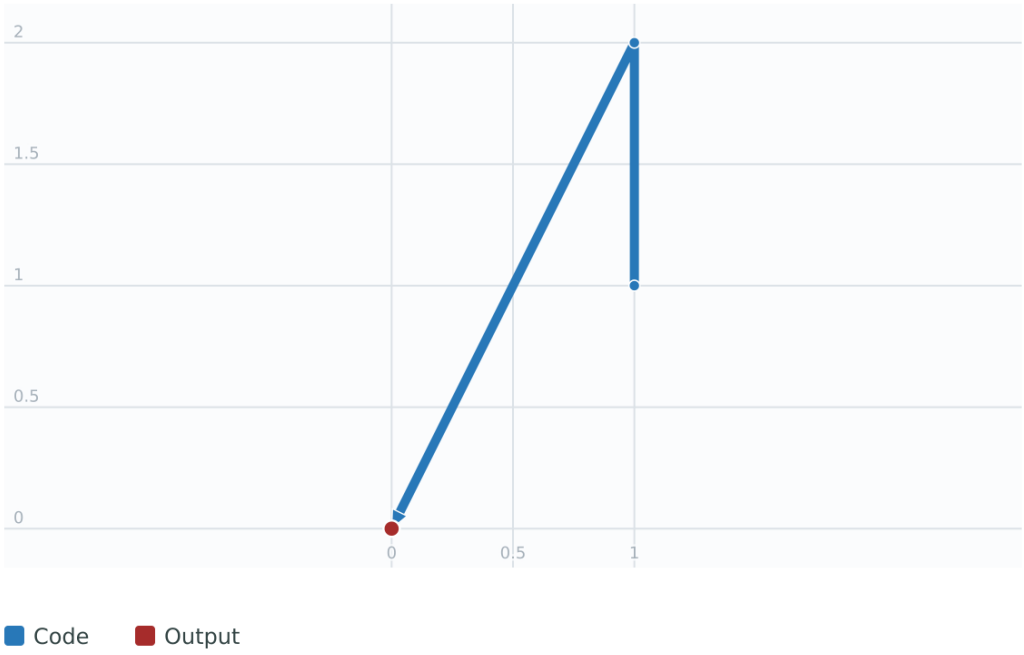
Code

```
SELECT ST_EndPoint('LINESTRING(1 1 2,1 2 3,0 0 5)');
```

Output

```
POINT(0 0 5)
```

Figure



Gibt die Anzahl der Stützpunkte eines ST_LineString oder eines ST_CircularString zurück.

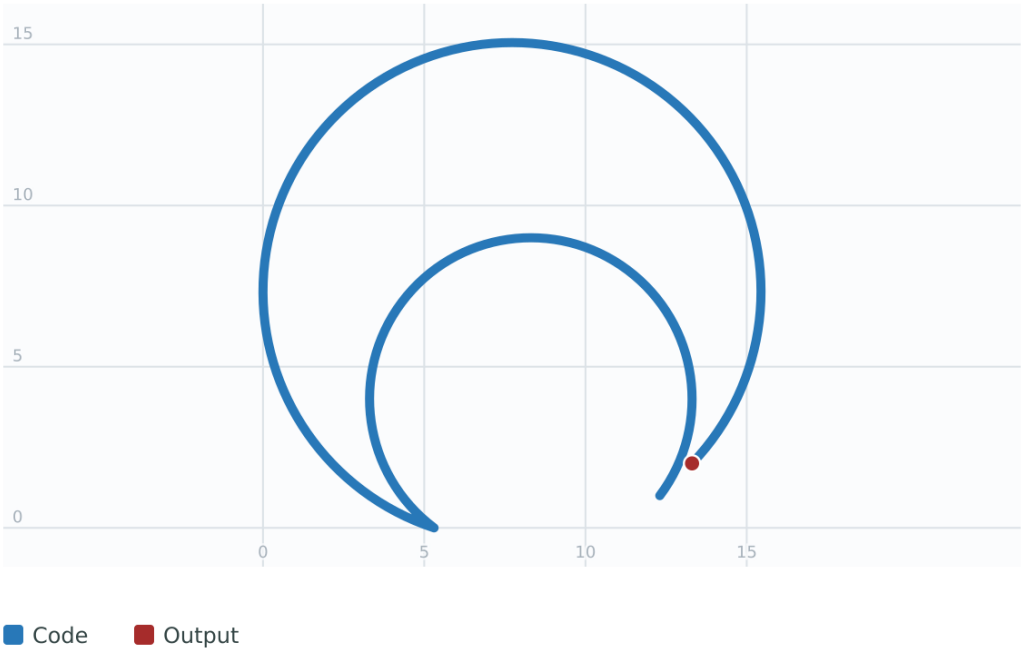
Code

```
SELECT ST_EndPoint('CIRCULARSTRING(5 2,-3 2,-2 1,-4 2,6 3)'::geometry);
```

Output

```
POINT(6 3)
```

Figure



End point of a NURBSCurve

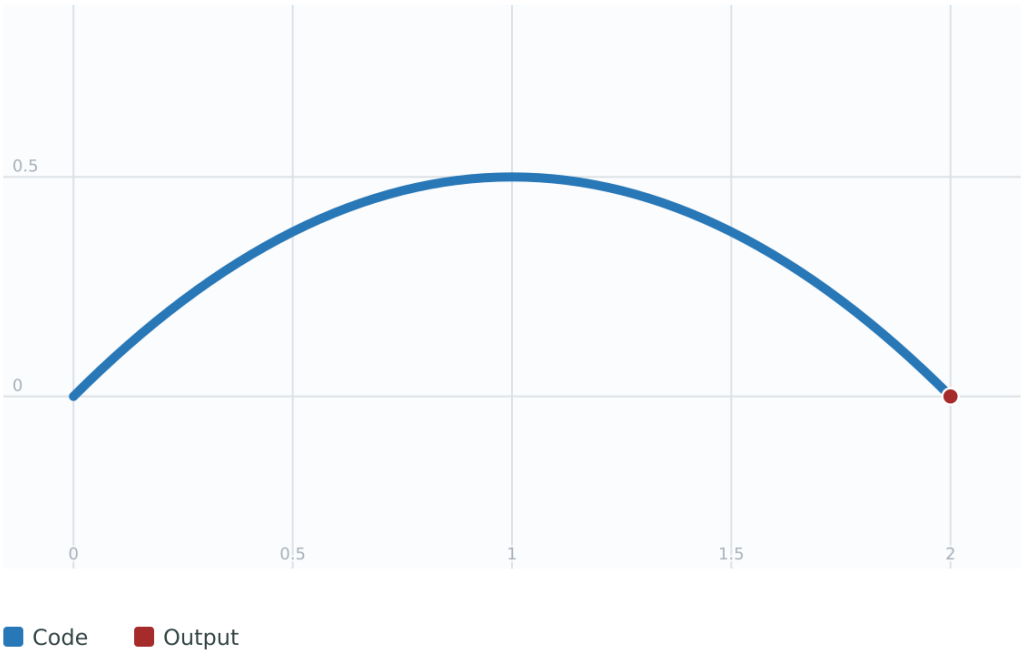
Code

```
SELECT ST_EndPoint('NURBSCURVE(2, (0 0, 1 1, 2 0))'::geometry);
```

Output

```
POINT(2 0)
```

Figure



Siehe auch

ST_PointN, ST_StartPoint

7.4.11 ST_Envelope

ST_Envelope — Gibt eine Geometrie in doppelter Genauigkeit (float8) zurück, welche das Umgebungsrechteck der beigestellten Geometrie darstellt.

Synopsis

geometry **ST_Envelope**(geometry g1);

Beschreibung

Gibt das kleinstmögliche Umgebungsrechteck der bereitgestellten Geometrie als Geometrie im Float8-Format zurück. Das Polygon wird durch die Eckpunkte des Umgebungsrechteckes beschrieben ((`MINX`, `MINY`), (`MINX`, `MAXY`), (`MAXX`, `MAXY`), (`MAXX`, `MINY`), (`MINX`, `MINY`)). (PostGIS fügt auch die `ZMIN`/`ZMAX` Koordinaten hinzu).

Spezialfälle (vertikale Linien, Punkte) geben eine Geometrie geringerer Dimension zurück als `POLYGON`, insbesondere `POINT` oder `LINESTRING`.

Verfügbarkeit: 1.5.0 Änderung der Verhaltensweise insofern, das die Ausgabe in Double Precision anstelle von Float4 erfolgt

Diese Methode implementiert die OGC Simple Features Implementation Specification for SQL 1.1. s2.1.1.1

Diese Methode setzt die SQL/MM-Spezifikation um. SQL-MM 3: 5.1.19

Beispiele

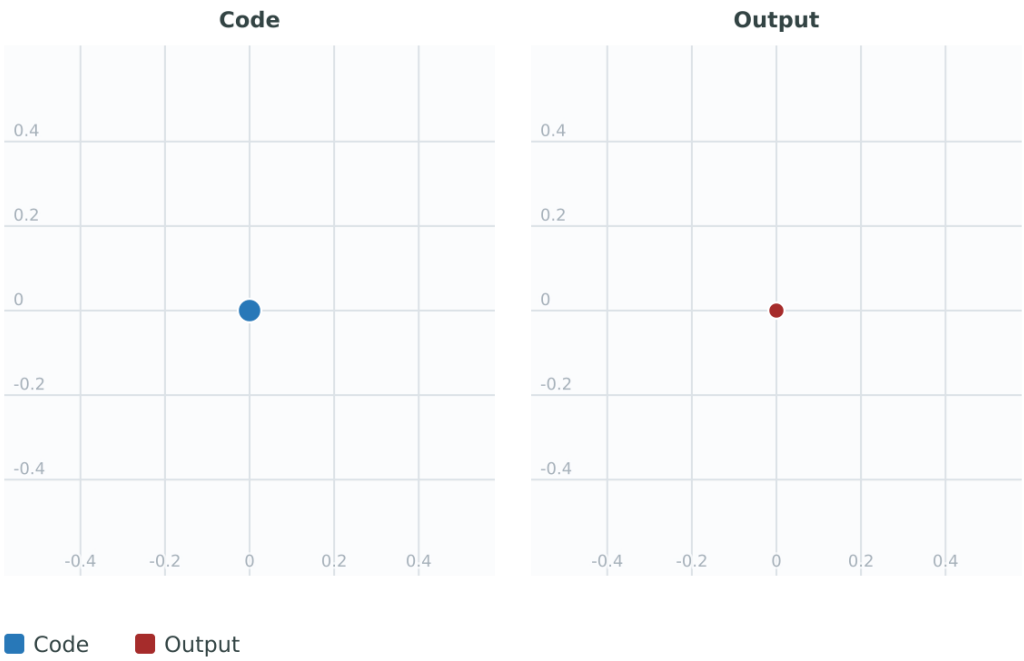
Code

```
SELECT ST_Envelope('POINT(1 3)'::geometry);
```

Output

```
POINT(1 3)
```

Figure



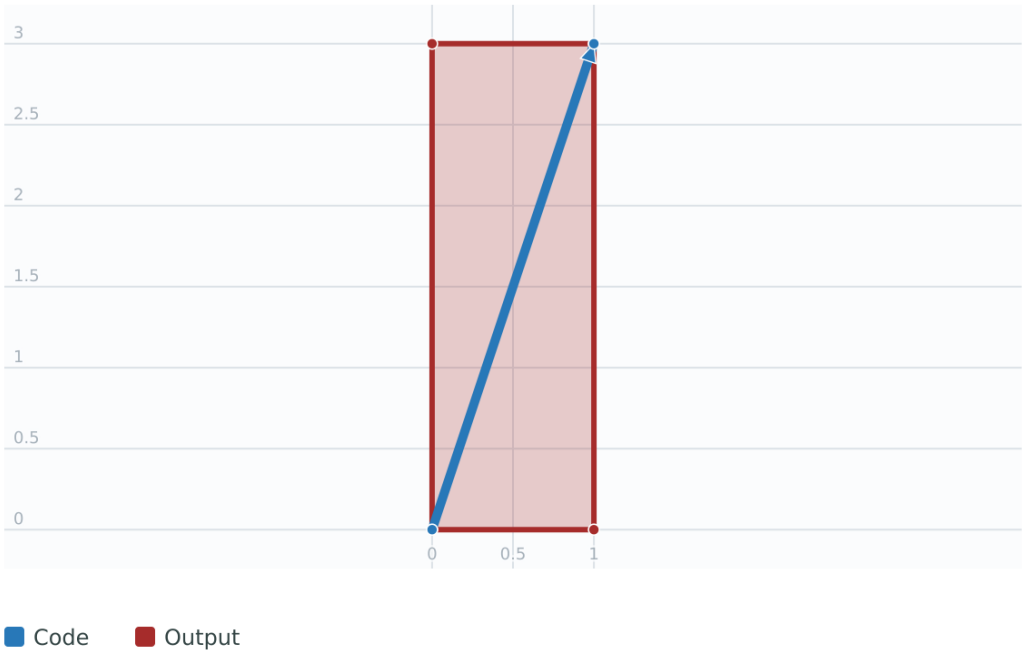
Code

```
SELECT ST_Envelope('LINESTRING(0 0,1 3)'::geometry);
```

Output

```
POLYGON((0 0,0 3,1 3,1 0,0 0))
```

Figure



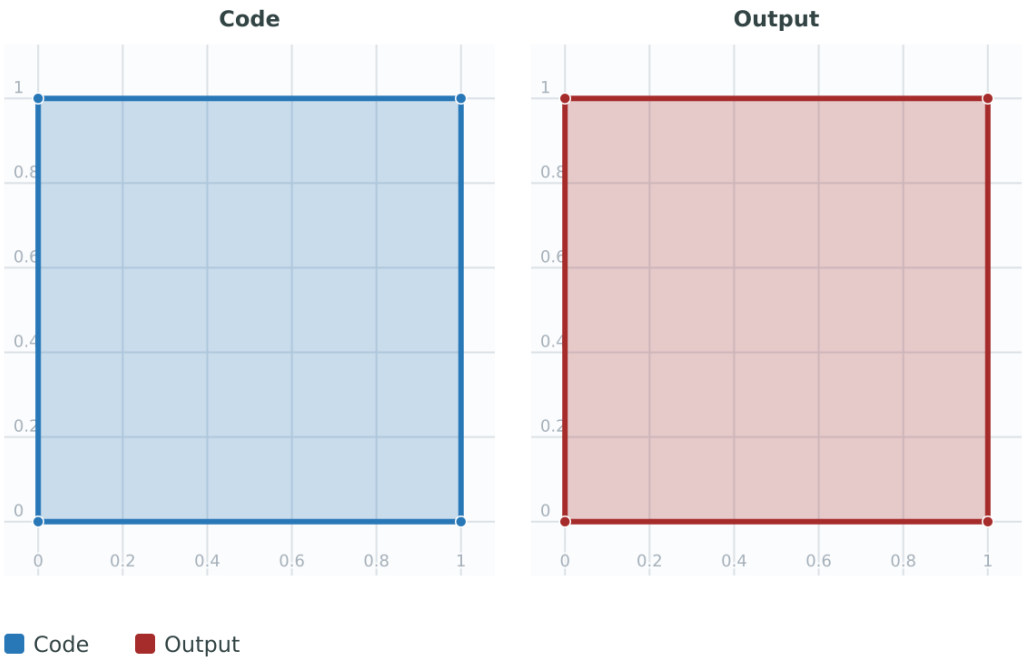
Code

```
SELECT ST_Envelope('POLYGON((0 0,0 1,1.0000001 1,1.0000001 0,0 0))'::geometry);
```

Output

```
POLYGON((0 0,0 1,1.0000001 1,1.0000001 0,0 0))
```

Figure



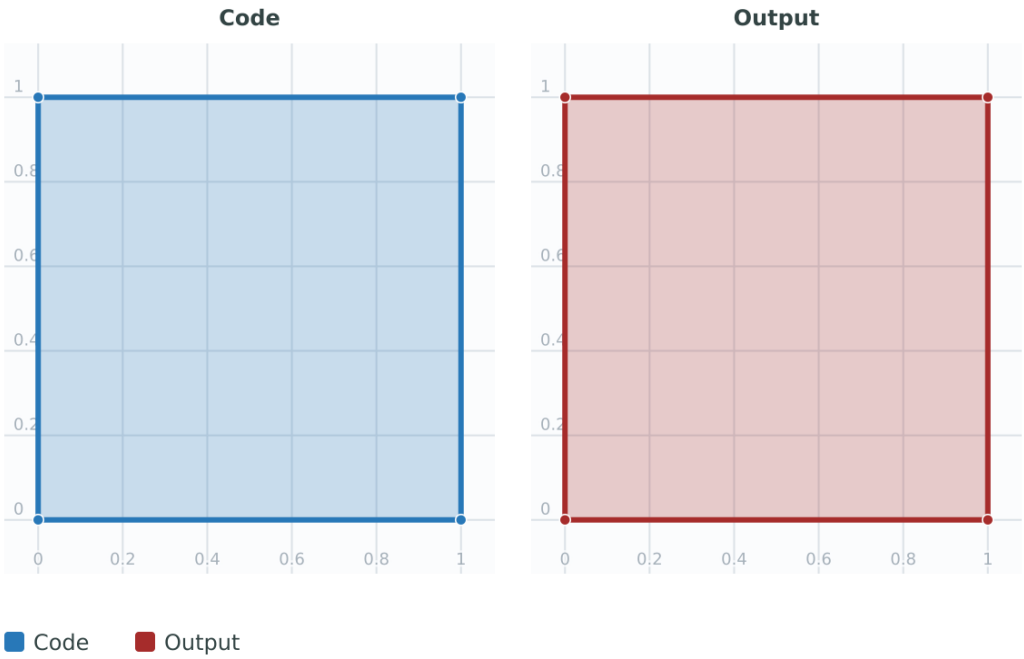
Code

```
SELECT ST_Envelope('POLYGON((0 0,0 1,1.0000000001 1,1.0000000001 0,0 0))'::geometry);
```

Output

```
POLYGON((0 0,0 1,1.0000000001 1,1.0000000001 0,0 0))
```

Figure



Code

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

Envelope of a point and LineString.

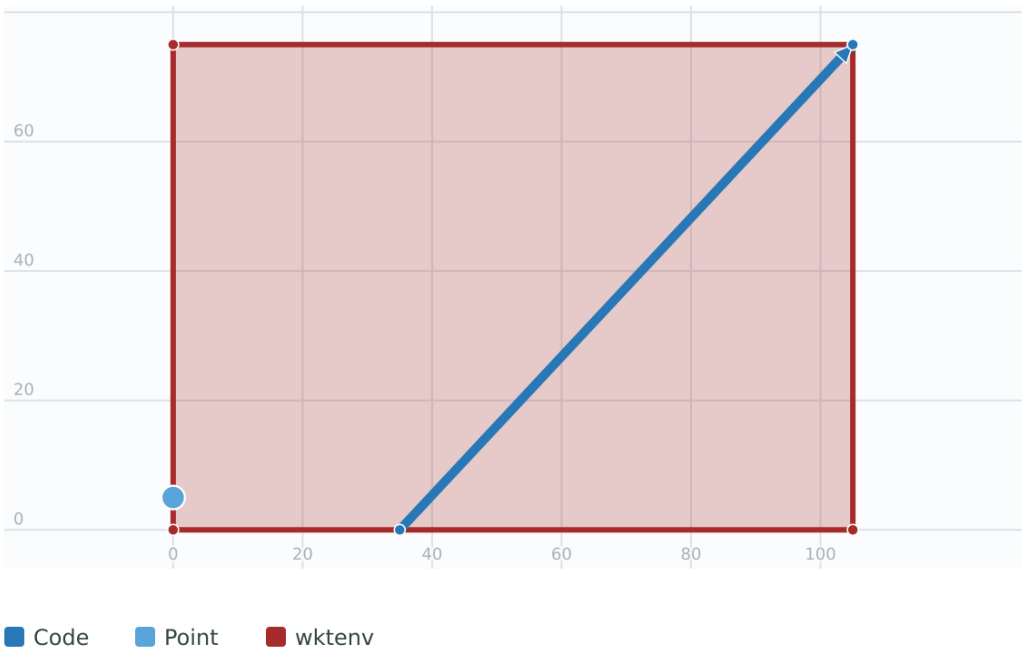
Code

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

Output

```
POLYGON((20 75,20 150,125 150,125 75,20 75))
```

Figure



Siehe auch

Box2D, Box3D, ST_OrientedEnvelope

7.4.12 ST_ExteriorRing

ST_ExteriorRing — Gibt die Anzahl der inneren Ringe einer Polygongeometrie aus.

Synopsis

geometry **ST_ExteriorRing**(geometry a_polygon);

Beschreibung

Gibt einen Linienzug zurück, welcher den äußeren Ring der `POLYGON` Geometrie darstellt. Gibt NULL zurück wenn es sich bei der Geometrie um kein Polygon handelt.



Anmerkung
Dies funktioniert nicht mit MULTIPOLYGONen. Verwenden Sie die Funktion bitte in Zusammenhang mit ST_Dump um sie auf MULTIPOLYGONe anzuwenden.

 Diese Methode implementiert die OGC Simple Features Implementation Specification for SQL 1.1. 2.1.5.1

 Diese Methode setzt die SQL/MM-Spezifikation um. SQL-MM 3: 8.2.3, 8.3.3

 Diese Funktion unterstützt 3d und lässt den Z-Index nicht fallen.

Beispiele

Extract the exterior ring from a Polygon with a hole.

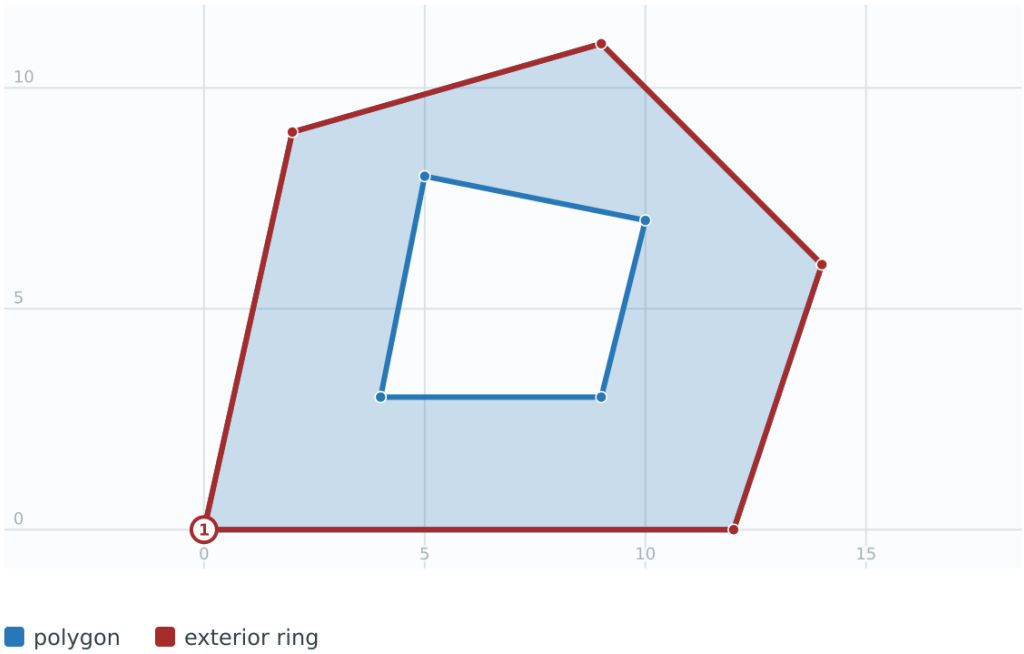
Code

```
WITH input(geom) AS (VALUES (
  'POLYGON((0 0,2 9,9 11,14 6,12 0,0 0),(4 3,9 3,10 7,5 8,4 3))'::geometry
))
SELECT geom AS input_polygon,
       ST_ExteriorRing(geom) AS exterior_ring
FROM input;
```

Output

```
POLYGON((0 0,2 9,9 11,14 6,12 0,0 0),(4 3,9 3,10 7,5 8,4 3)) | LINESTRING(0 0,2 9,9 11,14 6,12 0,0 0)
```

Figure



For a MultiPolygon, dump the component Polygons before extracting their exterior rings. Both component Polygons in this example have holes.

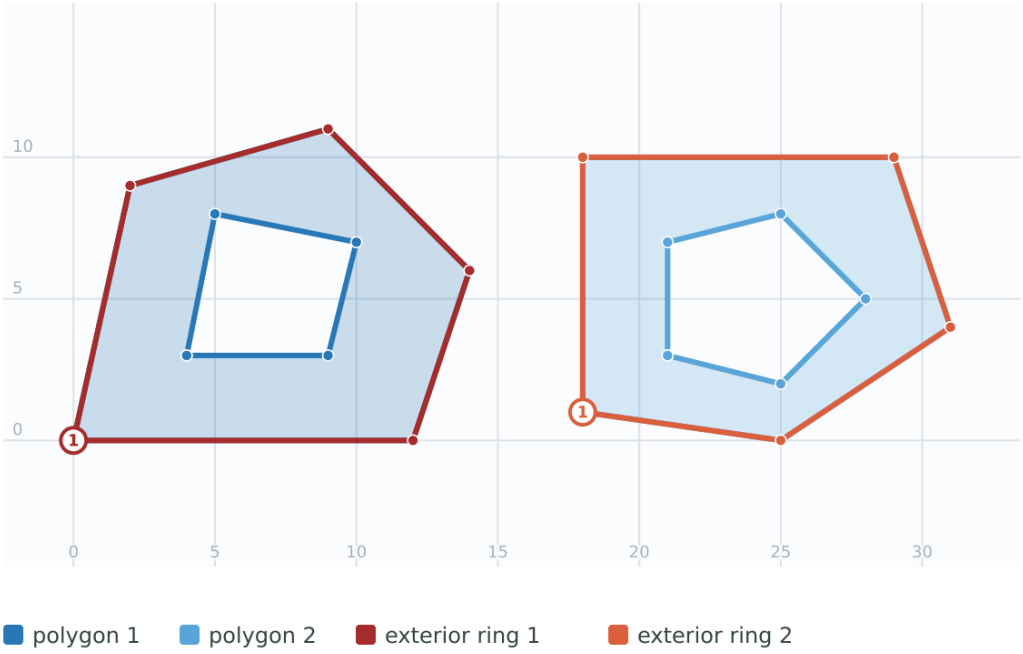
Code

```
WITH input(geom) AS (VALUES (
  'MULTIPOLYGON(
    ((0 0,2 9,9 11,14 6,12 0,0 0),(4 3,9 3,10 7,5 8,4 3)),
    ((18 1,18 10,29 10,31 4,25 0,18 1),(21 3,25 2,28 5,25 8,21 7,21 3))
  )'::geometry
))
SELECT part.geom AS input_polygon,
       ST_ExteriorRing(part.geom) AS exterior_ring
FROM input
CROSS JOIN LATERAL ST_Dump(geom) AS part
ORDER BY part.path;
```

Output

```
POLYGON((0 0,2 9,9 11,14 6,12 0,0 0),(4 3,9 3,10 7,5 8,4 3)) | LINESTRING(0 0,2 9,9 11,14 6,12 0,0 0)
POLYGON((18 1,18 10,29 10,31 4,25 0,18 1),(21 3,25 2,28 5,25 8,21 7,21 3)) | LINESTRING(18 1,18 10,29 10,31 4,25 0,18 ↩
    1)
```

Figure



Extract exterior rings from a table of Polygons.

Code

```
SELECT gid, ST_ExteriorRing(geom) AS ering
FROM sometable;
```

For a table of MultiPolygons, dump the component Polygons and collect their exterior rings into a MultiLineString.

Code

```
SELECT gid, ST_Collect(ST_ExteriorRing(geom)) AS erings
FROM (
  SELECT gid, (ST_Dump(geom)).geom AS geom
  FROM sometable
) AS foo
GROUP BY gid;
```

This example returns the 3D exterior ring of a Polygon.

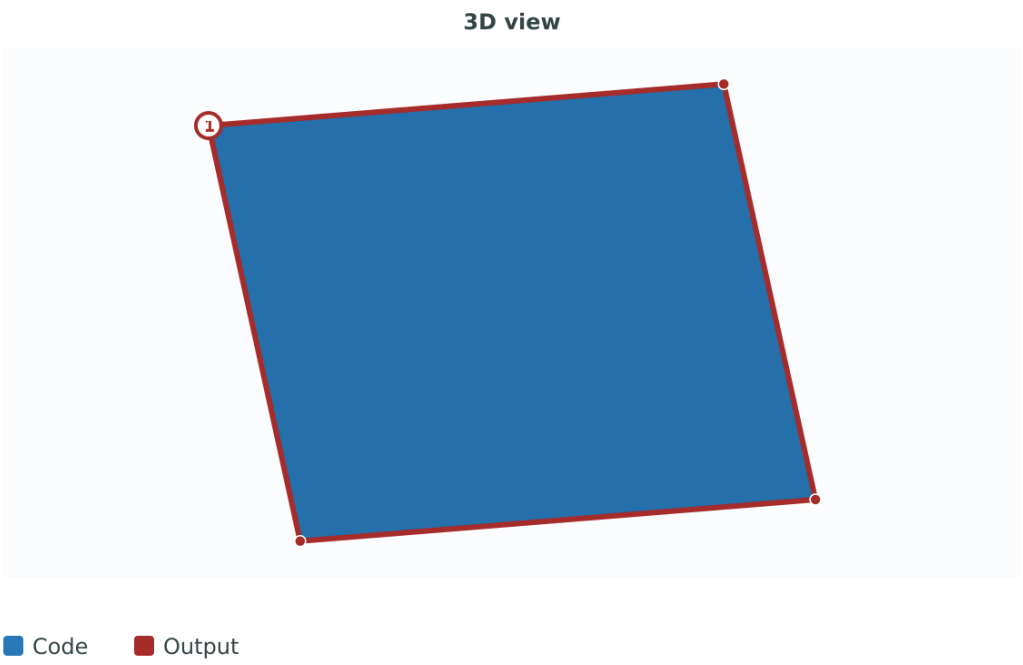
Code

```
SELECT ST_ExteriorRing(
    'POLYGON Z ((0 0 5,4 0 5,4 3 5,0 3 5,0 0 5))'
    );
```

Output

```
LINESTRING(0 0 5,4 0 5,4 3 5,0 3 5,0 0 5)
```

Figure



Siehe auch

ST_InteriorRingN, ST_Boundary, ST_NumInteriorRings

7.4.13 ST_GeometryN

ST_GeometryN — Gibt den Geometrietyp des ST_Geometry Wertes zurück.

Synopsis

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

Beschreibung

Returns the 1-based Nth element of a GEOMETRYCOLLECTION, MULTIPOINT, MULTILINESTRING, MULTICURVE, or MULTIPOLYGON. For a unitary geometry, including a TIN or PolyhedralSurface, an index of 1 returns the input geometry and any other index returns NULL. Use ST_PatchN and ST_NumPatches to access and count the faces of a TIN or PolyhedralSurface.



Anmerkung
Seit Version 0.8.0 basiert der Index auf 1, so wie in der OGC Spezifikation. Vorhergegangene Versionen waren 0-basiert.



Anmerkung
Falls Sie alle Geometrien einer Geometrie entnehmen wollen, so ist ST_Dump wesentlich leistungsfähiger und es funktioniert auch mit Einzelgeometrien.

Erweiterung: Mit 2.0.0 wurde die Unterstützung für polyedrische Oberflächen, Dreiecke und TIN eingeführt.

Änderung: 2.0.0 Vorangegangene Versionen geben bei Einzelgeometrien NULL zurück. Dies wurde geändert um die Geometrie für den ST_GeometrieN(..,1) Fall zurückzugeben.

Changed: 3.6.0 TIN and PolyhedralSurface are treated as unitary geometries and are not decomposed into patches; use ST_PatchN to access faces.

 Diese Methode implementiert die OGC Simple Features Implementation Specification for SQL 1.1.

 Diese Methode setzt die SQL/MM-Spezifikation um. SQL-MM 3: 9.1.5

 Diese Funktion unterstützt 3d und lässt den Z-Index nicht fallen.

 Diese Methode unterstützt kreisförmige Strings und Kurven.

 Diese Funktion unterstützt polyedrische Flächen.

 Diese Funktion unterstützt Dreiecke und dreieckige unregelmäßige Netzoberflächen (TIN).

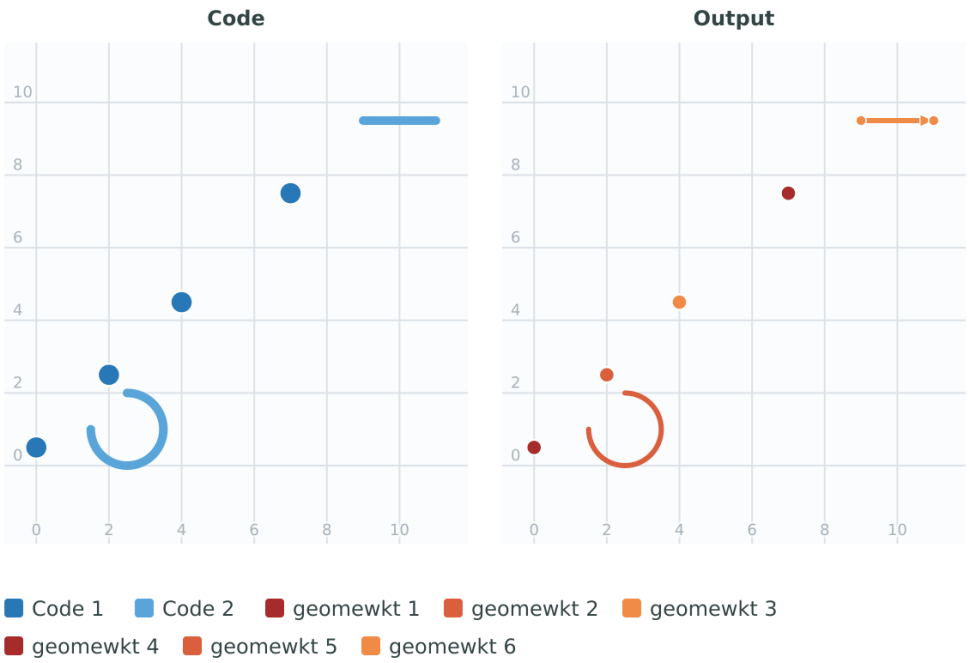
Standard Beispiele

Extracting a subset of points from a 3D multipoint.

Code

```
SELECT n, ST_GeometryN(geom, n) As geomewkt
FROM (
VALUES
    (1, 'MULTIPOINT((1 2 7), (3 4 7), (5 6 7), (8 9 10))'::geometry ),
    (2, 'MULTICURVE(CIRCULARSTRING(2.5 2.5,4.5 2.5, 3.5 3.5), (10 11, 12 11))'::geometry )
    ) As foo(id, geom)
    CROSS JOIN generate_series(1,100) n
WHERE n <= ST_NumGeometries(geom)
ORDER BY id, n;
```

Figure



Extracting all geometries (useful when you want to assign an id).

Code

```
SELECT gid, n, ST_GeometryN(geom, n)
FROM sometable CROSS JOIN generate_series(1,100) n
WHERE n <= ST_NumGeometries(geom);
```

Beispiele für polyedrische Oberflächen, TIN und Dreieck

A PolyhedralSurface or TIN is treated as a single geometry, not as a collection of faces. `ST_GeometryN` does not decompose the surface into patches, so an index greater than 1 returns `NULL`. Use ST_PatchN and ST_NumPatches to access individual faces.

This example requests the third element of a PolyhedralSurface.

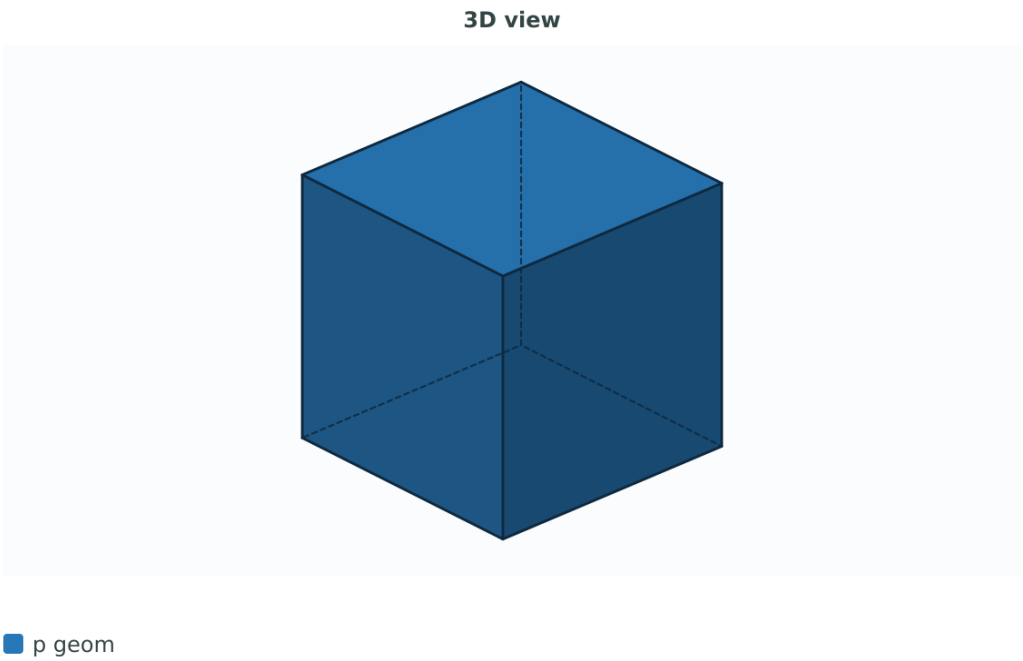
Code

```
SELECT ST_GeometryN(p_geom, 3) As geom_ewkt
  FROM (SELECT 'POLYHEDRALSURFACE(
((0 0 0,0 0 1,0 1 1,0 1 0,0 0 0)),
((0 0 0,0 1 0,1 1 0,1 0 0,0 0 0)),
((0 0 0,1 0 0,1 0 1,0 0 1,0 0 0)),
((1 1 0,1 1 1,1 0 1,1 0 0,1 1 0)),
((0 1 0,0 1 1,1 1 1,1 1 0,0 1 0)),
((0 0 1,1 0 1,1 1 1,0 1 1,0 0 1))
)'::geometry  AS p_geom )  AS a;
```

Output

```
NULL
```

Figure



This example requests the second element of a TIN.

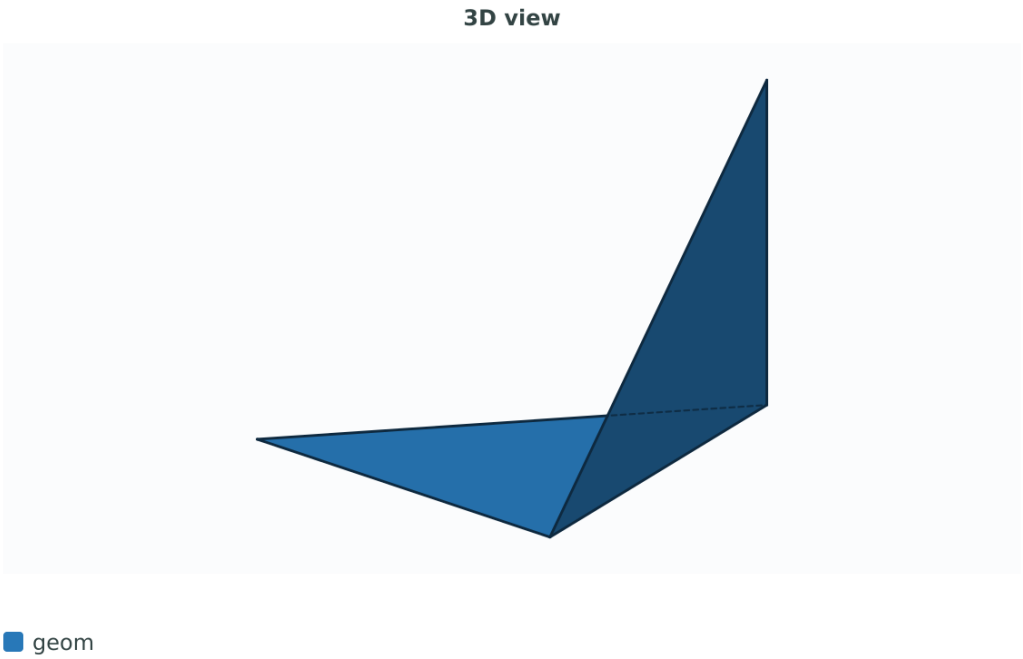
Code

```
SELECT ST_GeometryN(geom, 2) as wkt
  FROM
    (SELECT
       'TIN (((
                0 0 0,
                0 0 1,
                0 1 0,
                0 0 0
            )),((
                0 0 0,
                0 1 0,
                1 1 0,
                0 0 0
            ))
            )'::geometry  AS geom
    ) AS g;
```

Output

```
NULL
```

Figure



Siehe auch

ST_Dump, ST_NumGeometries, ST_PatchN, ST_NumPatches

7.4.14 ST_GeometryType

ST_GeometryType — Gibt den Geometrietyp des ST_Geometry Wertes zurück.

Synopsis

text **ST_GeometryType**(geometry g1);

Beschreibung

Gibt den Geometrietyp als Zeichenkette zurück. Z.B.: 'ST_LineString', 'ST_Polygon','ST_MultiPolygon' etc. Diese Funktion unterscheidet sich von GeometryType(geometry) durch den Präfix ST_ und dadurch, das nicht angezeigt wird, ob die Geometrie eine Maßzahl besitzt.

Erweiterung: Mit 2.0.0 wurde die Unterstützung für polyedrische Oberflächen eingeführt.

Diese Methode setzt die SQL/MM-Spezifikation um. SQL-MM 3: 5.1.4

Diese Funktion unterstützt 3d und lässt den Z-Index nicht fallen.

Diese Funktion unterstützt polyedrische Flächen.

Beispiele

Code

```
SELECT ST_GeometryType('LINESTRING(77.29 29.07,77.42 29.26,77.27 29.31,77.29 29.07)':: ↩
    geometry);
```

Output

```
ST_LineString
```

Code

```
SELECT ST_GeometryType('POLYHEDRALSURFACE( ((0 0 0,0 0 1,0 1 1,0 1 0,0 0 0)),
((0 0 0,0 1 0,1 1 0,1 0 0,0 0 0)),((0 0 0,1 0 0,1 0 1,0 0 1,0 0 0)),
((1 1 0,1 1 1,1 0 1,1 0 0,1 1 0)),
((0 1 0,0 1 1,1 1 1,1 1 0,0 1 0)),((0 0 1,1 0 1,1 1 1,0 1 1,0 0 1)) )'::geometry);
```

Output

```
ST_PolyhedralSurface
```

Code

```
SELECT ST_GeometryType(geom) AS result
FROM
  (SELECT
     'TIN (((
               0 0 0,
               0 0 1,
               0 1 0,
               0 0 0
           )),((
               0 0 0,
               0 1 0,
               1 1 0,
               0 0 0
           ))
           )'::geometry AS geom
  ) AS g;
```

Output

```
ST_Tin
```

Siehe auch

GeometryType

7.4.15 ST_HasArc

ST_HasArc — Prüft, ob eine Geometrie einen Kreisbogen enthält

Synopsis

boolean **ST_HasArc**(geometry geomA);

Beschreibung

Gibt den Wert TRUE zurück, falls es sich bei der Geometrie um eine leere GeometryCollection, Polygon, Point etc. handelt.

Verfügbarkeit: 1.2.2

Diese Funktion unterstützt 3d und lässt den Z-Index nicht fallen.

Diese Methode unterstützt kreisförmige Strings und Kurven.

Beispiele

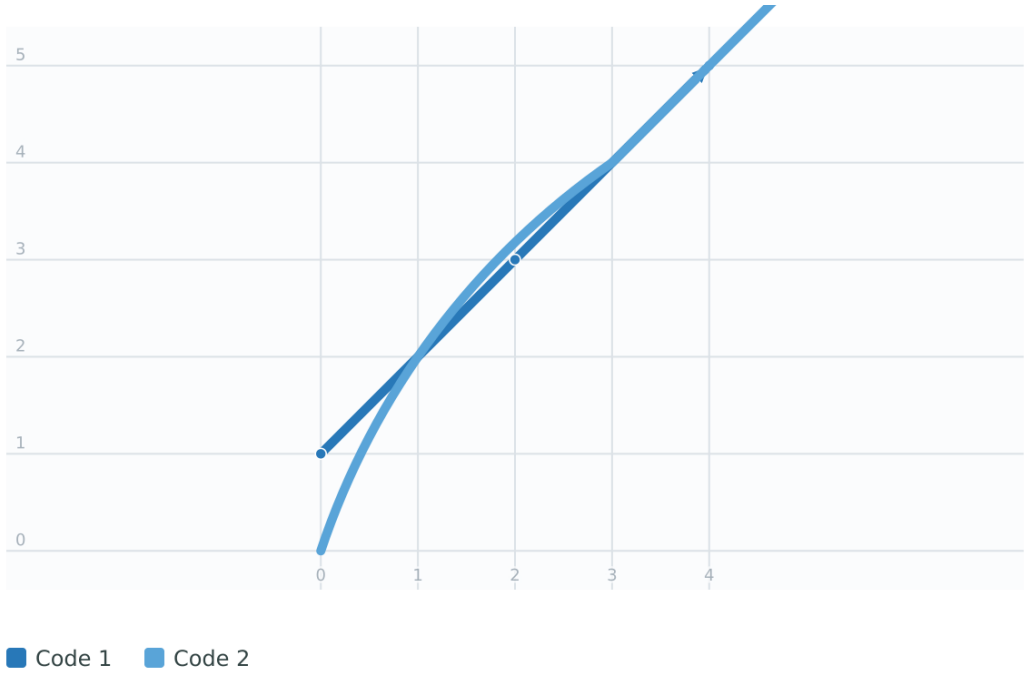
Code

```
SELECT ST_HasArc(ST_Collect('LINESTRING(1 2,3 4,5 6)', 'CIRCULARSTRING(1 1,2 3,4 5,6 7,5 6) ↩
    '));
```

Output

```
t
```

Figure



Siehe auch

ST_CurveToLine, ST_PointN

7.4.16 ST_InteriorRingN

ST_InteriorRingN — Gibt die Anzahl der inneren Ringe einer Polygongeometrie aus.

Synopsis

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

Beschreibung

Gibt den Nten innenliegenden Linienzug des Ringes der Polygongeometrie zurück. Gibt NULL zurück, falls es sich bei der Geometrie nicht um ein Polygon handelt, oder sich das angegebene N außerhalb des zulässigen Bereiches befindet.



Anmerkung
Dies funktioniert nicht mit MULTIPOLYGONen. Verwenden Sie die Funktion bitte in Zusammenhang mit ST_Dump um sie auf MULTIPOLYGONe anzuwenden.

 Diese Methode implementiert die OGC Simple Features Implementation Specification for SQL 1.1.

 Diese Methode setzt die SQL/MM-Spezifikation um. SQL-MM 3: 8.2.6, 8.3.5

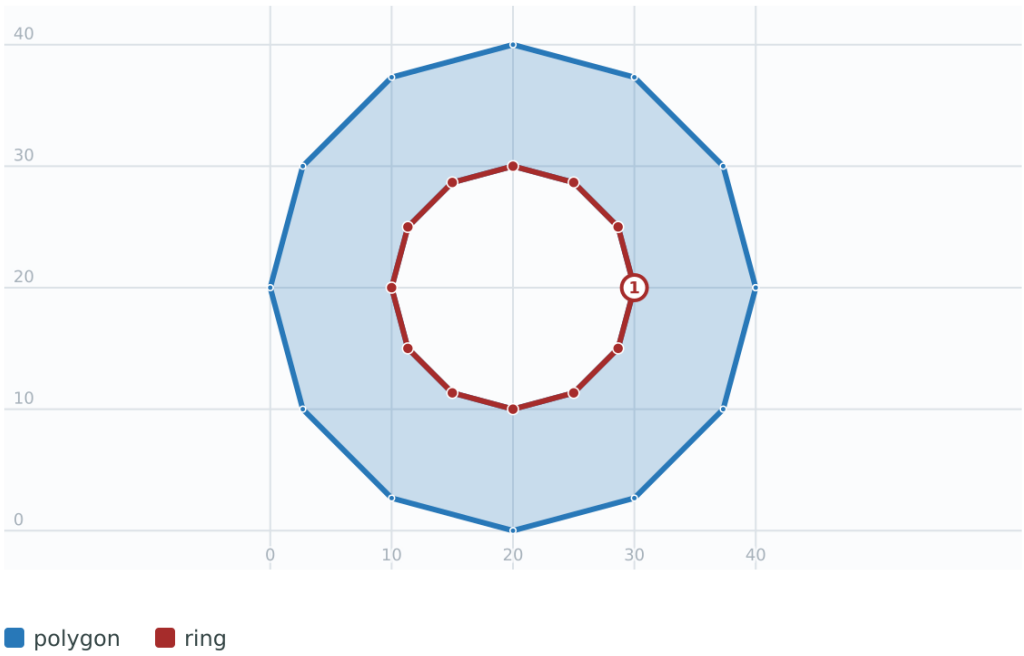
 Diese Funktion unterstützt 3d und lässt den Z-Index nicht fallen.

Beispiele

Code

```
WITH data AS (
  SELECT ST_BuildArea(ST_Collect(
    ST_Buffer(ST_Point(1, 2), 20, 3),
    ST_Buffer(ST_Point(1, 2), 10, 3)
  )) AS geom
)
SELECT geom AS input_polygon,
       ST_InteriorRingN(geom, 1) AS ring
FROM data;
```

Figure



Siehe auch

ST_ExteriorRing, ST_M, ST_X, ST_Y, ST_ZMax, ST_ZMin

7.4.17 ST_NumCurves

ST_NumCurves — Gibt die Anzahl der Kurvenkomponenten einer CompoundCurve zurück.

Synopsis

integer **ST_NumCurves**(geometry a_compoundcurve);

Beschreibung

Gibt die Anzahl der Kurvenkomponenten einer CompoundCurve zurück, Null steht für eine leere CompoundCurve und NULL für keine Eingabe einer CompoundCurve.

 Diese Methode setzt die SQL/MM-Spezifikation um. SQL-MM 3: 8.2.6, 8.3.5

 Diese Funktion unterstützt 3d und lässt den Z-Index nicht fallen.

Beispiele

This CompoundCurve contains three component curves.

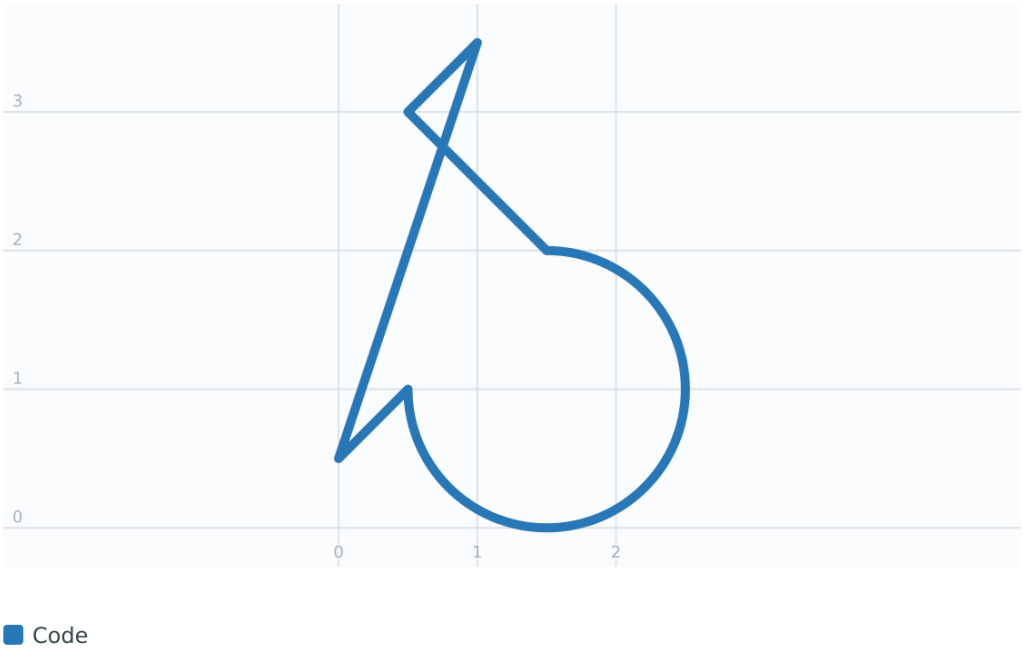
Code

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

Output

```
3
```

Figure



An empty CompoundCurve has no component curves.

Code

```
SELECT ST_NumCurves('COMPOUNDCURVE EMPTY');
```

Output

```
0
```

Siehe auch

ST_CurveN, ST_Dump, ST_ExteriorRing, ST_NumInteriorRings, ST_NumGeometries

7.4.18 ST_CurveN

ST_CurveN — Gibt die N-te Geometrie einer Kurvenkomponente in einer CompoundCurve zurück.

Synopsis

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

Beschreibung

Gibt die N-te Komponente einer Kurvengeometrie von einer CompoundCurve zurück. Der Index beginnt mit 1. Gibt NULL zurück, wenn die Geometrie keine CompoundCurve ist oder der Index außerhalb des Bereichs liegt.

Diese Methode setzt die SQL/MM-Spezifikation um. SQL-MM 3: 8.2.6, 8.3.5

Diese Funktion unterstützt 3d und lässt den Z-Index nicht fallen.

Beispiele

This example extracts the second component curve, a CircularString, from a CompoundCurve.

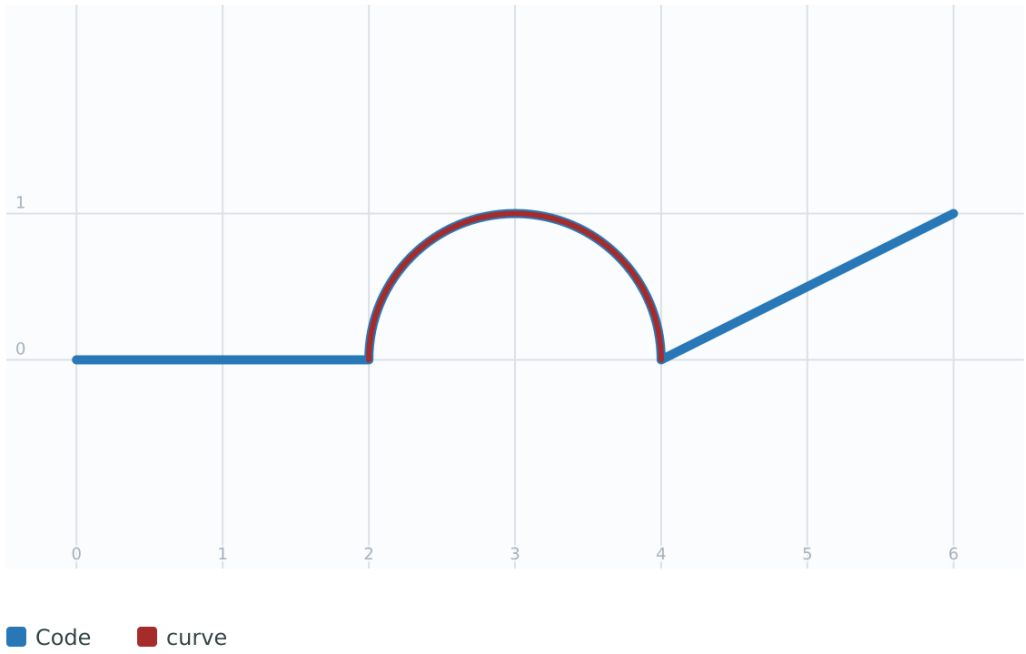
Code

```
SELECT ST_CurveN(
  'COMPOUNDCURVE(
    (0 0,2 0),
    CIRCULARSTRING(2 0,3 1,4 0),
    (4 0,6 1)
  )',
  2) AS curve;
```

Output

```
CIRCULARSTRING(2 0,3 1,4 0)
```

Figure



Siehe auch

ST_NumCurves, ST_Dump, ST_ExteriorRing, ST_NumInteriorRings, ST_NumGeometries

7.4.19 ST_IsClosed

ST_IsClosed — Gibt den Wert `TRUE` zurück, wenn die Anfangs- und Endpunkte des `LINESTRING`'s zusammenfallen. Bei polyedrischen Oberflächen, wenn sie geschlossen (volumetrisch) sind.

Synopsis

boolean **ST_IsClosed**(geometry g);

Beschreibung

Gibt den Wert `TRUE` zurück, wenn die Anfangs- und Endpunkte des `LINESTRING`'s zusammenfallen. Bei polyedrischen Oberflächen wird angezeigt, ob die Oberfläche eine Fläche (offen) oder ein Volumen (geschlossen) beschreibt.

Diese Methode implementiert die OGC Simple Features Implementation Specification for SQL 1.1.

Diese Methode setzt die SQL/MM-Spezifikation um. SQL-MM 3: 7.1.5, 9.3.3

Anmerkung
SQL-MM definiert das Ergebnis von `ST_IsClosed(NULL)` als 0, während PostGIS `NULL` zurückgibt.

Diese Funktion unterstützt 3d und lässt den Z-Index nicht fallen.

Diese Methode unterstützt kreisförmige Strings und Kurven.

Erweiterung: Mit 2.0.0 wurde die Unterstützung für polyedrische Oberflächen eingeführt.

Diese Funktion unterstützt polyedrische Flächen.

Beispiele für Linienzüge und Punkte

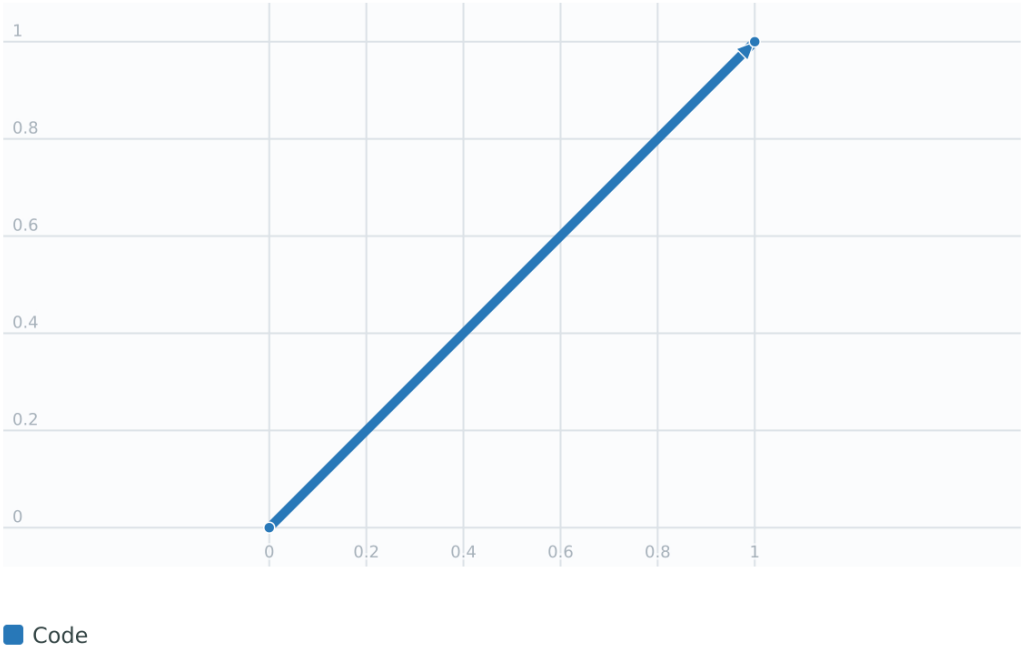
Code

```
SELECT ST_IsClosed('LINESTRING(0 0,1 1)'::geometry);
```

Output

```
f
```

Figure



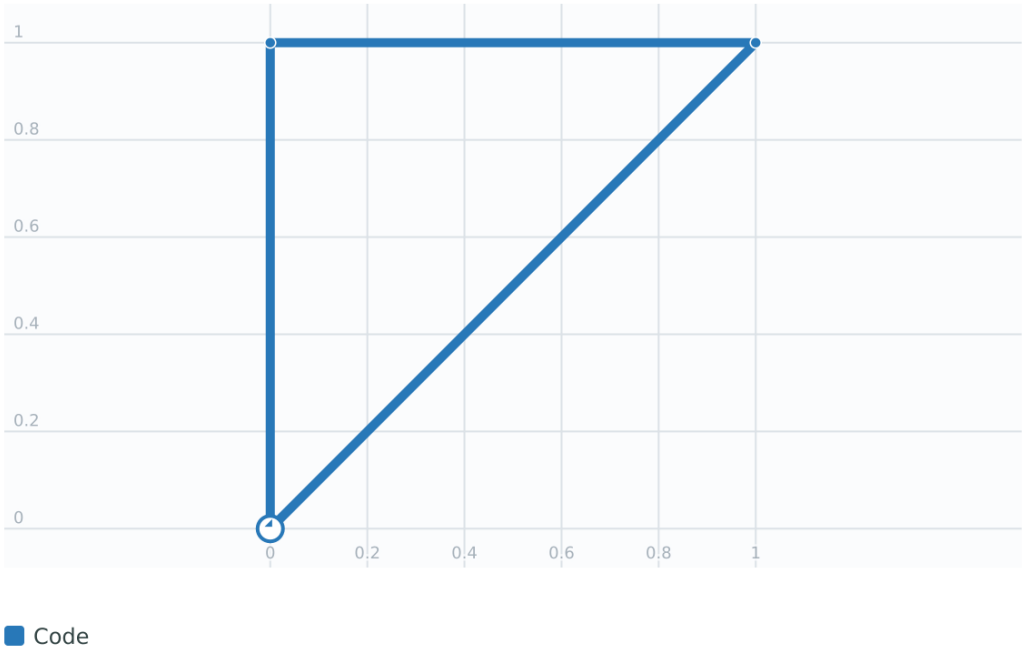
Code

```
SELECT ST_IsClosed('LINESTRING(0 0,0 1,1 1,0 0)'::geometry);
```

Output

```
t
```

Figure



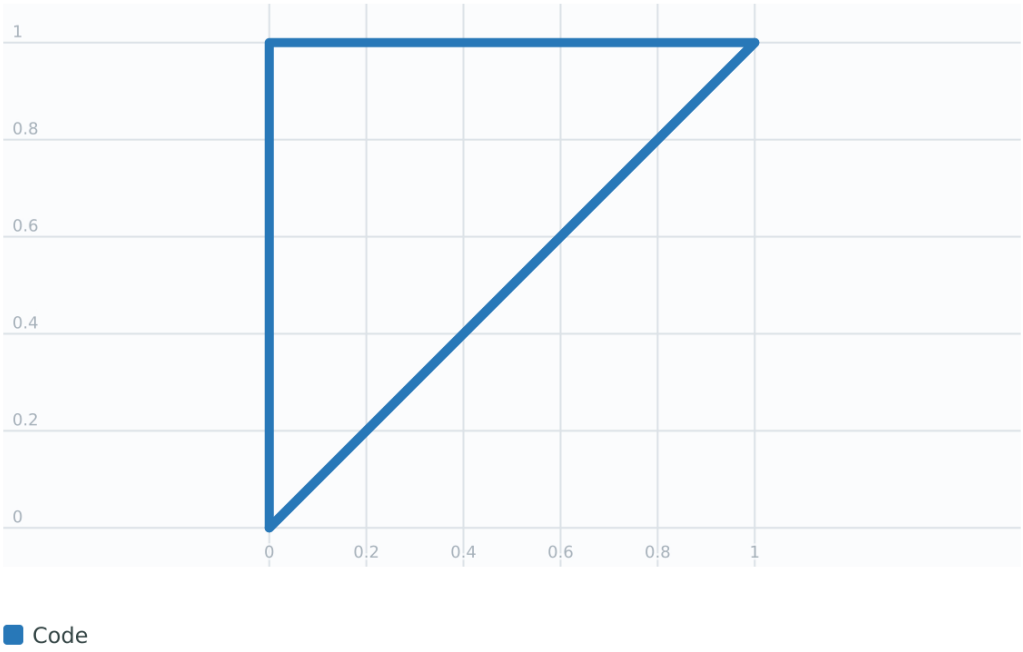
Code

```
SELECT ST_IsClosed('MULTILINESTRING((0 0,0 1,1 1,0 0),(0 0,1 1))'::geometry);
```

Output

```
f
```

Figure



Code

```
SELECT ST_IsClosed('POINT(0 0)'::geometry);
```

Output

```
t
```

Code

```
SELECT ST_IsClosed('MULTIPOINT((0 0),(1 1))'::geometry);
```

Output

```
t
```

Beispiel für eine polyedrische Oberfläche

This example uses a cube.

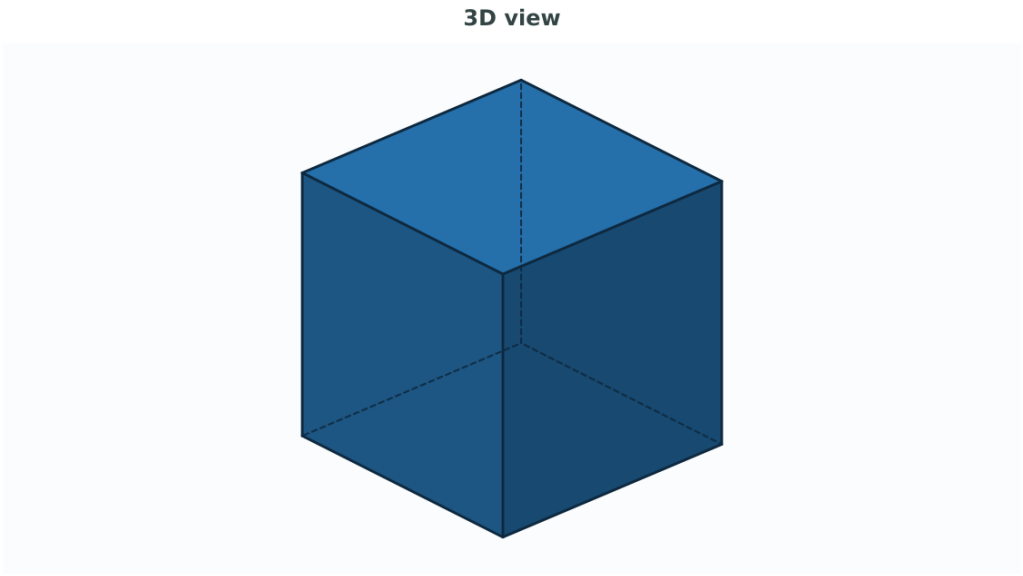
Code

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

Output

```
t
```

Figure



Code

This example uses the same cube with one side missing.

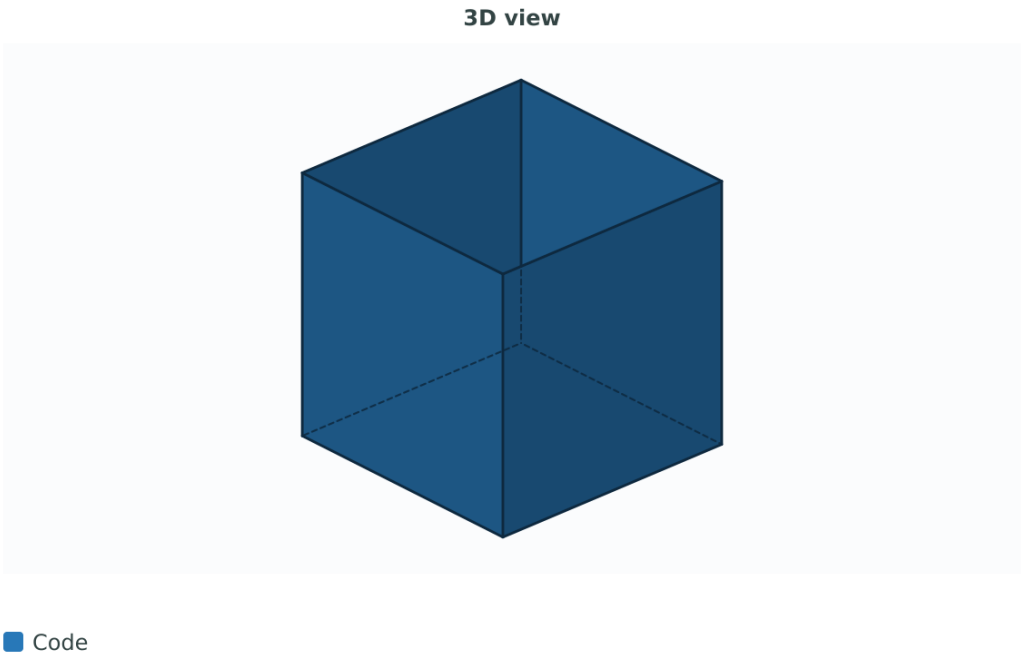
Code

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

Output

```
f
```

Figure



Siehe auch

ST_IsRing

7.4.20 ST_IsCollection

ST_IsCollection — Gibt den Wert TRUE zurück, falls es sich bei der Geometrie um eine leere GeometryCollection, Polygon, Point etc. handelt.

Synopsis

boolean **ST_IsCollection**(geometry g);

Beschreibung

Gibt den Wert `TRUE` zurück, wenn der Geometrietyp einer der folgenden Gemetrietypen entspricht:

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



Anmerkung
Diese Funktion wertet den Geometrietyp aus. D.h.: sie gibt den Wert `TRUE` für Geometriekollektionen zurück, wenn diese leer sind, oder nur ein einziges Element aufweisen.

 Diese Funktion unterstützt 3d und lässt den Z-Index nicht fallen.

 Diese Methode unterstützt kreisförmige Strings und Kurven.

Beispiele

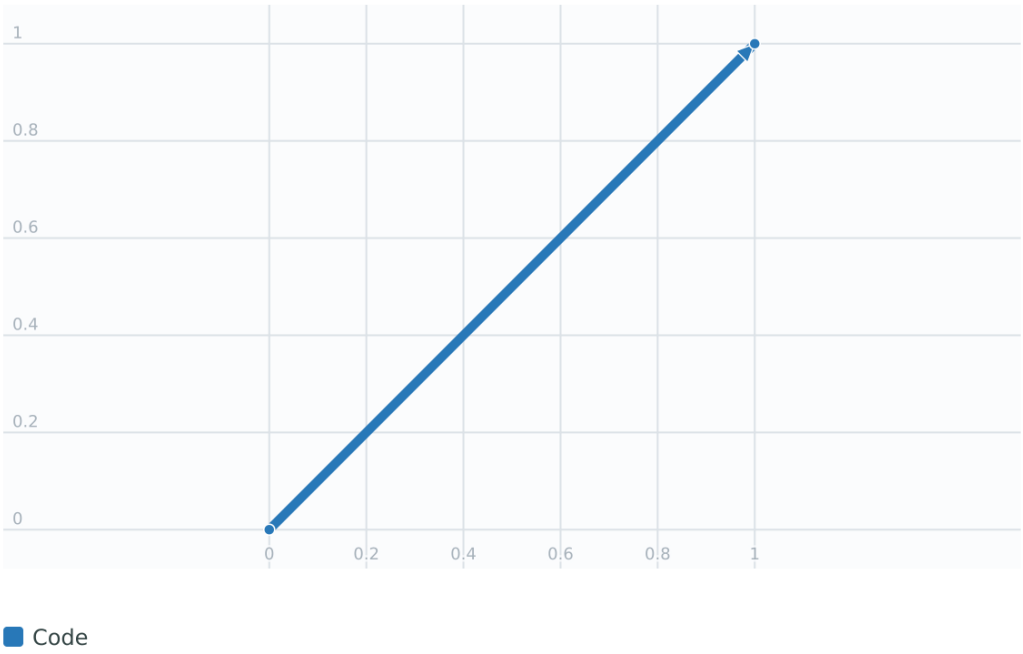
Code

```
SELECT ST_IsCollection('LINESTRING(0 0,1 1)'::geometry);
```

Output

```
f
```

Figure



Code

```
SELECT ST_IsCollection('MULTIPOINT EMPTY'::geometry);
```

Output

```
t
```

Code

```
SELECT ST_IsCollection('MULTIPOINT((0 0))'::geometry);
```

Output

```
t
```

Code

```
SELECT ST_IsCollection('MULTIPOINT((0 0),(42 42))'::geometry);
```

Output

```
t
```

Code

```
SELECT ST_IsCollection('GEOMETRYCOLLECTION(POINT(0 0))'::geometry);
```

Output

```
t
```

Siehe auch

ST_NumGeometries

7.4.21 ST_IsEmpty

ST_IsEmpty — Prüft, ob eine Geometrie leer ist.

Synopsis

boolean **ST_IsEmpty**(geometry geomA);

Beschreibung

Gibt den Wert TRUE zurück, wenn es sich um eine leere Geometrie handelt. Falls TRUE, dann repräsentiert diese Geometrie eine leere GeometryCollection, Polygon, Point etc.



Anmerkung
SQL-MM gibt vor, daß das Ergebnis von ST_IsEmpty(NULL) der Wert 0 ist, während PostGIS den Wert NULL zurückgibt.

 Diese Methode implementiert die OGC Simple Features Implementation Specification for SQL 1.1. s2.1.1.1

 Diese Methode setzt die SQL/MM-Spezifikation um. SQL-MM 3: 5.1.7

 Diese Methode unterstützt kreisförmige Strings und Kurven.

Änderung: 2.0.0 - In Vorgängerversionen von PostGIS war ST_GeomFromText('GEOMETRYCOLLECTION(EMPTY)') erlaubt. Um eine bessere Übereinstimmung mit der SQL/MM Norm zu erreichen, ist dies nun nicht mehr gestattet.

Beispiele

Code

```
SELECT ST_IsEmpty('GEOMETRYCOLLECTION EMPTY'::geometry);
```

Output

```
t
```

Code

```
SELECT ST_IsEmpty('POLYGON EMPTY'::geometry);
```

Output

```
t
```

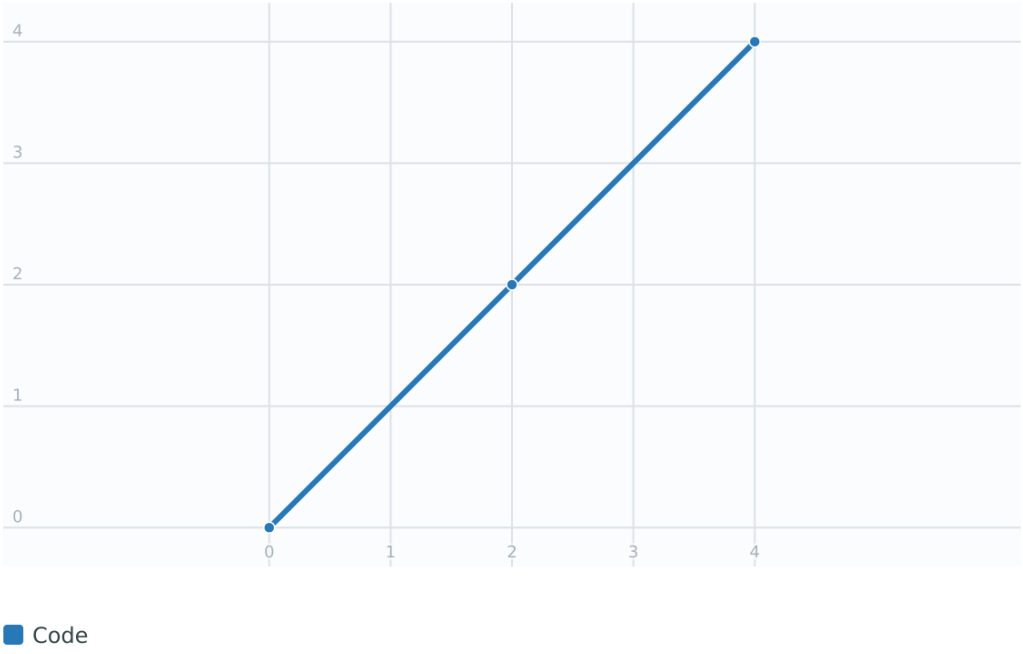
Code

```
SELECT ST_IsEmpty('POLYGON((1 2,3 4,5 6,1 2))'::geometry);
```

Output

```
f
```

Figure



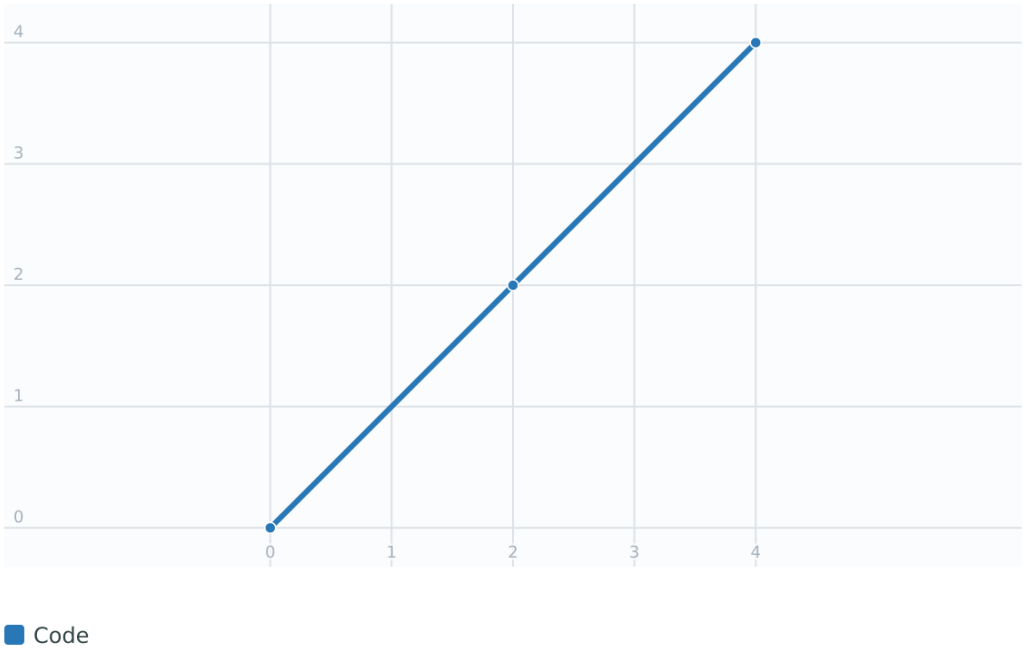
Code

```
SELECT ST_IsEmpty('POLYGON((1 2,3 4,5 6,1 2))'::geometry) = false;
```

Output

```
t
(1 row)
```

Figure



Code

```
SELECT ST_IsEmpty('CIRCULARSTRING EMPTY'::geometry);
```

Output

```
t
```

7.4.22 ST_IsPolygonCCW

ST_IsPolygonCCW — Gibt TRUE zurück, wenn alle äußeren Ringe gegen den Uhrzeigersinn orientiert sind und alle inneren Ringe im Uhrzeigersinn ausgerichtet sind.

Synopsis

boolean **ST_IsPolygonCCW**(geometry geom);

Beschreibung

Gibt TRUE zurück, wenn für alle Bestandteile der angegebenen Geometrie gilt: die äußeren Ringe sind gegen den Uhrzeigersinn und die inneren Ringe im Uhrzeigersinn ausgerichtet.

Gibt TRUE zurück, wenn die Geometrie keine Polygonbestandteile aufweist.



Anmerkung
Da geschlossene Linienzüge nicht als Polygonbestandteile betrachtet werden, erhalten Sie auch dann TRUE, wenn Sie einen einzelnen geschlossenen Linienzug eingeben und zwar unabhängig von dessen Ausrichtung.



Anmerkung
Wenn bei einer Polygongeometrie die inneren Ringe nicht entgegengesetzt orientiert sind (insbesondere, wenn einer oder mehrere innere Ringe die selbe Ausrichtung wie die äußeren Ringe haben), dann geben sowohl ST_IsPolygonCW als auch ST_IsPolygonCCW den Wert FALSE zurück.

Verfügbarkeit: 2.4.0


Diese Funktion unterstützt 3d und lässt den Z-Index nicht fallen.


Diese Funktion unterstützt M-Koordinaten.

Siehe auch

ST_ForcePolygonCW, ST_ForcePolygonCCW, ST_IsPolygonCW

7.4.23 ST_IsPolygonCW

ST_IsPolygonCW — Gibt den Wert TRUE zurück, wenn alle äußeren Ringe im Uhrzeigersinn und alle inneren Ringe gegen den Uhrzeigersinn ausgerichtet sind.

Synopsis

boolean **ST_IsPolygonCW**(geometry geom);

Beschreibung

Gibt den Wert TRUE zurück, wenn für alle Polygonbestandteile der eingegebenen Geometrie gilt: die äußeren Ringe sind im Uhrzeigersinn orientiert, die inneren Ringe entgegen dem Uhrzeigersinn.

Gibt TRUE zurück, wenn die Geometrie keine Polygonbestandteile aufweist.



Anmerkung
Da geschlossene Linienzüge nicht als Polygonbestandteile betrachtet werden, erhalten Sie auch dann TRUE, wenn Sie einen einzelnen geschlossenen Linienzug eingeben und zwar unabhängig von dessen Ausrichtung.



Anmerkung
Wenn bei einer Polygongeometrie die inneren Ringe nicht entgegengesetzt orientiert sind (insbesondere, wenn einer oder mehrere innere Ringe die selbe Ausrichtung wie die äußeren Ringe haben), dann geben sowohl ST_IsPolygonCW als auch ST_IsPolygonCCW den Wert FALSE zurück.

Verfügbarkeit: 2.4.0


Diese Funktion unterstützt 3d und lässt den Z-Index nicht fallen.


Diese Funktion unterstützt M-Koordinaten.

Siehe auch

ST_ForcePolygonCW, ST_ForcePolygonCCW, ST_IsPolygonCW

7.4.24 ST_IsRing

ST_IsRing — Prüft, ob ein LineString geschlossen und einfach ist.

Synopsis

boolean **ST_IsRing**(geometry g);

Beschreibung

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

Diese Methode implementiert die OGC Simple Features Implementation Specification for SQL 1.1. 2.1.5.1

Diese Methode setzt die SQL/MM-Spezifikation um. SQL-MM 3: 7.1.6



Anmerkung
SQL-MM gibt vor, daß das Ergebnis von`ST_IsRing(NULL)` der Wert 0 sein soll, während PostGIS den Wert `NULL` zurückgibt.

Beispiele

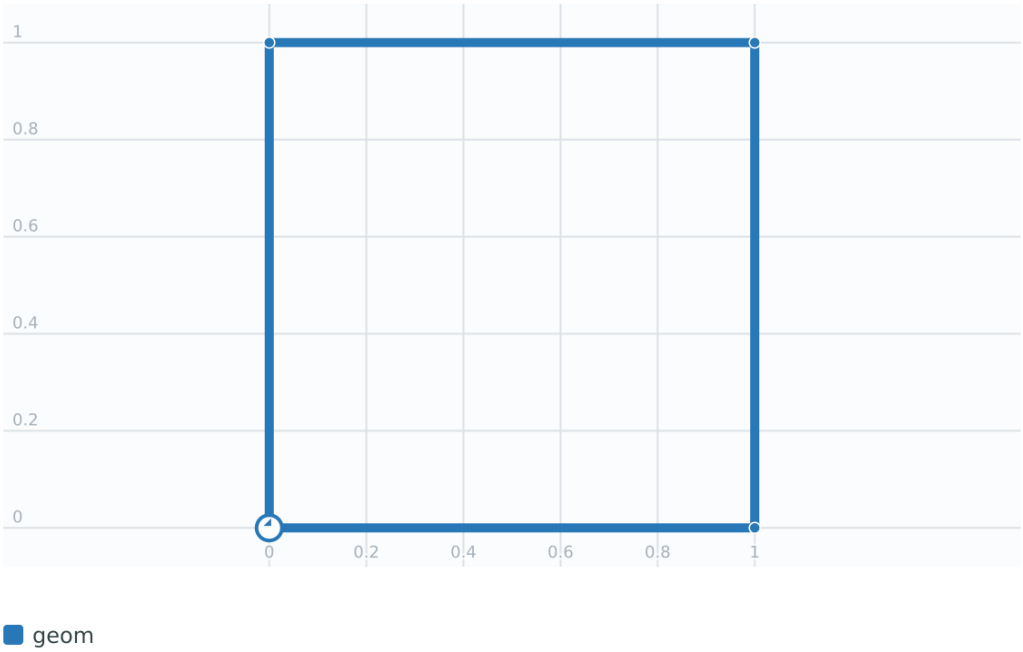
Code

```
SELECT ST_IsRing(geom), ST_IsClosed(geom), ST_IsSimple(geom)
FROM (SELECT 'LINESTRING(0 0,0 1,1 1,1 0,0 0)'::geometry AS geom) AS foo;
```

Output

```
st_isring | st_isclosed | st_issimple
-----------+-------------+-------------
 t         | t           | t
(1 row)
```

Figure



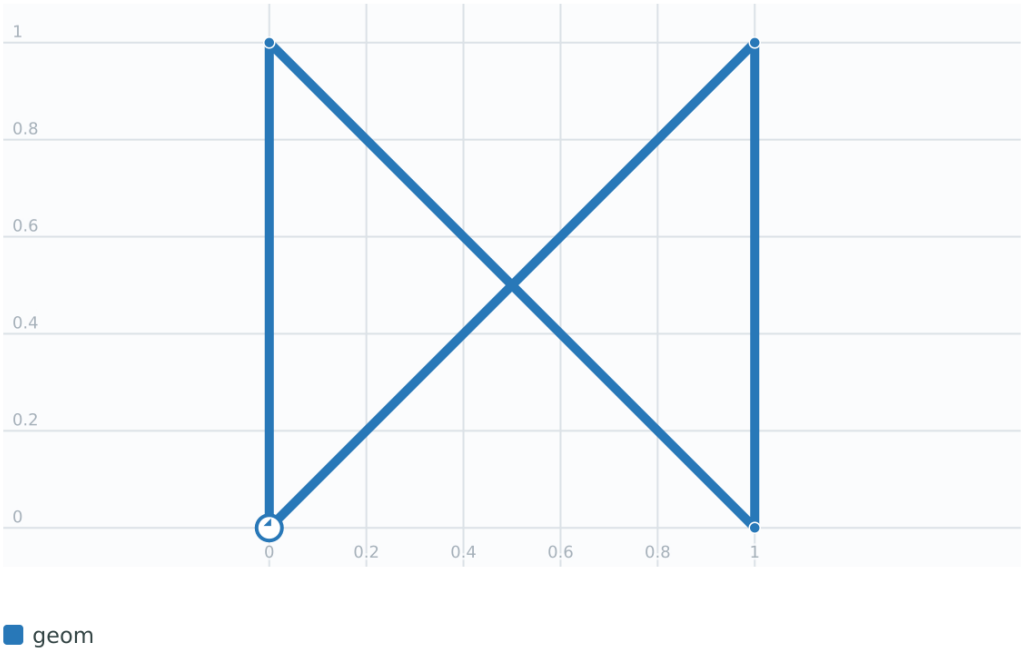
Code

```
SELECT ST_IsRing(geom), ST_IsClosed(geom), ST_IsSimple(geom)
FROM (SELECT 'LINESTRING(0 0,0 1,1 0,1 1,0 0)'::geometry AS geom) AS foo;
```

Output

```
st_isring | st_isclosed | st_issimple
-----------+-------------+-------------
 f         | t           | f
(1 row)
```

Figure



Siehe auch

ST_IsClosed, ST_IsSimple, ST_StartPoint, ST_EndPoint

7.4.25 ST_IsSimple

ST_IsSimple — Gibt den Wert (TRUE) zurück, wenn die Geometrie keine irregulären Stellen, wie Selbstüberschneidungen oder Selbstberührungen, aufweist.

Synopsis

boolean **ST_IsSimple**(geometry geomA);

Beschreibung

Gibt TRUE zurück, wenn keine regelwidrigen geometrischen Merkmale, wie Geometrien die sich selbst kreuzen oder berühren, auftreten. Für weiterführende Information zur OGC-Definition von Simplizität und Gültigkeit von Geometrien, siehe "Ensuring OpenGIS compliance of geometries"

Anmerkung
SQL-MM definiert das Ergebnis von ST_IsSimple(NULL) als 0, während PostGIS NULL zurückgibt.

Diese Methode implementiert die OGC Simple Features Implementation Specification for SQL 1.1. s2.1.1.1

Diese Methode setzt die SQL/MM-Spezifikation um. SQL-MM 3: 5.1.8

Diese Funktion unterstützt 3d und lässt den Z-Index nicht fallen.

Beispiele

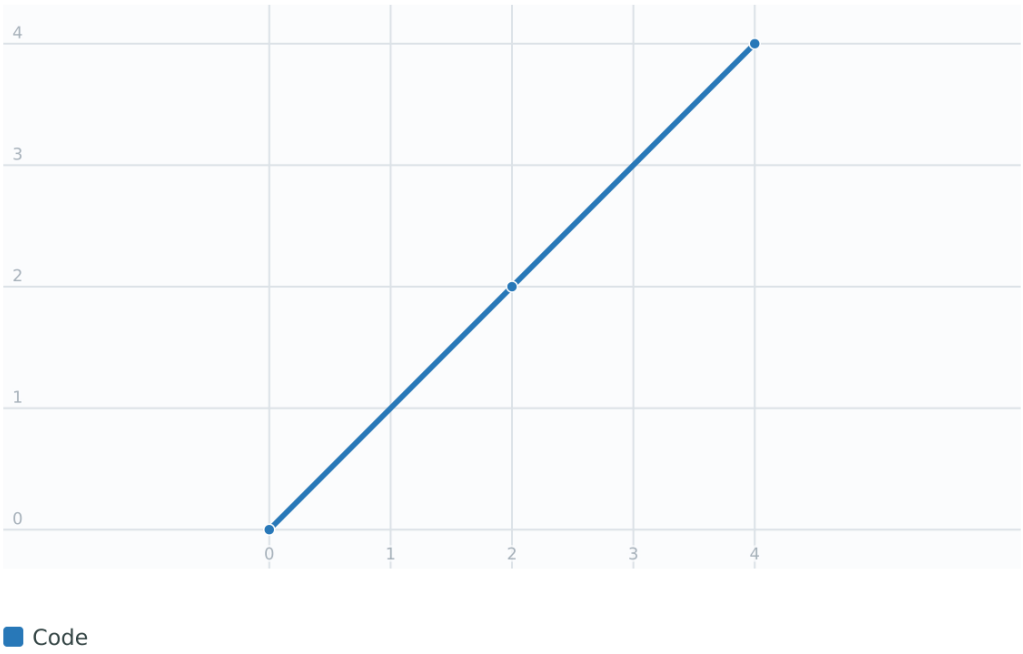
Code

```
SELECT ST_IsSimple('POLYGON((1 2,3 4,5 6,1 2))');
```

Output

```
f
```

Figure



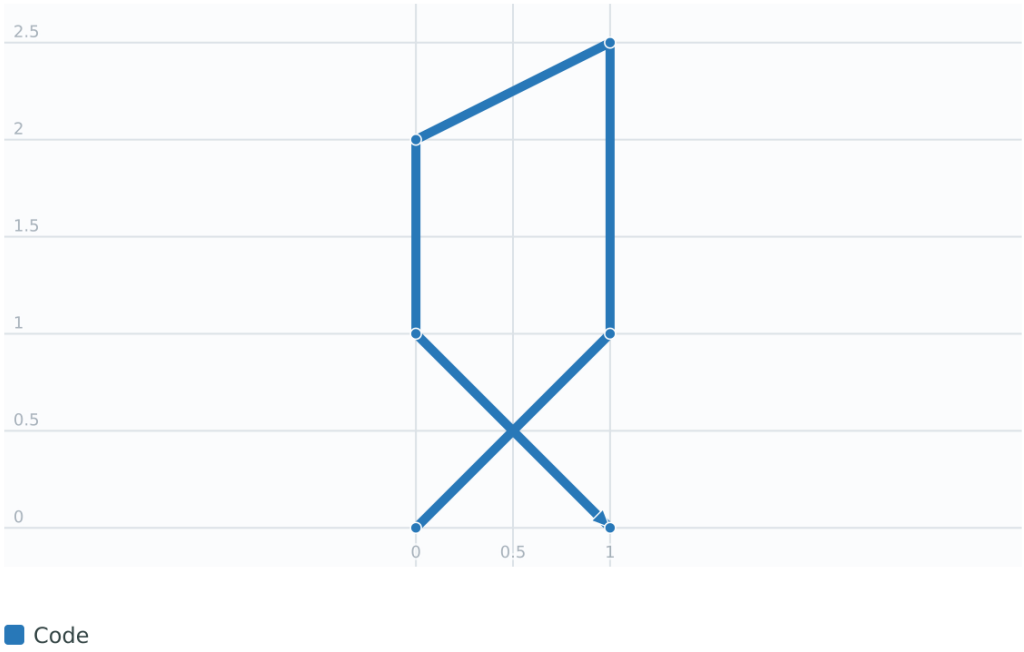
Code

```
SELECT ST_IsSimple('LINESTRING(1 1,2 2,2 3.5,1 3,1 2,2 1)');
```

Output

```
f
```

Figure



Siehe auch

ST_IsValid

7.4.26 ST_IsSimpleDetail

ST_IsSimpleDetail — Report locations of self-intersection or self-tangency, if any.

Synopsis

simple_detail **ST_IsSimpleDetail**(geometry geom, integer findAllLocations = 0, integer boundaryNodeRule = 1);

Beschreibung

Returns a `simple_detail` row, containing a boolean (`simple`) stating if a geometry is simple and a geometry (`location`) pointing out where it is non-simple.

The optional `findAllLocations` parameter can be set to 1 to report all non-simple locations, rather than just one.

The optional `boundaryNodeRule` values are:

- `1`: **OGC-Mod2** - Linienendpunkte befinden sich in der Boundary, wenn sie eine ungerade Anzahl von Malen vorkommen. Dies ist die vom OGC SFS-Standard definierte Regel und ist die Vorgabe für `ST_Relate`.
- `2`: **Endpunkt** - alle Endpunkte befinden sich in der Boundary.
- `3`: **MultivalentEndpoint** - Endpunkte befinden sich in der Boundary, wenn sie mehr als einmal vorkommen. Mit anderen Worten, die Grenze sind alle "verbundenen" oder "inneren" Endpunkte (aber nicht die "unverbundenen/äußeren").
- `4`: **MonovalenterEndpunkt** - Endpunkte befinden sich in der Boundary, wenn sie nur einmal vorkommen. Mit anderen Worten, die Begrenzung sind alle "unverbundenen" oder "äußeren" Endpunkte.

Availability: 3.7.0

Beispiele

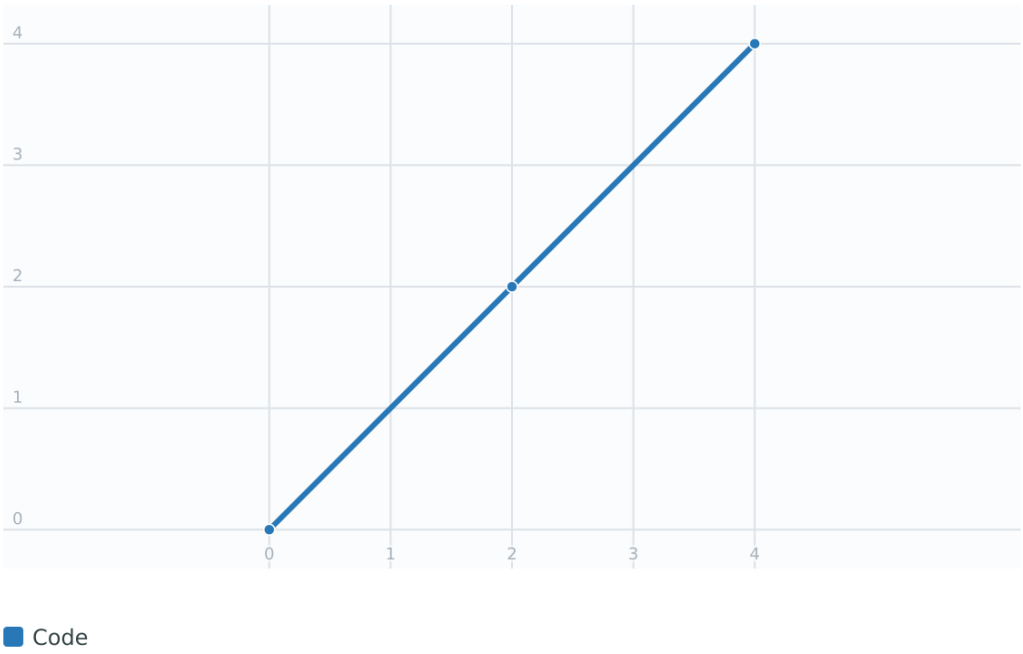
Code

```
SELECT ST_IsSimple('POLYGON((1 2,3 4,5 6,1 2))');
```

Output

```
f
```

Figure



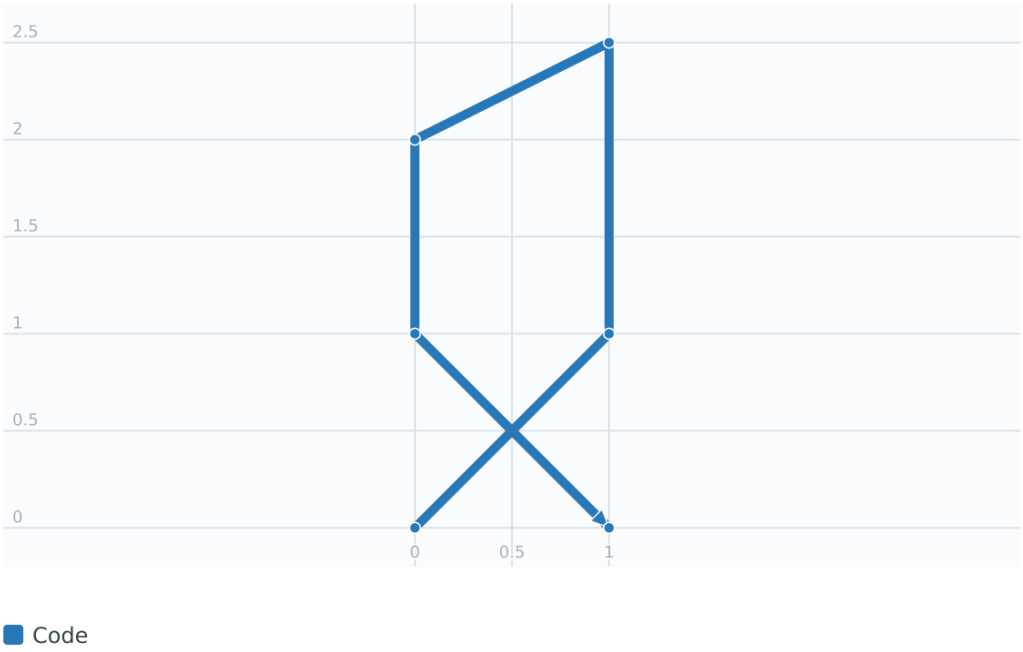
Code

```
SELECT ST_IsSimple('LINESTRING(1 1,2 2,2 3.5,1 3,1 2,2 1)');
```

Output

```
f
```

Figure



Siehe auch

ST_IsSimple

7.4.27 ST_M

ST_M — Gibt die M-Koordinate eines Punktes zurück.

Synopsis

float **ST_M**(geometry a_point);

Beschreibung

Gibt die M-Koordinate des Punktes zurück, oder NULL wenn keine vorhanden ist. Der Einabewert muss ein Punkt sein.



Anmerkung
Dies ist (noch) kein Teil der OGC Spezifikation, wird aber hier aufgeführt um die Liste von Funktionen zum Auslesen von Punktkoordinaten zu vervollständigen.

 Diese Methode implementiert die OGC Simple Features Implementation Specification for SQL 1.1.

 Diese Methode setzt die SQL/MM-Spezifikation um.

 Diese Funktion unterstützt 3d und lässt den Z-Index nicht fallen.

Beispiele

Code

```
SELECT ST_M('POINT(1 2 3 4)');
```

Output

```
4
```

Siehe auch

ST_GeomFromEWKT, ST_X, ST_Y, ST_Z

7.4.28 ST_MemSize

ST_MemSize — Gibt den Geometrietyp des ST_Geometry Wertes zurück.

Synopsis

integer **ST_MemSize**(geometry geomA);

Beschreibung

Gibt den Geometrietyp des ST_Geometry Wertes zurück.

Dies ergänzt die in PostgreSQL eingebauten Datenbankobjektfunktionen pg_column_size, pg_size_pretty, pg_relation_size, pg_total_relation_size.



Anmerkung
pg_relation_size, das die Bytegröße einer Tabelle angibt, kann eine geringere Bytegröße als ST_MemSize zurückgeben. Der Grund dafür ist, dass pg_relation_size den Beitrag von TOAST-Tabellen nicht berücksichtigt und große Geometrien in TOAST-Tabellen gespeichert werden.
pg_total_relation_size - schließt die Tabelle, die TOAST-Tabellen und di Indizes mit ein.
pg_column_size gibt zurück, wie viel Platz eine Geometrie in einer Spalte unter Berücksichtigung der Komprimierung einnehmen würde, kann also niedriger sein als ST_MemSize

 Diese Funktion unterstützt 3d und lässt den Z-Index nicht fallen.

 Diese Methode unterstützt kreisförmige Strings und Kurven.

 Diese Funktion unterstützt polyedrische Flächen.

 Diese Funktion unterstützt Dreiecke und dreieckige unregelmäßige Netzoberflächen (TIN).

Geändert: 2.2.0 Name geändert in ST_MemSize, um der Namenskonvention zu folgen.

Beispiele

Compare the memory used by all geometries in a Massachusetts town dataset with the subset for Boston.

Code

```
SELECT pg_size_pretty(sum(ST_MemSize(geom))) AS totgeomsum,
       pg_size_pretty(sum(CASE WHEN town = 'BOSTON' THEN ST_MemSize(geom) ELSE 0 END)) AS ↩
    bossum,
       cast(
           sum(CASE WHEN town = 'BOSTON' THEN ST_MemSize(geom) ELSE 0 END) * 100.0
           / sum(ST_MemSize(geom)) AS numeric(10, 2)
       ) AS perbos
FROM towns;
```

Output

```
totgeomsum | bossum | perbos
-----------+--------+--------
 1522 kB    | 30 kB  |   1.99
(1 row)
```

Return the memory size of a CircularString in bytes.

Code

```
SELECT ST_MemSize('CIRCULARSTRING(220268 150415,220227 150505,220227 150406)'::geometry);
```

Output

```
st_memsize
------------
         80
(1 row)
```

Calculate the percentage of a table's total size used by geometry data.

Code

```
SELECT pg_total_relation_size('public.neighborhoods') AS fulltable_size,
       sum(ST_MemSize(geom)) AS geomsize,
       sum(ST_MemSize(geom)) * 100.0
           / pg_total_relation_size('public.neighborhoods') AS pergeom
FROM neighborhoods;
```

Output

```
fulltable_size | geomsize |         pergeom
---------------+----------+-------------------------
         262144 |    96238 | 36.71188354492187500000
(1 row)
```

7.4.29 ST_NDims

ST_NDims — Gibt die Dimension der Koordinaten für den Wert von ST_Geometry zurück.

Synopsis

integer **ST_NDims**(geometry g1);

Beschreibung

Gibt die Koordinatendimension der Geometrie zurück. PostGIS unterstützt 2- (x,y), 3- (x,y,z) oder 2D mit Kilometrierung - x,y,m, und 4- dimensionalen Raum - 3D mit Kilometrierung x,y,z,m .

Diese Funktion unterstützt 3d und lässt den Z-Index nicht fallen.

Beispiele

Code

```
SELECT ST_NDims('POINT(1 1)') As d2point,
ST_NDims('POINT(1 1 2)') As d3point,
ST_NDims('POINTM(1 1 0.5)') As d2pointm;
```

Output

```
d2point | d3point | d2pointm
---------+---------+----------
       2 |       3 |        3
```

Siehe auch

ST_CoordDim, ST_Dimension, ST_GeomFromEWKT

7.4.30 ST_NPoints

ST_NPoints — Gibt die Anzahl der Punkte (Knoten) einer Geometrie zurück.

Synopsis

integer **ST_NPoints**(geometry g1);

Beschreibung

Gibt die Anzahl der Punkte einer Geometrie zurück. Funktioniert für alle Geometrien.

Erweiterung: Mit 2.0.0 wurde die Unterstützung für polyedrische Oberflächen eingeführt.



Anmerkung
Vor 1.3.4 ist diese Funktion abgestürzt, wenn die Geometrien CURVES enthalten. Dies wurde mit 1.3.4+ behoben

Diese Funktion unterstützt 3d und lässt den Z-Index nicht fallen.

Diese Methode unterstützt kreisförmige Strings und Kurven.

Diese Funktion unterstützt polyedrische Flächen.

Beispiele

Count the points in a 2D LineString.

Code

```
SELECT ST_NPoints('LINESTRING(77.29 29.07,77.42 29.26,77.27 29.31,77.29 29.07)');
```

Output

```
st_npoints
------------
           4
(1 row)
```

Count the points in a 3D LineString.

Code

```
SELECT ST_NPoints('LINESTRING(77.29 29.07 1,77.42 29.26 0,77.27 29.31 -1,77.29 29.07 3)');
```

Output

```
st_npoints
------------
           4
(1 row)
```

Siehe auch

ST_NumPoints

7.4.31 ST_NRings

ST_NRings — Gibt die Anzahl der inneren Ringe einer Polygongeometrie aus.

Synopsis

integer **ST_NRings**(geometry geomA);

Beschreibung

Wenn es sich bei der Geometrie um ein Polygon oder um ein MultiPolygon handelt, wird die Anzahl der Ringe zurückgegeben. Anders als NumInteriorRings werden auch die äußeren Ringe gezählt.

 Diese Funktion unterstützt 3d und lässt den Z-Index nicht fallen.

 Diese Methode unterstützt kreisförmige Strings und Kurven.

Beispiele

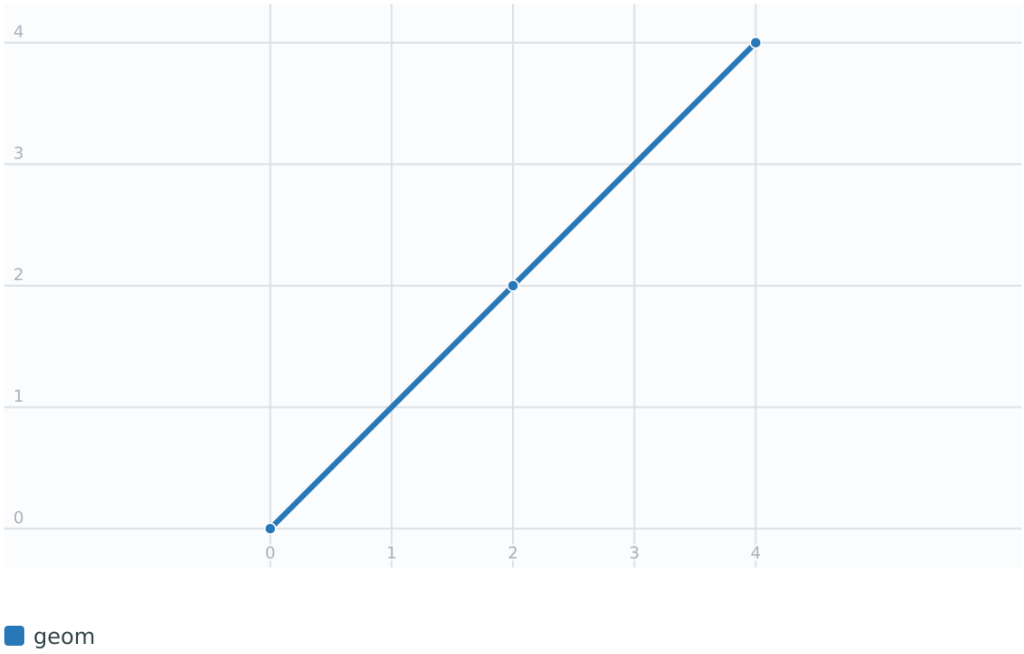
Code

```
SELECT ST_NRings(geom) As Nrings, ST_NumInteriorRings(geom) As ninterrings
FROM (SELECT 'POLYGON((1 2,3 4,5 6,1 2))'::geometry As geom) As foo;
```

Output

```
nrings | ninterrings
--------+-------------
      1 |           0
(1 row)
```

Figure



Siehe auch

ST_NumInteriorRings

7.4.32 ST_NumGeometries

ST_NumGeometries — Gibt die Anzahl der Punkte einer Geometrie zurück. Funktioniert für alle Geometrien.

Synopsis

integer **ST_NumGeometries**(geometry geom);

Beschreibung

Gibt die Anzahl der Elemente in einer Geometriesammlung (GEOMETRYCOLLECTION oder MULTI*) zurück. Für nicht leere atomare Geometrien wird 1 zurückgegeben. Für leere Geometrien wird 0 zurückgegeben.

Erweiterung: Mit 2.0.0 wurde die Unterstützung für polyedrische Oberflächen, Dreiecke und TIN eingeführt.

Änderung: 2.0.0 Bei früheren Versionen wurde NULL zurückgegeben, wenn die Geometrie nicht vom Typ GEOMETRYCOLLECTION/MULTI war. 2.0.0+ gibt nun 1 für Einzelgeometrien, wie POLYGON, LINESTRING, POINT zurück.

Changed: 3.6.0 TIN and PolyhedralSurface are treated as unitary geometries and return 1; use ST_NumPatches to count faces.

Diese Methode setzt die SQL/MM-Spezifikation um. SQL-MM 3: 9.1.4

Diese Funktion unterstützt 3d und lässt den Z-Index nicht fallen.

Diese Funktion unterstützt polyedrische Flächen.

Diese Funktion unterstützt Dreiecke und dreieckige unregelmäßige Netzoberflächen (TIN).

Beispiele

This example returns 1; prior versions would have returned NULL.

Code

```
SELECT ST_NumGeometries('LINESTRING(77.29 29.07,77.42 29.26,77.27 29.31,77.29 29.07)');
```

Output

```
st_numgeometries
------------------
                 1
(1 row)
```

A MultiPoint counts as one geometry within a GeometryCollection.

Code

```
SELECT ST_NumGeometries('GEOMETRYCOLLECTION(MULTIPOINT((-2 3),(-2 2)),
LINESTRING(5 5,10 10),
POLYGON((-7 4.2,-7.1 5,-7.1 4.3,-7 4.2)))');
```

Output

```
st_numgeometries
------------------
                 3
(1 row)
```

Siehe auch

ST_GeometryN, ST_Multi

7.4.33 ST_NumInteriorRings

ST_NumInteriorRings — Gibt die Anzahl der inneren Ringe einer Polygongeometrie aus.

Synopsis

integer **ST_NumInteriorRings**(geometry a_polygon);

Beschreibung

Gibt die Anzahl der inneren Ringe einer Polygongeometrie aus. Gibt NULL zurück, wenn die Geometrie kein Polygon ist.

Diese Methode setzt die SQL/MM-Spezifikation um. SQL-MM 3: 8.2.5

Änderung: 2.0.0 - In füheren Versionen war ein MULTIPOLYGON als Eingabe erlaubt, wobei die Anzahl der inneren Ringe des ersten Polygons ausgegeben wurde.

Beispiele

Count the interior rings of a Polygon.

Code

```
SELECT gid, field1, field2, ST_NumInteriorRings(geom) AS numholes
FROM sometable;
```

For a MultiPolygon, dump its component Polygons and sum their interior ring counts.

Code

```
SELECT gid, field1, field2, SUM(ST_NumInteriorRings(geom)) AS numholes
FROM (
    SELECT gid, field1, field2, (ST_Dump(geom)).geom AS geom
    FROM sometable
) AS foo
GROUP BY gid, field1, field2;
```

Siehe auch

ST_NumInteriorRing, ST_PointN

7.4.34 ST_NumInteriorRing

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

Synopsis

integer **ST_NumInteriorRing**(geometry a_polygon);

Beschreibung

This function is an alias for ST_NumInteriorRings.

Verfügbarkeit: 1.2.2

Siehe auch

ST_NumInteriorRings, ST_PointN

7.4.35 ST_NumPatches

ST_NumPatches — Returns the number of faces in a PolyhedralSurface or TIN.

Synopsis

integer **ST_NumPatches**(geometry g1);

Beschreibung

Returns the number of faces in a PolyhedralSurface or TIN. Returns NULL for other geometries. Unlike ST_NumGeometries, this function counts individual faces.

Verfügbarkeit: 2.0.0

Diese Funktion unterstützt 3d und lässt den Z-Index nicht fallen.

Diese Methode implementiert die OGC Simple Features Implementation Specification for SQL 1.1.

Diese Methode setzt die SQL/MM-Spezifikation um. SQL-MM ISO/IEC 13249-3: 8.5

Diese Funktion unterstützt polyedrische Flächen.

Beispiele

Code

```
SELECT ST_NumPatches('POLYHEDRALSURFACE( ((0 0 0,0 0 1,0 1 1,0 1 0,0 0 0)),
((0 0 0,0 1 0,1 1 0,1 0 0,0 0 0)),((0 0 0,1 0 0,1 0 1,0 0 1,0 0 0)),
((1 1 0,1 1 1,1 0 1,1 0 0,1 1 0)),
((0 1 0,0 1 1,1 1 1,1 1 0,0 1 0)),((0 0 1,1 0 1,1 1 1,0 1 1,0 0 1)) )');
```

Output

```
st_numpatches
---------------
             6
(1 row)
```

Siehe auch

ST_GeomFromEWKT, ST_NumGeometries, ST_PatchN

7.4.36 ST_NumPoints

ST_NumPoints — Gibt die Anzahl der Stützpunkte eines ST_LineString oder eines ST_CircularString zurück.

Synopsis

integer **ST_NumPoints**(geometry g1);

Beschreibung

Gibt die Anzahl der Stützpunkte eines ST_LineString oder eines ST_CircularString zurück. Vor 1.4 funktionierte dies nur mit ST_LineString, wie von der Spezifikation festgelegt. Ab 1.4 aufwärts handelt es sich um einen Alias für ST_NPoints, das die Anzahl der Knoten nicht nur für Linienzüge ausgibt. Erwägen Sie stattdessen die Verwendung von ST_NPoints, das vielseitig ist und mit vielen Geometrietypen funktioniert.

Diese Methode implementiert die OGC Simple Features Implementation Specification for SQL 1.1.

Diese Methode setzt die SQL/MM-Spezifikation um. SQL-MM 3: 7.2.4

Beispiele

Code

```
SELECT ST_NumPoints('LINESTRING(77.29 29.07,77.42 29.26,77.27 29.31,77.29 29.07)');
```

Output

```
st_numpoints
--------------
           4
(1 row)
```

Siehe auch

ST_NPoints

7.4.37 ST_PatchN

ST_PatchN — Returns the Nth face of a PolyhedralSurface or TIN.

Synopsis

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

Beschreibung

Returns the 1-based Nth face of a PolyhedralSurface or TIN. Returns NULL for other geometries or an out-of-range index. Use this function to access faces, because ST_GeometryN treats PolyhedralSurface and TIN as unitary geometries.



Anmerkung
Der Index ist auf 1 basiert.



Anmerkung
Falls Sie alle Geometrien einer Geometrie entnehmen wollen, so ist ST_Dump wesentlich leistungsfähiger.

Verfügbarkeit: 2.0.0

Diese Methode setzt die SQL/MM-Spezifikation um. SQL-MM ISO/IEC 13249-3: 8.5

Diese Funktion unterstützt 3d und lässt den Z-Index nicht fallen.

Diese Funktion unterstützt polyedrische Flächen.

Diese Funktion unterstützt Dreiecke und dreieckige unregelmäßige Netzoberflächen (TIN).

Beispiele

Extract the 2nd face of the polyhedral surface.

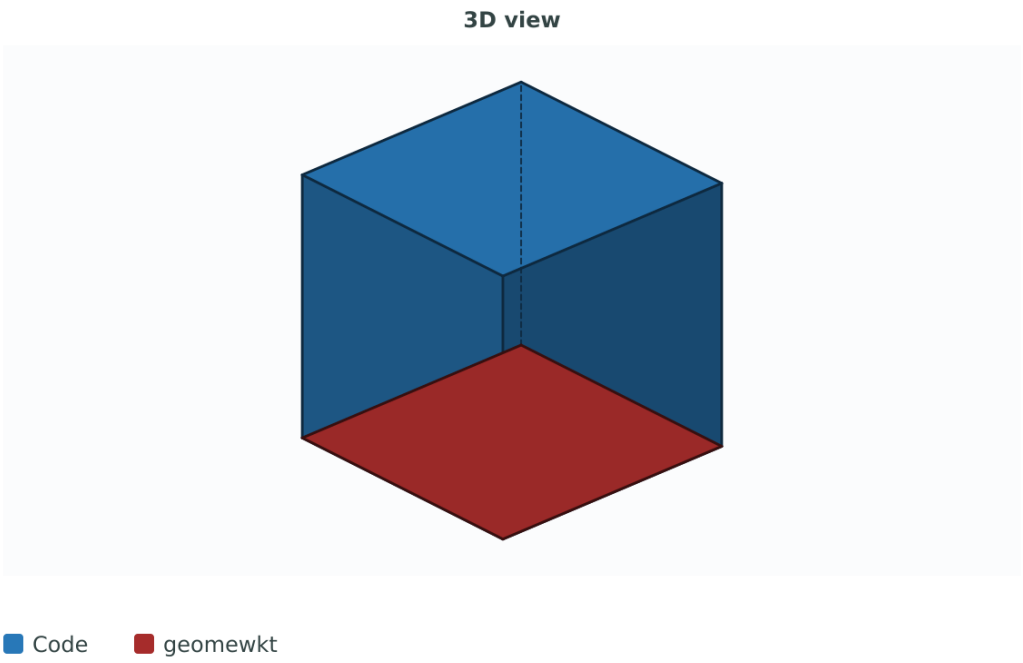
Code

```
SELECT ST_PatchN(geom, 2) As geomewkt
FROM (
VALUES ('POLYHEDRALSURFACE( ((0 0 0,0 0 1,0 1 1,0 1 0,0 0 0)),
    ((0 0 0,0 1 0,1 1 0,1 0 0,0 0 0)),((0 0 0,1 0 0,1 0 1,0 0 1,0 0 0)),
    ((1 1 0,1 1 1,1 0 1,1 0 0,1 1 0)),
    ((0 1 0,0 1 1,1 1 1,1 1 0,0 1 0)),((0 0 1,1 0 1,1 1 1,0 1 1,0 0 1)) )'::geometry) ) As ↩
    foo(geom);
```

Output

```
POLYGON((0 0 0,0 1 0,1 1 0,1 0 0,0 0 0))
```

Figure



Siehe auch

ST_AsEWKT, ST_GeomFromEWKT, ST_Dump, ST_GeometryN, ST_NumGeometries, ST_NumPatches

7.4.38 ST_PointN

ST_PointN — Gibt die Anzahl der Stützpunkte eines ST_LineString oder eines ST_CircularString zurück.

Synopsis

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

Beschreibung

Gibt den n-ten Punkt des ersten LineString's oder des kreisförmigen LineStrings's einer Geometrie zurück. Negative Werte werden rückwärts, vom Ende des LineString's her gezählt, sodass -1 der Endpunkt ist. Gibt NULL aus, wenn die Geometrie keinen LineString enthält.



Anmerkung
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.



Anmerkung
Falls Sie den n-ten Punkt eines jeden LineString's in einem MultiLinestring wollen, nutzen Sie diese Funktion gemeinsam mit ST_Dump.


Diese Methode implementiert die OGC Simple Features Implementation Specification for SQL 1.1.


Diese Methode setzt die SQL/MM-Spezifikation um. SQL-MM 3: 7.2.5, 7.3.5


Diese Funktion unterstützt 3d und lässt den Z-Index nicht fallen.


Diese Methode unterstützt kreisförmige Strings und Kurven.

Änderung: 2.0.0 arbeitet nicht mehr mit MultiLinestring's, die nur eine einzelne Geometrie enthalten. In früheren Versionen von PostGIS gab die Funktion bei einem, aus einer einzelnen Linie bestehender MultiLinestring, den Anfangspunkt zurück. Ab 2.0.0 wird, so wie bei jedem anderen MultiLinestring auch, NULL zurückgegeben.

Änderung: 2.3.0 : negatives Indizieren verfügbar (-1 entspricht dem Endpunkt)

Beispiele

This example extracts all POINT geometries from a LineString.

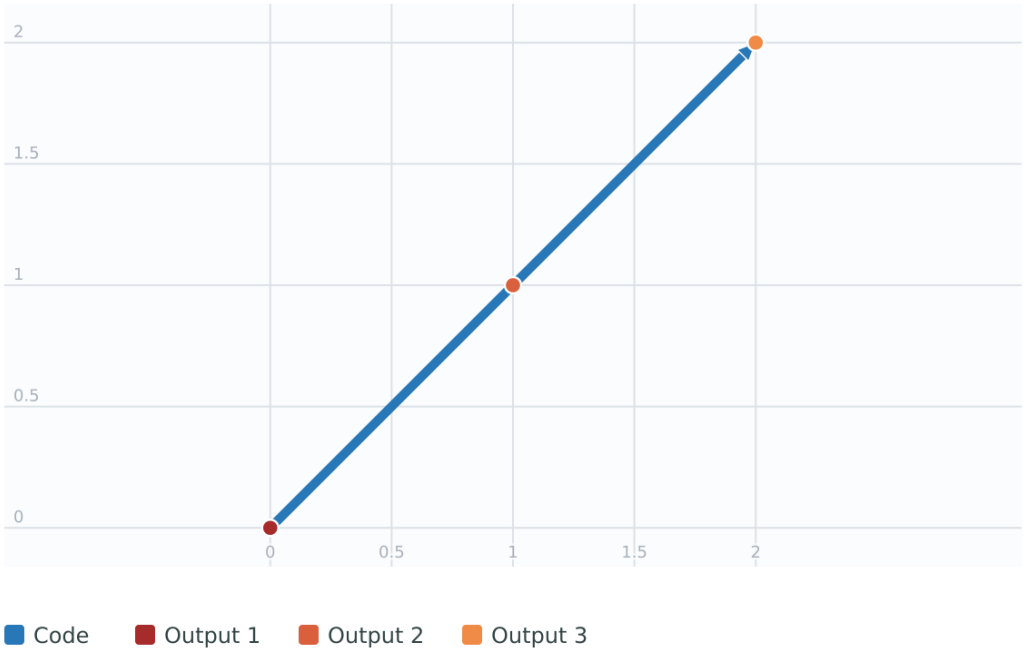
Code

```
SELECT ST_PointN(
      column1,
      generate_series(1, ST_NPoints(column1))
   )
FROM ( VALUES ('LINESTRING(0 0,1 1,2 2)'::geometry) ) AS foo;
```

Output

```
POINT(0 0)
 POINT(1 1)
 POINT(2 2)
(3 rows)
```

Figure



This example uses a CircularString.

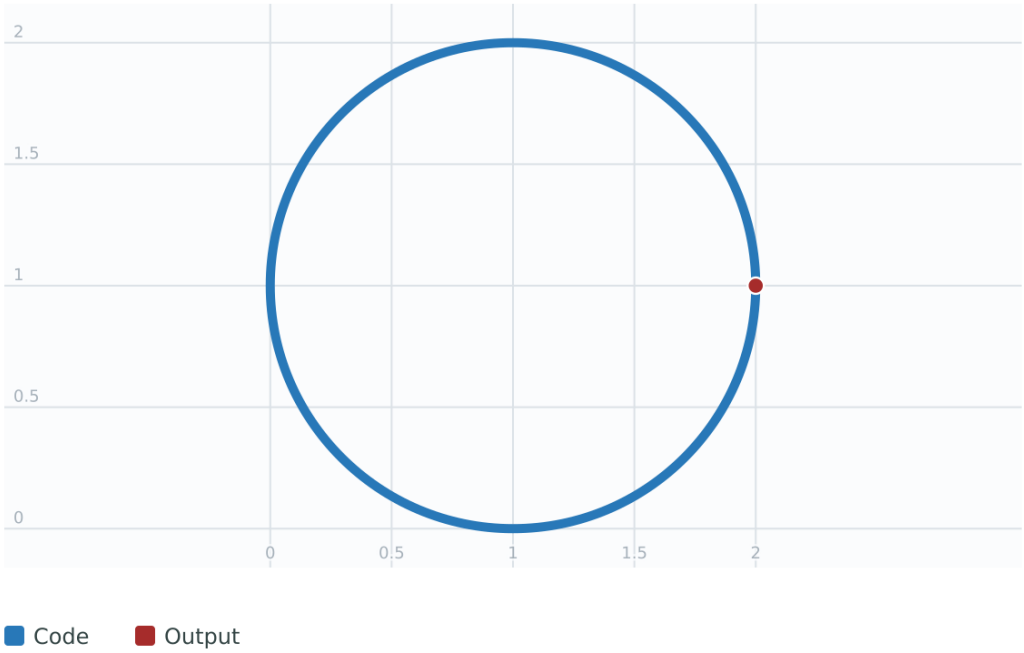
Code

```
SELECT ST_PointN('CIRCULARSTRING(1 2,3 2,1 2)', 2);
```

Output

```
POINT(3 2)
```

Figure



The index is 1-based. A negative index counts backward from the end of the LineString.

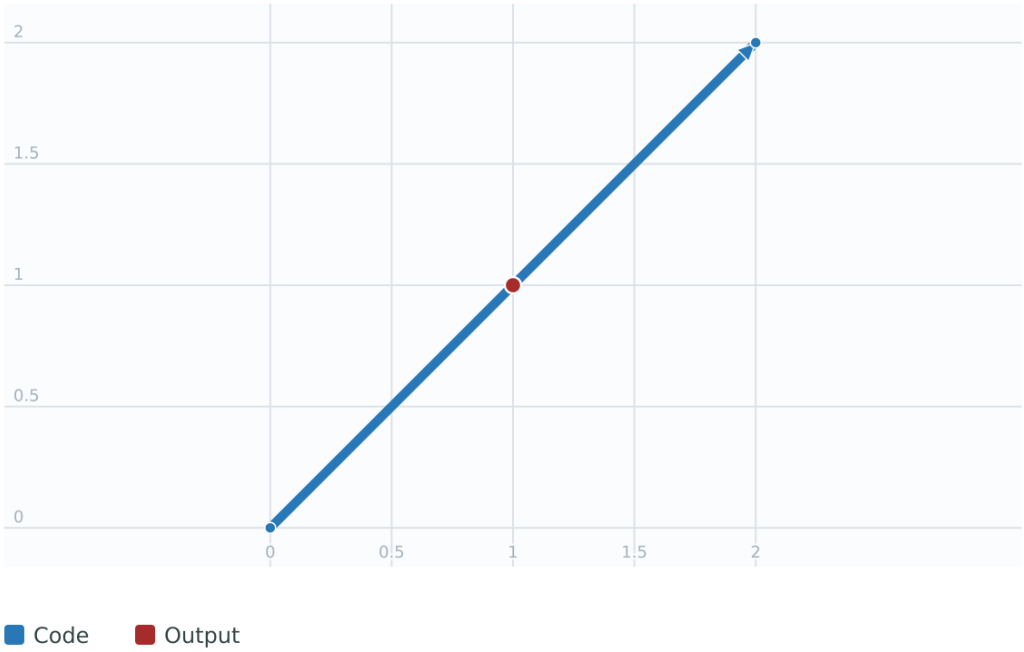
Code

```
SELECT f
FROM ST_GeomFromText('LINESTRING(0 0 0,1 1 1,2 2 2)') AS g
  CROSS JOIN ST_PointN(g, -2) AS f;
```

Output

```
POINT Z (1 1 1)
```

Figure



Siehe auch

ST_NPoints

7.4.39 ST_Points

ST_Points — Gibt einen MultiPoint zurück, welcher alle Koordinaten einer Geometrie enthält.

Synopsis

geometry **ST_Points**(geometry geom);

Beschreibung

Gibt einen MultiPoint zurück, der alle Koordinaten einer Geometrie enthält. Doppelte Punkte werden beibehalten, einschließlich der Start- und Endpunkte von Ringgeometrien. (Falls gewünscht, können doppelte Punkte durch den Aufruf von ST_RemoveRepeatedPoi für das Ergebnis entfernt werden).

Um Informationen über die Position der einzelnen Koordinaten in der übergeordneten Geometrie zu erhalten, verwenden Sie ST_DumpPoints.

M- und Z-Koordinaten werden beibehalten, falls vorhanden.

Diese Methode unterstützt kreisförmige Strings und Kurven.

Diese Funktion unterstützt 3d und lässt den Z-Index nicht fallen.

Verfügbarkeit: 2.3.0

Beispiele

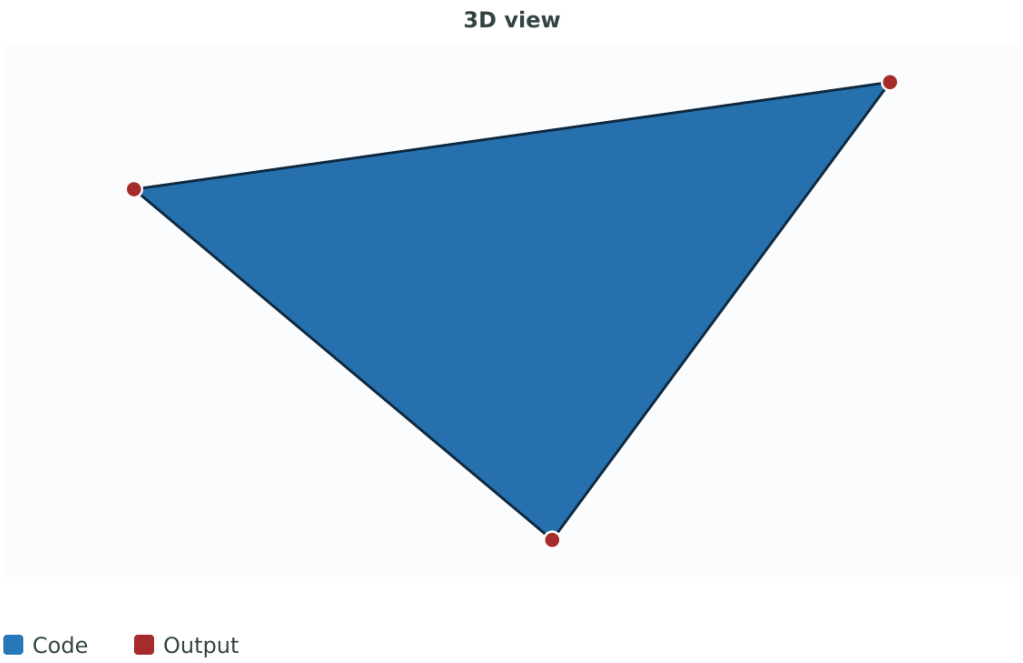
Code

```
SELECT ST_Points('POLYGON Z ((30 10 4,10 30 5,40 40 6,30 10 4))');
```

Output

```
MULTIPOINT Z ((30 10 4),(10 30 5),(40 40 6),(30 10 4))
```

Figure



Siehe auch

ST_RemoveRepeatedPoints, ST_PointN

7.4.40 ST_StartPoint

ST_StartPoint — Returns the first point of a LineString, CircularLineString, or NURBSCurve.

Synopsis

geometry **ST_StartPoint**(geometry geomA);

Beschreibung

Returns the first point of a `LINESTRING`, `CIRCULARLINESTRING`, or `NURBSCURVE` geometry as a `POINT`. For other geometries, returns the first point in coordinate order.

Diese Methode setzt die SQL/MM-Spezifikation um. SQL-MM 3: 7.1.3

Diese Funktion unterstützt 3d und lässt den Z-Index nicht fallen.

Diese Methode unterstützt kreisförmige Strings und Kurven.

Verbessert: 3.2.0 gibt einen Punkt für alle Geometrien zurück. Vorheriges Verhalten gibt NULLs zurück, wenn die Eingabe kein LineString war.

Änderung: 2.0.0 unterstützt die Verarbeitung von MultiLinestring's die nur aus einer einzelnen Geometrie bestehen, nicht mehr. In früheren Versionen von PostGIS gab die Funktion bei einem aus einer einzelnen Linie bestehender MultiLinestring den Anfangspunkt zurück. Ab 2.0.0 gibt sie nur NULL zurück, so wie bei jedem anderen MultiLinestring. Die alte Verhaltensweise war undokumentiert, aber Anwender, die annahmen, dass Sie Ihre Daten als LINESTRING vorliegen haben, könnten in 2.0 dieses zurückgegebene NULL bemerken.

Beispiele

Startpunkt eines LineString

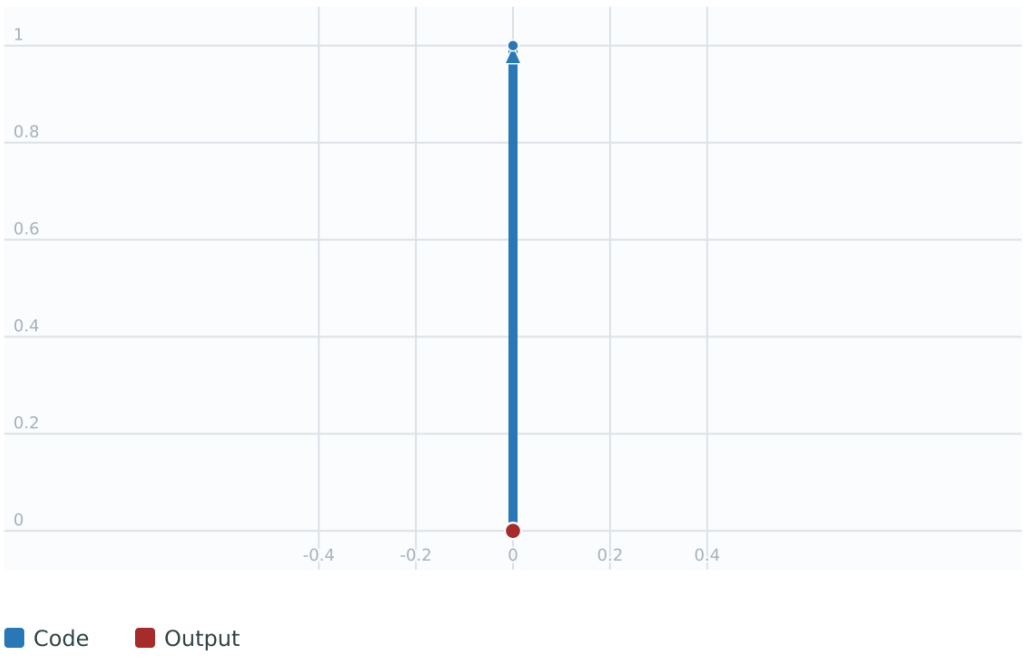
Code

```
SELECT ST_StartPoint('LINESTRING(0 1,0 2)'::geometry);
```

Output

```
POINT(0 1)
```

Figure



Start point of a Point is the Point itself

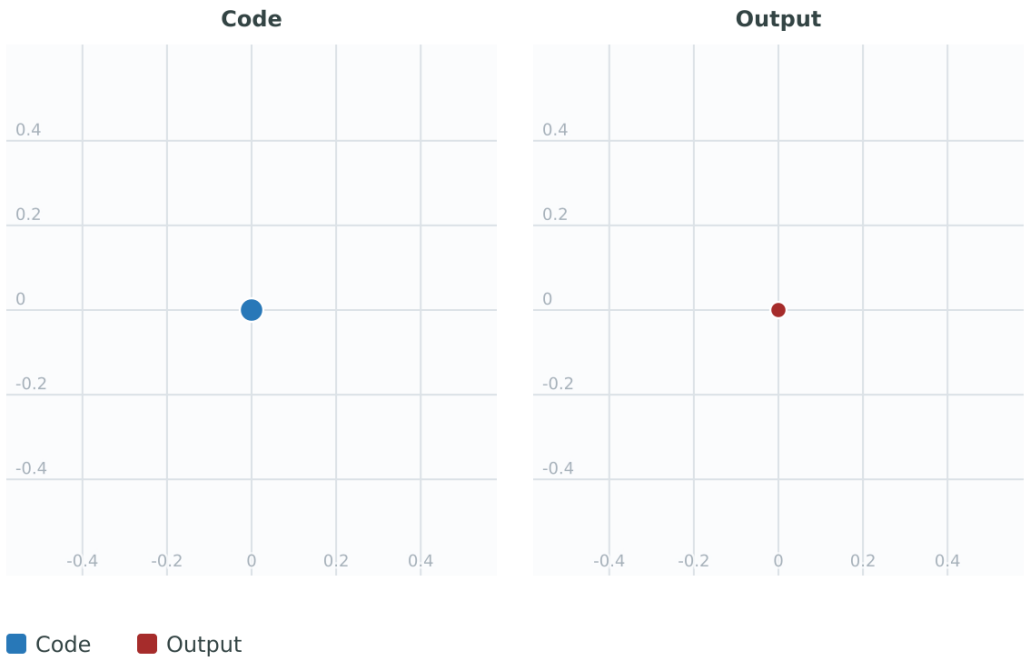
Code

```
SELECT ST_StartPoint('POINT(0 1)'::geometry);
```

Output

```
POINT(0 1)
```

Figure



Startpunkt einer 3D-LinieString

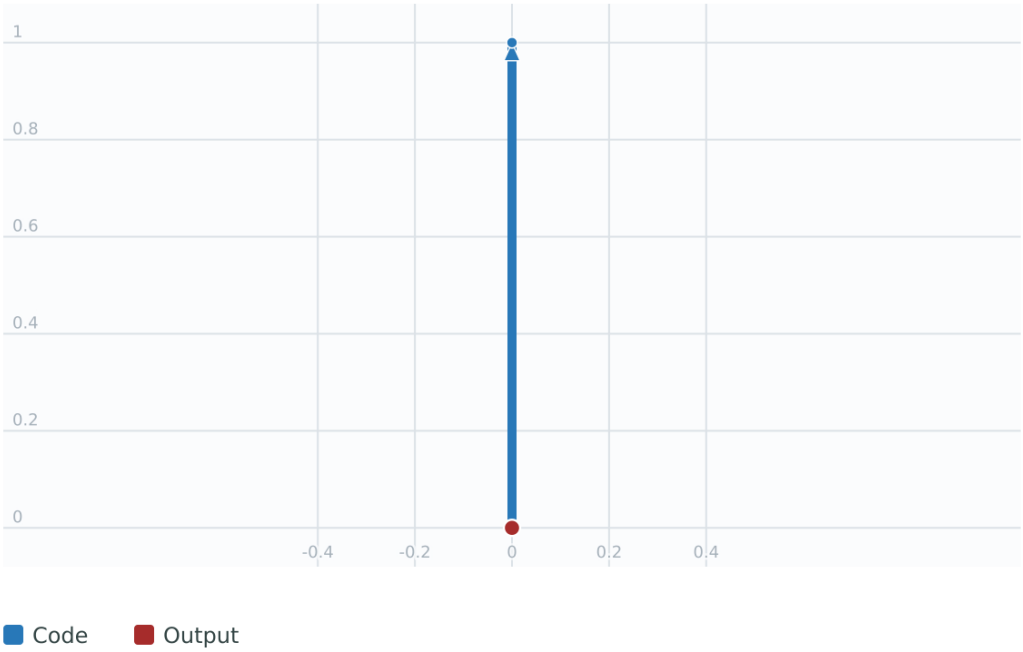
Code

```
SELECT ST_StartPoint('LINESTRING(0 1 1,0 2 2)'::geometry);
```

Output

```
POINT(0 1 1)
```

Figure



Gibt die Anzahl der Stützpunkte eines ST_LineString oder eines ST_CircularString zurück.

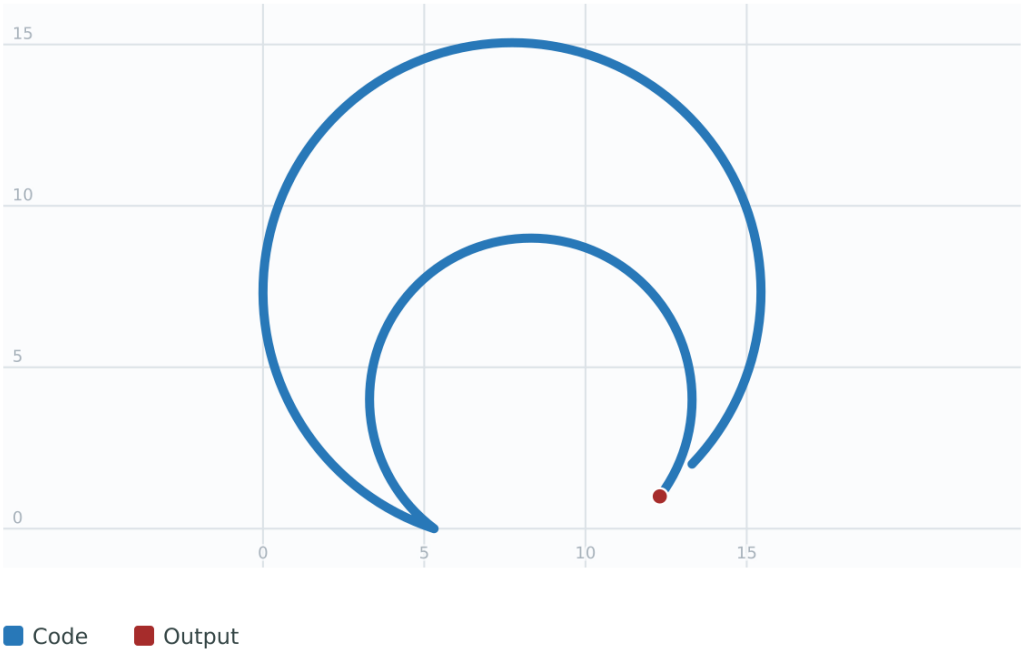
Code

```
SELECT ST_StartPoint('CIRCULARSTRING(5 2,-3 2,-2 1,-4 2,6 3)'::geometry);
```

Output

```
POINT(5 2)
```

Figure



Start point of a NURBSCurve

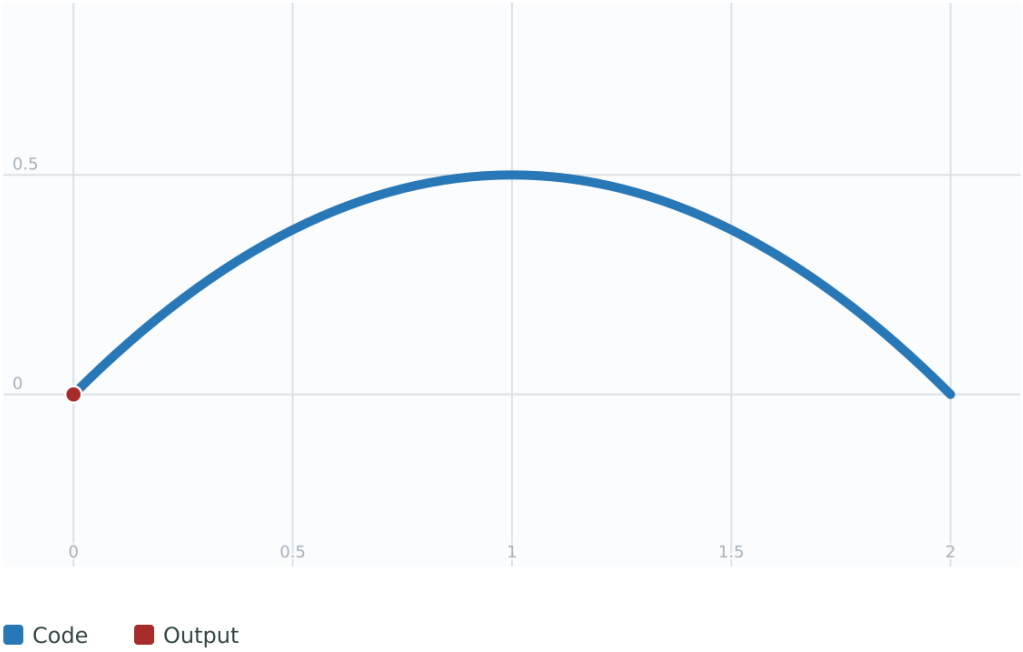
Code

```
SELECT ST_StartPoint('NURBSCURVE(2, (0 0, 1 1, 2 0))'::geometry);
```

Output

```
POINT(0 0)
```

Figure



Build edge node references from the start and end points of a LineString table. The example uses exact endpoint equality, so linework with nearly coincident endpoints should be snapped or otherwise cleaned first.

Code

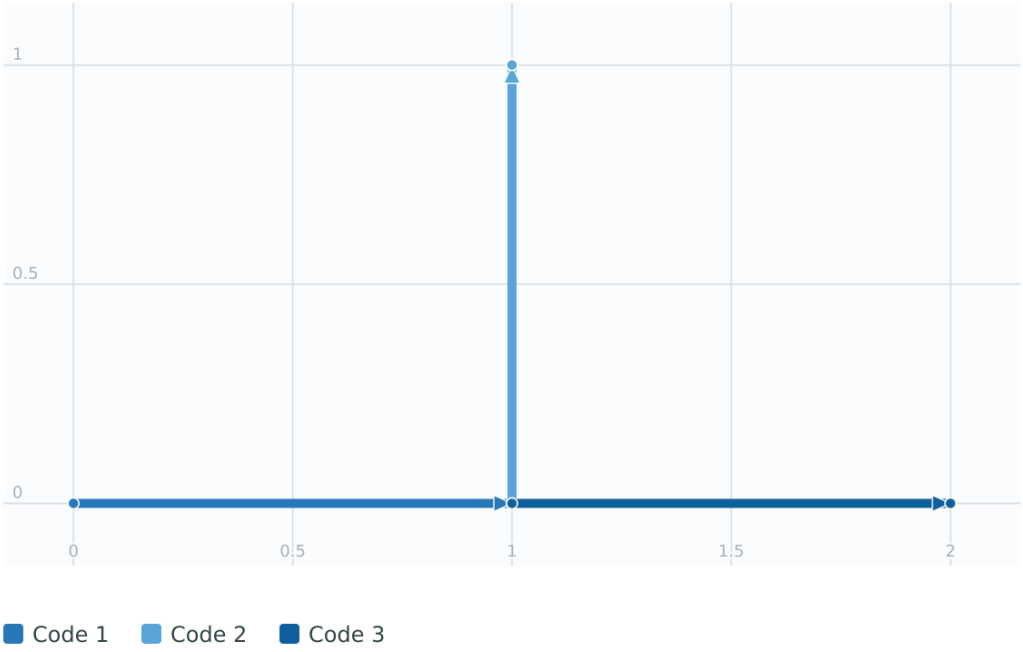
```
WITH roads(road_id, geom) AS (
  VALUES
    (1, 'LINESTRING(0 0, 1 0)'::geometry),
    (2, 'LINESTRING(1 0, 1 1)'::geometry),
    (3, 'LINESTRING(1 0, 2 0)'::geometry)
),
endpoints AS (
  SELECT road_id, 'from' AS end_name, ST_StartPoint(geom) AS geom,
         ST_AsEWKB(ST_StartPoint(geom)) AS endpoint_key
  FROM roads
  UNION ALL
  SELECT road_id, 'to' AS end_name, ST_EndPoint(geom) AS geom,
         ST_AsEWKB(ST_EndPoint(geom)) AS endpoint_key
  FROM roads
),
nodes AS (
  SELECT dense_rank() OVER (ORDER BY ST_X(geom), ST_Y(geom), endpoint_key) AS node_id,
         endpoint_key
  FROM (
    SELECT DISTINCT ON (endpoint_key) endpoint_key, geom
    FROM endpoints
    ORDER BY endpoint_key
  ) AS distinct_endpoints
```

```
)
SELECT e.road_id,
       max(n.node_id) FILTER (WHERE e.end_name = 'from') AS from_node,
       max(n.node_id) FILTER (WHERE e.end_name = 'to') AS to_node
FROM endpoints AS e
JOIN nodes AS n USING (endpoint_key)
GROUP BY e.road_id
ORDER BY e.road_id;
```

Output

```
road_id | from_node | to_node
---------+-----------+---------
     1 |         1 |       2
     2 |         2 |       3
     3 |         2 |       4
```

Figure



Siehe auch

ST_EndPoint, ST_PointN

7.4.41 ST_StartM

ST_StartM — Returns the M coordinate of the first point of a LineString, CircularLineString, or NURBSCurve.

Synopsis

float8 **ST_StartM**(geometry geom);

Beschreibung

Returns the M coordinate of the first point of a LINESTRING, CIRCULARLINESTRING, or NURBSCURVE geometry. Returns NULL if the input geometry does not have an M dimension or is not a LINESTRING, CIRCULARLINESTRING, or NURBSCURVE.

 Diese Methode unterstützt kreisförmige Strings und Kurven.

Beispiele

Start M value of a LineString

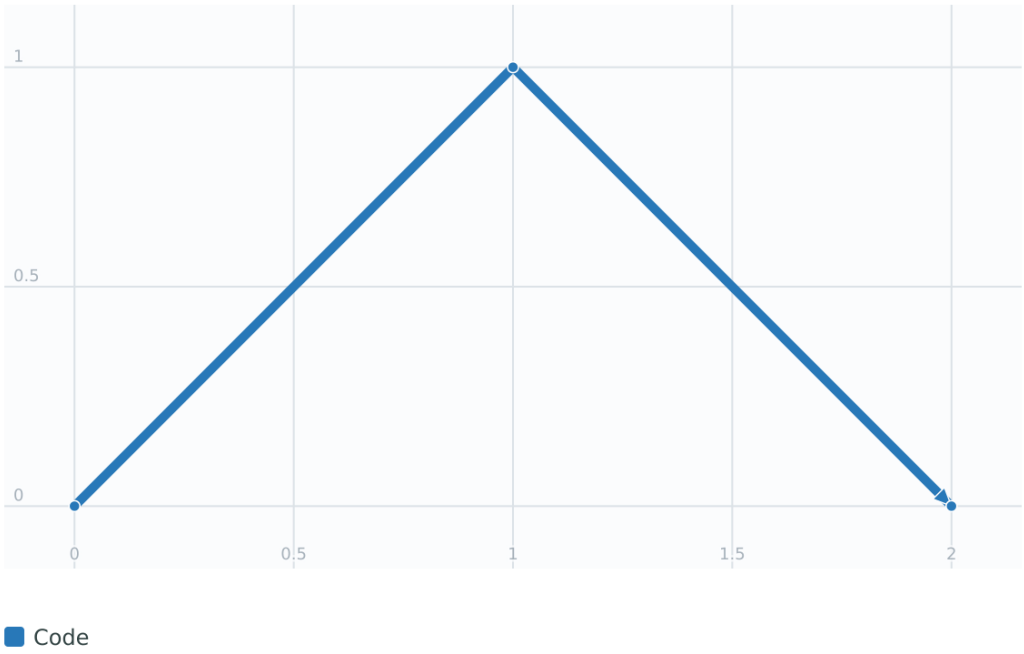
Code

```
SELECT ST_StartM('LINESTRING M(0 0 10,1 1 20,2 0 30)'::geometry);
```

Output

```
10
```

Figure



Start M value of a CircularString

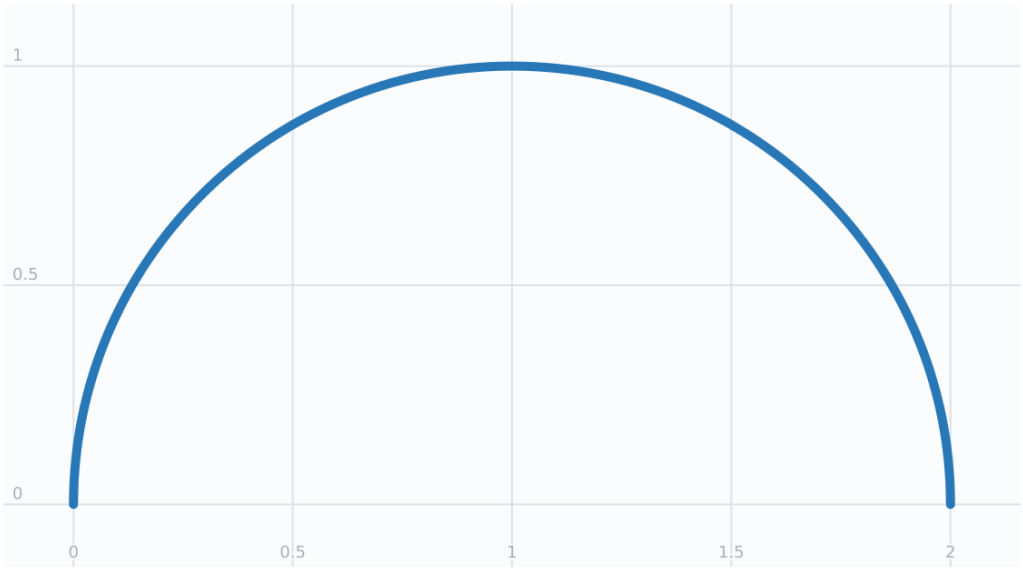
Code

```
SELECT ST_StartM('CIRCULARSTRING M(0 0 5,1 1 10,2 0 15)'::geometry);
```

Output

```
5
```

Figure



Start M value of a NURBSCurve

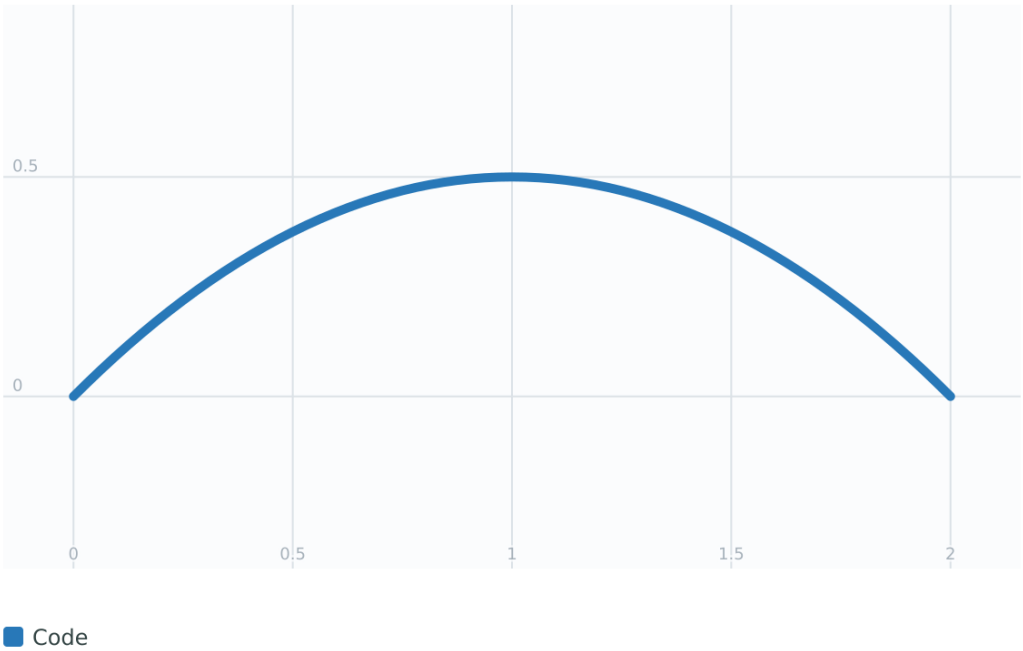
Code

```
SELECT ST_StartM('NURBSCURVE M(2, (0 0 100, 1 1 200, 2 0 300))'::geometry);
```

Output

```
100
```

Figure



Geometry without M dimension returns NULL

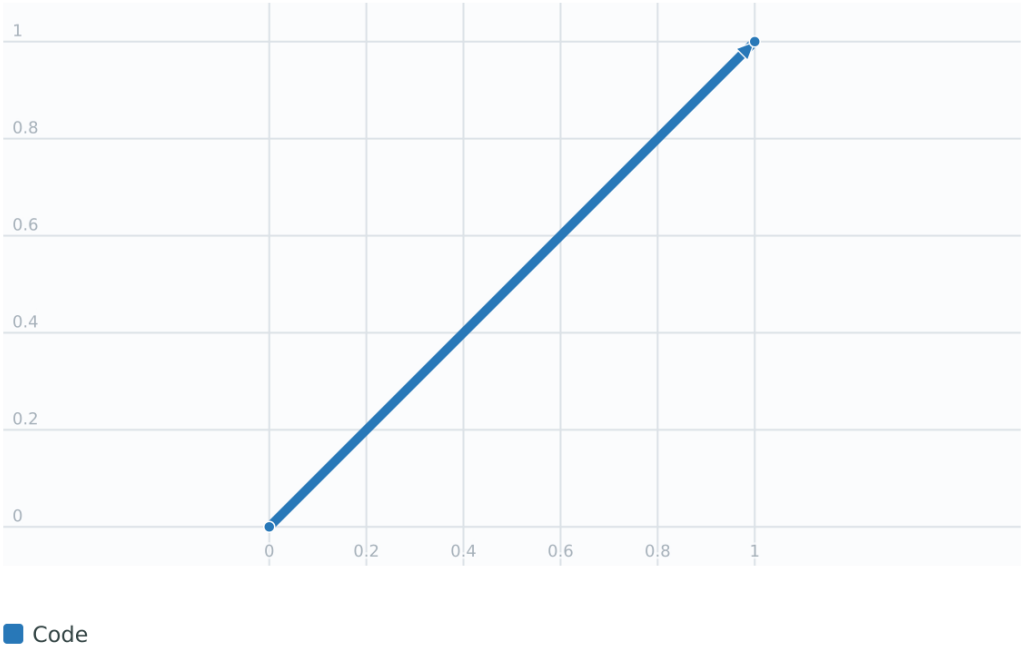
Code

```
SELECT ST_StartM('LINESTRING(0 0,1 1)'::geometry) IS NULL AS is_null;
```

Output

```
t
```

Figure



Siehe auch

ST_EndM, ST_SetStartM, ST_StartPoint

7.4.42 ST_EndM

ST_EndM — Returns the M coordinate of the last point of a LineString, CircularLineString, or NURBSCurve.

Synopsis

float8 **ST_EndM**(geometry geom);

Beschreibung

Returns the M coordinate of the last point of a `LINESTRING`, `CIRCULARLINESTRING`, or `NURBSCURVE` geometry. Returns `NULL` if the input geometry does not have an M dimension or is not a `LINESTRING`, `CIRCULARLINESTRING`, or `NURBSCURVE`.

Diese Methode unterstützt kreisförmige Strings und Kurven.

Beispiele

End M value of a LineString

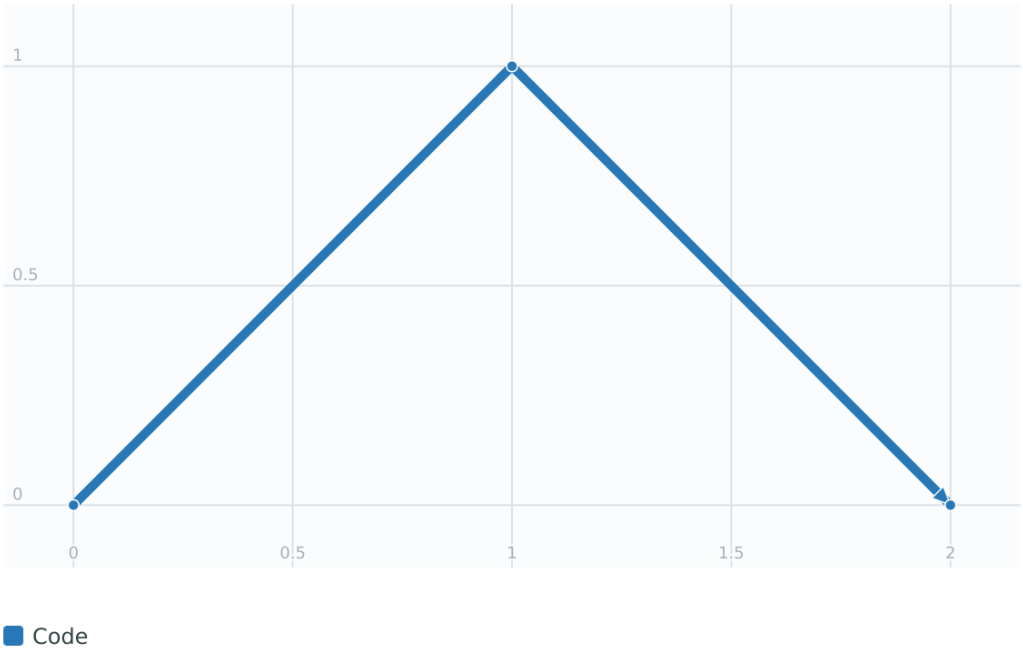
Code

```
SELECT ST_EndM('LINESTRING M(0 0 10,1 1 20,2 0 30)'::geometry);
```

Output

```
30
```

Figure



End M value of a CircularString

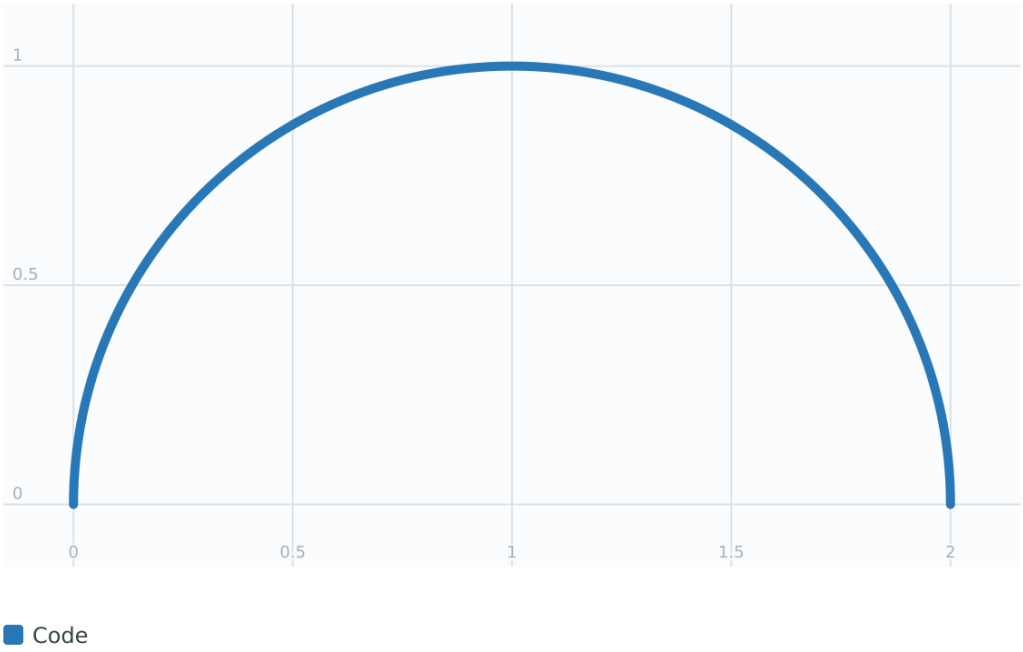
Code

```
SELECT ST_EndM('CIRCULARSTRING M(0 0 5,1 1 10,2 0 15)'::geometry);
```

Output

```
15
```

Figure



End M value of a NURBSCurve

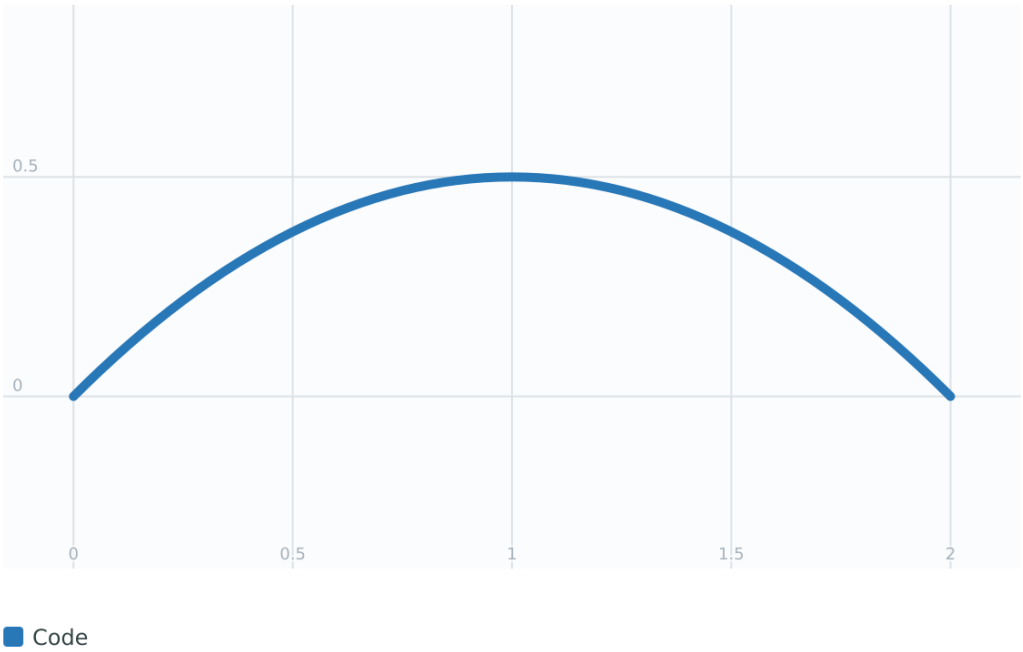
Code

```
SELECT ST_EndM('NURBSCURVE M(2, (0 0 100, 1 1 200, 2 0 300))'::geometry);
```

Output

```
300
```

Figure



Geometry without M dimension returns NULL

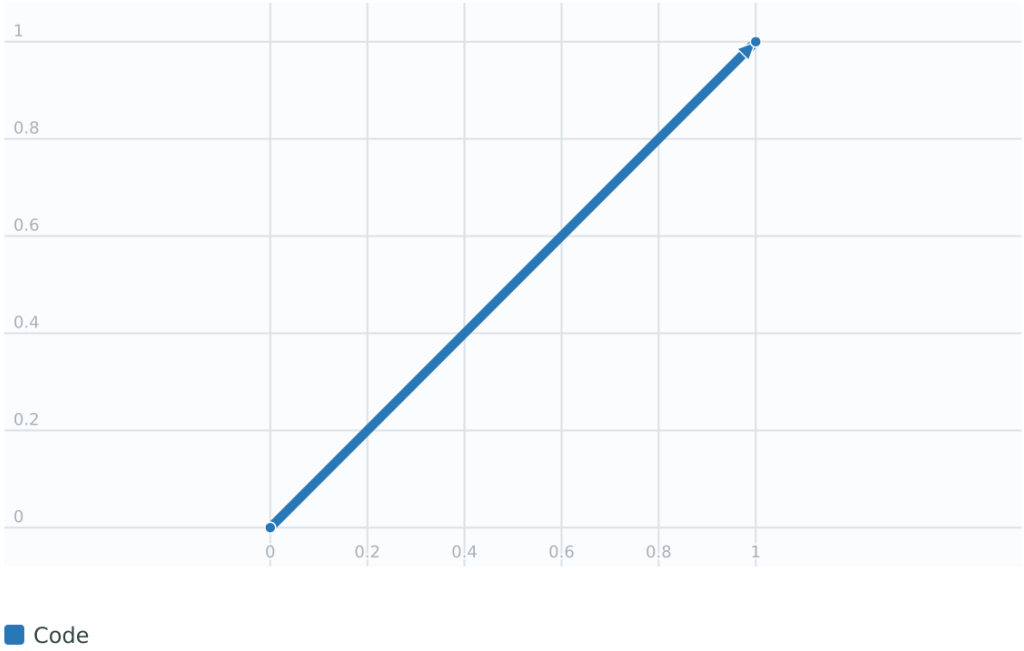
Code

```
SELECT ST_EndM('LINESTRING(0 0,1 1)'::geometry) IS NULL AS is_null;
```

Output

```
t
```

Figure



Siehe auch

ST_StartM, ST_SetEndM, ST_EndPoint

7.4.43 ST_SetStartM

ST_SetStartM — Sets the M coordinate of the first point of a LineString, CircularLineString, or NURBSCurve.

Synopsis

geometry **ST_SetStartM**(geometry geom, float8 m);

Beschreibung

Returns a copy of the input `LINESTRING`, `CIRCULARLINESTRING`, or `NURBSCURVE` geometry with the M coordinate of the first point set to the specified value. If the input geometry does not have an M dimension, it will be added. Returns `NULL` if the input is not a `LINESTRING`, `CIRCULARLINESTRING`, or `NURBSCURVE`.

Diese Methode unterstützt kreisförmige Strings und Kurven.

Beispiele

Set start M value of a LineString

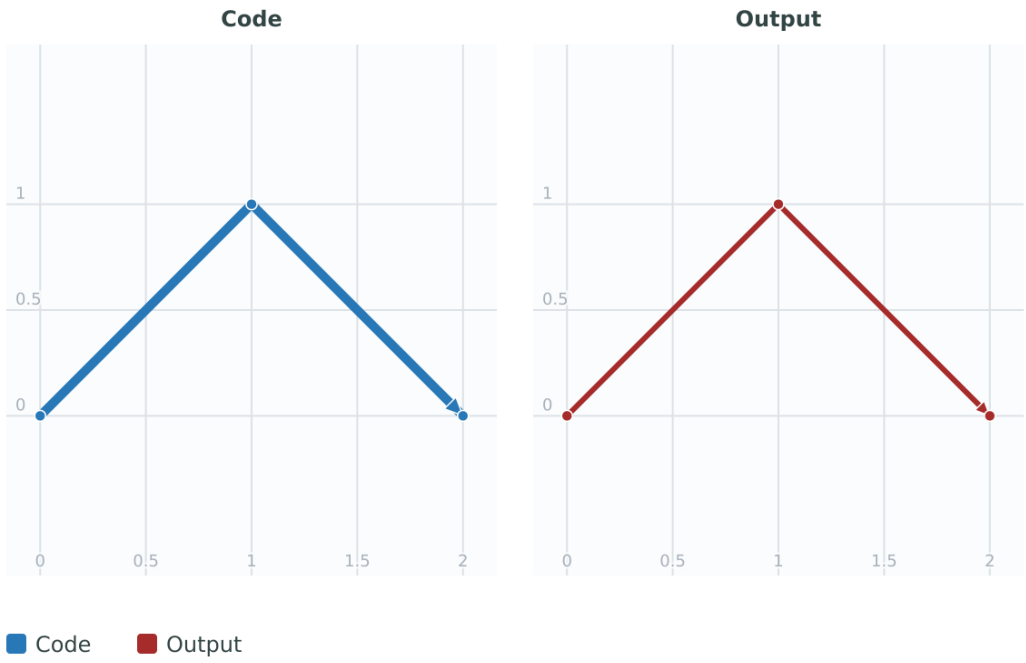
Code

```
SELECT ST_SetStartM('LINESTRING M(0 0 10,1 1 20,2 0 30)'::geometry, 100);
```

Output

```
LINESTRING M (0 0 100,1 1 20,2 0 30)
```

Figure



Add M dimension and set start M value

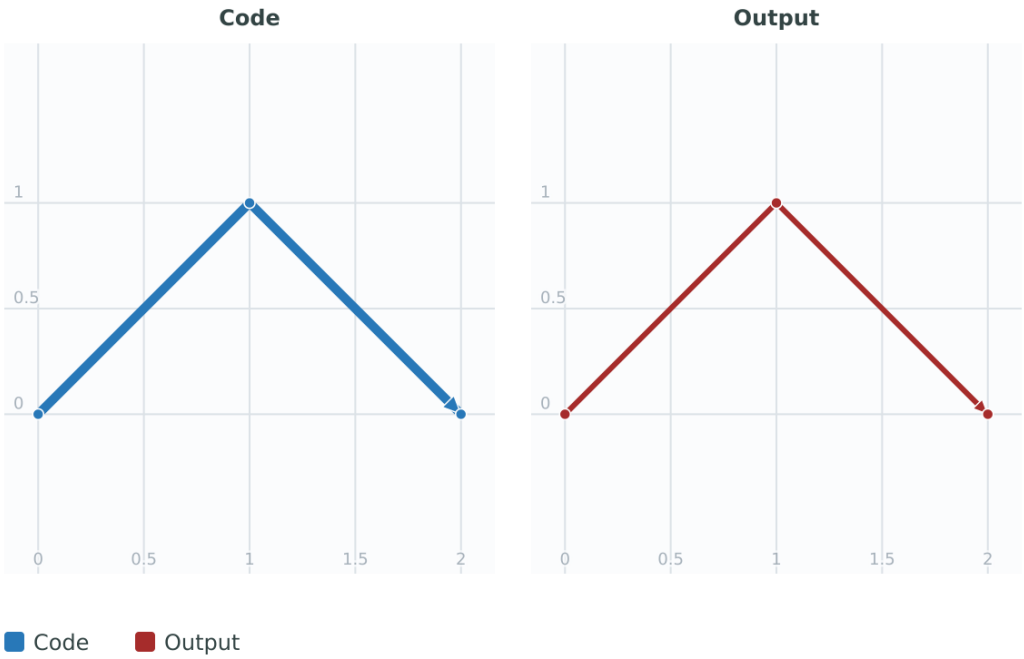
Code

```
SELECT ST_SetStartM('LINESTRING(0 0,1 1,2 0)'::geometry, 50);
```

Output

```
LINESTRING M (0 0 50,1 1 0,2 0 0)
```

Figure



Set start M value of a CircularString

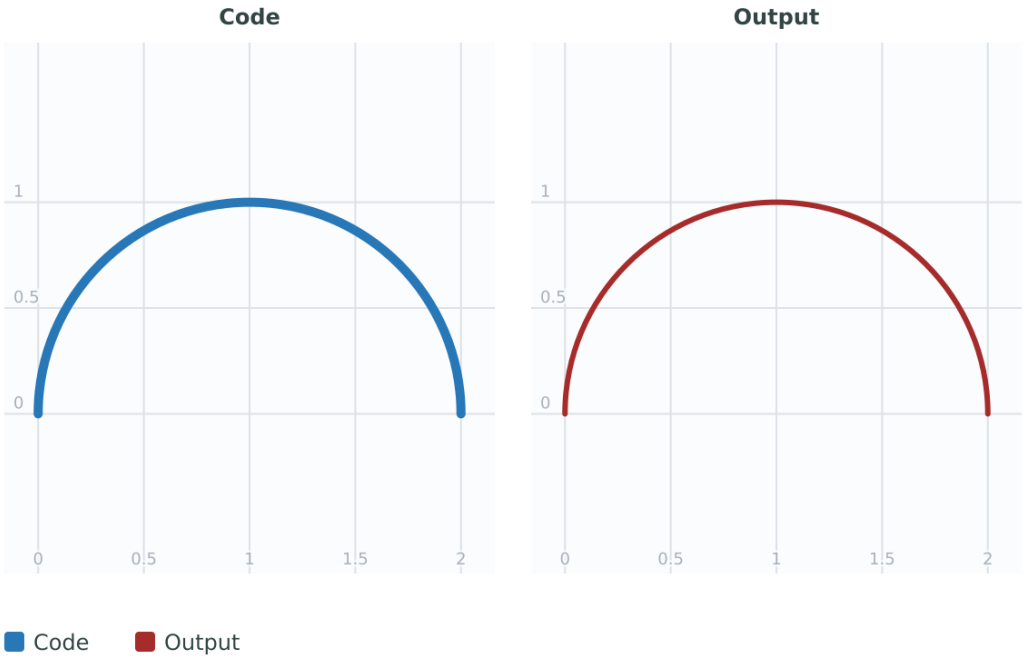
Code

```
SELECT ST_SetStartM('CIRCULARSTRING M(0 0 5,1 1 10,2 0 15)'::geometry, 99);
```

Output

```
CIRCULARSTRING M (0 0 99,1 1 10,2 0 15)
```

Figure



Set start M value of a NURBSCurve

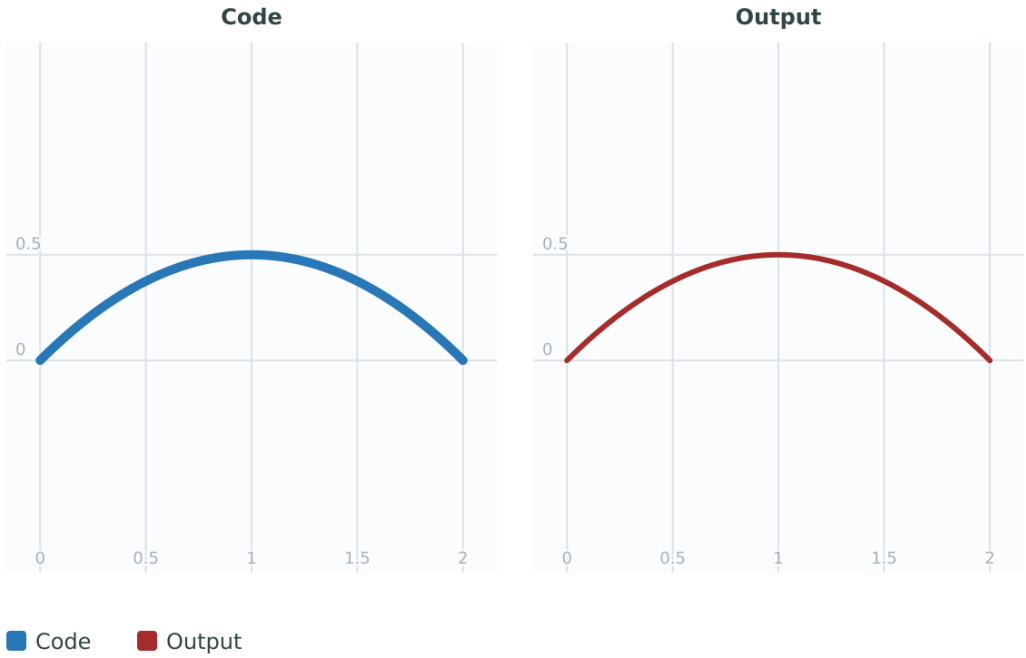
Code

```
SELECT ST_SetStartM('NURBSCURVE M(2, (0 0 100, 1 1 200, 2 0 300))'::geometry, 500);
```

Output

```
NURBSCURVE M (DEGREE 2,CONTROLPOINTS M (NURBSPOINT(WEIGHTEDPOINT M (0 0 500),WEIGHT 1), NURBSPOINT(WEIGHTEDPOINT M (1 1 ↩
     200),WEIGHT 1), NURBSPOINT(WEIGHTEDPOINT M (2 0 300),WEIGHT 1)),KNOTS (KNOT(0,3), KNOT(1,3)))
```

Figure



Siehe auch

ST_SetEndM, ST_StartM, ST_StartPoint

7.4.44 ST_SetEndM

ST_SetEndM — Sets the M coordinate of the last point of a LineString, CircularLineString, or NURBSCurve.

Synopsis

geometry **ST_SetEndM**(geometry geom, float8 m);

Beschreibung

Returns a copy of the input `LINESTRING`, `CIRCULARLINESTRING`, or `NURBSCURVE` geometry with the M coordinate of the last point set to the specified value. If the input geometry does not have an M dimension, it will be added. Returns `NULL` if the input is not a `LINESTRING`, `CIRCULARLINESTRING`, or `NURBSCURVE`.

Diese Methode unterstützt kreisförmige Strings und Kurven.

Beispiele

Set end M value of a LineString

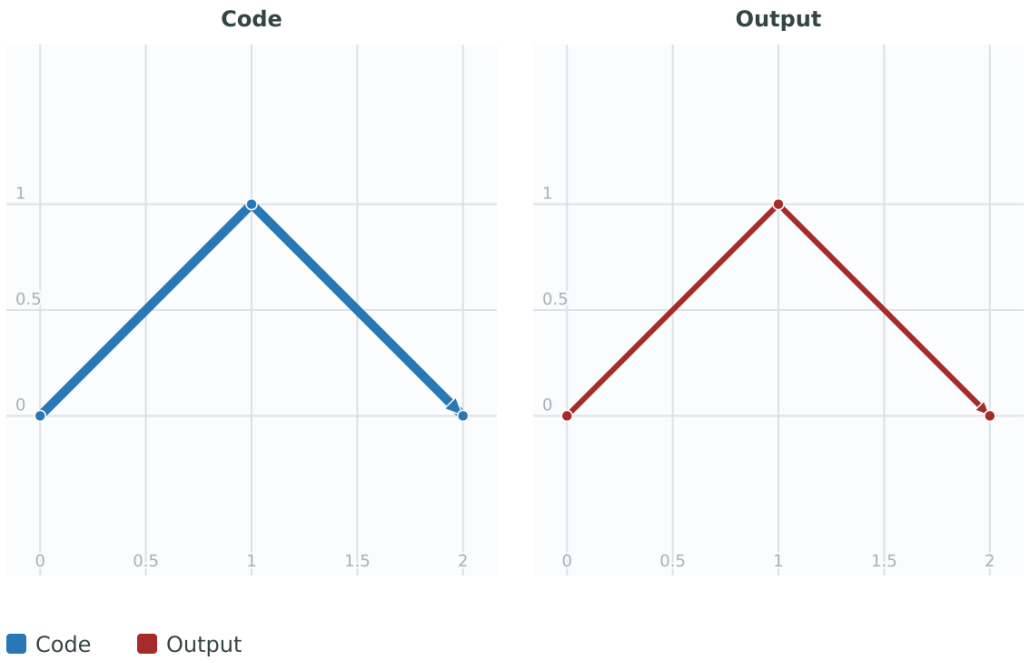
Code

```
SELECT ST_SetEndM('LINESTRING M(0 0 10,1 1 20,2 0 30)'::geometry, 999);
```

Output

```
LINESTRING M (0 0 10,1 1 20,2 0 999)
```

Figure



Add M dimension and set end M value

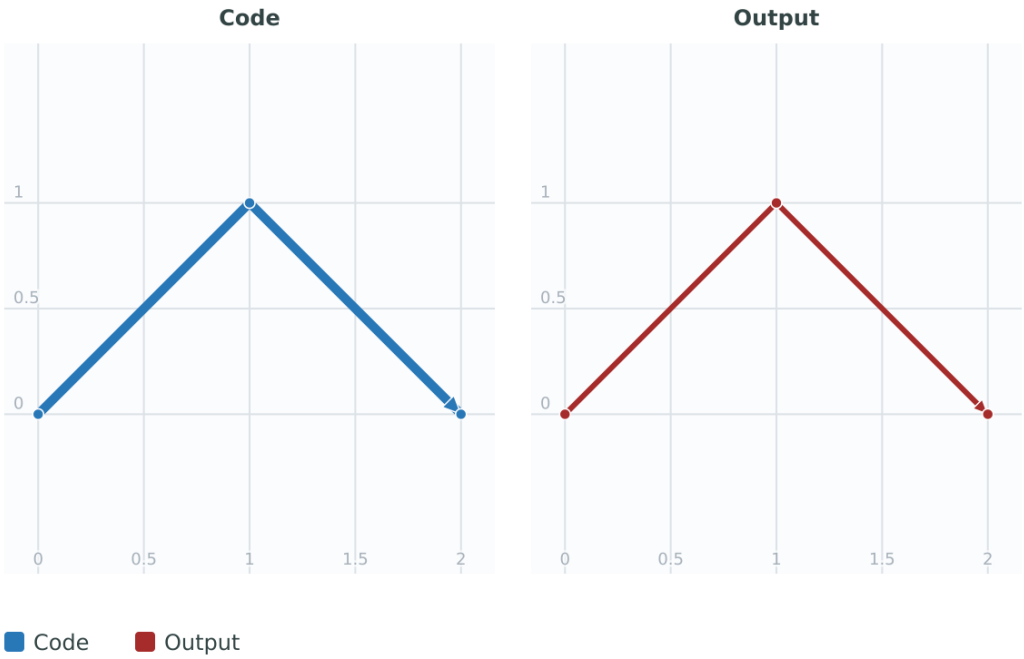
Code

```
SELECT ST_SetEndM('LINESTRING(0 0,1 1,2 0)'::geometry, 75);
```

Output

```
LINESTRING M (0 0 0,1 1 0,2 0 75)
```

Figure



Set end M value of a CircularString

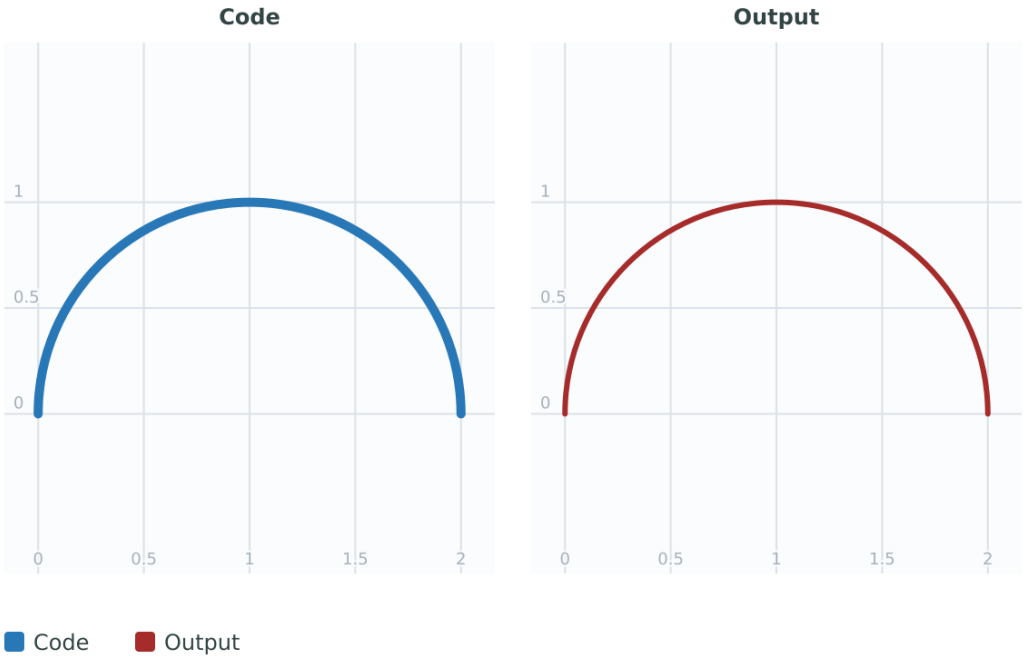
Code

```
SELECT ST_SetEndM('CIRCULARSTRING M(0 0 5,1 1 10,2 0 15)'::geometry, 88);
```

Output

```
CIRCULARSTRING M (0 0 5,1 1 10,2 0 88)
```

Figure



Set end M value of a NURBSCurve

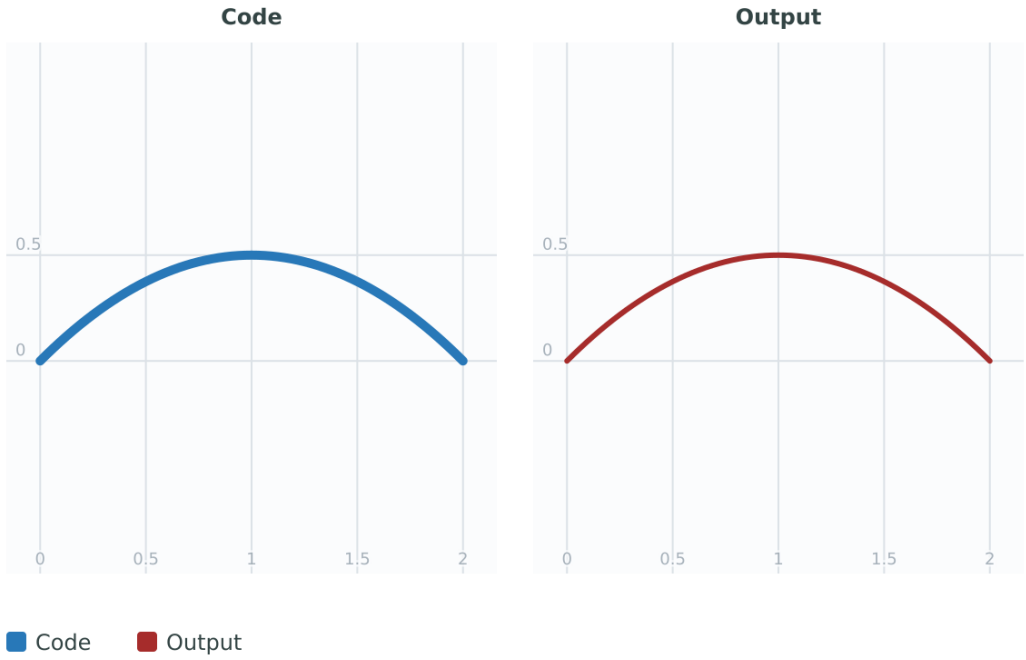
Code

```
SELECT ST_SetEndM('NURBSCURVE M(2, (0 0 100, 1 1 200, 2 0 300))'::geometry, 999);
```

Output

```
NURBSCURVE M (DEGREE 2,CONTROLPOINTS M (NURBSPOINT(WEIGHTEDPOINT M (0 0 100),WEIGHT 1), NURBSPOINT(WEIGHTEDPOINT M (1 1 ↩
      200),WEIGHT 1), NURBSPOINT(WEIGHTEDPOINT M (2 0 999),WEIGHT 1)),KNOTS (KNOT(0,3), KNOT(1,3)))
```

Figure



Siehe auch

ST_SetStartM, ST_EndM, ST_EndPoint

7.4.45 ST_Summary

ST_Summary — Gibt eine Zusammenfassung des Inhalts einer Geometrie wieder.

Synopsis

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

Beschreibung

Gibt eine Zusammenfassung des Inhalts einer Geometrie wieder.

Die Bedeutung der Flags, welche in eckigen Klammern hinter dem Geometrietyp angezeigt werden, ist wie folgt:

- M: besitzt eine M-Ordinate
- Z: besitzt eine Z-Ordinate

- B: besitzt ein zwischengespeichertes Umgebungsrechteck
- G: ist geodätisch (Geographie)
- S: besitzt ein räumliches Koordinatenreferenzsystem

 Diese Methode unterstützt kreisförmige Strings und Kurven.

 Diese Funktion unterstützt polyedrische Flächen.

 Diese Funktion unterstützt Dreiecke und dreieckige unregelmäßige Netzoberflächen (TIN).

Verfügbarkeit: 1.2.2

Erweiterung: 2.0.0 Unterstützung für geographische Koordinaten hinzugefügt

Erweiterung: 2.1.0 S-Flag, diese zeigt an ob das Koordinatenreferenzsystem bekannt ist

Erweiterung: 2.2.0 Unterstützung für TIN und Kurven

Beispiele

Code

```
SELECT ST_Summary('LINESTRING(0 0,1 1)'::geometry) as geom,
ST_Summary('POLYGON((0 0,1 1,1 2,1 1,0 0))'::geography) geog;
```

Output

```
geom              |          geog
------------------------------+--------------------------
 LineString[B] with 2 points | Polygon[BGS] with 1 rings
                              | ring 0 has 5 points
                              :
(1 row)
```

Code

```
SELECT ST_Summary('LINESTRING(0 0 1,1 1 1)'::geography) As geog_line,
ST_Summary('SRID=4326;POLYGON((0 0 1,1 1 2,1 2 3,1 1 1,0 0 1))'::geometry) As geom_poly;
```

Output

```
geog_line              |        geom_poly
------------------------------- +--------------------------
 LineString[ZBGS] with 2 points | Polygon[ZBS] with 1 rings
                                :    ring 0 has 5 points
                                :
(1 row)
```

Siehe auch

PostGIS_DropBBox, PostGIS_AddBBox, ST_Force3DM, ST_Force3DZ, ST_Force2D, geography

ST_IsValid, ST_IsValid, ST_IsValidReason, ST_IsValidDetail

7.4.46 ST_X

ST_X — Gibt die X-Koordinate eines Punktes zurück.

Synopsis

float **ST_X**(geometry a_point);

Beschreibung

Gibt die X-Koordinate eines Punktes, oder NULL wenn diese nicht vorhanden ist, zurück. Die Eingabe muss ein Punkt sein.



Anmerkung
Um den minimalen und maximalen X-Wert der Geometriekoordinaten zu ermitteln, verwenden Sie die Funktionen ST_XMin und ST_XMax.

Diese Methode setzt die SQL/MM-Spezifikation um. SQL-MM 3: 6.1.3

Diese Funktion unterstützt 3d und lässt den Z-Index nicht fallen.

Beispiele

Code

```
SELECT ST_X('POINT(1 2 3 4)');
```

Output

```
1
```

Code

```
SELECT ST_Y(ST_Centroid('LINESTRING(1 2 3 4,1 1 1 1)'::geometry));
```

Output

```
1.5
```

Siehe auch

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

7.4.47 ST_Y

ST_Y — Gibt die Y-Koordinate eines Punktes zurück.

Synopsis

float **ST_Y**(geometry a_point);

Beschreibung

Gibt die Y-Koordinate eines Punktes, oder NULL wenn diese nicht vorhanden ist, zurück. Die Eingabe muss ein Punkt sein.



Anmerkung
Um den minimalen und maximalen Y-Wert der Geometriekoordinaten zu ermitteln, verwenden Sie die Funktionen ST_YMin und ST_YMax.

Diese Methode implementiert die OGC Simple Features Implementation Specification for SQL 1.1.

Diese Methode setzt die SQL/MM-Spezifikation um. SQL-MM 3: 6.1.4

Diese Funktion unterstützt 3d und lässt den Z-Index nicht fallen.

Beispiele

Code

```
SELECT ST_Y('POINT(1 2 3 4)');
```

Output

```
2
```

Code

```
SELECT ST_Y(ST_Centroid('LINESTRING(1 2 3 4,1 1 1 1)'::geometry));
```

Output

```
1.5
```

Siehe auch

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

7.4.48 ST_Z

ST_Z — Gibt die Z-Koordinate eines Punktes zurück.

Synopsis

float **ST_Z**(geometry a_point);

Beschreibung

Gibt die Z-Koordinate eines Punktes, oder NULL wenn diese nicht vorhanden ist, zurück. Die Eingabe muss ein Punkt sein.



Anmerkung
Um den minimalen und maximalen Z-Wert der Geometriekoordinaten zu ermitteln, verwenden Sie die Funktionen ST_ZMin und ST_ZMax.

Diese Methode setzt die SQL/MM-Spezifikation um.

Diese Funktion unterstützt 3d und lässt den Z-Index nicht fallen.

Beispiele

Code

```
SELECT ST_Z('POINT(1 2 3 4)');
```

Output

```
3
```

Siehe auch

ST_GeomFromEWKT, ST_M, ST_X, ST_Y, ST_ZMax, ST_ZMin

7.4.49 ST_ZMFlag

ST_ZMFlag — Gibt die Dimension der Koordinaten von ST_Geometry zurück.

Synopsis

smallint **ST_ZMFlag**(geometry geomA);

Beschreibung

Gibt die Dimension der Koordinaten für den Wert von ST_Geometry zurück.

Die Werte sind: 0 = 2D, 1 = 3D-M, 2 = 3D-Z, 3 = 4D.

Diese Funktion unterstützt 3d und lässt den Z-Index nicht fallen.

Diese Methode unterstützt kreisförmige Strings und Kurven.

Beispiele

Code

```
SELECT ST_ZMFlag('LINESTRING(1 2,3 4)');
```

Output

```
0
```

Code

```
SELECT ST_ZMFlag('LINESTRINGM(1 2 3, 3 4 3)');
```

Output

```
1
```

Code

```
SELECT ST_ZMFlag('CIRCULARSTRING(1 2 3,3 4 3,5 6 3)');
```

Output

```
2
```

Code

```
SELECT ST_ZMFlag('POINT(1 2 3 4)');
```

Output

```
3
```

Siehe auch

ST_CoordDim, ST_NDims, ST_Dimension

7.4.50 ST_HasZ

ST_HasZ — Prüft, ob eine Geometrie eine Z-Dimension hat.

Synopsis

boolean **ST_HasZ**(geometry geom);

Beschreibung

Prüft, ob die Eingabegeometrie eine Z-Dimension hat und gibt einen booleschen Wert zurück. Wenn die Geometrie eine Z-Dimension hat, wird true zurückgegeben, andernfalls false.

Geometrieobjekte mit einer Z-Dimension stellen typischerweise dreidimensionale (3D) Geometrien dar, während solche ohne Z-Dimension zweidimensionale (2D) Geometrien sind.

Diese Funktion ist nützlich, um festzustellen, ob eine Geometrie Höhen- oder Breiteninformationen enthält.

Verfügbarkeit: 3.5.0

 Diese Funktion unterstützt 3d und lässt den Z-Index nicht fallen.

 Diese Funktion unterstützt M-Koordinaten.

Beispiele

Code

```
SELECT ST_HasZ('POINT(1 2 3)');
```

Output

```
t
```

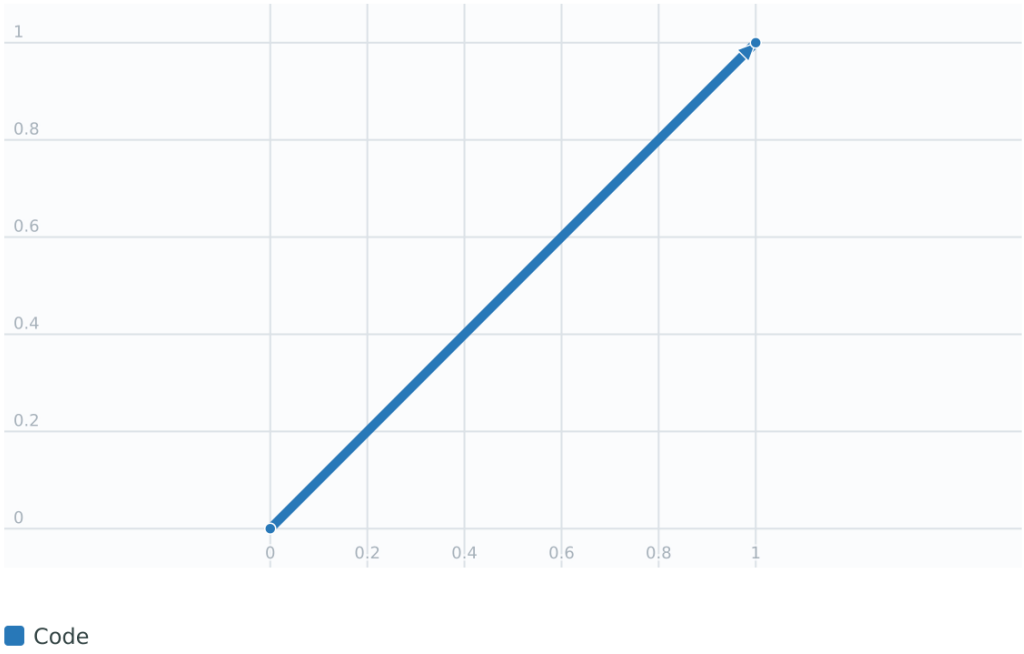
Code

```
SELECT ST_HasZ('LINESTRING(0 0,1 1)');
```

Output

```
f
```

Figure



Siehe auch

ST_ZMFlag

ST_HasM

7.4.51 ST_HasM

ST_HasM — Prüft, ob eine Geometrie eine M-Dimension (Maß) hat.

Synopsis

boolean **ST_HasM**(geometry geom);

Beschreibung

Prüft, ob die Eingabegeometrie eine M-Dimension (Maß) hat und gibt einen booleschen Wert zurück. Wenn die Geometrie ein M-Maß hat, wird true zurückgegeben, andernfalls false.

Geometrieobjekte mit einer M-Dimension stellen in der Regel Messungen oder zusätzliche Daten dar, die mit räumlichen Merkmalen verbunden sind.

Diese Funktion ist nützlich, um festzustellen, ob eine Geometrie Messinformationen enthält.

Verfügbarkeit: 3.5.0

Diese Funktion unterstützt 3d und lässt den Z-Index nicht fallen.

Diese Funktion unterstützt M-Koordinaten.

Beispiele

Code

```
SELECT ST_HasM('POINTM(1 2 3)');
```

Output

```
t
```

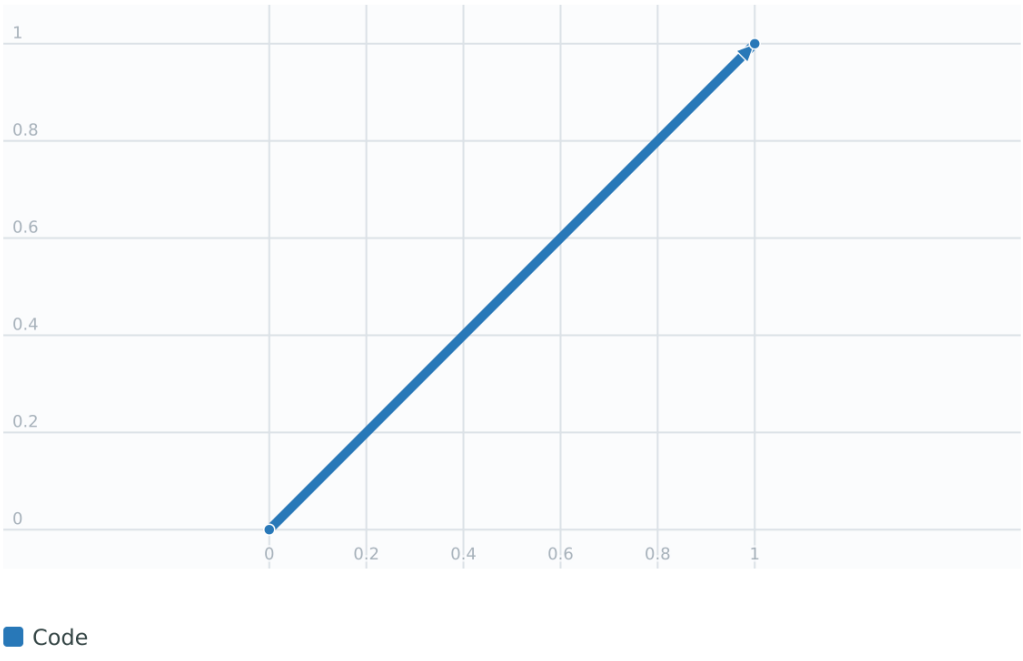
Code

```
SELECT ST_HasM('LINESTRING(0 0,1 1)');
```

Output

```
f
```

Figure



Siehe auch

ST_ZMFlag
ST_HasZ

7.4.52 ST_ControlPoints

ST_ControlPoints — Returns the control points of a NURBS curve as a MULTIPOINT geometry.

Synopsis

geometry **ST_ControlPoints**(geometry nurbscurve);

Beschreibung

Extracts the control points from a NURBS curve and returns them as a MULTIPOINT geometry. Control points define the control polygon that influences the shape of the NURBS curve.

The function raises an error if the input is not a NURBS curve. Returns NULL if the input is NULL.

Availability: 3.7.0

Beispiele

This example extracts control points from a NURBS curve.

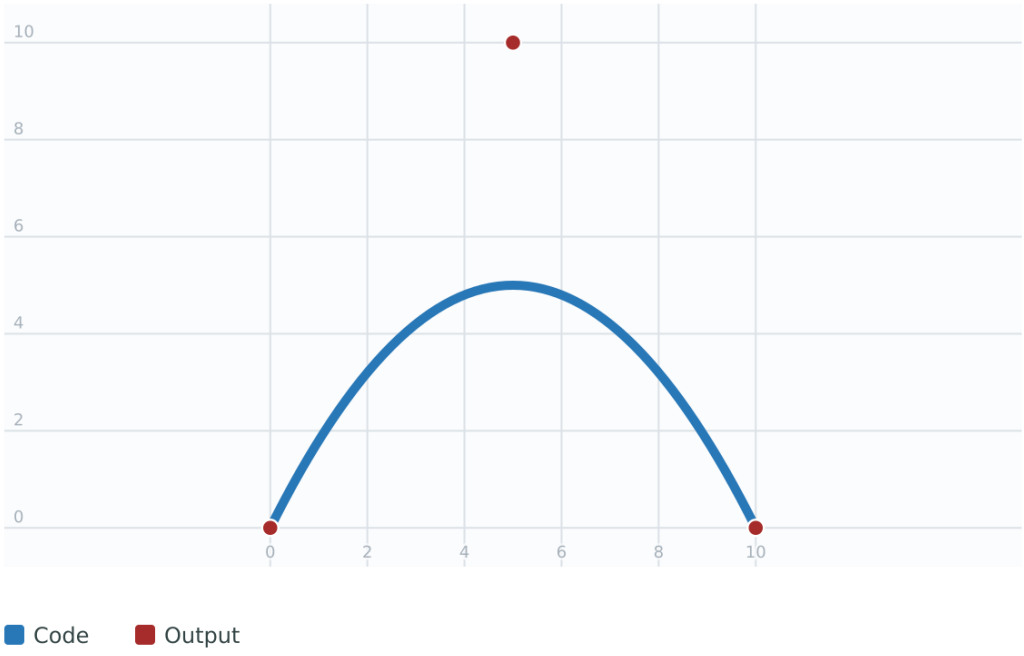
Code

```
SELECT ST_ControlPoints('NURBSCURVE(2, (0 0, 5 10, 10 0))'::geometry);
```

Output

```
MULTIPOINT((0 0),(5 10),(10 0))
```

Figure



This example counts the control points.

Code

```
SELECT ST_NPoints(ST_ControlPoints('NURBSCURVE(2, (0 0, 5 10, 10 0))'::geometry));
```

Output

```
3
```

Siehe auch

ST_MakeNurbsCurve, ST_NumControlPoints, ST_Weights

7.4.53 ST_Degree

ST_Degree — Returns the polynomial degree of a NURBS curve.

Synopsis

integer **ST_Degree**(geometry nurbscurve);

Beschreibung

Returns the polynomial degree of a NURBS curve as an integer. The degree determines the curve's smoothness and flexibility. Higher degrees produce smoother curves but require more control points.

The function raises an error if the input is not a NURBS curve. Returns NULL if the input is NULL.

Availability: 3.7.0

Beispiele

This example returns the degree of a NURBS curve.

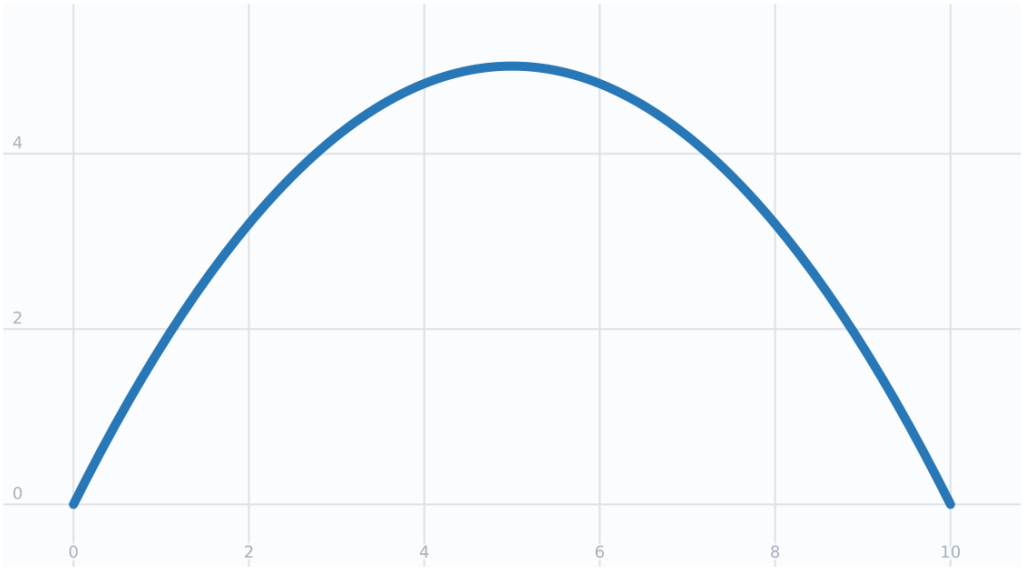
Code

```
SELECT ST_Degree('NURBSCURVE(2, (0 0, 5 10, 10 0))'::geometry);
```

Output

```
2
```

Figure



This example checks whether a NURBS curve is cubic.

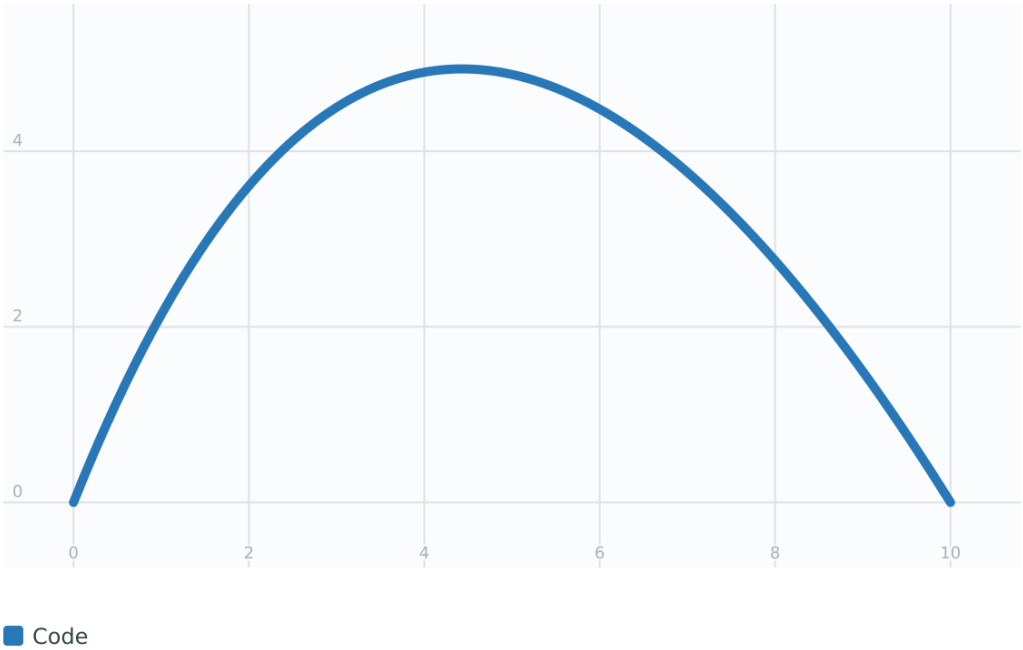
Code

```
SELECT ST_Degree('NURBSCURVE(3, (0 0, 2 5, 5 8, 10 0))'::geometry) = 3;
```

Output

```
t
```

Figure



Siehe auch

ST_MakeNurbsCurve, ST_NumControlPoints

7.4.54 ST_Weights

ST_Weights — Returns the weight array of a rational NURBS curve.

Synopsis

float8[] **ST_Weights**(geometry nurbscurve);

Beschreibung

Returns the weight values of a rational NURBS curve as an array of double precision values. Weights control the influence of each control point on the curve shape. Higher weights pull the curve closer to the corresponding control point.

Returns NULL if the curve has no weights (non-rational B-spline) or if the input is NULL. Raises an error if the input is not a NURBS curve.

Availability: 3.7.0

Beispiele

This example returns weights from a rational NURBS curve.

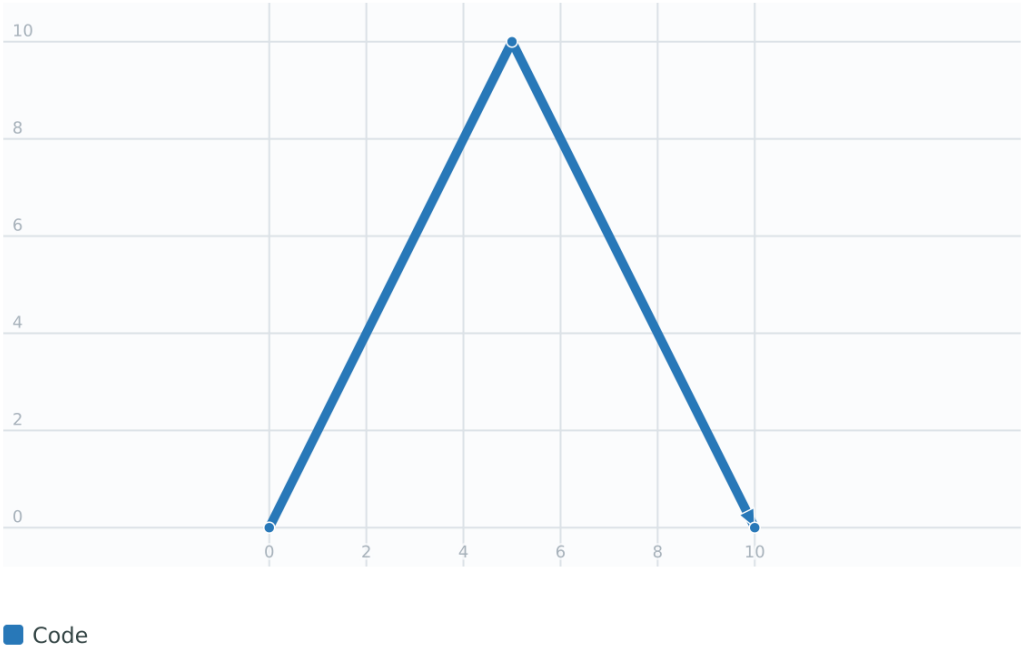
Code

```
SELECT ST_Weights(ST_MakeNurbsCurve(2, 'LINESTRING(0 0,5 10,10 0)'::geometry, ARRAY[1.0,  ↩
    2.0, 1.0]));
```

Output

```
{1,2,1}
```

Figure



This example checks the weights of a non-rational curve.

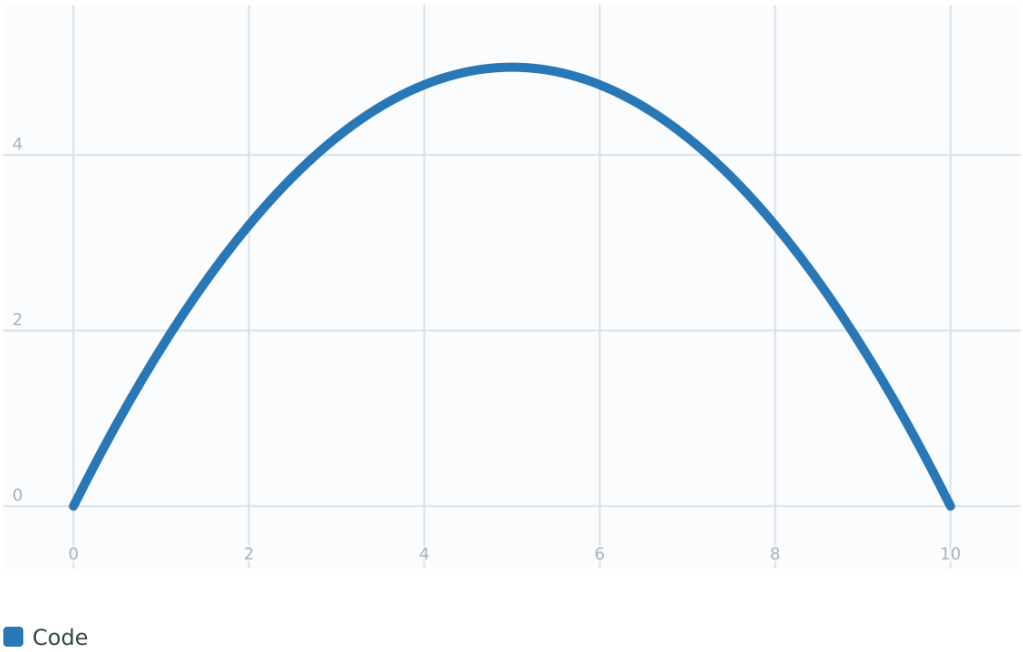
Code

```
SELECT ST_Weights('NURBSCURVE(2, (0 0, 5 10, 10 0))'::geometry);
```

Output

```
null
```

Figure



Siehe auch

ST_MakeNurbsCurve, ST_NurbsCurveIsRational, ST_ControlPoints

7.4.55 ST_Knots

ST_Knots — Returns the knot vector of a NURBS curve.

Synopsis

float8[] **ST_Knots**(geometry nurbscurve);

Beschreibung

Returns the knot vector of a NURBS curve as an array of double precision values. The knot vector defines the parameter space and controls where and how the control points affect the curve. It must be a non-decreasing sequence with length equal to (number of control points + degree + 1).

Returns NULL if the curve uses the default uniform clamped knot vector or if the input is NULL. Raises an error if the input is not a NURBS curve.

Availability: 3.7.0

Beispiele

This example returns knots from a NURBS curve with a custom knot vector.

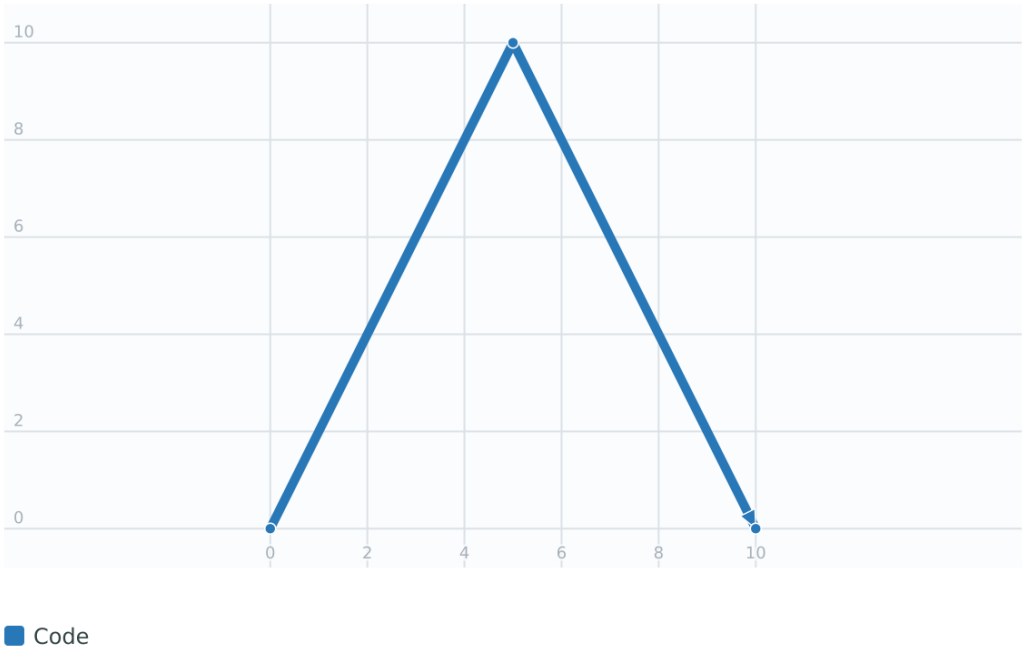
Code

```
SELECT ST_Knots(ST_MakeNurbsCurve(
        2,
        'LINESTRING(0 0,5 10,10 0)'::geometry,
        ARRAY[1.0, 2.0, 1.0],
        ARRAY[0, 0, 0, 1, 1, 1]));
```

Output

```
{0,0,0,1,1,1}
```

Figure



This example checks the knots of a curve using the default knot vector.

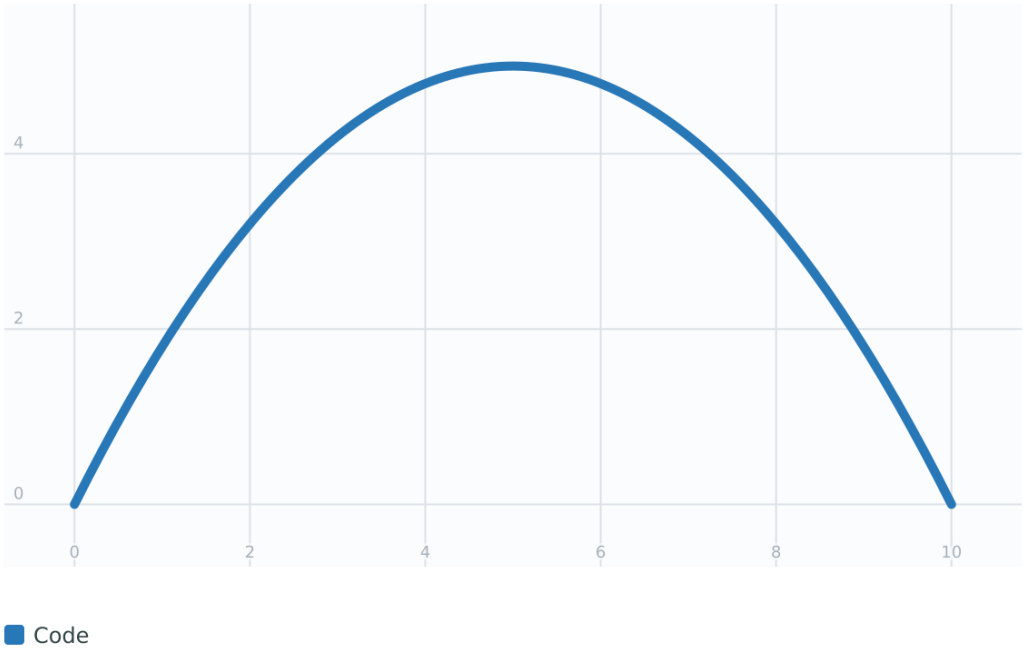
Code

```
SELECT ST_Knots('NURBSCURVE(2, (0 0, 5 10, 10 0))'::geometry);
```

Output

```
null
```

Figure



Siehe auch

ST_MakeNurbsCurve, ST_Degree, ST_NumControlPoints

7.4.56 ST_NumControlPoints

ST_NumControlPoints — Returns the number of control points in a NURBS curve.

Synopsis

integer **ST_NumControlPoints**(geometry nurbscurve);

Beschreibung

Returns the number of control points in a NURBS curve. The number of control points must be at least (degree + 1) for a valid NURBS curve.

Raises an error if the input is not a NURBS curve. Returns NULL if the input is NULL.

Availability: 3.7.0

Beispiele

This example returns the number of control points.

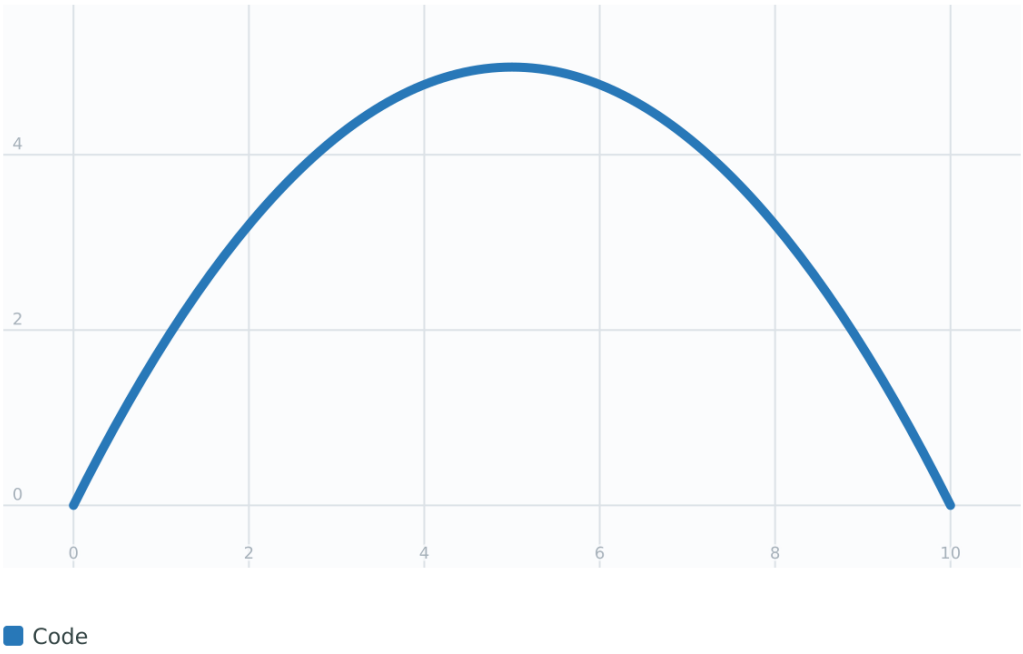
Code

```
SELECT ST_NumControlPoints('NURBSCURVE(2, (0 0, 5 10, 10 0))'::geometry);
```

Output

```
3
```

Figure



This example verifies that a curve has enough control points for its degree.

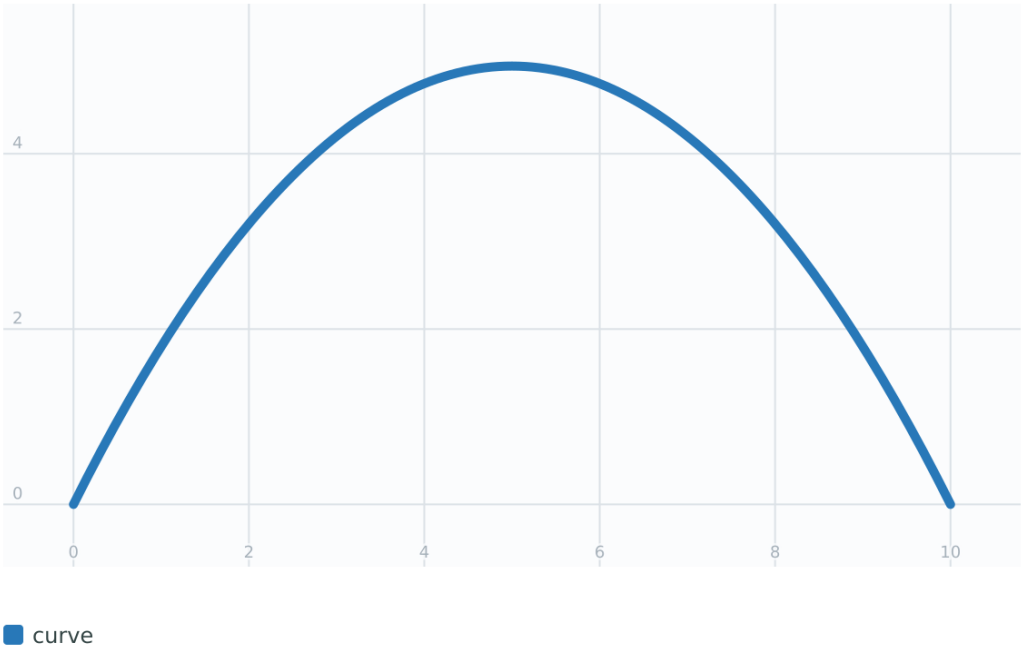
Code

```
SELECT ST_NumControlPoints(curve)
>= ST_Degree(curve) + 1 AS valid
FROM (SELECT 'NURBSCURVE(2, (0 0, 5 10, 10 0))'::geometry AS curve) AS t;
```

Output

```
t
```

Figure



Siehe auch

ST_MakeNurbsCurve, ST_ControlPoints, ST_Degree

7.4.57 ST_NurbsCurveIsRational

ST_NurbsCurveIsRational — Checks if a NURBS curve is rational (has weights).

Synopsis

boolean **ST_NurbsCurveIsRational**(geometry nurbscurve);

Beschreibung

Returns TRUE if the NURBS curve has weights assigned to its control points (rational NURBS), FALSE if it uses uniform weights (non-rational B-spline).

Rational NURBS curves provide more flexibility in curve shape control through variable weights. Non-rational B-splines use uniform weights and are a special case of NURBS curves.

Raises an error if the input is not a NURBS curve. Returns NULL if the input is NULL.

Availability: 3.7.0

Beispiele

This example checks whether a curve is rational.

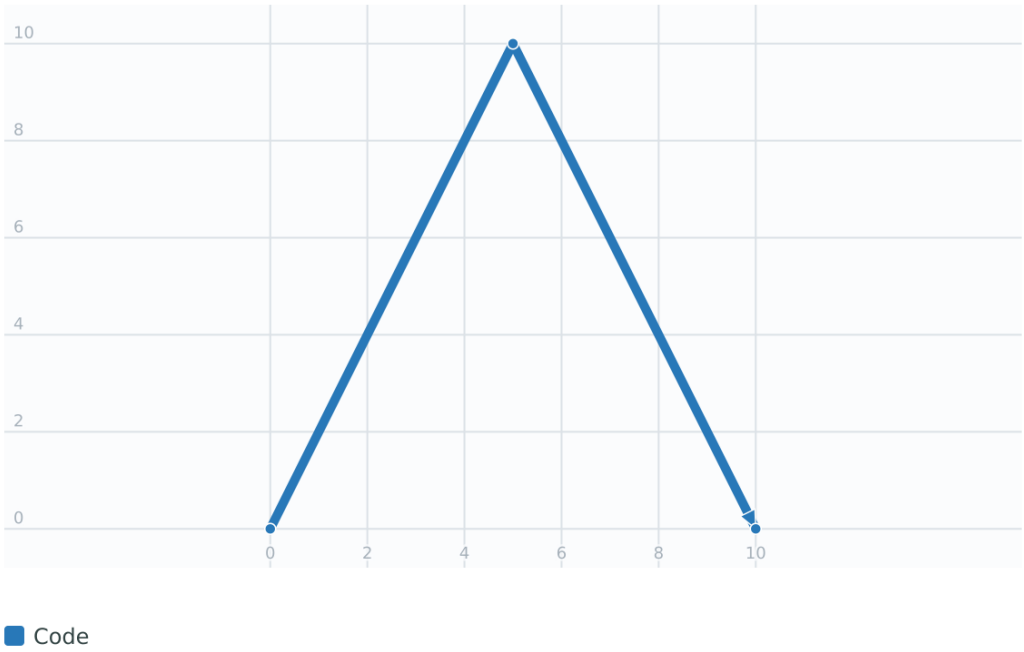
Code

```
SELECT ST_NurbsCurveIsRational(ST_MakeNurbsCurve(
        2,
        'LINESTRING(0 0,5 10,10 0)'::geometry,
        ARRAY[1.0, 2.0, 1.0]));
```

Output

```
t
```

Figure



This example checks a non-rational curve.

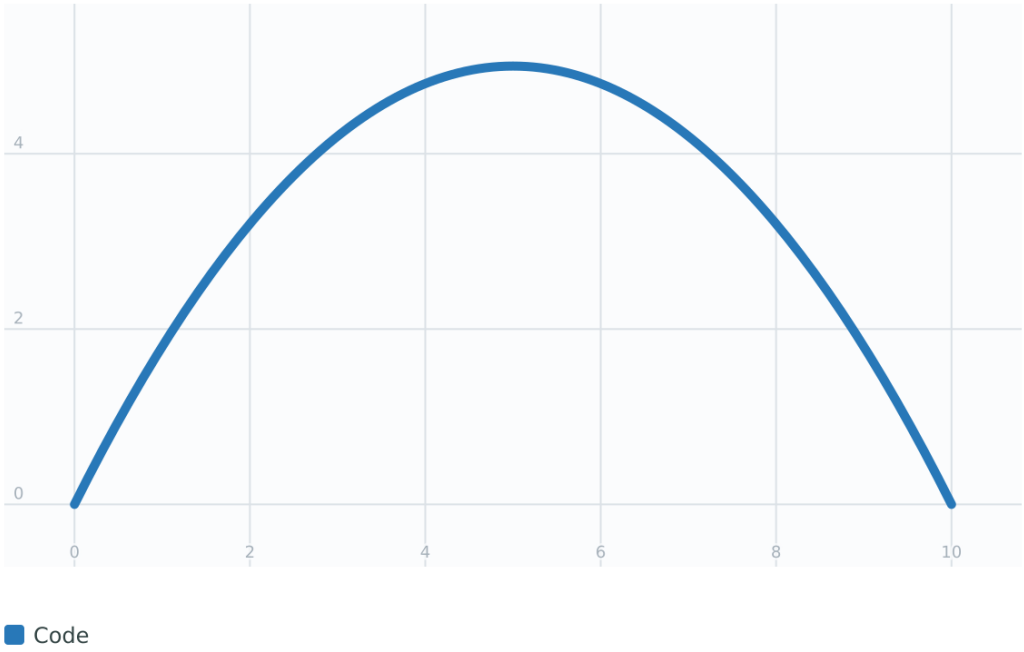
Code

```
SELECT ST_NurbsCurveIsRational('NURBSCURVE(2, (0 0, 5 10, 10 0))'::geometry);
```

Output

```
f
```

Figure



Siehe auch

ST_MakeNurbsCurve, ST_Weights, ST_IsNurbsCurve

7.4.58 ST_IsNurbsCurve

ST_IsNurbsCurve — Checks if a geometry is a NURBS curve.

Synopsis

boolean **ST_IsNurbsCurve**(geometry geom);

Beschreibung

Returns TRUE if the geometry is a NURBS curve, FALSE otherwise. This function can be used to test geometry types before applying NURBS-specific functions.

Returns NULL if the input is NULL.

Availability: 3.7.0

Beispiele

This example checks whether a geometry is a NURBS curve.

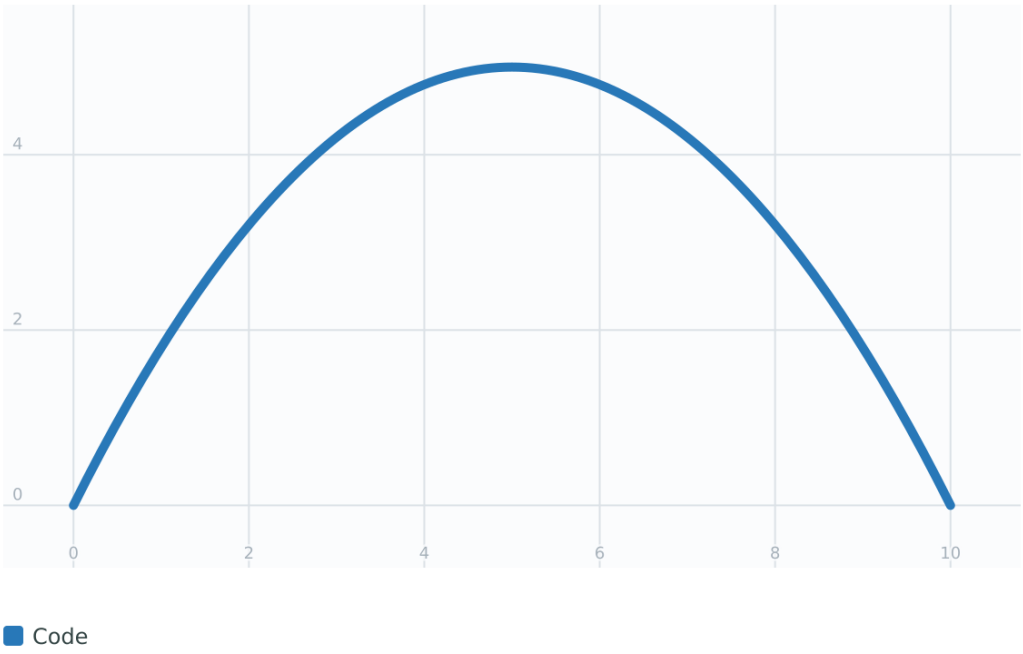
Code

```
SELECT ST_IsNurbsCurve('NURBSCURVE(2, (0 0, 5 10, 10 0))'::geometry);
```

Output

```
t
```

Figure



This example tests a regular LineString.

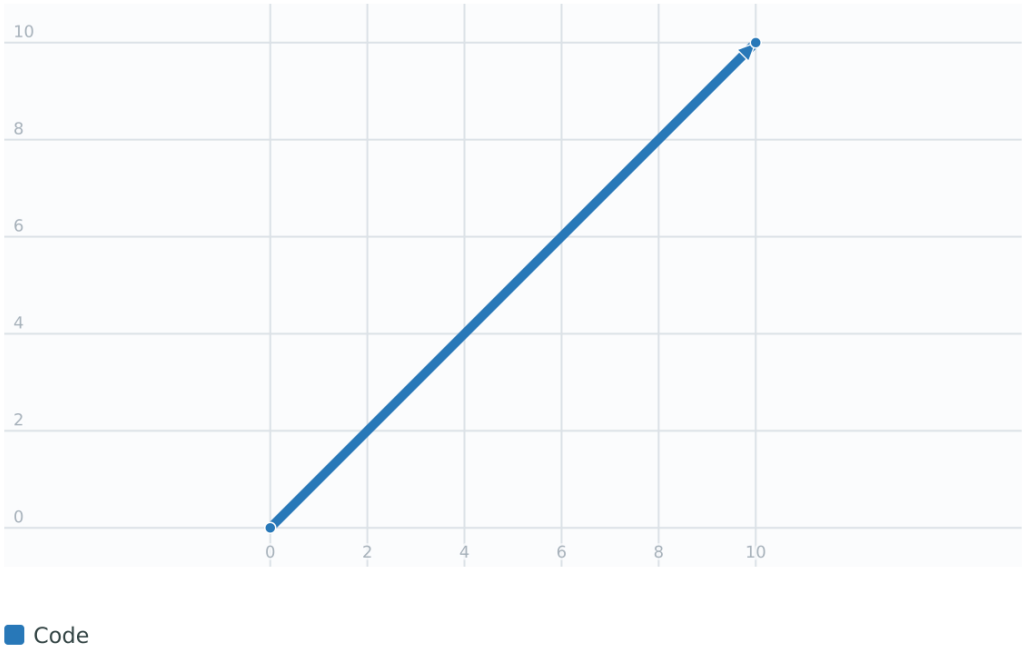
Code

```
SELECT ST_IsNurbsCurve('LINESTRING(0 0,10 10)'::geometry);
```

Output

```
f
```

Figure



Siehe auch

ST_MakeNurbsCurve, ST_GeometryType, ST_NurbsCurveIsRational

7.4.59 ST_Evaluate

ST_Evaluate — Evaluates a NURBS curve at a specific parameter value and returns the resulting point.

Synopsis

geometry **ST_Evaluate**(geometry nurbscurve, float8 parameter);

Beschreibung

Evaluates a NURBS curve at the specified parameter value and returns a POINT geometry representing the position on the curve. The parameter typically ranges from 0.0 (start of curve) to 1.0 (end of curve), though the actual range depends on the knot vector.

Raises an error if the input is not a NURBS curve or if evaluation fails. Returns NULL if either input is NULL.

Availability: 3.7.0

Beispiele

This example returns the midpoint of a NURBS curve.

Code

```
SELECT ST_Evaluate('NURBSCURVE(2, (0 0, 5 10, 10 0))'::geometry, 0.5);
```

This example samples multiple points along a curve.

Code

```
SELECT t, ST_Evaluate(curve, t) AS point
FROM (SELECT 'NURBSCURVE(2, (0 0, 5 10, 10 0))'::geometry AS curve) AS c,
         generate_series(0, 1, 0.25) AS t;
```

Siehe auch

ST_MakeNurbsCurve, ST_NurbsToLineString

7.4.60 ST_NurbsToLineString

ST_NurbsToLineString — Converts a NURBS curve to a LINESTRING by uniform sampling.

Synopsis

geometry **ST_NurbsToLineString**(geometry nurbscurve, integer num_segments=32);

Beschreibung

Converts a NURBS curve to a LINESTRING geometry by creating a piecewise linear approximation. The curve is sampled at uniformly distributed parameter values and the resulting points are connected with straight line segments.

The optional `num_segments` parameter specifies the number of line segments in the output (default: 32, range: 2-10000). Higher values produce smoother approximations but increase geometry complexity.

Raises an error if the input is not a NURBS curve. Returns NULL if the input is NULL.

Availability: 3.7.0

Beispiele

This example converts a NURBS curve to a LineString using the default quality.

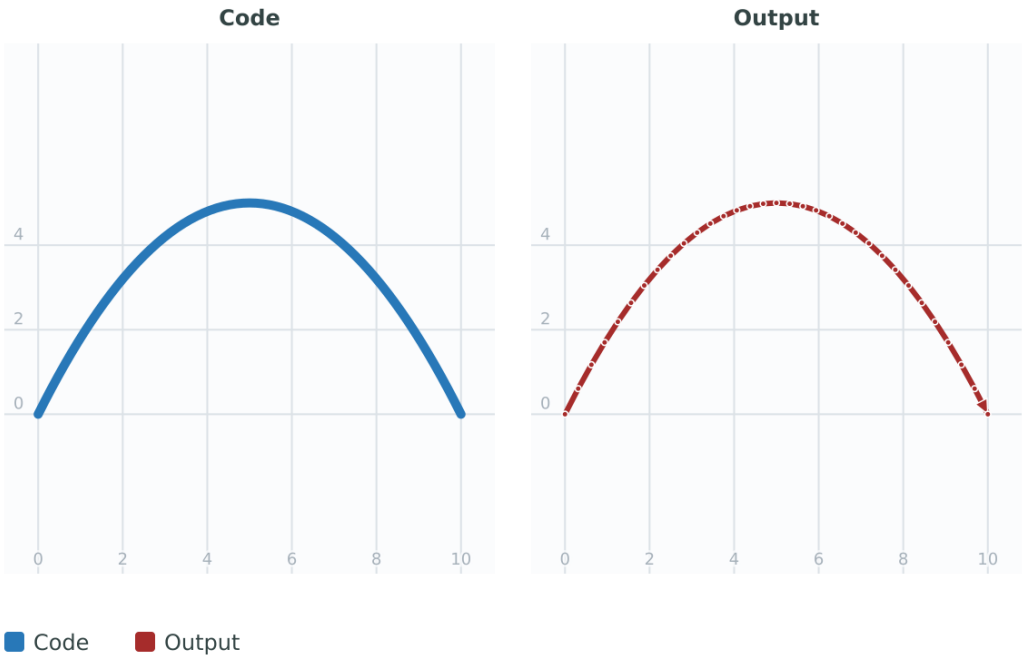
Code

```
SELECT ST_NurbsToLineString('NURBSCURVE(2, (0 0, 5 10, 10 0))'::geometry);
```

Output

```
LINESTRING(0 0,0.3125 0.60546875,0.625 1.171875,0.9375 1.69921875,1.25 2.1875,1.5625 2.63671875,1.875 3.046875,2.1875 ↩
    3.41796875,2.5 3.75,2.8125 4.04296875,3.125 4.296875,3.4375 4.51171875,3.75 4.6875,4.0625 4.82421875,4.375 ↩
    4.921875,4.6875 4.98046875,5 5,5.3125 4.98046875,5.625 4.921875,5.9375 4.82421875,6.25 4.6875,6.5625 ↩
    4.51171875,6.875 4.296875,7.1875 4.04296875,7.5 3.75,7.8125 3.41796875,8.125 3.046875,8.4375 2.63671875,8.75 ↩
    2.1875,9.0625 1.69921875,9.375 1.171875,9.6875 0.60546875,10 0)
```

Figure

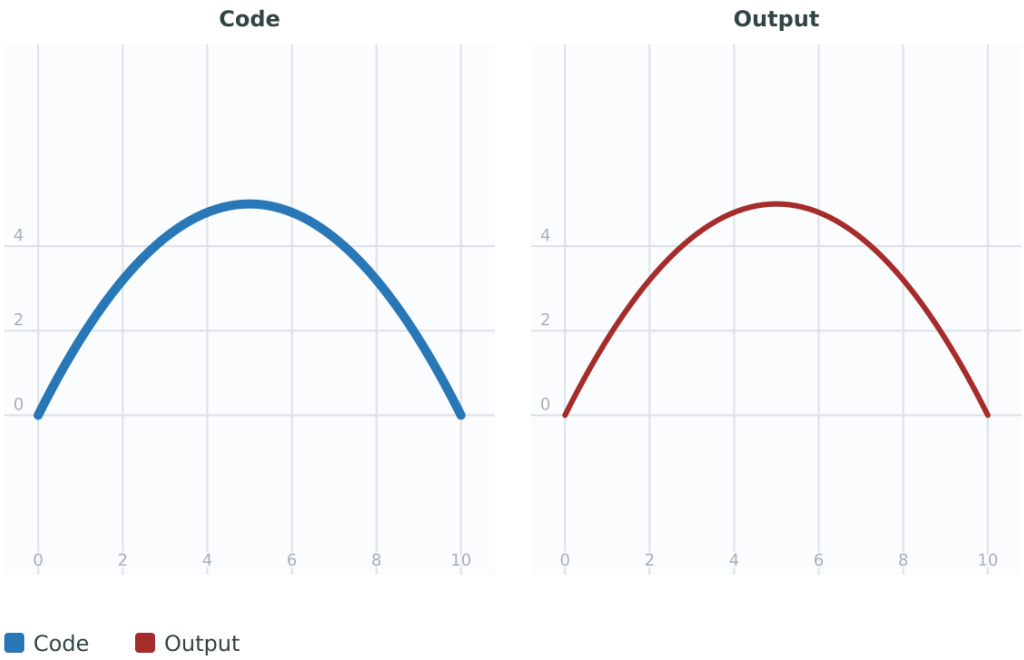


This example creates a high-quality approximation with 64 segments.

Code

```
SELECT ST_NurbsToLineString('NURBSCURVE(2, (0 0, 5 10, 10 0))'::geometry, 64);
```

Figure



This example creates a lower-quality, faster approximation with 16 segments.

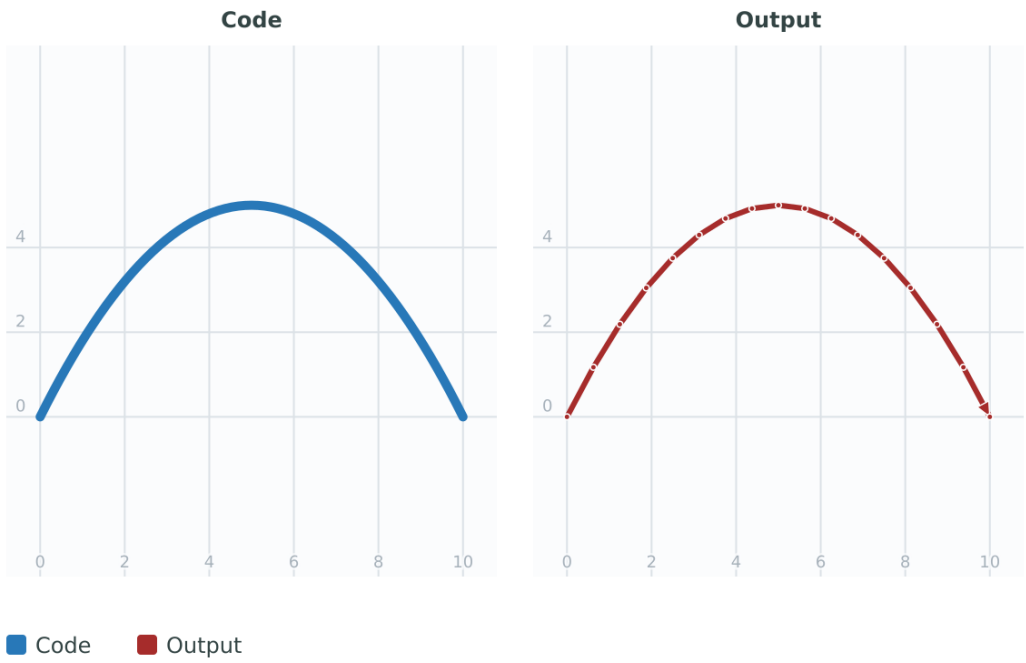
Code

```
SELECT ST_NurbsToLineString('NURBSCURVE(2, (0 0, 5 10, 10 0))'::geometry, 16);
```

Output

```
LINESTRING(0 0,0.625 1.171875,1.25 2.1875,1.875 3.046875,2.5 3.75,3.125 4.296875,3.75 4.6875,4.375 4.921875,5 5,5.625 ↩
    4.921875,6.25 4.6875,6.875 4.296875,7.5 3.75,8.125 3.046875,8.75 2.1875,9.375 1.171875,10 0)
```

Figure



Siehe auch

ST_MakeNurbsCurve, ST_Evaluate, ST_Segmentize

7.5 Geometrische Editoren

7.5.1 ST_AddPoint

ST_AddPoint — Fügt einem Linienzug einen Punkt hinzu.

Synopsis

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

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

Beschreibung

Fügt einen Punkt zu einem LineString vor dem Index *`Position`* hinzu (unter Verwendung eines 0-basierten Index). Wenn der Parameter *`position`* weggelassen wird oder -1 ist, wird der Punkt an das Ende des LineString angehängt.

Verfügbarkeit: 1.1.0

Diese Funktion unterstützt 3d und lässt den Z-Index nicht fallen.

Beispiele

Hinzufügen eines Punktes am Ende einer 3D-Linie

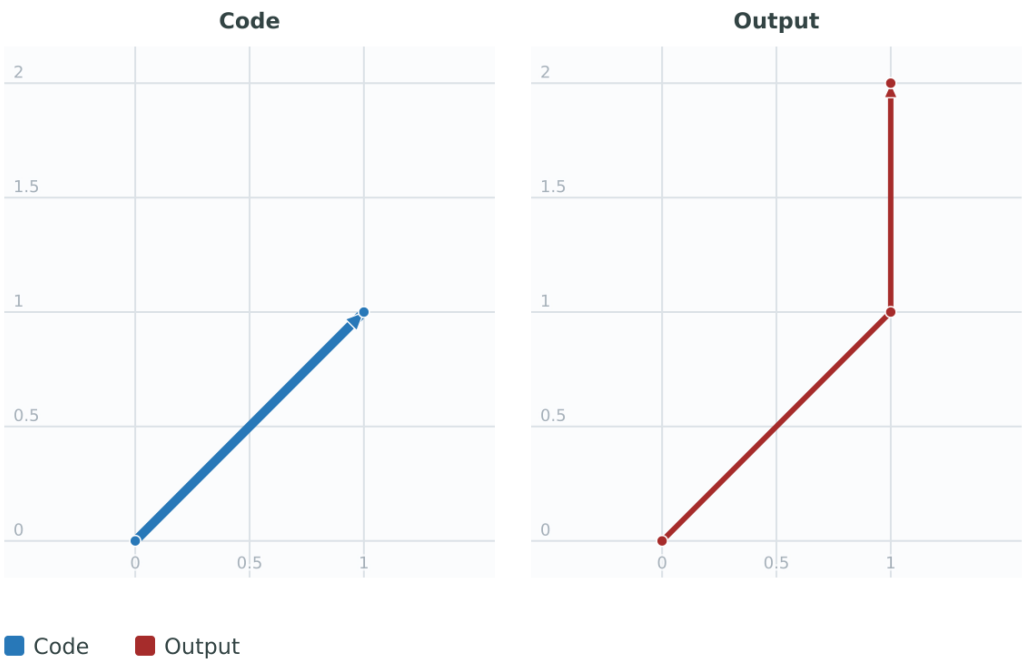
Code

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

Output

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

Figure



Garantiert, dass alle Zeilen einer Tabelle geschlossen sind, indem der Anfangspunkt jeder Zeile nur bei den nicht geschlossenen Zeilen an das Zeilenende angefügt wird.

Code

```
UPDATE sometable
SET geom = ST_AddPoint(geom, ST_StartPoint(geom))
FROM sometable
WHERE ST_IsClosed(geom) = false;
```

Siehe auch

ST_RemovePoint, ST_SetPoint

7.5.2 ST_CollectionExtract

ST_CollectionExtract — Gibt bei einer Geometriesammlung eine Multi-Geometrie zurück, die nur Elemente eines bestimmten Typs enthält.

Synopsis

geometry **ST_CollectionExtract**(geometry collection);

geometry **ST_CollectionExtract**(geometry collection, integer type);

Beschreibung

Gibt bei einer Geometriesammlung eine homogene Multi-Geometrie zurück.

Wenn der Typ nicht angegeben ist, wird eine Multi-Geometrie zurückgegeben, die nur Geometrien der höchsten Dimension enthält. Polygone werden also gegenüber Linien bevorzugt, die wiederum gegenüber Punkten bevorzugt werden.

Wenn der Typ angegeben ist, wird eine Multi-Geometrie zurückgegeben, die nur diesen Typ enthält. Gibt es keine Untergeometrien des richtigen Typs, wird eine EMPTY-Geometrie zurückgegeben. Es werden nur Punkte, Linien und Polygone unterstützt. Die Typnummern sind:

- 1 == PUNKT
- 2 == ZEILENUMBRUCH
- 3 == POLYGON

Bei atomaren Geometrieeingaben wird die Geometrie unverändert wiedergegeben, wenn der Eingabetyp mit dem angeforderten Typ übereinstimmt. Andernfalls ist das Ergebnis eine LEERE Geometrie des angegebenen Typs. Falls erforderlich, können diese mit ST_Multi in Multi-Geometrien umgewandelt werden.



Warnung
MultiPolygon-Ergebnisse werden nicht auf ihre Gültigkeit geprüft. Wenn die Polygonkomponenten nebeneinander liegen oder sich überlappen, ist das Ergebnis ungültig. (Dies kann z.B. bei der Anwendung dieser Funktion auf ein ST_Split Ergebnis auftreten.) Diese Situation kann mit ST_IsValid überprüft und mit ST_MakeValid behoben werden.

Verfügbarkeit: 1.5.0



Anmerkung
Vor 1.5.3 gab diese Funktion atomare Eingaben unverändert zurück, unabhängig vom Typ. In 1.5.3 lieferten nicht übereinstimmende Einzelgeometrien ein NULL-Ergebnis. In 2.0.0 geben nicht übereinstimmende Einzelgeometrien ein LEERES Ergebnis des angeforderten Typs zurück.

Beispiele

Extrahieren Sie den Typ mit der höchsten Dimension:

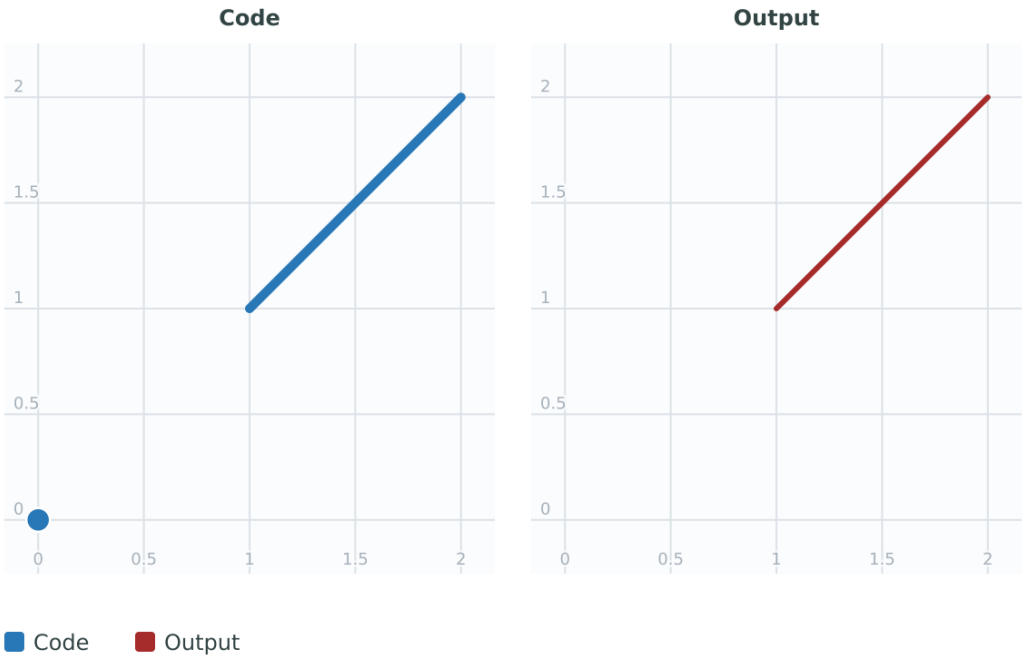
Code

```
SELECT ST_CollectionExtract('GEOMETRYCOLLECTION( POINT(0 0), LINESTRING(1 1,2 2) )');
```

Output

```
MULTILINESTRING((1 1, 2 2))
```

Figure



Punkte extrahieren (Typ 1 == POINT):

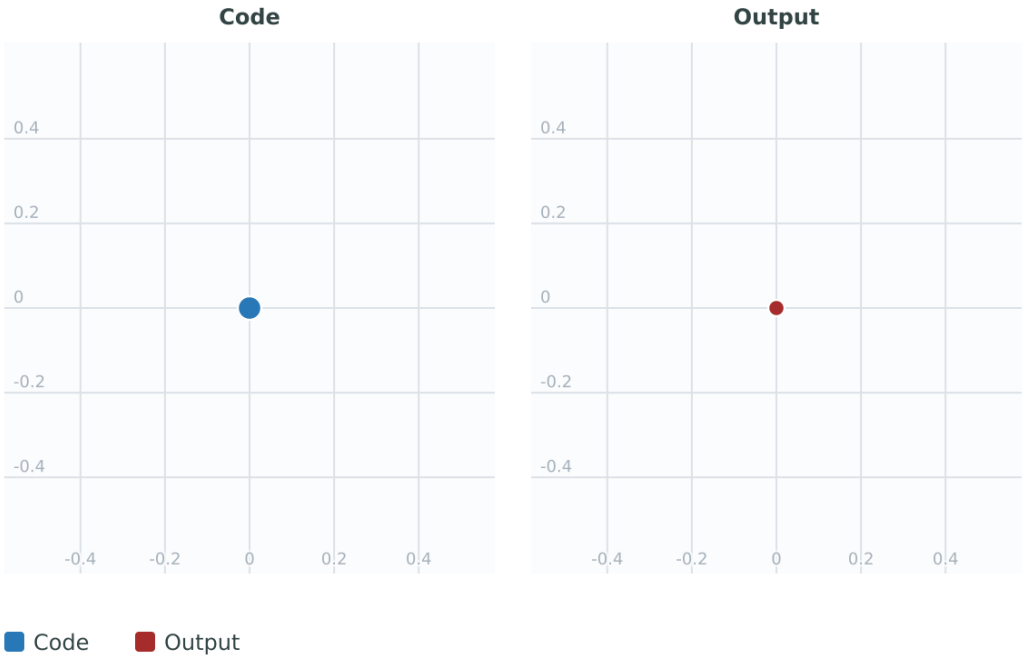
Code

```
SELECT ST_CollectionExtract('GEOMETRYCOLLECTION(GEOMETRYCOLLECTION(POINT(0 0)))',
        1);
```

Output

```
MULTIPOINT((0 0))
```

Figure



Zeilen extrahieren (Typ 2 == LINESTRING):

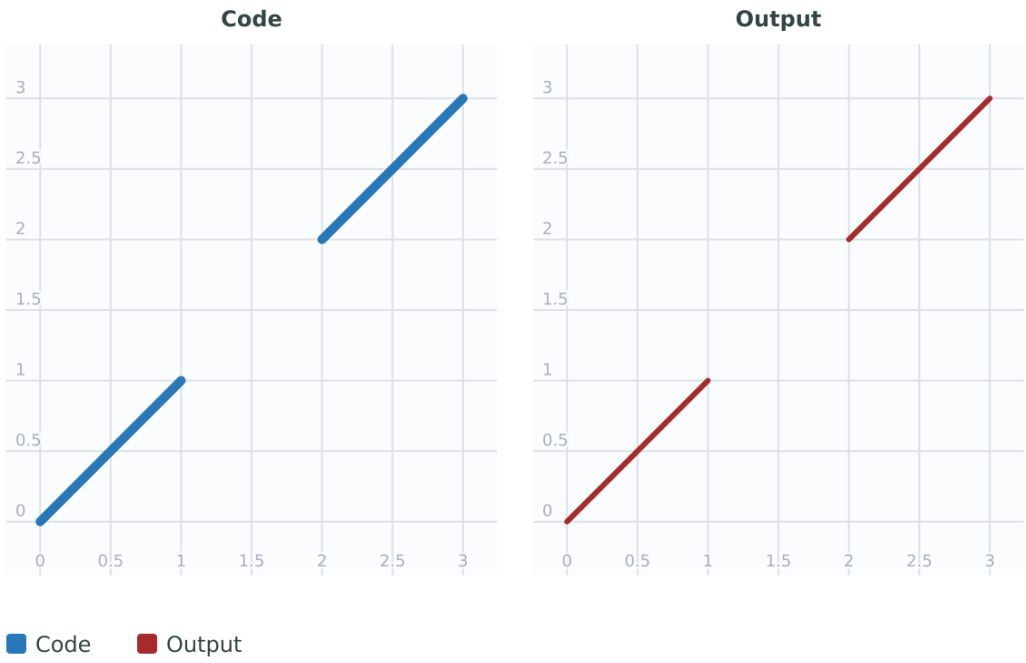
Code

```
SELECT ST_CollectionExtract('GEOMETRYCOLLECTION(GEOMETRYCOLLECTION(LINESTRING(0 0,1 1)),  ↩
    LINESTRING(2 2,3 3))',
        2);
```

Output

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

Figure



Siehe auch

ST_CollectionHomogenize, ST_Multi, ST_IsValid, ST_MakeValid

7.5.3 ST_CollectionHomogenize

ST_CollectionHomogenize — Gibt die einfachste Darstellung einer Geometriesammlung zurück.

Synopsis

geometry **ST_CollectionHomogenize**(geometry collection);

Beschreibung

Gibt bei einer Geometriesammlung die "einfachste" Darstellung des Inhalts zurück.

- Homogene (einheitliche) Sammlungen werden als die entsprechende Multi-Geometrie zurückgegeben.
- Heterogene (gemischte) Sammlungen werden zu einer einzigen GeometryCollection zusammengeführt.

- Sammlungen, die ein einzelnes atomares Element enthalten, werden als dieses Element zurückgegeben.
- Atomare Geometrien werden unverändert zurückgegeben. Bei Bedarf können diese mit ST_Multi in eine Multi-Geometrie umgewandelt werden.



Warnung
Diese Funktion stellt nicht sicher, dass das Ergebnis gültig ist. Insbesondere wird eine Sammlung, die benachbarte oder überlappende Polygone enthält, ein ungültiges MultiPolygon erzeugen. Diese Situation kann mit ST_IsValid überprüft und mit ST_MakeValid behoben werden.

Verfügbarkeit: 2.0.0

Beispiele

In eine atomare Geometrie umgewandelte Einzelelementsammlung

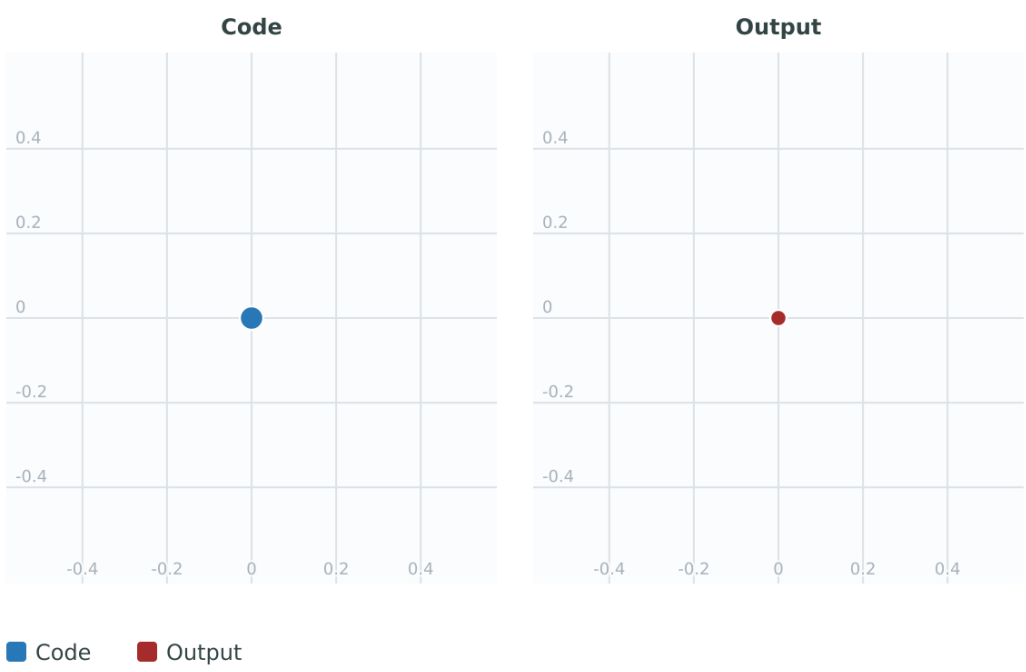
Code

```
SELECT ST_CollectionHomogenize('GEOMETRYCOLLECTION(POINT(0 0))');
```

Output

```
POINT(0 0)
```

Figure



Verschachtelte Einzelelementsammlung, die in eine atomare Geometrie umgewandelt wurde:

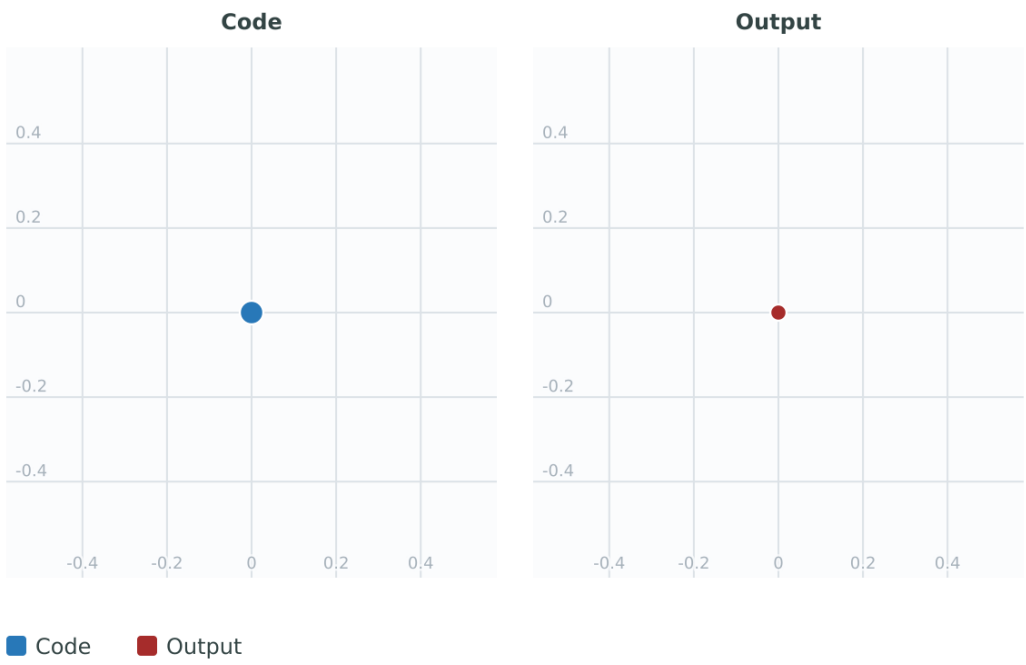
Code

```
SELECT ST_CollectionHomogenize('GEOMETRYCOLLECTION(MULTIPOINT((0 0)))');
```

Output

```
POINT(0 0)
```

Figure



Sammlung in eine Multi-Geometrie umgewandelt:

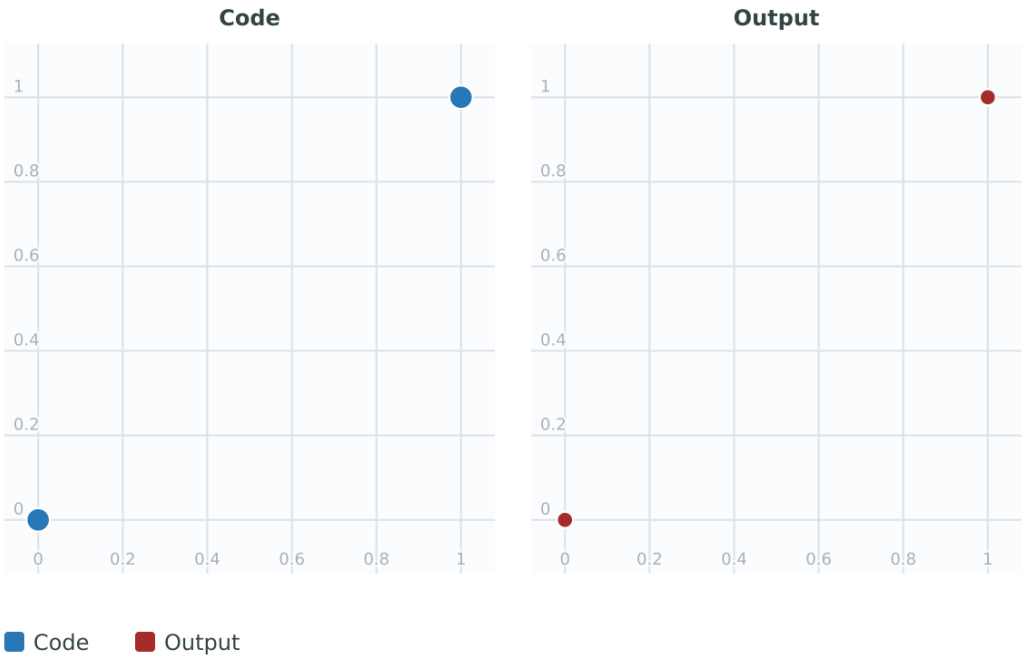
Code

```
SELECT ST_CollectionHomogenize('GEOMETRYCOLLECTION(POINT(0 0), POINT(1 1))');
```

Output

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

Figure



Verschachtelte heterogene Sammlung, die zu einer GeometryCollection abgeflacht wurde:

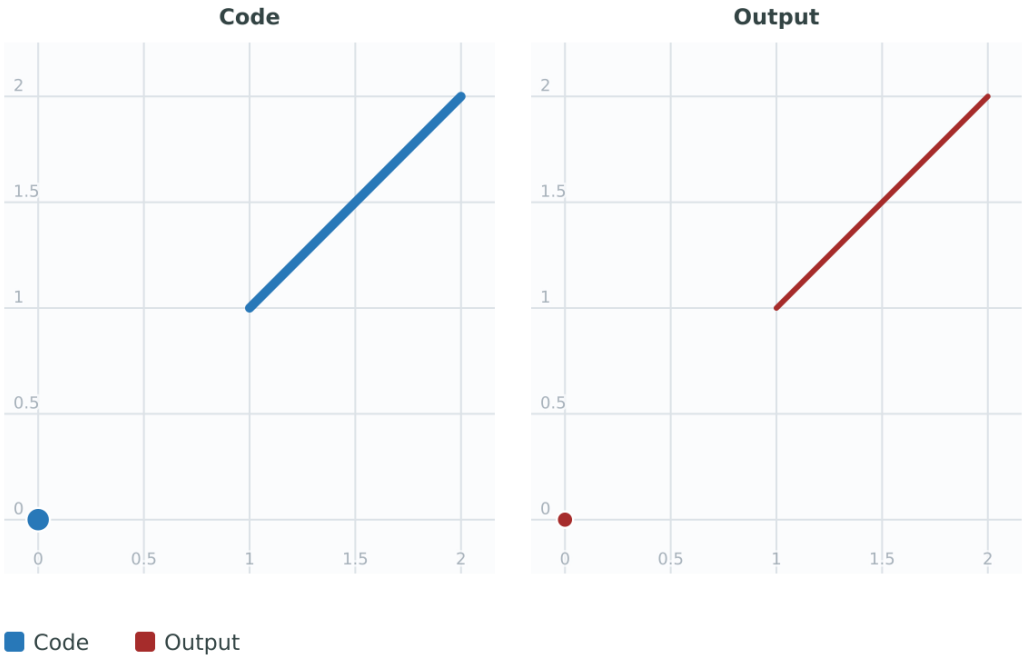
Code

```
SELECT ST_CollectionHomogenize('GEOMETRYCOLLECTION(POINT(0 0), GEOMETRYCOLLECTION(
    LINESTRING(1 1,2 2)))');
```

Output

```
GEOMETRYCOLLECTION(POINT(0 0), LINESTRING(1 1,2 2))
```

Figure



Sammlung von Polygonen, die in ein (ungültiges) MultiPolygon umgewandelt wurden:

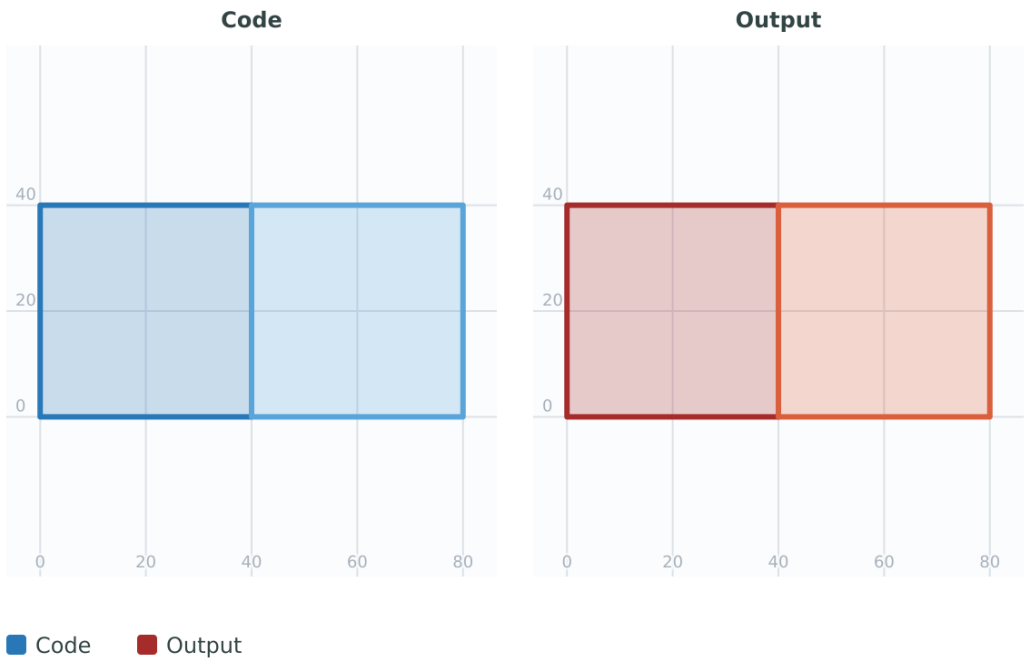
Code

```
SELECT ST_CollectionHomogenize('GEOMETRYCOLLECTION (POLYGON ((10 50,50 50,50 10,10 10,10 50) ↩
    ),POLYGON ((90 50,90 10,50 10,50 50,90 50)))');
```

Output

```
MULTIPOLYGON(((10 50,50 50,50 10,10 10,10 50)),((90 50,90 10,50 10,50 50,90 50)))
```

Figure



Siehe auch

ST_CollectionExtract, ST_Multi, ST_IsValid, ST_MakeValid

7.5.4 ST_CurveToLine

ST_CurveToLine — Konvertiert eine Geometrie mit Kurven in eine lineare Geometrie.

Synopsis

geometry **ST_CurveToLine**(geometry curveGeom, float tolerance, integer tolerance_type, integer flags);

Beschreibung

Konvertiert einen CIRCULAR STRING in einen normalen LINESTRING, ein CURVEPOLYGON in ein POLYGON und ein MULTISURFACE in ein MULTIPOLYGON. Ist nützlich zur Ausgabe an Geräten, die keine Kreisbögen unterstützen.

Wandelt eine gegebene Geometrie in eine lineare Geometrie um. Jede Kurvengeometrie und jedes Kurvensegment wird in linearer Näherung mit der gegebenen Toleranz und Optionen konvertiert (Standardmäßig 32 Segmenten pro Viertelkreis und keine Optionen).

Der Übergabewert 'tolerance_type' gibt den Toleranztyp an. Er kann die folgenden Werte annehmen:

- 0 (default): die Toleranz wird über die maximale Anzahl der Segmente pro Viertelkreis angegeben.
- 1: Die Toleranz wird als maximale Abweichung der Linie von der Kurve in der Einheit der Herkunftsdaten angegeben.
- 2: Die Toleranz entspricht dem maximalen Winkel zwischen zwei erzeugten Radien.

Der Parameter 'flags' ist ein Bitfeld mit dem Standardwert 0. Es werden folgende Bits unterstützt:

- 1: Symmetrische (orientierungsunabhängige) Ausgabe.
- 2: Erhält den Winkel, vermeidet die Winkel (Segmentlängen) bei der symmetrischen Ausgabe zu reduzieren. Hat keine Auswirkung, wenn die Symmetrie-Flag nicht aktiviert ist.

Verfügbarkeit: 1.3.0

Erweiterung: ab 2.4.0 kann die Toleranz über die 'maximale Abweichung' und den 'maximalen Winkel' angegeben werden. Die symmetrische Ausgabe wurde hinzugefügt.

Erweiterung: 3.0.0 führte eine minimale Anzahl an Segmenten pro linearisierten Bogen ein, um einem topologischen Kollaps vorzubeugen.

Diese Methode implementiert die OGC Simple Features Implementation Specification for SQL 1.1.

Diese Methode setzt die SQL/MM-Spezifikation um. SQL-MM 3: 7.1.7

Diese Funktion unterstützt 3d und lässt den Z-Index nicht fallen.

Diese Methode unterstützt kreisförmige Strings und Kurven.

Beispiele

Code

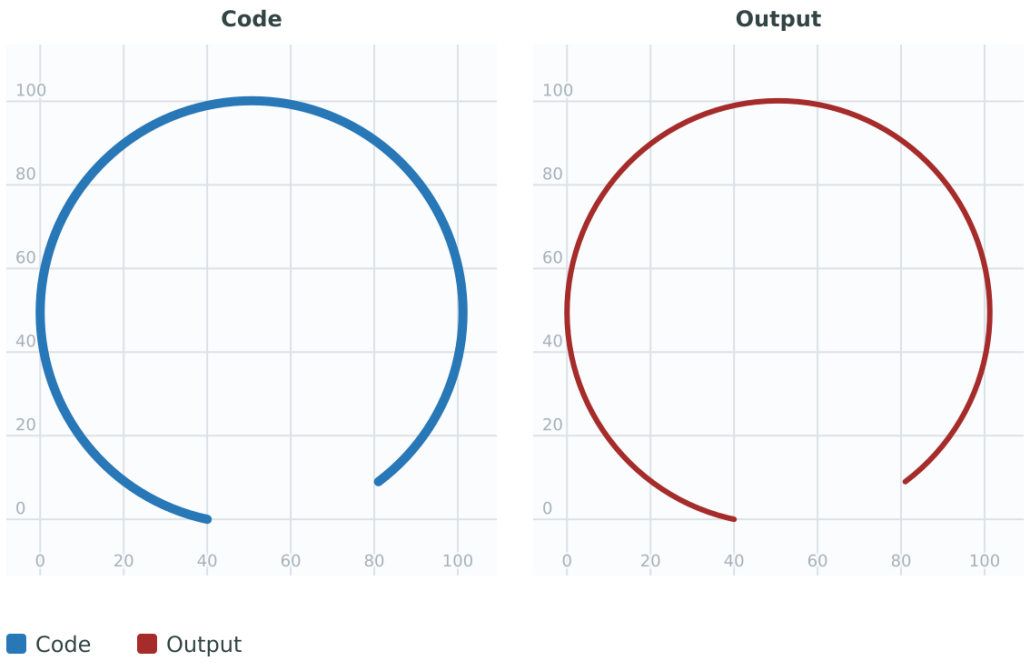
```
SELECT ST_CurveToLine('CIRCULARSTRING(220268 150415,220227 150505,220227 150406)');
```

Output

```
LINESTRING(220268 150415,220269.95064912 150416.539364228,220271.823415575 150418.17258804,220273.613787707 ↩
    150419.895736857,
220275.317452352 150421.704659462,220276.930305234 150423.594998003,220278.448460847 150425.562198489,
220279.868261823 150427.60152176,220281.186287736 150429.708054909,220282.399363347 150431.876723113,
220283.50456625 150434.10230186,220284.499233914 150436.379429536,220285.380970099 150438.702620341,220286.147650624 ↩
    150441.066277505,
220286.797428488 150443.464706771,220287.328738321 150445.892130112,220287.740300149 150448.342699654,
220288.031122486 150450.810511759,220288.200504713 150453.289621251,220288.248038775 150455.77405574,
220288.173610157 150458.257830005,220287.977398166 150460.734960415,220287.659875492 150463.199479347,
220287.221807076 150465.64544956,220286.664248262 150468.066978495,220285.988542259 150470.458232479,220285.196316903 ↩
    150472.81345077,
220284.289480732 150475.126959442,220283.270218395 150477.39318505,220282.140985384 150479.606668057,
220280.90450212 150481.762075989,220279.5637474 150483.85421628,220278.12195122 150485.87804878,
220276.582586992 150487.828697901,220274.949363179 150489.701464356,220273.226214362 150491.491836488,
220271.417291757 150493.195501133,220269.526953216 150494.808354014,220267.559752731 150496.326509628,
220265.520429459 150497.746310603,220263.41389631 150499.064336517,220261.245228106 150500.277412127,
220259.019649359 150501.38261503,220256.742521683 150502.377282695,220254.419330878 150503.259018879,
220252.055673714 150504.025699404,220249.657244448 150504.675477269,220247.229821107 150505.206787101,
220244.779251566 150505.61834893,220242.311439461 150505.909171266,220239.832329968 150506.078553494,
220237.347895479 150506.126087555,220234.864121215 150506.051658938,220232.386990804 150505.855446946,
220229.922471872 150505.537924272,220227.47650166 150505.099855856,220225.054972724 150504.542297043,
220222.663718741 150503.86659104,220220.308500449 150503.074365683,
220217.994991777 150502.167529512,220215.72876617 150501.148267175,
220213.515283163 150500.019034164,220211.35987523 150498.7825509,
220209.267734939 150497.441796181,220207.243902439 150496,
220205.293253319 150494.460635772,220203.420486864 150492.82741196,220201.630114732 150491.104263143,
220199.926450087 150489.295340538,220198.313597205 150487.405001997,220196.795441592 150485.437801511,
220195.375640616 150483.39847824,220194.057614703 150481.291945091,220192.844539092 150479.123276887,220191.739336189 ↩
    150476.89769814,
220190.744668525 150474.620570464,220189.86293234 150472.297379659,220189.096251815 150469.933722495,
220188.446473951 150467.535293229,220187.915164118 150465.107869888,220187.50360229 150462.657300346,
220187.212779953 150460.189488241,220187.043397726 150457.710378749,220186.995863664 150455.22594426,
```

```
220187.070292282 150452.742169995,220187.266504273 150450.265039585,220187.584026947 150447.800520653,
220188.022095363 150445.35455044,220188.579654177 150442.933021505,220189.25536018 150440.541767521,
220190.047585536 150438.18654923,220190.954421707 150435.873040558,220191.973684044 150433.60681495,
220193.102917055 150431.393331943,220194.339400319 150429.237924011,220195.680155039 150427.14578372,220197.12195122 ↩
    150425.12195122,
220198.661315447 150423.171302099,220200.29453926 150421.298535644,220202.017688077 150419.508163512,220203.826610682 ↩
    150417.804498867,
220205.716949223 150416.191645986,220207.684149708 150414.673490372,220209.72347298 150413.253689397,220211.830006129 ↩
    150411.935663483,
220213.998674333 150410.722587873,220216.22425308 150409.61738497,220218.501380756 150408.622717305,220220.824571561 ↩
    150407.740981121,
220223.188228725 150406.974300596,220225.586657991 150406.324522731,220227 150406)
```

Figure



This example uses 3D input.

Code

```
SELECT ST_CurveToLine('CIRCULARSTRING(220268 150415 1,220227 150505 2,220227 150406 3)');
```

Output

```
LINESTRING(220268 150415 1,220269.95064912 150416.539364228 1.0181172856673,
220271.823415575 150418.17258804 1.03623457133459,220273.613787707 150419.895736857 1.05435185700189,....AD INFINITUM ↩
    ....
   220225.586657991 150406.324522731 1.32611114201132,220227 150406 3)
```

Use only 2 segments to approximate quarter circle.

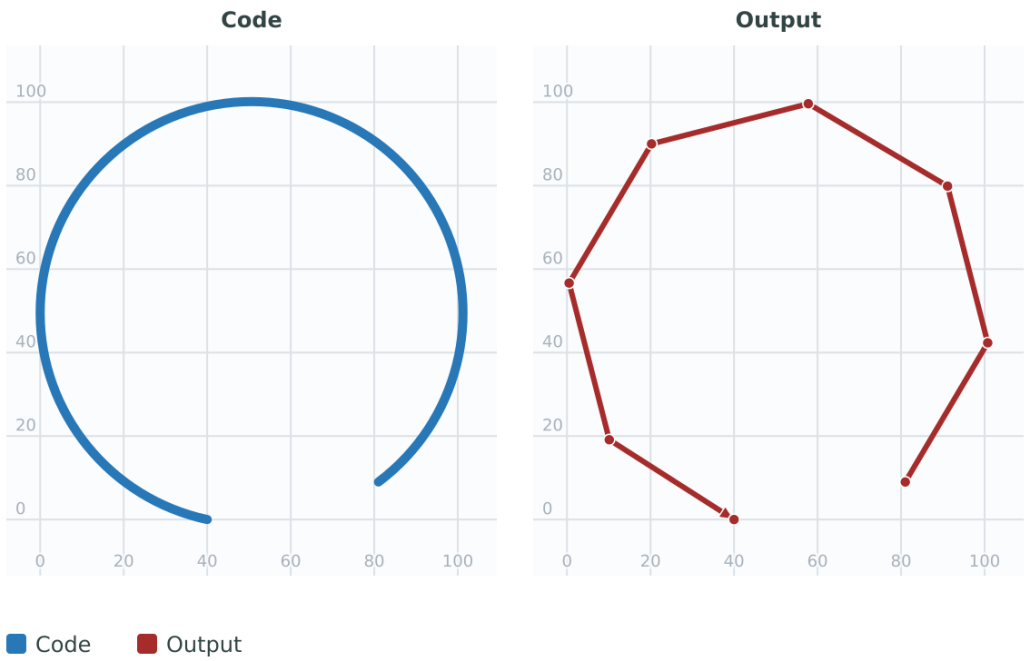
Code

```
SELECT ST_CurveToLine('CIRCULARSTRING(220268 150415,220227 150505,220227 150406)', 2);
```

Output

```
LINESTRING(220268 150415,220287.740300149 150448.342699654,220278.12195122 150485.87804878,
220244.779251566 150505.61834893,220207.243902439 150496,220187.50360229 150462.657300346,
220197.12195122 150425.12195122,220227 150406)
```

Figure



Ensure the approximated line is no more than 20 units away from the original curve, and make the result direction-neutral.

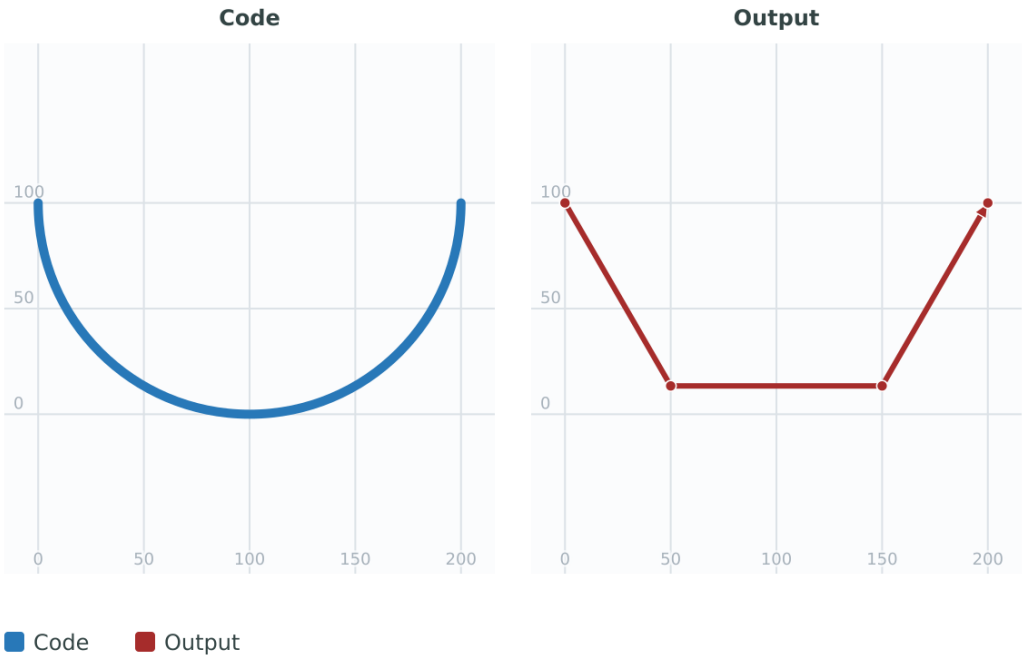
Code

```
SELECT ST_CurveToLine(
    'CIRCULARSTRING(0 0,100 -100,200 0)'::geometry,
    20,
    1,
    1
  );
```

Output

```
LINESTRING(0 0,49.99999999999996 -86.60254037844383,149.99999999999994 -86.6025403784439,200 0)
```

Figure



Siehe auch

ST_LineToCurve

7.5.5 ST_Scroll

ST_Scroll — Startpunkt eines geschlossenen LineStrings ändern.

Synopsis

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

Beschreibung

Ändert den Start-/Endpunkt eines geschlossenen LineStrings auf den angegebenen Scheitelpunkt .

Verfügbarkeit: 3.2.0

Diese Funktion unterstützt 3d und lässt den Z-Index nicht fallen.

Diese Funktion unterstützt M-Koordinaten.

Beispiele

Make a closed line start at its 3rd vertex.

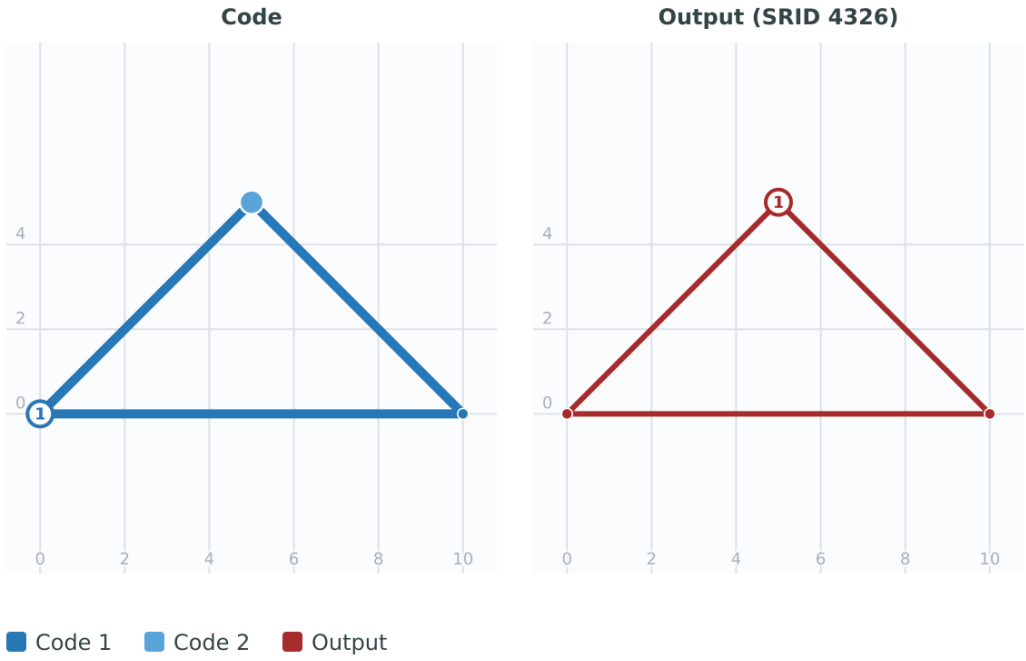
Code

```
SELECT ST_Scroll('SRID=4326;LINESTRING(0 0 0 1,10 0 2 0,5 5 4 2,0 0 0 1)', 'POINT(5 5 4 2)') ↩
    ;
```

Output

```
SRID=4326;LINESTRING(5 5 4 2,0 0 0 1,10 0 2 0,5 5 4 2)
```

Figure



Siehe auch

ST_Normalize

7.5.6 ST_FlipCoordinates

ST_FlipCoordinates — Gibt eine Version einer Geometrie mit gespiegelter X- und Y-Achse zurück.

Synopsis

geometry **ST_FlipCoordinates**(geometry geom);

Beschreibung

Liefert eine Version der angegebenen Geometrie mit gespiegelter X- und Y-Achse. Nützlich zum Fixieren von Geometrien, die Koordinaten enthalten, die als Breitengrad/Längengrad (Y,X) ausgedrückt sind.

Verfügbarkeit: 2.0.0

Diese Methode unterstützt kreisförmige Strings und Kurven.

Diese Funktion unterstützt 3d und lässt den Z-Index nicht fallen.

Diese Funktion unterstützt M-Koordinaten.

Diese Funktion unterstützt polyedrische Flächen.

Diese Funktion unterstützt Dreiecke und dreieckige unregelmäßige Netzoberflächen (TIN).

Beispiel

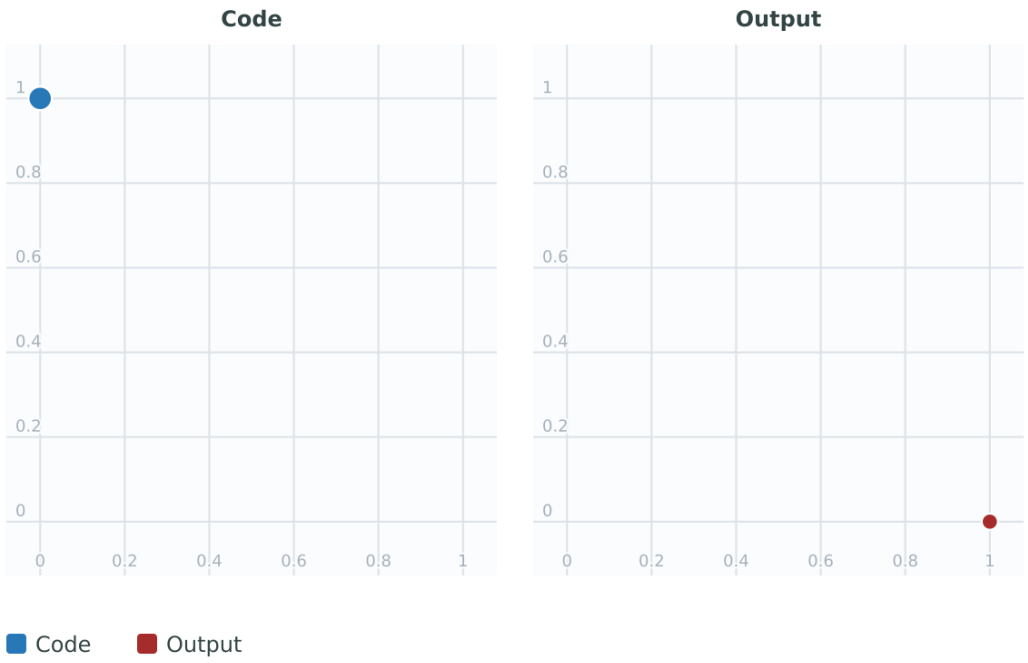
Code

```
SELECT ST_FlipCoordinates(GeomFromEWKT('POINT(1 2)'));
```

Output

```
POINT(2 1)
```

Figure



Siehe auch

ST_SwapOrdinates

7.5.7 ST_Force2D

ST_Force2D — Die Geometrien in einen "2-dimensionalen Modus" zwingen.

Synopsis

geometry **ST_Force2D**(geometry geomA);

Beschreibung

Zwingt die Geometrien in einen "2-dimensionalen Modus", sodass in der Ausgabe nur die X- und Y-Koordinaten dargestellt werden. Nützlich um eine OGC-konforme Ausgabe zu erhalten (da OGC nur 2-D Geometrien spezifiziert).

Erweiterung: Mit 2.0.0 wurde die Unterstützung für polyedrische Oberflächen eingeführt.

Änderung: 2.1.0. Bis zu 2.0.x wurde diese Funktion mit ST_Force_2D bezeichnet.

Diese Methode unterstützt kreisförmige Strings und Kurven.

Diese Funktion unterstützt polyedrische Flächen.

Diese Funktion unterstützt 3d und lässt den Z-Index nicht fallen.

Beispiele

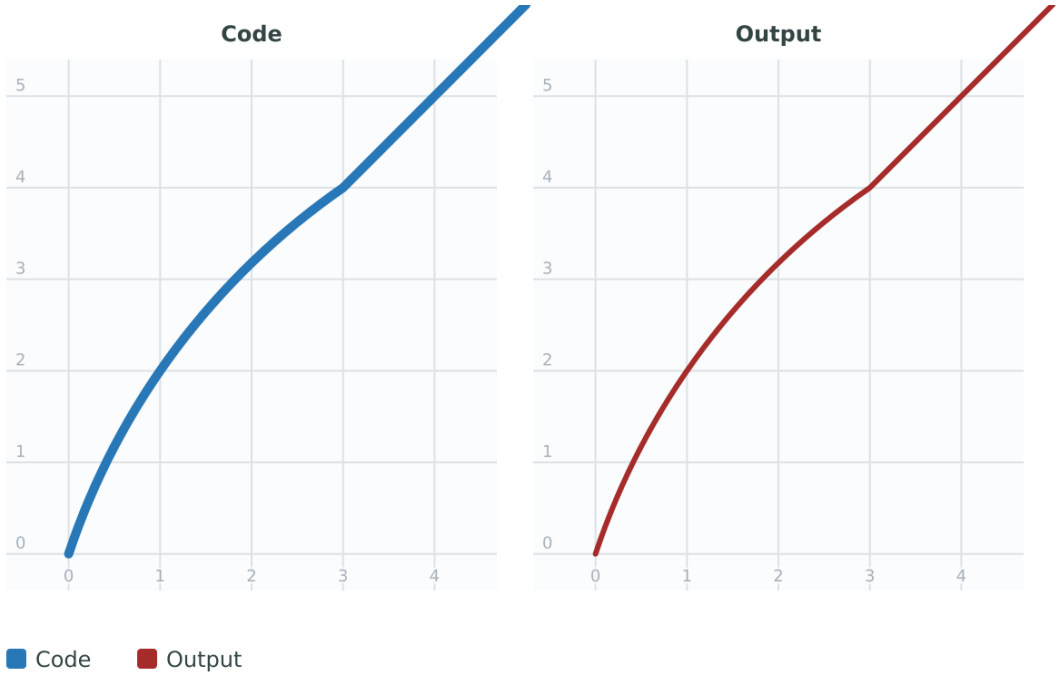
Code

```
SELECT ST_Force2D('CIRCULARSTRING(1 1 2,2 3 2,4 5 2,6 7 2,5 6 2)');
```

Output

```
CIRCULARSTRING(1 1,2 3,4 5,6 7,5 6)
```

Figure



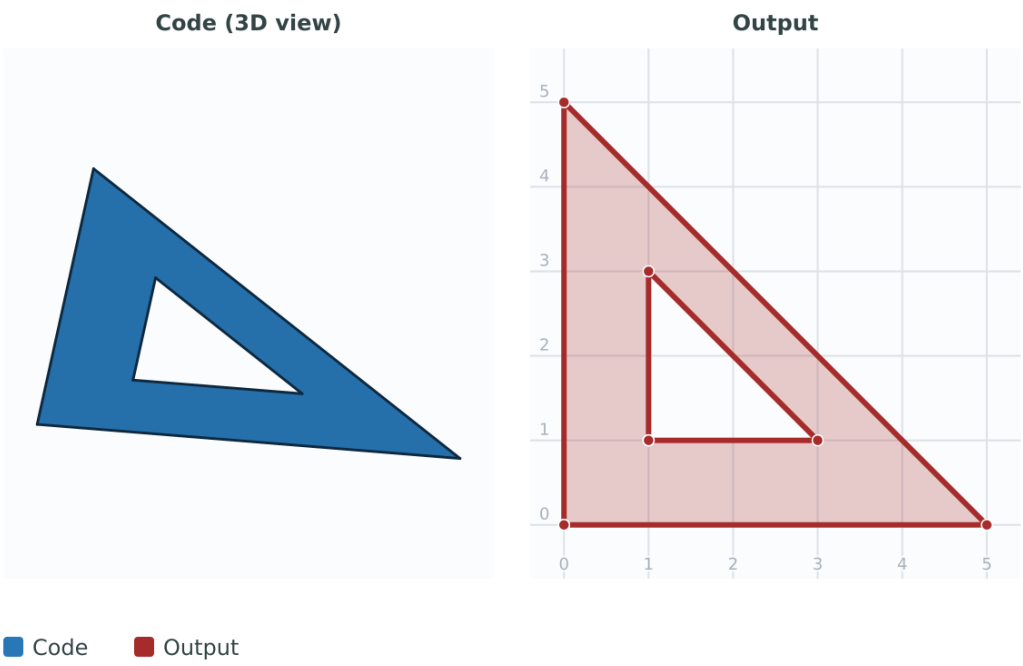
Code

```
SELECT ST_Force2D('POLYGON((0 0 2,0 5 2,5 0 2,0 0 2),(1 1 2,3 1 2,1 3 2,1 1 2))');
```

Output

```
POLYGON((0 0,0 5,5 0,0 0),(1 1,3 1,1 3,1 1))
```

Figure



Siehe auch

ST_Force3D

7.5.8 ST_Force3D

ST_Force3D — Zwingt die Geometrien in einen XYZ Modus. Dies ist ein Alias für ST_Force3DZ.

Synopsis

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

Beschreibung

Zwingt die Geometrien in den XYZ-Modus. Dies ist ein Alias für ST_Force3DZ. Wenn eine Geometrie keine Z-Komponente hat, wird ein *Z-Wert* Z-Koordinate angehängt.

Erweiterung: Mit 2.0.0 wurde die Unterstützung für polyedrische Oberflächen eingeführt.

Änderung: 2.1.0. Bis zu 2.0.x wurde diese Funktion mit ST_Force_3D bezeichnet.

Geändert: 3.1.0. Unterstützung für die Angabe eines Z-Wertes ungleich Null wurde zugefügt.

Diese Funktion unterstützt polyedrische Flächen.

Diese Methode unterstützt kreisförmige Strings und Kurven.

Diese Funktion unterstützt 3d und lässt den Z-Index nicht fallen.

Beispiele

Nothing happens to an already 3D geometry.

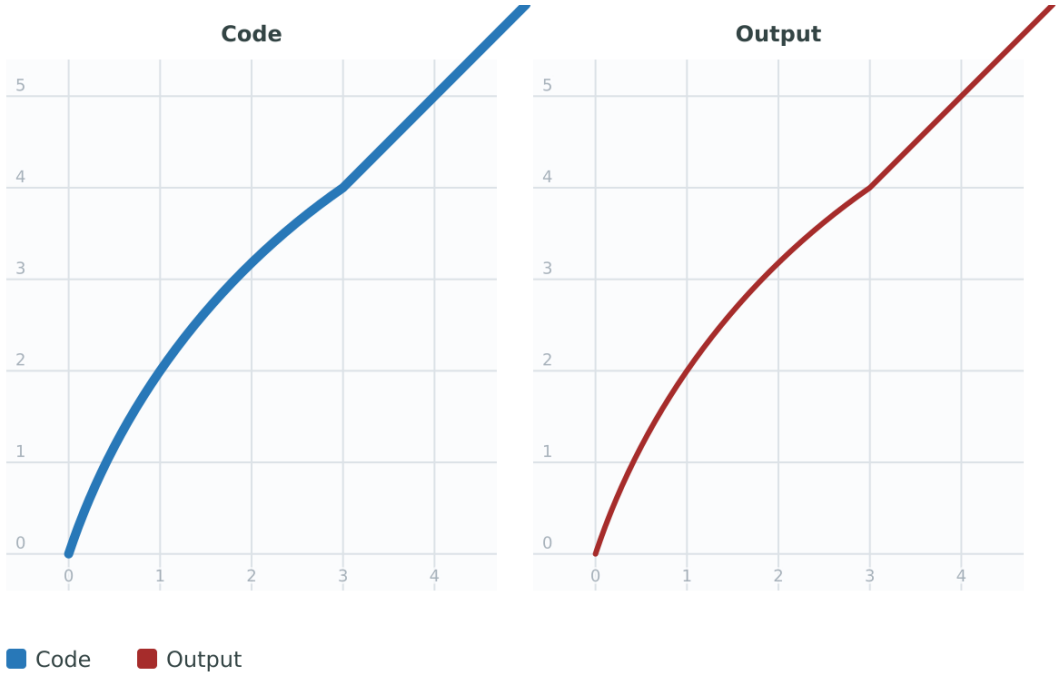
Code

```
SELECT ST_Force3D('CIRCULARSTRING(1 1 2,2 3 2,4 5 2,6 7 2,5 6 2)');
```

Output

```
CIRCULARSTRING(1 1 2,2 3 2,4 5 2,6 7 2,5 6 2)
```

Figure



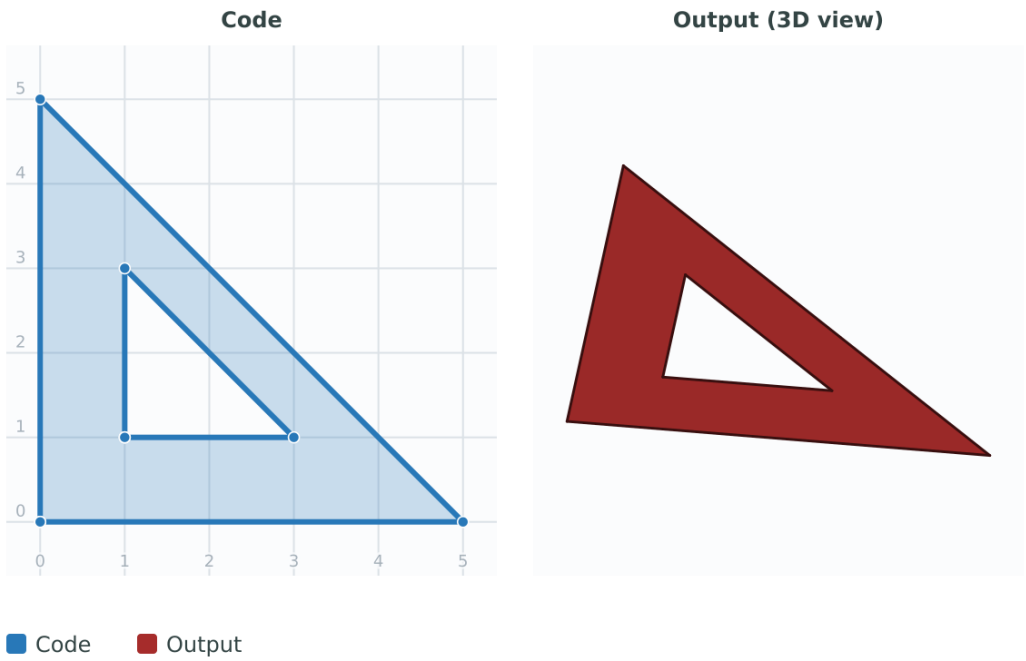
Code

```
SELECT ST_Force3D('POLYGON((0 0,0 5,5 0,0 0),(1 1,3 1,1 3,1 1))');
```

Output

```
POLYGON((0 0 0,0 5 0,5 0 0,0 0 0),(1 1 0,3 1 0,1 3 0,1 1 0))
```

Figure



Siehe auch

ST_AsEWKT, ST_Force2D, ST_Force3DM, ST_Force3DZ

7.5.9 ST_Force3DZ

ST_Force3DZ — Zwingt die Geometrien in einen XYZ Modus.

Synopsis

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

Beschreibung

Zwingt die Geometrien in den XYZ-Modus. Wenn eine Geometrie keine Z-Komponente hat, wird ein `Z-Wert` Z-Koordinate angehängt.

Erweiterung: Mit 2.0.0 wurde die Unterstützung für polyedrische Oberflächen eingeführt.

Änderung: 2.1.0. Bis zu 2.0.x wurde diese Funktion mit ST_Force_3DZ bezeichnet.

Geändert: 3.1.0. Unterstützung für die Angabe eines Z-Wertes ungleich Null wurde zugefügt.

Diese Funktion unterstützt polyedrische Flächen.

Diese Funktion unterstützt 3d und lässt den Z-Index nicht fallen.

Diese Methode unterstützt kreisförmige Strings und Kurven.

Beispiele

Nothing happens to an already 3D geometry.

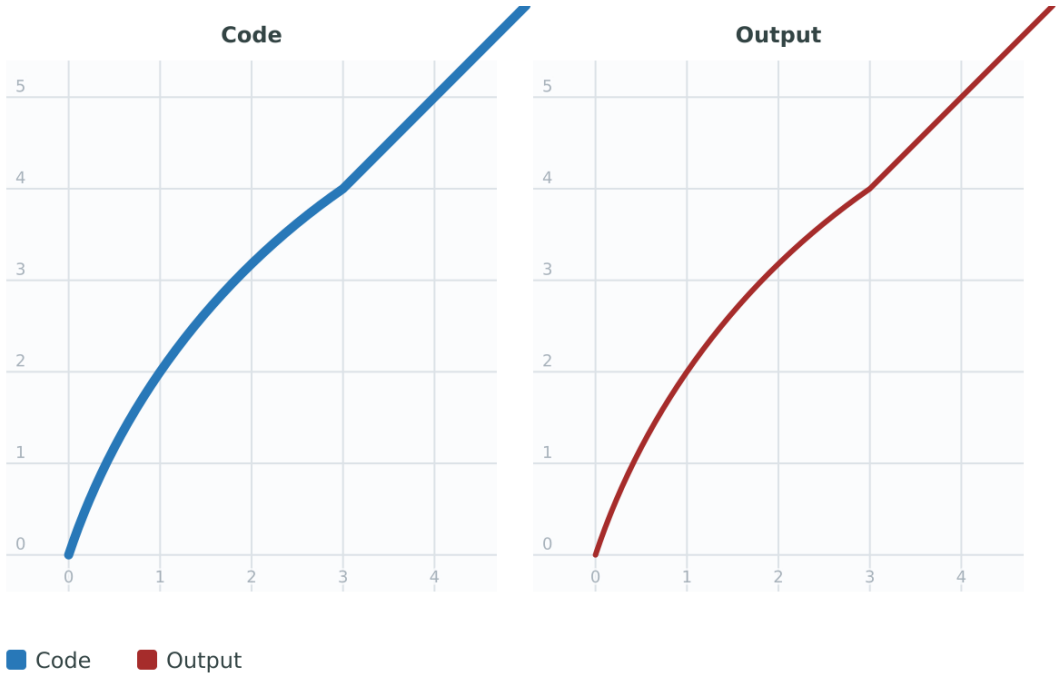
Code

```
SELECT ST_Force3DZ('CIRCULARSTRING(1 1 2,2 3 2,4 5 2,6 7 2,5 6 2)');
```

Output

```
CIRCULARSTRING(1 1 2,2 3 2,4 5 2,6 7 2,5 6 2)
```

Figure



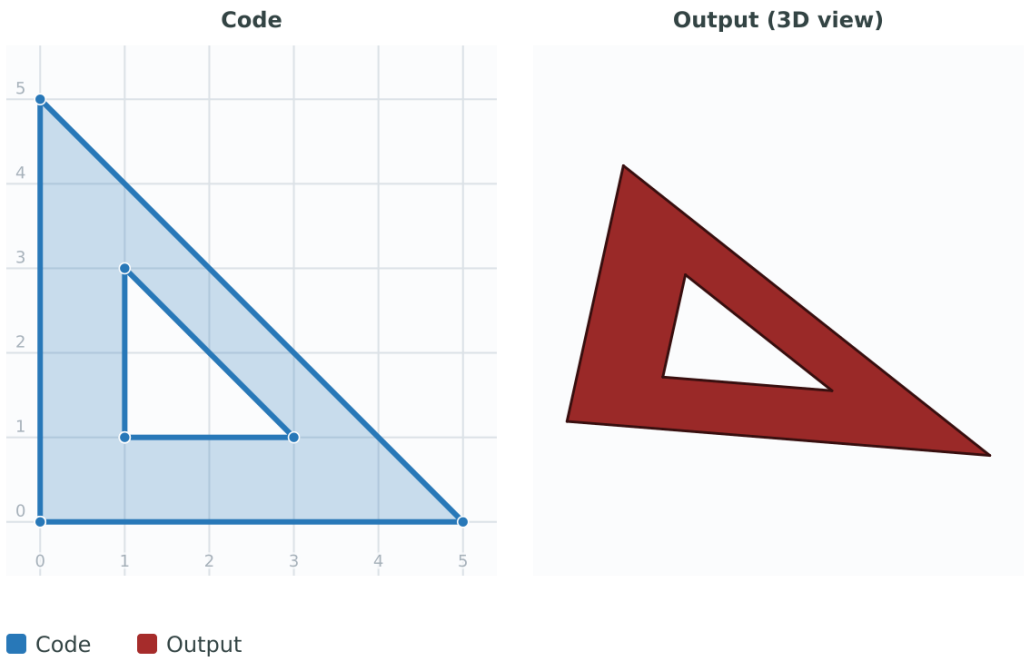
Code

```
SELECT ST_Force3DZ('POLYGON((0 0,0 5,5 0,0 0),(1 1,3 1,1 3,1 1))');
```

Output

```
POLYGON((0 0 0,0 5 0,5 0 0,0 0 0),(1 1 0,3 1 0,1 3 0,1 1 0))
```

Figure



Siehe auch

ST_AsEWKT, ST_Force2D, ST_Force3DM, ST_Force3D

7.5.10 ST_Force3DM

ST_Force3DM — Zwingt die Geometrien in einen XYM Modus.

Synopsis

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

Beschreibung

Zwingt die Geometrien in den XYM-Modus. Wenn eine Geometrie keine M-Komponente hat, wird ein `M-Wert` M-Koordinate angehängt. Wenn sie eine Z-Komponente hat, wird Z entfernt.

Änderung: 2.1.0. Bis zu 2.0.x wurde diese Funktion mit ST_Force_3DM bezeichnet.

Geändert: 3.1.0. Unterstützung für die Angabe eines M-Wertes ungleich Null wurde hinzugefügt.

Diese Methode unterstützt kreisförmige Strings und Kurven.

Beispiele

Nothing happens to an already 3D geometry.

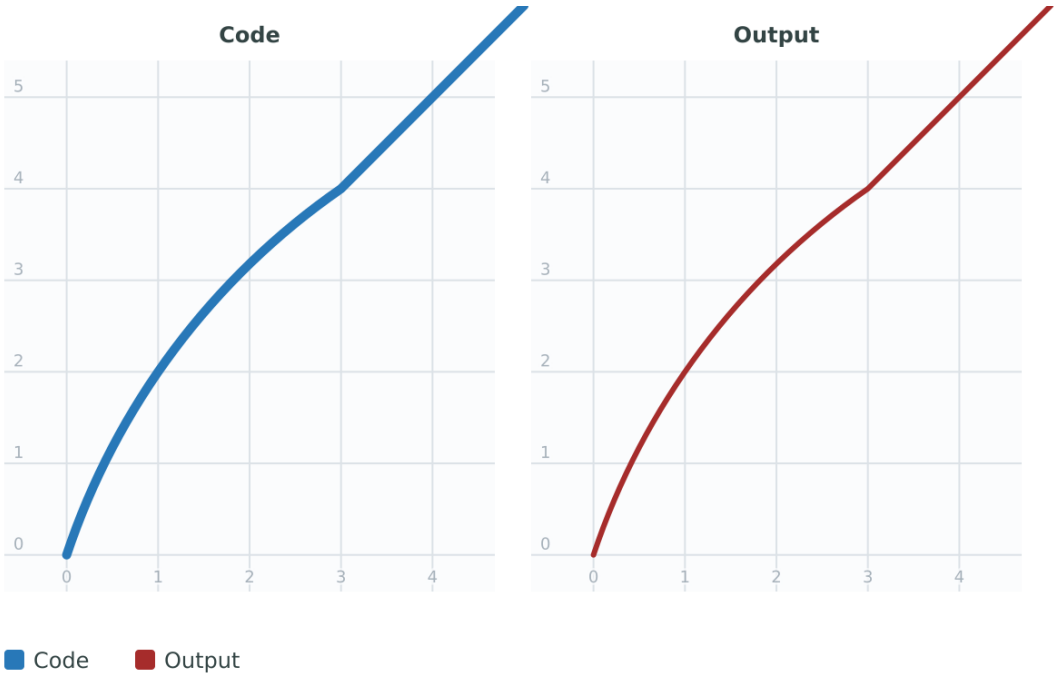
Code

```
SELECT ST_Force3DM('CIRCULARSTRING(1 1 2,2 3 2,4 5 2,6 7 2,5 6 2)');
```

Output

```
CIRCULARSTRINGM(1 1 0,2 3 0,4 5 0,6 7 0,5 6 0)
```

Figure



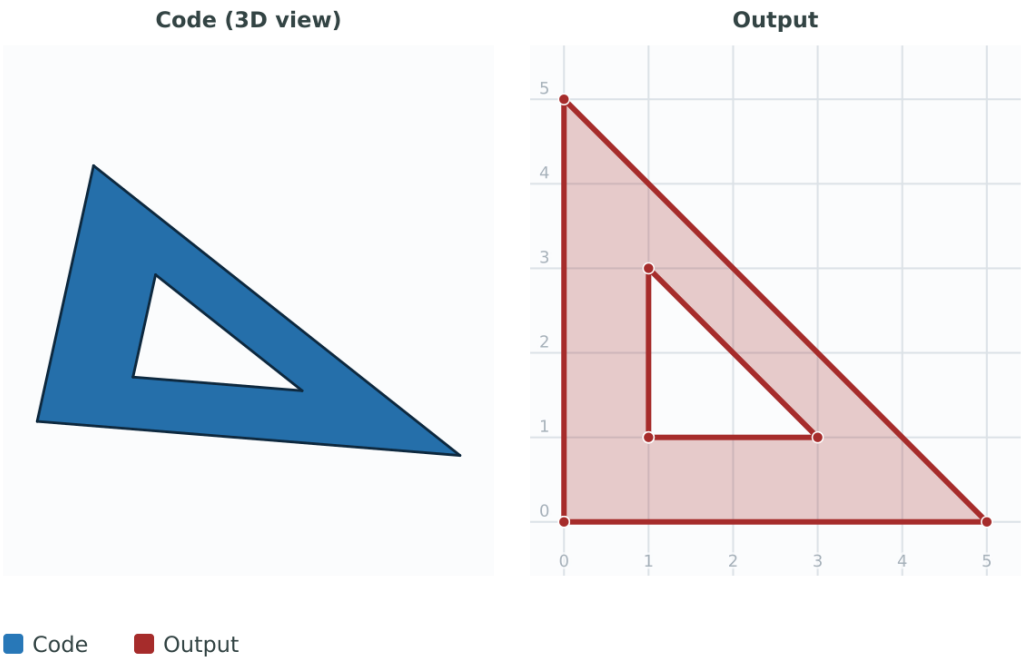
Code

```
SELECT ST_Force3DM('POLYGON((0 0 1,0 5 1,5 0 1,0 0 1),(1 1 1,3 1 1,1 3 1,1 1 1))');
```

Output

```
POLYGONM((0 0 0,0 5 0,5 0 0,0 0 0),(1 1 0,3 1 0,1 3 0,1 1 0))
```

Figure



Siehe auch

ST_AsEWKT, ST_Force2D, ST_Force3DM, ST_Force3D, ST_GeomFromEWKT

7.5.11 ST_Force4D

ST_Force4D — Zwingt die Geometrien in einen XYZM Modus.

Synopsis

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

Beschreibung

Zwingt die Geometrien in den XYZM-Modus. `Zvalue` und `Mvalue` wird für fehlende Z- bzw. M-Dimensionen angehängt.

Änderung: 2.1.0. Bis zu 2.0.x wurde diese Funktion mit ST_Force_4D bezeichnet.

Geändert: 3.1.0. Unterstützung für die Angabe von Z- und M-Werten ungleich Null wurde hinzugefügt.

Diese Funktion unterstützt 3d und lässt den Z-Index nicht fallen.

Diese Methode unterstützt kreisförmige Strings und Kurven.

Beispiele

Nothing happens to an already 3D geometry.

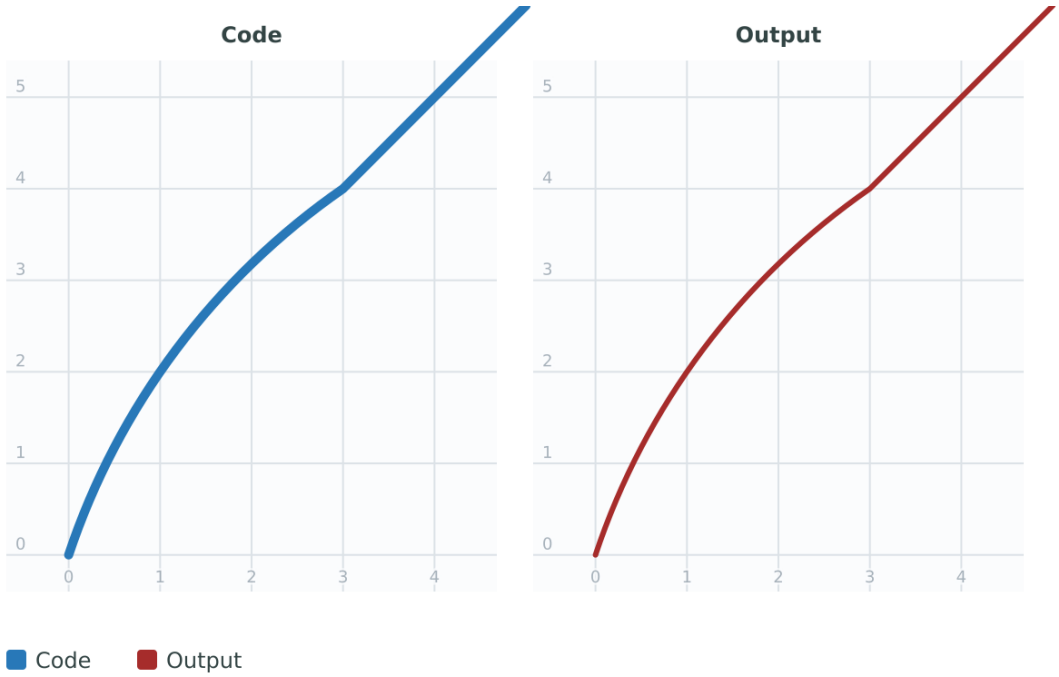
Code

```
SELECT ST_Force4D('CIRCULARSTRING(1 1 2,2 3 2,4 5 2,6 7 2,5 6 2)');
```

Output

```
CIRCULARSTRING(1 1 2 0,2 3 2 0,4 5 2 0,6 7 2 0,5 6 2 0)
```

Figure



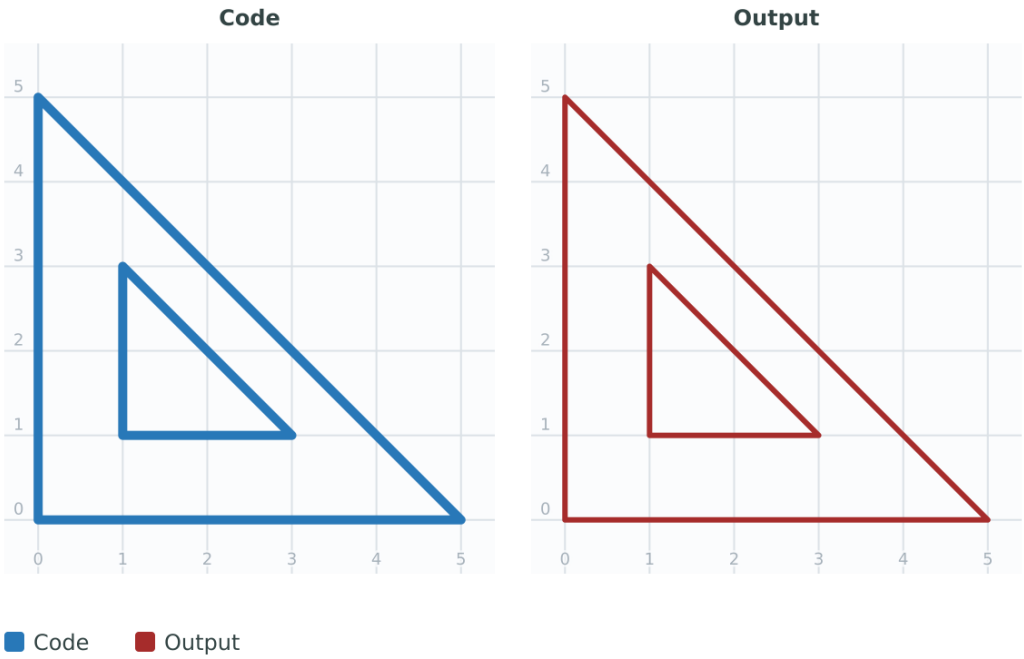
Code

```
SELECT ST_Force4D('MULTILINESTRINGM((0 0 1,0 5 2,5 0 3,0 0 4),(1 1 1,3 1 1,1 3 1,1 1 1))');
```

Output

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

Figure



Siehe auch

ST_AsEWKT, ST_Force2D, ST_Force3DM, ST_Force3D

7.5.12 ST_ForceCollection

ST_ForceCollection — Wandelt eine Geometrie in eine GEOMETRYCOLLECTION um.

Synopsis

geometry **ST_ForceCollection**(geometry geomA);

Beschreibung

Wandelt eine Geometrie in eine GEOMETRYCOLLECTION um. Nützlich um eine WKB-Darstellung zu vereinfachen.

Erweiterung: Mit 2.0.0 wurde die Unterstützung für polyedrische Oberflächen eingeführt.

Verfügbarkeit: 1.2.2, Vor 1.3.4 ist diese Funktion bei CURVES abgestürzt. Dies wurde mit 1.3.4+ behoben

Änderung: 2.1.0. Bis zu 2.0.x wurde diese Funktion mit ST_Force_Collection bezeichnet.

Diese Funktion unterstützt polyedrische Flächen.

Diese Funktion unterstützt 3d und lässt den Z-Index nicht fallen.

Diese Methode unterstützt kreisförmige Strings und Kurven.

Beispiele

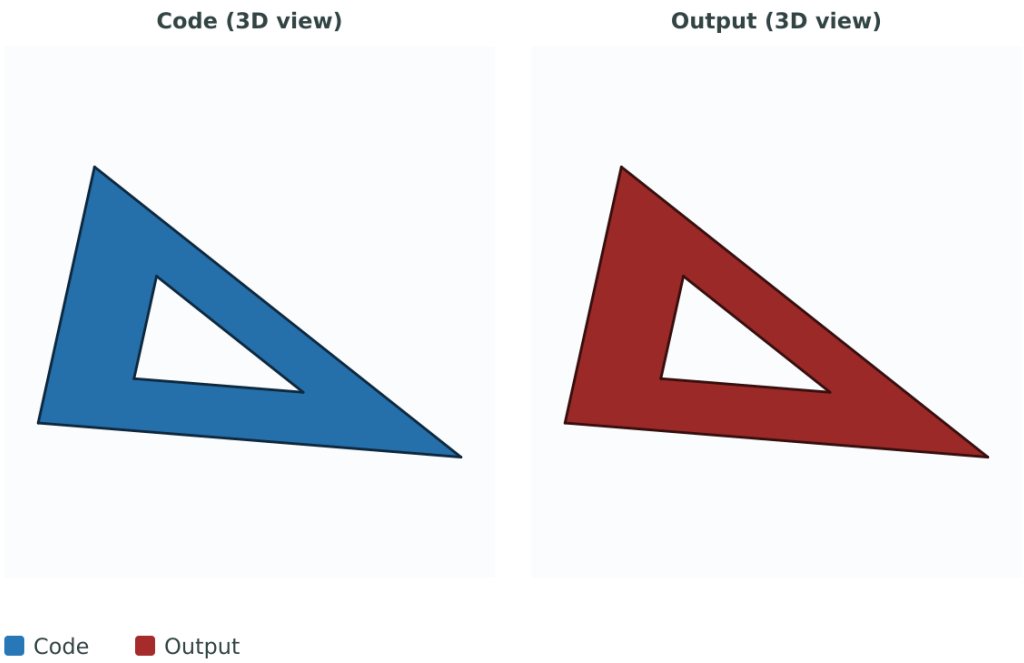
Code

```
SELECT ST_ForceCollection('POLYGON((0 0 1,0 5 1,5 0 1,0 0 1),(1 1 1,3 1 1,1 3 1,1 1 1))');
```

Output

```
GEOMETRYCOLLECTION(POLYGON((0 0 1,0 5 1,5 0 1,0 0 1),(1 1 1,3 1 1,1 3 1,1 1 1)))
```

Figure



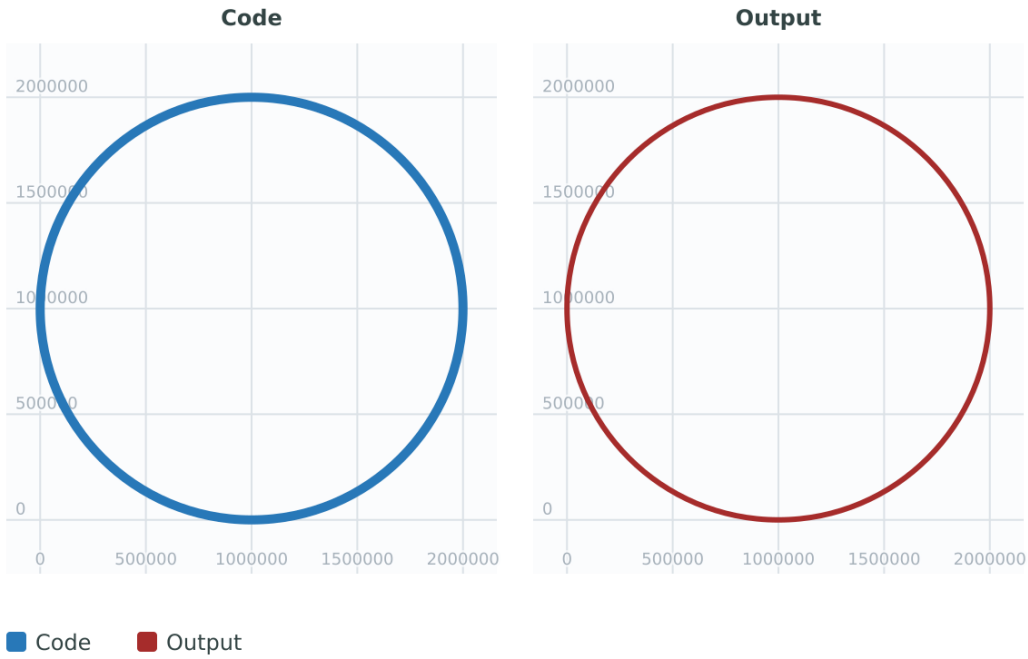
Code

```
SELECT ST_ForceCollection('CIRCULARSTRING(220227 150406,2220227 150407,220227 150406)');
```

Output

```
GEOMETRYCOLLECTION(CIRCULARSTRING(220227 150406,2220227 150407,220227 150406))
```

Figure



This example uses a PolyhedralSurface.

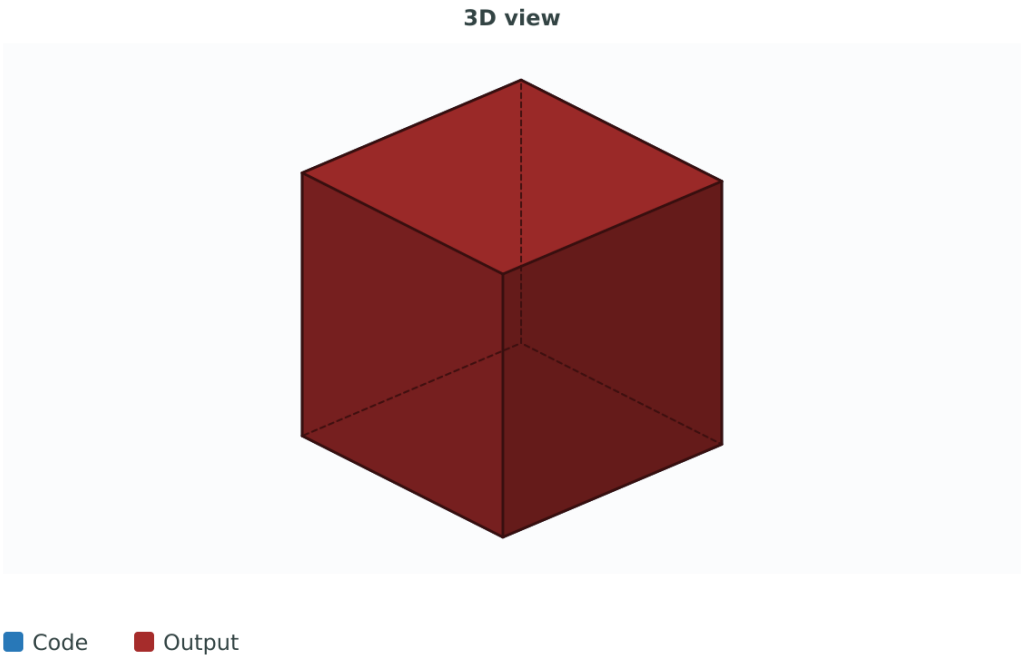
Code

```
SELECT ST_ForceCollection('POLYHEDRALSURFACE(((0 0 0,0 0 1,0 1 1,0 1 0,0 0 0)),
  ((0 0 0,0 1 0,1 1 0,1 0 0,0 0 0)),
  ((0 0 0,1 0 0,1 0 1,0 0 1,0 0 0)),
  ((1 1 0,1 1 1,1 0 1,1 0 0,1 1 0)),
  ((0 1 0,0 1 1,1 1 1,1 1 0,0 1 0)),
  ((0 0 1,1 0 1,1 1 1,0 1 1,0 0 1)))');
```

Output

```
GEOMETRYCOLLECTION(
  POLYGON((0 0 0,0 0 1,0 1 1,0 1 0,0 0 0)),
  POLYGON((0 0 0,0 1 0,1 1 0,1 0 0,0 0 0)),
  POLYGON((0 0 0,1 0 0,1 0 1,0 0 1,0 0 0)),
  POLYGON((1 1 0,1 1 1,1 0 1,1 0 0,1 1 0)),
  POLYGON((0 1 0,0 1 1,1 1 1,1 1 0,0 1 0)),
  POLYGON((0 0 1,1 0 1,1 1 1,0 1 1,0 0 1))
)
```

Figure



Siehe auch

ST_AsEWKT, ST_Force2D, ST_Force3DM, ST_Force3D, ST_GeomFromEWKT

7.5.13 ST_ForceCurve

ST_ForceCurve — Wandelt einen geometrischen in einen Kurven Datentyp um, soweit anwendbar.

Synopsis

geometry **ST_ForceCurve**(geometry g);

Beschreibung

Wandelt eine Geometrie in eine Kurvendarstellung um, soweit anwendbar: Linien werden CompundCurves, MultiLines werden MultiCurves, Polygone werden zu CurvePolygons, Multipolygons werden MultiSurfaces. Wenn die Geometrie bereits in Kurvendarstellung vorliegt, wird sie unverändert zurückgegeben.

Verfügbarkeit: 2.2.0

Diese Funktion unterstützt 3d und lässt den Z-Index nicht fallen.

Diese Methode unterstützt kreisförmige Strings und Kurven.

Beispiele

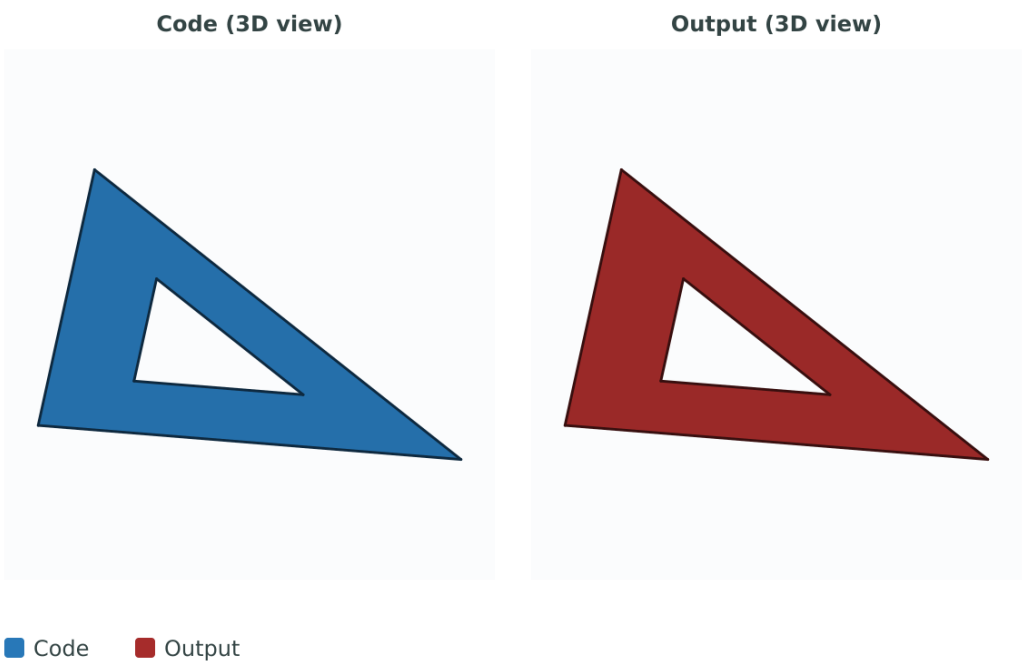
Code

```
SELECT ST_ForceCurve('POLYGON((0 0 2,5 0 2,0 5 2,0 0 2),(1 1 2,1 3 2,3 1 2,1 1 2))':: ↩
    geometry
  );
```

Output

```
CURVEPOLYGON Z ((0 0 2,5 0 2,0 5 2,0 0 2),(1 1 2,1 3 2,3 1 2,1 1 2))
```

Figure



Siehe auch

ST_LineToCurve

7.5.14 ST_ForcePolygonCCW

ST_ForcePolygonCCW — Richtet alle äußeren Ringe gegen den Uhrzeigersinn und alle inneren Ringe mit dem Uhrzeigersinn aus.

Synopsis

geometry **ST_ForcePolygonCCW**(geometry geom);

Beschreibung

Zwingt (Multi)Polygone, den äusseren Ring gegen den Uhrzeigersinn und die inneren Ringe im Uhrzeigersinn zu orientieren. Andere Geometrien werden unverändert zurückgegeben.

Verfügbarkeit: 2.4.0

Diese Funktion unterstützt 3d und lässt den Z-Index nicht fallen.

Diese Funktion unterstützt M-Koordinaten.

Beispiele

Force a polygon and its hole to use counter-clockwise exterior rings and clockwise interior rings.

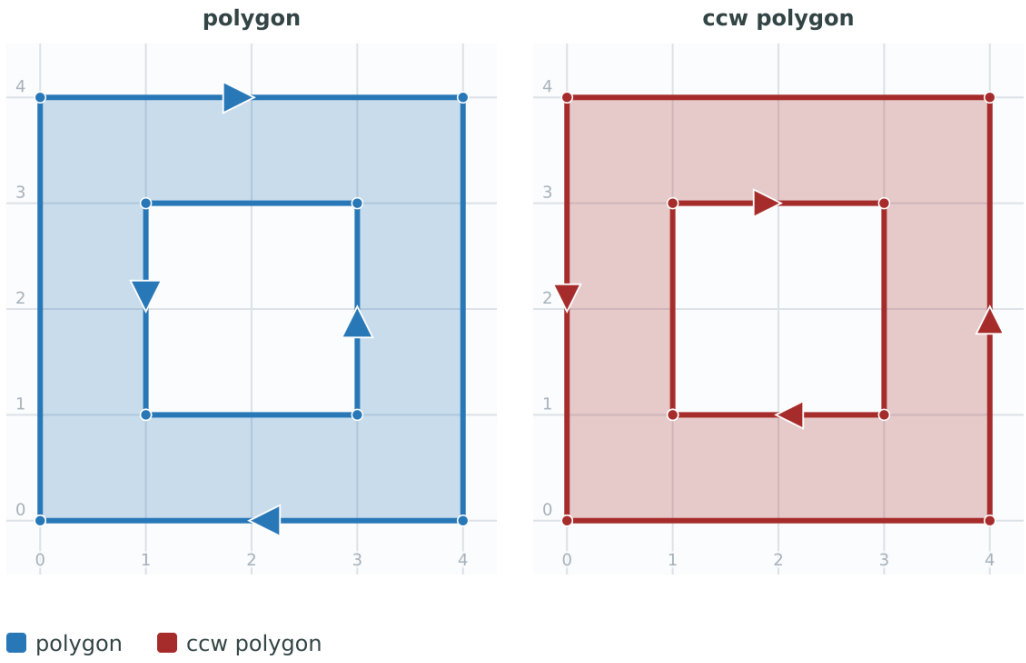
Code

```
WITH data AS (
  SELECT 'POLYGON((0 0,0 4,4 4,4 0,0 0),
                  (1 1,3 1,3 3,1 3,1 1))'::geometry AS geom
)
SELECT geom AS input_polygon,
       ST_ForcePolygonCCW(geom) AS ccw_polygon
FROM data;
```

Output

```
POLYGON((0 0,0 4,4 4,4 0,0 0),(1 1,3 1,3 3,1 3,1 1)) | POLYGON((0 0,4 0,4 4,0 4,0 0),(1 1,1 3,3 3,3 1,1 1))
```

Figure



Siehe auch

ST_ForcePolygonCW, ST_IsPolygonCCW, ST_IsPolygonCW

7.5.15 ST_ForcePolygonCW

ST_ForcePolygonCW — Richtet alle äußeren Ringe im Uhrzeigersinn und alle inneren Ringe gegen den Uhrzeigersinn aus.

Synopsis

geometry **ST_ForcePolygonCW**(geometry geom);

Beschreibung

Zwingt (Multi)Polygone, den äusseren Ring im Uhrzeigersinn und die inneren Ringe gegen den Uhrzeigersinn zu orientieren. Andere Geometrien werden unverändert zurückgegeben.

Verfügbarkeit: 2.4.0

Diese Funktion unterstützt 3d und lässt den Z-Index nicht fallen.

Diese Funktion unterstützt M-Koordinaten.

Beispiele

Force a polygon and its hole to use clockwise exterior rings and counter-clockwise interior rings.

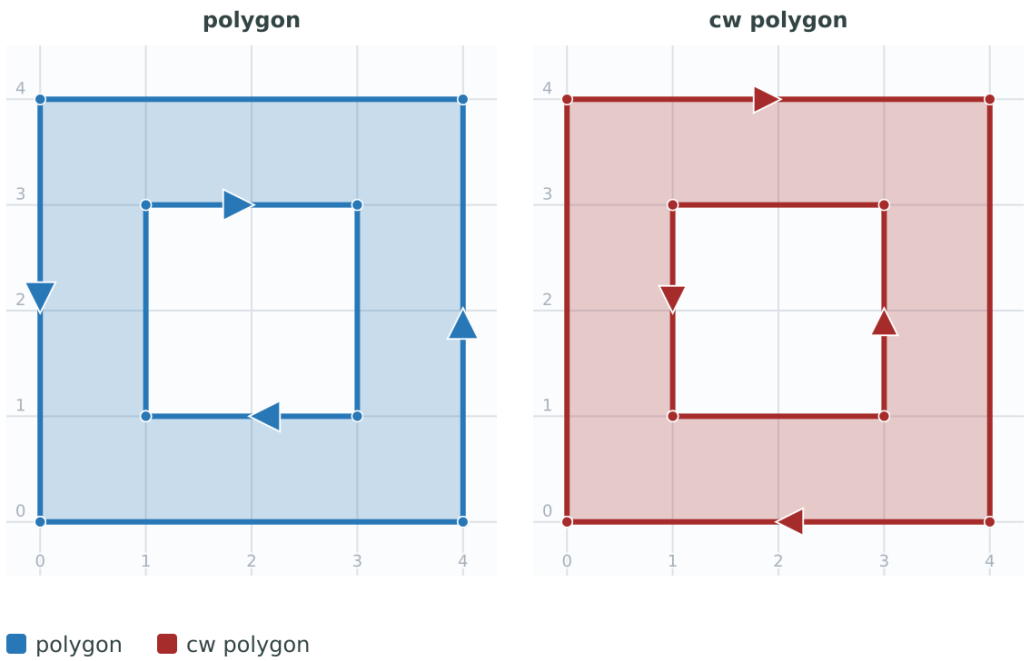
Code

```
WITH data AS (
  SELECT 'POLYGON((0 0,4 0,4 4,0 4,0 0),
                  (1 1,1 3,3 3,3 1,1 1))'::geometry AS geom
)
SELECT geom AS input_polygon,
       ST_ForcePolygonCW(geom) AS cw_polygon
FROM data;
```

Output

```
POLYGON((0 0,4 0,4 4,0 4,0 0),(1 1,1 3,3 3,3 1,1 1)) | POLYGON((0 0,0 4,4 4,4 0,0 0),(1 1,3 1,3 3,1 3,1 1))
```

Figure



Siehe auch

ST_ForcePolygonCCW, ST_IsPolygonCCW, ST_IsPolygonCW

7.5.16 ST_ForceSFS

ST_ForceSFS — Force geometries to use SFS 1.1 or 1.2 geometry types.

Synopsis

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

Beschreibung

Forces a geometry to use the geometry types defined by the requested version of the OGC Simple Features Specification. The default is SFS 1.1. Curved geometries are converted to linear representations, `TRIANGLE` is converted to `POLYGON`, and `TIN` and `POLYHEDRALSURFACE` are converted to `GEOMETRYCOLLECTION` values containing `POLYGON` elements.

If *`version`* begins with `1.2`, curved geometries are still converted to linear representations, but `TRIANGLE`, `TIN`, and `POLYHEDRALSU` are preserved. Other version values use the default SFS 1.1 behavior. Geometry collections are processed recursively, and the SRID and Z/M coordinates are preserved.

Verfügbarkeit: 2.1.0

Diese Funktion unterstützt polyedrische Flächen.

Diese Funktion unterstützt Dreiecke und dreieckige unregelmäßige Netzoberflächen (TIN).

Diese Methode unterstützt kreisförmige Strings und Kurven.

Diese Funktion unterstützt 3d und lässt den Z-Index nicht fallen.

Beispiele

Compare the geometry types produced for SFS 1.1 (the default) and SFS 1.2. Read each row from the input type in the direction of the result columns.

Code

```
WITH inputs(geom) AS (VALUES
    ('CIRCULARSTRING(0 0, 1 1, 2 0)'::geometry),
    ('TRIANGLE((0 0, 0 2, 2 0, 0 0))'::geometry),
    ('TIN(((0 0, 0 2, 2 0, 0 0)))'::geometry),
    ('POLYHEDRALSURFACE(((0 0, 0 2, 2 0, 0 0)))'::geometry)
)
SELECT ST_GeometryType(geom) || ' →' AS input,
       ST_GeometryType(ST_ForceSFS(geom)) AS "SFS 1.1 (default)",
       ST_GeometryType(ST_ForceSFS(geom, '1.2')) AS "SFS 1.2"
FROM inputs;
```

Output

```
input          |   SFS 1.1 (default)   |       SFS 1.2
-----------------------+-----------------------+----------------------
 ST_CircularString →    | ST_LineString         | ST_LineString
 ST_Triangle →          | ST_Polygon            | ST_Triangle
 ST_Tin →               | ST_GeometryCollection | ST_Tin
 ST_PolyhedralSurface → | ST_GeometryCollection | ST_PolyhedralSurface
(4 rows)
```

Siehe auch

ST_CurveToLine, ST_ForceCurve

7.5.17 ST_ForceRHR

ST_ForceRHR — Orientiert die Knoten in einem Polygon so, dass sie der Drei-Finger-Regel folgen.

Synopsis

geometry **ST_ForceRHR**(geometry g);

Beschreibung

Orientiert die Knoten in einem Polygon so, dass sie der Drei-Finger-Regel folgen. Dadurch kommt die durch das Polygon begrenzte Fläche auf der rechten Seite der Begrenzung zu liegen. Insbesondere sind der äussere Ring im Uhrzeigersinn und die inneren Ringe gegen den Uhrzeigersinn orientiert. Diese Funktion ist ein Synonym für ST_ForcePolygonCW



Anmerkung
Die obere Definition mit der Drei-Finger-Regel widerspricht den Definitionen, die in anderen Zusammenhängen verwendet werden. Um Verwirrung zu vermeiden, wird die Verwendung von ST_ForcePolygonCW empfohlen.

Erweiterung: Mit 2.0.0 wurde die Unterstützung für polyedrische Oberflächen eingeführt.

 Diese Funktion unterstützt 3d und lässt den Z-Index nicht fallen.

 Diese Funktion unterstützt polyedrische Flächen.

Beispiele

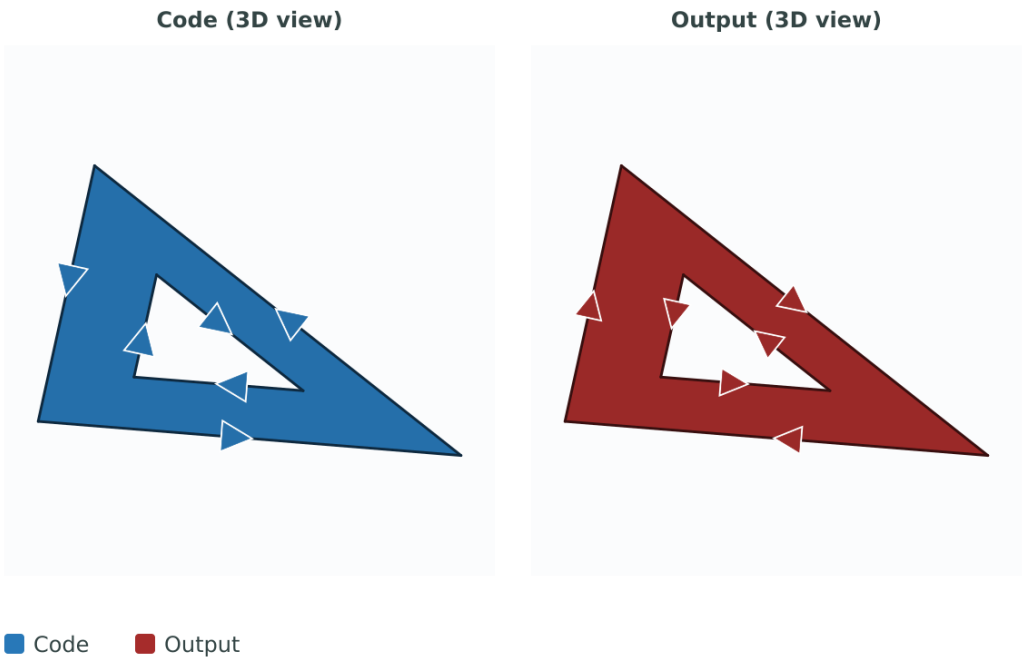
Code

```
SELECT ST_ForceRHR('POLYGON((0 0 2,5 0 2,0 5 2,0 0 2),(1 1 2,1 3 2,3 1 2,1 1 2))'
  );
```

Output

```
POLYGON((0 0 2,0 5 2,5 0 2,0 0 2),(1 1 2,3 1 2,1 3 2,1 1 2))
```

Figure



Siehe auch

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

7.5.18 ST_LineExtend

ST_LineExtend — Gibt eine Linie zurück, die um die angegebenen Abstände vorwärts und rückwärts verlängert wurde.

Synopsis

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

Beschreibung

Gibt eine vorwärts und rückwärts verlängerte Linie zurück, indem neue Start- (und End-) Punkte in den angegebenen Abständen hinzugefügt werden. Bei einem Abstand von Null wird kein Punkt hinzugefügt. Nur nicht-negative Abstände sind erlaubt. Die Richtung des/der hinzugefügten Punkte(s) wird durch die ersten (und letzten) zwei unterschiedlichen Punkte der Linie bestimmt. Doppelte Punkte werden ignoriert.

Verfügbarkeit: 3.4.0

Beispiel: Erweitert eine Linie um 5 Einheiten vorwärts und 6 Einheiten rückwärts

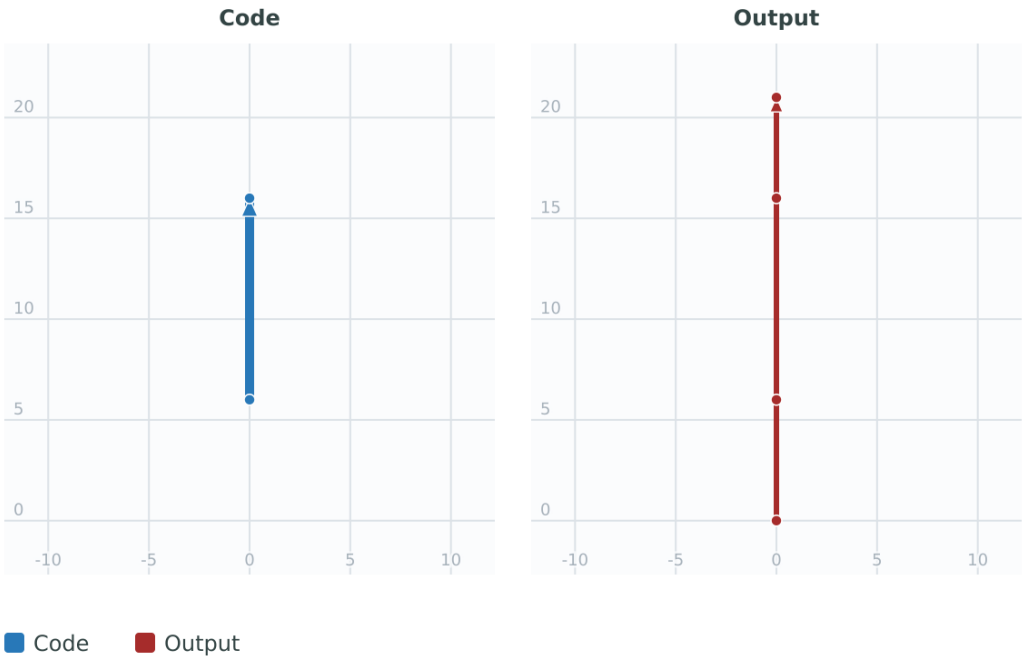
Code

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

Output

```
LINESTRING(0 -6,0 0,0 10,0 15)
```

Figure



Siehe auch

ST_LineSubstring, ST_LocateAlong, ST_Project

7.5.19 ST_LineToCurve

ST_LineToCurve — Konvertiert eine lineare Geometrie in eine gekrümmte Geometrie.

Synopsis

geometry **ST_LineToCurve**(geometry geomANoncircular);

Beschreibung

Wandelt einen einfachen LineString/Polygon in Kreisbögen/CIRCULARSTRINGs und Kurvenpolygone um. Beachten Sie, dass wesentlich weniger Punkte zur Beschreibung des Kurvenäquivalents benötigt werden.



Anmerkung
Wenn der gegebene LINESTRING/POLYGON nicht genug gekrümmt ist um eine deutliche Kurve zu repräsentieren, wird die Geometrie von der Funktion unverändert zurückgegeben.

Verfügbarkeit: 1.3.0

 Diese Funktion unterstützt 3d und lässt den Z-Index nicht fallen.

 Diese Methode unterstützt kreisförmige Strings und Kurven.

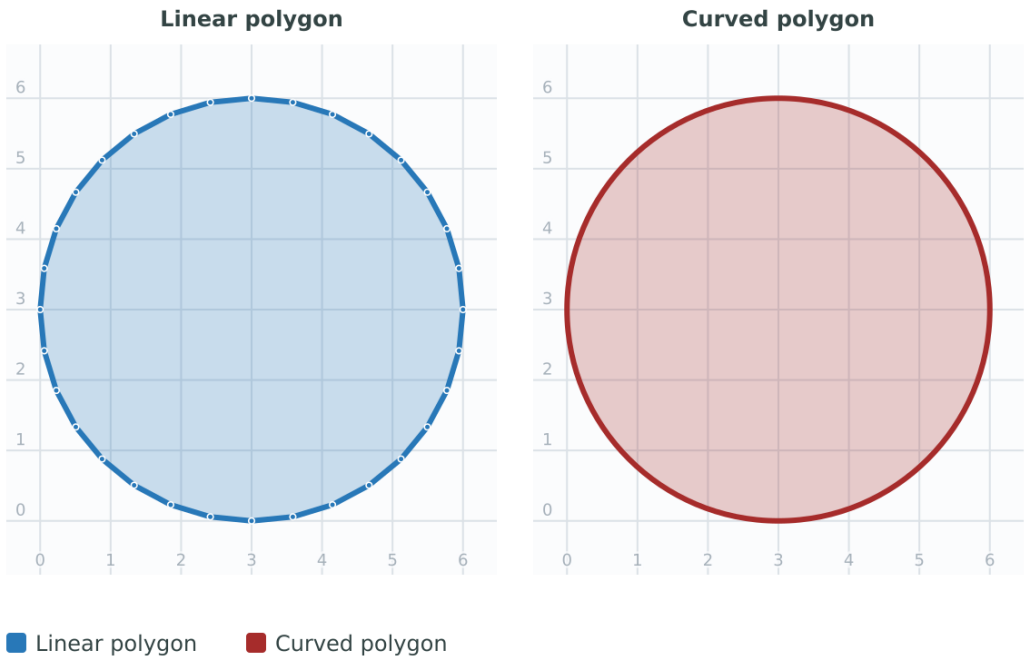
Beispiele

This example converts a linear Polygon approximating a circle to a CurvePolygon.

Code

```
WITH data AS (
    SELECT ST_Buffer('POINT(1 3)'::geometry, 3) AS geom
)
SELECT geom AS "input_Linear polygon",
       ST_LineToCurve(geom) AS "Curved polygon"
FROM data;
```

Figure

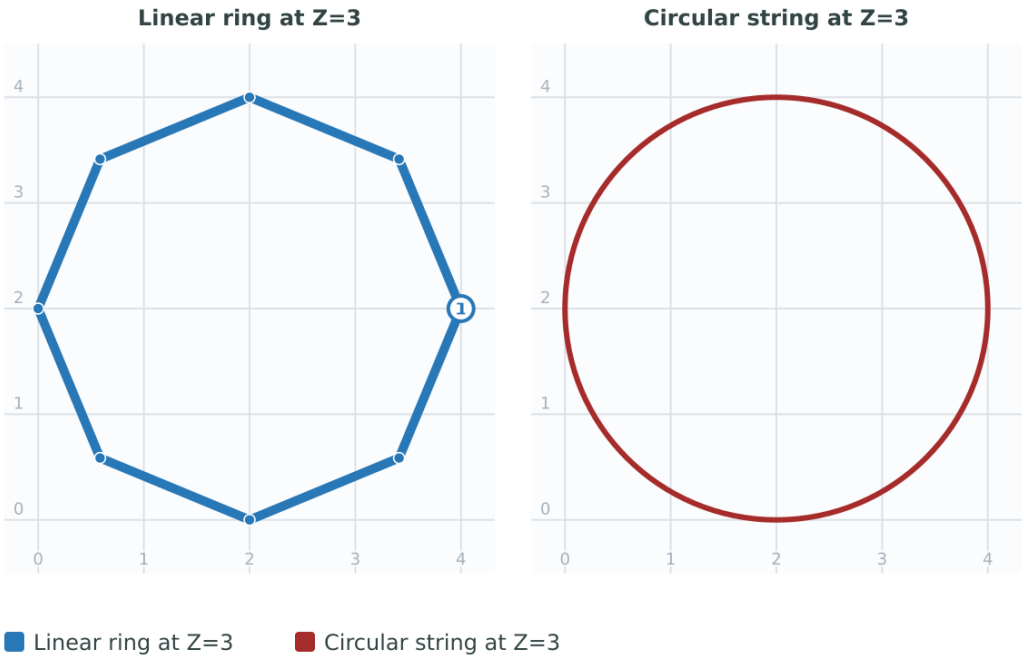


This example converts a 3D linear ring to a CircularString while preserving Z.

Code

```
WITH data AS (
    SELECT ST_Translate(
        ST_Force3D(ST_Boundary(ST_Buffer(ST_Point(1, 3), 2, 2))),
        0, 0, 3
    ) AS geom
)
SELECT geom AS "input_Linear ring at Z=3",
       ST_LineToCurve(geom) AS "Circular string at Z=3"
FROM data;
```

Figure



Siehe auch

ST_CurveToLine

7.5.20 ST_Multi

ST_Multi — Gibt die Geometrie als MULTI* Geometrie zurück.

Synopsis

geometry **ST_Multi**(geometry geom);

Beschreibung

Gibt die Geometrie als MULTI*-Geometriesammlung zurück. Wenn die Geometrie bereits eine Sammlung ist, wird sie unverändert zurückgegeben.

Beispiele

A Polygon is converted to a single-element MultiPolygon. The coordinates are unchanged.

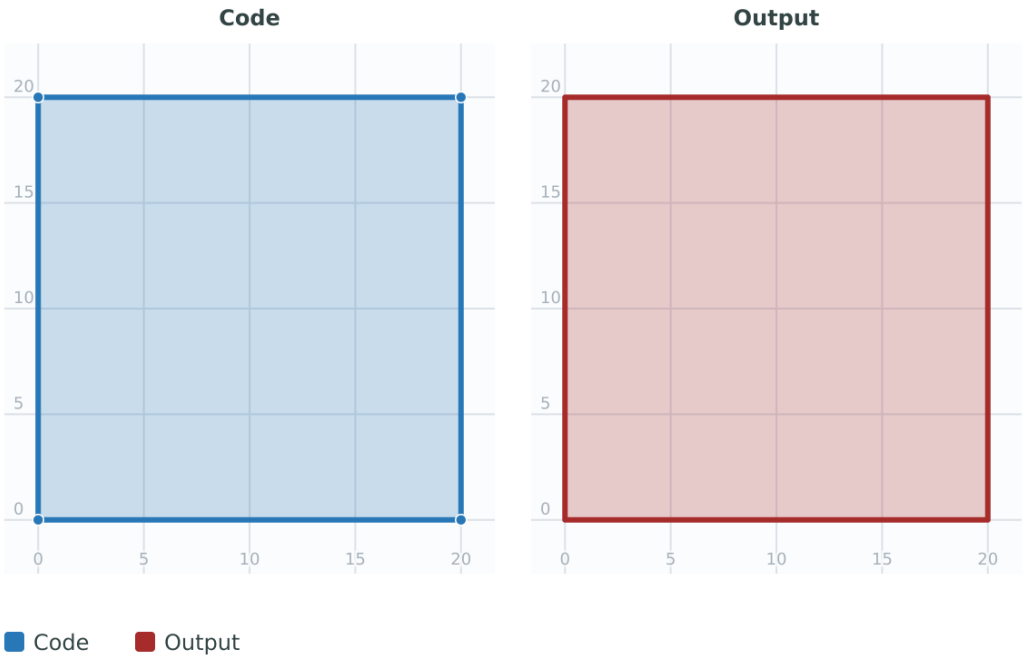
Code

```
SELECT ST_Multi('POLYGON ((10 30,30 30,30 10,10 10,10 30))');
```

Output

```
MULTIPOLYGON(((10 30,30 30,30 10,10 10,10 30)))
```

Figure



Siehe auch

ST_AsText

7.5.21 ST_Normalize

ST_Normalize — Gibt die Geometrie in Normalform zurück.

Synopsis

geometry **ST_Normalize**(geometry geom);

Beschreibung

Gibt die Geometrie in Normalform aus. Möglicherweise werden die Knoten der Polygonringe, die Ringe eines Polygons oder die Elemente eines Komplexes von Mehrfachgeometrien neu gereiht.

Hauptsächlich für Testzwecke sinnvoll (zum Vergleich von erwarteten und erhaltenen Ergebnissen).

Verfügbarkeit: 2.3.0

Beispiele

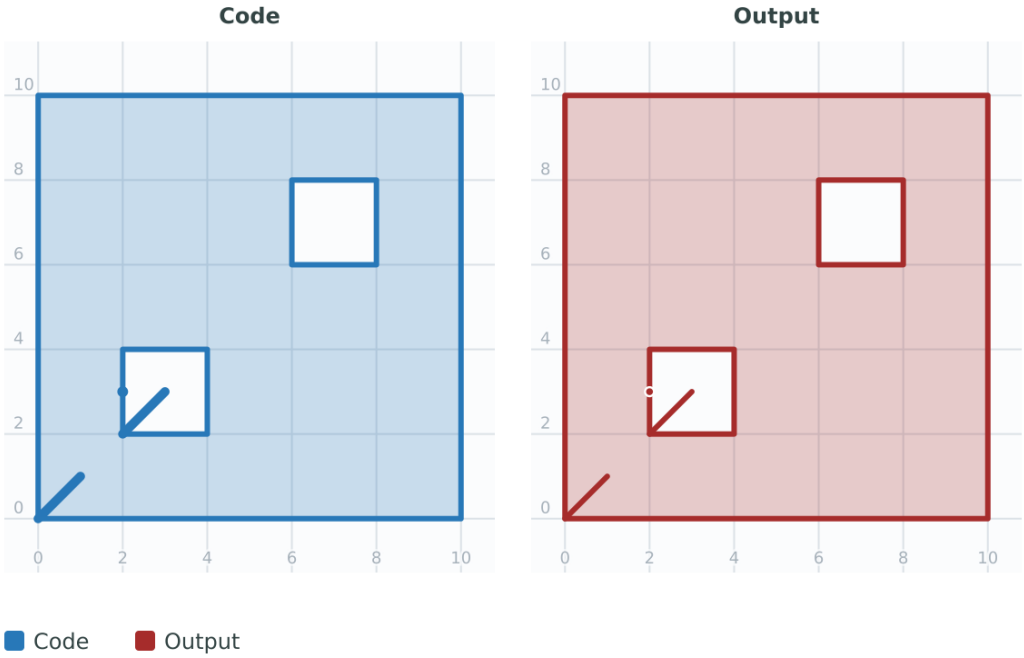
Code

```
SELECT ST_Normalize('GEOMETRYCOLLECTION(
    POINT(2 3),
    MULTILINESTRING((0 0,1 1),(2 2,3 3)),
    POLYGON(
      (0 10,0 0,10 0,10 10,0 10),
      (4 2,2 2,2 4,4 4,4 2),
      (6 8,8 8,8 6,6 6,6 8)
    )
  )');
```

Output

```
GEOMETRYCOLLECTION(POLYGON((0 0,0 10,10 10,10 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))
```

Figure



Siehe auch

ST_Equals,

7.5.22 ST_Project

ST_Project — Gibt einen Punkt zurück, der von einem Startpunkt um eine bestimmte Entfernung und Peilung (Azimut) projiziert wird.

Synopsis

geometry **ST_Project**(geometry g1, float distance, float azimuth);
geometry **ST_Project**(geometry g1, geometry g2, float distance);
geography **ST_Project**(geography g1, float distance, float azimuth);
geography **ST_Project**(geography g1, geography g2, float distance);

Beschreibung

Gibt einen Punkt zurück, der von einem Punkt entlang einer Geodäte projiziert wird, wobei eine bestimmte Entfernung und ein bestimmter Azimut (Peilung) verwendet werden. Dies ist bekannt als das direkte geodätische Problem.

Bei der Zweipunktversion wird der Weg vom ersten zum zweiten Punkt verwendet, um implizit den Azimut zu definieren, und die Entfernung wird wie zuvor verwendet.

Die Entfernung wird in Metern angegeben. Negative Werte werden unterstützt.

Der Azimut (auch als Kurs oder Peilung bezeichnet) wird im Bogenmaß angegeben. Er wird im Uhrzeigersinn vom geografischen Norden aus gemessen.

- Norden ist Azimut Null (0 Grad)
- Osten ist Azimut $\pi/2$ (90 Grad)
- Süden ist Azimut π (180 Grad)
- West ist Azimut $3\pi/2$ (270 Grad)

Negative Azimutwerte und Werte größer als 2π (360 Grad) werden unterstützt.

Verfügbarkeit: 2.0.0

Verbessert: 2.4.0 Erlaubt negative Entfernungen und nicht-normierte Azimute.

Verbessert: 3.4.0 Erlaubt Geometrieargumente und Zweipunktform ohne Azimut.

Beispiel: Projizierter Punkt auf 100.000 Meter und Peilung 45 Grad

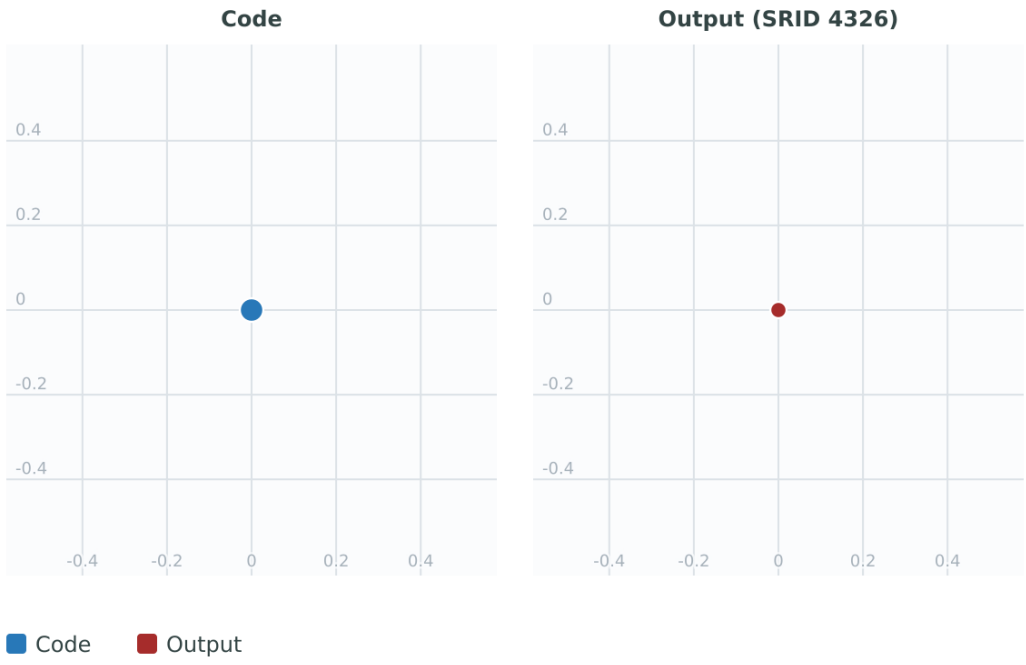
Code

```
SELECT ST_Project('POINT(0 0)'::geography, 100000, radians(45.0));
```

Output

```
POINT(0.635231029125537 0.639472334729198)
```

Figure



Siehe auch

ST_Azimuth, ST_Distance, PostgreSQL-Funktion radians()

7.5.23 ST_QuantizeCoordinates

ST_QuantizeCoordinates — Setzt die niedrigwertigsten Bits der Koordinaten auf Null

Synopsis

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

Beschreibung

`ST_QuantizeCoordinates` ermittelt die Anzahl der Bits (`N`), die erforderlich sind, um einen Koordinatenwert mit einer bestimmten Anzahl von Nachkommastellen darzustellen, und setzt dann alle Bits bis auf die `N` höchstwertigen auf Null. Der resultierende Koordinatenwert wird immer noch auf den ursprünglichen Wert gerundet, hat aber eine verbesserte Kompressibilität. Dies kann zu einer erheblichen Verringerung der Festplattennutzung führen, vorausgesetzt, die Geometriespalte verwendet einen komprimierbaren Speichertyp. Die Funktion erlaubt die Angabe einer unterschiedlichen Anzahl von Nachkommastellen in jeder Dimension; bei nicht spezifizierten Dimensionen wird die Genauigkeit der `x` Dimension angenommen. Negative Ziffern werden so interpretiert, dass sie sich auf Ziffern links vom Dezimalpunkt beziehen (d.h. `prec_x=-2` erhält Koordinatenwerte auf die nächsten 100.

Die von `ST_QuantizeCoordinates` erzeugten Koordinaten sind unabhängig von der Geometrie, die diese Koordinaten und die relative Position dieser Koordinaten in der Geometrie enthält. Daher sind vorhandene topologische Beziehungen zwischen Geometrien durch die Verwendung dieser Funktion nicht betroffen. Die Funktion erzeugt möglicherweise ungültige Geometrie, wenn sie mit einer Anzahl von Stellen aufgerufen wird, die Koordinaten innerhalb der Geometrie zusammenfallen lassen.

Verfügbarkeit: 2.5.0

Technischer Hintergrund

PostGIS speichert alle Koordinatenwerte als Gleitkommazahlen mit doppelter Genauigkeit, die 15 signifikante Stellen zuverlässig darstellen können. PostGIS kann jedoch verwendet werden, um Daten zu verwalten, die weniger als 15 signifikante Ziffern enthalten. Ein Beispiel sind TIGER-Daten, die als geografische Koordinaten mit sechs Nachkommastellen zur Verfügung gestellt werden (so dass nur neun signifikante Ziffern des Längengrads und acht signifikante Breitengrade erforderlich sind).

Wenn 15 signifikante Ziffern verfügbar sind, gibt es viele mögliche Darstellungen einer Zahl mit 9 signifikanten Ziffern. Eine Gleitkommazahl mit doppelter Genauigkeit verwendet 52 explizite Bits, um den Mantisse der Koordinate darzustellen. Nur 30 Bits werden benötigt, um eine Mantisse mit 9 signifikanten Ziffern darzustellen, wobei 22 unbedeutende Bits übrig bleiben; Wir können ihren Wert auf alles setzen, was wir wollen, und erhalten trotzdem eine zum Eingabewert passende Zahl. Beispielsweise kann der Wert 100.123456 durch die nächstliegenden Zahlen 100.123456000000, 100.123456000001 und 100.123456432199 dargestellt werden. Alle sind gleichermaßen gültig, da `ST_AsText (geom, 6)` bei allen dieser Eingaben das gleiche Ergebnis liefert. Da wir diese Bits auf einen beliebigen Wert setzen können, setzt `ST_QuantizeCoordinates` die 22 nicht signifikanten Bits auf Null. Für eine lange Koordinatensequenz wird dadurch ein Muster aus Blöcken von aufeinanderfolgenden Nullen erzeugt, das von PostgreSQL effizienter komprimiert wird.



Anmerkung
Von `ST_QuantizeCoordinates` ist möglicherweise nur die Größe der Geometrie auf der Festplatte betroffen. ST_MemSize, das die speicherinterne Verwendung der Geometrie meldet, gibt unabhängig vom von einer Geometrie belegten Speicherplatz den gleichen Wert zurück.

Beispiele

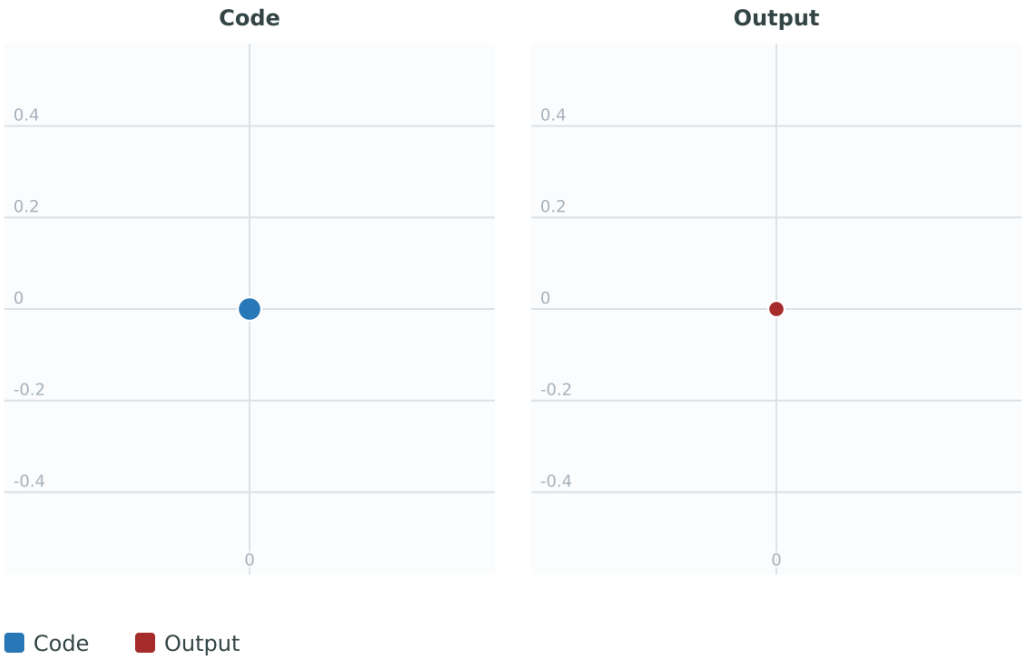
Code

```
SELECT ST_QuantizeCoordinates('POINT (100.123456 0)'::geometry, 4);
```

Output

```
POINT(100.12344360351562 0)
```

Figure



Code

```
WITH test AS (SELECT 'POINT (123.456789123456 123.456789123456)'::geometry AS geom)
SELECT
  digits,
  encode(ST_QuantizeCoordinates(geom, digits), 'hex'),
  ST_QuantizeCoordinates(geom, digits)
FROM test, generate_series(15, -15, -1) AS digits;
```

Output

```
digits |                  encode                  |                 st_astext
--------+------------------------------------------+-------------------------------------------
     15 | 01010000005f9a72083cdd5e405f9a72083cdd5e40 | POINT(123.456789123456 123.456789123456)
     14 | 01010000005f9a72083cdd5e405f9a72083cdd5e40 | POINT(123.456789123456 123.456789123456)
     13 | 01010000005e9a72083cdd5e405e9a72083cdd5e40 | POINT(123.45678912345599 123.45678912345599)
     12 | 0101000000409a72083cdd5e40409a72083cdd5e40 | POINT(123.45678912345556 123.45678912345556)
     11 | 0101000000009a72083cdd5e40009a72083cdd5e40 | POINT(123.45678912345466 123.45678912345466)
     10 | 0101000000009872083cdd5e40009872083cdd5e40 | POINT(123.45678912344738 123.45678912344738)
      9 | 0101000000008072083cdd5e40008072083cdd5e40 | POINT(123.45678912336007 123.45678912336007)
      8 | 0101000000000070083cdd5e40000070083cdd5e40 | POINT(123.45678912103176 123.45678912103176)
      7 | 0101000000000060083cdd5e40000060083cdd5e40 | POINT(123.4567891061306 123.4567891061306)
      6 | 0101000000000000083cdd5e40000000083cdd5e40 | POINT(123.45678901672363 123.45678901672363)
      5 | 0101000000000000003cdd5e40000000003cdd5e40 | POINT(123.456787109375 123.456787109375)
      4 | 0101000000000000003cdd5e40000000003cdd5e40 | POINT(123.456787109375 123.456787109375)
      3 | 0101000000000000038dd5e400000000038dd5e40 | POINT(123.45654296875 123.45654296875)
      2 | 0101000000000000000dd5e400000000000dd5e40 | POINT(123.453125 123.453125)
      1 | 0101000000000000000dc5e400000000000dc5e40 | POINT(123.4375 123.4375)
      0 | 0101000000000000000c05e400000000000c05e40 | POINT(123 123)
     -1 | 0101000000000000000005e400000000000005e40 | POINT(120 120)
     -2 | 0101000000000000000005840000000000005840 | POINT(96 96)
     -3 | 0101000000000000000005840000000000005840 | POINT(96 96)
     -4 | 0101000000000000000005840000000000005840 | POINT(96 96)
     -5 | 0101000000000000000005840000000000005840 | POINT(96 96)
```

```
     -6 | 0101000000000000000000584000000000000005840 | POINT(96 96)
     -7 | 0101000000000000000000584000000000000005840 | POINT(96 96)
     -8 | 0101000000000000000000584000000000000005840 | POINT(96 96)
     -9 | 0101000000000000000000584000000000000005840 | POINT(96 96)
    -10 | 0101000000000000000000584000000000000005840 | POINT(96 96)
    -11 | 0101000000000000000000584000000000000005840 | POINT(96 96)
    -12 | 0101000000000000000000584000000000000005840 | POINT(96 96)
    -13 | 0101000000000000000000584000000000000005840 | POINT(96 96)
    -14 | 0101000000000000000000584000000000000005840 | POINT(96 96)
    -15 | 0101000000000000000000584000000000000005840 | POINT(96 96)
(31 rows)
```

Siehe auch

ST_SnapToGrid

7.5.24 ST_RemovePoint

ST_RemovePoint — Einen Punkt aus einem Linienzug entfernen.

Synopsis

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

Beschreibung

Entfernt einen Punkt aus einem LineString unter Angabe seines Index (0-basiert). Nützlich, um eine geschlossene Linie (Ring) in einen offenen LineString zu verwandeln.

Verbessert: 3.2.0

Verfügbarkeit: 1.1.0

 Diese Funktion unterstützt 3d und lässt den Z-Index nicht fallen.

Beispiele

Garantiert, dass keine Linien geschlossen sind, indem der Endpunkt von geschlossenen Linien (Ringen) entfernt wird. Setzt voraus, dass geom vom Typ LINESTRING ist

Code

```
UPDATE sometable
    SET geom = ST_RemovePoint(geom, ST_NPoints(geom) - 1)
    FROM sometable
    WHERE ST_IsClosed(geom);
```

Siehe auch

ST_AddPoint, ST_NPoints, ST_NumPoints

7.5.25 ST_RemoveRepeatedPoints

ST_RemoveRepeatedPoints — Gibt eine Version einer Geometrie zurück, bei der doppelte Punkte entfernt wurden.

Synopsis

geometry **ST_RemoveRepeatedPoints**(geometry geom, float8 tolerance = 0.0);

Beschreibung

Gibt eine Version der angegebenen Geometrie zurück, bei der doppelte aufeinanderfolgende Punkte entfernt wurden. Die Funktion verarbeitet nur (Multi)LineStrings, (Multi)Polygone und MultiPoints, kann aber mit jeder Art von Geometrie aufgerufen werden. Elemente von GeometryCollections werden einzeln verarbeitet. Die Endpunkte von LineStrings werden beibehalten.

If a non-zero *tolerance* parameter is provided, vertices within the tolerance distance of one another are considered to be duplicates. The distance is computed in 2D (XY plane).

Verbessert: 3.2.0

Verfügbarkeit: 2.2.0

Diese Funktion unterstützt polyedrische Flächen.

Diese Funktion unterstützt 3d und lässt den Z-Index nicht fallen.

Beispiele

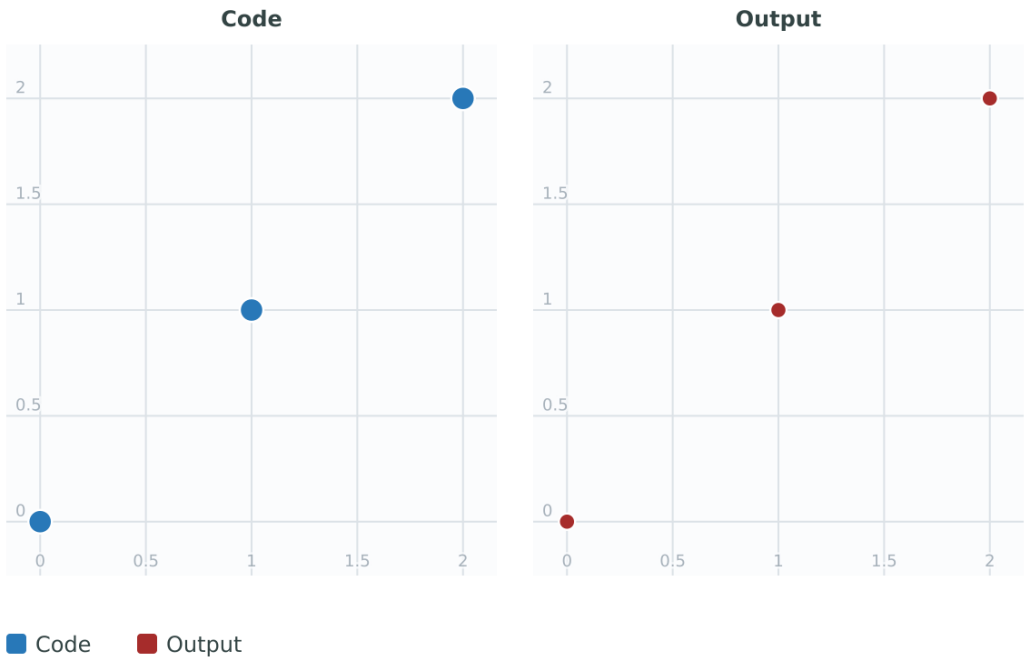
Code

```
SELECT ST_RemoveRepeatedPoints('MULTIPOINT ((1 1),(2 2),(3 3),(2 2))');
```

Output

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

Figure



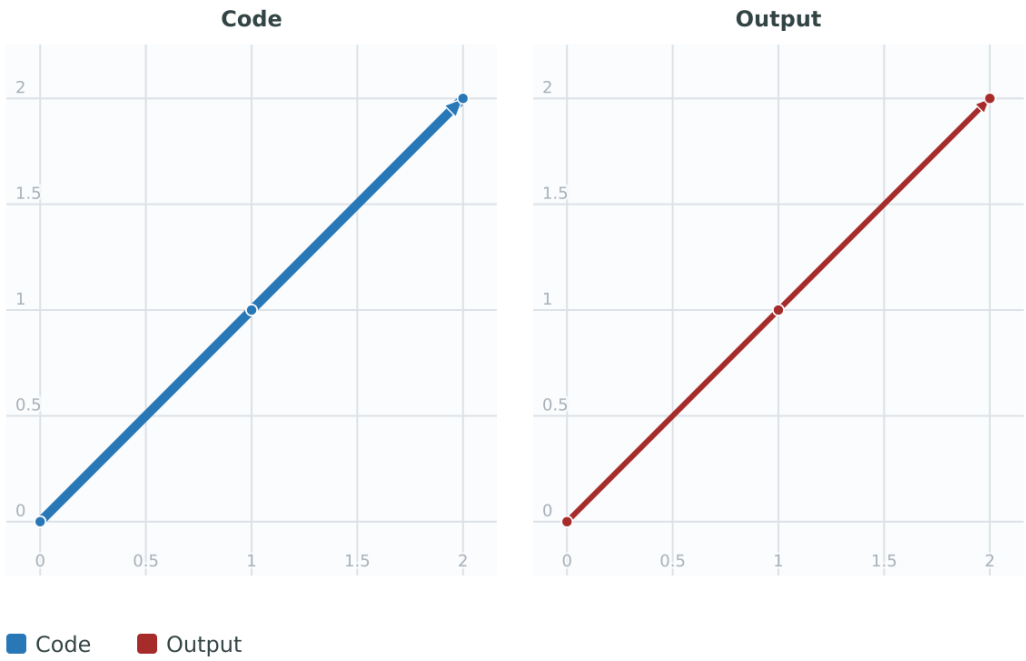
Code

```
SELECT ST_RemoveRepeatedPoints('LINESTRING (0 0,0 0,1 1,0 0,1 1,2 2)');
```

Output

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

Figure



Beispiel: Sammlungselemente werden einzeln bearbeitet.

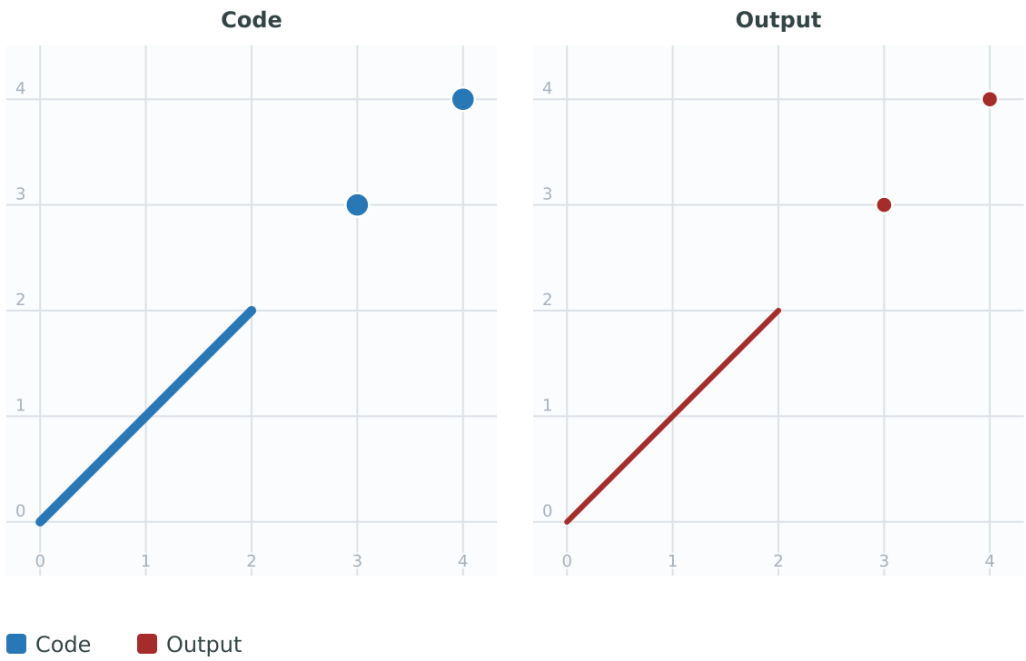
Code

```
SELECT ST_RemoveRepeatedPoints('GEOMETRYCOLLECTION (LINESTRING (1 1,2 2,2 2,3 3),POINT (4 4) ↩
    ,POINT (4 4),POINT (5 5))');
```

Output

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

Figure



Beispiel: Wiederholte Punktentfernung mit einer Abstandstoleranz.

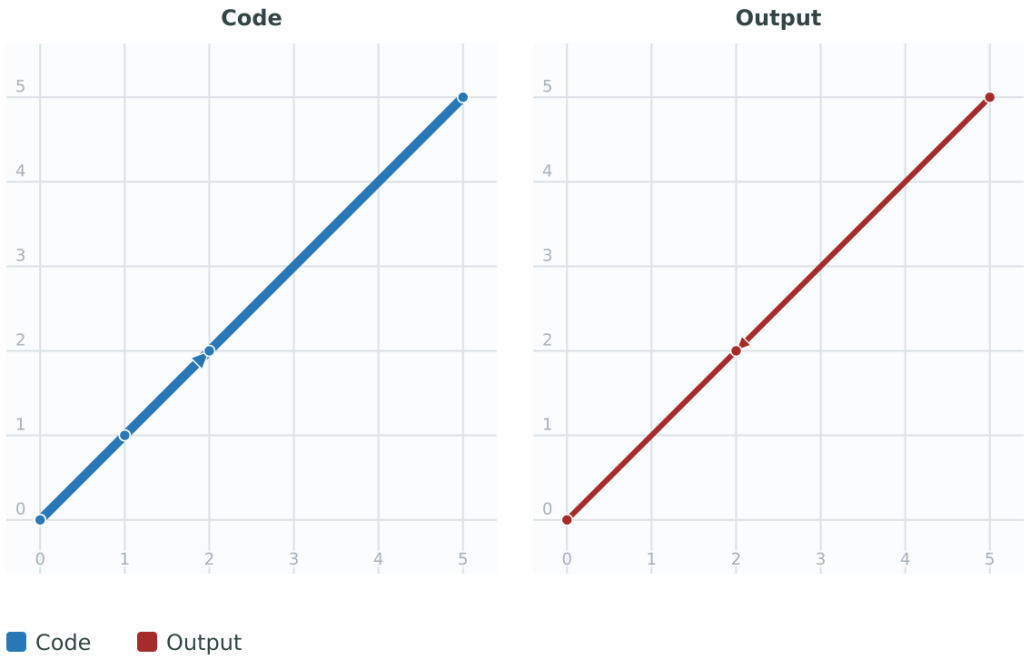
Code

```
SELECT ST_RemoveRepeatedPoints('LINESTRING (0 0,0 0,1 1,5 5,1 1,2 2)', 2);
```

Output

```
LINESTRING(0 0,5 5,2 2)
```

Figure



Siehe auch

ST_Simplify

7.5.26 ST_RemoveIrrelevantPointsForView

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

Synopsis

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

Beschreibung

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

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

Es wird nur die Geometrie vom Typ (MULTI)POLYGON und (MULTI)LINESTRING ausgewertet. Die restliche Geometrie bleibt unverändert.

Im Gegensatz zu `ST_ClipByBox2D()` werden von dieser Funktion

- sortiert Punkte aus, ohne neue Schnittpunkte zu berechnen, was Rundungsfehler vermeidet und in der Regel die Leistung erhöht,
- gibt eine Geometrie mit gleicher oder ähnlicher Anzahl an Punkten zurück,
- führt zu demselben Rendering-Ergebnis innerhalb der gegebenen Ansicht, und
- es kann zu Selbstüberschneidungen kommen, welche die resultierende Geometrie ungültig machen können (siehe Beispiel unten).

Wenn `cartesian_hint` auf `true` gesetzt ist, wendet der Algorithmus zusätzliche Optimierungen unter Verwendung kartesischer Mathematik an, um die resultierende Punktanzahl weiter zu reduzieren. Beachten Sie bitte, dass die Verwendung dieser Option Rendering-Artefakte verursachen kann, wenn die resultierenden Koordinaten in ein anderes (nicht-kartesisches) Koordinatensystem projiziert werden, bevor sie gerendert werden.“



Warnung
Für Polygone stellt diese Funktion derzeit nicht sicher, dass das Ergebnis gültig ist. Diese Situation kann mit ST_IsValid überprüft und mit ST_MakeValid repariert werden.

Verfügbarkeit: 3.5.0

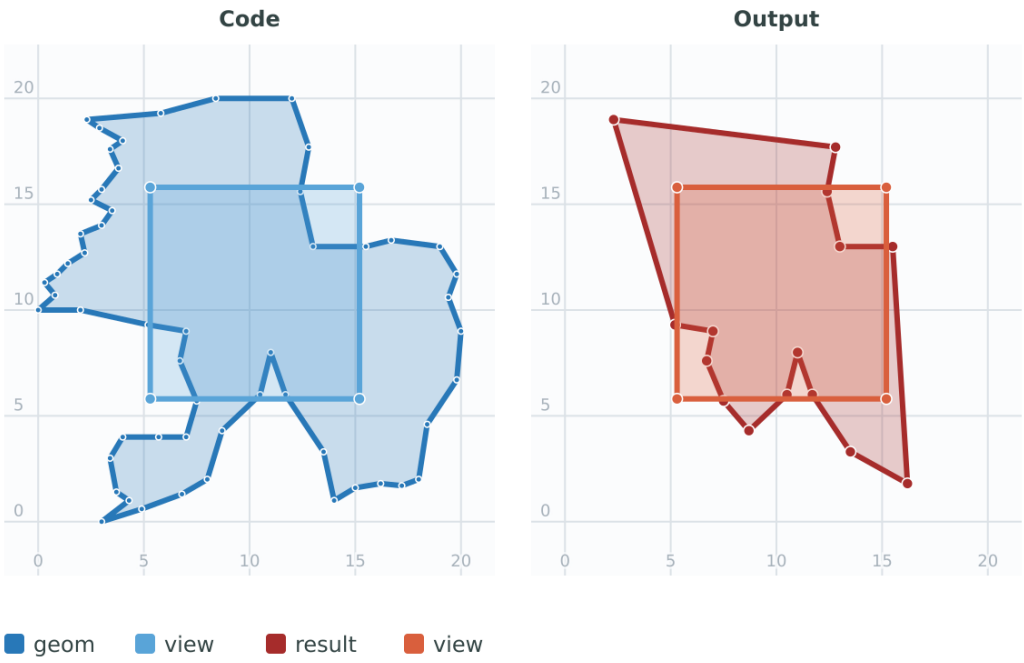
Beispiele

The input polygon has 55 points. The result retains 15 points, while rendering the same shape inside the view envelope.

Code

```
WITH data(geom, view) AS (VALUES (
  'POLYGON((3.2 14.2,6.8 14.2,7.6 11.9,7.2 9.8,7.8 7.2,10.3 7.2,11.5 7.5,13.8 7.2,14.6 ↩
    5.9,14.2 4.8,14.8 3.2,14.6 0.9,13.2 -1.2,12.8 -3.8,12 -4.1,11 -4,9.8 -4.2,8.8 -4.8,8.3 ↩
    -2.5,6.5 0.2,5.8 2.2,5.3 0.2,3.5 -1.5,2.8 -3.8,1.6 -4.5,-0.3 -5.2,-2.2 -5.8,-0.9 ↩
    -4.8,-1.5 -4.4,-1.8 -2.8,-1.2 -1.8,0.5 -1.8,1.8 -1.8,2.3 -0.1,1.5 1.8,1.8 3.2,0 3.5,-3.2 ↩
     4.2,-5.2 4.2,-4.4 4.9,-4.9 5.5,-4.3 5.9,-3.8 6.4,-3 6.9,-3.2 7.8,-2.2 8.2,-1.7 8.9,-2.7 ↩
     9.4,-2.2 9.9,-1.4 10.9,-1.8 11.8,-1.2 12.2,-2.3 12.8,-2.9 13.2,0.6 13.5,3.2 14.2))'':: ↩
    geometry,
  ST_MakeEnvelope(0.1, 0, 10, 10)
))
SELECT geom AS input_geom,
       view AS input_view,
       ST_RemoveIrrelevantPointsForView(geom, view, true) AS result,
       view AS view
FROM data;
```

Figure

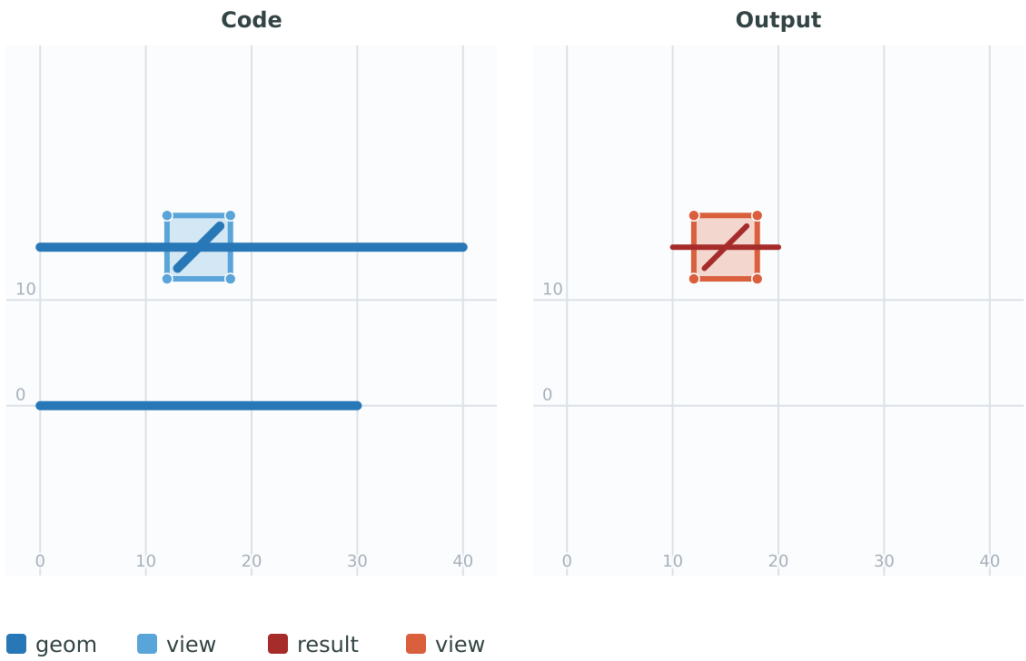


Only linework affecting the view is retained; existing vertices are reused.

Code

```
WITH data(geom, view) AS (VALUES (
  'MULTILINESTRING((0 0,10 0,20 0,30 0),(0 15,5 15,10 15,15 15,20 15,25 15,30 15,40 15),(13 ↩
    13,15 15,17 17))'::geometry,
  ST_MakeEnvelope(12, 12, 18, 18)
))
SELECT geom AS input_geom,
       view AS input_view,
       ST_RemoveIrrelevantPointsForView(geom, view, true) AS result,
       view AS view
FROM data;
```

Figure



A line with no segment relevant to the view becomes empty.

Code

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

Output

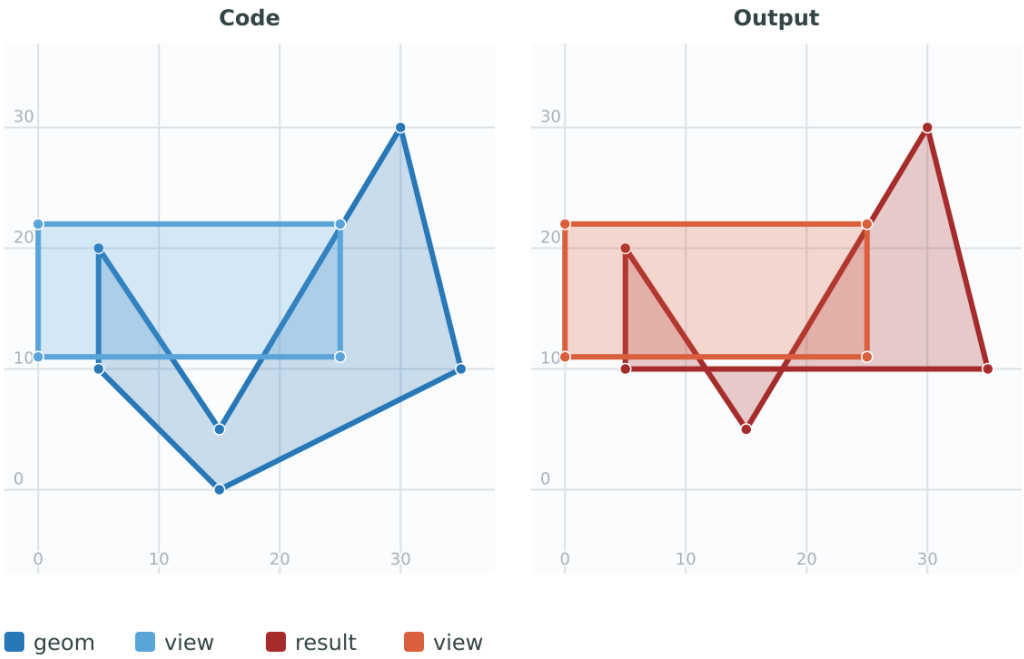
```
ZEILENSTRING LEER
```

Removing vertices without computing intersections can introduce self-intersections in the result.

Code

```
WITH data(geom, view) AS (VALUES (
  'POLYGON((0 10,0 0,10 -10,30 0,25 20,10 -5,0 10))'::geometry,
  ST_MakeEnvelope(-5, 1, 20, 12)
))
SELECT geom AS input_geom,
       view AS input_view,
       ST_RemoveIrrelevantPointsForView(geom, view, true) AS result,
       view AS view
FROM data;
```

Figure

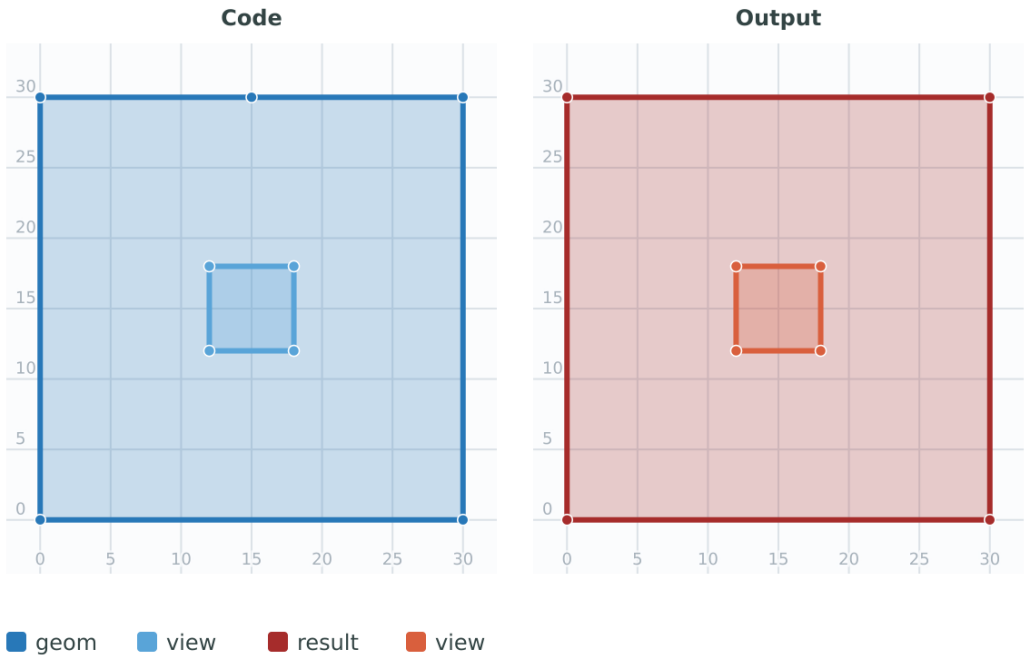


With the default *cartesian_hint*, the function removes only points known to be irrelevant to the view.

Code

```
WITH data(geom, view) AS (VALUES (
  'POLYGON((0 30,15 30,30 30,30 0,0 0,0 30))'::geometry,
  ST_MakeEnvelope(12, 12, 18, 18)
))
SELECT geom AS input_geom,
       view AS input_view,
       ST_RemoveIrrelevantPointsForView(geom, view) AS result,
       view AS view
FROM data;
```

Figure



Siehe auch

ST_ClipByBox2D, ST_Intersection

7.5.27 ST_RemoveSmallParts

ST_RemoveSmallParts — Entfernt kleine Teile (Polygonringe oder Linienzüge) einer Geometrie.

Synopsis

geometry **ST_RemoveSmallParts**(geometry geom, double precision minSizeX, double precision minSizeY);

Beschreibung

Gibt eine geometry ohne kleine Teile (äußere oder innere Polygonringe oder Linienzüge) zurück.

Diese Funktion kann als Vorverarbeitungsschritt zur Erstellung vereinfachter Karten verwendet werden, z. B. um kleine Inseln oder Löcher zu entfernen.

Es wertet nur Geometrien vom Typ (MULTI)POLYGON und (MULTI)LINESTRING aus. Andere Geometrien bleiben unverändert.

Wenn `minSizeX` größer als 0 ist, werden Teile ausgeschlossen, wenn ihre Breite kleiner als `minSizeX` ist.

Wenn `minSizeY` größer als 0 ist, werden Teile ausgeschlossen, wenn ihre Höhe kleiner als `minSizeY` ist.

Sowohl `minSizeX` als auch `minSizeY` werden in den Koordinatensystemeinheiten der Geometrie gemessen.

Bei Polygonen wird die Auswertung für jeden Ring getrennt vorgenommen, was zu einem der folgenden Ergebnisse führen kann:

- die ursprüngliche Geometrie,
- ein POLYGON mit allen Ringen, die weniger Scheitelpunkte haben

- ein POLYGON mit einer reduzierten Anzahl an inneren Ringen (möglicherweise mit weniger Scheitelpunkten),
- ein POLYGON EMPTY, oder
- ein MULTIPOLYGON mit einer reduzierten Anzahl an Polygonen (möglicherweise weniger innere Ringe oder Scheitelpunkte), oder
- ein MULTIPOLYGON EMPTY.

Bei Linienzügen wird die Auswertung für jeden Linienzug getrennt vorgenommen, was zu einem der folgenden Ergebnisse führen kann:

- die ursprüngliche Geometrie,
- ein LINESTRING mit einer reduzierten Anzahl an Scheitelpunkten,
- ein LINESTRING LEER,
- ein MULTILINESTRING mit einer reduzierten Anzahl an Linienzügen (möglicherweise weniger Scheitelpunkten), oder
- ein MULTILINESTRING LEER.

Verfügbarkeit: 3.5.0

Beispiele

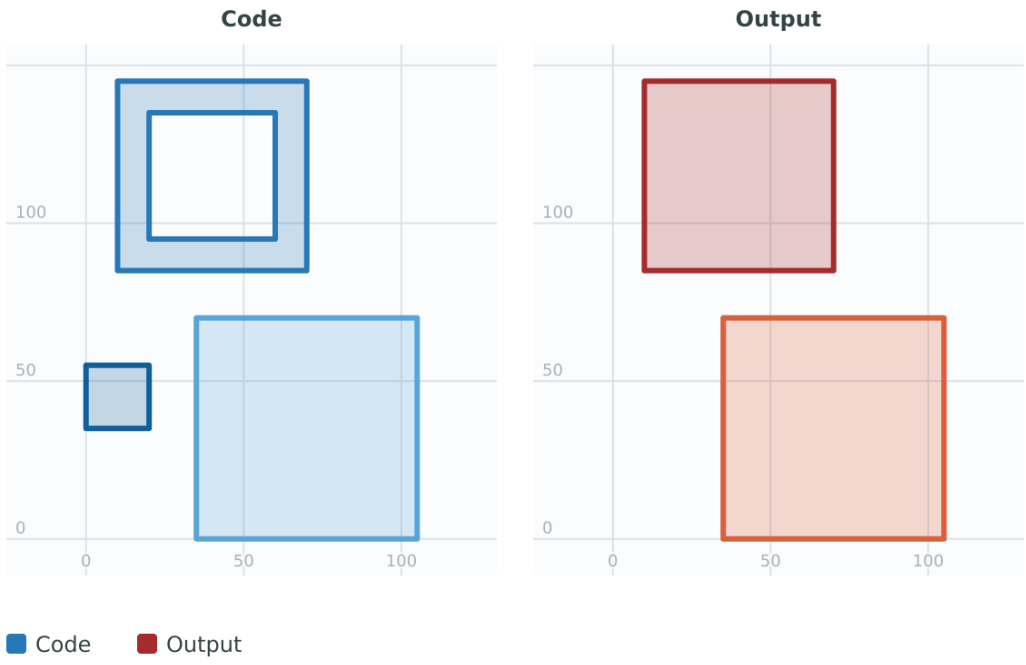
Code

```
SELECT ST_RemoveSmallParts('MULTIPOLYGON(
    ((60 160,120 160,120 220,60 220,60 160),(70 170,70 210,110 210,110 170,70 170)),
    ((85 75,155 75,155 145,85 145,85 75)),
    ((50 110,70 110,70 130,50 130,50 110)))',
    50, 50);
```

Output

```
MULTIPOLYGON(((60 160,120 160,120 220,60 220,60 160)),((85 75,155 75,155 145,85 145,85 75)))
```

Figure



Code

```
SELECT ST_RemoveSmallParts('LINESTRING(10 10,20 20)',
    50, 50);
```

Output

```
ZEILENSTRING LEER
```

7.5.28 ST_Reverse

ST_Reverse — Gibt die Geometrie in umgekehrter Knotenreihenfolge zurück.

Synopsis

geometry **ST_Reverse**(geometry g1);

Beschreibung

Kann mit jedem geometrischen Datentyp verwendet werden; kehrt die Reihenfolge der Knoten um

Erweiterung: mit 2.4.0 wurde die Unterstützung für Kurven eingeführt.

 Diese Funktion unterstützt 3d und lässt den Z-Index nicht fallen.

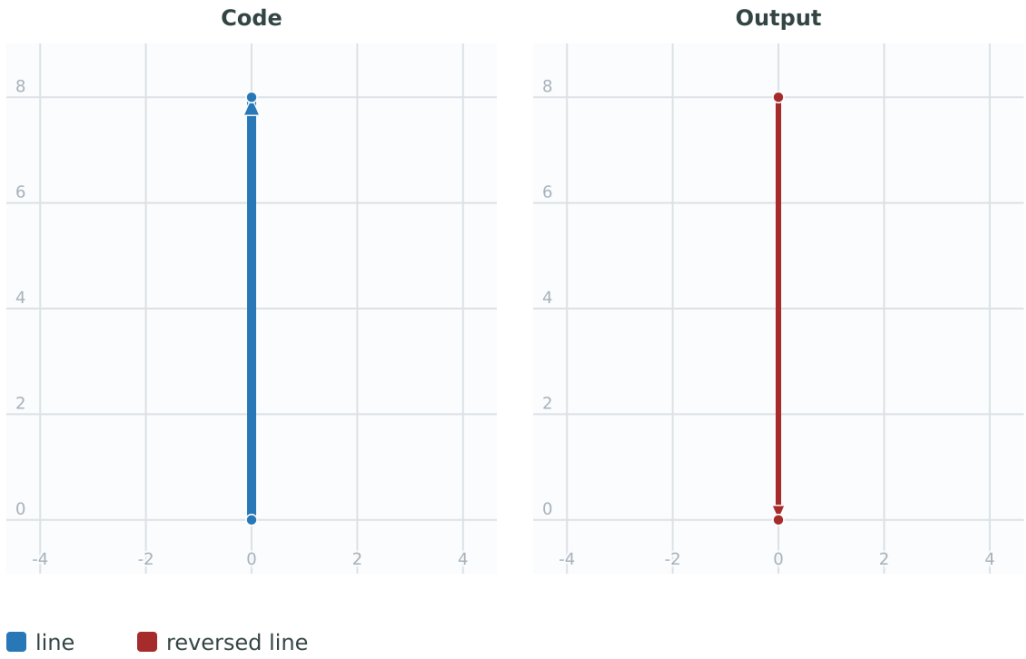
 Diese Funktion unterstützt polyedrische Flächen.

Beispiele

Code

```
WITH data(geom) AS (VALUES (
  ST_MakeLine(ST_Point(1, 2), ST_Point(1, 10))
))
SELECT geom AS input_line,
       ST_Reverse(geom) AS reversed_line
FROM data;
```

Figure



7.5.29 ST_Segmentize

ST_Segmentize — Gibt eine geänderte Geometrie/Geografie zurück, bei der kein Segment länger als eine bestimmte Entfernung ist.

Synopsis

geometry **ST_Segmentize**(geometry geom, float max_segment_length);
geography **ST_Segmentize**(geography geog, float max_segment_length);

Beschreibung

Gibt eine geänderte Geometrie/Geografie zurück, bei der kein Segment länger ist als `max_segment_length`. Die Länge wird in 2D berechnet. Segmente werden immer in Untersegmente gleicher Länge aufgeteilt.

- In der Geometrie wird die maximale Länge in den Einheiten des räumlichen Bezugssystems angegeben.
- In der Geografie wird die maximale Länge in Metern angegeben. Die Entfernungen werden auf der Kugel berechnet. Hinzugefügte Scheitelpunkte werden entlang der sphärischen Großkreisbögen erstellt, die durch die Endpunkte der Segmente definiert sind.



Anmerkung
Dadurch werden nur lange Segmente verkürzt. Segmente, die kürzer als die maximale Länge sind, werden nicht verlängert.



Warnung
Bei Eingaben, die lange Segmente enthalten, kann die Angabe einer relativ kurzen `max_segment_length` dazu führen, dass eine sehr große Anzahl von Scheitelpunkten hinzugefügt wird. Dies kann unbeabsichtigt geschehen, wenn das Argument versehentlich als Anzahl der Segmente und nicht als maximale Länge angegeben wird.

Verfügbarkeit: 1.2.2

Verbessert: 3.0.0 Segmentize-Geometrie erzeugt jetzt Teilsegmente gleicher Länge

Verbessert: 2.3.0 Die Segmentierung der Geografie erzeugt nun Teilsegmente gleicher Länge

Erweiterung: mit 2.1.0 wurde die Unterstützung des geographischen Datentyps eingeführt.

Geändert: 2.1.0 Infolge der Einführung der Geographie-Unterstützung verursacht die Verwendung `ST_Segmentize('LINESTRING 2, 3 4)', 0.5)` einen mehrdeutigen Funktionsfehler. Die Eingabe muss korrekt als Geometrie oder Geografie eingegeben werden. Verwenden Sie ST_GeomFromText, ST_GeogFromText oder eine Umwandlung in den erforderlichen Typ (z.B. `ST_Segmentize('LINESTRING(1 2, 3 4)'::geometry, 0.5) )`

Beispiele

Segmentierung einer Zeile. Lange Segmente werden gleichmäßig aufgeteilt, kurze Segmente werden nicht aufgeteilt.

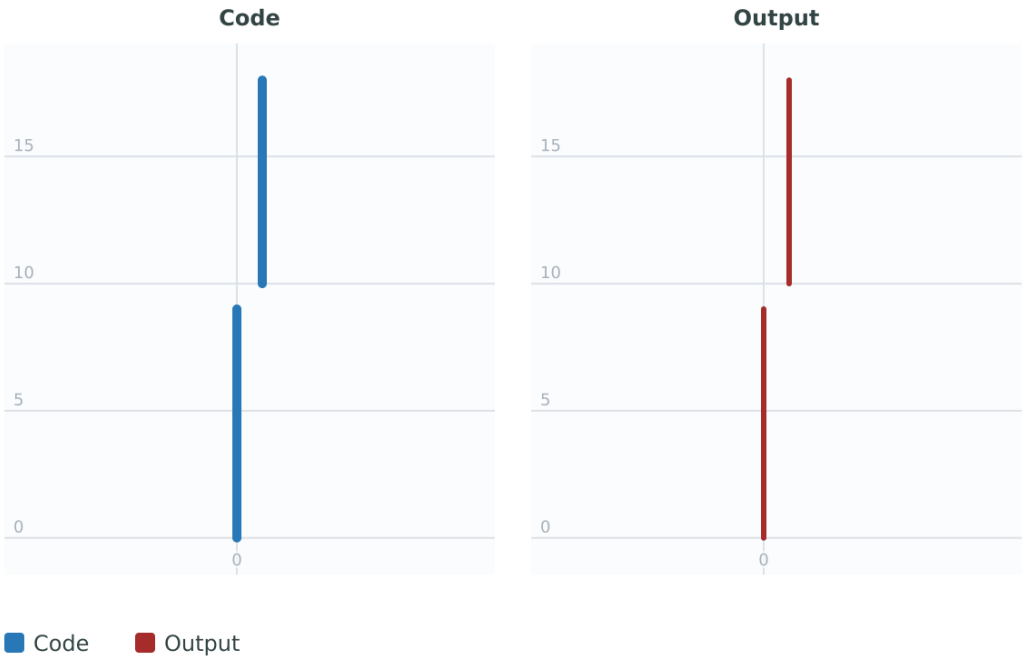
Code

```
SELECT ST_Segmentize('MULTILINESTRING((0 0,0 1,0 9),(1 10,1 18))'::geometry,
5);
```

Output

```
MULTILINESTRING((0 0,0 1,0 5,0 9),(1 10,1 14,1 18))
```

Figure



Segmentierung eines Polygons:

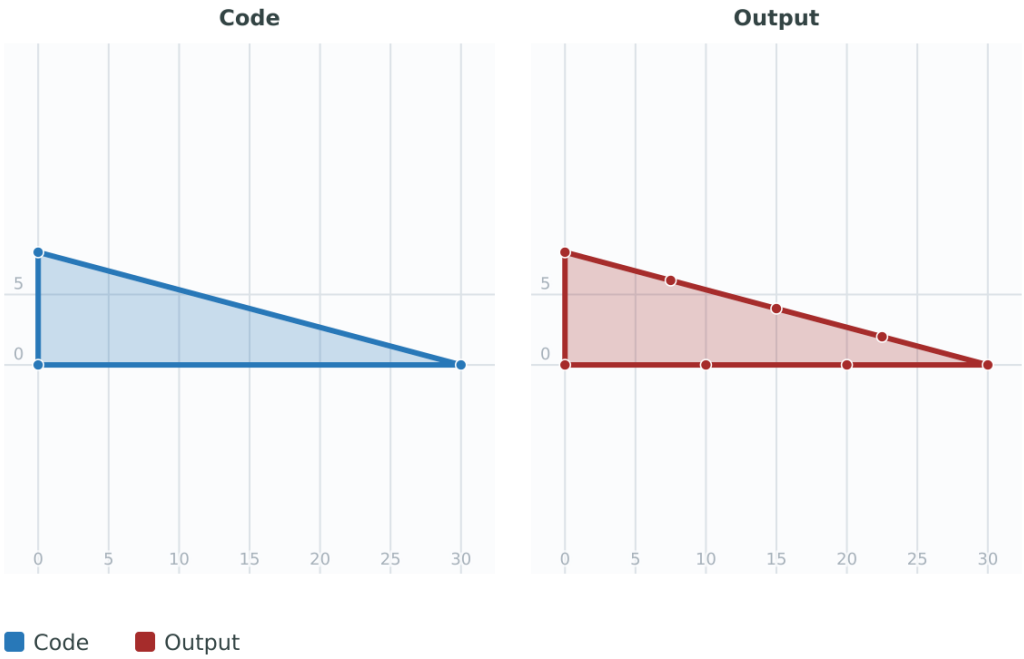
Code

```
SELECT ST_Segmentize('POLYGON((0 0,0 8,30 0,0 0))'::geometry,
        10);
```

Output

```
POLYGON((0 0,0 8,7.5 6,15 4,22.5 2,30 0,20 0,10 0,0 0))
```

Figure



Segmentierung einer geografischen Linie unter Verwendung einer maximalen Segmentlänge von 2000 Kilometern. Scheitelpunkte werden entlang des Großkreisbogens hinzugefügt, der die Endpunkte verbindet.

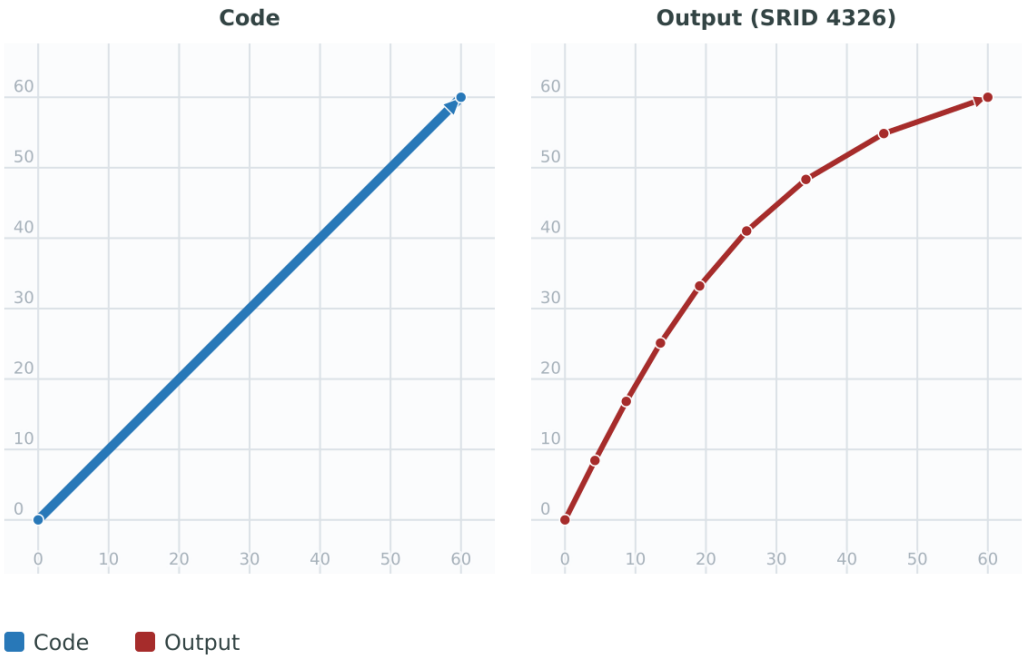
Code

```
SELECT ST_Segmentize(('LINESTRING (0 0,60 60)'::geography), 2000000);
```

Output

```
LINESTRING(0 0,4.252632294621186 8.43596525986862,8.69579947419404 16.824093489701564,13.550465473227048 ↩
    25.107950473646188,19.1066053508691 33.21091076089908,25.779290201459894 41.01711439406505,34.188839517966954 ↩
    48.337222885886,45.238153936612264 54.84733442373889,60 60)
```

Figure



Siehe auch

ST_LineSubstring

7.5.30 ST_SetPoint

ST_SetPoint — Einen Punkt eines Linienzuges durch einen gegebenen Punkt ersetzen.

Synopsis

geometry **ST_SetPoint**(geometry linestring, integer zerobasedposition, geometry point);

Beschreibung

Ersetzt den Punkt N eines Linienzuges mit dem gegebenen Punkt. Der Index beginnt mit 0. Negative Indizes werden rückwärts gezählt, sodasss -1 der letzte Punkt ist. Dies findet insbesondere bei Triggern verwendung, wenn man die Beziehung zwischen den Verbindungsstücken beim Verschieben von Knoten erhalten will

Verfügbarkeit: 1.1.0

Änderung: 2.3.0 : negatives Indizieren

Diese Funktion unterstützt 3d und lässt den Z-Index nicht fallen.

Beispiele

Change first point in line string from -1 3 to -1 1.

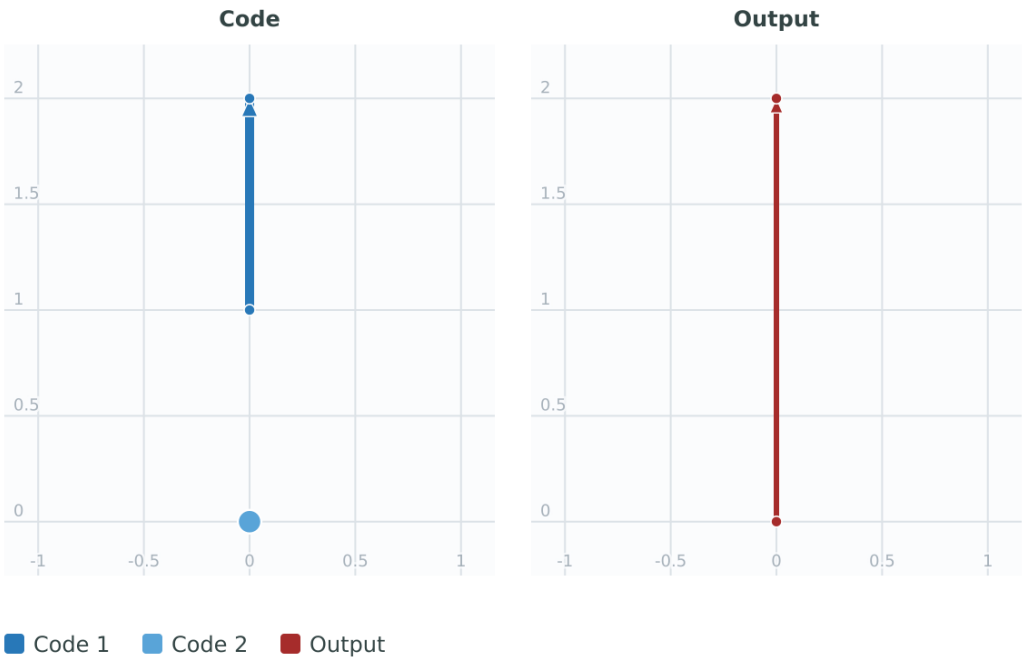
Code

```
SELECT ST_SetPoint('LINESTRING(-1 2,-1 3)', 0, 'POINT(-1 1)');
```

Output

```
LINESTRING(-1 1,-1 3)
```

Figure



Change last point in a line string (let's play with 3D linestring this time)

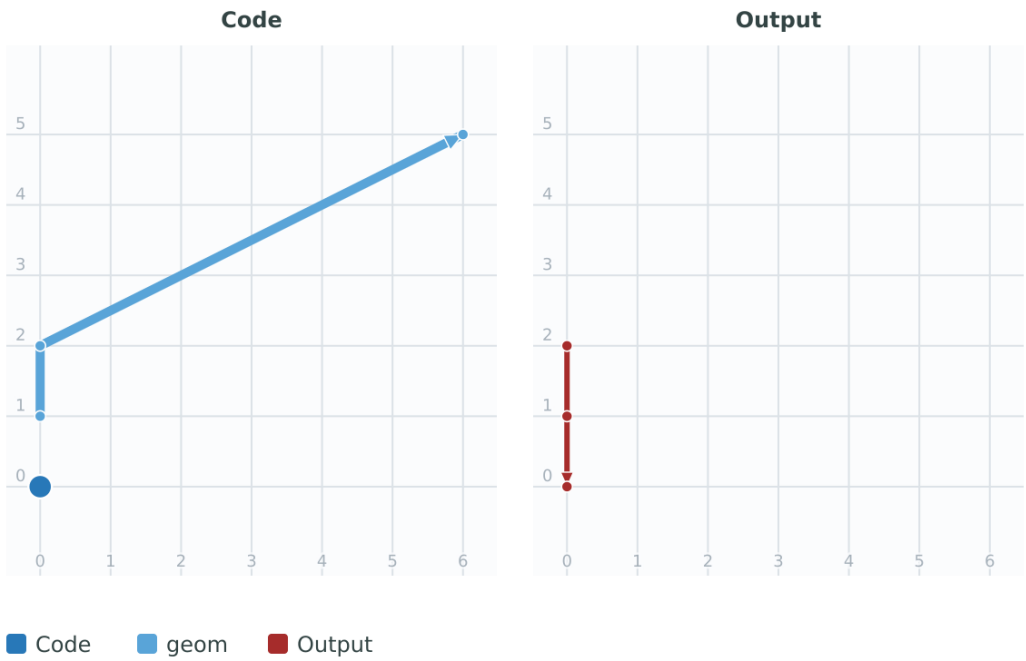
Code

```
SELECT ST_SetPoint(foo.geom, ST_NumPoints(foo.geom) - 1, 'POINT(-1 1 3)')
FROM (SELECT 'LINESTRING(-1 2 3,-1 3 4,5 6 7)'::geometry As geom) As foo;
```

Output

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

Figure



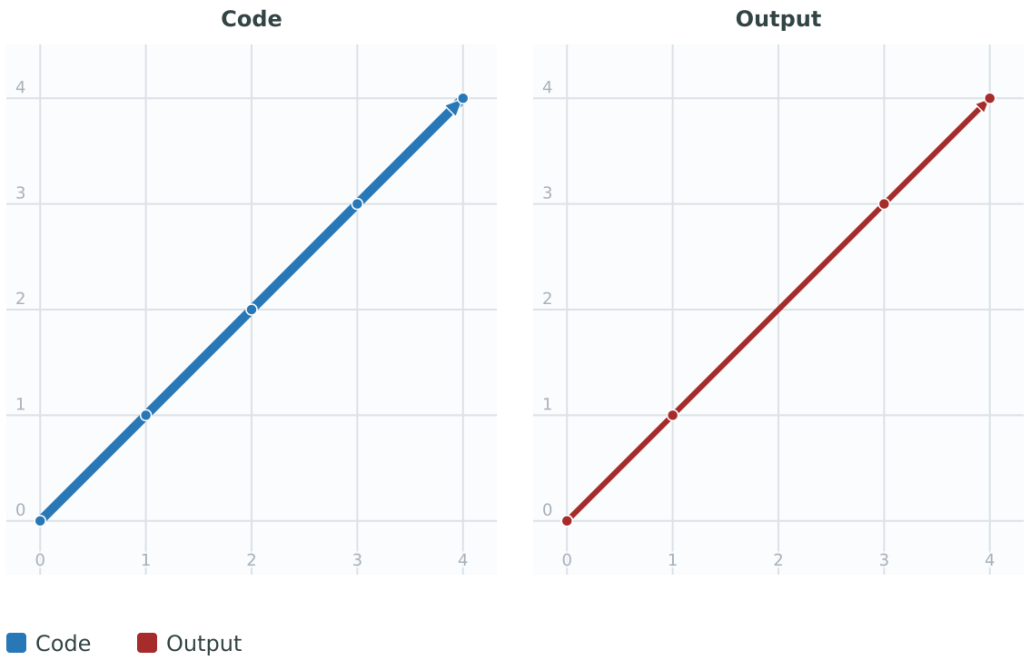
Code

```
SELECT ST_SetPoint(g, -3, p)
FROM ST_GeomFromText('LINESTRING(0 0,1 1,2 2,3 3,4 4)') AS g
    CROSS JOIN ST_PointN(g, 1) as p;
```

Output

```
LINESTRING(0 0,1 1,0 0,3 3,4 4)
```

Figure



Siehe auch

ST_AddPoint, ST_NPoints, ST_NumPoints, ST_PointN, ST_RemovePoint

7.5.31 ST_ShiftLongitude

ST_ShiftLongitude — Verschiebt die Längenkoordinaten einer Geometrie zwischen -180..180 und 0..360.

Synopsis

geometry **ST_ShiftLongitude**(geometry geom);

Beschreibung

Liest jeden Punkt/Vertex in einer Geometrie und verschiebt seine Längenkoordinate von -180..0 auf 180..360 und umgekehrt, wenn sie zwischen diesen Bereichen liegt. Diese Funktion ist symmetrisch, so dass das Ergebnis eine 0..360-Darstellung von -180..180-Daten und eine -180..180-Darstellung von 0..360-Daten ist.



Anmerkung
Dies ist nur für Daten mit Koordinaten in geografischer Länge/Breite nützlich, z. B. SRID 4326 (WGS 84 geografisch)



Warnung
Pre-1.3.4 Aufgrund eines Bugs funktionierte dies für MULTIPOINT nicht. Mit 1.3.4+ funktioniert es auch mit MULTIPOINT.



Diese Funktion unterstützt 3d und lässt den Z-Index nicht fallen.

Erweiterung: Mit 2.0.0 wurde die Unterstützung für polyedrische Oberflächen und TIN eingeführt.

Anmerkung: Vor 2.2.0 hieß diese Funktion "ST_Shift_Longitude"



Diese Funktion unterstützt polyedrische Flächen.



Diese Funktion unterstützt Dreiecke und dreieckige unregelmäßige Netzoberflächen (TIN).

Beispiele

This example performs a forward longitude shift on a single point.

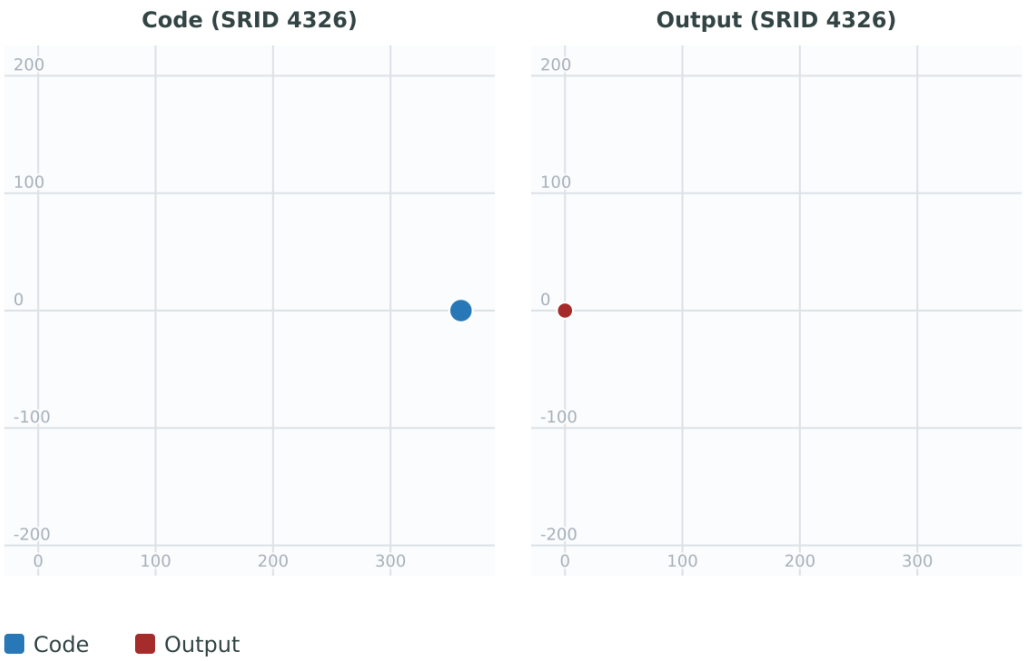
Code

```
SELECT ST_ShiftLongitude('SRID=4326;POINT(270 0)'::geometry)
```

Output

```
POINT(-90 0)
```

Figure



This example performs a reverse longitude shift on a single point.

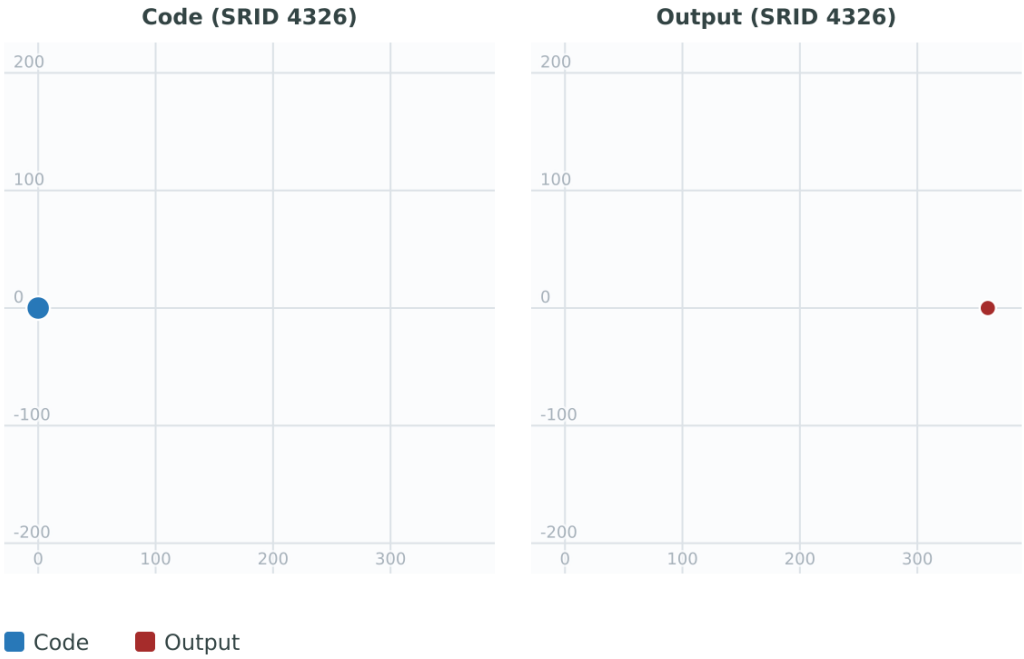
Code

```
SELECT ST_ShiftLongitude('SRID=4326;POINT(-90 0)'::geometry)
```

Output

```
POINT(270 0)
```

Figure



This example shows that LineStrings are shifted only where coordinates require it.

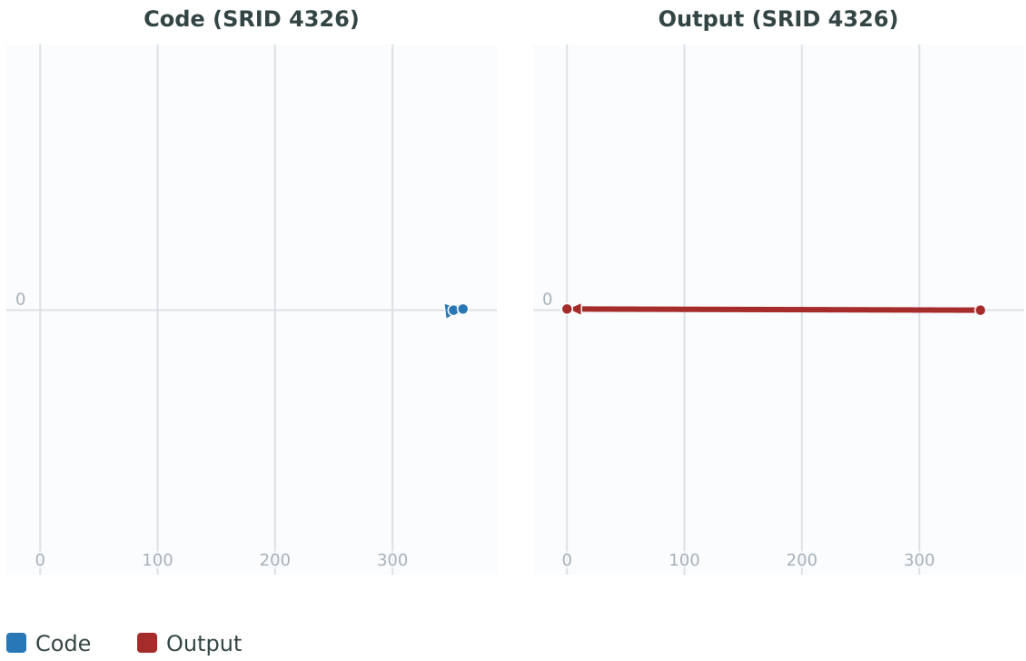
Code

```
SELECT ST_ShiftLongitude('SRID=4326;LINESTRING(174 12,182 13)'::geometry)
```

Output

```
LINESTRING(174 12,-178 13)
```

Figure



Siehe auch

ST_WrapX

7.5.32 ST_WrapX

ST_WrapX — Versammelt eine Geometrie um einen X-Wert

Synopsis

geometry **ST_WrapX**(geometry geom, float8 wrap, float8 move);

Beschreibung

Diese Funktion teilt die eingegebenen Geometrien auf und verschiebt dann jede resultierende Komponente, die rechts (bei negativem 'move') oder links (bei positivem 'move') von der gegebenen 'wrap'-Linie fällt, in die durch den 'move'-Parameter angegebene Richtung und fügt die Teile schließlich wieder zusammen.



Anmerkung
Nützlich, um eine Eingabe in Länge und Breite neu zu zentrieren, damit die wesentlichen Geoobjekte nicht von einer Seite bis zur anderen abgebildet werden.

Verfügbarkeit: 2.3.0 erfordert GEOS

 Diese Funktion unterstützt 3d und lässt den Z-Index nicht fallen.

Beispiele

Move all components whose bounding boxes fall completely to the left of `x = 0` by +360.

Code

```
SELECT ST_WrapX(geom, 0, 360);
```

Move all components whose bounding boxes fall completely to the left of `x = -30` by +360.

Code

```
SELECT ST_WrapX(geom, -30, 360);
```

Siehe auch

ST_ShiftLongitude

7.5.33 ST_SnapToGrid

ST_SnapToGrid — Fängt alle Punkte der Eingabegeometrie auf einem regelmäßigen Gitter.

Synopsis

geometry **ST_SnapToGrid**(geometry geomA, float originX, float originY, float sizeX, float sizeY);
geometry **ST_SnapToGrid**(geometry geomA, float sizeX, float sizeY);
geometry **ST_SnapToGrid**(geometry geomA, float size);
geometry **ST_SnapToGrid**(geometry geomA, geometry pointOrigin, float sizeX, float sizeY, float sizeZ, float sizeM);

Beschreibung

Variante 1, 2 und 3: Fängt alle Punkte der Eingabegeometrie auf den Gitterpunkten, die durch Ursprung und Gitterkästchengröße festgelegt sind. Aufeinanderfolgende Punkte, die in dasselbe Gitterkästchen fallen, werden gelöscht, wobei NULL zurückgegeben wird, wenn nicht mehr genug Punkte für den jeweiligen geometrischen Datentyp vorhanden sind. Collapsed geometries in a collection are stripped from it. Kollabierte Geometrien einer Kollektion werden von dieser entfernt. Nützlich um die Genauigkeit zu verringern.

Variante 4: wurde mit 1.1.0 eingeführt - Fängt alle Punkte der Eingabegeometrie auf den Gitterpunkten, welche durch den Ursprung des Gitters (der zweite Übergabewert muss ein Punkt sein) und die Gitterkästchengröße bestimmt sind. Geben Sie 0 als Größe für jene Dimension an, die nicht auf den Gitterpunkten gefangen werden soll.



Anmerkung
Die zurückgegebene Geometrie kann ihre Simplizität verlieren (siehe ST_IsSimple).



Anmerkung
Vor Release 1.1.0 gab diese Funktion immer eine 2D-Geometrie zurück. Ab 1.1.0 hat die zurückgegebene Geometrie dieselbe Dimensionalität wie die Eingabegemetrie, wobei höhere Dimensionen unangetastet bleiben. Verwenden Sie die Version, welche einen zweiten geometrischen Übergabewert annimmt, um sämtliche Grid-Dimensionen zu bestimmen.

Verfügbarkeit: 1.0.0RC1

Verfügbarkeit: 1.1.0, Unterstützung für Z und M

Diese Funktion unterstützt 3d und lässt den Z-Index nicht fallen.

Beispiele

Snap a geometry column to a precision grid of 10^-3.

Code

```
UPDATE mytable
   SET geom = ST_SnapToGrid(geom, 0.001);
```

Display how the same grid size rounds a LineString.

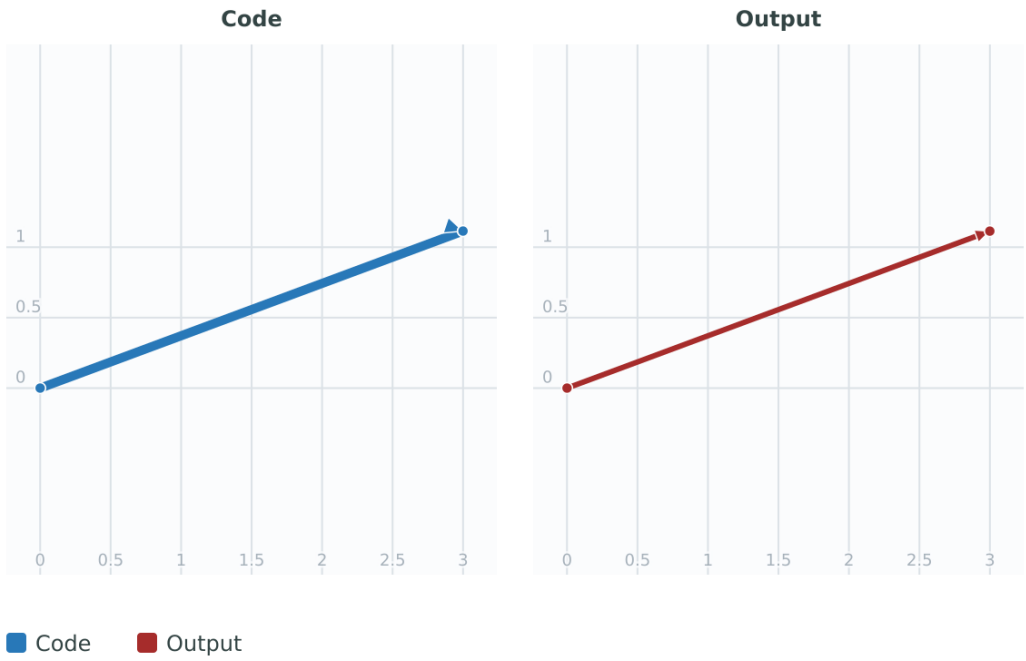
Code

```
SELECT ST_SnapToGrid('LINESTRING(1.1115678 2.123,4.111111 3.2374897,4.11112 3.23748667)':: ↩
    geometry,
        0.001);
```

Output

```
LINESTRING(1.112 2.123,4.111 3.237)
```

Figure



This example snaps a 4D geometry.

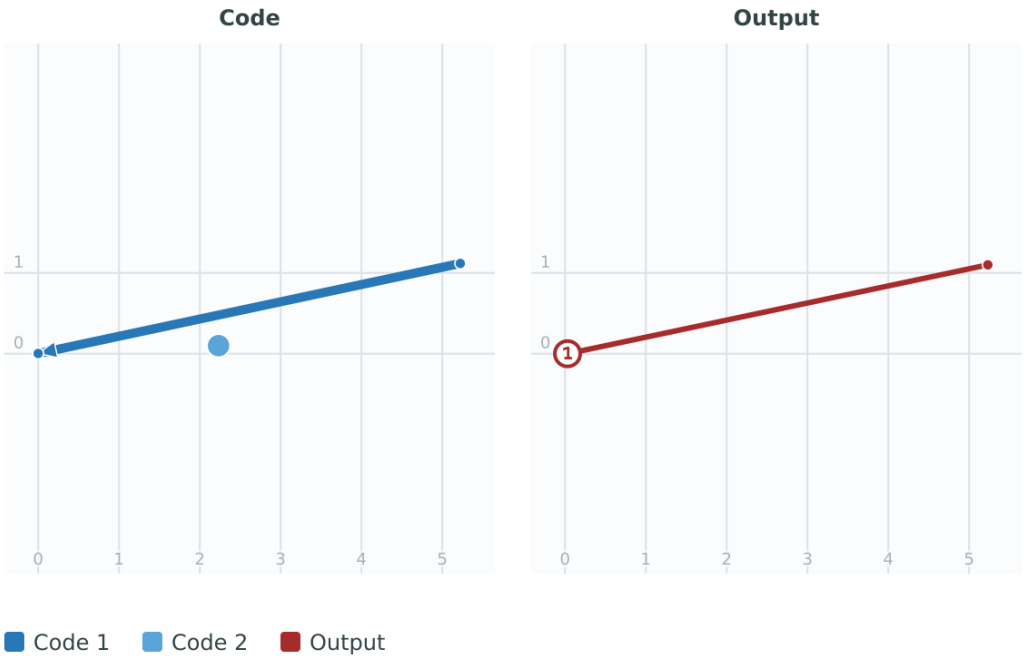
Code

```
SELECT ST_SnapToGrid('LINESTRING(-1.1115678 2.123 2.3456 1.11111,
        4.111111 3.2374897 3.1234 1.1111,-1.11111112 2.123 2.3456 1.1111112)'::geometry,
        'POINT(1.12 2.22 3.2 4.4444)',
        0.1, 0.1, 0.1, 0.01);
```

Output

```
LINESTRING(-1.08 2.12 2.3 1.1144,4.12 3.22 3.1 1.1144,-1.08 2.12 2.3 1.1144)
```

Figure



With a 4D geometry - the ST_SnapToGrid(geom,size) only touches x and y coords but keeps m and z the same.

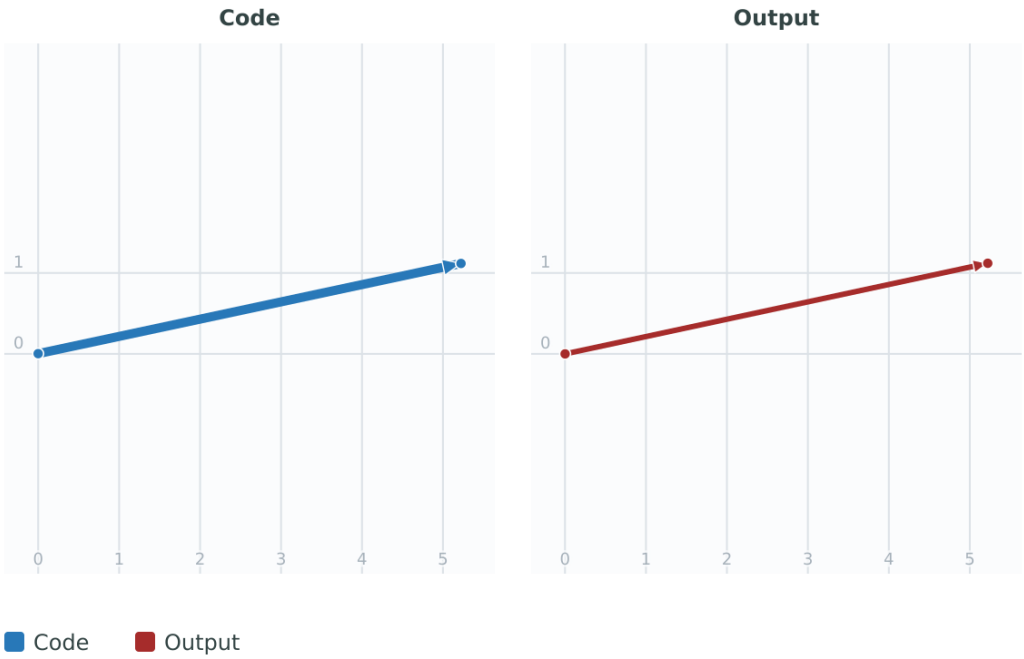
Code

```
SELECT ST_SnapToGrid('LINESTRING(-1.1115678 2.123 3 2.3456,
        4.111111 3.2374897 3.1234 1.1111)'::geometry,
       0.01);
```

Output

```
LINESTRING(-1.11 2.12 3 2.3456,4.11 3.24 3.1234 1.1111)
```

Figure



Siehe auch

ST_Snap, ST_AsEWKT, ST_AsText, ST_GeomFromText, ST_GeomFromEWKT, ST_Simplify

7.5.34 ST_Snap

ST_Snap — Fängt die Segmente und Knoten einer Eingabegeometrie an den Knoten einer Referenzgeometrie.

Synopsis

geometry **ST_Snap**(geometry input, geometry reference, float tolerance);

Beschreibung

Fängt die Knoten und Segmente einer Geometrie an den Knoten einer anderen Geometrie. Eine Entfernungstoleranz bestimmt, wo das Fangen durchgeführt wird. Die Ergebnisgeometrie ist die Eingabegeometrie mit gefangenen Knoten. Wenn kein Fangen auftritt, wird die Eingabegeometrie unverändert ausgegeben..

Eine Geometrie an einer anderen zu fangen, kann die Robustheit von Überlagerungs-Operationen verbessern, indem nahe zusammenfallende Kanten beseitigt werden (diese verursachen Probleme bei der Knoten- und Verschneidungsberechnung).

Übermäßiges Fangen kann zu einer invaliden Topologie führen. Die Anzahl und der Ort an dem Knoten sicher gefangen werden können wird mittels Heuristik bestimmt. Dies kann allerdings dazu führen, dass einige potentielle Knoten nicht gefangen werden.



Anmerkung
Die zurückgegebene Geometrie kann ihre Simplizität (see ST_IsSimple) und Valididät (see ST_IsValid) verlieren.

Wird vom GEOS Modul ausgeführt

Verfügbarkeit: 2.0.0

Beispiele

A MultiPolygon snapped to a LineString at 1.01 times their distance.

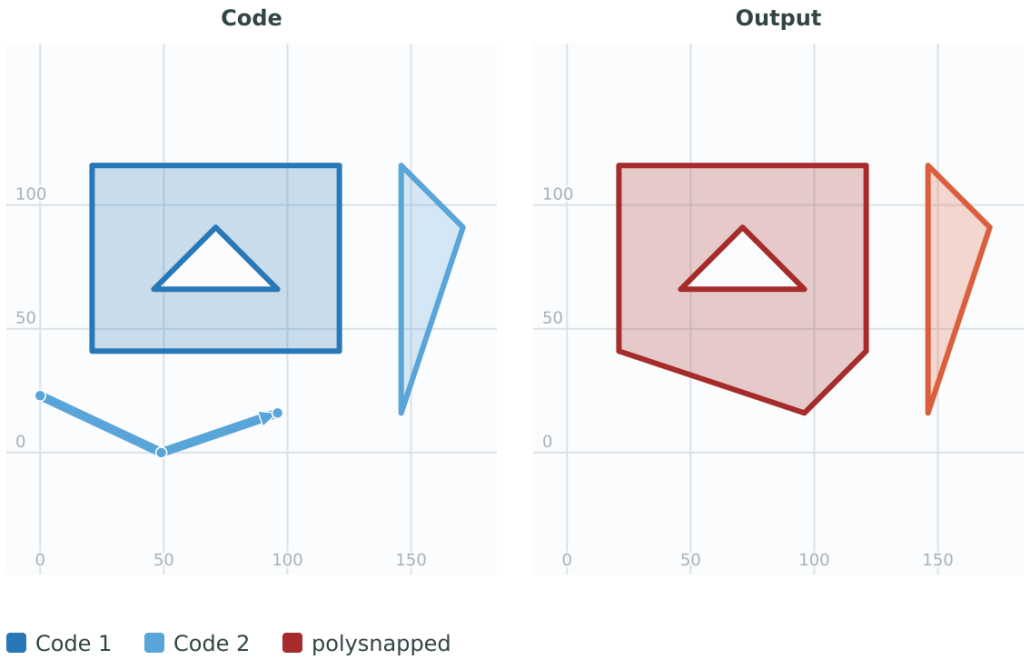
Code

```
SELECT ST_Snap(poly, line, ST_Distance(poly, line)*1.01) AS polysnapped
FROM (SELECT
   ST_GeomFromText('MULTIPOLYGON(
     ((26 125,26 200,126 200,126 125,26 125 ),
      ( 51 150,101 150,76 175,51 150 )),
      (( 151 100,151 200,176 175,151 100 )))') As poly,
       ST_GeomFromText('LINESTRING (5 107,54 84,101 100)') As line
    ) As foo;
```

Output

```
MULTIPOLYGON(((26 125,26 200,126 200,126 125,101 100,26 125),
(51 150,101 150,76 175,51 150)),((151 100,151 200,176 175,151 100)))
```

Figure



A MultiPolygon snapped to a LineString at 1.25 times their distance.

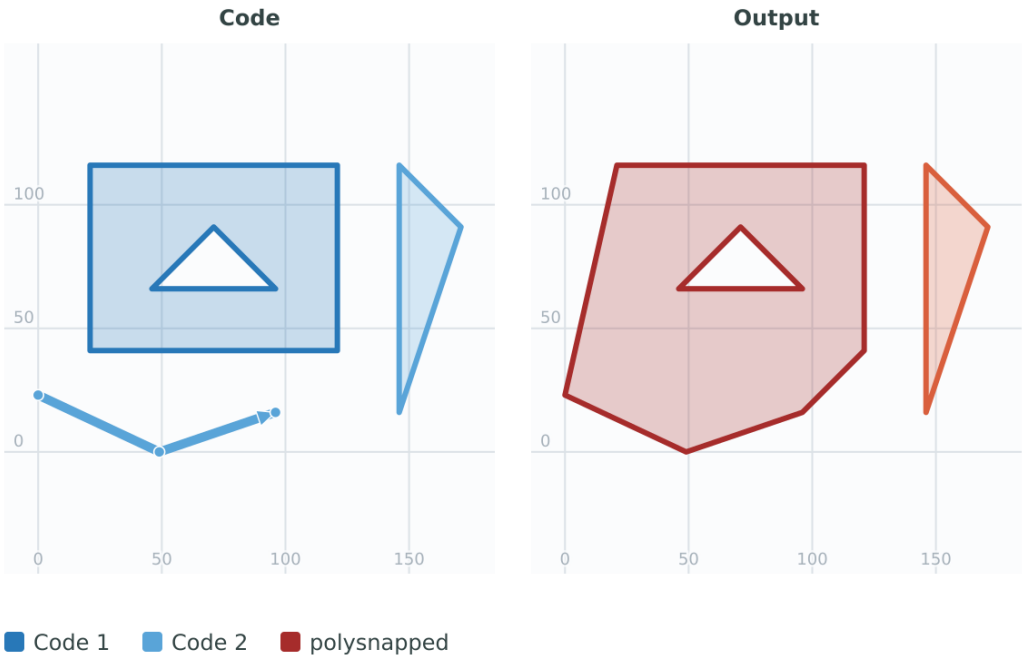
Code

```
SELECT ST_Snap(poly, line, ST_Distance(poly, line)*1.25) AS polysnapped
FROM (SELECT
  ST_GeomFromText('MULTIPOLYGON(
    (( 26 125,26 200,126 200,126 125,26 125 ),
      ( 51 150,101 150,76 175,51 150 )),
      (( 151 100,151 200,176 175,151 100 )))') As poly,
       ST_GeomFromText('LINESTRING (5 107,54 84,101 100)') As line
    ) As foo;
```

Output

```
MULTIPOLYGON(((5 107,26 200,126 200,126 125,101 100,54 84,5 107),
(51 150,101 150,76 175,51 150)),((151 100,151 200,176 175,151 100)))
```

Figure



The LineString snapped to the original MultiPolygon at 1.01 times their distance.

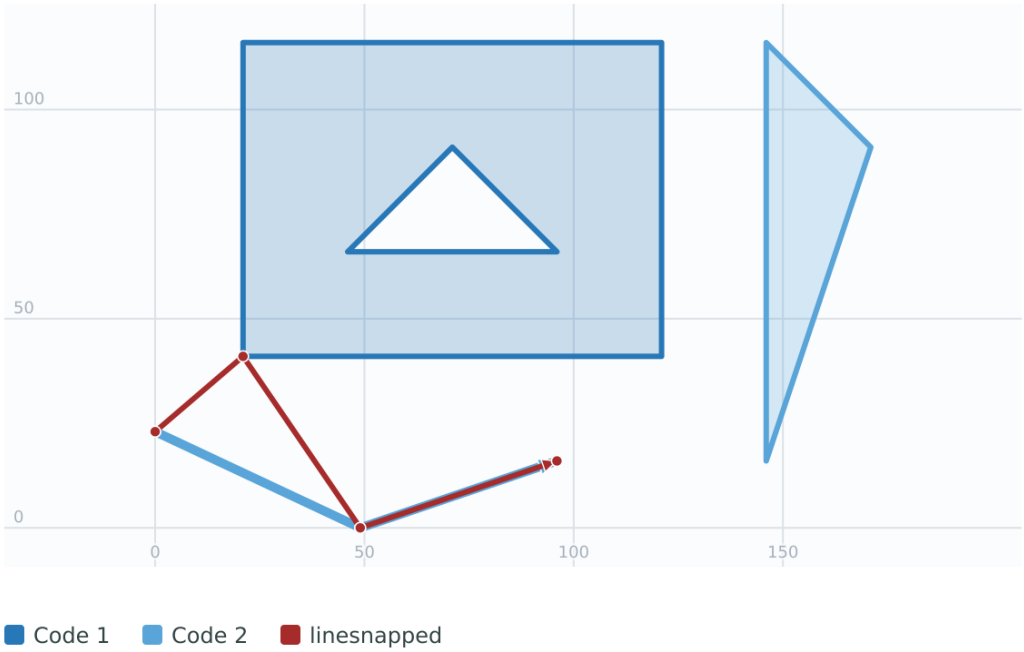
Code

```
SELECT ST_Snap(line, poly, ST_Distance(poly, line)*1.01) AS linesnapped
FROM (SELECT
  ST_GeomFromText('MULTIPOLYGON(
     ((26 125,26 200,126 200,126 125,26 125),
      (51 150,101 150,76 175,51 150 )),
      ((151 100,151 200,176 175,151 100)))') As poly,
       ST_GeomFromText('LINESTRING (5 107,54 84,101 100)') As line
    ) As foo;
```

Output

```
LINESTRING(5 107,26 125,54 84,101 100)
```

Figure



The LineString snapped to the original MultiPolygon at 1.25 times their distance.

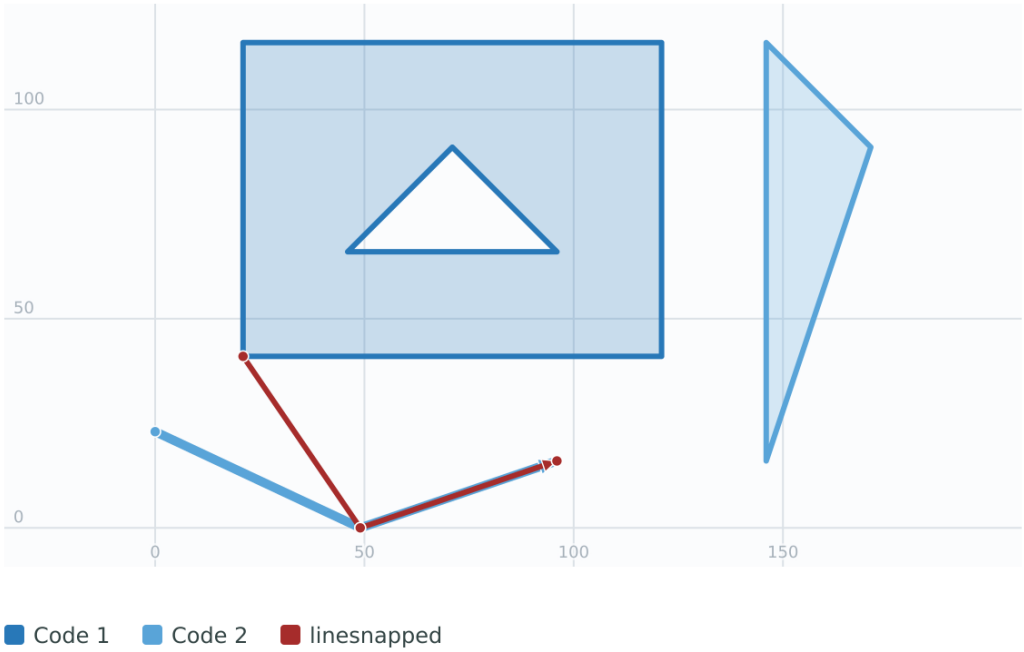
Code

```
SELECT ST_Snap(line, poly, ST_Distance(poly, line)*1.25) AS linesnapped
FROM (SELECT
  ST_GeomFromText('MULTIPOLYGON(
     (( 26 125,26 200,126 200,126 125,26 125 ),
      (51 150,101 150,76 175,51 150 )),
      ((151 100,151 200,176 175,151 100 )))') As poly,
       ST_GeomFromText('LINESTRING (5 107,54 84,101 100)') As line
    ) As foo;
```

Output

```
LINESTRING(26 125,54 84,101 100)
```

Figure



Siehe auch

ST_SnapToGrid

7.5.35 ST_SwapOrdinates

ST_SwapOrdinates — Gibt eine Version der Ausgangsgeometrie zurück, in der die angegebenen Ordinatenwerte ausgetauscht werden.

Synopsis

geometry **ST_SwapOrdinates**(geometry geom, cstring ords);

Beschreibung

Gibt eine Version der Ausgangsgeometrie zurück, in der die angegebenen Ordinaten ausgetauscht werden.

Der `ords` Parameter ist eine Zeichenkette aus 2 Zeichen, welche die Ordinate benennt die getauscht werden soll. Gültige Bezeichnungen sind: x,y,z und m.

Verfügbarkeit: 2.2.0

Diese Methode unterstützt kreisförmige Strings und Kurven.

Diese Funktion unterstützt 3d und lässt den Z-Index nicht fallen.

Diese Funktion unterstützt M-Koordinaten.

Diese Funktion unterstützt polyedrische Flächen.

Diese Funktion unterstützt Dreiecke und dreieckige unregelmäßige Netzoberflächen (TIN).

Beispiel

This example scales the M value by 2.

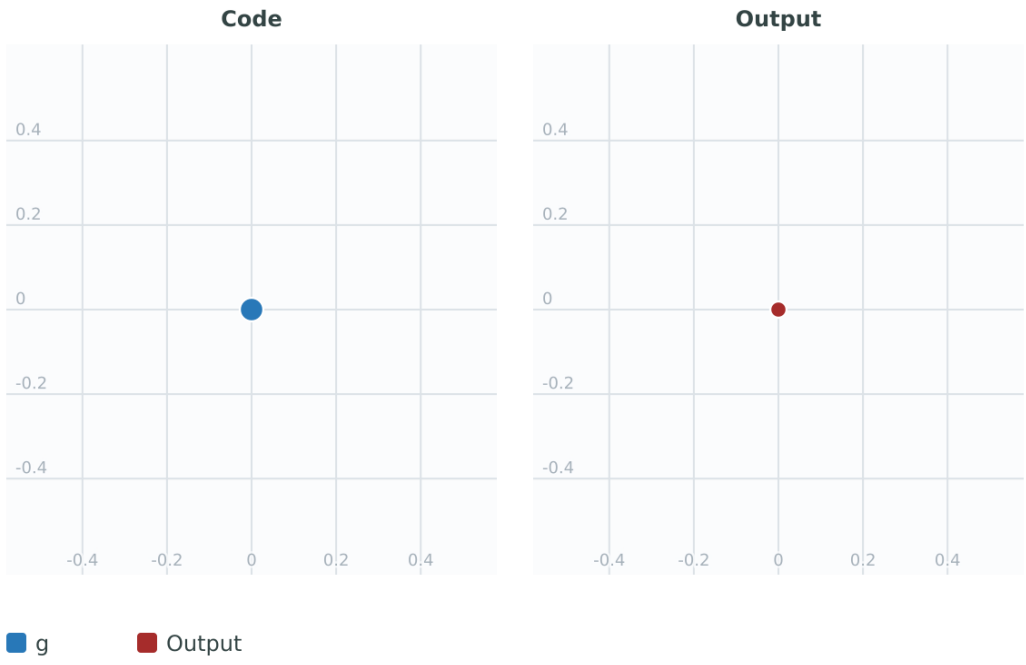
Code

```
SELECT ST_SwapOrdinates(ST_Scale(ST_SwapOrdinates(g, 'xm'),
      2, 1
    ),
  'xm') FROM ( SELECT 'POINT ZM (0 0 0 2)'::geometry g ) foo;
```

Output

```
POINT ZM (0 0 0 4)
```

Figure



Siehe auch

ST_FlipCoordinates

7.6 Geometrievalidierung

7.6.1 ST_IsValid

ST_IsValid — Prüft, ob eine Geometrie in 2D wohlgeformt ist.

Synopsis

boolean **ST_IsValid**(geometry g);
boolean **ST_IsValid**(geometry g, integer flags);

Beschreibung

Prüft, ob ein ST_Geometry-Wert wohlgeformt und in 2D gemäß den OGC-Regeln gültig ist. Bei Geometrien mit 3 und 4 Dimensionen wird die Gültigkeit weiterhin nur in 2 Dimensionen geprüft. Für Geometrien, die ungültig sind, wird ein PostgreSQL-NOTICE ausgegeben, der die Gründe für die Ungültigkeit angibt.

Für die Version mit dem Parameter `flags` sind die unterstützten Werte in ST_IsValidDetail dokumentiert. Diese Version druckt keinen HINWEIS, der die Ungültigkeit erklärt.

Weitere Informationen über die Definition der Geometriegültigkeit finden Sie unter Abschnitt 4.4



Anmerkung
SQL-MM definiert das Ergebnis von ST_IsValid(NULL) als 0, während PostGIS NULL zurückgibt.

Wird vom GEOS Modul ausgeführt

Die Version, die Flaggen akzeptiert, ist ab 2.0.0 verfügbar.



Diese Methode implementiert die OGC Simple Features Implementation Specification for SQL 1.1.



Diese Methode setzt die SQL/MM-Spezifikation um. SQL-MM 3: 5.1.9



Anmerkung
Weder die OGC-SFS- noch die SQL-MM-Spezifikationen enthalten ein Flag-Argument für ST_IsValid. Das Flag ist eine PostGIS-Erweiterung.

Beispiele

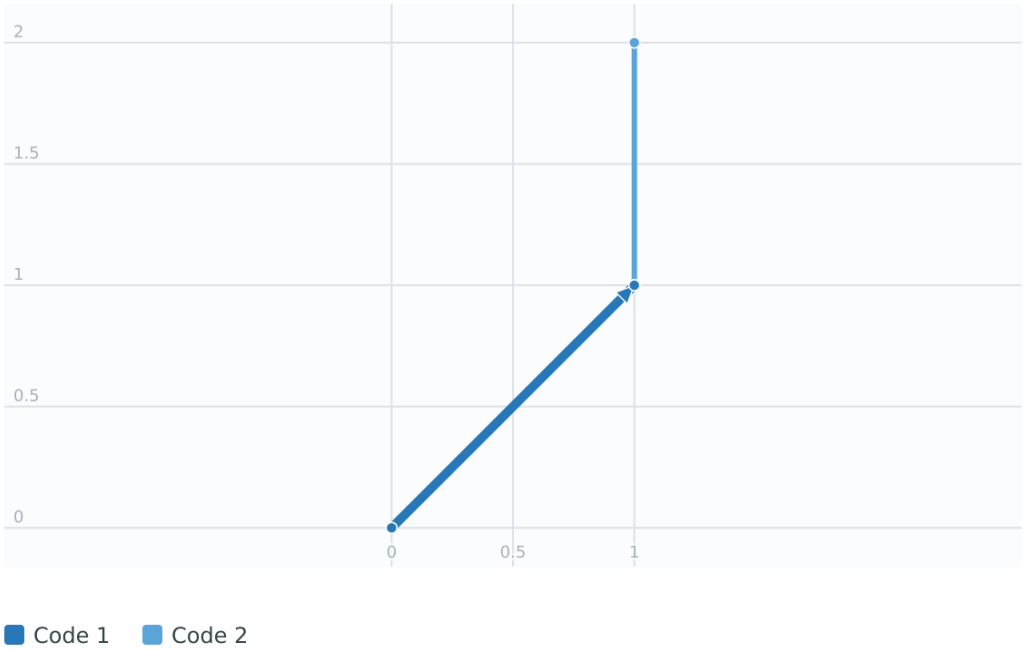
Code

```
SELECT ST_IsValid('LINESTRING(0 0,1 1)') As good_line,
ST_IsValid('POLYGON((0 0,1 1,1 2,1 1,0 0))') As bad_poly;
```

Output

```
NOTICE:  Self-intersection at or near point 0 0
 good_line | bad_poly
-----------+----------
 t         | f
```

Figure



Find tables with invalid geometries in registered geometry columns.

Code

```
CREATE TEMP TABLE invalid_geometry_columns (
    table_name text,
    column_name text,
    invalid_count bigint
);

DO $$
DECLARE
    rec record;
    invalid_count bigint;
BEGIN
    FOR rec IN
        SELECT f_table_schema, f_table_name, f_geometry_column
        FROM geometry_columns
        ORDER BY f_table_schema, f_table_name, f_geometry_column
    LOOP
        EXECUTE format(
            'SELECT count(*) FROM %I.%I WHERE NOT ST_IsValid(%I, 0)',
            rec.f_table_schema,
            rec.f_table_name,
            rec.f_geometry_column
        ) INTO invalid_count;

        IF invalid_count
> 0 THEN
            INSERT INTO invalid_geometry_columns
            VALUES (
                format('%I.%I', rec.f_table_schema, rec.f_table_name),
                rec.f_geometry_column,
                invalid_count
            );
        END IF;
    END LOOP;
```

```
END;
$$;

SELECT * FROM invalid_geometry_columns;
```

Siehe auch

ST_IsSimple, ST_IsValidReason, ST_IsValidDetail,

7.6.2 ST_IsValidDetail

ST_IsValidDetail — Gibt eine Zeile `valid_detail` zurück, die angibt, ob eine Geometrie gültig ist oder, falls nicht, einen Grund und einen Ort.

Synopsis

valid_detail **ST_IsValidDetail**(geometry geom, integer flags);

Beschreibung

Gibt eine Zeile `valid_detail` zurück, die einen booleschen Wert (`valid`) enthält, der angibt, ob eine Geometrie gültig ist, einen varchar-Wert (`reason`), der einen Grund angibt, warum sie ungültig ist, und eine Geometrie (`location`), die angibt, wo sie ungültig ist.

Nützlich zur Verbesserung der Kombination von ST_IsValid und ST_IsValidReason, um einen detaillierten Bericht über ungültige Geometrien zu erstellen.

Der optionale Parameter `flags` ist ein Bitfeld. Er kann die folgenden Werte haben:

- 0: Verwendung der üblichen OGC SFS-Gültigkeitssemantik.
- 1: Bestimmte Arten von sich selbst berührenden Ringen (umgekehrte Schalen und umgekehrte Löcher) als gültig betrachten. Dies ist auch als "ESRI-Flagge" bekannt, da dies das von diesen Werkzeugen verwendete Gültigkeitsmodell ist. Beachten Sie, dass dies nach dem OGC-Modell ungültig ist.

Wird vom GEOS Modul ausgeführt

Verfügbarkeit: 2.0.0

Beispiele

This example returns the first three rejects from a successful quintuplet experiment.

Code

```
SELECT gid, reason(ST_IsValidDetail(geom)), location(ST_IsValidDetail(geom)) as location
FROM
(SELECT ST_MakePolygon(ST_ExteriorRing(e.buff), array_agg(f.line)) As geom, gid
FROM (SELECT ST_Buffer(ST_Point(x1*10,y1), z1) As buff, x1*10 + y1*100 + z1*1000 As gid
    FROM generate_series(-4,6) x1
    CROSS JOIN generate_series(2,5) y1
    CROSS JOIN generate_series(1,8) z1
    WHERE x1
> y1*0.5 AND z1 < x1*y1) As e
    INNER JOIN (SELECT ST_Translate(ST_ExteriorRing(ST_Buffer(ST_Point(x1*10,y1), z1)),y1*1, ↩
     z1*2) As line
```

```
    FROM generate_series(-3,6) x1
    CROSS JOIN generate_series(2,5) y1
    CROSS JOIN generate_series(1,10) z1
    WHERE x1
> y1*0.75 AND z1 < x1*y1) As f
ON (ST_Area(e.buff)
> 78 AND ST_Contains(e.buff, f.line))
GROUP BY gid, e.buff) As quintuplet_experiment
WHERE ST_IsValid(geom) = false
ORDER BY gid
LIMIT 3;
```

Output

```
 gid  |      reason       |  location
------+-------------------+-------------
 5330 | Self-intersection | POINT(32 5)
 5340 | Self-intersection | POINT(42 5)
 5350 | Self-intersection | POINT(52 5)
```

This example reports a valid LineString.

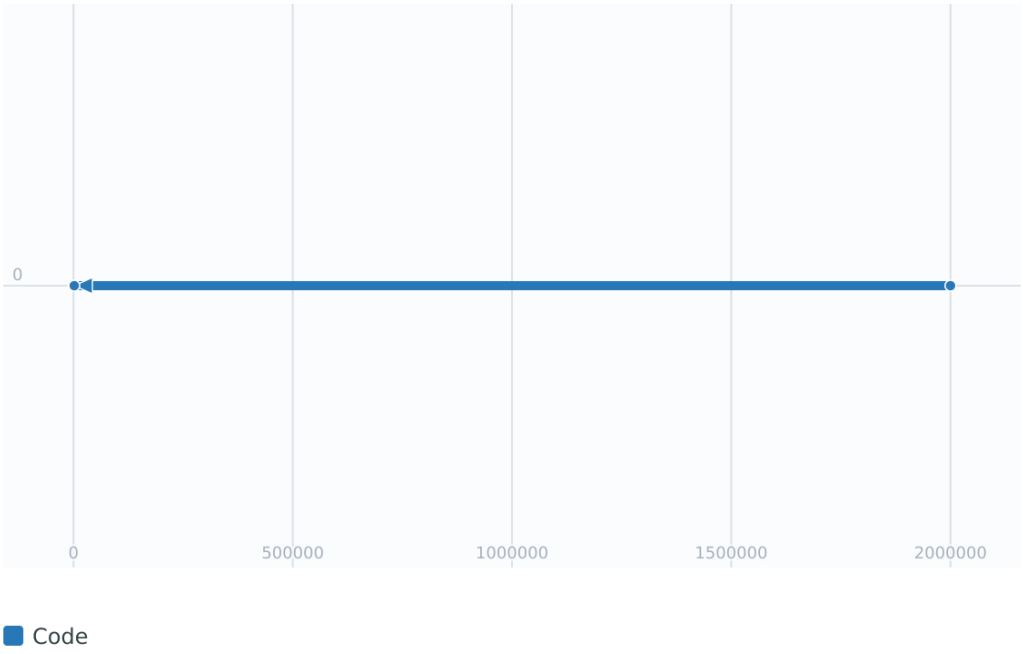
Code

```
SELECT * FROM ST_IsValidDetail('LINESTRING(220227 150406,2220227 150407,222020 150410)');
```

Output

```
 valid | reason | location
-------+--------+----------
 t     |        |
```

Figure



Siehe auch

ST_IsValid, ST_IsValidReason

7.6.3 ST_IsValidReason

ST_IsValidReason — Gibt einen Text zurück, der angibt, ob eine Geometrie gültig ist, oder einen Grund für die Ungültigkeit.

Synopsis

text **ST_IsValidReason**(geometry geomA);
text **ST_IsValidReason**(geometry geomA, integer flags);

Beschreibung

Gibt einen Text zurück, der angibt, ob eine Geometrie gültig ist, oder, falls ungültig, einen Grund dafür.

Nützlich in Kombination mit ST_IsValid, um einen detaillierten Bericht über ungültige Geometrien und Gründe zu erstellen.

Erlaubte `Flaggen` sind in ST_IsValidDetail dokumentiert.

Wird vom GEOS Modul ausgeführt

Verfügbarkeit: 1.4

Verfügbarkeit: Version 2.0 mit Flaggen.

Beispiele

This example uses an invalid bow-tie polygon.

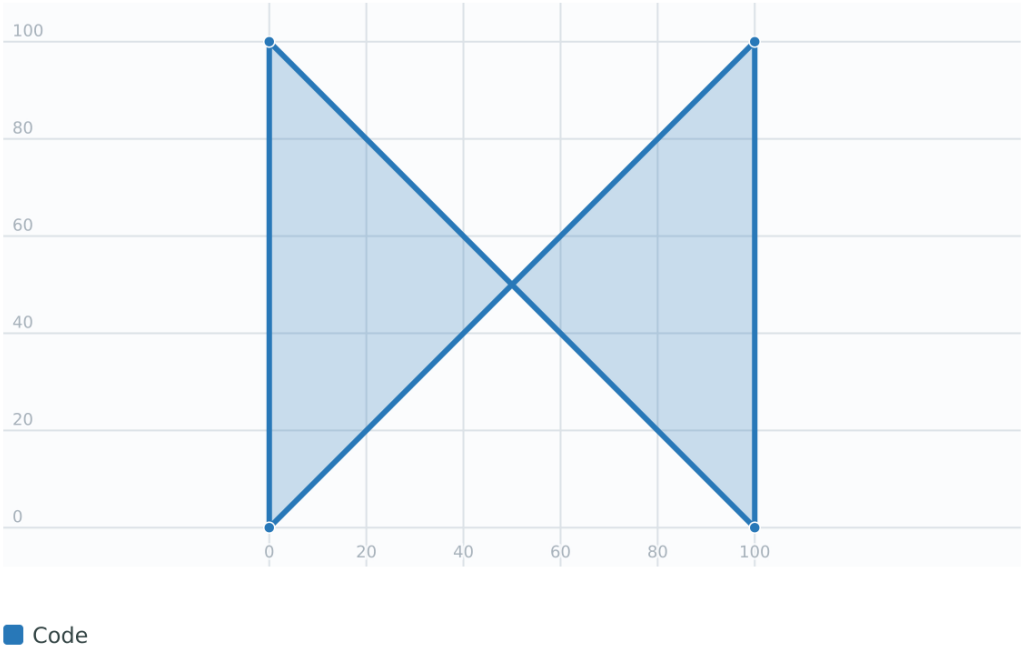
Code

```
SELECT ST_IsValidReason(
    'POLYGON ((100 200,100 100,200 200,
     200 100,100 200))'::geometry) as validity_info;
```

Output

```
Self-intersection[150 150]
```

Figure



This example returns the first three rejects from a successful quintuplet experiment.

Code

```
SELECT gid, ST_IsValidReason(geom) as validity_info
FROM
(SELECT ST_MakePolygon(ST_ExteriorRing(e.buff), array_agg(f.line)) As geom, gid
FROM (SELECT ST_Buffer(ST_Point(x1*10,y1), z1) As buff, x1*10 + y1*100 + z1*1000 As gid
    FROM generate_series(-4,6) x1
    CROSS JOIN generate_series(2,5) y1
    CROSS JOIN generate_series(1,8) z1
    WHERE x1
> y1*0.5 AND z1 < x1*y1) As e
    INNER JOIN (SELECT ST_Translate(ST_ExteriorRing(ST_Buffer(ST_Point(x1*10,y1), z1)),y1*1, ↩
     z1*2) As line
    FROM generate_series(-3,6) x1
    CROSS JOIN generate_series(2,5) y1
    CROSS JOIN generate_series(1,10) z1
    WHERE x1
> y1*0.75 AND z1 < x1*y1) As f
ON (ST_Area(e.buff)
> 78 AND ST_Contains(e.buff, f.line))
GROUP BY gid, e.buff) As quintuplet_experiment
WHERE ST_IsValid(geom) = false
ORDER BY gid
LIMIT 3;
```

Output

```
 gid  |      validity_info
------+-------------------------
 5330 | Self-intersection [32 5]
 5340 | Self-intersection [42 5]
 5350 | Self-intersection [52 5]
```

Simple example.

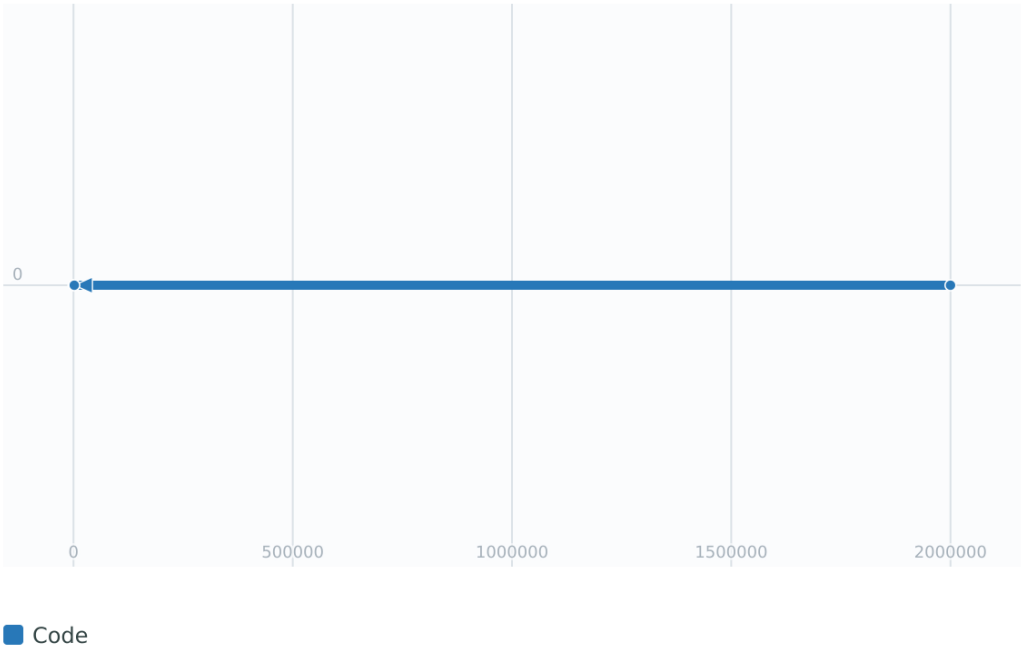
Code

```
SELECT ST_IsValidReason('LINESTRING(220227 150406,2220227 150407,222020 150410)');
```

Output

```
 st_isvalidreason
------------------
 Valid Geometry
```

Figure



Siehe auch

ST_IsValid, ST_Summary

7.6.4 ST_MakeValid

ST_MakeValid — Versucht, eine ungültige Geometrie gültig zu machen, ohne dass Scheitelpunkte verloren gehen.

Synopsis

geometry **ST_MakeValid**(geometry input);
geometry **ST_MakeValid**(geometry input, text params);

Beschreibung

Die Funktion versucht, eine gültige Darstellung einer gegebenen ungültigen Geometrie zu erstellen, ohne dass einer der Eingabepunkte verloren geht. Gültige Geometrien werden unverändert zurückgegeben.

Unterstützte Eingaben sind: PUNKTE, MEHRPUNKTE, LINIENSTRÄNGE, MEHRLINIENSTRÄNGE, POLYGONEN, MEHRPOLYGONEN und GEOMETRIESAMMLUNGEN, die eine beliebige Mischung dieser Elemente enthalten.

Bei vollständiger oder teilweiser Dimensionsreduzierung kann die Ausgangsgeometrie eine Sammlung von Geometrien mit geringerer bis gleicher Dimension oder eine Geometrie mit geringerer Dimension sein.

Einzelne Polygone können im Falle von Selbstüberschneidungen zu Multi-Geometrien werden.

Mit dem Argument `params` kann eine Optionszeichenfolge angegeben werden, mit der die Methode für die Erstellung gültiger Geometrien ausgewählt wird. Die Optionszeichenfolge hat das Format "method=linework|structure keepcollapsed=true|false". Wird kein "params"-Argument angegeben, wird standardmäßig der "linework"-Algorithmus verwendet.

Der Schlüssel "Methode" hat zwei Werte.

- "linework" is the default algorithm, and builds valid geometries by first extracting all lines, noding that linework together, then building a value output from the linework. The requirement that no vertices are lost can generate complex collections as outputs.
- "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 results tend to be more intuitive, where collapsed lines are discarded in the output.

The "keepcollapsed" key is only valid for the "structure" algorithm, and takes a value of "true" or "false".

- Keep collapsed of "false" is the default, and means that geometry components that collapse to a lower dimensionality, for example a one-point linestring will be dropped.
- Keep collapsed of "true" means geometry components that collapse to a lower dimensionality will be retained.

Wird vom GEOS Modul ausgeführt

Verfügbarkeit: 2.0.0

Verbessert: 2.0.1, Geschwindigkeitsverbesserungen

Verbessert: 2.1.0, Unterstützung für GEOMETRYCOLLECTION und MULTIPOINT hinzugefügt.

Verbessert: 3.1.0, Entfernen von Koordinaten mit NaN-Werten hinzugefügt.

Verbessert: 3.2.0, zusätzliche Algorithmus-Optionen, 'Linienwerk' und 'Struktur', die GEOS >= 3.10.0 erfordern.

 Diese Funktion unterstützt 3d und lässt den Z-Index nicht fallen.

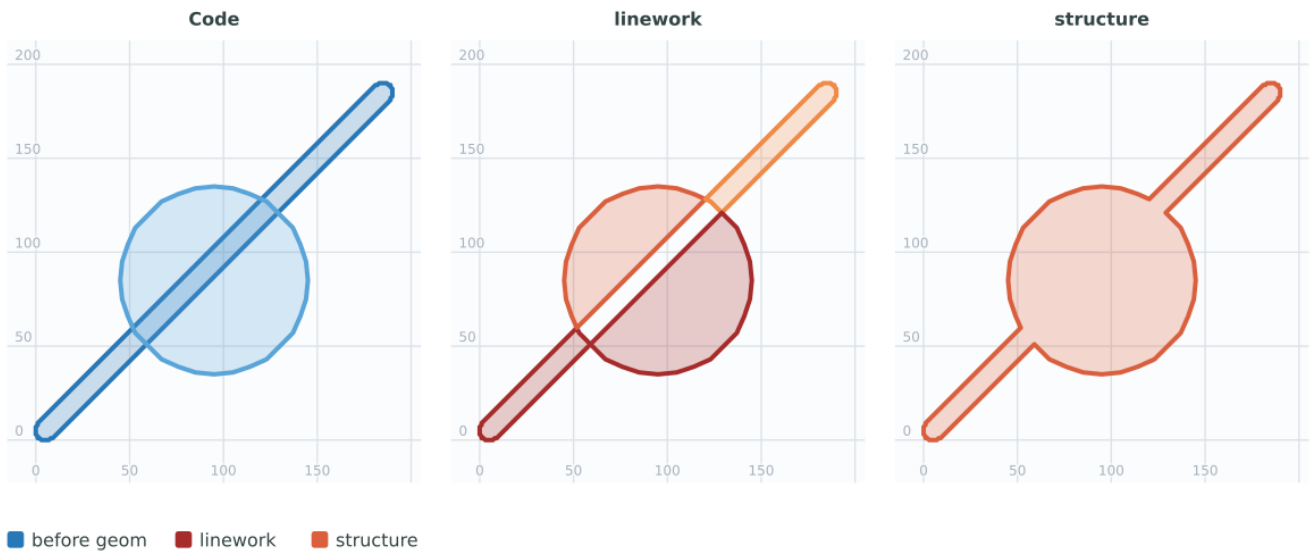
Beispiele

Make valid a MultiPolygon of two overlapping polygons using the linework and structure methods.

Code

```
SELECT ST_MakeValid(before_geom) AS linework,
  ST_MakeValid(before_geom, 'method=structure') AS structure
FROM (SELECT 'MULTIPOLYGON(((186 194,187 194,188 195,189 195,190 195,
191 195,192 195,193 194,194 194,194 193,195 192,195 191,
195 190,195 189,195 188,194 187,194 186,14 6,13 6,12 5,11 5,
10 5,9 5,8 5,7 6,6 6,6 7,5 8,5 9,5 10,5 11,5 12,6 13,6 14,186 194)),
((150 90,149 80,146 71,142 62,135 55,128 48,119 44,110 41,100 40,
90 41,81 44,72 48,65 55,58 62,54 71,51 80,50 90,51 100,
54 109,58 118,65 125,72 132,81 136,90 139,100 140,110 139,
119 136,128 132,135 125,142 118,146 109,149 100,150 90)))'::geometry AS before_geom) AS f;
```

Figure

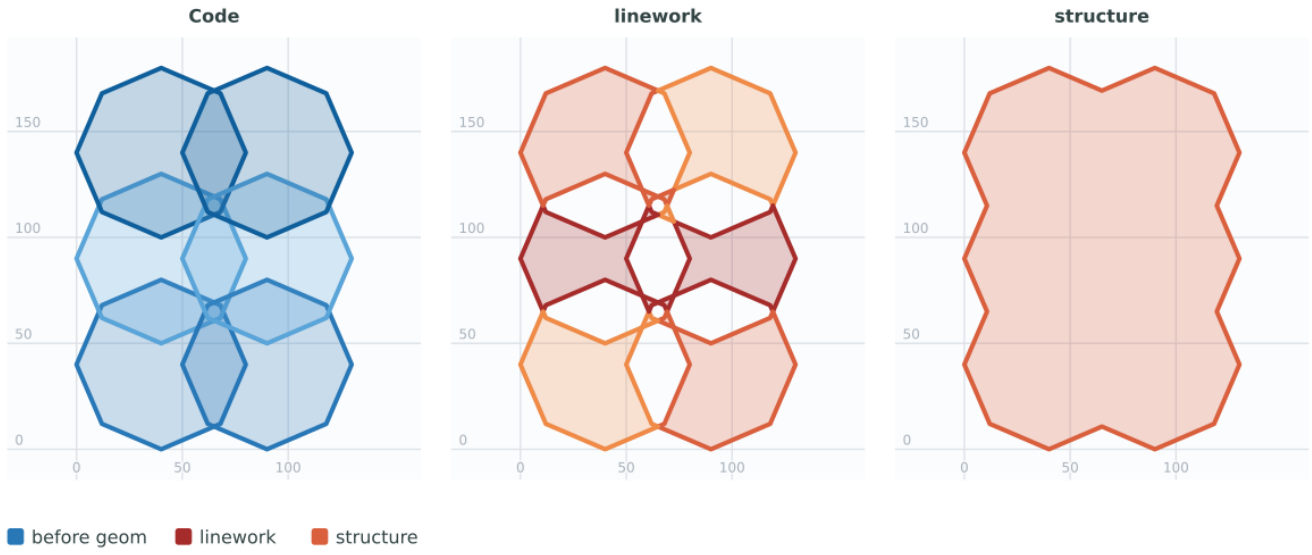


Make valid a MultiPolygon of six overlapping polygons using the linework and structure methods.

Code

```
SELECT ST_MakeValid(before_geom) AS linework,
  ST_MakeValid(before_geom, 'method=structure') AS structure
FROM (SELECT 'MULTIPOLYGON(((91 50,79 22,51 10,23 22,11 50,23 78,51 90,79 78,91 50)),
      ((91 100,79 72,51 60,23 72,11 100,23 128,51 140,79 128,91 100)),
      ((91 150,79 122,51 110,23 122,11 150,23 178,51 190,79 178,91 150)),
      ((141 50,129 22,101 10,73 22,61 50,73 78,101 90,129 78,141 50)),
      ((141 100,129 72,101 60,73 72,61 100,73 128,101 140,129 128,141 100)),
      ((141 150,129 122,101 110,73 122,61 150,73 178,101 190,129 178,141 150)))'::geometry ↩
   AS before_geom) AS c;
```

Figure



Collapsed Geometry Examples

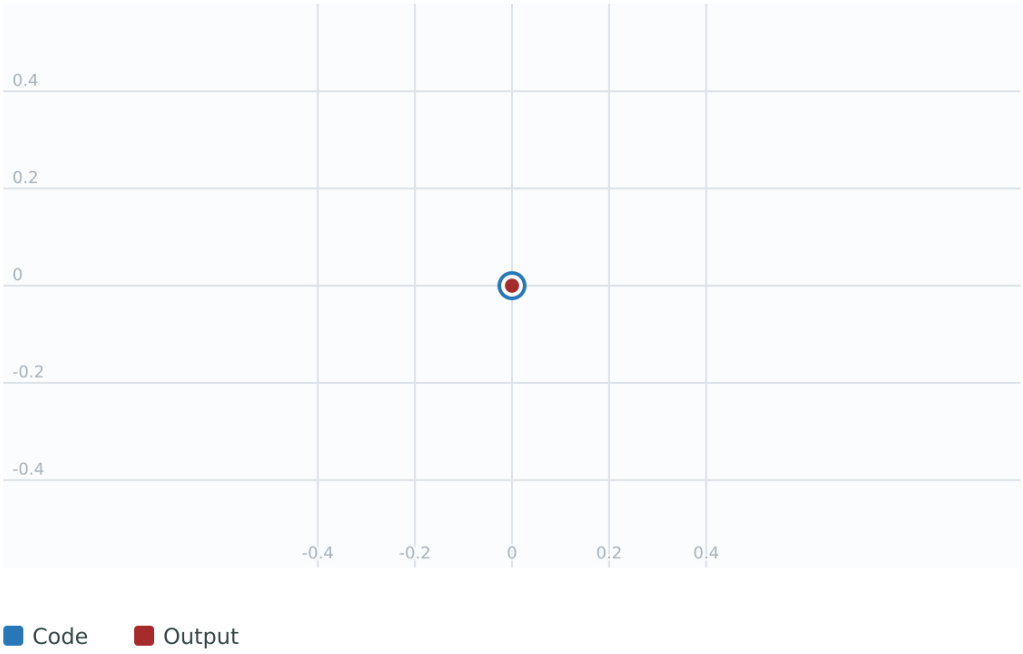
Code

```
SELECT ST_MakeValid(
'LINESTRING(0 0,0 0)',
'method=structure keepcollapsed=true'
);
```

Output

```
POINT(0 0)
```

Figure



Code

```
SELECT ST_MakeValid(
'LINESTRING(0 0,0 0)',
'method=structure keepcollapsed=false'
);
```

Output

```
ZEILENSTRING LEER
```

Siehe auch

ST_IsValid, ST_Collect, ST_CollectionExtract

7.7 Funktionen des räumlichen Bezugssystems

7.7.1 ST_InverseTransformPipeline

ST_InverseTransformPipeline — Rückgabe einer neuen Geometrie mit in ein anderes räumliches Bezugssystem transformierten Koordinaten unter Verwendung der Umkehrung einer definierten Koordinatentransformationspipeline.

Synopsis

geometry **ST_InverseTransformPipeline**(geometry geom, text pipeline, integer to_srid);

Beschreibung

Rückgabe einer neuen Geometrie mit in ein anderes räumliches Bezugssystem transformierten Koordinaten unter Verwendung einer definierten Koordinatentransformationspipeline, die in umgekehrter Richtung verläuft.

Einzelheiten zum Schreiben einer Transformationspipeline finden Sie unter ST_TransformPipeline.

Verfügbarkeit: 3.4.0

Die SRID der Eingabegeometrie wird ignoriert, und die SRID der Ausgabegeometrie wird auf Null gesetzt, sofern nicht über den optionalen Parameter `to_srid` ein Wert angegeben wird. Bei Verwendung von ST_TransformPipeline wird die Pipeline in Vorwärtsrichtung ausgeführt. Bei Verwendung von `ST_InverseTransformPipeline()` wird die Pipeline in umgekehrter Richtung ausgeführt.

Transformationen unter Verwendung von Pipelines sind eine spezialisierte Version von ST_Transform. In den meisten Fällen wählt "ST_Transform" die richtigen Operationen für die Konvertierung zwischen Koordinatensystemen und sollte daher bevorzugt werden.

Beispiele

Ändern Sie WGS 84 long lat in UTM 31N unter Verwendung der Konvertierung EPSG:16031

This example transforms in the inverse direction.

Code

```
SELECT ST_InverseTransformPipeline(
        'POINT(426857.9877165967 5427937.523342293)'::geometry,
        'urn:ogc:def:coordinateOperation:EPSG::16031'
    ) AS wgs_geom;
```

Output

```
POINT(2 48.99999999999999)
```

Figure



Beispiel GDA2020.

This example uses ST_Transform with automatic selection of a conversion pipeline.

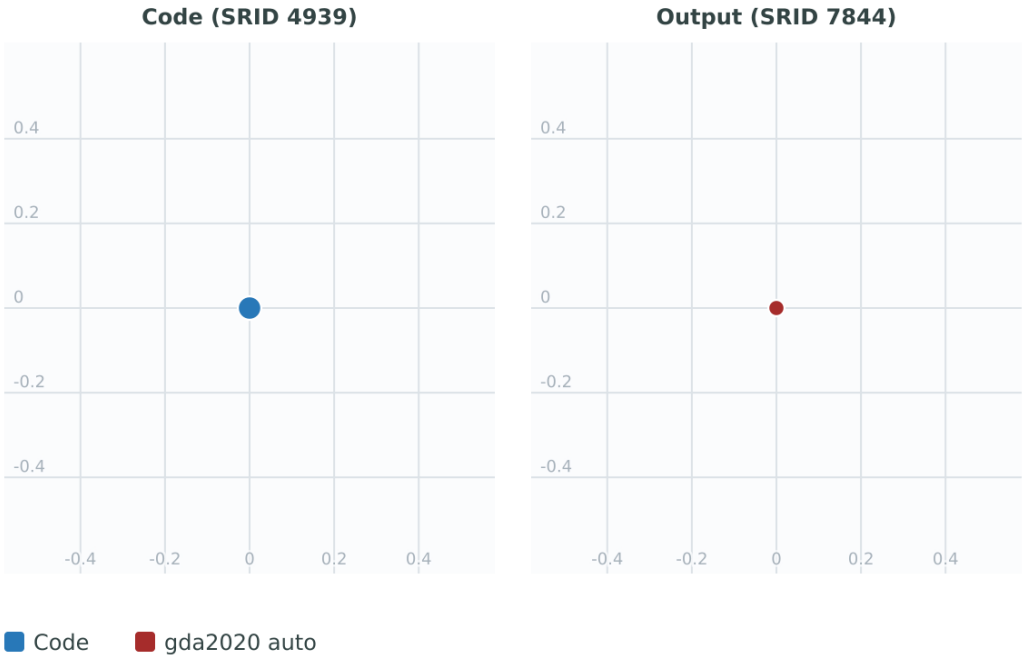
Code

```
SELECT ST_Transform('SRID=4939;POINT(143.0 -37.0)'::geometry, 7844) AS gda2020_auto;
```

Output

```
POINT(143.00000635638918 -36.999986706128176)
```

Figure



Siehe auch

ST_Transform, ST_TransformPipeline

7.7.2 ST_SetSRID

ST_SetSRID — Legen Sie den SRID für eine Geometrie fest.

Synopsis

geometry **ST_SetSRID**(geometry geom, integer srid);

Beschreibung

Setzt den SRID einer Geometrie auf einen bestimmten Integer-Wert. Nützlich bei der Erstellung von Begrenzungsrahmen für Abfragen.



Anmerkung
Diese Funktion transformiert die Geometriekoordinaten in keiner Weise - sie setzt lediglich die Metadaten, die das räumliche Bezugssystem definieren, in dem die Geometrie angenommen wird. Verwenden Sie ST_Transform, wenn Sie die Geometrie in eine neue Projektion transformieren möchten.

 Diese Methode implementiert die OGC Simple Features Implementation Specification for SQL 1.1.

 Diese Methode unterstützt kreisförmige Strings und Kurven.

Beispiele

Mark a point as WGS 84 longitude/latitude.

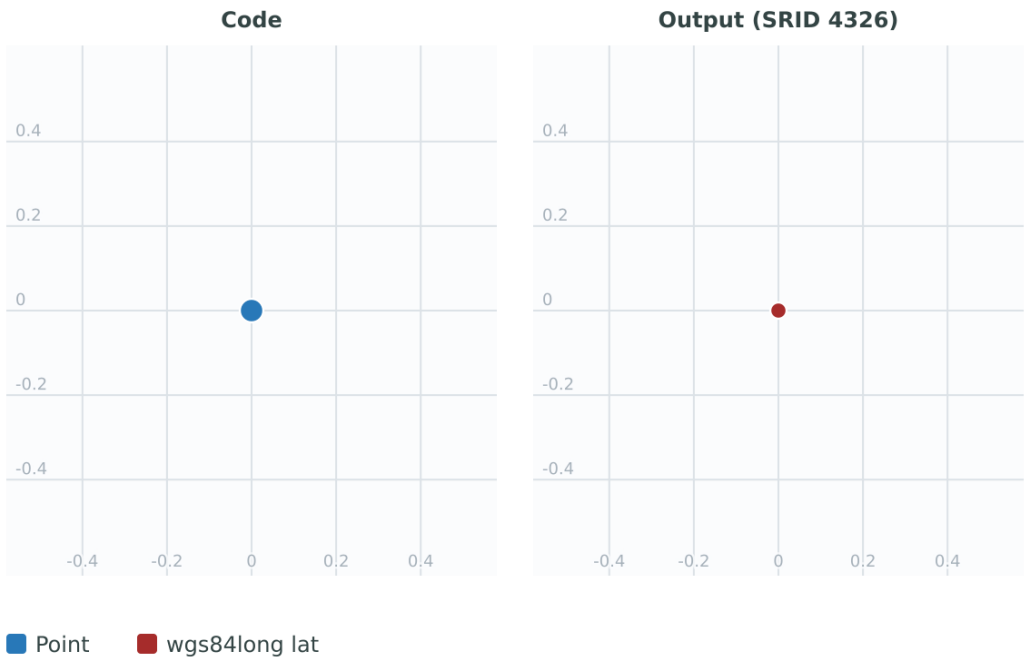
Code

```
SELECT ST_SetSRID(ST_Point(-123.365556, 48.428611), 4326) AS wgs84long_lat;
```

Output

```
SRID=4326;POINT(-123.365556 48.428611)
```

Figure



Mark a point as WGS 84 longitude/latitude and then transform it to Web Mercator.

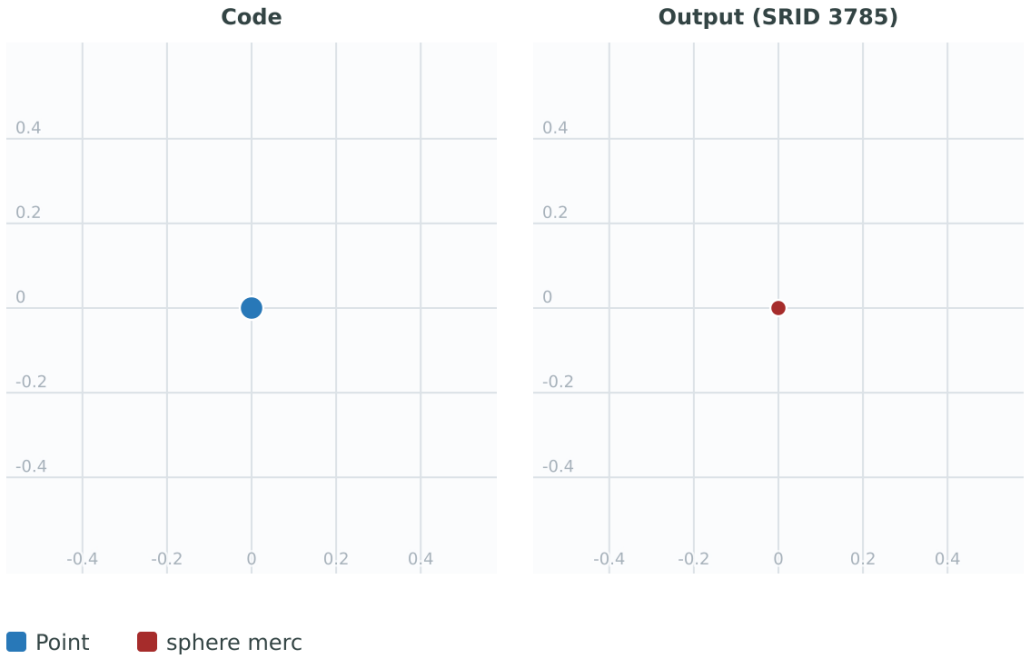
Code

```
SELECT ST_Transform(ST_SetSRID(ST_Point(-123.365556, 48.428611), 4326), 3785) AS sphere_merc ↩
    ;
```

Output

```
SRID=3785;POINT(-13732990.8753491 6178458.96425423)
```

Figure



Siehe auch

Abschnitt 4.5, ST_SRID, ST_Transform, UpdateGeometrySRID

7.7.3 ST_SRID

ST_SRID — Gibt die Raumbezugskennung für eine Geometrie zurück.

Synopsis

integer **ST_SRID**(geometry g1);

Beschreibung

Liefert die Kennung des Raumbezugs für die ST_Geometrie, wie in der Tabelle spatial_ref_sys definiert. Abschnitt 4.5



Anmerkung
Die Tabelle spatial_ref_sys ist eine Tabelle, die alle PostGIS bekannten räumlichen Bezugssysteme katalogisiert und für Transformationen von einem räumlichen Bezugssystem in ein anderes verwendet wird. Daher ist es wichtig, dass Sie die richtige Kennung für das räumliche Bezugssystem haben, wenn Sie Ihre Geometrien transformieren möchten.

 Diese Methode implementiert die OGC Simple Features Implementation Specification for SQL 1.1. s2.1.1.1

 Diese Methode setzt die SQL/MM-Spezifikation um. SQL-MM 3: 5.1.5

 Diese Methode unterstützt kreisförmige Strings und Kurven.

Beispiele

Code

```
SELECT ST_SRID(ST_GeomFromText('POINT(-71.1043 42.315)', 4326));
```

Output

```
st_srid
---------
    4326
(1 row)
```

Siehe auch

Abschnitt 4.5, ST_SetSRID, ST_Transform, ST_SRID, ST_SRID

7.7.4 ST_Transform

ST_Transform — Rückgabe einer neuen Geometrie mit in ein anderes räumliches Bezugssystem transformierten Koordinaten.

Synopsis

geometry **ST_Transform**(geometry g1, integer srid);
geometry **ST_Transform**(geometry geom, text to_proj);
geometry **ST_Transform**(geometry geom, text from_proj, text to_proj);
geometry **ST_Transform**(geometry geom, text from_proj, integer to_srid);

Beschreibung

Gibt eine neue Geometrie zurück, deren Koordinaten in ein anderes räumliches Bezugssystem transformiert wurden. Der Ziel-Raumbezug `to_srid` kann durch einen gültigen SRID-Integer-Parameter identifiziert werden (d. h. er muss in der Tabelle `spatial_ref_sys` vorhanden sein). Alternativ kann ein als PROJ.4-String definierter Raumbezug für `to_proj` und/oder `from_proj` verwendet werden, allerdings sind diese Methoden nicht optimiert. Wenn das Ziel-Raumbezugssystem durch einen PROJ.4-String anstelle eines SRID ausgedrückt wird, wird der SRID der Ausgangsgeometrie auf Null gesetzt. Mit Ausnahme der Funktionen mit `from_proj` müssen Eingabegeometrien einen definierten SRID haben.

ST_Transform wird oft mit ST_SetSRID verwechselt. ST_Transform ändert tatsächlich die Koordinaten einer Geometrie von einem räumlichen Bezugssystem in ein anderes, während ST_SetSRID() lediglich den SRID-Bezeichner der Geometrie ändert.

ST_Transform wählt automatisch eine geeignete Konvertierungspipeline für das Quell- und Ziel-Raumbezugssystem aus. Um eine bestimmte Konvertierungsmethode zu verwenden, verwenden Sie ST_TransformPipeline.



Anmerkung
Erfordert, dass PostGIS mit PROJ-Unterstützung kompiliert ist. Verwenden Sie PostGIS_Full_Version, um zu bestätigen, dass Sie PROJ-Unterstützung einkompiliert haben.



Anmerkung
Wenn mehr als eine Transformation verwendet wird, ist es sinnvoll, einen funktionalen Index für die am häufigsten verwendeten Transformationen zu haben, um die Vorteile der Indexnutzung zu nutzen.



Anmerkung
Vor 1.3.4 ist diese Funktion abgestürzt, wenn die Geometrien CURVES enthalten. Dies wurde mit 1.3.4+ behoben

---

Erweiterung: Mit 2.0.0 wurde die Unterstützung für polyedrische Oberflächen eingeführt.

Verbessert: In Version 2.3.0 wurde die Unterstützung für direkten PROJ.4 Text eingeführt.

 Diese Methode setzt die SQL/MM-Spezifikation um. SQL-MM 3: 5.1.6

 Diese Methode unterstützt kreisförmige Strings und Kurven.

 Diese Funktion unterstützt polyedrische Flächen.

Beispiele

Änderung der Geometrie der Ebene des Bundesstaates Massachusetts von US-Fuß auf WGS 84 long lat

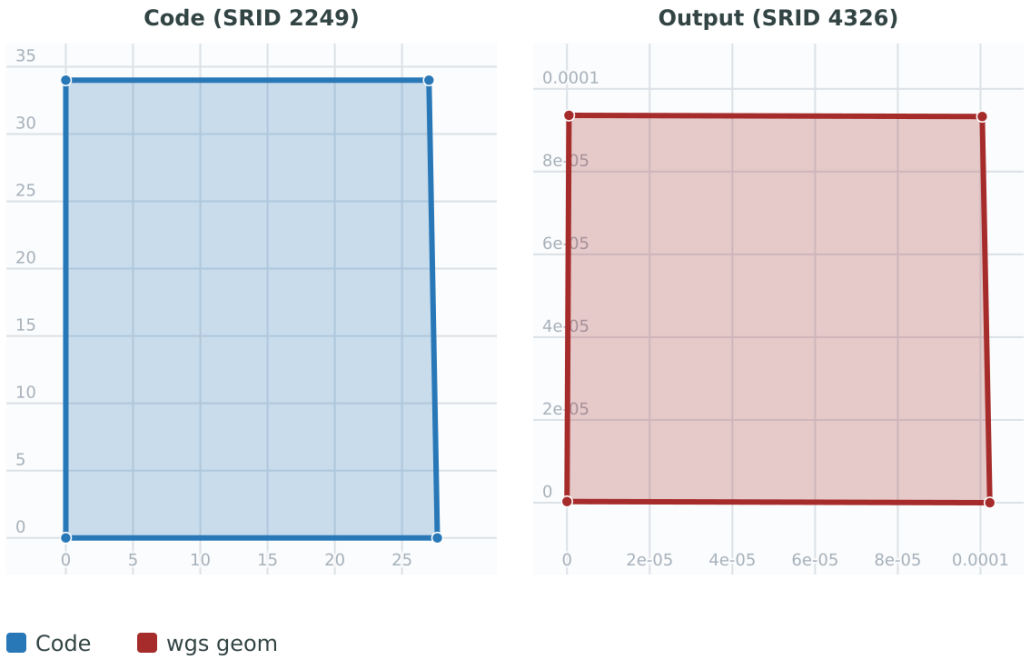
Code

```
SELECT ST_Transform(
  'SRID=2249;POLYGON((743238 2967416,743238 2967450,
    743265 2967450,743265.625 2967416,743238 2967416))'::geometry,
  4326) AS wgs_geom;
```

Output

```
SRID=4326;POLYGON((-71.1776848522251 42.3902896512903,-71.17768437663261 42.390382947801015,-71.17758443054647 ↩
    42.390382667791854,-71.17758259272304 42.390289364798825,-71.1776848522251 42.3902896512903))
```

Figure



3D Circular String example.

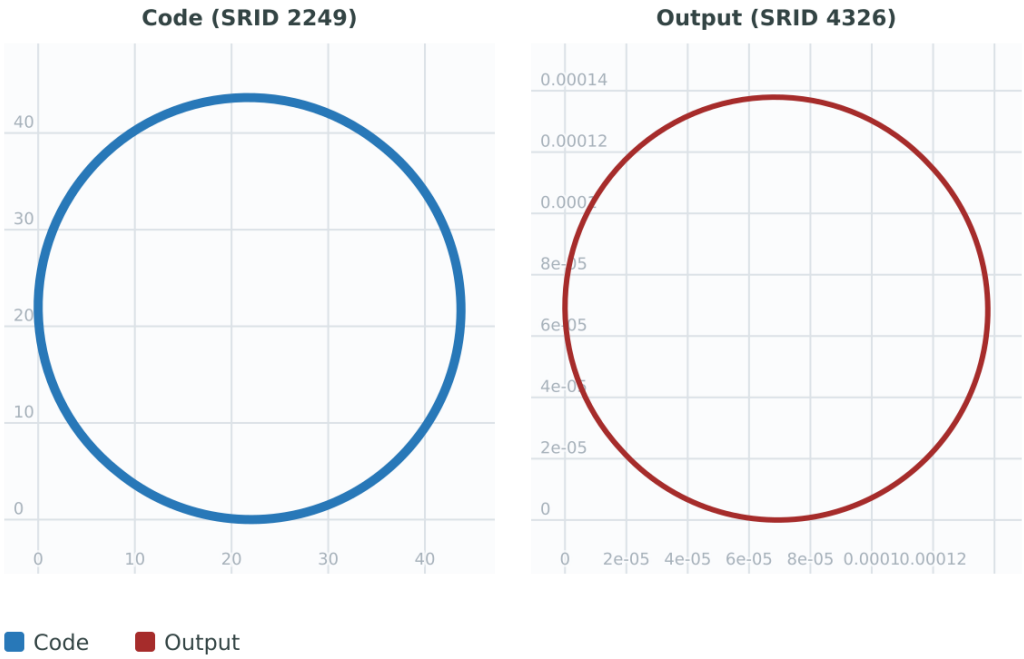
Code

```
SELECT ST_Transform('SRID=2249;CIRCULARSTRING(743238 2967416 1,743238 2967450 2,743265 ↩
    2967450 3,743265.625 2967416 3,743238 2967416 4)'::geometry,
    4326);
```

Output

```
SRID=4326;CIRCULARSTRING(-71.1776848522251 42.3902896512903 1,-71.17768437663261 42.390382947801015 2,
-71.17758443054647 42.390382667791854 3,
-71.17758259272304 42.390289364798825 3,-71.1776848522251 42.3902896512903 4)
```

Figure



Beispiel für die Erstellung eines partiellen funktionalen Indexes. Für Tabellen, bei denen Sie nicht sicher sind, dass alle Geometrien ausgefüllt werden, ist es am besten, einen partiellen Index zu verwenden, der Null-Geometrien auslässt, was sowohl Platz spart als auch Ihren Index kleiner und effizienter macht.

Code

```
CREATE INDEX idx_geom_26986_parcels
  ON parcels
  USING gist
  (ST_Transform(geom, 26986))
  WHERE geom IS NOT NULL;
```

Beispiele für die Verwendung von PROJ.4 Text zur Transformation mit benutzerdefinierten räumlichen Bezügen.

Find intersection of two polygons near the North pole, using a custom Gnomic projection. See http://boundlessgeo.com/2012/02/flattening the-peel/.

Code

```
WITH data AS (
  SELECT
    ST_GeomFromText('POLYGON((170 50,170 72,-130 72,-130 50,170 50))', 4326) AS p1,
    ST_GeomFromText('POLYGON((-170 68,-170 90,-141 90,-141 68,-170 68))', 4326) AS p2,
    '+proj=gnom +ellps=WGS84 +lat_0=70 +lon_0=-160 +no_defs'::text AS gnom
)
SELECT ST_Transform(ST_Intersection(ST_Transform(p1, gnom), ST_Transform(p2, gnom)),
  gnom, 4326)
FROM data;
```

Output

```
POLYGON((-140.9999840290591 73.42768865981984,-141 68,-170 68,-170.0000026915361 74.05492179637055,-140.9999840290591 ↩
    73.42768865981984))
```

Figure



Konfigurieren des Transformationsverhaltens

Manchmal kann eine Koordinatentransformation, die eine Rasterverschiebung beinhaltet, fehlschlagen, z.B. wenn PROJ.4 nicht mit Rasterverschiebungsdateien erstellt wurde oder die Koordinate nicht innerhalb des Bereichs liegt, für den die Rasterverschiebung definiert ist. Standardmäßig gibt PostGIS einen Fehler aus, wenn keine Rasterverschiebungsdatei vorhanden ist, aber dieses Verhalten kann auf einer pro-SRID-Basis konfiguriert werden, indem entweder verschiedene `to_proj` Werte des PROJ.4-Textes getestet oder der `proj4text` Wert in der `spatial_ref_sys` Tabelle geändert wird.

Der proj4text-Parameter +datum=NAD87 ist zum Beispiel eine Kurzform für den folgenden +nadgrids-Parameter:

Code

```
+nadgrids=@conus, @alaska, @ntv2_0.gsb, @ntv1_can.dat
```

Das Präfix @ bedeutet, dass kein Fehler gemeldet wird, wenn die Dateien nicht vorhanden sind, aber wenn das Ende der Liste erreicht wird, ohne dass eine Datei gefunden wurde (d. h. gefunden und überschneidend), wird ein Fehler ausgegeben.

Wenn Sie dagegen sicherstellen wollten, dass zumindest die Standarddateien vorhanden sind, aber wenn alle Dateien ohne Treffer durchsucht wurden, eine Nulltransformation angewendet wird, könnten Sie dies verwenden:

Code

```
+nadgrids=@conus, @alaska, @ntv2_0.gsb, @ntv1_can.dat, null
```

Die Null-Rasterverschiebungsdatei ist eine gültige Rasterverschiebungsdatei, die die ganze Welt abdeckt und keine Verschiebung anwendet. Wenn Sie also PostGIS so ändern wollten, dass bei Transformationen zu SRID 4267, die nicht im korrekten Bereich liegen, kein ERROR ausgelöst wird, würden Sie das folgende Beispiel verwenden:

Code

```
UPDATE spatial_ref_sys SET proj4text = '+proj=longlat +ellps=clrk66 +nadgrids=@conus,@alaska ↩
    ,@ntv2_0.gsb,@ntv1_can.dat,null +no_defs' WHERE srid = 4267;
```

Siehe auch

Abschnitt 4.5, ST_SetSRID, ST_SRID, UpdateGeometrySRID, ST_TransformPipeline

7.7.5 ST_TransformPipeline

ST_TransformPipeline — Rückgabe einer neuen Geometrie mit in ein anderes räumliches Bezugssystem transformierten Koordinaten unter Verwendung einer definierten Koordinatentransformationspipeline.

Synopsis

geometry **ST_TransformPipeline**(geometry g1, text pipeline, integer to_srid);

Beschreibung

Rückgabe einer neuen Geometrie mit in ein anderes räumliches Bezugssystem transformierten Koordinaten unter Verwendung einer definierten Koordinatentransformationspipeline.

Transformations-Pipelines werden mit einem der folgenden String-Formate definiert:

- `urn:ogc:def:coordinateOperation:AUTHORITY::CODE`. Beachten Sie, dass ein einfacher `EPSG:CODE` String eine Koordinatenoperation nicht eindeutig identifiziert: Der gleiche EPSG-Code kann für eine CRS-Definition verwendet werden.
- Eine PROJ-Pipeline-Zeichenkette in der Form: `+proj=pipeline ...`. Die automatische Achsennormalisierung wird nicht angewendet, und der Aufrufer muss gegebenenfalls einen zusätzlichen Pipelineschritt hinzufügen oder `axisswap` Schritte entfernen.
- Verkettete Operationen der Form: `urn:ogc:def:coordinateOperation,coordinateOperation:EPSG::3895,coo`

Verfügbarkeit: 3.4.0

Die SRID der Eingangsgeometrie wird ignoriert, und die SRID der Ausgangsgeometrie wird auf Null gesetzt, sofern nicht über den optionalen Parameter `to_srid` ein Wert angegeben wird. Bei Verwendung von `ST_TransformPipeline()` wird die Pipeline in Vorwärtsrichtung ausgeführt. Bei Verwendung von ST_InverseTransformPipeline wird die Pipeline in der umgekehrten Richtung ausgeführt.

Transformationen unter Verwendung von Pipelines sind eine spezialisierte Version von ST_Transform. In den meisten Fällen wählt "ST_Transform" die richtigen Operationen für die Konvertierung zwischen Koordinatensystemen und sollte daher bevorzugt werden.

Beispiele

Ändern Sie WGS 84 long lat in UTM 31N unter Verwendung der Konvertierung EPSG:16031

This example transforms in the forward direction.

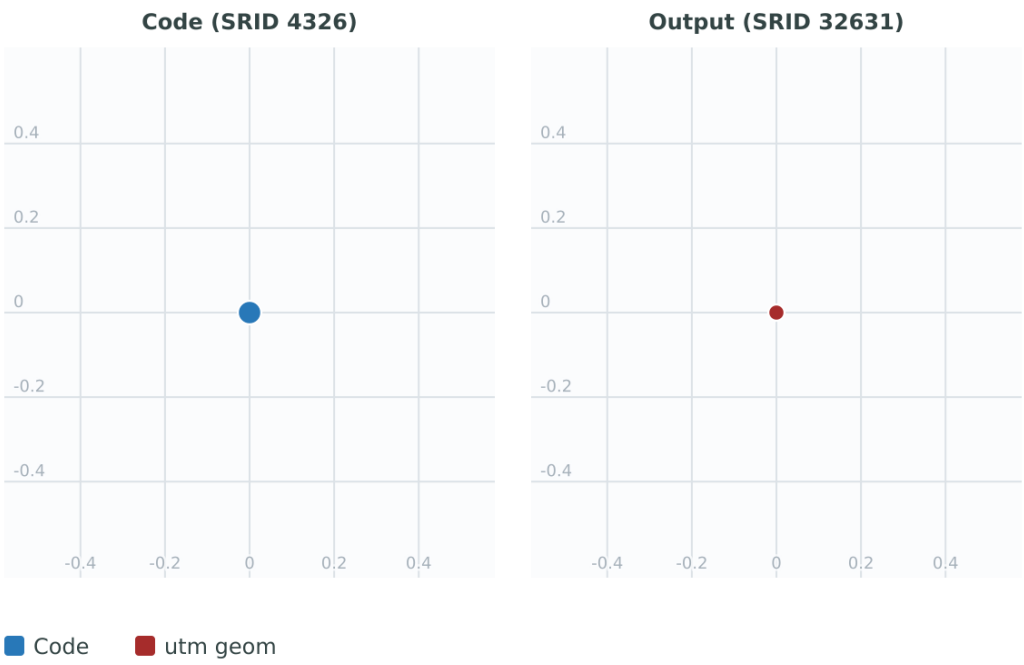
Code

```
SELECT
    ST_SetSRID(ST_TransformPipeline(
        'SRID=4326;POINT(2 49)'::geometry,
        'urn:ogc:def:coordinateOperation:EPSG::16031'
    ), 32631) AS utm_geom;
```

Output

```
SRID=32631;POINT(426857.9877165967 5427937.523342293)
```

Figure



This example transforms in the inverse direction.

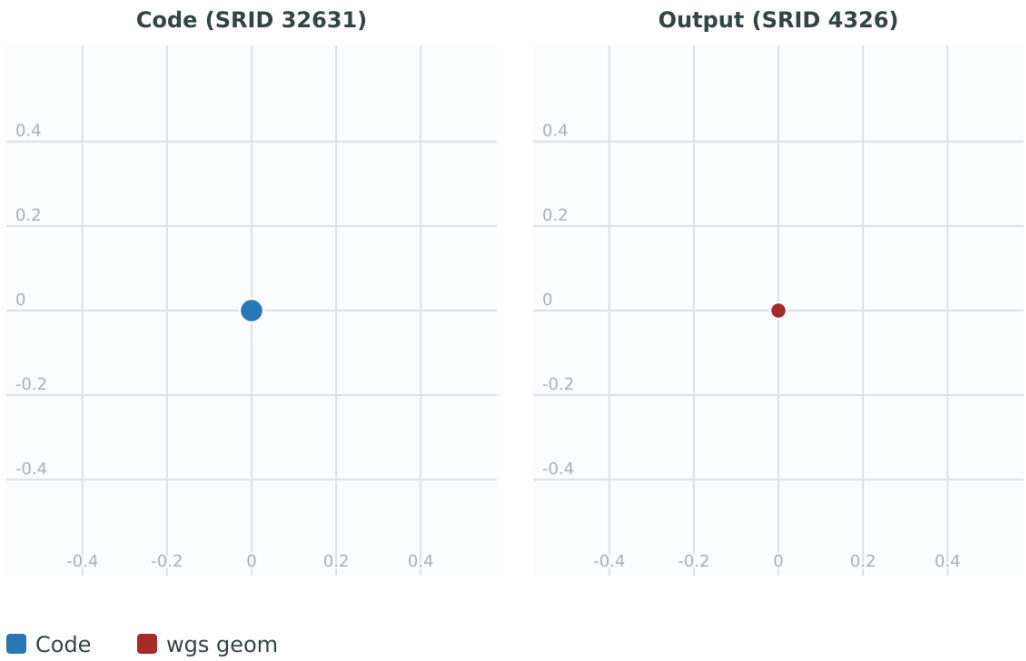
Code

```
SELECT
    ST_SetSRID(ST_InverseTransformPipeline(
        'SRID=32631;POINT(426857.9877165967 5427937.523342293)'::geometry,
        'urn:ogc:def:coordinateOperation:EPSG::16031'
    ), 4326) AS wgs_geom;
```

Output

```
SRID=4326;POINT(2 48.99999999999999)
```

Figure



Beispiel GDA2020.

This example uses ST_Transform with automatic selection of a conversion pipeline.

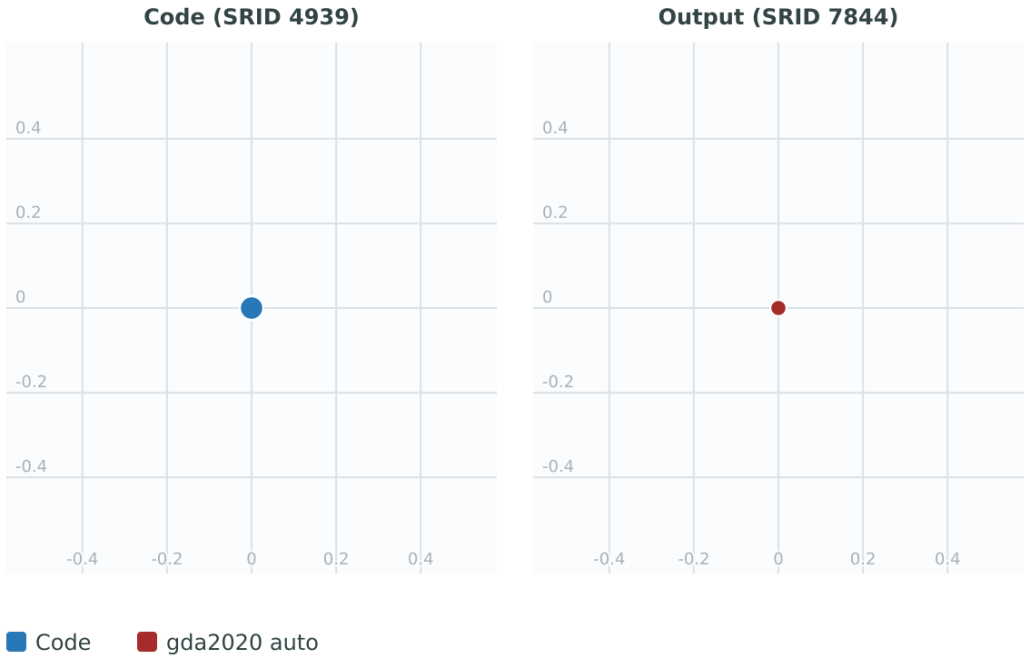
Code

```
SELECT ST_Transform('SRID=4939;POINT(143.0 -37.0)'::geometry, 7844) AS gda2020_auto;
```

Output

```
POINT(143.00000635638918 -36.999986706128176)
```

Figure



Using a defined conversion (EPSG:8447)

Code

```
SELECT
    ST_TransformPipeline(
        'SRID=4939;POINT(143.0 -37.0)'::geometry,
        'urn:ogc:def:coordinateOperation:EPSG::8447'
    ) AS gda2020_code;
```

Output

```
POINT(143.0000063280214 -36.999986718287545)
```

This example uses a PROJ pipeline definition matching EPSG:8447, as returned from projinfo. Any axisswap steps must be removed.

Code

```
SELECT
    ST_TransformPipeline(
        'SRID=4939;POINT(143.0 -37.0)'::geometry,
        '+proj=pipeline
   +step +proj=unitconvert +xy_in=deg +xy_out=rad
   +step +proj=hgridshift +grids=au_icsm_GDA94_GDA2020_conformal_and_distortion.tif
   +step +proj=unitconvert +xy_in=rad +xy_out=deg'
    ) AS gda2020_pipeline;
```

Output

```
POINT(143.0000063280214 -36.999986718287545)
```

Siehe auch

ST_Transform, ST_InverseTransformPipeline

7.7.6 postgis_srs_codes

postgis_srs_codes — Gibt die Liste der SRS-Codes zurück, die mit der angegebenen Behörde verbunden sind.

Synopsis

setof text **postgis_srs_codes**(text auth_name);

Beschreibung

Gibt eine Menge aller `auth_srid` für den angegebenen `auth_name` zurück.

Verfügbarkeit: 3.4.0

Proj Version 6+

Beispiele

Geben Sie die ersten zehn Codes für die EPSG-Behörde an.

Code

```
SELECT * FROM postgis_srs_codes('EPSG') ORDER BY 1 LIMIT 10;
```

Output

```
10156
10157
10158
10160
10162
10163
10164
10165
10166
10167
```

Siehe auch

postgis_srs, postgis_srs_all, postgis_srs_search

7.7.7 postgis_srs

postgis_srs — Rückgabe eines Metadatensatzes für die angefragte Behörde und srid.

Synopsis

table(auth_name text, auth_srid text, srname text, srtext text, proj4text text, point_sw geometry, point_ne geometry) **postgis_srs**(text auth_name, text auth_srid);

Beschreibung

Gibt einen Metadatensatz für den angeforderten `auth_srid` für den angegebenen `auth_name` zurück. Der Datensatz enthält den `auth_name`, `auth_srid`, `srname`, `srtext`, `proj4text` und die Ecken des Nutzungsbereichs, `point_sw` und `point_ne`.

Verfügbarkeit: 3.4.0

Proj Version 6+

Beispiele

Abrufen der Metadaten für EPSG:3005.

Code

```
SELECT auth_name, auth_srid, srname,point_sw AS point_sw,point_ne AS point_ne
FROM postgis_srs('EPSG', '3005');
```

Output

```
-[ RECORD 1 ]-------------------
auth_name | EPSG
auth_srid | 3005
srname    | NAD83 / BC Albers
point_sw  | POINT(-139.04 48.25)
point_ne  | POINT(-114.08 60.01)
```

Siehe auch

postgis_srs_codes, postgis_srs_all, postgis_srs_search

7.7.8 postgis_srs_all

postgis_srs_all — Gibt Metadatensätze für jedes räumliche Bezugssystem in der zugrunde liegenden Proj-Datenbank zurück.

Synopsis

table(auth_name text, auth_srid text, srname text, srtext text, proj4text text, point_sw geometry, point_ne geometry) **postgis_srs_all**(void);

Beschreibung

Gibt eine Menge aller Metadatensätze in der zugrunde liegenden Proj-Datenbank zurück. Die Datensätze haben die Bezeichnungen `auth_name`, `auth_srid`, `srname`, `srtext`, `proj4text` und die Ecken des Verwendungsbereichs, `point_sw` und `point_ne`.

Verfügbarkeit: 3.4.0

Proj Version 6+

Beispiele

Abruf der ersten 10 Metadatensätze aus der Proj-Datenbank.

Code

```
SELECT auth_name, auth_srid, srname FROM postgis_srs_all() ORDER BY auth_name, auth_srid ↩
    LIMIT 10;
```

Output

```
auth_name | auth_srid |                                srname
-----------+-----------+-------------------------------------------------------------------------
 EPSG      | 10156     | ETRS89 + MSL UK & Ireland VORF08 depth
 EPSG      | 10157     | ETRS89 + CD UK & Ireland VORF08 depth
 EPSG      | 10158     | S34J-IRF
 EPSG      | 10160     | S34J reconstruction east-orientated
 EPSG      | 10162     | JGD2011 / Japan Plane Rectangular CS I + JGD2011 (vertical) height
 EPSG      | 10163     | JGD2011 / Japan Plane Rectangular CS II + JGD2011 (vertical) height
 EPSG      | 10164     | JGD2011 / Japan Plane Rectangular CS III + JGD2011 (vertical) height
 EPSG      | 10165     | JGD2011 / Japan Plane Rectangular CS IV + JGD2011 (vertical) height
 EPSG      | 10166     | JGD2011 / Japan Plane Rectangular CS V + JGD2011 (vertical) height
 EPSG      | 10167     | JGD2011 / Japan Plane Rectangular CS VI + JGD2011 (vertical) height
```

Siehe auch

postgis_srs_codes, postgis_srs, postgis_srs_search

7.7.9 postgis_srs_search

postgis_srs_search — Gibt Metadatensätze für projizierte Koordinatensysteme zurück, die Nutzungsbereiche haben, die den Parameter bounds vollständig enthalten.

Synopsis

table(auth_name text, auth_srid text, srname text, srtext text, proj4text text, point_sw geometry, point_ne geometry) **postgis_srs_search**(geometry bounds, text auth_name=EPSG);

Beschreibung

Gibt eine Reihe von Metadatensätzen für projizierte Koordinatensysteme zurück, die Nutzungsbereiche haben, die den Parameter bounds vollständig enthalten. Jeder Datensatz enthält `auth_name`, `auth_srid`, `srname`, `srtext`, `proj4text`, und die Ecken des Nutzungsbereichs, `point_sw` und `point_ne`.

Die Suche sucht nur nach projizierten Koordinatensystemen und ist dafür gedacht, dass die Nutzer die möglichen Systeme erkunden, die für den Umfang ihrer Daten geeignet sind.

Verfügbarkeit: 3.4.0

Proj Version 6+

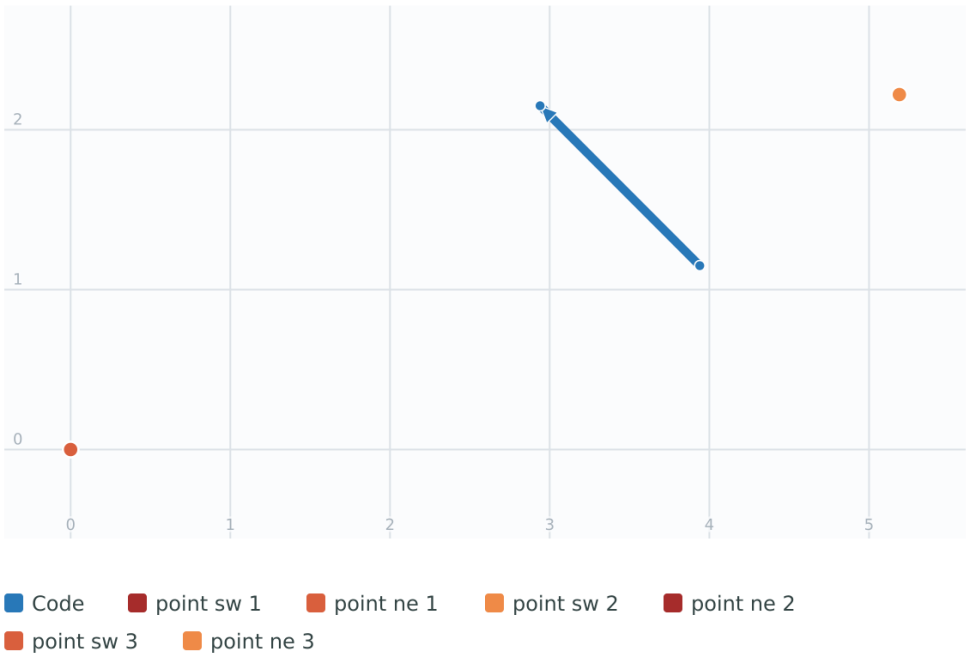
Beispiele

Suche nach projizierten Koordinatensystemen in Louisiana.

Code

```
SELECT auth_name, auth_srid, srname,point_sw AS point_sw,point_ne AS point_ne
FROM postgis_srs_search('SRID=4326;LINESTRING(-90 30,-91 31)')
WHERE auth_srid IN ('2801', '3452', '3457')
ORDER BY auth_name, auth_srid::integer
```

Figure



Durchsuchen Sie eine Tabelle nach maximaler Ausdehnung und finden Sie passende projizierte Koordinatensysteme.

Code

```
WITH ext AS (
  SELECT ST_Extent(geom) AS geom, Max(ST_SRID(geom)) AS srid
  FROM foo
)
SELECT auth_name, auth_srid, srname,
  point_sw AS point_sw,
  point_ne AS point_ne
FROM ext
CROSS JOIN postgis_srs_search(ST_SetSRID(ext.geom, ext.srid))
LIMIT 3;
```

Siehe auch

postgis_srs_codes, postgis_srs_all, postgis_srs

7.8 Geometrische Eingabe

7.8.1 Well-known-Text (WKT) Repräsentation

7.8.1.1 ST_BdPolyFromText

ST_BdPolyFromText — Konstruiert ein Polygon aus einer beliebigen Ansammlung von geschlossenen Linienzügen, welche als MultiLineString in der Well-Known Text Darstellung vorliegen müssen.

Synopsis

geometry **ST_BdPolyFromText**(text WKT, integer srid);

Beschreibung

Konstruiert ein Polygon aus einer beliebigen Ansammlung von geschlossenen Linienzügen, welche als MultiLineString in der Well-Known Text Darstellung vorliegen müssen.



Anmerkung
Meldet einen Fehler, wenn es sich bei dem WKT nicht um einen MULTILINESTRING handelt. Meldet einen Fehler, wenn die Ausgabe ein MULTIPOLYGON ist; in diesem Fall verwenden Sie bitte ST_BdMPolyFromText, oder vergleichen ST_BuildArea() für einen PostGIS orientierten Ansatz.



Diese Methode implementiert die OGC Simple Features Implementation Specification for SQL 1.1. s3.2.6.2

Wird vom GEOS Modul ausgeführt

Verfügbarkeit: 1.1.0

Siehe auch

ST_BuildArea, ST_BdMPolyFromText

7.8.1.2 ST_BdMPolyFromText

ST_BdMPolyFromText — Konstruiert ein MultiPolygon aus einer beliebigen Ansammlung von geschlossenen Linienzügen, welche als MultiLineString in der Well-Known Text Darstellung vorliegen müssen.

Synopsis

geometry **ST_BdMPolyFromText**(text WKT, integer srid);

Beschreibung

Konstruiert ein Polygon aus einer beliebigen Ansammlung von geschlossenen Linienzügen, Polygonen und MultiLineStrings, welche in der Well-Known Text Darstellung vorliegen müssen.



Anmerkung
Meldet einen Fehler wenn der WKT kein MULTILINESTRING ist. Erzwingt die MULTIPOLYGON Ausgabe sogar dann, wenn das Ergebnis nur aus einem einzelnen POLYGON besteht; verwenden Sie bitte ST_BdPolyFromText , wenn Sie sicher sind, daß nur ein einzelnes POLYGON entsteht, oder vegleichen Sie ST_BuildArea() für einen PostGIS orientierten Ansatz.

 Diese Methode implementiert die OGC Simple Features Implementation Specification for SQL 1.1. s3.2.6.2

Wird vom GEOS Modul ausgeführt

Verfügbarkeit: 1.1.0

Siehe auch

ST_BuildArea, ST_BdPolyFromText

7.8.1.3 ST_GeogFromText

ST_GeogFromText — Gibt einen geographischen Datentyp aus einer Well-known-Text (WKT), oder einer erweiterten WKT (EWKT), Darstellung zurück.

Synopsis

geography **ST_GeogFromText**(text EWKT);

Beschreibung

Gibt ein geographisches Objekt in der Well-known-Text oder in der erweiterten Well-known-Text Darstellung zurück. Falls nicht angegeben, wird die SRID 4326 angenommen. Dies ist ein Alias für ST_GeographyFromText. Punkte werden immer in Form von Länge und Breite ausgedrückt.

Beispiele

Add a WGS 84 geography point column to a table.

Code

```
ALTER TABLE sometable ADD COLUMN geog geography(POINT, 4326);
```

Populate the geography column from longitude and latitude columns.

Code

```
UPDATE sometable
SET geog = ST_GeogFromText('SRID=4326;POINT(' || lon || ' ' || lat || ')');
```

Specify a geography point using EPSG:4267, NAD27.

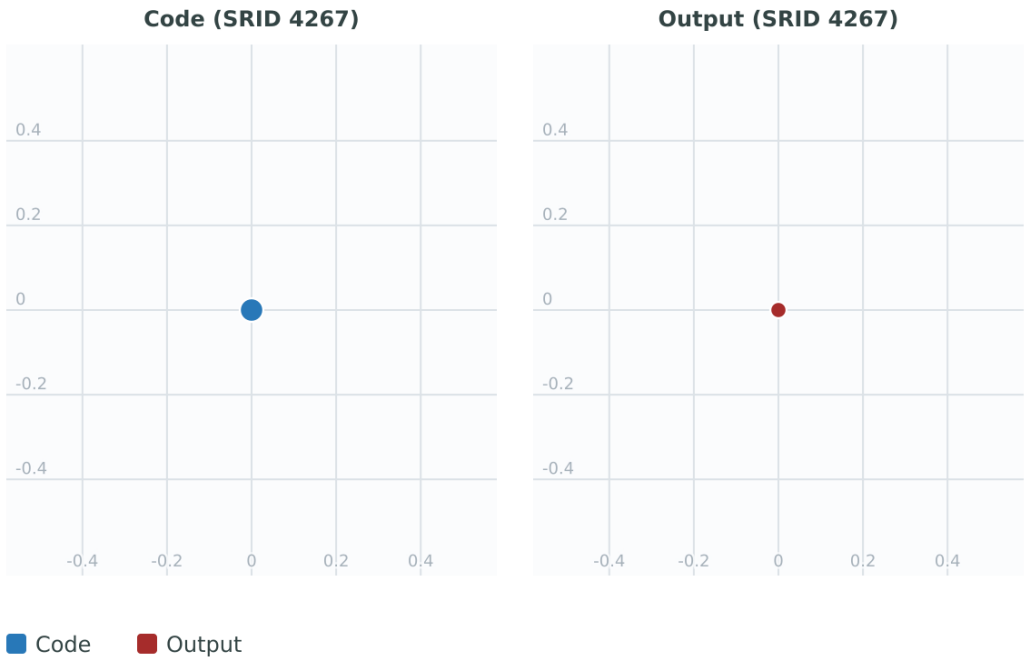
Code

```
SELECT ST_GeogFromText('SRID=4267;POINT(-77.0092 38.889588)');
```

Output

```
SRID=4267;POINT(-77.0092 38.889588)
```

Figure



Siehe auch

ST_AsText, ST_GeographyFromText

7.8.1.4 ST_GeographyFromText

ST_GeographyFromText — Gibt einen geographischen Datentyp aus einer Well-known-Text (WKT), oder einer erweiterten WKT (EWKT), Darstellung zurück.

Synopsis

geography **ST_GeographyFromText**(text EWKT);

Beschreibung

Gibt ein geographisches Objekt in der Well-known-Text Darstellung zurück. Falls nicht angegeben, wird die SRID 4326 angenommen.

Siehe auch

ST_GeogFromText, ST_AsText

7.8.1.5 ST_GeomCollFromText

ST_GeomCollFromText — Erzeugt eine Sammelgeometrie mit der gegebenen SRID aus einer WKT-Kollektion. Wenn keine SRID angegeben ist, wird diese standardmäßig auf 0 gesetzt.

Synopsis

geometry **ST_GeomCollFromText**(text WKT, integer srid);
geometry **ST_GeomCollFromText**(text WKT);

Beschreibung

Erzeugt eine Sammelgeometrie mit der gegebenen SRID aus einer Well-known-Text (WKT) Darstellung. Wenn keine SRID angegeben ist, wird diese standardmäßig auf 0 gesetzt.

OGC SPEC 3.2.6.2 - Die Option SRID stammt aus dem Konformitätstest

Gibt NULL zurück, wenn der WKT keine GEOMETRYCOLLECTION ist



Anmerkung
Verwenden Sie diese Funktion nicht, wenn Sie sich vollkommen sicher sind, dass ihre WKT Geometrie eine Sammelgeometrie ist. Sie ist langsamer als ST_GeomFromText, da sie einen zusätzlicher Validierungsschritt ausführt.


Diese Methode implementiert die OGC Simple Features Implementation Specification for SQL 1.1. s3.2.6.2


Diese Methode setzt die SQL/MM-Spezifikation um.

Beispiele

Code

```
SELECT ST_GeomCollFromText('GEOMETRYCOLLECTION(POINT(1 2), LINESTRING(1 2,3 4))');
```

Siehe auch

ST_GeomFromText, ST_SRID

7.8.1.6 ST_GeomFromEWKT

ST_GeomFromEWKT — Gibt einen spezifizierten ST_Geometry-Wert von einer erweiterten Well-known-Text Darstellung (EWKT) zurück.

Synopsis

geometry **ST_GeomFromEWKT**(text EWKT);

Beschreibung

Erzeugt ein PostGIS ST_Geometry Objekt aus der erweiterten OGC Well-known-Text (EWKT) Darstellung.



Anmerkung
EWKT ist kein Format des OGC Standards, sondern ein PostGIS eigenes Format, welches den Identifikator (SRID) des räumlichen Koordinatenreferenzsystem mit einbindet

Erweiterung: Mit 2.0.0 wurde die Unterstützung für polyedrische Oberflächen und TIN eingeführt.

 Diese Funktion unterstützt 3d und lässt den Z-Index nicht fallen.

 Diese Methode unterstützt kreisförmige Strings und Kurven.

 Diese Funktion unterstützt polyedrische Flächen.

 Diese Funktion unterstützt Dreiecke und dreieckige unregelmäßige Netzoberflächen (TIN).

Beispiele

Create a LineString with SRID 4269.

Code

```
SELECT ST_GeometryType(geom) AS geometry_type, ST_SRID(geom) AS srid
FROM ST_GeomFromEWKT('SRID=4269;LINESTRING(-71.160281 42.258729,-71.160837 ↩
    42.259113,-71.161144 42.25932)') AS geom;
```

Output

```
geometry_type | srid
---------------+------
 ST_LineString | 4269
(1 row)
```

Create a MultiLineString with SRID 4269.

Code

```
SELECT ST_GeometryType(geom) AS geometry_type, ST_SRID(geom) AS srid
FROM ST_GeomFromEWKT('SRID=4269;MULTILINESTRING((-71.160281 42.258729,-71.160837 ↩
    42.259113,-71.161144 42.25932))') AS geom;
```

Output

```
geometry_type    | srid
---------------------+------
 ST_MultiLineString  | 4269
(1 row)
```

Create a Point with SRID 4269.

Code

```
SELECT ST_GeometryType(geom) AS geometry_type, ST_SRID(geom) AS srid
FROM ST_GeomFromEWKT('SRID=4269;POINT(-71.064544 42.28787)') AS geom;
```

Output

```
geometry_type | srid
--------------+------
 ST_Point     | 4269
(1 row)
```

Create a Polygon with SRID 4269.

Code

```
SELECT ST_GeometryType(geom) AS geometry_type, ST_SRID(geom) AS srid
FROM ST_GeomFromEWKT('SRID=4269;POLYGON((-71.1776585052917 ↩
    42.3902909739571,-71.1776820268866 42.3903701743239,
-71.1776063012595 42.3903825660754,-71.1775826583081 42.3903033653531,-71.1776585052917 ↩
    42.3902909739571))') AS geom;
```

Output

```
geometry_type | srid
--------------+------
 ST_Polygon   | 4269
(1 row)
```

Create a MultiPolygon with SRID 4269.

Code

```
SELECT ST_GeometryType(geom) AS geometry_type, ST_SRID(geom) AS srid
FROM ST_GeomFromEWKT('SRID=4269;MULTIPOLYGON(((-71.1031880899493 42.3152774590236,
-71.1031627617667 42.3152960829043,-71.102923838298 42.3149156848307,
-71.1023097974109 42.3151969047397,-71.1019285062273 42.3147384934248,
-71.102505233663 42.3144722937587,-71.10277487471 42.3141658254797,
-71.103113945163 42.3142739188902,-71.10324876416 42.31402489987,
-71.1033002961013 42.3140393340215,-71.1033488797549 42.3139495090772,
-71.103396240451 42.3138632439557,-71.1041521907712 42.3141153348029,
-71.1041411411543 42.3141545014533,-71.1041287795912 42.3142114839058,
-71.1041188134329 42.3142693656241,-71.1041112482575 42.3143272556118,
-71.1041072845732 42.3143851580048,-71.1041057218871 42.3144430686681,
-71.1041065602059 42.3145009876017,-71.1041097995362 42.3145589148055,
-71.1041166403905 42.3146168544148,-71.1041258822717 42.3146748022936,
-71.1041375307579 42.3147318674446,-71.1041492906949 42.3147711126569,
-71.1041598612795 42.314808571739,-71.1042515013869 42.3151287620809,
-71.1041173835118 42.3150739481917,-71.1040809891419 42.3151344119048,
-71.1040438678912 42.3151191367447,-71.1040194562988 42.3151832057859,
-71.1038734225584 42.3151140942995,-71.1038446938243 42.3151006300338,
-71.1038315271889 42.315094347535,-71.1037393329282 42.315054824985,
-71.1035447555574 42.3152608696313,-71.1033436658644 42.3151648370544,
-71.1032580383161 42.3152269126061,-71.103223066939 42.3152517403219,
-71.1031880899493 42.3152774590236)),
((-71.1043632495873 42.315113108546,-71.1043583974082 42.3151211109857,
-71.1043443253471 42.3150676015829,-71.1043850704575 42.3150793250568,-71.1043632495873 ↩
    42.315113108546)))') AS geom;
```

Output

```
geometry_type   | srid
------------------+------
 ST_MultiPolygon   | 4269
(1 row)
```

Create a 3D CircularString without an SRID.

Code

```
SELECT ST_GeometryType(geom) AS geometry_type, ST_SRID(geom) AS srid
FROM ST_GeomFromEWKT('CIRCULARSTRING(220268 150415 1,220227 150505 2,220227 150406 3)') AS ↩
    geom;
```

Output

```
geometry_type    | srid
-------------------+------
 ST_CircularString  | 0
(1 row)
```

Create a PolyhedralSurface without an SRID.

Code

```
SELECT ST_GeometryType(geom) AS geometry_type, ST_SRID(geom) AS srid
FROM ST_GeomFromEWKT('POLYHEDRALSURFACE(
    ((0 0 0,0 0 1,0 1 1,0 1 0,0 0 0)),
    ((0 0 0,0 1 0,1 1 0,1 0 0,0 0 0)),
    ((0 0 0,1 0 0,1 0 1,0 0 1,0 0 0)),
    ((1 1 0,1 1 1,1 0 1,1 0 0,1 1 0)),
    ((0 1 0,0 1 1,1 1 1,1 1 0,0 1 0)),
    ((0 0 1,1 0 1,1 1 1,0 1 1,0 0 1))
)') AS geom;
```

Output

```
geometry_type       | srid
-------------------------+------
 ST_PolyhedralSurface     | 0
(1 row)
```

Siehe auch

ST_AsEWKT, ST_GeomFromText

7.8.1.7 ST_GeomFromMARC21

ST_GeomFromMARC21 — Nimmt MARC21/XML-Geodaten als Eingabe und gibt ein PostGIS-Geometrieobjekt zurück.

Synopsis

geometry **ST_GeomFromMARC21** (text marcxml);

Beschreibung

Diese Funktion erstellt eine PostGIS-Geometrie aus einem MARC21/XML-Datensatz, der einen POINT oder ein POLYGON enthalten kann. Bei mehreren geografischen Dateneinträgen im selben MARC21/XML-Datensatz wird ein MULTIPOINT oder MULTIPOLYGON zurückgegeben. Wenn der Datensatz gemischte Geometrietypen enthält, wird GEOMETRYCOLLECTION zurückgegeben. Es wird NULL zurückgegeben, wenn der MARC21/XML-Datensatz keine geografischen Daten enthält (Datenfeld:034).

LOC MARC21/XML-Versionen werden unterstützt:

- MARC21/XML 1.1

Verfügbarkeit: 3.3.0, erfordert libxml2 2.6+



Anmerkung
MARC21/XML Coded Cartographic Mathematical Data bietet derzeit keine Möglichkeit, das räumliche Bezugssystem der kodierten Koordinaten zu beschreiben, so dass diese Funktion immer eine Geometrie mit SRID 0 zurückgibt.



Anmerkung
Die zurückgegebenen POLYGON Geometrien sind immer im Uhrzeigersinn ausgerichtet.

Beispiele

Konvertierung von MARC21/XML-Geodaten, die einen einzelnen POINT enthalten, kodiert als hddd.dddddd

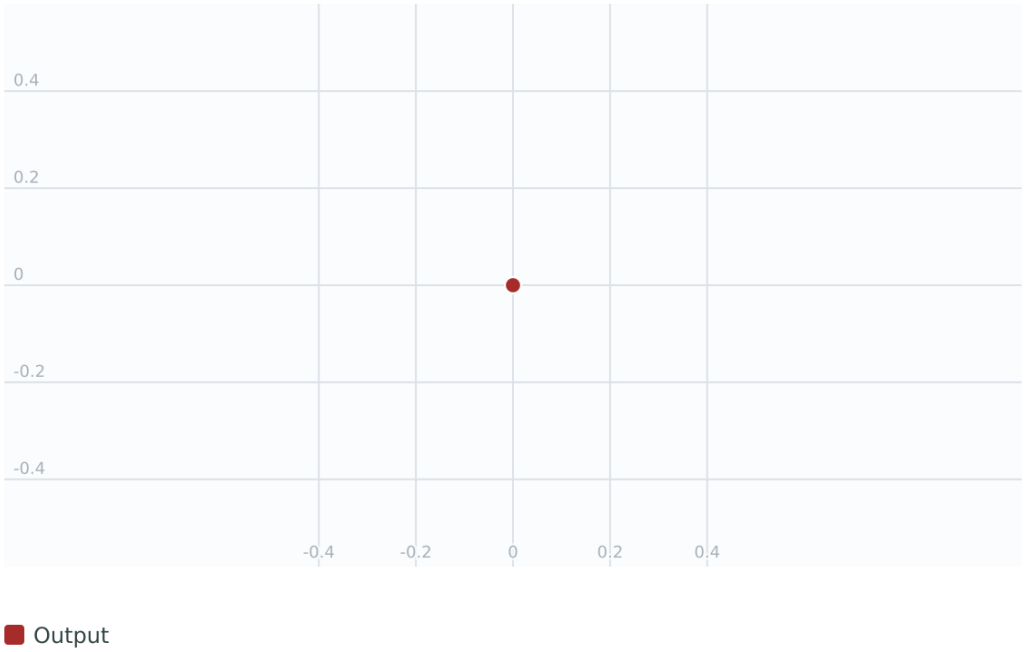
Code

```
SELECT ST_GeomFromMARC21(
'<record xmlns="http://www.loc.gov/MARC21/slim">
    <leader
>00000nz a2200000nc 4500</leader>
    <controlfield tag="001"
>040277569</controlfield>
    <datafield tag="034" ind1=" " ind2=" ">
        <subfield code="d"
>W004.500000</subfield>
        <subfield code="e"
>W004.500000</subfield>
        <subfield code="f"
>N054.250000</subfield>
        <subfield code="g"
>N054.250000</subfield>
    </datafield>
</record
>'
);
```

Output

```
POINT(-4.5 54.25)
```

Figure



Konvertierung von MARC21/XML-Geodaten, die eine einzelne `POLYGON` kodiert als `hdddmmss`

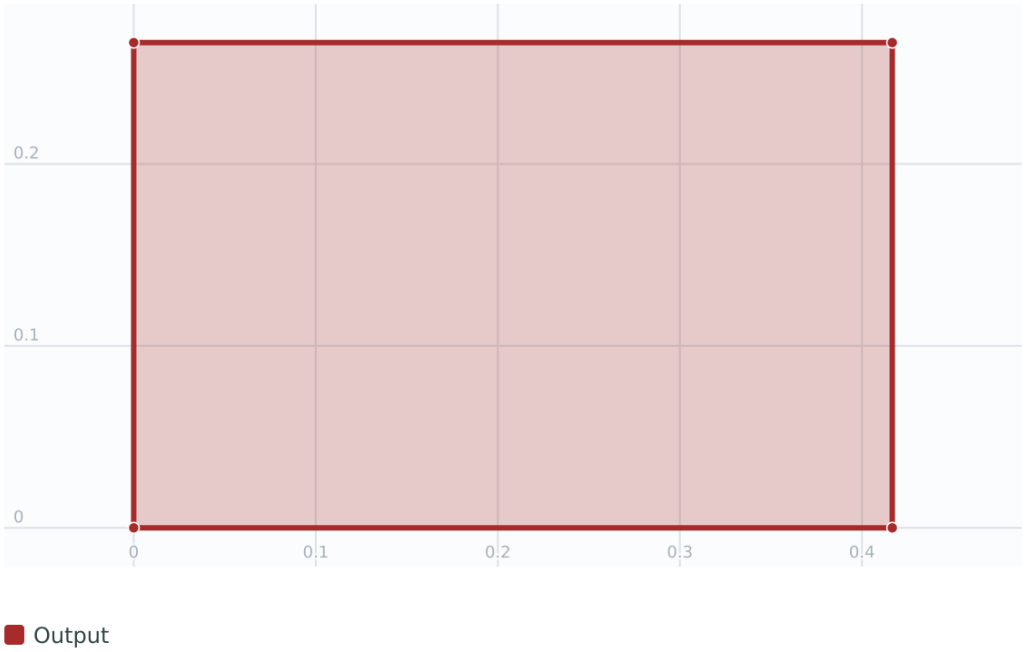
Code

```
SELECT ST_GeomFromMARC21(
'<record xmlns="http://www.loc.gov/MARC21/slim">
    <leader
>01062cem a2200241 a 4500</leader>
    <controlfield tag="001"
>   84696781 </controlfield>
    <datafield tag="034" ind1="1" ind2=" ">
        <subfield code="a"
>a</subfield>
        <subfield code="b"
>50000</subfield>
        <subfield code="d"
>E0130600</subfield>
        <subfield code="e"
>E0133100</subfield>
        <subfield code="f"
>N0523900</subfield>
        <subfield code="g"
>N0522300</subfield>
    </datafield>
</record
>'
);
```

Output

```
POLYGON((13.1 52.65,13.516666666666667 52.65,13.516666666666667 52.38333333333333,13.1 52.38333333333333,13.1 52.65))
```

Figure



Konvertierung von MARC21/XML-Geodaten, die ein `POLYGON` und ein `POINT` enthalten:

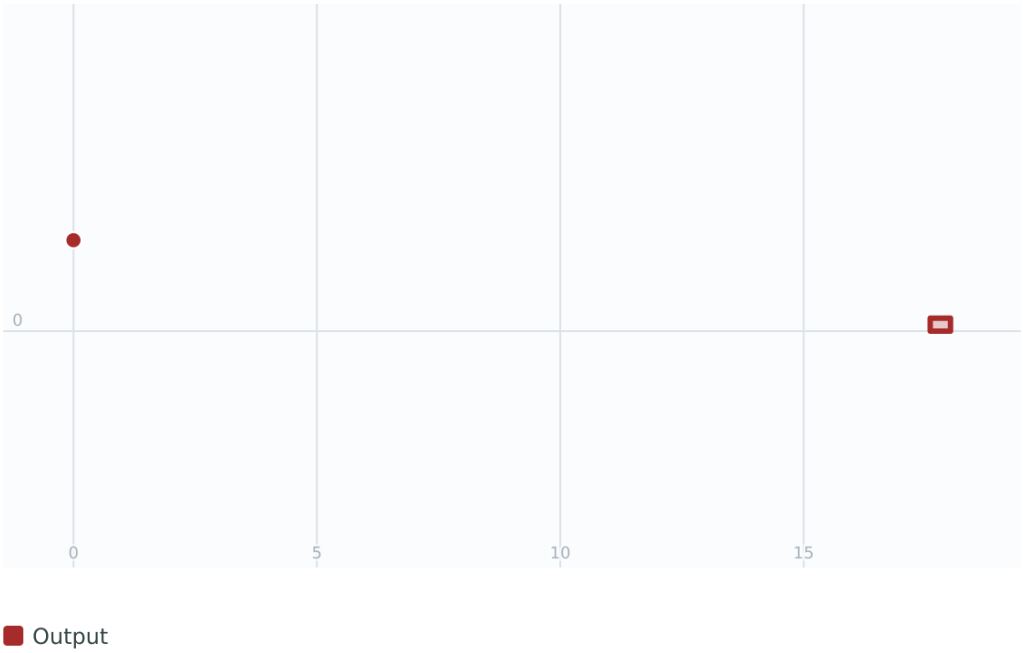
Code

```
SELECT ST_GeomFromMARC21(
'<record xmlns="http://www.loc.gov/MARC21/slim">
    <datafield tag="034" ind1="1" ind2=" ">
        <subfield code="a"
>a</subfield>
        <subfield code="b"
>50000</subfield>
        <subfield code="d"
>E0130600</subfield>
        <subfield code="e"
>E0133100</subfield>
        <subfield code="f"
>N0523900</subfield>
        <subfield code="g"
>N0522300</subfield>
    </datafield>
    <datafield tag="034" ind1=" " ind2=" ">
        <subfield code="d"
>W004.500000</subfield>
        <subfield code="e"
>W004.500000</subfield>
        <subfield code="f"
>N054.250000</subfield>
        <subfield code="g"
>N054.250000</subfield>
    </datafield>
</record
>'
);
```

Output

```
GEOMETRYCOLLECTION(POLYGON((13.1 52.65,13.516666666666667 52.65,13.516666666666667 52.38333333333333,13.1 ↩
    52.38333333333333,13.1 52.65)), POINT(-4.5 54.25))
```

Figure



Siehe auch

ST_AsMARC21

7.8.1.8 ST_GeometryFromText

ST_GeometryFromText — Returns a geometry from Well-Known Text (WKT). Alias for ST_GeomFromText.

Synopsis

geometry **ST_GeometryFromText**(text WKT);
geometry **ST_GeometryFromText**(text WKT, integer srid);

Beschreibung

This function is an alias for ST_GeomFromText and provides the same signatures.

Verfügbarkeit: 1.2.2

Diese Methode implementiert die OGC Simple Features Implementation Specification for SQL 1.1.

Diese Methode setzt die SQL/MM-Spezifikation um. SQL-MM 3: 5.1.40

Siehe auch

ST_GeomFromText

7.8.1.9 ST_GeomFromText

ST_GeomFromText — Gibt einen spezifizierten ST_Geometry Wert aus einer Well-known-Text Darstellung (WKT) zurück.

Synopsis

geometry **ST_GeomFromText**(text WKT);
geometry **ST_GeomFromText**(text WKT, integer srid);

Beschreibung

Erzeugt ein PostGIS ST_Geometry Objekt aus der OGC Well-known-Text Darstellung.



Anmerkung
Die Funktion ST_GeomFromText hat zwei Varianten. Die erste Variante nimmt keine SRID entgegen und gibt eine Geometrie ohne ein bestimmtes Koordinatenreferenzsystem aus (SRID=0) . Die zweite Variante nimmt eine SRID als zweiten Übergabewert entgegen und gibt eine Geometrie zurück, die diese SRID als Teil ihrer Metadaten beinhaltet.

 Diese Methode implementiert die OGC Simple Features Implementation Specification for SQL 1.1. s3.2.6.2 - die Option SRID ist vom Konformitätstest.

 Diese Methode setzt die SQL/MM-Spezifikation um. SQL-MM 3: 5.1.40

 Diese Methode unterstützt kreisförmige Strings und Kurven.



Anmerkung
Obwohl nicht OGC-konform, ist ST_MakePoint schneller als ST_GeomFromText und ST_PointFromText. Es ist auch einfacher für numerische Koordinatenwerte zu verwenden. ST_Point ist eine weitere Option, die ähnlich schnell wie ST_MakePoint ist und OGC-konform ist, aber nur 2D-Punkte unterstützt.

Änderung: 2.0.0 - In Vorgängerversionen von PostGIS war ST_GeomFromText('GEOMETRYCOLLECTION(EMPTY)') erlaubt. Um eine bessere Übereinstimmung mit der SQL/MM Norm zu erreichen, ist dies in PostGIS 2.0.0 nun nicht mehr gestattet. Hier sollte nun ST_GeomFromText('GEOMETRYCOLLECTION EMPTY') geschrieben werden.

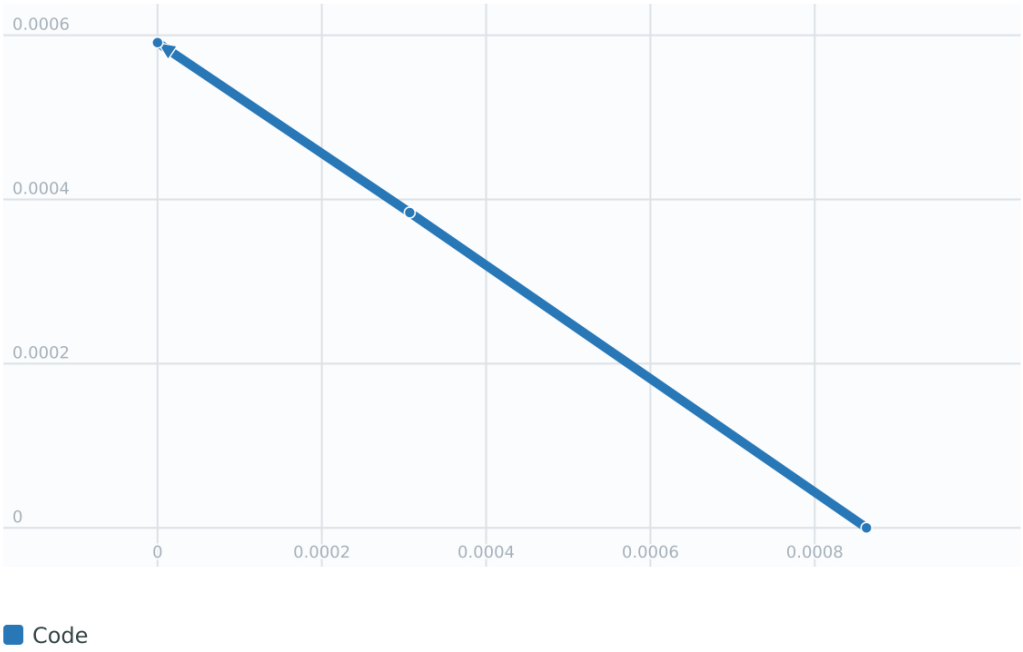
Beispiele

Create a LineString with SRID 0.

Code

```
SELECT ST_SRID(ST_GeomFromText('LINESTRING(-71.160281 42.258729,-71.160837 ↩
    42.259113,-71.161144 42.25932)'));
```

Figure

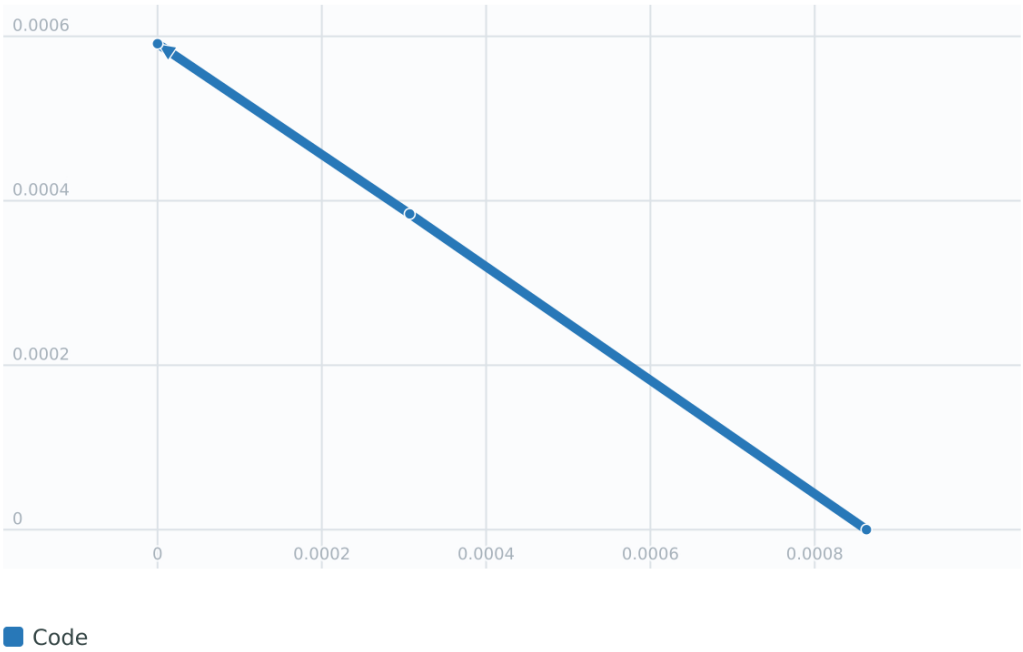


Create the same LineString with SRID 4269.

Code

```
SELECT ST_SRID(ST_GeomFromText('LINESTRING(-71.160281 42.258729,-71.160837 ↩
    42.259113,-71.161144 42.25932)', 4269));
```

Figure

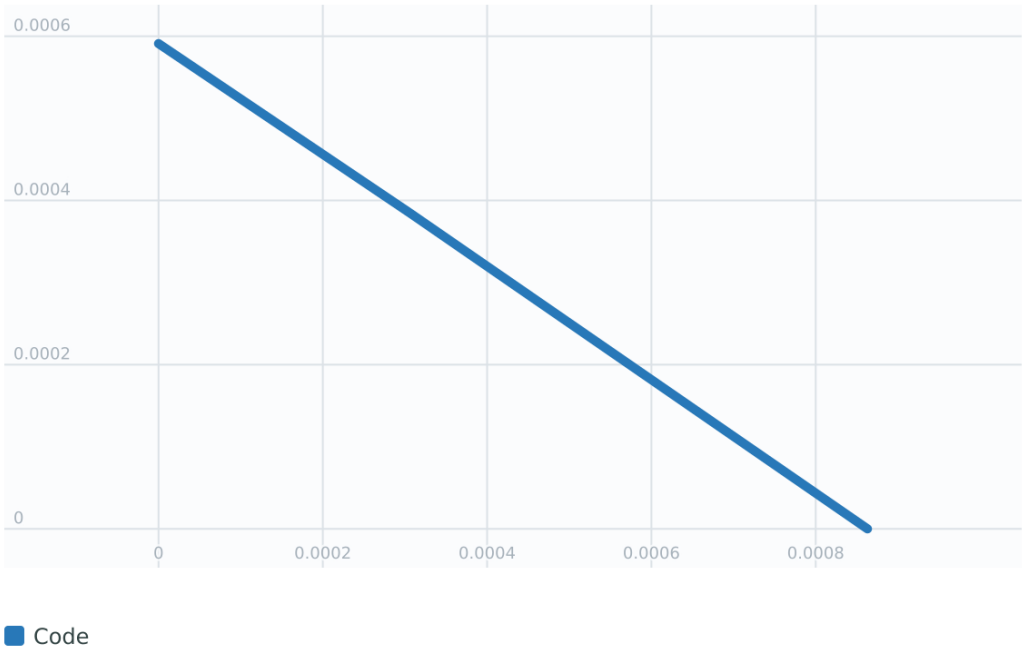


Create a MultiLineString.

Code

```
SELECT ST_GeometryType(ST_GeomFromText('MULTILINESTRING((-71.160281 42.258729,-71.160837 ↩
    42.259113,-71.161144 42.25932)'));
```

Figure



Create a Point.

Code

```
SELECT ST_GeometryType(ST_GeomFromText('POINT(-71.064544 42.28787)'));
```

Output

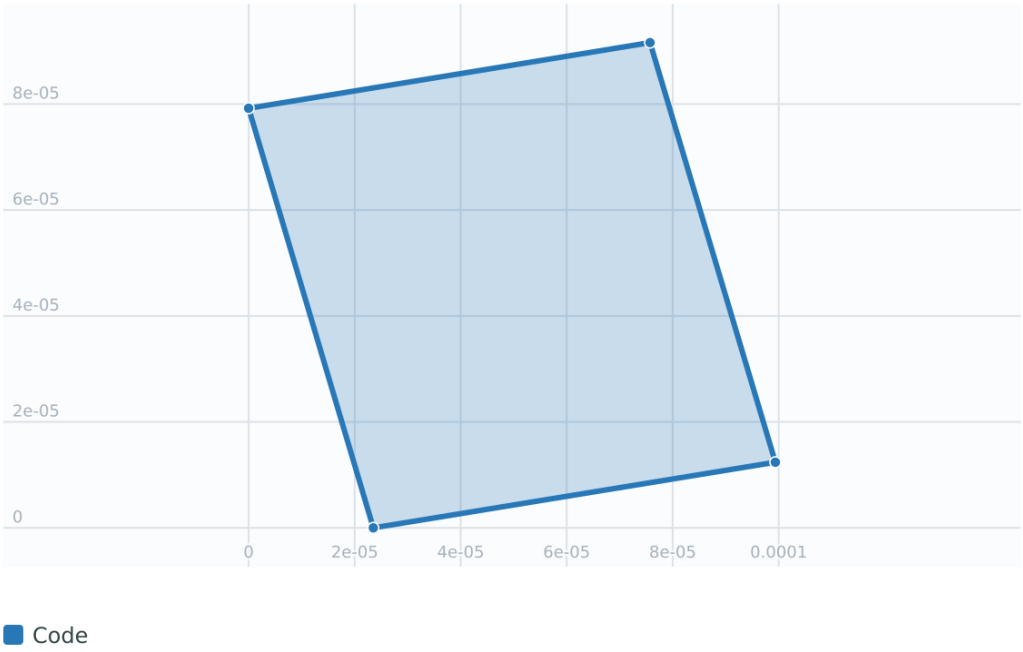
```
ST_Point
```

Create a Polygon.

Code

```
SELECT ST_GeometryType(ST_GeomFromText('POLYGON((-71.1776585052917 ↩
    42.3902909739571,-71.1776820268866 42.3903701743239,
-71.1776063012595 42.3903825660754,-71.1775826583081 42.3903033653531,-71.1776585052917 ↩
    42.3902909739571))'));
```

Figure

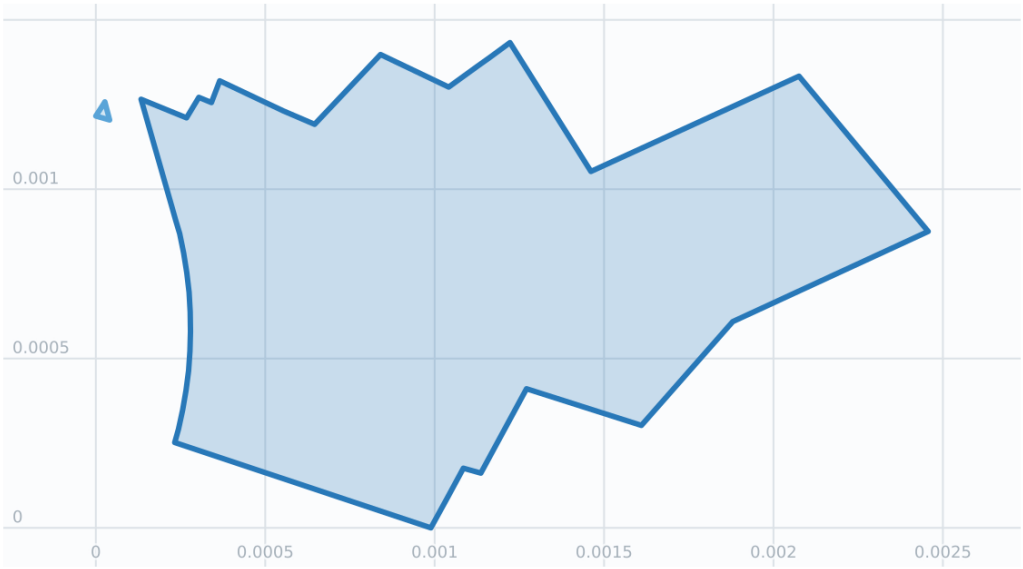


Create a MultiPolygon with SRID 4326.

Code

```
SELECT ST_GeometryType(ST_GeomFromText('MULTIPOLYGON(((-71.1031880899493 42.3152774590236,
-71.1031627617667 42.3152960829043,-71.102923838298 42.3149156848307,
-71.1023097974109 42.3151969047397,-71.1019285062273 42.3147384934248,
-71.102505233663 42.3144722937587,-71.10277487471 42.3141658254797,
-71.103113945163 42.3142739188902,-71.10324876416 42.31402489987,
-71.1033002961013 42.3140393340215,-71.1033488797549 42.3139495090772,
-71.103396240451 42.3138632439557,-71.1041521907712 42.3141153348029,
-71.1041411411543 42.3141545014533,-71.1041287795912 42.3142114839058,
-71.1041188134329 42.3142693656241,-71.1041112482575 42.3143272556118,
-71.1041072845732 42.3143851580048,-71.1041057218871 42.3144430686681,
-71.1041065602059 42.3145009876017,-71.1041097995362 42.3145589148055,
-71.1041166403905 42.3146168544148,-71.1041258822717 42.3146748022936,
-71.1041375307579 42.3147318674446,-71.1041492906949 42.3147711126569,
-71.1041598612795 42.314808571739,-71.1042515013869 42.3151287620809,
-71.1041173835118 42.3150739481917,-71.1040809891419 42.3151344119048,
-71.1040438678912 42.3151191367447,-71.1040194562988 42.3151832057859,
-71.1038734225584 42.3151140942995,-71.1038446938243 42.3151006300338,
-71.1038315271889 42.315094347535,-71.1037393329282 42.315054824985,
-71.1035447555574 42.3152608696313,-71.1033436658644 42.3151648370544,
-71.1032580383161 42.3152269126061,-71.103223066939 42.3152517403219,
-71.1031880899493 42.3152774590236)),
((-71.1043632495873 42.315113108546,-71.1043583974082 42.3151211109857,
-71.1043443253471 42.3150676015829,-71.1043850704575 42.3150793250568,-71.1043632495873 ↩
    42.315113108546)))', 4326));
```

Figure

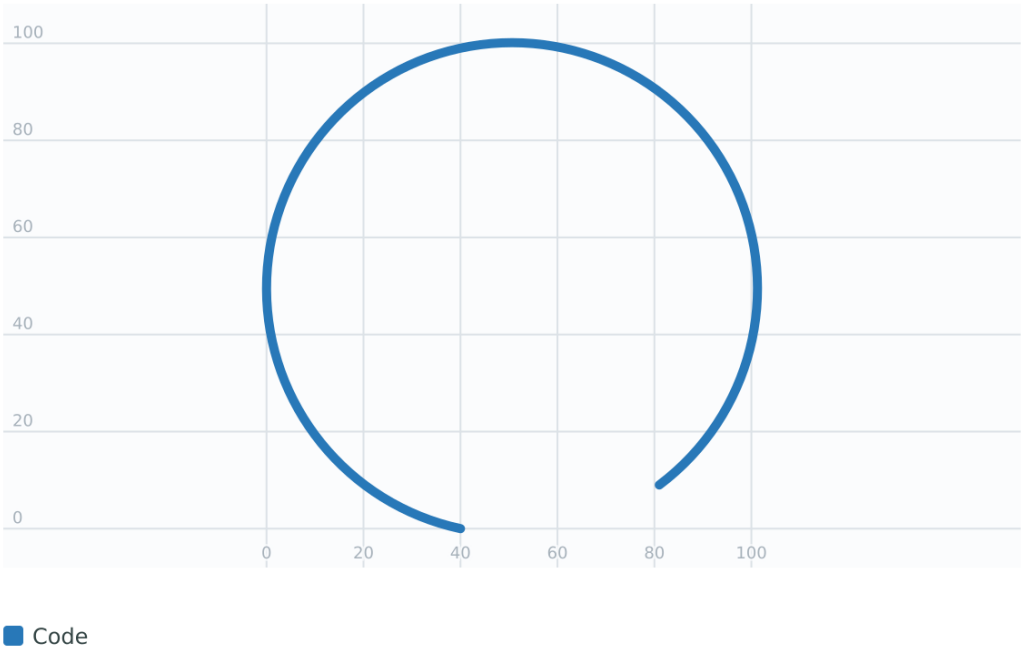


Create a CircularString.

Code

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

Figure



Siehe auch

ST_GeomFromEWKT, ST_GeomFromWKB, ST_SRID

7.8.1.10 ST_LineFromText

ST_LineFromText — Erzeugt eine Geometrie aus einer WKT Darstellung mit der angegebenen SRID. Wenn keine SRID angegeben wird, wird diese standardmäßig auf 0 gesetzt.

Synopsis

geometry **ST_LineFromText**(text WKT);
geometry **ST_LineFromText**(text WKT, integer srid);

Beschreibung

Erzeugt eine Geometrie aus einer WKT Darstellung mit der angegebenen SRID. Wenn keine SRID angegeben wird, wird diese standardmäßig auf 0 gesetzt. Wenn das übergebene WKT kein LineString ist, wird NULL zurückgegeben.



Anmerkung
OGC SPEC 3.2.6.2 - optionale SRID ist vom Konformitätstest.



Anmerkung
Wenn Sie wissen, dass die Geometrie nur aus LINESTRINGs besteht, ist es effizienter einfach ST_GeomFromText zu verwenden. Diese Funktion ruft auch nur ST_GeomFromText auf und fügt die Information hinzu, dass es sich um einen Linienzug handelt.

 Diese Methode implementiert die OGC Simple Features Implementation Specification for SQL 1.1. s3.2.6.2

 Diese Methode setzt die SQL/MM-Spezifikation um. SQL-MM 3: 7.2.8

Beispiele

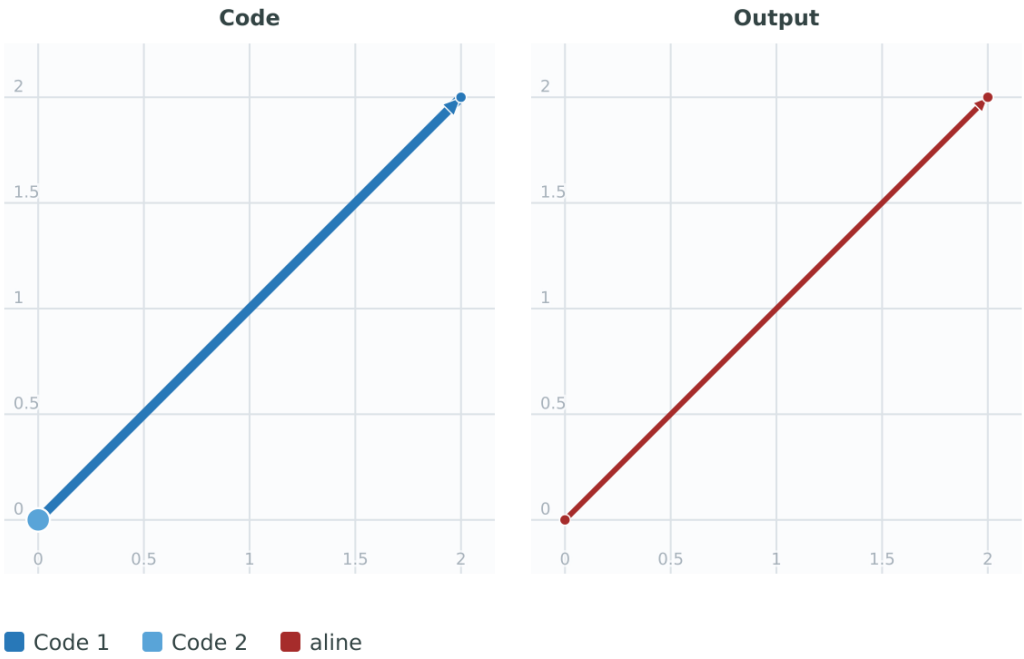
Code

```
SELECT ST_LineFromText('LINESTRING(1 2,3 4)') AS aline,
ST_LineFromText('POINT(1 2)') IS NULL AS null_return;
```

Output

```
aline                | null_return
-------------------------------------
LINESTRING(1 2,3 4)  | t
```

Figure



Siehe auch

ST_GeomFromText

7.8.1.11 ST_MLineFromText

ST_MLineFromText — Liest einen festgelegten ST_MultiLineString Wert von einer WKT-Darstellung aus.

Synopsis

geometry **ST_MLineFromText**(text WKT, integer srid);
geometry **ST_MLineFromText**(text WKT);

Beschreibung

Erzeugt eine Geometrie aus einer Well-known-Text (WKT) Darstellung mit der angegebenen SRID. Wenn keine SRID angegeben wird, wird diese standardmäßig auf 0 gesetzt.

OGC SPEC 3.2.6.2 - Die Option SRID stammt aus dem Konformitätstest

Gibt NULL zurück wenn der WKT kein MULTILINESTRING ist.



Anmerkung
Verwenden Sie diese Funktion nicht, wenn Sie sich vollkommen sicher sind, dass ihre WKT Geometrie nur aus Punkten besteht. Sie ist langsamer als ST_GeomFromText, da sie einen zusätzlichen Validierungsschritt hinzufügt.

 Diese Methode implementiert die OGC Simple Features Implementation Specification for SQL 1.1. s3.2.6.2

 Diese Methode setzt die SQL/MM-Spezifikation um. SQL-MM 3: 9.4.4

Beispiele

Code

```
SELECT ST_MLineFromText('MULTILINESTRING((1 2,3 4),(4 5,6 7))');
```

Siehe auch

ST_GeomFromText

7.8.1.12 ST_MPointFromText

ST_MPointFromText — Erzeugt eine Geometrie aus WKT mit der angegebenen SRID. Wenn keine SRID angegeben wird, wird diese standardmäßig auf 0 gesetzt.

Synopsis

geometry **ST_MPointFromText**(text WKT, integer srid);
geometry **ST_MPointFromText**(text WKT);

Beschreibung

Erzeugt eine Geometrie aus WKT mit der angegebenen SRID. Wenn SRID nicht angegeben ist, wird sie standardmäßig auf 0 gesetzt.

OGC SPEC 3.2.6.2 - Die Option SRID stammt aus dem Konformitätstest

Gibt NULL zurück, wenn der WKT kein MULTIPOINT ist.



Anmerkung
Verwenden Sie diese Funktion nicht, wenn Sie sich vollkommen sicher sind, dass ihre WKT Geometrie nur aus Punkten besteht. Sie ist langsamer als ST_GeomFromText, da sie einen zusätzlichen Validierungsschritt hinzufügt.

 Diese Methode implementiert die OGC Simple Features Implementation Specification for SQL 1.1. 3.2.6.2

 Diese Methode setzt die SQL/MM-Spezifikation um. SQL-MM 3: 9.2.4

Beispiele

Code

```
SELECT ST_MPointFromText('MULTIPOINT((1 2),(3 4))');
SELECT ST_MPointFromText('MULTIPOINT((-70.9590 42.1180),(-70.9611 42.1223))', 4326);
```

Siehe auch

ST_GeomFromText

7.8.1.13 ST_MPolyFromText

ST_MPolyFromText — Erzeugt eine MultiPolygon Geometrie aus WKT mit der angegebenen SRID. Wenn SRID nicht angegeben ist, wird sie standardmäßig auf 0 gesetzt.

Synopsis

geometry **ST_MPolyFromText**(text WKT, integer srid);
geometry **ST_MPolyFromText**(text WKT);

Beschreibung

Erzeugt ein MultiPolygon von WKT mit der gegebenen SRID. Wenn SRID nicht angegeben ist, wird sie standardmäßig auf 0 gesetzt.

OGC SPEC 3.2.6.2 - Die Option SRID stammt aus dem Konformitätstest

Meldet einen Fehler, wenn der WKT kein MULTIPOLYGON ist.



Anmerkung
Verwenden Sie diese Funktion nicht, wenn Sie sich vollkommen sicher sind, dass ihre WKT Geometrie nur aus Multi-Polygonen besteht. Sie ist langsamer als ST_GeomFromText, da sie einen zusätzlichen Validierungsschritt hinzufügt.

 Diese Methode implementiert die OGC Simple Features Implementation Specification for SQL 1.1. s3.2.6.2

 Diese Methode setzt die SQL/MM-Spezifikation um. SQL-MM 3: 9.6.4

Beispiele

Create a 3D MultiPolygon with an interior ring and SRID 0.

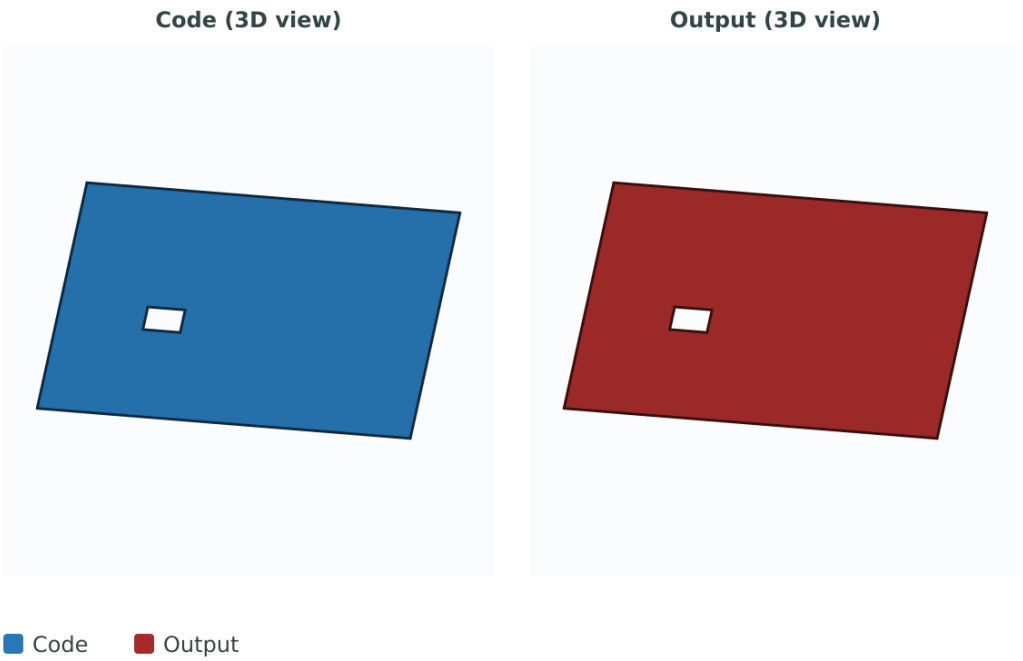
Code

```
SELECT ST_MPolyFromText(
    'MULTIPOLYGON(((0 0 1,20 0 1,20 20 1,0 20 1,0 0 1),
                   (5 5 3,5 7 3,7 7 3,7 5 3,5 5 3)))'
);
```

Output

```
MULTIPOLYGON(((0 0 1,20 0 1,20 20 1,0 20 1,0 0 1),(5 5 3,5 7 3,7 7 3,7 5 3,5 5 3)))
```

Figure

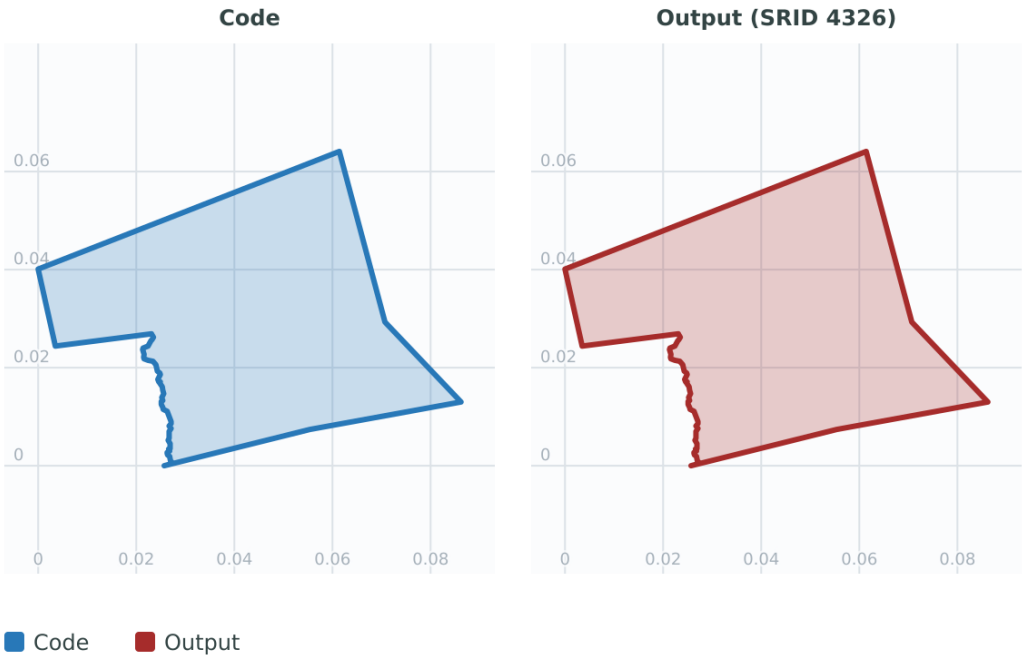


Create a MultiPolygon with SRID 4326.

Code

```
SELECT ST_MPolyFromText(
    'MULTIPOLYGON(((-70.916 42.1002,-70.9468 42.0946,-70.9765 42.0872,
      -70.9754 42.0875,-70.9749 42.0879,-70.9752 42.0881,-70.9754 42.0891,
      -70.9758 42.0894,-70.9759 42.0897,-70.9759 42.0899,-70.9754 42.0902,
      -70.9756 42.0906,-70.9753 42.0907,-70.9753 42.0917,-70.9757 42.0924,
      -70.9755 42.0928,-70.9755 42.0942,-70.9751 42.0948,-70.9755 42.0953,
      -70.9751 42.0958,-70.9751 42.0962,-70.9759 42.0983,-70.9767 42.0987,
      -70.9768 42.0991,-70.9771 42.0997,-70.9771 42.1003,-70.9768 42.1005,
      -70.977 42.1011,-70.9766 42.1019,-70.9768 42.1026,-70.9769 42.1033,
      -70.9775 42.1042,-70.9773 42.1043,-70.9776 42.1043,-70.9778 42.1048,
      -70.9773 42.1058,-70.9774 42.1061,-70.9779 42.1065,-70.9782 42.1078,
      -70.9788 42.1085,-70.9798 42.1087,-70.9806 42.109,-70.9807 42.1093,
      -70.9806 42.1099,-70.9809 42.1109,-70.9808 42.1112,-70.9798 42.1116,
      -70.9792 42.1127,-70.979 42.1129,-70.9787 42.1134,-70.979 42.1139,
      -70.9791 42.1141,-70.9987 42.1116,-71.0022 42.1273,-70.9408 42.1513,
      -70.9315 42.1165,-70.916 42.1002)))',
    4326
);
```

Figure



Siehe auch

ST_GeomFromText, ST_SRID

7.8.1.14 ST_PointFromText

ST_PointFromText — Erzeugt eine Punktgeometrie mit gegebener SRID von WKT. Wenn SRID nicht angegeben ist, wird sie standardmäßig auf 0 gesetzt.

Synopsis

geometry **ST_PointFromText**(text WKT);
geometry **ST_PointFromText**(text WKT, integer srid);

Beschreibung

Erzeugt ein PostGIS ST_Geometrie Punktobjekt von der OGC Well-known-Text Darstellung. Wenn die SRID nicht angegeben ist, wird sie standardmäßig auf "unknown" (zurzeit 0) gesetzt. Falls die Geometrie nicht in der WKT Punktdarstellung vorliegt, wird NULL zurückgegeben. Bei einer invaliden WKT Darstellung wird eine Fehlermeldung angezeigt.



Anmerkung
Die Funktion ST_PointFromText hat zwei Varianten. Die erste Variante nimmt keine SRID entgegen und gibt eine Geometrie ohne ein bestimmtes Koordinatenreferenzsystem aus. Die zweite Variante nimmt eine SRID als zweiten Übergabewert entgegen und gibt eine Geometrie zurück, die diese SRID als Teil ihrer Metadaten beinhaltet. Die SRID muss in der Tabelle "spatial_ref_sys" definiert sein.



Anmerkung
Verwenden Sie diese Funktion nicht, wenn Sie sich vollkommen sicher sind, dass ihre WKT Geometrie nur aus Punkten besteht. Sie ist langsamer als ST_GeomFromText, da sie einen zusätzlichen Validierungsschritt hinzufügt. Wenn Sie Punkte aus Koordinaten in Länge und Breite erstellen und mehr auf Rechenleistung und Genauigkeit wertlegen als auf OGC-Konformität, so verwenden Sie bitte ST_MakePoint oder den OGC-konformen Alias ST_Point.

 Diese Methode implementiert die OGC Simple Features Implementation Specification for SQL 1.1. s3.2.6.2 - die Option SRID ist vom Konformitätstest.

 Diese Methode setzt die SQL/MM-Spezifikation um. SQL-MM 3: 6.1.8

Beispiele

Create a point with the default SRID 0.

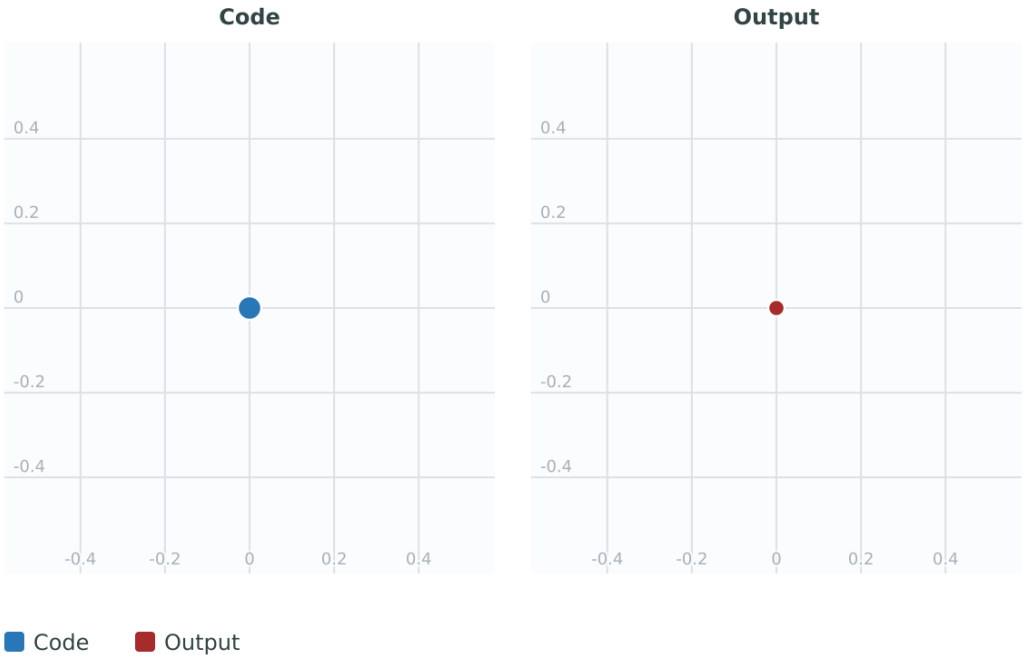
Code

```
SELECT ST_PointFromText('POINT(-71.064544 42.28787)');
```

Output

```
POINT(-71.064544 42.28787)
```

Figure



Create a point with SRID 4326.

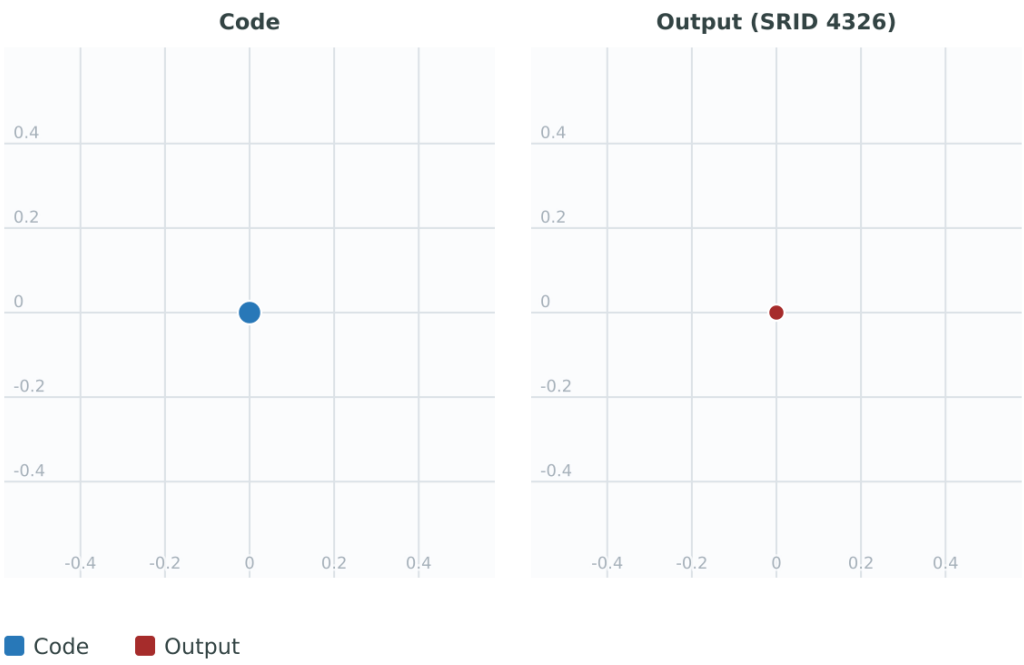
Code

```
SELECT ST_PointFromText('POINT(-71.064544 42.28787)', 4326);
```

Output

```
SRID=4326;POINT(-71.064544 42.28787)
```

Figure



Siehe auch

ST_GeomFromText, ST_MakePoint, ST_Point, ST_SRID

7.8.1.15 ST_PolygonFromText

ST_PolygonFromText — Erzeugt eine Geometrie aus WKT mit der angegebenen SRID. Wenn keine SRID angegeben wird, wird diese standardmäßig auf 0 gesetzt.

Synopsis

geometry **ST_PolygonFromText**(text WKT);
geometry **ST_PolygonFromText**(text WKT, integer srid);

Beschreibung

Erzeugt eine Geometrie mit gegebener SRID von WKT. Wenn SRID nicht angegeben ist, wird sie standardmäßig auf 0 gesetzt. Gibt NULL zurück, wenn WKT kein Polygon ist.

OGC SPEC 3.2.6.2 - Die Option SRID stammt aus dem Konformitätstest



Anmerkung
Verwenden Sie diese Funktion nicht, wenn Sie sich vollkommen sicher sind, dass ihre WKT Geometrie nur aus Polygonen besteht. Sie ist langsamer als ST_GeomFromText, da sie einen zusätzlichen Validierungsschritt hinzufügt.

Diese Methode implementiert die OGC Simple Features Implementation Specification for SQL 1.1. s3.2.6.2

Diese Methode setzt die SQL/MM-Spezifikation um. SQL-MM 3: 8.3.6

Beispiele

The result is returned as a native geometry. The documentation test runner compares that value directly with the readable EWKT shown below and uses the same geometry for the generated figure.

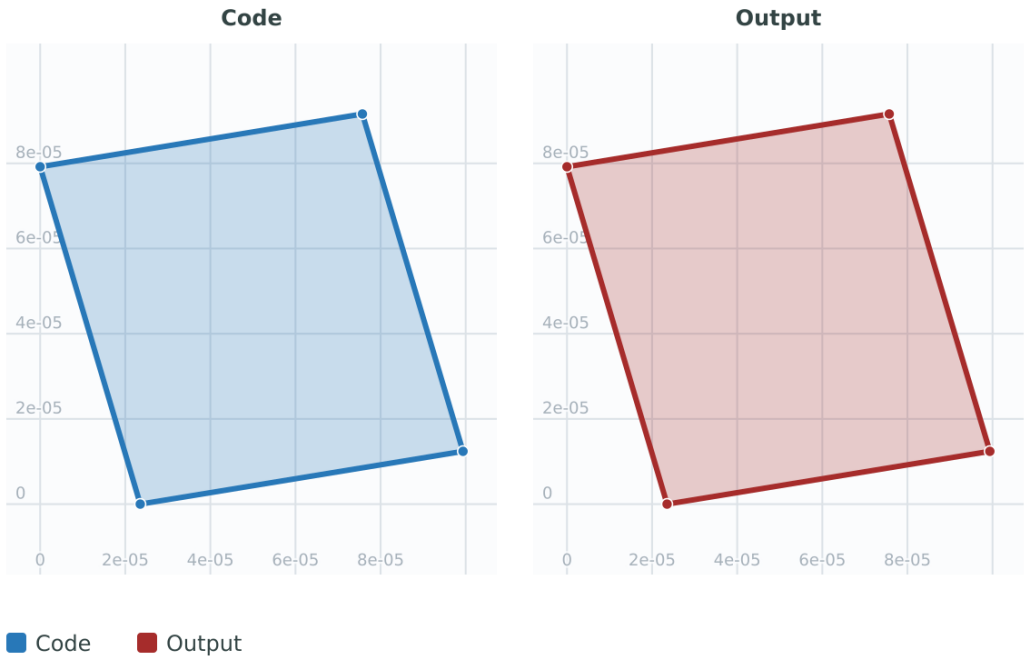
Code

```
SELECT ST_PolygonFromText('POLYGON((-71.1776585052917 42.3902909739571,-71.1776820268866 ↩
    42.3903701743239,
-71.1776063012595 42.3903825660754,-71.1775826583081 42.3903033653531,-71.1776585052917 ↩
    42.3902909739571))');
```

Output

```
POLYGON((-71.1776585052917 42.3902909739571,-71.1776820268866 42.3903701743239,-71.1776063012595 ↩
    42.3903825660754,-71.1775826583081 42.3903033653531,-71.1776585052917 42.3902909739571))
```

Figure



Code

```
SELECT ST_PolygonFromText('POINT(1 2)') IS NULL AS point_is_notpoly;
```

Output

```
t
```

Siehe auch

ST_GeomFromText

7.8.1.16 ST_WKTToSQL

ST_WKTToSQL — Returns a geometry from Well-Known Text (WKT). SQL/MM alias for ST_GeomFromText.

Synopsis

geometry **ST_WKTToSQL**(text WKT);

Beschreibung

This function is the SQL/MM alias for ST_GeomFromText with no SRID argument.

Diese Methode setzt die SQL/MM-Spezifikation um. SQL-MM 3: 5.1.34

Siehe auch

ST_GeomFromText

7.8.2 Well-known-Binary (WKB) Repräsentation

7.8.2.1 ST_GeogFromWKB

ST_GeogFromWKB — Erzeugt ein geographisches Objekt aus der Well-known-Binary (WKB) oder der erweiterten Well-known-Binary (EWKB) Darstellung.

Synopsis

geography **ST_GeogFromWKB**(bytea wkb);

Beschreibung

Die Funktion `ST_GeogFromWKB` empfängt eine Well-known-Binary (WKB) oder eine erweiterte PostGIS WKB (EWKB) Darstellung einer Geometrie und erzeugt eine Instanz des entsprechenden geographischen Datentyps. Diese Funktion übernimmt die Rolle der Geometrie-Factory/Fabrik in SQL.

Wenn die SRID nicht festgelegt ist, wird 4326 (WGS 84) angenommen.

Diese Methode unterstützt kreisförmige Strings und Kurven.

Beispiele

Although bytea rep contains single \, these need to be escaped when inserting into a table.

Code

```
SELECT ST_GeogFromWKB(E'\\001\\002\\000\\000\\000\\002\\000\\000\\000\\037\\205\\353Q ↩
    \\270~\\\\\\300\\323Mb\\020X\\231C@\\020X9\\264\\310~\\\\\\300)\\\\\\217\\302\\365\\230 ↩
    C@');
```

Output

```
LINESTRING(-113.98 39.198,-113.981 39.195)
```

Figure



Siehe auch

ST_GeogFromText, ST_AsBinary

7.8.2.2 ST_GeomFromEWKB

ST_GeomFromEWKB — Gibt einen geometrischen Datentyp (ST_Geometry) aus einer Well-known-Binary (WKB) Darstellung zurück.

Synopsis

geometry **ST_GeomFromEWKB**(bytea EWKB);

Beschreibung

Constructs a PostGIS geometry from an Extended Well-Known Binary (EWKB) representation.



Anmerkung
EWKB ist kein Format des OGC Standards, sondern ein PostGIS eigenes Format, welches den Identifikator (SRID) des räumlichen Koordinatenreferenzsystem mit einbindet

Erweiterung: Mit 2.0.0 wurde die Unterstützung für polyedrische Oberflächen und TIN eingeführt.

 Diese Funktion unterstützt 3d und lässt den Z-Index nicht fallen.

Diese Methode unterstützt kreisförmige Strings und Kurven.

Diese Funktion unterstützt polyedrische Flächen.

Diese Funktion unterstützt Dreiecke und dreieckige unregelmäßige Netzoberflächen (TIN).

Beispiele

This example constructs `LINESTRING(-71.160281 42.258729,-71.160837 42.259113,-71.161144 42.25932)` in NAD83 longitude/latitude (SRID 4269) from its EWKB representation.

PostgreSQL escape string syntax (`E'...'`) always interprets backslash escapes. Each backslash therefore has to be doubled so that the bytea input parser receives the octal escapes.

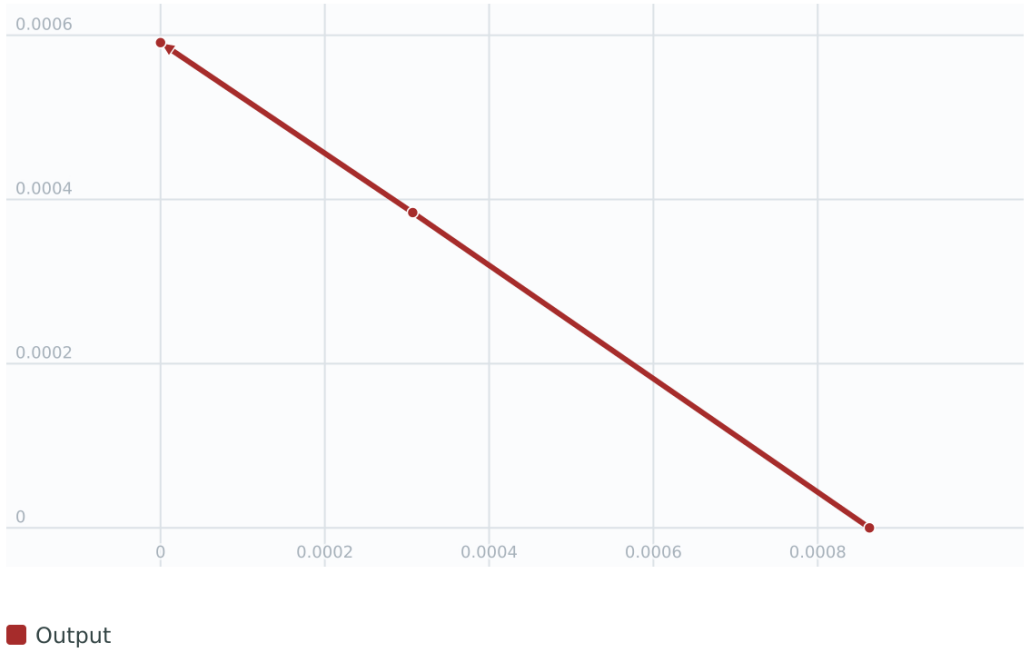
Code

```
SELECT ST_GeomFromEWKB((
    E'\\001\\002\\000\\000\\040\\255\\020\\000\\000\\003\\000\\000\\000\\344\\112\\075' ||
    E'\\013\\102\\312\\121\\300\\156\\303\\050\\010\\036\\041\\105\\100\\047\\277\\105' ||
    E'\\047\\113\\312\\121\\300\\366\\173\\142\\235\\052\\041\\105\\100\\225\\174\\354' ||
    E'\\056\\120\\312\\121\\300\\160\\231\\323\\145\\061\\041\\105\\100'
)::bytea);
```

Output

```
0102000020 ↩
    AD10000003000000E44A3D0B42CA51C06EC328081E21454027BF45274BCA51C0F67B629D2A214540957CEC2E50CA51C07099D36531214540
```

Figure



With `standard_conforming_strings` enabled (the default), ordinary string literals preserve backslashes. The same EWKB can therefore be written with a single backslash before each octal byte value.

Code

```
SELECT ST_GeomFromEWKB((
    '\001\002\000\000\040\255\020\000\000\003\000\000\000\344\112\075' ||
    '\013\102\312\121\300\156\303\050\010\036\041\105\100\047\277\105' ||
    '\047\113\312\121\300\366\173\142\235\052\041\105\100\225\174\354' ||
    '\056\120\312\121\300\160\231\323\145\061\041\105\100'
)::bytea);
```

Output

```
0102000020 ↩
    AD10000003000000E44A3D0B42CA51C06EC328081E21454027BF45274BCA51C0F67B629D2A214540957CEC2E50CA51C0
```

Siehe auch

ST_AsBinary, ST_AsEWKB, ST_GeomFromWKB

7.8.2.3 ST_GeomFromWKB

ST_GeomFromWKB — Erzeugt ein geometrisches Objekt aus der Well-known-Binary (WKB) Darstellung und einer optionalen SRID.

Synopsis

geometry **ST_GeomFromWKB**(bytea geom);
geometry **ST_GeomFromWKB**(bytea geom, integer srid);

Beschreibung

Die Funktion `ST_GeogFromWKB` nimmt eine Well-known-Binary (WKB) Darstellung und eine Id für das Koordinatenreferenzsystem (`SRID`) entgegen und erzeugt eine Instanz des entsprechenden geometrischen Datentyps. Diese Funktion übernimmt die Rolle der Geometrie-Factory in SQL. Ist eine alternative Bezeichnung für ST_WKBToSQL.

Wenn die SRID nicht festgelegt ist, wird sie standardmäßig auf 0 (Unknown) gesetzt.

Diese Methode implementiert die OGC Simple Features Implementation Specification for SQL 1.1. s3.2.7.2 - die optionale SRID kommt vom Konformitätstest.

Diese Methode setzt die SQL/MM-Spezifikation um. SQL-MM 3: 5.1.41

Diese Methode unterstützt kreisförmige Strings und Kurven.

Beispiele

Although bytea rep contains single \, these need to be escaped when inserting into a table. Unless standard_conforming_strings is set to on.

Code

```
SELECT ST_GeomFromWKB(E'\\001\\002\\000\\000\\000\\002\\000\\000\\000\\037\\205\\353Q ↩
    \\270~\\\\\\300\\323Mb\\020X\\231C@\\020X9\\264\\310~\\\\\\300)\\\\\\217\\302\\365\\230 ↩
    C@', 4326);
```

Output

```
SRID=4326;LINESTRING(-113.98 39.198,-113.981 39.195)
```

Figure



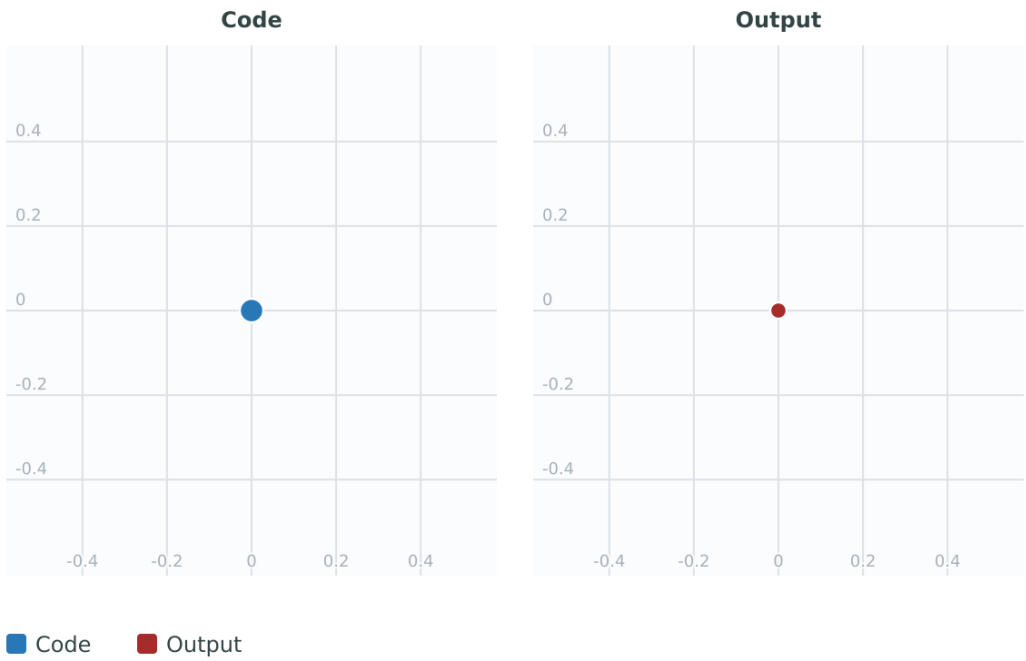
Code

```
SELECT ST_GeomFromWKB(ST_AsEWKB('POINT(2 5)'::geometry)
  );
```

Output

```
POINT(2 5)
```

Figure



The PostgreSQL `decode` function converts a hexadecimal WKB string to the bytea value accepted by `ST_GeomFromWKB`.

Code

```
SELECT ST_AsEWKT(ST_GeomFromWKB(
    decode('0101000000e5d022dbf93e2e40dbf97e6abc743540', 'hex'),
    4326
)) AS geom;
```

Output

```
SRID=4326;POINT(15.123 21.456)
```

Siehe auch

ST_WKBToSQL, ST_AsBinary, ST_AsEWKT, ST_GeomFromEWKB

7.8.2.4 ST_LineFromWKB

ST_LineFromWKB — Creates a `LINESTRING` from a WKB representation, optionally using a given SRID.

Synopsis

geometry **ST_LineFromWKB**(bytea WKB);
geometry **ST_LineFromWKB**(bytea WKB, integer srid);

Beschreibung

The `ST_LineFromWKB` function takes a Well-Known Binary (WKB) representation of a geometry and an optional Spatial Reference System ID (`SRID`), and creates a `LINESTRING` geometry. It is a type-specific geometry factory in SQL.

If an SRID is not specified, it defaults to 0. `NULL` is returned if the input `bytea` does not represent a `LINESTRING` geometry.

Anmerkung
OGC SPEC 3.2.6.2 - optionale SRID ist vom Konformitätstest.

Anmerkung
If you know all your geometries are `LINESTRING`s, it is more efficient to use ST_GeomFromWKB directly. This function calls ST_GeomFromWKB and adds validation that the result is a `LINESTRING`.

Diese Methode implementiert die OGC Simple Features Implementation Specification for SQL 1.1. s3.2.6.2

Diese Methode setzt die SQL/MM-Spezifikation um. SQL-MM 3: 7.2.9

Beispiele

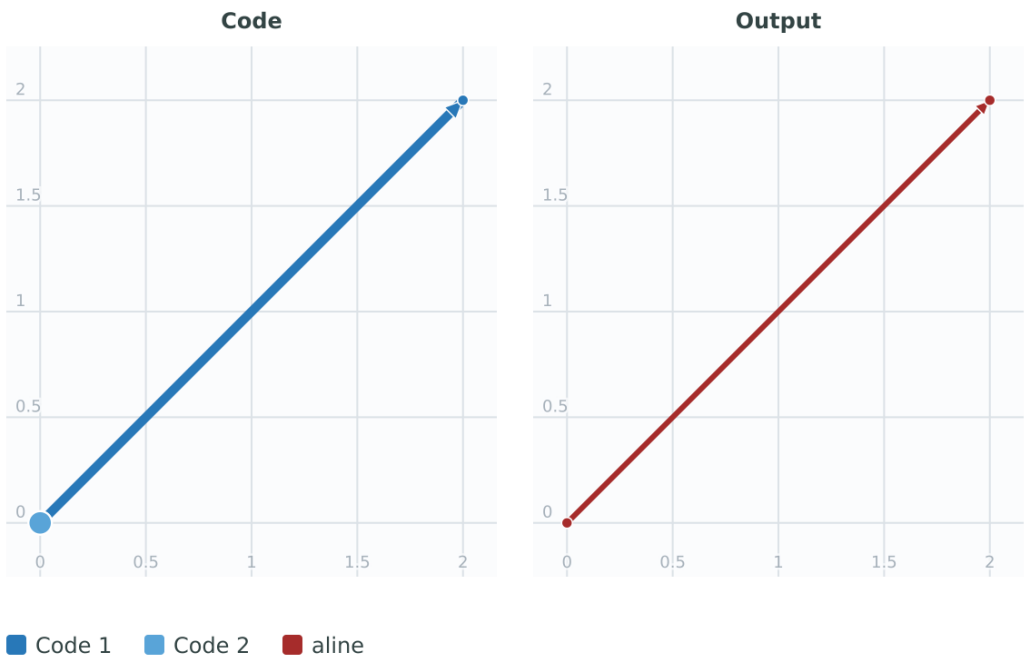
Code

```
SELECT ST_LineFromWKB(
    ST_AsBinary(ST_GeomFromText('LINESTRING(1 2,3 4)'))
) AS aline,
ST_LineFromWKB(
    ST_AsBinary(ST_GeomFromText('POINT(1 2)'))
) IS NULL AS null_return;
```

Output

```
aline                | null_return
-------------------------------------
LINESTRING(1 2,3 4)  | t
```

Figure



Siehe auch

ST_GeomFromWKB, ST_LinestringFromWKB

7.8.2.5 ST_LinestringFromWKB

ST_LinestringFromWKB — Creates a `LINESTRING` from a WKB representation, optionally using a given SRID.

Synopsis

geometry **ST_LinestringFromWKB**(bytea WKB);
geometry **ST_LinestringFromWKB**(bytea WKB, integer srid);

Beschreibung

The `ST_LinestringFromWKB` function takes a Well-Known Binary (WKB) representation of a geometry and an optional Spatial Reference System ID (`SRID`), and creates a `LINESTRING` geometry. It is an alias for ST_LineFromWKB.

If an SRID is not specified, it defaults to 0. `NULL` is returned if the input `bytea` does not represent a `LINESTRING` geometry.



Anmerkung
OGC SPEC 3.2.6.2 - optionale SRID ist vom Konformitätstest.



Anmerkung
If you know all your geometries are `LINESTRING`s, it is more efficient to use ST_GeomFromWKB directly. This function calls ST_GeomFromWKB and adds validation that the result is a `LINESTRING`.



Diese Methode implementiert die OGC Simple Features Implementation Specification for SQL 1.1. s3.2.6.2



Diese Methode setzt die SQL/MM-Spezifikation um. SQL-MM 3: 7.2.9

Beispiele

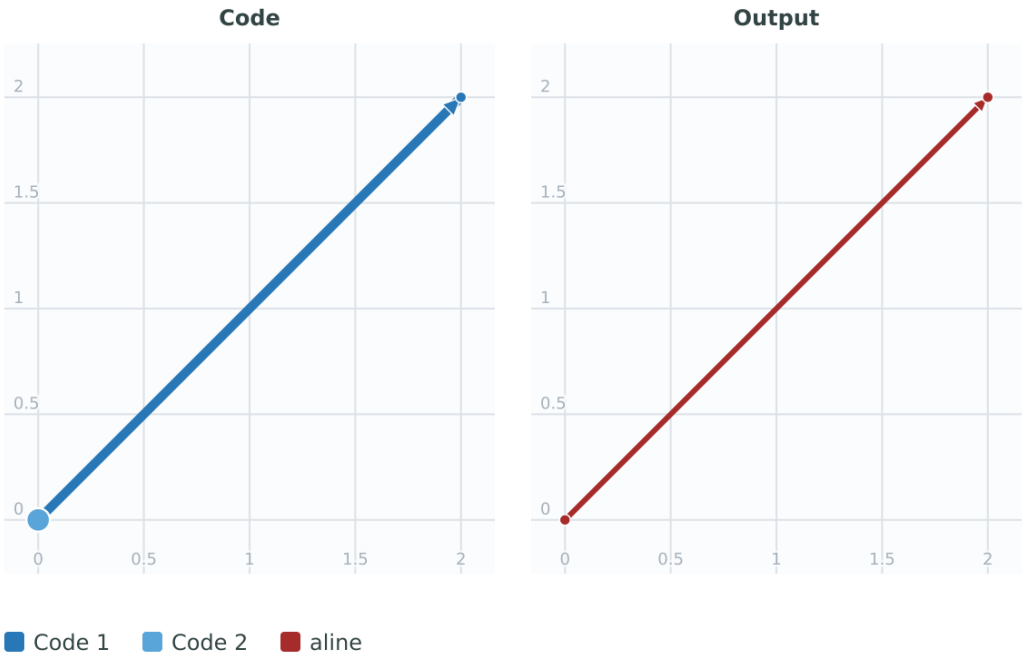
Code

```
SELECT ST_LinestringFromWKB(
    ST_AsBinary(ST_GeomFromText('LINESTRING(1 2,3 4)'))
) AS aline,
ST_LinestringFromWKB(
    ST_AsBinary(ST_GeomFromText('POINT(1 2)'))
) IS NULL AS null_return;
```

Output

```
aline                | null_return
---------------------------------
LINESTRING(1 2,3 4)  | t
```

Figure



Siehe auch

ST_GeomFromWKB, ST_LineFromWKB

7.8.2.6 ST_PointFromWKB

ST_PointFromWKB — Erzeugt eine Geometrie mit gegebener SRID von WKB.

Synopsis

geometry **ST_GeomFromWKB**(bytea geom);
geometry **ST_GeomFromWKB**(bytea geom, integer srid);

Beschreibung

Die Funktion `ST_PointFromWKB` nimmt eine Well-known-Binary Darstellung der Geometrie und eine Id für das Koordinatenreferenzsystem (`SRID`) entgegen und erzeugt eine Instanz des entsprechenden geometrischen Datentyps - in diesem Fall eine Geometrie vom Typ `POINT`. Diese Funktion übernimmt die Rolle der Geometrie-Factory in SQL.

Wenn keine SRID angegeben ist, wird diese standardmäßig auf 0 gesetzt. `NULL` wird zurückgegeben, wenn die Eingabe `bytea` keinen `POINT` darstellt.

Diese Methode implementiert die OGC Simple Features Implementation Specification for SQL 1.1. s3.2.7.2

Diese Methode setzt die SQL/MM-Spezifikation um. SQL-MM 3: 6.1.9

Diese Funktion unterstützt 3d und lässt den Z-Index nicht fallen.

Diese Methode unterstützt kreisförmige Strings und Kurven.

Beispiele

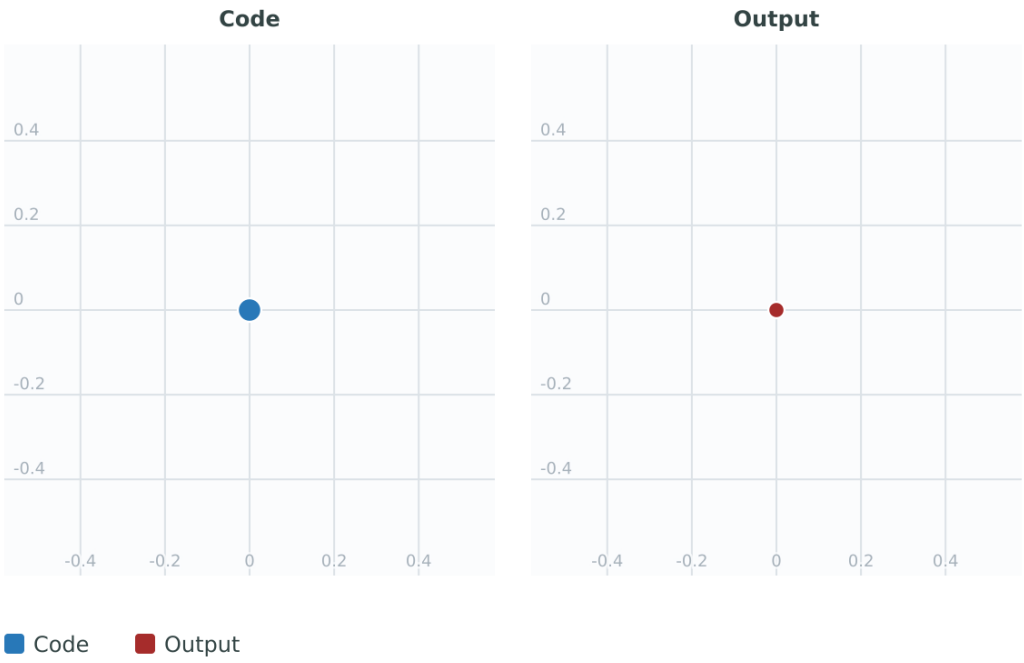
Code

```
SELECT ST_PointFromWKB(ST_AsEWKB('POINT(2 5)'::geometry));
```

Output

```
POINT(2 5)
```

Figure



Code

```
SELECT ST_PointFromWKB(ST_AsEWKB('LINESTRING(2 5,2 6)'::geometry));
```

Output

```
null
```

Siehe auch

ST_GeomFromWKB, ST_LineFromWKB

7.8.2.7 ST_WKBToSQL

ST_WKBToSQL — Returns a geometry from Well-Known Binary (WKB). SQL/MM alias for ST_GeomFromWKB without an SRID argument.

Synopsis

geometry **ST_WKBToSQL**(bytea WKB);

Beschreibung

This function is the SQL/MM alias for ST_GeomFromWKB with no SRID argument.

Diese Methode setzt die SQL/MM-Spezifikation um. SQL-MM 3: 5.1.36

Siehe auch

ST_GeomFromWKB

7.8.3 Weitere Formate

7.8.3.1 ST_Box2dFromGeoHash

ST_Box2dFromGeoHash — Gibt die BOX2D einer GeoHash Zeichenkette zurück.

Synopsis

box2d **ST_Box2dFromGeoHash**(text geohash, integer precision=full_precision_of_geohash);

Beschreibung

Gibt die BOX2D einer GeoHash Zeichenkette zurück.

Wenn keine Genauigkeit angegeben ist, gibt ST_Box2dFromGeoHash ein BOX2D zurück, das auf der vollen Genauigkeit des eingegebenen GeoHash-Strings basiert.

Wenn `precision` angegeben wird, verwendet ST_Box2dFromGeoHash entsprechend viele Zeichen des GeoHash um die BOX2D zu erzeugen. Niedrigere Werte erzeugen eine größere BOX2D und höhere Werte erhöhen die Genauigkeit.

Verfügbarkeit: 2.1.0

Beispiele

Code

```
SELECT ST_Box2dFromGeoHash('9qqj7nmxncgyy4d0dbxqz0');
```

Output

```
BOX(-115.172816 36.114646,-115.172816 36.114646)
```

Code

```
SELECT ST_Box2dFromGeoHash('9qqj7nmxncgyy4d0dbxqz0', 0);
```

Output

```
BOX(-180 -90,180 90)
```

Code

```
SELECT ST_Box2dFromGeoHash('9qqj7nmxncgyy4d0dbxqz0', 10);
```

Output

```
BOX(-115.17282128334 36.1146408319473,-115.172810554504 36.1146461963654)
```

Siehe auch

ST_GeoHash, ST_GeomFromGeoHash, ST_PointFromGeoHash

7.8.3.2 ST_GeomFromGeoHash

ST_GeomFromGeoHash — Gibt die Geometrie einer GeoHash Zeichenfolge zurück.

Synopsis

geometry **ST_GeomFromGeoHash**(text geohash, integer precision=full_precision_of_geohash);

Beschreibung

Gibt die Geometrie einer GeoHash Zeichenfolge zurück. Der geometrische Datentyp ist ein Polygon, das den GeoHash begrenzt.

Wenn keine `precision` angegeben wird, dann gibt ST_GeomFromGeoHash ein Polygon zurück, das auf der vollständigen Genauigkeit der GeoHash Zeichenfolge beruht.

Wenn `precision` angegeben wird, verwendet ST_GeomFromGeoHash entsprechend viele Zeichen des GeoHash, um das Polygon zu erzeugen.

Verfügbarkeit: 2.1.0

Beispiele

The cells are returned together so their relative sizes remain visible in a shared coordinate frame.

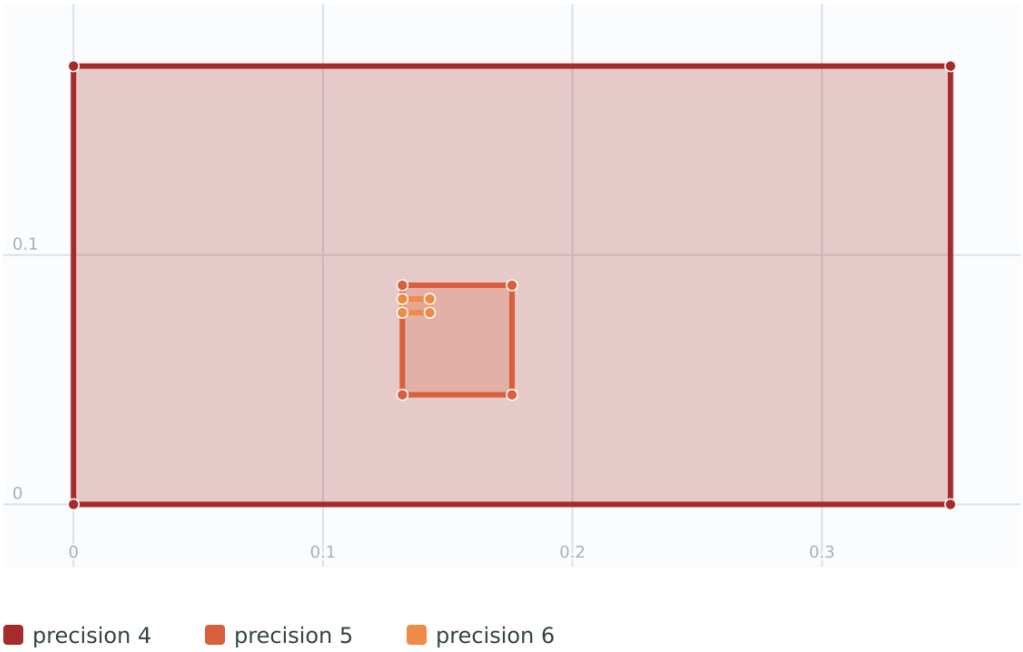
Code

```
SELECT ST_GeomFromGeoHash('9qqj7nmxncgyy4d0dbxqz0', 4) AS "precision 4", ST_GeomFromGeoHash( ↩
    '9qqj7nmxncgyy4d0dbxqz0', 5) AS "precision 5", ST_GeomFromGeoHash('9 ↩
    qqj7nmxncgyy4d0dbxqz0', 6) AS "precision 6";
```

Output

```
-[ RECORD 1 ]----------
precision 4 | POLYGON((-115.3125 36.03515625,-115.3125 36.2109375,-114.9609375 36.2109375,-114.9609375  ↩
     36.03515625,-115.3125 36.03515625))
precision 5 | POLYGON((-115.1806640625 36.0791015625,-115.1806640625 36.123046875,-115.13671875  ↩
     36.123046875,-115.13671875 36.0791015625,-115.1806640625 36.0791015625))
precision 6 | POLYGON((-115.1806640625 36.112060546875,-115.1806640625 36.1175537109375,-115.169677734375  ↩
     36.1175537109375,-115.169677734375 36.112060546875,-115.1806640625 36.112060546875))
```

Figure



Siehe auch

ST_GeoHash,ST_Box2dFromGeoHash, ST_PointFromGeoHash

7.8.3.3 ST_GeomFromGML

ST_GeomFromGML — Nimmt als Eingabe eine GML-Darstellung der Geometrie und gibt ein geometrisches PostGIS-Objekt aus.

Synopsis

geometry **ST_GeomFromGML**(text geomgml);
geometry **ST_GeomFromGML**(text geomgml, integer srid);

Beschreibung

Erzeugt ein PostGIS ST_Geometry Objekt aus der OGC GML Darstellung.

ST_GeomFromGML funktioniert nur bei Fragmenten von GML-Geometrien. Auf das ganze GML-Dokument angewendet führt zu einer Fehlermeldung.

Unterstützte OGC GML Versionen:

- GML 3.2.1 Namespace
- GML 3.1.1 Simple Features profile SF-2 (inkl. GML 3.1.0 und 3.0.0 Rückwertskompatibilität)
- GML 2.1.2

OGC GML Standards, vgl.: http://www.opengeospatial.org/standards/gml:

Verfügbarkeit: 1.5, benötigt libxml2 1.6+

Erweiterung: Mit 2.0.0 wurde die Unterstützung für polyedrische Oberflächen und TIN eingeführt.

Erweiterung: 2.0.0 Standardwert für den optionalen Parameter SRID eingefügt.

Enhanced: 3.7.0 support for GML ArcString, curved Ring, and CurvePolygon geometries was introduced.

 Diese Funktion unterstützt 3d und lässt den Z-Index nicht fallen.

 Diese Funktion unterstützt polyedrische Flächen.

 Diese Funktion unterstützt Dreiecke und dreieckige unregelmäßige Netzoberflächen (TIN).

GML erlaubt das Mischen von Dimensionen (z.B. 2D und 3D innerhalb der selben MultiGeometry). Da PostGIS Geometrien dies nicht zulassen, wandelt ST_GeomFromGML die gesamte Geometrie in 2D um, sobald eine fehlende Z-Dimension existiert.

GML unterstützt uneinheitliche Koordinatenreferenzsysteme innerhalb derselben Mehrfachgeometrie. Da dies der geometrische Datentyp von PostGIS nicht unterstützt, wird in diesem Fall die Subgeometrie in das Referenzsystem des Knotens, der die Wurzel darstellt, umprojiziert. Wenn kein Attribut "srsName" für den Knoten der GML-Wurzel vorhanden ist, gibt die Funktion eine Fehlermeldung aus.

Die Funktion ST_GeomFromGML ist nicht kleinlich, was die explizite Vergabe eines GML-Namensraums betrifft. Bei üblichen Anwendungen können Sie die explizite Vergabe weglassen. Wenn Sie aber das XLink Feature von GML verwenden wollen, müssen Sie den Namensraum explizit angeben.

Beispiele

A single geometry with srsName.

Code

```
SELECT ST_AsEWKT(ST_GeomFromGML($$
    <gml:LineString xmlns:gml="http://www.opengis.net/gml"
            srsName="EPSG:4269">
        <gml:coordinates>
    -71.16028,42.258729 -71.160837,42.259112 -71.161143,42.25932
        </gml:coordinates>
    </gml:LineString>
$$));
```

Output

```
SRID=4269;LINESTRING(-71.16028 42.258729,-71.160837 42.259112,-71.161143 42.25932)
```

Figure



XLink usage.

Code

```
        SELECT ST_AsEWKT(ST_GeomFromGML($$
            <gml:LineString xmlns:gml="http://www.opengis.net/gml"
                    xmlns:xlink="http://www.w3.org/1999/xlink"
                    srsName="EPSG:4269">
        <gml:pointProperty>
            <gml:Point gml:id="p1"
><gml:pos
>-71.16028 42.258729</gml:pos
></gml:Point>
        </gml:pointProperty>
        <gml:pos
>-71.160837 42.259112</gml:pos>
        <gml:pos
>-71.161143 42.258729</gml:pos>
                <gml:pointProperty>
                    <gml:Point xlink:type="simple" xlink:href="#p1"/>
                </gml:pointProperty>
            </gml:LineString>
        $$));
```

Output

```
SRID=4269;LINESTRING(-71.16028 42.258729,-71.160837 42.259112,-71.161143 42.258729,-71.16028 42.258729)
```

Figure



Polyhedral Surface.

Code

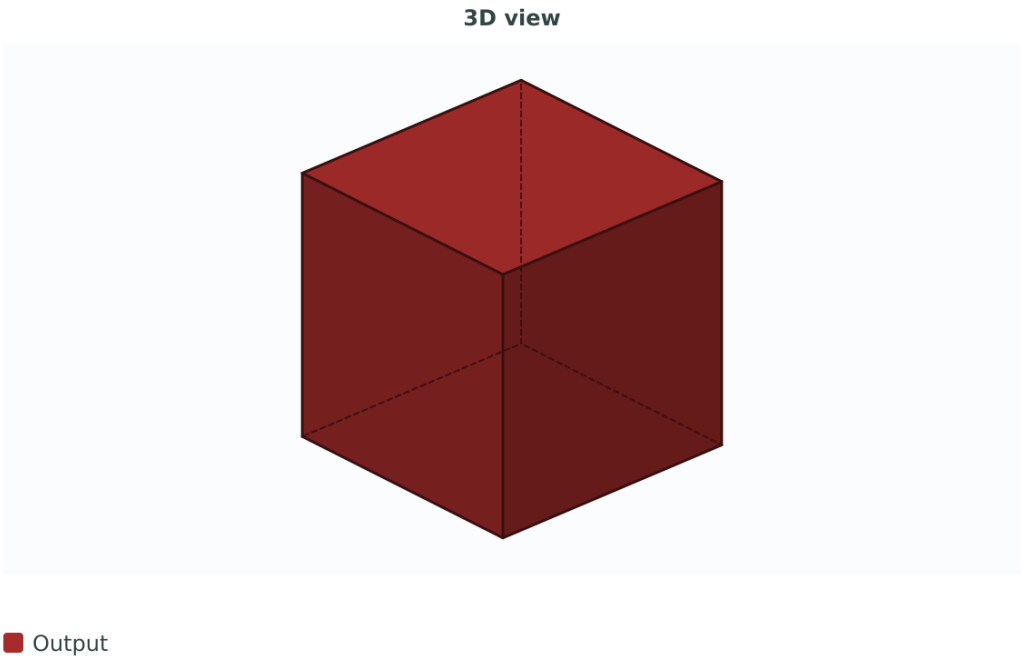
```
SELECT ST_AsEWKT(ST_GeomFromGML('
<gml:PolyhedralSurface xmlns:gml="http://www.opengis.net/gml">
<gml:polygonPatches>
  <gml:PolygonPatch>
    <gml:exterior>
      <gml:LinearRing
><gml:posList srsDimension="3"
>0 0 0 0 0 1 0 1 1 0 1 0 0 0 0</gml:posList
></gml:LinearRing>
    </gml:exterior>
  </gml:PolygonPatch>
  <gml:PolygonPatch>
    <gml:exterior>
    <gml:LinearRing
><gml:posList srsDimension="3"
>0 0 0 0 1 0 1 1 0 1 0 0 0 0 0</gml:posList
></gml:LinearRing>
    </gml:exterior>
  </gml:PolygonPatch>
  <gml:PolygonPatch>
    <gml:exterior>
    <gml:LinearRing
><gml:posList srsDimension="3"
>0 0 0 1 0 0 1 0 1 0 0 1 0 0 0</gml:posList
></gml:LinearRing>
    </gml:exterior>
  </gml:PolygonPatch>
  <gml:PolygonPatch>
    <gml:exterior>
    <gml:LinearRing
><gml:posList srsDimension="3"
>1 1 0 1 1 1 1 0 1 1 0 0 1 1 0</gml:posList
></gml:LinearRing>
```

```
    </gml:exterior>
  </gml:PolygonPatch>
  <gml:PolygonPatch>
    <gml:exterior>
    <gml:LinearRing
><gml:posList srsDimension="3"
>0 1 0 0 1 1 1 1 1 1 0 0 1 0</gml:posList
></gml:LinearRing>
    </gml:exterior>
  </gml:PolygonPatch>
  <gml:PolygonPatch>
    <gml:exterior>
    <gml:LinearRing
><gml:posList srsDimension="3"
>0 0 1 1 0 1 1 1 1 0 1 1 0 0 1</gml:posList
></gml:LinearRing>
    </gml:exterior>
  </gml:PolygonPatch>
</gml:polygonPatches>
</gml:PolyhedralSurface
>'));
```

Output

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

Figure



Siehe auch

Abschnitt 2.2.3, ST_AsGML, ST_GMLToSQL

7.8.3.4 ST_GeomFromGeoJSON

ST_GeomFromGeoJSON — Nimmt als Eingabe eine GeoJSON-Darstellung der Geometrie und gibt ein geometrisches PostGIS-Objekt aus.

Synopsis

geometry **ST_GeomFromGeoJSON**(text geomjson);
geometry **ST_GeomFromGeoJSON**(json geomjson);
geometry **ST_GeomFromGeoJSON**(jsonb geomjson);

Beschreibung

Erzeugt ein geometrisches PostGIS Objekt aus der GeoJSON Darstellung.

ST_GeomFromGeoJSON funktioniert nur bei Fragmenten von JSON-Geometrien. Auf das ganze JSON-Dokument angewendet führt zu einer Fehlermeldung.

Verbessert: 3.0.0 Geometrie wird standardmäßig auf SRID=4326 gesetzt, wenn nicht anders angegeben.

Erweiterung: 2.5.0 unterstützt nun auch die Eingabe von json und jsonb.

Verfügbarkeit: 2.0.0 benötigt - JSON-C >= 0.9



Anmerkung
Wenn Sie die JSON-C Unterstützung nicht aktiviert haben, sehen Sie eine Fehlermeldung anstatt einer Ausgabe. Um JSON-C zu aktivieren, führen Sie bitte configure --with-jsondir=/path/to/json-c aus. Für Einzelheiten siehe Abschnitt 2.2.3.

 Diese Funktion unterstützt 3d und lässt den Z-Index nicht fallen.

Beispiele

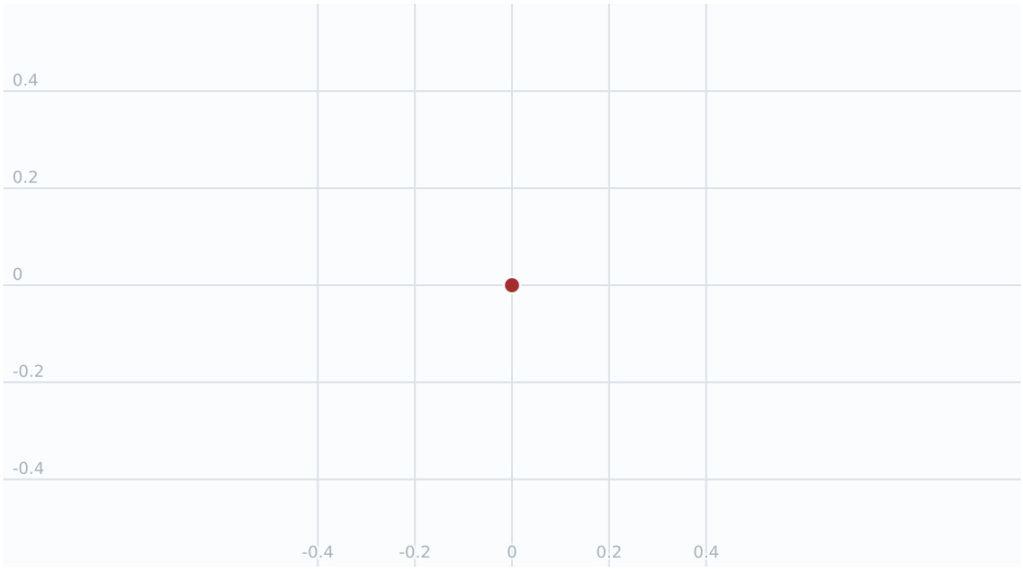
Code

```
SELECT ST_GeomFromGeoJSON('{"type":"Point","coordinates":[-48.23456,20.12345]}') AS wkt;
```

Output

```
POINT(-48.23456 20.12345)
```

Figure



This example creates a 3D LineString.

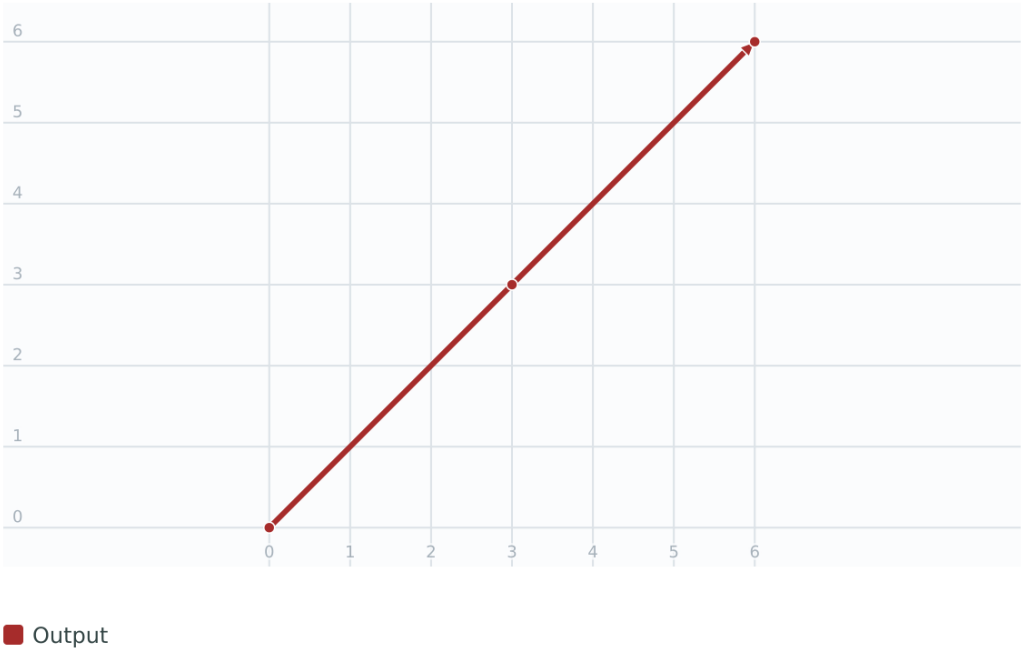
Code

```
SELECT ST_GeomFromGeoJSON('{"type":"LineString","coordinates":[[1,2,3],[4,5,6],[7,8,9]]}') ↩
    AS wkt;
```

Output

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

Figure



Siehe auch

ST_AsText, ST_AsGeoJSON, Abschnitt 2.2.3

7.8.3.5 ST_GeomFromKML

ST_GeomFromKML — Nimmt als Eingabe eine KML-Darstellung der Geometrie und gibt ein geometrisches PostGIS-Objekt aus.

Synopsis

geometry **ST_GeomFromKML**(text geomkml);

Beschreibung

Constructs a PostGIS geometry from an OGC KML representation.

T_GeomFromKML funktioniert nur bei Fragmenten von KML-Geometrien. Auf das ganze KML-Dokument angewendet führt zu einer Fehlermeldung.

Unterstützte OGC KML Versionen:

- KML 2.2.0 Namespace

OGC KML Standards, vgl.: http://www.opengeospatial.org/standards/kml:

Verfügbarkeit: 1.5, benötigt libxml2 2.6+

Diese Funktion unterstützt 3d und lässt den Z-Index nicht fallen.

Anmerkung
ST_GeomFromKML does not support SQL/MM curve geometries.

Beispiele

A single LineString geometry.

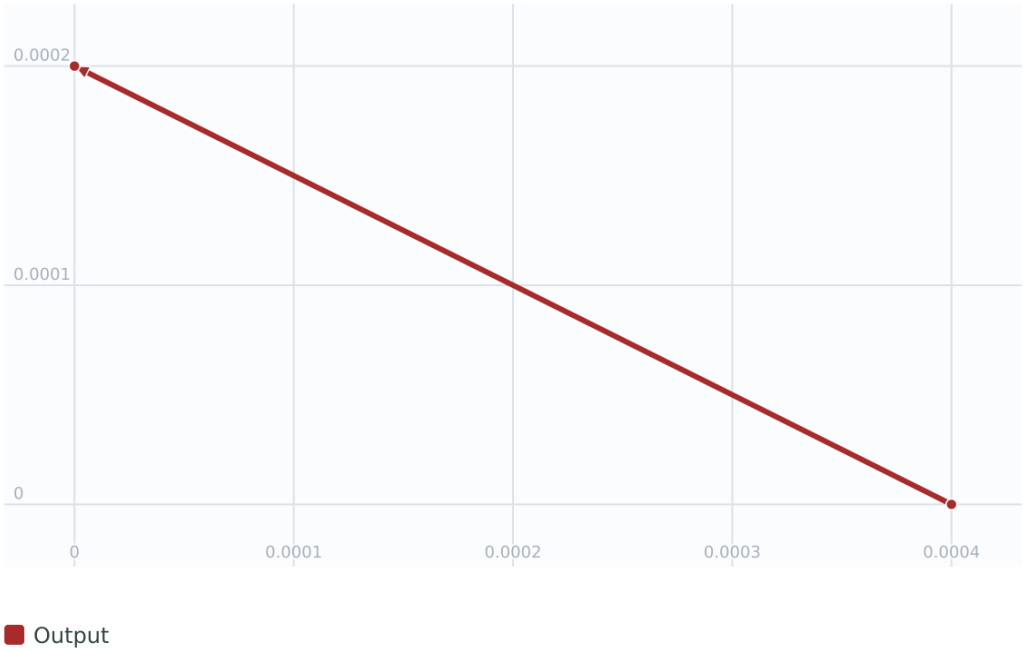
Code

```
        SELECT ST_AsEWKT(ST_GeomFromKML($$
            <LineString>
                <coordinates
>-71.1663, 42.2614
                    -71.1667, 42.2616</coordinates>
            </LineString>
        $$));
```

Output

```
SRID=4326;LINESTRING(-71.1663 42.2614,-71.1667 42.2616)
```

Figure



Siehe auch

Abschnitt 2.2.3, ST_AsKML

7.8.3.6 ST_GeomFromTWKB

ST_GeomFromTWKB — Erzeugt eine Geometrie aus einer TWKB ("Tiny Well-Known Binary") Darstellung.

Synopsis

geometry **ST_GeomFromTWKB**(bytea twkb);

Beschreibung

Die Funktion `ST_GeomFromTWKB` nimmt eine TWKB ("Tiny Well-Known Binary") Darstellung und erzeugt ein Objekt mit dem entsprechenden geometrischen Datentyp.

Beispiele

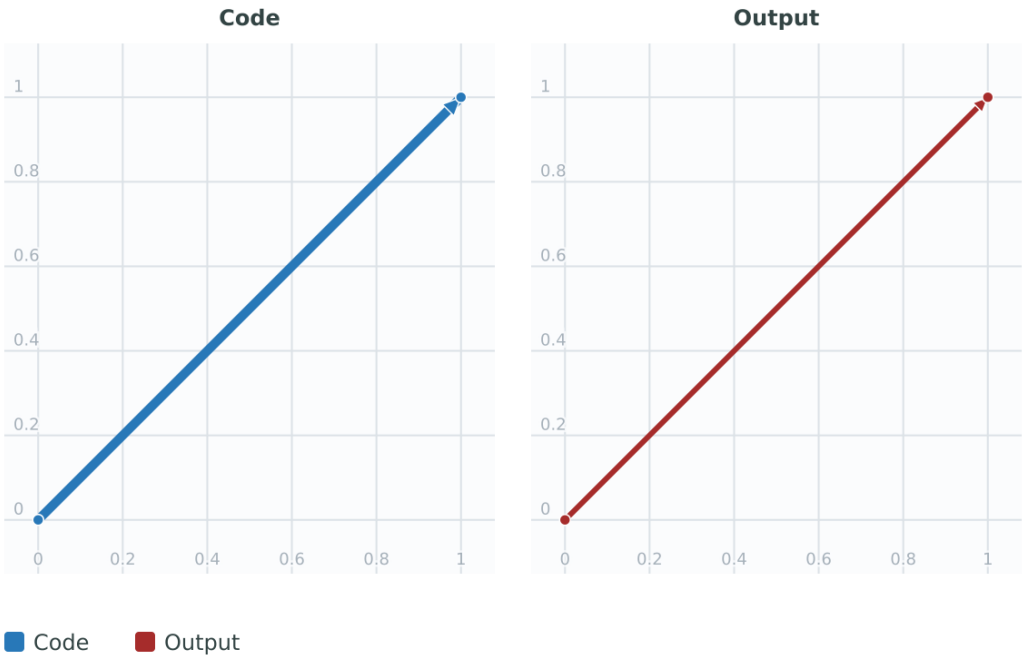
Code

```
SELECT ST_GeomFromTWKB(ST_AsTWKB('LINESTRING(126 34,127 35)'::geometry));
```

Output

```
LINESTRING(126 34, 127 35)
```

Figure



Code

```
SELECT ST_GeomFromTWKB(E'\\x620002f7f40dbce4040105');
```

Output

```
LINESTRING(-113.98 39.198,-113.981 39.195)
```

Figure



Siehe auch

ST_AsTWKB

7.8.3.7 ST_GMLToSQL

ST_GMLToSQL — Returns a geometry from GML. Alias for ST_GeomFromGML.

Synopsis

geometry **ST_GMLToSQL**(text geomgml);
geometry **ST_GMLToSQL**(text geomgml, integer srid);

Beschreibung

This function is an alias for ST_GeomFromGML and provides the same signatures.

 Diese Methode setzt die SQL/MM-Spezifikation um. SQL-MM 3: 5.1.50.

Verfügbarkeit: 1.5, benötigt libxml2 1.6+

Erweiterung: Mit 2.0.0 wurde die Unterstützung für polyedrische Oberflächen und TIN eingeführt.

Erweiterung: 2.0.0 Standardwert für den optionalen Parameter SRID eingefügt.

Enhanced: 3.7.0 support for GML ArcString, curved Ring, and CurvePolygon geometries was introduced.

Siehe auch

Abschnitt 2.2.3, ST_GeomFromGML, ST_AsGML

7.8.3.8 ST_LineFromEncodedPolyline

ST_LineFromEncodedPolyline — Erzeugt einen LineString aus einem codierten Linienzug.

Synopsis

geometry **ST_LineFromEncodedPolyline**(text polyline, integer precision=5);

Beschreibung

Erzeugt einen LineString aus einem codierten Linienzug.

Der optionale Parameter `precision` gibt an wieviele Dezimalstellen der kodierten Polylinie erhalten bleiben. Dieser Wert sollte beim Dekodieren und beim Kodieren ident sein, sonst entstehen inkorrekte Koordinaten.

Siehe http://developers.google.com/maps/documentation/utilities/polylinealgorithm

Verfügbarkeit: 2.2.0

Beispiele

This example creates a LineString from a polyline.

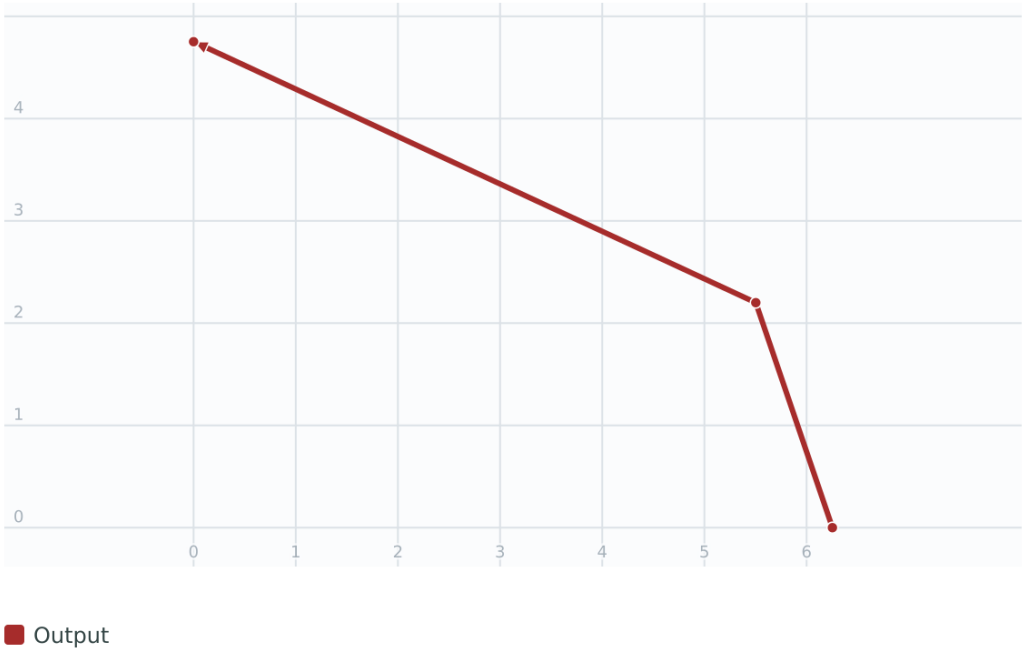
Code

```
SELECT ST_LineFromEncodedPolyline('_p~iF~ps|U_ulLnnqC_mqNvxq`@');
```

Output

```
SRID=4326;LINESTRING(-120.2 38.5,-120.95 40.7,-126.453 43.252)
```

Figure



This example selects a different precision than the one used for polyline encoding.

Code

```
SELECT ST_LineFromEncodedPolyline('_p~iF~ps|U_ulLnnqC_mqNvxq`@', 6);
```

Output

```
SRID=4326;LINESTRING(-12.02 3.85,-12.095 4.07,-12.6453 4.3252)
```

Figure



Siehe auch

ST_AsEncodedPolyline

7.8.3.9 ST_PointFromGeoHash

ST_PointFromGeoHash — Gibt einen Punkt von einer GeoHash Zeichenfolge zurück.

Synopsis

point **ST_PointFromGeoHash**(text geohash, integer precision=full_precision_of_geohash);

Beschreibung

Gibt die Geometrie einer GeoHash Zeichenfolge zurück. Der Punkt entspricht dem Mittelpunkt des GeoHas.

Wenn keine `precision` angegeben wird, dann gibt ST_PointFromGeoHash einen Punkt zurück, der auf der vollständigen Genauigkeit der gegebenen GeoHash Zeichenfolge beruht.

Wenn `precision` angegeben wird, verwendet ST_PointFromGeoHash entsprechend viele Zeichen des GeoHash, um den Punkt zu erzeugen.

Verfügbarkeit: 2.1.0

Beispiele

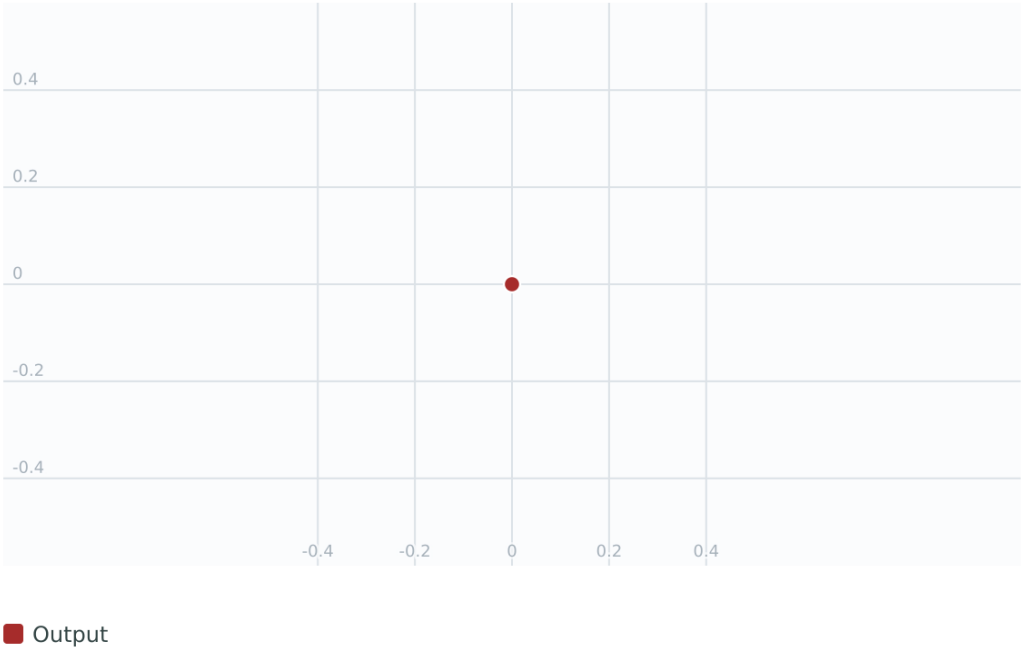
Code

```
SELECT ST_PointFromGeoHash('9qqj7nmxncgyy4d0dbxqz0');
```

Output

```
POINT(-115.172816 36.114646)
```

Figure



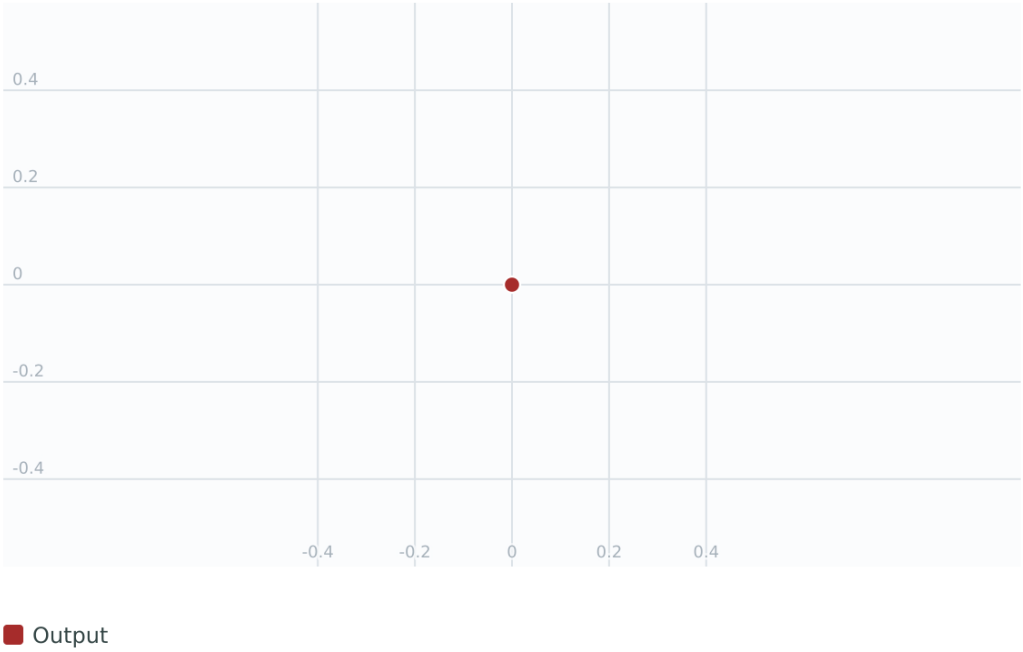
Code

```
SELECT ST_PointFromGeoHash('9qqj7nmxncgyy4d0dbxqz0', 4);
```

Output

```
POINT(-115.13671875 36.123046875)
```

Figure



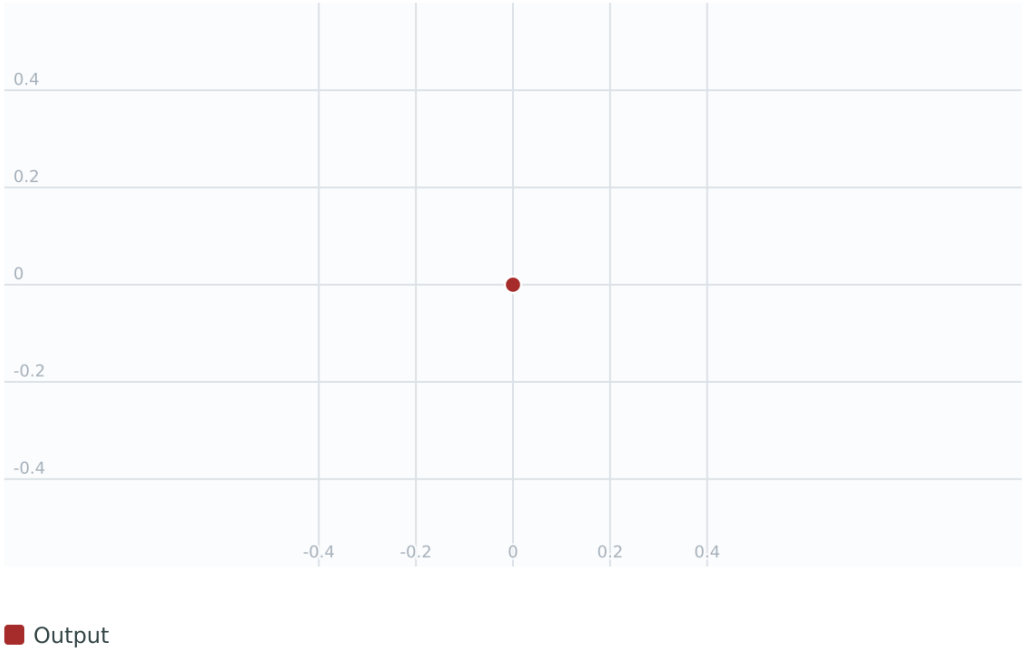
Code

```
SELECT ST_PointFromGeoHash('9qqj7nmxncgyy4d0dbxqz0', 10);
```

Output

```
POINT(-115.172815918922 36.1146435141563)
```

Figure



Siehe auch

ST_GeoHash, ST_Box2dFromGeoHash, ST_GeomFromGeoHash

7.8.3.10 ST_FromFlatGeobufToTable

ST_FromFlatGeobufToTable — Erstellt eine Tabelle auf der Grundlage der Struktur der FlatGeobuf-Daten.

Synopsis

void **ST_FromFlatGeobufToTable**(text schemaname, text tablename, bytea FlatGeobuf input data);

Beschreibung

Erzeugt eine Tabelle auf der Grundlage der Struktur der FlatGeobuf-Daten. (http://flatgeobuf.org).

`schema` Name des Schemas.

`Tabelle` Tabellenname.

`data` Eingabe von FlatGeobuf-Daten.

Verfügbarkeit: 3.2.0

7.8.3.11 ST_FromFlatGeobuf

ST_FromFlatGeobuf — Liest FlatGeobuf-Daten.

Synopsis

setof anyelement **ST_FromFlatGeobuf**(anyelement Table reference, bytea FlatGeobuf input data);

Beschreibung

Liest FlatGeobuf-Daten (http://flatgeobuf.org). HINWEIS: PostgreSQL bytea kann 1GB nicht überschreiten.

`tabletype` Verweis auf einen Tabellentyp.

`data` Eingabe FlatGeobuf-Daten.

Verfügbarkeit: 3.2.0

7.9 Geometrieausgabe

7.9.1 Well-known-Text (WKT) Repräsentation

7.9.1.1 ST_AsEWKT

ST_AsEWKT — Gibt die Well-known-Text(WKT) Darstellung der Geometrie mit den SRID-Metadaten zurück.

Synopsis

text **ST_AsEWKT**(geometry g1);
text **ST_AsEWKT**(geometry g1, integer maxdecimaldigits=15);
text **ST_AsEWKT**(geography g1);
text **ST_AsEWKT**(geography g1, integer maxdecimaldigits=15);

Beschreibung

Gibt die Well-Known-Text-Darstellung der Geometrie mit dem Präfix SRID zurück. Das optionale Argument *`maxdecimaldigits`* kann verwendet werden, um die maximale Anzahl der in der Ausgabe verwendeten Dezimalstellen nach der Fließkommazahl zu verringern (Standardwert: 15).

Um die inverse Konvertierung der EWKT-Darstellung in eine PostGIS-Geometrie durchzuführen, verwenden Sie ST_GeomFromEWKT.



Warnung
Die Verwendung des Parameters *`maxdecimaldigits`* kann dazu führen, dass die Ausgabegeometrie ungültig wird. Um dies zu vermeiden, verwenden Sie zuerst ST_ReducePrecision mit einer geeigneten Rastergröße.



Anmerkung
Die WKT-Spezifikation enthält keine SRID. Um das OGC-WKT-Format zu erhalten, verwenden Sie ST_AsText.

Warnung
Im WKT-Format wird die Genauigkeit nicht beibehalten. Verwenden Sie daher für den Transport das Format ST_AsBinary oder ST_AsEWKB, um ein fließendes Abschneiden zu verhindern.

Verbessert: 3.1.0 Unterstützung für optionale Präzisionsparameter.

Erweiterung: Mit 2.0.0 wurde die Unterstützung für den geographischen Datentyp, polyedrische Oberflächen, Dreiecke und TIN eingeführt.

Diese Funktion unterstützt 3d und lässt den Z-Index nicht fallen.

Diese Methode unterstützt kreisförmige Strings und Kurven.

Diese Funktion unterstützt polyedrische Flächen.

Diese Funktion unterstützt Dreiecke und dreieckige unregelmäßige Netzoberflächen (TIN).

Beispiele

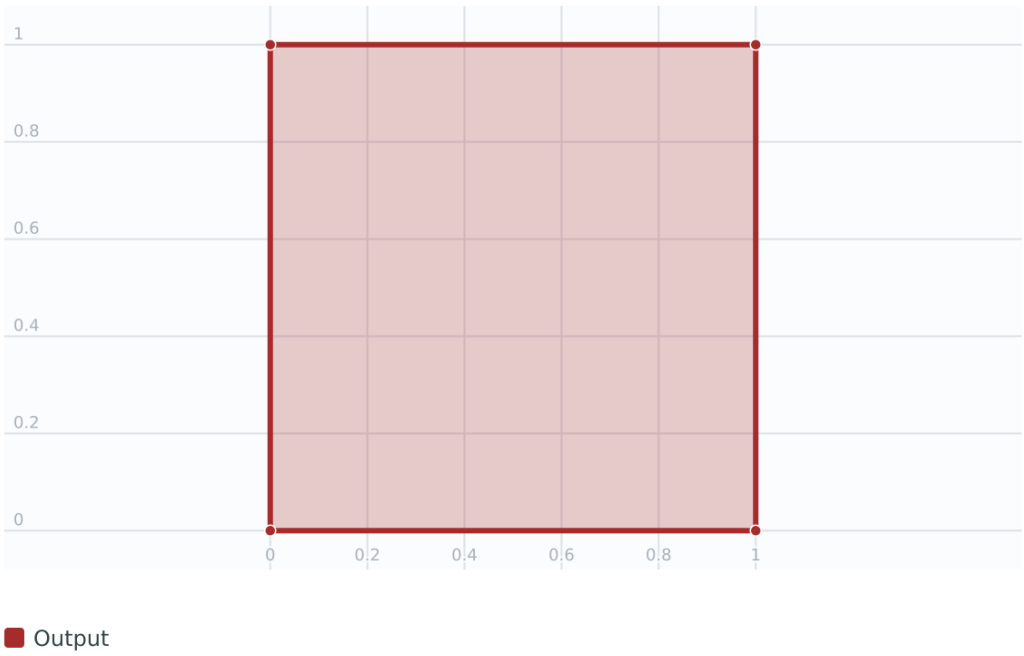
Code

```
SELECT ST_AsEWKT('0103000020 ↩
    e610000001000000050000000000000000000000000000000000000000000000000000000000000000000000000000000000f03f000000000
    '::geometry);
```

Output

```
SRID=4326;POLYGON((0 0,0 1,1 1,1 0,0 0))
```

Figure



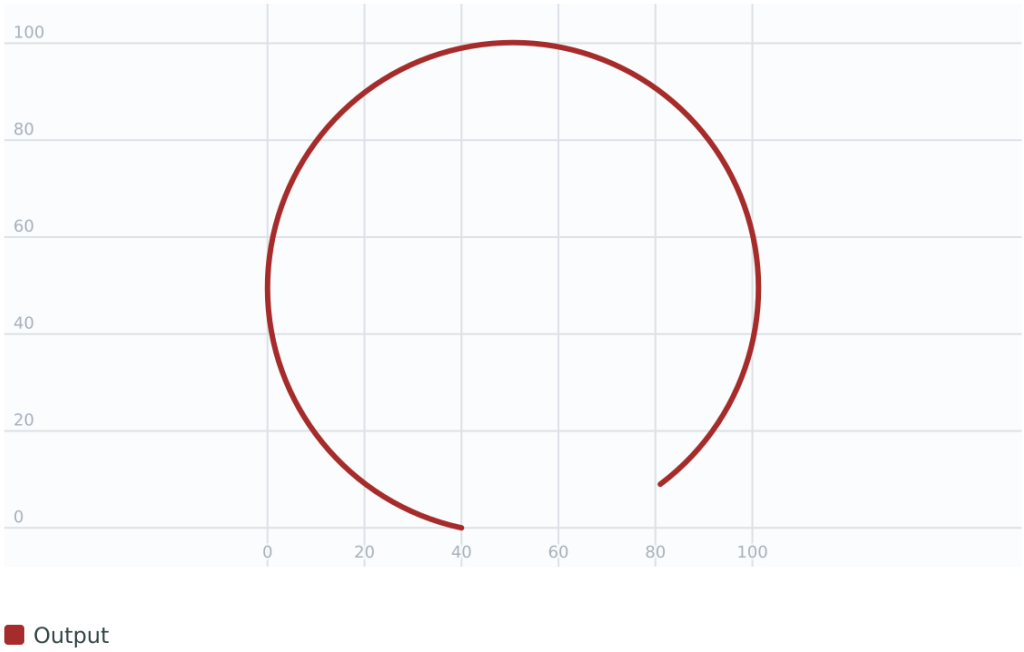
Code

```
SELECT ST_AsEWKT('0108000080030000000000000060 ↩
    E30A4100000000785C0241000000000000F03F0000000018E20A4100000000485F024100000000000000004000000000018
    ');
```

Output

```
CIRCULARSTRING(220268 150415 1,220227 150505 2,220227 150406 3)
```

Figure



Siehe auch

ST_AsBinary, ST_AsEWKB, ST_AsText, ST_GeomFromEWKT

7.9.1.2 ST_AsText

ST_AsText — Gibt die Well-known-Text(WKT) Darstellung der Geometrie/Geographie ohne die SRID Metadaten zurück.

Synopsis

text **ST_AsText**(geometry g1);
text **ST_AsText**(geometry g1, integer maxdecimaldigits = 15);
text **ST_AsText**(geography g1);
text **ST_AsText**(geography g1, integer maxdecimaldigits = 15);

Beschreibung

Liefert die OGC Well-Known Text (WKT) Darstellung der Geometrie/Geographie. Das optionale Argument *maxdecimaldigits* kann verwendet werden, um die Anzahl der Nachkommastellen in der Ausgabe von Ordinaten zu begrenzen (Standardwert: 15).

Um die umgekehrte Konvertierung der WKT-Darstellung in PostGIS-Geometrie durchzuführen, verwenden Sie ST_GeomFromText.



Anmerkung
Die standardmäßige OGC-WKT-Darstellung enthält den SRID nicht. Um den SRID als Teil der Ausgabedarstellung aufzunehmen, verwenden Sie die nicht standardisierte PostGIS-Funktion ST_AsEWKT



Warnung
Bei der textuellen Darstellung von Zahlen in WKT wird möglicherweise nicht die volle Gleitkommagenauigkeit eingehalten. Um die volle Genauigkeit bei der Datenspeicherung oder beim Datentransport zu gewährleisten, ist es am besten, das Format Well-Known Binary (WKB) zu verwenden (siehe ST_AsBinary und *maxdecimaldigits*).



Warnung
Die Verwendung des Parameters *maxdecimaldigits* kann dazu führen, dass die Ausgabegeometrie ungültig wird. Um dies zu vermeiden, verwenden Sie zuerst ST_ReducePrecision mit einer geeigneten Rastergröße.

Verfügbarkeit: 1.5 - Unterstützung von geograpischen Koordinaten.

Erweiterung: 2.5 - der optionale Parameter "precision" wurde eingeführt.


Diese Methode implementiert die OGC Simple Features Implementation Specification for SQL 1.1. s2.1.1.1


Diese Methode setzt die SQL/MM-Spezifikation um. SQL-MM 3: 5.1.25


Diese Methode unterstützt kreisförmige Strings und Kurven.

Beispiele

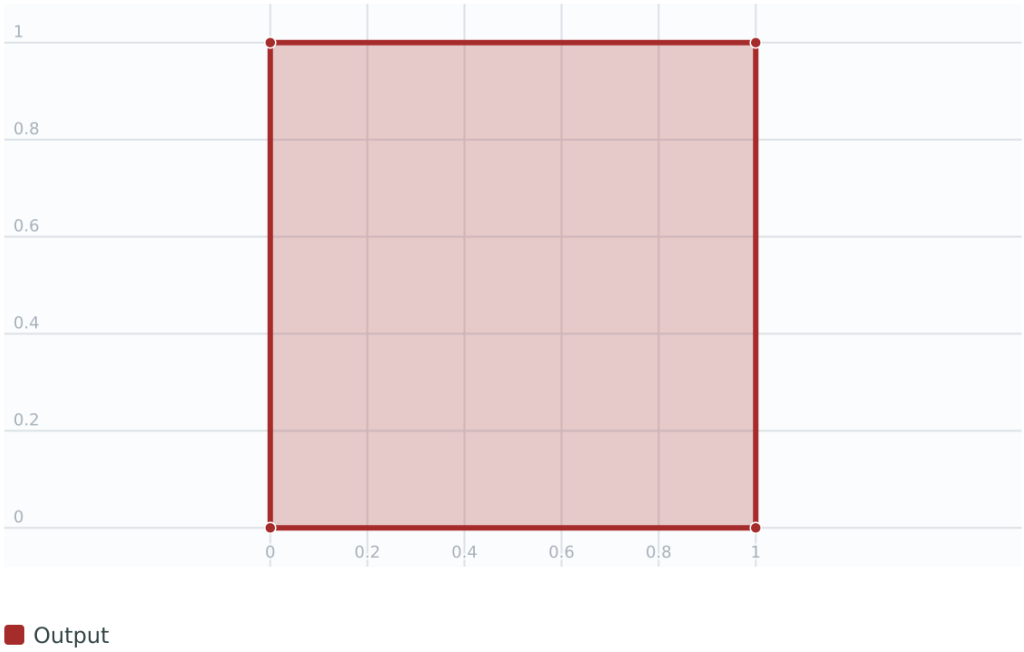
Code

```
SELECT ST_AsText(' ↩
    01030000000100000005000000000000000000000000000000000000000000000000000000000000000000000000000000 ↩
    f03f000000000000f03f000000000000f03f000000000000f03f00000000000000000000000000000000000000000000000
    ');
```

Output

```
POLYGON((0 0,0 1,1 1,1 0,0 0))
```

Figure



Die Standardeinstellung ist die Ausgabe mit voller Präzision.

Code

```
SELECT ST_AsText('POINT(111.1111111 1.1111111)'::geometry);
```

Output

```
POINT(111.1111111 1.1111111)
```

Das Argument *maxdecimaldigits* kann verwendet werden, um die Ausgabegenauigkeit zu begrenzen.

Code

```
SELECT ST_AsText('POINT(111.1111111 1.1111111)'::geometry, 2);
```

Output

```
POINT(111.11 1.11)
```

Siehe auch

ST_AsBinary, ST_AsEWKB, ST_AsEWKT, ST_GeomFromText

7.9.2 Well-known-Binary (WKB) Repräsentation

7.9.2.1 ST_AsBinary

ST_AsBinary — Rückgabe der OGC/ISO Well-Known Binary (WKB)-Darstellung der Geometrie/Geografie ohne SRID-Metadaten.

Synopsis

bytea **ST_AsBinary**(geometry g1);
bytea **ST_AsBinary**(geometry g1, text NDR_or_XDR);
bytea **ST_AsBinary**(geography g1);
bytea **ST_AsBinary**(geography g1, text NDR_or_XDR);

Beschreibung

Returns the OGC/ISO Well-Known Binary (WKB) representation of the geometry. The first function variant defaults to encoding using server machine endian. The second function variant takes a text argument specifying the endian encoding: either 'NDR' for little-endian; or 'XDR' for big-endian. Supplying unknown arguments will result in little-endian output.

Das WKB-Format ist nützlich, um Geometriedaten aus der Datenbank zu lesen und dabei die volle numerische Präzision beizubehalten. Dadurch wird die Präzisionsrundung vermieden, die bei Textformaten wie WKT auftreten kann.

Um die inverse Konvertierung von WKB in PostGIS-Geometrie durchzuführen, verwenden Sie ST_GeomFromWKB.



Anmerkung
Das OGC/ISO WKB-Format enthält keinen SRID. Um das EWKB-Format zu erhalten, das den SRID enthält, verwenden Sie ST_AsEWKB



Anmerkung
PostgreSQL outputs `bytea` values in hex encoding by default. If your GUI tools require the old behavior, then SET `bytea_output='escape'` in your database.

Erweiterung: Mit 2.0.0 wurde die Unterstützung für polyedrische Oberflächen, Dreiecke und TIN eingeführt.

Erweiterung: 2.0.0 - Unterstützung für höherdimensionale Koordinatensysteme eingeführt.

Erweiterung: 2.0.0 Unterstützung zum Festlegen des Endian beim geographischen Datentyp eingeführt.

Verfügbarkeit: 1.5.0 Unterstützung von geograpischen Koordinaten.

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`.

Diese Methode implementiert die OGC Simple Features Implementation Specification for SQL 1.1. s2.1.1.1

Diese Methode setzt die SQL/MM-Spezifikation um. SQL-MM 3: 5.1.37

Diese Methode unterstützt kreisförmige Strings und Kurven.

Diese Funktion unterstützt polyedrische Flächen.

Diese Funktion unterstützt Dreiecke und dreieckige unregelmäßige Netzoberflächen (TIN).

Diese Funktion unterstützt 3d und lässt den Z-Index nicht fallen.

Beispiele

Code

```
SELECT ST_AsBinary(ST_GeomFromText('POLYGON((0 0,0 1,1 1,1 0,0 0))', 4326));
```

Output

```
\x0103000000010000000500000000000000000000000000000000000000000000000000000000000000000000000000000000
000000f03f000000000000f03f000000000000f03f000000000000f03f000000000000000000000000
00000000000000000000000000
```

Code

```
SELECT ST_AsBinary(ST_GeomFromText('POLYGON((0 0,0 1,1 1,1 0,0 0))', 4326), 'XDR');
```

Output

```
\x0000000003000000010000000500000000000000000000000000000000000000000000000000000000000000000003ff000
00000000003ff0000000000000003ff0000000000003ff000000000000000000000000000000000000
00000000000000000000000000
```

Use the PostgreSQL `encode` function to produce a hexadecimal WKB string without the `\x` prefix used for bytea display. Specifying `NDR` makes the byte order deterministic.

Code

```
SELECT encode(
    ST_AsBinary('POINT(15.123 21.456)'::geometry, 'NDR'),
    'hex'
) AS wkb_hex;
```

Output

```
0101000000e5d022dbf93e2e40dbf97e6abc743540
```

Siehe auch

ST_GeomFromWKB, ST_AsEWKB, ST_AsTWKB, ST_AsText,

7.9.2.2 ST_AsEWKB

ST_AsEWKB — Rückgabe der Extended Well-Known Binary (EWKB) Darstellung der Geometrie mit SRID-Metadaten.

Synopsis

bytea **ST_AsEWKB**(geometry g1);
bytea **ST_AsEWKB**(geometry g1, text NDR_or_XDR);

Beschreibung

Returns the Extended Well-Known Binary (EWKB) representation of the geometry with SRID metadata. The first function variant defaults to encoding using server machine endian. The second function variant takes a text argument specifying the endian encoding: either 'NDR' for little-endian; or 'XDR' for big-endian. Supplying unknown arguments will result in little-endian output.

Das WKB-Format ist nützlich, um Geometriedaten aus der Datenbank zu lesen und dabei die volle numerische Präzision beizubehalten. Dadurch wird die Präzisionsrundung vermieden, die bei Textformaten wie WKT auftreten kann.

Um die inverse Konvertierung von EWKB in PostGIS-Geometrie durchzuführen, verwenden Sie ST_GeomFromEWKB.



Anmerkung
Um das OGC/ISO WKB-Format zu erhalten, verwenden Sie ST_AsBinary. Beachten Sie, dass das OGC/ISO WKB-Format nicht den SRID enthält.

Erweiterung: Mit 2.0.0 wurde die Unterstützung für polyedrische Oberflächen, Dreiecke und TIN eingeführt.

 Diese Funktion unterstützt 3d und lässt den Z-Index nicht fallen.

 Diese Methode unterstützt kreisförmige Strings und Kurven.

 Diese Funktion unterstützt polyedrische Flächen.

 Diese Funktion unterstützt Dreiecke und dreieckige unregelmäßige Netzoberflächen (TIN).

Beispiele

Code

```
SELECT ST_AsEWKB(ST_GeomFromText('POLYGON((0 0,0 1,1 1,1 0,0 0))', 4326));
```

Output

```
\x0103000020e6100000010000000500000000000000000000000000000000000000000000000000000000000000000000
0000000000000f03f000000000000f03f000000000000f03f000000000000f03f000000000000000000000000000000000
0000000000000000000000000000000000
```

Code

```
SELECT ST_AsEWKB(ST_GeomFromText('POLYGON((0 0,0 1,1 1,1 0,0 0))', 4326), 'XDR');
```

Output

```
\x00200000030000010e60000000100000005000000000000000000000000000000000000000000000000000000000000
003ff0000000000003ff0000000000003ff0000000000003ff0000000000000000000000000000000000
0000000000000000000000000000000000
```

Siehe auch

ST_AsBinary, ST_GeomFromEWKB, ST_SRID

7.9.2.3 ST_AsHEXEWKB

ST_AsHEXEWKB — Gibt eine Geometrie im HEXEWKB Format (als Text) aus; verwendet entweder die Little-Endian (NDR) oder die Big-Endian (XDR) Zeichenkodierung.

Synopsis

text **ST_AsHEXEWKB**(geometry g1, text NDRorXDR);
text **ST_AsHEXEWKB**(geometry g1);

Beschreibung

Gibt eine Geometrie im HEXEWKB Format (als Text) aus; verwendet entweder die Little-Endian (NDR) oder die Big-Endian (XDR) Zeichenkodierung. Wenn keine Zeichenkodierung angegeben wurde, wird NDR verwendet.

Verfügbarkeit: 1.2.2

Diese Funktion unterstützt 3d und lässt den Z-Index nicht fallen.

Diese Methode unterstützt kreisförmige Strings und Kurven.

Beispiele

Code

```
SELECT ST_AsHEXEWKB(ST_GeomFromText('POLYGON((0 0,0 1,1 1,1 0,0 0))', 4326));
```

Output

```
0103000020E6100000010000000500
0000000000000000000000000000000000000000
0000000000000000000000000000000000F03F
000000000000F03F000000000000F03F000000000000F03
F0000000000000000000000000000000000000000000000000000000
```

Which gives the same answer as:

Code

```
SELECT ST_GeomFromText('POLYGON((0 0,0 1,1 1,1 0,0 0))', 4326)::text;
```

7.9.3 Weitere Formate

7.9.3.1 ST_AsEncodedPolyline

ST_AsEncodedPolyline — Erzeugt eine codierte Polylinie aus einer LineString Geometrie.

Synopsis

text **ST_AsEncodedPolyline**(geometry geom, integer precision=5);

Beschreibung

Returns the geometry in the Encoded Polyline Algorithm Format. The format stores latitude and longitude deltas as integers after multiplying coordinates by 10 raised to the requested `precision`.

The informal names *polyline5* and *polyline6* refer to the same encoding with precision 5 and 6. The precision must match when encoding and decoding; otherwise the coordinates are scaled incorrectly.

The input geometry must use SRID 4326, with longitude as X and latitude as Y. Use ST_Transform to transform geometries in other coordinate systems before encoding.

The OSRM API returns precision 5 for `geometries=polyline` and precision 6 for `geometries=polyline6`. Decode the resulting `route.geometry` with ST_LineFromEncodedPolyline using the matching precision.

Verfügbarkeit: 2.2.0

Beispiele

Encode the same WGS 84 LineString using precision 5 and 6.

Code

```
WITH route AS (
  SELECT ST_GeomFromEWKT(
    'SRID=4326;LINESTRING(-120.2 38.5,-120.95 40.7,-126.453 43.252)'
  ) AS geom
)
SELECT ST_AsEncodedPolyline(geom, 5) AS polyline5,
       ST_AsEncodedPolyline(geom, 6) AS polyline6
FROM route;
```

Output

```
polyline5          |           polyline6
---------------------------+----------------------------------
 _p~iF~ps|U_ulLnnqC_mqNvxq`@ | _izlhA~rlgdF_{geC~ywl@_kwzCn`{nI
```

Decode an OSRM `route.geometry` returned with `geometries=polyline6`.

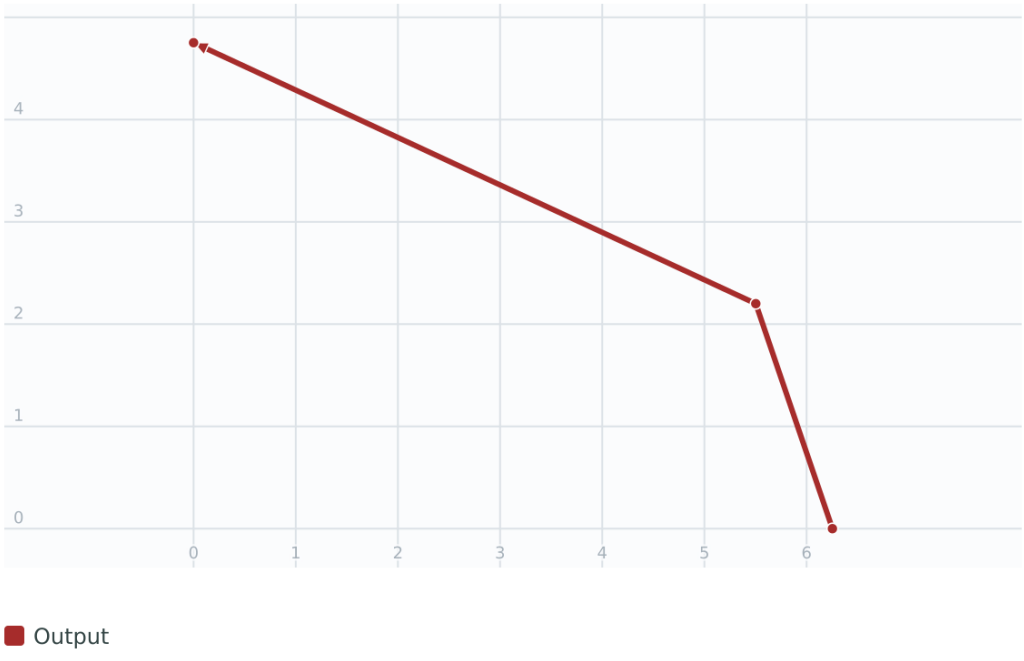
Code

```
SELECT ST_AsEWKT(ST_LineFromEncodedPolyline(
  '_izlhA~rlgdF_{geC~ywl@_kwzCn`{nI',
  6
)) AS route;
```

Output

```
SRID=4326;LINESTRING(-120.2 38.5,-120.95 40.7,-126.453 43.252)
```

Figure



The Google Maps JavaScript Geometry library decodes precision-5 strings. This browser example assumes that `map` is an initialized Google map.

Code

```
<script src="https://maps.googleapis.com/maps/api/js?key=YOUR_API_KEY&libraries=geometry"
></script>
<script>
  const encodedPath = "_p~iF~ps|U_ulLnnqC_mqNvxq`@";
  const flightPath = new google.maps.Polyline({
    path: google.maps.geometry.encoding.decodePath(encodedPath),
    map: map,
    strokeColor: "#0000CC",
    strokeOpacity: 1.0,
    strokeWeight: 4
  });
</script>
```

Siehe auch

ST_LineFromEncodedPolyline, ST_Transform

7.9.3.2 ST_AsFlatGeobuf

ST_AsFlatGeobuf — Rückgabe einer FlatGeobuf-Darstellung einer Reihe von Zeilen.

Synopsis

bytea **ST_AsFlatGeobuf**(anyelement set row);
bytea **ST_AsFlatGeobuf**(anyelement row, bool index);
bytea **ST_AsFlatGeobuf**(anyelement row, bool index, text geom_name);

Beschreibung

Gibt eine FlatGeobuf-Darstellung (http://flatgeobuf.org) einer Reihe von Zeilen zurück, die einer FeatureCollection entsprechen. HINWEIS: PostgreSQL bytea kann 1GB nicht überschreiten.

`row` Datenzeilen mit zumindest einer Geometriespalte.

`index` schaltet die Erstellung von räumlichen Indizes ein. Die Voreinstellung ist false.

`geom_name` ist die Bezeichnung der Geometriespalte in den Datenzeilen. Wenn NULL, dann wird standardmäßig die erste aufgefundene Geometriespalte verwendet.

Verfügbarkeit: 3.2.0

7.9.3.3 ST_AsGeobuf

ST_AsGeobuf — Gibt eine Menge an Zeilen in der Geobuf Darstellung aus.

Synopsis

bytea **ST_AsGeobuf**(anyelement set row);
bytea **ST_AsGeobuf**(anyelement row, text geom_name);

Beschreibung

Gibt Zeilen einer FeatureCollection in der Geobuf Darstellung (https://github.com/mapbox/geobuf) aus. Von jeder Eingabegeometrie wird die maximale Genauigkeit analysiert, um eine optimale Speicherung zu erreichen. Anmerkung: In der jetzigen Form kann Geobuf nicht "gestreamt" werden, wodurch die gesamte Ausgabe im Arbeitsspeicher zusammengestellt wird.

`row` Datenzeilen mit zumindest einer Geometriespalte.

`geom_name` ist die Bezeichnung der Geometriespalte in den Datenzeilen. Wenn NULL, dann wird standardmäßig die erste aufgefundene Geometriespalte verwendet.

Verfügbarkeit: 2.4.0

Beispiele

Code

```
SELECT encode(ST_AsGeobuf(q, 'geom'), 'base64')
FROM (SELECT 'POLYGON((0 0,0 1,1 1,1 0,0 0))'::geometry AS geom) AS q;
```

Output

```
GAAiEAoOCgwIBBoIAAAAAgIAAAE=
```

7.9.3.4 ST_AsGeoJSON

ST_AsGeoJSON — Rückgabe einer Geometrie oder eines Merkmals im 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);

Beschreibung

Gibt eine Geometrie als GeoJSON "geometry"-Objekt oder eine Zeile als GeoJSON "feature"-Objekt zurück.

Die resultierenden GeoJSON-Geometrie- und Merkmalsdarstellungen entsprechen den GeoJSON-Spezifikationen RFC 7946, außer wenn die geparsten Geometrien mit einem anderen CRS als WGS84-Längen- und Breitengrad referenziert sind (EPSG:4326, urn:ogc:def:crs:OGC::CRS84); dem GeoJSON-Geometrieobjekt wird dann standardmäßig ein kurzer CRS-SRID-Bezeichner beigefügt. Es werden sowohl 2D- als auch 3D-Geometrien unterstützt. GeoJSON unterstützt nur SFS 1.1 Geometrietypen (z.B. keine Unterstützung für Kurven).

Der Parameter `geom_column` wird verwendet, um zwischen mehreren Geometriespalten zu unterscheiden. Wird er weggelassen, wird die erste Geometriespalte im Datensatz ermittelt. Umgekehrt erspart die Übergabe des Parameters die Suche nach dem Spaltentyp.

Der Parameter `maxdecimaldigits` kann zur Reduzierung der Nachkommastellen in der Ausgabe verwendet werden (standardmäßig 9). Wenn EPSG:4326 verwendet wird, kann `maxdecimaldigits=6` eine gute Wahl für viele Karten bei der Bildschirmausgabe sein.



Warnung
Die Verwendung des Parameters *`maxdecimaldigits`* kann dazu führen, dass die Ausgabegeometrie ungültig wird. Um dies zu vermeiden, verwenden Sie zuerst ST_ReducePrecision mit einer geeigneten Rastergröße.

Das Argument `options` kann verwendet werden, um BBOX oder CRS in der GeoJSON-Ausgabe hinzuzufügen:

- 0: keine option
- 1: GeoJSON BBOX
- 2: GeoJSON CRS-Kurzform (z.B. EPSG:4326)
- 4: GeoJSON CRS-Langform (z.B. urn:ogc:def:crs:EPSG::4326)
- 8: GeoJSON CRS-Kurzform, außer bei EPSG:4326 (default)

Der Parameter `id_column` wird verwendet, um das Element "id" der zurückgegebenen GeoJSON-Features festzulegen. Gemäß GeoJSON RFC SOLLTE dies immer dann verwendet werden, wenn ein Feature einen häufig verwendeten Bezeichner hat, wie z.B. einen Primärschlüssel. Wenn dies nicht angegeben wird, erhalten die erzeugten Features kein "id"-Mitglied und alle anderen Spalten als die Geometrie, einschließlich potenzieller Schlüssel, landen einfach im "properties"-Mitglied des Features.

When the input record includes duplicate column names, the resulting "properties" object will contain repeated keys. `ST_AsGeoJSON` raises a warning in that case because PostgreSQL jsonb keeps only the last value for duplicate keys.

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 kann online getestet und angesehen werden unter geojson.io und geojsonlint.com. Es wird von vielen Web-Mapping-Frameworks unterstützt:

- Beispiel für ein OpenLayers GeoJSON
- Beispiel für ein Leaflet GeoJSON
- Beispiel für ein Mapbox GL GeoJSON

Verfügbarkeit: 1.3.4

Verfügbarkeit: 1.5.0 Unterstützung von geograpischen Koordinaten.

Änderung: 2.0.0 Unterstützung für Standardargumente und benannte Argumente.

Änderung: 3.0.0 Unterstützung von Datensätzen bei der Eingabe

Änderung: 3.0.0 Ausgabe der SRID wenn nicht EPSG:4326

Geändert: 3.5.0 erlaubt die Angabe der Spalte, die die Feature-ID enthält

Changed: 3.7.0 added warning about duplicate keys

 Diese Funktion unterstützt 3d und lässt den Z-Index nicht fallen.

Beispiele

Erzeugen Sie eine FeatureCollection:

Code

```
SELECT json_build_object(
    'type', 'FeatureCollection',
    'features', json_agg(ST_AsGeoJSON(t.*, id_column =
> 'id')::json)
    )
FROM ( VALUES (1, 'one', 'POINT(1 1)'::geometry),
              (2, 'two', 'POINT(2 2)'),
              (3, 'three', 'POINT(3 3)')
     ) as t(id, name, geom);
```

Output

```
{"type" : "FeatureCollection", "features" : [{"type": "Feature", "geometry": {"type":"Point","coordinates":[1,1]}, "id ↩
    ": 1, "properties": {"name": "one"}}, {"type": "Feature", "geometry": {"type":"Point","coordinates":[2,2]}, "id": ↩
    2, "properties": {"name": "two"}}, {"type": "Feature", "geometry": {"type":"Point","coordinates":[3,3]}, "id": 3, ↩
    "properties": {"name": "three"}}]}
```

Erzeugen Sie ein Feature:

Code

```
SELECT ST_AsGeoJSON(t.*, id_column =
> 'id')
FROM (VALUES (1, 'one', 'POINT(1 1)'::geometry)) AS t(id, name, geom);
```

Output

```
{"type": "Feature", "geometry": {"type":"Point","coordinates":[1,1]}, "id": 1, "properties": {"name": "one"}}
```

Vergessen Sie nicht, Ihre Daten in WGS84 Längen- und Breitengrade umzuwandeln, um der GeoJSON-Spezifikation zu entsprechen:

Code

```
SELECT ST_AsGeoJSON(ST_Transform(geom, 4326)) FROM fe_edges LIMIT 1;
```

Output

```
{"type":"MultiLineString","coordinates":[[[-89.734634999999997,31.492072000000000],
[-89.734955999999997,31.492237999999997]]]}
```

3D-Geometrien werden unterstützt:

Code

```
SELECT ST_AsGeoJSON('LINESTRING(1 2 3,4 5 6)');
```

Output

```
{"type":"LineString","coordinates":[[1,2,3],[4,5,6]]}
```

Das Argument options kann verwendet werden, um BBOX oder CRS in der GeoJSON-Ausgabe hinzuzufügen:

Code

```
SELECT ST_AsGeoJSON(ST_SetSRID('POINT(1 1)'::geometry, 4326), 9, 4|1);
```

Output

```
{"type":"Point","crs":{"type":"name","properties":{"name":"urn:ogc:def:crs:EPSG::4326"}},"bbox ↩
    ":[1.000000000,1.000000000,1.000000000,1.000000000],"coordinates":[1,1]}
```

Siehe auch

ST_GeomFromGeoJSON, ST_ForcePolygonCCW, ST_Transform

7.9.3.5 ST_AsGML

ST_AsGML — Gibt die Geometrie als GML-Element - Version 2 oder 3 - zurück.

Synopsis

text **ST_AsGML**(geometry geom, integer maxdecimaldigits=15, integer options=0);
text **ST_AsGML**(geography geog, integer maxdecimaldigits=15, integer options=0, text nprefix=null, text id=null);
text **ST_AsGML**(integer version, geometry geom, integer maxdecimaldigits=15, integer options=0, text nprefix=null, text id=null);
text **ST_AsGML**(integer version, geography geog, integer maxdecimaldigits=15, integer options=0, text nprefix=null, text id=null);

Beschreibung

Gibt die Geometrie als ein Geography Markup Language (GML) Element zurück. Ein Versionsparameter kann mit 2 oder 3 angegeben werden. Wenn kein Versionsparameter angegeben ist, wird dieser standardmäßig Version 2 angenommen. Der Parameter `maxdecimaldigits` kann verwendet werden, um die Anzahl der Nachkommastellen bei der Ausgabe zu reduzieren (standardmäßig 15).



Warnung
Die Verwendung des Parameters *`maxdecimaldigits`* kann dazu führen, dass die Ausgabegeometrie ungültig wird. Um dies zu vermeiden, verwenden Sie zuerst ST_ReducePrecision mit einer geeigneten Rastergröße.

GML 2 verweist auf Version 2.1.2, GML 3 auf Version 3.1.1

Der Übergabewert "options" ist ein Bitfeld. Es kann verwendet werden um das Koordinatenreferenzsystem bei der GML Ausgabe zu bestimmen und um die Daten in Länge/Breite anzugeben.

- 0: GML Kurzform für das CRS (z.B. EPSG:4326), Standardwert
- 1: GML Langform für das CRS (z.B. urn:ogc:def:crs:EPSG::4326)
- 2: Nur für GML 3, entfernt das srsDimension Attribut von der Ausgabe.
- 4: Nur für GML 3, Für Linien verwenden Sie bitte den Tag <LineString> anstatt <Curve>.
- 16: Deklarieren, dass die Daten in Breite/Länge (z.B. SRID=4326) vorliegen. Standardmäßig wird angenommen, dass die Daten planar sind. Diese Option ist nur bei Ausgabe in GML 3.1.1, in Bezug auf die Anordnung der Achsen sinnvoll. Falls Sie diese setzen, werden die Koordinaten von Länge/Breite auf Breite/Länge vertauscht.
- 32: Ausgabe der BBox der Geometrie (Umhüllende/Envelope).

Der Übergabewert 'namespace prefix' kann verwendet werden, um ein benutzerdefiniertes Präfix für den Namensraum anzugeben, oder kein Präfix (wenn leer). Wenn Null oder weggelassen, so wird das Präfix "gml" verwendet.

Verfügbarkeit: 1.3.2

Verfügbarkeit: 1.5.0 Unterstützung von geograpischen Koordinaten.

Erweiterung: 2.0.0 Unterstützung durch Präfix eingeführt. Für GML3 wurde die Option 4 eingeführt, um die Verwendung von LineString anstatt von Kurven für Linien zu erlauben. Ebenfalls wurde die GML3 Unterstützung für polyedrische Oberflächen und TINS eingeführt, sowie die Option 32 zur Ausgabe der BBox.

Änderung: 2.0.0 verwendet standardmäßig benannte Argumente.

Erweiterung: 2.1.0 Für GML 3 wurde die Unterstützung einer ID eingeführt.



Anmerkung
Nur die Version 3+ von ST_AsGML unterstützt polyedrische Oberflächen und TINs.

Diese Methode setzt die SQL/MM-Spezifikation um. SQL-MM IEC 13249-3: 17.2

Diese Funktion unterstützt 3d und lässt den Z-Index nicht fallen.

Diese Funktion unterstützt polyedrische Flächen.

Diese Funktion unterstützt Dreiecke und dreieckige unregelmäßige Netzoberflächen (TIN).

Beispiele

Version 2.

Code

```
SELECT ST_AsGML(ST_GeomFromText('POLYGON((0 0,0 1,1 1,1 0,0 0))', 4326));
```

Output

```
<gml:Polygon srsName="EPSG:4326"
><gml:outerBoundaryIs
><gml:LinearRing
><gml:coordinates
>0,0 0,1 1,1 1,0 0,0</gml:coordinates
></gml:LinearRing
></gml:outerBoundaryIs
></gml:Polygon
>
```

Version 3.

Flip coordinates and output extended EPSG (16 | 1)

Code

```
SELECT ST_AsGML(3, ST_GeomFromText('POINT(5.234234233242 6.34534534534)', 4326), 5, 17);
```

Output

```
<gml:Point srsName="urn:ogc:def:crs:EPSG::4326"
><gml:pos srsDimension="2"
>6.34535 5.23423</gml:pos
></gml:Point
>
```

Output the envelope (32).

Code

```
SELECT ST_AsGML(3, ST_GeomFromText('LINESTRING(1 2, 3 4, 10 20)',4326), 5, 32);
```

Output

```
<gml:Envelope srsName="EPSG:4326" srsDimension="2"
><gml:lowerCorner
>1 2</gml:lowerCorner
><gml:upperCorner
>10 20</gml:upperCorner
></gml:Envelope
>
```

Output the envelope (32), reverse (lat lon instead of lon lat) (16), long srs (1) = 32 | 16 | 1 = 49.

Code

```
SELECT ST_AsGML(3, ST_GeomFromText('LINESTRING(1 2, 3 4, 10 20)',4326), 5, 49);
```

Output

```
<gml:Envelope srsName="urn:ogc:def:crs:EPSG::4326" srsDimension="2"
><gml:lowerCorner
>2 1</gml:lowerCorner
><gml:upperCorner
>20 10</gml:upperCorner
></gml:Envelope
>
```

This example uses a PolyhedralSurface.

Code

```
SELECT ST_AsGML(
3,
'POLYHEDRALSURFACE( ((0 0 0, 0 0 1, 0 1 1, 0 1 0, 0 0 0)),
((0 0 0, 0 1 0, 1 1 0, 1 0 0, 0 0 0)), ((0 0 0, 1 0 0, 1 0 1, 0 0 1, 0 0 0)),
((1 1 0, 1 1 1, 1 0 1, 1 0 0, 1 1 0)),
((0 1 0, 0 1 1, 1 1 1, 1 1 0, 0 1 0)), ((0 0 1, 1 0 1, 1 1 1, 0 1 1, 0 0 1)) )'::geometry);
```

Output

```
 <gml:PolyhedralSurface>
<gml:polygonPatches>
   <gml:PolygonPatch>
    <gml:exterior>
        <gml:LinearRing>
           <gml:posList srsDimension="3"
>0 0 0 0 0 1 0 1 1 0 1 0 0 0 0</gml:posList>
        </gml:LinearRing>
    </gml:exterior>
   </gml:PolygonPatch>
   <gml:PolygonPatch>
    <gml:exterior>
        <gml:LinearRing>
           <gml:posList srsDimension="3"
>0 0 0 0 1 0 1 1 0 1 0 0 0 0 0</gml:posList>
        </gml:LinearRing>
    </gml:exterior>
   </gml:PolygonPatch>
   <gml:PolygonPatch>
    <gml:exterior>
        <gml:LinearRing>
           <gml:posList srsDimension="3"
>0 0 0 1 0 0 1 0 1 0 0 1 0 0 0</gml:posList>
        </gml:LinearRing>
    </gml:exterior>
   </gml:PolygonPatch>
   <gml:PolygonPatch>
    <gml:exterior>
        <gml:LinearRing>
           <gml:posList srsDimension="3"
>1 1 0 1 1 1 1 0 1 1 0 0 1 1 0</gml:posList>
        </gml:LinearRing>
    </gml:exterior>
   </gml:PolygonPatch>
   <gml:PolygonPatch>
    <gml:exterior>
        <gml:LinearRing>
           <gml:posList srsDimension="3"
>0 1 0 0 1 1 1 1 1 1 1 0 0 1 0</gml:posList>
        </gml:LinearRing>
    </gml:exterior>
   </gml:PolygonPatch>
   <gml:PolygonPatch>
    <gml:exterior>
        <gml:LinearRing>
           <gml:posList srsDimension="3"
>0 0 1 1 0 1 1 1 1 0 1 1 0 0 1</gml:posList>
        </gml:LinearRing>
    </gml:exterior>
   </gml:PolygonPatch>
```

```
</gml:polygonPatches>
</gml:PolyhedralSurface
>
```

Siehe auch

ST_GeomFromGML

7.9.3.6 ST_AsKML

ST_AsKML — Gibt die Geometrie als GML-Element - Version 2 oder 3 - zurück.

Synopsis

text **ST_AsKML**(geometry geom, integer maxdecimaldigits=15, text nprefix=NULL);
text **ST_AsKML**(geography geog, integer maxdecimaldigits=15, text nprefix=NULL);

Beschreibung

Gibt die Geometrie als ein Keyhole Markup Language (KML) Element zurück. Diese Funktion verfügt über mehrere Varianten. Die maximale Anzahl der Dezimalstellen die bei der Ausgabe verwendet wird (standardmäßig 15), die Version ist standardmäßig 2 und der Standardnamensraum hat kein Präfix.



Warnung
Die Verwendung des Parameters *maxdecimaldigits* kann dazu führen, dass die Ausgabegeometrie ungültig wird. Um dies zu vermeiden, verwenden Sie zuerst ST_ReducePrecision mit einer geeigneten Rastergröße.

Verfügbarkeit: 1.2.2 - spätere Varianten ab 1.3.2 nehmen den Versionsparameter mit auf

Erweiterung: 2.0.0 - Präfix Namensraum hinzugefügt. Standardmäßig kein Präfix

Geändert: 3.0.0 - Die Signatur der "versionierten" Variante wurde entfernt.



Anmerkung
Setzt voraus, dass PostGIS mit Proj-Unterstützung kompiliert wurde. Verwenden Sie bitte PostGIS_Full_Version, um festzustellen ob mit proj kompiliert wurde.



Anmerkung
Die Ausgabe AsKML funktioniert nicht bei Geometrien ohne SRID


Diese Funktion unterstützt 3d und lässt den Z-Index nicht fallen.

Beispiele

Code

```
SELECT ST_AsKML(ST_GeomFromText('POLYGON((0 0,0 1,1 1,1 0,0 0))', 4326));
```

Output

```
<Polygon
><outerBoundaryIs
><LinearRing
><coordinates
>0,0 0,1 1,1 1,0 0,0</coordinates
></LinearRing
></outerBoundaryIs
></Polygon
>
```

This example outputs a 3D LineString.

Code

```
SELECT ST_AsKML('SRID=4326;LINESTRING(1 2 3,4 5 6)');
```

Output

```
<LineString
><coordinates
>1,2,3 4,5,6</coordinates
></LineString
>
```

Siehe auch

ST_AsSVG, ST_AsGML

7.9.3.7 ST_AsLatLonText

ST_AsLatLonText — Gibt die "Grad, Minuten, Sekunden"-Darstellung für den angegebenen Punkt aus.

Synopsis

text **ST_AsLatLonText**(geometry pt, text format='');

Beschreibung

Gibt die "Grad, Minuten, Sekunden"-Darstellung des Punktes aus.



Anmerkung
Es wird angenommen, dass der Punkt in einer Breite/Länge-Projektion vorliegt. Die X (Länge) und Y (Breite) Koordinaten werden bei der Ausgabe in den "üblichen" Bereich (-180 to +180 für die Länge, -90 to +90 für die Breite) normalisiert.

Der Textparameter ist eine Zeichenkette für die Formatierung der Ausgabe, ähnlich wie die Zeichenkette für die Formatierung der Datumsausgabe. Gültige Zeichen sind "D" für Grad/Degrees, "M" für Minuten, "S" für Sekunden, und "C" für die Himmelsrichtung (NSEW). DMS Zeichen können wiederholt werden, um die gewünschte Zeichenbreite und Genauigkeit anzugeben ("SSS.SSSS" bedeutet z.B. " 1.0023").

"M", "S", und "C" sind optional. Wenn "C" weggelassen wird, werden Grad mit einem "-" Zeichen versehen, wenn Süd oder West. Wenn "S" weggelassen wird, werden die Minuten als Dezimalzahl mit der vorgegebenen Anzahl an Kommastellen angezeigt. Wenn "M" wegggelassen wird, werden die Grad als Dezimalzahl mit der vorgegebenen Anzahl an Kommastellen angezeigt.

Wenn die Zeichenkette für das Ausgabeformat weggelassen wird (oder leer ist) wird ein Standardformat verwendet.

Verfügbarkeit: 2.0

Beispiele

Standardformat.

Code

```
SELECT (ST_AsLatLonText('POINT (-3.2342342 -2.32498)'));
```

Output

```
2\textdegree{}19'29.928"S 3\textdegree{}14'3.243"W
```

Ein Format angeben (identisch mit Standardformat).

Code

```
SELECT (ST_AsLatLonText('POINT (-3.2342342 -2.32498)', 'D\textdegree{}M''S.SSS"C'));
```

Output

```
2\textdegree{}19'29.928"S 3\textdegree{}14'3.243"W
```

Andere Zeichen als D, M, S, C und "." werden lediglich durchgereicht.

Code

```
SELECT (ST_AsLatLonText('POINT (-3.2342342 -2.32498)', 'D degrees, M minutes, S seconds to ↩
    the C'));
```

Output

```
2 degrees, 19 minutes, 30 seconds to the S 3 degrees, 14 minutes, 3 seconds to the W
```

Grad mit einem Vorzeichen versehen - anstatt der Himmelsrichtung.

Code

```
SELECT (ST_AsLatLonText('POINT (-3.2342342 -2.32498)', 'D\textdegree{}M''S.SSS"'));
```

Output

```
-2\textdegree{}19'29.928" -3\textdegree{}14'3.243"
```

Dezimalgrad.

Code

```
SELECT (ST_AsLatLonText('POINT (-3.2342342 -2.32498)', 'D.DDDD degrees C'));
```

Output

```
2.3250 degrees S 3.2342 degrees W
```

To convert parsed degree, minute, and second components to decimal degrees, apply the cardinal-direction sign explicitly.

Code

```
WITH dms(label, degrees, minutes, seconds, hemisphere) AS (
  VALUES
    ('latitude', 43::numeric, 0::numeric, 50.60::numeric, 'S'),
    ('longitude', 147::numeric, 12::numeric, 18.20::numeric, 'E')
)
SELECT label,
       round(
         (CASE WHEN hemisphere IN ('S', 'W') THEN -1 ELSE 1 END)
         * (degrees + minutes / 60 + seconds / 3600),
         9
       ) AS decimal_degrees
FROM dms;
```

Output

```
   label    | decimal_degrees
------------+-----------------
 latitude   |   -43.014055556
 longitude  |   147.205055556
```

Überhöhte Werte werden normalisiert.

Code

```
SELECT (ST_AsLatLonText('POINT (-302.2342342 -792.32498)'));
```

Output

```
72\textdegree{}19'29.928"S 57\textdegree{}45'56.757"E
```

7.9.3.8 ST_AsMARC21

ST_AsMARC21 — Gibt die Geometrie als MARC21/XML-Datensatz mit einem geografischen Datenfeld (034) zurück.

Synopsis

text **ST_AsMARC21** (geometry geom , text format='hdddmmss');

Beschreibung

Diese Funktion gibt einen MARC21/XML-Datensatz mit Codierten kartographisch-mathematischen Daten zurück, der die Bounding Box einer bestimmten Geometrie darstellt. Der Parameter `Format` erlaubt die Codierung der Koordinaten in den Unterfeldern `$d`,`$e`,`$f` und `$g` in allen vom MARC21/XML-Standard unterstützten Formaten zu codieren. Gültige Formate sind:

- Himmelsrichtung, Grad, Minuten und Sekunden (Standard): `hdddmmss`
- Dezimalgrad mit Himmelsrichtung: `hddd.dddddd`
- Dezimalgrad ohne Himmelsrichtung: `ddd.dddddd`
- Dezimalminuten mit Himmelsrichtung: `hdddmm.mmmm`
- Dezimalminuten ohne Himmelsrichtung: `dddmm.mmmm`
- Dezimalsekunden mit Himmelsrichtung: `hdddmmss.sss`

Das Dezimalzeichen kann auch ein Komma sein, z.B. `hdddmm,mmmm`.

Die Genauigkeit von Dezimalformaten kann durch die Anzahl der Zeichen nach dem Dezimalzeichen begrenzt werden, z. B. `hdddmm.mm` für Dezimalminuten mit einer Genauigkeit von zwei Dezimalstellen.

Diese Funktion ignoriert die Dimensionen Z und M.

LOC MARC21/XML-Versionen werden unterstützt:

- MARC21/XML 1.1

Verfügbarkeit: 3.3.0



Anmerkung
Diese Funktion unterstützt keine Geometrien, die nicht lon/lat sind, da diese nicht vom MARC21/XML-Standard (Coded Cartographic Mathematical Data) unterstützt werden.



Anmerkung
Der MARC21/XML-Standard bietet keine Möglichkeit, das räumliche Referenzsystem für kartografisch-mathematische Daten zu kommentieren, was bedeutet, dass diese Information nach der Konvertierung in MARC21/XML verloren geht.

Beispiele

Konvertierung eines `POINT` in MARC21/XML im Format hdddmmss (Standard)

Code

```
SELECT ST_AsMARC21('SRID=4326;POINT(-4.504289 54.253312)'::geometry);
```

Output

```
                                st_asmarc21
-------------------------------------------------
<record xmlns="http://www.loc.gov/MARC21/slim">
    <datafield tag="034" ind1="1" ind2=" ">
        <subfield code="a"
>a</subfield>
        <subfield code="d"
>W0043015</subfield>
        <subfield code="e"
>W0043015</subfield>
        <subfield code="f"
>N0541512</subfield>
        <subfield code="g"
>N0541512</subfield>
    </datafield>
</record>
```

Konvertierung eines `POLYGON` in MARC21/XML in dezimaler Form

Code

```
SELECT ST_AsMARC21('SRID=4326;POLYGON((-4.5792388916015625 ↩
    54.18172660239091,-4.56756591796875 54.196993557130355,-4.546623229980469 ↩
    54.18313300502024,-4.5792388916015625 54.18172660239091))'::geometry,'hddd.dddd');

<record xmlns="http://www.loc.gov/MARC21/slim">
    <datafield tag="034" ind1="1" ind2=" ">
        <subfield code="a"
>a</subfield>
        <subfield code="d"
>W004.5792</subfield>
        <subfield code="e"
>W004.5466</subfield>
        <subfield code="f"
>N054.1970</subfield>
        <subfield code="g"
>N054.1817</subfield>
    </datafield>
</record>
```

Konvertierung einer `GEOMETRYCOLLECTION` in MARC21/XML im Dezimalminutenformat. Die Reihenfolge der Geometrien in der MARC21/XML-Ausgabe entspricht der Reihenfolge in der Sammlung.

Code

```
SELECT ST_AsMARC21('SRID=4326;GEOMETRYCOLLECTION(POLYGON((13.1 52.65,13.516666666666667 ↩
    52.65,13.516666666666667 52.38333333333333,13.1 52.38333333333333,13.1 52.65)),POINT ↩
    (-4.5 54.25))'::geometry,'hdddmm.mmmm');
```

Output

```
                                st_asmarc21
-----------------------------------------------------
<record xmlns="http://www.loc.gov/MARC21/slim">
    <datafield tag="034" ind1="1" ind2=" ">
        <subfield code="a"
>a</subfield>
        <subfield code="d"
>E01307.0000</subfield>
        <subfield code="e"
>E01331.0000</subfield>
        <subfield code="f"
>N05240.0000</subfield>
        <subfield code="g"
>N05224.0000</subfield>
    </datafield>
    <datafield tag="034" ind1="1" ind2=" ">
        <subfield code="a"
>a</subfield>
        <subfield code="d"
>W00430.0000</subfield>
        <subfield code="e"
>W00430.0000</subfield>
        <subfield code="f"
>N05415.0000</subfield>
        <subfield code="g"
>N05415.0000</subfield>
    </datafield>
</record>
```

Siehe auch

ST_GeomFromMARC21

7.9.3.9 ST_AsMVTGeom

ST_AsMVTGeom — Transformiert eine Geometrie in den Koordinatenraum einer MVT-Kachel.

Synopsis

geometry **ST_AsMVTGeom**(geometry geom, box2d bounds, integer extent=4096, integer buffer=256, boolean clip_geom=true);

Beschreibung

Transformiert eine Geometrie in den Koordinatenraum einer MVT-Kachel (Mapbox Vector Tile) und beschneidet sie gegebenenfalls auf die Kachelgrenzen. Die Geometrie muss im Koordinatensystem der Zielkarte liegen (ggf. mit ST_Transform). Üblicherweise ist dies Web Mercator (SRID:3857).

Die Funktion versucht, die Gültigkeit der Geometrie zu erhalten, und korrigiert sie gegebenenfalls. Dies kann dazu führen, dass die Ergebnisgeometrie auf eine niedrigere Dimension kollabiert.

The function performs geometry simplification appropriate to the integer tile coordinate space. After transforming to tile coordinates it snaps coordinates to the integer grid, removes repeated points, and removes points that lie on straight lines. It does not perform general-purpose, scale-dependent simplification such as ST_Simplify. To reduce detail before tile conversion, simplify the geometry in map coordinates before calling `ST_AsMVTGeom`.

Die rechteckigen Grenzen der Kachel im Koordinatenraum der Zielkarte müssen angegeben werden, damit die Geometrie transformiert und bei Bedarf beschnitten werden kann. Die Begrenzungen können mit ST_TileEnvelope erzeugt werden.

Diese Funktion wird verwendet, um die Geometrie in den von ST_AsMVT benötigten Kachelkoordinatenraum zu konvertieren.

`geom` ist die zu transformierende Geometrie, im Koordinatensystem der Zielkarte.

`bounds` ist die rechteckige Begrenzung der Kachel im Kartenkoordinatenraum, ohne Puffer.

`extent` ist die Größe der Kachelausdehnung im Kachelkoordinatenraum, wie in der MVT-Spezifikation definiert. Der Standardwert ist 4096.

`buffer` ist die Puffergröße im Kachelkoordinatenraum für das Clipping der Geometrie. Der Standardwert ist 256.

`clip_geom` ist ein boolescher Wert, der bestimmt, ob Geometrien abgeschnitten oder unverändert kodiert werden. Der Standardwert ist true.

Verfügbarkeit: 2.4.0



Anmerkung
Ab 3.0 kann zum Ausschneiden und Validieren von MVT-Polygonen bei der Konfiguration Wagyu gewählt werden. Diese Bibliothek ist schneller und liefert genauere Ergebnisse als die standardmäßige GEOS-Bibliothek, sie kann aber kleine Polygone verwerfen.

Beispiele

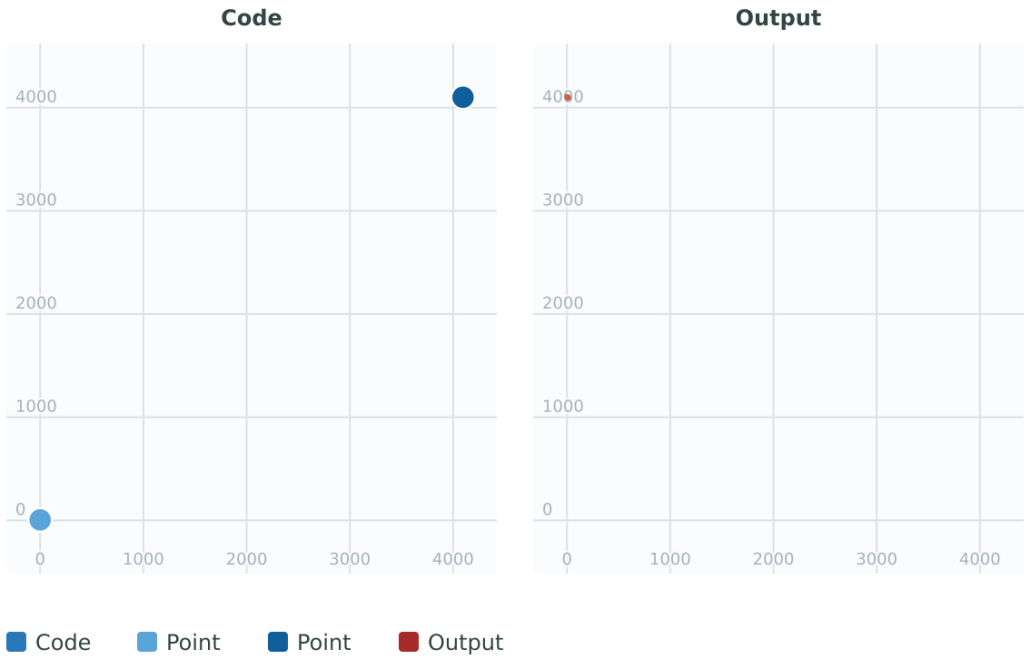
Code

```
SELECT ST_AsMVTGeom('POLYGON ((0 0,10 0,10 5,0 -5,0 0))',
    ST_MakeBox2D(ST_Point(0, 0), ST_Point(4096, 4096)),
    4096, 0, false);
```

Output

```
MULTIPOLYGON(((5 4096,10 4091,10 4096,5 4096)),((5 4096,0 4101,0 4096,5 4096)))
```

Figure



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.

Code

```
SELECT ST_AsMVTGeom(ST_Transform(geom, 3857 ),
            ST_TileEnvelope(12, 513, 412),
            extent =
> 4096,
            buffer =
> 64) AS geom
  FROM data
  WHERE geom && ST_Transform(ST_TileEnvelope(12, 513, 412, margin =
> (64.0 / 4096)), 4326)
```

Siehe auch

ST_AsMVT, ST_TileEnvelope, PostGIS_Wagyu_Version

7.9.3.10 ST_AsMVT

ST_AsMVT — Aggregatfunktion, die eine MVT-Darstellung einer Reihe von Zeilen zurückgibt.

Synopsis

bytea **ST_AsMVT**(anyelement set row);
bytea **ST_AsMVT**(anyelement row, text name);
bytea **ST_AsMVT**(anyelement row, text name, integer extent);
bytea **ST_AsMVT**(anyelement row, text name, integer extent, text geom_name);
bytea **ST_AsMVT**(anyelement row, text name, integer extent, text geom_name, text feature_id_name);

Beschreibung

Eine Aggregatfunktion, die eine binäre Mapbox Vector Tile-Darstellung eines Satzes von Zeilen zurückgibt, die einer Kachelebene entsprechen. Die Zeilen müssen eine Geometriespalte enthalten, die als Feature-Geometrie kodiert wird. Die Geometrie muss im Kachelkoordinatenraum vorliegen und gemäß der MVT-Spezifikation gültig sein. ST_AsMVTGeom kann zur Transformation der Geometrie in den Kachelkoordinatenraum verwendet werden. Andere Zeilenspalten werden als Feature-Attribute kodiert.

Das Mapbox Vector Tile Format kann Geoobjekte mit unterschiedlichen Attributen speichern. Um diese Möglichkeit zu nutzen, muss eine JSONB-Spalte in den Datensätzen mitgeliefert werden, welche in einer tieferen Ebene die JSON-Objekte enthält. Die Schlüssel und Werte im JSONB werden als Featureattribute kodiert.

Durch das Aneinanderhängen mehrerer Funktionsaufrufe mittels `||`, können Tiles mit mehreren Ebenen erstellt werden.



Wichtig
Darf nicht mit einer `GEOMETRYCOLLECTION` im Datensatz aufgerufen werden. Es kann aber ST_AsMVTGeom verwendet werden, um eine Sammelgeometrie einzubinden.

`row` Datenzeilen mit zumindest einer Geometriespalte.

`name` ist die Bezeichnung der Ebene. Standardmäßig die Zeichenkette "default".

`extent` ist die Ausdehnung der Tiles in Bildschirmeinheiten. Standardmäßig 4096.

`geom_name` ist der Name der Geometriespalte in den Zeilendaten. Standard ist die erste Geometriespalte. Beachten Sie, dass PostgreSQL standardmäßig automatisch unquotierte Bezeichner in Kleinbuchstaben umwandelt, was bedeutet, dass dieser Parameter in Kleinbuchstaben angegeben werden muss, es sei denn, die Geometriespalte ist in Anführungszeichen gesetzt, z. B. `"MyMVTGeom"`.

`feature_id_name` ist die Bezeichnung der Feature-ID Spalte im Datensatz. Ist der Übergabewert NULL oder negativ, dann wird die Feature-ID nicht gesetzt. Die erste Spaltee mit einem passenden Namen und einem gültigen Datentyp (Smallint, Integer, Bigint) wird als Feature-ID verwendet, alle nachfolgenden Spalten werden als Eigenschaften hinzugefügt. JSON-Properties werden nicht unterstützt.

Erweiterung: 3.0 - Unterstützung für eine Feature-ID.

Erweiterung: 2.5.0 - Unterstützung von nebenläufigen Abfragen.

Verfügbarkeit: 2.4.0

Beispiele

Code

```
WITH mvtgeom AS
(
  SELECT ST_AsMVTGeom(geom, ST_TileEnvelope(12, 513, 412), extent =
> 4096, buffer =
> 64) AS geom, name, description
  FROM points_of_interest
  WHERE geom && ST_TileEnvelope(12, 513, 412, margin =
> (64.0 / 4096))
)
SELECT ST_AsMVT(mvtgeom.*)
FROM mvtgeom;
```

Siehe auch

ST_AsMVTGeom, ST_TileEnvelope

7.9.3.11 ST_AsSVG

ST_AsSVG — Gibt eine Geometrie als SVG-Pfad aus.

Synopsis

text **ST_AsSVG**(geometry geom, integer rel=0, integer maxdecimaldigits=15);
text **ST_AsSVG**(geography geog, integer rel=0, integer maxdecimaldigits=15);

Beschreibung

Gibt die Geometrie als Skalare Vektor Graphik (SVG-Pfadgeometrie) aus. Verwenden Sie 1 als zweiten Übergabewert um die Pfadgeometrie in relativen Schritten zu implementieren; Standardmäßig (oder 0) verwendet absolute Schritte. Der dritte Übergabewert kann verwendet werden, um die maximale Anzahl der Dezimalstellen bei der Ausgabe einzuschränken (standardmäßig 15). Punktgeometrie wird als cx/cy übersetzt wenn der Übergabewert 'rel' gleich 0 ist, x/y wenn 'rel' 1 ist. Mehrfachgeometrie wird durch Beistriche (",") getrennt, Sammelgeometrie wird durch Strichpunkt (";") getrennt.

Für die Arbeit mit PostGIS-SVG-Grafiken gibt es die pg_svg-Bibliothek, die plpgsql-Funktionen für die Arbeit mit Ausgaben von ST_AsSVG bereitstellt.

Verbessert: 3.4.0 zur Unterstützung aller Kurventypen

Änderung: 2.0.0 verwendet Standardargumente und unterstützt benannte Argumente.

Verfügbarkeit: 1.2.2. Änderung: 1.4.0 L-Befehl beim absoluten Pfad aufgenommen, um mit http://www.w3.org/TR/SVG/paths.html#Path konform zu sein.

Diese Methode unterstützt kreisförmige Strings und Kurven.

Beispiele

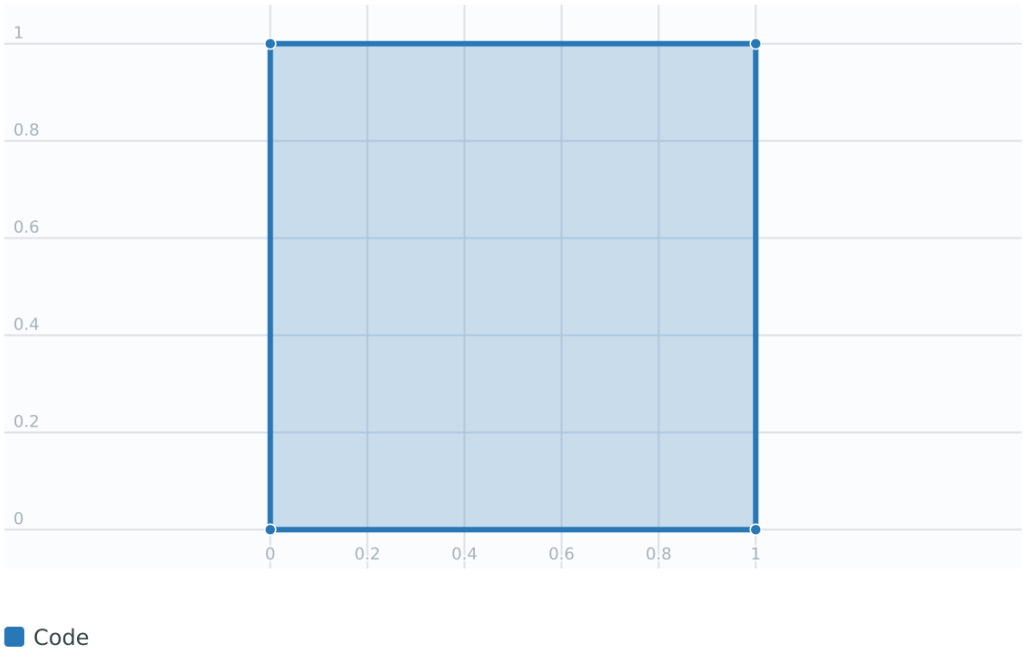
Code

```
SELECT ST_AsSVG('POLYGON((0 0,0 1,1 1,1 0,0 0))'::geometry);
```

Output

```
M 0 0 L 0 -1 1 -1 1 0 Z
```

Figure



Kreisförmige Schnur

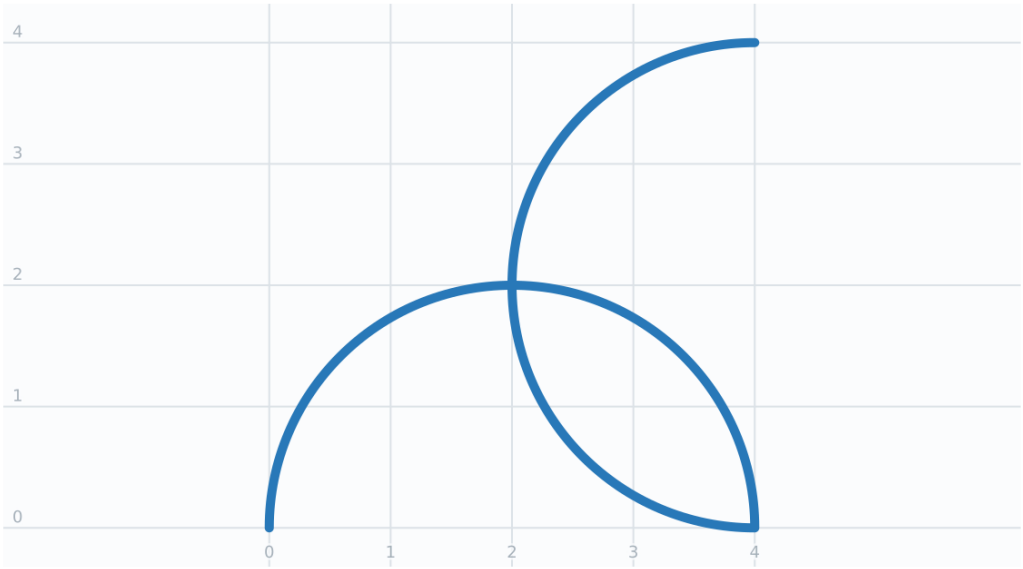
Code

```
SELECT ST_AsSVG('CIRCULARSTRING(-2 0,0 2,2 0,0 2,2 4)'::geometry);
```

Output

```
M -2 0 A 2 2 0 0 1 2 0 A 2 2 0 0 1 2 -4
```

Figure



Multi-Kurve

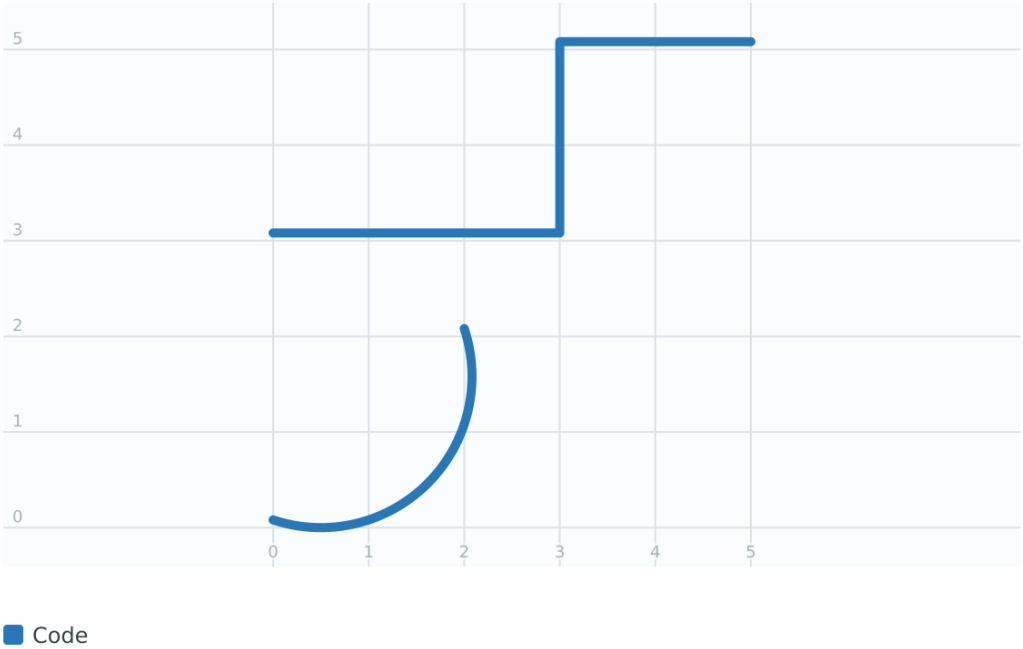
Code

```
SELECT ST_AsSVG('MULTICURVE((5 5,3 5,3 3,0 3),
CIRCULARSTRING(0 0,2 1,2 2))'::geometry, 0, 0);
```

Output

```
M 5 -5 L 3 -5 3 -3 0 -3 M 0 0 A 2 2 0 0 0 2 -2
```

Figure



Multi-Surface

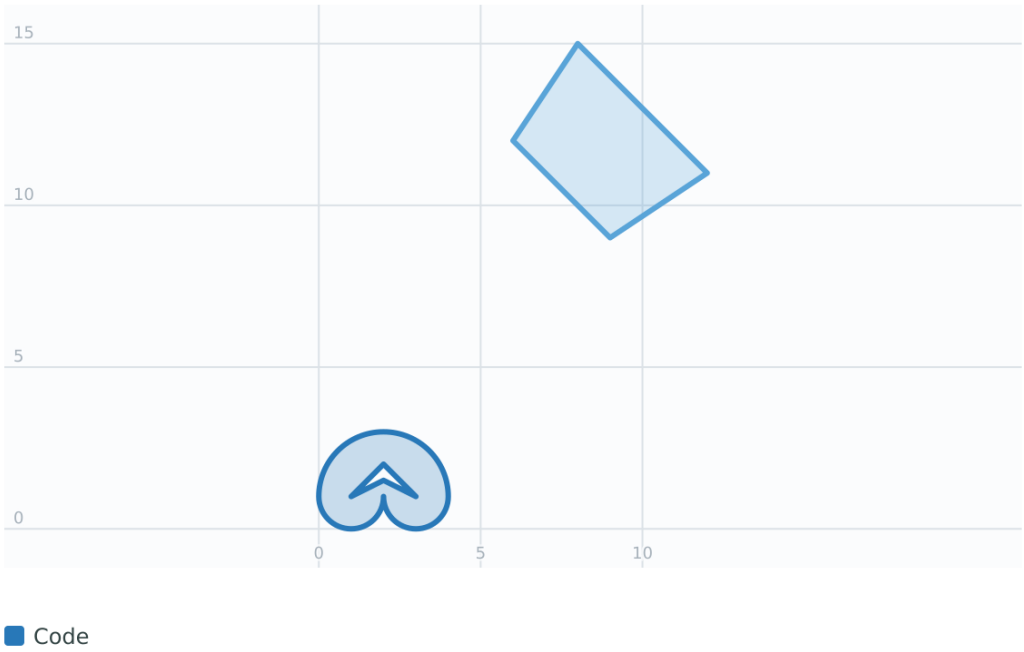
Code

```
SELECT ST_AsSVG('MULTISURFACE(
CURVEPOLYGON(CIRCULARSTRING(-2 0,-1 -1,0 0,1 -1,2 0,0 2,-2 0),
    (-1 0,0 0.5,1 0,0 1,-1 0)),
((7 8,10 10,6 14,4 11,7 8)))'::geometry, 0, 2);
```

Output

```
M -2 0 A 1 1 0 0 0 0 0 A 1 1 0 0 0 2 0 A 2 2 0 0 0 -2 0 Z
M -1 0 L 0 -0.5 1 0 0 -1 -1 0 Z
M 7 -8 L 10 -10 6 -14 4 -11 Z
```

Figure



7.9.3.12 ST_AsTWKB

ST_AsTWKB — Gibt die Geometrie als TWKB, aka "Tiny Well-known Binary" zurück

Synopsis

bytea **ST_AsTWKB**(geometry geom, integer prec=0, integer prec_z=0, integer prec_m=0, boolean with_sizes=false, boolean with_boxes=false);
bytea **ST_AsTWKB**(geometry[] geom, bigint[] ids, integer prec=0, integer prec_z=0, integer prec_m=0, boolean with_sizes=false, boolean with_boxes=false);

Beschreibung

Gibt die Geometrie im TWKB ("Tiny Well-Known Binary") Format aus. TWKB ist ein komprimiertes binäres Format mit dem Schwerpunkt, die Ausgabegröße zu minimieren.

Der Parameter 'decimaldigits' bestimmt die Anzahl der Dezimalstellen bei der Ausgabe. Standardmäßig werden die Werte vor der Zeichenkodierung auf die Einserstelle gerundet. Wenn Sie die Daten mit höherer Genauigkeit übergeben wollen, erhöhen Sie bitte die Anzahl der Dezimalstellen. Zum Beispiel bedeutet ein Wert von 1, dass die erste Dezimalstelle erhalten bleibt.

Die Parameter "sizes" und "bounding_boxes" bestimmen ob zusätzliche Information über die kodierte Länge und die Abgrenzung des Objektes in der Ausgabe eingebunden werden. Standardmäßig passiert dies nicht. Drehen Sie diese bitte nicht auf, solange dies nicht von Ihrer Client-Software benötigt wird, da dies nur unnötig Speicherplatz verbraucht (Einsparen von Speicherplatz ist der Sinn von TWKB).

Das Feld-Eingabeformat dieser Funktion wird verwendet um eine Sammelgeometriee und eindeutige Identifikatoren in eine TWKB-Collection zu konvertieren, welche die Identifikatoren erhält. Dies ist nützlich für Clients, die davon ausgehen, eine Sammelgeometrie auszupacken, um so auf zusätzliche Information über die internen Objekte zuzugreifen. Sie können das Feld mit der Funktion array_agg erstellen. Die anderen Parameter bewirken dasselbe wie bei dem einfachen Format dieser Funktion.



Anmerkung
Die Formatspezifikation steht Online unter https://github.com/TWKB/Specification zur Verfügung, und Code zum Aufbau eines JavaScript Clints findet sich unter https://github.com/TWKB/twkb.js.

Erweiterung: 2.4.0 Hauptspeicher- und Geschwindigkeitsverbesserungen.

Verfügbarkeit: 2.2.0

Beispiele

Code

```
SELECT ST_AsTWKB('LINESTRING(1 1,5 5)'::geometry);
```

Output

```
\x02000202020808
```

Um ein aggregiertes TWKB-Objekt inklusive Identifikatoren zu erzeugen, fassen Sie bitte die gewünschte Geometrie und Objekte zuerst mittels "array_agg()" zusammen und rufen anschließend die passende TWKB Funktion auf.

Code

```
SELECT ST_AsTWKB(array_agg(geom), array_agg(gid)) FROM mytable;
```

Output

```
\x040402020400000202
```

Siehe auch

ST_GeomFromTWKB, ST_AsBinary, ST_AsEWKB, ST_AsEWKT, ST_GeomFromText

7.9.3.13 ST_AsX3D

ST_AsX3D — Gibt eine Geometrie im X3D XML Knotenelement-Format zurück: ISO-IEC-19776-1.2-X3DEncodings-XML

Synopsis

text **ST_AsX3D**(geometry g1, integer maxdecimaldigits=15, integer options=0);

Beschreibung

Gibt eine Geometrie als X3D knotenformatiertes XML Element zurück http://www.web3d.org/standards/number/19776-1. Falls `maxdecimaldigits` (Genauigkeit) nicht angegeben ist, wird sie standardmäßig 15.



Anmerkung
Es gibt verschiedene Möglichkeiten eine PostGIS Geometrie in X3D zu übersetzen, da sich der X3D Geometrietyp nicht direkt in den geometrischen Datentyp von PostGIS abbilden lässt. Einige neuere X3D Datentypen, die sich besser abbilden lassen könnten haben wir vermieden, da diese von den meisten Rendering-Tools zurzeit nicht untestützt werden. Dies sind die Abbildungen für die wir uns entschieden haben. Falls Sie Ideen haben, wie wir es den Anwendern ermöglichen können ihre bevorzugten Abbildungen anzugeben, können Sie gerne ein Bug-Ticket senden.
Im Folgenden wird beschrieben, wie der PostGIS 2D/3D Datentyp derzeit in den X3D Datentyp abgebildet wird

The 'options' argument is a bitfield that denotes whether to represent coordinates with the X3D GeoCoordinates geospatial node and whether to flip the x/y axis. By default, `ST_AsX3D` outputs in database form (long,lat or X,Y), but the X3D default of lat/lon, y/x may be preferred.

- 0: X/Y in der Datenbankreihenfolge (z.B. ist Länge/Breite = X,Y die standardmäßige Datenbankreihenfolge), Standardwert, und nicht-spatiale Koordinaten (nur der normale alte Koordinaten-Tag).
- 1: X und Y umdrehen. In Verbindung mit der Option für GeoKoordinaten wird bei der Standardausgabe die Breite zuerst/"latitude_first" ausgegeben und die Koordinaten umgedreht.
- 2: Die Koordinaten werden als geospatiale GeoKoordinaten ausgegeben. Diese Option gibt eine Fehlermeldung aus, falls die Geometrie nicht in WGS 84 Länge/Breite (SRID: 4326) vorliegt. Dies ist zurzeit der einzige GeoKoordinaten-Typ der unterstützt wird.Siehe die X3D Spezifikation für Koordinatenreferenzsysteme. Die Standardausgabe ist `GeoCoordinate geoSystem='"GD" "WE" "longitude_first"'`. Wenn Sie den X3D Standard bevorzugen `GeoCoordinate geoSyste "WE" "latitude_first"'` verwenden Sie bitte `(2+1) = 3`

PostGIS Datentyp	2D X3D Datentyp	3D X3D Datentyp
LINESTRING	LineSet (coordinates are promoted to 3D)	LineSet
MULTILINESTRING	IndexedLineSet (coordinates are promoted to 3D)	IndexedLineSet
MULTIPOINT	Polypoint2D	PointSet
POINT	gibt leerzeichengetrennte Koordinaten aus	gibt leerzeichengetrennte Koordinaten aus
(MULTI) POLYGON, POLYHEDRALSURFACE	IndexedFaceSet (coordinates are promoted to 3D; inner rings currently output as another faceset)	IndexedFaceSet (die inneren Ringe werden zurzeit als ein weiteres FaceSet abgebildet)
TIN	TriangleSet2D (zurzeit nicht implementiert)	IndexedTriangleSet
CIRCULARSTRING, COMPOUNDCURVE, CURVEPOLYGON, MULTICURVE, MULTISURFACE	linearized, then emitted with the linear geometry mapping	linearized, then emitted with the linear geometry mapping
NURBSCURVE	NurbsCurve with controlPoint, knot, weight, and order fields	NurbsCurve with controlPoint, knot, weight, and order fields



Anmerkung
Die Unterstützung von 2D-Geometrie ist noch nicht vollständig. Die inneren Ringe werden zur Zeit lediglich als gesonderte Polygone abgebildet. Wir arbeiten daran.

M coordinates are not represented in X3D output.

Viele Fortschritte im 3D-Bereich, insbesondere mit X3D-Integration mit HTML5

Es gibt auch einen feinen OpenSource X3D Viewer, den Sie benützen können, um Geometrien darzustellen. Free Wrl http://freewrl.source Binärdateien sind für Mac, Linux und Windows verfügbar. Sie können den mitgelieferten FreeWRL_Launcher verwenden, um Gemetrien darzustellen.

Schauen Sie sich auch PostGIS minimalistischer X3D-Viewer an, der diese Funktion nutzt, und x3dDom html/js open source toolkit.

Verfügbarkeit: 2.0.0: ISO-IEC-19776-1.2-X3DEncodings-XML

Erweiterung: 2.2.0: Unterstützung für geographische Koordinaten und Vertauschen der Achsen (x/y, Länge/Breite). Für nähere Details siehe Optionen.

Enhanced: 3.7.0: Support for curve geometries. NURBSCURVE is emitted as X3D NurbsCurve; other curve types are linearized before output.

Diese Funktion unterstützt 3d und lässt den Z-Index nicht fallen.

Diese Funktion unterstützt polyedrische Flächen.

Diese Funktion unterstützt Dreiecke und dreieckige unregelmäßige Netzoberflächen (TIN).

Beispiele

Create a complete X3D document containing a cube that is viewable in FreeWRL and other X3D viewers.

Code

```
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;
```

Output

```
    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 1 0 1 1' />
     </IndexedFaceSet>
     </Shape>
    </Transform>
  </Scene>
</X3D>
```

PostGIS buildings.

Kopieren Sie die Ausgabe dieser Abfrage und fügen Sie sie in x3d scene viewer ein und klicken Sie auf Anzeigen

Code

```
SELECT string_agg('<Shape
>' || ST_AsX3D(CG_Extrude(geom, 0,0, i*0.5)) ||
    '<Appearance>
          <Material diffuseColor="' || (0.01*i)::text || ' 0.8 0.2" specularColor="' ||  ↩
    (0.05*i)::text || ' 0 0.5"/>
        </Appearance>
    </Shape
>', '')
FROM ST_Subdivide(ST_Letters('PostGIS'),20) WITH ORDINALITY AS f(geom,i);
```

The following preview shows the extruded geometry generated from a simplified version of the same source lettering. It is a geometry preview, not X3D markup; use the preceding `ST_AsX3D` query output in an X3D viewer to inspect the native X3D scene.

Code

```
SELECT
    CG_Extrude(
        ST_SimplifyPreserveTopology(ST_Letters('PostGIS'), 2),
        0, 0, 10
    ) AS buildings;
```

Figure



An Octagon elevated 3 Units and decimal precision of 6.

Code

```
SELECT ST_AsX3D(ST_Translate(ST_Force3D(ST_Buffer(ST_Point(10,10),5, 'quad_segs=2')), 0,0,
    3),
  6) As x3dfrag;
```

Output

```
x3dfrag
--------
<IndexedFaceSet  convex='false' coordIndex='0 1 2 3 4 5 6 7'
><Coordinate point='15 10 3 13.535534 6.464466 3 10 5 3 6.464466 6.464466 3 5 10 3 6.464466 13.535534 3 10 15 3 ↩
     13.535534 13.535534 3 ' /></IndexedFaceSet>
```

TIN.

Code

```
SELECT ST_AsX3D(ST_GeomFromEWKT('TIN (((
                0 0 0,
                0 0 1,
                0 1 0,
                0 0 0
            )), ((
                0 0 0,
                0 1 0,
                1 1 0,
                0 0 0
            ))
            )')) As x3dfrag;
```

Output

```
    x3dfrag
    --------
<IndexedTriangleSet  index='0 1 2 3 4 5'
><Coordinate point='0 0 0 0 0 1 0 1 0 0 0 0 0 1 0 1 1 0'/></IndexedTriangleSet>
```

Closed multilinestring (the boundary of a polygon with holes).

Code

```
SELECT ST_AsX3D(ST_GeomFromEWKT('MULTILINESTRING((20 0 10,16 -12 10,0 -16 10,-12 -12 10,-20 ↩
    0 10,-12 16 10,0 24 10,16 16 10,20 0 10),
  (12 0 10,8 8 10,0 12 10,-8 8 10,-8 0 10,-8 -4 10,0 -8 10,8 -4 10,12 0 10))')
) As x3dfrag;
```

Output

```
    x3dfrag
    --------
<IndexedLineSet  coordIndex='0 1 2 3 4 5 6 7 0 -1 8 9 10 11 12 13 14 15 8'>
    <Coordinate point='20 0 10 16 -12 10 0 -16 10 -12 -12 10 -20 0 10 -12 16 10 0 24 10 16 16 10 12 0 10 8 8 10 0 12 10 ↩
      -8 8 10 -8 0 10 -8 -4 10 0 -8 10 8 -4 10 ' />
 </IndexedLineSet>
```

7.9.3.14 ST_GeoHash

ST_GeoHash — Gibt die Geometrie in der GeoHash Darstellung aus.

Synopsis

text **ST_GeoHash**(geometry geom, integer maxchars=full_precision_of_point);

Beschreibung

Berechnet eine GeoHash-Darstellung einer Geometrie. Ein GeoHash kodiert einen geografischen Punkt in eine Textform, die auf der Grundlage von Präfixen sortierbar und durchsuchbar ist. Ein kürzerer GeoHash ist eine weniger präzise Darstellung eines Punktes. Man kann ihn sich als eine Box vorstellen, die den Punkt enthält.

Auch Nicht-Punkt-Geometriewerte mit einer Ausdehnung ungleich Null können auf GeoHash-Codes abgebildet werden. Die Genauigkeit des Codes hängt von der geografischen Ausdehnung der Geometrie ab.

Wenn `maxchars` nicht angegeben wird, ist der zurückgegebene GeoHash-Code für die kleinste Zelle, die die Eingabegeometrie enthält. Punkte geben einen GeoHash mit einer Genauigkeit von 20 Zeichen zurück (ungefähr genug, um die volle doppelte Genauigkeit der Eingabe zu speichern). Andere Geometrietypen können je nach Umfang der Geometrie einen GeoHash mit geringerer Genauigkeit zurückgeben. Größere Geometrien werden mit geringerer, kleinere mit höherer Genauigkeit dargestellt. Der durch den GeoHash-Code ermittelte Kasten enthält immer das eingegebene Merkmal.

Wenn `maxchars` angegeben ist, hat der zurückgegebene GeoHash-Code höchstens so viele Zeichen. Er entspricht einer (möglicherweise) weniger genauen Darstellung der Eingabegeometrie. Bei Nicht-Punkten ist der Ausgangspunkt der Berechnung der Mittelpunkt des Begrenzungsrahmens der Geometrie.

Verfügbarkeit: 1.4.0



Anmerkung
ST_GeoHash setzt voraus, dass die Eingabegeometrie in geografischen (Lon/Lat) Koordinaten vorliegt.

 Diese Methode unterstützt kreisförmige Strings und Kurven.

Beispiele

Code

```
SELECT ST_GeoHash(ST_Point(-126, 48));
```

Output

```
c0w3hf1s70w3hf1s70w3
```

Code

```
SELECT ST_GeoHash(ST_Point(-126, 48), 5);
```

Output

```
c0w3h
```

This example shows that the GeoHash for a line containing a point is a prefix of the point code.

Code

```
SELECT ST_GeoHash('LINESTRING(-126 48,-126.1 48.1)'::geometry);
```

Output

```
c0w3
```

Siehe auch

ST_GeomFromGeoHash, ST_PointFromGeoHash, ST_Box2dFromGeoHash

7.10 Operatoren

7.10.1 Bounding-Box-Operatoren

7.10.1.1 &&

&& — Gibt TRUE zurück, wenn die 2D Bounding Box von A die 2D Bounding Box von B schneidet.

Synopsis

boolean **&&**(geometry A , geometry B);
boolean **&&**(geography A , geography B);

Beschreibung

Der && Operator gibt TRUE zurück, wenn die 2D Bounding Box von Geometrie A die 2D Bounding Box der Geometrie von B schneidet.



Anmerkung
Dieser Operand benützt jeden Index, der für die Geometrie zur Verfügung stellt.

Erweiterung: Mit 2.0.0 wurde die Unterstützung für polyedrische Oberflächen eingeführt.

Verfügbarkeit: Mit 1.5.0 wurde die Unterstützung von geograpischen Koordinaten eingeführt

 Diese Methode unterstützt kreisförmige Strings und Kurven.

 Diese Funktion unterstützt polyedrische Flächen.

Beispiele

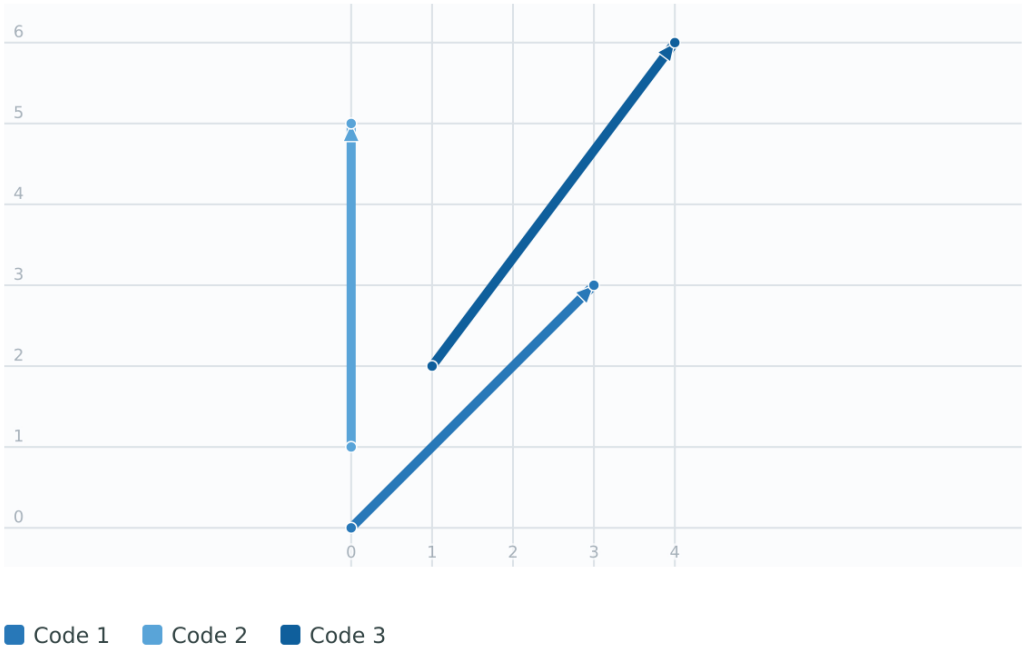
Code

```
SELECT tbl1.column1, tbl2.column1, tbl1.column2 && tbl2.column2 AS overlaps
FROM ( VALUES
    (1, 'LINESTRING(0 0, 3 3)'::geometry),
    (2, 'LINESTRING(0 1, 0 5)'::geometry)) AS tbl1,
( VALUES
    (3, 'LINESTRING(1 2, 4 6)'::geometry)) AS tbl2;
```

Output

```
 column1 | column1 | overlaps
---------+---------+----------
       1 |       3 | t
       2 |       3 | f
(2 rows)
```

Figure



Siehe auch

ST_Intersects, &>, &<|, &<, ~, @

7.10.1.2 &&(geometry,box2df)

&&(geometry,box2df) — Gibt `TRUE` zurück, wenn sich die 2D Bounding Box (cached) einer Geometrie mit einer 2D Bounding Box mit Gleitpunktgenauigkeit (BOX2DF) überschneidet.

Synopsis

boolean **&&**(geometry A , box2df B);

Beschreibung

Der `&&` Operator gibt `TRUE` zurück, wenn die im Cache befindliche 2D Bounding Box der Geometrie A sich mit der 2D Bounding Box von B, unter Verwendung von Gleitpunktgenauigkeit überschneidet. D.h.: falls B eine (double precision) box2d ist, wird diese intern in eine auf Gleitpunkt genaue 2D Bounding Box (BOX2DF) umgewandelt.



Anmerkung
Dieser Operand ist eher für die interne Nutzung durch BRIN Indizes, als durch die Anwender, gedacht.

Availability: 2.3.0 support for Block Range INdexes (BRIN) was introduced.

Diese Methode unterstützt kreisförmige Strings und Kurven.

Diese Funktion unterstützt polyedrische Flächen.

Beispiele

Code

```
SELECT ST_Point(1,1) && ST_MakeBox2D(ST_Point(0,0), ST_Point(2,2)) AS overlaps;
```

Output

```
 overlaps
----------
 t
(1 row)
```

Siehe auch

&&(box2df,geometry), &&(box2df,box2df), ~(geometry,box2df), ~(box2df,geometry), ~(box2df,box2df), @(geometry,box2df), @(box2df,geometry), @(box2df,box2df)

7.10.1.3 &&(box2df,geometry)

&&(box2df,geometry) — Gibt TRUE zurück, wenn eine 2D float precision bounding box (BOX2DF) eine Geometrie (cached) 2D bounding box schneidet.

Synopsis

boolean **&&**(box2df A , geometry B);

Beschreibung

Der && Operator gibt TRUE zurück, wenn die 2D Bounding Box A die zwischengespeicherte 2D Bounding Box der Geometrie B, unter Benutzung von Fließpunktgenauigkeit, schneidet. D.h.: wenn A eine (double precision) box2d ist, wird diese intern in eine float precision 2D bounding box (BOX2DF) umgewandelt.



Anmerkung
Dieser Operand ist eher für die interne Nutzung durch BRIN Indizes, als durch die Anwender, gedacht.

Availability: 2.3.0 support for Block Range INdexes (BRIN) was introduced.

Diese Methode unterstützt kreisförmige Strings und Kurven.

Diese Funktion unterstützt polyedrische Flächen.

Beispiele

Code

```
SELECT ST_MakeBox2D(ST_Point(0,0), ST_Point(2,2)) && ST_Point(1,1) AS overlaps;
```

Output

```
 overlaps
----------
 t
(1 row)
```

Siehe auch

&&(geometry,box2df), &&(box2df,box2df), ~(geometry,box2df), ~(box2df,geometry), ~(box2df,box2df), @(geometry,box2df), @(box2df,geometry), @(box2df,box2df)

7.10.1.4 &&(box2df,box2df)

&&(box2df,box2df) — Gibt TRUE zurück, wenn sich zwei 2D float precision Bounding Boxes (BOX2DF) überschneiden.

Synopsis

boolean **&&**(box2df A , box2df B);

Beschreibung

Der && Operator gibt TRUE zurück, wenn sich zwei 2D Bounding Boxes A und B, unter Benutzung von float precision, gegenseitig überschneiden. D.h.: Wenn A (oder B) eine (double precision) box2d ist, wird diese intern in eine float precision 2D bounding box (BOX2DF) umgewandelt



Anmerkung
Dieser Operator ist für die interne Nutzung durch BRIN Indizes, und nicht so sehr durch Anwender, vorgesehen.

Availability: 2.3.0 support for Block Range INdexes (BRIN) was introduced.

Diese Methode unterstützt kreisförmige Strings und Kurven.

Diese Funktion unterstützt polyedrische Flächen.

Beispiele

Code

```
SELECT ST_MakeBox2D(ST_Point(0,0), ST_Point(2,2)) && ST_MakeBox2D(ST_Point(1,1), ST_Point ↩
    (3,3)) AS overlaps;
```

Output

```
 overlaps
----------
 t
(1 row)
```

Siehe auch

&&(geometry,box2df), &&(box2df,geometry), ~(geometry,box2df), ~(box2df,geometry), ~(box2df,box2df), @(geometry,box2df), @(box2df,geometry), @(box2df,box2df)

7.10.1.5 &&&

&&& — Gibt TRUE zurück, wenn A's n-D bounding box B's n-D bounding box schneidet.

Synopsis

boolean **&&&**(geometry A , geometry B);

Beschreibung

Der `&&&` Operator gibt `TRUE` zurück, wenn die n-D bounding box der Geometrie A die n-D bounding box der Geometrie B schneidet.



Anmerkung
Dieser Operand benützt jeden Index, der für die Geometrie zur Verfügung stellt.

Verfügbarkeit: 2.0.0

 Diese Methode unterstützt kreisförmige Strings und Kurven.

 Diese Funktion unterstützt polyedrische Flächen.

 Diese Funktion unterstützt Dreiecke und dreieckige unregelmäßige Netzoberflächen (TIN).

 Diese Funktion unterstützt 3d und lässt den Z-Index nicht fallen.

Beispiele

3D LineStrings.

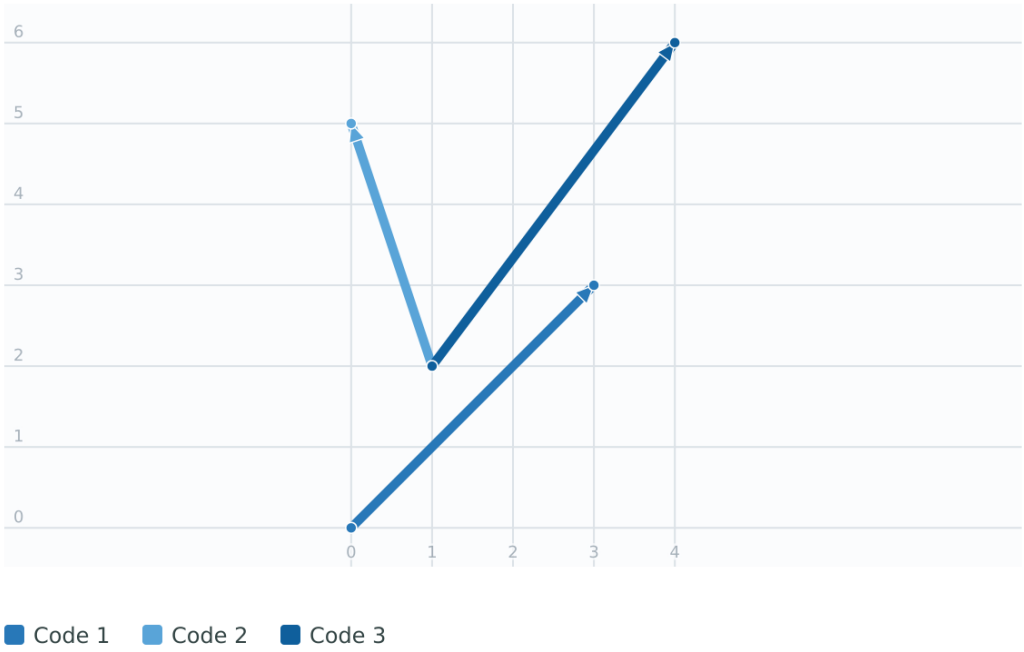
Code

```
SELECT tbl1.column1, tbl2.column1, tbl1.column2 &&& tbl2.column2 AS overlaps_3d,
                       tbl1.column2 && tbl2.column2 AS overlaps_2d
FROM ( VALUES
    (1, 'LINESTRING Z(0 0 1, 3 3 2)'::geometry),
    (2, 'LINESTRING Z(1 2 0, 0 5 -1)'::geometry)) AS tbl1,
( VALUES
    (3, 'LINESTRING Z(1 2 1, 4 6 1)'::geometry)) AS tbl2;
```

Output

```
 column1 | column1 | overlaps_3d | overlaps_2d
---------+---------+-------------+-------------
       1 |       3 | t           | t
       2 |       3 | f           | t
```

Figure



3M LineStrings.

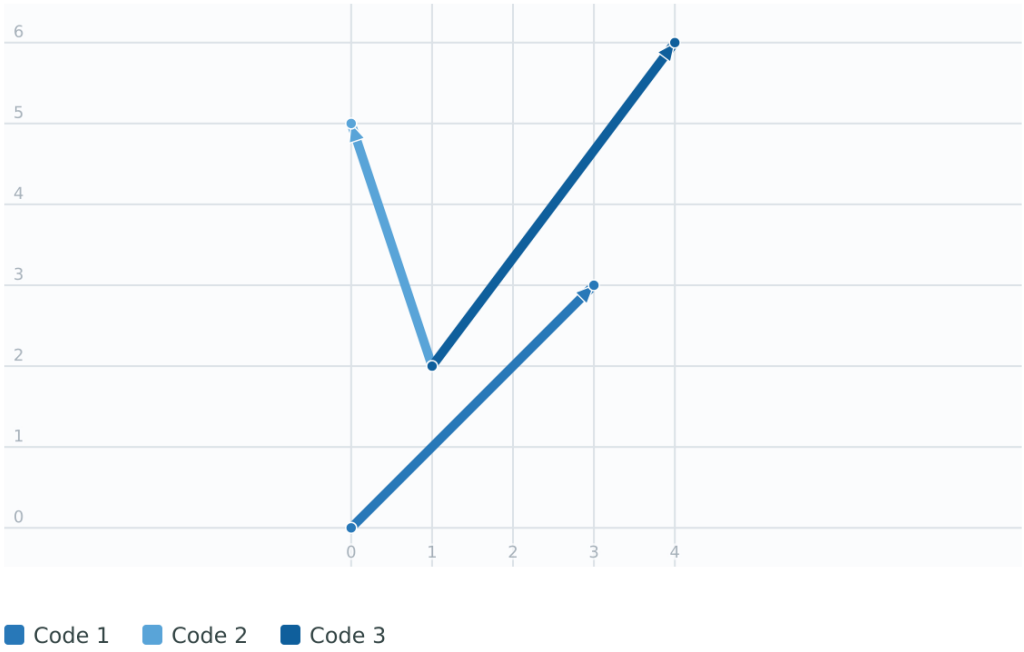
Code

```
SELECT tbl1.column1, tbl2.column1, tbl1.column2 &&& tbl2.column2 AS overlaps_3zm,
                     tbl1.column2 && tbl2.column2 AS overlaps_2d
FROM ( VALUES
    (1, 'LINESTRING M(0 0 1, 3 3 2)'::geometry),
    (2, 'LINESTRING M(1 2 0, 0 5 -1)'::geometry)) AS tbl1,
( VALUES
    (3, 'LINESTRING M(1 2 1, 4 6 1)'::geometry)) AS tbl2;
```

Output

```
 column1 | column1 | overlaps_3zm | overlaps_2d
---------+---------+--------------+-------------
       1 |       3 | t            | t
       2 |       3 | f            | t
```

Figure



Siehe auch

&&

7.10.1.6 &&&(geometry,gidx)

&&&(geometry,gidx) — Gibt `TRUE` zurück, wenn die (cached) n-D bounding box einer Geometrie eine n-D float precision bounding box (GIDX) schneidet.

Synopsis

boolean **&&&**(geometry A , gidx B);

Beschreibung

Der `&&&` Operator gibt `TRUE` zurück, wenn die zwischengespeicherte n-D bounding box der Geometrie A die n-D bounding box B, unter Benutzung von float precision, schneidet. D.h.: Wenn B eine (double precision) box3d ist, wird diese intern in eine float precision 3D bounding box (GIDX) umgewandelt



Anmerkung
Dieser Operator ist für die interne Nutzung durch BRIN Indizes, und nicht so sehr durch Anwender, vorgesehen.

Availability: 2.3.0 support for Block Range INdexes (BRIN) was introduced.

Diese Methode unterstützt kreisförmige Strings und Kurven.

Diese Funktion unterstützt polyedrische Flächen.

Diese Funktion unterstützt Dreiecke und dreieckige unregelmäßige Netzoberflächen (TIN).

Diese Funktion unterstützt 3d und lässt den Z-Index nicht fallen.

Beispiele

Code

```
SELECT ST_MakePoint(1,1,1) &&& ST_3DMakeBox(ST_MakePoint(0,0,0), ST_MakePoint(2,2,2)) AS ↩
    overlaps;
```

Output

```
 overlaps
----------
 t
(1 row)
```

Siehe auch

&&&(gidx,geometry), &&&(gidx,gidx)

7.10.1.7 &&&(gidx,geometry)

&&&(gidx,geometry) — Gibt `TRUE` zurück, wenn eine n-D float precision bounding box (GIDX) eine (cached) n-D bounding box einer Geometrie schneidet.

Synopsis

boolean **&&&**(gidx A , geometry B);

Beschreibung

Der `&&&` Operator gibt `TRUE` zurück, wenn die n-D bounding box A die cached n-D bounding box der Geometrie B, unter Benutzung von float precision, schneidet. D.h.: wenn A eine (double precision) box3d ist, wir diese intern in eine float precision 3D bounding box (GIDX) umgewandelt



Anmerkung
Dieser Operator ist für die interne Nutzung durch BRIN Indizes, und nicht so sehr durch Anwender, vorgesehen.

Availability: 2.3.0 support for Block Range INdexes (BRIN) was introduced.

Diese Methode unterstützt kreisförmige Strings und Kurven.

Diese Funktion unterstützt polyedrische Flächen.

Diese Funktion unterstützt Dreiecke und dreieckige unregelmäßige Netzoberflächen (TIN).

Diese Funktion unterstützt 3d und lässt den Z-Index nicht fallen.

Beispiele

Code

```
SELECT ST_3DMakeBox(ST_MakePoint(0,0,0), ST_MakePoint(2,2,2)) &&& ST_MakePoint(1,1,1) AS ↩
    overlaps;
```

Output

```
 overlaps
----------
 t
(1 row)
```

Siehe auch

&&&(geometry,gidx), &&&(gidx,gidx)

7.10.1.8 &&&(gidx,gidx)

&&&(gidx,gidx) — Gibt TRUE zurück, wenn sich zwei n-D float precision bounding boxes (GIDX) gegenseitig überschneiden.

Synopsis

boolean **&&&**(gidx A , gidx B);

Beschreibung

Der &&& Operator gibt TRUE zurück, wenn sich zwei n-D bounding boxes A und B, unter Benutzung von float precision, gegenseitig überschneiden. D.h.: wenn A (oder B) eine (double precision) box3d ist, wird diese intern in eine float precision 3D bounding box (GIDX) umgewandelt



Anmerkung
Dieser Operator ist für die interne Nutzung durch BRIN Indizes, und nicht so sehr durch Anwender, vorgesehen.

Availability: 2.3.0 support for Block Range INdexes (BRIN) was introduced.

Diese Methode unterstützt kreisförmige Strings und Kurven.

Diese Funktion unterstützt polyedrische Flächen.

Diese Funktion unterstützt Dreiecke und dreieckige unregelmäßige Netzoberflächen (TIN).

Diese Funktion unterstützt 3d und lässt den Z-Index nicht fallen.

Beispiele

Code

```
SELECT ST_3DMakeBox(ST_MakePoint(0,0,0), ST_MakePoint(2,2,2)) &&& ST_3DMakeBox(ST_MakePoint ↩
    (1,1,1), ST_MakePoint(3,3,3)) AS overlaps;
```

Output

```
 overlaps
----------
 t
(1 row)
```

Siehe auch

&&&(geometry,gidx), &&&(gidx,geometry)

7.10.1.9 &<

&< — Gibt TRUE zurück, wenn die bounding box der Geometrie A, die bounding box der Geometrie B überlagert oder links davon liegt.

Synopsis

boolean **&<**(geometry A , geometry B);

Beschreibung

Der &< Operator gibt TRUE zurück, wenn die bounding box der Geometrie A die bounding box der Geometrie B überlagert oder links davon liegt, oder präziser, überlagert und NICHT rechts von der bounding box der Geometrie B liegt.



Anmerkung
Dieser Operand benützt jeden Index, der für die Geometrie zur Verfügung stellt.

Beispiele

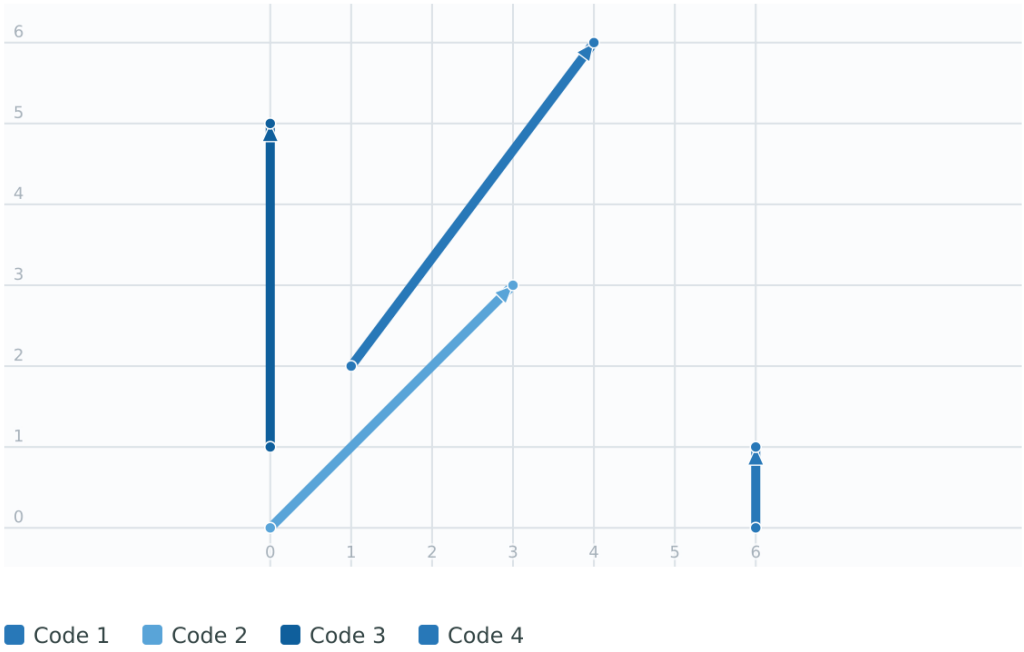
Code

```
SELECT tbl1.column1, tbl2.column1, tbl1.column2 &< tbl2.column2 AS overleft
FROM
  ( VALUES
    (1, 'LINESTRING(1 2, 4 6)'::geometry)) AS tbl1,
  ( VALUES
    (2, 'LINESTRING(0 0, 3 3)'::geometry),
    (3, 'LINESTRING(0 1, 0 5)'::geometry),
    (4, 'LINESTRING(6 0, 6 1)'::geometry)) AS tbl2;
```

Output

```
 column1 | column1 | overleft
---------+---------+----------
       1 |       2 | f
       1 |       3 | f
       1 |       4 | t
(3 rows)
```

Figure



Siehe auch

&&, |&>, &>, &<|

7.10.1.10 &<|

&<| — Gibt `TRUE` zurück, wenn die bounding box von A jene von B überlagert oder unterhalb liegt.

Synopsis

boolean **&<|**(geometry A , geometry B);

Beschreibung

Der `&<|` Operator gibt `TRUE` zurück, wenn die Bounding Box der Geometrie A die Bounding Box der Geometrie B überlagert oder unterhalb liegt, oder präziser, überlagert oder NICHT oberhalb der Bounding der Geometrie B liegt.

Diese Methode unterstützt kreisförmige Strings und Kurven.

Diese Funktion unterstützt polyedrische Flächen.

Anmerkung
Dieser Operand benützt jeden Index, der für die Geometrie zur Verfügung stellt.

Beispiele

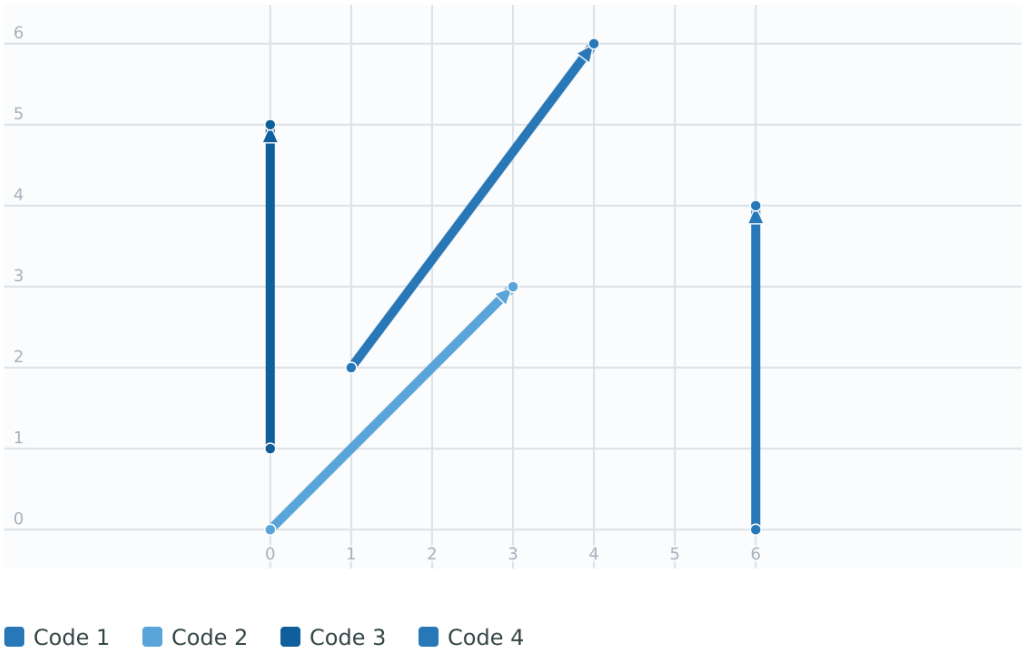
Code

```
SELECT tbl1.column1, tbl2.column1, tbl1.column2 &<| tbl2.column2 AS overbelow
FROM
  ( VALUES
    (1, 'LINESTRING(6 0, 6 4)'::geometry)) AS tbl1,
  ( VALUES
    (2, 'LINESTRING(0 0, 3 3)'::geometry),
    (3, 'LINESTRING(0 1, 0 5)'::geometry),
    (4, 'LINESTRING(1 2, 4 6)'::geometry)) AS tbl2;
```

Output

```
 column1 | column1 | overbelow
---------+---------+-----------
       1 |       2 | f
       1 |       3 | t
       1 |       4 | t
(3 rows)
```

Figure



Siehe auch

&&, |&>, &>, &<

7.10.1.11 &>

&> — Gibt TRUE zurück, wenn die Bounding Box von A jene von B überlagert oder rechts davon liegt.

Synopsis

boolean **&>**(geometry A , geometry B);

Beschreibung

Der `&>` Operator gibt `TRUE` zurück, wenn die Bounding Box der Geometrie A die Bounding Box der Geometrie B überlagert oder rechts von ihr liegt, oder präziser, überlagert und NICHT links von der Bounding Box der Geometrie B liegt.



Anmerkung
Dieser Operand benützt jeden Index, der für die Geometrie zur Verfügung stellt.

Beispiele

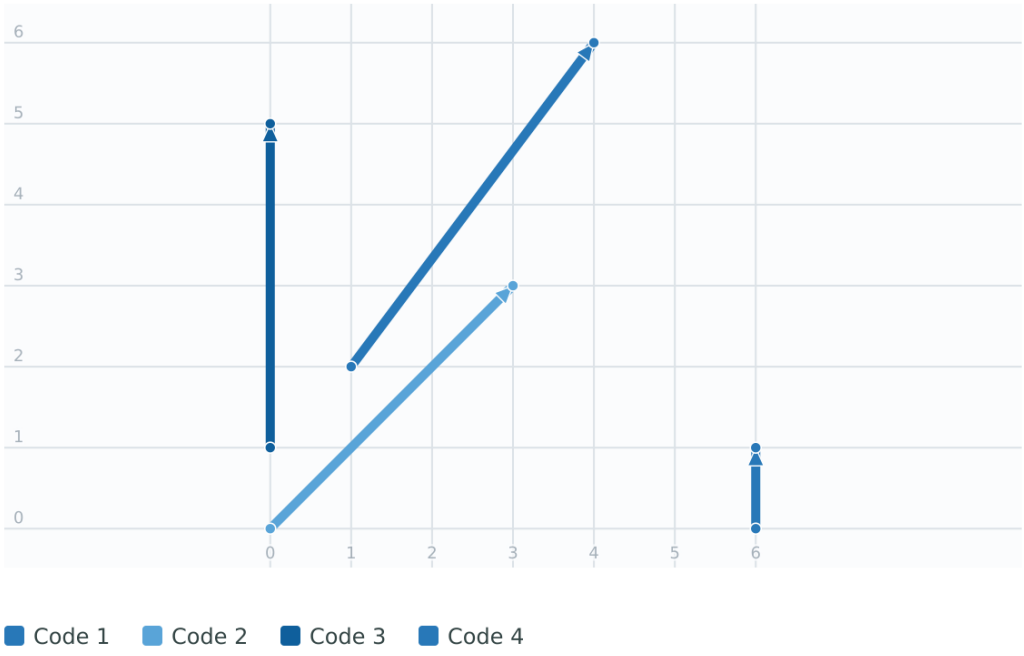
Code

```
SELECT tbl1.column1, tbl2.column1, tbl1.column2 &
> tbl2.column2 AS overright
FROM
  ( VALUES
    (1, 'LINESTRING(1 2,4 6)'::geometry)) AS tbl1,
  ( VALUES
    (2, 'LINESTRING(0 0,3 3)'::geometry),
    (3, 'LINESTRING(0 1,0 5)'::geometry),
    (4, 'LINESTRING(6 0,6 1)'::geometry)) AS tbl2;
```

Output

```
 column1 | column1 | overright
---------+---------+-----------
       1 |       2 | t
       1 |       3 | t
       1 |       4 | f
(3 rows)
```

Figure



Siehe auch

&&, |&>, &<|, &<

7.10.1.12 <<

<< — Gibt TRUE zurück, wenn die Bounding Box von A zur Gänze links von der von B liegt.

Synopsis

boolean **<<**(geometry A , geometry B);

Beschreibung

Der << Operator gibt TRUE zurück, wenn die Bounding Box der Geometrie A zur Gänze links der Bounding Box der Geometrie B liegt.



Anmerkung
Dieser Operand benützt jeden Index, der für die Geometrie zur Verfügung stellt.

Beispiele

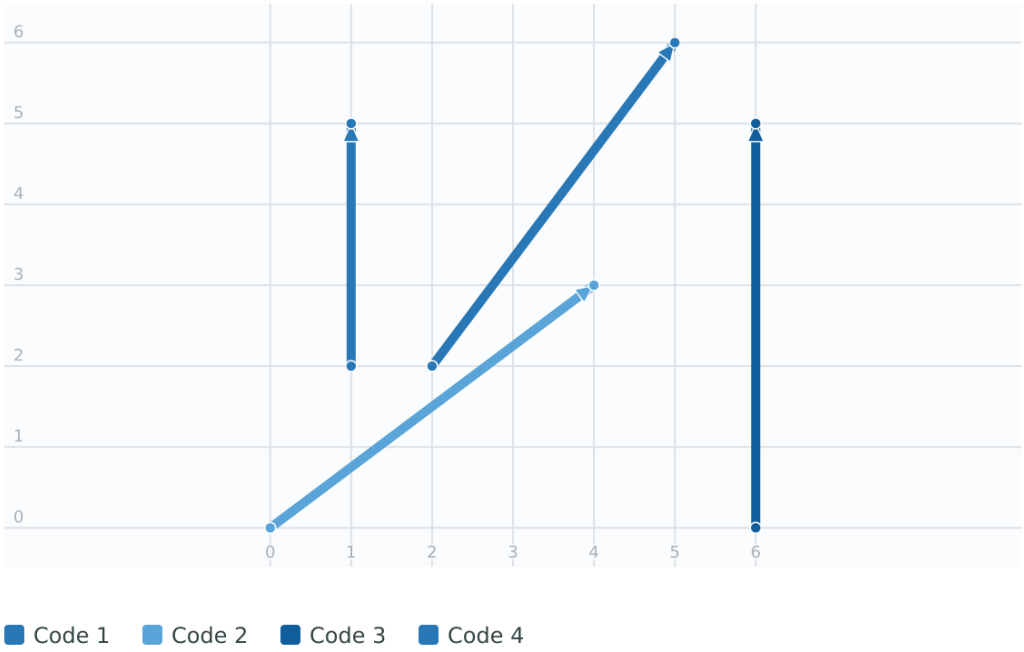
Code

```
SELECT tbl1.column1, tbl2.column1, tbl1.column2 << tbl2.column2 AS left
FROM
  ( VALUES
    (1, 'LINESTRING (1 2, 1 5)'::geometry)) AS tbl1,
  ( VALUES
    (2, 'LINESTRING (0 0, 4 3)'::geometry),
    (3, 'LINESTRING (6 0, 6 5)'::geometry),
    (4, 'LINESTRING (2 2, 5 6)'::geometry)) AS tbl2;
```

Output

```
 column1 | column1 | left
---------+---------+------
       1 |       2 | f
       1 |       3 | t
       1 |       4 | t
(3 rows)
```

Figure



Siehe auch

>>, |>>, <<|

7.10.1.13 <<|

<<| — Gibt `TRUE` zurück, wenn A's Bounding Box zur Gänze unterhalb von der von B liegt.

Synopsis

boolean **<<|**(geometry A , geometry B);

Beschreibung

Der `<<|` Operator gibt `TRUE` zurück, wenn die Bounding Box der Geometrie A zur Gänze unterhalb der Bounding Box von Geometrie B liegt.



Anmerkung
Dieser Operand benützt jeden Index, der für die Geometrie zur Verfügung stellt.

Beispiele

Code

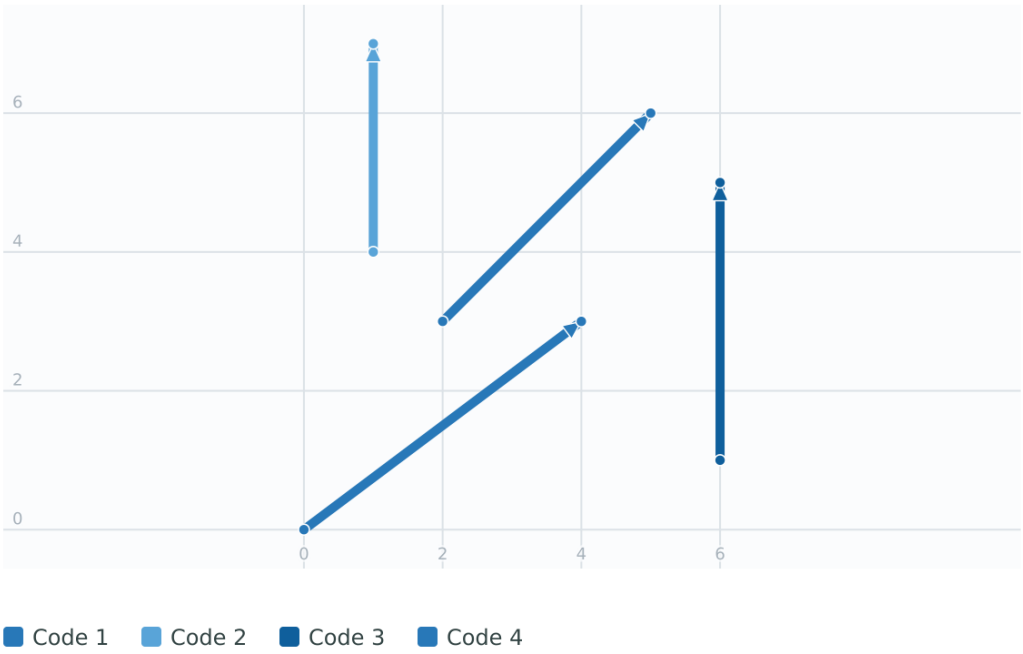
```
SELECT tbl1.column1, tbl2.column1, tbl1.column2 <<| tbl2.column2 AS below
FROM
  ( VALUES
    (1, 'LINESTRING (0 0, 4 3)'::geometry)) AS tbl1,
```

```
  ( VALUES
    (2, 'LINESTRING (1 4, 1 7)'::geometry),
    (3, 'LINESTRING (6 1, 6 5)'::geometry),
    (4, 'LINESTRING (2 3, 5 6)'::geometry)) AS tbl2;
```

Output

```
 column1 | column1 | below
---------+---------+-------
       1 |       2 | t
       1 |       3 | f
       1 |       4 | f
(3 rows)
```

Figure



Siehe auch

<<, >>, |>>

7.10.1.14 =

= — Gibt `TRUE` zurück, wenn die Koordinaten und die Reihenfolge der Koordinaten der Geometrie/Geographie A und der Geometrie/Geographie B ident sind.

Synopsis

boolean =(geometry A , geometry B);
boolean =(geography A , geography B);

Beschreibung

Der Operator = gibt TRUE zurück, wenn die Koordinaten und die Reihenfolge der Koordinaten der Geometrie/Geographie A und der Geometrie/Geographie B ident sind. PostgreSQL verwendet die =, <, und > Operatoren um die interne Sortierung und den Vergleich von Geometrien durchzuführen (z.B.: in einer GROUP BY oder ORDER BY Klausel).



Anmerkung
Nur die Geometrie/Geographie die in allen Gesichtspunkten übereinstimmt, d.h. mit den selben Koordinaten in der gleichen Reihenfolge, werden von diesem Operator als gleich betrachtet. Für "räumliche Gleichheit", bei der Dinge wie die Reihenfolge der Koordinaten außer Acht gelassen werden, und die es ermöglicht Geoobjekte zu erfassen, die denselben räumlichen Bereich mit unterschiedlicher Darstellung abdecken, verwenden Sie bitte ST_OrderingEquals oder ST_Equals



Achtung
Dieser Operator verwendet NICHT die Indizes, welche für die Geometrien vorhanden sind. Um eine Überprüfung auf exakte Gleichheit indexgestützt durchzuführen, kombinieren Sie bitte = mit &&.

Änderung: 2.4.0, in Vorgängerversionen war dies die Gleichheit der umschreibenden Rechtecke, nicht die geometrische Gleichheit. Falls Sie auf Gleichheit der umschreibenden Rechtecke prüfen wollen, verwenden Sie stattdesse bitte ~=.

 Diese Methode unterstützt kreisförmige Strings und Kurven.

 Diese Funktion unterstützt polyedrische Flächen.

Beispiele

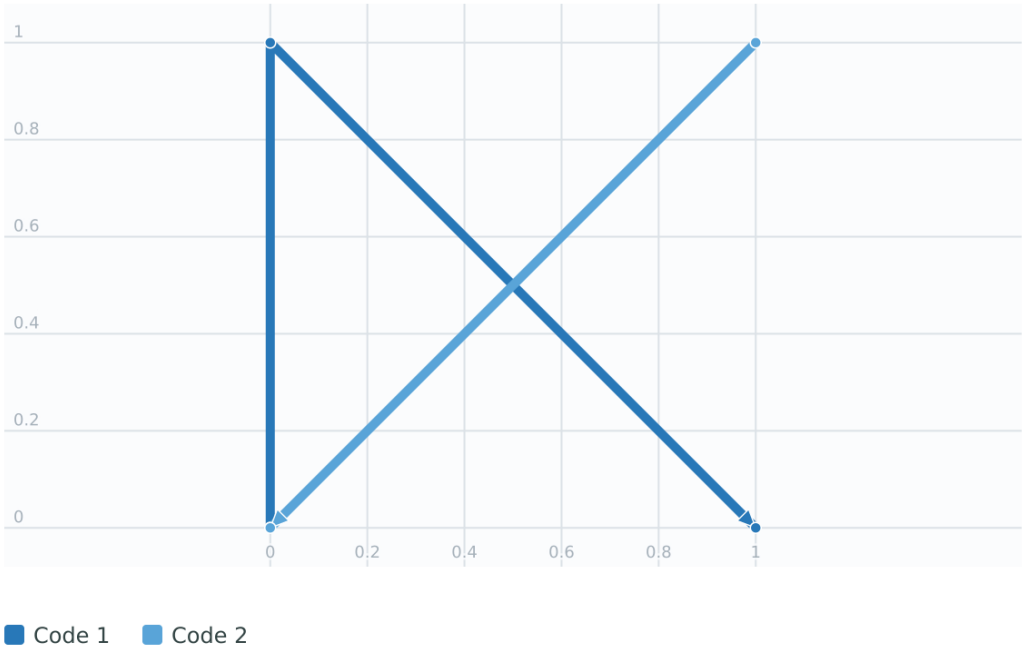
Code

```
SELECT 'LINESTRING(0 0,0 1,1 0)'::geometry = 'LINESTRING(1 1,0 0)'::geometry;
```

Output

```
 ?column?
----------
 f
(1 row)
```

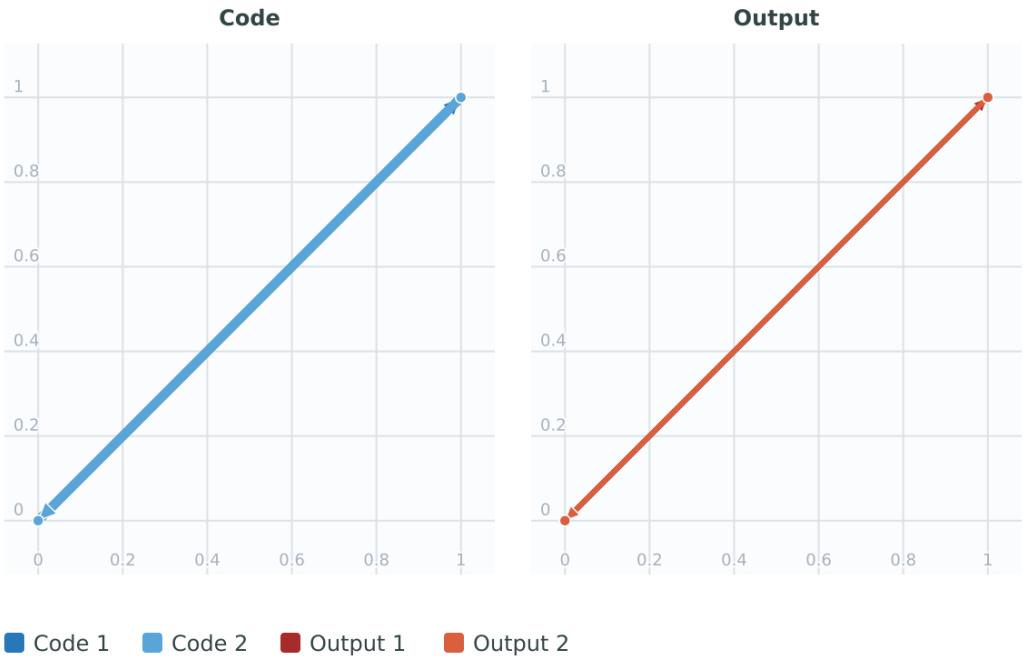
Figure



Code

```
SELECT column1
FROM ( VALUES
    ('LINESTRING(0 0, 1 1)'::geometry),
    ('LINESTRING(1 1, 0 0)'::geometry)) AS foo;
```

Figure

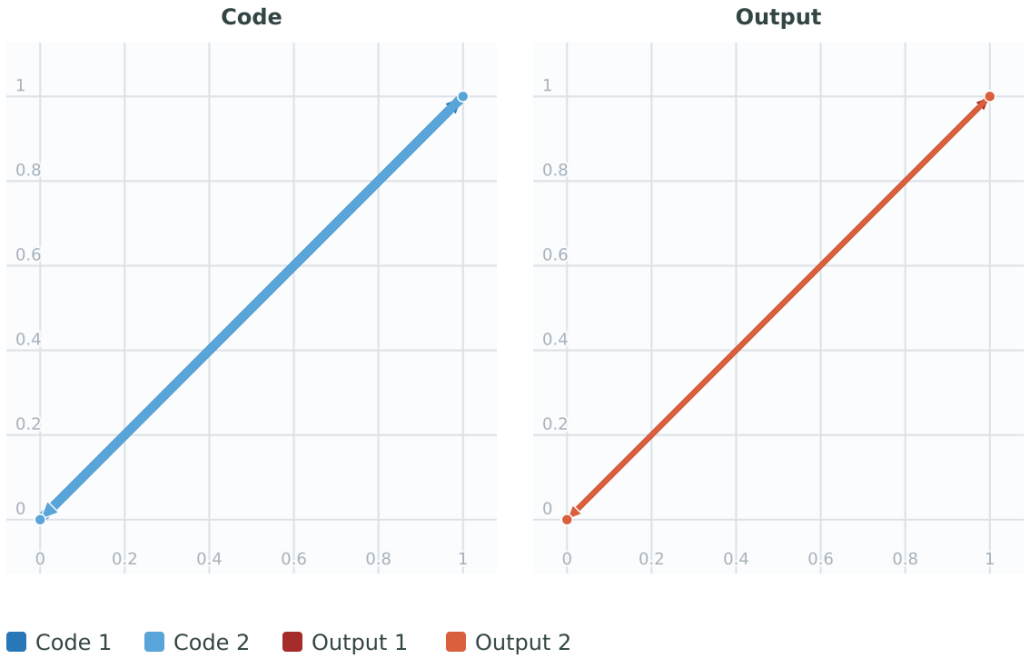


Note: the GROUP BY uses the "=" to compare for geometry equivalency.

Code

```
SELECT column1
FROM ( VALUES
    ('LINESTRING(0 0, 1 1)'::geometry),
    ('LINESTRING(1 1, 0 0)'::geometry)) AS foo
GROUP BY column1;
```

Figure



In versions prior to 2.0, this used to return true.

Code

```
SELECT 'POINT(1707296.37 4820536.77)'::geometry =
   'POINT(1707296.27 4820536.87)'::geometry As pt_intersect;
```

Output

```
 pt_intersect
--------------
 f
(1 row)
```

Siehe auch

ST_Equals, ST_OrderingEquals, ~=

7.10.1.15 >>

>> — Gibt TRUE zurück, wenn A's bounding box zur Gänze rechts von der von B liegt.

Synopsis

boolean **>>**(geometry A , geometry B);

Beschreibung

Der >> Operator gibt `TRUE` zurück, wenn die Bounding Box von Geometrie A zur Gänze rechts der Bounding Box von Geometrie B liegt.



Anmerkung
Dieser Operand benützt jeden Index, der für die Geometrie zur Verfügung stellt.

Beispiele

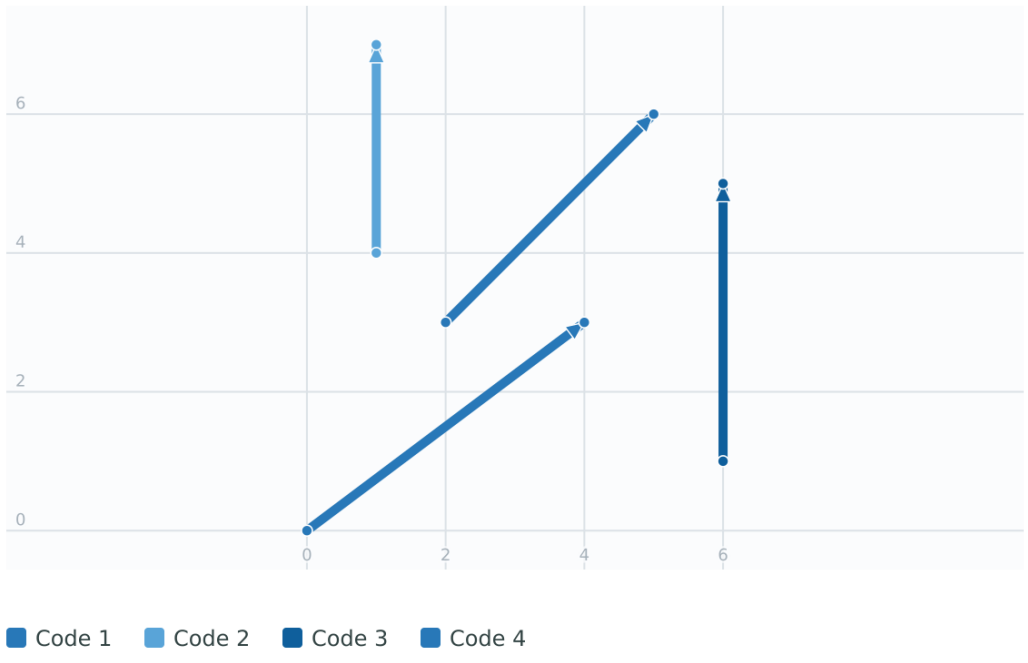
Code

```
SELECT tbl1.column1, tbl2.column1, tbl1.column2
>
> tbl2.column2 AS right
FROM
  ( VALUES
    (1, 'LINESTRING (2 3,5 6)'::geometry)) AS tbl1,
  ( VALUES
    (2, 'LINESTRING (1 4,1 7)'::geometry),
    (3, 'LINESTRING (6 1,6 5)'::geometry),
    (4, 'LINESTRING (0 0,4 3)'::geometry)) AS tbl2;
```

Output

```
 column1 | column1 | right
---------+---------+-------
       1 |       2 | t
       1 |       3 | f
       1 |       4 | f
(3 rows)
```

Figure



Siehe auch

<<, |>>, <<|

7.10.1.16 @

@ — Gibt TRUE zurück, wenn die Bounding Box von A in jener von B enthalten ist.

Synopsis

boolean @(geometry A , geometry B);

Beschreibung

Der @ Operator gibt TRUE zurück, wenn die Bounding Box der Geometrie A vollstänig in der Bounding Box der Geometrie B enthalten ist.



Anmerkung
Dieser Operand benützt jeden Index, der für die Geometrie zur Verfügung stellt.

Beispiele

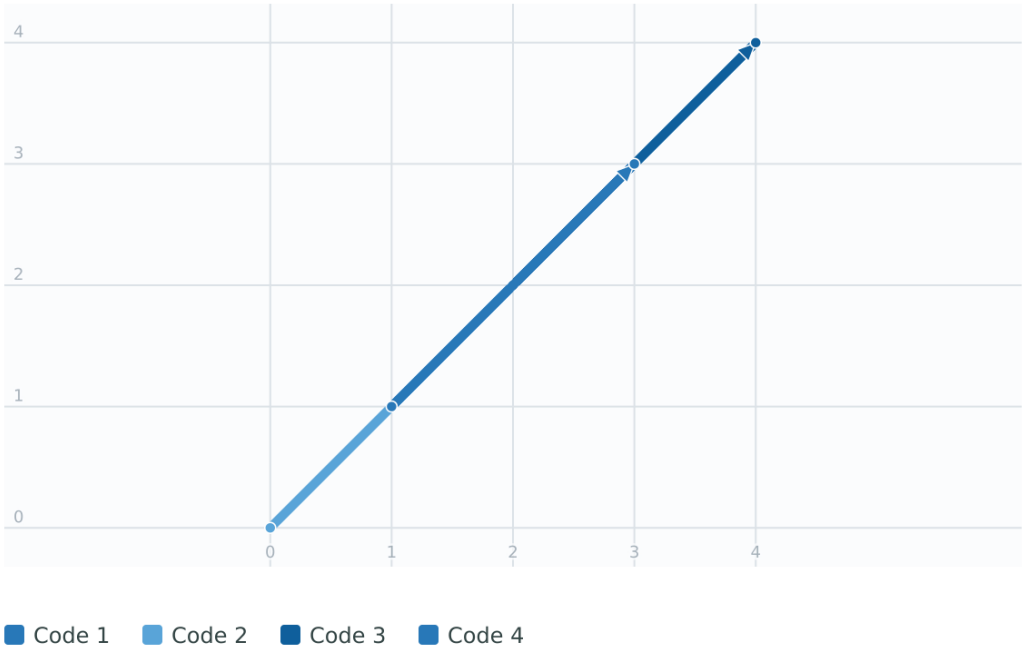
Code

```
SELECT tbl1.column1, tbl2.column1, tbl1.column2 @ tbl2.column2 AS contained
FROM
  ( VALUES
    (1, 'LINESTRING (1 1, 3 3)'::geometry)) AS tbl1,
  ( VALUES
    (2, 'LINESTRING (0 0, 4 4)'::geometry),
    (3, 'LINESTRING (2 2, 4 4)'::geometry),
    (4, 'LINESTRING (1 1, 3 3)'::geometry)) AS tbl2;
```

Output

```
 column1 | column1 | contained
---------+---------+-----------
       1 |       2 | t
       1 |       3 | f
       1 |       4 | t
(3 rows)
```

Figure



Siehe auch

~, &&

7.10.1.17 @(geometry,box2df)

@(geometry,box2df) — Gibt `TRUE` zurück, wenn die 2D Bounding Box einer Geometrie in einer 2D float precision Bbounding Box (BOX2DF) enthalten ist.

Synopsis

boolean @(geometry A , box2df B);

Beschreibung

Der `@` Operator gibt `TRUE` zurück, wenn die 2D Bounding Box der Geometrie A in der 2D Bounding Box der Geometrie B , unter Benutzung von float precision, enthalten ist. D.h.: wenn B eine (double precision) box2d ist, wird diese intern in eine float precision 2D bounding box (BOX2DF) übersetzt.



Anmerkung
Dieser Operand ist eher für die interne Nutzung durch BRIN Indizes, als durch die Anwender, gedacht.

Availability: 2.3.0 support for Block Range INdexes (BRIN) was introduced.

 Diese Methode unterstützt kreisförmige Strings und Kurven.

 Diese Funktion unterstützt polyedrische Flächen.

Beispiele

Code

```
SELECT ST_Buffer('POINT(2 2)'::geometry, 1) @ ST_MakeBox2D(ST_Point(0, 0), ST_Point(5, 5))  ↩
    AS is_contained;
```

Output

```
t
```

Siehe auch

&&(geometry,box2df), &&(box2df,geometry), &&(box2df,box2df), ~(geometry,box2df), ~(box2df,geometry), ~(box2df,box2df), @(box2df,geometry), @(box2df,box2df)

7.10.1.18 @(box2df,geometry)

@(box2df,geometry) — Gibt `TRUE` zurück, wenn eine 2D float precision bounding box (BOX2DF) in der 2D Bounding Box einer Geometrie enthalten ist..

Synopsis

boolean @(box2df A , geometry B);

Beschreibung

Der `@` Operator gibt `TRUE` zurück, wenn die 2D bounding box A in der 2D bounding box der Geometrie B, unter Verwendung von float precision, enthalten ist. D.h.: wenn B eine (double precision) box2d ist, wird diese intern in eine float precision 2D bounding box (BOX2DF) umgewandelt



Anmerkung
Dieser Operand ist eher für die interne Nutzung durch BRIN Indizes, als durch die Anwender, gedacht.

Availability: 2.3.0 support for Block Range INdexes (BRIN) was introduced.

Diese Methode unterstützt kreisförmige Strings und Kurven.

Diese Funktion unterstützt polyedrische Flächen.

Beispiele

Code

```
SELECT ST_MakeBox2D(ST_Point(2, 2), ST_Point(3, 3)) @ ST_Buffer('POINT(1 1)'::geometry, 10)  ↩
    AS is_contained;
```

Output

```
t
```

Siehe auch

&&(geometry,box2df), &&(box2df,geometry), &&(box2df,box2df), ~(geometry,box2df), ~(box2df,geometry), ~(box2df,box2df), @(geometry,box2df), @(box2df,box2df)

7.10.1.19 @(box2df,box2df)

@(box2df,box2df) — Gibt `TRUE` zurück, wenn eine 2D float precision bounding box (BOX2DF) innerhalb einer anderen 2D float precision bounding box enthalten ist.

Synopsis

boolean @(box2df A , box2df B);

Beschreibung

Der `@` Operator gibt `TRUE` zurück, wenn die 2D bounding box A innerhalb der 2D bounding box B, unter Verwendung von float precision, enthalten ist. D.h.: wenn A (oder B) eine (double precision) box2d ist, wird diese intern in eine float precision 2D bounding box (BOX2DF) umgewandelt.



Anmerkung
Dieser Operand ist eher für die interne Nutzung durch BRIN Indizes, als durch die Anwender, gedacht.

Availability: 2.3.0 support for Block Range INdexes (BRIN) was introduced.


Diese Methode unterstützt kreisförmige Strings und Kurven.


Diese Funktion unterstützt polyedrische Flächen.

Beispiele

Code

```
SELECT ST_MakeBox2D(ST_Point(2, 2), ST_Point(3, 3)) @ ST_MakeBox2D(ST_Point(0, 0), ST_Point ↩
    (5, 5)) AS is_contained;
```

Output

```
t
```

Siehe auch

&&(geometry,box2df), &&(box2df,geometry), &&(box2df,box2df), ~(geometry,box2df), ~(box2df,geometry), ~(box2df,box2df), @(geometry,box2df), @(box2df,geometry)

7.10.1.20 |&>

|&> — Gibt `TRUE` zurück, wenn A's bounding box diejenige von B überlagert oder oberhalb von B liegt.

Synopsis

boolean **|&>**(geometry A , geometry B);

Beschreibung

Der `|&>` Operator gibt `TRUE` zurück, wenn die bounding box der Geometrie A die bounding box der Geometrie B überlagert oder oberhalb liegt, oder präziser, überlagert oder NICHT unterhalb der Bounding Box der Geometrie B liegt.



Anmerkung
Dieser Operand benützt jeden Index, der für die Geometrie zur Verfügung stellt.

Beispiele

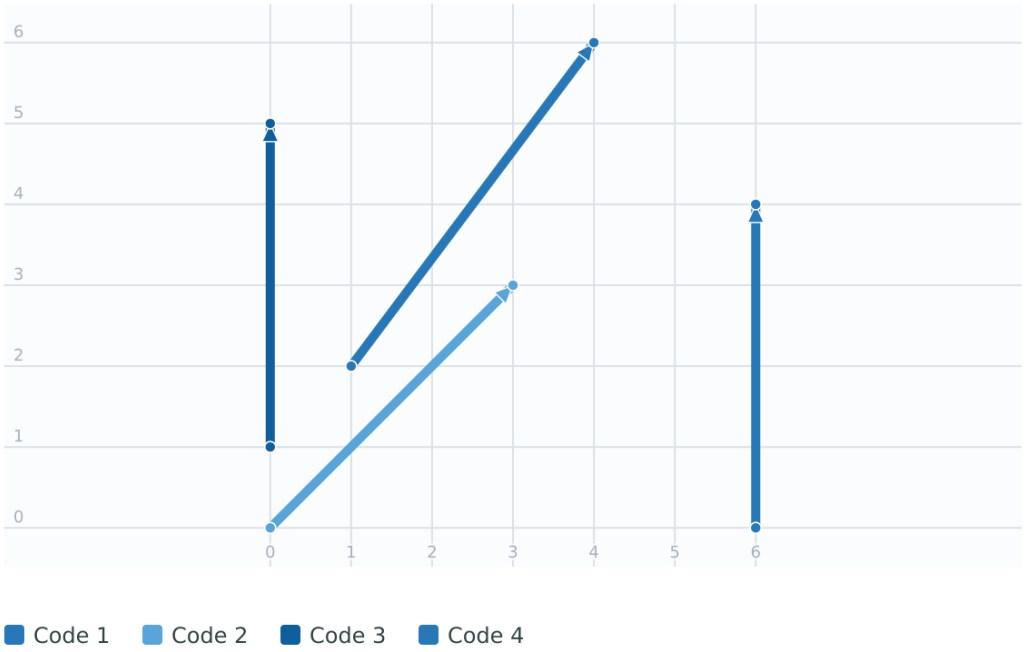
Code

```
SELECT tbl1.column1, tbl2.column1, tbl1.column2 |&
> tbl2.column2 AS overabove
FROM
  ( VALUES
    (1, 'LINESTRING(6 0,6 4)'::geometry)) AS tbl1,
  ( VALUES
    (2, 'LINESTRING(0 0,3 3)'::geometry),
    (3, 'LINESTRING(0 1,0 5)'::geometry),
    (4, 'LINESTRING(1 2,4 6)'::geometry)) AS tbl2;
```

Output

```
 column1 | column1 | overabove
---------+---------+-----------
       1 |       2 | t
       1 |       3 | f
       1 |       4 | f
(3 rows)
```

Figure



Siehe auch

&&, &>, &<|, &<

7.10.1.21 |>>

|>> — Gibt TRUE zurück, wenn A's bounding box is zur Gänze oberhalb der von B liegt.

Synopsis

boolean **|>>**(geometry A , geometry B);

Beschreibung

Der Operator |>> gibt TRUE zurück, wenn die Bounding Box der Geometrie A zur Gänze oberhalb der Bounding Box von Geometrie B liegt.



Anmerkung
Dieser Operand benützt jeden Index, der für die Geometrie zur Verfügung stellt.

Beispiele

Code

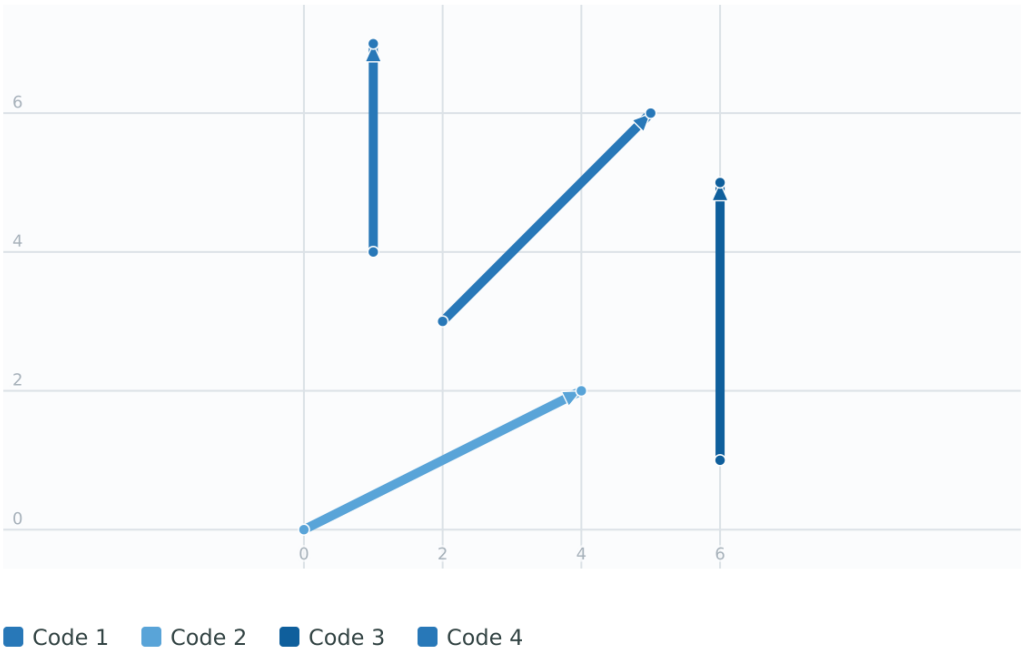
```
SELECT tbl1.column1, tbl2.column1, tbl1.column2 |>> tbl2.column2 AS above
FROM
  ( VALUES
    (1, 'LINESTRING (1 4,1 7)'::geometry)) AS tbl1,
```

```
  ( VALUES
    (2, 'LINESTRING (0 0,4 2)'::geometry),
    (3, 'LINESTRING (6 1,6 5)'::geometry),
    (4, 'LINESTRING (2 3,5 6)'::geometry)) AS tbl2;
```

Output

```
column1 | column1 | above
---------+---------+-------
       1 |       2 | t
       1 |       3 | f
       1 |       4 | f
(3 rows)
```

Figure



Siehe auch

<<, >>, <<|

7.10.1.22 ~

~ — Gibt TRUE zurück, wenn A's bounding box die von B enthält.

Synopsis

boolean ~(geometry A , geometry B);

Beschreibung

Der ~ Operator gibt TRUE zurück, wenn die bounding box der Geometrie A zur Gänze die bounding box der Geometrie B enthält.



Anmerkung
Dieser Operand benützt jeden Index, der für die Geometrie zur Verfügung stellt.

Beispiele

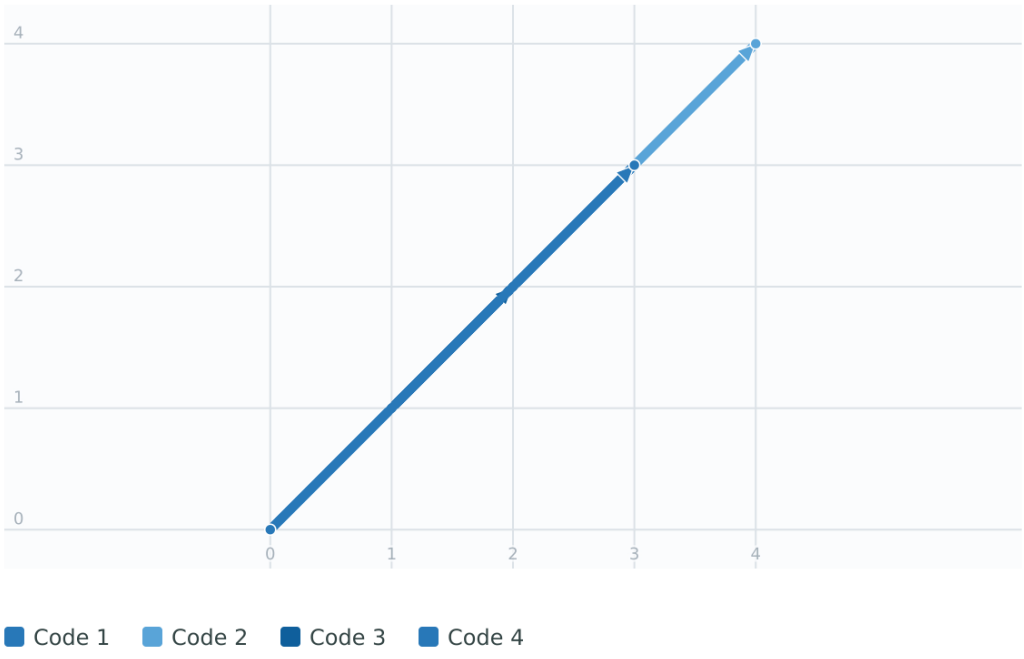
Code

```
SELECT tbl1.column1, tbl2.column1, tbl1.column2 ~ tbl2.column2 AS contains
FROM
  ( VALUES
    (1, 'LINESTRING (0 0,3 3)'::geometry)) AS tbl1,
  ( VALUES
    (2, 'LINESTRING (0 0,4 4)'::geometry),
    (3, 'LINESTRING (1 1,2 2)'::geometry),
    (4, 'LINESTRING (0 0,3 3)'::geometry)) AS tbl2;
```

Output

```
column1 | column1 | contains
---------+---------+----------
       1 |       2 | f
       1 |       3 | t
       1 |       4 | t
(3 rows)
```

Figure



Siehe auch

@, &&

7.10.1.23 ~(geometry,box2df)

~(geometry,box2df) — Gibt TRUE zurück, wenn die 2D bounding box einer Geometrie eine 2D float precision bounding box (GIDX) enthält.

Synopsis

boolean ~(geometry A , box2df B);

Beschreibung

Der ~ Operator gibt TRUE zurück, wenn die 2D bounding box einer Geometrie A die 2D bounding box B, unter Verwendung von float precision, enthält. D.h.: wenn B eine (double precision) box2d ist, wird diese intern in eine float precision 2D bounding box (BOX2DF) übersetzt



Anmerkung
Dieser Operand ist eher für die interne Nutzung durch BRIN Indizes, als durch die Anwender, gedacht.

Availability: 2.3.0 support for Block Range INdexes (BRIN) was introduced.


Diese Methode unterstützt kreisförmige Strings und Kurven.


Diese Funktion unterstützt polyedrische Flächen.

Beispiele

Code

```
SELECT ST_Buffer('POINT(1 1)'::geometry, 10) ~ ST_MakeBox2D(ST_Point(0, 0), ST_Point(2, 2))  ↩
    AS contains;
```

Output

```
t
```

Siehe auch

&&(geometry,box2df), &&(box2df,geometry), &&(box2df,box2df), ~(box2df,geometry), ~(box2df,box2df), @(geometry,box2df), @(box2df,geometry), @(box2df,box2df)

7.10.1.24 ~(box2df,geometry)

~(box2df,geometry) — Gibt TRUE zurück, wenn eine 2D float precision bounding box (BOX2DF) die 2D Bounding Box einer Geometrie enthält.

Synopsis

boolean ~(box2df A , geometry B);

Beschreibung

Der ~ Operator gibt `TRUE` zurück, wenn die 2D bounding box A die Bounding Box der Geometrie B, unter Verwendung von float precision, enthält. D.h.: wenn A eine (double precision) box2d ist, wird diese intern in eine float precision 2D bounding box (BOX2DF) umgewandelt.



Anmerkung
Dieser Operand ist eher für die interne Nutzung durch BRIN Indizes, als durch die Anwender, gedacht.

Availability: 2.3.0 support for Block Range INdexes (BRIN) was introduced.


Diese Methode unterstützt kreisförmige Strings und Kurven.


Diese Funktion unterstützt polyedrische Flächen.

Beispiele

Code

```
SELECT ST_MakeBox2D(ST_Point(0, 0), ST_Point(5, 5)) ~ ST_Buffer('POINT(2 2)'::geometry, 1)  ↩
    AS contains;
```

Output

```
t
```

Siehe auch

&&(geometry,box2df), &&(box2df,geometry), &&(box2df,box2df), ~(geometry,box2df), ~(box2df,box2df), @(geometry,box2df), @(box2df,geometry), @(box2df,box2df)

7.10.1.25 ~(box2df,box2df)

~(box2df,box2df) — Gibt `TRUE` zurück, wenn eine 2D float precision bounding box (BOX2DF) eine andere 2D float precision bounding box (BOX2DF) enthält.

Synopsis

boolean **~**(box2df A , box2df B);

Beschreibung

Der ~ Operator gibt `TRUE` zurück, wenn die 2D bounding box A die 2D bounding box B, unter Verwendung von float precision, enthält. D.h.: wenn A eine (double precision) box2d ist, wird diese intern in eine float precision 2D bounding box (BOX2DF) umgewandelt



Anmerkung
Dieser Operand ist eher für die interne Nutzung durch BRIN Indizes, als durch die Anwender, gedacht.

Availability: 2.3.0 support for Block Range INdexes (BRIN) was introduced.

Diese Methode unterstützt kreisförmige Strings und Kurven.

Diese Funktion unterstützt polyedrische Flächen.

Beispiele

Code

```
SELECT ST_MakeBox2D(ST_Point(0, 0), ST_Point(5, 5)) ~ ST_MakeBox2D(ST_Point(2, 2), ST_Point ↩
    (3, 3)) AS contains;
```

Output

```
t
```

Siehe auch

&&(geometry,box2df), &&(box2df,geometry), &&(box2df,box2df), ~(geometry,box2df), ~(box2df,geometry), @(geometry,box2df), @(box2df,geometry), @(box2df,box2df)

7.10.1.26 ~=

~= — Gibt TRUE zurück, wenn die bounding box von A ident mit jener von B ist.

Synopsis

boolean **~=**(geometry A , geometry B);

Beschreibung

Der ~= Operator gibt TRUE zurück, wenn die bounding box der Geometrie/Geographie A ident mit der bounding box der Geometrie/Geographie B ist.



Anmerkung
Dieser Operand benützt jeden Index, der für die Geometrie zur Verfügung stellt.

Verfügbarkeit: 1.5.0 "Verhaltensänderung"

Diese Funktion unterstützt polyedrische Flächen.



Warnung
Dieser Operator verhält sich ab PostGIS 1.5 insofern anders, als er vom Prüfen der Übereinstimmung der tatsächlichen Geometrie auf eine ledigliche Überprüfung der Gleichheit der Bounding Boxes abgeändert wurde. Um die Sache noch weiter zu komplizieren, hängt dieses Verhalten der Datenbank davon ab, ob ein hard oder soft upgrade durchgeführt wurde. Um herauszufinden, wie sich die Datenbank in dieser Beziehung verhält, führen Sie bitte die untere Abfrage aus. Um auf exakte Gleichheit zu prüfen benutzen Sie bitte ST_OrderingEquals oder ST_Equals.

Beispiele

Code

```
SELECT 'LINESTRING(0 0,1 1)'::geometry ~= 'LINESTRING(0 1,1 0)'::geometry AS equality;
```

Output

```
t
```

Figure



Siehe auch

ST_Equals, ST_OrderingEquals, =

7.10.2 Operatoren

7.10.2.1 <->

<-> — Gibt die 2D Entfernung zwischen A und B zurück.

Synopsis

double precision **<->**(geometry A , geometry B);
double precision **<->**(geography A , geography B);

Beschreibung

The <-> operator returns the 2D distance between two geometries. Used in the "ORDER BY" clause provides index-assisted nearest-neighbor result sets with true KNN distance search for geometries, and distance on the sphere for geographies.



Anmerkung
Dieser Operand verwendet 2D GiST Indizes, falls diese für die Geometrien vorhanden sind. Er unterscheidet sich insofern von anderen Operatoren, die räumliche Indizes verwenden, indem der räumliche Index nur dann verwendet wird, wenn sich der Operator in einer ORDER BY Klausel befindet.



Anmerkung
Der Index kommt nur zum Tragen, wenn eine der Geometrien eine Konstante ist (sich nicht in einer Subquery/CTE befindet). Z.B. 'SRID=3005;POINT(1011102 450541)'::geometry und nicht a.geom

Siehe OpenGeo workshop: Nearest-Neighbour Searching für ein praxisbezogenes Anwendungsbeispiel.

Enhanced: 2.2.0 -- True KNN ("K nearest neighbor") behavior for geometry and geography. Note for geography KNN is based on sphere rather than spheroid.

Changed: 2.2.0 -- Old hybrid-syntax workarounds may be slower once true KNN is available. 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.

Beispiele

Code

```
SELECT ST_Distance(geom, 'SRID=3005;POINT(1011102 450541)'::geometry) as d, edabbr, vaabbr
FROM va2005
ORDER BY d limit 10;
```

Output

```
d          | edabbr | vaabbr
-----------------+--------+--------
                 0 | ALQ    | 128
 5541.57712511724 | ALQ    | 129A
 5579.67450712005 | ALQ    | 001
  6083.4207708641 | ALQ    | 131
  7691.2205404848 | ALQ    | 003
 7900.75451037313 | ALQ    | 122
 8694.20710669982 | ALQ    | 129B
 9564.24289057111 | ALQ    | 130
  12089.665931705 | ALQ    | 127
 18472.5531479404 | ALQ    | 002
(10 rows)
```

Then the KNN raw answer:

Code

```
SELECT ST_Distance(geom, 'SRID=3005;POINT(1011102 450541)'::geometry) as d,edabbr, vaabbr
FROM va2005
ORDER BY geom <-> 'SRID=3005;POINT(1011102 450541)'::geometry limit 10;
```

Output

```
        d          | edabbr | vaabbr
-----------------+--------+--------
                 0 | ALQ    | 128
 5541.57712511724 | ALQ    | 129A
```

```
 5579.67450712005 | ALQ    | 001
  6083.4207708641 | ALQ    | 131
  7691.2205404848 | ALQ    | 003
 7900.75451037313 | ALQ    | 122
 8694.20710669982 | ALQ    | 129B
 9564.24289057111 | ALQ    | 130
  12089.665931705 | ALQ    | 127
 18472.5531479404 | ALQ    | 002
(10 rows)
```

Wenn Sie "EXPLAIN ANALYZE" an den zwei Abfragen ausführen, sollte eine Performance Verbesserung im Ausmaß von einer Sekunde auftreten.

When a plan cannot produce exact distances directly, 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:

Code

```
WITH index_query AS (
  SELECT ST_Distance(geom, 'SRID=3005;POINT(1011102 450541)'::geometry) as d,edabbr, vaabbr
    FROM va2005
  ORDER BY geom <-> 'SRID=3005;POINT(1011102 450541)'::geometry LIMIT 100)
  SELECT *
    FROM index_query
  ORDER BY d limit 10;
```

Output

```
        d         | edabbr | vaabbr
------------------+--------+--------
                0 | ALQ    | 128
 5541.57712511724 | ALQ    | 129A
 5579.67450712005 | ALQ    | 001
  6083.4207708641 | ALQ    | 131
  7691.2205404848 | ALQ    | 003
 7900.75451037313 | ALQ    | 122
 8694.20710669982 | ALQ    | 129B
 9564.24289057111 | ALQ    | 130
  12089.665931705 | ALQ    | 127
 18472.5531479404 | ALQ    | 002
(10 rows)
```

Siehe auch

ST_DWithin, ST_Distance, <#>

7.10.2.2 |=|

|=| — Gibt die Entfernung zwischen den Trajektorien A und B, am Ort der dichtesten Annäherung, an.

Synopsis

double precision **|=|**(geometry A , geometry B);

Beschreibung

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.



Anmerkung
Dieser Operand verwendet die ND GiST Indizes, welche für Geometrien vorhanden sein können. Er unterscheidet sich insofern von anderen Operatoren, die ebenfalls räumliche Indizes verwenden, als der räumliche Index nur dann angewandt wird, wenn sich der Operand in einer ORDER BY Klausel befindet.



Anmerkung
Der Index kommt nur zum Tragen, wenn eine der Geometrien eine Konstante ist (sich nicht in einer Subquery/CTE befindet). Z.B. 'SRID=3005;LINESTRINGM(0 0 0,0 0 1)'::geometry und nicht a.geom

Availability: 2.2.0.

Beispiele

This example orders stored trajectories by their distance from a literal query trajectory.

Code

```
\set qt 'ST_AddMeasure(ST_MakeLine(ST_MakePointM(-350, 300, 0), ST_MakePointM(-410, 490, 0)) ↩
    , 10, 20)'

SELECT track_id, dist FROM (
  SELECT track_id,
    ST_DistanceCPA(tr, :qt) dist
  FROM trajectories
  ORDER BY tr |=| :qt
  LIMIT 5
) foo;
```

Output

```
track_id        dist
----------+------------------
      395 | 0.576496831518066
      380 |  5.06797130410151
      390 |  7.72262293958322
      385 |   9.8004461358071
      405 |  10.9534397988433
(5 rows)
```

Siehe auch

ST_DistanceCPA, ST_ClosestPointOfApproach, ST_IsValidTrajectory

7.10.2.3 <#>

<#> — Gibt die 2D Entfernung zwischen den Bounding Boxes von A und B zurück

Synopsis

double precision **<#>**(geometry A , geometry B);

Beschreibung

The <#> operator returns distance between two floating point bounding boxes, possibly reading them from a spatial index. Useful for doing nearest neighbor **approximate** distance ordering.



Anmerkung
Dieser Operand verwendet sämtliche Indizes, welche für die Geometrien vorhanden sind. Er unterscheidet sich insofern von anderen Operatoren, welche ebenfalls räumliche Indizes verwenden, als der räumliche Index nur dann verwendet wird, falls sich der Operand in einer ORDER BY Klausel befindet.



Anmerkung
Der Index kommt nur zum Tragen, wenn eine der Geometrien eine Konstante ist; z.B.: ORDER BY (ST_GeomFromText('POINT(1 2)') <#> geom) anstatt g1.geom <#>.

Availability: 2.0.0.

Beispiele

Code

```
SELECT *
FROM (
SELECT b.tlid, b.mtfcc,
    b.geom <#
> ST_GeomFromText('LINESTRING(746149 2948672,745954 2948576,
        745787 2948499,745740 2948468,745712 2948438,
        745690 2948384,745677 2948319)', 2249) As b_dist,
        ST_Distance(b.geom, ST_GeomFromText('LINESTRING(746149 2948672,745954 2948576,
        745787 2948499,745740 2948468,745712 2948438,
        745690 2948384,745677 2948319)', 2249)) As act_dist
    FROM bos_roads As b
    ORDER BY b_dist, b.tlid
    LIMIT 100) As foo
    ORDER BY act_dist, tlid LIMIT 10;
```

Output

```
   tlid    | mtfcc |      b_dist      |     act_dist
-----------+-------+------------------+------------------
  85732027 | S1400 |                0 |                0
  85732029 | S1400 |                0 |                0
  85732031 | S1400 |                0 |                0
  85734335 | S1400 |                0 |                0
  85736037 | S1400 |                0 |                0
 624683742 | S1400 |                0 | 128.528874268666
  85719343 | S1400 | 260.839270432962 | 260.839270432962
  85741826 | S1400 | 164.759294123275 | 260.839270432962
  85732032 | S1400 |           277.75 | 311.830282365264
  85735592 | S1400 |           222.25 | 311.830282365264
(10 rows)
```

Siehe auch

ST_DWithin, ST_Distance, <->

7.10.2.4 <<->>

<<->> — Gibt den n-D-Abstand zwischen den Geometrien oder Begrenzungsrahmen von A und B zurück

Synopsis

double precision **<<->>**(geometry A , geometry B);

Beschreibung

Der <<−>> Operator gibt die n-D (euklidische) Entfernung zwischen den geometrischen Schwerpunkten der Begrenzungsrechtecke zweier Geometrien zurück. Praktikabel für nearest neighbor **approximate** distance ordering.



Anmerkung
Dieser Operator verwendet n-D GiST Indizes, falls diese für die Geometrien vorhanden sind. Er unterscheidet sich insofern von anderen Operatoren, die räumliche Indizes verwenden, indem der räumliche Index nur dann verwendet wird, wenn sich der Operator in einer ORDER BY Klausel befindet.



Anmerkung
Der Index kommt nur zum Tragen, wenn eine der Geometrien eine Konstante ist (sich nicht in einer Subquery/CTE befindet). Z.B. 'SRID=3005;POINT(1011102 450541)'::geometry und nicht a.geom

Availability: 2.2.0.

Siehe auch

<->

7.11 Lagevergleiche

7.11.1 Topologische Beziehungen

7.11.1.1 ST_3DIntersects

ST_3DIntersects — Prüft, ob sich zwei Geometrien in 3D räumlich schneiden - nur für Punkte, Linienzüge, Polygone, polyedrische Flächen (Bereich)

Synopsis

boolean **ST_3DIntersects**(geometry geomA , geometry geomB);

Beschreibung

Overlaps, Touches und Within implizieren räumliche Überschneidung. Wenn irgendeine dieser Eigenschaften TRUE zurückgibt, dann überschneiden sich die geometrischen Objekte auch. Disjoint impliziert FALSE für die räumliche Überschneidung.



Anmerkung
Diese Funktion beinhaltet automatisch einen Bounding-Box-Vergleich, der alle räumlichen Indizes verwendet, die für die Geometrien verfügbar sind.



Anmerkung
Aufgrund von Floating-Robustness-Fehlern überschneiden sich Geometrien nicht immer so, wie man es nach der geometrischen Bearbeitung erwarten würde. So liegt beispielsweise der Punkt, der einer Geometrie am nächsten liegt, nicht unbedingt auf dem Linienzug. Für diese Art von Problemen, bei denen ein Abstand von einem Zentimeter einfach als Schnittpunkt betrachtet werden soll, verwenden Sie ST_3DDWithin.

Änderung: 3.0.0 das SFCGAL Back-end wurde entfernt, das GEOS Back-end unterstützt TIN.

Verfügbarkeit: 2.0.0

 Diese Funktion unterstützt 3d und lässt den Z-Index nicht fallen.

 Diese Funktion unterstützt polyedrische Flächen.

 Diese Funktion unterstützt Dreiecke und dreieckige unregelmäßige Netzoberflächen (TIN).

 Diese Methode setzt die SQL/MM-Spezifikation um. SQL-MM IEC 13249-3: 5.1

Beispiele mit dem geometrischen Datentyp

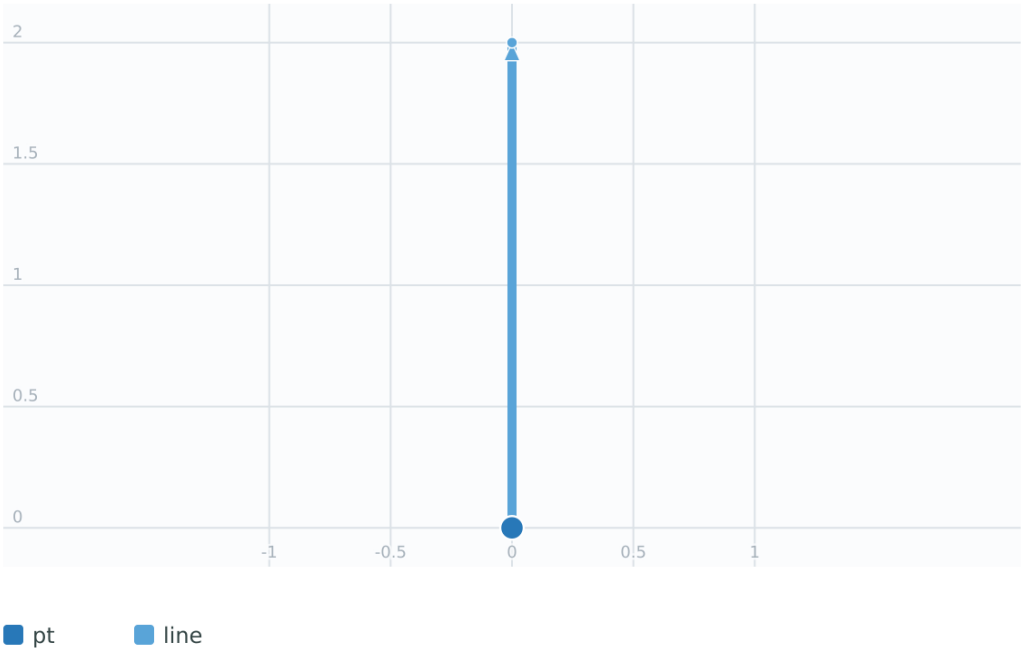
Code

```
SELECT ST_3DIntersects(pt, line), ST_Intersects(pt, line)
FROM (SELECT 'POINT(0 0 2)'::geometry As pt, 'LINESTRING (0 0 1,0 2 3)'::geometry As line)  ↩
    As foo;
```

Output

```
st_3dintersects | st_intersects
----------------+---------------
 f              | t
(1 row)
```

Figure



Beispiele mit dem Datentyp TIN

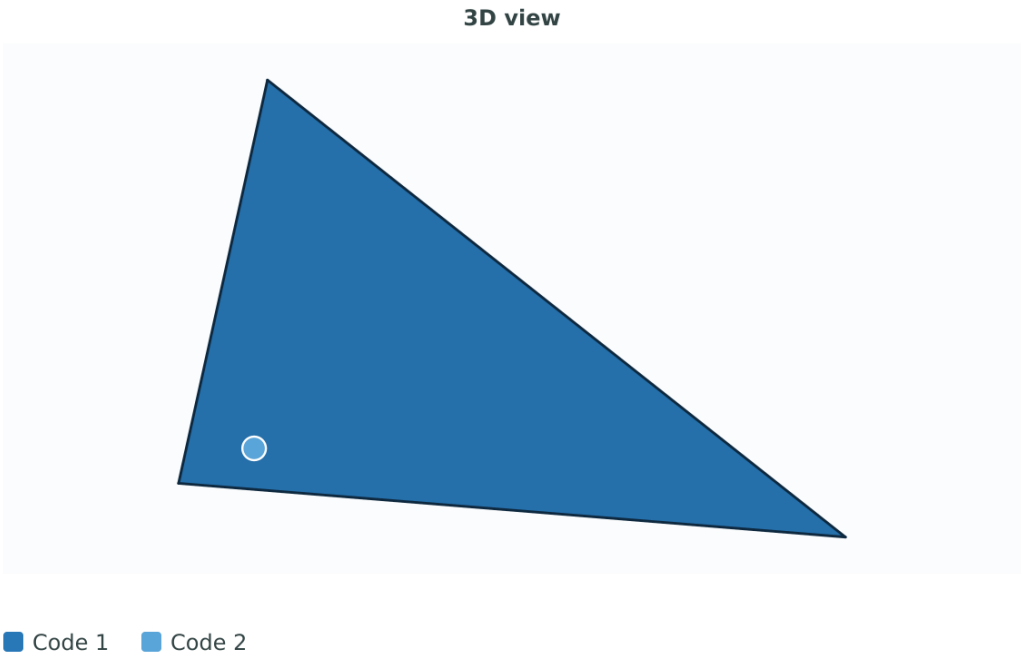
Code

```
SELECT ST_3DIntersects('TIN(((0 0 0,1 0 0,0 1 0,0 0 0)))'::geometry, 'POINT(.1 .1 0)':: ↩
    geometry);
```

Output

```
t
```

Figure



Siehe auch

ST_3DDWithin, ST_Intersects

7.11.1.2 ST_Contains

ST_Contains — Tests, wenn jeder Punkt von B in A liegt und ihre Innenräume einen gemeinsamen Punkt haben

Synopsis

boolean **ST_Contains**(geometry geomA, geometry geomB);

Beschreibung

Gibt TRUE zurück, wenn die Geometrie A die Geometrie B enthält. A enthält B nur dann, wenn alle Punkte von B innerhalb von A liegen (d. h. im Inneren oder am Rand von A) (oder gleichwertig, wenn keine Punkte von B im Äußeren von A liegen) und die Innenräume von A und B mindestens einen Punkt gemeinsam haben.

Mathematisch ausgedrückt: $ST_Contains(A, B) \Leftrightarrow (A \bigcap B = B) \wedge (Int(A) \bigcap Int(B) \neq \varnothing)$

Die Enthält-Beziehung ist reflexiv: Jede Geometrie enthält sich selbst. (Im Gegensatz dazu enthält im Prädikat ST_ContainsProperly eine Geometrie *nicht* sich selbst.) Die Beziehung ist antisymmetrisch: Wenn `ST_Contains(A,B) = true` und `ST_Contains(B, = true`, dann müssen die beiden Geometrien topologisch gleich sein (`ST_Equals(A,B) = true`).

ST_Contains ist die Umkehrung von ST_Within. `ST_Contains(A,B) = ST_Within(B,A)`.



Anmerkung
Da die Innenräume einen gemeinsamen Punkt haben müssen, besteht eine Feinheit der Definition darin, dass Polygone und Linien *nicht* Linien und Punkte enthalten, die vollständig in ihrer Begrenzung liegen. Für weitere Details siehe Subtleties of OGC Covers, Contains, Within. Das Prädikat ST_Covers bietet eine umfassendere Beziehung.



Anmerkung
Diese Funktion beinhaltet automatisch einen Bounding-Box-Vergleich, der alle räumlichen Indizes verwendet, die für die Geometrien verfügbar sind. Um die Verwendung eines Indices zu vermeiden, kann die Funktion `_ST_Contains` verwendet werden.

Wird durch das GEOS Modul ausgeführt

Verbessert: 2.3.0 Verbesserung des PIP-Kurzschlusses erweitert um die Unterstützung von MultiPoints mit wenigen Punkten. Frühere Versionen unterstützten nur Punkte in Polygonen.

Verbessert: 3.0.0 ermöglicht die Unterstützung von `GEOMETRYCOLLECTION`



Wichtig
Verwenden Sie diese Funktion nicht mit ungültigen Geometrien. Sie werden unerwartete Ergebnisse erhalten.

HINWEIS: Dies ist die "zulässige" Version, die einen booleschen Wert und keine ganze Zahl zurückgibt.

 Diese Methode implementiert die OGC Simple Features Implementation Specification for SQL 1.1. s2.1.1.2 // s2.1.13.3 - dasselbe wie innerhalb (Geometrie B, Geometrie A)

 Diese Methode setzt die SQL/MM-Spezifikation um. SQL-MM 3: 5.1.31

Beispiele

`ST_Contains` gibt `TRUE` in den folgenden Situationen zurück:

A LineString contains a MultiPoint whose points lie in its interior or boundary.

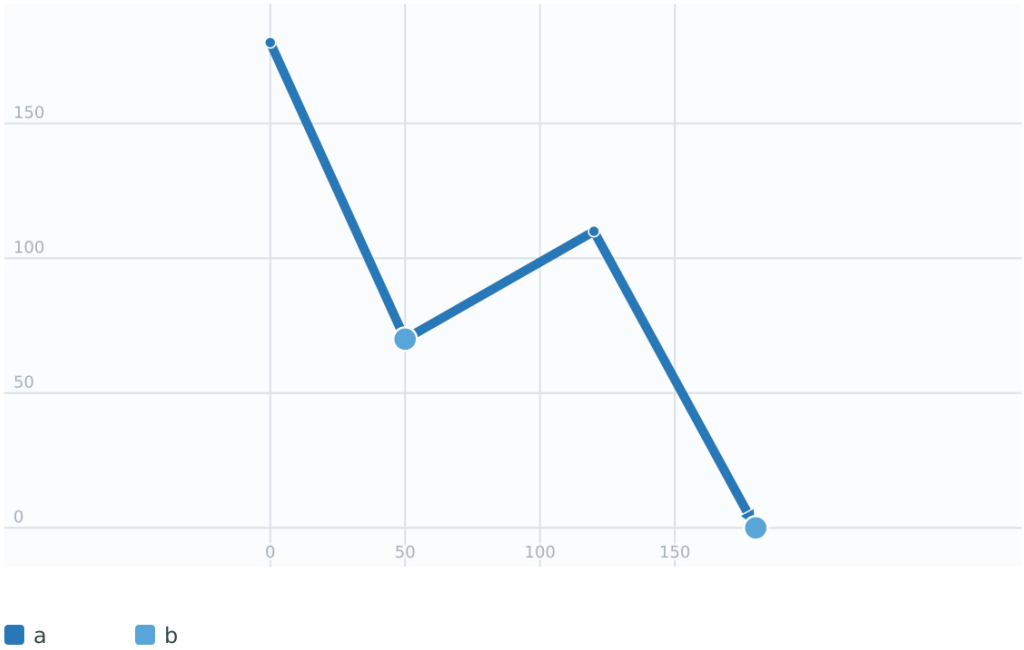
Code

```
SELECT ST_Contains(a, b)
FROM (SELECT
  'LINESTRING (10 190,60 80,130 120,190 10)'::geometry AS a,
  'MULTIPOINT (60 80,190 10)'::geometry AS b
) AS example;
```

Output

```
t
```

Figure



A Polygon contains a Point in its interior.

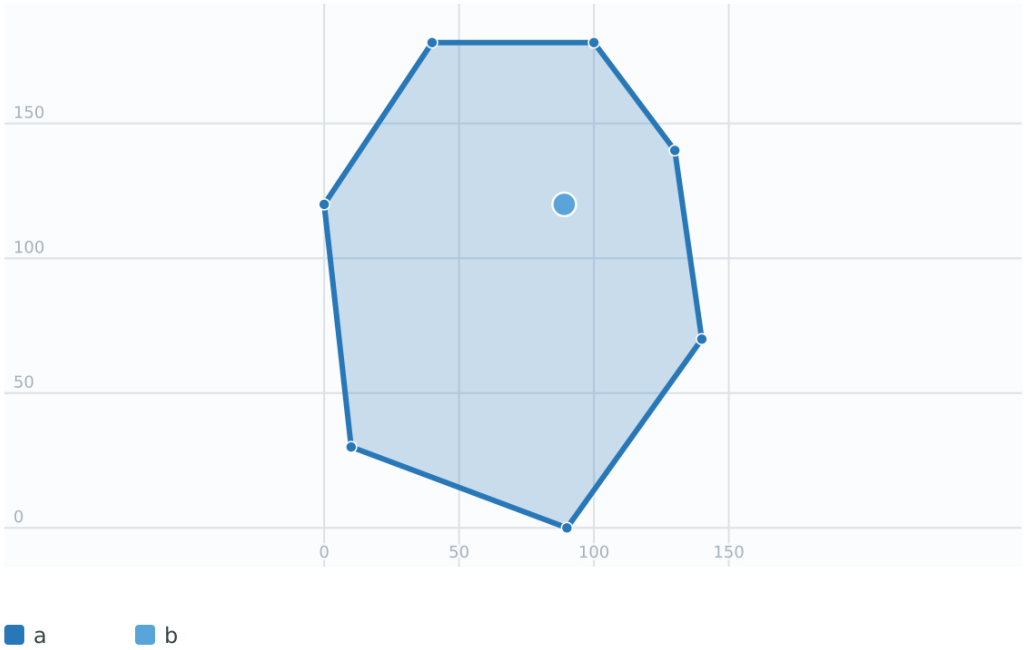
Code

```
SELECT ST_Contains(a, b)
FROM (SELECT
  'POLYGON ((10 130,50 190,110 190,140 150,150 80,100 10,20 40,10 130))'::geometry AS a,
  'POINT (99 130)'::geometry AS b
) AS example;
```

Output

```
t
```

Figure



A Polygon contains a LineString with interior points inside the Polygon.

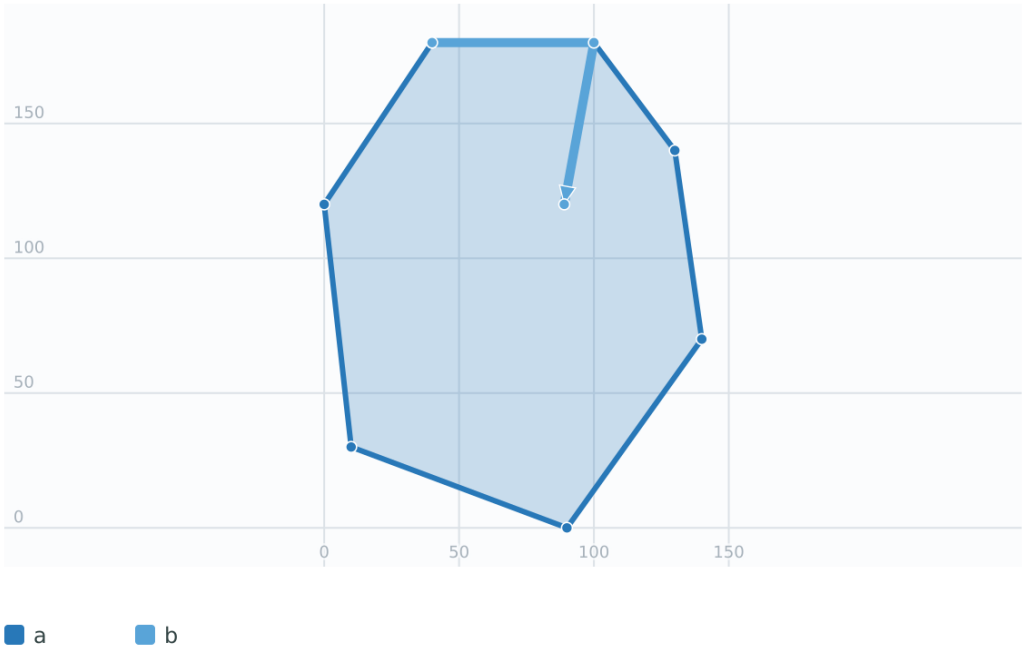
Code

```
SELECT ST_Contains(a, b)
FROM (SELECT
  'POLYGON ((10 130,50 190,110 190,140 150,150 80,100 10,20 40,10 130))'::geometry AS a,
  'LINESTRING (50 190,110 190,99 130)'::geometry AS b
) AS example;
```

Output

```
t
```

Figure



A Polygon contains another Polygon in its interior.

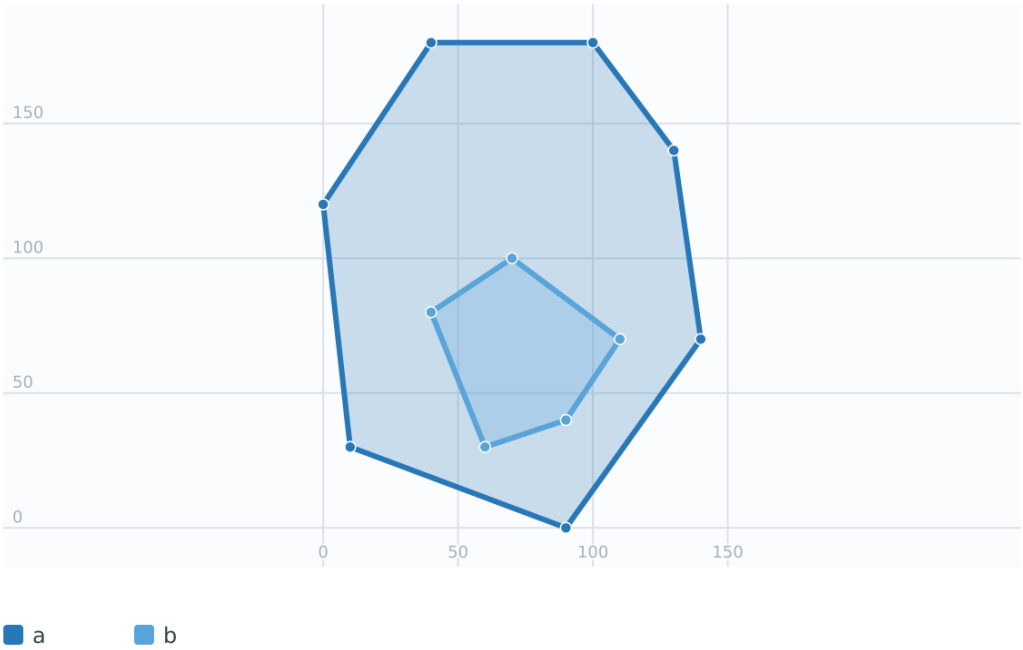
Code

```
SELECT ST_Contains(a, b)
FROM (SELECT
  'POLYGON ((10 130,50 190,110 190,140 150,150 80,100 10,20 40,10 130))'::geometry AS a,
  'POLYGON ((70 40,100 50,120 80,80 110,50 90,70 40))'::geometry AS b
) AS example;
```

Output

```
t
```

Figure



`ST_Contains` gibt `FALSE` in den folgenden Situationen zurück:

A Polygon does not contain a MultiPoint when one point lies in a hole.

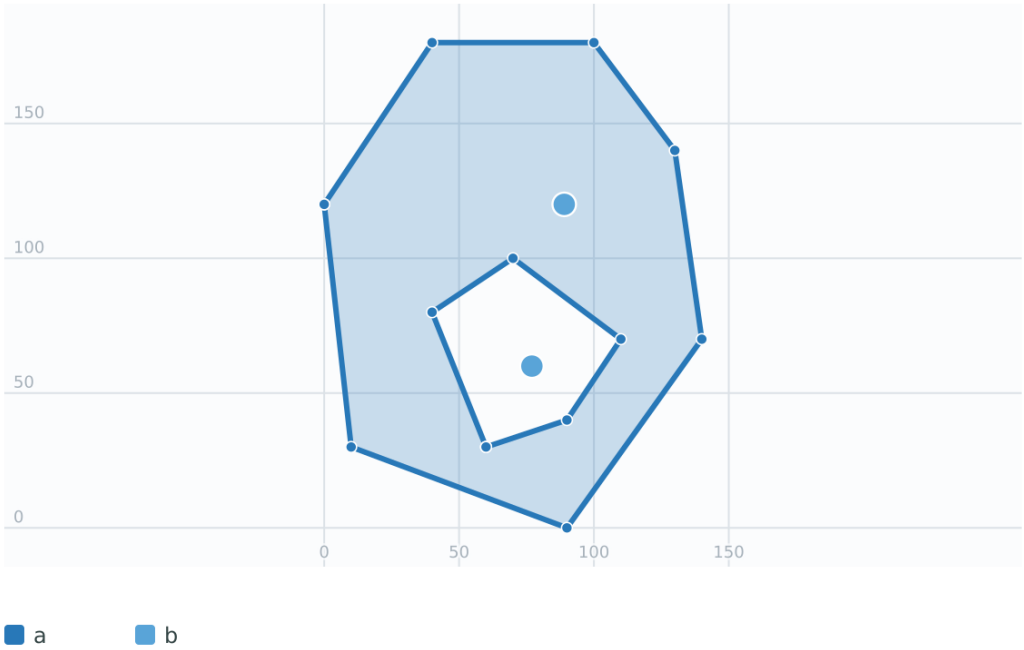
Code

```
SELECT ST_Contains(a, b)
FROM (SELECT
  'POLYGON ((10 130,50 190,110 190,140 150,150 80,100 10,20 40,10 130),(70 40,100 50,120 ↩
    80,80 110,50 90,70 40))'::geometry AS a,
  'MULTIPOINT (99 130,87 70)'::geometry AS b
) AS example;
```

Output

```
f
```

Figure



A Polygon does not contain a LineString extending outside it.

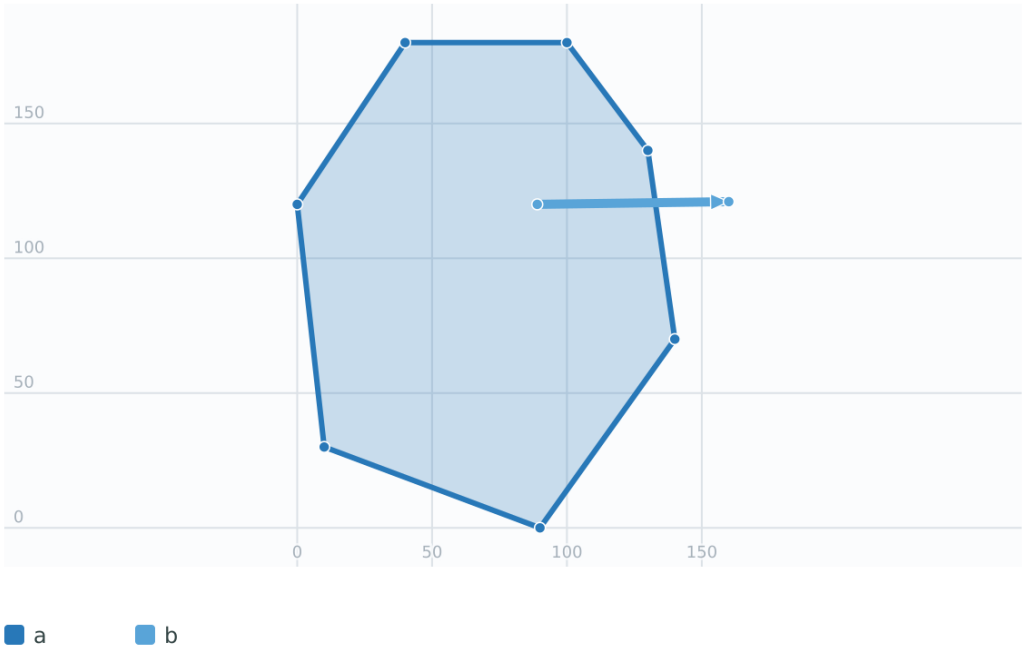
Code

```
SELECT ST_Contains(a, b)
FROM (SELECT
  'POLYGON ((10 130,50 190,110 190,140 150,150 80,100 10,20 40,10 130))'::geometry AS a,
  'LINESTRING (99 130,170 131)'::geometry AS b
) AS example;
```

Output

```
f
```

Figure



Aufgrund der inneren Schnittpunktbedingung `ST_Contains` liefert `FALSE` in den folgenden Situationen (während `ST_Covers` `TRUE` liefert):

A LineString does not contain a Point lying only on its boundary.

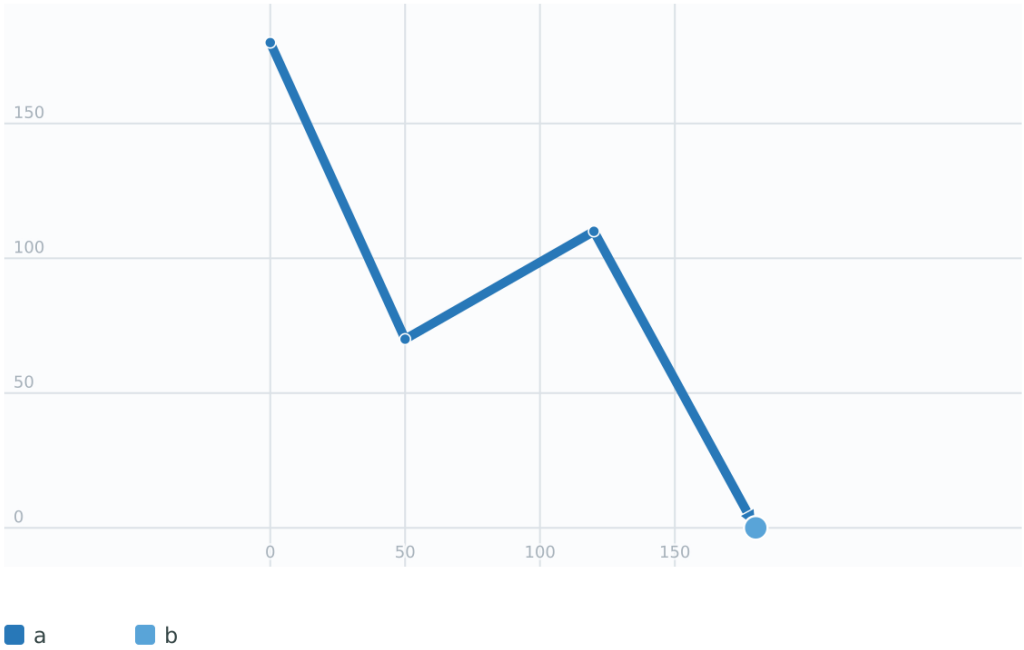
Code

```
SELECT ST_Contains(a, b), ST_Covers(a, b)
FROM (SELECT
  'LINESTRING (10 190,60 80,130 120,190 10)'::geometry AS a,
  'POINT (190 10)'::geometry AS b
) AS example;
```

Output

```
f | t
```

Figure



A Polygon does not contain a LineString lying entirely on its boundary.

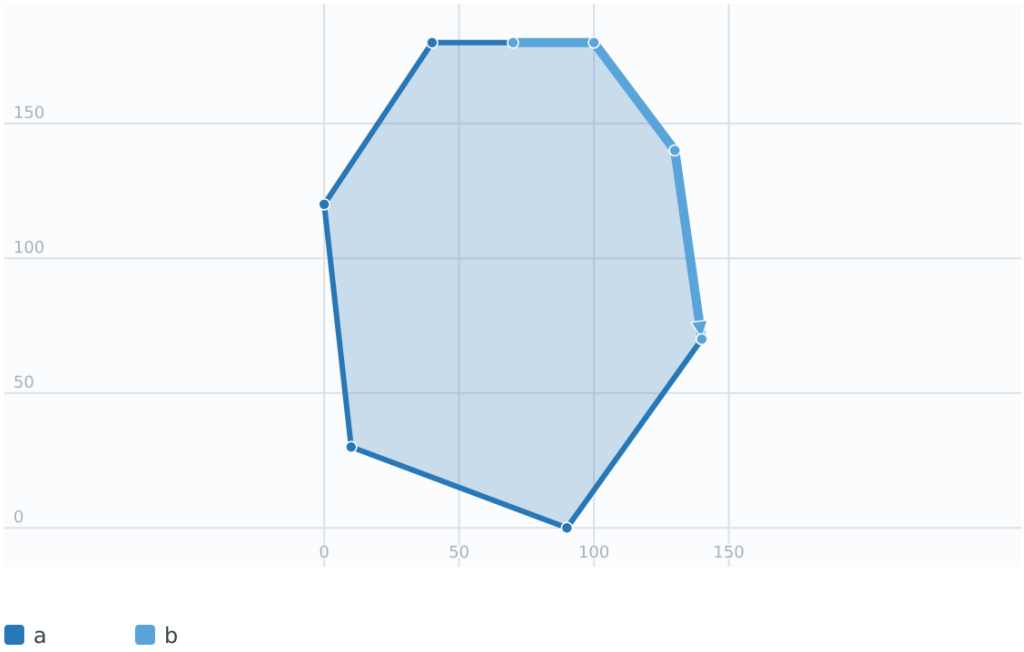
Code

```
SELECT ST_Contains(a, b), ST_Covers(a, b)
FROM (SELECT
  'POLYGON ((10 130,50 190,110 190,140 150,150 80,100 10,20 40,10 130))'::geometry AS a,
  'LINESTRING (80 190,110 190,140 150,150 80)'::geometry AS b
) AS example;
```

Output

```
f | t
```

Figure



This example tests a circle within another circle.

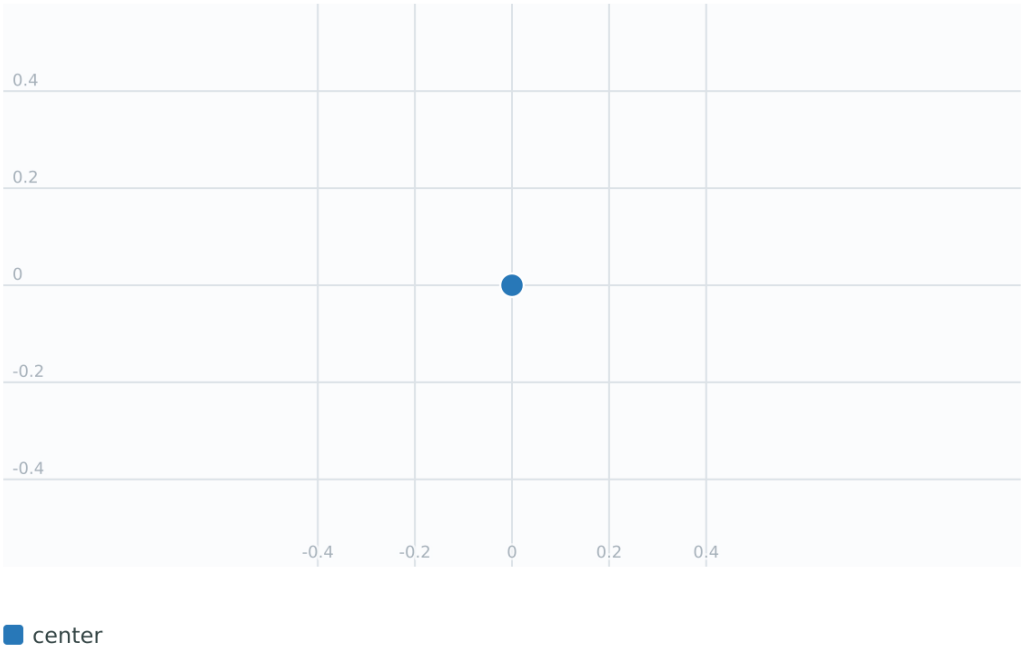
Code

```
WITH circles AS (
  SELECT ST_Buffer(center, 10) AS smallc,
         ST_Buffer(center, 20) AS bigc
  FROM (SELECT 'POINT(1 2)'::geometry AS center) AS p
)
SELECT
  ST_Contains(smallc, bigc) AS smallcontainsbig,
  ST_Contains(bigc, smallc) AS bigcontainssmall,
  ST_Contains(bigc, ST_Union(smallc, bigc)) AS bigcontainsunion,
  ST_Equals(bigc, ST_Union(smallc, bigc)) AS bigisunion,
  ST_Covers(bigc, ST_ExteriorRing(bigc)) AS bigcoversexterior,
  ST_Contains(bigc, ST_ExteriorRing(bigc)) AS bigcontainsexterior
FROM circles;
```

Output

```
-[ RECORD 1 ]-----
smallcontainsbig   | f
bigcontainssmall   | t
bigcontainsunion   | t
bigisunion         | t
bigcoversexterior  | t
bigcontainsexterior | f
```

Figure



This example demonstrates the difference between contains and contains-properly predicates.

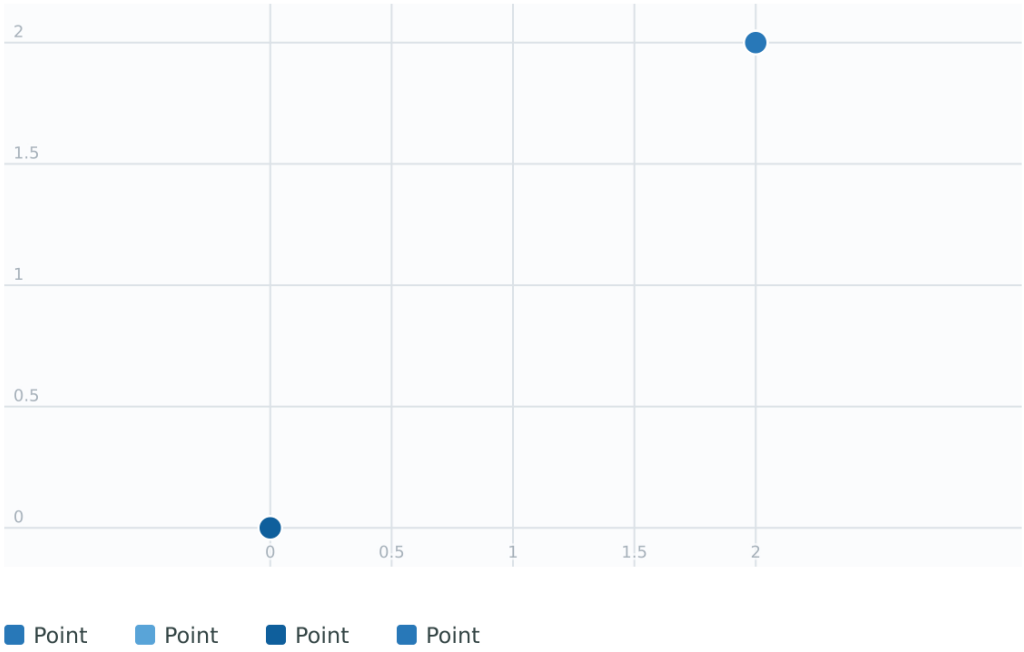
Code

```
SELECT
  ST_GeometryType(geom_a) AS geomtype,
  ST_Contains(geom_a, geom_a) AS acontainsa,
  ST_ContainsProperly(geom_a, geom_a) AS acontainspropa,
  ST_Contains(geom_a, ST_Boundary(geom_a)) AS acontainsba,
  ST_ContainsProperly(geom_a, ST_Boundary(geom_a)) AS acontainspropba
FROM (VALUES
    (ST_Buffer(ST_Point(1, 1), 5, 1)),
    (ST_MakeLine(ST_Point(1, 1), ST_Point(-1, -1))),
    (ST_Point(1, 1))
  ) AS foo(geom_a);
```

Output

```
geomtype     | acontainsa | acontainspropa | acontainsba | acontainspropba
-------------+------------+----------------+-------------+-----------------
ST_Polygon   | t          | f              | f           | f
ST_LineString | t          | f              | f           | f
ST_Point     | t          | t              | f           | f
```

Figure



Siehe auch

ST_Boundary, ST_ContainsProperly, ST_Covers, ST_CoveredBy, ST_Equals, ST_Within

7.11.1.3 ST_ContainsProperly

ST_ContainsProperly — Prüft, ob jeder Punkt von B im Inneren von A liegt

Synopsis

boolean **ST_ContainsProperly**(geometry geomA, geometry geomB);

Beschreibung

Gibt `true` zurück, wenn jeder Punkt von B im Inneren von A liegt (oder äquivalent dazu, kein Punkt von B liegt im Rand oder außerhalb von A).

Mathematisch ausgedrückt: *ST_EnthältEcht(A, B)* $\Leftrightarrow$ $Int(A) \bigcap B = B$

A enthält B ordnungsgemäß, wenn die DE-9IM-Schnittpunktmatrix für die beiden Geometrien mit [T**FF*FF*]

A enthält sich selbst nicht richtig, aber es enthält sich selbst.

Eine Anwendung für dieses Prädikat ist die Berechnung der Schnittpunkte einer Menge von Geometrien mit einer großen polygonalen Geometrie. Da die Schnittmenge eine recht langsame Operation ist, kann es effizienter sein, containsProperly zu verwenden, um Testgeometrien herauszufiltern, die vollständig innerhalb des Bereichs liegen. In diesen Fällen ist von vornherein bekannt, dass der Schnittpunkt genau die ursprüngliche Testgeometrie ist.



Anmerkung
Diese Funktion beinhaltet automatisch einen Bounding-Box-Vergleich, der alle räumlichen Indizes verwendet, die für die Geometrien verfügbar sind. Um die Verwendung von Indizes zu vermeiden, verwenden Sie die Funktion `_ST_ContainsProperly`.



Anmerkung
Der Vorteil dieses Prädikats gegenüber ST_Contains und ST_Intersects ist, dass es effizienter berechnet werden kann, ohne dass die Topologie an einzelnen Punkten berechnet werden muss.

Wird vom GEOS Modul ausgeführt

Verfügbarkeit: 1.4.0

Verbessert: 3.0.0 ermöglicht die Unterstützung von `GEOMETRYCOLLECTION`



Wichtig
Verwenden Sie diese Funktion nicht mit ungültigen Geometrien. Sie werden unerwartete Ergebnisse erhalten.

Beispiele

This example tests a circle within another circle.

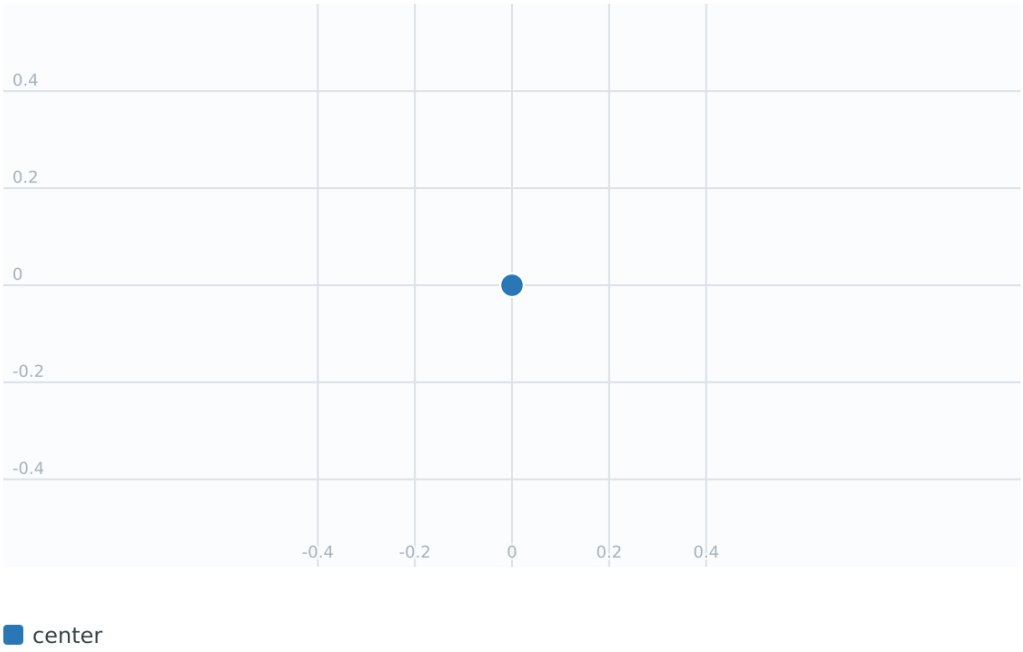
Code

```
WITH circles AS (
  SELECT ST_Buffer(center, 10) AS smallc,
         ST_Buffer(center, 20) AS bigc
  FROM (SELECT 'POINT(1 2)'::geometry AS center) AS p
)
SELECT
  ST_ContainsProperly(smallc, bigc) AS smallcontainspropbig,
  ST_ContainsProperly(bigc, smallc) AS bigcontainspropsmall,
  ST_ContainsProperly(bigc, ST_Union(smallc, bigc)) AS bigcontainspropunion,
  ST_Equals(bigc, ST_Union(smallc, bigc)) AS bigisunion,
  ST_Covers(bigc, ST_ExteriorRing(bigc)) AS bigcoversexterior,
  ST_ContainsProperly(bigc, ST_ExteriorRing(bigc)) AS bigcontainsexterior
FROM circles;
```

Output

```
-[ RECORD 1 ]-----
smallcontainspropbig   | f
bigcontainspropsmall   | t
bigcontainspropunion   | f
bigisunion             | t
bigcoversexterior      | t
bigcontainsexterior    | f
```

Figure



This example demonstrates the difference between contains and contains-properly predicates.

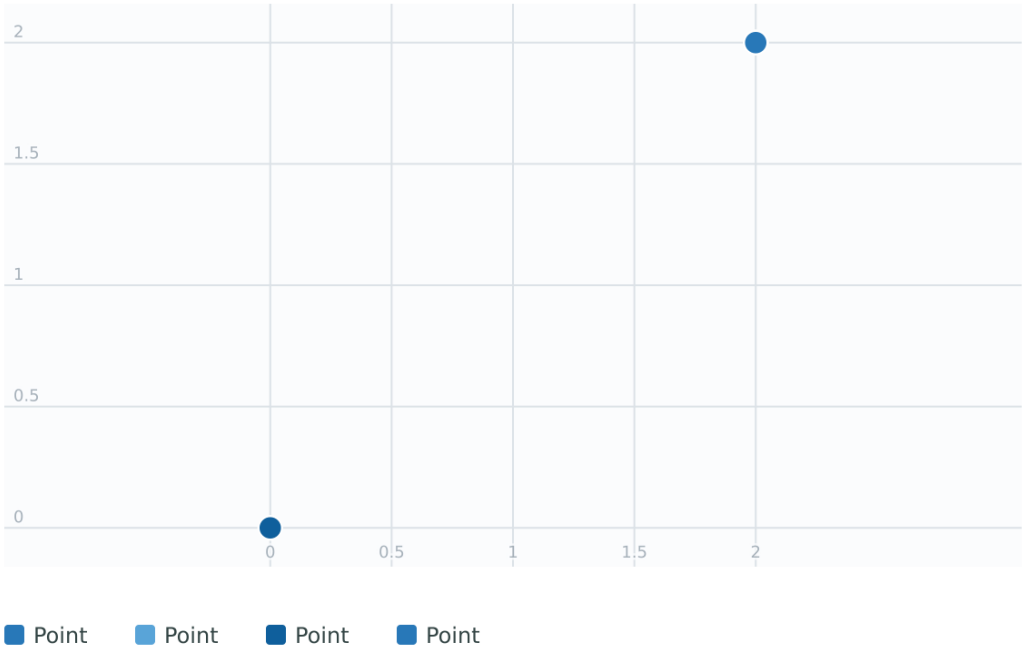
Code

```
WITH input(geom) AS (VALUES
  (ST_Buffer(ST_Point(1, 1), 5, 1)),
  (ST_MakeLine(ST_Point(1, 1), ST_Point(-1, -1))),
  (ST_Point(1, 1))
)
SELECT ST_GeometryType(geom) AS geomtype,
       ST_Contains(geom, geom) AS acontainsa,
       ST_ContainsProperly(geom, geom) AS acontainspropa,
       ST_Contains(geom, ST_Boundary(geom)) AS acontainsba,
       ST_ContainsProperly(geom, ST_Boundary(geom)) AS acontainspropba
FROM input;
```

Output

```
geomtype    | acontainsa | acontainspropa | acontainsba | acontainspropba
------------+------------+----------------+-------------+-----------------
ST_Polygon    | t          | f              | f           | f
ST_LineString | t          | f              | f           | f
ST_Point      | t          | t              | f           | f
```

Figure



Siehe auch

ST_GeometryType, ST_Boundary, ST_Contains, ST_Covers, ST_CoveredBy, ST_Equals, ST_Relate, ST_Within

7.11.1.4 ST_CoveredBy

ST_CoveredBy — Prüft, ob jeder Punkt von A in B liegt

Synopsis

boolean **ST_CoveredBy**(geometry geomA, geometry geomB);
boolean **ST_CoveredBy**(geography geogA, geography geogB);

Beschreibung

Gibt `true` zurück, wenn jeder Punkt in Geometrie/Geografie A innerhalb von Geometrie/Geografie B liegt (d. h. das Innere oder die Grenze von B schneidet). Gleichbedeutend wird getestet, dass kein Punkt von A außerhalb von B liegt.

Mathematisch ausgedrückt: *ST_CoveredBy(A, B)* $\Leftrightarrow A \bigcap B = A$

ST_CoveredBy ist die Umkehrung von ST_Covers. `ST_CoveredBy(A,B) = ST_Covers(B,A)`.

Im Allgemeinen sollte diese Funktion anstelle von ST_Within verwendet werden, da sie eine einfachere Definition hat, die nicht die Eigenart hat, dass "Grenzen nicht innerhalb ihrer Geometrie liegen".



Anmerkung
Diese Funktion beinhaltet automatisch einen Bounding-Box-Vergleich, der alle räumlichen Indizes verwendet, die für die Geometrien verfügbar sind. Um die Verwendung von Indizes zu vermeiden, verwenden Sie die Funktion `_ST_CoveredBy`.

Verbessert: 3.0.0 ermöglicht die Unterstützung von `GEOMETRYCOLLECTION`



Wichtig
Verwenden Sie diese Funktion nicht mit ungültigen Geometrien. Sie werden unerwartete Ergebnisse erhalten.

Wird durch das GEOS Modul ausgeführt

Verfügbarkeit: 1.2.2

HINWEIS: Dies ist die "zulässige" Version, die einen booleschen Wert und keine ganze Zahl zurückgibt.

Kein OGC-Standard, aber Oracle hat ihn auch.

Beispiele

This example tests a circle covered by another circle.

Code

```
WITH circles AS (
  SELECT ST_Buffer(center, 10) AS smallc,
         ST_Buffer(center, 20) AS bigc
  FROM (SELECT 'POINT(1 2)'::geometry AS center) AS p
)
SELECT
  ST_CoveredBy(smallc, smallc) AS smallinsmall,
  ST_CoveredBy(smallc, bigc) AS smallcoveredbybig,
  ST_CoveredBy(ST_ExteriorRing(bigc), bigc) AS exteriorcoveredbybig,
  ST_Within(ST_ExteriorRing(bigc), bigc) AS exteriorwithinbig
FROM circles;
```

Output

```
smallinsmall | smallcoveredbybig | exteriorcoveredbybig | exteriorwithinbig
--------------+-------------------+----------------------+-------------------
 t            | t                 | t                    | f
(1 row)
```

Figure



Siehe auch

ST_Contains, ST_Covers, ST_ExteriorRing, ST_Within

7.11.1.5 ST_Covers

ST_Covers — Prüft, ob jeder Punkt von B in A liegt

Synopsis

boolean **ST_Covers**(geometry geomA, geometry geomB);
boolean **ST_Covers**(geography geogpolyA, geography geogpointB);

Beschreibung

Gibt `true` zurück, wenn jeder Punkt in Geometrie/Geografie B innerhalb von Geometrie/Geografie A liegt (d. h. das Innere oder den Rand von A schneidet). Äquivalent dazu wird getestet, dass kein Punkt von B außerhalb (im Äußeren von) A liegt.

Mathematisch ausgedrückt: *ST_Covers(A, B)* $\Leftrightarrow A \bigcap B = B$

ST_Covers ist die Umkehrung von ST_CoveredBy. `ST_Covers(A,B) = ST_CoveredBy(B,A)`.

Im Allgemeinen sollte diese Funktion anstelle von ST_Contains verwendet werden, da sie eine einfachere Definition hat, die nicht die Eigenart hat, dass "Geometrien ihre Begrenzung nicht enthalten".



Anmerkung
Diese Funktion beinhaltet automatisch einen Bounding-Box-Vergleich, der alle räumlichen Indizes verwendet, die für die Geometrien verfügbar sind. Um die Verwendung von Indizes zu vermeiden, verwenden Sie die Funktion `_ST_Covers`.

Verbessert: 3.0.0 ermöglicht die Unterstützung von `GEOMETRYCOLLECTION`



Wichtig
Verwenden Sie diese Funktion nicht mit ungültigen Geometrien. Sie werden unerwartete Ergebnisse erhalten.

Wird durch das GEOS Modul ausgeführt

Verbessert: 2.4.0 Unterstützung für Polygon in Polygon und Linie in Polygon für Geografietypen hinzugefügt

Verbessert: 2.3.0 Verbesserung des PIP-Kurzschlusses für Geometrien, erweitert um die Unterstützung von MultiPoints mit wenigen Punkten. Frühere Versionen unterstützten nur Punkte in Polygonen.

Verfügbarkeit: 1.5 - Unterstützung von geograpischen Koordinaten.

Verfügbarkeit: 1.2.2

HINWEIS: Dies ist die "zulässige" Version, die einen booleschen Wert und keine ganze Zahl zurückgibt.

Kein OGC-Standard, aber Oracle hat ihn auch.

Beispiele

Beispiel für Geometrie

This example tests a circle covering another circle.

Code

```
WITH circles AS (
  SELECT ST_Buffer(center, 10) AS smallc,
         ST_Buffer(center, 20) AS bigc
  FROM (SELECT 'POINT(1 2)'::geometry AS center) AS p
)
SELECT
  ST_Covers(smallc, smallc) AS smallinsmall,
  ST_Covers(smallc, bigc) AS smallcoversbig,
  ST_Covers(bigc, ST_ExteriorRing(bigc)) AS bigcoversexterior,
  ST_Contains(bigc, ST_ExteriorRing(bigc)) AS bigcontainsexterior
FROM circles;
```

Output

```
smallinsmall | smallcoversbig | bigcoversexterior | bigcontainsexterior
-------------+----------------+-------------------+--------------------
 t           | f              | t                 | f
(1 row)
```

Figure



Geography example

This example compares a point with a 300-meter buffer against a point and against a 10-meter buffer around that point.

Code

```
SELECT ST_Covers(geog_poly, geog_pt) As poly_covers_pt,
  ST_Covers(ST_Buffer(geog_pt, 10), geog_pt) As buff_10m_covers_cent
  FROM (SELECT ST_Buffer('SRID=4326;POINT(-99.327 31.4821)'::geography, 300) As geog_poly,
        'SRID=4326;POINT(-99.33 31.483)'::geography As geog_pt ) As foo;
```

Output

```
poly_covers_pt | buff_10m_covers_cent
---------------+-------------------
 f             | t
```

Siehe auch

ST_Contains, ST_CoveredBy, ST_Within

7.11.1.6 ST_Crosses

ST_Crosses — Prüft, ob zwei Geometrien einige, aber nicht alle, innere Punkte gemeinsam haben

Synopsis

boolean **ST_Crosses**(geometry g1, geometry g2);

Beschreibung

Vergleicht zwei Geometrieobjekte und gibt `true` zurück, wenn sich ihre Schnittpunkte "räumlich kreuzen"; das heißt, die Geometrien haben einige, aber nicht alle Innenpunkte gemeinsam. Die Schnittmenge der Innenräume der Geometrien darf nicht leer sein und muss eine Dimension haben, die kleiner ist als die maximale Dimension der beiden Eingabegeometrien, und die Schnittmenge der beiden Geometrien darf keiner der beiden Geometrien entsprechen. Andernfalls gibt es `false` zurück. Die Kreuzungsbeziehung ist symmetrisch und irreflexiv.

Mathematisch ausgedrückt: $ST_Crosses(A, B) \Leftrightarrow (dim(\ Int(A) \bigcap Int(B)\ ) < max(\ dim(\ Int(A)\ ), dim(\ Int(B)\ )\ )) \wedge (A \bigcap B \neq A) \wedge (A \bigcap B \neq B)$

Geometrien kreuzen sich, wenn ihre DE-9IM Schnittpunktmatrix übereinstimmt:

- `T*T******` für die Situationen Punkt/Linie, Punkt/Fläche und Linie/Fläche
- `T*****T**` für die Situationen Linie/Punkt, Bereich/Punkt und Bereich/Linie
- `0********` für Line/Line-Situationen
- das Ergebnis ist `falsch` für Punkt/Punkt und Bereich/Fläche Situationen



Anmerkung
Die OpenGIS Simple Features Specification definiert dieses Prädikat nur für die Situationen Punkt/Linie, Punkt/Fläche, Linie/Linie und Linie/Fläche. JTS / GEOS erweitert die Definition so, dass sie auch für die Situationen Linie/Punkt, Fläche/Punkt und Fläche/Linie gilt. Dadurch wird die Beziehung symmetrisch.



Anmerkung
Diese Funktion beinhaltet automatisch einen Bounding-Box-Vergleich, der alle räumlichen Indizes verwendet, die für die Geometrien verfügbar sind.

Verbessert: 3.0.0 ermöglicht die Unterstützung von `GEOMETRYCOLLECTION`

Diese Methode implementiert die OGC Simple Features Implementation Specification for SQL 1.1. s2.1.13.3

Diese Methode setzt die SQL/MM-Spezifikation um. SQL-MM 3: 5.1.29

Beispiele

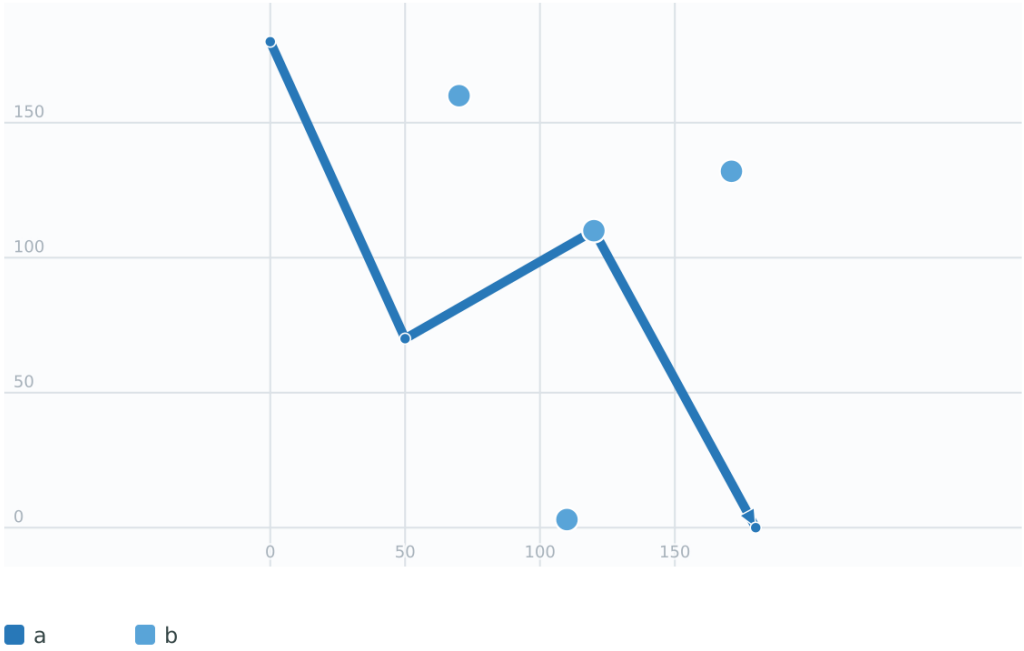
Die folgenden Situationen ergeben alle `true`.

A LineString crosses a MultiPoint when some, but not all, points lie on the line.

Code

```
SELECT ST_Crosses(a, b)
FROM (SELECT
  'LINESTRING (10 190,60 80,130 120,190 10)'::geometry AS a,
  'MULTIPOINT (80 170,120 13,130 120,181 142)'::geometry AS b
) AS example;
```

Figure

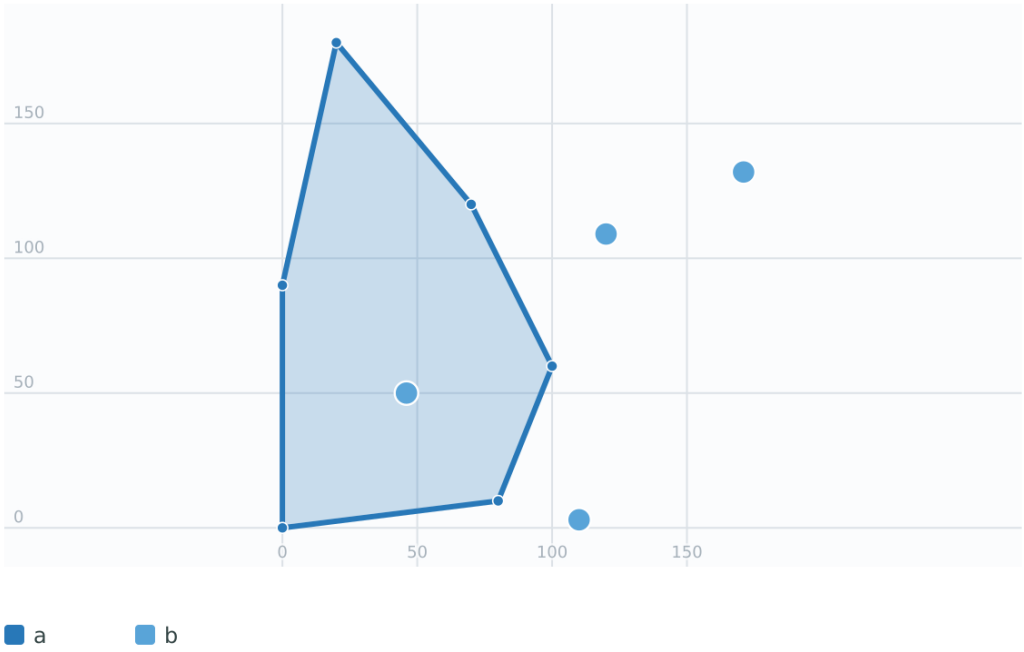


A Polygon crosses a MultiPoint when some, but not all, points lie in its interior.

Code

```
SELECT ST_Crosses(a, b)
FROM (SELECT
  'POLYGON ((30 190,10 100,10 10,90 20,110 70,80 130,30 190))'::geometry AS a,
  'MULTIPOINT (56 60,120 13,130 119,181 142)'::geometry AS b
) AS example;
```

Figure

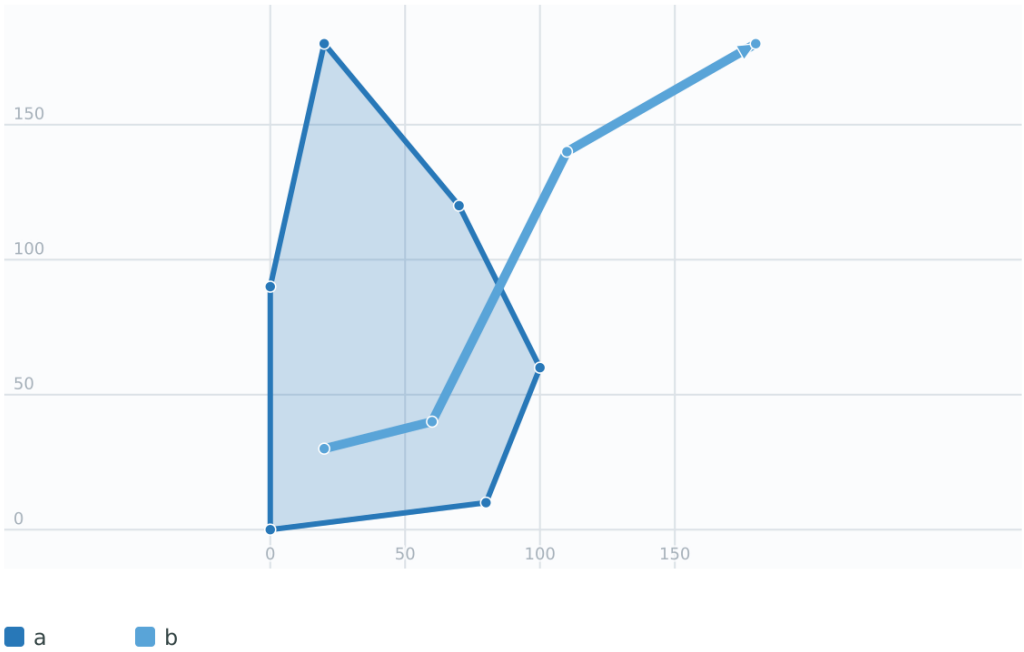


A LineString crosses a Polygon when part of the line lies inside and part outside.

Code

```
SELECT ST_Crosses(a, b)
FROM (SELECT
  'POLYGON ((30 190,10 100,10 10,90 20,110 70,80 130,30 190))'::geometry AS a,
  'LINESTRING (30 40,70 50,120 150,190 190)'::geometry AS b
) AS example;
```

Figure

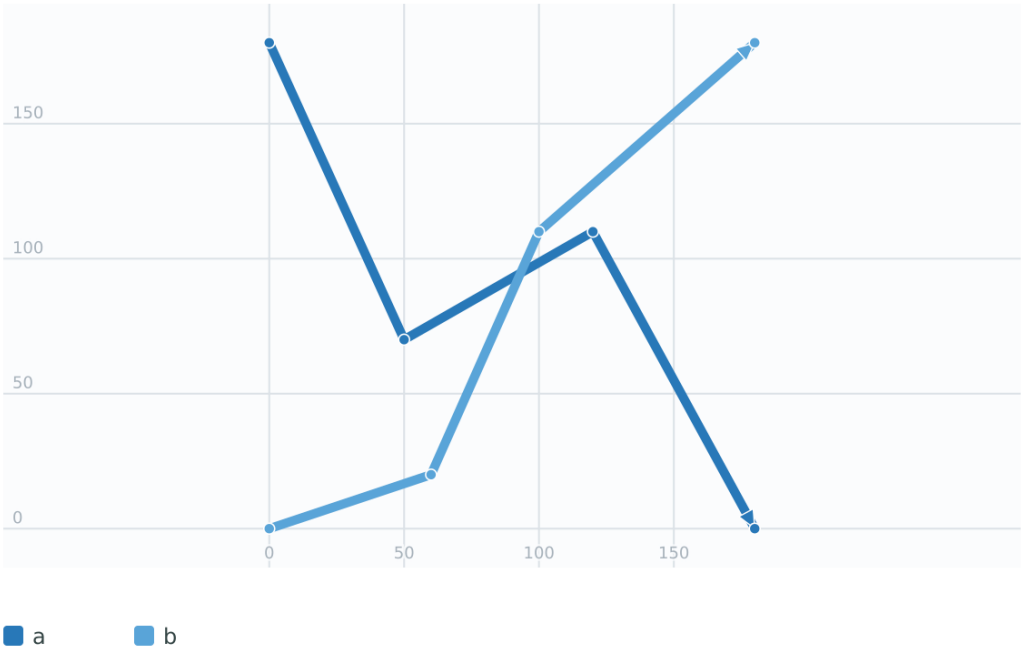


Two LineStrings cross when they share an interior point.

Code

```
SELECT ST_Crosses(a, b)
FROM (SELECT
  'LINESTRING (10 190,60 80,130 120,190 10)'::geometry AS a,
  'LINESTRING (10 10,70 30,110 120,190 190)'::geometry AS b
) AS example;
```

Figure



Stellen Sie sich eine Situation vor, in der ein Benutzer zwei Tabellen hat: eine Tabelle mit Straßen und eine Tabelle mit Autobahnen.

Code

```
CREATE TABLE roads (
  id serial NOT NULL,
  geom geometry,
  CONSTRAINT roads_pkey PRIMARY KEY ( ↩
    road_id)
);
```

Code

```
CREATE TABLE highways (
  id serial NOT NULL,
  the_gem geometry,
  CONSTRAINT roads_pkey PRIMARY KEY ( ↩
    road_id)
);
```

Um eine Liste von Straßen zu ermitteln, die eine Autobahn kreuzen, verwenden Sie eine Abfrage, die der folgenden ähnelt:

Code

```
SELECT roads.id
FROM roads, highways
WHERE ST_Crosses(roads.geom, highways.geom);
```

Siehe auch

ST_Contains, ST_Overlaps

7.11.1.7 ST_Disjoint

ST_Disjoint — Prüft, ob zwei Geometrien keine gemeinsamen Punkte haben

Synopsis

boolean **ST_Disjoint**(geometry A , geometry B);

Beschreibung

Gibt `true` zurück, wenn zwei Geometrien disjunkt sind. Geometrien sind unzusammenhängend, wenn sie keinen gemeinsamen Punkt haben.

Wenn eine andere räumliche Beziehung für ein Paar von Geometrien gilt, sind sie nicht disjunkt. Disjunkt bedeutet, dass ST_Intersects falsch ist.

Mathematisch ausgedrückt: *ST_Disjoint(A, B)* $\Leftrightarrow A \bigcap B = \varnothing$

Verbessert: 3.0.0 ermöglicht die Unterstützung von `GEOMETRYCOLLECTION`

Wird durch das GEOS Modul ausgeführt



Anmerkung
Dieser Funktionsaufruf verwendet keine Indizes. Ein negiertes ST_Intersects Prädikat kann als leistungsfähigere Alternative verwendet werden, die Indizes verwendet: `ST_Disjoint(A,B) = NOT ST_Intersects(A,B)`



Anmerkung
HINWEIS: Dies ist die "zulässige" Version, die einen booleschen Wert und keine ganze Zahl zurückgibt.

Diese Methode implementiert die OGC Simple Features Implementation Specification for SQL 1.1. s2.1.1.2 //s2.1.13.3 - a.Relate(b, 'FF*FF****')

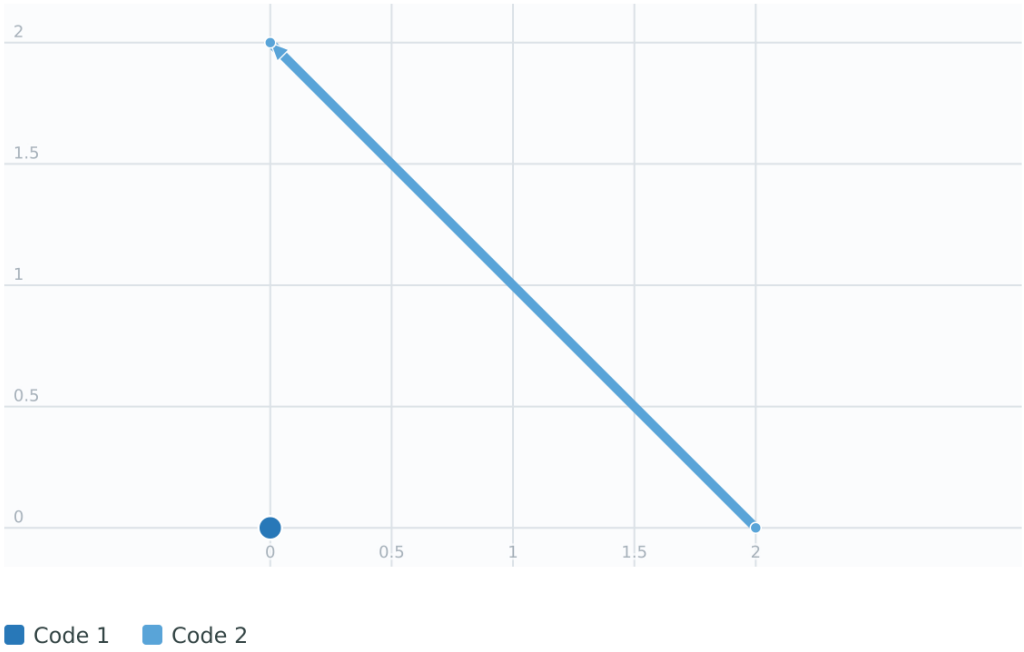
Diese Methode setzt die SQL/MM-Spezifikation um. SQL-MM 3: 5.1.26

Beispiele

Code

```
SELECT ST_Disjoint('POINT(0 0)'::geometry, 'LINESTRING ( 2 0,0 2 )'::geometry);
```

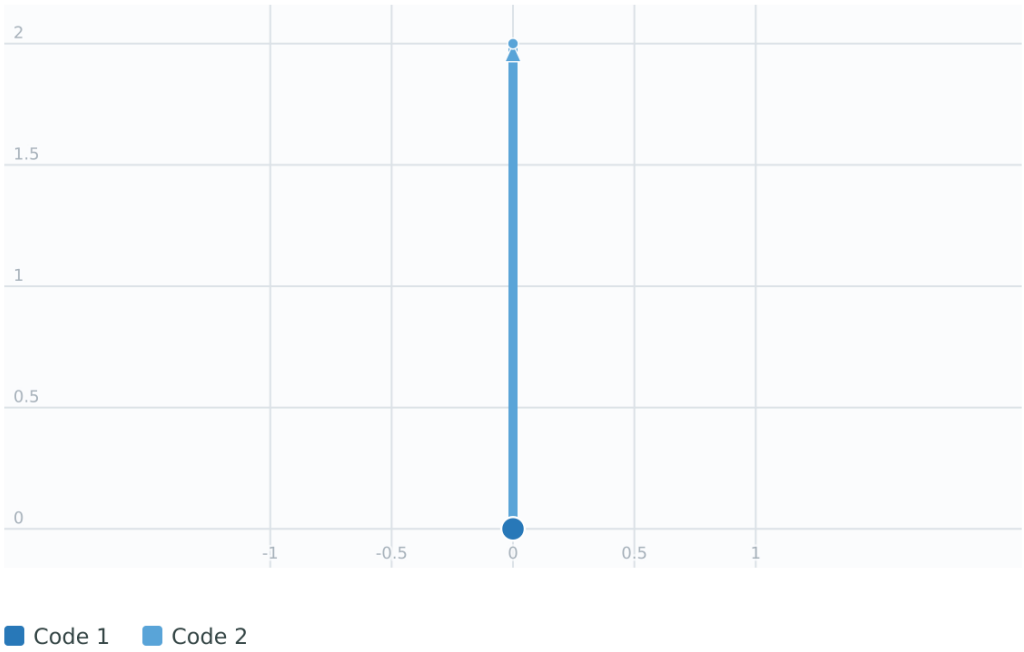
Figure



Code

```
SELECT ST_Disjoint('POINT(0 0)'::geometry, 'LINESTRING ( 0 0,0 2 )'::geometry);
```

Figure



Siehe auch

ST_Intersects

7.11.1.8 ST_Equals

ST_Equals — Prüft, ob zwei Geometrien dieselbe Menge von Punkten enthalten

Synopsis

boolean **ST_Equals**(geometry A, geometry B);

Beschreibung

Gibt `true` zurück, wenn die angegebenen Geometrien "topologisch gleich" sind. Verwenden Sie dies für eine "bessere" Antwort als "=". Topologische Gleichheit bedeutet, dass die Geometrien die gleiche Dimension haben und ihre Punktmengen den gleichen Raum einnehmen. Das bedeutet, dass die Reihenfolge der Eckpunkte in topologisch gleichen Geometrien unterschiedlich sein kann. Um zu überprüfen, ob die Reihenfolge der Punkte konsistent ist, verwenden Sie ST_OrderingEquals (es ist zu beachten, dass ST_OrderingEquals etwas strenger ist als die einfache Überprüfung, ob die Reihenfolge der Punkte gleich ist).

Mathematisch ausgedrückt: *ST_Equals(A, B)* $\Leftrightarrow A = B$

Es gilt die folgende Beziehung: *ST_Equals(A, B)* $\Leftrightarrow$ *ST_Within(A,B)* $\wedge$ *ST_Within(B,A)*

Verbessert: 3.0.0 ermöglicht die Unterstützung von `GEOMETRYCOLLECTION`

Diese Methode implementiert die OGC Simple Features Implementation Specification for SQL 1.1. s2.1.1.2

Diese Methode setzt die SQL/MM-Spezifikation um. SQL-MM 3: 5.1.24

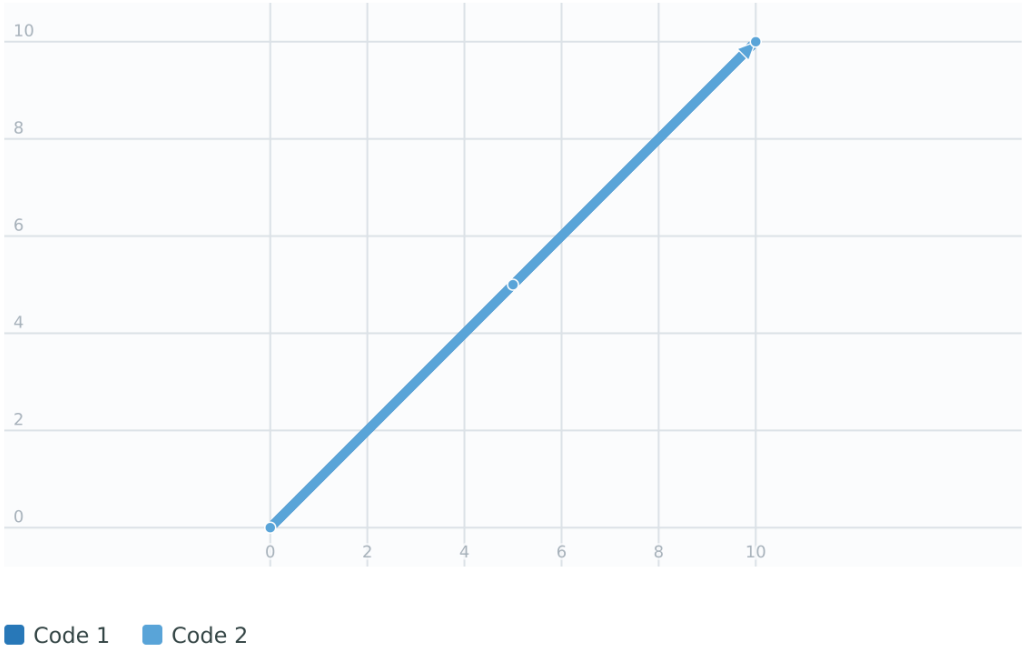
Geändert: 2.2.0 Gibt auch bei ungültigen Geometrien true zurück, wenn sie binär gleich sind

Beispiele

Code

```
SELECT ST_Equals('LINESTRING(0 0,10 10)',
'LINESTRING(0 0,5 5,10 10)');
```

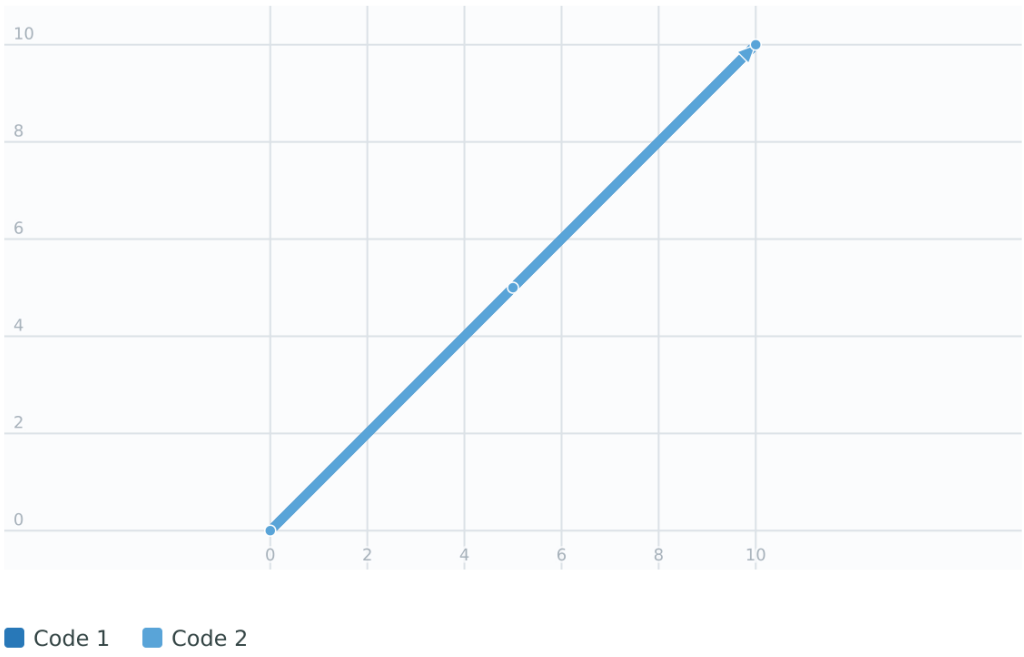
Figure



Code

```
SELECT ST_Equals(ST_Reverse('LINESTRING(0 0,10 10)'),
'LINESTRING(0 0,5 5,10 10)');
```

Figure



Siehe auch

ST_IsValid, ST_OrderingEquals, ST_Reverse, ST_Within

7.11.1.9 ST_Intersects

ST_Intersects — Prüft, ob sich zwei Geometrien schneiden (sie haben mindestens einen Punkt gemeinsam)

Synopsis

boolean **ST_Intersects**(geometry geomA , geometry geomB);
boolean **ST_Intersects**(geography geogA , geography geogB);

Beschreibung

Gibt `true` zurück, wenn sich zwei Geometrien überschneiden. Geometrien überschneiden sich, wenn sie einen Punkt gemeinsam haben.

Für die Geografie wird eine Entfernungstoleranz von 0,00001 Metern verwendet (d. h. Punkte, die sehr nahe beieinander liegen, werden als sich überschneidend betrachtet).

Mathematisch ausgedrückt: *ST_Intersects(A, B)* $\Leftrightarrow A \bigcap B \neq \varnothing$

Geometrien überschneiden sich, wenn ihre DE-9IM Schnittpunktmatrix mit einer der folgenden übereinstimmt:

- `T********`
- `*T*******`

- ***T*****
- ****T****

Eine räumliche Überschneidung wird von allen anderen Tests für räumliche Beziehungen impliziert, mit Ausnahme von ST_Disjoint, das prüft, dass sich Geometrien NICHT überschneiden.



Anmerkung
Diese Funktion beinhaltet automatisch einen Bounding-Box-Vergleich, der alle räumlichen Indizes verwendet, die für die Geometrien verfügbar sind.

Geändert: 3.0.0 SFCGAL Version entfernt und native Unterstützung für 2D TINS hinzugefügt.

Verbessert: 2.5.0 Unterstützt GEOMETRYCOLLECTION.

Verbessert: 2.3.0 Verbesserung des PIP-Kurzschlusses erweitert um die Unterstützung von MultiPoints mit wenigen Punkten. Frühere Versionen unterstützten nur Punkte in Polygonen.

Wird vom GEOS-Modul (für Geometrie) durchgeführt, Geographie ist nativ

Verfügbarkeit: Mit Version 1.5 wurde die Unterstützung für Geografie eingeführt.



Anmerkung
Für die Geografie hat diese Funktion eine Entfernungstoleranz von etwa 0,00001 Metern und verwendet die Kugelstatt der Sphäroidberechnung.



Anmerkung
HINWEIS: Dies ist die "zulässige" Version, die einen booleschen Wert und keine ganze Zahl zurückgibt.

 Diese Methode implementiert die 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))

 Diese Methode setzt die SQL/MM-Spezifikation um. SQL-MM 3: 5.1.27

 Diese Methode unterstützt kreisförmige Strings und Kurven.

 Diese Funktion unterstützt Dreiecke und dreieckige unregelmäßige Netzoberflächen (TIN).

Beispiele mit dem geometrischen Datentyp

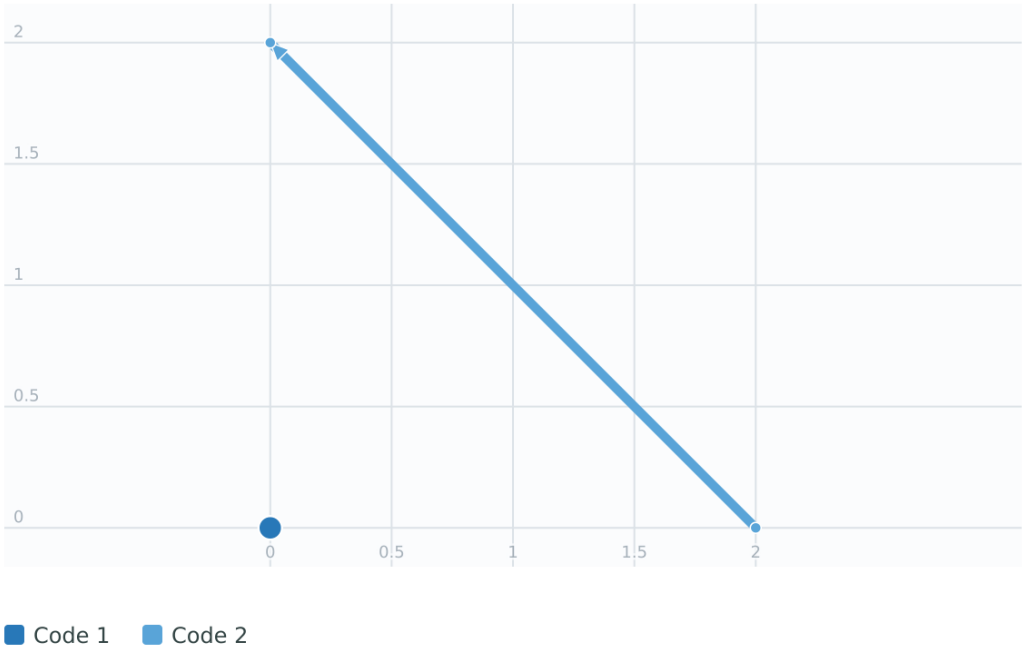
Code

```
SELECT ST_Intersects('POINT(0 0)'::geometry, 'LINESTRING ( 2 0,0 2 )'::geometry);
```

Output

```
f
```

Figure



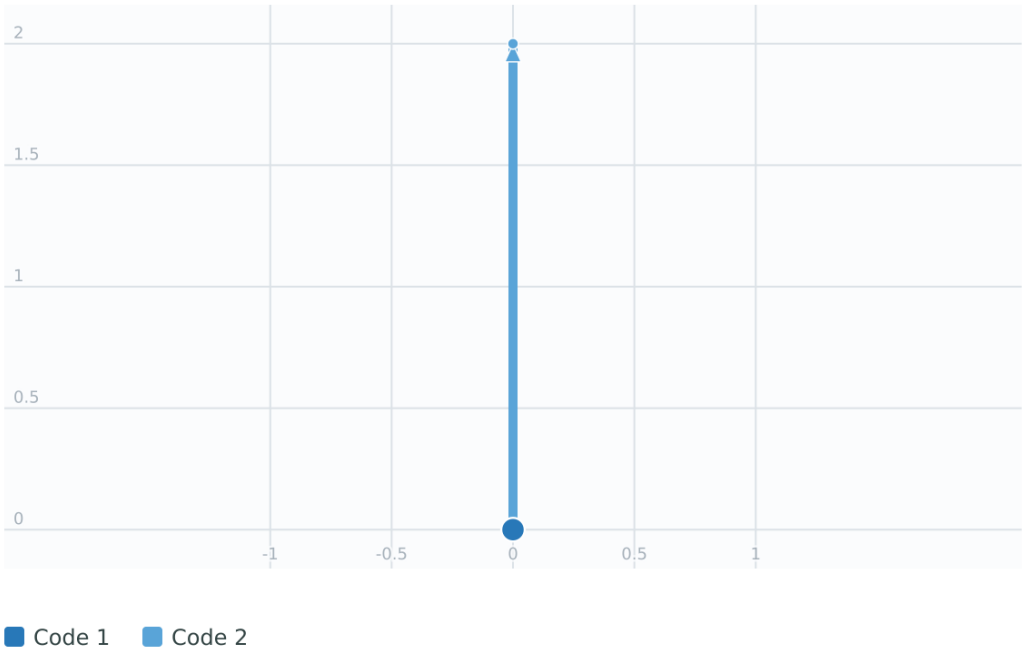
Code

```
SELECT ST_Intersects('POINT(0 0)'::geometry, 'LINESTRING ( 0 0,0 2 )'::geometry);
```

Output

```
t
```

Figure



This example looks up rows in a table; for faster lookup, the geometry column should have a GiST index.

Code

```
SELECT id, name FROM cities WHERE ST_Intersects(geom, 'SRID=4326;POLYGON((28 53,27.707 ↩
    52.293,27 52,26.293 52.293,26 53,26.293 53.707,27 54,27.707 53.707,28 53))');
```

Output

```
id | name
----+-------
  2 | Minsk
(1 row)
```

Beispiele für den geographischen Datentyp

Code

```
SELECT ST_Intersects('SRID=4326;LINESTRING(-43.23456 72.4567,-43.23456 72.4568)'::geography,
'SRID=4326;POINT(-43.23456 72.4567772)'::geography
);
```

Output

```
t
```

Siehe auch

&&, ST_3DIntersects, ST_Disjoint

7.11.1.10 ST_LineCrossingDirection

ST_LineCrossingDirection — Gibt eine Zahl zurück, die das Kreuzungsverhalten von zwei LineStrings angibt

Synopsis

integer **ST_LineCrossingDirection**(geometry linestringA, geometry linestringB);

Beschreibung

Bei zwei Linienstrings wird eine ganze Zahl zwischen -3 und 3 zurückgegeben, die angibt, welche Art von Kreuzungsverhalten zwischen ihnen besteht. 0 bedeutet keine Kreuzung. Dies wird nur für `LINESTRING`s unterstützt.

Die Kreuzungsnummer hat die folgende Bedeutung:

- 0: LINIE KEIN KREUZ
- -1: LINIENKREUZ LINKS
- 1: LINIE QUER RECHTS
- -2: LINIE MULTIKREUZ ENDE LINKS
- 2: LINIE MULTICROSS ENDE RECHTS
- -3: LINIE MULTIKREUZ ENDE GLEICH ERSTE LINKS
- 3: LINIE MULTICROSS ENDE GLEICH ERSTE RECHTS

Verfügbarkeit: 1.4

Beispiele

Beispiel: LINIENKREUZUNG LINKS und LINIENKREUZUNG RECHTS

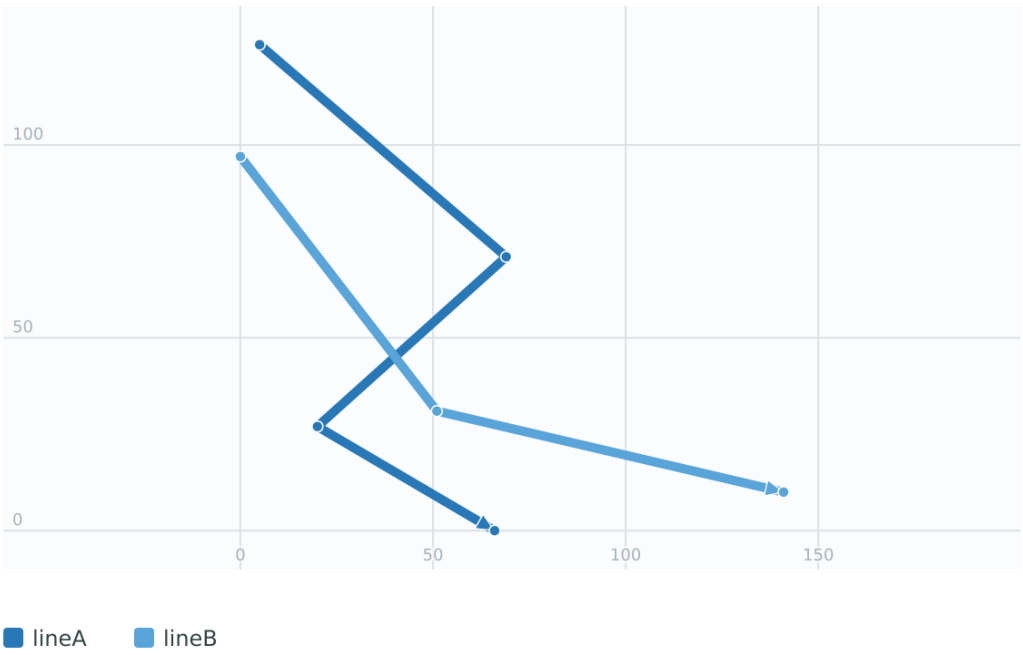
Code

```
SELECT ST_LineCrossingDirection(lineA, lineB) As A_cross_B,
       ST_LineCrossingDirection(lineB, lineA) As B_cross_A
FROM (SELECT
  'LINESTRING(25 169,89 114,40 70,86 43)'::geometry As lineA,
  'LINESTRING (20 140,71 74,161 53)'::geometry As lineB
  ) As foo;
```

Output

```
A_cross_B | B_cross_A
----------+----------
       -1 |         1
```

Figure



Beispiel: LINE MULTICROSS END SAME FIRST LEFT und LINE MULTICROSS END SAME FIRST RIGHT

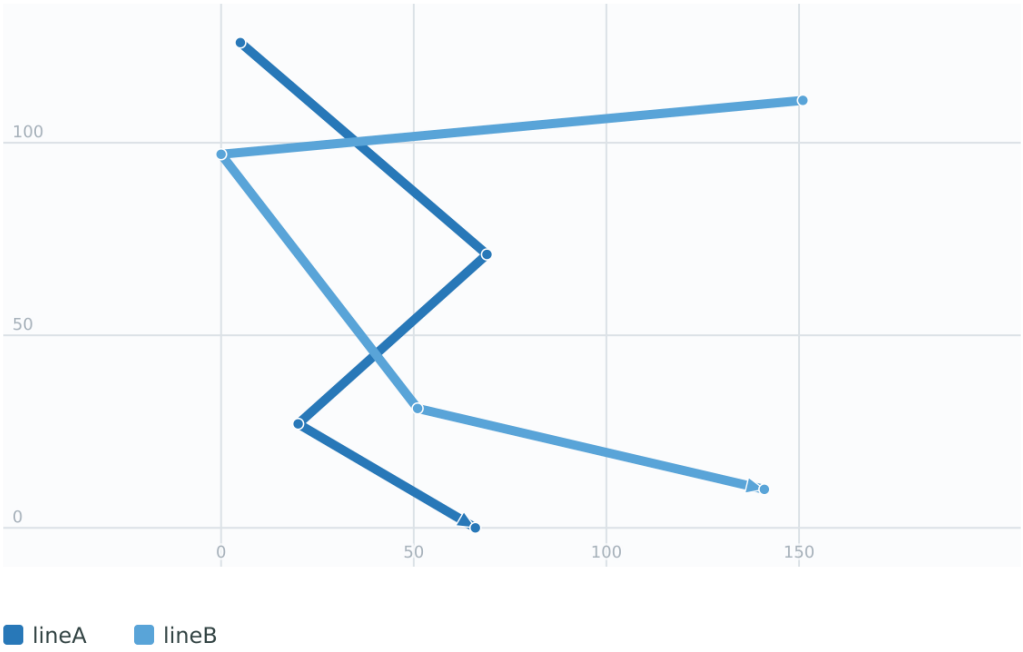
Code

```
SELECT ST_LineCrossingDirection(lineA, lineB) As A_cross_B,
       ST_LineCrossingDirection(lineB, lineA) As B_cross_A
FROM (SELECT
 'LINESTRING(25 169,89 114,40 70,86 43)'::geometry As lineA,
 'LINESTRING(171 154,20 140,71 74,161 53)'::geometry As lineB
  ) As foo;
```

Output

```
A_cross_B | B_cross_A
----------+----------
        3 |        -3
```

Figure



Beispiel: LINE MULTICROSS END LEFT und LINE MULTICROSS END RIGHT

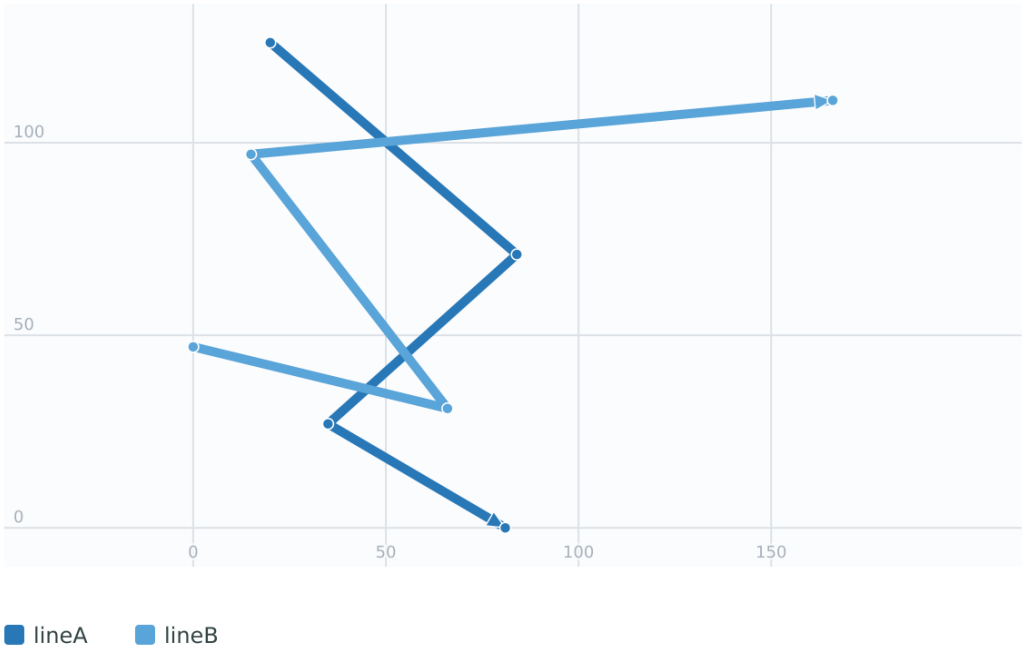
Code

```
SELECT ST_LineCrossingDirection(lineA, lineB) As A_cross_B,
       ST_LineCrossingDirection(lineB, lineA) As B_cross_A
FROM (SELECT
  'LINESTRING(25 169,89 114,40 70,86 43)'::geometry As lineA,
  'LINESTRING(5 90,71 74,20 140,171 154)'::geometry As lineB
  ) As foo;
```

Output

```
A_cross_B | B_cross_A
-----------+-----------
        -2 |         2
```

Figure



Beispiel: Findet alle Straßen, die sich kreuzen

Code

```
SELECT s1.gid, s2.gid, ST_LineCrossingDirection(s1.geom, s2.geom)
  FROM streets s1 CROSS JOIN streets s2
        ON (s1.gid != s2.gid AND s1.geom && s2.geom )
WHERE ST_LineCrossingDirection(s1.geom, s2.geom)
> 0;
```

Siehe auch

ST_Crosses

7.11.1.11 ST_OrderingEquals

ST_OrderingEquals — Prüft, ob zwei Geometrien die gleiche Geometrie darstellen und Punkte in der gleichen Richtungsreihenfolge haben

Synopsis

boolean **ST_OrderingEquals**(geometry A, geometry B);

Beschreibung

ST_OrderingEquals vergleicht zwei Geometrien und gibt t (TRUE) zurück, wenn die Geometrien gleich sind und die Koordinaten in der gleichen Reihenfolge sind; andernfalls gibt es f (FALSE) zurück.

Anmerkung
Diese Funktion ist gemäß der ArcSDE-SQL-Spezifikation und nicht gemäß SQL-MM implementiert. http://edndoc.esri.com/arcsde/9.1/sql_api/sqlapi3.htm#ST_OrderingEquals

Diese Methode setzt die SQL/MM-Spezifikation um. SQL-MM 3: 5.1.43

Beispiele

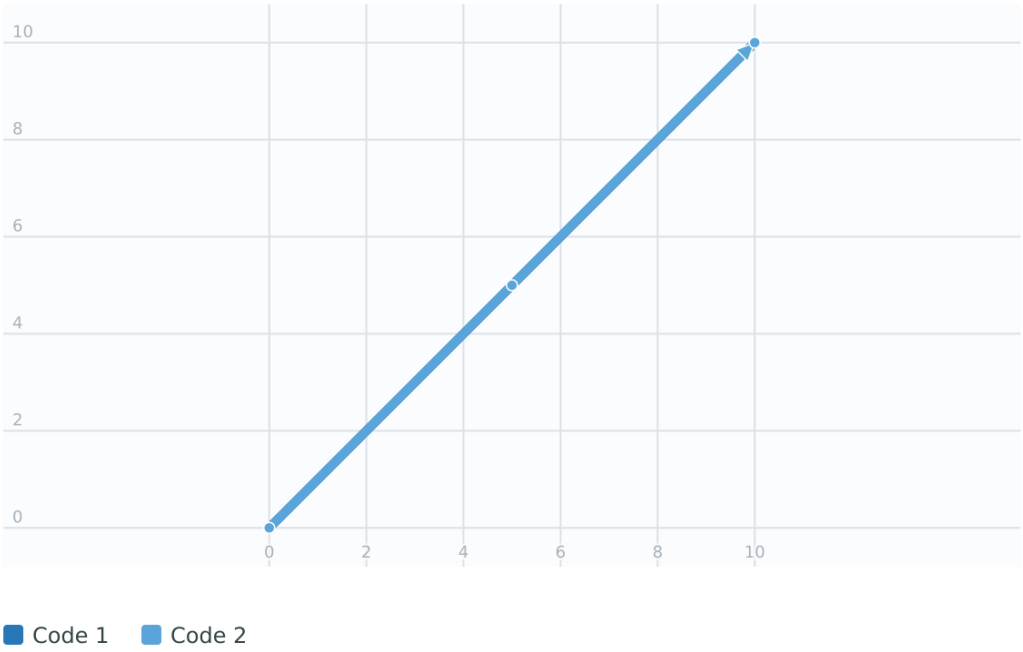
Code

```
SELECT ST_OrderingEquals('LINESTRING(0 0,10 10)',
'LINESTRING(0 0,5 5,10 10)');
```

Output

```
f
```

Figure



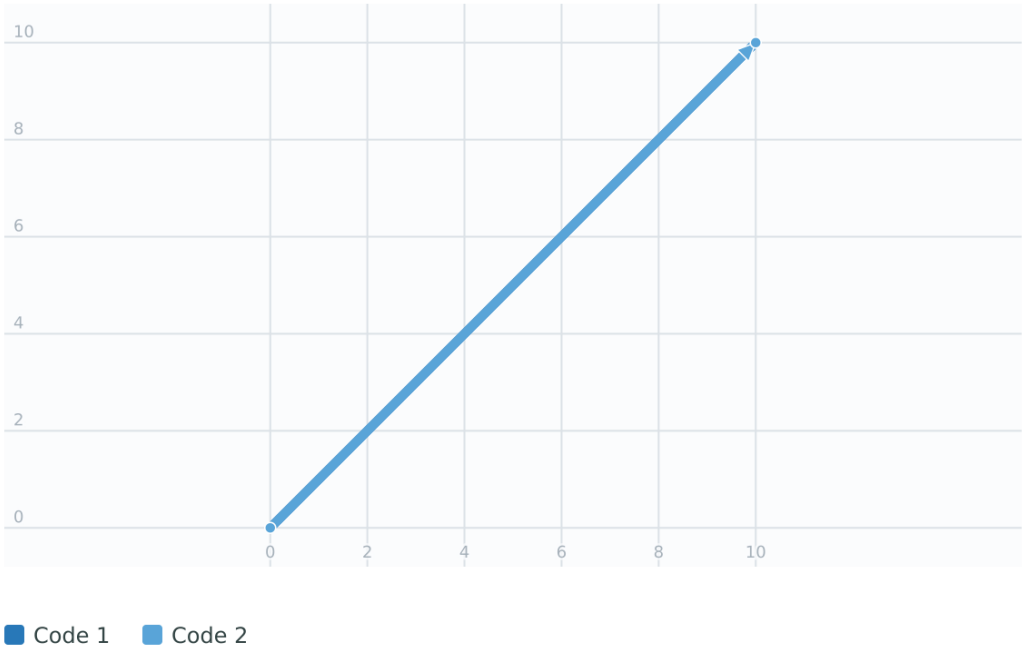
Code

```
SELECT ST_OrderingEquals('LINESTRING(0 0,10 10)',
'LINESTRING(0 0,10 10)');
```

Output

```
t
```

Figure



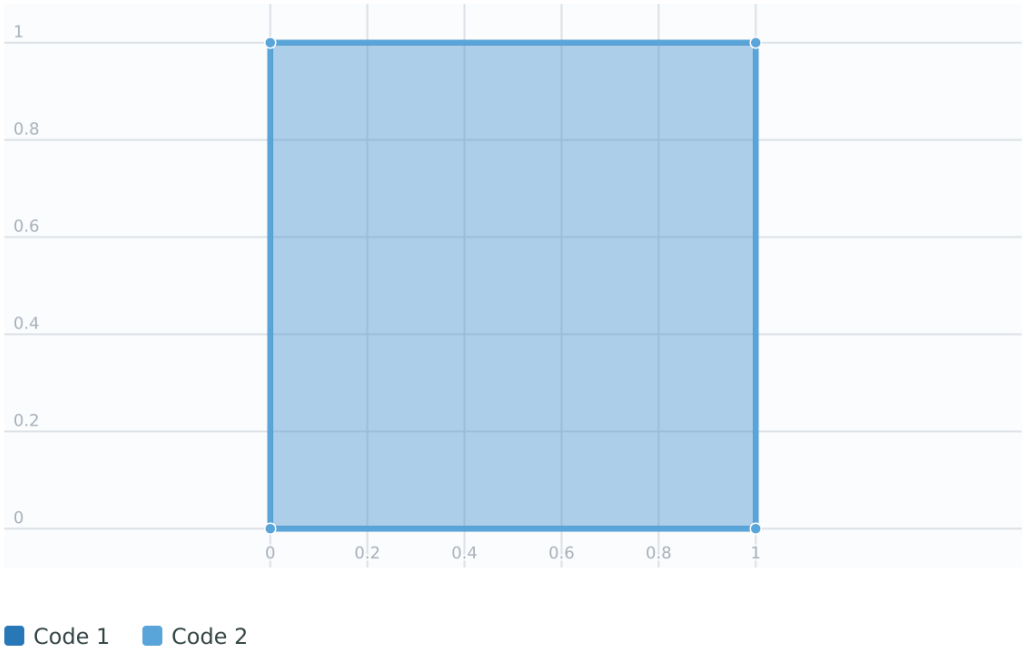
Code

```
SELECT ST_OrderingEquals('POLYGON((0 0,0 1,1 1,1 0,0 0))',
'POLYGON((0 0,1 0,1 1,0 1,0 0))');
```

Output

```
f
```

Figure



Siehe auch

&&, ST_Equals, ST_Reverse

7.11.1.12 ST_Overlaps

ST_Overlaps — Prüft, ob zwei Geometrien die gleiche Abmessung haben und sich schneiden, aber jede mindestens einen Punkt hat, der nicht in der anderen liegt

Synopsis

boolean **ST_Overlaps**(geometry A, geometry B);

Beschreibung

Gibt TRUE zurück, wenn sich Geometrie A und B "räumlich überschneiden". Zwei Geometrien überlappen sich, wenn sie dieselbe Dimension haben, ihre Innenräume sich in dieser Dimension schneiden und jede Geometrie mindestens einen Punkt im Inneren der anderen hat (oder äquivalent dazu, keine der beiden Geometrien die andere überdeckt). Die Überlappungsbeziehung ist symmetrisch und irreflexiv.

Mathematisch ausgedrückt: $ST_Overlaps(A, B) \Leftrightarrow (dim(A) = dim(B) = dim(Int(A) \bigcap Int(B))) \wedge (A \bigcap B \neq A) \wedge (A \bigcap B \neq B)$



Anmerkung
Diese Funktion beinhaltet automatisch einen Bounding-Box-Vergleich, der alle räumlichen Indizes verwendet, die für die Geometrien verfügbar sind. Um die Verwendung des Index zu vermeiden, verwenden Sie die Funktion `_ST_Overlaps`.

Wird durch das GEOS Modul ausgeführt

Verbessert: 3.0.0 ermöglicht die Unterstützung von `GEOMETRYCOLLECTION`

HINWEIS: Dies ist die "zulässige" Version, die einen booleschen Wert und keine ganze Zahl zurückgibt.

 Diese Methode implementiert die OGC Simple Features Implementation Specification for SQL 1.1. s2.1.1.2 // s2.1.13.3

 Diese Methode setzt die SQL/MM-Spezifikation um. SQL-MM 3: 5.1.32

Beispiele

`ST_Overlaps` gibt `TRUE` in den folgenden Situationen zurück:

Two MultiPoints overlap when they share some, but not all, points.

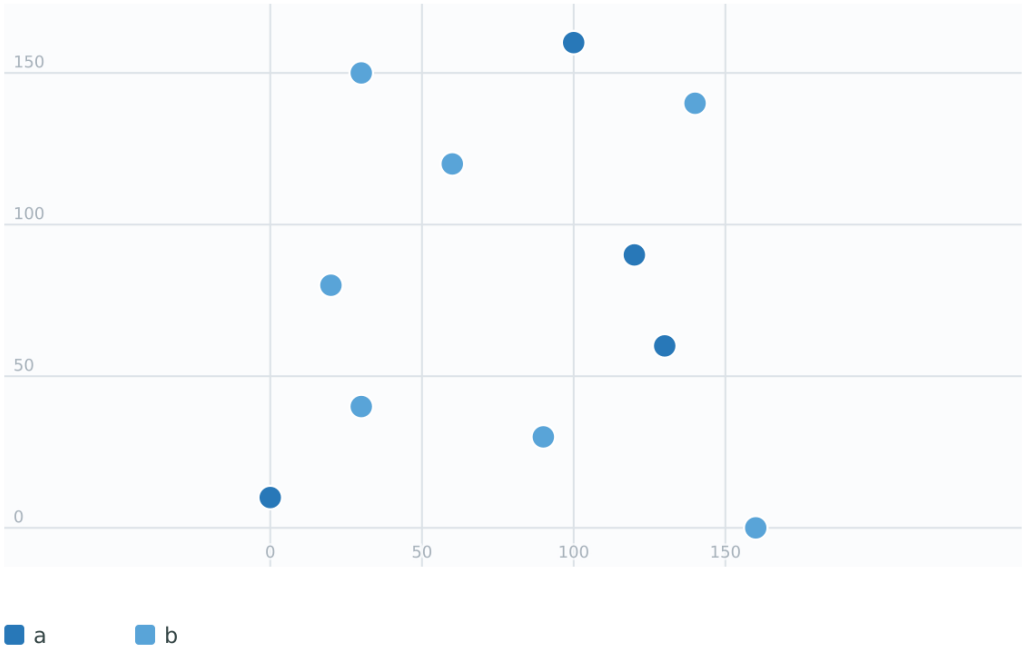
Code

```
SELECT ST_Overlaps(a, b)
FROM (SELECT
  'MULTIPOINT ((20 20),(40 90),(80 130),(140 100),(110 40),(120 170),(150 70))'::geometry AS ↩
    a,
  'MULTIPOINT ((180 10),(40 90),(80 130),(160 150),(110 40),(50 50),(50 160))'::geometry AS  ↩
    b
) AS example;
```

Output

```
t
```

Figure



Two LineStrings overlap along part of their paths.

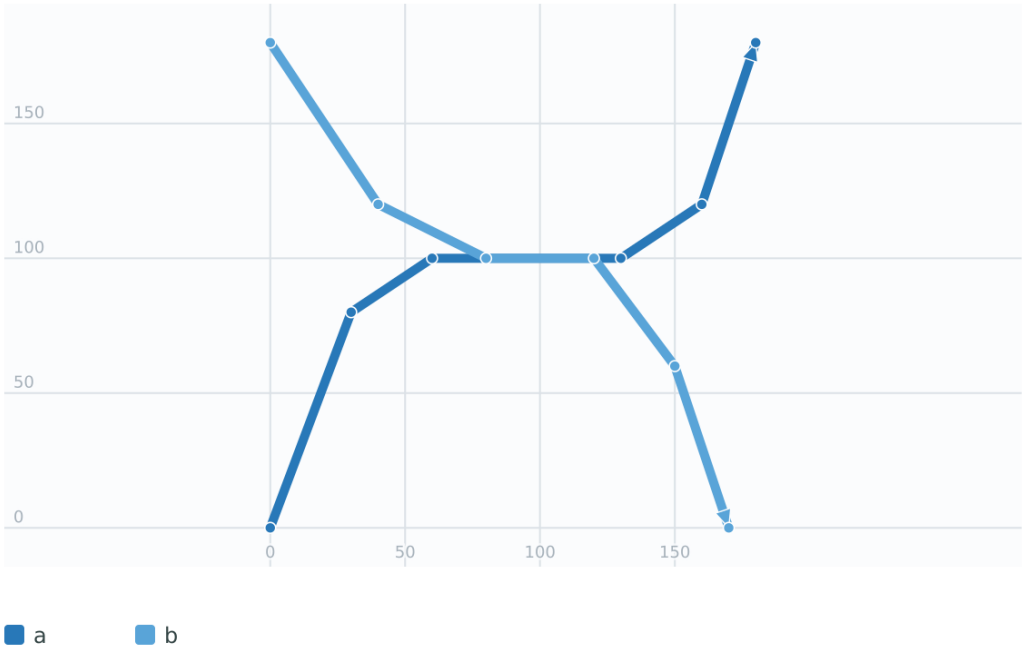
Code

```
SELECT ST_Overlaps(a, b)
FROM (SELECT
  'LINESTRING (10 10,40 90,70 110,140 110,170 130,190 190)'::geometry AS a,
  'LINESTRING (10 190,50 130,90 110,130 110,160 70,180 10)'::geometry AS b
) AS example;
```

Output

```
t
```

Figure



Two Polygons overlap where their interiors intersect and neither contains the other.

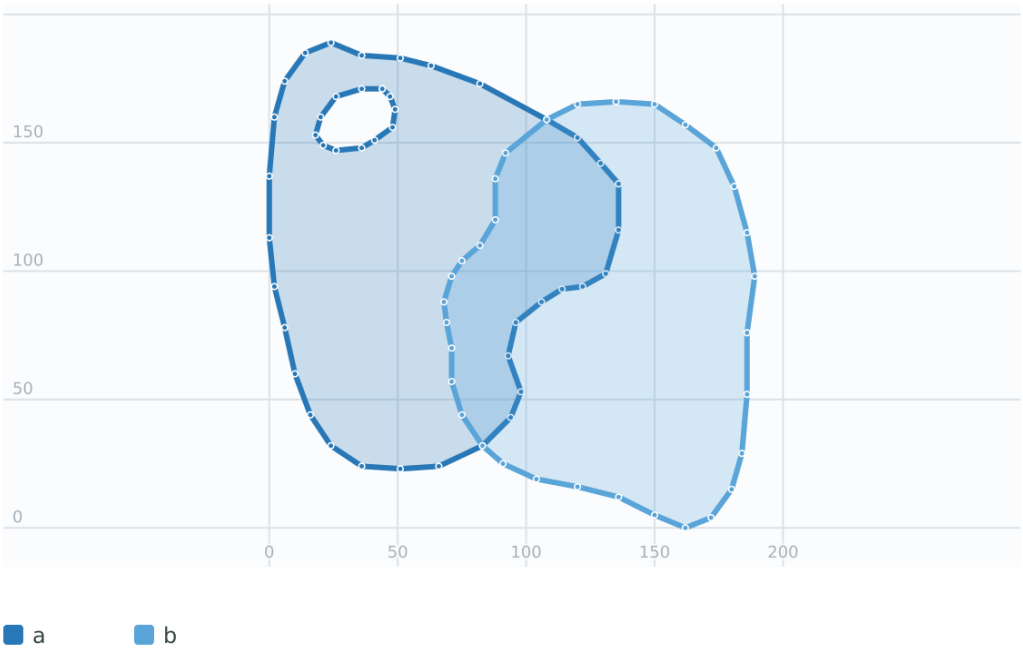
Code

```
SELECT ST_Overlaps(a, b)
FROM (SELECT
  'POLYGON ((140 140,140 122,135 105,126 100,118 99,110 94,100 86,97 73,102 59,98 49,87 ↩
    38,70 30,55 29,40 30,28 38,20 50,14 66,10 84,6 100,4 119,4 143,6 166,10 180,18 191,28 ↩
    195,40 190,55 189,67 186,86 179,112 165,124 158,133 148,140 140),(48 177,40 177,30 ↩
    174,24 166,22 159,25 155,30 153,40 154,45 157,52 162,53 169,51 174,48 177))'::geometry ↩
    AS a,
  'POLYGON ((75 63,79 50,87 38,95 31,108 25,124 22,140 18,154 11,166 6,176 10,184 21,188 ↩
    35,190 58,190 82,193 104,190 121,185 139,178 154,166 163,154 171,139 172,124 171,112 ↩
    165,96 152,92 142,92 126,86 116,79 110,75 104,72 94,73 86,75 76,75 63))'::geometry AS b
) AS example;
```

Output

```
t
```

Figure



Ein Punkt auf einem LineString ist enthalten, aber da er eine geringere Dimension hat, überschneidet oder kreuzt er sich nicht.

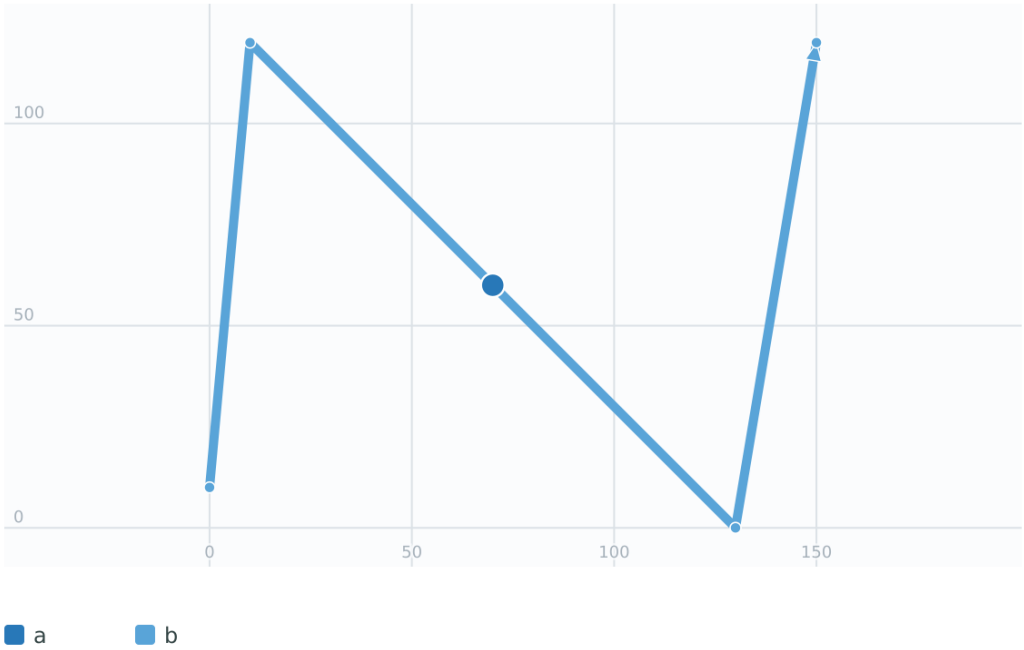
Code

```
SELECT ST_Overlaps(a, b) AS overlaps, ST_Crosses(a, b) AS crosses,
       ST_Intersects(a, b) AS intersects, ST_Contains(b, a) AS b_contains_a
FROM (SELECT 'POINT (100 100)'::geometry As a,
             'LINESTRING (30 50,40 160,160 40,180 160)'::geometry  AS b) AS t
```

Output

```
overlaps | crosses | intersects | b_contains_a
---------+---------+------------+--------------
f        | f       | t          | t
```

Figure



Ein LineString, der ein Polygon teilweise abdeckt, schneidet und kreuzt sich, überschneidet sich aber nicht, da er unterschiedliche Abmessungen hat.

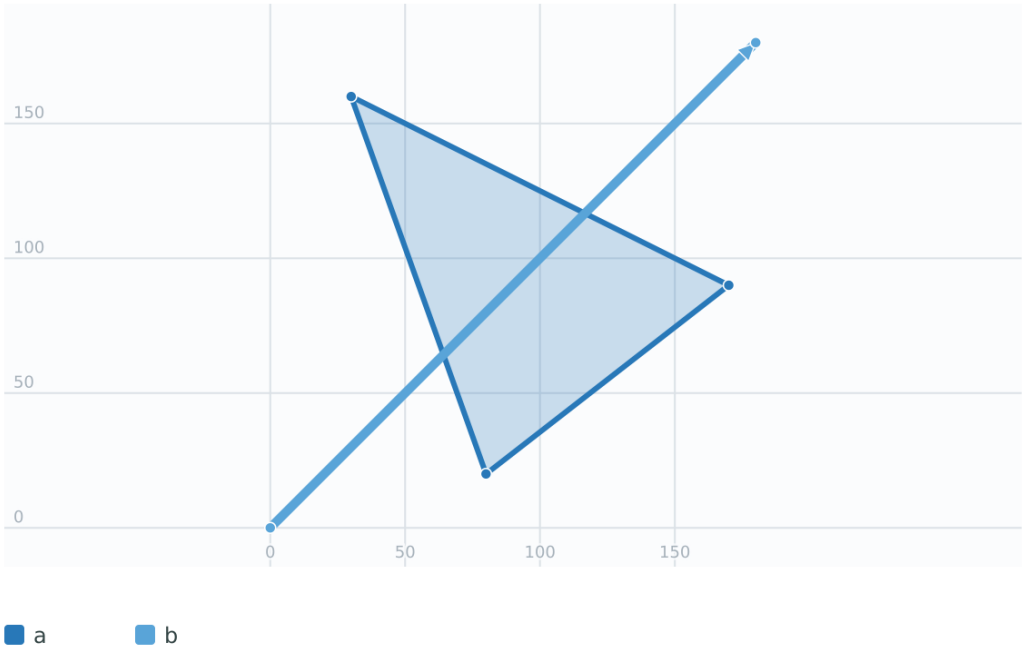
Code

```
SELECT ST_Overlaps(a, b) AS overlaps, ST_Crosses(a, b) AS crosses,
       ST_Intersects(a, b) AS intersects, ST_Contains(a, b) AS contains
FROM (SELECT 'POLYGON ((40 170,90 30,180 100,40 170))'::geometry AS a,
             'LINESTRING(10 10,190 190)'::geometry AS b) AS t;
```

Output

```
overlap | crosses | intersects | contains
---------+---------+------------+--------------
 f       | t       | t          | f
```

Figure



Zwei Polygone, die sich schneiden, ohne dass das eine vom anderen enthalten ist, überschneiden sich, aber nicht, weil ihr Schnittpunkt die gleiche Dimension hat.

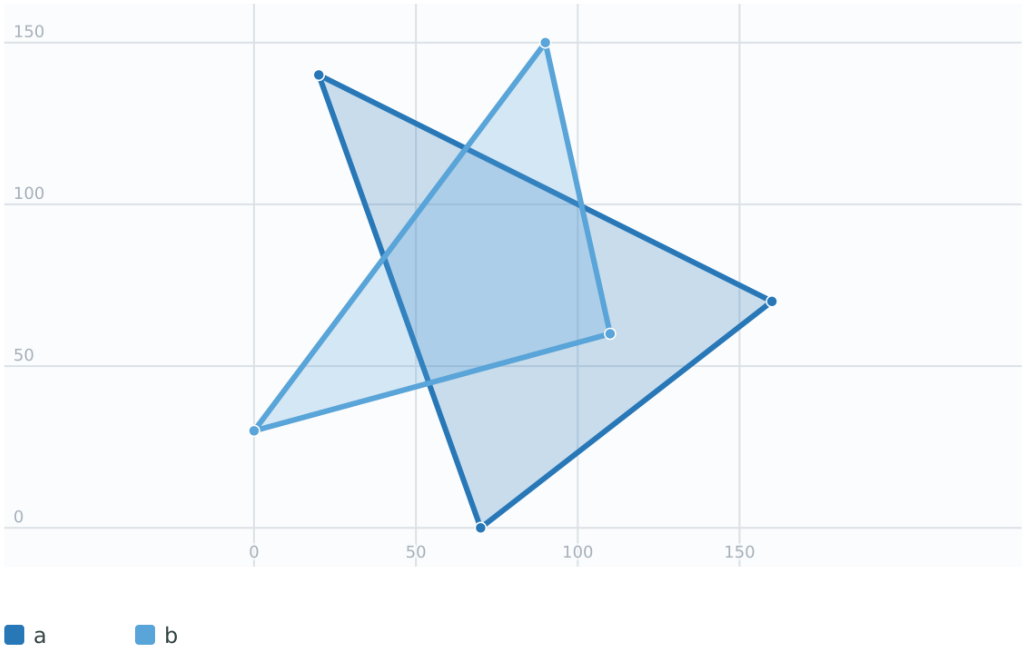
Code

```
SELECT ST_Overlaps(a, b) AS overlaps, ST_Crosses(a, b) AS crosses,
       ST_Intersects(a, b) AS intersects, ST_Contains(b, a) AS b_contains_a,
       ST_Dimension(a) AS dim_a, ST_Dimension(b) AS dim_b,
       ST_Dimension(ST_Intersection(a, b)) AS dim_int
FROM (SELECT 'POLYGON ((40 170,90 30,180 100,40 170))'::geometry AS a,
             'POLYGON ((110 180,20 60,130 90,110 180))'::geometry AS b) As t;
```

Output

```
overlaps | crosses | intersects | b_contains_a | dim_a | dim_b | dim_int
---------+---------+------------+--------------+-------+-------+----------
 t       | f       | t          | f            |     2 |     2 |       2
```

Figure



Siehe auch

ST_Contains, ST_Crosses, ST_Dimension, ST_Intersects

7.11.1.13 ST_Relate

ST_Relate — Prüft, ob zwei Geometrien eine topologische Beziehung haben, die einem Schnittpunktmatrixmuster entspricht, oder berechnet ihre Schnittpunktmatrix

Synopsis

boolean **ST_Relate**(geometry geomA, geometry geomB, text intersectionMatrixPattern);
text **ST_Relate**(geometry geomA, geometry geomB);
text **ST_Relate**(geometry geomA, geometry geomB, integer boundaryNodeRule);

Beschreibung

Diese Funktionen ermöglichen die Prüfung und Bewertung der räumlichen (topologischen) Beziehung zwischen zwei Geometrien, wie sie durch das Dimensionally Extended 9-Intersection Model (DE-9IM) definiert sind.

DE-9IM wird als 9-Element-Matrix spezifiziert, die die Dimension der Schnittpunkte zwischen dem Inneren, dem Rand und dem Äußeren von zwei Geometrien angibt. Sie wird durch eine 9-stellige Zeichenkette mit den Symbolen "F", "0", "1", "2" dargestellt (z. B. `"FF1FF0102"`).

Eine bestimmte Art von räumlicher Beziehung kann getestet werden, indem die Kreuzungsmatrix mit einem Muster der *Kreuzungsmatrix* verglichen wird. Muster können die zusätzlichen Symbole 'T' (bedeutet "Schnittpunkt ist nicht leer") und '*' (bedeutet "beliebiger Wert") enthalten. Allgemeine räumliche Beziehungen werden durch die benannten Funktionen ST_Contains, ST_ContainsProperly, ST_Covers, ST_CoveredBy, ST_Crosses, ST_Disjoint, ST_Equals, ST_Intersects, ST_Overlaps, ST_Touches und ST_Within bereitgestellt. Die Verwendung eines expliziten Musters ermöglicht es, mehrere Bedingungen für Überschneidungen, Kreuzungen usw. in einem Schritt zu testen. Es ermöglicht auch das Testen räumlicher Beziehungen, die keine benannte räumliche Beziehungsfunktion haben. Zum Beispiel hat die Beziehung "Interior-Intersects" das DE-9IM-Muster `T********`, das von keinem benannten Prädikat ausgewertet wird.

Weitere Informationen finden Sie unter Abschnitt 5.1.

Variante 1: Prüft, ob zwei Geometrien nach dem vorgegebenen `intersectionMatrixPattern` räumlich zusammenhängen.



Anmerkung
Im Gegensatz zu den meisten benannten räumlichen Beziehungsprädikaten enthält dieses NICHT automatisch einen Indexaufruf. Der Grund dafür ist, dass einige Beziehungen für Geometrien wahr sind, die sich NICHT schneiden (z. B. Disjoint). Wenn Sie ein Beziehungsmuster verwenden, das eine Überschneidung erfordert, dann schließen Sie den Indexaufruf && ein.



Anmerkung
Es ist besser, eine benannte Beziehungsfunktion zu verwenden, wenn sie verfügbar ist, da sie automatisch einen räumlichen Index verwendet, wenn einer vorhanden ist. Außerdem können sie Leistungsoptimierungen implementieren, die bei einer vollständigen Beziehungsauswertung nicht verfügbar sind.

Variante 2: Gibt die DE-9IM-Matrixzeichenfolge für die räumliche Beziehung zwischen den beiden Eingabegeometrien zurück. Die Matrixzeichenfolge kann mit ST_RelateMatch auf Übereinstimmung mit einem DE-9IM-Muster getestet werden.

Variante 3: Wie Variante 2, erlaubt aber die Angabe einer **Boundary Node Rule**. Eine Boundary Node Rule erlaubt eine feinere Kontrolle darüber, ob die Endpunkte von MultiLineStrings als im DE-9IM Interior oder Boundary liegend betrachtet werden. Die `boundaryNodeRule` Werte sind:

- `1`: **OGC-Mod2** - Linienendpunkte befinden sich in der Boundary, wenn sie eine ungerade Anzahl von Malen vorkommen. Dies ist die vom OGC SFS-Standard definierte Regel und ist die Vorgabe für `ST_Relate`.
- `2`: **Endpunkt** - alle Endpunkte befinden sich in der Boundary.
- `3`: **MultivalentEndpoint** - Endpunkte befinden sich in der Boundary, wenn sie mehr als einmal vorkommen. Mit anderen Worten, die Grenze sind alle "verbundenen" oder "inneren" Endpunkte (aber nicht die "unverbundenen/äußeren").
- `4`: **MonovalenterEndpunkt** - Endpunkte befinden sich in der Boundary, wenn sie nur einmal vorkommen. Mit anderen Worten, die Begrenzung sind alle "unverbundenen" oder "äußeren" Endpunkte.

Diese Funktion ist nicht in der OGC-Spezifikation enthalten, wird aber impliziert. siehe s2.1.13.2

Diese Methode implementiert die OGC Simple Features Implementation Specification for SQL 1.1. s2.1.1.2 // s2.1.13.3

Diese Methode setzt die SQL/MM-Spezifikation um. SQL-MM 3: 5.1.25

Wird durch das GEOS Modul ausgeführt

Verbessert: 2.0.0 - Unterstützung für die Angabe von Grenzknotenregeln hinzugefügt.

Verbessert: 3.0.0 ermöglicht die Unterstützung von `GEOMETRYCOLLECTION`

Beispiele

Verwendung der booleschen Funktion zur Prüfung räumlicher Beziehungen.

Code

```
SELECT ST_Relate('POINT(1 2)', ST_Buffer('POINT(1 2)', 2), '0FFFFF212');
```

Output

```
t
```

Code

```
SELECT ST_Relate('POINT(1 2)', ST_Buffer('POINT(1 2)', 2), '*FF*FF212');
```

Output

```
t
```

Testen eines benutzerdefinierten räumlichen Beziehungsmusters als Abfragebedingung, mit `&&` , um die Verwendung eines räumlichen Indexes zu ermöglichen.

Find compounds that properly intersect (not just touch) a poly (Interior Intersects)

Code

```
SELECT c.* , p.name As poly_name
    FROM polys AS p
    INNER JOIN compounds As c
          ON c.geom && p.geom
             AND ST_Relate(p.geom, c.geom, 'T********');
```

Berechnung der Kreuzungsmatrix für räumliche Beziehungen.

Code

```
SELECT ST_Relate('POINT(1 2)',
ST_Buffer('POINT(1 2)', 2));
```

Output

```
0FFFFF212
```

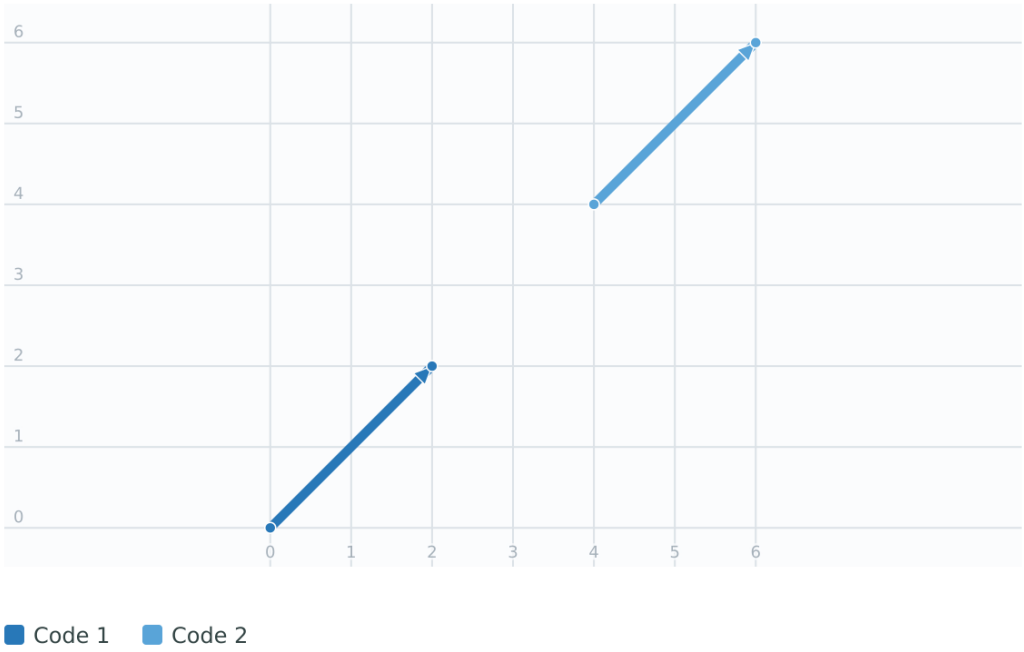
Code

```
SELECT ST_Relate('LINESTRING(1 2,3 4)',
'LINESTRING(5 6,7 8)' );
```

Output

```
FF1FF0102
```

Figure



Verwendung verschiedener Boundary Node Rules zur Berechnung der räumlichen Beziehung zwischen einem LineString und einem MultiLineString mit einem doppelten Endpunkt `(3 3)`:

- Unter Verwendung der **OGC-Mod2** Regel (1) liegt der doppelte Endpunkt im **Inneren** des MultiLineString, so dass der DE-9IM Matrixeintrag [aB:bI] `0` und [aB:bB] `F` ist.
- Bei Anwendung der Regel **Endpunkt** (2) liegt der doppelte Endpunkt in der **Grenze** des MultiLineString, so dass der DE-9IM Matrixeintrag [aB:bI] `F` und [aB:bB] `0` ist.

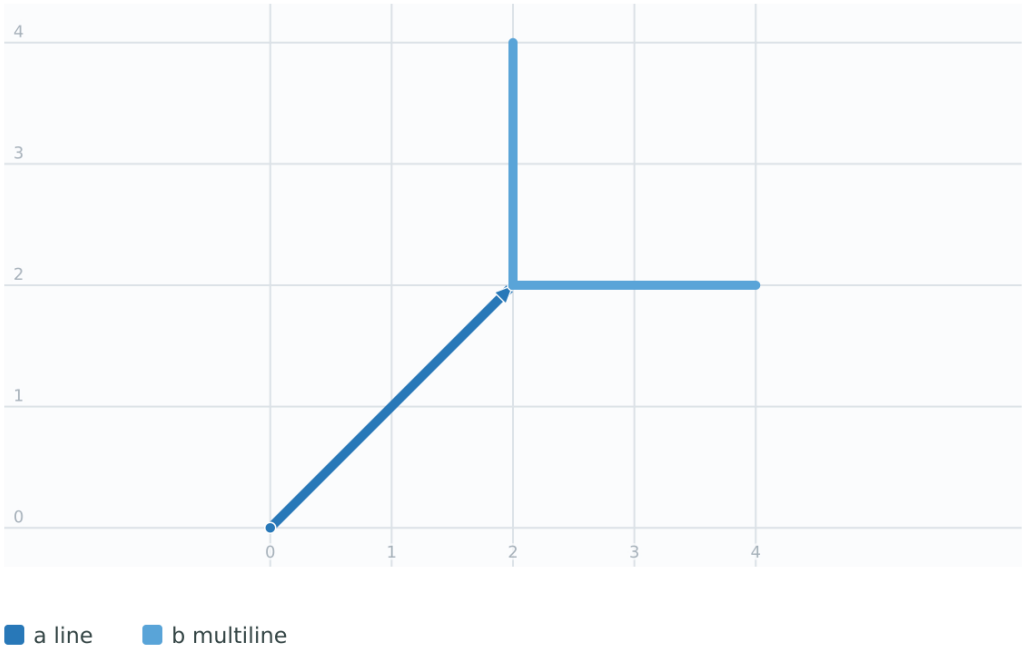
Code

```
WITH data AS (SELECT
  'LINESTRING(1 1,3 3)'::geometry AS a_line,
  'MULTILINESTRING((3 3,3 5),(3 3,5 3))'::geometry AS b_multiline
)
SELECT ST_Relate(a_line, b_multiline, 1) AS bnr_mod2,
       ST_Relate(a_line, b_multiline, 2) AS bnr_endpoint
    FROM data;
```

Output

```
bnr_mod2  | bnr_endpoint
----------+-------------
 FF10F0102 | FF1F00102
```

Figure



Siehe auch

Abschnitt 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 — Testet, ob eine DE-9IM Schnittpunktmatrix mit einem Schnittpunktmuster übereinstimmt

Synopsis

boolean **ST_RelateMatch**(text intersectionMatrix, text intersectionMatrixPattern);

Beschreibung

Prüft, ob ein Dimensionally Extended 9-Intersection Model (DE-9IM) `intersectionMatrix` Wert ein `intersectionMatrixPa` erfüllt. Schnittpunktmatrixwerte können mit ST_Relate berechnet werden.

Weitere Informationen finden Sie unter Abschnitt 5.1.

Wird durch das GEOS Modul ausgeführt

Verfügbarkeit: 2.0.0

Beispiele

Code

```
SELECT ST_RelateMatch('101202FFF', 'TTTTTTFFF') ;
```

Output

```
t
```

Muster für gemeinsame räumliche Beziehungen, die mit den Werten der Kreuzungsmatrix abgeglichen werden, für eine Linie in verschiedenen Positionen relativ zu einem Polygon

Code

```
SELECT pat.name AS relationship, pat.val AS pattern,
       mat.name AS position, mat.val AS matrix,
       ST_RelateMatch(mat.val, pat.val) AS match
    FROM (VALUES ( 'Equality', 'T1FF1FFF1' ),
                 ( 'Overlaps', 'T*T***T**' ),
                 ( 'Within', 'T*F**F***' ),
                 ( 'Disjoint', 'FF*FF****' )) AS pat(name, val)
    CROSS JOIN
        (VALUES  ('non-intersecting', 'FF1FF0212'),
                 ('overlapping', '1010F0212'),
                 ('inside', '1FF0FF212')) AS mat(name, val);
```

Output

```
relationship |  pattern  |     position     |  matrix   | match
-------------+-----------+------------------+-----------+-------
 Equality    | T1FF1FFF1 | non-intersecting | FF1FF0212 | f
 Equality    | T1FF1FFF1 | overlapping      | 1010F0212 | f
 Equality    | T1FF1FFF1 | inside           | 1FF0FF212 | f
 Overlaps    | T*T***T** | non-intersecting | FF1FF0212 | f
 Overlaps    | T*T***T** | overlapping      | 1010F0212 | t
 Overlaps    | T*T***T** | inside           | 1FF0FF212 | f
 Within      | T*F**F*** | non-intersecting | FF1FF0212 | f
 Within      | T*F**F*** | overlapping      | 1010F0212 | f
 Within      | T*F**F*** | inside           | 1FF0FF212 | t
 Disjoint    | FF*FF**** | non-intersecting | FF1FF0212 | t
 Disjoint    | FF*FF**** | overlapping      | 1010F0212 | f
 Disjoint    | FF*FF**** | inside           | 1FF0FF212 | f
```

Siehe auch

Abschnitt 5.1, ST_Relate

7.11.1.15 ST_Touches

ST_Touches — Prüft, ob zwei Geometrien mindestens einen Punkt gemeinsam haben, aber ihre Innenräume sich nicht schneiden

Synopsis

boolean **ST_Touches**(geometry A, geometry B);

Beschreibung

Rückgabe `TRUE` wenn A und B sich schneiden, aber ihre Innenräume sich nicht schneiden. Äquivalent dazu haben A und B mindestens einen Punkt gemeinsam, und die gemeinsamen Punkte liegen in mindestens einer Begrenzung. Bei Punkt/Punkt-Eingaben ist die Beziehung immer `FALSE`, da Punkte keine Begrenzung haben.

In mathematical terms: $ST_Touches(A, B) \Leftrightarrow (Int(A) \cap Int(B) = \varnothing) \wedge (A \cap B \neq \varnothing)$

Diese Beziehung ist gegeben, wenn die DE-9IM-Schnittpunktmatrix für die beiden Geometrien einem der folgenden Werte entspricht:

- FT*******

- F**T*****
- F***T****



Anmerkung
Diese Funktion beinhaltet automatisch einen Bounding-Box-Vergleich, der alle räumlichen Indizes verwendet, die für die Geometrien verfügbar sind. Um die Verwendung eines Indexes zu vermeiden, verwenden Sie stattdessen `_ST_Touches`.

Verbessert: 3.0.0 ermöglicht die Unterstützung von `GEOMETRYCOLLECTION`

 Diese Methode implementiert die OGC Simple Features Implementation Specification for SQL 1.1. s2.1.1.2 // s2.1.13.3

 Diese Methode setzt die SQL/MM-Spezifikation um. SQL-MM 3: 5.1.28

Beispiele

Das Prädikat `ST_Touches` liefert `TRUE` in den folgenden Beispielen.

Two Polygons touching along a boundary segment.

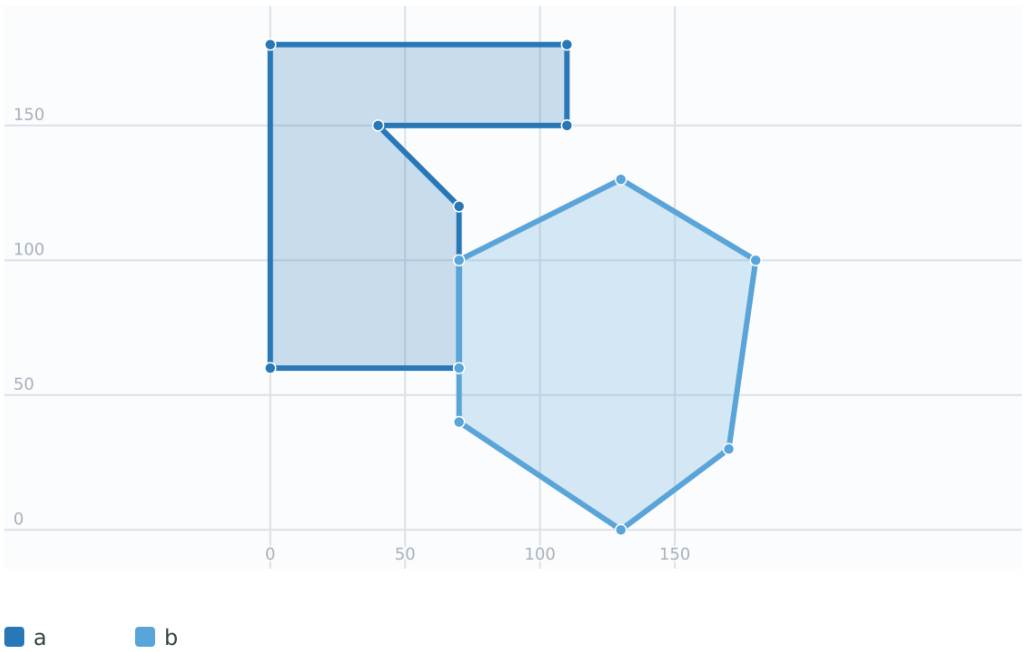
Code

```
SELECT ST_Touches(a, b)
FROM (SELECT
  'POLYGON ((10 190,10 70,80 70,80 130,50 160,120 160,120 190,10 190))'::geometry AS a,
  'POLYGON ((80 110,140 140,190 110,180 40,140 10,80 50,80 70,80 110))'::geometry AS b
) AS example;
```

Output

```
t
```

Figure



Two Polygons touching at a boundary point.

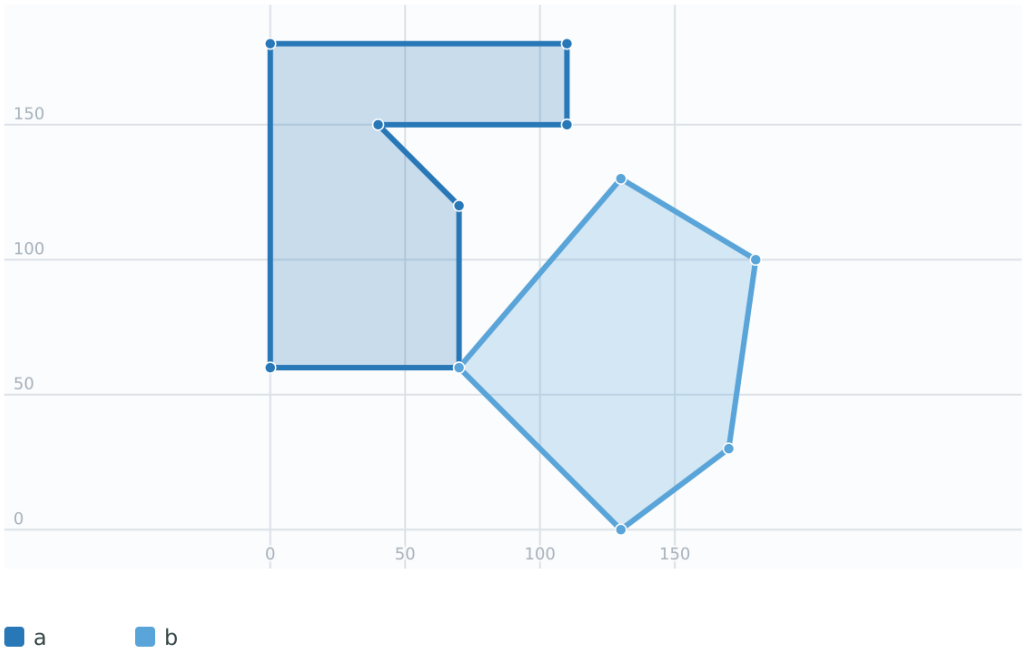
Code

```
SELECT ST_Touches(a, b)
FROM (SELECT
  'POLYGON ((10 190,10 70,80 70,80 130,50 160,120 160,120 190,10 190))'::geometry AS a,
  'POLYGON ((140 140,190 110,180 40,140 10,80 70,140 140))'::geometry AS b
) AS example;
```

Output

```
t
```

Figure



A Polygon and a LineString touching at the line endpoint.

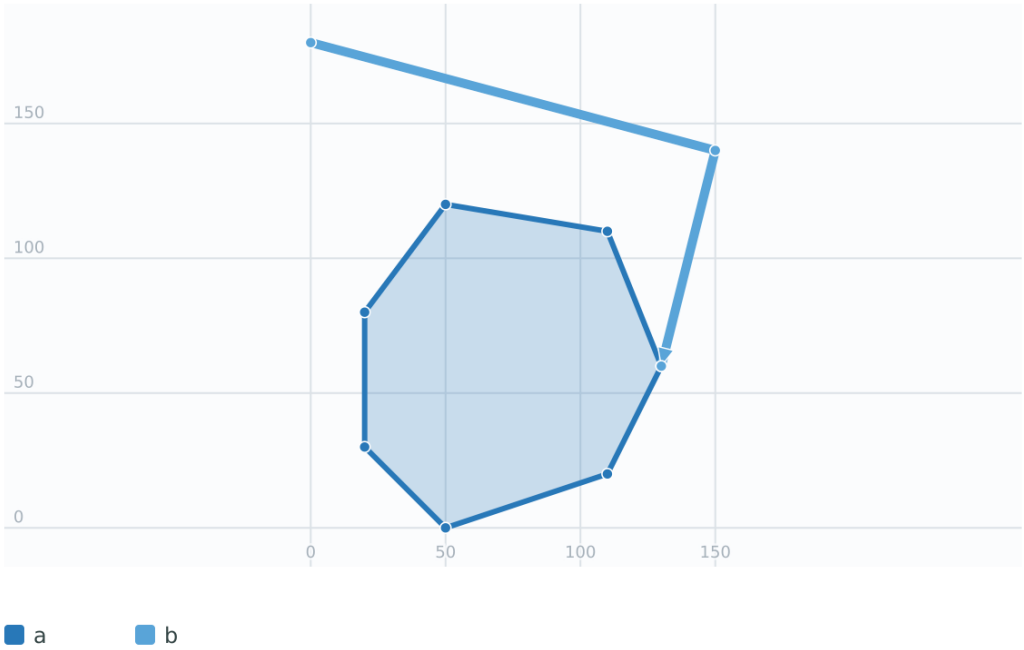
Code

```
SELECT ST_Touches(a, b)
FROM (SELECT
  'POLYGON ((80 10,50 40,50 90,80 130,140 120,160 70,140 30,80 10))'::geometry AS a,
  'LINESTRING (30 190,180 150,160 70)'::geometry AS b
) AS example;
```

Output

```
t
```

Figure



Two LineStrings touching at their endpoints.

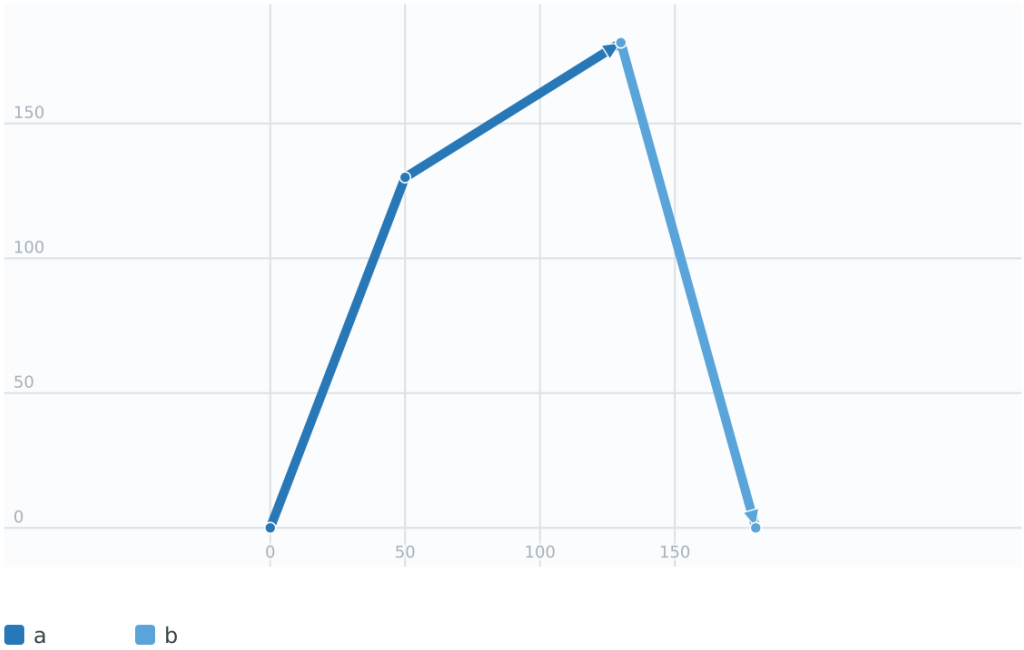
Code

```
SELECT ST_Touches(a, b)
FROM (SELECT
  'LINESTRING (10 10,60 140,140 190)'::geometry AS a,
  'LINESTRING (140 190,190 10)'::geometry AS b
) AS example;
```

Output

```
t
```

Figure



Two LineStrings touching where an endpoint meets the other line.

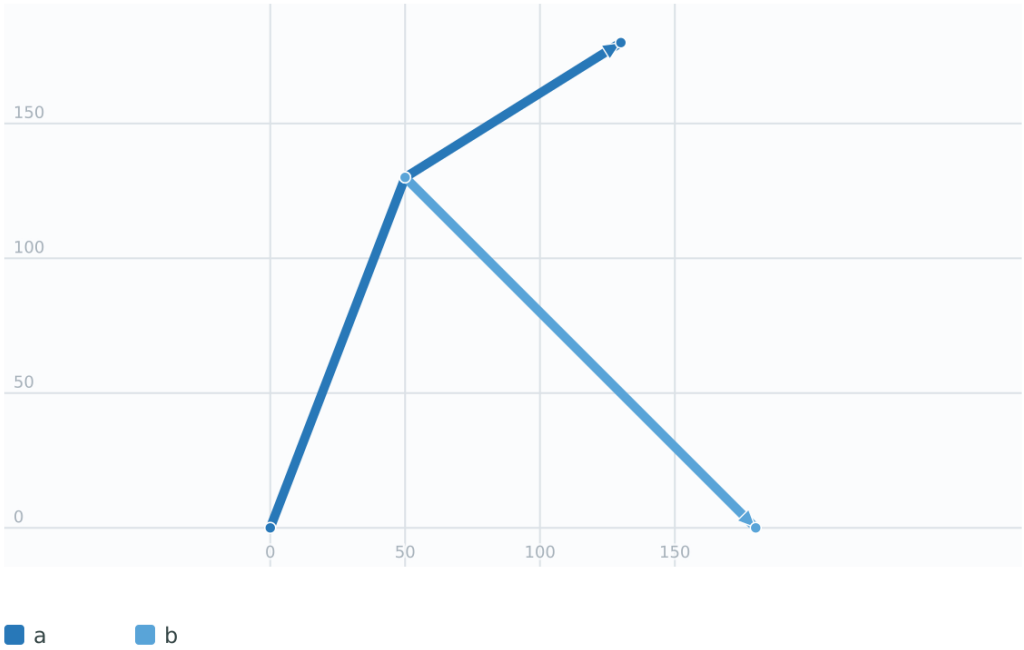
Code

```
SELECT ST_Touches(a, b)
FROM (SELECT
  'LINESTRING (10 10,60 140,140 190)'::geometry AS a,
  'LINESTRING (60 140,190 10)'::geometry AS b
) AS example;
```

Output

```
t
```

Figure



A Point touching a Polygon boundary.

Code

```
SELECT ST_Touches(a, b)
FROM (SELECT
  'POLYGON ((30 30,10 110,40 190,120 170,180 180,170 100,130 10,30 30))'::geometry AS a,
  'POINT (170 100)'::geometry AS b
) AS example;
```

Output

```
t
```

Figure



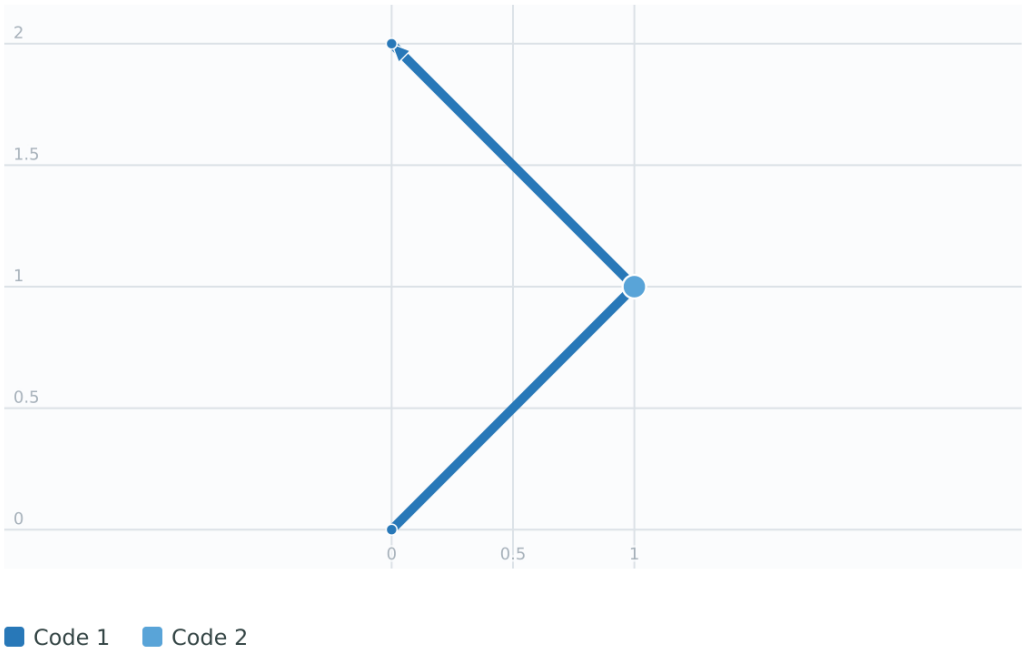
Code

```
SELECT ST_Touches('LINESTRING(0 0,1 1,0 2)'::geometry, 'POINT(1 1)'::geometry);
```

Output

```
f
```

Figure



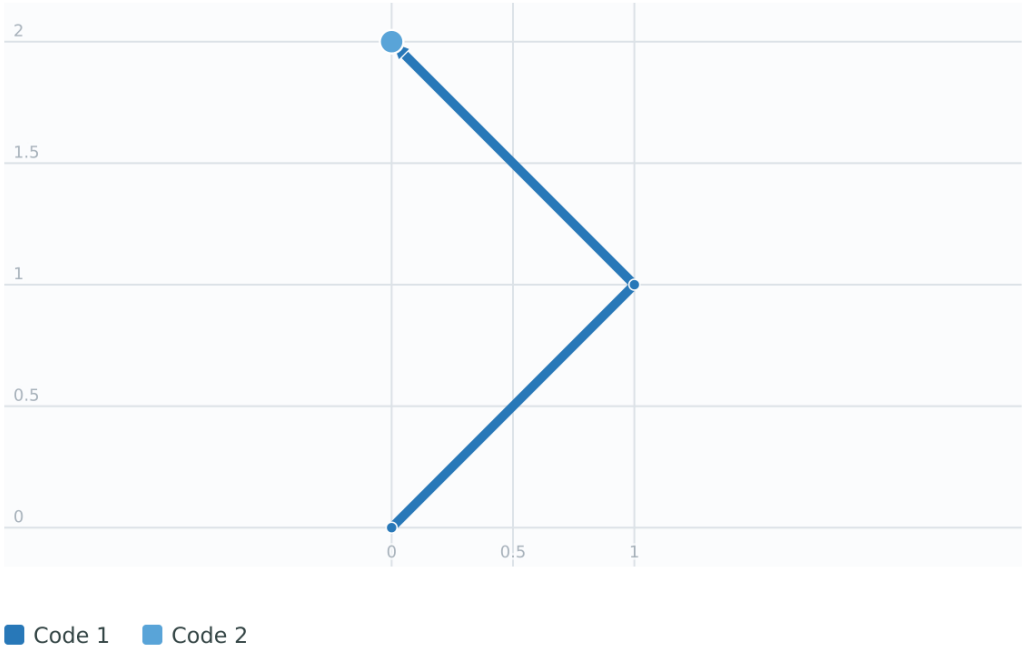
Code

```
SELECT ST_Touches('LINESTRING(0 0,1 1,0 2)'::geometry, 'POINT(0 2)'::geometry);
```

Output

```
t
```

Figure



7.11.1.16 ST_Within

ST_Within — Tests, wenn jeder Punkt von A in B liegt und ihre Innenräume einen gemeinsamen Punkt haben

Synopsis

boolean **ST_Within**(geometry A, geometry B);

Beschreibung

Gibt TRUE zurück, wenn Geometrie A innerhalb von Geometrie B liegt. A liegt nur dann innerhalb von B, wenn alle Punkte von A innerhalb (d. h. im Inneren oder am Rand) von B liegen (oder gleichwertig, keine Punkte von A liegen im Äußeren von B) und die Innenräume von A und B mindestens einen Punkt gemeinsam haben.

Damit diese Funktion sinnvoll ist, müssen die Ausgangsgeometrien beide die gleiche Koordinatenprojektion und den gleichen SRID haben.

Mathematisch ausgedrückt: $ST_Within(A, B) \Leftrightarrow (A \bigcap B = A) \wedge (Int(A) \bigcap Int(B) \neq \varnothing)$

Die Within-Relation ist reflexiv: jede Geometrie ist in sich selbst. Die Beziehung ist antisymmetrisch: Wenn `ST_Within(A,B) = true` und `ST_Within(B,A) = true`, dann müssen die beiden Geometrien topologisch gleich sein (`ST_Equals(A,B) = true`).

ST_Within ist die Umkehrung von ST_Contains. `ST_Within(A,B) = ST_Contains(B,A)`.



Anmerkung
Da die Innenräume einen gemeinsamen Punkt haben müssen, besteht eine Feinheit der Definition darin, dass Linien und Punkte, die vollständig in der Begrenzung von Polygonen oder Linien liegen, *nicht* innerhalb der Geometrie sind. Für weitere Details siehe Subtleties of OGC Covers, Contains, Within. Das Prädikat ST_CoveredBy bietet eine umfassendere Beziehung.



Anmerkung
Diese Funktion beinhaltet automatisch einen Bounding-Box-Vergleich, der alle räumlichen Indizes verwendet, die für die Geometrien verfügbar sind. Um die Verwendung von Indizes zu vermeiden, verwenden Sie die Funktion `_ST_Within`.

Wird durch das GEOS Modul ausgeführt

Verbessert: 2.3.0 Verbesserung des PIP-Kurzschlusses für Geometrien, erweitert um die Unterstützung von MultiPoints mit wenigen Punkten. Frühere Versionen unterstützten nur Punkte in Polygonen.

Verbessert: 3.0.0 ermöglicht die Unterstützung von `GEOMETRYCOLLECTION`



Wichtig
Verwenden Sie diese Funktion nicht mit ungültigen Geometrien. Sie werden unerwartete Ergebnisse erhalten.

HINWEIS: Dies ist die "zulässige" Version, die einen booleschen Wert und keine ganze Zahl zurückgibt.

 Diese Methode implementiert die OGC Simple Features Implementation Specification for SQL 1.1. s2.1.1.2 // s2.1.13.3 - a.Relate(b, "T*F**F***")

 Diese Methode setzt die SQL/MM-Spezifikation um. SQL-MM 3: 5.1.30

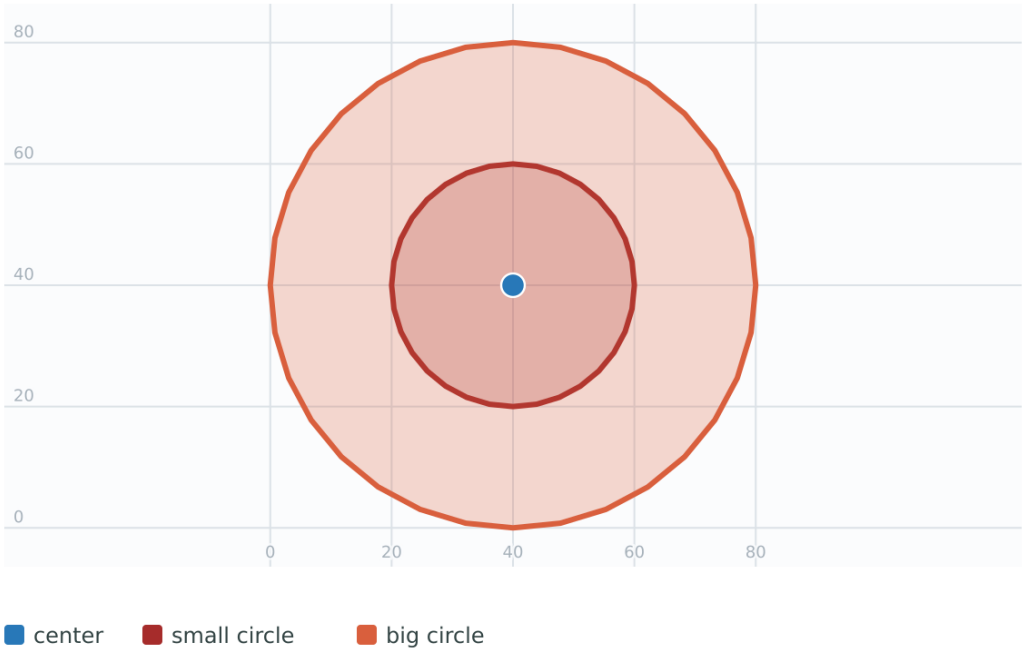
Beispiele

This example defines two concentric circles once in a CTE so the same inputs can be rendered and then reused in related predicate checks.

Code

```
WITH circles AS (
  SELECT ST_Buffer(center, 20) AS smallc,
         ST_Buffer(center, 40) AS bigc
  FROM (SELECT 'POINT(50 50)'::geometry AS center) AS p
)
SELECT smallc AS small_circle,
       bigc AS big_circle
FROM circles;
```

Figure



Using the same circles, `ST_Within` is reflexive on each circle, false from the larger circle into the smaller one, and true for `ST_Union` only when the union does not extend beyond the larger circle.

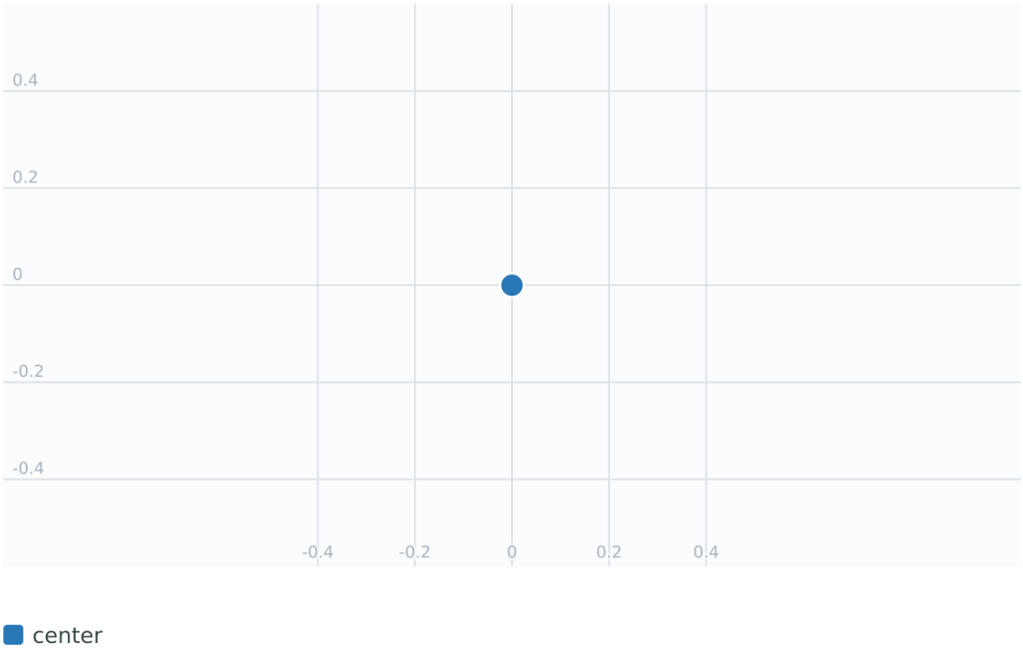
Code

```
WITH circles AS (
  SELECT ST_Buffer(center, 20) AS smallc,
         ST_Buffer(center, 40) AS bigc
  FROM (SELECT 'POINT(50 50)'::geometry AS center) AS p
)
SELECT
  ST_Within(smallc, smallc) AS smallinsmall,
  ST_Within(smallc, bigc) AS smallinbig,
  ST_Within(bigc, smallc) AS biginsmall,
  ST_Within(ST_Union(smallc, bigc), bigc) AS unioninbig,
  ST_Within(bigc, ST_Union(smallc, bigc)) AS biginunion,
  ST_Equals(bigc, ST_Union(smallc, bigc)) AS bigisunion
FROM circles;
```

Output

```
t | t | f | t | t | t
```

Figure



Siehe auch

ST_Contains, ST_CoveredBy, ST_Equals, ST_IsValid

7.11.2 Fernbeziehungen

7.11.2.1 ST_3DDWithin

ST_3DDWithin — Prüft, ob zwei 3D-Geometrien innerhalb eines bestimmten 3D-Abstands liegen

Synopsis

boolean **ST_3DDWithin**(geometry g1, geometry g2, double precision distance_of_srid);

Beschreibung

Gibt true zurück, wenn der 3D-Abstand zwischen zwei Geometriewerten nicht größer ist als distance `distance_of_srid`. Der Abstand wird in Einheiten angegeben, die durch das räumliche Bezugssystem der Geometrien definiert sind. Damit diese Funktion sinnvoll ist, müssen sich die Ausgangsgeometrien im selben Koordinatensystem befinden (denselben SRID haben).



Anmerkung
Diese Funktion beinhaltet automatisch einen Bounding-Box-Vergleich, der alle räumlichen Indizes verwendet, die für die Geometrien verfügbar sind.

 Diese Funktion unterstützt 3d und lässt den Z-Index nicht fallen.

 Diese Funktion unterstützt polyedrische Flächen.

 Diese Methode setzt die SQL/MM-Spezifikation um. SQL-MM ?

Verfügbarkeit: 2.0.0

Beispiele

Compare the same transformed point and line in both 3D and 2D. The 3D test fails because the Z difference matters even though the 2D footprints are close enough.

Code

```
WITH sample AS (
  SELECT ST_Transform('SRID=4326;POINT(-72.1235 42.3521 4)'::geometry, 2163) AS input_point,
         ST_Transform(
           'SRID=4326;LINESTRING(-72.1260 42.45 15,-72.123 42.1546 20)'::geometry,
           2163
         ) AS input_line
)
SELECT ST_3DDWithin(input_point, input_line, 126.8) AS within_dist_3d,
       ST_DWithin(input_point, input_line, 126.8) AS within_dist_2d,
       input_point AS input_point,
       input_line AS input_line
FROM sample;
```

Output

```
-[ RECORD 1 ↩
    ]--+-----------------------------------------------------------------------------------------------------------------------------------------------------------

within_dist_3d | f
within_dist_2d | t
input_point    | POINT Z (-11774128.9 5237160.2 4)
input_line     | LINESTRING Z (-11774339.5 5243556.8 15,-11773706.7 5215256.3 20)
```

Siehe auch

ST_3DDFullyWithin, ST_DWithin, ST_DFullyWithin, ST_3DDistance, ST_Distance, ST_3DMaxDistance, ST_Transform

7.11.2.2 ST_3DDFullyWithin

ST_3DDFullyWithin — Prüft, ob zwei 3D-Geometrien vollständig innerhalb eines bestimmten 3D-Abstands liegen

Synopsis

boolean **ST_3DDFullyWithin**(geometry g1, geometry g2, double precision distance);

Beschreibung

Gibt true zurück, wenn die 3D-Geometrien vollständig innerhalb des angegebenen Abstands zueinander liegen. Der Abstand wird in Einheiten angegeben, die durch das räumliche Bezugssystem der Geometrien definiert sind. Damit diese Funktion sinnvoll ist, müssen die Ausgangsgeometrien beide der gleichen Koordinatenprojektion angehören und den gleichen SRID haben.



Anmerkung
Diese Funktion beinhaltet automatisch einen Bounding-Box-Vergleich, der alle räumlichen Indizes verwendet, die für die Geometrien verfügbar sind.

Verfügbarkeit: 2.0.0

 Diese Funktion unterstützt 3d und lässt den Z-Index nicht fallen.

 Diese Funktion unterstützt polyedrische Flächen.

Beispiele

This example compares distance-within predicates in 2D and 3D. The point is distance-within the line in 3D, but the full 3D line is not within the same distance of the point.

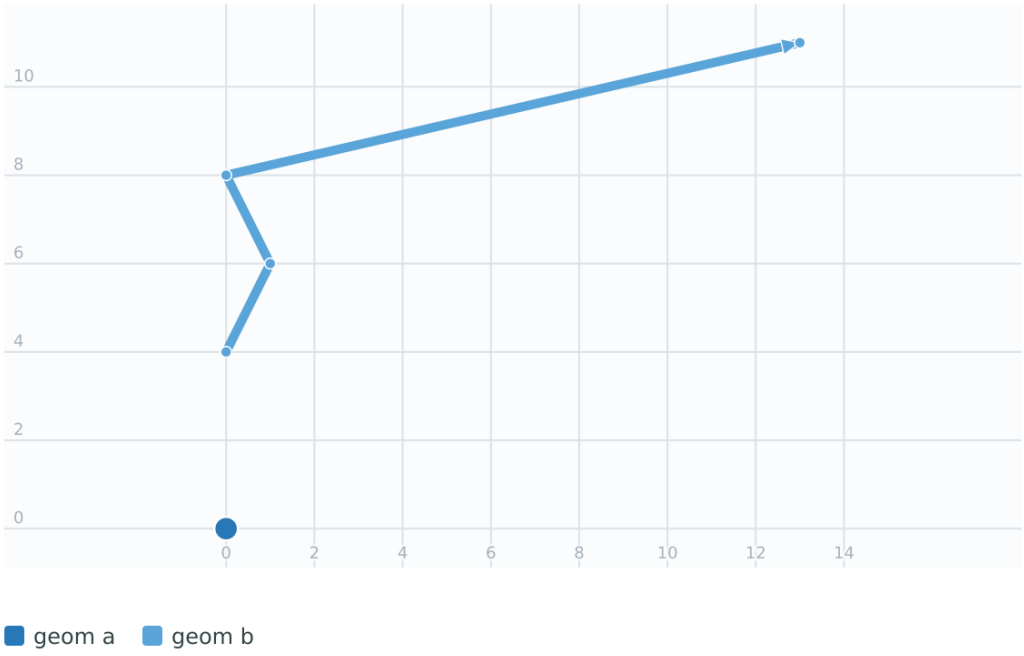
Code

```
WITH data AS (
  SELECT
    'POINT(1 1 2)'::geometry AS geom_a,
    'LINESTRING(1 5 2,2 7 20,1 9 100,14 12 3)'::geometry AS geom_b
)
SELECT
  ST_3DDFullyWithin(geom_a, geom_b, 10) AS d3dfullywithin10,
  ST_3DDWithin(geom_a, geom_b, 10) AS d3dwithin10,
  ST_DFullyWithin(geom_a, geom_b, 20) AS d2dfullywithin20,
  ST_3DDFullyWithin(geom_a, geom_b, 20) AS d3dfullywithin20
FROM data;
```

Output

```
d3dfullywithin10 | d3dwithin10 | d2dfullywithin20 | d3dfullywithin20
-----------------+-------------+------------------+-----------------
 f               | t           | t                | f
```

Figure



Siehe auch

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);

Beschreibung

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.



Anmerkung
Diese Funktion beinhaltet automatisch einen Bounding-Box-Vergleich, der alle räumlichen Indizes verwendet, die für die Geometrien verfügbar sind.

Verfügbarkeit: 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.

Beispiele

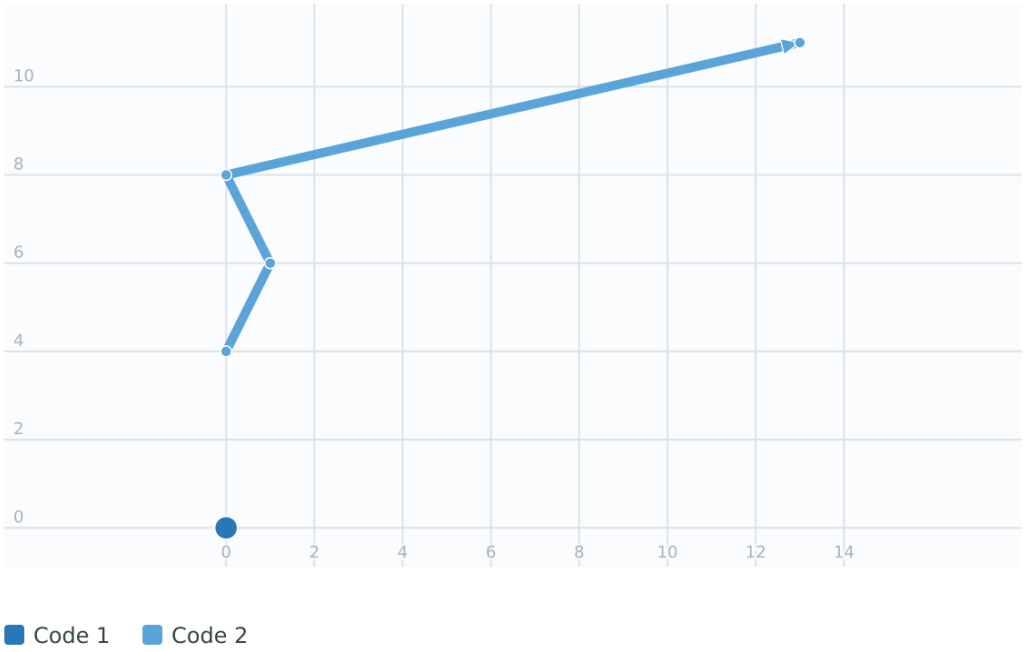
Code

```
SELECT
    ST_DFullyWithin(geom_a, geom_b, 10) AS DFullyWithin10,
    ST_DWithin(geom_a, geom_b, 10) AS DWithin10,
    ST_DFullyWithin(geom_a, geom_b, 20) AS DFullyWithin20
FROM (VALUES
    ('POINT(1 1)'::geometry, 'LINESTRING(1 5,2 7,1 9,14 12)'::geometry)
    ) AS v(geom_a, geom_b)
```

Output

```
dfullywithin10 | dwithin10 | dfullywithin20
---------------+-----------+---------------
 f             | t         | t
```

Figure



Siehe auch

ST_MaxDistance, ST_DWithin, ST_3DDWithin, ST_3DDFullyWithin

7.11.2.4 ST_DWithin

ST_DWithin — Prüft, ob zwei Geometrien innerhalb eines bestimmten Abstands liegen

Synopsis

boolean **ST_DWithin**(geometry g1, geometry g2, double precision distance_of_srid);
boolean **ST_DWithin**(geography gg1, geography gg2, double precision distance_meters, boolean use_spheroid = true);

Beschreibung

Gibt true zurück, wenn die Geometrien innerhalb eines bestimmten Abstands liegen

Für geometry: Der Abstand wird in Einheiten angegeben, die durch das räumliche Bezugssystem der Geometrien definiert sind. Damit diese Funktion sinnvoll ist, müssen sich die Ausgangsgeometrien im selben Koordinatensystem befinden (denselben SRID haben).

Für Geografie: Die Einheiten sind in Metern und die Entfernungsmessung ist standardmäßig auf `use_spheroid = true` eingestellt. Für eine schnellere Auswertung verwenden Sie `use_spheroid = false`, um auf der Kugel zu messen.



Anmerkung
Verwenden Sie ST_3DDWithin für 3D-Geometrien.



Anmerkung
Dieser Funktionsaufruf beinhaltet einen Bounding-Box-Vergleich, der alle für die Geometrien verfügbaren Indizes verwendet.

 Diese Methode implementiert die OGC Simple Features Implementation Specification for SQL 1.1.

Verfügbarkeit: Mit Version 1.5.0 wurde die Unterstützung für Geografie eingeführt.

Erweitert: 2.1.0 verbesserte die Geschwindigkeit für Geographie. Weitere Details finden Sie unter Making Geography faster.

Verbessert: 2.1.0 Unterstützung für gekrümmte Geometrien wurde eingeführt.

Vor 1.3 wurde ST_Expand üblicherweise in Verbindung mit && und ST_Distance verwendet, um den Abstand zu testen, und vor 1.3.4 verwendete diese Funktion diese Logik. Ab 1.3.4 verwendet ST_DWithin eine schnellere Kurzschluss-Abstandsfunktion.

Beispiele

This self-contained geometry example keeps the compared shapes visible while also returning the boolean results.

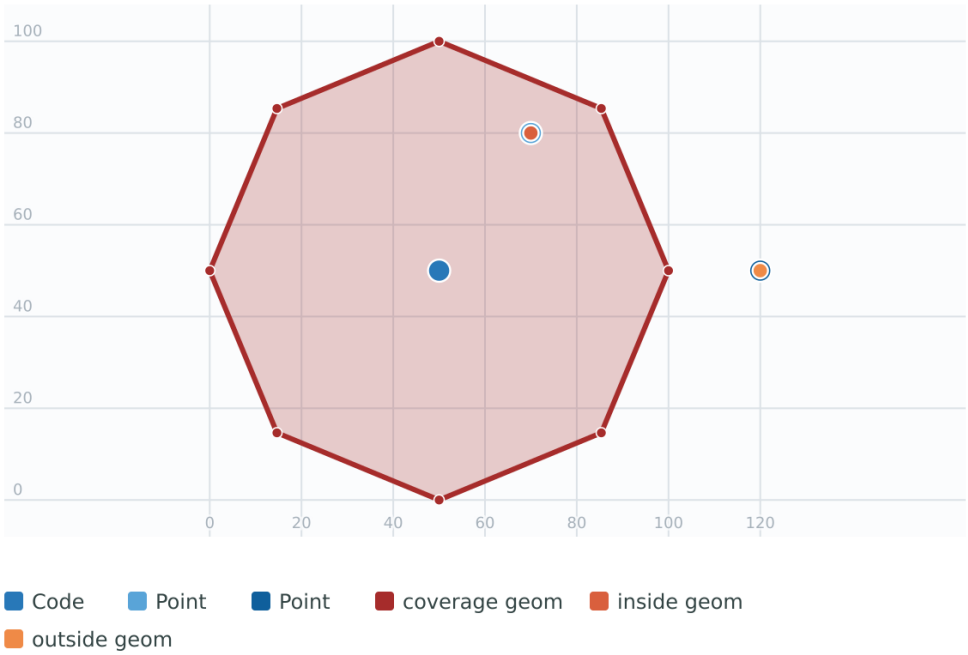
Code

```
WITH sample AS (
  SELECT ST_Buffer('POINT(100 90)'::geometry, 50, 'quad_segs=2') AS coverage,
         ST_Point(120, 120) AS inside_pt,
         ST_Point(170, 90) AS outside_pt
)
SELECT ST_DWithin(coverage, inside_pt, 0) AS inside_now,
       ST_DWithin(coverage, outside_pt, 10) AS outside_within_10,
       coverage AS coverage_geom,
       inside_pt AS inside_geom,
       outside_pt AS outside_geom
FROM sample;
```

Output

```
-[ RECORD 1 ]-----
inside_now        | t
outside_within_10 | f
coverage_geom     | POLYGON((150 90,135 55,100 40,65 55,50 90,65 125,100 140,135 125,150 90))
inside_geom       | POINT(120 120)
outside_geom      | POINT(170 90)
```

Figure



Find the nearest hospital to each school. That is within 3000 units of the school. We do an ST_DWithin search to utilize indexes to limit our search list. That the non-indexable ST_Distance needs to process. If the units of the spatial reference is meters then units would be meters.

Code

```
SELECT DISTINCT ON (s.gid) s.gid, s.school_name, s.geom, h.hospital_name
  FROM schools s
    LEFT JOIN hospitals h ON ST_DWithin(s.geom, h.geom, 3000)
  ORDER BY s.gid, ST_Distance(s.geom, h.geom);
```

The schools with no close hospitals. Find all schools with no hospital within 3000 units away from the school. Units are in units of spatial reference (for example meters, feet, or degrees).

Code

```
SELECT s.gid, s.school_name
  FROM schools s
    LEFT JOIN hospitals h ON ST_DWithin(s.geom, h.geom, 3000)
  WHERE h.gid IS NULL;
```

Find locations more than 3000 units away from their nearest hospital. Use `NOT EXISTS` so that a location is returned only when no nearby hospital exists.

Code

```
SELECT l.location_name
  FROM locations l
  WHERE NOT EXISTS (
    SELECT 1
    FROM hospitals h
    WHERE ST_DWithin(l.geom, h.geom, 3000)
  );
```

Find broadcasting towers that a receiver with limited range can receive. Data is geometry in Spherical Mercator (SRID=3857), ranges are approximate.

Code

```
CREATE INDEX ON broadcasting_towers USING gist (geom);
CREATE INDEX ON broadcasting_towers USING gist (ST_Expand(geom, sending_range));
```

Query towers that a 4-kilometer receiver in Minsk Hackerspace can get. Note: two conditions are required because the shorter `LEAST(b.sending_range, 4000)` will not use an index.

Code

```
SELECT b.tower_id, b.geom
  FROM broadcasting_towers b
  WHERE ST_DWithin(b.geom, 'SRID=3857;POINT(3072163.4 7159374.1)', 4000)
    AND ST_DWithin(b.geom, 'SRID=3857;POINT(3072163.4 7159374.1)', b.sending_range);
```

Siehe auch

ST_Distance, ST_3DDWithin

7.11.2.5 ST_PointInsideCircle

ST_PointInsideCircle — Prüft, ob ein geometrischer Punkt innerhalb eines Kreises liegt, der durch einen Mittelpunkt und einen Radius definiert ist

Synopsis

boolean **ST_PointInsideCircle**(geometry a_point, float center_x, float center_y, float radius);

Beschreibung

Gibt true zurück, wenn die Geometrie ein Punkt ist und innerhalb des Kreises mit Zentrum `center_x`,`center_y` und Radius `radius` liegt.



Warnung
Verwendet keine räumlichen Indizes. Verwenden Sie stattdessen ST_DWithin.

Verfügbarkeit: 1.2

Geändert: 2.2.0 In früheren Versionen hieß dies ST_Point_Inside_Circle

Beispiele

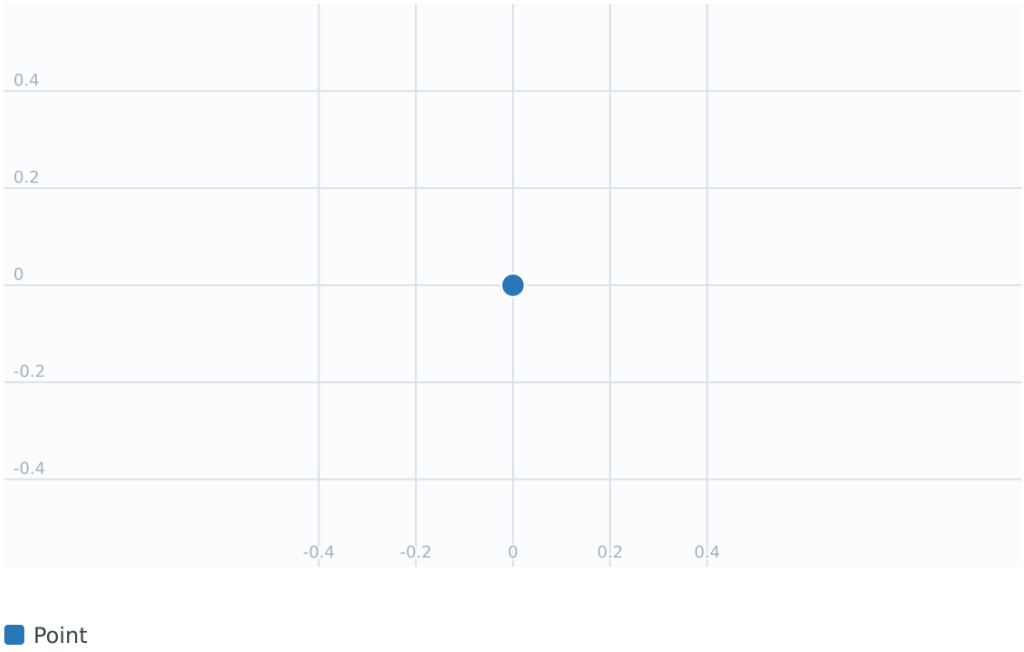
Code

```
SELECT ST_PointInsideCircle(ST_Point(1, 2), 0.5, 2, 3);
```

Output

```
t
```

Figure



Siehe auch

ST_DWithin

7.12 Messfunktionen

7.12.1 ST_Area

ST_Area — Gibt den geometrischen Schwerpunkt einer Geometrie zurück.

Synopsis

float **ST_Area**(geometry g1);
float **ST_Area**(geography geog, boolean use_spheroid = true);

Beschreibung

Gibt den Flächeninhalt von Polygonen und Mehrfachpolygonen zurück. Gibt den Flächeninhalt der Datentypen "ST_Surface" und "ST_MultiSurface" zurück. Beim geometrischen Datentyp wird die kartesische 2D-Fläche ermittelt und in den Einheiten des SRID ausgegeben. Beim geographischen Datentyp wird die Fläche standardmäßig auf einem Referenzellipsoid ermittelt und in Quadratmeter ausgegeben. Mit ST_Area(geog,false) kann der Flächeninhalt auf einer Kugel ermittelt werden; dies ist zwar schneller aber auch weniger genau.

Erweiterung: Mit 2.0.0 wurde 2D-Unterstützung für polyedrische Oberflächen eingeführt.

Erweiterung: 2.2.0 - die Messung auf dem Referenzellipsoid wird mit der Bibliothek "GeographicLib" durchgeführt. Dadurch wurde die Genauigkeit und die Robustheit erhöht. Um die Vorteile dieser neuen Funktionalität zu nutzen, benötigen Sie Proj >= 4.9.0.

Geändert: 3.0.0 - hängt nicht mehr von SFCGAL ab.

Diese Methode implementiert die OGC Simple Features Implementation Specification for SQL 1.1.

Diese Methode setzt die SQL/MM-Spezifikation um. SQL-MM 3: 8.1.2, 9.5.3

Diese Funktion unterstützt polyedrische Flächen.

Anmerkung
Bei polyedrischen Oberflächen wird nur 2D (nicht 2.5D) unterstützt. Bei 2.5D kann ein Ergebnis ungleich null geliefert werden, wenn die Oberflächen vollständig in der XY-Ebene liegen.

Beispiele

Gibt den Flächeninhalt eines Grundstücks in Massachusetts - in Quadratfuß und konvertiert in Quadratmeter - zurück. Anmerkung: Wegen "Massachusetts State Plane Feet" (EPSG:2249) wird der Flächeninhalt in Quadratfuß ausgegeben

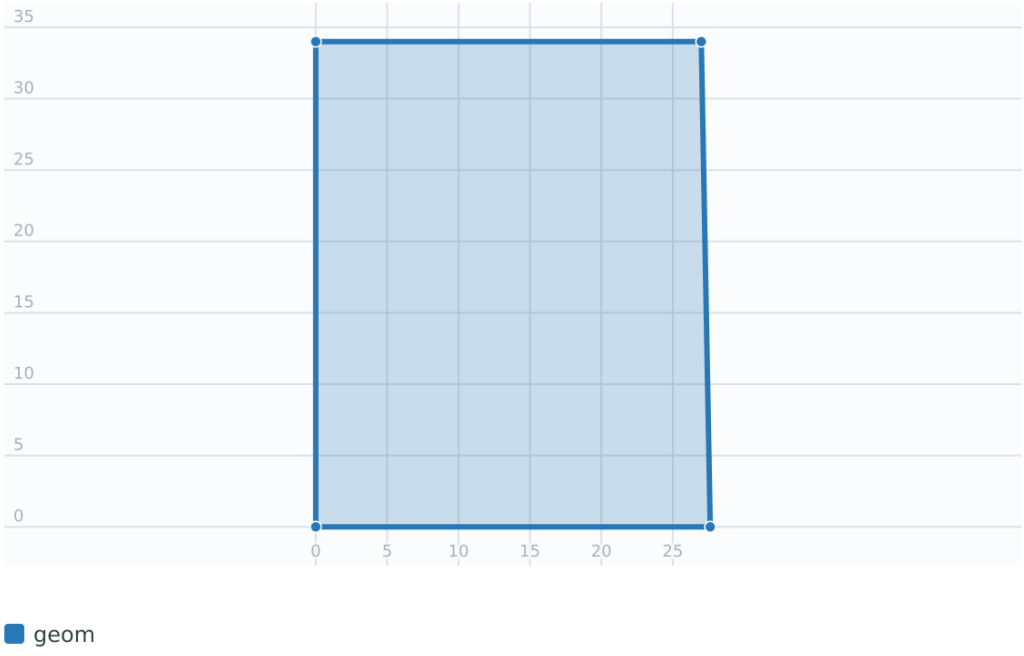
Code

```
SELECT ST_Area(geom) AS sqft,
       ST_Area(geom) * 0.3048 ^ 2 AS sqm
FROM (
  SELECT 'SRID=2249;POLYGON((743238 2967416,743238 2967450,
          743265 2967450,743265.625 2967416,743238 2967416))'::geometry AS geom
) AS subquery;
```

Output

```
sqft    |     sqm
--------+-------------
928.625 | 86.27208552
```

Figure



Return area square feet and transform to Massachusetts state plane meters (EPSG:26986) to get square meters. Note this is in square feet because 2249 is Massachusetts State Plane Feet and transformed area is in square meters since EPSG:26986 is state plane Massachusetts meters.

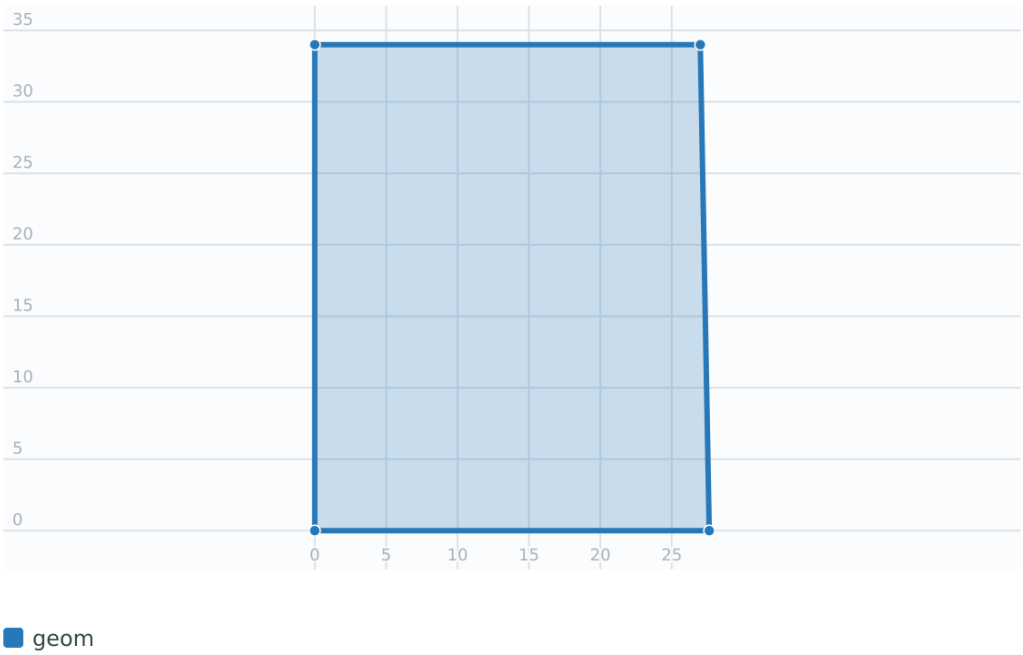
Code

```
SELECT ST_Area(geom) AS sqft,
       round(ST_Area(ST_Transform(geom, 26986))::numeric, 6) AS sqm
FROM (
  SELECT 'SRID=2249;POLYGON((743238 2967416,743238 2967450,
          743265 2967450,743265.625 2967416,743238 2967416))'::geometry AS geom
) AS subquery;
```

Output

```
sqft   |    sqm
--------+-----------
928.625 | 86.272431
```

Figure



Gibt den Flächeninhalt in Quadratfuß und in Quadratmeter für den geographischen Datentyp zurück. Beachten Sie bitte, dass wir den geometrischen in den geographischen Datentyp umwandeln (dafür muss die Geometrie in WGS84 lon lat 4326 vorliegen). Beim geographischen Datentyp wird immer in Meter gemessen. Dies ist nur für Vergleichszwecke gedacht, da Ihre Tabelle üblicherweise bereits den geographischen Datentyp aufweisen wird.

Code

```
SELECT round((ST_Area(geog) / 0.3048 ^ 2)::numeric, 2) AS sqft_spheroid,
       round((ST_Area(geog, false) / 0.3048 ^ 2)::numeric, 2) AS sqft_sphere,
       round(ST_Area(geog)::numeric, 2) AS sqm_spheroid
FROM (
  SELECT ST_Transform(
           'SRID=2249;POLYGON((743238 2967416,743238 2967450,
           743265 2967450,743265.625 2967416,743238 2967416))'::geometry,
           4326
         )::geography AS geog
) AS subquery;
```

Output

```
sqft_spheroid | sqft_sphere | sqm_spheroid
---------------+-------------+--------------
         928.68 |      926.61 |        86.28
```

Wenn Ihre Daten bereits in der Geografie enthalten sind:

Code

```
SELECT ST_Area(geog) / 0.3048 ^ 2 AS sqft,
               ST_Area(the_geog) AS sqm
        FROM somegeogtable;
```

Siehe auch

CG_3DArea, ST_GeomFromEWKT, ST_LengthSpheroid, ST_Perimeter, ST_Transform

7.12.2 ST_Azimuth

ST_Azimuth — Gibt die 2-dimenionale kürzeste Strecke zwischen zwei Geometrien als Linie zurück

Synopsis

float **ST_Azimuth**(geometry origin, geometry target);
float **ST_Azimuth**(geography origin, geography target);

Beschreibung

Gibt den Azimutwinkel des Zielpunkts vom Ursprungspunkt in Bogenmaß zurück oder NULL, wenn die beiden Punkte übereinstimmen. Der Azimutwinkel ist ein positiver Winkel im Uhrzeigersinn, der sich auf die positive Y-Achse (Geometrie) oder den Nordmeridian (Geografie) bezieht: Nord = 0; Nordost = $\pi/4$; Ost = $\pi/2$; Südost = $3\pi/4$; Süd = π; Südwest $5\pi/4$; West = $3\pi/2$; Nordwest = $7\pi/4$.

Für den Typ Geografie ist die Azimutlösung als inverses geodätisches Problem bekannt.

Der Azimut ist ein mathematisches Konzept, das als der Winkel zwischen einem Referenzvektor und einem Punkt definiert ist, mit Winkeleinheiten im Bogenmaß. Der Ergebniswert in Bogenmaß kann mit der PostgreSQL-Funktion `degrees()` in Grad umgerechnet werden.

For offsetting lines relative to their direction, use ST_OffsetCurve. For projecting a point by distance and bearing, use ST_Project.

Verfügbarkeit: 1.1.0

Erweiterung: mit 2.0.0 wurde die Unterstützung des geographischen Datentyps eingeführt.

Erweiterung: 2.2.0 die Messungen auf dem Referenzellipsoid werden mit der Bibliothek "GeographicLib" durchgeführt. Dadurch wurde die Genauigkeit und die Robustheit erhöht. Um die Vorteile dieser neuen Funktionalität zu nutzen, benötigen Sie Proj >= 4.9.0.

Beispiele

Geometrischer Datentyp - Azimut in Grad

Code

```
SELECT degrees(ST_Azimuth(ST_Point(25, 45), ST_Point(75, 100))) AS degA_B,
       degrees(ST_Azimuth(ST_Point(75, 100), ST_Point(25, 45) )) AS degB_A;
```

Output

```
dega_b       |     degb_a
-----------------+-----------------
42.27368900609371 | 222.27368900609372
```

Compass direction labels can be computed from the azimuth.

Code

```
WITH directions(deg) AS (VALUES
  (0), (45), (90), (135), (180), (225), (270), (315)
),
bearings AS (
  SELECT deg,
         degrees(ST_Azimuth(ST_Point(0, 0),
                            ST_MakePoint(sin(radians(deg)),
                                         cos(radians(deg))))) AS azimuth
  FROM directions
)
SELECT deg,
       round(azimuth::numeric, 1) AS azimuth,
       (ARRAY['N', 'NE', 'E', 'SE', 'S', 'SW', 'W', 'NW', 'N'])
         [floor((azimuth + 22.5) / 45)::int + 1] AS direction
FROM bearings
ORDER BY deg;
```

Output

```
deg | azimuth | direction
-----+---------+-----------
   0 |     0.0 | N
  45 |    45.0 | NE
  90 |    90.0 | E
 135 |   135.0 | SE
 180 |   180.0 | S
 225 |   225.0 | SW
 270 |   270.0 | W
 315 |   315.0 | NW
```

Azimuth from Point(25, 45) to Point(75, 100), shown against the positive Y axis (north).

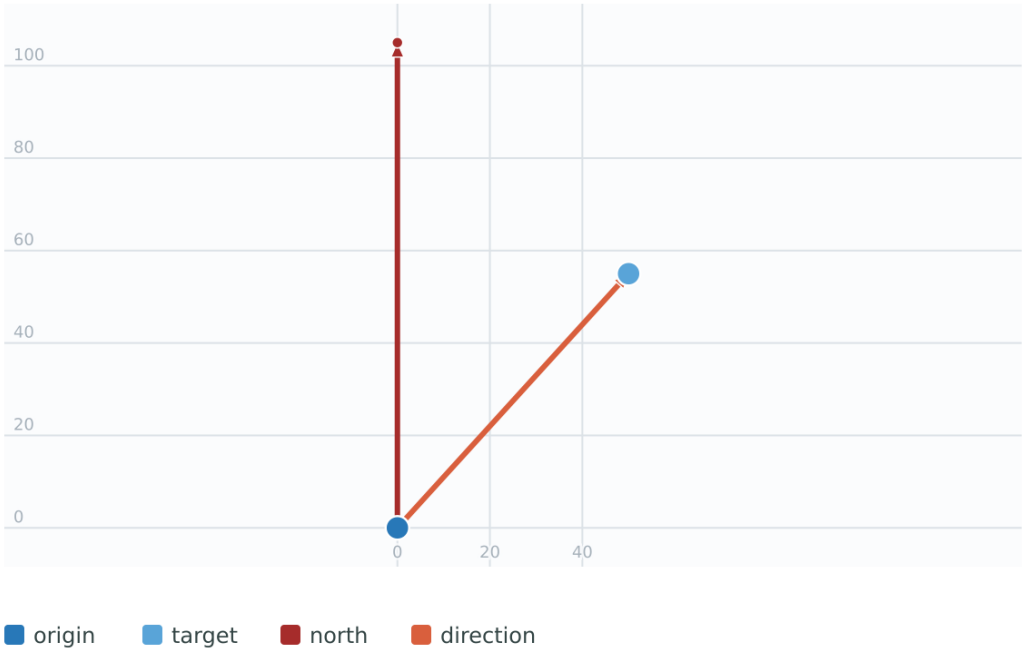
Code

```
WITH points AS (
  SELECT 'POINT(25 45)'::geometry AS origin,
         'POINT(75 100)'::geometry AS target
)
SELECT round(degrees(ST_Azimuth(origin, target))::numeric, 6) AS degrees,
       ST_MakeLine(origin, ST_Translate(origin, 0, 105)) AS north,
       ST_MakeLine(origin, target) AS direction
FROM points;
```

Output

```
42.273689 | LINESTRING(25 45,25 150) | LINESTRING(25 45,75 100)
```

Figure



Reverse azimuth from Point(75, 100) to Point(25, 45).

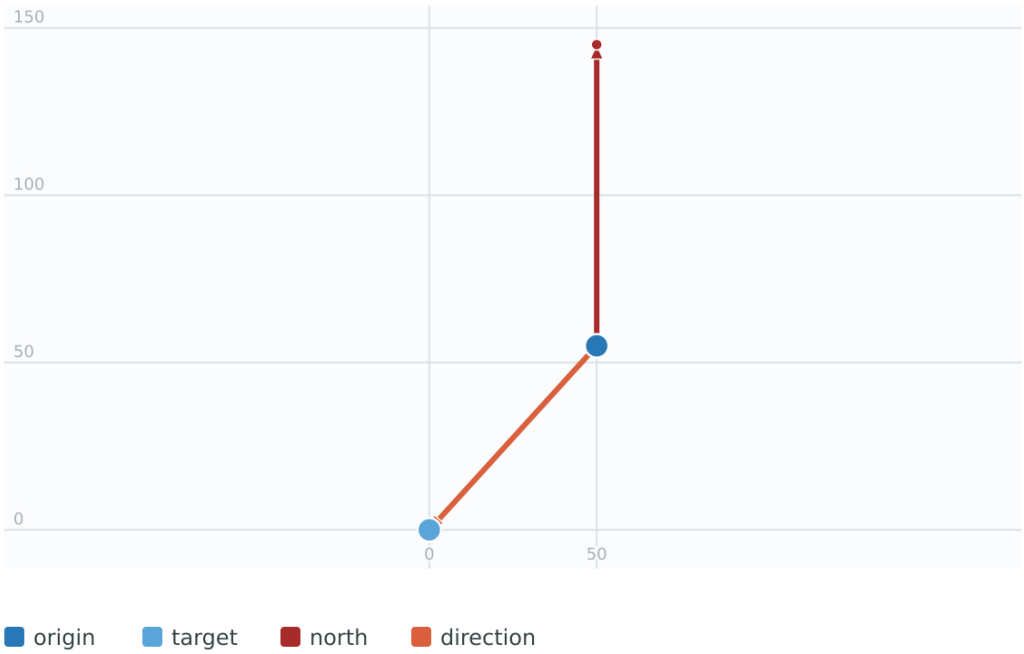
Code

```
WITH points AS (
  SELECT 'POINT(75 100)'::geometry AS origin,
         'POINT(25 45)'::geometry AS target
)
SELECT round(degrees(ST_Azimuth(origin, target))::numeric, 6) AS degrees,
       ST_MakeLine(origin, ST_Translate(origin, 0, 90)) AS north,
       ST_MakeLine(origin, target) AS direction
FROM points;
```

Output

```
222.273689 | LINESTRING(75 100,75 190) | LINESTRING(75 100,25 45)
```

Figure



Siehe auch

ST_Angle, ST_Translate, ST_Project, PostgreSQL Math Functions

7.12.3 ST_Angle

ST_Angle — Gibt den Winkel zwischen 3 Punkten oder zwischen 2 Vektoren (4 Punkte oder 2 Linien) zurück.

Synopsis

float **ST_Angle**(geometry point1, geometry point2, geometry point3, geometry point4);
float **ST_Angle**(geometry line1, geometry line2);

Beschreibung

Gibt die größte 3-dimensionale Distanz zwischen zwei geometrischen Objekten als Linie zurück

Variante 1: berechnet den Winkel, der von den Punkten P1-P2-P3 eingeschlossen wird. Wenn ein 4. Punkt vorgesehen ist, werden die Winkel P1-P2 und P3-P4 berechnet

Variante 2: berechnet den Winkel zwischen zwei Vektoren S1-E1 und S2-E2, die durch die Anfangs- und Endpunkte der Eingabelinien definiert sind

Das Ergebnis wird in Radiant ausgegeben. Wie im folgenden Beispiel gezeigt, kann mit der in PostgreSQL integrierten Funktion "degrees()" von der Einheit Radiant auf die Einheit Grad umgerechnet werden.

ST_Angle(P1,P2,P3) = ST_Angle(P2,P1,P2,P3)

Verfügbarkeit: 2.5.0

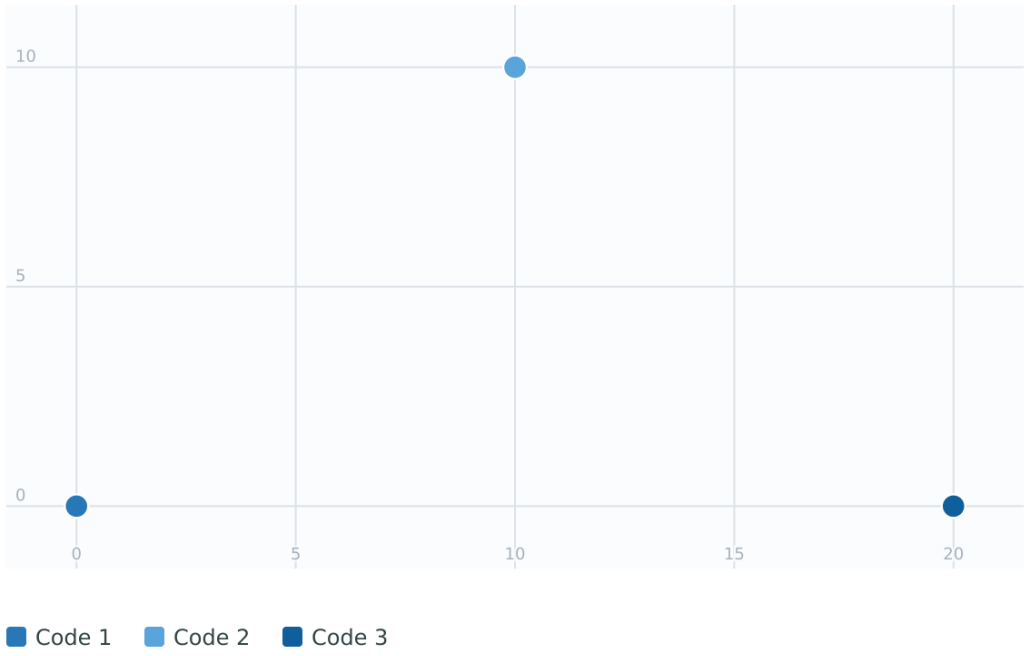
Beispiele

Längste Strecke zwischen Polygon und Polygon

Code

```
SELECT degrees(ST_Angle('POINT(0 0)', 'POINT(10 10)', 'POINT(20 0)'));
```

Figure

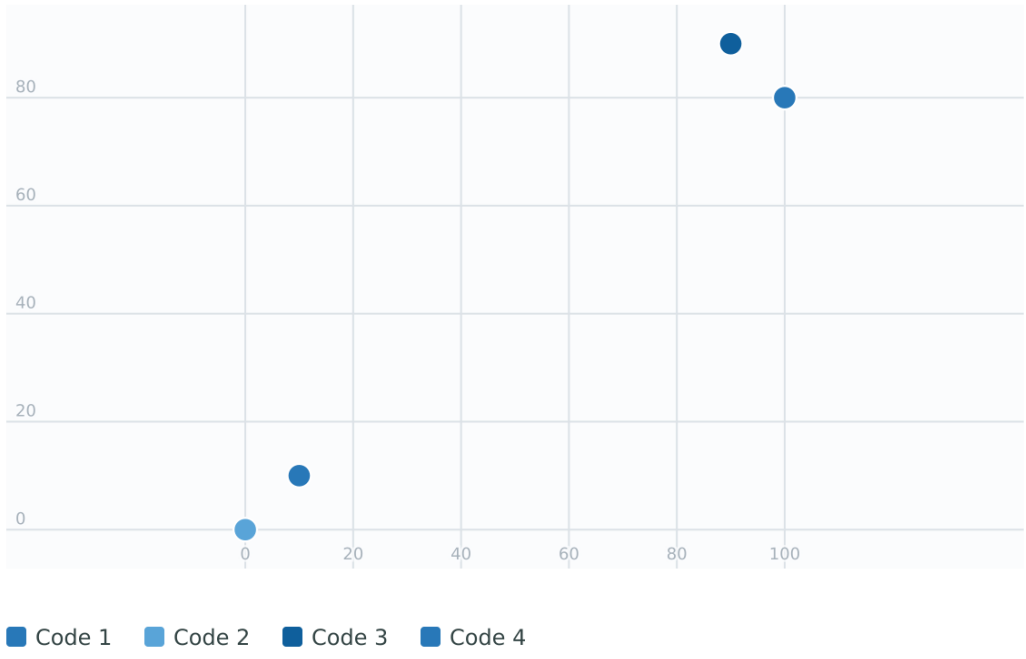


Winkel zwischen Vektoren, die durch vier Punkte definiert sind

Code

```
SELECT degrees(ST_Angle('POINT (10 10)', 'POINT (0 0)', 'POINT(90 90)', 'POINT (100 80)'));
```

Figure

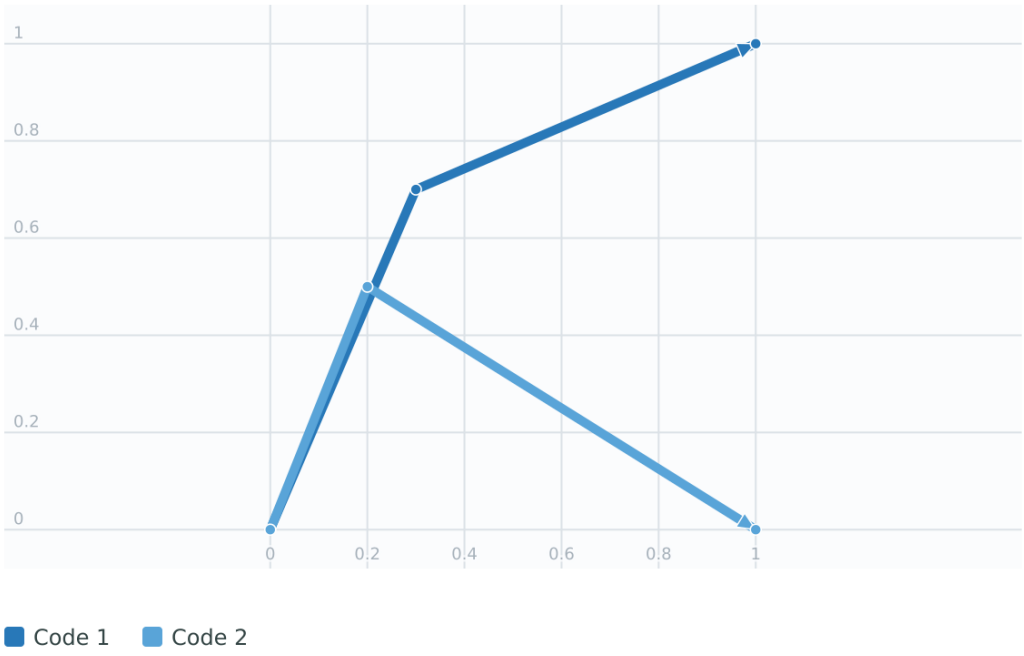


Winkel zwischen Vektoren, die durch die Anfangs- und Endpunkte von Linien definiert sind

Code

```
SELECT degrees(ST_Angle('LINESTRING(0 0,0.3 0.7,1 1)', 'LINESTRING(0 0,0.2 0.5,1 0)'));
```

Figure



Siehe auch

ST_Azimuth

7.12.4 ST_ClosestPoint

ST_ClosestPoint — Gibt den 2D-Punkt auf g1 zurück, der g2 am nächsten ist. Dies ist der erste Punkt der kürzesten Linie von einer Geometrie zur anderen.

Synopsis

geometry **ST_ClosestPoint**(geometry geom1, geometry geom2);
geography **ST_ClosestPoint**(geography geom1, geography geom2, boolean use_spheroid = true);

Beschreibung

Gibt den 2-dimensionalen Punkt auf `geom1` zurück, der `geom2` am nächsten liegt. Dies ist der erste Punkt der kürzesten Linie zwischen den Geometrien (wie von ST_ShortestLine berechnet).



Anmerkung
Falls es sich um eine 3D-Geometrie handelt, sollten Sie ST_3DClosestPoint vorziehen.

Verbessert: 3.4.0 - Unterstützung für Geographie.

Verfügbarkeit: 1.5.0

Beispiele

Der nächstgelegene Punkt für einen Point und einen LineString ist der Punkt selbst. Der nächstgelegene Punkt für einen LineString und einen Punkt ist ein Punkt auf der Linie.

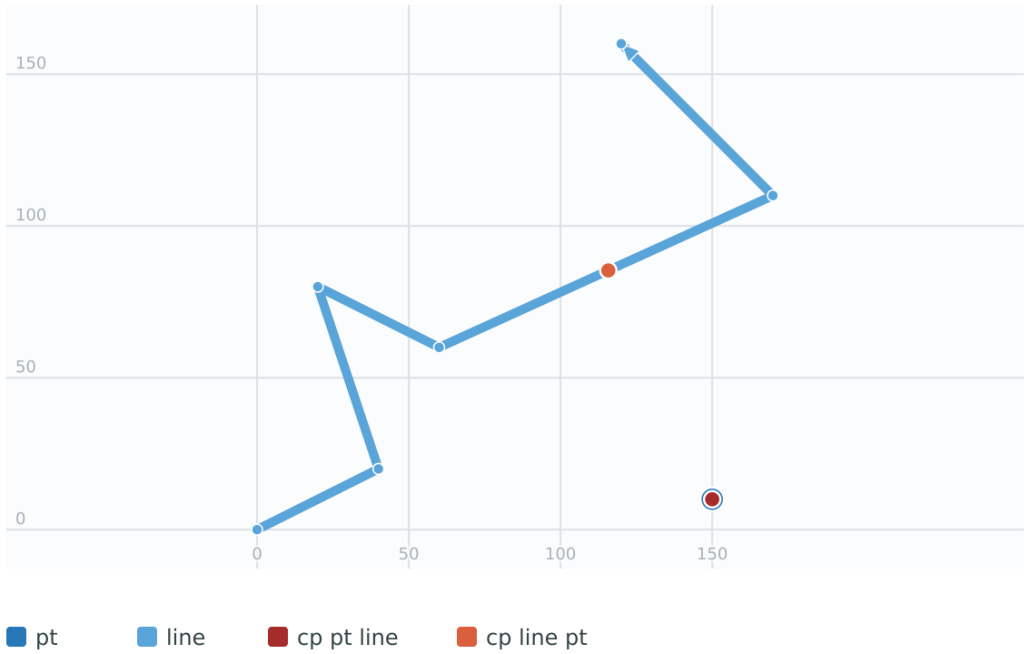
Code

```
SELECT ST_ClosestPoint(pt, line) AS cp_pt_line, ST_ClosestPoint(line, pt) AS cp_line_pt
    FROM (SELECT 'POINT (160 40)'::geometry AS pt,
                 'LINESTRING (10 30,50 50,30 110,70 90,180 140,130 190)'::geometry AS line ) ↩
     AS t;
```

Output

```
cp_pt_line    |               cp_line_pt
---------------+-----------------------------------------
 POINT(160 40)  | POINT(125.75342465753425 115.34246575342466)
```

Figure

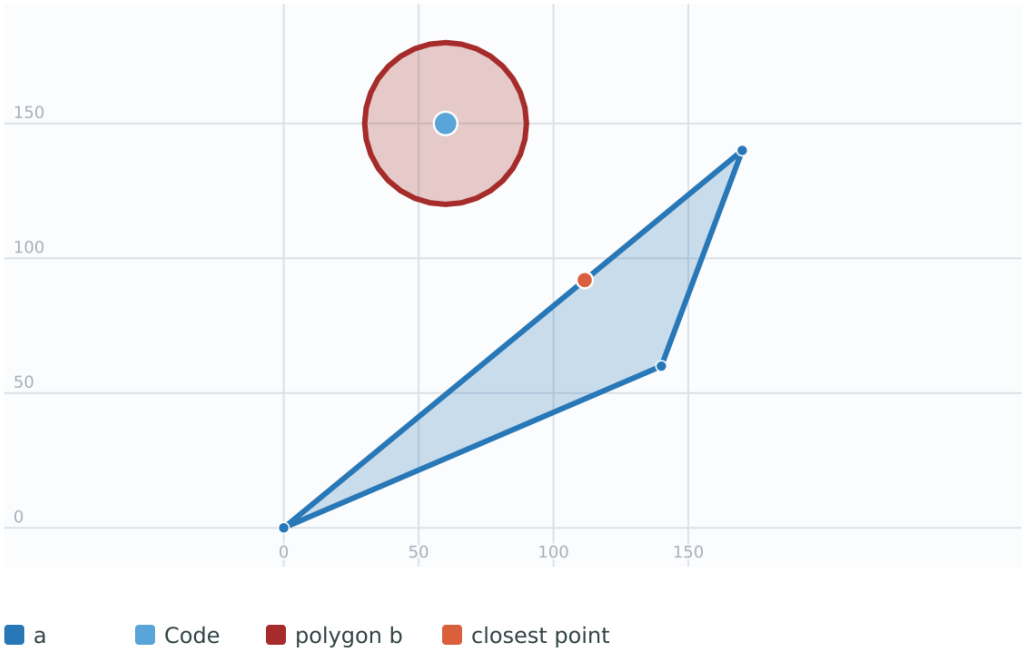


Find the closest point on polygon A to polygon B.

Code

```
WITH source AS (
  SELECT 'POLYGON ((190 150,20 10,160 70,190 150))'::geometry AS a,
         ST_Buffer('POINT(80 160)'::geometry, 30) AS b
)
SELECT b AS polygon_b,
       ST_ClosestPoint(a, b) AS closest_point
FROM source;
```

Figure



Siehe auch

ST_3DClosestPoint, ST_Distance, ST_LongestLine, ST_ShortestLine, ST_MaxDistance

7.12.5 ST_3DClosestPoint

ST_3DClosestPoint — Gibt den 3-dimensionalen Punkt auf g1 zurück, der den kürzesten Abstand zu g2 hat. Dies ist der Anfangspunkt des kürzesten Abstands in 3D.

Synopsis

geometry **ST_3DClosestPoint**(geometry g1, geometry g2);

Beschreibung

Gibt den 3-dimensionalen Punkt auf g1 zurück, der den kürzesten Abstand zu g2 hat. Dies ist der Anfangspunkt des kürzesten Abstands in 3D. Die Länge des kürzesten Abstands in 3D ergibt sich aus der 3D-Distanz.

Diese Funktion unterstützt 3d und lässt den Z-Index nicht fallen.

Diese Funktion unterstützt polyedrische Flächen.

Verfügbarkeit: 2.0.0

Änderung: 2.2.0 - Wenn 2 geometrische Objekte in 2D übergegeben werden, wird ein 2D-Punkt zurückgegeben (anstelle wie früher 0 für ein fehlendes Z). Im Falle von 2D und 3D, wird für fehlende Z nicht länger 0 angenommen.

Beispiele

Compare the 3D closest point to the 2D closest point for a LineString and a Point.

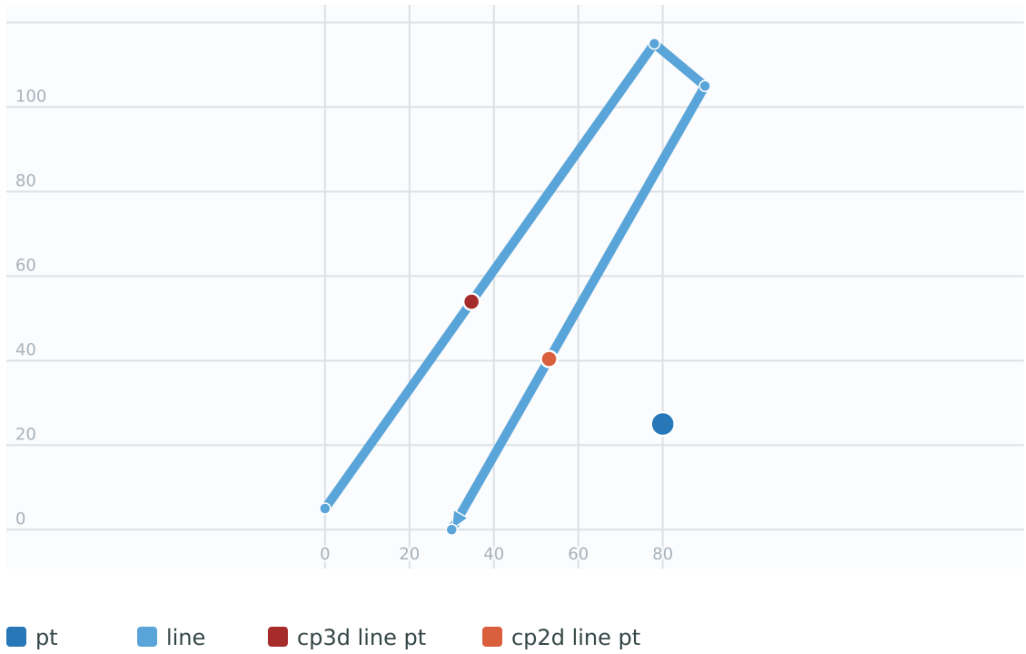
Code

```
WITH input AS (
  SELECT 'POINT(100 100 30)'::geometry AS pt,
         'LINESTRING (20 80 20,98 190 1,110 180 3,50 75 1000)'::geometry AS line
)
SELECT ST_3DClosestPoint(line, pt) AS cp3d_line_pt,
       ST_ClosestPoint(line, pt) AS cp2d_line_pt
FROM input;
```

Output

```
-[ RECORD 1 ]--+------------------------------------------------------
cp3d_line_pt   | POINT(54.69937988676193 128.93502291722837 11.547586950660556)
cp2d_line_pt   | POINT(73.07692307692307 115.38461538461539)
```

Figure



Compare the 3D closest point to the 2D closest point for a LineString and a MultiPoint.

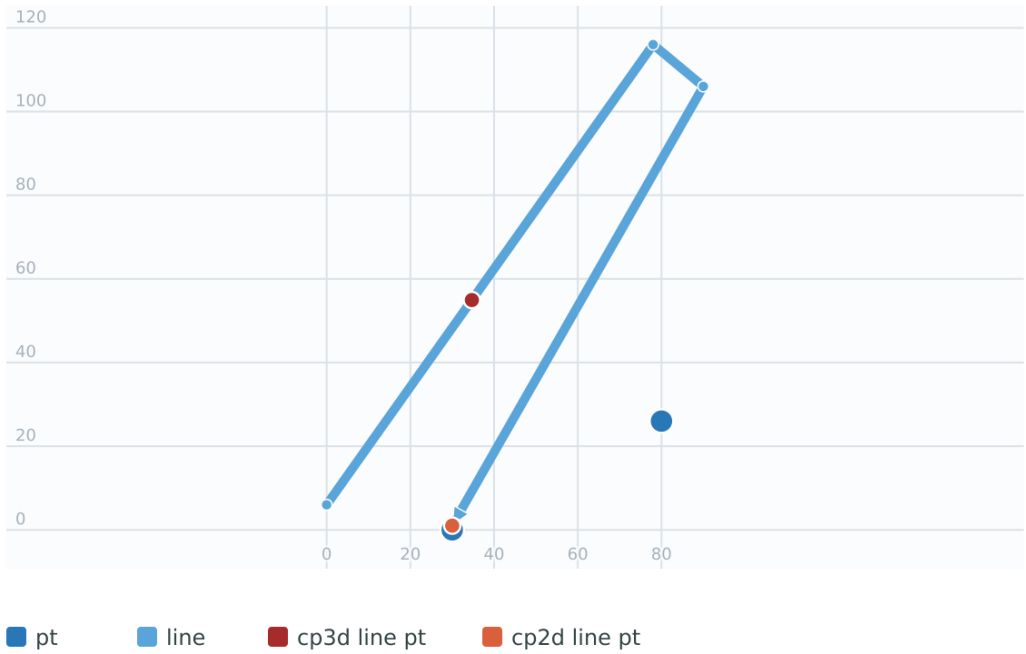
Code

```
WITH input AS (
  SELECT 'MULTIPOINT(100 100 30,50 74 1000)'::geometry AS pt,
         'LINESTRING (20 80 20,98 190 1,110 180 3,50 75 900)'::geometry AS line
)
SELECT ST_3DClosestPoint(line, pt) AS cp3d_line_pt,
       ST_ClosestPoint(line, pt) AS cp2d_line_pt
FROM input;
```

Output

```
-[ RECORD 1 ]--+------------------------------------------------------
cp3d_line_pt   | POINT(54.69937988676193 128.93502291722837 11.547586950660556)
cp2d_line_pt   | POINT(50 75)
```

Figure



Compare the 3D closest point to the 2D closest point for a Polygon and a MultiLineString.

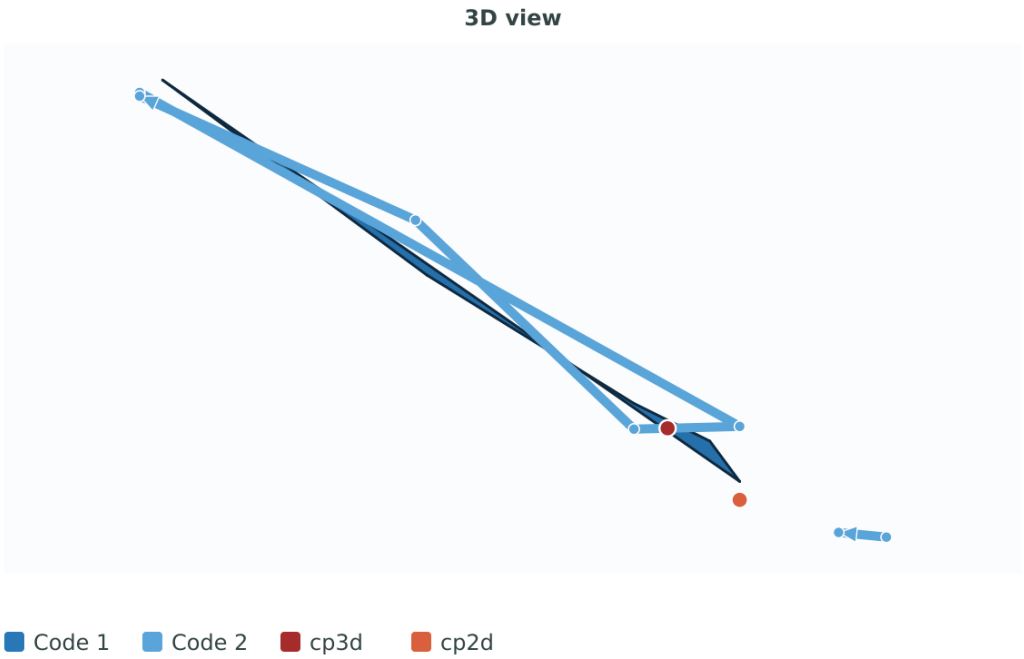
Code

```
WITH input AS (
  SELECT ST_GeomFromEWKT(
           'POLYGON((175 150 5,20 40 5,35 45 5,50 60 5,100 100 5,175 150 5))'
         ) AS poly,
         ST_GeomFromEWKT(
           'MULTILINESTRING((175 155 2,20 40 20,50 60 -2,125 100 1,175 155 1),
                            (1 10 2,5 20 1))'
         ) AS mline
)
SELECT ST_3DClosestPoint(poly, mline) AS cp3d,
       ST_ClosestPoint(poly, mline) AS cp2d
FROM input;
```

Output

```
-[ RECORD 1 ]--+---------------------------------------
cp3d           | POINT(40.45454545454545 53.63636363636364 5)
cp2d           | POINT(20 40)
```

Figure



Siehe auch

ST_AsEWKT, ST_ClosestPoint, ST_3DDistance, ST_3DShortestLine

7.12.6 ST_Distance

ST_Distance — Gibt die größte 3-dimensionale Distanz zwischen zwei geometrischen Objekten als Linie zurück

Synopsis

float **ST_Distance**(geometry g1, geometry g2);
float **ST_Distance**(geography geog1, geography geog2, boolean use_spheroid = true);

Beschreibung

Für den geometrischen Datentyp. Es wird der geringste 3-dimensionale kartesische Abstand zwischen zwei geometrischen Objekten in projizierten Einheiten (Einheiten des Koordinatenreferenzsystem) zurückgegeben.

Beim geometrischen Datentyp geometry wird die geringste kartesische Distanz in 2D zwischen zwei geometrischen Objekten - in projizierten Einheiten (Einheiten des Koordinatenreferenzsystem) - zurückgegeben. Beim geographischen Datentyp geography wird standardmäßig die geringste geodätische Distanz zwischen zwei geographischen Objekten in Meter zurückgegeben. Wenn use_speroid FALSE ist, erfolgt eine schnellere Berechnung auf einer Kugel anstatt auf dem Referenzellipsoid.

Diese Methode implementiert die OGC Simple Features Implementation Specification for SQL 1.1.

Diese Methode setzt die SQL/MM-Spezifikation um. SQL-MM 3: 5.1.23

Diese Methode unterstützt kreisförmige Strings und Kurven.

Verfügbarkeit: 1.5.0 die Unterstützung des geograpischen Datentyps wurde eingeführt. Geschwindigkeitsverbesserungen bei einer umfangreichen Geometrie und bei einer Geometrie mit vielen Knoten

Erweitert: 2.1.0 verbesserte die Geschwindigkeit für Geographie. Weitere Details finden Sie unter Making Geography faster.

Erweiterung: 2.1.0 - Unterstützung für Kurven beim geometrischen Datentyp eingeführt.

Erweiterung: 2.2.0 - die Messung auf dem Referenzellipsoid wird mit der Bibliothek "GeographicLib" durchgeführt. Dadurch wurde die Genauigkeit und die Robustheit erhöht. Um die Vorteile dieser neuen Funktionalität zu nutzen, benötigen Sie Proj >= 4.9.0.

Geändert: 3.0.0 - hängt nicht mehr von SFCGAL ab.

Beispiele mit dem geometrischen Datentyp

Geometriebeispiel - Einheiten in Grad 4326 ist WGS 84 long lat, Einheiten sind Grad.

Code

```
SELECT ST_Distance(
'SRID=4326;POINT(-72.1235 42.3521)'::geometry,
'SRID=4326;LINESTRING(-72.1260 42.45,-72.123 42.1546)'::geometry );
```

Output

```
0.00150567726382282
```

Geometriebeispiel - Einheiten in Metern (SRID: 3857, proportional zu Pixeln auf gängigen Webkarten). Obwohl der Wert nicht stimmt, können nahe gelegene Werte korrekt verglichen werden, was ihn zu einer guten Wahl für Algorithmen wie KNN oder KMeans macht.

Code

```
SELECT ST_Distance(
ST_Transform('SRID=4326;POINT(-72.1235 42.3521)'::geometry, 3857),
ST_Transform('SRID=4326;LINESTRING(-72.1260 42.45,-72.123 42.1546)'::geometry, 3857) );
```

Output

```
167.441410065196
```

Geometriebeispiel - Einheiten in Metern (SRID: 3857 wie oben, aber korrigiert um cos(lat), um die Verzerrung zu berücksichtigen)

Code

```
SELECT ST_Distance(
ST_Transform('SRID=4326;POINT(-72.1235 42.3521)'::geometry, 3857),
ST_Transform('SRID=4326;LINESTRING(-72.1260 42.45,-72.123 42.1546)'::geometry, 3857)
        ) * cosd(42.3521);
```

Output

```
123.742351254151
```

Geometriebeispiel - Einheiten in Metern (SRID: 26986 Massachusetts state plane meters) (am genauesten für Massachusetts)

Code

```
SELECT ST_Distance(
ST_Transform('SRID=4326;POINT(-72.1235 42.3521)'::geometry, 26986),
ST_Transform('SRID=4326;LINESTRING(-72.1260 42.45,-72.123 42.1546)'::geometry, 26986) );
```

Output

```
123.797937878454
```

Geometriebeispiel - Einheiten in Metern (SRID: 2163 US National Atlas Equal area) (am wenigsten genau)

Code

```
SELECT ST_Distance(
ST_Transform('SRID=4326;POINT(-72.1235 42.3521)'::geometry, 2163),
ST_Transform('SRID=4326;LINESTRING(-72.1260 42.45,-72.123 42.1546)'::geometry, 2163) );
```

Output

```
126.664256056812
```

Beispiele für den geographischen Datentyp

Wie im Geometrie-Beispiel, aber beachten Sie die Einheiten in Metern - verwenden Sie die Kugel für eine etwas schnellere und weniger genaue Berechnung.

Code

```
SELECT ST_Distance(gg1, gg2) As spheroid_dist, ST_Distance(gg1, gg2, false) As sphere_dist
FROM (SELECT
    'SRID=4326;POINT(-72.1235 42.3521)'::geography as gg1,
    'SRID=4326;LINESTRING(-72.1260 42.45,-72.123 42.1546)'::geography as gg2
    ) As foo  ;
```

Output

```
spheroid_dist   |   sphere_dist
------------------+------------------
 123.802076746848 | 123.475736916397
```

Siehe auch

ST_3DDistance, ST_DWithin, ST_DistanceSphere, ST_DistanceSpheroid, ST_MaxDistance, ST_HausdorffDistance, ST_FrechetDistan
ST_Transform

7.12.7 ST_3DDistance

ST_3DDistance — Für den geometrischen Datentyp. Es wird der geringste 3-dimensionale kartesische Abstand (basierend auf dem Koordinatenreferenzsystem) zwischen zwei geometrischen Objekten in projizierten Einheiten zurückgegeben.

Synopsis

float **ST_3DDistance**(geometry g1, geometry g2);

Beschreibung

Für den geometrischen Datentyp. Es wird der geringste 3-dimensionale kartesische Abstand zwischen zwei geometrischen Objekten in projizierten Einheiten (Einheiten des Koordinatenreferenzsystem) zurückgegeben.

Diese Funktion unterstützt 3d und lässt den Z-Index nicht fallen.

Diese Funktion unterstützt polyedrische Flächen.

Diese Methode setzt die SQL/MM-Spezifikation um. SQL-MM ISO/IEC 13249-3

Verfügbarkeit: 2.0.0

Änderung: 2.2.0 - Im Falle von 2D und 3D wird für ein fehlendes Z nicht mehr 0 angenommen.

Geändert: 3.0.0 - SFCGAL-Version entfernt

Beispiele

Geometry example - units in meters (SRID: 2163 US National Atlas Equal area) (3D point and line compared 2D point and line) Note that currently no vertical datum support so Z is not transformed and assumed to be same units as final.

Code

```
SELECT ST_3DDistance(
            ST_Transform('SRID=4326;POINT(-72.1235 42.3521 4)'::geometry, 2163),
                ST_Transform('SRID=4326;LINESTRING(-72.1260 42.45 15,-72.123 42.1546 20)':: ↩
    geometry, 2163)
            ) As dist_3d,
            ST_Distance(
                ST_Transform('SRID=4326;POINT(-72.1235 42.3521)'::geometry, 2163),
                ST_Transform('SRID=4326;LINESTRING(-72.1260 42.45,-72.123 42.1546)':: ↩
    geometry, 2163)
            ) As dist_2d;
```

Output

```
dist_3d      |     dist_2d
-----------------+-----------------
 127.295059324629 | 126.66425605671
```

Multilinestring and polygon both 3D and 2D distance. Same example as 3D closest point example.

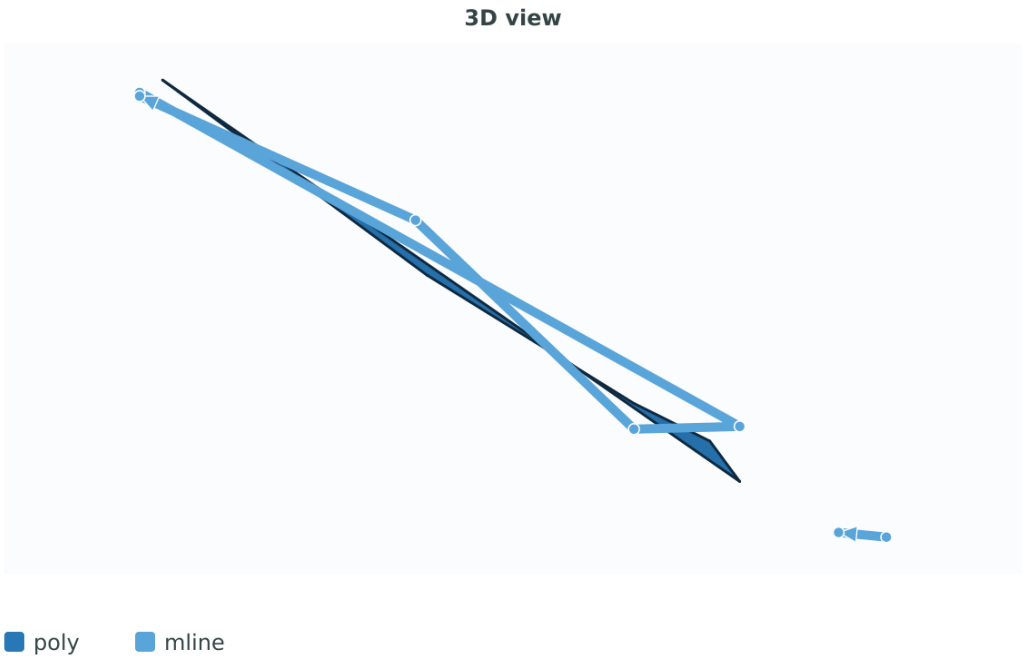
Code

```
SELECT ST_3DDistance(poly, mline) As dist3d,
    ST_Distance(poly, mline) As dist2d
        FROM (SELECT  'POLYGON((175 150 5,20 40 5,35 45 5,50 60 5,100 100 5,175 150 5))':: ↩
    geometry as poly,
                'MULTILINESTRING((175 155 2,20 40 20,50 60 -2,125 100 1,175 155 1),(1 10 2,5  ↩
    20 1))'::geometry as mline) as foo;
```

Output

```
dist3d       | dist2d
------------------+--------
                0 |      0
```

Figure



Siehe auch

ST_Distance, ST_3DClosestPoint, ST_3DDWithin, ST_3DMaxDistance, ST_3DShortestLine, ST_Transform

7.12.8 ST_DistanceSphere

ST_DistanceSphere — Gibt die kürzeste Distanz zwischen zwei geometrischen Objekten zurück, die über Länge, Breite und ein bestimmtes Referenzellipsoid gegeben sind. Vorgängerversionen von PostGIS 1.5 unterstützten nur Punkte.

Synopsis

float **ST_DistanceSphere**(geometry geomlonlatA, geometry geomlonlatB, float8 radius=6371008);

Beschreibung

Gibt die kürzeste Distanz zwischen zwei Punkten zurück, die über Länge und Breite gegeben sind. Verwendet die Kugelform für die Erde und den Radius des Referenzellipsoids, der durch die SRID festgelegt ist. Ist schneller als ST_DistanceSpheroid, aber weniger genau. Vorgängerversionen von PostGIS 1.5 unterstützten nur Punkte.

Verfügbarkeit: 1.5 die Unterstützung für weitere geometrische Datentypen neben Punkten eingeführt. Bei Vorgängerversionen wurden nur Punkte unterstützt.

Änderung: 2.2.0 In Vorgängerversionen als ST_Distance_Sphere bezeichet.

Beispiele

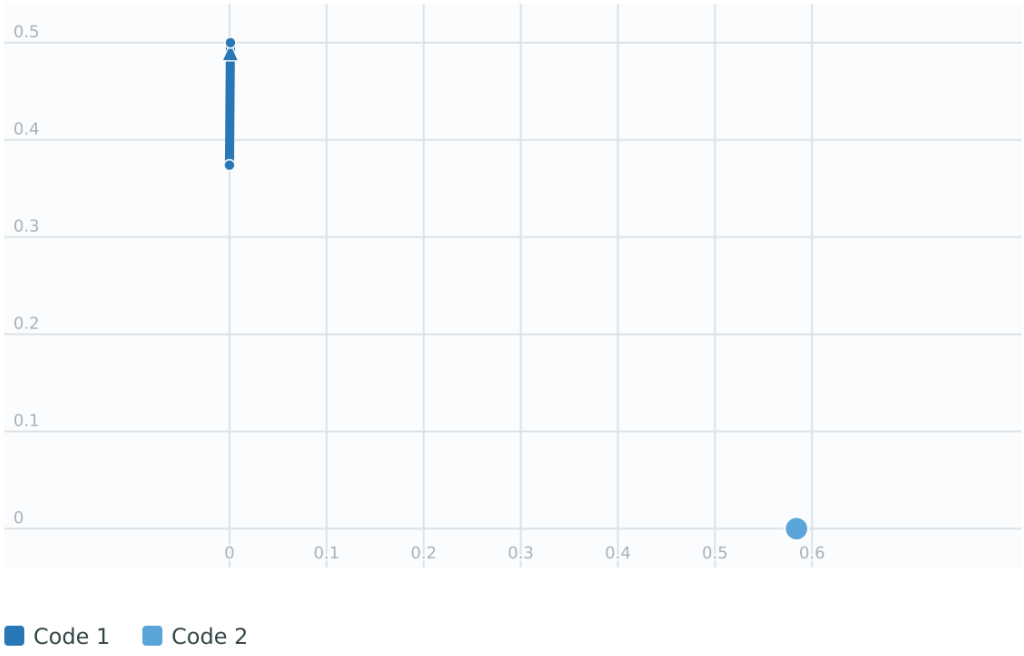
Code

```
WITH sample AS (
    SELECT
        ST_GeomFromText('LINESTRING(-118.584 38.374,-118.583 38.5)', 4326) AS geom,
        ST_GeomFromText('POINT(-118 38)', 4326) AS pt
)
SELECT
    round(ST_DistanceSphere(ST_Centroid(geom), pt)::numeric, 2) AS dist_meters,
    round(ST_Distance(
        ST_Transform(ST_Centroid(geom), 32611),
        ST_Transform(pt, 32611)
    )::numeric, 2) AS dist_utm11_meters,
    round(ST_Distance(ST_Centroid(geom), pt)::numeric, 5) AS dist_degrees,
    round(ST_Distance(
        ST_Transform(geom, 32611),
        ST_Transform(pt, 32611)
    )::numeric, 2) AS min_dist_line_point_meters
FROM sample;
```

Output

```
dist_meters | dist_utm11_meters | dist_degrees | min_dist_line_point_meters
------------+-------------------+--------------+---------------------------
    70424.71 |          70438.00 |      0.72900 |                   65871.18
```

Figure



Siehe auch

ST_Distance, ST_DistanceSpheroid

7.12.9 ST_DistanceSpheroid

ST_DistanceSpheroid — Gibt die kürzeste Distanz zwischen zwei geometrischen Objekten zurück, die über Länge, Breite und ein bestimmtes Referenzellipsoid gegeben sind. Vorgängerversionen von PostGIS 1.5 unterstützten nur Punkte.

Synopsis

float **ST_DistanceSpheroid**(geometry geomlonlatA, geometry geomlonlatB, spheroid measurement_spheroid=WGS84);

Beschreibung

Gibt die kürzeste Distanz zwischen zwei geometrischen Objekten in Meter zurück, die über Länge, Breite und ein bestimmtes Referenzellipsoid gegeben sind. Siehe die Erklärung zu Referenzellipsoiden unter ST_LengthSpheroid. Vorgängerversionen von PostGIS 1.5 unterstützten nur Punkte.



Anmerkung
Aktuell schaut diese Funktion nicht auf die SRID der Geometrie, sondern nimmt an, dass die Geometrie in den Koordinaten des gegebenen Referenzellipsoids vorliegt. Vorgängerversionen von PostGIS 1.5 unterstützten nur Punkte.

Verfügbarkeit: 1.5 die Unterstützung für weitere geometrische Datentypen neben Punkten eingeführt. Bei Vorgängerversionen wurden nur Punkte unterstützt.

Änderung: 2.2.0 In Vorgängerversionen als ST_Distance_Spheroid bezeichet.

Beispiele

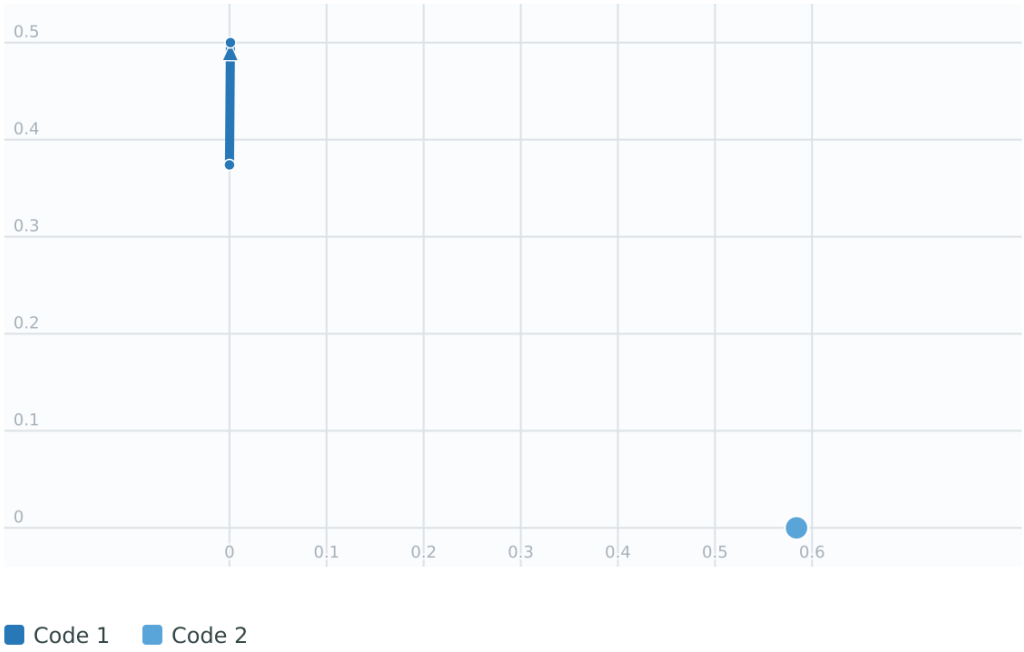
Code

```
WITH input AS (
  SELECT ST_Centroid(
           ST_GeomFromText(
             'LINESTRING(-118.584 38.374,-118.583 38.5)', 4326
           )
         ) AS source,
         ST_GeomFromText('POINT(-118 38)', 4326) AS target
)
SELECT round(
         ST_DistanceSpheroid(
           source,
           target,
           'SPHEROID["WGS 84",6378137,298.257223563]'
         )::numeric,
         2
       ) AS dist_meters_spheroid,
       round(ST_DistanceSphere(source, target)::numeric, 2)
         AS dist_meters_sphere,
       round(
         ST_Distance(
           ST_Transform(source, 32611),
           ST_Transform(target, 32611)
         )::numeric,
         2
       ) AS dist_utm11_meters
FROM input;
```

Output

```
dist_meters_spheroid | dist_meters_sphere | dist_utm11_meters
---------------------+--------------------+-------------------
             70454.92 |           70424.71 |          70438.00
```

Figure



Siehe auch

ST_Distance, ST_DistanceSphere

7.12.10 ST_FrechetDistance

ST_FrechetDistance — Gibt den kürzesten 3-dimensionalen Abstand zwischen zwei geometrischen Objekten als Linie zurück

Synopsis

float **ST_FrechetDistance**(geometry g1, geometry g2, float densifyFrac = -1);

Beschreibung

Implementiert einen Algorithmus zur Berechnung der Fréchet-Metrik, der für beide geometrischen Objekte auf diskrete Punkte beschränkt ist und auf Computing Discrete Fréchet Distance beruht. Die Fréchet-Metrik ist ein Maß für die Ähnlichkeit von Kurven, welches die Position und die Reihenfolge der Kurvenstützpunkte mit einbezieht. Daher ist sie oft besser geeignet als die Hausdorff-Metrik.

Wenn der optionale Parameter "densifyFrac" vorgegeben wird, dann führt diese Funktion eine Verdichtung der Linienstücke durch, bevor die diskrete Fréchet-Metrik berechnet wird. Der Parameter "densifyFrac" bestimmt um welchen Anteil die Linienstücke verdichtet werden. Jedes Linienstück wird in gleichlange Teilsegmente zerlegt, deren Anteil an der Gesamtlänge am nächsten an den vorgegebenen Anteil herankommt.

Die Einheiten sind in den Einheiten des räumlichen Bezugssystems der Geometrien angegeben.



Anmerkung
Bei der aktuellen Implementierung können die diskreten Punkte nur Knoten sein. Dies könnte erweitert werden, um eine beliebige Punktdichte zu ermöglichen.



Anmerkung
Umso kleiner wir densifyFrac wählen, umso genauer wird die Fréchet-Metrik. Aber die Rechenzeit und der Speicherplatzbedarf steigen quadratisch mit der Anzahl der Teilabschnitte.

Wird vom GEOS Modul ausgeführt

Verfügbarkeit: 2.4.0 - benötigt GEOS >= 3.7.0

Beispiele

Both lines are traversed from left to right. The first line is straight, while the second bows upward through its middle vertex. The point ordering matters, so keeping both inputs in the same direction makes the comparison clearer.

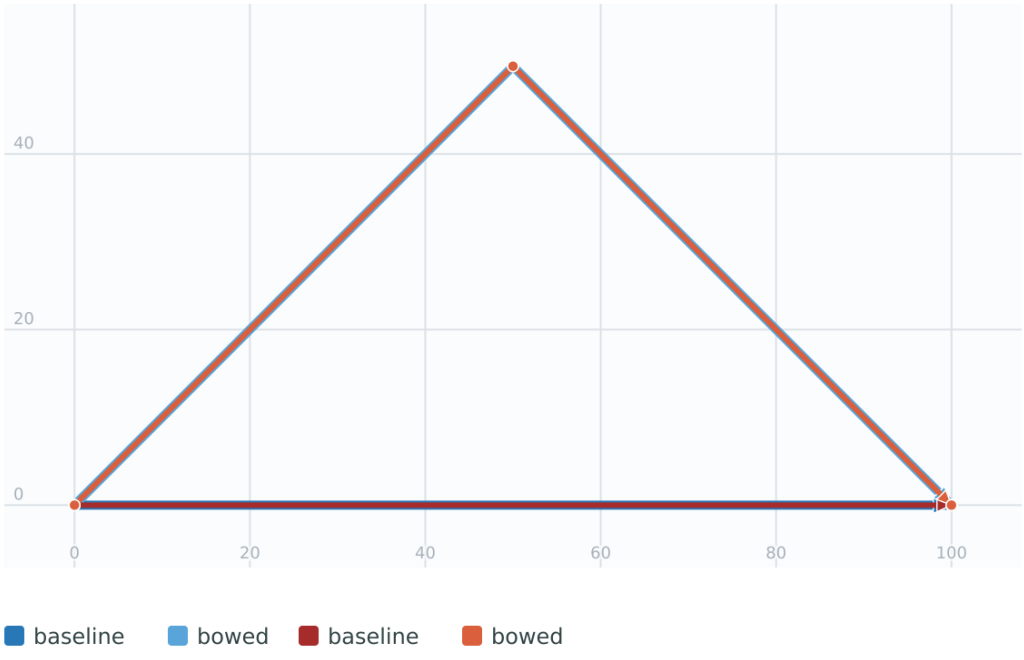
Code

```
WITH lines AS (
  SELECT 'LINESTRING (0 0,100 0)'::geometry AS baseline,
         'LINESTRING (0 0,50 50,100 0)'::geometry AS bowed
)
SELECT baseline AS baseline,
       bowed AS bowed,
       ST_FrechetDistance(baseline, bowed) AS discrete_distance,
       ST_FrechetDistance(baseline, bowed, 0.5) AS densified_distance
FROM lines;
```

Output

```
-[ RECORD 1 ]-----+----------------------------
baseline          | LINESTRING(0 0,100 0)
bowed             | LINESTRING(0 0,50 50,100 0)
discrete_distance | 70.7106781186548
densified_distance | 50
```

Figure



Siehe auch

ST_HausdorffDistance

7.12.11 ST_HausdorffDistance

ST_HausdorffDistance — Gibt den kürzesten 3-dimensionalen Abstand zwischen zwei geometrischen Objekten als Linie zurück

Synopsis

float **ST_HausdorffDistance**(geometry g1, geometry g2);
float **ST_HausdorffDistance**(geometry g1, geometry g2, float densifyFrac);

Beschreibung

Liefert den Hausdorff-Abstand zwischen zwei Geometrien. Der Hausdorff-Abstand ist ein Maß dafür, wie ähnlich oder unähnlich 2 Geometrien sind.

Die Funktion berechnet die "Diskrete Hausdorff-Distanz". Dies ist der Hausdorff-Abstand, der an diskreten Punkten der Geometrien berechnet wird. Der Parameter *densifyFrac* kann angegeben werden, um durch Verdichtung der Segmente vor der Berechnung des diskreten Hausdorff-Abstands eine genauere Antwort zu erhalten. Jedes Segment wird in eine Anzahl von Untersegmenten gleicher Länge aufgeteilt, deren Anteil an der Segmentlänge dem angegebenen Anteil am nächsten kommt.

Die Einheiten sind in den Einheiten des räumlichen Bezugssystems der Geometrien angegeben.



Anmerkung
Dieser Algorithmus ist NICHT gleichwertig mit dem Standard-Hausdorff-Abstand. Er berechnet jedoch eine Annäherung, die für eine große Teilmenge nützlicher Fälle korrekt ist. Ein wichtiger Fall sind Linien, die ungefähr parallel zueinander verlaufen und ungefähr gleich lang sind. Dies ist eine nützliche Metrik für die Anpassung von Linien.

Verfügbarkeit: 1.5.0

Beispiele

Hausdorff distance and minimum distance between two lines. The Hausdorff witness segment from the original illustration is retained and its length is checked against the computed distance.

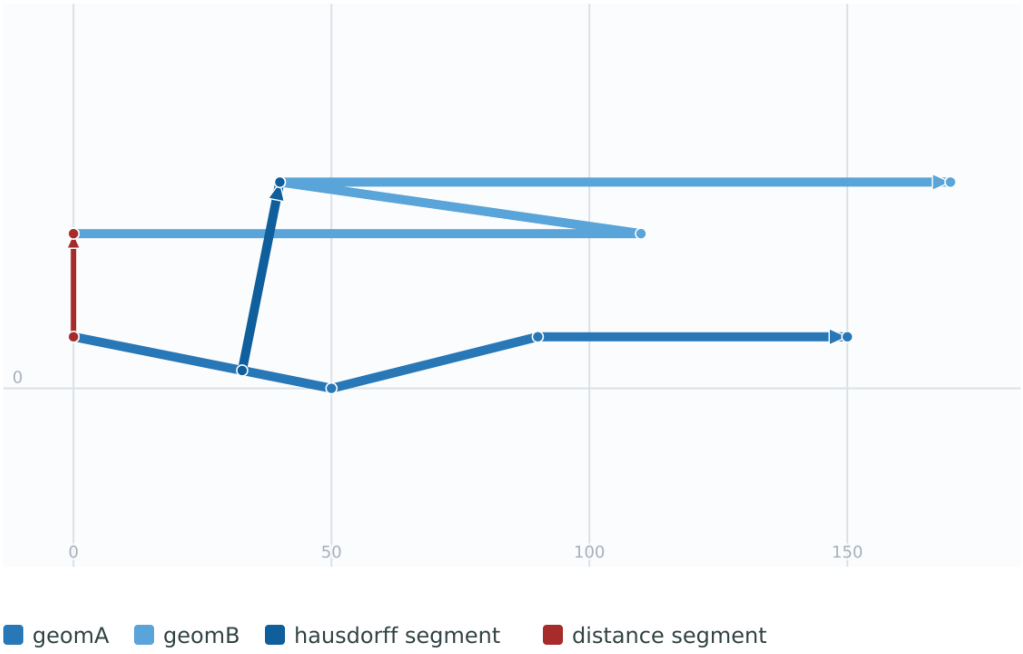
Code

```
WITH lines AS (
  SELECT 'LINESTRING (20 70,70 60,110 70,170 70)'::geometry AS geomA,
         'LINESTRING (20 90,130 90,60 100,190 100)'::geometry AS geomB,
         'LINESTRING (52.69230769230769 63.46153846153846,60 100)'::geometry AS ↩
    hausdorff_segment
)
SELECT round(ST_HausdorffDistance(geomA, geomB)::numeric, 6) AS hausdorff_distance,
       round(ST_Distance(geomA, geomB)::numeric, 6) AS distance,
       round(ST_Length(hausdorff_segment)::numeric, 6) AS witness_length,
       ST_ShortestLine(geomA, geomB) AS distance_segment
FROM lines;
```

Output

```
37.262066 | 20.000000 | 37.262066 | LINESTRING(20 70,20 90)
```

Figure



Beispiel: Hausdorff-Abstand mit Verdichtung.

Code

```
SELECT ST_HausdorffDistance(
            'LINESTRING (130 0,0 0,0 150)'::geometry,
            'LINESTRING (10 10,10 150,130 10)'::geometry,
            0.5);
```

Figure



Beispiel: Finden Sie für jedes Gebäude die Parzelle, die es am besten repräsentiert. Zunächst muss sich die Parzelle mit der Geometrie des Gebäudes schneiden. `DISTINCT ON` garantiert, dass jedes Gebäude nur einmal aufgeführt wird. `ORDER BY .. ST_HausdorffDistance` wählt die Parzelle aus, die dem Gebäude am ähnlichsten ist.

Code

```
SELECT DISTINCT ON (buildings.gid) buildings.gid, parcels.parcel_id
   FROM buildings
       INNER JOIN parcels
       ON ST_Intersects(buildings.geom, parcels.geom)
   ORDER BY buildings.gid, ST_HausdorffDistance(buildings.geom, parcels.geom);
```

Siehe auch

ST_FrechetDistance

7.12.12 ST_Length

ST_Length — Gibt den geometrischen Schwerpunkt einer Geometrie zurück.

Synopsis

float **ST_Length**(geometry a_2dlinestring);
float **ST_Length**(geography geog, boolean use_spheroid = true);

Beschreibung

Beim geometrischen Datentyp wird die kartesische 2D Länge der Geometrie zurückgegeben. Dabei muss es sich um einen LineString, einen MultiLineString, eine ST_Curve oder eine ST_MultiCurve handeln. Bei einer flächigen Geometrie wird 0 zurückgegeben. Für eine flächige Geometrie können Sie ST_Perimeter verwenden. Bei den geometrischen Datentypen sind die Einheiten für die Längenmessungen durch das Koordinatenreferenzsystem festgelegt.

For geography types: computation is performed using the inverse geodesic calculation. Units of length are in meters. The spheroid is specified by the SRID; when no SRID is provided, WGS84 is used. If `use_spheroid = false`, then the calculation is based on a sphere instead of a spheroid.

Zur Zeit ein Synonym für ST_Length2D; dies kann sich allerdings ändern, wenn höhere Dimensionen unterstützt werden.

Änderung: 2.0.0 Wesentliche Änderung -- In früheren Versionen ergab die Anwendung auf ein MULTI/POLYGON vom geographischen Datentyp den Umfang des POLYGON/MULTIPOLYGON. In 2.0.0 wurde dies geändert und es wird jetzt 0 zurückgegeben, damit es mit der Verhaltensweise beim geometrischen Datentyp übereinstimmt. Verwenden Sie bitte ST_Perimeter, wenn Sie den Umfang eines Polygons wissen wollen

---



Anmerkung
Beim geographischer Datentyp werden die Messungen standardmäßig am Referenzellipsoid durchgeführt. Für die schnellere, aber ungenauere Berechnung auf einer Kugel können Sie ST_Length(gg,false) verwenden;

 Diese Methode implementiert die OGC Simple Features Implementation Specification for SQL 1.1. s2.1.5.1

 Diese Methode setzt die SQL/MM-Spezifikation um. SQL-MM 3: 7.1.2, 9.3.4

Verfügbarkeit: 1.5.0 Unterstützung von geograpischen Koordinaten.

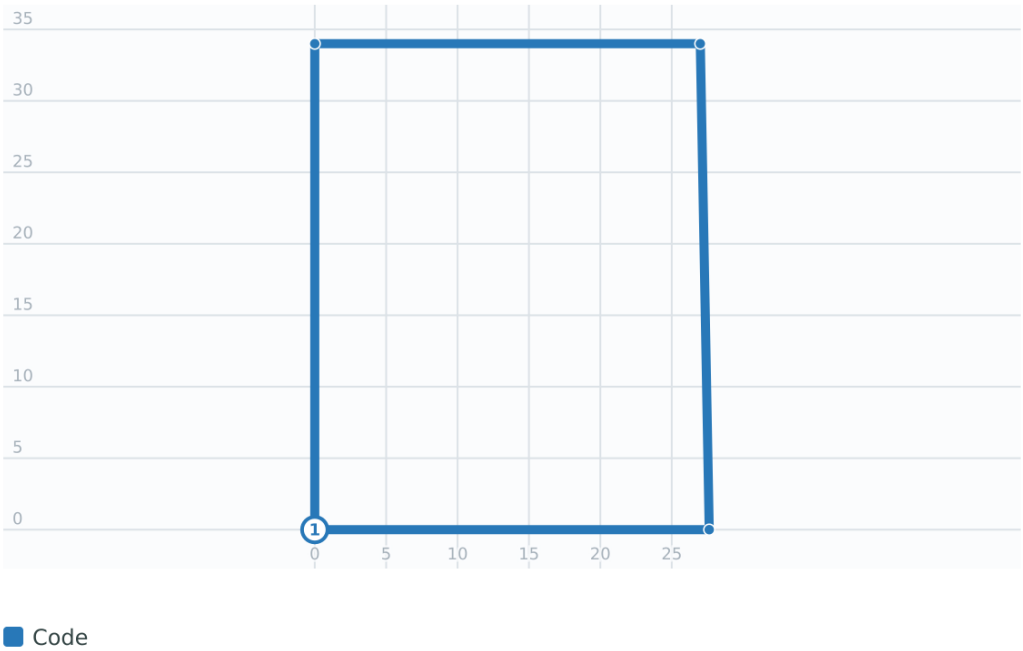
Beispiele mit dem geometrischen Datentyp

Gibt die Länge eines Linienstücks zurück. Beachten Sie, dass die Einheit Fuß ist, da EPSG:2249 "Massachusetts State Plane Feet" ist

Code

```
SELECT ST_Length(ST_GeomFromText(
'LINESTRING(743238 2967416,743238 2967450,743265 2967450,
743265.625 2967416,743238 2967416)',
2249));
```

Figure

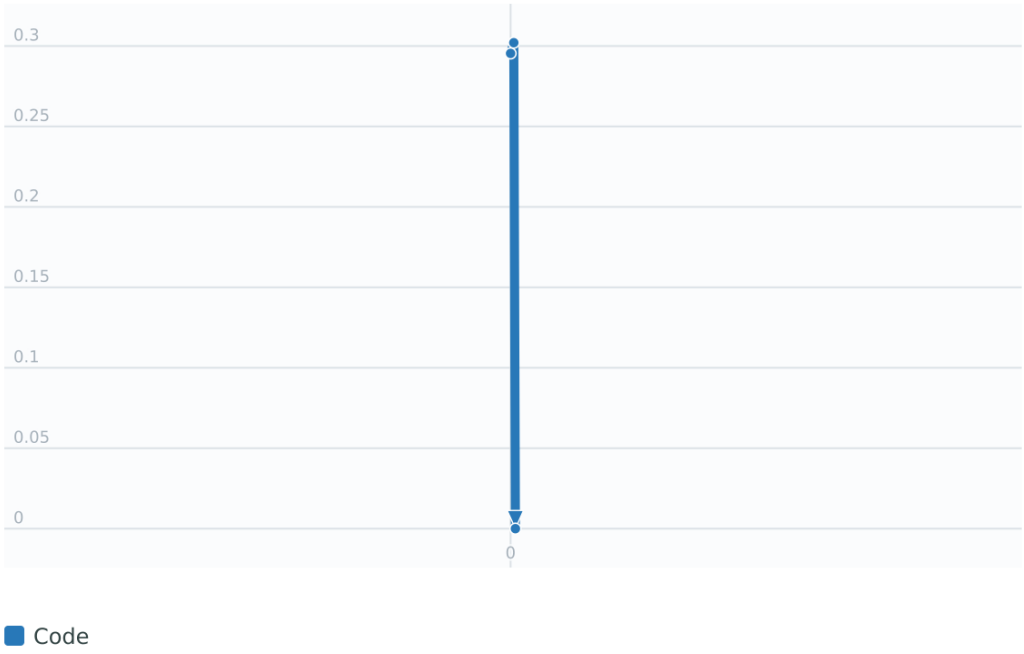


This example transforms a WGS 84 LineString to Massachusetts State Plane meters.

Code

```
SELECT ST_Length(ST_Transform(
        'SRID=4326;LINESTRING(-72.1260 42.45,-72.1240 42.45666,-72.123 42.1546)'::geometry,
            26986
    )
);
```

Figure



Beispiele für den geographischen Datentyp

Gibt die Länge einer Linie zurück, die in geographischen WGS 84 Koordinaten vorliegt.

This example uses the default spheroid calculation.

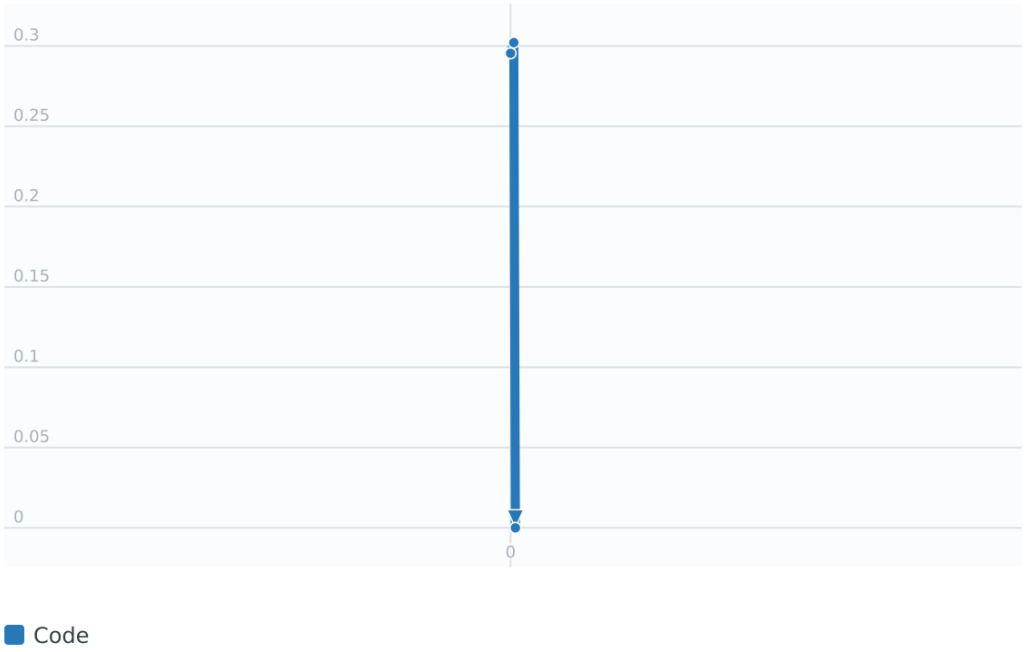
Code

```
SELECT ST_Length(the_geog) As length_spheroid, ST_Length(the_geog, false) As length_sphere
FROM (SELECT ST_GeographyFromText('SRID=4326;LINESTRING(-72.1260 42.45,-72.1240 ↩
    42.45666,-72.123 42.1546)') As the_geog)
 As foo;
```

Output

```
length_spheroid  |  length_sphere
-----------------+-----------------
34310.5703627284 | 34346.20609607416
```

Figure



Siehe auch

ST_GeographyFromText, ST_GeomFromEWKT, ST_LengthSpheroid, ST_Perimeter, ST_Transform

7.12.13 ST_Length2D

ST_Length2D — Gibt die 2-dimensionale Länge einer Linie oder einer Mehrfachlinie zurück. Dies ist ein Alias für `ST_Length`

Synopsis

float **ST_Length2D**(geometry a_2dlinestring);

Beschreibung

Gibt die 2-dimensionale Länge einer Linie oder einer Mehrfachlinie zurück. Dies ist ein Alias für `ST_Length`

Siehe auch

ST_Length, ST_3DLength

7.12.14 ST_3DLength

ST_3DLength — Gibt den geometrischen Schwerpunkt einer Geometrie zurück.

Synopsis

float **ST_3DLength**(geometry a_3dlinestring);

Beschreibung

Gibt die 2- oder 3-dimensionale Länge einer Linie oder einer Mehrfachlinie zurück. Bei einer 2-D Linie wird die Länge nur in 2D zurückgegeben (gleich wie ST_Length und ST_Length2D)

Diese Funktion unterstützt 3d und lässt den Z-Index nicht fallen.

Diese Methode setzt die SQL/MM-Spezifikation um. SQL-MM IEC 13249-3: 7.1, 10.3

Änderung: 2.0.0 In Vorgängerversionen als ST_Length3D bezeichet.

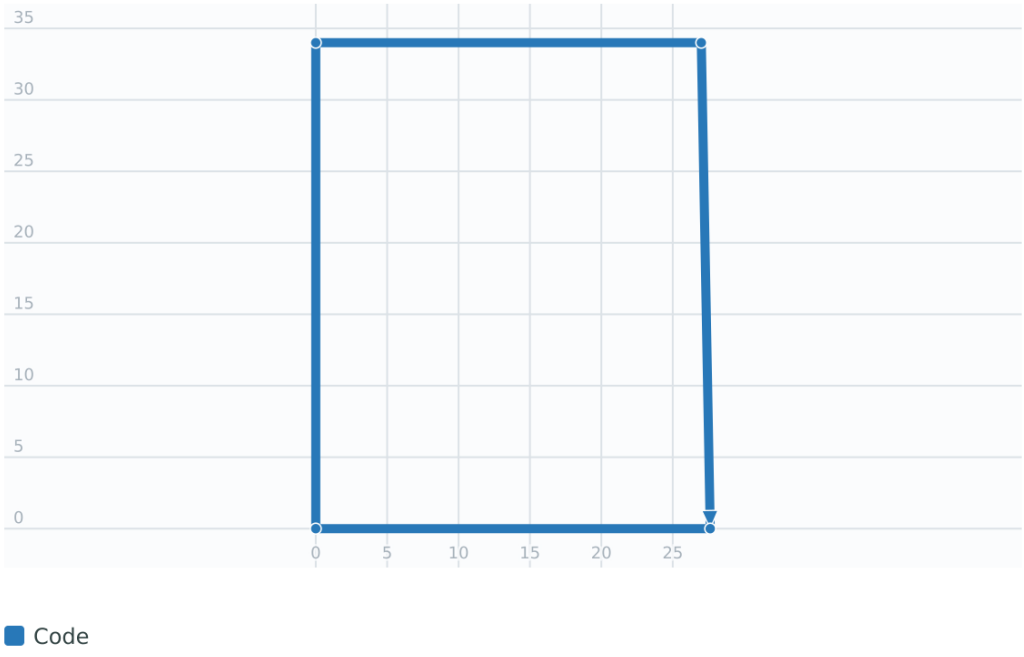
Beispiele

Gibt die Länge eines 3D-Kabels zurück. Beachten Sie, dass die Einheit Fuß ist, da EPSG:2249 "Massachusetts State Plane Feet" ist

Code

```
SELECT ST_3DLength(ST_GeomFromText(
'LINESTRING(743238 2967416 1,743238 2967450 1,743265 2967450 3,
743265.625 2967416 3,743238 2967416 3)',
2249));
```

Figure



Siehe auch

ST_Length, ST_Length2D

7.12.15 ST_LengthSpheroid

ST_LengthSpheroid — Gibt den geometrischen Schwerpunkt einer Geometrie zurück.

Synopsis

float **ST_LengthSpheroid**(geometry a_geometry, spheroid a_spheroid);

Beschreibung

Calculates the length or perimeter of a geometry on a spheroid. This is useful if the coordinates of the geometry are in longitude/latitude and a length is desired without reprojection. The spheroid is specified by a text value as follows:

```
SPHEROID[<NAME>,<SEMI-MAJOR AXIS>,<INVERSE FLATTENING>]
```

Geometrie Beispiel

```
SPHEROID["GRS_1980",6378137,298.257222101]
```

Verfügbarkeit: 1.2.2

Änderung: 2.2.0 In Vorgängerversionen als ST_Length_Spheroid bezeichet.und mit dem Alias "ST_3DLength_Spheroid" versehen

 Diese Funktion unterstützt 3d und lässt den Z-Index nicht fallen.

Beispiele

Measure a table geometry directly on a spheroid when the coordinates are stored in longitude and latitude.

Code

```
SELECT ST_LengthSpheroid(
    geometry_column,
    'SPHEROID["GRS_1980",6378137,298.257222101]'
    )
FROM geometry_table;
```

This example measures a 2D MultiLineString and each component separately on the same spheroid.

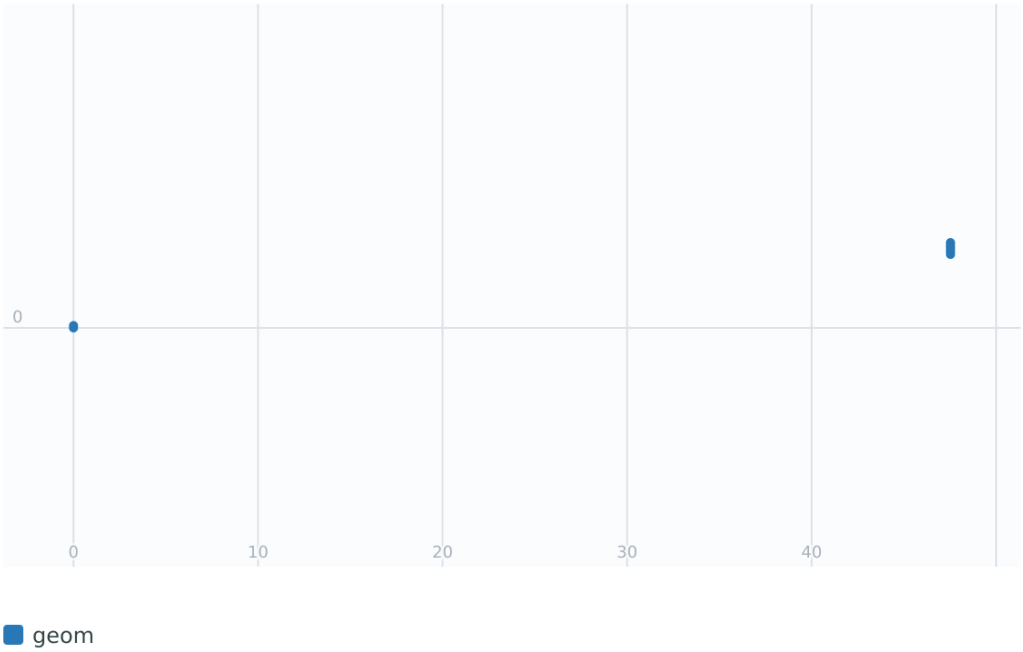
Code

```
WITH sample AS (
  SELECT 'MULTILINESTRING((-118.584 38.374,-118.583 38.5),
          (-71.05957 42.3589,-71.061 43))'::geometry AS geom,
         CAST('SPHEROID["GRS_1980",6378137,298.257222101]' AS spheroid) AS sph_m
)
SELECT ST_LengthSpheroid(geom, sph_m) AS tot_len,
       ST_LengthSpheroid(ST_GeometryN(geom, 1), sph_m) AS len_line1,
       ST_LengthSpheroid(ST_GeometryN(geom, 2), sph_m) AS len_line2
FROM sample;
```

Output

```
tot_len      |    len_line1     |    len_line2
-----------------+------------------+------------------
 85204.52077117894 | 13986.87252824332 | 71217.64824293563
```

Figure



This example uses 3D input; Z contributes to the measured result.

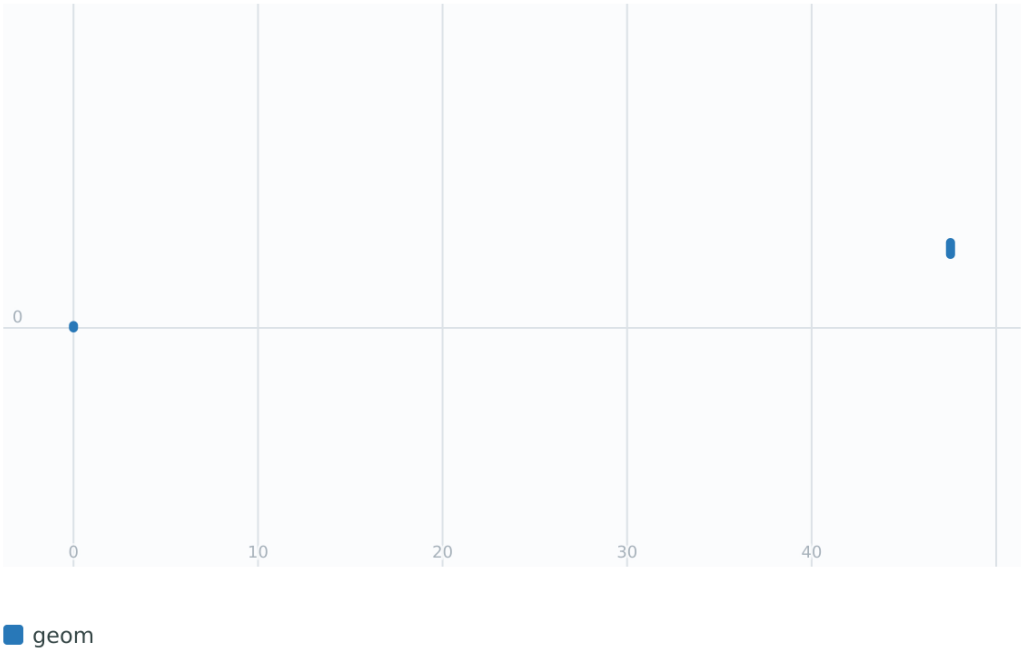
Code

```
WITH sample AS (
  SELECT 'MULTILINESTRING((-118.584 38.374 20,-118.583 38.5 30),
          (-71.05957 42.3589 75,-71.061 43 90))'::geometry AS geom,
         CAST('SPHEROID["GRS_1980",6378137,298.257222101]' AS spheroid) AS sph_m
)
SELECT ST_LengthSpheroid(geom, sph_m) AS tot_len,
       ST_LengthSpheroid(ST_GeometryN(geom, 1), sph_m) AS len_line1,
       ST_LengthSpheroid(ST_GeometryN(geom, 2), sph_m) AS len_line2
FROM sample;
```

Output

```
tot_len      |    len_line1    |    len_line2
-----------------+-----------------+-----------------
 85204.52592562366 | 13986.876103023424 | 71217.64982260024
```

Figure



Siehe auch

ST_GeometryN, ST_Length

7.12.16 ST_LongestLine

ST_LongestLine — Gibt die größte 3-dimensionale Distanz zwischen zwei geometrischen Objekten als Linie zurück

Synopsis

geometry **ST_LongestLine**(geometry g);
geometry **ST_LongestLine**(geometry g1, geometry g2);

Beschreibung

Liefert die 2-dimensionale längste Linie zwischen den Punkten zweier Geometrien. Die zurückgegebene Linie beginnt auf `g1` und endet auf `g2`.

Die längste Linie liegt immer zwischen zwei Scheitelpunkten. Die Funktion gibt die erste längste Linie zurück, wenn mehr als eine gefunden wird. Die Länge der Linie ist gleich dem von ST_MaxDistance zurückgegebenen Abstand.

If only one geometry is provided, or g1 and g2 are the same geometry, returns the line between the two vertices farthest apart in the geometry. The endpoints of the line lie on the circle computed by ST_MinimumBoundingCircle.

Verfügbarkeit: 1.5.0

Enhanced: 3.7.0 - support for a single geometry input.

Beispiele

Longest line between a Point and a LineString.

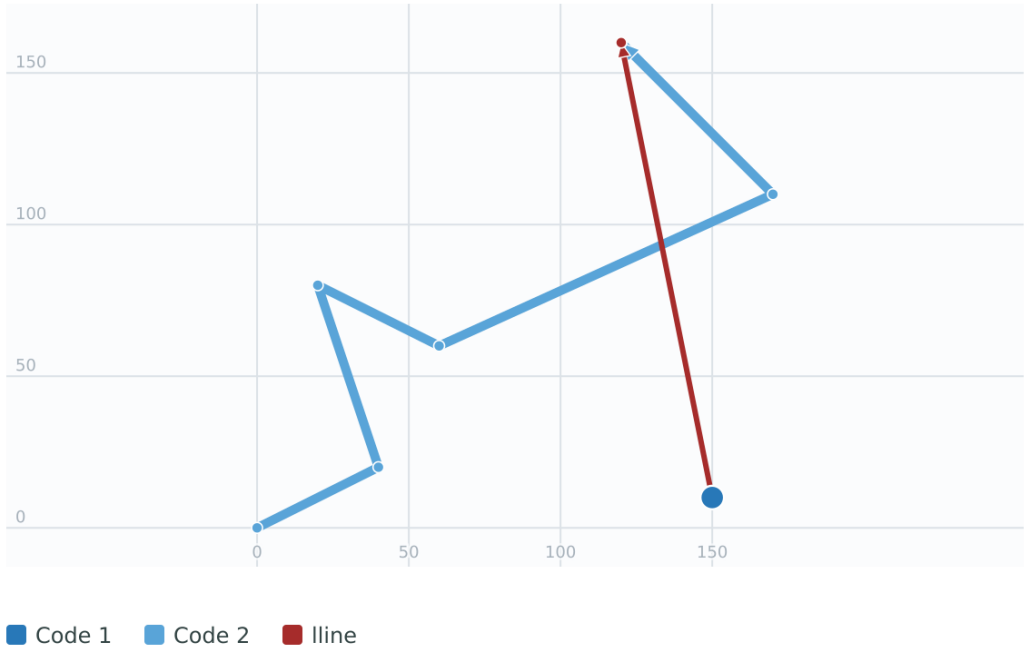
Code

```
SELECT ST_LongestLine(
        'POINT (160 40)',
        'LINESTRING (10 30,50 50,30 110,70 90,180 140,130 190)' ) AS lline;
```

Output

```
LINESTRING(160 40,130 190)
```

Figure



Longest line between two polygons.

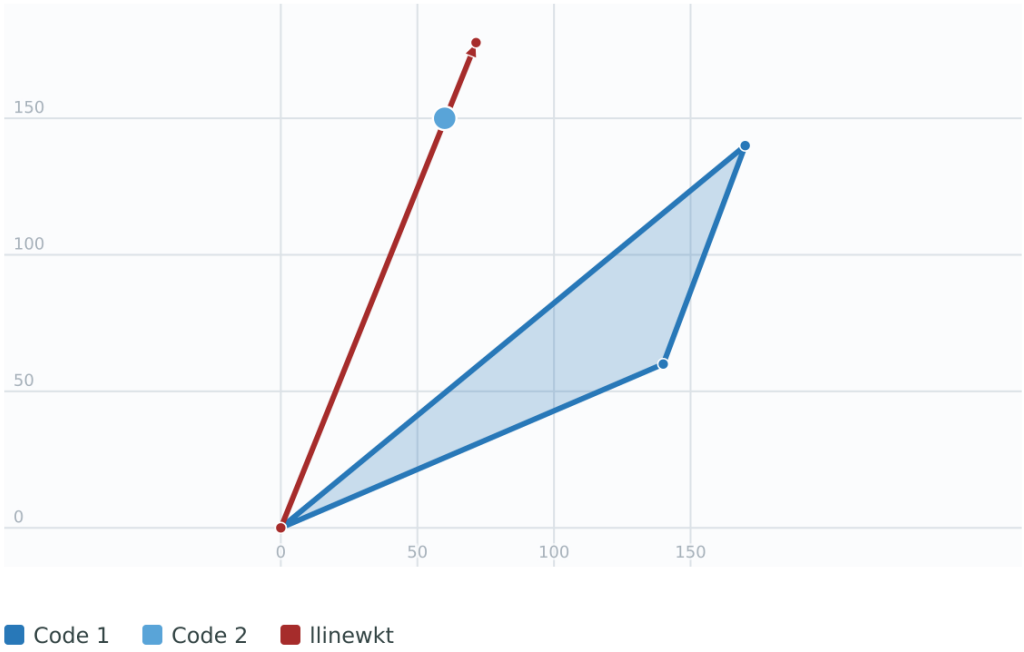
Code

```
SELECT ST_LongestLine(
        'POLYGON ((190 150,20 10,160 70,190 150))',
        ST_Buffer('POINT(80 160)', 30)
            ) AS llinewkt;
```

Output

```
LINESTRING(20 10,91.4805029709527 187.7163859753386)
```

Figure



Längste Linie durch eine einzelne Geometrie. Die Länge der Linie ist gleich dem Maximalen Abstand. Die Endpunkte der Linie liegen auf dem Minimalen Begrenzungskreis.

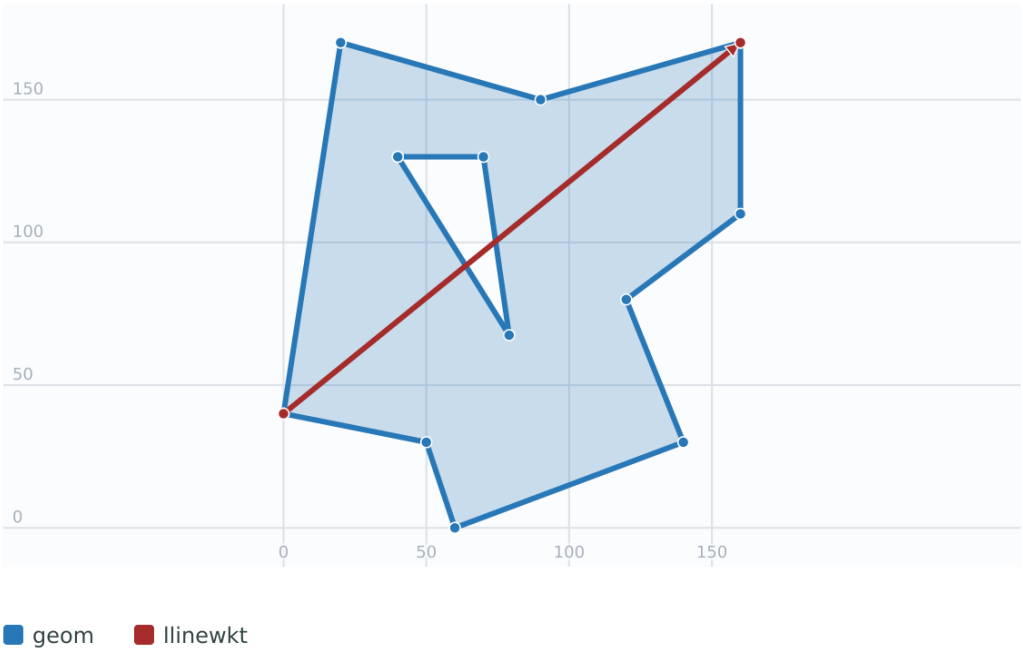
Code

```
SELECT ST_LongestLine(geom) AS llinewkt,
                  ST_MaxDistance(geom) AS max_dist,
                  ST_Length(ST_LongestLine(geom)) AS lenll
FROM (SELECT 'POLYGON ((40 180,110 160,180 180,180 120,140 90,160 40,80 10,70 40,20 50,40 ↩
    180),
              (60 140,99 77.5,90 140,60 140))'::geometry AS geom) AS t;
```

Output

```
llinewkt          |      max_dist      |       lenll
--------------------------+--------------------+--------------------
 LINESTRING(20 50,180 180) | 206.15528128088303 | 206.15528128088303
```

Figure



Siehe auch

ST_MaxDistance, ST_ShortestLine, ST_3DLongestLine, ST_MinimumBoundingCircle

7.12.17 ST_3DLongestLine

ST_3DLongestLine — Gibt die größte 3-dimensionale Distanz zwischen zwei geometrischen Objekten als Linie zurück

Synopsis

geometry **ST_3DLongestLine**(geometry g1, geometry g2);

Beschreibung

Gibt den größten 3-dimensionalen Abstand zwischen zwei geometrischen Objekten als Linie zurück. Wenn es mehr als einen größten Abstand gibt, dann wird nur die erste zurückgegeben. Die zurückgegebene Linie fängt immer mit "g1" an und endet mit "g2". Die Länge der 3D-Linie die von dieser Funktion zurückgegeben wird ist immer ident mit der von ST_3DMaxDistance für "g1" und "g2" zurückgegebenen Distanz.

Verfügbarkeit: 2.0.0

Änderung: 2.2.0 - Wenn 2 geometrische Objekte in 2D übergegeben werden, wird ein 2D-Punkt zurückgegeben (anstelle wie früher 0 für ein fehlendes Z). Im Falle von 2D und 3D, wird für fehlende Z nicht länger 0 angenommen.

Diese Funktion unterstützt 3d und lässt den Z-Index nicht fallen.

Diese Funktion unterstützt polyedrische Flächen.

Beispiele

Longest line between a LineString and a Point, comparing the 3D and 2D results.

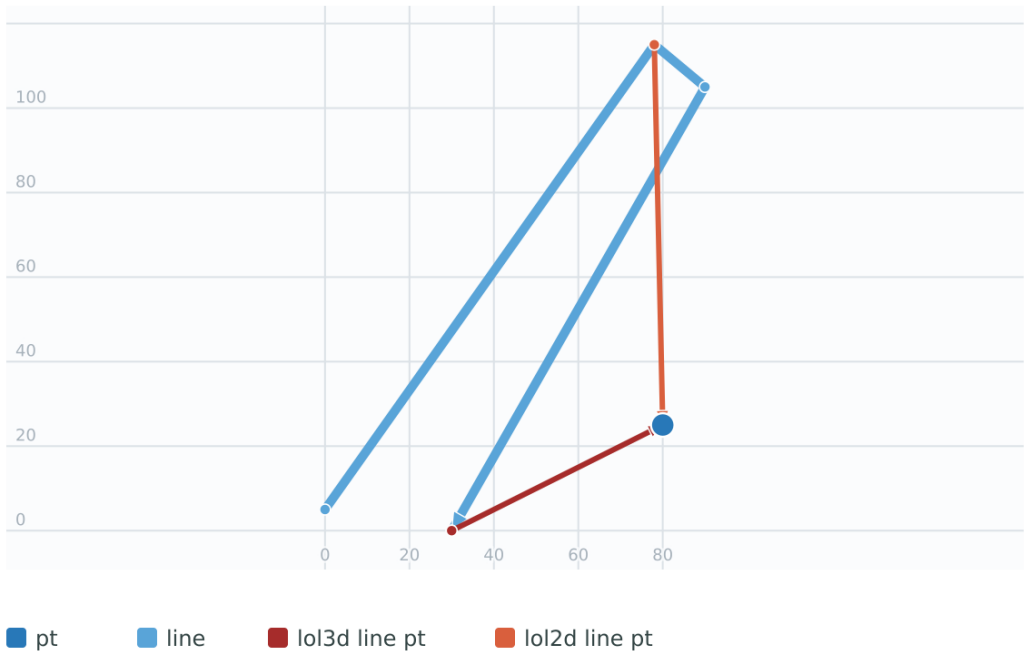
Code

```
SELECT ST_3DLongestLine(line, pt) AS lol3d_line_pt, ST_LongestLine(line, pt) AS  ↩
    lol2d_line_pt
FROM (
    SELECT 'POINT(100 100 30)'::geometry AS pt,
           'LINESTRING (20 80 20,98 190 1,110 180 3,50 75 1000)'::geometry AS line
) AS foo;
```

Output

```
lol3d_line_pt           |       lol2d_line_pt
--------------------------------+---------------------------
 LINESTRING(50 75 1000,100 100 30) | LINESTRING(98 190,100 100)
```

Figure



Longest line between a LineString and a MultiPoint, comparing the 3D and 2D results.

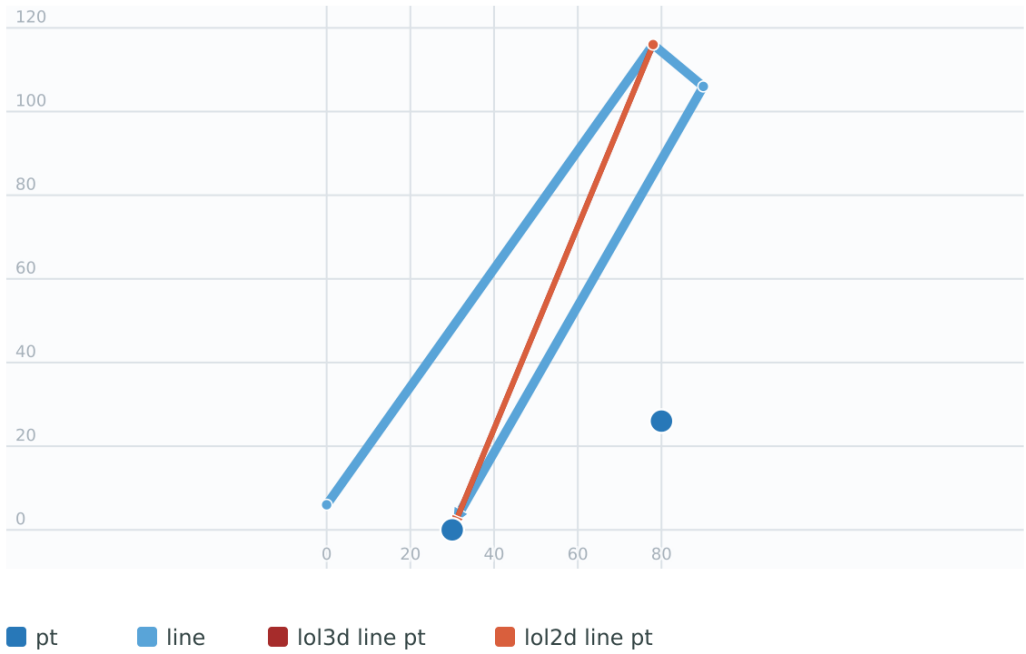
Code

```
SELECT ST_3DLongestLine(line, pt) AS lol3d_line_pt, ST_LongestLine(line, pt) AS  ↩
    lol2d_line_pt
FROM (
    SELECT 'MULTIPOINT(100 100 30,50 74 1000)'::geometry AS pt,
           'LINESTRING (20 80 20,98 190 1,110 180 3,50 75 900)'::geometry AS line
) AS foo;
```

Output

```
lol3d_line_pt          |      lol2d_line_pt
-------------------------------+--------------------------
 LINESTRING(98 190 1,50 74 1000) | LINESTRING(98 190,50 74)
```

Figure



Longest line between a Polygon and a MultiLineString, comparing the 3D and 2D results.

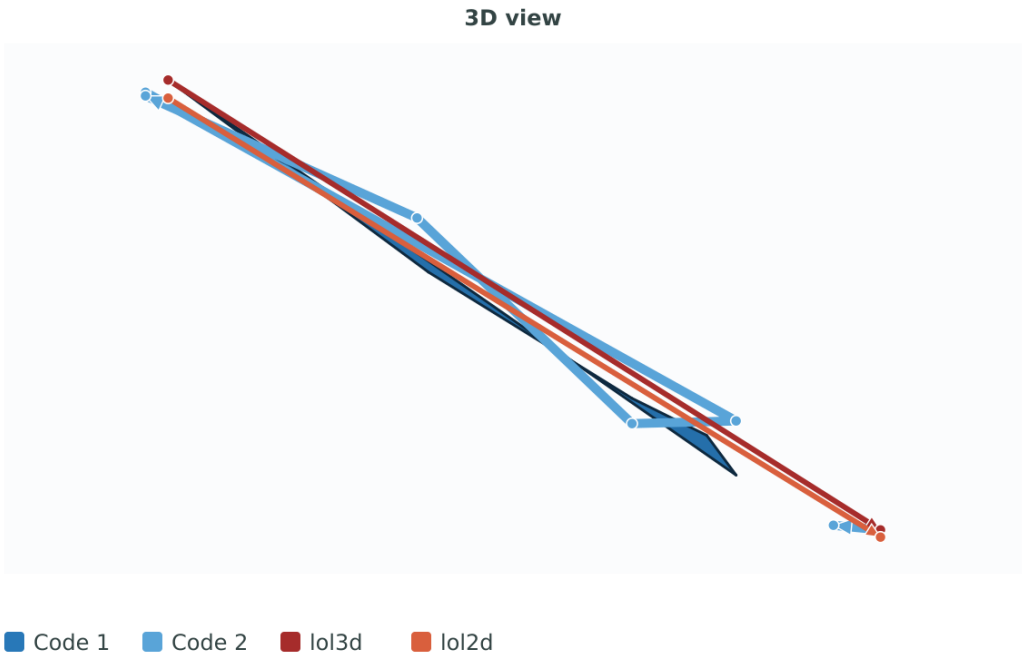
Code

```
SELECT ST_3DLongestLine(poly, mline) AS lol3d, ST_LongestLine(poly, mline) AS lol2d
FROM (
    SELECT ST_GeomFromEWKT(
            'POLYGON((175 150 5,20 40 5,35 45 5,50 60 5,100 100 5,175 150 5))'
        ) AS poly,
        ST_GeomFromEWKT(
            'MULTILINESTRING((175 155 2,20 40 20,50 60 -2,125 100 1,175 155 1),
            (1 10 2,5 20 1))'
        ) AS mline
) AS foo;
```

Output

```
lol3d                        |          lol2d
-----------------------------+-------------------------
 LINESTRING(175 150 5,1 10 2) | LINESTRING(175 150,1 10)
```

Figure



Siehe auch

ST_3DClosestPoint, ST_3DDistance, ST_LongestLine, ST_3DShortestLine, ST_3DMaxDistance

7.12.18 ST_MaxDistance

ST_MaxDistance — Gibt die größte 2-dimensionale Distanz zwischen zwei geometrischen Objekten in projizierten Einheiten zurück.

Synopsis

float **ST_MaxDistance**(geometry g);
float **ST_MaxDistance**(geometry g1, geometry g2);

Beschreibung

Gibt die größte 2-dimensionale Distanz zwischen zwei geometrischen Objekten in projizierten Einheiten zurück. Wenn g1 und g2 dieselbe Geometrie sind, dann gibt die Funktion die Distanz zwischen den beiden am weitesten entfernten Knoten in dieser Geometrie zurück.

If only one geometry is provided, or g1 and g2 are the same geometry, returns the distance between the two vertices farthest apart in that geometry.

Verfügbarkeit: 1.5.0

Enhanced: 3.7.0 - support for a single geometry input.

Beispiele

Längste Strecke zwischen Punkt und Linie

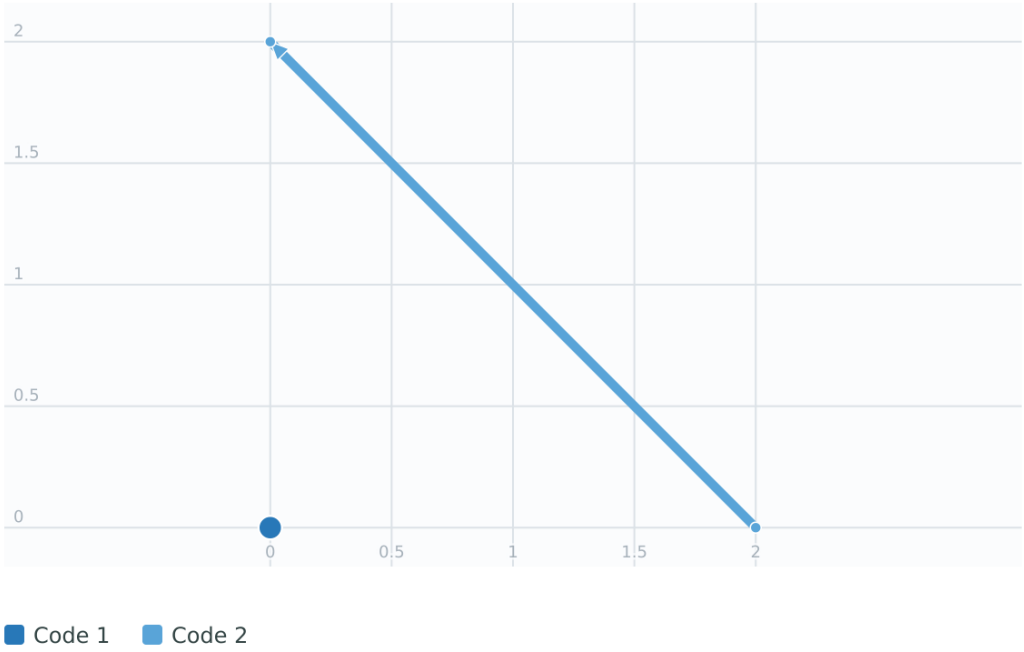
Code

```
SELECT ST_MaxDistance('POINT(0 0)'::geometry, 'LINESTRING ( 2 0,0 2 )'::geometry);
```

Output

```
2
```

Figure



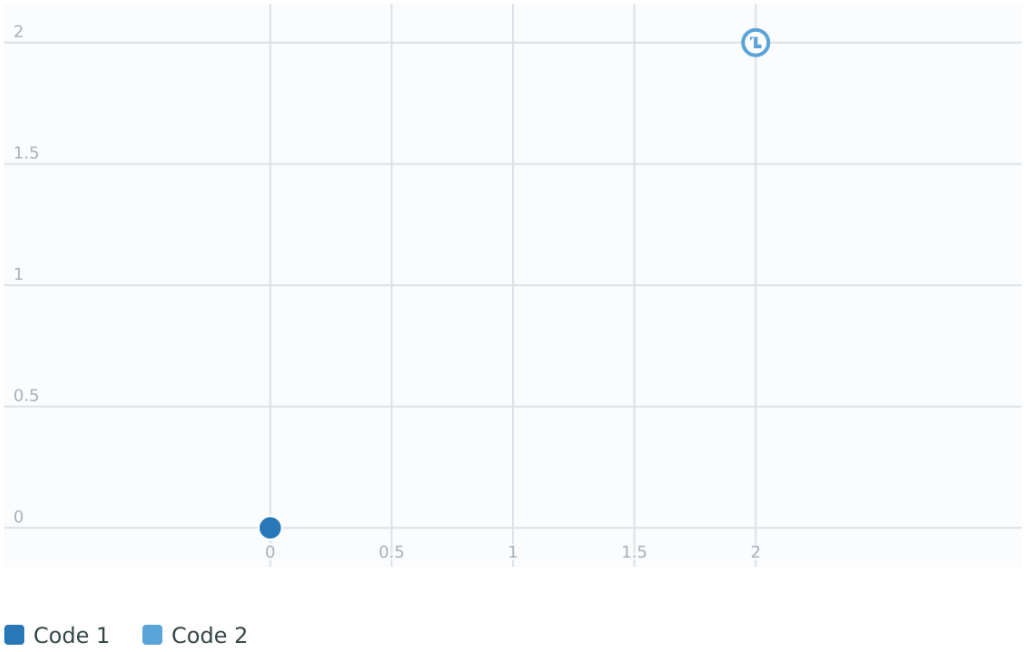
Code

```
SELECT ST_MaxDistance('POINT(0 0)'::geometry, 'LINESTRING ( 2 2,2 2 )'::geometry);
```

Output

```
2.82842712474619
```

Figure



Maximaler Abstand zwischen Scheitelpunkten einer einzelnen Geometrie.

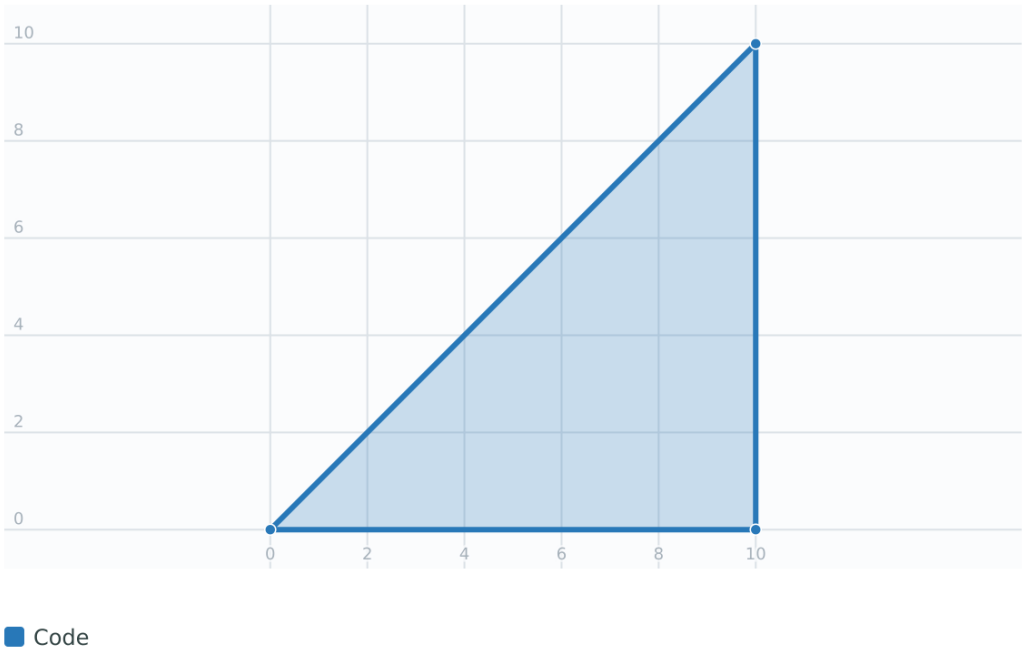
Code

```
SELECT ST_MaxDistance('POLYGON ((10 10,10 0,0 0,10 10))'::geometry);
```

Output

```
14.142135623730951
```

Figure



Siehe auch

ST_Distance, ST_LongestLine, ST_DFullyWithin

7.12.19 ST_3DMaxDistance

ST_3DMaxDistance — Für den geometrischen Datentyp. Gibt die maximale 3-dimensionale kartesische Distanz (basierend auf dem Koordinatenreferenzsystem) zwischen zwei geometrischen Objekten in projizierten Einheiten zurück.

Synopsis

float **ST_3DMaxDistance**(geometry g1, geometry g2);

Beschreibung

Für den geometrischen Datentyp. Gibt die maximale 3-dimensionale kartesische Distanz zwischen zwei geometrischen Objekten in projizierten Einheiten (Einheiten des Koordinatenreferenzsystem) zurück.

 Diese Funktion unterstützt 3d und lässt den Z-Index nicht fallen.

 Diese Funktion unterstützt polyedrische Flächen.

Verfügbarkeit: 2.0.0

Änderung: 2.2.0 - Im Falle von 2D und 3D wird für ein fehlendes Z nicht mehr 0 angenommen.

Beispiele

Geometry example - units in meters (SRID: 2163 US National Atlas Equal area) (3D point and line compared 2D point and line) Note that currently no vertical datum support so Z is not transformed and assumed to be same units as final.

Code

```
SELECT ST_3DMaxDistance(
            ST_Transform('SRID=4326;POINT(-72.1235 42.3521 10000)'::geometry, 2163),
                ST_Transform('SRID=4326;LINESTRING(-72.1260 42.45 15,-72.123 42.1546 20)':: ↩
    geometry, 2163)
            ) As dist_3d,
            ST_MaxDistance(
                ST_Transform('SRID=4326;POINT(-72.1235 42.3521 10000)'::geometry, 2163),
                ST_Transform('SRID=4326;LINESTRING(-72.1260 42.45 15,-72.123 42.1546 20)':: ↩
    geometry, 2163)
            ) As dist_2d;
```

Output

```
dist_3d      |     dist_2d
-----------------+------------------
 24383.7467488441 | 22247.8472107251
```

Siehe auch

ST_Distance, ST_3DDWithin, ST_3DMaxDistance, ST_Transform

7.12.20 ST_MinimumClearance

ST_MinimumClearance — Gibt das Mindestabstandsmaß für eine Geometrie zurück; ein Maß für die Robustheit einer Geometrie.

Synopsis

float **ST_MinimumClearance**(geometry g);

Beschreibung

Es ist nicht ungewöhnlich dass eine Geometrie, welche die Kriterien für Validität gemäß ST_IsValid (Polygone) oder ST_IsSimple (Linien) erfüllt, durch eine geringe Verschiebung von einem Knoten invalid wird. Dies kann während einer Konvertierung in Textformate (wie WKT, KML, GML und GeoJSON) vorkommen, oder bei binären Formaten, welche die Koordinaten nicht als Gleitpunkt-Zahl mit doppelter Genauigkeit (double-precision floating point) abspeichern (MapInfo TAB).

Der Mindestabstand ist ein quantitatives Maß für die Robustheit einer Geometrie gegenüber Änderungen der Koordinatengenauigkeit. Es ist der größte Abstand, um den die Eckpunkte der Geometrie verschoben werden können, ohne dass eine ungültige Geometrie entsteht. Größere Werte des Mindestabstands bedeuten eine größere Robustheit.

Wenn eine Geometrie ein Mindestabstandsmaß von e hat, dann gilt:

- Zwei sich unterscheidende Knoten der Geometrie sind nicht weniger als e voneinander entfernt.
- Kein Knoten liegt näher als e bei einem Liniensegment, außer es ist ein Endpunkt.

Wenn für eine Geometrie kein Mindestabstandsmaß existiert (zum Beispiel ein einzelner Punkt, oder ein Mehrfachpunkt, dessen Punkte identisch sind), dann gibt ST_MinimumClearance unendlich zurück.

Um durch Präzisionsverluste verursachte Gültigkeitsprobleme zu vermeiden, kann ST_ReducePrecision die Koordinatengenauigkeit verringern und gleichzeitig sicherstellen, dass die polygonale Geometrie gültig bleibt.

Verfügbarkeit: 2.3.0

Beispiele

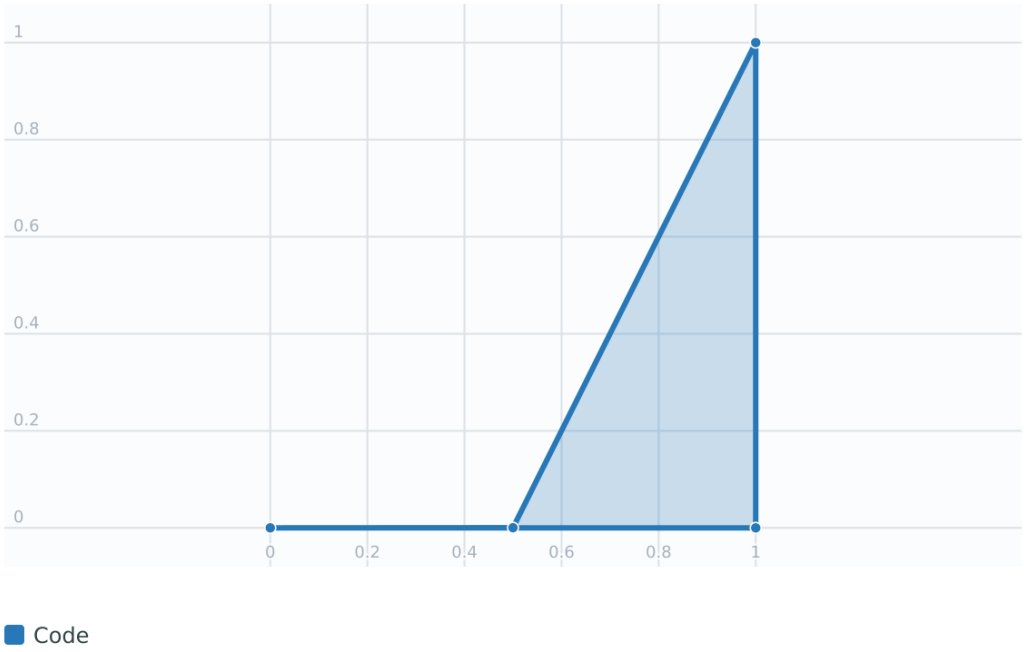
Code

```
SELECT ST_MinimumClearance('POLYGON ((0 0,1 0,1 1,0.5 3.2e-4,0 0))');
```

Output

```
0.00032
```

Figure



Siehe auch

ST_MinimumClearanceLine, ST_Crosses, ST_Dimension, ST_Intersects

7.12.21 ST_MinimumClearanceLine

ST_MinimumClearanceLine — Gibt ein Linienstück mit zwei Punkten zurück, welche sich über das Mindestabstandsmaß erstreckt.

Synopsis

Geometry **ST_MinimumClearanceLine**(geometry g);

Beschreibung

Gibt den Zweipunkt-LineString zurück, der den Mindestabstand einer Geometrie umspannt. Wenn die Geometrie keinen Mindestabstand hat, wird `LINESTRING EMPTY` zurückgegeben.

Wird vom GEOS Modul ausgeführt

Verfügbarkeit: 2.3.0 - benötigt GEOS >= 3.6.0

Beispiele

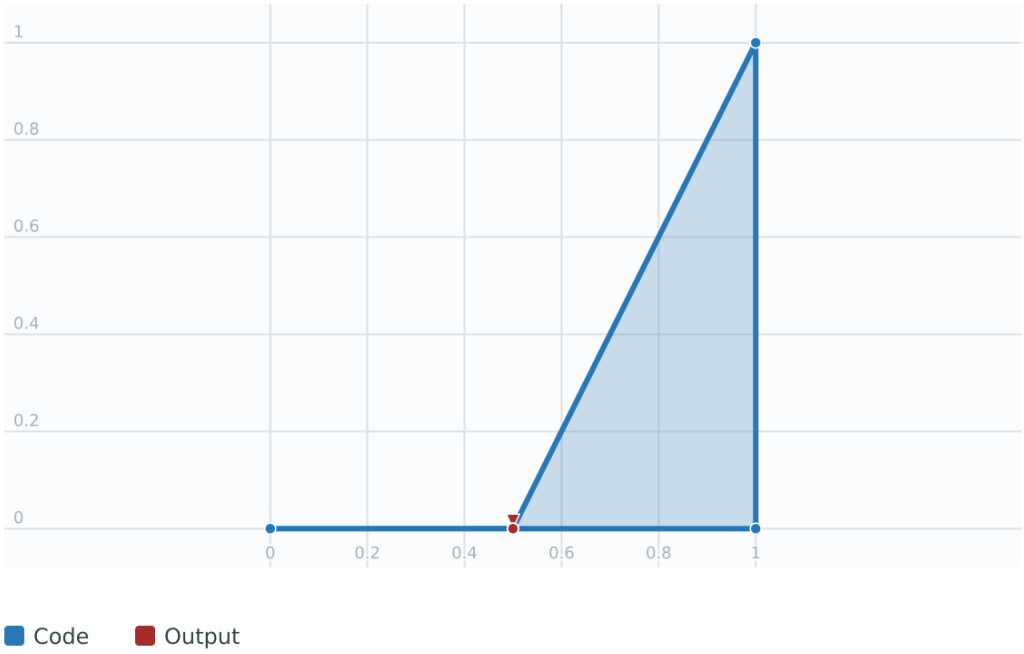
Code

```
SELECT ST_MinimumClearanceLine('POLYGON ((0 0,1 0,1 1,0.5 3.2e-4,0 0))');
```

Output

```
LINESTRING(0.5 0.00032,0.5 0)
```

Figure



Siehe auch

ST_MinimumClearance

7.12.22 ST_Perimeter

ST_Perimeter — Gibt die Länge der Begrenzung einer polygonalen Geometrie oder Geografie zurück.

Synopsis

float **ST_Perimeter**(geometry g1);
float **ST_Perimeter**(geography geog, boolean use_spheroid = true);

Beschreibung

Gibt für die geometrischen/geographischen Datentypen ST_Surface, ST_MultiSurface (Polygon, MultiPolygon) den Umfang in 2D zurück. Bei einer nicht flächigen Geometrie wird 0 zurückgegeben. Für eine lineare Geometrie können Sie ST_Length verwenden. Beim geometrischen Datentyp sind die Einheiten der Umfangsmessung durch das Koordinatenreferenzsystem der Geometrie festgelegt.

For geography types, the calculations are performed using the inverse geodesic problem, where perimeter units are in meters. The spheroid is specified by the SRID; when no SRID is provided, WGS84 is used. If `use_spheroid = false`, then calculations will approximate a sphere instead of a spheroid.

Zur Zeit ein Synonym für ST_Perimeter2D; dies kann sich allerdings ändern, wenn höhere Dimensionen unterstützt werden.

Diese Methode implementiert die OGC Simple Features Implementation Specification for SQL 1.1. s2.1.5.1

Diese Methode setzt die SQL/MM-Spezifikation um. SQL-MM 3: 8.1.3, 9.5.4

Verfügbarkeit: Mit 2.0.0 wurde die Unterstützung für geograpischen Koordinaten eingeführt

Beispiele

Geometry.

Den Umfang eines Polygons und eines Mehrfachpolygons in Fuß ausgeben. Beachten Sie bitte, dass die Einheit Fuß ist, da EPSG:2249 "Massachusetts State Plane Feet" ist

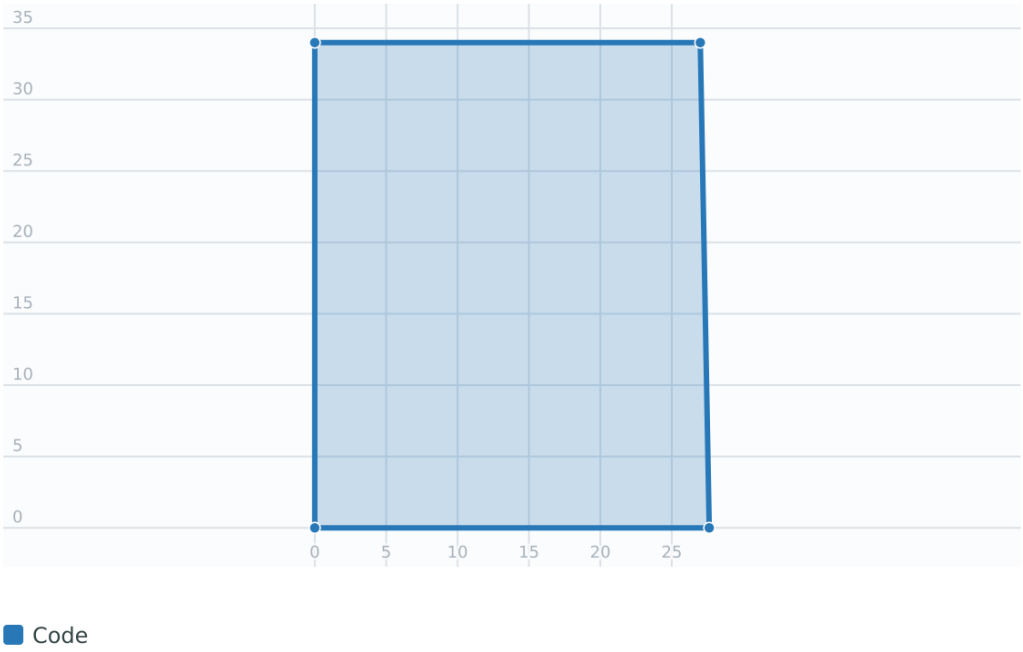
Code

```
SELECT ST_Perimeter(ST_GeomFromText(
'POLYGON((743238 2967416,743238 2967450,743265 2967450,
743265.625 2967416,743238 2967416))',
2249));
```

Output

```
122.63074400009504
```

Figure



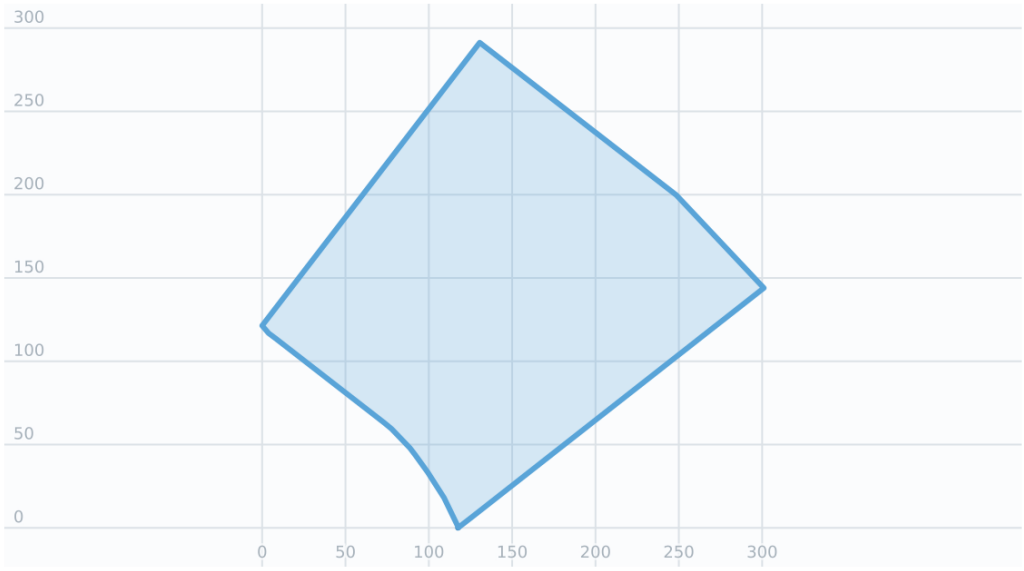
Code

```
SELECT ST_Perimeter(ST_GeomFromText(
'MULTIPOLYGON(((763104.471273676 2949418.44119003,
763104.477769673 2949418.42538203,
763104.189609677 2949418.22343004,763104.471273676 2949418.44119003)),
((763104.471273676 2949418.44119003,763095.804579742 2949436.33850239,
763086.132105649 2949451.46730207,763078.452329651 2949462.11549407,
763075.354136904 2949466.17407812,763064.362142565 2949477.64291974,
763059.953961626 2949481.28983009,762994.637609571 2949532.04103014,
762990.568508415 2949535.06640477,762986.710889563 2949539.61421415,
763117.237897679 2949709.50493431,763235.236617789 2949617.95619822,
763287.718121842 2949562.20592617,763111.553321674 2949423.91664605,
763104.471273676 2949418.44119003)))',
2249));
```

Output

```
845.227713366825
```

Figure



Gibt den Umfang eines Polygons und eines Mehrfachpolygons in Meter und in Fuß aus. Beachten Sie, dass diese in geographischen Koordinaten (WGS 84 Länge/Breite) vorliegen.

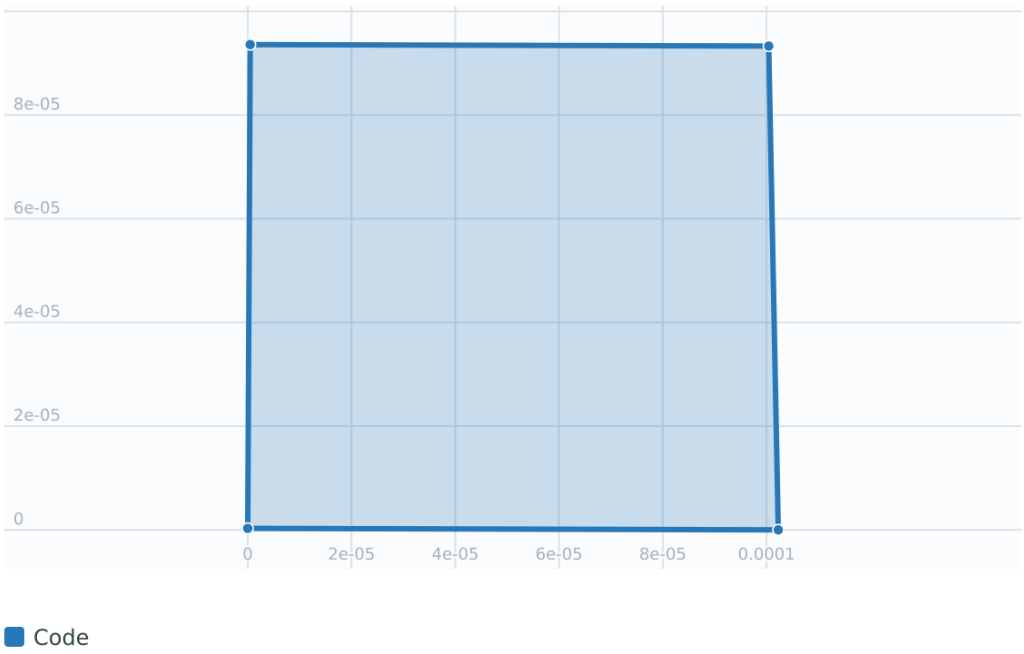
Code

```
SELECT round(ST_Perimeter(geog)::numeric, 8) As per_meters,
       round((ST_Perimeter(geog)/0.3048)::numeric, 8) As per_ft
FROM ST_GeogFromText('POLYGON((-71.1776848522251 42.3902896512902,-71.1776843766326 ↩
    42.3903829478009,
-71.1775844305465 42.3903826677917,-71.1775825927231 42.3902893647987,-71.1776848522251 ↩
    42.3902896512902))') As geog;
```

Output

```
per_meters  |    per_ft
------------+--------------
 37.37904625 | 122.63466619
```

Figure



This example uses a MultiPolygon.

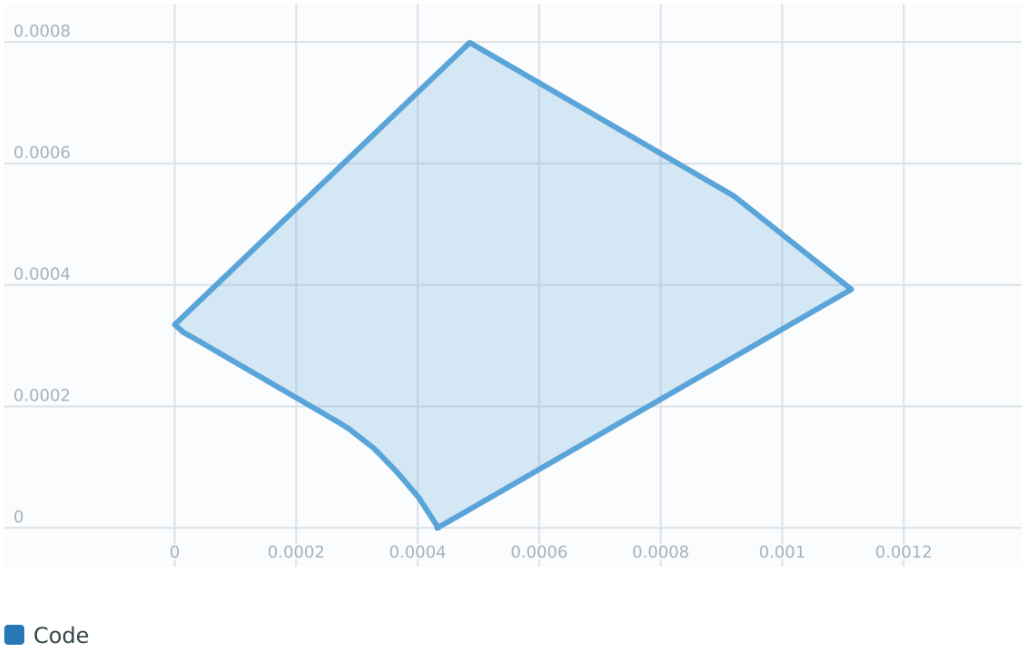
Code

```
SELECT round(ST_Perimeter(geog)::numeric, 8) As per_meters,
       round(ST_Perimeter(geog, false)::numeric, 8) As per_sphere_meters,
       round((ST_Perimeter(geog)/0.3048)::numeric, 8) As per_ft
FROM ST_GeogFromText('MULTIPOLYGON(((-71.1044543107478 42.340674480411,-71.1044542869917 ↩
    42.3406744369506,
-71.1044553562977 42.340673886454,-71.1044543107478 42.340674480411)),
((-71.1044543107478 42.340674480411,-71.1044860600303 42.3407237015564,-71.1045215770124 ↩
    42.3407653385914,
-71.1045498002983 42.3407946553165,-71.1045611902745 42.3408058316308,-71.1046016507427 ↩
    42.340837442371,
-71.104617893173 42.3408475056957,-71.1048586153981 42.3409875993595,-71.1048736143677 ↩
    42.3409959528211,
-71.1048878050242 42.3410084812078,-71.1044020965803 42.3414730072048,
-71.1039672113619 42.3412202916693,-71.1037740497748 42.3410666421308,
-71.1044280218456 42.3406894151355,-71.1044543107478 42.340674480411)))') As geog;
```

Output

```
per_meters   | per_sphere_meters |    per_ft
-------------+-------------------+-------------
 257.63428368 |       257.41231145 | 845.25683623
```

Figure



Siehe auch

ST_GeogFromText, ST_GeomFromText, ST_Length

7.12.23 ST_Perimeter2D

ST_Perimeter2D — Gibt den 2D-Umfang einer polygonalen Geometrie zurück. Alias für `ST_Perimeter`.

Synopsis

float **ST_Perimeter2D**(geometry geomA);

Beschreibung

Gibt den 2-dimensionalen Umfang eines Polygons oder eines Mehrfachpolygons zurück.



Anmerkung
Zurzeit ein Alias für ST_Perimeter. In zukünftigen Versionen dürfte ST_Perimeter den Umfang in der höchsten Dimension einer Geometrie zurückgeben. Dies befindet sich jedoch noch im Aufbau.

Siehe auch

ST_Perimeter

7.12.24 ST_3DPerimeter

ST_3DPerimeter — Gibt den geometrischen Schwerpunkt einer Geometrie zurück.

Synopsis

float **ST_3DPerimeter**(geometry geomA);

Beschreibung

Gibt den 3-dimensionalen Umfang eines Polygons oder eines Mehrfachpolygons zurück. Wenn es sich um eine 2-dimensionale Geometrie handelt wird der 2-dimensionale Umfang zurückgegeben.

Diese Funktion unterstützt 3d und lässt den Z-Index nicht fallen.

Diese Methode setzt die SQL/MM-Spezifikation um. SQL-MM ISO/IEC 13249-3: 8.1, 10.5

Änderung: 2.0.0 In Vorgängerversionen als ST_Perimeter3D bezeichet.

Beispiele

Umfang eines leicht erhöhten Polygons in "Massachusetts state plane feet"

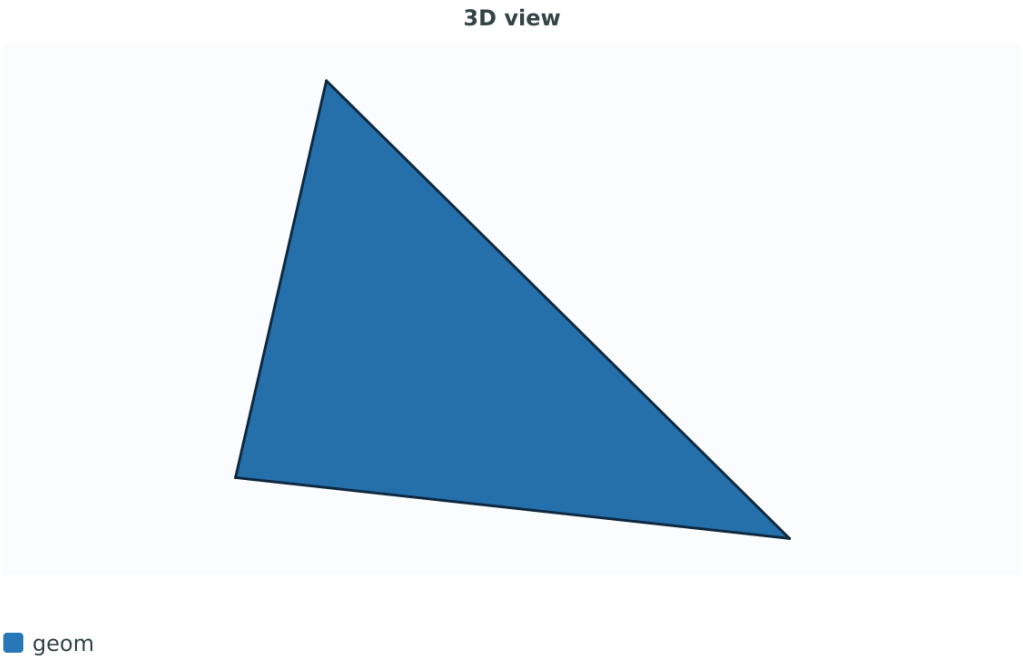
Code

```
SELECT ST_3DPerimeter(geom), ST_Perimeter2D(geom), ST_Perimeter(geom) FROM
            (SELECT 'SRID=2249;POLYGON((743238 2967416 2,743238 2967450 1,
743265.625 2967416 1,743238 2967416 2))'::geometry As geom) As foo;
```

Output

```
ST_3DPerimeter  |  st_perimeter2d  |   st_perimeter
------------------+------------------+------------------
105.46579359767418 | 105.43299727218763 | 105.43299727218763
```

Figure



Siehe auch

ST_GeomFromEWKT, ST_Perimeter, ST_Perimeter2D

7.12.25 ST_ShortestLine

ST_ShortestLine — Gibt die 2-dimenionale kürzeste Strecke zwischen zwei Geometrien als Linie zurück

Synopsis

geometry **ST_ShortestLine**(geometry geom1, geometry geom2);
geography **ST_ShortestLine**(geography geom1, geography geom2, boolean use_spheroid = true);

Beschreibung

Gibt die 2-dimensionale kürzeste Linie zwischen zwei Geometrien zurück. Die zurückgegebene Linie beginnt in `geom1` und endet in `geom2`. Wenn `geom1` und `geom2` sich schneiden, ist das Ergebnis eine Linie mit Start und Ende am Schnittpunkt. Die Länge der Linie ist dieselbe, die ST_Distance für g1 und g2 zurückgibt.

Verbessert: 3.4.0 - Unterstützung für Geographie.

Verfügbarkeit: 1.5.0

Beispiele

Shortest line between a Point and a LineString.

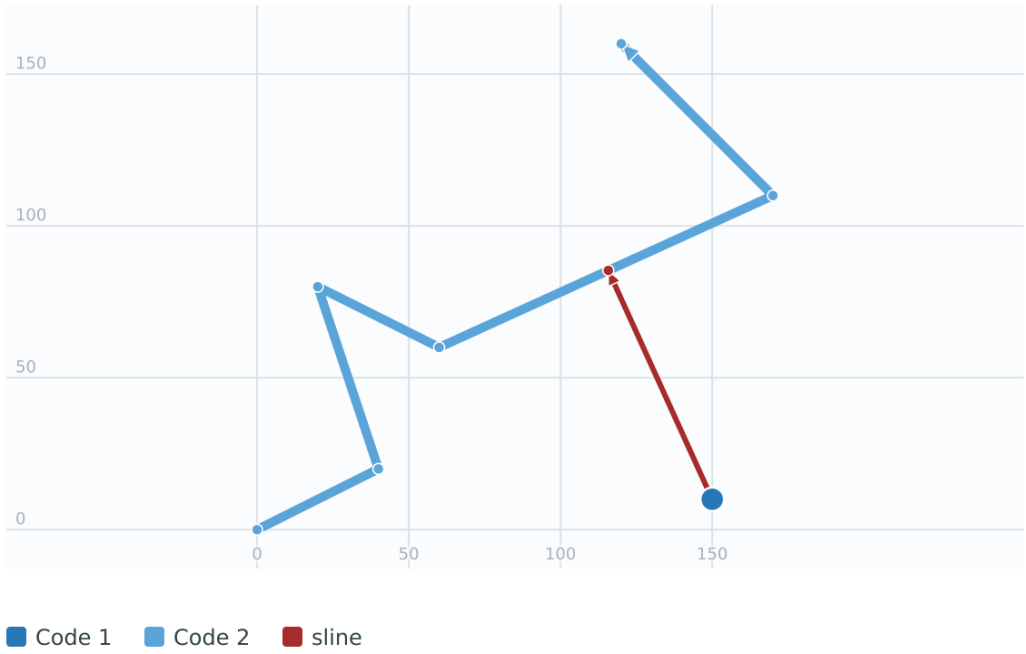
Code

```
SELECT ST_ShortestLine(
        'POINT (160 40)',
        'LINESTRING (10 30,50 50,30 110,70 90,180 140,130 190)') As sline;
```

Output

```
LINESTRING(160 40,125.75342465753425 115.34246575342466)
```

Figure



Shortest line between polygons.

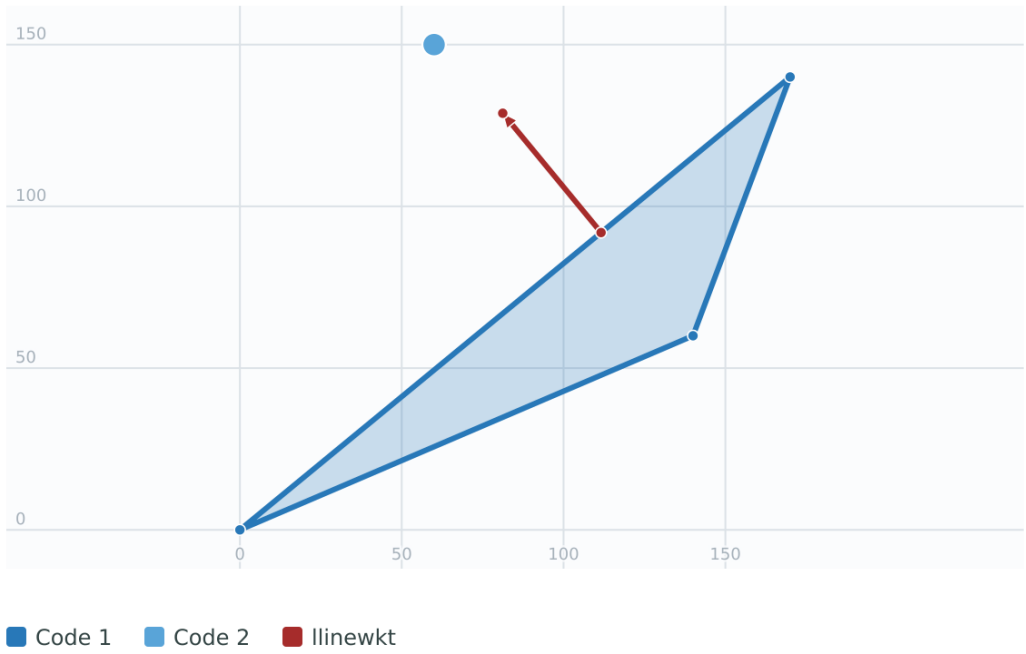
Code

```
SELECT ST_ShortestLine(
         'POLYGON ((190 150,20 10,160 70,190 150))',
         ST_Buffer('POINT(80 160)', 30)
                 ) AS llinewkt;
```

Output

```
LINESTRING(131.59149149528952 101.89887534906195,101.21320343559643 138.78679656440357)
```

Figure



Siehe auch

ST_ClosestPoint, ST_Distance, ST_LongestLine, ST_MaxDistance

7.12.26 ST_3DShortestLine

ST_3DShortestLine — Gibt den kürzesten 3-dimensionalen Abstand zwischen zwei geometrischen Objekten als Linie zurück

Synopsis

geometry **ST_3DShortestLine**(geometry g1, geometry g2);

Beschreibung

Gibt den kürzesten 3-dimensionalen Abstand zwischen zwei geometrischen Objekten als Linie zurück. Wenn es mehrere kürzeste Abstände gibt, dann wird nur der erste zurückgegeben, der von der Funktion gefunden wurde. Wenn sich g1 und g2 nur in einem Punkt schneiden, dann gibt die Funktion eine Linie zurück, die ihren Anfang und ihr Ende in dem Schnittpunkt hat. Wenn sich g1 und g2 in mehreren Punkten schneiden, dann gibt die Funktion eine Linie zurück, die Anfang und Ende in irgendeinem der Schnittpunkte hat. Die zurückgegebene Linie beginnt immer mit g1 und endet mit g2. Die Länge der 3D-Linie die von dieser Funktion zurückgegeben wird ist immer ident mit der von ST_3DDistance für g1 und g2 zurückgegebenen Distanz.

Verfügbarkeit: 2.0.0

Änderung: 2.2.0 - Wenn 2 geometrische Objekte in 2D übergegeben werden, wird ein 2D-Punkt zurückgegeben (anstelle wie früher 0 für ein fehlendes Z). Im Falle von 2D und 3D, wird für fehlende Z nicht länger 0 angenommen.

Diese Funktion unterstützt 3d und lässt den Z-Index nicht fallen.

Diese Funktion unterstützt polyedrische Flächen.

Beispiele

Compare the 3D shortest line to the 2D shortest line for a LineString and a Point.

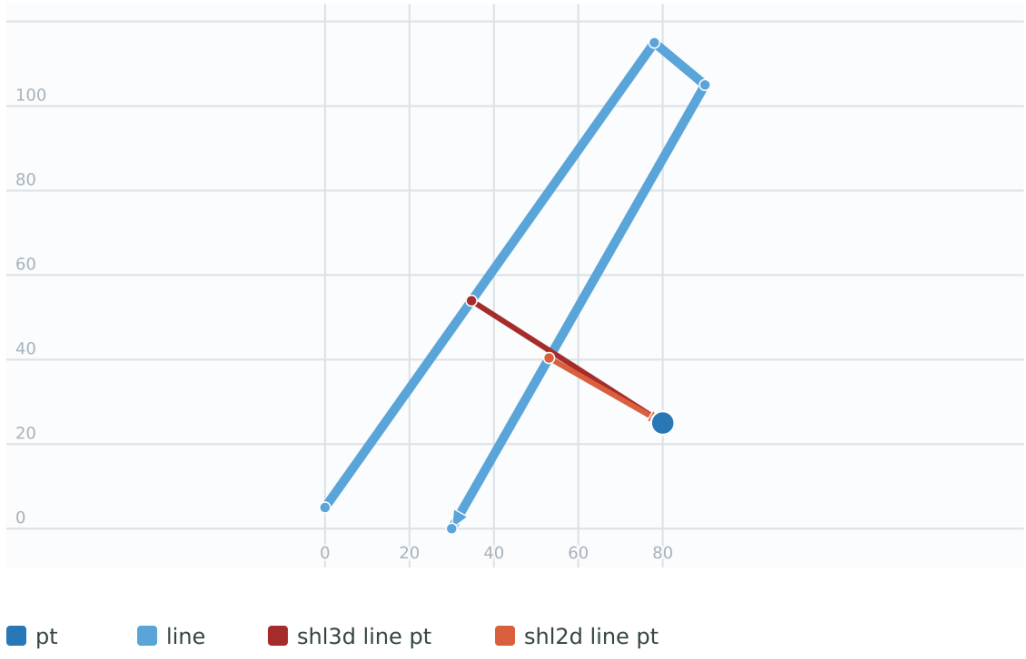
Code

```
WITH input AS (
  SELECT 'POINT(100 100 30)'::geometry AS pt,
         'LINESTRING (20 80 20,98 190 1,110 180 3,50 75 1000)'::geometry AS line
)
SELECT ST_3DShortestLine(line, pt) AS shl3d_line_pt,
       ST_ShortestLine(line, pt) AS shl2d_line_pt
FROM input;
```

Output

```
-[ RECORD 1 ]--+-----------------------------------------------------------------------------------
shl3d_line_pt  | LINESTRING(54.69937988676193 128.93502291722837 11.547586950660556,100 100 30)
shl2d_line_pt  | LINESTRING(73.07692307692307 115.38461538461539,100 100)
```

Figure



Compare the 3D shortest line to the 2D shortest line for a LineString and a MultiPoint.

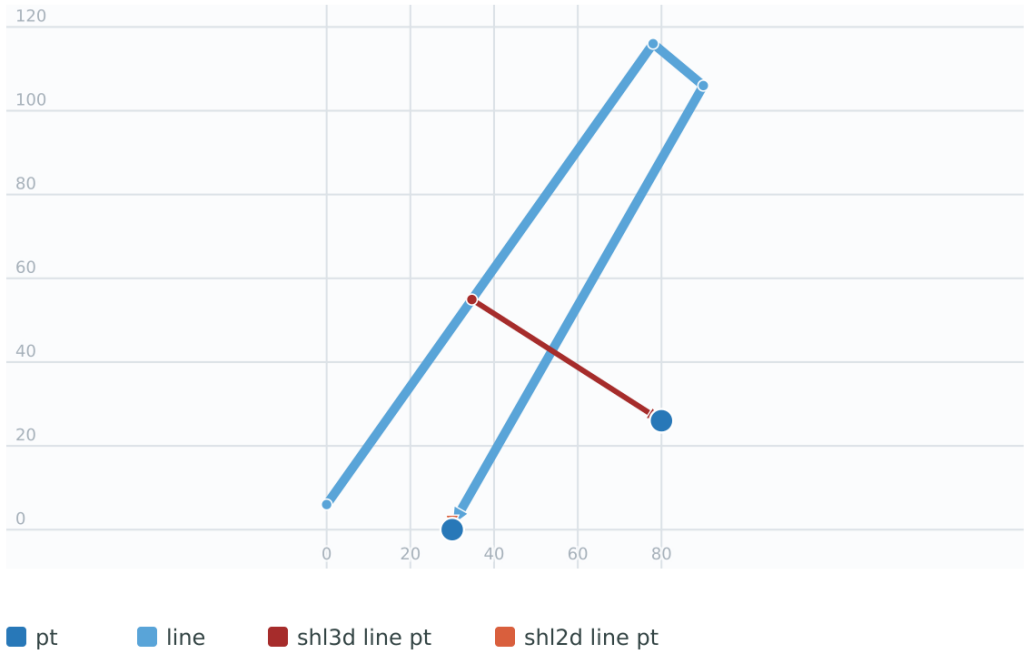
Code

```
WITH input AS (
  SELECT 'MULTIPOINT(100 100 30,50 74 1000)'::geometry AS pt,
         'LINESTRING (20 80 20,98 190 1,110 180 3,50 75 900)'::geometry AS line
)
SELECT ST_3DShortestLine(line, pt) AS shl3d_line_pt,
       ST_ShortestLine(line, pt) AS shl2d_line_pt
FROM input;
```

Output

```
-[ RECORD 1 ]--+-----------------------------------------------------------------------------
shl3d_line_pt  | LINESTRING(54.69937988676193 128.93502291722837 11.547586950660556,100 100 30)
shl2d_line_pt  | LINESTRING(50 75,50 74)
```

Figure



Compare the 3D shortest line to the 2D shortest line for a Polygon and a MultiLineString.

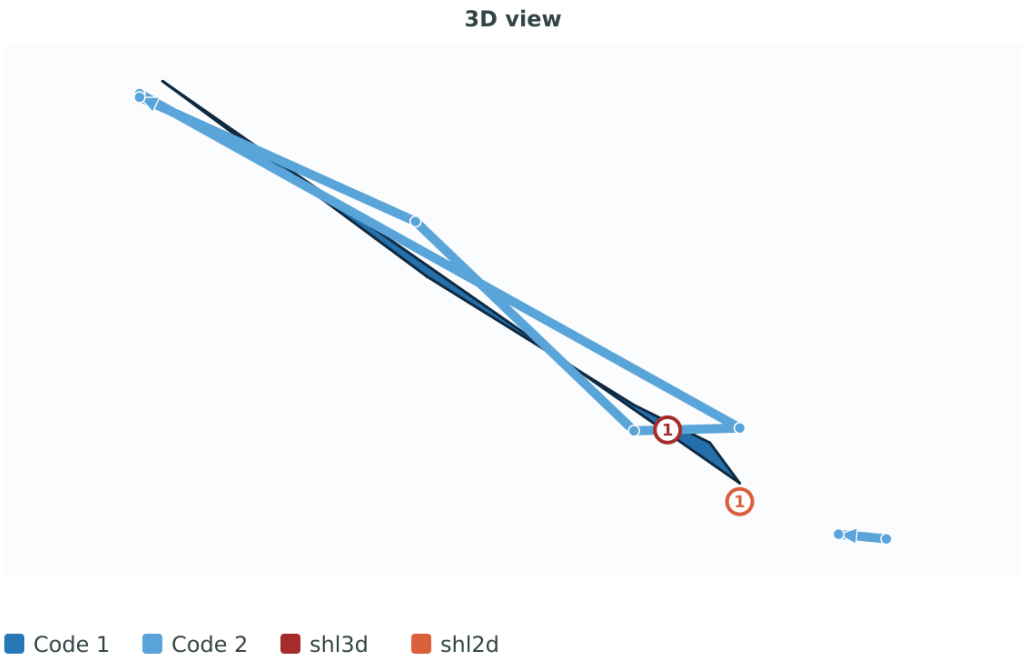
Code

```
WITH input AS (
  SELECT ST_GeomFromEWKT(
           'POLYGON((175 150 5,20 40 5,35 45 5,50 60 5,100 100 5,175 150 5))'
         ) AS poly,
         ST_GeomFromEWKT(
           'MULTILINESTRING((175 155 2,20 40 20,50 60 -2,125 100 1,175 155 1),
                            (1 10 2,5 20 1))'
         ) AS mline
)
SELECT ST_3DShortestLine(poly, mline) AS shl3d,
       ST_ShortestLine(poly, mline) AS shl2d
FROM input;
```

Output

```
-[ RECORD 1 ]--+-----------------------------------------------------------------------------------
shl3d          | LINESTRING(40.45454545454545 53.63636363636364 5,40.45454545454545 53.63636363636364 5)
shl2d          | LINESTRING(20 40,20 40)
```

Figure



Siehe auch

ST_3DClosestPoint, ST_3DDistance, ST_LongestLine, ST_ShortestLine, ST_3DMaxDistance

7.13 Overlay-Funktionen

7.13.1 ST_ClipByBox2D

ST_ClipByBox2D — Berechnet den Teil einer Geometrie, der innerhalb eines Rechtecks liegt.

Synopsis

geometry **ST_ClipByBox2D**(geometry geom, box2d box);

Beschreibung

Beschneidet eine Geometrie durch eine 2D-Box auf eine schnelle und tolerante, aber möglicherweise ungültige Weise. Topologisch ungültige Eingabegeometrien führen nicht zum Auslösen von Exceptions. Es ist nicht garantiert, dass die Ausgabegeometrie gültig ist (insbesondere können Selbstüberschneidungen für ein Polygon eingeführt werden).

Wird vom GEOS Modul ausgeführt

Verfügbarkeit: 2.2.0

Beispiele

This example relies on the implicit cast from geometry to box2d for the second parameter.

Code

```
SELECT ST_ClipByBox2D(geom, ST_MakeEnvelope(0, 0, 10, 10)) FROM mytab;
```

Siehe auch

ST_Intersection, ST_MakeBox2D, ST_MakeEnvelope

7.13.2 ST_Difference

ST_Difference — Berechnet eine Geometrie, die den Teil der Geometrie A darstellt, der die Geometrie B nicht schneidet.

Synopsis

geometry **ST_Difference**(geometry geomA, geometry geomB, float8 gridSize = -1);

Beschreibung

Gibt eine Geometrie zurück, die den Teil der Geometrie A darstellt, der die Geometrie B nicht schneidet. Dies entspricht `A - ST_Intersection(A,B)`. Wenn A vollständig in B enthalten ist, wird eine leere atomare Geometrie des entsprechenden Typs zurückgegeben.



Anmerkung
Dies ist die einzige Überlagerungsfunktion, bei der die Reihenfolge der Eingabe eine Rolle spielt. ST_Difference(A, B) gibt immer einen Teil von A zurück.

If the optional `gridSize` parameter is given (GEOS-3.9.0 or higher required), all result vertices are guaranteed to fall on a snap-rounded grid of the specified size. Note that operations performed on a grid may contain small artifacts produced during grid alignment, see ST_ReducePrecision.

Wird durch das GEOS Modul ausgeführt

Verbessert: 3.1.0 akzeptiert einen gridSize-Parameter.

Erfordert GEOS >= 3.9.0 zur Verwendung des Parameters gridSize.

Diese Methode implementiert die OGC Simple Features Implementation Specification for SQL 1.1. s2.1.1.3

Diese Methode setzt die SQL/MM-Spezifikation um. SQL-MM 3: 5.1.20

Diese Funktion unterstützt 3d und lässt den Z-Index nicht fallen. Das Ergebnis wird jedoch nur mit XY berechnet. Die Z-Werte des Ergebnisses werden kopiert, gemittelt oder interpoliert.

Beispiele

Die Differenz der 2D-Linien.

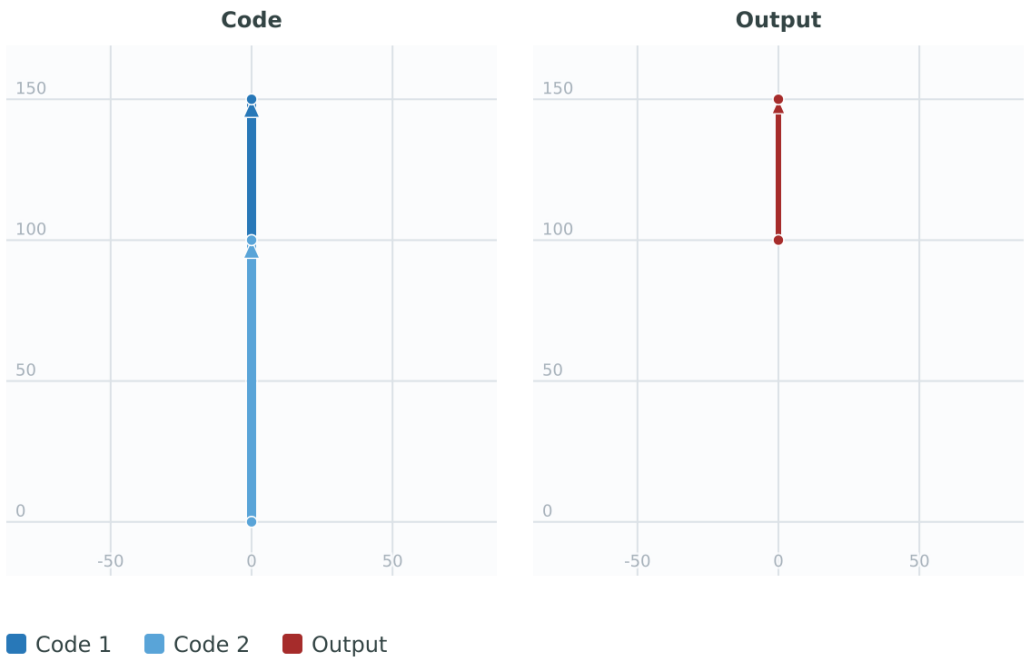
Code

```
SELECT ST_Difference('LINESTRING(50 100,50 200)'::geometry,
        'LINESTRING(50 50,50 150)'::geometry
    );
```

Output

```
LINESTRING(50 150,50 200)
```

Figure



Die Differenz der 3D-Punkte.

Code

```
SELECT ST_Difference('MULTIPOINT(-118.58 38.38 5,-118.60 38.329 6,-118.614 38.281 7)' ::  ↩
    geometry,
 'POINT(-118.614 38.281 5)' :: geometry
);
```

Output

```
MULTIPOINT(-118.6 38.329 6,-118.58 38.38 5)
```

Figure



Siehe auch

ST_SymDifference, ST_Intersection, ST_Union, ST_ReducePrecision

7.13.3 ST_Intersection

ST_Intersection — Berechnet eine Geometrie, die den gemeinsamen Teil der Geometrien A und B darstellt.

Synopsis

geometry **ST_Intersection**(geometry geomA , geometry geomB , float8 gridSize = -1);
geography **ST_Intersection**(geography geogA , geography geogB);

Beschreibung

Gibt eine Geometrie zurück, die die Punktmengenüberschneidung zweier Geometrien darstellt. Mit anderen Worten: der Teil von Geometrie A und Geometrie B, der von den beiden Geometrien gemeinsam genutzt wird.

Wenn die Geometrien keine gemeinsamen Punkte haben (d. h. disjunkt sind), wird eine leere atomare Geometrie des entsprechenden Typs zurückgegeben.

If the optional `gridSize` parameter is given (GEOS-3.9.0 or higher required), all result vertices are guaranteed to fall on a snap-rounded grid of the specified size. Note that operations performed on a grid may contain small artifacts produced during grid alignment, see ST_ReducePrecision.

ST_Intersection in Verbindung mit ST_Intersects ist nützlich, um Geometrien zu beschneiden, z. B. in Bounding Box-, Puffer- oder Regionsabfragen, bei denen Sie nur den Teil einer Geometrie benötigen, der sich innerhalb eines Landes oder einer Region von Interesse befindet.



Anmerkung
Spatial predicates such as ST_Intersects and overlay functions such as `ST_Intersection` are computed by different algorithms. For very close or nearly coincident coordinates, floating-point precision and geometric robustness may make a predicate return true while the computed intersection is empty, or place an intersection point slightly away from the input line. For repeatable overlay results, snap inputs to a suitable grid first, for example with ST_ReducePrecision, or use the `gridSize` parameter when all input vertices already lie on the grid.



Anmerkung

Für die Geografie ist dies eine dünne Hülle um die Geometrieimplementierung. Zunächst wird der beste SRID ermittelt, der in die Bounding Box der beiden geografischen Objekte passt (wenn die geografischen Objekte innerhalb einer halben UTM-Zone liegen, aber nicht in der gleichen UTM-Zone, wird eines davon ausgewählt) (wobei UTM oder der Nord-/Südpol von Lambert Azimuthal Equal Area (LAEA) bevorzugt werden und im schlimmsten Fall auf Mercator zurückgegriffen wird), und dann wird die Schnittmenge in diesem am besten passenden planaren Raumbezug ermittelt und zurück in WGS84-Geografie transformiert.



Warnung
Mit dieser Funktion werden die M-Koordinatenwerte, falls vorhanden, gelöscht.



Warnung
If working with 3D geometries, you may want to use SFCGAL based CG_3DIntersection which does a proper 3D intersection for 3D geometries. Although this function works with Z-coordinate, it does an averaging of Z-Coordinate.

Wird durch das GEOS Modul ausgeführt

Verbessert: 3.1.0 akzeptiert einen gridSize Parameter

Erfordert GEOS >= 3.9.0 zur Verwendung des Parameters gridSize

Geändert: 3.0.0 ist nicht von SFCGAL abhängig.

Verfügbarkeit: Mit Version 1.5 wurde die Unterstützung für den Datentyp Geographie eingeführt.


Diese Methode implementiert die OGC Simple Features Implementation Specification for SQL 1.1. s2.1.1.3


Diese Methode setzt die SQL/MM-Spezifikation um. SQL-MM 3: 5.1.18


Diese Funktion unterstützt 3d und lässt den Z-Index nicht fallen. Das Ergebnis wird jedoch nur mit XY berechnet. Die Z-Werte des Ergebnisses werden kopiert, gemittelt oder interpoliert.

Beispiele

Code

```
SELECT ST_Intersection('POINT(0 0)'::geometry, 'LINESTRING ( 2 0,0 2 )'::geometry);
```

Output

```
PUNKT LEER
```

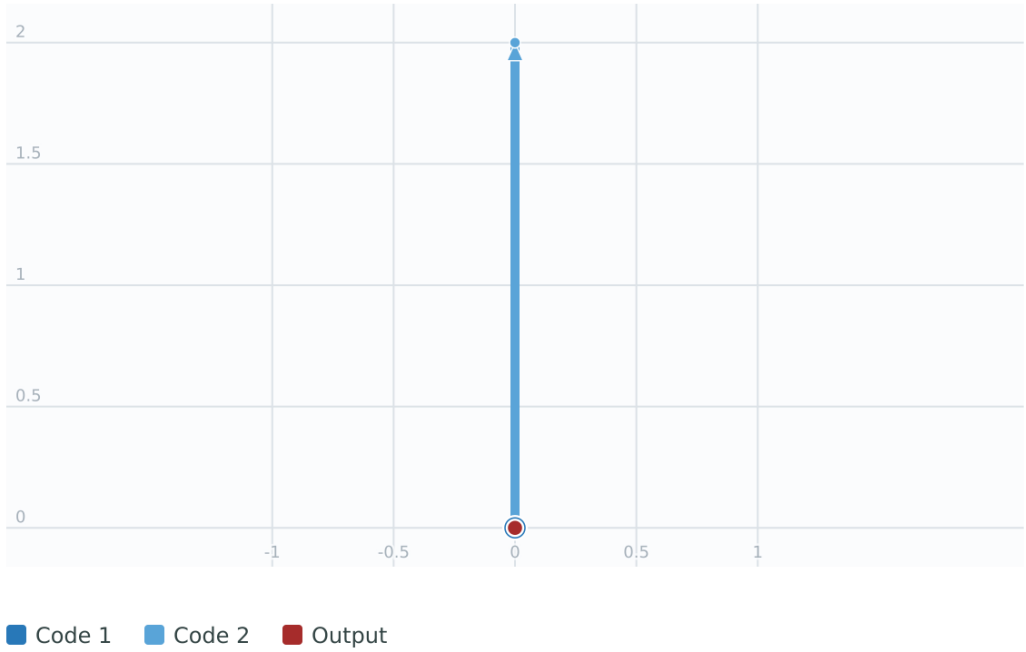
Code

```
SELECT ST_Intersection('POINT(0 0)'::geometry, 'LINESTRING ( 0 0,0 2 )'::geometry);
```

Output

```
POINT(0 0)
```

Figure



Schneiden Sie alle Linien (Trails) nach Land. Hier nehmen wir an, dass die Ländergeometrien POLYGON oder MULTIPOLYGON sind. HINWEIS: Wir behalten nur Schnittpunkte, die zu einem LINESTRING oder MULTILINESTRING führen, da wir uns nicht um Wege kümmern, die nur einen Punkt teilen. Der Dump wird benötigt, um eine Geometriesammlung in einzelne MULT*-Teile zu zerlegen. Das folgende Beispiel ist recht allgemein gehalten und funktioniert auch für Polygone usw., indem einfach die Where-Klausel geändert wird.

Code

```
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;
```

Für Polygone, z. B. Polygon-Landmarken, können Sie auch den manchmal schnelleren Hack verwenden, dass das Puffern von allem um 0,0 außer einem Polygon zu einer leeren Geometriesammlung führt. (Eine Geometriesammlung mit Polygonen, Linien und Punkten, die mit 0,0 gepuffert wird, würde also nur die Polygone übrig lassen und die Sammlungshülle auflösen).

Code

```
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));
```

Note this is not a true 3D intersection. It uses 2.5D geometries (geometries with Z ordinates, but where calculations are performed in 2D). The same input is compared with CG_3DIntersection, which performs a true 3D intersection.

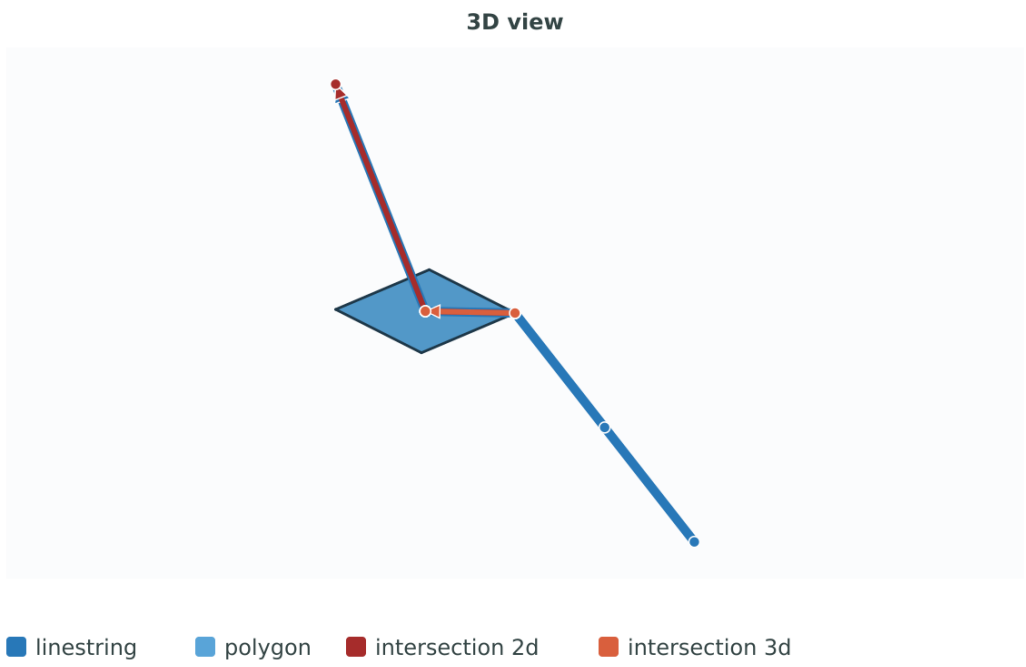
Code

```
WITH data AS (
  SELECT 'LINESTRING Z (2 2 6,1.5 1.5 7,1 1 8,0.5 0.5 8,0 0 10)'::geometry AS ↩
    input_linestring,
         'POLYGON Z ((0 0 8,0 1 8,1 1 8,1 0 8,0 0 8))'::geometry AS input_polygon
)
SELECT input_linestring AS input_linestring,
       input_polygon AS input_polygon,
       ST_Intersection(input_linestring, input_polygon) AS intersection_2d,
       CG_3DIntersection(input_linestring, input_polygon) AS intersection_3d
FROM data;
```

Output

```
LINESTRING Z (2 2 6,1.5 1.5 7,1 1 8,0.5 0.5 8,0 0 10) | POLYGON Z ((0 0 8,0 1 8,1 1 8,1 0 8,0 0 8)) | LINESTRING Z (1 1 ↩
     8,0.5 0.5 8,0 0 10) | LINESTRING Z (1 1 8,0.5 0.5 8)
```

Figure



Siehe auch

CG_3DIntersection, ST_Difference, ST_Union, ST_ClipByBox2D, ST_Dimension, ST_Dump, ST_Force2D, ST_SymDifference, ST_Intersects, ST_Multi, ST_ReducePrecision

7.13.4 ST_MemUnion

ST_MemUnion — Aggregatfunktion, die Geometrien auf eine speichereffiziente, aber langsamere Weise zusammenfasst

Synopsis

geometry **ST_MemUnion**(geometry set geomfield);

Beschreibung

Eine Aggregatfunktion, die die Eingabegeometrien vereinigt und zu einer überschneidungsfreien Ergebnisgeometrie zusammenführt. Die Ausgabe kann eine einzelne Geometrie, eine MultiGeometry oder eine Geometriesammlung sein.



Anmerkung
Ergibt das gleiche Ergebnis wie ST_Union, benötigt aber weniger Speicher und mehr Prozessorzeit. Bei dieser Aggregatfunktion werden die Geometrien inkrementell vereinigt, im Gegensatz zum Aggregat ST_Union, das zunächst ein Array akkumuliert und dann den Inhalt mit einem schnellen Algorithmus vereinigt.

 Diese Funktion unterstützt 3d und lässt den Z-Index nicht fallen. Das Ergebnis wird jedoch nur mit XY berechnet. Die Z-Werte des Ergebnisses werden kopiert, gemittelt oder interpoliert.

Beispiele

Code

```
SELECT id,
       ST_MemUnion(geom) as singlegeom
FROM sometable f
GROUP BY id;
```

Siehe auch

ST_Union

7.13.5 ST_Node

ST_Node — Knoten eine Sammlung von Linien.

Synopsis

geometry **ST_Node**(geometry geom);

Beschreibung

Gibt einen (Multi)LineString zurück, der die vollständig nodierte Version einer Sammlung von Linestrings darstellt. Bei der Kodierung bleiben alle Eingabeknoten erhalten, und es werden so wenig neue Knoten wie möglich hinzugefügt. Das resultierende Linienwerk wird aufgelöst (doppelte Linien werden entfernt).

Dies ist ein guter Weg, um ein vollständig codiertes Linienwerk zu erstellen, das als Input für ST_Polygonize verwendet werden kann.

ST_UnaryUnion kann auch zum Verknüpfen und Auflösen von Linien verwendet werden. Es bietet die Möglichkeit, eine gridSize anzugeben, die eine einfachere und robustere Ausgabe ermöglichen kann. Siehe auch ST_Union für eine aggregierte Variante.

Diese Funktion unterstützt 3d und lässt den Z-Index nicht fallen.

Wird vom GEOS Modul ausgeführt

Verfügbarkeit: 2.0.0

Geändert: 2.4.0 verwendet diese Funktion intern GEOSNode anstelle von GEOSUnaryUnion. Dies kann dazu führen, dass die resultierenden Linestrings eine andere Reihenfolge und Richtung haben als in PostGIS < 2.4.

Beispiele

Noding a 3D LineString which self-intersects.

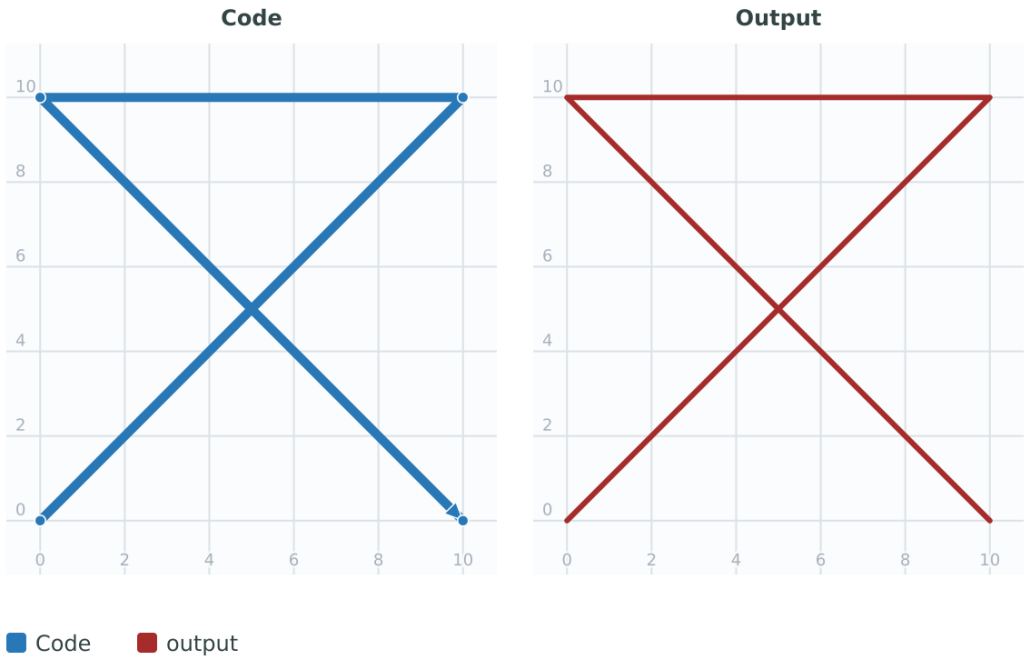
Code

```
SELECT ST_Node('LINESTRINGZ(0 0 0, 10 10 10, 0 10 5, 10 0 3)'::geometry) As  output;
```

Output

```
MULTILINESTRING Z ((0 0 0,5 5 4.5),(5 5 4.5,10 10 10,0 10 5,5 5 4.5),(5 5 4.5,10 0 3))
```

Figure



Kodierung zweier LineStrings, die ein gemeinsames Liniengerüst haben. Beachten Sie, dass das Liniengerüst des Ergebnisses aufgelöst wird.

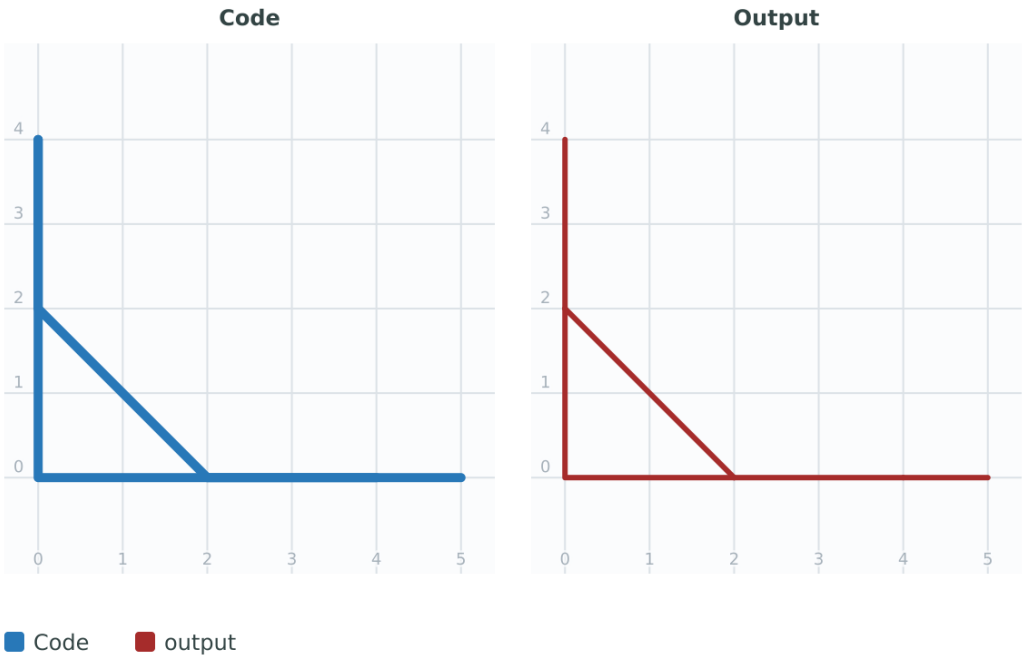
Code

```
SELECT ST_Node('MULTILINESTRING ((2 5,2 1,7 1),(6 1,4 1,2 3,2 5))'::geometry) As  output;
```

Output

```
MULTILINESTRING((2 5,2 3),(2 3,2 1,4 1),(4 1,2 3),(4 1,6 1),(6 1,7 1))
```

Figure



Siehe auch

ST_UnaryUnion, ST_AsBinary

7.13.6 ST_Split

ST_Split — Gibt eine Sammlung von Geometrien zurück, die durch Aufteilung einer Geometrie durch eine andere Geometrie entstanden sind.

Synopsis

geometry **ST_Split**(geometry input, geometry blade);

Beschreibung

Die Funktion unterstützt die Aufteilung eines LineStrings durch eine (Multi)Point-, (Multi)LineString- oder (Multi)Polygon-Grenze, oder eines (Multi)Polygons durch einen LineString. Wenn ein (Multi)Polygon als Klinge verwendet wird, werden seine linearen Komponenten (die Begrenzung) für die Aufteilung der Eingabe verwendet. Die Ergebnisgeometrie ist immer eine Sammlung.

Diese Funktion ist in gewisser Weise das Gegenteil von ST_Union. Die Anwendung von ST_Union auf die zurückgegebene Sammlung sollte theoretisch die ursprüngliche Geometrie ergeben (obwohl dies aufgrund numerischer Rundungen nicht unbedingt der Fall ist).



Anmerkung
Wenn sich Eingabe und Schaufel aufgrund von Problemen mit der numerischen Präzision nicht überschneiden, wird die Eingabe möglicherweise nicht wie erwartet geteilt. Um dies zu vermeiden, kann es notwendig sein, die Eingabe zuerst an der Klinge zu fangen, indem Sie ST_Snap mit einer kleinen Toleranz verwenden.

Verfügbarkeit: 2.0.0 erfordert GEOS

Verbessert: In Version 2.2.0 wurde die Unterstützung für die Aufteilung einer Linie durch eine Mehrlinien-, eine Mehrpunkt- oder eine (Mehr-)Polygonbegrenzung eingeführt.

Verbessert: In Version 2.5.0 wurde die Unterstützung für die Aufteilung eines Polygons durch eine Mehrlinie eingeführt.

Beispiele

Split a circular polygon by a diagonal line.

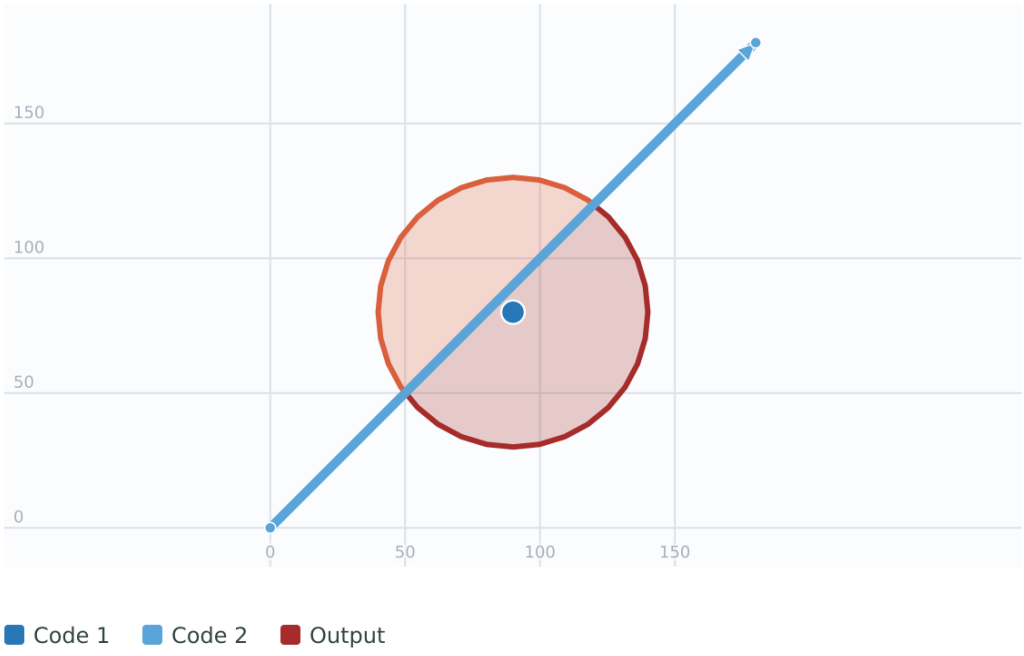
Code

```
SELECT
          ST_Split(
            ST_Buffer('POINT(100 90)'::geometry, 50),
            'LINESTRING(10 10,190 190)'::geometry
          );
```

Output

```
GEOMETRYCOLLECTION(POLYGON((150 90,149 80.2,146.2 70.9,141.6 62.2,135.4 54.6,127.8 48.4,119.1 43.8,109.8 41,100 40,90.2 ↩
      41,80.9 43.8,72.2 48.4,64.6 54.6,60.1 60.1,129.9 129.9,135.4 125.4,141.6 117.8,146.2 109.1,149 99.8,150 90)),  ↩
     POLYGON((60.1 60.1,58.4 62.2,53.8 70.9,51 80.2,50 90,51 99.8,53.8 109.1,58.4 117.8,64.6 125.4,72.2 131.6,80.9  ↩
     136.2,90.2 139,100 140,109.8 139,119.1 136.2,127.8 131.6,129.9 129.9,60.1 60.1)))
```

Figure



Teilt einen MultiLineString durch einen Punkt, wobei der Punkt genau auf beiden LineString-Elementen liegt.

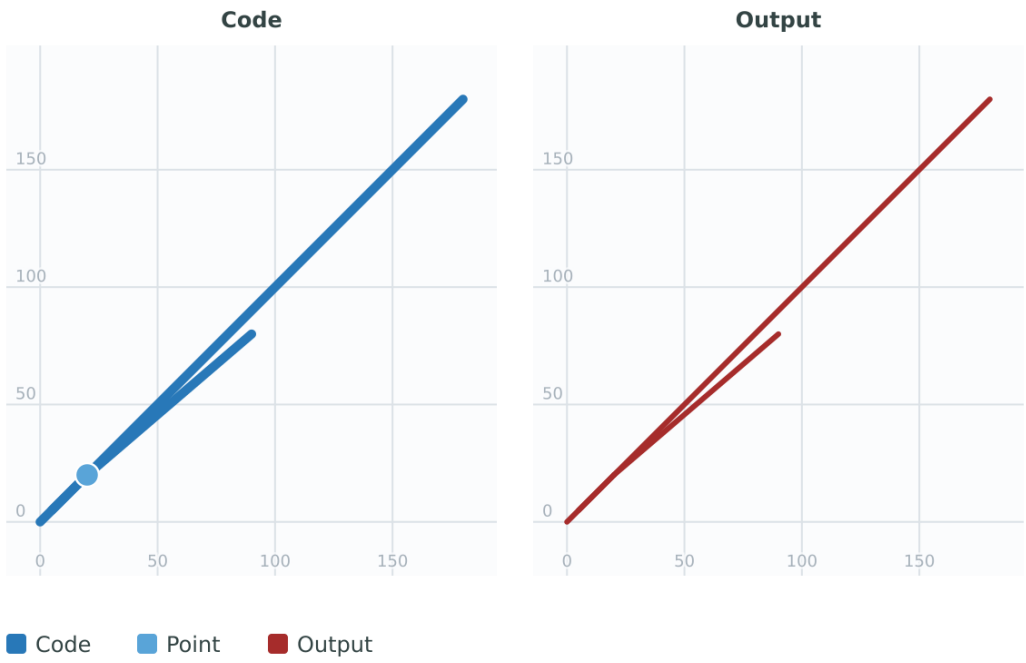
Code

```
SELECT ST_Split('MULTILINESTRING((10 10,190 190),(15 15,30 30,100 90))',
    ST_Point(30, 30));
```

Output

```
GEOMETRYCOLLECTION(
    LINESTRING(10 10,30 30),
    LINESTRING(30 30,190 190),
    LINESTRING(15 15,30 30),
    LINESTRING(30 30,100 90)
)
```

Figure

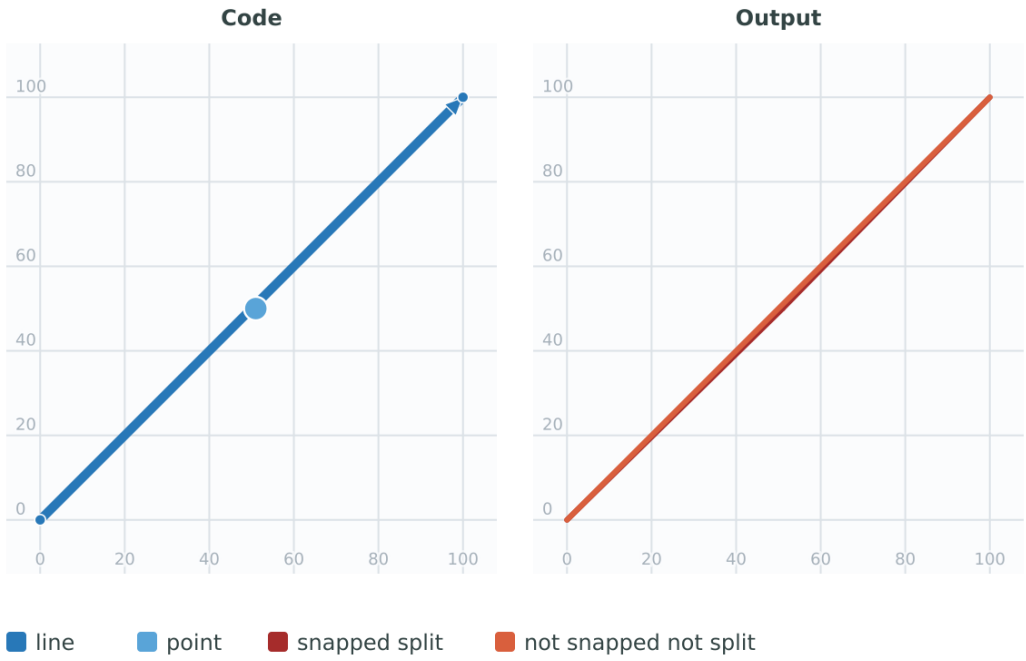


Aufteilung eines LineString durch einen Punkt, wobei der Punkt nicht genau auf der Linie liegt. Zeigt die Verwendung von ST_Snap, um die Linie an dem Punkt zu fangen, damit sie geteilt werden kann.

Code

```
WITH data AS (SELECT
  'LINESTRING(0 0,100 100)'::geometry AS line,
  'POINT(51 50)':: geometry AS point
)
SELECT ST_Split(ST_Snap(line, point, 1), point) AS snapped_split,
       ST_Split(line, point) AS not_snapped_not_split
       FROM data;
```

Figure



Siehe auch

ST_Snap, ST_AsBinary

7.13.7 ST_Subdivide

ST_Subdivide — Berechnet eine geradlinige Unterteilung einer Geometrie.

Synopsis

setof geometry **ST_Subdivide**(geometry geom, integer max_vertices=256, float8 gridSize = -1);

Beschreibung

Gibt eine Menge von Geometrien zurück, die das Ergebnis der Unterteilung von `geom` in Teile mit geradlinigen Linien sind, wobei jeder Teil nicht mehr als `max_vertices` enthält.

`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 snap-rounded grid of the specified size. Note that operations performed on a grid may contain small artifacts produced during grid alignment, see ST_ReducePrecision.

Punkt-in-Polygon und andere räumliche Operationen sind bei indizierten, unterteilten Datensätzen normalerweise schneller. Da die Boundingboxen für die Teile in der Regel einen kleineren Bereich abdecken als die ursprüngliche Geometriebox, führen Indexabfragen zu weniger "Treffer"-Fällen. Die "Treffer"-Fälle sind schneller, weil die räumlichen Operationen, die von der Indexnachprüfung ausgeführt werden, weniger Punkte verarbeiten.



Anmerkung
When casting a subdivided geometry to geography, the resulting geography may differ from the original. Subdivision adds vertices in planar (geometry) space. If vertices are inserted along the boundary, they will alter the geographical representation, where edges are interpreted as geodesic segments. To minimize distortion, first densify the geography using ST_Segmentize to add geodesic vertices, then cast to geometry before subdivision.



Anmerkung
Dies ist eine set-returning function (SRF), die eine Reihe von Zeilen mit einzelnen Geometriewerten zurückgibt. Sie kann in einer SELECT-Liste oder einer FROM-Klausel verwendet werden, um eine Ergebnismenge mit einem Datensatz für jede Ergebnisgeometrie zu erzeugen.

Wird vom GEOS Modul ausgeführt

Verfügbarkeit: 2.2.0

Verbessert: 2.5.0 verwendet vorhandene Punkte bei der Polygonaufteilung wieder, die Anzahl der Scheitelpunkte wurde von 8 auf 5 gesenkt.

Verbessert: 3.1.0 akzeptiert einen gridSize-Parameter.

Erfordert GEOS >= 3.9.0 zur Verwendung des Parameters gridSize

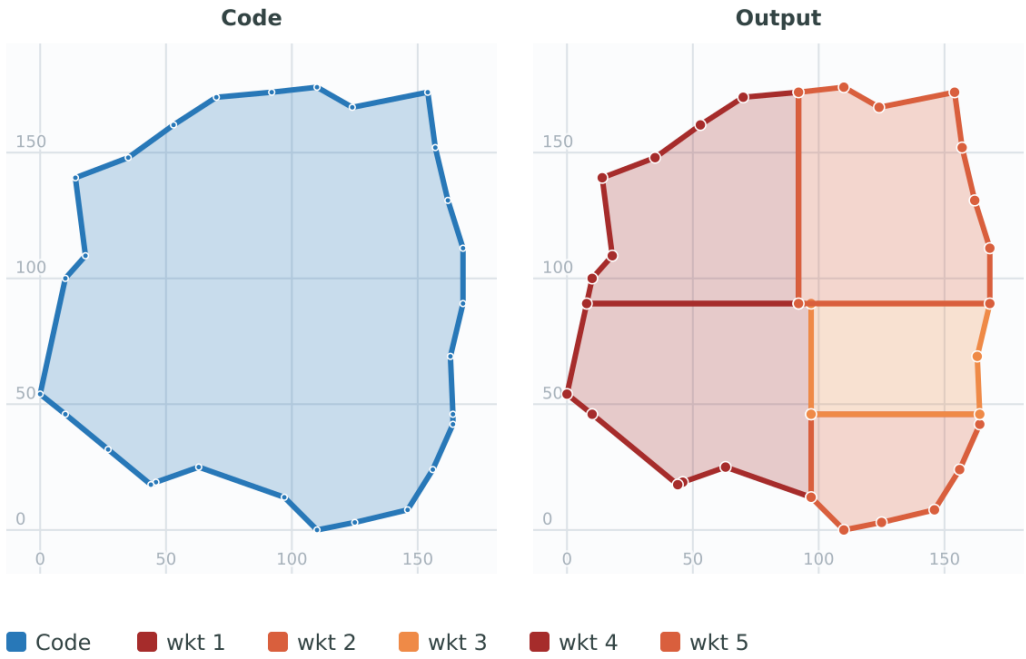
Beispiele

Beispiel: Unterteilen Sie ein Polygon in Teile mit nicht mehr als 10 Scheitelpunkten, und weisen Sie jedem Teil eine eindeutige ID zu.

Code

```
SELECT row_number() OVER() As rn,geom As wkt
    FROM (SELECT ST_Subdivide('POLYGON((132 10,119 23,85 35,68 29,66 28,49 42,32 56,22 64,32 ↩
     110,40 119,36 150,
        57 158,75 171,92 182,114 184,132 186,146 178,176 184,179 162,184 141,190 122,
        190 100,185 79,186 56,186 52,178 34,168 18,147 13,132 10))'::geometry, 10))  AS f( ↩
    geom);
```

Figure



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. Densification minimizes the impact of changes to the geography representation of a geometry when subdividing.

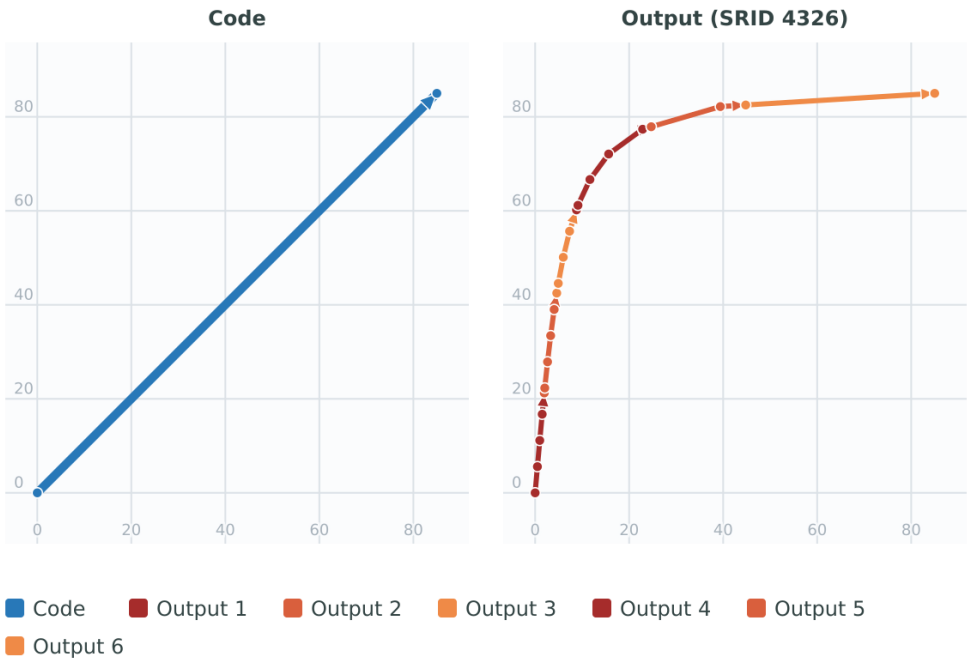
Code

```
SELECT ST_Subdivide(ST_Segmentize('LINESTRING(0 0,85 85)'::geography,
                          1200000)::geometry, 8);
```

Output

```
LINESTRING(0 0,0.487578359029357 5.57659056746196,0.984542144675897 11.1527721155093,1.50101059639722 ↩
    16.7281035483571,1.94532113630331 21.25)
LINESTRING(1.94532113630331 21.25,2.04869538062779 22.3020741387339,2.64204641967673 27.8740533545155,3.29994062412787 ↩
    33.443216802941,4.04836719489742 39.0084282520239,4.59890468420694 42.5)
LINESTRING(4.59890468420694 42.5,4.92498503922732 44.5680389206321,5.98737409390639 50.1195229244701,7.3290919767674 ↩
    55.6587646879025,8.79638749938413 60.1969505994924)
LINESTRING(8.79638749938413 60.1969505994924,9.11375579533779 61.1785363177625,11.6558166691368 ↩
    66.6648504160202,15.642041247655 72.0867690601745,22.8716627200212 77.3609628116894,24.6991785131552 ↩
    77.8939011989848)
LINESTRING(24.6991785131552 77.8939011989848,39.4046096622744 82.1822848017636,44.7994523421035 82.5156766227011)
LINESTRING(44.7994523421035 82.5156766227011,85 85)
```

Figure



Beispiel: Die komplexen Geometrien einer Tabelle werden an Ort und Stelle unterteilt. Die ursprünglichen Geometriedatensätze werden aus der Quelltabelle gelöscht, und neue Datensätze für jede unterteilte Ergebnisgeometrie werden eingefügt.

Code

```
WITH complex_areas_to_subdivide AS (
    DELETE from polygons_table
    WHERE ST_NPoints(geom)
> 255
    RETURNING id, column1, column2, column3, geom
)
INSERT INTO polygons_table (fid, column1, column2, column3, geom)
    SELECT fid, column1, column2, column3,
           ST_Subdivide(geom, 255) as geom
    FROM complex_areas_to_subdivide;
```

Beispiel: Erstellen einer neuen Tabelle mit unterteilten Geometrien unter Beibehaltung des Schlüssels der ursprünglichen Geometrie, so dass die neue Tabelle mit der Quelltabelle verbunden werden kann. Da ST_Subdivide eine mengenrückgebende (Tabellen-)Funktion ist, die eine Menge von Einzelwertzeilen zurückgibt, erzeugt diese Syntax automatisch eine Tabelle mit einer Zeile für jedes Ergebnisteil.

Code

```
CREATE TABLE subdivided_geoms AS
    SELECT pkey, ST_Subdivide(geom) AS geom
    FROM original_geoms;
```

Siehe auch

ST_ClipByBox2D, ST_Segmentize, ST_Split, ST_NPoints, ST_ReducePrecision

7.13.8 ST_SymDifference

ST_SymDifference — Berechnet eine Geometrie, die die Teile der Geometrien A und B darstellt, die sich nicht überschneiden.

Synopsis

geometry **ST_SymDifference**(geometry geomA, geometry geomB, float8 gridSize = -1);

Beschreibung

Gibt eine Geometrie zurück, die die Teile der Geodäten A und B darstellt, die sich nicht schneiden. Dies ist äquivalent zu `ST_Union(A,B) - ST_Intersection(A,B)`. Es wird als symmetrische Differenz bezeichnet, weil `ST_SymDifference(A, = ST_SymDifference(B,A)`.

If the optional `gridSize` parameter is given (GEOS-3.9.0 or higher required), all result vertices are guaranteed to fall on a snap-rounded grid of the specified size. Note that operations performed on a grid may contain small artifacts produced during grid alignment, see ST_ReducePrecision.

Wird durch das GEOS Modul ausgeführt

Verbessert: 3.1.0 akzeptiert einen gridSize-Parameter.

Erfordert GEOS >= 3.9.0 zur Verwendung des Parameters gridSize

Diese Methode implementiert die OGC Simple Features Implementation Specification for SQL 1.1. s2.1.1.3

Diese Methode setzt die SQL/MM-Spezifikation um. SQL-MM 3: 5.1.21

Diese Funktion unterstützt 3d und lässt den Z-Index nicht fallen. Das Ergebnis wird jedoch nur mit XY berechnet. Die Z-Werte des Ergebnisses werden kopiert, gemittelt oder interpoliert.

Beispiele

This example shows a 2D-safe symmetric difference of two LineStrings.

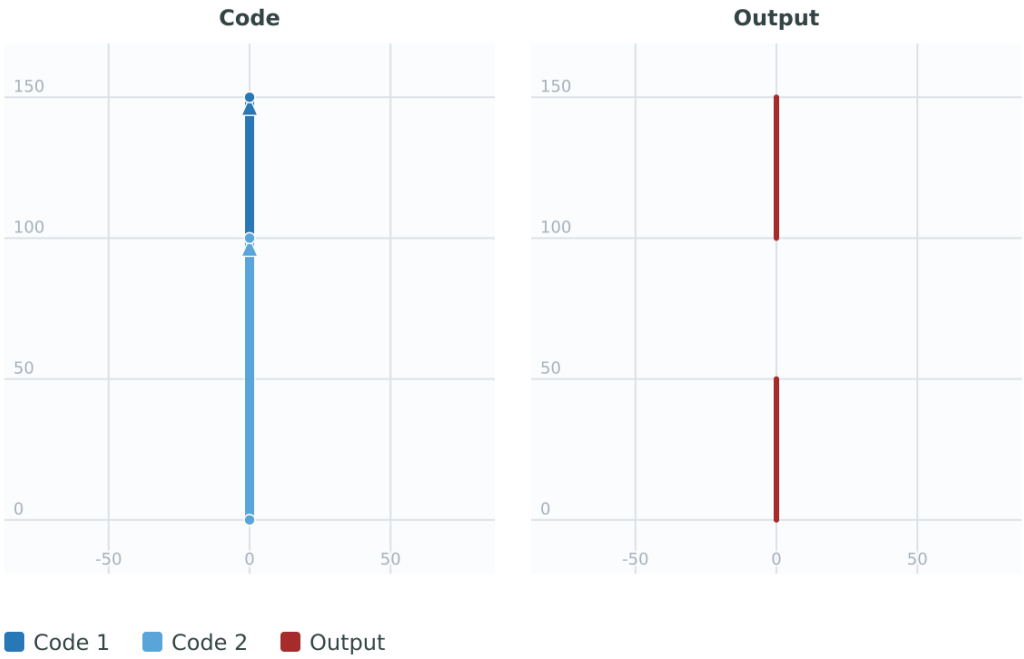
Code

```
SELECT ST_SymDifference('LINESTRING(50 100,50 200)',
        'LINESTRING(50 50,50 150)'
    );
```

Output

```
MULTILINESTRING((50 150,50 200),(50 50,50 100))
```

Figure



In 3D, this example does not produce the expected result.

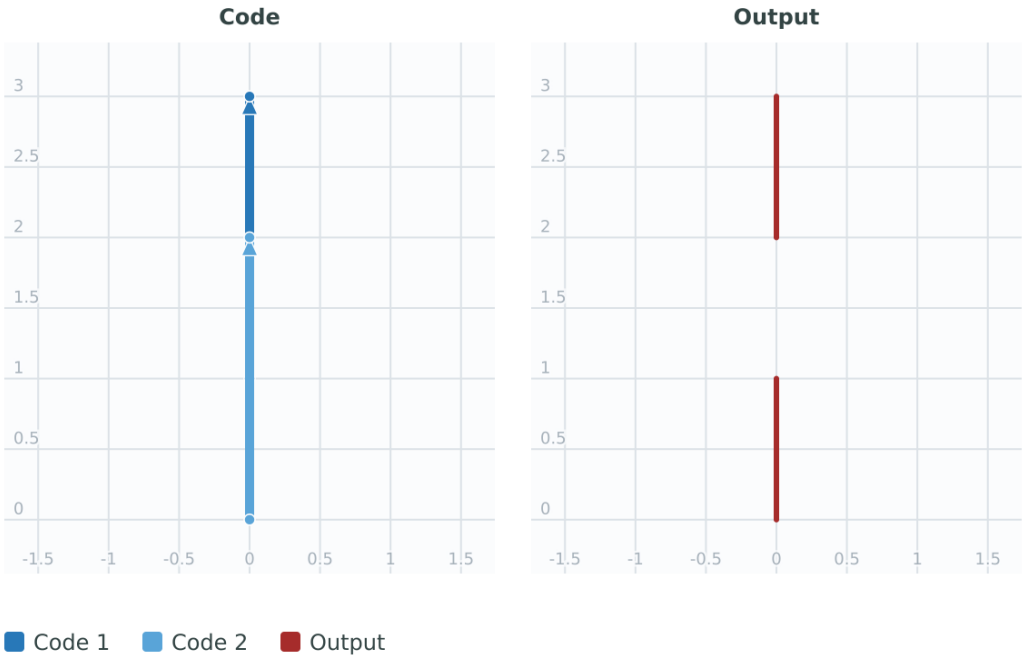
Code

```
SELECT ST_SymDifference('LINESTRING(1 2 1,1 4 2)',
    'LINESTRING(1 1 3,1 3 4)');
```

Output

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

Figure



Siehe auch

ST_Difference, ST_Intersection, ST_Union, ST_ReducePrecision

7.13.9 ST_UnaryUnion

ST_UnaryUnion — Berechnet die Vereinigung der Komponenten einer einzelnen Geometrie.

Synopsis

geometry **ST_UnaryUnion**(geometry geom, float8 gridSize = -1);

Beschreibung

Eine Variante von ST_Union mit einer einzigen Eingabe. Die Eingabe kann eine einzelne Geometrie, eine MultiGeometry oder eine GeometryCollection sein. Die Vereinigung wird auf die einzelnen Elemente der Eingabe angewendet.

Diese Funktion kann verwendet werden, um MultiPolygone zu korrigieren, die aufgrund von überlappenden Komponenten ungültig sind. Allerdings müssen die Eingabekomponenten jeweils gültig sein. Eine ungültige Eingabekomponente, wie z.B. ein Schleifenpolygon, kann einen Fehler verursachen. Aus diesem Grund kann es besser sein, ST_MakeValid zu verwenden.

Eine weitere Anwendung dieser Funktion ist das Verknüpfen und Auflösen einer Sammlung von Linien, die sich kreuzen oder überlappen, um sie einfach zu machen. (ST_Node tut dies auch, bietet aber nicht die Option `gridSize`.)

Es ist möglich, ST_UnaryUnion mit ST_Collect zu kombinieren, um eine Feinabstimmung darüber vorzunehmen, wie viele Geometrien auf einmal vereinigt werden sollen. Dies ermöglicht eine Abwägung zwischen Speichernutzung und Rechenzeit, wobei ein Gleichgewicht zwischen ST_Union und ST_MemUnion hergestellt wird.

If the optional `gridSize` parameter is given (GEOS-3.9.0 or higher required), all result vertices are guaranteed to fall on a snap-rounded grid of the specified size. Note that operations performed on a grid may contain small artifacts produced during grid alignment, see ST_ReducePrecision.

 Diese Funktion unterstützt 3d und lässt den Z-Index nicht fallen. Das Ergebnis wird jedoch nur mit XY berechnet. Die Z-Werte des Ergebnisses werden kopiert, gemittelt oder interpoliert.

Verbessert: 3.1.0 akzeptiert einen gridSize-Parameter.

Erfordert GEOS >= 3.9.0 zur Verwendung des Parameters gridSize

Verfügbarkeit: 2.0.0

Siehe auch

ST_Union, ST_MemUnion, ST_MakeValid, ST_Collect, ST_Node, ST_ReducePrecision

7.13.10 ST_Union

ST_Union — Berechnet eine Geometrie, die die Punktmengenvereinigung der Eingabegeometrien darstellt.

Synopsis

geometry **ST_Union**(geometry g1, geometry g2);
geometry **ST_Union**(geometry g1, geometry g2, float8 gridSize);
geometry **ST_Union**(geometry[] g1_array);
geometry **ST_Union**(geometry set g1field);
geometry **ST_Union**(geometry set g1field, float8 gridSize);

Beschreibung

Vereinigt die Eingabegeometrien und führt die Geometrien zusammen, um eine überschneidungsfreie Ergebnisgeometrie zu erzeugen. Die Ausgabe kann eine atomare Geometrie, eine MultiGeometry oder eine Geometriesammlung sein. Gibt es in mehreren Varianten:

Variante mit zwei Eingaben: gibt eine Geometrie zurück, die die Vereinigung von zwei Eingabegeometrien ist. Wenn eine der beiden Eingaben NULL ist, wird NULL zurückgegeben.

Array-Variante: gibt eine Geometrie zurück, die die Vereinigung eines Arrays von Geometrien ist.

Aggregat-Variante: gibt eine Geometrie zurück, die die Vereinigung eines Rowsets von Geometrien ist. Die Funktion ST_Union() ist eine "Aggregat"-Funktion in der Terminologie von PostgreSQL. Das bedeutet, dass sie mit Datenzeilen arbeitet, so wie es auch die Funktionen SUM() und AVG() tun, und wie die meisten Aggregate ignoriert sie auch NULL-Geometrien.

Siehe ST_UnaryUnion für eine nicht aggregierte Variante mit einem Eingang.

Die ST_Union Array- und Set-Varianten verwenden den schnellen Cascaded Union-Algorithmus, der in http://blog.cleverelephant.ca/-2009/01/must-faster-unions-in-postgis-14.html beschrieben ist.

If the optional `gridSize` parameter is given (GEOS-3.9.0 or higher required), all result vertices are guaranteed to fall on a snap-rounded grid of the specified size. Note that operations performed on a grid may contain small artifacts produced during grid alignment, see ST_ReducePrecision.



Anmerkung
ST_Collect kann manchmal anstelle von ST_Union verwendet werden, wenn das Ergebnis nicht überlappungsfrei sein muss. ST_Collect ist in der Regel schneller als ST_Union, da es keine Verarbeitung der gesammelten Geometrien vornimmt.

Wird vom GEOS Modul ausgeführt

ST_Union erzeugt MultiLineString und fügt LineStrings nicht zu einem einzigen LineString zusammen. Verwenden Sie ST_LineMerge, um LineStrings zusammenzufügen.

HINWEIS: Diese Funktion hieß früher GeomUnion() und wurde von "Union" umbenannt, da UNION ein reserviertes SQL-Wort ist.

Verbessert: 3.1.0 akzeptiert einen gridSize-Parameter.

Erfordert GEOS >= 3.9.0 zur Verwendung des Parameters gridSize

Geändert: 3.0.0 ist nicht von SFCGAL abhängig.

Verfügbarkeit: 1.4.0 - ST_Union wurde verbessert. ST_Union(geomarray) wurde eingeführt und auch schnellere Aggregat-Sammlung in PostgreSQL.

 Diese Methode implementiert die OGC Simple Features Implementation Specification for SQL 1.1. s2.1.1.3



Anmerkung
Die Aggregatversion ist in der OGC SPEC nicht explizit definiert.

 Diese Methode setzt die SQL/MM-Spezifikation um. SQL-MM 3: 5.1.19 der z-index (Elevation), wenn Polygone beteiligt sind.

 Diese Funktion unterstützt 3d und lässt den Z-Index nicht fallen. Das Ergebnis wird jedoch nur mit XY berechnet. Die Z-Werte des Ergebnisses werden kopiert, gemittelt oder interpoliert.

Beispiele

Beispiel für ein Aggregat

Code

```
SELECT id,
       ST_Union(geom) as singlegeom
FROM sometable f
GROUP BY id;
```

Nicht-Aggregat-Beispiel

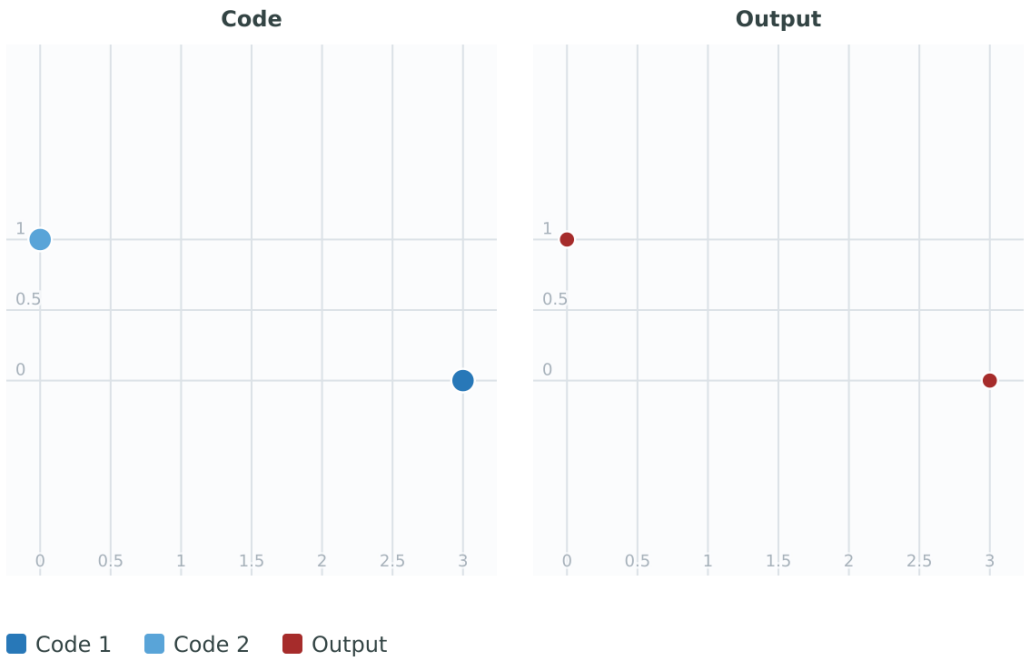
Code

```
SELECT ST_Union('POINT(1 2)'::geometry, 'POINT(-2 3)'::geometry);
```

Output

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

Figure



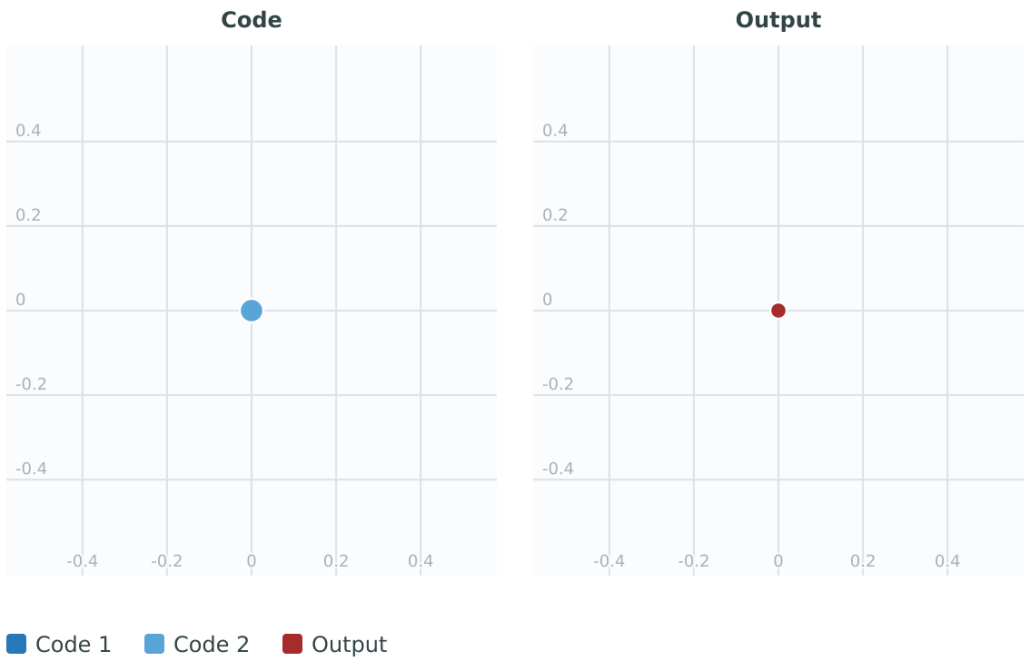
Code

```
SELECT ST_Union('POINT(1 2)'::geometry, 'POINT(1 2)'::geometry);
```

Output

```
POINT(1 2)
```

Figure



3D example - sort of supports 3D (and with mixed geometry types!)

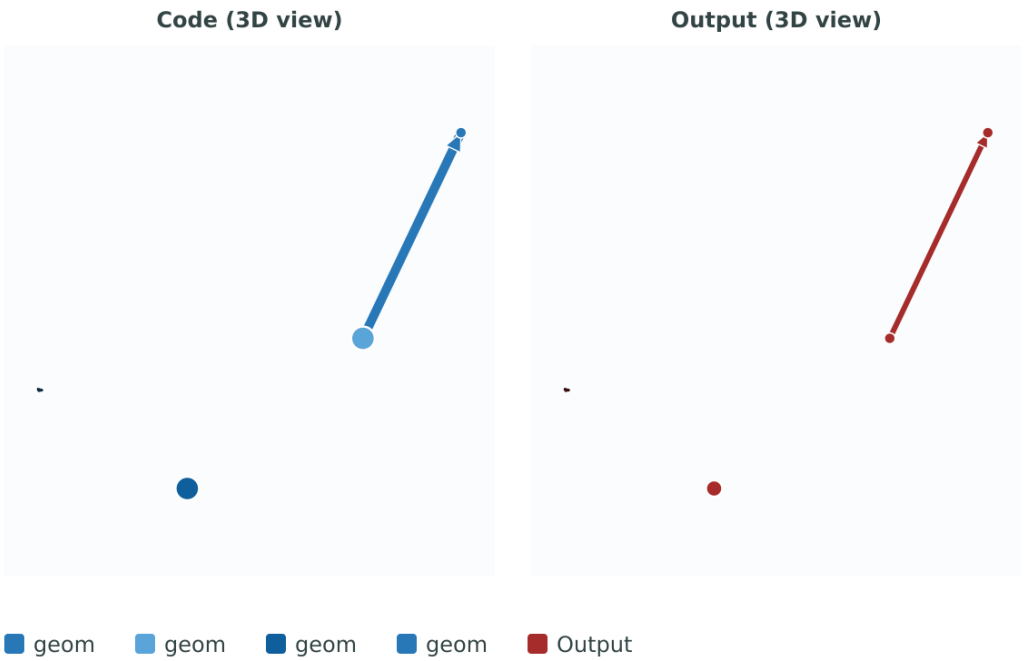
Code

```
SELECT ST_Union(geom)
FROM (
    SELECT 'POLYGON Z((-7 4.2 5,-7.1 4.2 5,-7.1 4.3 5,-7 4.2 5))'::geometry geom
    UNION ALL
    SELECT 'POINT Z(5 5 5)'::geometry geom
    UNION ALL
    SELECT 'POINT Z(-2 3 1)'::geometry geom
    UNION ALL
    SELECT 'LINESTRING Z(5 5 5,10 10 10)'::geometry geom
) AS foo;
```

Output

```
GEOMETRYCOLLECTION(POINT(-2 3 1), LINESTRING(5 5 5,10 10 10), POLYGON((-7 4.2 5,-7.1 4.2 5,-7.1 4.3 5,-7 4.2 5)))
```

Figure



This 3D example does not mix dimensions.

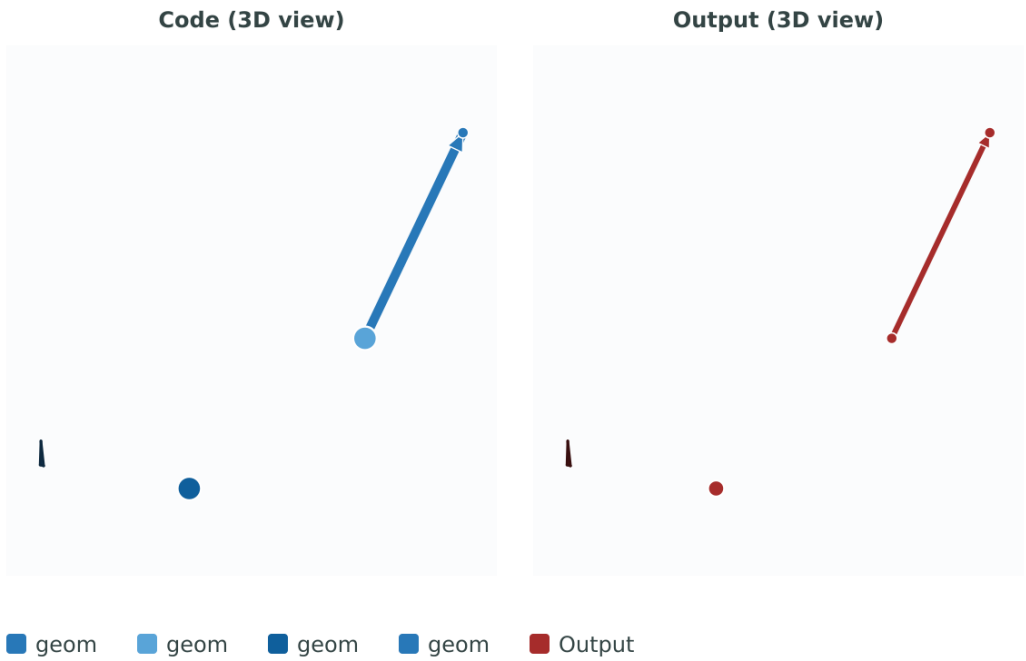
Code

```
SELECT ST_Union(geom)
FROM (
    SELECT 'POLYGON((-7 4.2 2,-7.1 4.2 3,-7.1 4.3 2,-7 4.2 2))'::geometry geom
    UNION ALL
    SELECT 'POINT(5 5 5)'::geometry geom
    UNION ALL
    SELECT 'POINT(-2 3 1)'::geometry geom
    UNION ALL
    SELECT 'LINESTRING(5 5 5,10 10 10)'::geometry geom
) AS foo;
```

Output

```
GEOMETRYCOLLECTION(POINT(-2 3 1), LINESTRING(5 5 5,10 10 10), POLYGON((-7 4.2 2,-7.1 4.2 3,-7.1 4.3 2,-7 4.2 2)))
```

Figure



This example unions an array built from a query.

Code

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

This example unions an array of linestrings.

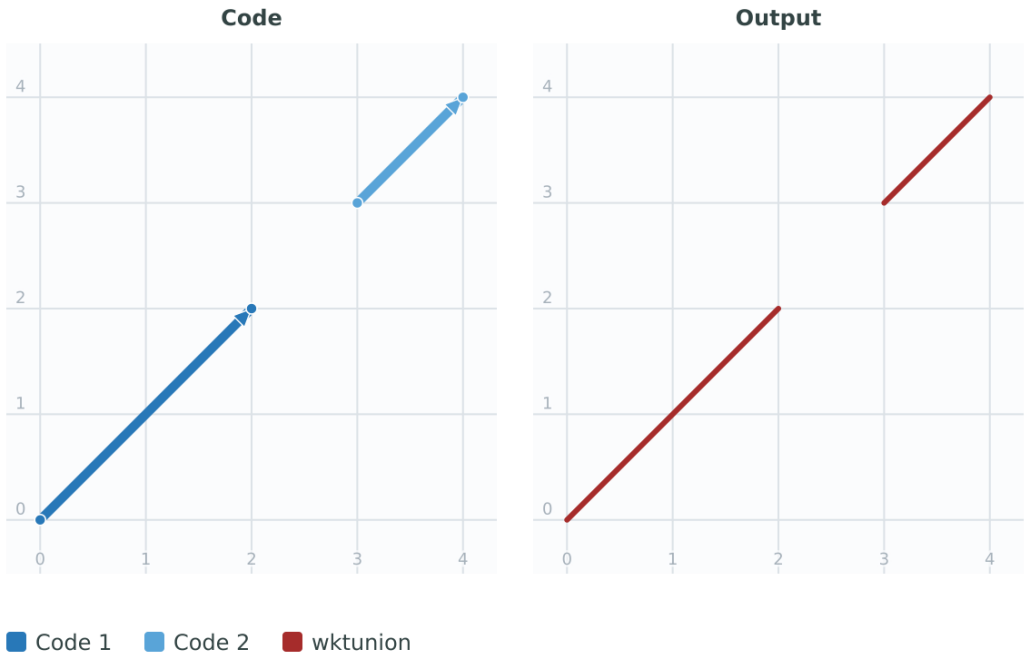
Code

```
SELECT ST_Union(ARRAY[
    'LINESTRING(1 2,3 4)'::geometry,
    'LINESTRING(4 5,5 6)'::geometry
]) As wktunion;
```

Output

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

Figure



Siehe auch

ST_CoverageUnion, ST_UnaryUnion, ST_MemUnion, ST_Collect, ST_Intersection, ST_Difference, ST_SymDifference, ST_LineMerg

7.14 Geometrieverarbeitung

7.14.1 ST_Buffer

ST_Buffer — Berechnet eine Geometrie, die alle Punkte innerhalb eines bestimmten Abstands zu einer Geometrie umfasst.

Synopsis

geometry **ST_Buffer**(geometry g1, float radius_of_buffer, text buffer_style_parameters = '');
geometry **ST_Buffer**(geometry g1, float radius_of_buffer, integer num_seg_quarter_circle);
geography **ST_Buffer**(geography g1, float radius_of_buffer, text buffer_style_parameters);
geography **ST_Buffer**(geography g1, float radius_of_buffer, integer num_seg_quarter_circle);

Beschreibung

Berechnet ein POLYGON oder MULTIPOLYGON, das alle Punkte darstellt, deren Abstand zu einer Geometrie/Geografie kleiner oder gleich einem bestimmten Abstand ist. Ein negativer Abstand verkleinert die Geometrie, anstatt sie zu vergrößern. Ein negativer Abstand kann ein Polygon vollständig schrumpfen lassen; in diesem Fall wird POLYGON EMPTY zurückgegeben. Für Punkte und Linien geben negative Abstände immer leere Ergebnisse zurück.

Bei Geometrie wird die Entfernung in den Einheiten des räumlichen Bezugssystems der Geometrie angegeben. Für die Geografie wird die Entfernung in Metern angegeben.

Der optionale dritte Parameter steuert die Genauigkeit und den Stil des Puffers. Die Genauigkeit der Kreisbögen im Puffer wird als die Anzahl der Liniensegmente angegeben, die zur Annäherung an einen Viertelkreis verwendet werden (Standardwert ist 8). Der Pufferstil kann durch Angabe einer Liste von durch Leerzeichen getrennten Schlüssel=Wert-Paaren wie folgt spezifiziert werden:

- quad_segs=#' : Anzahl der Liniensegmente, die zur Annäherung an einen Viertelkreis verwendet werden (Standard ist 8).
- endcap=round|flat|square' : Stil der Endkappe (Standardwert ist "round"); "butt" wird als Synonym für "flat" akzeptiert.
- join=round|mitre|bevel' : Art der Verbindung (Standardeinstellung ist "round"). 'miter' wird als Synonym für 'mitre' akzeptiert.
- 'mitre_limit=#.#' : Begrenzung des Gehrungsverhältnisses (wirkt sich nur auf Gehrungsverbindungen aus). 'miter_limit' wird als Synonym für 'mitre_limit' akzeptiert.
- 'side=both|left|right' : defaults to 'both'. '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 when 'left' or 'right' are specified.



Anmerkung



Für die Geografie ist dies eine dünne Hülle um die Geometrieimplementierung. Es wird ein planares räumliches Bezugssystem bestimmt, das am besten zum Begrenzungsrahmen des geografischen Objekts passt (versucht werden UTM, Lambert Azimuthal Equal Area (LAEA) Nord-/Südpol und schließlich Mercator). Der Puffer wird im planaren Raum berechnet und dann in WGS84 zurücktransformiert. Dies führt möglicherweise nicht zu dem gewünschten Verhalten, wenn das Eingabeobjekt viel größer als eine UTM-Zone ist oder die Datumsgrenze überschreitet.



Anmerkung

Buffer can handle invalid inputs and the output is always a valid polygonal geometry. Buffering by distance 0 is sometimes used as a way of repairing invalid polygons. ST_MakeValid is more suitable for this process as it can handle multi-polygons.



Anmerkung

Die Pufferung wird manchmal verwendet, um eine Suche innerhalb der Entfernung durchzuführen. Für diesen Anwendungsfall ist es effizienter, ST_DWithin zu verwenden.



Anmerkung

Diese Funktion ignoriert die Z-Dimension. Sie liefert immer ein 2D-Ergebnis, auch wenn sie auf eine 3D-Geometrie angewendet wird.

Erweiterung: 2.5.0 - ST_Buffer ermöglicht jetzt auch eine seitliche Pufferzonenberechnung über `side=both|left|right`.

Verfügbarkeit: 1.5 - ST_Buffer wurde um die Unterstützung von Abschlusstücken/endcaps und Join-Typen erweitert. Diese können zum Beispiel dazu verwendet werden, um Linienzüge von Straßen in Straßenpolygone mit flachen oder rechtwinkeligen Abschlüssen anstatt mit runden Enden umzuwandeln. Ein schlanker Adapter für den geographischen Datentyp wurde hinzugefügt.

Wird vom GEOS Modul ausgeführt



Diese Methode implementiert die OGC Simple Features Implementation Specification for SQL 1.1. s2.1.1.3



Diese Methode setzt die SQL/MM-Spezifikation um. SQL-MM IEC 13249-3: 5.1.30

Beispiele

The textual results are rounded to whole coordinate units for readability. Each result remains available as text, while the manual initially favors the shared input/output figure.

quad_segs=8 (Standardwert)

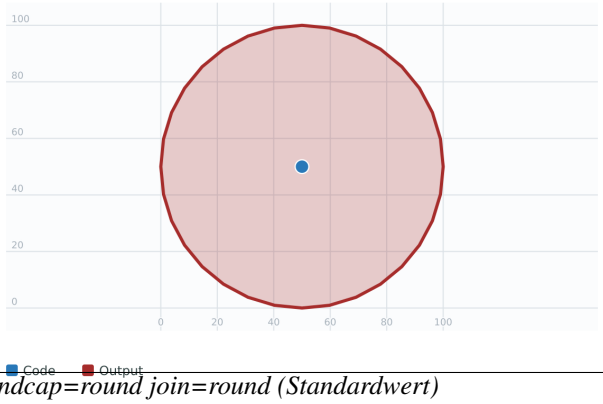
Code

```
SELECT ST_Buffer(
  ST_GeomFromText('POINT(100 90)'),
  50, 'quad_segs=8');
```

Output

```
POLYGON((150 90,149 80,146 71,142 62,135 55,128 48,119
44,110 41,100 40,90 41,81 44,72 48,65 55,58 62,54 71,51
80,50 90,51 100,54 109,58 118,65 125,72 132,81 136,90
139,100 140,110 139,119 136,128 132,135 125,142 118,146
109,149 100,150 90))
```

Figure



quad_segs=2

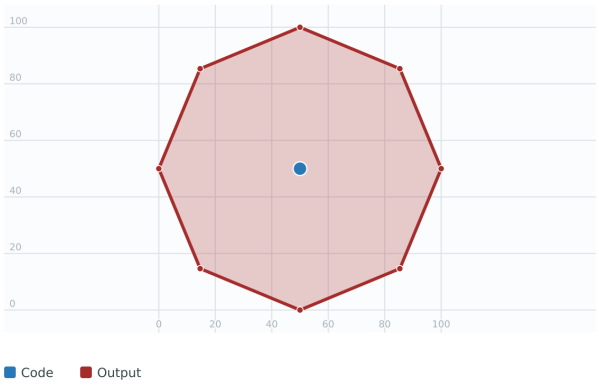
Code

```
SELECT ST_Buffer(
  ST_GeomFromText('POINT(100 90)'),
  50, 'quad_segs=2');
```

Output

```
POLYGON((150 90,135 55,100 40,65 55,50 90,65 125,100
140,135 125,150 90))
```

Figure



endcap=round join=round (Standardwert)

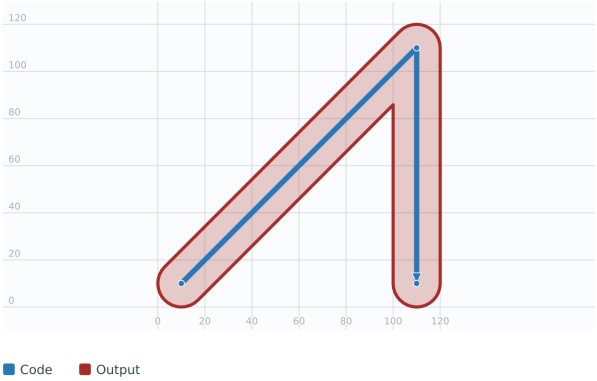
Code

```
SELECT ST_Buffer(
  ST_GeomFromText('LINESTRING(50 50,150 ↩
    150,150 50)'),
  10, 'endcap=round join=round');
```

Output

```
POLYGON((143 157,144 158,146 159,148 160,150 160,152
160,154 159,156 158,157 157,158 156,159 154,160 152,160
150,160 50,160 48,159 46,158 44,157 43,156 42,154 41,152
40,150 40,148 40,146 41,144 42,143 43,142 44,141 46,140
48,140 50,140 126,57 43,56 42,54 41,52 40,50 40,48 40,46
41,44 42,43 43,42 44,41 46,40 48,40 50,40 52,41 54,42
56,43 57,143 157))
```

Figure



endcap=square

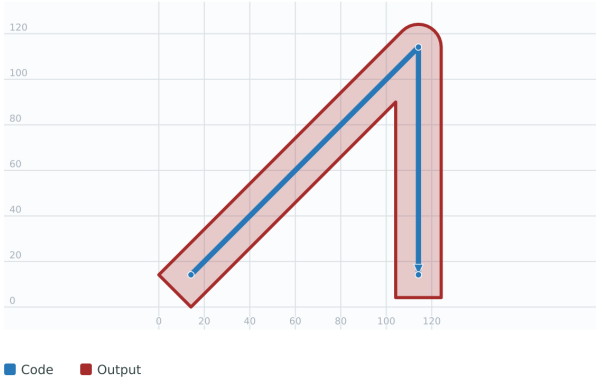
Code

```
SELECT ST_Buffer(
  ST_GeomFromText('LINESTRING(50 50,150 ↩
    150,150 50)'),
  10, 'endcap=square join=round');
```

Output

```
POLYGON((143 157,144 158,146 159,148 160,150 160,152
160,154 159,156 158,157 157,158 156,159 154,160 152,160
150,160 50,160 40,140 40,140 126,57 43,50 36,36 50,143
157))
```

Figure



endcap=flat

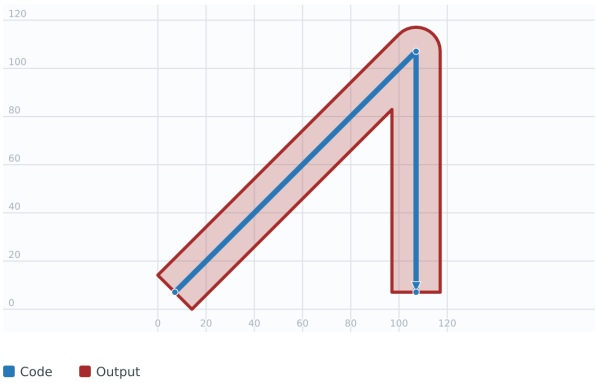
Code

```
SELECT ST_Buffer(
  ST_GeomFromText('LINESTRING(50 50,150 ↩
    150,150 50)'),
  10, 'endcap=flat join=round');
```

Output

```
POLYGON((143 157,144 158,146 159,148 160,150 160,152
160,154 159,156 158,157 157,158 156,159 154,160 152,160
150,160 50,140 50,140 126,57 43,43 57,143 157))
```

Figure



join=bevel

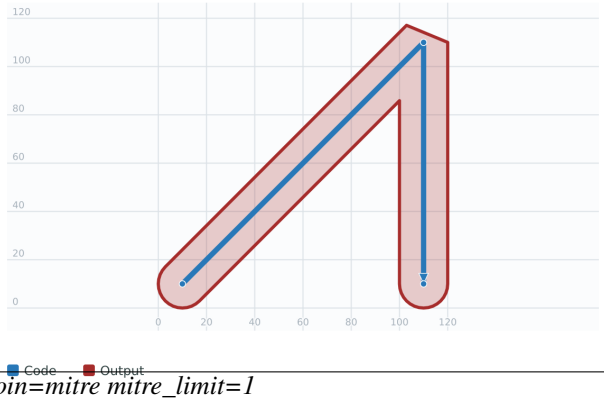
Code

```
SELECT ST_Buffer(
  ST_GeomFromText('LINESTRING(50 50,150 ↩
    150,150 50)'),
  10, 'join=bevel');
```

Output

```
POLYGON((143 157,160 150,160 50,160 48,159 46,158 44,157
43,156 42,154 41,152 40,150 40,148 40,146 41,144 42,143
43,142 44,141 46,140 48,140 50,140 126,57 43,56 42,54
41,52 40,50 40,48 40,46 41,44 42,43 43,42 44,41 46,40
48,40 50,40 52,41 54,42 56,43 57,143 157))
```

Figure



join=mitre mitre_limit=5.0 (default mitre limit)

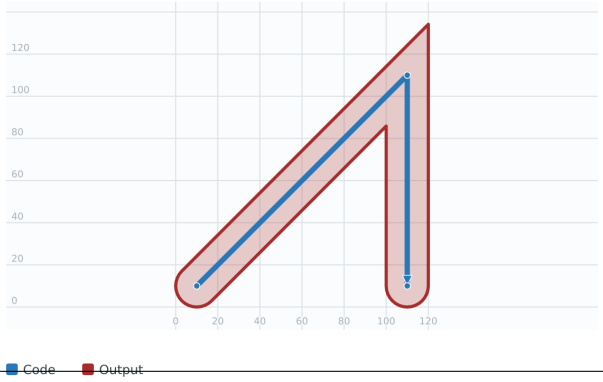
Code

```
SELECT ST_Buffer(
  ST_GeomFromText('LINESTRING(50 50,150 ↩
    150,150 50)'),
  10, 'join=mitre mitre_limit=5.0');
```

Output

```
POLYGON((160 174,160 50,160 48,159 46,158 44,157 43,156
42,154 41,152 40,150 40,148 40,146 41,144 42,143 43,142
44,141 46,140 48,140 50,140 126,57 43,56 42,54 41,52 40,
50 40,48 40,46 41,44 42,43 43,42 44,41 46,40 48,40 50,40
52,41 54,42 56,43 57,160 174))
```

Figure



join=mitre mitre_limit=1

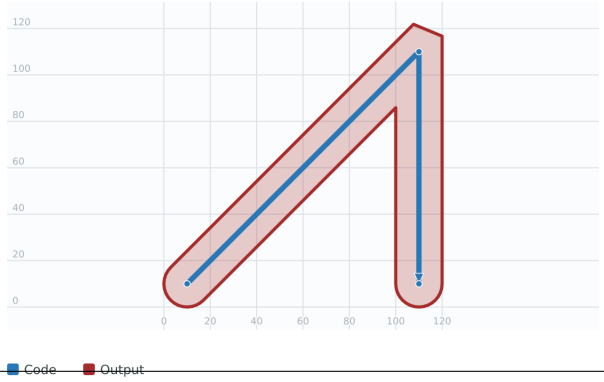
Code

```
SELECT ST_Buffer(
  ST_GeomFromText('LINESTRING(50 50,150 ↩
    150,150 50)'),
  10, 'join=mitre mitre_limit=1.0');
```

Output

```
POLYGON((148 162,160 157,160 50,160 48,159 46,158 44,157
43,156 42,154 41,152 40,150 40,148 40,146 41,144 42,143
43,142 44,141 46,140 48,140 50,140 126,57 43,56 42,54
41,52 40,50 40,48 40,46 41,44 42,43 43,42 44,41 46,40
48,40 50,40 52,41 54,42 56,43 57,148 162))
```

Figure



side=left

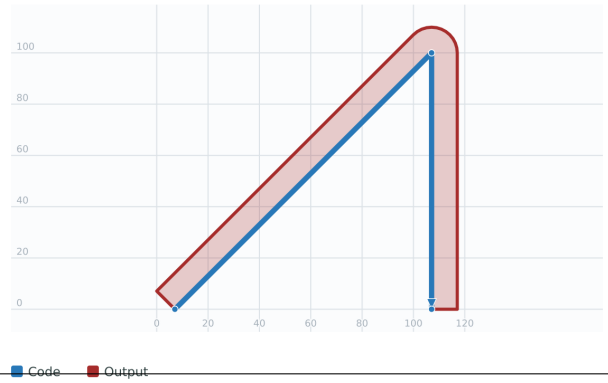
Code

```
SELECT ST_Buffer(
  ST_GeomFromText('LINESTRING(50 50,150 ↩
    150,150 50)'),
  10, 'side=left');
```

Output

```
POLYGON((150 50,150 150,50 50,43 57,143 157,144 158,146
159,148 160,150 160,152 160,154 159,156 158,157 157,158
156,159 154,160 152,160 150,160 50,150 50))
```

Figure



side=right

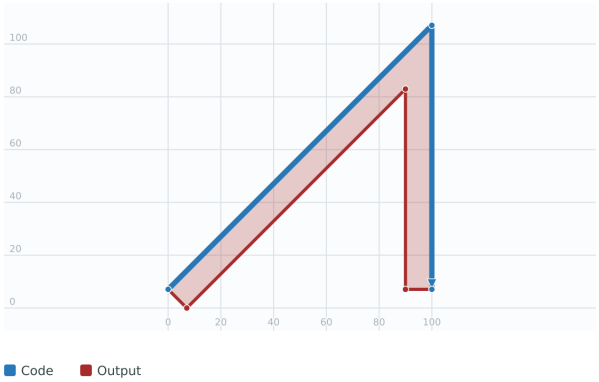
Code

```
SELECT ST_Buffer(
  ST_GeomFromText('LINESTRING(50 50,150 ↩
    150,150 50)'),
  10, 'side=right');
```

Output

```
POLYGON((50 50,150 150,150 50,140 50,140 126,57 43,50
50))
```

Figure



side=left join=mitre

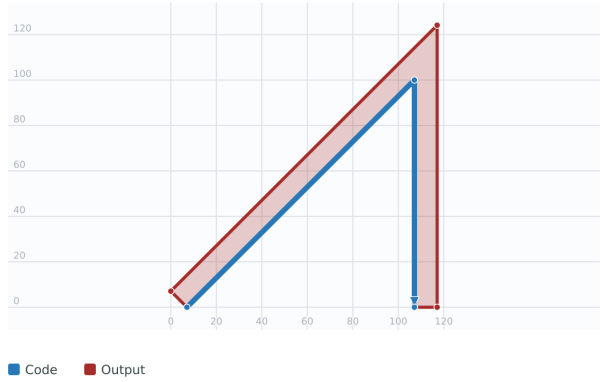
Code

```
SELECT ST_Buffer(
  ST_GeomFromText('LINESTRING(50 50,150 ↩
    150,150 50)'),
  10, 'side=left join=mitre');
```

Output

```
POLYGON((150 50,150 150,50 50,43 57,160 174,160 50,150
50))
```

Figure



right-hand winding, polygon boundary side=left

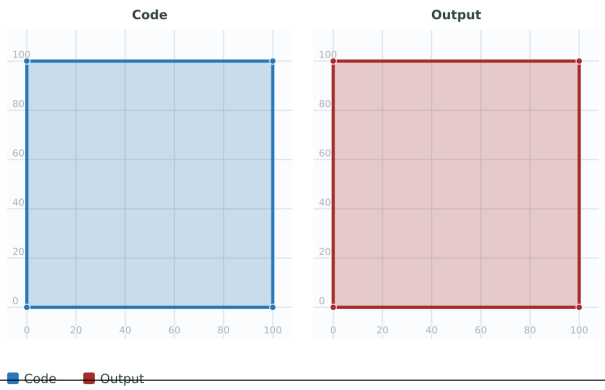
Code

```
SELECT ST_Buffer(
  ST_ForceRHR(ST_Boundary(ST_GeomFromText ↩
    (
    'POLYGON ((50 50,50 150,150 150,150 ↩
    50,50 50))'))),
  20, 'side=left');
```

Output

```
POLYGON((50 50,50 150,150 150,150 50,50 50))
```

Figure



right-hand winding, polygon boundary side=right

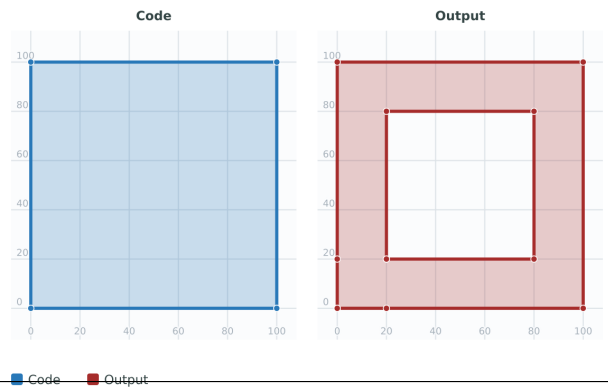
Code

```
SELECT ST_Buffer(
  ST_ForceRHR(ST_Boundary(ST_GeomFromText ↩
    (
    'POLYGON ((50 50,50 150,150 150,150  ↩
    50,50 50))')))),
  20, 'side=right');
```

Output

```
POLYGON((50 50,50 70,50 150,150 150,150 50,70 50,50 50),
(70 70,130 70,130 130,70 130,70 70))
```

Figure



A buffered point approximates a circle. A buffered point forcing approximation of (see diagram) 2 points per quarter circle is poly with 8 sides (see diagram)

Code

```
SELECT ST_NPoints(ST_Buffer('POINT(100 90)'::geometry, 50)) As promisingcircle_pcount,
ST_NPoints(ST_Buffer('POINT(100 90)'::geometry, 50, 2)) As lamecircle_pcount;
```

Output

```
promisingcircle_pcount | lamecircle_pcount
-----------------------+-------------------
                    33 |                 9
```

A lighter, less smooth circle using only 2 points per quarter circle is an octagon.

The following example creates a 100 meter octagon in NAD83 longitude/latitude by transforming to the Massachusetts state plane meter projection and then buffering. The figure shows the input and output in separate panels because the coordinates are in different SRIDs.

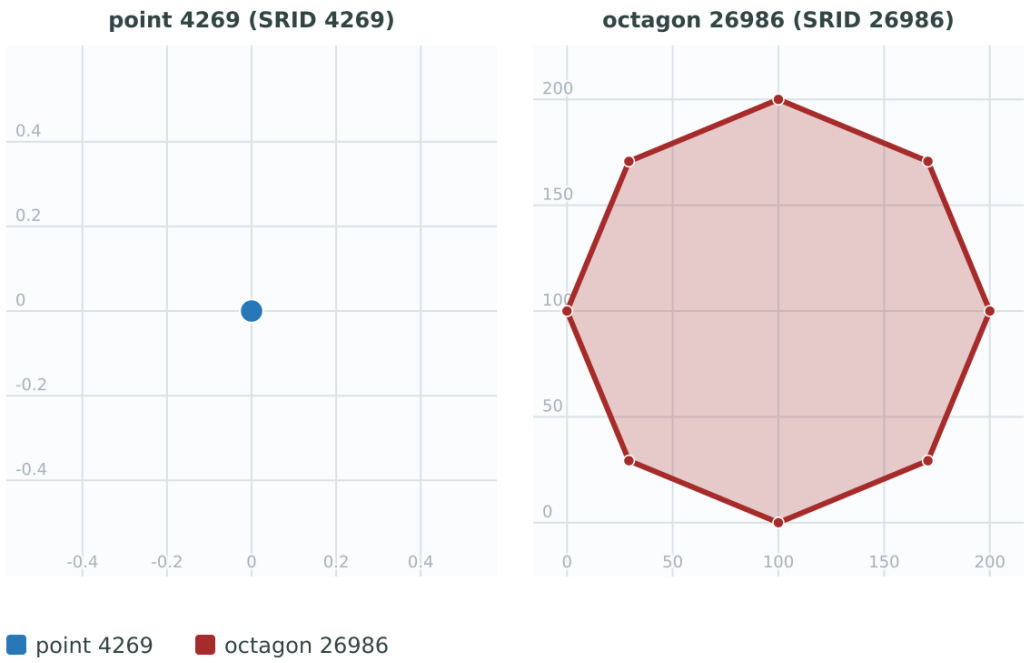
Code

```
WITH input AS (
  SELECT ST_SetSRID(ST_Point(-71.063526, 42.35785), 4269) AS point_4269
), projected AS (
  SELECT point_4269,
         ST_Transform(point_4269, 26986) AS point_26986
  FROM input
)
SELECT point_4269 AS input_point_4269,
       ST_Buffer(point_26986, 100, 2) AS octagon_26986
FROM projected;
```

Output

```
SRID=4269;POINT(-71.063526 42.35785) | SRID=26986;POLYGON((236057.5905746494 900908.7599186979,236028.30125276805 ↩
    900838.0492405792,235957.5905746494 900808.7599186979,235886.87989653074 900838.0492405792,235857.5905746494 ↩
    900908.7599186979,235886.87989653074 900979.4705968165,235957.5905746494 901008.7599186979,236028.30125276805 ↩
    900979.4705968165,236057.5905746494 900908.7599186979))
```

Figure



Buffering a polygon by a positive distance and then by the same negative distance is a morphological closing operation. It can be useful for smoothing noisy boundaries, such as simplifying a coastline by removing narrow inlets and bays. The result intentionally changes the shape and area of the input, and the distance must be chosen in the units of the input SRS. For data in a geographic SRS, transform to a projected SRS suited to the operation first; for cartographic visualization in Web Mercator, using that display SRS can be appropriate.

Smooth a coastline using a 1 km closing operation, then simplify the result. The example uses Massachusetts State Plane meter units for the operation.

Code

```
WITH projected AS (
  SELECT id, ST_Transform(geom, 26986) AS geom
  FROM coast
),
closed AS (
  SELECT id, ST_Buffer(ST_Buffer(geom, 1000), -1000) AS geom
  FROM projected
)
SELECT id, ST_Transform(ST_SimplifyPreserveTopology(geom, 50), 4326) AS geom
FROM closed;
```

The same technique is described in Paul Ramsey's Removing Complexities. The final simplification step is optional; use ST_SimplifyPres when polygonal validity matters, and check the result against the cartographic or analytical tolerance required by the application.

A small negative buffer can also be used as a heuristic to find narrow polygon spikes. The query below compares how much area and perimeter remain after erosion. A large `erosion_index` indicates that the perimeter dropped proportionally more than the area. This is only a preselection aid; the threshold and buffer distance must be chosen for the data units and the kind of spike being checked.

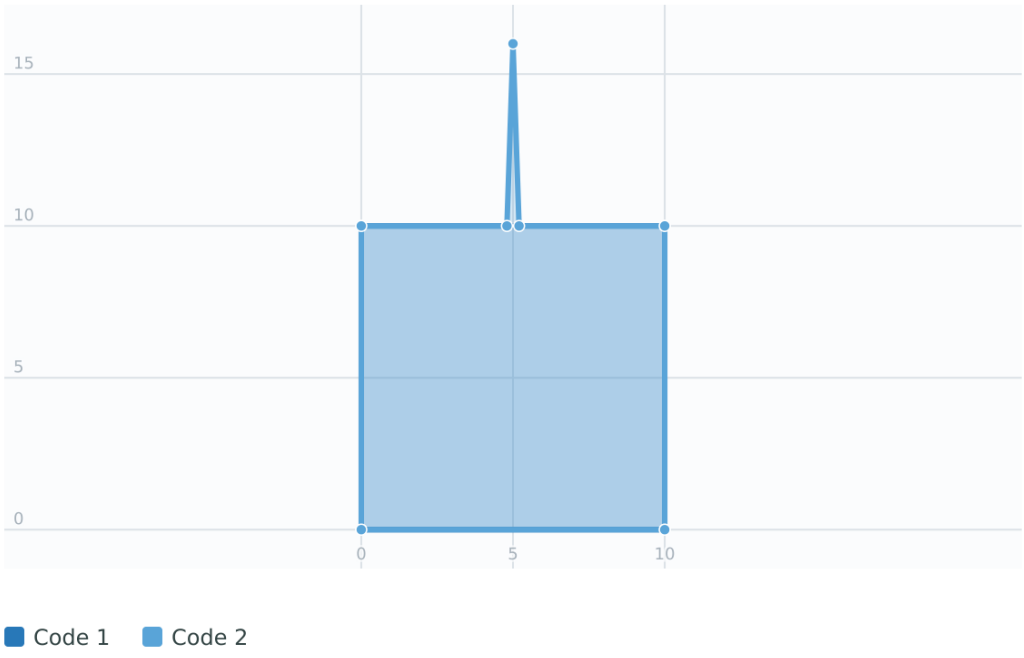
Code

```
WITH sample(id, geom) AS (
  VALUES
    ('plain', 'POLYGON((0 0,10 0,10 10,0 10,0 0))'::geometry),
    ('spike', 'POLYGON((0 0,10 0,10 10,5.2 10,5 16,4.8 10,0 10,0 0))'::geometry)
),
eroded AS (
  SELECT id, geom, ST_Buffer(geom, -0.5) AS eroded_geom
  FROM sample
),
scored AS (
  SELECT id,
         round((
           ST_Area(eroded_geom) / ST_Area(geom) /
           NULLIF(ST_Perimeter(eroded_geom) / ST_Perimeter(geom), 0)
         )::numeric, 3) AS erosion_index
  FROM eroded
  WHERE NOT ST_IsEmpty(eroded_geom)
)
SELECT id, erosion_index, erosion_index > 1.1 AS possible_spike
FROM scored
ORDER BY id;
```

Output

```
id    | erosion_index | possible_spike
-------+---------------+----------------
 plain |         0.900 | f
 spike |         1.147 | t
```

Figure



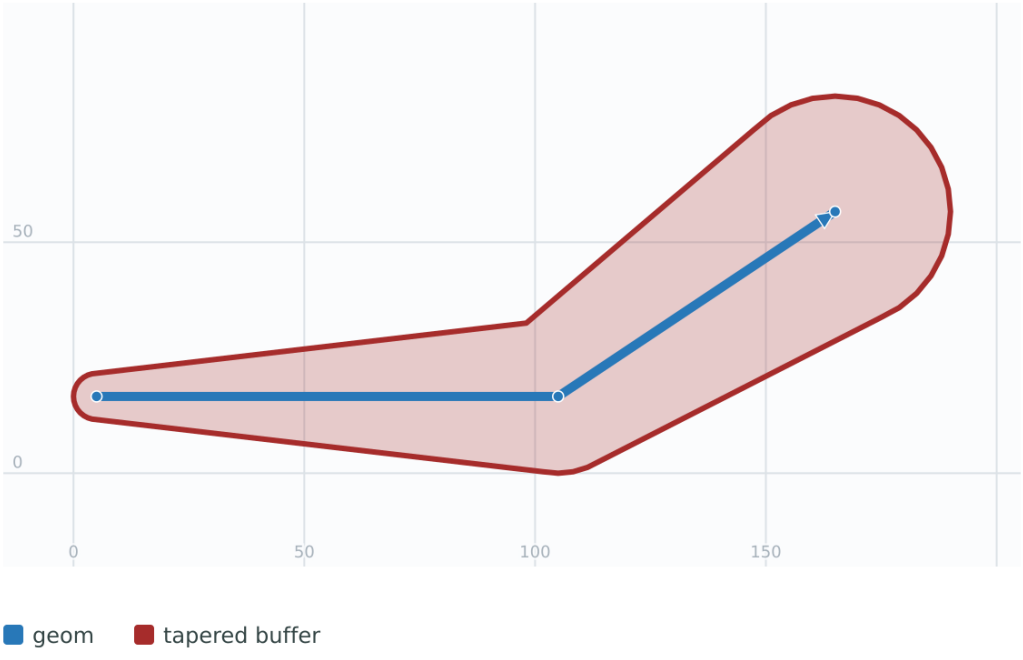
A variable-distance buffer can be approximated by buffering each vertex of a line by a distance interpolated along the line, building the convex hull for each consecutive pair of vertex buffers, and unioning the segment hulls. As with any geometry buffer, distances are in the units of the input SRS.

Taper a line buffer from 5 to 25 units along the line.

Code

```
WITH line AS (
  SELECT 'LINESTRING(0 0, 100 0, 160 40)'::geometry AS geom
),
vertices AS (
  SELECT (dump).path[1] AS n,
         (dump).geom AS geom,
         line.geom AS line_geom
  FROM line
  CROSS JOIN LATERAL ST_DumpPoints(line.geom) AS dump
),
radii AS (
  SELECT n,
         geom,
         5 + (25 - 5) * ST_LineLocatePoint(line_geom, geom) AS radius
  FROM vertices
),
segments AS (
  SELECT n,
         geom,
         radius,
         lead(geom) OVER (ORDER BY n) AS next_geom,
         lead(radius) OVER (ORDER BY n) AS next_radius
  FROM radii
)
SELECT ST_Union(
         ST_ConvexHull(
           ST_Collect(
             ST_Buffer(geom, radius),
             ST_Buffer(next_geom, next_radius)
           )
         )
       ) AS tapered_buffer
FROM segments
WHERE next_geom IS NOT NULL;
```

Figure



Siehe auch

ST_Collect, ST_ConvexHull, ST_DWithin, ST_DumpPoints, ST_LineLocatePoint, ST_SetSRID, ST_SimplifyPreserveTopology, ST_Transform, ST_Union, ST_MakeValid

7.14.2 ST_BuildArea

ST_BuildArea — Erzeugt eine polygonale Geometrie, die aus dem Linienwerk einer Geometrie gebildet wird.

Synopsis

geometry **ST_BuildArea**(geometry geom);

Beschreibung

Erzeugt eine flächige Geometrie, die aus den konstituierenden Linienzügen der Eingabegeometrie gebildet wird. Die Eingabe kann ein LineString, MultiLineString, Polygon, MultiPolygon oder eine GeometryCollection sein. Das Ergebnis ist ein Polygon oder MultiPolygon, je nach Eingabe. Wenn das eingegebene Liniengerüst keine Polygone bildet, wird NULL zurückgegeben.

Im Gegensatz zu ST_MakePolygon akzeptiert diese Funktion Ringe, die aus mehreren Linien bestehen, und kann eine beliebige Anzahl von Polygonen bilden.

Diese Funktion wandelt innere Ringe in Löcher um. Um auch innere Ringe in Polygone zu verwandeln, verwenden Sie ST_Polygonize.



Anmerkung
Damit diese Funktion ordnungsgemäß funktioniert, muss das eingegebene Linienwerk korrekt genodet sein. ST_Node kann zum Knoten von Linien verwendet werden.
Wenn sich das eingegebene Linienwerk kreuzt, erzeugt diese Funktion ungültige Polygone. ST_MakeValid kann verwendet werden, um sicherzustellen, dass die Ausgabe gültig ist.

Verfügbarkeit: 1.1.0

Beispiele

Building an area from five input lines.

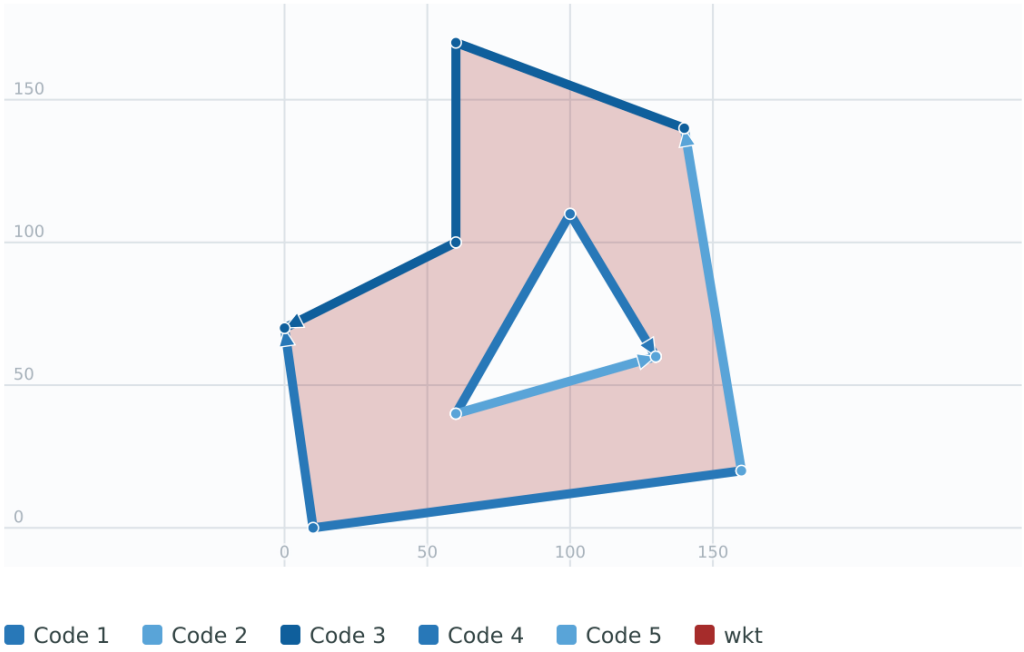
Code

```
WITH data(geom) AS (VALUES
   ('LINESTRING (180 40,30 20,20 90)'::geometry),
  ('LINESTRING (180 40,160 160)'::geometry),
  ('LINESTRING (160 160,80 190,80 120,20 90)'::geometry),
  ('LINESTRING (80 60,120 130,150 80)'::geometry),
  ('LINESTRING (80 60,150 80)'::geometry)
)
SELECT ST_BuildArea(ST_Collect(geom )) AS wkt
    FROM data;
```

Output

```
POLYGON((180 40,30 20,20 90,80 120,80 190,160 160,180 40),(150 80,120 130,80 60,150 80))
```

Figure

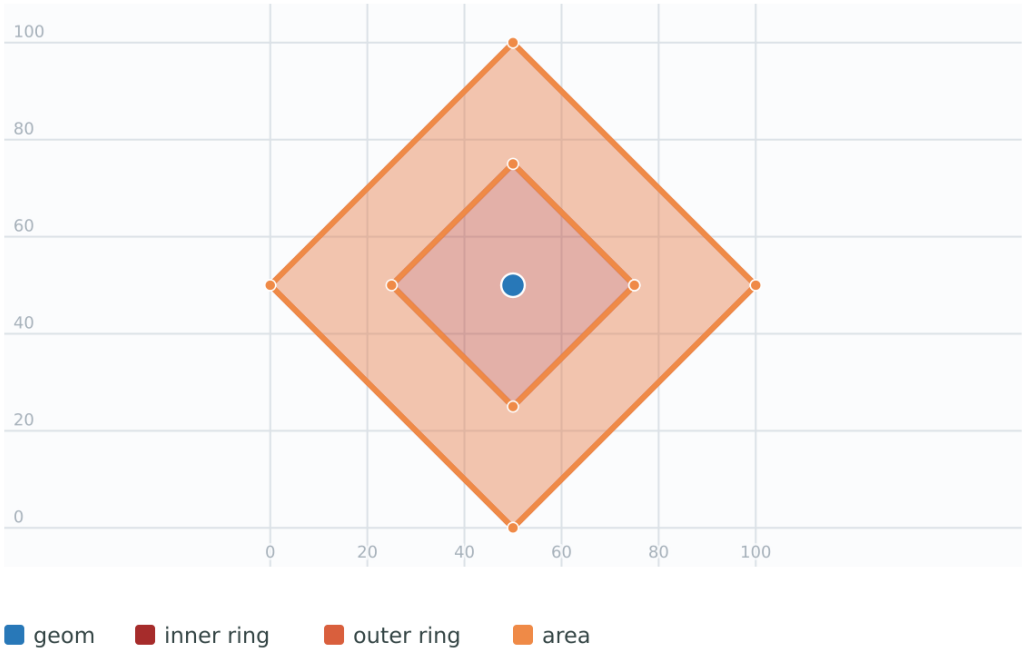


Create a donut from two circular polygons.

Code

```
WITH input AS (
  SELECT 'POINT(100 90)'::geometry AS geom
),
rings AS (
  SELECT ST_Buffer(geom, 25, 'quad_segs=1') AS inring,
         ST_Buffer(geom, 50, 'quad_segs=1') AS outring
  FROM input
)
SELECT inring AS inner_ring,
       outring AS outer_ring,
       ST_BuildArea(ST_Collect(inring, outring)) AS area
FROM rings;
```

Figure



Input linework that crosses at non-endpoint intersections must be noded before building areas:

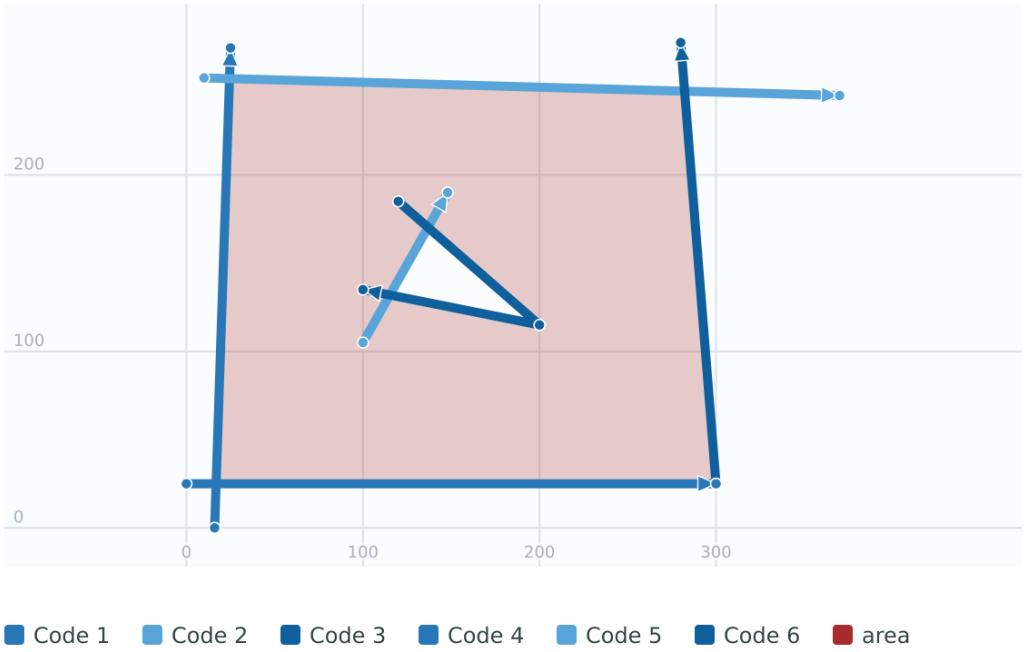
Code

```
WITH lines(geom) AS (VALUES
  ('LINESTRING(36 35,45 307)'::geometry),
  ('LINESTRING(30 290,390 280)'::geometry),
  ('LINESTRING(320 60,300 310)'::geometry),
  ('LINESTRING(20 60,320 60)'::geometry),
  ('LINESTRING(120 140,168 225)'::geometry),
  ('LINESTRING(140 220,220 150,120 170)'::geometry)
),
noded AS (
  SELECT ST_Node(ST_UnaryUnion(ST_Collect(geom))) AS geom FROM lines
)
SELECT ST_BuildArea(geom) AS area
FROM noded;
```

Output

```
POLYGON((320 60,36.8 60,44.4 289.6,302.2 282.4,320 60),(156.9 205.3,135.2 167,220 150,156.9 205.3))
```

Figure



Siehe auch

ST_Collect, ST_MakePolygon, ST_MakeValid, ST_Node, ST_Polygonize, ST_BdPolyFromText, ST_BdMPolyFromText (Wrapper zu dieser Funktion mit Standard OGC Schnittstelle)

7.14.3 ST_Centroid

ST_Centroid — Gibt den geometrischen Schwerpunkt einer Geometrie zurück.

Synopsis

geometry **ST_Centroid**(geometry g1);
geography **ST_Centroid**(geography g1, boolean use_spheroid = true);

Beschreibung

Berechnet einen Punkt, der den geometrischen Schwerpunkt einer Geometrie darstellt. Für [MULTI]POINTs ist der Schwerpunkt das arithmetische Mittel der eingegebenen Koordinaten. Für [MULTI]LINESTRINGs wird der Schwerpunkt anhand der gewichteten Länge der einzelnen Liniensegmente berechnet. Für [MULTI]POLYGONs wird der Schwerpunkt anhand der Fläche berechnet. Wenn eine leere Geometrie übergeben wird, wird eine leere GEOMETRYCOLLECTION zurückgegeben. Wenn NULL übergeben wird, wird NULL zurückgegeben. Wenn CIRCULARSTRING oder COMPOUNDCURVE übergeben werden, werden sie zuerst mit CurveToLine in einen Linestring umgewandelt, dann genauso wie bei LINESTRING

Bei gemischtdimensionalen Eingaben entspricht das Ergebnis dem Schwerpunkt der Komponente Geometrien mit der höchsten Dimension (da die Geometrien mit geringerer Dimension kein "Gewicht" zum Schwerpunkt beitragen).

Beachten Sie, dass bei polygonalen Geometrien der Schwerpunkt nicht unbedingt im Inneren des Polygons liegt. Siehe zum Beispiel das folgende Diagramm des Schwerpunkts eines C-förmigen Polygons. Um einen Punkt zu konstruieren, der garantiert im Inneren eines Polygons liegt, verwenden Sie ST_PointOnSurface.

Neu in 2.3.0 : unterstützt CIRCULARSTRING und COMPOUNDCURVE (mit CurveToLine)

Verfügbarkeit: Mit 2.4.0 wurde die Unterstützung für den geograpischen Datentyp eingeführt.

Diese Methode implementiert die OGC Simple Features Implementation Specification for SQL 1.1.

Diese Methode setzt die SQL/MM-Spezifikation um. SQL-MM 3: 8.1.4, 9.5.5

Beispiele

Centroid of a MultiPoint.

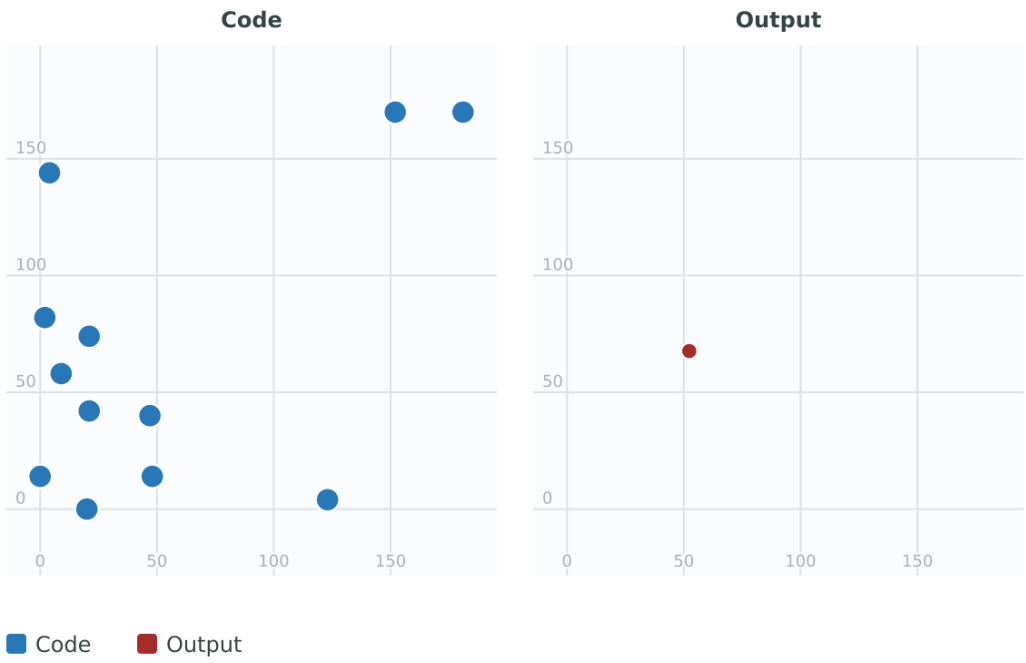
Code

```
SELECT ST_Centroid(
  'MULTIPOINT (8 24,10 92,12 154,17 68,28 10,29 52,29 84,55 50,56 24,131 14,160 180,189 180) ↩
    ');
```

Output

```
POINT(60.333 77.667)
```

Figure



Centroid of a LineString.

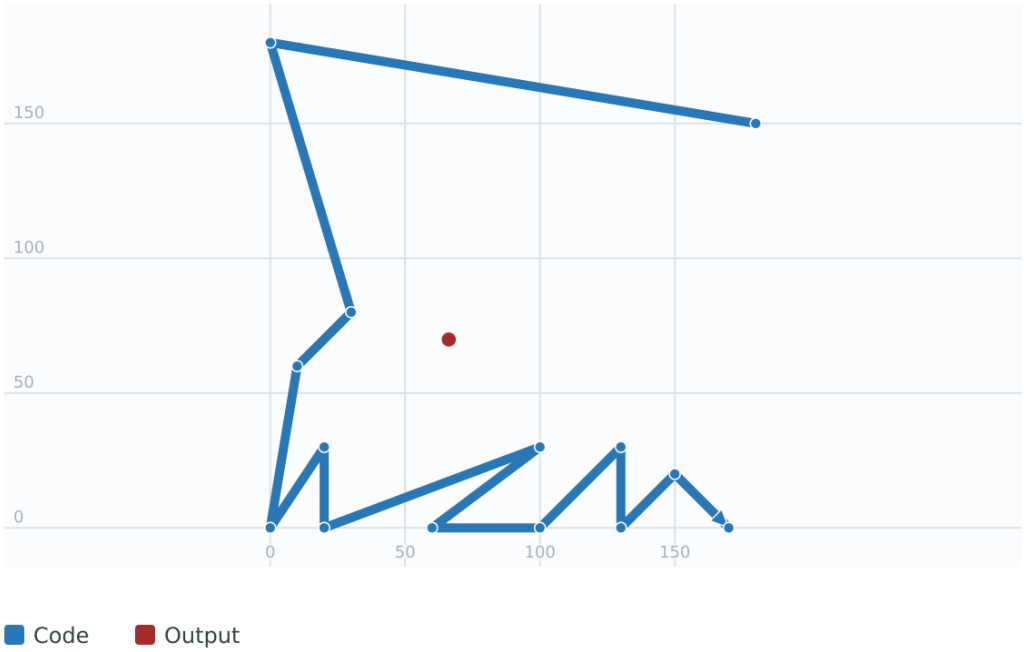
Code

```
SELECT ST_Centroid(
  'LINESTRING (190 160,10 190,40 90,20 70,10 10,30 40,30 10,110 40,70 10,110 10,140 40,140  ↩
    10,160 30,180 10)');
```

Output

```
POINT(76.191 79.876)
```

Figure



Centroid of a Polygon.

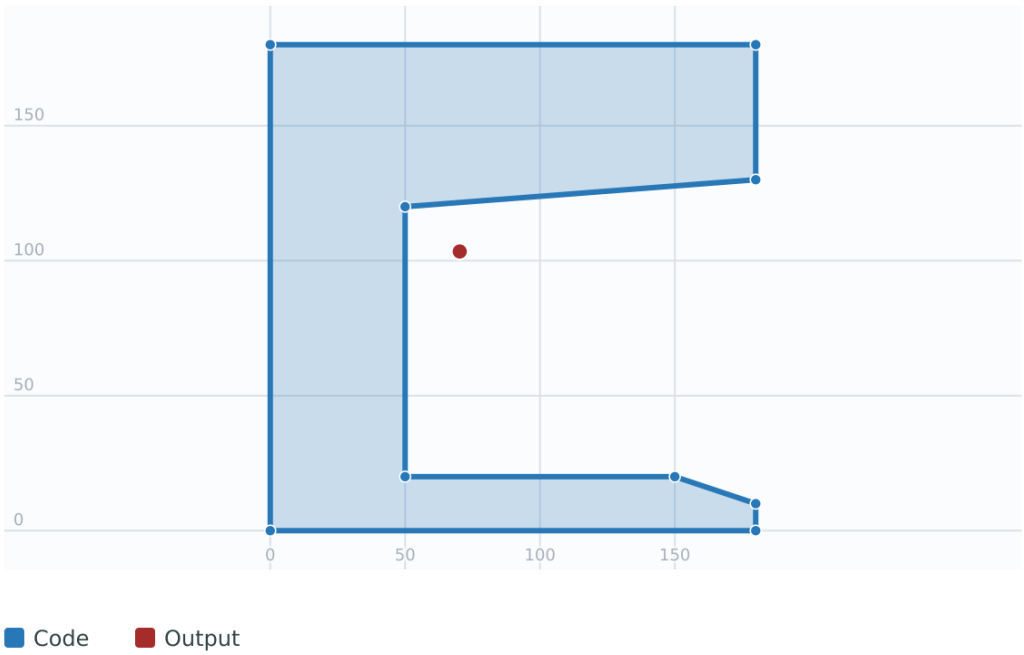
Code

```
SELECT ST_Centroid(
  'POLYGON ((190 190,10 190,10 10,190 10,190 20,160 30,60 30,60 130,190 140,190 190))');
```

Output

```
POINT(80.251 113.405)
```

Figure



Centroid of the highest-dimension components of a GeometryCollection.

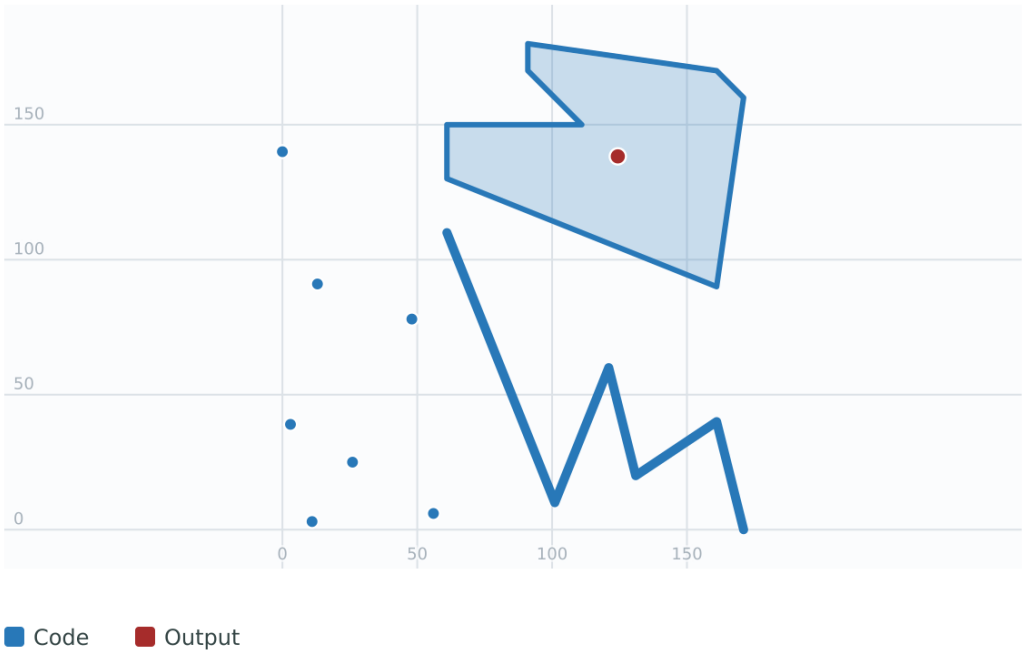
Code

```
SELECT ST_Centroid(
  'GEOMETRYCOLLECTION (POLYGON ((190 170,180 100,80 140,80 160,130 160,110 180,110 190,180 ↩
   180,190 170)),
    LINESTRING (80 120,120 20,140 70,150 30,180 50,190 10),
    MULTIPOINT (19 150,22 49,30 13,32 101,45 35,67 88,75 16))');
```

Output

```
POINT(143.361 148.263)
```

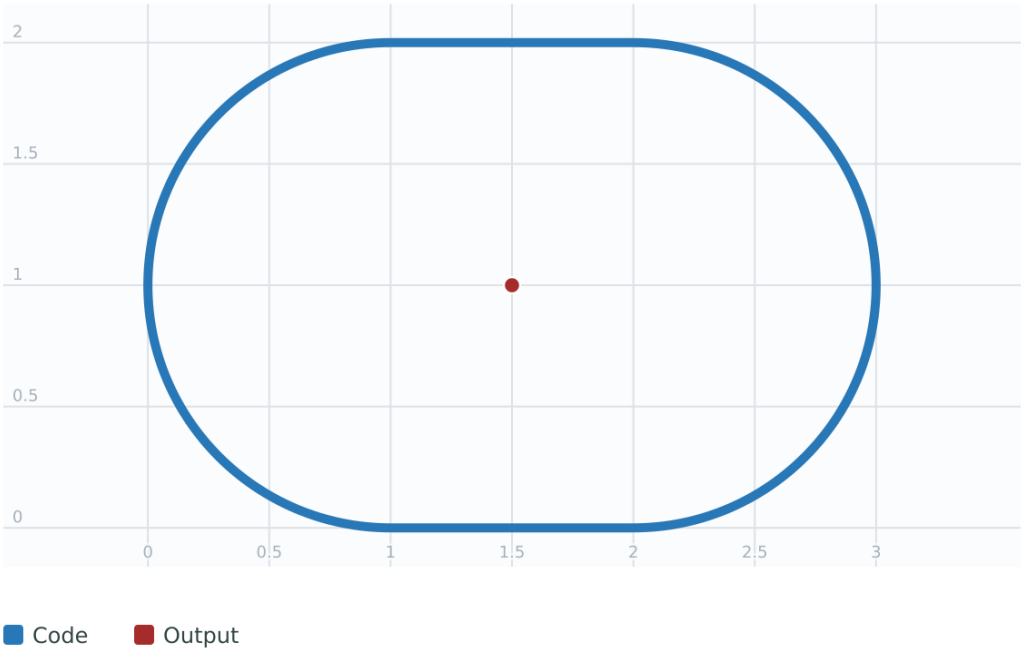
Figure



Code

```
SELECT ST_Centroid(g)
FROM  ST_GeomFromText('CIRCULARSTRING(0 2,-1 1,0 0,0.5 0,1 0,2 1,1 2,0.5 2,0 2)')  AS g ;
```

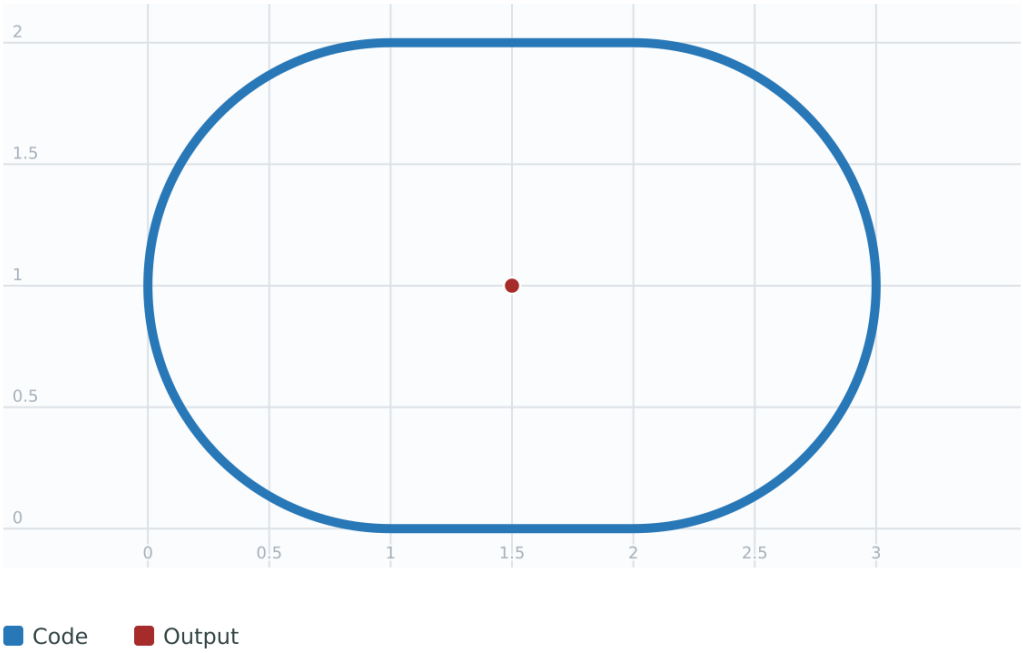
Figure



Code

```
SELECT ST_Centroid(g)
FROM  ST_GeomFromText('COMPOUNDCURVE(CIRCULARSTRING(0 2,-1 1,0 0),(0 0,0.5 0,1 0),  ↩
    CIRCULARSTRING( 1 0,2 1,1 2),(1 2,0.5 2,0 2))' ) AS g;
```

Figure



Centroids of the parts of a MultiPolygon.

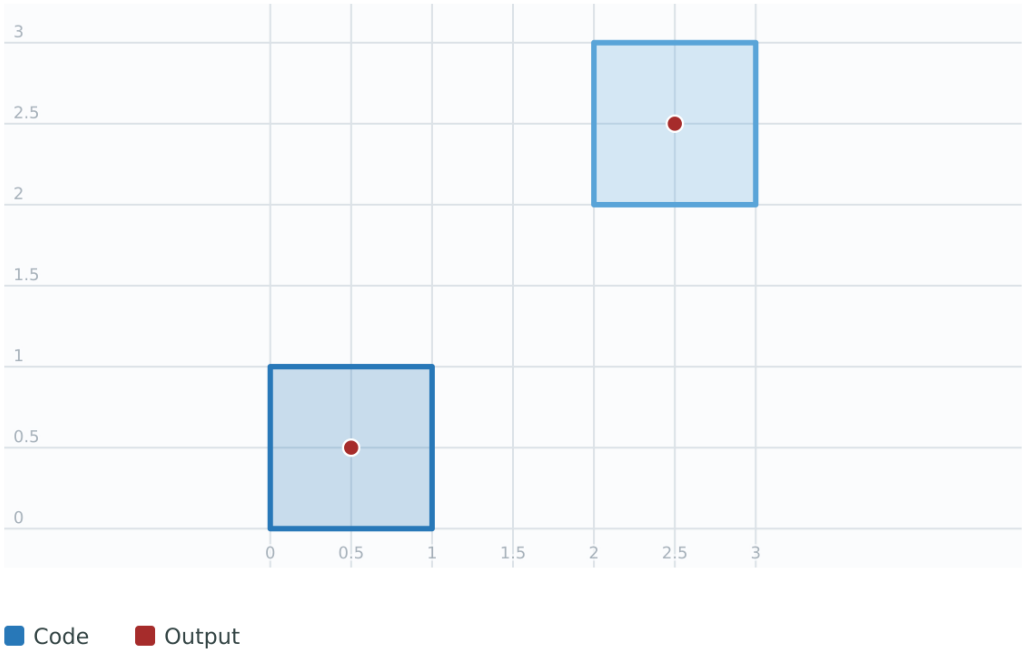
Code

```
SELECT ST_Collect(ST_Centroid(geom) ORDER BY path)
FROM ST_Dump('MULTIPOLYGON (
    ((0 0,0 1,1 1,1 0,0 0)),
    ((2 2,2 3,3 3,3 2,2 2))
  )'::geometry);
```

Output

```
MULTIPOINT((0.5 0.5),(2.5 2.5))
```

Figure



Siehe auch

ST_Collect, ST_Dump, ST_PointOnSurface, ST_GeometricMedian

7.14.4 ST_ChaikinSmoothing

ST_ChaikinSmoothing — Gibt eine geglättete Version einer Geometrie zurück, die den Chaikin-Algorithmus verwendet

Synopsis

geometry **ST_ChaikinSmoothing**(geometry geom, integer nIterations = 1, boolean preserveEndPoints = false);

Beschreibung

Glättet eine lineare oder polygonale Geometrie mit Chaikin-Algorithmus. Der Grad der Glättung wird durch den Parameter `nIterationen` gesteuert. Bei jeder Iteration wird jeder innere Scheitelpunkt durch zwei Scheitelpunkte ersetzt, die bei 1/4 der Länge der Liniensegmente vor und nach dem Scheitelpunkt liegen. Ein angemessenes Maß an Glättung wird durch 3 Iterationen erreicht; das Maximum ist auf 5 begrenzt.

Wenn `preserveEndPoints` true ist, werden die Endpunkte von Polygonringen nicht geglättet. Die Endpunkte von LineStrings werden immer beibehalten.



Anmerkung
Die Anzahl der Scheitelpunkte verdoppelt sich mit jeder Iteration, so dass die Ergebnisgeometrie viel mehr Punkte als die Eingabe haben kann. Um die Anzahl der Punkte zu reduzieren, verwenden Sie eine Vereinfachungsfunktion für das Ergebnis (siehe ST_Simplify, ST_SimplifyPreserveTopology und ST_SimplifyVW).

Das Ergebnis enthält interpolierte Werte für die Z- und M-Dimensionen, sofern vorhanden.

 Diese Funktion unterstützt 3d und lässt den Z-Index nicht fallen.

Verfügbarkeit: 2.5.0

Beispiele

Smoothing a Polygon using 1, 2 and 3 iterations.

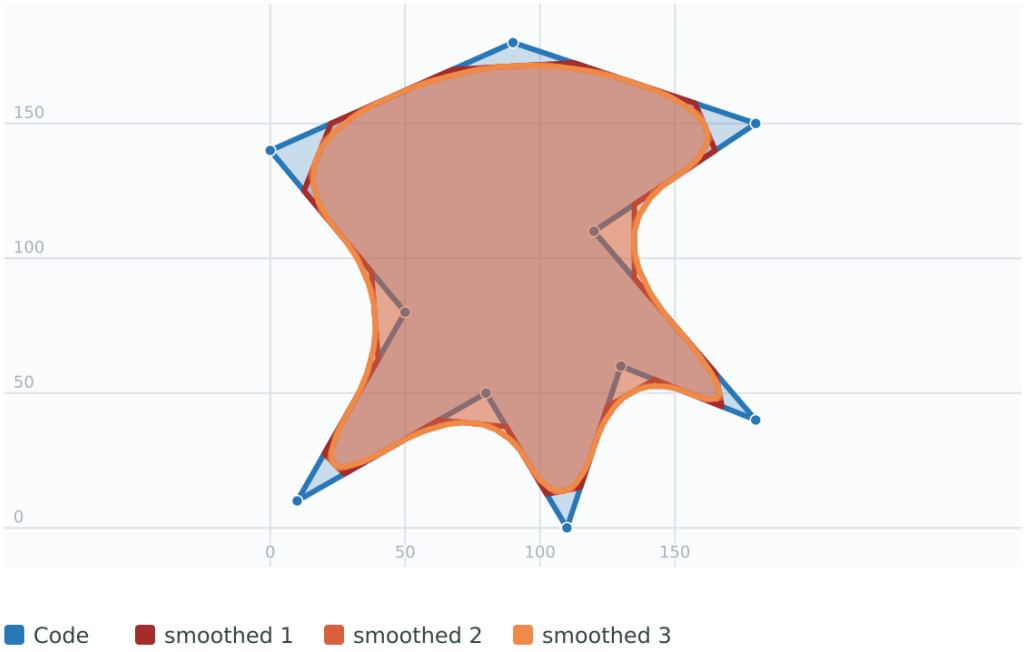
Code

```
SELECT n AS iterations, ST_ChaikinSmoothing(
    'POLYGON ((20 20,60 90,10 150,100 190,190 160,130 120,190 50,140 70,120 10,90 60,20 20))'
    , n
  ) AS smoothed
FROM generate_series(1, 3) AS n;
```

Output

```
1 | POLYGON((30 37.5,50 72.5,47.5 105,22.5 135,32.5 160,77.5 180,122.5 182.5,167.5 167.5,175 150,145 130,145 102.5,175
    67.5,177.5 55,152.5 65,135 55,125 25,112.5 22.5,97.5 47.5,72.5 50,37.5 30,30 37.5))
2 | POLYGON((35 46.25,45 63.75,49.375 80.625,48.125 96.875,41.25 112.5,28.75 127.5,25 141.25,30 153.75,43.75 165,66.25
    175,88.75 180.625,111.25 181.875,133.75 178.75,156.25 171.25,169.375 163.125,173.125 154.375,167.5 145,152.5
    135,145 123.125,145 109.375,152.5 93.75,167.5 76.25,175.625 64.375,176.875 58.125,171.25 57.5,158.75 62.5,148.125
    62.5,139.375 57.5,132.5 47.5,127.5 32.5,121.875 24.375,115.625 23.125,108.75 28.75,101.25 41.25,91.25 48.125,78.75
     49.375,63.75 45,46.25 35,35.625 31.875,31.875 35.625,35 46.25))
3 | POLYGON((37.5 50.625,42.5 59.375,46.09375 67.96875,48.28125 76.40625,49.0625 84.6875,48.4375 92.8125,46.40625
    100.78125,42.96875 108.59375,38.125 116.25,31.875 123.75,27.8125 130.9375,25.9375 137.8125,26.25 144.375,28.75
    150.625,33.4375 156.5625,40.3125 162.1875,49.375 167.5,60.625 172.5,71.875 176.40625,83.125 179.21875,94.375
    180.9375,105.625 181.5625,116.875 181.09375,128.125 179.53125,139.375 176.875,150.625 173.125,159.53125
    169.21875,166.09375 165.15625,170.3125 160.9375,172.1875 156.5625,171.71875 152.03125,168.90625 147.34375,163.75
    142.5,156.25 137.5,150.625 132.03125,146.875 126.09375,145 119.6875,145 112.8125,146.875 105.46875,150.625
    97.65625,156.25 89.375,163.75 80.625,169.53125 73.28125,173.59375 67.34375,175.9375 62.8125,176.5625
    59.6875,175.46875 57.96875,172.65625 57.65625,168.125 58.75,161.875 61.25,156.09375 62.5,150.78125 62.5,145.9375
    61.25,141.5625 58.75,137.65625 55,134.21875 50,131.25 43.75,128.75 36.25,126.09375 30.46875,123.28125
    26.40625,120.3125 24.0625,117.1875 23.4375,113.90625 24.53125,110.46875 27.34375,106.875 31.875,103.125
    38.125,98.75 42.96875,93.75 46.40625,88.125 48.4375,81.875 49.0625,75 48.28125,67.5 46.09375,59.375 42.5,50.625
    37.5,43.59375 34.21875,38.28125 32.65625,34.6875 32.8125,32.8125 34.6875,32.65625 38.28125,34.21875 43.59375,37.5
    50.625))
```

Figure



Smoothing a LineString using 1, 2 and 3 iterations.

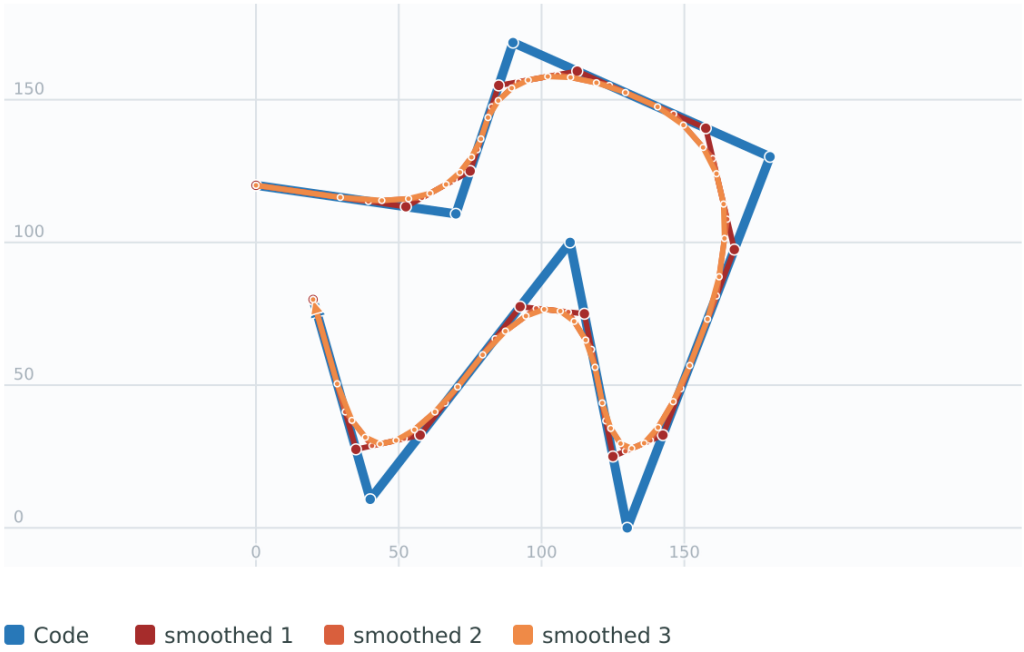
Code

```
SELECT n AS iterations, ST_ChaikinSmoothing(
    'LINESTRING (10 140,80 130,100 190,190 150,140 20,120 120,50 30,30 100)', n
  ) AS smoothed
FROM generate_series(1, 3) AS n;
```

Output

```
1 | LINESTRING(10 140,62.5 132.5,85 145,95 175,122.5 180,167.5 160,177.5 117.5,152.5 52.5,135 45,125 95,102.5 97.5,67.5 ↩
     52.5,45 47.5,30 100)
2 | LINESTRING(10 140,49.375 134.375,68.125 135.625,79.375 141.875,87.5 152.5,92.5 167.5,101.875 176.25,115.625  ↩
    178.75,133.75 175,156.25 165,170 149.375,175 128.125,171.25 101.25,158.75 68.75,148.125 50.625,139.375  ↩
    46.875,132.5 57.5,127.5 82.5,119.375 95.625,108.125 96.875,93.75 86.25,76.25 63.75,61.875 51.25,50.625 48.75,41.25 ↩
     60.625,30 100)
3 | LINESTRING(10 140,39.53125 135.78125,54.0625 134.6875,63.4375 135.3125,70.9375 137.1875,76.5625 140.3125,81.40625  ↩
    144.53125,85.46875 149.84375,88.75 156.25,91.25 163.75,94.84375 169.6875,99.53125 174.0625,105.3125  ↩
    176.875,112.1875 178.125,120.15625 177.8125,129.21875 175.9375,139.375 172.5,150.625 167.5,159.6875  ↩
    161.09375,166.5625 153.28125,171.25 144.0625,173.75 133.4375,174.0625 121.40625,172.1875 107.96875,168.125  ↩
    93.125,161.875 76.875,156.09375 64.21875,150.78125 55.15625,145.9375 49.6875,141.5625 47.8125,137.65625  ↩
    49.53125,134.21875 54.84375,131.25 63.75,128.75 76.25,125.46875 85.78125,121.40625 92.34375,116.5625  ↩
    95.9375,110.9375 96.5625,104.53125 94.21875,97.34375 88.90625,89.375 80.625,80.625 69.375,72.65625 60.625,65.46875 ↩
     54.375,59.0625 50.625,53.4375 49.375,48.28125 51.71875,43.59375 57.65625,38.4375 70.46875,30 100)
```

Figure



Smoothing a triangle shows the weakest visual change, so it is kept last.

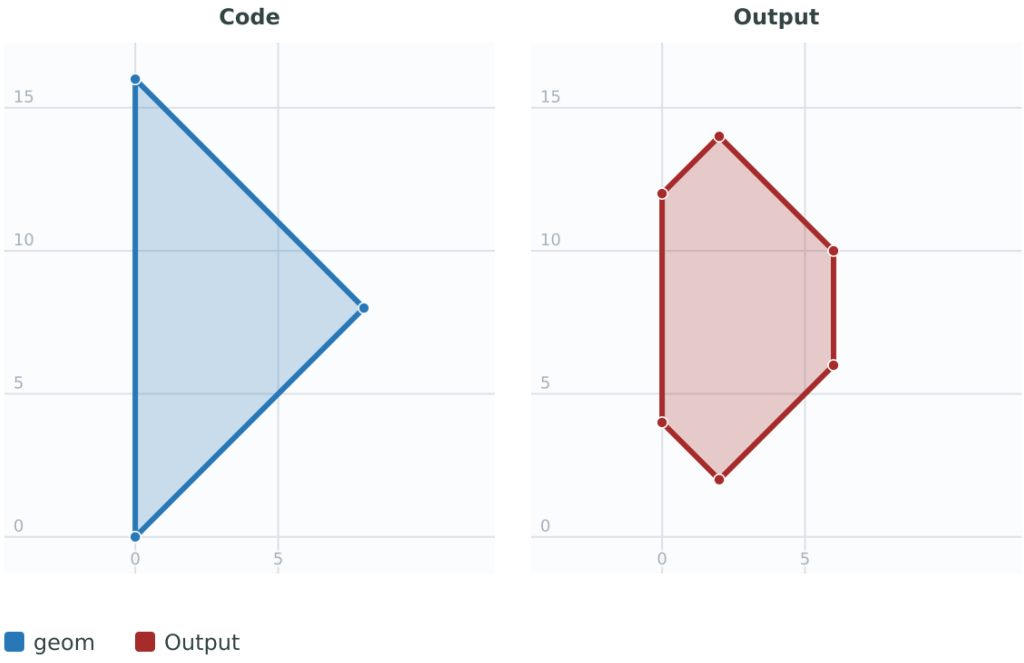
Code

```
SELECT ST_ChaikinSmoothing(geom) smoothed
FROM (SELECT  'POLYGON((0 0,8 8,0 16,0 0))'::geometry geom) AS foo;
```

Output

```
POLYGON((2 2,6 6,6 10,2 14,0 12,0 4,2 2))
```

Figure



Siehe auch

ST_Simplify, ST_SimplifyPreserveTopology, ST_SimplifyVW

7.14.5 ST_CatmullRomSmoothing

ST_CatmullRomSmoothing — Returns a smoothed version of a geometry, using the Catmull-Rom spline algorithm

Synopsis

geometry **ST_CatmullRomSmoothing**(geometry geom, integer nSegments = 5);

Beschreibung

Smoothes a linear or polygonal geometry using the Catmull-Rom spline algorithm. Unlike ST_ChaikinSmoothing, this is an *interpolating* spline: the output curve passes through every original vertex. Between each pair of consecutive original vertices, `nSegments` - 1 new intermediate points are inserted.

The `nSegments` parameter controls the density of the output. With `nSegments` = 5 (the default), each original span is divided into 5 sub-segments (inserting 4 new points per span). The minimum value is 2.

At least 4 input vertices are required to apply smoothing; geometries with fewer vertices are returned unchanged. Points and multipoints are always returned unchanged.



Anmerkung
The output vertex count grows as `1 + (N-1) * nSegments` for open lines, and `1 + N * nSegments` for closed rings, where N is the number of input vertices. For large geometries, use a simplification function on the result to reduce the number of points (see ST_Simplify, ST_SimplifyPreserveTopology and ST_SimplifyVW).

Das Ergebnis enthält interpolierte Werte für die Z- und M-Dimensionen, sofern vorhanden.

 Diese Funktion unterstützt 3d und lässt den Z-Index nicht fallen.

Availability: 3.7.0

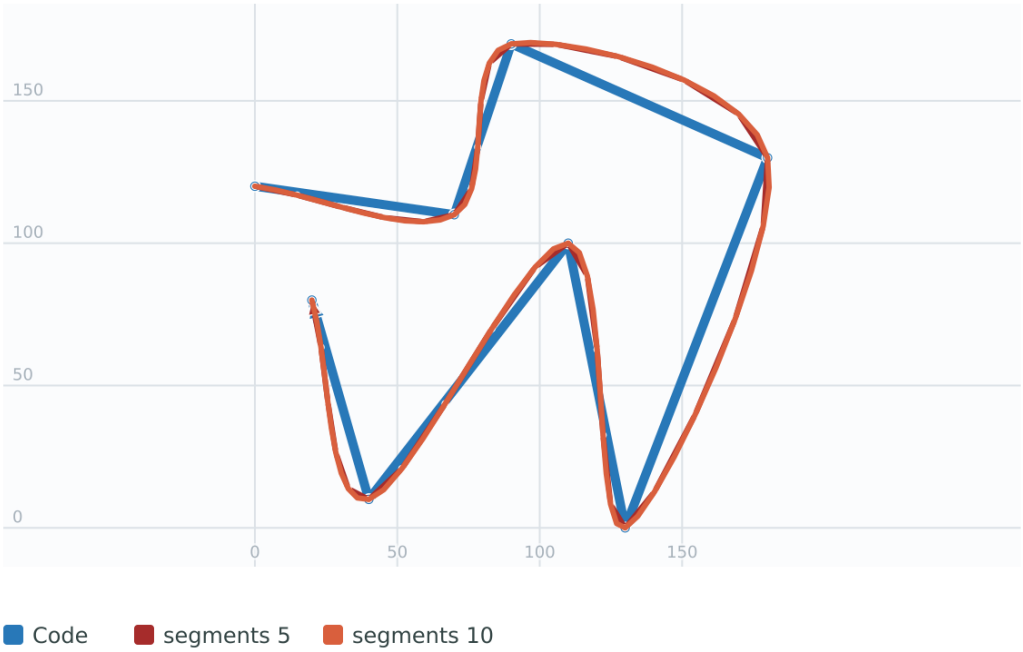
Beispiele

Smoothing a LineString using nSegments = 5 and 10.

Code

```
WITH input(geom) AS (VALUES
  ('LINESTRING (10 140,80 130,100 190,190 150,140 20,120 120,50 30,30 100)'::geometry)
)
SELECT ST_CatmullRomSmoothing(geom, 5) AS segments_5,
  ST_CatmullRomSmoothing(geom, 10) AS segments_10
FROM input;
```

Figure



Smoothing a 4-point collinear line with default nSegments=5 shows the minimal change case.

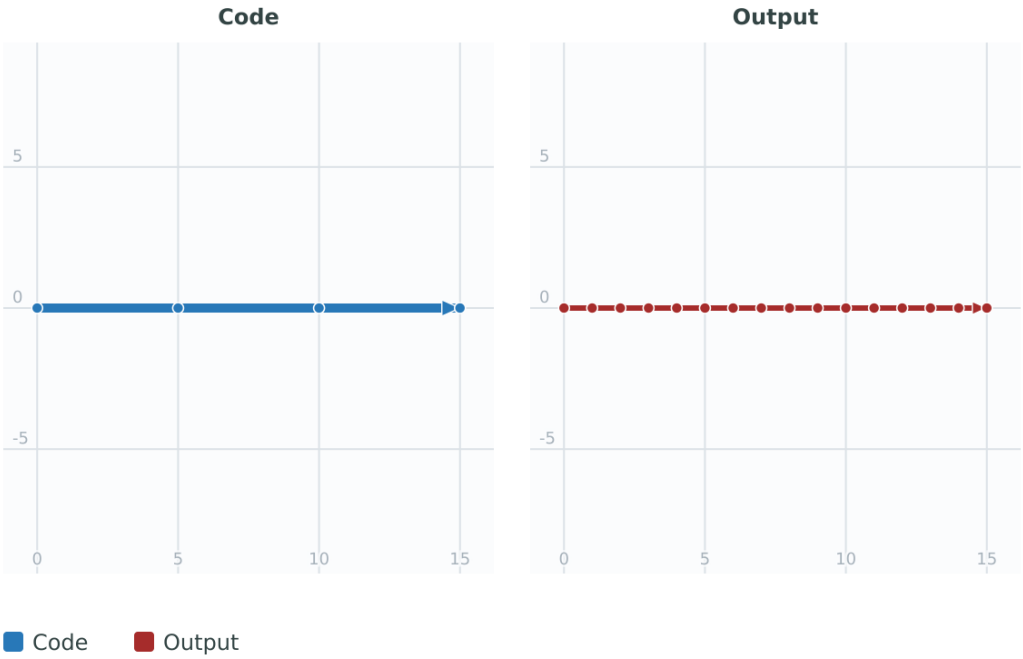
Code

```
SELECT ST_CatmullRomSmoothing('LINESTRING(0 0,5 0,10 0,15 0)');
```

Output

```
LINESTRING(0 0,1 0,2 0,3 0,4 0,5 0,6 0,7 0,8 0,9 0,10 0,11 0,12 0,13 0,14 0,15 0)
```

Figure

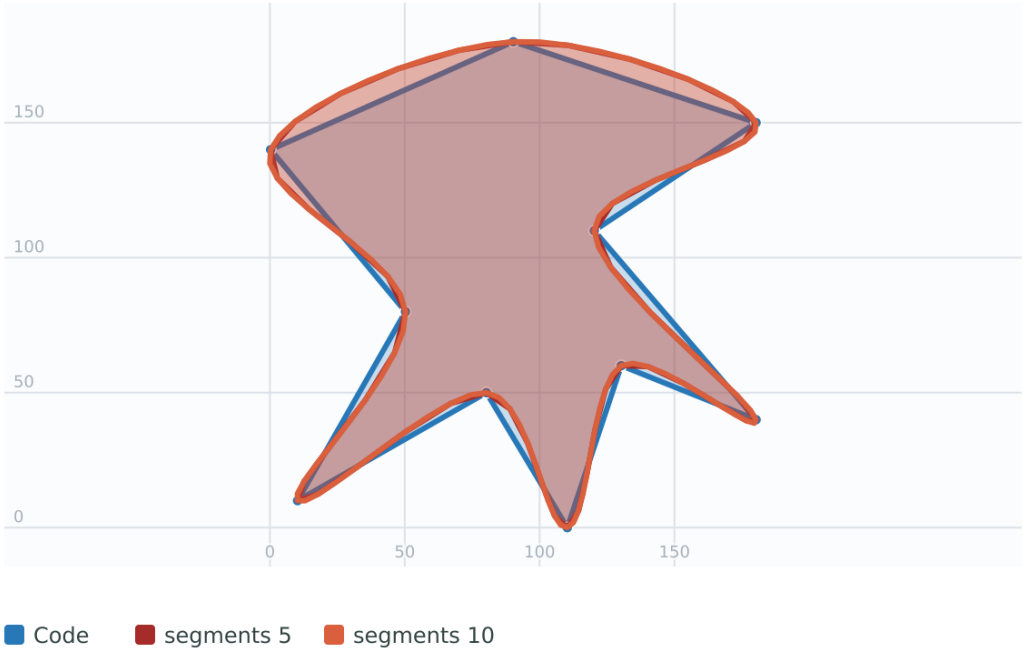


Smoothing a Polygon using nSegments = 5 and 10. This larger example is kept last.

Code

```
WITH input(geom) AS (VALUES
  ('POLYGON ((20 20,60 90,10 150,100 190,190 160,130 120,190 50,140 70,120 10,90 60,20 20))' ↩
    ::geometry)
)
SELECT ST_CatmullRomSmoothing(geom, 5) AS segments_5,
  ST_CatmullRomSmoothing(geom, 10) AS segments_10
FROM input;
```

Figure



Siehe auch

ST_ChaikinSmoothing, ST_Simplify, ST_SimplifyPreserveTopology, ST_SimplifyVW

7.14.6 ST_ConcaveHull

ST_ConcaveHull — Berechnet eine möglicherweise konkave Geometrie, die alle Eckpunkte der Eingabegeometrie enthält

Synopsis

geometry **ST_ConcaveHull**(geometry param_geom, float param_pctconvex, boolean param_allow_holes = false);

Beschreibung

Eine konkave Hülle ist eine (normalerweise) konkave Geometrie, die die Eingabe enthält und deren Scheitelpunkte eine Teilmenge der Eingabepunkte sind. Im allgemeinen Fall ist die konkave Hülle ein Polygon. Die konkave Hülle von zwei oder mehr kollinearen Punkten ist ein Zwei-Punkt-LineString. Die konkave Hülle von einem oder mehreren identischen Punkten ist ein Punkt. Das Polygon enthält keine Löcher, es sei denn, das optionale Argument `param_allow_holes` wird als true angegeben.

Man kann sich eine konkave Hülle wie eine "Schrumpfverpackung" für eine Reihe von Punkten vorstellen. Dies unterscheidet sich von der konvexen Hülle, die eher wie ein Gummiband um die Punkte gewickelt ist. Eine konkave Hülle hat im Allgemeinen eine kleinere Fläche und stellt eine natürlichere Begrenzung für die Eingabepunkte dar.

Die `param_pctconvex` steuert die Konkavität der berechneten Hülle. Ein Wert von 1 ergibt die konvexe Hülle. Werte zwischen 1 und 0 erzeugen Hüllen mit zunehmender Konkavität. Ein Wert von 0 ergibt eine Hülle mit maximaler Konkavität (aber immer noch ein einzelnes Polygon). Die Wahl eines geeigneten Wertes hängt von der Art der Eingabedaten ab, aber häufig führen Werte zwischen 0,3 und 0,1 zu vernünftigen Ergebnissen.



Anmerkung
Technisch gesehen bestimmt die `param_pctconvex` eine Länge als Bruchteil der Differenz zwischen der längsten und der kürzesten Kante in der Delaunay-Triangulation der Eingabepunkte. Kanten, die länger als diese Länge sind, werden aus der Triangulation "abgetragen". Die verbleibenden Dreiecke bilden die konkave Hülle.

Bei punkt- und linienförmigen Eingaben umschließt die Hülle alle Punkte der Eingaben. Bei polygonalen Eingaben umschließt die Hülle alle Punkte der Eingabe *und auch* alle von der Eingabe abgedeckten Flächen. Wenn Sie eine punktweise Hülle einer polygonalen Eingabe wünschen, konvertieren Sie sie zunächst mit ST_Points in Punkte.

This is not an aggregate function. To compute the concave hull of a set of geometries use ST_Collect (e.g. `ST_ConcaveHull(ST_Col ), 0.80)`.

Verfügbarkeit: 2.0.0

Verbessert: 3.3.0, native GEOS-Implementierung aktiviert für GEOS 3.11+

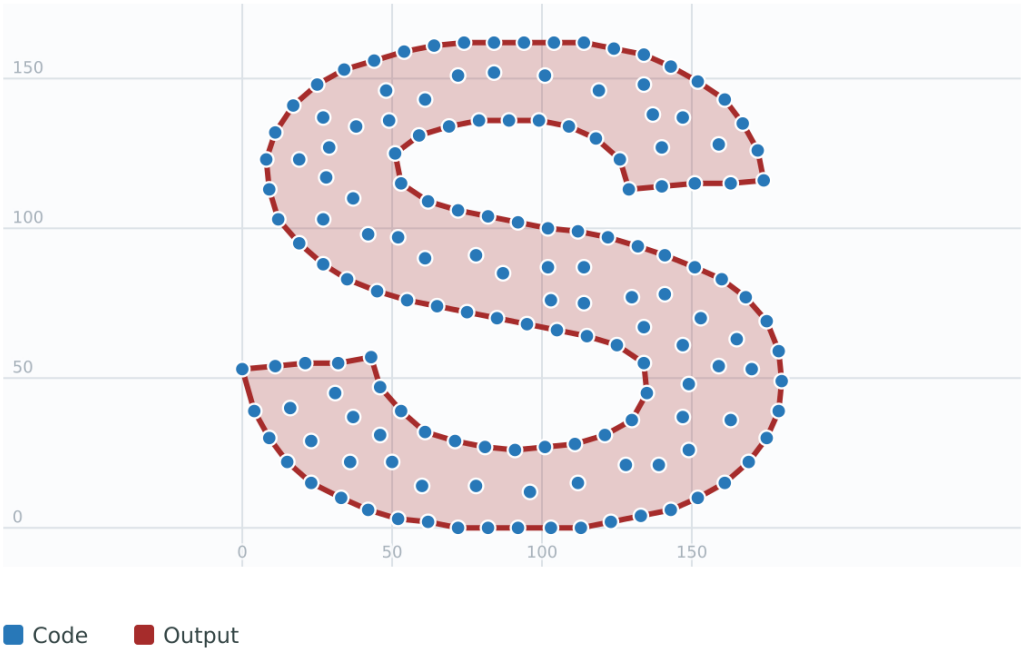
Beispiele

Concave hull of a MultiPoint.

Code

```
SELECT ST_ConcaveHull('MULTIPOINT ((10 72),(53 76),(56 66),(63 58),(71 51),(81 48),(91 46) ↩
    ,(101 45),(111 46),(121 47),(131 50),(140 55),(145 64),(144 74),(135 80),(125 83),(115  ↩
    85),(105 87),(95 89),(85 91),(75 93),(65 95),(55 98),(45 102),(37 107),(29 114),(22 122) ↩
    ,(19 132),(18 142),(21 151),(27 160),(35 167),(44 172),(54 175),(64 178),(74 180),(84  ↩
    181),(94 181),(104 181),(114 181),(124 181),(134 179),(144 177),(153 173),(162 168),(171 ↩
     162),(177 154),(182 145),(184 135),(139 132),(136 142),(128 149),(119 153),(109 155) ↩
    ,(99 155),(89 155),(79 153),(69 150),(61 144),(63 134),(72 128),(82 125),(92 123),(102  ↩
    121),(112 119),(122 118),(132 116),(142 113),(151 110),(161 106),(170 102),(178 96),(185 ↩
     88),(189 78),(190 68),(189 58),(185 49),(179 41),(171 34),(162 29),(153 25),(143 23) ↩
    ,(133 21),(123 19),(113 19),(102 19),(92 19),(82 19),(72 21),(62 22),(52 25),(43 29),(33 ↩
     34),(25 41),(19 49),(14 58),(21 73),(31 74),(42 74),(173 134),(161 134),(150 133),(97  ↩
    104),(52 117),(157 156),(94 171),(112 106),(169 73),(58 165),(149 40),(70 33),(147 157) ↩
    ,(48 153),(140 96),(47 129),(173 55),(144 86),(159 67),(150 146),(38 136),(111 170),(124 ↩
     94),(26 59),(60 41),(71 162),(41 64),(88 110),(122 34),(151 97),(157 56),(39 146),(88  ↩
    33),(159 45),(47 56),(138 40),(129 165),(33 48),(106 31),(169 147),(37 122),(71 109) ↩
    ,(163 89),(37 156),(82 170),(180 72),(29 142),(46 41),(59 155),(124 106),(157 80),(175  ↩
    82),(56 50),(62 116),(113 95),(144 167))',
         0.1 );
```

Figure

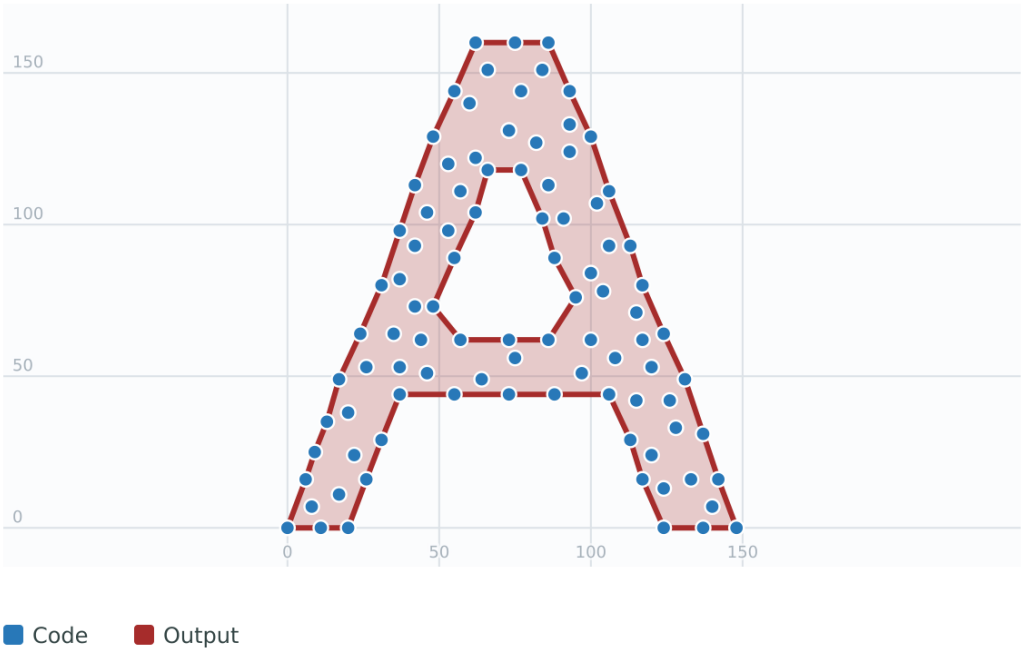


Concave hull of a MultiPoint, allowing holes.

Code

```
SELECT ST_ConcaveHull('MULTIPOINT ((132 64),(114 64),(99 64),(81 64),(63 64),(57 49),(52 36) ↩
    ,(46 20),(37 20),(26 20),(32 36),(39 55),(43 69),(50 84),(57 100),(63 118),(68 133),(74  ↩
    149),(81 164),(88 180),(101 180),(112 180),(119 164),(126 149),(132 131),(139 113),(143  ↩
    100),(150 84),(157 69),(163 51),(168 36),(174 20),(163 20),(150 20),(143 36),(139 49) ↩
    ,(132 64),(99 151),(92 138),(88 124),(81 109),(74 93),(70 82),(83 82),(99 82),(112 82) ↩
    ,(126 82),(121 96),(114 109),(110 122),(103 138),(99 151),(34 27),(43 31),(48 44),(46  ↩
    58),(52 73),(63 73),(61 84),(72 71),(90 69),(101 76),(123 71),(141 62),(166 27),(150 33) ↩
    ,(159 36),(146 44),(154 53),(152 62),(146 73),(134 76),(143 82),(141 91),(130 98),(126  ↩
    104),(132 113),(128 127),(117 122),(112 133),(119 144),(108 147),(119 153),(110 171) ↩
    ,(103 164),(92 171),(86 160),(88 142),(79 140),(72 124),(83 131),(79 118),(68 113),(63  ↩
    102),(68 93),(35 45))',
         0.15, true );
```

Figure



Vergleich einer konkaven Hülle eines Polygons mit der konkaven Hülle der Punkte, aus denen es besteht. Die Hülle respektiert die Begrenzung des Polygons, während die punktbasierte Hülle dies nicht tut.

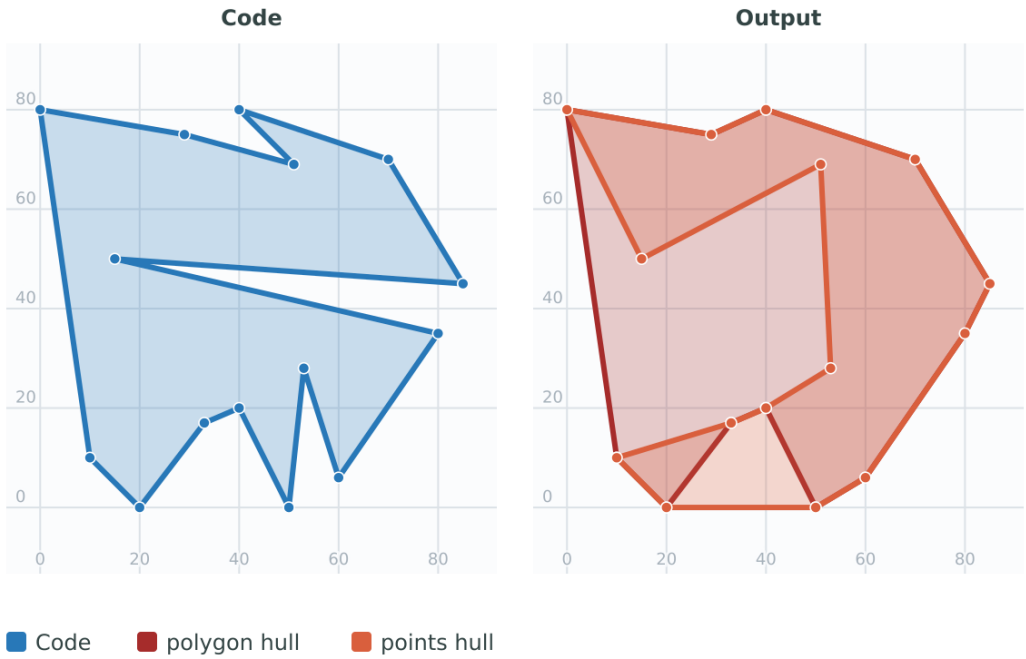
Code

```
WITH data(geom) AS (VALUES
    ('POLYGON ((10 90,39 85,61 79,50 90,80 80,95 55,25 60,90 45,70 16,63 38,60 10,50 30,43 ↩
     27,30 10,20 20,10 90))'::geometry)
)
SELECT ST_ConcaveHull(geom, 0.1) AS polygon_hull,
       ST_ConcaveHull(ST_Points(geom), 0.1) AS points_hull
     FROM data;
```

Output

```
POLYGON((60 10,50 30,43 27,30 10,20 20,10 90,39 85,50 90,80 80,95 55,90 45,70 16,60 10)) | POLYGON((25 60,10 90,39 ↩
     85,50 90,80 80,95 55,90 45,70 16,60 10,30 10,20 20,43 27,50 30,63 38,61 79,25 60))
```

Figure



Verwendung mit ST_Collect zur Berechnung der konvexen Hüllen einer Geometrie.

This example estimates an infected area from point observations.

Code

```
SELECT disease_type,
    ST_ConcaveHull(ST_Collect(obs_pnt), 0.3 ) AS geom
  FROM disease_obs
  GROUP BY disease_type;
```

Siehe auch

ST_ConvexHull, ST_Collect, CG_AlphaShape, CG_OptimalAlphaShape

7.14.7 ST_ConvexHull

ST_ConvexHull — Berechnet die konvexe Hülle einer Geometrie.

Synopsis

geometry **ST_ConvexHull**(geometry geomA);

Beschreibung

Berechnet die konvexe Hülle einer Geometrie. Die konvexe Hülle stellt die kleinste konvexe Geometrie dar, welche die gesamte Geometrie einschließt.

Man kann sich die konvexe Hülle als die Geometrie vorstellen, die man erhält, wenn man ein Gummiband um eine Reihe von Geometrien wickelt. Dies unterscheidet sich von einer konkaven Hülle, die dem "Einschweißen" der Geometrien entspricht. Eine konvexe Hülle wird häufig verwendet, um ein betroffenes Gebiet auf der Grundlage einer Reihe von Punktbeobachtungen zu bestimmen.

Im allgemeinen Fall ist die konvexe Hülle ein Polygon. Die konvexe Hülle von zwei oder mehr kollinearen Punkten ist ein Zweipunkt-Linienzug (LineString). Die konvexe Hülle von einem oder mehreren identischen Punkten ist ein Punkt.

Dies ist keine Aggregatfunktion. Um die konvexe Hülle einer Menge von Geometrien zu berechnen, verwenden Sie ST_Collect, um sie zu einer Geometriesammlung zu aggregieren (z. B. `ST_ConvexHull(ST_Collect(geom))`.

Wird durch das GEOS Modul ausgeführt

Diese Methode implementiert die OGC Simple Features Implementation Specification for SQL 1.1. s2.1.1.3

Diese Methode setzt die SQL/MM-Spezifikation um. SQL-MM IEC 13249-3: 5.1.16

Diese Funktion unterstützt 3d und lässt den Z-Index nicht fallen.

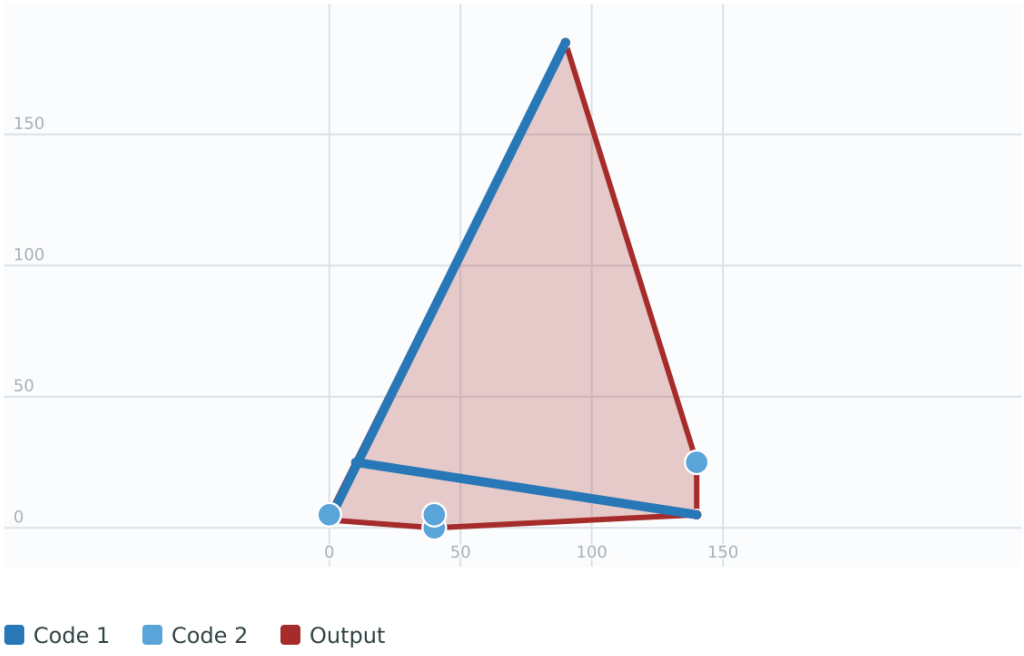
Beispiele

Compute the convex hull of a MultiLineString and a MultiPoint.

Code

```
SELECT ST_ConvexHull(ST_Collect(ST_GeomFromText('MULTILINESTRING((100 190,10 8),(150 10,20 ↩
    30))'),
            ST_GeomFromText('MULTIPOINT(50 5,150 30,50 10,10 10)')
            ));
```

Figure



Verwendung mit ST_Collect zur Berechnung der konvexen Hüllen einer Geometrie.

This example returns an estimated infected area from point observations.

Code

```
SELECT d.disease_type,
    ST_ConvexHull(ST_Collect(d.geom)) As geom
    FROM disease_obs As d
    GROUP BY d.disease_type;
```

Siehe auch

ST_Collect, ST_ConcaveHull, ST_MinimumBoundingCircle

7.14.8 ST_DelaunayTriangles

ST_DelaunayTriangles — Gibt die Delaunay-Triangulation der Scheitelpunkte einer Geometrie zurück.

Synopsis

geometry **ST_DelaunayTriangles**(geometry g1, float tolerance = 0.0, int4 flags = 0);

Beschreibung

Berechnet die Delaunay-Triangulation der Scheitelpunkte der Eingabegeometrie. Die optionale `Toleranz` kann verwendet werden, um nahegelegene Eingabepunkte zusammenzufassen, was in einigen Situationen die Robustheit verbessert. Die Ergebnisgeometrie wird durch die konvexe Hülle der Eingabescheitelpunkte begrenzt. Die Darstellung der Ergebnisgeometrie wird durch den Code `flags` bestimmt:

- `0` - eine GEOMETRIESAMMLUNG von dreieckigen POLYGONEN (Standard)
- `1` - ein MULTILINESTRING der Kanten der Triangulation
- `2` - Eine TIN der Triangulation

Wird vom GEOS Modul ausgeführt

Verfügbarkeit: 2.1.0

Diese Funktion unterstützt 3d und lässt den Z-Index nicht fallen.

Diese Funktion unterstützt Dreiecke und dreieckige unregelmäßige Netzoberflächen (TIN).

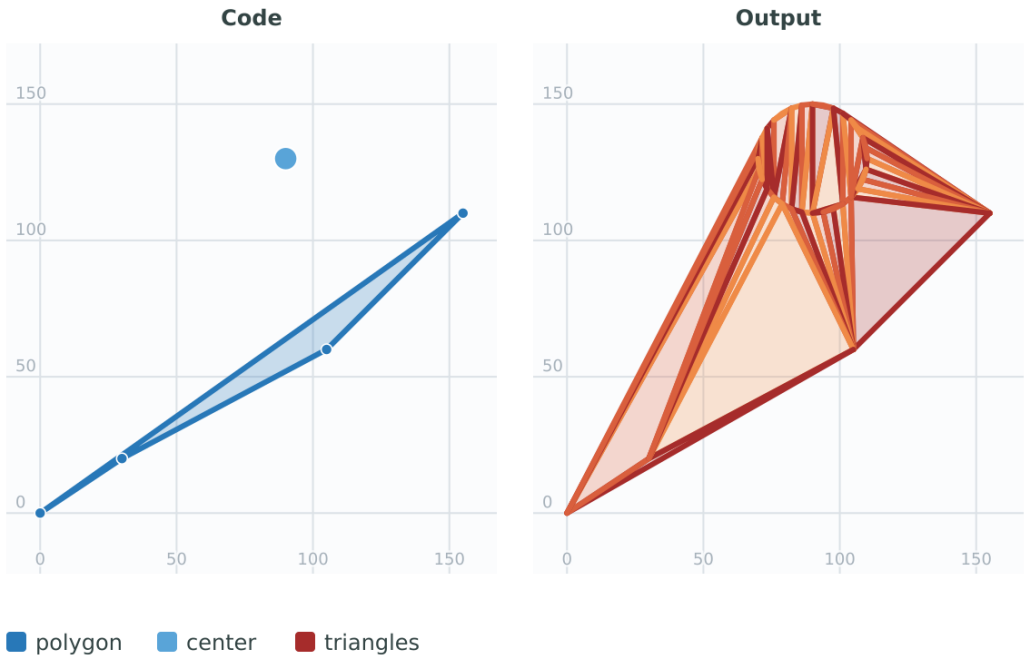
Beispiele

Triangulation of the vertices of a polygon and a buffered point.

Code

```
WITH data AS (
  SELECT polygon, ST_Buffer(center, 20) AS point_buffer
  FROM (SELECT
    'POLYGON((175 150,20 40,50 60,125 100,175 150))'::geometry AS polygon,
    'POINT(110 170)'::geometry AS center
  ) AS input
)
SELECT ST_DelaunayTriangles(ST_Union(polygon, point_buffer)) AS triangles
FROM data;
```

Figure

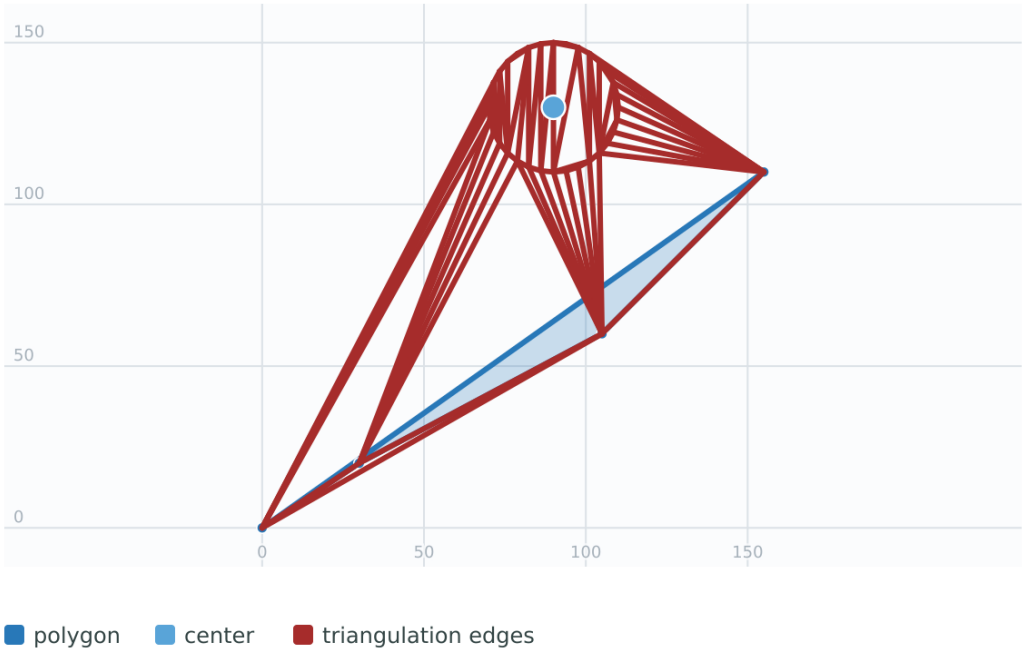


Triangulation edges returned as a `MULTILINESTRING`.

Code

```
WITH data AS (
  SELECT polygon, ST_Buffer(center, 20) AS point_buffer
  FROM (SELECT
    'POLYGON((175 150,20 40,50 60,125 100,175 150))'::geometry AS polygon,
    'POINT(110 170)'::geometry AS center
  ) AS input
)
SELECT ST_DelaunayTriangles(ST_Union(polygon, point_buffer), 0.001, 1) AS ↩
    triangulation_edges
FROM data;
```

Figure

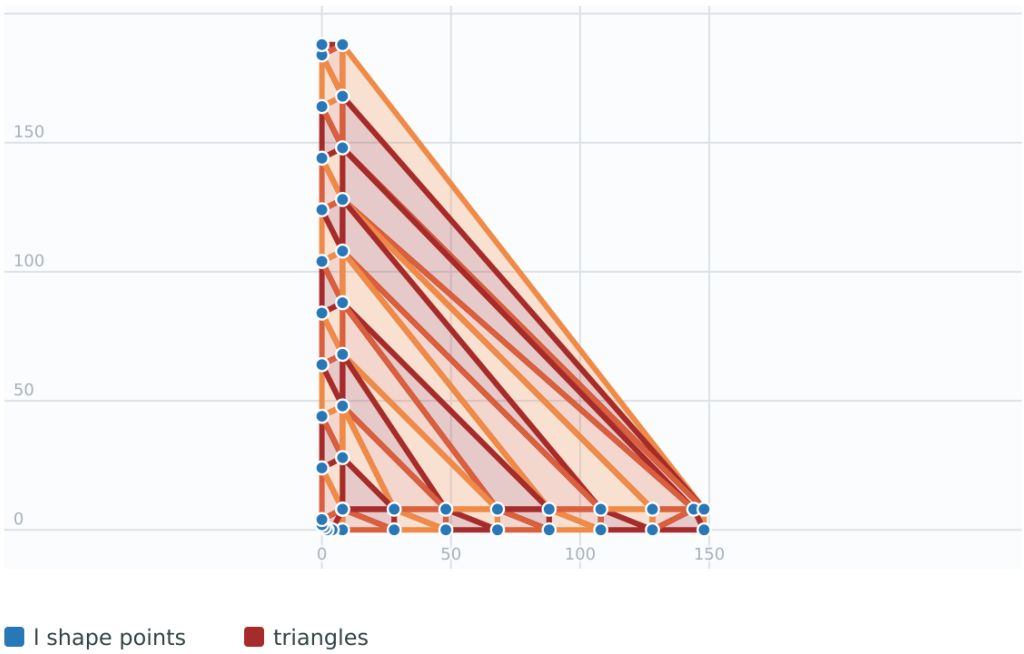


Triangulation of 42 points forming an L shape.

Code

```
WITH data AS (
  SELECT 'MULTIPOINT(14 14,34 14,54 14,74 14,94 14,114 14,134 14,
    150 14,154 14,154 6,134 6,114 6,94 6,74 6,54 6,34 6,
    14 6,10 6,8 6,7 7,6 8,6 10,6 30,6 50,6 70,6 90,6 110,6 130,
    6 150,6 170,6 190,6 194,14 194,14 174,14 154,14 134,14 114,
    14 94,14 74,14 54,14 34,14 14)'::geometry AS l_shape_points
)
SELECT ST_DelaunayTriangles(l_shape_points) AS triangles
FROM data;
```

Figure



Beispiel mit Scheitelpunkten mit Z-Werten.

Code

```
SELECT ST_DelaunayTriangles(
  'MULTIPOINT Z(14 14 10,150 14 100,34 6 25,20 10 150)'::geometry);
```

Output

```
GEOMETRYCOLLECTION Z (POLYGON Z ((14 14 10,20 10 150,34 6 25,14 14 10)),POLYGON Z ((14 14 ↩
    10,34 6 25,150 14 100,14 14 10)))
```

Siehe auch

ST_VoronoiPolygons, ST_TriangulatePolygon, CG_ConstrainedDelaunayTriangles, ST_VoronoiLines, ST_ConvexHull

7.14.9 ST_FilterByM

ST_FilterByM — Entfernt Scheitelpunkte basierend auf ihrem M-Wert

Synopsis

geometry **ST_FilterByM**(geometry geom, double precision min, double precision max = null, boolean returnM = false);

Beschreibung

Filtert Scheitelpunkte auf der Grundlage ihres M-Werts heraus. Gibt eine Geometrie zurück, die nur Scheitelpunkte enthält, deren M-Wert größer oder gleich dem Min-Wert und kleiner oder gleich dem Max-Wert ist. Wenn das Argument max-value weggelassen wird, wird nur der min-Wert berücksichtigt. Wenn das vierte Argument weggelassen wird, wird der M-Wert nicht in der resultierenden Geometrie enthalten sein. Wenn die resultierende Geometrie zu wenig Scheitelpunkte für ihren Geometrietyp

hat, wird eine leere Geometrie zurückgegeben. In einer Geometriesammlung werden Geometrien, die nicht genügend Punkte haben, einfach stillschweigend ausgelassen.

Diese Funktion ist hauptsächlich für die Verwendung in Verbindung mit ST_SetEffectiveArea gedacht. ST_EffectiveArea legt die effektive Fläche eines Scheitelpunkts in seinem m-Wert fest. Mit ST_FilterByM ist es dann möglich, eine vereinfachte Version der Geometrie ohne jegliche Berechnungen zu erhalten, einfach durch Filtern



Anmerkung
Es gibt einen Unterschied zwischen ST_SimplifyVW und ST_FilterByM, was ST_SimplifyVW zurückgibt, wenn nicht genügend Punkte die Kriterien erfüllen. ST_SimplifyVW gibt die Geometrie mit genügend Punkten zurück, während ST_FilterByM eine leere Geometrie zurückgibt.



Anmerkung
Beachten Sie, dass die zurückgegebene Geometrie ungültig sein kann



Anmerkung
Diese Funktion liefert alle Dimensionen, einschließlich der Z- und M-Werte

Verfügbarkeit: 2.5.0

Beispiele

Eine gefilterte Linie

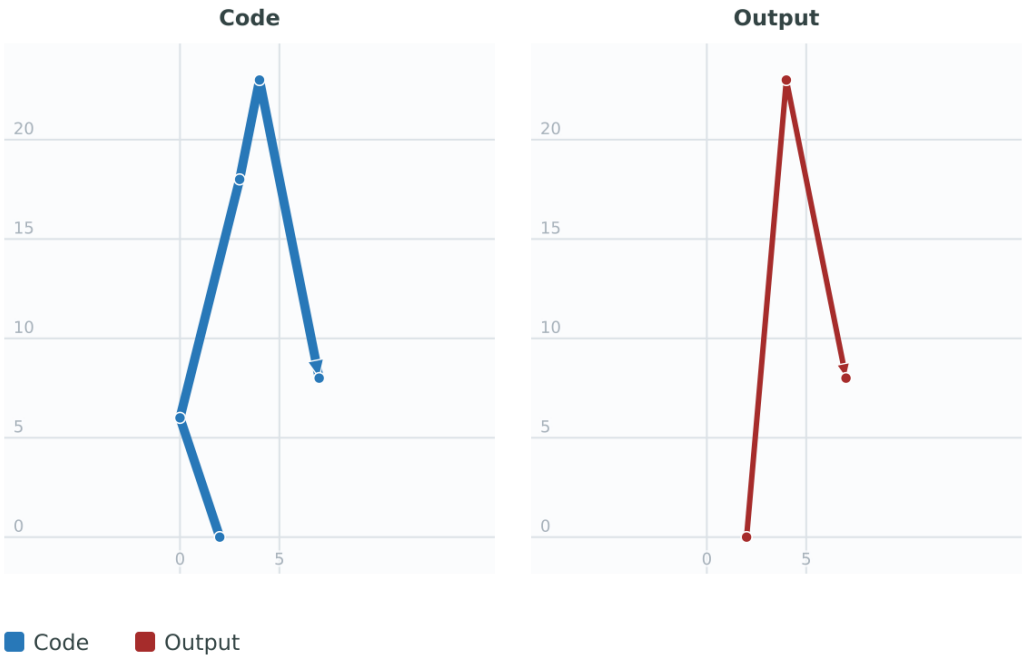
Code

```
SELECT ST_FilterByM(geom, 30) simplified
FROM (SELECT  ST_SetEffectiveArea('LINESTRING(5 2,3 8,6 20,7 25,10 10)'::geometry) geom) As ↩
    foo;
```

Output

```
LINESTRING(5 2,7 25,10 10)
```

Figure



Siehe auch

ST_SetEffectiveArea, ST_SimplifyVW

7.14.10 ST_GeneratePoints

ST_GeneratePoints — Erzeugt einen Multipunkt aus zufälligen Punkten, die in einem Polygon oder MultiPolygon enthalten sind.

Synopsis

geometry **ST_GeneratePoints**(geometry g, integer npoints, integer seed = 0);

Beschreibung

ST_GeneratePoints erzeugt einen Multipunkt, der aus einer bestimmten Anzahl von Pseudo-Zufallspunkten besteht, die innerhalb des Eingabebereichs liegen. Der optionale Wert `seed` wird verwendet, um eine deterministische Folge von Punkten zu erzeugen, und muss größer als Null sein.

Verfügbarkeit: 2.3.0

Erweiterung: mit 3.0.0 wurde das Argument "seed" hinzugefügt

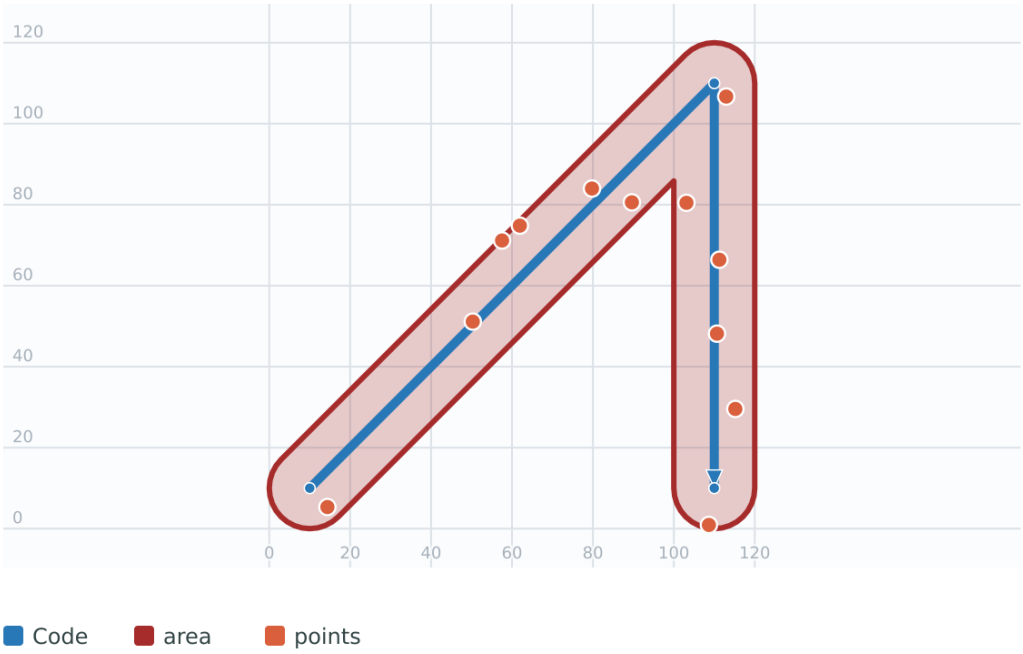
Beispiele

Generate 12 points inside a buffered LineString using the deterministic seed 1996.

Code

```
WITH source AS (
  SELECT ST_Buffer(ST_GeomFromText('LINESTRING(50 50,150 150,150 50)'),
           10, 'endcap=round join=round') AS geom
)
SELECT geom AS area,
       ST_GeneratePoints(geom, 12, 1996) AS points
FROM source;
```

Figure



Bei einer Tabelle mit Polygonen s sind 12 einzelne Punkte pro Polygon zu ermitteln. Die Ergebnisse sind bei jedem Durchlauf anders.

Code

```
SELECT s.id, dp.path[1] AS pt_id, dp.geom
FROM s, ST_DumpPoints(ST_GeneratePoints(s.geom, 12)) AS dp;
```

Generate dot-density points from a polygon table, using the absolute value of an attribute as the number of points and keeping the sign of the attribute for styling.

Code

```
SELECT z.id,
       CASE WHEN z.count_delta < 0 THEN -1 ELSE 1 END AS sign,
       dp.path[1] AS pt_id,
       dp.geom
FROM zones AS z
CROSS JOIN LATERAL ST_DumpPoints(
    ST_GeneratePoints(z.geom, abs(z.count_delta)::integer, z.id + 1)
) AS dp
WHERE z.count_delta <> 0;
```

Siehe auch

ST_DumpPoints

7.14.11 ST_GeometricMedian

ST_GeometricMedian — Gibt den geometrischen Median eines Mehrfachpunktes zurück.

Synopsis

geometry **ST_GeometricMedian** (geometry geom, float8 tolerance = NULL, int max_iter = 10000, boolean fail_if_not_converged = false);

Beschreibung

Berechnet den ungefähren geometrischen Median einer MultiPoint-Geometrie unter Verwendung des Weiszfeld-Algorithmus. Der geometrische Median ist der Punkt, der die Summe der Abstände zu den Eingabepunkten minimiert. Er liefert ein Zentralitätsmaß, das weniger empfindlich auf Ausreißerpunkte reagiert als der Schwerpunkt (Center of Mass).

Der Algorithmus iteriert, bis die Abstandsänderung zwischen aufeinanderfolgenden Iterationen kleiner ist als der angegebene `tolerance` Parameter. Wenn diese Bedingung nach `max_iterations` Iterationen nicht erfüllt ist, erzeugt die Funktion einen Fehler und wird beendet, es sei denn, `fail_if_not_converged` ist auf `false` (die Voreinstellung) gesetzt.

Wenn das Argument `tolerance` nicht angegeben wird, wird der Toleranzwert auf der Grundlage der Ausdehnung der Eingabegeometrie berechnet.

Falls vorhanden, werden die Werte der Eingabepunkte M als deren relative Gewichte interpretiert.

Verfügbarkeit: 2.3.0

Erweiterung: 2.5.0 Unterstützung für M zur Gewichtung nach Punkten.

Diese Funktion unterstützt 3d und lässt den Z-Index nicht fallen.

Diese Funktion unterstützt M-Koordinaten.

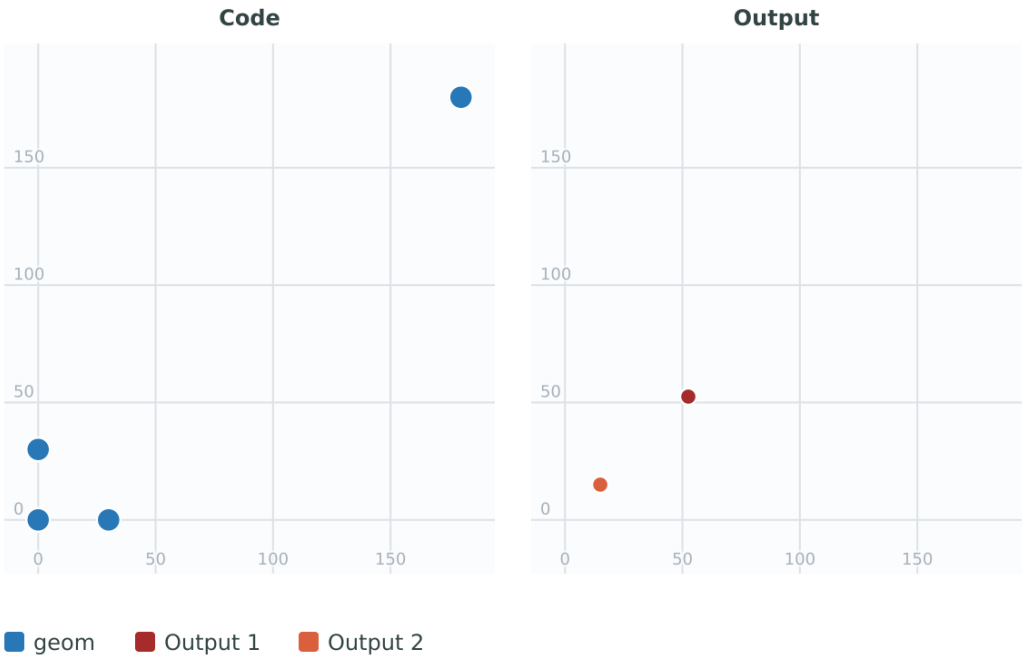
Beispiele

Compare the geometric median and centroid of a MultiPoint.

Code

```
WITH test AS (
SELECT 'MULTIPOINT((10 10),(10 40),(40 10),(190 190))'::geometry geom)
SELECT
  ST_Centroid(geom) centroid,
  ST_GeometricMedian(geom) median
FROM test;
```

Figure



Siehe auch

ST_Centroid

7.14.12 ST_LineMerge

ST_LineMerge — Gibt die Linien zurück, die durch das Zusammenfügen eines MultiLineString gebildet werden.

Synopsis

geometry **ST_LineMerge**(geometry amultilinestring);
geometry **ST_LineMerge**(geometry amultilinestring, boolean directed);

Beschreibung

Gibt einen LineString oder MultiLineString zurück, der durch Zusammenfügen der Linienelemente eines MultiLineString gebildet wird. Linien werden an ihren Endpunkten an 2-Wege-Kreuzungen verbunden. Linien werden nicht über Schnittpunkte mit 3 oder mehr Richtungen verbunden.

Wenn **directed** TRUE ist, dann ändert ST_LineMerge nicht die Reihenfolge der Punkte innerhalb von LineStrings, so dass Linien mit entgegengesetzten Richtungen nicht zusammengeführt werden



Anmerkung
Nur mit MultiLineString/LineStrings verwenden. Andere Geometrietypen geben eine leere GeometryCollection zurück

Wird vom GEOS Modul ausgeführt

Verbessert: 3.3.0 akzeptiert einen gerichteten Parameter.

Erfordert GEOS >= 3.11.0 zur Verwendung des gerichteten Parameters.

Verfügbarkeit: 1.1.0



Warnung
Diese Funktion entfernt die Dimension M.

Beispiele

Cardinality 2

Nodes with cardinality 2 are merged away.

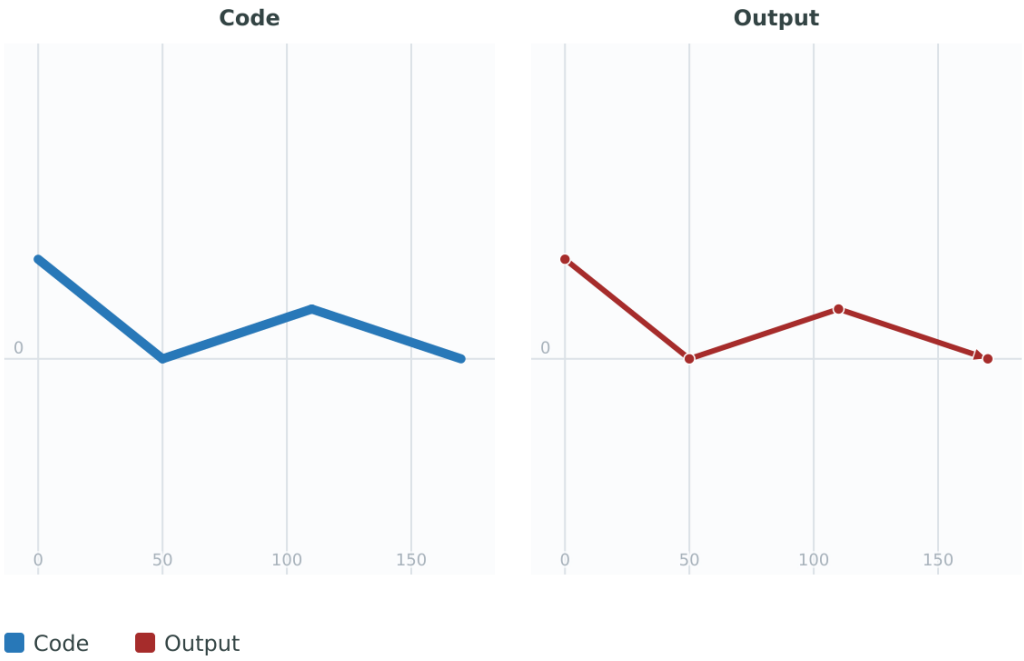
Code

```
SELECT ST_LineMerge('MULTILINESTRING((10 160,60 120),(120 140,60 120),(120 140,180 120))'
);
```

Output

```
LINESTRING(10 160,60 120,120 140,180 120)
```

Figure



Cardinality 3 or More

Linien werden nicht über Schnittpunkte mit dem Grad > 2 zusammengeführt.

Code

```
SELECT ST_LineMerge('MULTILINESTRING((10 160,60 120),(120 140,60 120),(120 140,180 120),(100 ↩
    180,120 140))'
);
```

Output

```
MULTILINESTRING((10 160,60 120,120 140),(100 180,120 140),(120 140,180 120))
```

Figure



Non-Touching Lines

Ist das Zusammenführen aufgrund von sich nicht berührenden Zeilen nicht möglich, wird der ursprüngliche MultiLineString zurückgegeben.

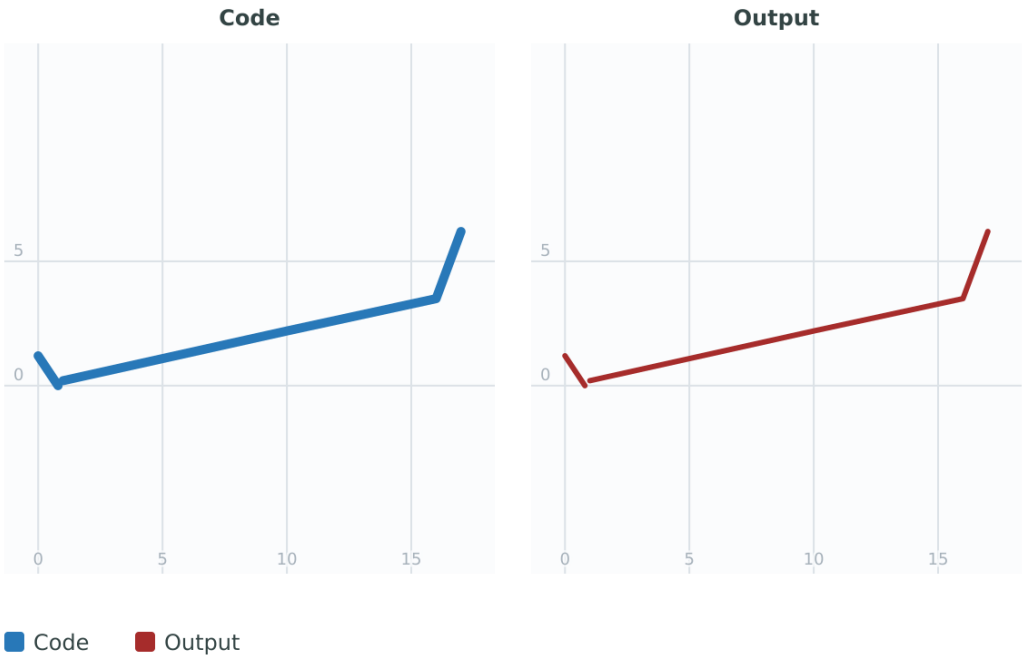
Code

```
SELECT ST_LineMerge('MULTILINESTRING((-29 -27,-30 -29.7,-36 -31,-45 -33),(-45.2 -33.2,-46 ↩
    -32))'
);
```

Output

```
MULTILINESTRING((-45.2 -33.2,-46 -32),(-29 -27,-30 -29.7,-36 -31,-45 -33))
```

Figure



Directed Parameter

Lines with opposite directions are not merged if *directed* is TRUE.

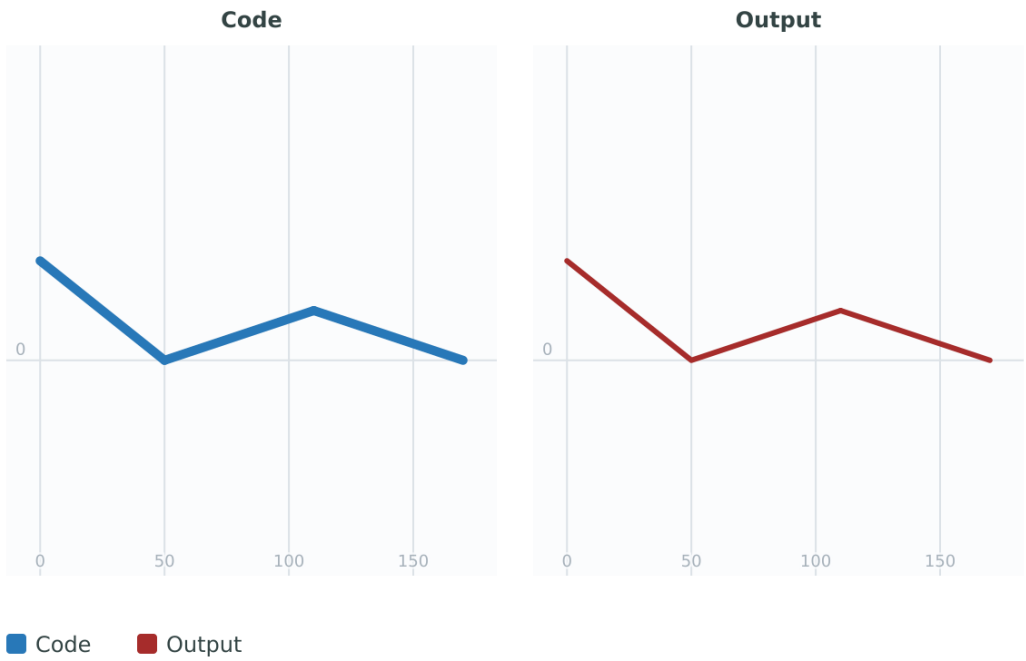
Code

```
SELECT ST_LineMerge('MULTILINESTRING((60 30,10 70),(120 50,60 30),(120 50,180 30))',
TRUE);
```

Output

```
MULTILINESTRING((120 50,60 30,10 70),(120 50,180 30))
```

Figure



Z-dimension Handling

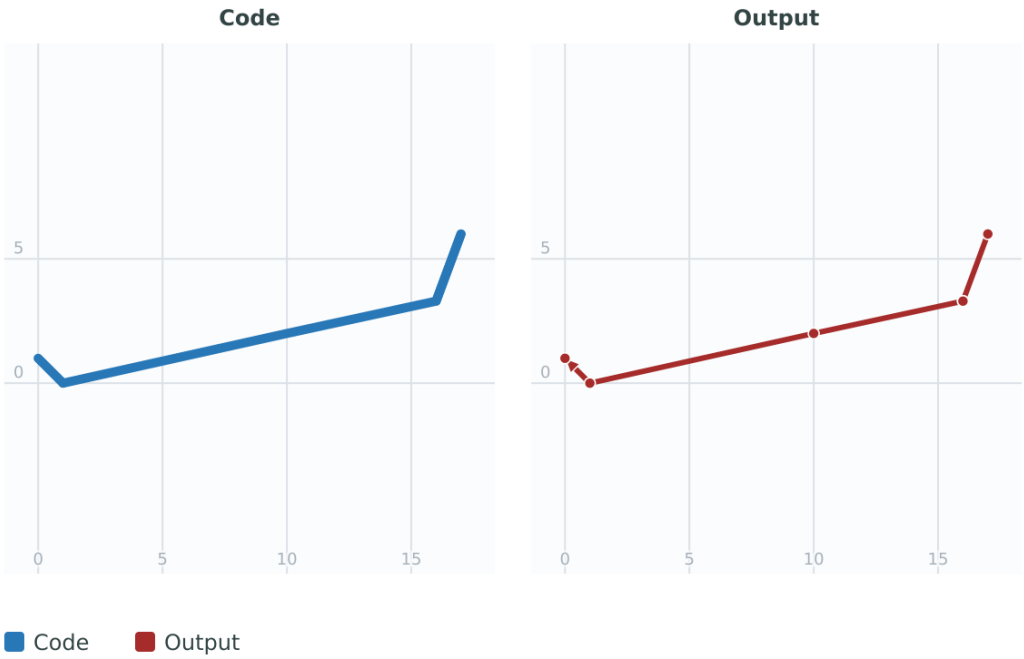
Code

```
SELECT ST_LineMerge('MULTILINESTRING((-29 -27 11,-30 -29.7 10,-36 -31 5,-45 -33 6),(-29 -27 ↩
    12,-30 -29.7 5),(-45 -33 1,-46 -32 11))'
        );
```

Output

```
LINESTRING Z (-30 -29.7 5,-29 -27 11,-30 -29.7 10,-36 -31 5,-45 -33 1,-46 -32 11)
```

Figure



Siehe auch

ST_Segmentize, ST_LineSubstring

7.14.13 ST_MaximumInscribedCircle

ST_MaximumInscribedCircle — Berechnet die konvexe Hülle einer Geometrie.

Synopsis

(geometry, geometry, double precision) **ST_MaximumInscribedCircle**(geometry geom);

Beschreibung

Findet den größten Kreis, der in einem (Mehr-)Polygon enthalten ist oder der keine Linien und Punkte überschneidet. Gibt einen Datensatz mit Feldern zurück:

- `center` - Mittelpunkt des Kreises
- `nächstgelegener` - ein Punkt auf der Geometrie, der dem Zentrum am nächsten liegt
- `Radius` - Radius des Kreises

Bei polygonalen Eingaben wird der Kreis innerhalb der Begrenzungsringe eingeschrieben, wobei die inneren Ringe als Begrenzungen verwendet werden. Bei linearen und punktförmigen Eingaben wird der Kreis in die konvexe Hülle der Eingabe eingeschrieben, wobei die eingegebenen Linien und Punkte als weitere Begrenzungen verwendet werden.

Verfügbarkeit: 3.1.0.

Erfordert GEOS >= 3.9.0.

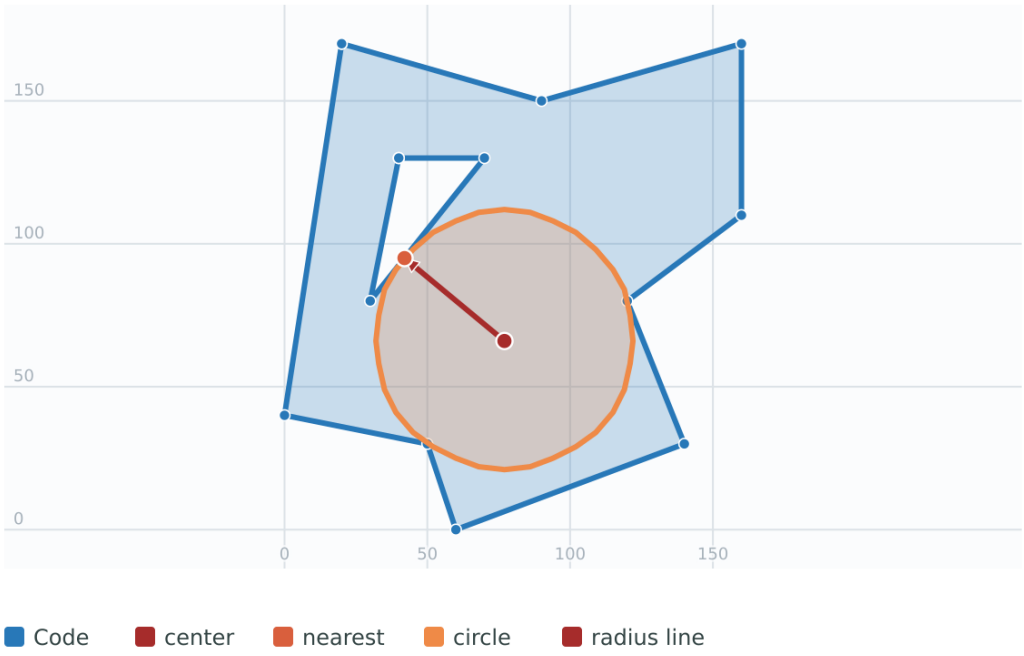
Beispiele

Maximum inscribed circle of a polygon. Center, nearest point, radius, circle, and radius line are returned.

Code

```
WITH mic AS (
  SELECT * FROM ST_MaximumInscribedCircle(
    'POLYGON ((40 180,110 160,180 180,180 120,140 90,160 40,80 10,70 40,20 50,40 180),
              (60 140,50 90,90 140,60 140))')
)
SELECT round(radius::numeric, 3) AS radius,
       ST_SnapToGrid(center, 1) AS center,
       ST_SnapToGrid(nearest, 1) AS nearest,
       ST_SnapToGrid(ST_Buffer(center, radius), 1) AS circle,
       ST_SnapToGrid(ST_MakeLine(center, nearest), 1) AS radius_line
FROM mic;
```

Figure

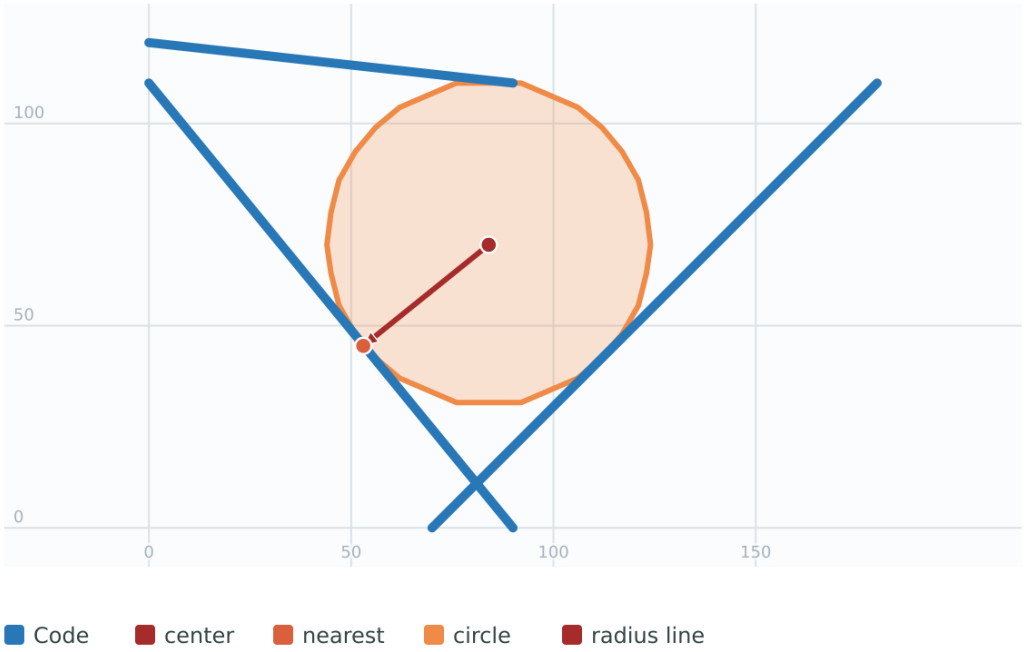


Maximum inscribed circle of a MultiLineString.

Code

```
WITH mic AS (
  SELECT * FROM ST_MaximumInscribedCircle(
    'MULTILINESTRING((100 40,10 150),(10 160,100 150),(190 150,80 40))')
)
SELECT round(radius::numeric, 3) AS radius,
       ST_SnapToGrid(center, 1) AS center,
       ST_SnapToGrid(nearest, 1) AS nearest,
       ST_SnapToGrid(ST_Buffer(center, radius), 1) AS circle,
       ST_SnapToGrid(ST_MakeLine(center, nearest), 1) AS radius_line
FROM mic;
```

Figure



Siehe auch

ST_MinimumBoundingRadius, ST_LargestEmptyCircle

7.14.14 ST_LargestEmptyCircle

ST_LargestEmptyCircle — Berechnet den größten Kreis, der eine Geometrie nicht überschneidet.

Synopsis

(geometry, geometry, double precision) **ST_LargestEmptyCircle**(geometry geom, double precision tolerance=0.0, geometry boundary=POINT EMPTY);

Beschreibung

Findet den größten Kreis, der einen Satz von Punkt- und Linienhindernissen nicht überschneidet. (Polygonale Geometrien können als Hindernisse einbezogen werden, aber nur ihre Begrenzungslinien werden verwendet). Der Mittelpunkt des Kreises muss innerhalb einer polygonalen Begrenzung liegen, die standardmäßig die konvexe Hülle der Eingabegeometrie ist. Der Kreismittelpunkt ist der Punkt im Inneren der Begrenzung, der den weitesten Abstand zu den Hindernissen hat. Der Kreis selbst wird durch den Mittelpunkt und einen nächstgelegenen Punkt auf einem Hindernis gebildet, der den Radius des Kreises bestimmt.

If a boundary is supplied it must be a valid polygonal geometry with positive area.

Der Kreismittelpunkt wird mit einem iterativen Algorithmus mit einer bestimmten Genauigkeit bestimmt, die durch eine Abstandstoleranz festgelegt wird. Wenn der Genauigkeitsabstand nicht angegeben ist, wird ein angemessener Standardwert verwendet.

Gibt einen Datensatz mit Feldern zurück:

- `center` - Mittelpunkt des Kreises
- `nächstgelegener` - ein Punkt auf der Geometrie, der dem Zentrum am nächsten liegt

- `Radius` - Radius des Kreises

Um den größten leeren Kreis im Inneren eines Polygons zu finden, siehe ST_MaximumInscribedCircle.

Verfügbarkeit: 3.4.0.

Erfordert GEOS >= 3.9.0.

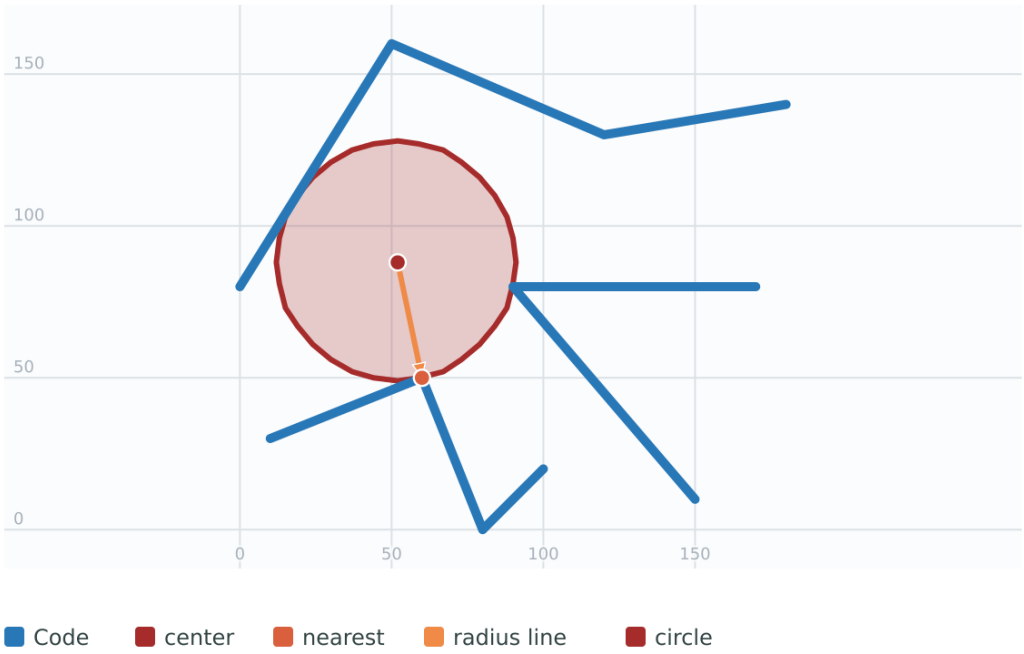
Beispiele

Largest empty circle within a set of lines.

Code

```
WITH obstacles(geom) AS (VALUES ('MULTILINESTRING (
        (10 100,60 180,130 150,190 160),
        (20 50,70 70,90 20,110 40),
        (160 30,100 100,180 100))'::geometry)),
circle AS (
  SELECT result.*
  FROM obstacles
  CROSS JOIN LATERAL ST_LargestEmptyCircle(obstacles.geom) AS result
)
SELECT round(radius::numeric, 6) AS radius,
       ST_SnapToGrid(center, 1) AS center,
       ST_SnapToGrid(nearest, 1) AS nearest,
       ST_SnapToGrid(ST_MakeLine(center, nearest), 1) AS radius_line,
       ST_SnapToGrid(ST_Buffer(center, radius), 1) AS circle
FROM circle;
```

Figure

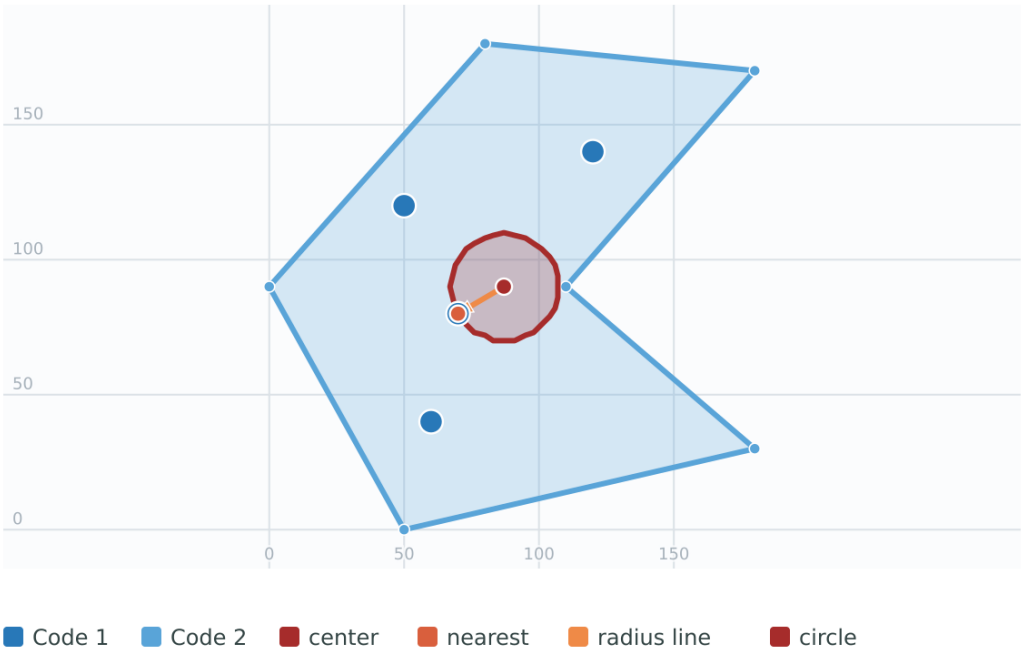


Largest empty circle within a set of points, constrained to lie in a polygon. The constraint polygon boundary is included both as an obstacle and as the constraint for the circle center.

Code

```
WITH inputs(points, boundary) AS (VALUES (
  'MULTIPOINT ((70 50),(60 130),(130 150),(80 90))'::geometry,
  'POLYGON ((90 190,10 100,60 10,190 40,120 100,190 180,90 190))'::geometry
)),
circle AS (
  SELECT result.*
  FROM inputs
  CROSS JOIN LATERAL ST_LargestEmptyCircle(
    ST_Collect(points, boundary), 0, boundary
  ) AS result
)
SELECT round(radius::numeric, 6) AS radius,
       ST_SnapToGrid(center, 1) AS center,
       ST_SnapToGrid(nearest, 1) AS nearest,
       ST_SnapToGrid(ST_MakeLine(center, nearest), 1) AS radius_line,
       ST_SnapToGrid(ST_Buffer(center, radius), 1) AS circle
FROM circle;
```

Figure



Siehe auch

ST_MinimumBoundingRadius

7.14.15 ST_MinimumBoundingCircle

ST_MinimumBoundingCircle — Gibt das kleinste Kreispolygon zurück, das eine Geometrie enthält.

Synopsis

geometry **ST_MinimumBoundingCircle**(geometry geomA, integer num_segs_per_qt_circ=48);

Beschreibung

Gibt das kleinste Kreispolygon zurück, das eine Geometrie enthält.



Anmerkung
Der Kreis wird standardmäßig durch ein Polygon mit 48 Segmenten pro Viertelkreis angenähert. Da das Polygon eine Annäherung an den minimalen Umgebungskreis ist, können einige Punkte der Eingabegeometrie nicht in dem Polygon enthalten sein. Die Annäherung kann durch Erhöhung der Anzahl der Segmente mit geringen Einbußen bei der Rechenleistung verbessert werden. Bei Anwendungen wo eine polygonale Annäherung nicht ausreicht, kann ST_MinimumBoundingRadius verwendet werden.

Verwendung mit ST_Collect, um den minimalen Begrenzungskreis einer Reihe von Geometrien zu ermitteln.

Um zwei Punkte zu berechnen, die auf dem minimalen Kreis (dem "maximalen Durchmesser") liegen, verwenden Sie ST_LongestLine.

Das Verhältnis zwischen der Fläche des Polygons und der Fläche ihres kleinstmöglichen Umgebungskreises wird öfter als Roeck Test bezeichnet.

Wird vom GEOS Modul ausgeführt

Verfügbarkeit: 1.4.0

Beispiele

Code

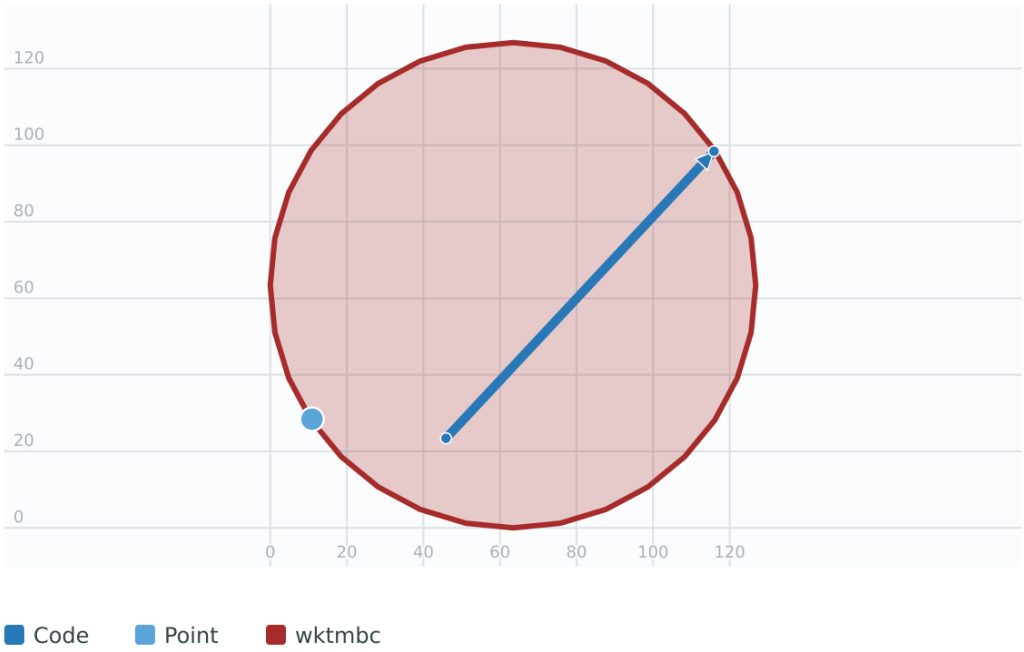
```
SELECT d.disease_type,
ST_MinimumBoundingCircle(ST_Collect(d.geom)) As geom
FROM disease_obs As d
GROUP BY d.disease_type;
```

Minimum bounding circle of a point and linestring, using 8 segments to approximate a quarter circle.

Code

```
SELECT ST_MinimumBoundingCircle(ST_Collect('LINESTRING(55 75,125 150)',
ST_Point(20, 80)), 8
) As wktmbc;
```

Figure



Siehe auch

ST_Collect, ST_MinimumBoundingRadius, ST_LargestEmptyCircle, ST_LongestLine

7.14.16 ST_MinimumBoundingRadius

ST_MinimumBoundingRadius — Gibt den Mittelpunkt und den Radius des kleinsten Kreises zurück, der eine Geometrie enthält.

Synopsis

(geometry, double precision) **ST_MinimumBoundingRadius**(geometry geom);

Beschreibung

Berechnet den Mittelpunkt und den Radius des kleinsten Kreises, der eine Geometrie enthält. Gibt einen Datensatz mit Feldern zurück:

- `center` - Mittelpunkt des Kreises
- `Radius` - Radius des Kreises

Verwendung mit ST_Collect, um den minimalen Begrenzungskreis einer Reihe von Geometrien zu ermitteln.

Um zwei Punkte zu berechnen, die auf dem minimalen Kreis (dem "maximalen Durchmesser") liegen, verwenden Sie ST_LongestLine.

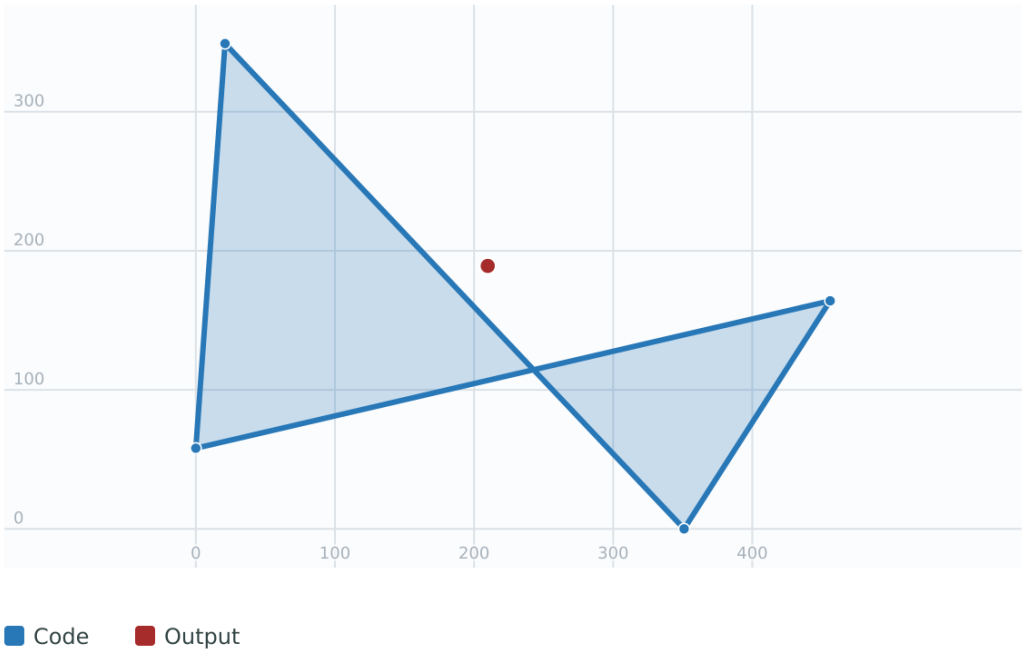
Verfügbarkeit: 2.3.0

Beispiele

Code

```
SELECT center, radius FROM ST_MinimumBoundingRadius('POLYGON((26426 65078,26531 65242,26075 ↩
    65136,26096 65427,26426 65078))');
```

Figure



Siehe auch

ST_Collect, ST_MinimumBoundingCircle, ST_LongestLine

7.14.17 ST_OrientedEnvelope

ST_OrientedEnvelope — Gibt ein Rechteck mit minimalem Flächeninhalt zurück, das eine Geometrie enthält.

Synopsis

geometry **ST_OrientedEnvelope**(geometry geom);

Beschreibung

Gibt das gedrehte Rechteck mit dem kleinsten Flächeninhalt zurück, das eine Geometrie umschließt. Beachten Sie, dass es mehr als ein solches Rechteck geben kann. Kann im Fall von degenerierten Eingaben einen Punkt oder einen LineString zurückgeben.

Verfügbarkeit: 2.5.0.

Erfordert GEOS >= 3.6.0.

Beispiele

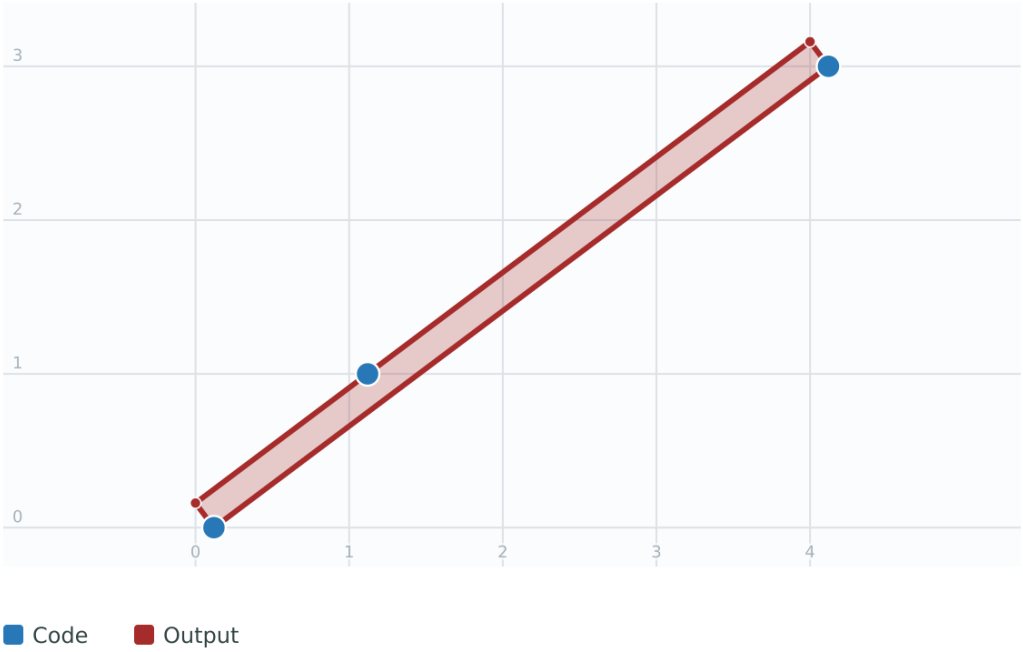
Code

```
SELECT ST_OrientedEnvelope('MULTIPOINT ((0 0),(-1 -1),(3 2))');
```

Output

```
POLYGON((3 2,-1 -1,-1.12 -0.84,2.88 2.16,3 2))
```

Figure



Oriented envelope of a Point and LineString.

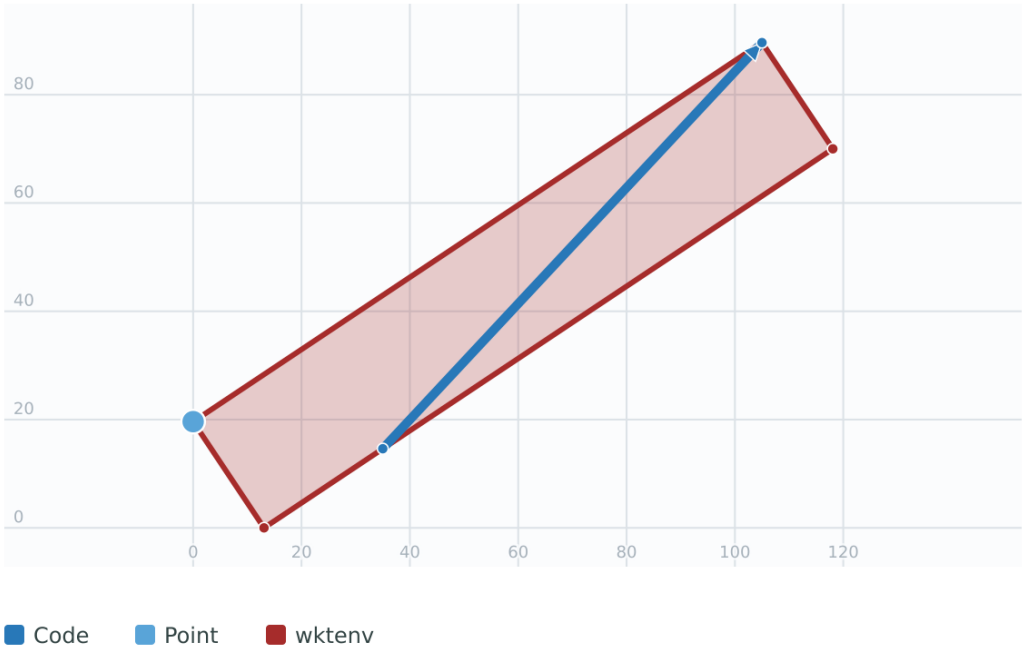
Code

```
SELECT ST_OrientedEnvelope(ST_Collect('LINESTRING(55 75,125 150)',
                ST_Point(20, 80))
                ) As wktenv;
```

Output

```
POLYGON((20 80,125 150,138.0769230769241 130.38461538461573,33.07692307692291 60.384615384615195,20 80))
```

Figure



Siehe auch

ST_Envelope ST_MinimumBoundingCircle

7.14.18 ST_OffsetCurve

ST_OffsetCurve — Gibt eine versetzte Linie in einem bestimmten Abstand und einer bestimmten Seite von einer Eingabelinie zurück.

Synopsis

geometry **ST_OffsetCurve**(geometry line, float signed_distance, text style_parameters='');

Beschreibung

Gibt eine versetzte Linie in einem bestimmten Abstand und einer bestimmten Seite von einer Eingabelinie zurück. Alle Punkte der zurückgegebenen Geometrien sind nicht weiter als der angegebene Abstand von der Eingabegeometrie entfernt. Nützlich für die Berechnung von parallelen Linien um eine Mittellinie.

Bei einem positiven Abstand befindet sich der Versatz auf der linken Seite der Eingabezeile und behält die gleiche Richtung bei. Bei einem negativen Abstand befindet er sich auf der rechten Seite und in der entgegengesetzten Richtung.

Die Einheiten der Entfernung werden in den Einheiten des Koordinatenreferenzsystems gemessen.

Bei einer Puzzlestück-förmigen Geometrie kann die Ausgabe manchmal ein MULTILINESTRING oder EMPTY sein.

Der optionale dritte Parameter ermöglicht es eine Liste von leerzeichengetrennten key=value Paaren anzulegen, um die Berechnungen wie folgt zu optimieren:

- 'quad_segs=#' : Anzahl der Segmente die verwendet werden um einen Viertelkreis anzunähern (standardmäßig 8).
- 'join=round|mitre|bevel' : join style (defaults to "round"). 'miter' kann auch als Synonym für 'mitre' verwendet werden.

- 'mitre_limit=#.#' : Gehrungsobergrenze (beeinflusst nur Gehrungsverbindungen). 'miter_limit' kann auch als Synonym von 'mitre_limit' verwendet werden.

Wird vom GEOS Modul ausgeführt

Durch die Änderung des Verhaltens in GEOS 3.11 haben versetzte Kurven nun die gleiche Richtung wie die Eingabelinie, sowohl bei positiven als auch bei negativen Versetzungen.

Verfügbarkeit: 2.0

Erweiterung: ab 2.5 wird auch GEOMETRYCOLLECTION und MULTILINESTRING unterstützt.



Anmerkung
Diese Funktion ignoriert die Z-Dimension. Sie liefert immer ein 2D-Ergebnis, auch wenn sie auf eine 3D-Geometrie angewendet wird.

Beispiele

Compute an open buffer around roads.

Code

```
SELECT ST_Union(
 ST_OffsetCurve(f.geom, f.width/2, 'quad_segs=4 join=round'),
 ST_OffsetCurve(f.geom, -f.width/2, 'quad_segs=4 join=round')
) as track
FROM someroadstable;
```

Offset 15 units with `quad_segs=4 join=round`.

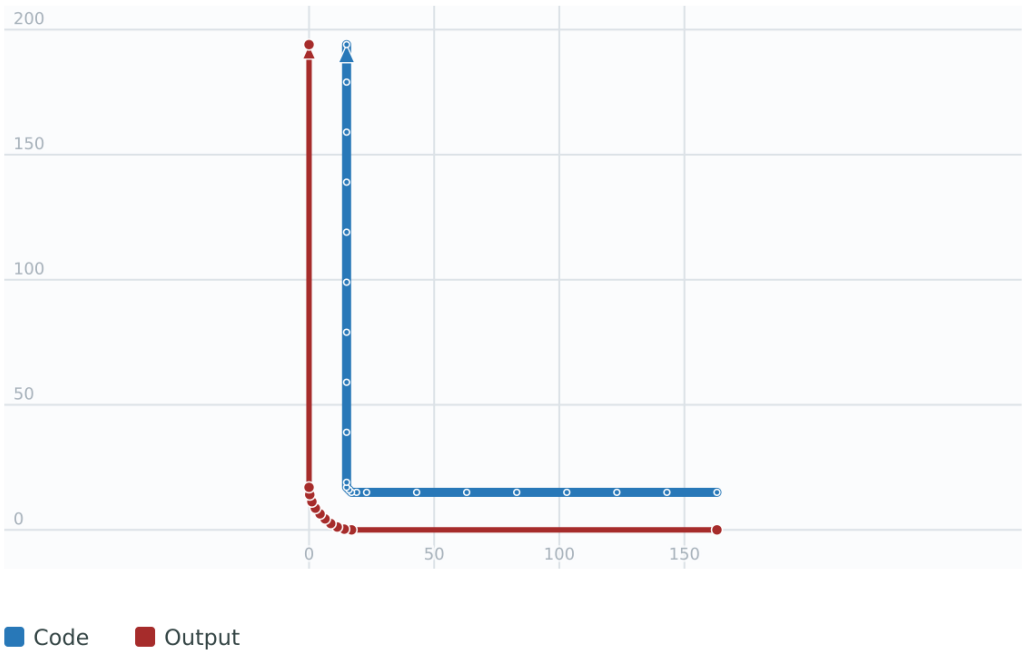
Code

```
SELECT ST_OffsetCurve(
    ST_GeomFromText('LINESTRING(164 16,144 16,124 16,104 16,84 16,64 16,
    44 16,24 16,20 16,18 16,17 17,
    16 18,16 20,16 40,16 60,16 80,16 100,
    16 120,16 140,16 160,16 180,16 195)'),
    15, 'quad_segs=4 join=round');
```

Output

```
LINESTRING(164 1,18 1,15.073645169758077 1.288220793951544,
    12.259748514523654 2.141807012330698,9.66644650470597 3.527955815461818,
    7.393398282201788 5.393398282201788,5.393398282201788 7.393398282201788,
    3.527955815461818 9.66644650470597,2.141807012330696 12.259748514523654,
    1.288220793951544 15.073645169758075,1 18,1 195)
```

Figure



Offset -15 units with `quad_segs=4 join=round`.

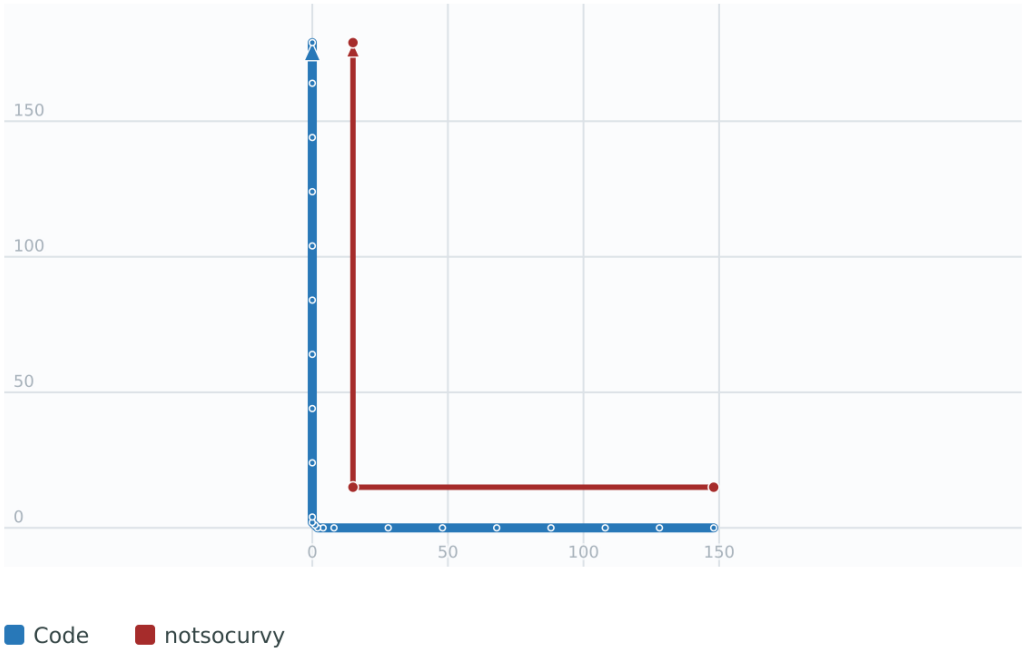
Code

```
SELECT ST_OffsetCurve(
    geom,
    -15,
    'quad_segs=4 join=round') As notsocurvy
    FROM ST_GeomFromText('LINESTRING(164 16,144 16,124 16,104 16,84 16,64 16,
    44 16,24 16,20 16,18 16,17 17,
    16 18,16 20,16 40,16 60,16 80,16 100,
    16 120,16 140,16 160,16 180,16 195)') As geom;
```

Output

```
LINESTRING(164 31,31 31,31 195)
```

Figure



Apply two negative offsets to obtain a smoother curve; the distances combine to -45 units.

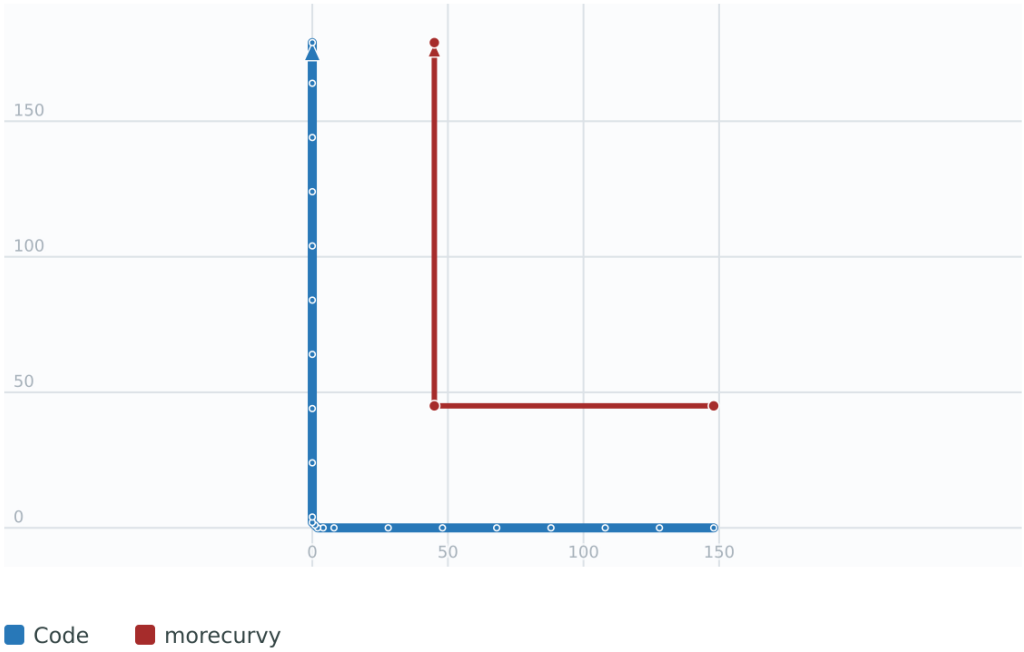
Code

```
SELECT ST_OffsetCurve(
    ST_OffsetCurve(
        geom,
        -30,
        'quad_segs=4 join=round'),
    -15,
    'quad_segs=4 join=round') As morecurvy
    FROM ST_GeomFromText('LINESTRING(164 16,144 16,124 16,104 16,84 16,64 16,
    44 16,24 16,20 16,18 16,17 17,
    16 18,16 20,16 40,16 60,16 80,16 100,
    16 120,16 140,16 160,16 180,16 195)') As geom;
```

Output

```
LINESTRING(164 61,61 61,61 195)
```

Figure



Collect the positive offset and the double-offset to produce parallel curves.

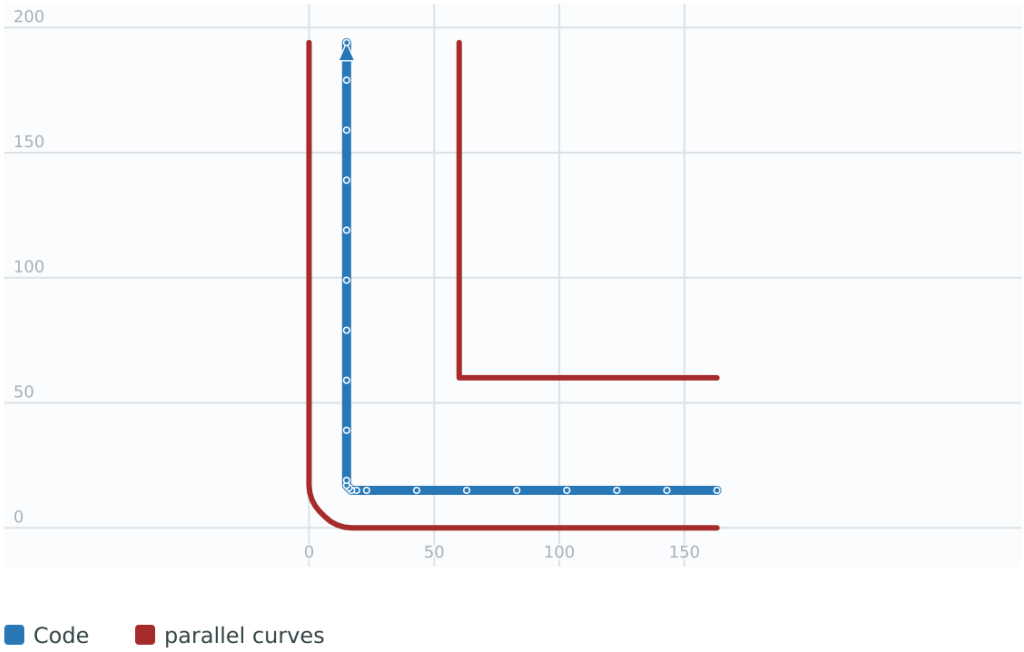
Code

```
SELECT ST_Collect(
    ST_OffsetCurve(geom, 15, 'quad_segs=4 join=round'),
    ST_OffsetCurve(
        ST_OffsetCurve(
            geom,
            -30,
            'quad_segs=4 join=round'),
        -15,
        'quad_segs=4 join=round')
) As parallel_curves
    FROM ST_GeomFromText('LINESTRING(164 16,144 16,124 16,104 16,84 16,64 16,
    44 16,24 16,20 16,18 16,17 17,
    16 18,16 20,16 40,16 60,16 80,16 100,
    16 120,16 140,16 160,16 180,16 195)') As geom;
```

Output

```
MULTILINESTRING((164 1,18 1,15.073645169758077 1.288220793951544,
12.259748514523654 2.141807012330698,9.66644650470597 3.527955815461818,
7.393398282201788 5.393398282201788,5.393398282201788 7.393398282201788,
3.527955815461818 9.66644650470597,2.141807012330696 12.259748514523654,
1.288220793951544 15.073645169758075,1 18,1 195),(164 61,61 61,61 195))
```

Figure



Offset 15 units with a bevel join.

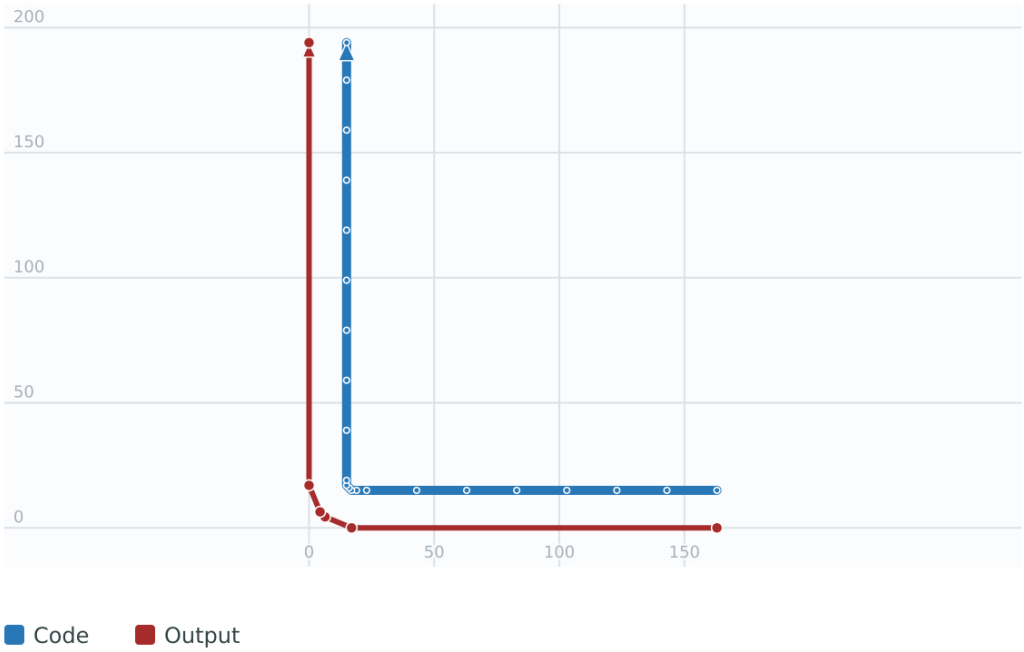
Code

```
SELECT ST_OffsetCurve(
    ST_GeomFromText('LINESTRING(164 16,144 16,124 16,104 16,84 16,64 16,
    44 16,24 16,20 16,18 16,17 17,
    16 18,16 20,16 40,16 60,16 80,16 100,
    16 120,16 140,16 160,16 180,16 195)'),
        15, 'quad_segs=4 join=bevel');
```

Output

```
LINESTRING(164 1,18 1,7.393398282201788 5.393398282201788,
    5.393398282201788 7.393398282201788,1 18,1 195)
```

Figure



Collect offsets on both sides using a mitre join and a mitre limit of 2.2.

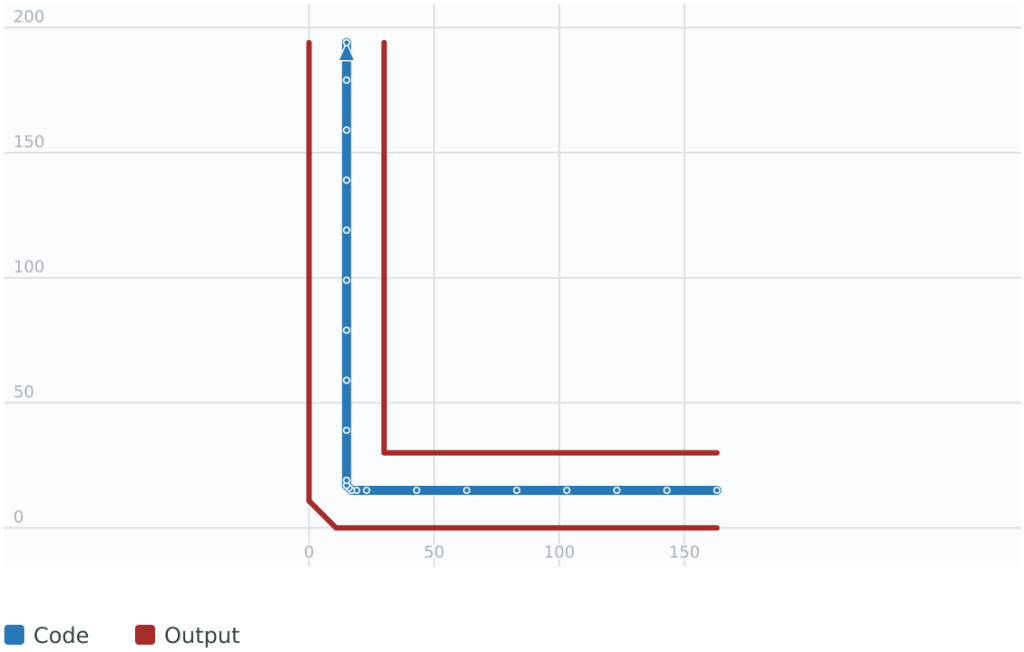
Code

```
SELECT ST_Collect(
    ST_OffsetCurve(geom, 15, 'quad_segs=4 join=mitre mitre_limit=2.2'),
    ST_OffsetCurve(geom, -15, 'quad_segs=4 join=mitre mitre_limit=2.2')
    )
    FROM ST_GeomFromText('LINESTRING(164 16,144 16,124 16,104 16,84 16,64 16,
    44 16,24 16,20 16,18 16,17 17,
    16 18,16 20,16 40,16 60,16 80,16 100,
    16 120,16 140,16 160,16 180,16 195)') As geom;
```

Output

```
MULTILINESTRING((164 1,11.786796564403577 1,1 11.786796564403577,1 195),
    (164 31,31 31,31 195))
```

Figure



Siehe auch

ST_Buffer

7.14.19 ST_PointOnSurface

ST_PointOnSurface — Berechnet einen Punkt, der garantiert in einem Polygon oder auf einer Geometrie liegt.

Synopsis

geometry **ST_PointOnSurface**(geometry g1);

Beschreibung

Gibt einen `POINT` zurück, der garantiert im Inneren einer Fläche liegt (`POLYGON`, `MULTIPOLYGON` und `CURVEPOLYGON`). In PostGIS funktioniert diese Funktion auch für Linien- und Punktgeometrien.

Diese Methode implementiert die OGC Simple Features Implementation Specification for SQL 1.1. s3.2.14.2 // s3.2.18.2

Diese Methode setzt die SQL/MM-Spezifikation um. SQL-MM 3: 8.1.5, 9.5.6. Die Spezifikationen definieren ST_PointOnSurface nur für Oberflächengeometrien. PostGIS erweitert die Funktion, um alle gängigen Geometrietypen zu unterstützen. Andere Datenbanken (Oracle, DB2, ArcSDE) scheinen diese Funktion nur für Flächen zu unterstützen. SQL Server 2008 unterstützt alle gängigen Geometrietypen.

Diese Funktion unterstützt 3d und lässt den Z-Index nicht fallen.

Beispiele

Point on surface of a MultiPoint.

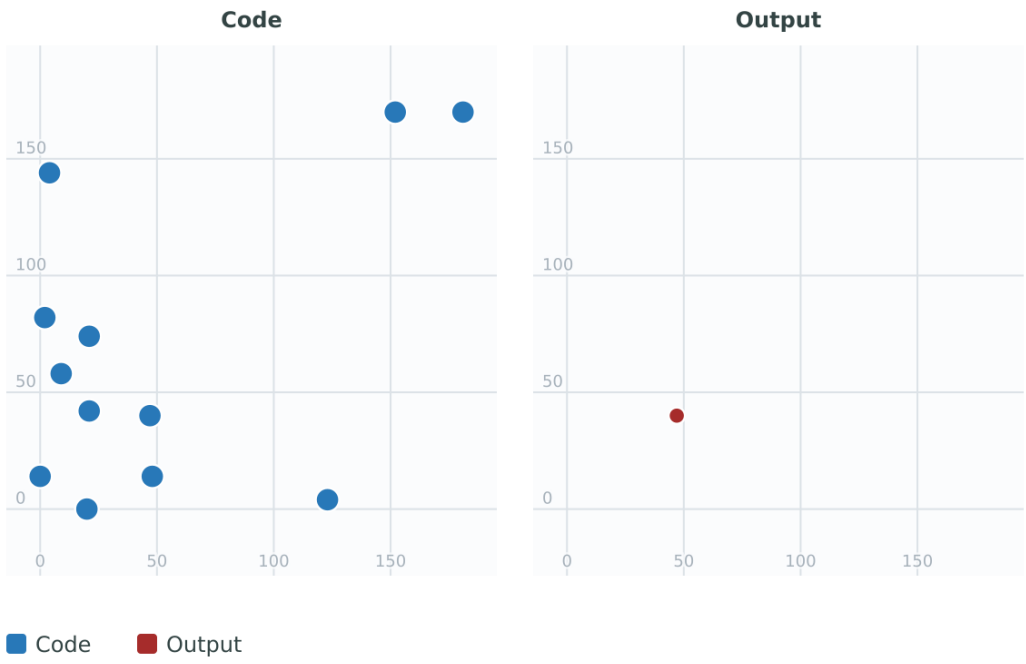
Code

```
SELECT ST_PointOnSurface(
  'MULTIPOINT (8 24,10 92,12 154,17 68,28 10,29 52,29 84,55 50,56 24,131 14,160 180,189 180) ↩
    ');
```

Output

```
POINT(55 50)
```

Figure



Point on surface of a LineString.

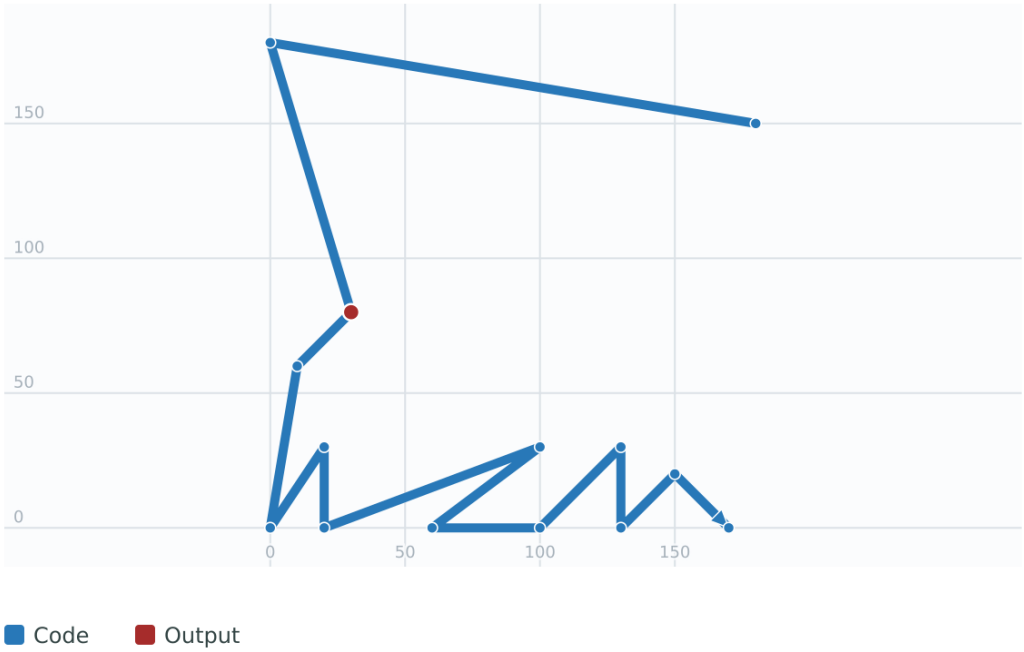
Code

```
SELECT ST_PointOnSurface(
  'LINESTRING (190 160,10 190,40 90,20 70,10 10,30 40,30 10,110 40,70 10,110 10,140 40,140  ↩
    10,160 30,180 10)');
```

Output

```
POINT(40 90)
```

Figure



Point on surface of a Polygon.

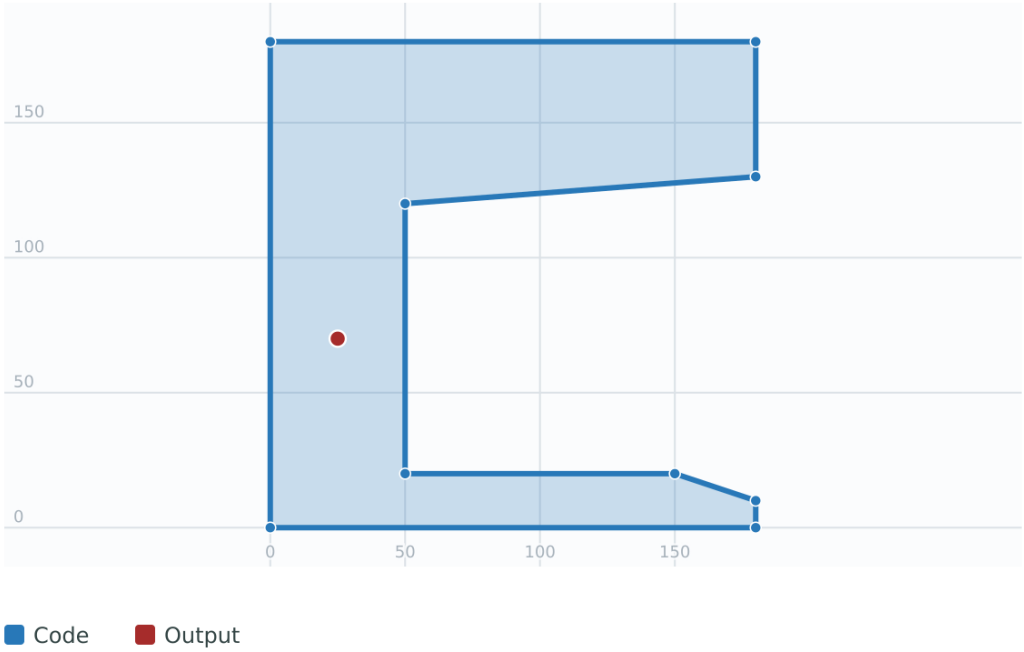
Code

```
SELECT ST_PointOnSurface(
  'POLYGON ((190 190,10 190,10 10,190 10,190 20,160 30,60 30,60 130,190 140,190 190))');
```

Output

```
POINT(35 80)
```

Figure



Point on surface of the highest-dimension components of a GeometryCollection.

Code

```
SELECT ST_PointOnSurface(
  'GEOMETRYCOLLECTION (POLYGON ((190 170,180 100,80 140,80 160,130 160,110 180,110 190,180 ↩
   180,190 170)),
    LINESTRING (80 120,120 20,140 70,150 30,180 50,190 10),
    MULTIPOINT (19 150,22 49,30 13,32 101,45 35,67 88,75 16))');
```

Output

```
POINT(133.571 150)
```

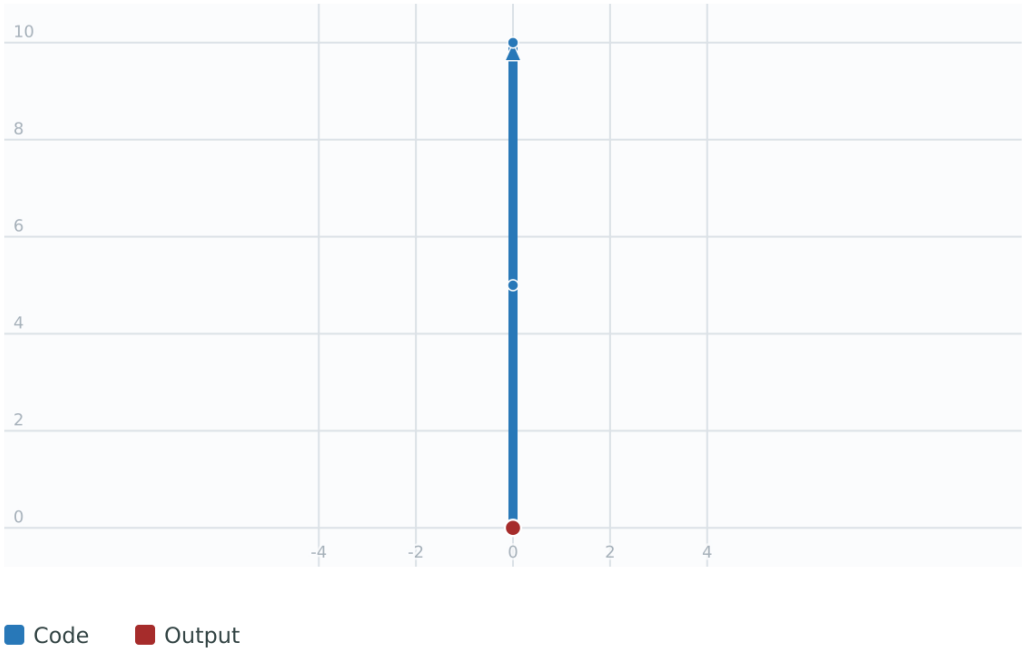
Figure



Code

```
SELECT ST_PointOnSurface('LINESTRING(0 5 1,0 0 1,0 10 2)');
```

Figure

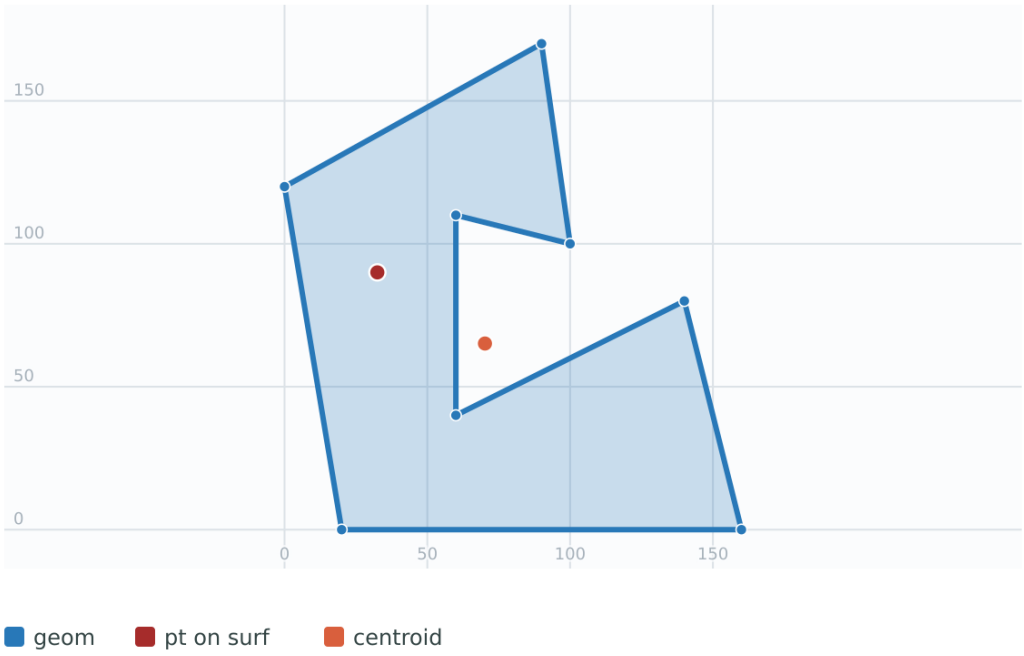


Beispiel: Das Ergebnis von ST_PointOnSurface liegt garantiert innerhalb von Polygonen, während der von ST_Centroid berechnete Punkt außerhalb liegen kann.

Code

```
SELECT ST_PointOnSurface(geom) AS pt_on_surf, ST_Centroid(geom) AS centroid
    FROM (SELECT 'POLYGON ((130 120,120 190,30 140,50 20,190 20,
                        170 100,90 60,90 130,130 120))'::geometry AS geom) AS t;
```

Figure



Siehe auch

ST_Centroid, ST_MaximumInscribedCircle

7.14.20 ST_Polygonize

ST_Polygonize — Berechnet eine Sammlung von Polygonen, die aus dem Linienwerk einer Reihe von Geometrien gebildet werden.

Synopsis

geometry **ST_Polygonize**(geometry set geomfield);
geometry **ST_Polygonize**(geometry[] geom_array);

Beschreibung

Erzeugt eine GeometryCollection, die die Polygone enthält, die durch das Liniengerüst einer Menge von Geometrien gebildet werden. Wenn das eingegebene Linienwerk keine Polygone bildet, wird eine leere GeometryCollection zurückgegeben.

Diese Funktion erzeugt Polygone, die alle abgegrenzten Bereiche abdecken. Wenn das Ergebnis eine gültige polygonale Geometrie bilden soll, verwenden Sie ST_BuildArea, um zu verhindern, dass Löcher gefüllt werden.



Anmerkung
Die Eingabelinie muss korrekt genodet sein, damit diese Funktion ordnungsgemäß funktioniert. Um sicherzustellen, dass die Eingabe nodiert ist, verwenden Sie ST_Node für die Eingabegeometrie vor der Polygonisierung.



Anmerkung
GeometryCollections können mit externen Werkzeugen schwer zu handhaben sein. Verwenden Sie ST_Dump, um das polygonisierte Ergebnis in separate Polygone umzuwandeln.

Wird vom GEOS Modul ausgeführt

Verfügbarkeit: 1.0.0RC1

Beispiele

Polygonize a set of input lines.

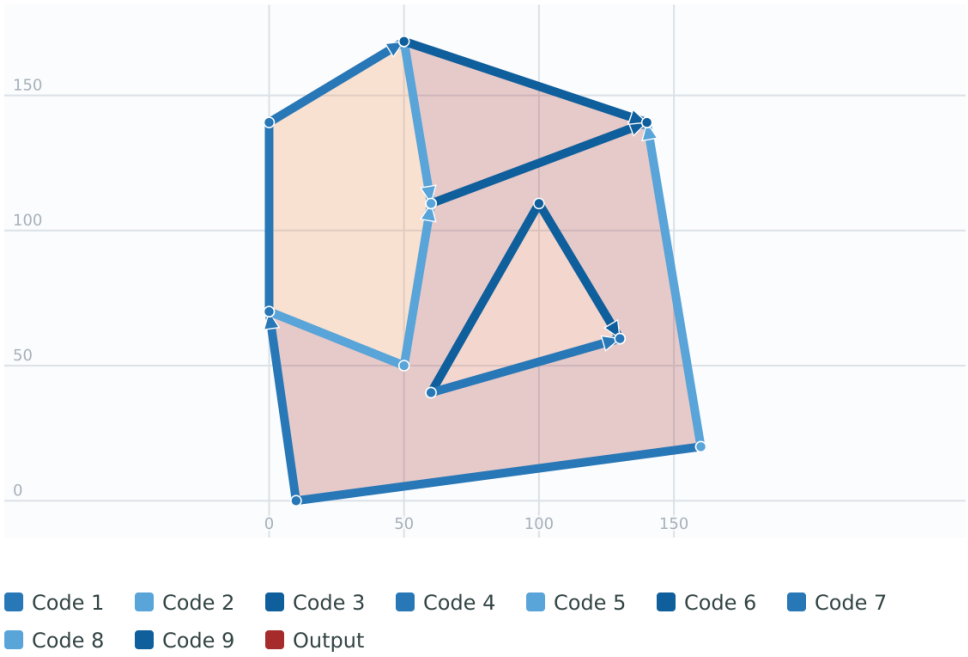
Code

```
WITH data(geom) AS (
  VALUES
    ('LINESTRING (180 40,30 20,20 90)'::geometry),
    ('LINESTRING (180 40,160 160)'::geometry),
    ('LINESTRING (80 60,120 130,150 80)'::geometry),
    ('LINESTRING (80 60,150 80)'::geometry),
    ('LINESTRING (20 90,70 70,80 130)'::geometry),
    ('LINESTRING (80 130,160 160)'::geometry),
    ('LINESTRING (20 90,20 160,70 190)'::geometry),
    ('LINESTRING (70 190,80 130)'::geometry),
    ('LINESTRING (70 190,160 160)'::geometry)
)
SELECT ST_Polygonize(geom)
FROM data;
```

Output

```
GEOMETRYCOLLECTION(POLYGON((180 40,30 20,20 90,70 70,80 130,160 160,180 40),(150 80,120 130,80 60,150 80)), POLYGON((80 ↩
      60,120 130,150 80,80 60)), POLYGON((80 130,70 70,20 90,20 160,70 190,80 130)), POLYGON((160 160,80 130,70 190,160 ↩
      160)))
```

Figure

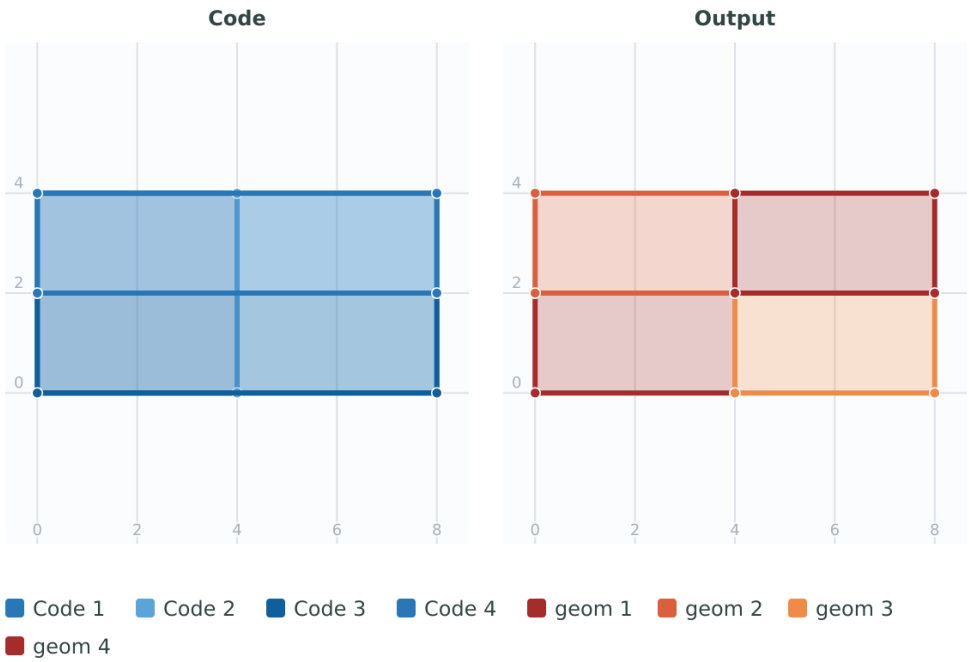


Polygonizing two polygon coverages and transferring attributes to the resultants:

Code

```
WITH poly_a(id, geom) AS (VALUES
  ('a1', 'POLYGON ((0 0, 4 0, 4 4, 0 4, 0 0))'::geometry),
  ('a2', 'POLYGON ((4 0, 8 0, 8 4, 4 4, 4 0))'::geometry)
),
poly_b(id, geom) AS (VALUES
  ('b1', 'POLYGON ((0 0, 8 0, 8 2, 0 2, 0 0))'::geometry),
  ('b2', 'POLYGON ((0 2, 8 2, 8 4, 0 4, 0 2))'::geometry)
),
lines AS (
  SELECT ST_Boundary(geom) AS geom FROM poly_a
  UNION ALL
  SELECT ST_Boundary(geom) FROM poly_b
),
noded AS (
  SELECT ST_UnaryUnion(ST_Collect(geom)) AS geom FROM lines
),
cells AS (
  SELECT (ST_Dump(ST_Polygonize(geom))).geom AS geom FROM noded
)
SELECT a.id AS id_a,
       b.id AS id_b,
       c.geom AS geom
FROM cells AS c
LEFT JOIN poly_a AS a ON ST_Within(ST_PointOnSurface(c.geom), a.geom)
LEFT JOIN poly_b AS b ON ST_Within(ST_PointOnSurface(c.geom), b.geom)
WHERE a.id IS NOT NULL OR b.id IS NOT NULL
ORDER BY id_a, id_b;
```

Figure



Polygonisierung einer Tabelle mit Linienzügen:

Code

```
SELECT ST_Polygonize(geom_4269) As geomtextrep
FROM (SELECT geom_4269 FROM ma.suffolk_edges) As foo;
```

Output

```
geomtextrep
-------------
 SRID=4269;GEOMETRYCOLLECTION(POLYGON((-71.040878 42.285678,-71.040943 42.2856,-71.04096  ↩
    42.285752,-71.040878 42.285678)),
 POLYGON((-71.17166 42.353675,-71.172026 42.354044,-71.17239 42.354358,-71.171794  ↩
    42.354971,-71.170511 42.354855,
 -71.17112 42.354238,-71.17166 42.353675)))
(1 row)
```

Use ST_Dump to dump out the polygonize geoms into individual polygons.

Code

```
SELECT (ST_Dump(t.polycoll)).geom AS geomtextrep
FROM (SELECT ST_Polygonize(geom_4269) AS polycoll
    FROM (SELECT geom_4269 FROM ma.suffolk_edges)
        As foo) AS t;
```

Output

```
geomtextrep
-------------
 SRID=4269;POLYGON((-71.040878 42.285678,-71.040943 42.2856,-71.04096 42.285752,
-71.040878 42.285678))
 SRID=4269;POLYGON((-71.17166 42.353675,-71.172026 42.354044,-71.17239 42.354358,
-71.171794 42.354971,-71.170511 42.354855,-71.17112 42.354238,-71.17166 42.353675))
(2 rows)
```

Siehe auch

ST_BuildArea, ST_Boundary, ST_Collect, ST_Dump, ST_Node, ST_PointOnSurface, ST_UnaryUnion, ST_Within

7.14.21 ST_ReducePrecision

ST_ReducePrecision — Gibt eine gültige Geometrie mit auf eine Rastertoleranz gerundeten Punkten zurück.

Synopsis

geometry **ST_ReducePrecision**(geometry g, float8 gridsize);

Beschreibung

Gibt eine gültige Geometrie zurück, bei der alle Punkte auf die angegebene Rastertoleranz gerundet und Features unterhalb der Toleranz entfernt wurden.

Im Gegensatz zu ST_SnapToGrid ist die zurückgegebene Geometrie gültig, ohne Ringselbstschnittpunkte oder kollabierte Komponenten.

Die Präzisionsreduzierung kann verwendet werden, um:

- Anpassung der Koordinatengenauigkeit an die Datengenauigkeit
- die Anzahl der für die Darstellung einer Geometrie erforderlichen Koordinaten zu verringern
- Gewährleistung einer gültigen Geometrieausgabe in Formaten, die eine geringere Genauigkeit verwenden (z. B. Textformate wie WKT, GeoJSON oder KML, wenn die Zahl der ausgegebenen Dezimalstellen begrenzt ist).
- Export gültiger Geometrie in Systeme, die eine geringere oder begrenzte Genauigkeit verwenden (z. B. SDE, Oracle-Toleranzwert)

Verfügbarkeit: 3.1.0.

Erfordert GEOS >= 3.9.0.

Beispiele

Code

```
SELECT ST_ReducePrecision('POINT(1.412 19.323)', 0.1);
```

Output

```
POINT(1.4 19.3)
```

Figure



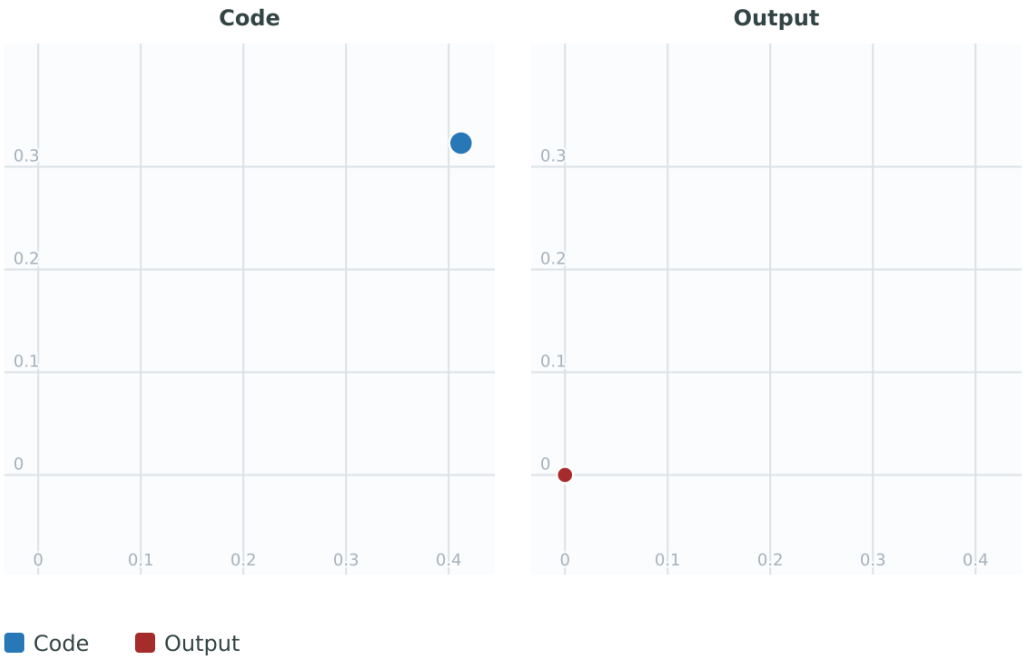
Code

```
SELECT ST_ReducePrecision('POINT(1.412 19.323)', 1.0);
```

Output

```
POINT(1 19)
```

Figure



Code

```
SELECT ST_ReducePrecision('POINT(1.412 19.323)', 10);
```

Output

```
POINT(0 20)
```

Figure



Präzisionsreduzierung kann die Anzahl der Scheitelpunkte reduzieren

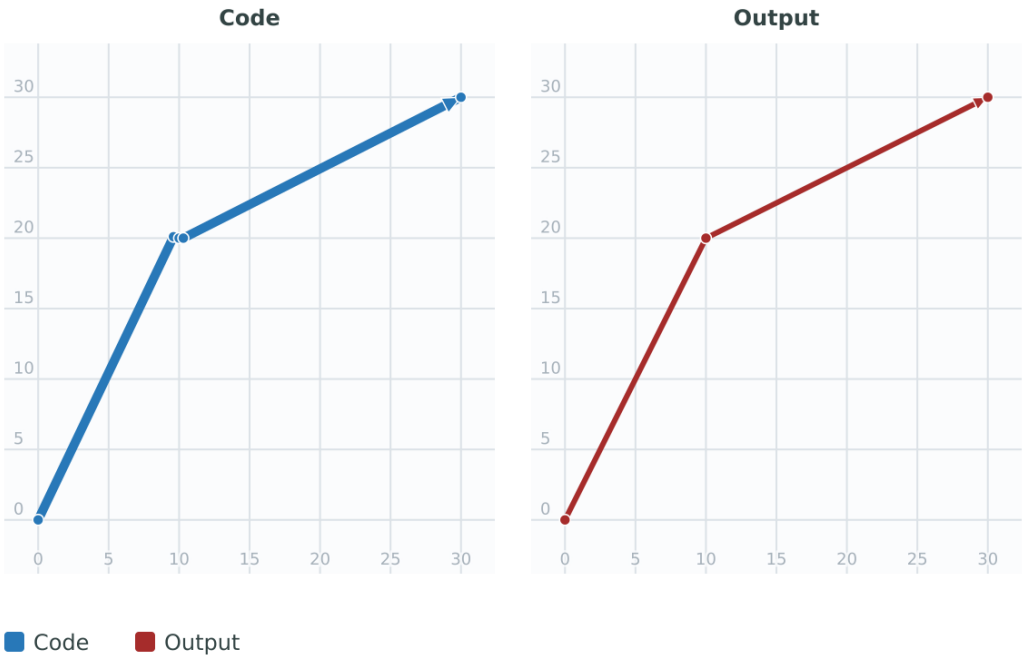
Code

```
SELECT ST_ReducePrecision('LINESTRING (10 10,19.6 30.1,20 30,20.3 30,40 40)', 1);
```

Output

```
LINESTRING (10 10, 20 30, 40 40)
```

Figure



Präzisionsreduzierung teilt Polygone bei Bedarf auf, um die Gültigkeit zu gewährleisten

Code

```
WITH data(geom) AS (
  VALUES ('POLYGON ((10 10,60 60.1,70 30,40 40,50 10,10 10))'::geometry)
)
SELECT geom AS input,
       ST_ReducePrecision(geom, 10) AS reduced
FROM data;
```

Output

```
-[ RECORD 1 ]-------------------------------------------------
input   | POLYGON((10 10,60 60.1,70 30,40 40,50 10,10 10))
reduced | MULTIPOLYGON(((10 10,40 40,50 10,10 10)),((40 40,60 60,70 30,40 40)))
```

Figure



Siehe auch

ST_SnapToGrid, ST_Simplify, ST_SimplifyVW

7.14.22 ST_SharedPaths

ST_SharedPaths — Gibt eine Sammelgeometrie zurück, welche die gemeinsamen Strecken der beiden eingegebenen LineStrings/MultiLinestrings enthält.

Synopsis

geometry **ST_SharedPaths**(geometry lineal1, geometry lineal2);

Beschreibung

Gibt eine Sammelgeometrie zurück, die die gemeinsamen Pfade zweier Eingabegeometrie enthält. Jene, die in derselben Richtung orientiert sind, werden im ersten Element der Sammelgeometrie, jene die in die entgegengesetzte Richtung orientiert sind, werden im zweiten Element gespeichert. Die Pfade selbst befinden sich in der ersten Geometrie.

Wird vom GEOS Modul ausgeführt

Verfügbarkeit: 2.0.0

Beispiele: Gemeinsame Strecken finden

Adjacent shared path segments may be returned as separate or merged LineStrings depending on the GEOS version.

Find the shared paths of a MultiLineString and a LineString.

Code

```
SELECT ST_SharedPaths(ST_GeomFromText('MULTILINESTRING((26 125,26 200,126 200,126 125,26 ↩
    125),
       (51 150,101 150,76 175,51 150))'),
   ST_GeomFromText('LINESTRING(151 100,126 156.25,126 125,101 150,76 175)')
   )
```

Output

```
GEOMETRYCOLLECTION(MULTILINESTRING((126 156.25,126 125),
(101 150,76 175)),MULTILINESTRING EMPTY)
```

Figure



Flip the LineString orientation to return the same paths in the opposite-direction member.

Code

```
SELECT ST_SharedPaths(ST_GeomFromText('LINESTRING(76 175,101 150,126 125,126 156.25,151 100) ↩
    '),
   ST_GeomFromText('MULTILINESTRING((26 125,26 200,126 200,126 125,26 125),
       (51 150,101 150,76 175,51 150))')
    )
```

Output

```
GEOMETRYCOLLECTION(MULTILINESTRING EMPTY,
MULTILINESTRING((76 175,101 150),(126 125,126 156.25)))
```

Figure



Siehe auch

ST_Dump, ST_GeometryN, ST_NumGeometries

7.14.23 ST_Simplify

ST_Simplify — Gibt eine vereinfachte Darstellung einer Geometrie unter Verwendung des Douglas-Peucker-Algorithmus zurück.

Synopsis

geometry **ST_Simplify**(geometry geom, float tolerance);
geometry **ST_Simplify**(geometry geom, float tolerance, boolean preserveCollapsed);

Beschreibung

Berechnet eine vereinfachte Darstellung einer Geometrie unter Verwendung des Douglas-Peucker-Algorithmus. Die Vereinfachung `Toleranz` ist ein Abstandswert, in den Einheiten der Eingabe SRS. Bei der Vereinfachung werden Eckpunkte entfernt, die innerhalb der Toleranzdistanz des vereinfachten Liniennetzes liegen. Das Ergebnis kann ungültig sein, auch wenn die Eingabe gültig ist.

Die Funktion kann mit jeder Art von Geometrie (einschließlich GeometryCollections) aufgerufen werden, aber nur Linien- und Polygonelemente werden vereinfacht. Die Endpunkte der linearen Geometrie bleiben erhalten.

Das Kennzeichen `preserveCollapsed` bewahrt kleine Geometrien, die andernfalls bei der angegebenen Toleranz entfernt werden würden. Wenn zum Beispiel eine 1 m lange Linie mit einer Toleranz von 10 m vereinfacht wird, verschwindet die Linie nicht, wenn `preserveCollapsed` true ist. Dieses Flag ist für Rendering-Zwecke nützlich, um zu verhindern, dass sehr kleine Merkmale aus einer Karte verschwinden.



Anmerkung
Die zurückgegebene Geometrie kann ihre Einfachheit verlieren (siehe ST_IsSimple), die Topologie wird möglicherweise nicht beibehalten, und polygonale Ergebnisse können ungültig sein (siehe ST_IsValid). Verwenden Sie ST_SimplifyPreserveTopology, um die Topologie zu erhalten und die Gültigkeit sicherzustellen.



Anmerkung
Diese Funktion erhält keine gemeinsamen Grenzen zwischen Polygonen. Verwenden Sie ST_CoverageSimplify, wenn dies erforderlich ist.

Verfügbarkeit: 1.2.2

Beispiele

A circle simplified with a medium tolerance becomes an octagon, and with a large tolerance disappears.

Code

```
SELECT ST_NPoints(geom) AS np_before,
     ST_NPoints(ST_Simplify(geom, 0.1)) AS np01_notbadcircle,
     ST_NPoints(ST_Simplify(geom, 0.5)) AS np05_notquitecircle,
     ST_NPoints(ST_Simplify(geom, 1)) AS np1_octagon,
     ST_NPoints(ST_Simplify(geom, 10)) AS np10_gone,
     (ST_Simplify(geom, 100) is null) AS  np100_geometrygoesaway
FROM (SELECT ST_Buffer('POINT(1 3)', 10, 12) As geom) AS t;
```

Output

```
-[ RECORD 1 ]------------
np_before                | 49
np01_notbadcircle        | 33
np05_notquitecircle      | 17
np1_octagon              | 9
np10_gone                | null
np100_geometrygoesaway   | t
```

Vereinfachung einer Reihe von Linien. Linien können sich nach der Vereinfachung kreuzen.

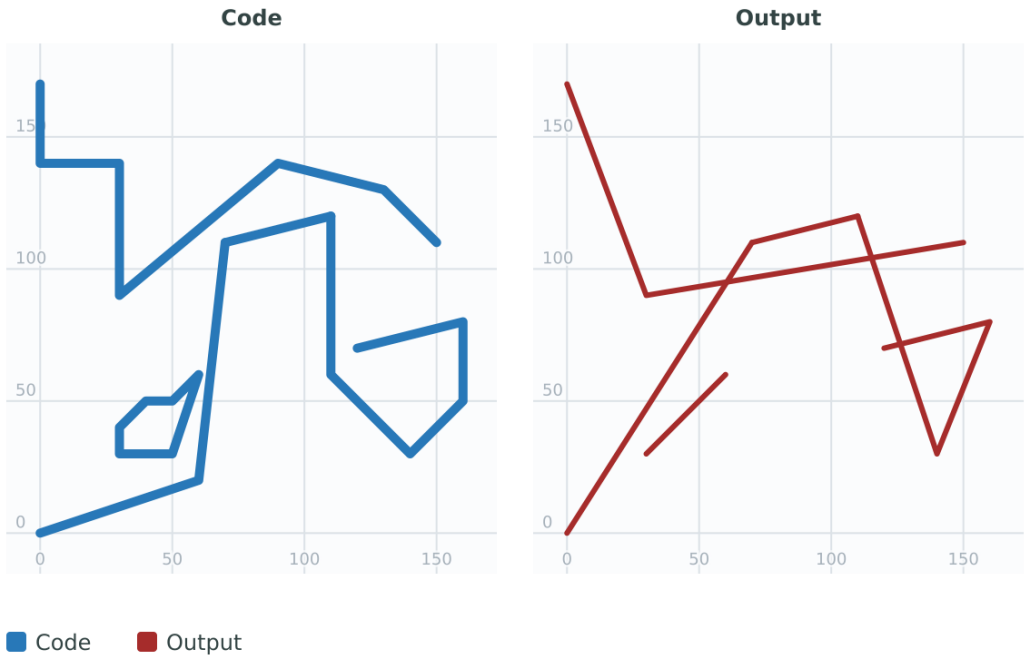
Code

```
SELECT ST_Simplify('MULTILINESTRING ((20 180,20 150,50 150,50 100,110 150,150 140,170 120) ↩
    ,(20 10,80 30,90 120),(90 120,130 130),(130 130,130 70,160 40,180 60,180 90,140 80),(50 ↩
    40,70 40,80 70,70 60,60 60,50 50,50 40))'::geometry,
40);
```

Output

```
MULTILINESTRING((20 180,50 100,170 120),(20 10,90 120),(90 120,130 130),(130 130,160 40,180 90,140 80),(50 40,80 70,50 ↩
    40))
```

Figure



Vereinfachung eines MultiPolygons. Polygonale Ergebnisse können ungültig sein.

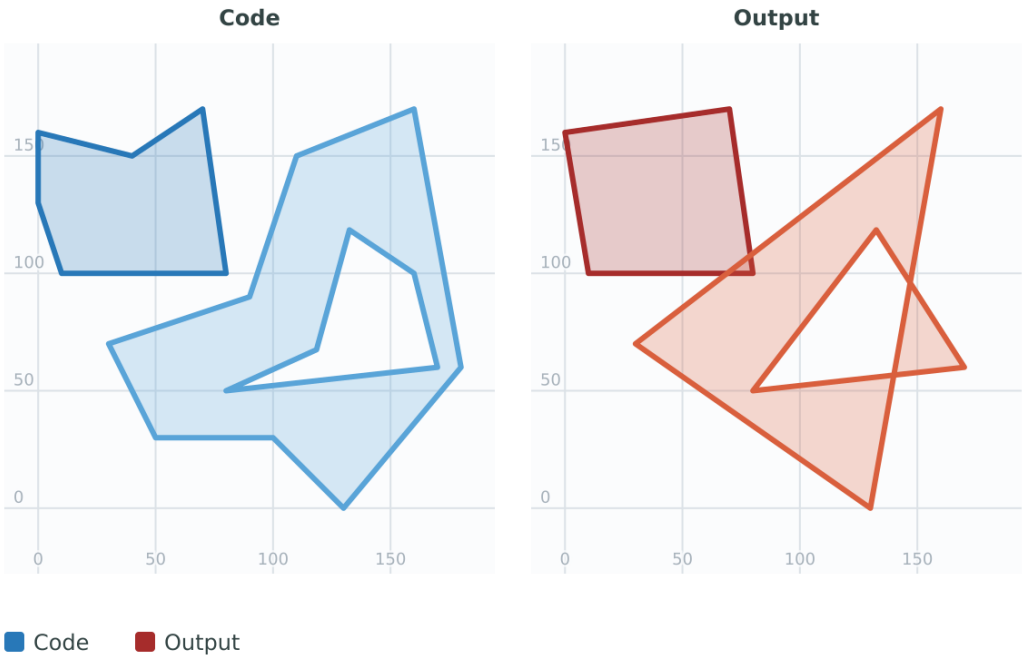
Code

```
SELECT ST_Simplify('MULTIPOLYGON (((90 110,80 180,50 160,10 170,10 140,20 110,90 110)),((40 ↩
    80,100 100,120 160,170 180,190 70,140 10,110 40,60 40,40 80),(180 70,170 110,142.5 ↩
    128.5,128.5 77.5,90 60,180 70)))'::geometry,
40);
```

Output

```
MULTIPOLYGON(((90 110,80 180,10 170,20 110,90 110)),((40 80,170 180,140 10,40 80),(180 70,142.5 128.5,90 60,180 70)))
```

Figure



Siehe auch

ST_IsSimple, ST_SimplifyPreserveTopology, ST_SimplifyVW, ST_CoverageSimplify, Topologie ST_Simplify

7.14.24 ST_SimplifyPreserveTopology

ST_SimplifyPreserveTopology — Gibt eine vereinfachte und gültige Darstellung einer Geometrie zurück, die den Douglas-Peucker-Algorithmus verwendet.

Synopsis

geometry **ST_SimplifyPreserveTopology**(geometry geom, float tolerance);

Beschreibung

Berechnet eine vereinfachte Darstellung einer Geometrie unter Verwendung einer Variante des Douglas-Peucker-Algorithmus, der die Vereinfachung einschränkt, um sicherzustellen, dass das Ergebnis die gleiche Topologie wie die Eingabe hat. Die Vereinfachung `Toleranz` ist ein Abstandswert in den Einheiten der Eingabe-SRS. Bei der Vereinfachung werden Eckpunkte entfernt, die innerhalb der Toleranzdistanz des vereinfachten Linienwerks liegen, solange die Topologie erhalten bleibt. Das Ergebnis wird gültig und einfach sein, wenn die Eingabe lautet.

Die Funktion kann mit jeder Art von Geometrie (einschließlich GeometryCollections) aufgerufen werden, aber nur Linien- und Polygonelemente werden vereinfacht. Bei polygonalen Eingaben hat das Ergebnis die gleiche Anzahl von Ringen (Schalen und Löcher), und die Ringe kreuzen sich nicht. Die Endpunkte der Ringe können vereinfacht werden. Bei linearen Eingaben hat das Ergebnis die gleiche Anzahl von Linien, und die Linien schneiden sich nicht, wenn sie dies in der ursprünglichen Geometrie nicht taten. Die Endpunkte der linearen Geometrie bleiben erhalten.



Anmerkung
Diese Funktion erhält keine gemeinsamen Grenzen zwischen Polygonen. Verwenden Sie ST_CoverageSimplify, wenn dies erforderlich ist.

Wird vom GEOS Modul ausgeführt

Verfügbarkeit: 1.3.3

Beispiele

For the same example as ST_Simplify, ST_SimplifyPreserveTopology prevents the geometry from disappearing.

Code

```
SELECT  ST_NPoints(geom) AS np_before,
        ST_NPoints(ST_SimplifyPreserveTopology(geom, 0.1)) AS np01_notbadcircle,
        ST_NPoints(ST_SimplifyPreserveTopology(geom, 0.5)) AS np05_notquitecircle,
        ST_NPoints(ST_SimplifyPreserveTopology(geom, 1))   AS np1_octagon,
        ST_NPoints(ST_SimplifyPreserveTopology(geom, 10))  AS np10_minimal,
        ST_NPoints(ST_SimplifyPreserveTopology(geom, 100)) AS np100_minimal
FROM (SELECT ST_Buffer('POINT(1 3)', 10, 12) AS geom) AS t;
```

Output

```
np_before | np01_notbadcircle | np05_notquitecircle | np1_octagon | np10_minimal |  ↩
    np100_minimal
-----------+-------------------+---------------------+-------------+--------------+--------------- ←

        49 |                33 |                  17 |           9 |            4 |          ↩
        4
```

Vereinfachung einer Reihe von Linien, wobei die Topologie der sich nicht schneidenden Linien erhalten bleibt.

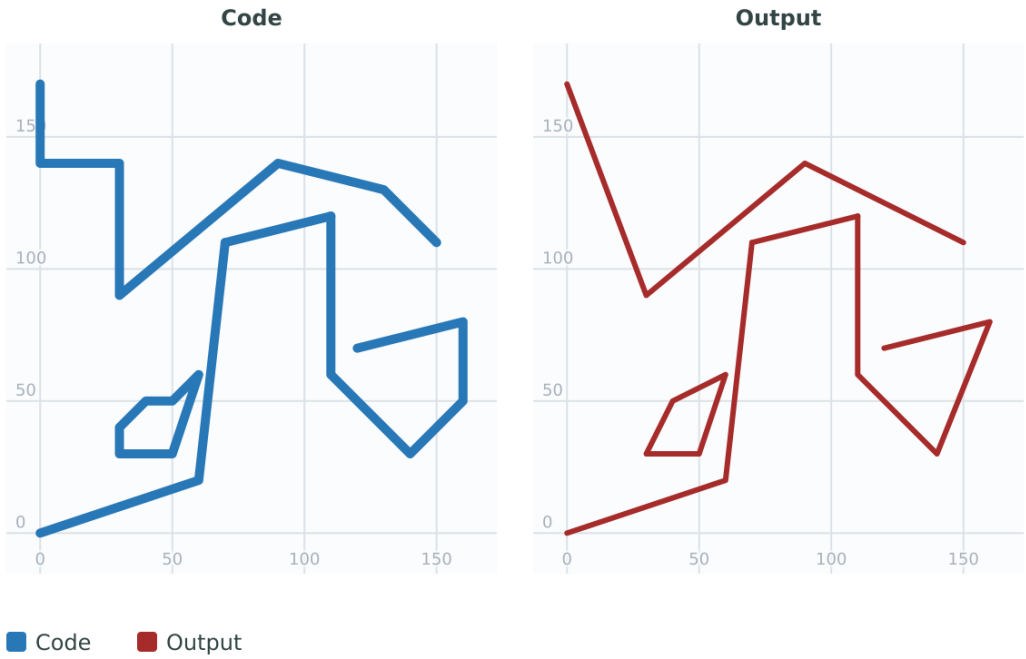
Code

```
SELECT ST_SimplifyPreserveTopology('MULTILINESTRING ((20 180,20 150,50 150,50 100,110  ↩
    150,150 140,170 120),(20 10,80 30,90 120),(90 120,130 130),(130 130,130 70,160 40,180  ↩
    60,180 90,140 80),(50 40,70 40,80 70,70 60,60 60,50 50,50 40))'::geometry,
40);
```

Output

```
MULTILINESTRING((20 180,50 100,110 150,170 120),(20 10,80 30,90 120),(90 120,130 130),(130 130,130 70,160 40,180 90,140 ↩
     80),(50 40,70 40,80 70,60 60,50 40))
```

Figure



Vereinfachung eines MultiPolygons unter Beibehaltung der Topologie von Schalen und Löchern.

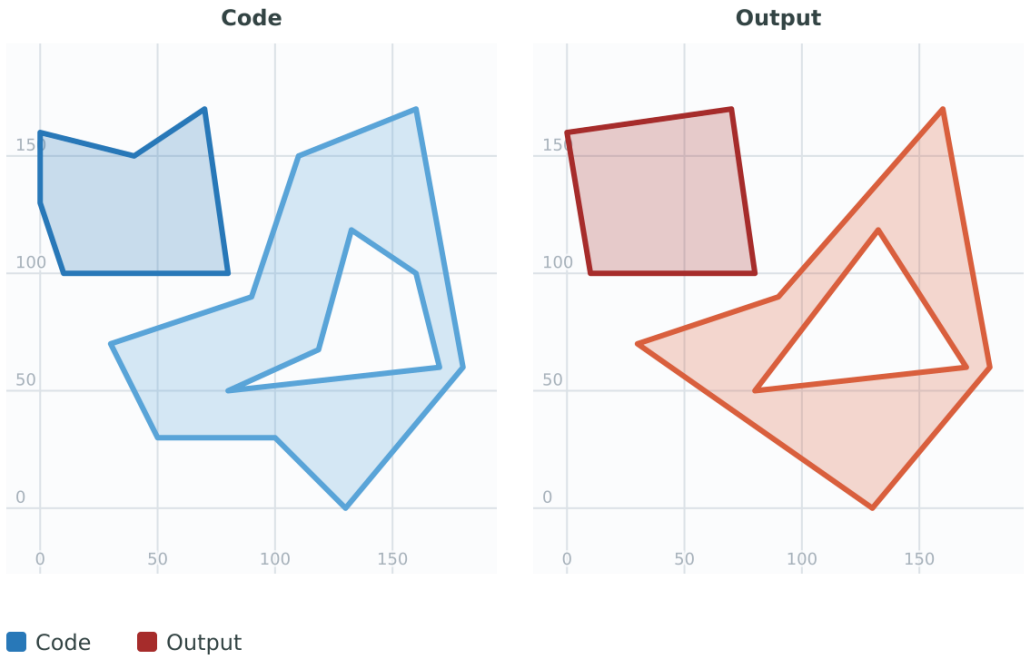
Code

```
SELECT ST_SimplifyPreserveTopology('MULTIPOLYGON (((90 110,80 180,50 160,10 170,10 140,20 ↩
    110,90 110)),((40 80,100 100,120 160,170 180,190 70,140 10,110 40,60 40,40 80),(180 ↩
    70,170 110,142.5 128.5,128.5 77.5,90 60,180 70)))'::geometry,
40);
```

Output

```
MULTIPOLYGON(((90 110,80 180,10 170,20 110,90 110)),((40 80,100 100,170 180,190 70,140 10,40 80),(180 70,142.5 128.5,90 ↩
      60,180 70)))
```

Figure



Siehe auch

ST_Simplify, ST_SimplifyVW, ST_CoverageSimplify

7.14.25 ST_SimplifyPolygonHull

ST_SimplifyPolygonHull — Berechnet eine vereinfachte topologieerhaltende äußere oder innere Hülle einer polygonalen Geometrie.

Synopsis

geometry **ST_SimplifyPolygonHull**(geometry param_geom, float vertex_fraction, boolean is_outer = true);

Beschreibung

Berechnet eine vereinfachte topologieerhaltende äußere oder innere Hülle einer polygonalen Geometrie. Eine äußere Hülle deckt die Eingabegeometrie vollständig ab. Eine innere Hülle wird vollständig von der Eingabegeometrie abgedeckt. Das Ergebnis ist eine polygonale Geometrie, die durch eine Teilmenge der Eingabepunkte gebildet wird. MultiPolygone und Löcher werden behandelt und führen zu einem Ergebnis mit derselben Struktur wie die Eingabe.

Die Verringerung der Scheitelpunktanzahl wird durch den Parameter `vertex_fraction` gesteuert, der eine Zahl im Bereich von 0 bis 1 ist. Niedrigere Werte führen zu einfacheren Ergebnissen mit einer geringeren Anzahl von Scheitelpunkten und einer geringeren Konkavität. Ein Scheitelpunktanteil von 1,0 ergibt sowohl für äußere als auch für innere Hüllen die ursprüngliche Geometrie. Für äußere Hüllen ergibt ein Wert von 0,0 die konvexe Hülle (für ein einzelnes Polygon); für innere Hüllen ergibt er ein Dreieck.

Bei der Vereinfachung werden nach und nach die konkaven Ecken entfernt, die die geringste Fläche enthalten, bis das Ziel für die Anzahl der Eckpunkte erreicht ist. Er verhindert, dass sich Kanten kreuzen, so dass das Ergebnis immer eine gültige polygonale Geometrie ist.

Um bessere Ergebnisse bei Geometrien zu erzielen, die relativ lange Liniensegmente enthalten, kann es notwendig sein, die Eingabe zu "segmentieren", wie unten gezeigt.

Wird vom GEOS Modul ausgeführt

Verfügbarkeit: 3.3.0.

Erfordert GEOS >= 3.11.0.

Beispiele

Outer hull of a Polygon.

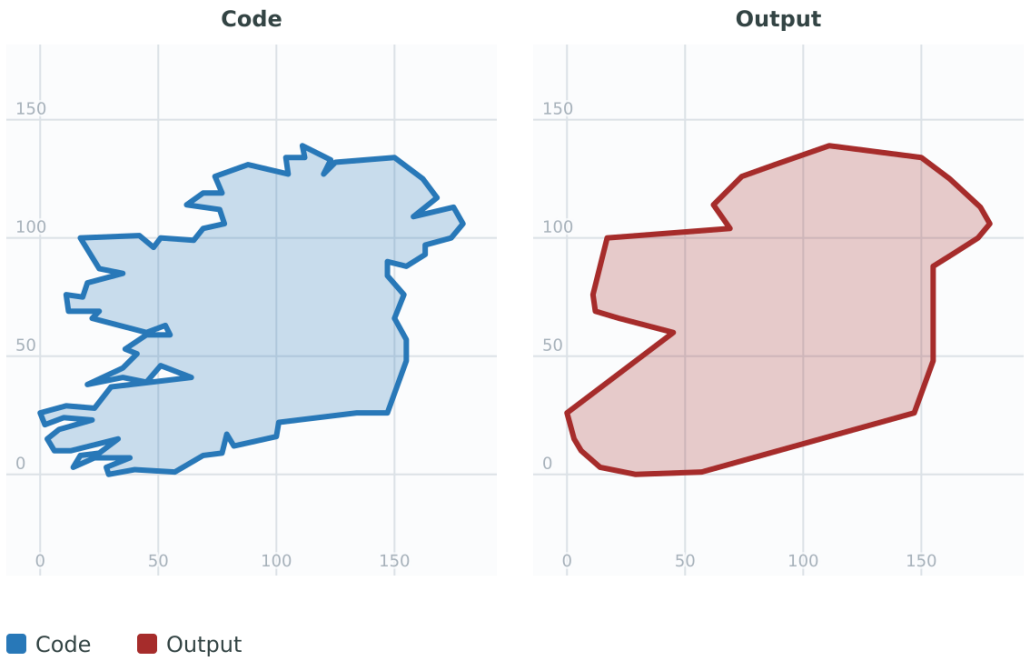
Code

```
SELECT ST_SimplifyPolygonHull(
  'POLYGON ((131 158,136 163,161 165,173 156,179 148,169 140,186 144,190 137,185 131,174 ↩
    128,174 124,166 119,158 121,158 115,165 107,161 97,166 88,166 79,158 57,145 57,112 ↩
    53,111 47,93 43,90 48,88 40,80 39,68 32,51 33,40 31,39 34,49 38,34 38,25 34,28 39,36 ↩
    40,44 46,24 41,17 41,14 46,19 50,33 54,21 55,13 52,11 57,22 60,34 59,41 68,75 72,62 ↩
    77,56 70,46 72,31 69,46 76,52 82,47 84,56 90,66 90,64 94,56 91,33 97,36 100,23 100,22 ↩
    107,29 106,31 112,46 116,36 118,28 131,53 132,59 127,62 131,76 130,80 135,89 137,87 ↩
    143,73 145,80 150,88 150,85 157,99 162,116 158,115 165,123 165,122 170,134 164,131 158)) ↩
    '::geometry,
  0.3
);
```

Output

```
POLYGON((161 165,173 156,186 144,190 137,185 131,174 124,166 119,166 79,158 57,68 32,40 31,25 34,17 41,14 46,11 57,56 ↩
    91,33 97,23 100,22 107,28 131,80 135,73 145,85 157,99 162,122 170,161 165))
```

Figure



Inner hull of a Polygon.

Code

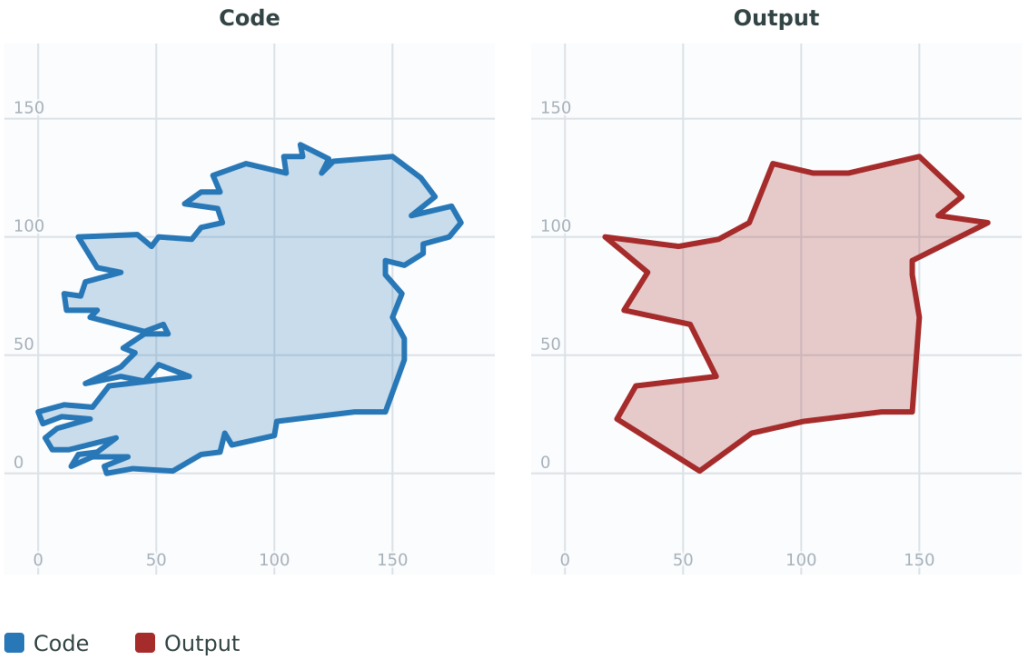
```
SELECT ST_SimplifyPolygonHull(
```

```
  'POLYGON ((131 158,136 163,161 165,173 156,179 148,169 140,186 144,190 137,185 131,174 ↩
    128,174 124,166 119,158 121,158 115,165 107,161 97,166 88,166 79,158 57,145 57,112 ↩
    53,111 47,93 43,90 48,88 40,80 39,68 32,51 33,40 31,39 34,49 38,34 38,25 34,28 39,36 ↩
    40,44 46,24 41,17 41,14 46,19 50,33 54,21 55,13 52,11 57,22 60,34 59,41 68,75 72,62 ↩
    77,56 70,46 72,31 69,46 76,52 82,47 84,56 90,66 90,64 94,56 91,33 97,36 100,23 100,22 ↩
    107,29 106,31 112,46 116,36 118,28 131,53 132,59 127,62 131,76 130,80 135,89 137,87 ↩
    143,73 145,80 150,88 150,85 157,99 162,116 158,115 165,123 165,122 170,134 164,131 158)) ↩
    '::geometry,
  0.3, false
);
```

Output

```
POLYGON((131 158,116 158,99 162,89 137,76 130,59 127,28 131,46 116,36 100,64 94,75 72,41 68,33 54,68 32,90 48,112 ↩
    53,145 57,158 57,161 97,158 115,158 121,190 137,169 140,179 148,161 165,131 158))
```

Figure

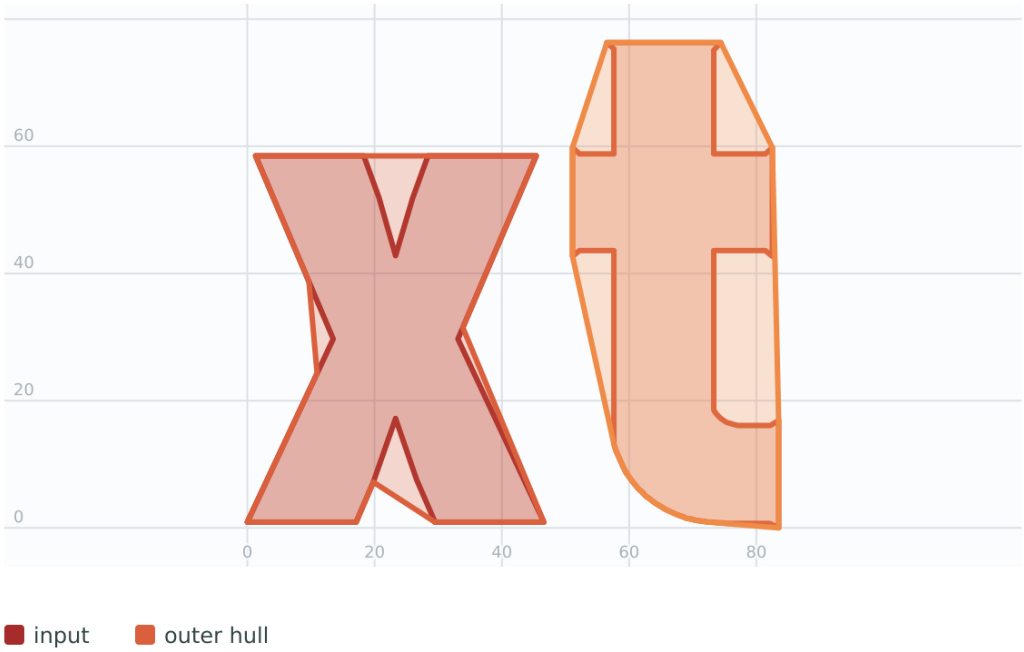


Outer hull simplification of a MultiPolygon, with segmentization.

Code

```
WITH letters AS (
  SELECT ST_Letters('xt') AS geom
)
SELECT geom AS input,
  ST_SimplifyPolygonHull(ST_Segmentize(geom, 2.0), 0.1) AS outer_hull
FROM letters;
```

Figure



Siehe auch

ST_ConvexHull, ST_SimplifyVW, ST_ConcaveHull, ST_Segmentize

7.14.26 ST_SimplifyVW

ST_SimplifyVW — Liefert eine vereinfachte Darstellung einer Geometrie unter Verwendung des Visvalingam-Whyatt-Algorithmus

Synopsis

geometry **ST_SimplifyVW**(geometry geom, float tolerance);

Beschreibung

Returns a simplified representation of a geometry using the Visvalingam-Whyatt algorithm. The simplification `tolerance` is an area value, in squared units of the input SRS. Simplification removes vertices which form "corners" with area less than the tolerance. The result may not be valid even if the input is.

Die Funktion kann mit jeder Art von Geometrie (einschließlich GeometryCollections) aufgerufen werden, aber nur Linien- und Polygonelemente werden vereinfacht. Die Endpunkte der linearen Geometrie bleiben erhalten.



Anmerkung
Die zurückgegebene Geometrie kann ihre Einfachheit verlieren (siehe ST_IsSimple), die Topologie bleibt möglicherweise nicht erhalten, und polygonale Ergebnisse können ungültig sein (siehe ST_IsValid). Verwenden Sie ST_SimplifyPreserveTopology, um die Topologie zu erhalten und die Gültigkeit sicherzustellen. ST_CoverageSimplify erhält ebenfalls die Topologie und die Gültigkeit.

This is a general-purpose simplification function. It may remove narrow polygon spikes when their effective area is below the tolerance, but it is not a dedicated spike-removal or polygon-repair operation. Use ST_IsValid to check the result and ST_MakeValid to repair invalid polygonal output if needed.

Diese Funktion erhält keine gemeinsamen Grenzen zwischen Polygonen. Verwenden Sie ST_CoverageSimplify, wenn dies erforderlich ist.

Diese Funktion kann mit 3D umgehen und die dritte Dimension beeinflusst auch das Ergebnis.

Verfügbarkeit: 2.2.0

Beispiele

This polygon has a narrow spike whose effective area is small enough to be removed by a tolerance of 20 square units.

Code

```
SELECT geom AS original, ST_SimplifyVW(geom, 20) AS simplified
FROM (SELECT 'POLYGON((0 0,10 0,10 10,5.2 10,5 16,4.8 10,0 10,0 0))'::geometry AS geom) AS t ↩
    ;
```

Output

```
POLYGON((0 0,10 0,10 10,5.2 10,5 16,4.8 10,0 10,0 0)) | POLYGON((0 0,10 0,10 10,0 10,0 0))
```

Figure



A LineString is simplified with a minimum-area tolerance of 30. For a linear simplification using ST_Simplify, a comparable tolerance would be a length in input-SRS units, so the value here is roughly that linear tolerance squared.

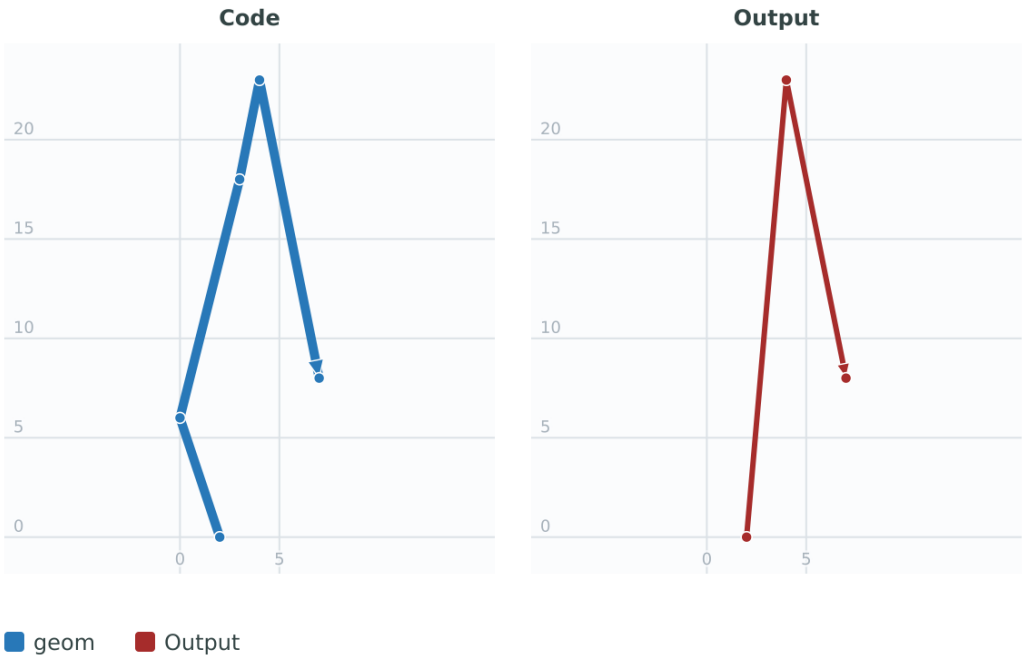
Code

```
SELECT ST_SimplifyVW(geom, 30) simplified
  FROM (SELECT 'LINESTRING(5 2,3 8,6 20,7 25,10 10)'::geometry AS geom) AS t;
```

Output

```
LINESTRING(5 2,7 25,10 10)
```

Figure



Vereinfachung einer Linie.

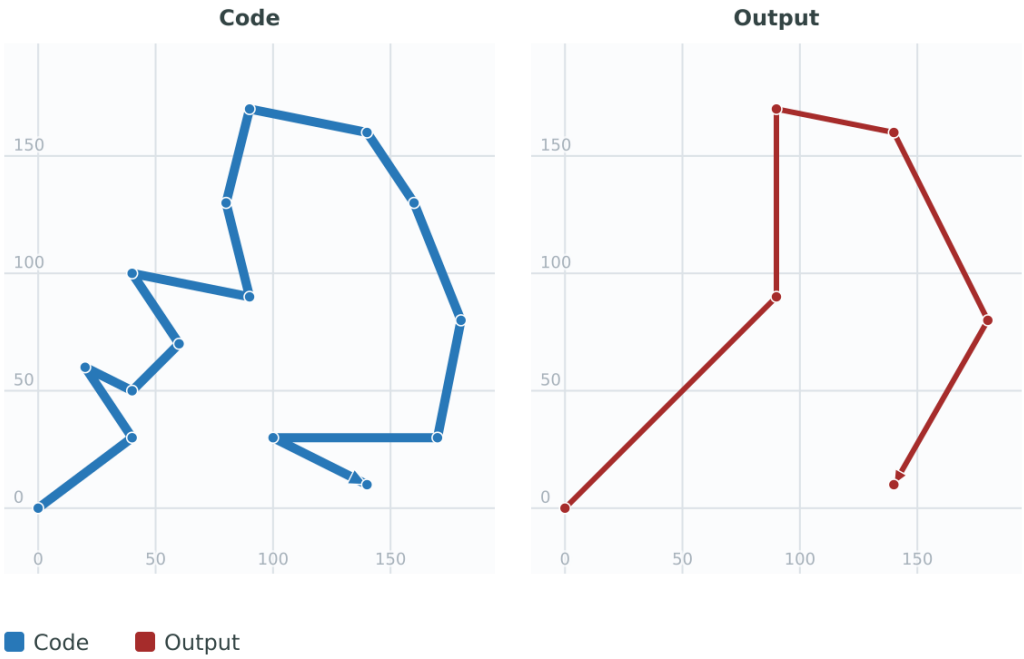
Code

```
SELECT ST_SimplifyVW('LINESTRING (10 10,50 40,30 70,50 60,70 80,50 110,100 100,90 140,100 ↩
    180,150 170,170 140,190 90,180 40,110 40,150 20)'::geometry,
1600);
```

Output

```
LINESTRING(10 10,100 100,100 180,150 170,190 90,150 20)
```

Figure



Vereinfachung eines Polygons.

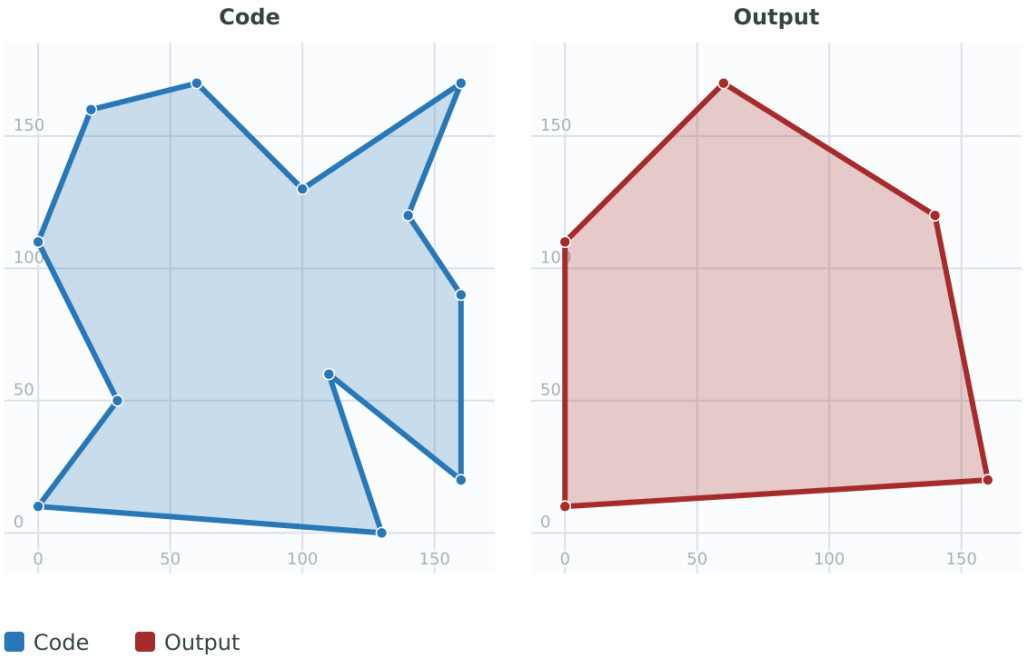
Code

```
SELECT ST_SimplifyVW('POLYGON ((20 20,50 60,20 120,40 170,80 180,120 140,180 180,160 130,180 ↩
    100,180 30,130 70,150 10,20 20))'::geometry,
1600);
```

Output

```
POLYGON((20 20,20 120,80 180,160 130,180 30,20 20))
```

Figure



Siehe auch

ST_SetEffectiveArea, ST_Simplify, ST_SimplifyPreserveTopology, ST_CoverageSimplify, Topologie ST_Simplify

7.14.27 ST_SetEffectiveArea

ST_SetEffectiveArea — Legt die effektive Fläche für jeden Scheitelpunkt unter Verwendung des Visvalingam-Whyatt-Algorithmus fest.

Synopsis

geometry **ST_SetEffectiveArea**(geometry geom, float threshold = 0, integer set_area = 1);

Beschreibung

Setzt die Nutzfläche für jeden Knoten. Verwendet den Visvalingam-Whyatt Algorithmus. Die Nutzfläche wird als M-Wert des Knoten gespeichert. Wird der optionale Parameter "threshold" verwendet, so wird eine vereinfachte Geometrie zurückgegeben, die nur jene Knoten enthält, deren Nutzfläche größer oder gleich dem Schwellenwert ist.

Diese Funktion kann für die serverseitige Vereinfachung, mittels eines Schwellenwerts verwendet werden. Eine andere Möglichkeit besteht darin, einen Schwellenwert von null anzugeben. In diesem Fall wird die gesamte Geometrie inklusive der Nutzflächen als M-Werte zurückgegeben, welche dann am Client für eine rasche Vereinfachung genutzt werden können.

Tut zurzeit nur mit (Multi)Lines und (Multi)Polygons etwas, kann aber gefahrlos mit jedem geometrischen Datentyp verwendet werden. Da die Vereinfachung auf einer Objekt zu Objekt Basis passiert, können Sie diese Funktion auch mit einer Sammelgeometrie speisen.



Anmerkung
Bemerke, dass die zurückgegebene Geometrie ihre Simplizität verlieren kann (siehe ST_IsSimple).



Anmerkung
Beachten Sie bitte, das die Topologie möglicherweise nicht erhalten bleibt und ungültige Geometrien entstehen können. Benutzen Sie bitte (see ST_SimplifyPreserveTopology) um die Topologie zu erhalten.



Anmerkung
Die Ausgabegeometrie verliert die gesamte vorhandene Information über die M-Werte



Anmerkung
Diese Funktion kann mit 3D umgehen und die dritte Dimension beeinflusst auch die tatsächliche Fläche

Verfügbarkeit: 2.2.0

Beispiele

Berechnung der Nutzfläche eines Linienzugs. Da wir einen Schwellenwert von null verwenden, werden alle Knoten der Eingabegeomettrie zurückgegeben.

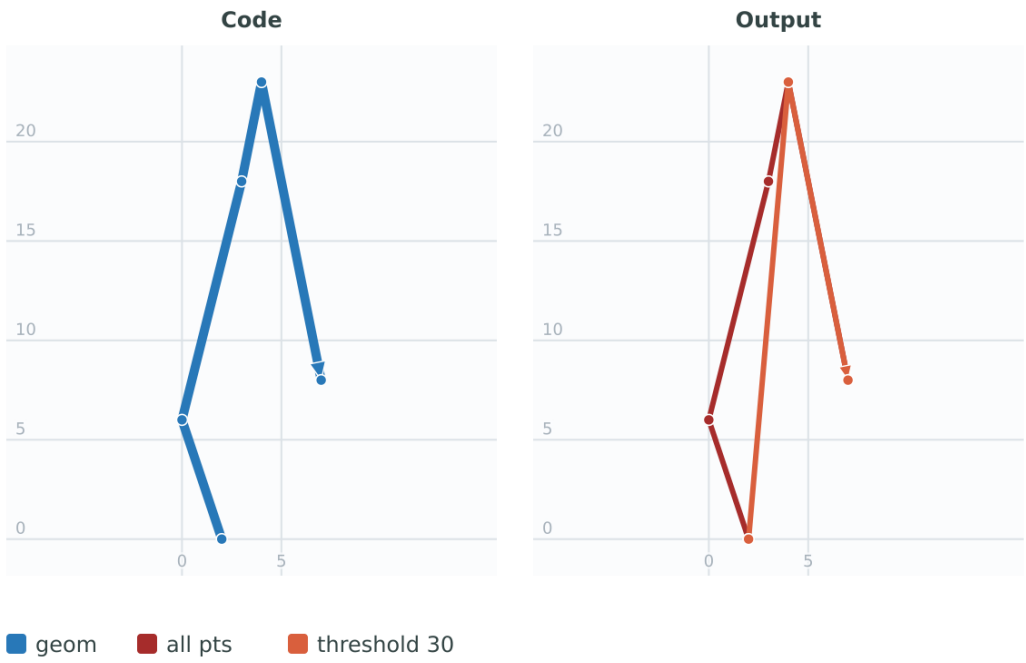
Code

```
SELECT ST_SetEffectiveArea(geom) AS all_pts,
       ST_SetEffectiveArea(geom, 30) AS threshold_30
FROM (SELECT  'LINESTRING(5 2,3 8,6 20,7 25,10 10)'::geometry geom) As foo;
```

Output

```
-[ RECORD 1 ]----------------
all_pts      | LINESTRING M (5 2 3.4e+38,3 8 29,6 20 1.5,7 25 49.5,10 10 3.4e+38)
threshold_30 | LINESTRING M (5 2 3.4e+38,7 25 49.5,10 10 3.4e+38)
```

Figure



Siehe auch

ST_SimplifyVW

7.14.28 ST_TriangulatePolygon

ST_TriangulatePolygon — Berechnet die eingeschränkte Delaunay-Triangulation von Polygonen

Synopsis

geometry **ST_TriangulatePolygon**(geometry geom);

Beschreibung

Berechnet die eingeschränkte Delaunay-Triangulation von Polygonen. Löcher und Multipolygone werden unterstützt.

Die "eingeschränkte Delaunay-Triangulation" eines Polygons ist eine Menge von Dreiecken, die aus den Scheitelpunkten des Polygons gebildet werden und es genau abdecken, mit dem maximalen Gesamtinnenwinkel aller möglichen Triangulationen. Sie liefert die "beste Qualität" der Triangulation des Polygons.

Verfügbarkeit: 3.3.0.

Erfordert GEOS >= 3.11.0.

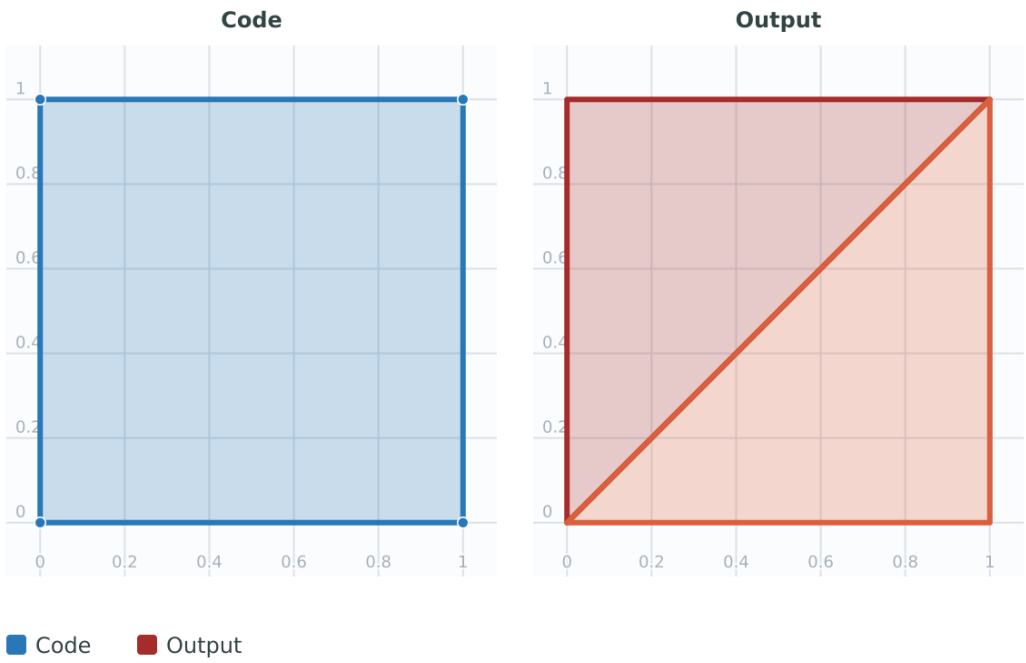
Beispiele

Triangulierung eines Quadrats.

Code

```
SELECT ST_TriangulatePolygon('POLYGON((0 0,0 1,1 1,1 0,0 0))');
```

Figure

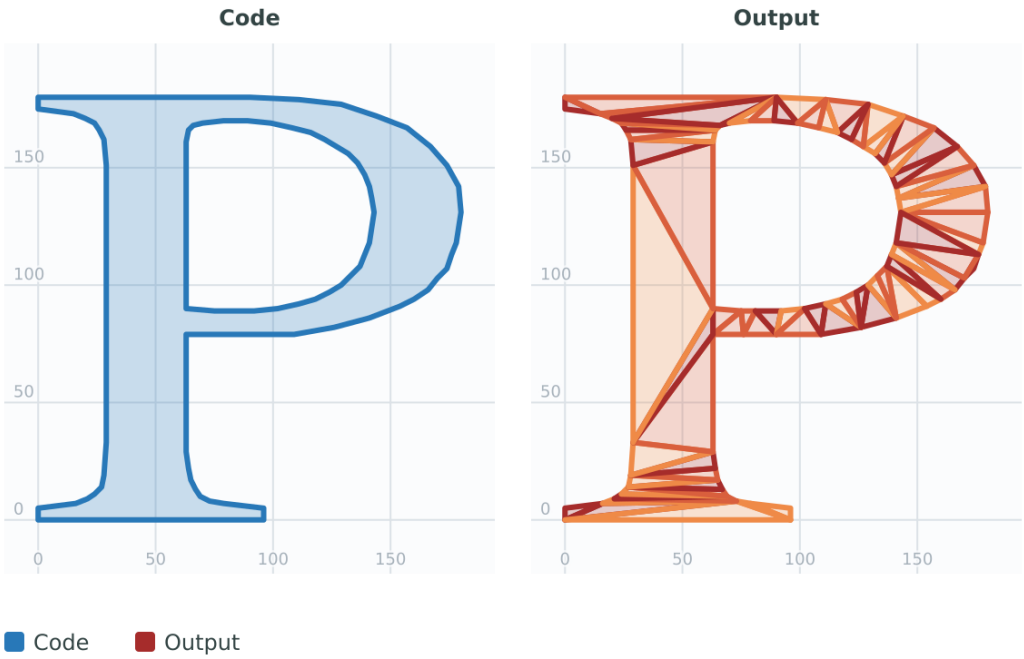


Triangulation des Buchstabens P.

Code

```
SELECT ST_TriangulatePolygon('POLYGON ((26 17,31 19,34 21,37 24,38 29,39 43,39 161,38 172,36 ↩
     176,34 179,30 181,25 183,10 185,10 190,100 190,121 189,139 187,154 182,167 177,177  ↩
    169,184 161,189 152,190 141,188 128,186 123,184 117,180 113,176 108,170 104,164 101,151  ↩
    96,136 92,119 89,100 89,86 89,73 89,73 39,74 32,75 27,77 23,79 20,83 18,89 17,106 15,106 ↩
     10,10 10,10 15,26 17),(152 147,151 152,149 157,146 162,142 166,137 169,132 172,126  ↩
    175,118 177,109 179,99 180,89 180,80 179,76 178,74 176,73 171,73 100,85 99,91 99,102  ↩
    99,112 100,121 102,128 104,134 107,139 110,143 114,147 118,149 123,151 128,153 141,152  ↩
    147))'
);
```

Figure

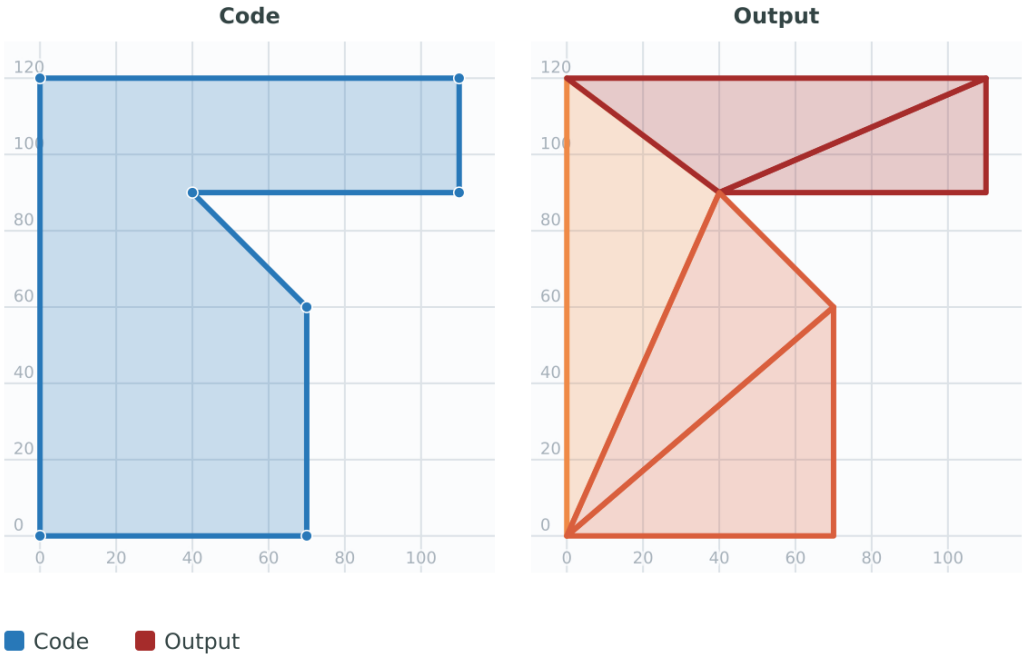


Triangulation of the polygon used in the CG_Tesselate example.

Code

```
SELECT ST_TriangulatePolygon('POLYGON (( 10 190,10 70,80 70,80 130,50 160,120 160,120 190,10 ↩
    190 ))'::geometry
);
```

Figure



Siehe auch

CG_ConstrainedDelaunayTriangles, ST_DelaunayTriangles, CG_Tesselate

7.14.29 ST_VoronoiLines

ST_VoronoiLines — Gibt die Grenzen des Voronoi-Diagramms der Eckpunkte einer Geometrie zurück.

Synopsis

geometry **ST_VoronoiLines**(geometry geom , float8 tolerance = 0.0 , geometry extend_to = NULL);

Beschreibung

Berechnet ein zweidimensionales Voronoi-Diagramm aus den Scheitelpunkten der angegebenen Geometrie und gibt die Grenzen zwischen den Zellen im Diagramm als MultiLineString zurück. Gibt null zurück, wenn die Eingabegeometrie null ist. Gibt eine leere Geometriesammlung zurück, wenn die Eingabegeometrie nur einen Scheitelpunkt enthält. Gibt eine leere Geometriesammlung zurück, wenn die `extend_to`-Hülle eine Fläche von Null hat.

Optionale Parameter:

- `Toleranz`: Der Abstand, innerhalb dessen Scheitelpunkte als gleichwertig betrachtet werden. Die Robustheit des Algorithmus kann durch Angabe einer Toleranzdistanz ungleich Null verbessert werden. (Voreinstellung = 0.0)
- `extend_to`: Wenn vorhanden, wird das Diagramm so erweitert, dass es die Hüllkurve der übergebenen Geometrie abdeckt, sofern diese nicht kleiner als die Standardhüllkurve ist (Standard = NULL, Standardhüllkurve ist die um etwa 50 % erweiterte Bounding Box der Eingabe).

Wird vom GEOS Modul ausgeführt

Verfügbarkeit: 2.3.0

Beispiele

Voronoi diagram lines with a tolerance of 30 units.

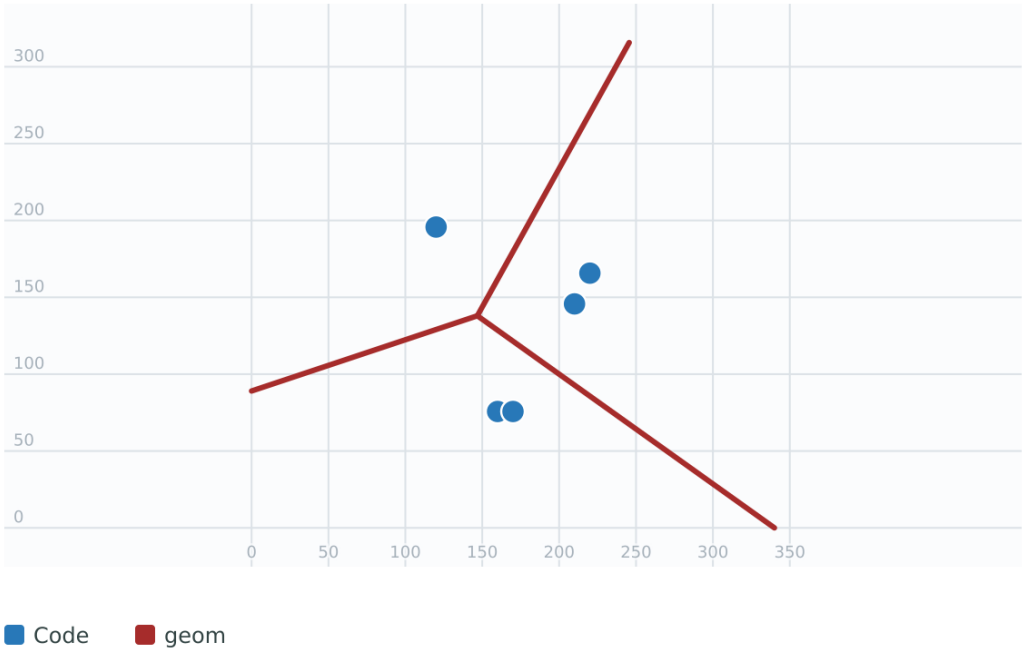
Code

```
SELECT ST_VoronoiLines('MULTIPOINT (50 30,60 30,100 100,10 150,110 120)'::geometry,
            30) AS geom;
```

Output

```
MULTILINESTRING((135.555555555556 270,36.8181818181818 92.2727272727273),(36.8181818181818 92.2727272727273,-110 ↩
    43.3333333333333),(230 -45.7142857142858,36.8181818181818 92.2727272727273))
```

Figure



Siehe auch

ST_DelaunayTriangles, ST_VoronoiPolygons

7.14.30 ST_VoronoiPolygons

ST_VoronoiPolygons — Gibt die Zellen des Voronoi-Diagramms der Scheitelpunkte einer Geometrie zurück.

Synopsis

geometry **ST_VoronoiPolygons**(geometry geom , float8 tolerance = 0.0 , geometry extend_to = NULL);

Beschreibung

Berechnet ein zweidimensionales Voronoi-Diagramm aus den Scheitelpunkten der angegebenen Geometrie. Das Ergebnis ist eine GEOMETRIE-SAMMLUNG von POLYGONEN, die einen Bereich abdeckt, der größer ist als die Ausdehnung der Eingabescheitelpunkte. Gibt null zurück, wenn die Eingabegeometrie null ist. Gibt eine leere Geometriesammlung zurück, wenn die Eingabegeometrie nur einen Scheitelpunkt enthält. Gibt eine leere Geometriesammlung zurück, wenn die `extend_to`-Hüllkurve eine Fläche von Null hat.

Optionale Parameter:

- `Toleranz`: Der Abstand, innerhalb dessen Scheitelpunkte als gleichwertig betrachtet werden. Die Robustheit des Algorithmus kann durch Angabe einer Toleranzdistanz ungleich Null verbessert werden. (Voreinstellung = 0.0)
- `extend_to`: Wenn vorhanden, wird das Diagramm so erweitert, dass es die Hüllkurve der übergebenen Geometrie abdeckt, sofern diese nicht kleiner als die Standardhüllkurve ist (Standard = NULL, Standardhüllkurve ist die um etwa 50 % erweiterte Bounding Box der Eingabe).

Wird vom GEOS Modul ausgeführt

Verfügbarkeit: 2.3.0

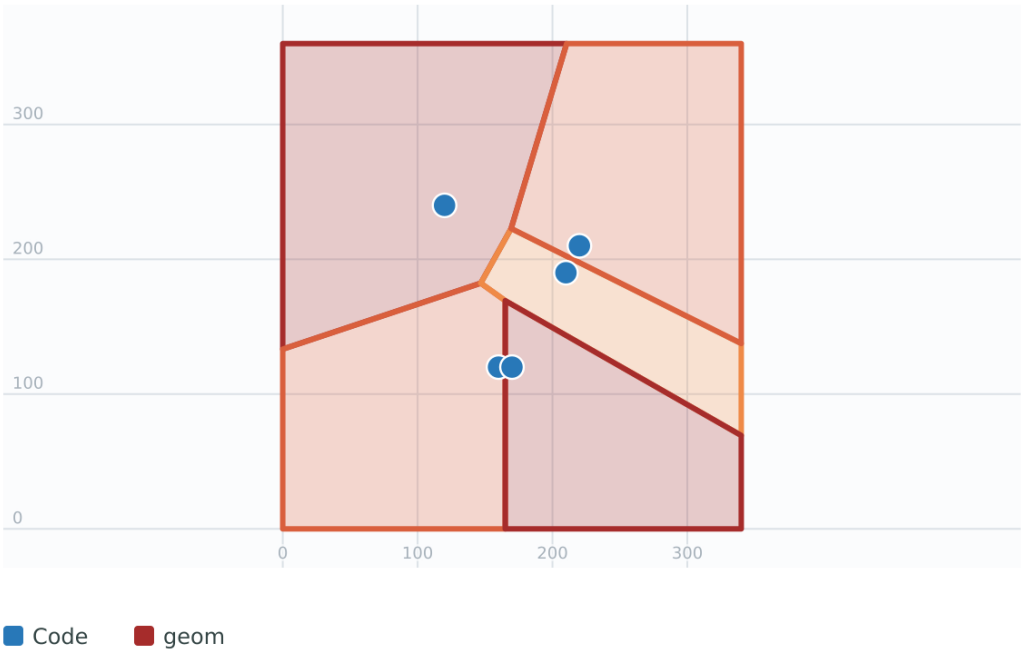
Beispiele

Points overlaid on top of the Voronoi diagram.

Code

```
SELECT ST_VoronoiPolygons('MULTIPOINT (50 30,60 30,100 100,10 150,110 120)'::geometry
            ) AS geom;
```

Figure

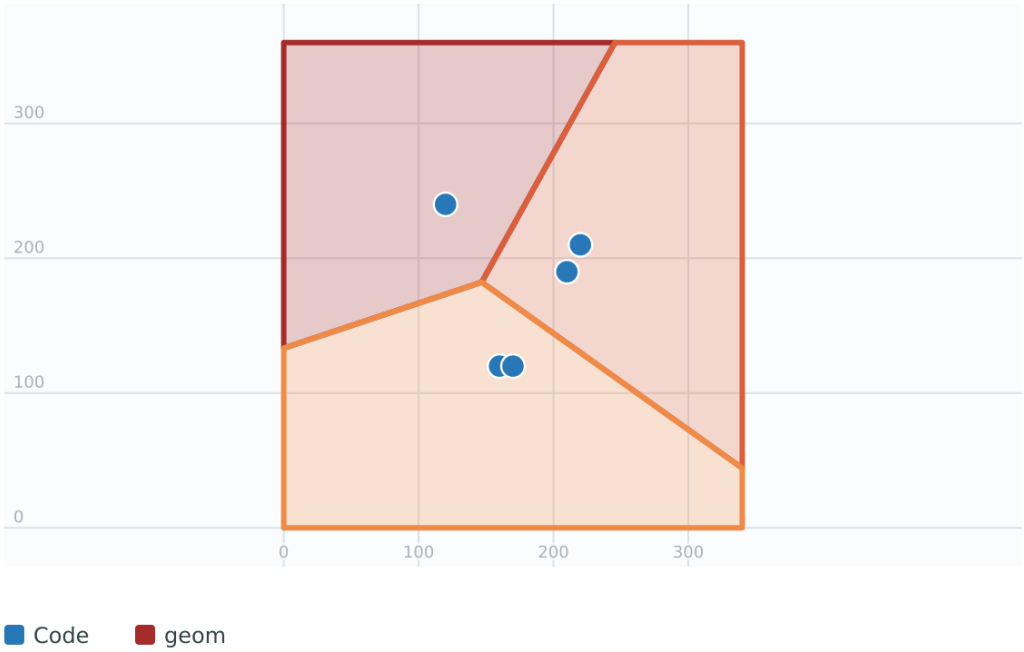


Voronoi diagram with a tolerance of 30 units.

Code

```
SELECT ST_VoronoiPolygons('MULTIPOINT (50 30,60 30,100 100,10 150,110 120)'::geometry,
            30) AS geom;
```

Figure



Siehe auch

ST_DelaunayTriangles, ST_VoronoiLines

7.15 Deckungen

7.15.1 ST_CoverageInvalidEdges

ST_CoverageInvalidEdges — Fensterfunktion, die Stellen findet, an denen die Polygone keine gültige Abdeckung bilden.

Synopsis

geometry **ST_CoverageInvalidEdges**(geometry winset geom, float8 tolerance = 0);

Beschreibung

Eine Fensterfunktion, die prüft, ob die Polygone in der Fensterpartition eine gültige polygonale Abdeckung bilden. Sie gibt lineare Indikatoren zurück, die die Lage der ungültigen Kanten (falls vorhanden) in jedem Polygon anzeigen.

Eine Menge gültiger Polygone ist eine gültige Abdeckung, wenn die folgenden Bedingungen erfüllt sind:

- **Nicht überlappend** - Polygone überlappen sich nicht (ihre Innenräume schneiden sich nicht)
- **Edge-Matched** - Scheitelpunkte entlang gemeinsamer Kanten sind identisch

Als Fensterfunktion wird für jedes Eingabepolygon ein Wert zurückgegeben. Für Polygone, die eine oder mehrere der Gültigkeitsbedingungen verletzen, ist der Rückgabewert ein MULTILINESTRING, der die problematischen Kanten enthält. Flächendeckend gültige Polygone geben den Wert NULL zurück. Nicht-polygonale oder leere Geometrien liefern ebenfalls NULL-Werte.

Die Bedingungen erlauben es, dass eine gültige Abdeckung Löcher (Lücken zwischen Polygonen) enthält, solange die umgebenden Polygone kantenangepasst sind. Sehr schmale Lücken sind jedoch oft unerwünscht. Wenn der Parameter `tolerance` mit einem Abstand ungleich Null angegeben wird, werden auch Kanten, die engere Lücken bilden, als ungültig zurückgegeben.

Die Polygone, die auf die Gültigkeit der Abdeckung geprüft werden, müssen ebenfalls gültige Geometrien sein. Dies kann mit ST_IsValid überprüft werden.

Verfügbarkeit: 3.4.0

Benötigt GEOS >= 3.12.0

Beispiele

Invalid edges caused by overlap and non-matching vertices.

Code

```
WITH coverage(id, geom) AS (VALUES
  (1, 'POLYGON ((10 190,30 160,40 110,100 70,120 10,10 10,10 190))'::geometry),
  (2, 'POLYGON ((100 190,10 190,30 160,40 110,50 80,74 110.5,100 130,140 120,140 160,100 ↩
    190))'::geometry),
  (3, 'POLYGON ((140 190,190 190,190 80,140 80,140 190))'::geometry),
  (4, 'POLYGON ((180 40,120 10,100 70,140 80,190 80,180 40))'::geometry)
)
SELECT id, ST_CoverageInvalidEdges(geom) OVER ()
  FROM coverage;
```

Figure



This example tests the entire table for coverage validity.

Code

```
SELECT true = ALL (
    SELECT ST_CoverageInvalidEdges(geom) OVER () IS NULL
    FROM coverage
    );
```

Siehe auch

ST_IsValid, ST_CoverageUnion, ST_CoverageClean, ST_CoverageSimplify

7.15.2 ST_CoverageClean

ST_CoverageClean — Computes a clean (edge matched, non-overlapping, gap-cleared) polygonal coverage, given a non-clean input.

Synopsis

geometry **ST_CoverageClean**(geometry winset geom, float8 gapMaximumWidth = 0, float8 snappingDistance = -1, text overlapMergeStrategy = 'MERGE_LONGEST_BORDER');

Beschreibung

A window function which adjusts the edges of a set of valid polygonal geometries to produce a clean coverage. Cleaning involves:

- snapping vertices and edges to remove small discrepancies and ensure common edges are identically noded
- merging overlaps into a parent polygon
- merging narrow gaps into adjacent polygons

`gapMaximumWidth` controls which gaps between polygons are merged. Gaps with width <= this distance are merged into an adjacent polygon.

`snappingDistance` controls snapping of vertices and edges. The default (-1) automatically determines a snapping distance based on the input extent. Set to 0.0 to turn off snapping.

`overlapMergeStrategy` specifies how overlaps are merged into a parent polygon:

- `MERGE_LONGEST_BORDER` - merges into polygon with longest common border
- `MERGE_MAX_AREA` - merges into polygon with maximum area
- `MERGE_MIN_AREA` - merges into polygon with minimum area
- `MERGE_MIN_INDEX` - merges into polygon with smallest input index (defined by order of input polygons)

The result is a clean polygonal coverage that will pass validation by ST_CoverageInvalidEdges and can be input to coverage processing functions.



Anmerkung
To aid in determining a maximum gap width, gaps can be computed by cleaning with `gapMaximumWidth = > 0` and using ST_CoverageUnion to union the result coverage. Holes in the union correspond to gaps in the original dataset. Gap widths can be measured by extracting the holes as polygons and running ST_MaximumInscribedCircle on them; the gap width is twice the computed radius.

Availability: 3.6.0 - requires GEOS >= 3.14.0

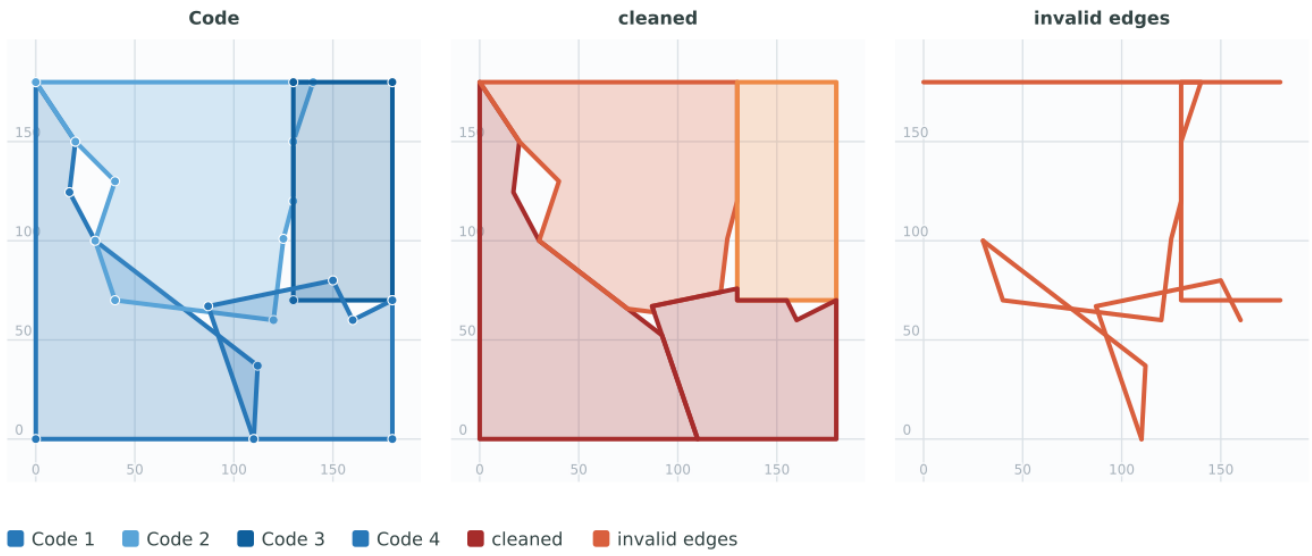
Beispiele

Clean a coverage containing overlaps and narrow gaps, and compare the cleaned polygons with the original invalid edges.

Code

```
WITH coverage(id, geom) AS (VALUES
  (1, 'POLYGON ((10 190,30 160,27 134.5,40 110,122 47,120 10,10 10,10 190))'::geometry),
  (2, 'POLYGON ((150 190,10 190,30 160,50 140,40 110,50 80,130 70,135 111,140 130,140 ↩
    160,150 190))'::geometry),
  (3, 'POLYGON ((140 190,190 190,190 80,140 80,140 190))'::geometry),
  (4, 'POLYGON ((190 10,120 10,97 77,160 90,170 70,190 80,190 10))'::geometry)
), processed AS (
  SELECT id,
         ST_CoverageClean(geom, 1) OVER () AS cleaned,
         ST_CoverageInvalidEdges(geom) OVER () AS invalid_edges
  FROM coverage
)
SELECT ST_Collect(cleaned ORDER BY id) AS cleaned,
       ST_Collect(invalid_edges ORDER BY id) AS invalid_edges
FROM processed;
```

Figure



Siehe auch

ST_CoverageInvalidEdges, ST_CoverageUnion, ST_CoverageSimplify, ST_MaximumInscribedCircle

7.15.3 ST_CoverageSimplify

ST_CoverageSimplify — Fensterfunktion, die die Kanten einer polygonalen Abdeckung vereinfacht.

Synopsis

geometry **ST_CoverageSimplify**(geometry winset geom, float8 tolerance, boolean simplifyBoundary = true);

Beschreibung

Eine Fensterfunktion, die die Kanten von Polygonen in einer polygonalen Abdeckung vereinfacht. Durch die Vereinfachung bleibt die Topologie der Abdeckung erhalten. Das bedeutet, dass die vereinfachten Ausgabepolygone entlang gemeinsamer Kanten konsistent sind und immer noch eine gültige Abdeckung bilden.

Die Vereinfachung verwendet eine Variante des Visvalingam-Whyatt-Algorithmus. Der Parameter *Toleranz* hat die Einheit Abstand und ist ungefähr gleich der Quadratwurzel der zu vereinfachenden Dreiecksflächen.

Um nur die "internen" Kanten der Abdeckung zu vereinfachen (diejenigen, die von zwei Polygonen geteilt werden), setzen Sie den Parameter *simplifyBoundary* auf false.

Use this function to simplify adjacent polygon layers when shared boundaries must remain edge-matched. For a single geometry where only self-intersection and ring validity matter, see ST_SimplifyPreserveTopology.



Anmerkung
Wenn die Eingabe keine gültige Abdeckung ist, kann es zu unerwarteten Artefakten in der Ausgabe kommen (z. B. Überschneidungen von Grenzen oder getrennte Grenzen, die gemeinsam zu sein schienen). Verwenden Sie ST_CoverageInvalidEdges, um festzustellen, ob eine Abdeckung gültig ist.

Verfügbarkeit: 3.4.0

Benötigt GEOS >= 3.12.0

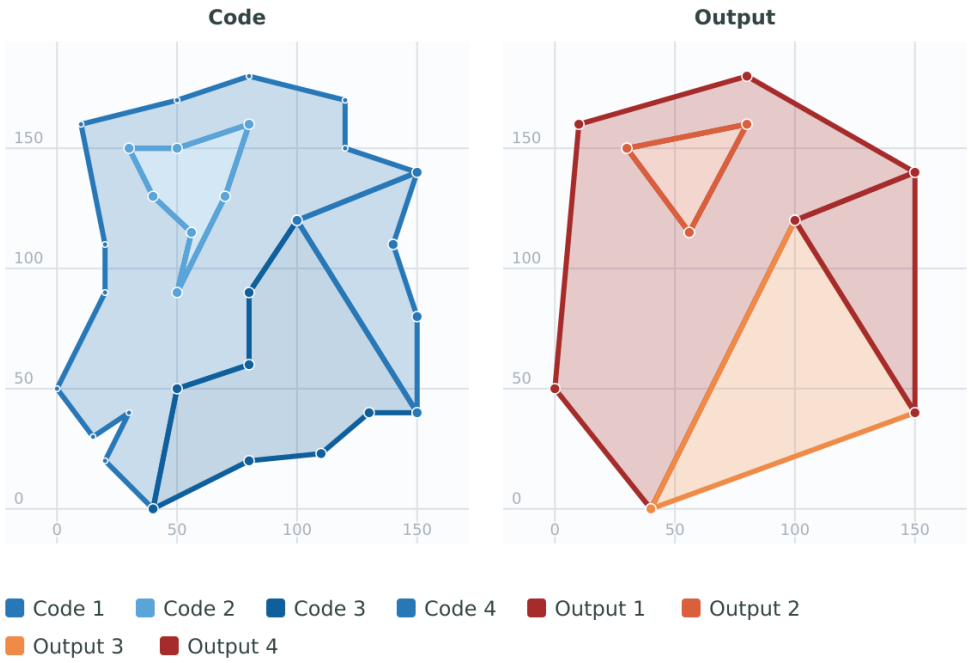
Beispiele

Simplify a polygonal coverage while preserving shared boundaries.

Code

```
WITH coverage(id, geom) AS (VALUES
  (1, 'POLYGON ((160 150,110 130,90 100,90 70,60 60,50 10,30 30,40 50,25 40,10 60,30 100,30 ↩
    120,20 170,60 180,90 190,130 180,130 160,160 150),(40 160,50 140,66 125,60 100,80 140,90 ↩
     170,60 160,40 160))'::geometry),
  (2, 'POLYGON ((40 160,60 160,90 170,80 140,60 100,66 125,50 140,40 160))'::geometry),
  (3, 'POLYGON ((110 130,160 50,140 50,120 33,90 30,50 10,60 60,90 70,90 100,110 130))':: ↩
    geometry),
  (4, 'POLYGON ((160 150,150 120,160 90,160 50,110 130,160 150))'::geometry)
)
SELECT id, ST_CoverageSimplify(geom, 30) OVER ()
  FROM coverage;
```

Figure



Siehe auch

ST_CoverageInvalidEdges, ST_CoverageUnion, ST_CoverageClean

7.15.4 ST_CoverageEdges

ST_CoverageEdges — Computes the unique edges of a polygonal coverage.

Synopsis

geometry **ST_CoverageEdges**(geometry geom, integer edgetype = 0);
geometry **ST_CoverageEdges**(geometry[] geoms, integer edgetype = 0);

Beschreibung

Returns a MultiLineString representing the unique edges of a polygonal coverage. A polygonal coverage is a set of non-overlapping polygons where adjacent polygons have matching vertices along shared edges.

The *edgetype* parameter can be used to select which edges are returned:

- **0 (ALL)** - all unique edges (default)
- **1 (EXTERIOR)** - only exterior edges (non-shared)
- **2 (INTERIOR)** - only interior edges (shared)

Availability: 3.7.0

Requires GEOS >= 3.15.0

Beispiele

This example returns all unique edges of a coverage.

Code

```
WITH coverage(geom) AS (
  VALUES (
    'GEOMETRYCOLLECTION(
      POLYGON((0 0,10 0,10 10,0 10,0 0)),
      POLYGON((10 0,20 0,20 10,10 10,10 0))
    )'::geometry
  )
)
SELECT geom AS input_coverage,
       ST_CoverageEdges(geom) AS all_edges
FROM coverage;
```

Output

```
-[ RECORD 1 ]--+-----------------------------------------------------------------------------------------------
input_coverage | GEOMETRYCOLLECTION(POLYGON((0 0,10 0,10 10,0 10,0 0)),POLYGON((10 0,20 0,20 10,10 10,10 0)))
all_edges      | MULTILINESTRING((10 0,10 10),(10 10,0 10,0 0,10 0),(10 0,20 0,20 10,10 10))
```

This example returns only the interior shared edges of a coverage.

Code

```
WITH coverage(geom) AS (
  VALUES (
    'GEOMETRYCOLLECTION(
      POLYGON((0 0,10 0,10 10,0 10,0 0)),
      POLYGON((10 0,20 0,20 10,10 10,10 0))
    )'::geometry
  )
)
SELECT geom AS input_coverage,
       ST_CoverageEdges(geom, 2) AS shared_edges
FROM coverage;
```

Output

```
-[ RECORD 1 ]--+-----------------------------------------------------------------------------------------
input_coverage | GEOMETRYCOLLECTION(POLYGON((0 0,10 0,10 10,0 10,0 0)),POLYGON((10 0,20 0,20 10,10 10,10 0)))
shared_edges   | MULTILINESTRING((10 0,10 10))
```

Siehe auch

ST_CoverageInvalidEdges, ST_CoverageSimplify, ST_CoverageUnion

7.15.5 ST_CoverageUnion

ST_CoverageUnion — Berechnet die Vereinigung einer Menge von Polygonen, die eine Abdeckung bilden, indem gemeinsame Kanten entfernt werden.

Synopsis

geometry **ST_CoverageUnion**(geometry set geom);

Beschreibung

Eine Aggregatfunktion, die einen Satz von Polygonen zu einer polygonalen Abdeckung zusammenfügt. Das Ergebnis ist eine polygonale Geometrie, die denselben Bereich wie die Abdeckung abdeckt. Diese Funktion liefert das gleiche Ergebnis wie ST_Union, nutzt aber die Struktur der Abdeckung, um die Vereinigung viel schneller zu berechnen.



Anmerkung
Wenn die Eingabe keine gültige Abdeckung ist, kann es zu unerwarteten Artefakten in der Ausgabe kommen (z. B. nicht verschmolzene oder überlappende Polygone). Verwenden Sie ST_CoverageInvalidEdges, um festzustellen, ob eine Abdeckung gültig ist.

Verfügbarkeit: 3.4.0 - erfordert GEOS >= 3.8.0

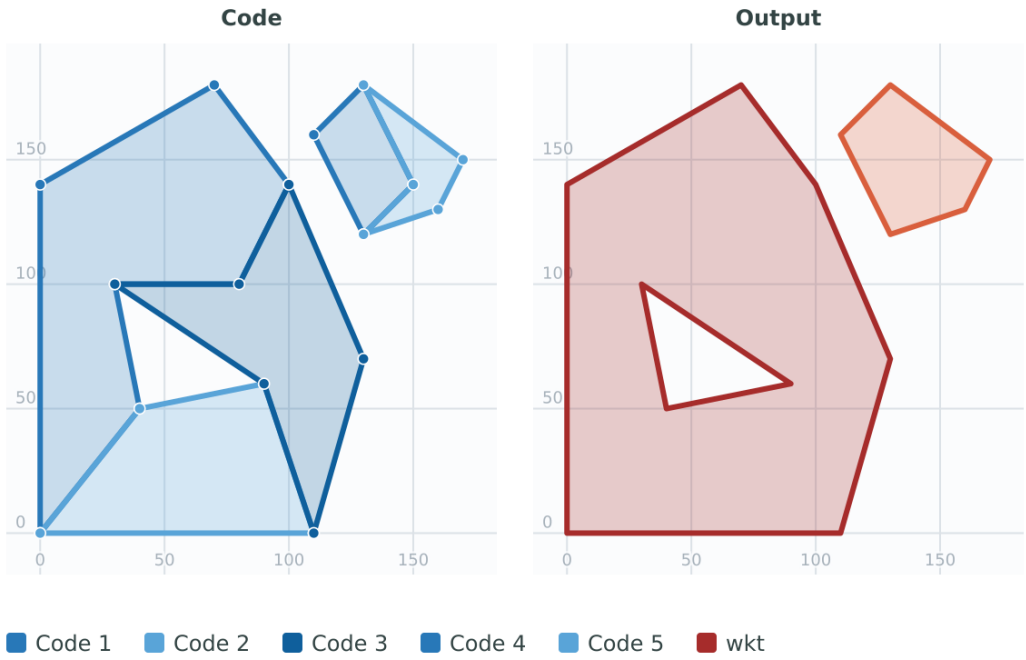
Beispiele

Union a polygonal coverage.

Code

```
WITH coverage(id, geom) AS (VALUES
  (1, 'POLYGON ((10 10,10 150,80 190,110 150,90 110,40 110,50 60,10 10))'::geometry),
  (2, 'POLYGON ((120 10,10 10,50 60,100 70,120 10))'::geometry),
  (3, 'POLYGON ((140 80,120 10,100 70,40 110,90 110,110 150,140 80))'::geometry),
  (4, 'POLYGON ((140 190,120 170,140 130,160 150,140 190))'::geometry),
  (5, 'POLYGON ((180 160,170 140,140 130,160 150,140 190,180 160))'::geometry)
)
SELECT ST_CoverageUnion(geom) AS wkt
  FROM coverage;
```

Figure



Siehe auch

ST_CoverageInvalidEdges, ST_CoverageSimplify, ST_CoverageClean, ST_Union

7.16 Affine Transformationen

7.16.1 ST_Affine

ST_Affine — Wenden Sie eine affine 3D-Transformation auf eine Geometrie an.

Synopsis

geometry **ST_Affine**(geometry geomA, float a, float b, float c, float d, float e, float f, float g, float h, float i, float xoff, float yoff, float zoff);
geometry **ST_Affine**(geometry geomA, float a, float b, float d, float e, float xoff, float yoff);

Beschreibung

Wendet eine affine 3D-Transformation auf die Geometrie an, um Dinge wie Verschieben, Drehen und Skalieren in einem Schritt durchzuführen.

Version 1: Der Aufruf

Code

```
ST_Affine(geom, a, b, c, d, e, f, g, h, i, xoff, yoff, zoff)
```

stellt die Transformationsmatrix

Code

```
/ a b c xoff \
| d e f yoff |
| g h i zoff |
\ 0 0 0 1 /
```

und die Scheitelpunkte werden wie folgt transformiert:

Code

```
x' = a*x + b*y + c*z + xoff
y' = d*x + e*y + f*z + yoff
z' = g*x + h*y + i*z + zoff
```

Alle nachstehenden Translations-/Skalierungsfunktionen werden durch eine solche affine Transformation ausgedrückt.

Version 2: Wendet eine 2d affine Transformation auf die Geometrie an. Der Aufruf

Code

```
ST_Affine(geom, a, b, d, e, xoff, yoff)
```

stellt die Transformationsmatrix

Code

```
/ a b 0 xoff \ / a b xoff \
| d e 0 yoff | rsp.  | d e yoff |
| 0 0 1 0 | \ 0 0 1 /
\ 0 0 0 1 /
```

und die Scheitelpunkte werden wie folgt transformiert:

Code

```
x' = a*x + b*y + xoff
y' = d*x + e*y + yoff
z' = z
```

Diese Methode ist ein Unterfall der obigen 3D-Methode.

Erweiterung: Mit 2.0.0 wurde die Unterstützung für polyedrische Oberflächen, Dreiecke und TIN eingeführt.

Verfügbarkeit: 1.1.2. Name geändert von Affine zu ST_Affine in 1.2.2



Anmerkung
Vor 1.3.4 ist diese Funktion abgestürzt, wenn die Geometrien CURVES enthalten. Dies wurde mit 1.3.4+ behoben

Diese Funktion unterstützt polyedrische Flächen.

Diese Funktion unterstützt Dreiecke und dreieckige unregelmäßige Netzoberflächen (TIN).

Diese Funktion unterstützt 3d und lässt den Z-Index nicht fallen.

Diese Methode unterstützt kreisförmige Strings und Kurven.

Beispiele

Rotate a 3D line 180 degrees about the z axis. Note this is long-hand for doing ST_Rotate();

Code

```
SELECT ST_Affine(geom, cos(pi()), -sin(pi()), 0, sin(pi()), cos(pi()), 0, 0, 0, 1, 0, 0, 0)  ↩
    As using_affine,
     ST_Rotate(geom, pi()) As using_rotate
   FROM (SELECT 'LINESTRING(1 2 3,1 4 3)'::geometry As geom) As foo;
```

Figure

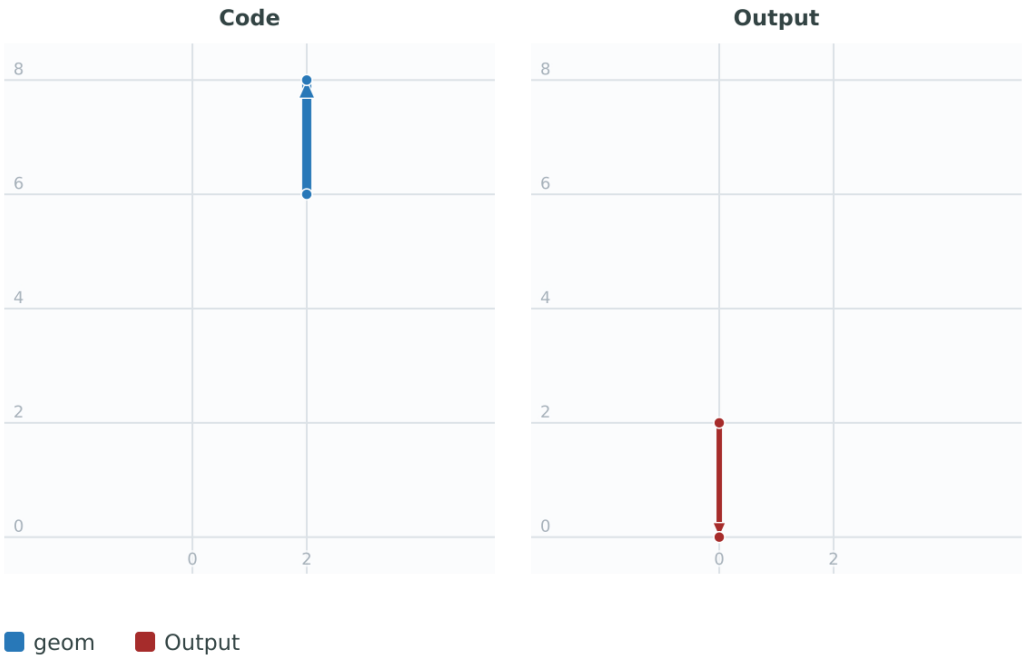


Rotate a 3D line 180 degrees in both the x and z axis.

Code

```
SELECT ST_Affine(geom, cos(pi()), -sin(pi()), 0, sin(pi()), cos(pi()), -sin(pi()), 0, sin(pi ↩
    ()), cos(pi()), 0, 0, 0)
    FROM (SELECT 'LINESTRING(1 2 3,1 4 3)'::geometry As geom) As foo;
```

Figure



Siehe auch

ST_Rotate, ST_Scale, ST_Translate, ST_TransScale

7.16.2 ST_Rotate

ST_Rotate — Dreht eine Geometrie um einen Ursprungspunkt.

Synopsis

geometry **ST_Rotate**(geometry geomA, float rotRadians);
geometry **ST_Rotate**(geometry geomA, float rotRadians, float x0, float y0);
geometry **ST_Rotate**(geometry geomA, float rotRadians, geometry pointOrigin);

Beschreibung

Dreht die Geometrie rotRadian gegen den Uhrzeigersinn um den Ursprungspunkt. Der Rotationsursprung kann entweder als POINT-Geometrie oder als x- und y-Koordinaten angegeben werden. Wenn der Ursprung nicht angegeben wird, wird die Geometrie um POINT(0 0) gedreht.

Erweiterung: Mit 2.0.0 wurde die Unterstützung für polyedrische Oberflächen, Dreiecke und TIN eingeführt.

Verbessert: In Version 2.0.0 wurden zusätzliche Parameter zur Angabe des Ursprungs der Drehung hinzugefügt.

Verfügbarkeit: 1.1.2. Name geändert von Rotate zu ST_Rotate in 1.2.2

Diese Funktion unterstützt 3d und lässt den Z-Index nicht fallen.

Diese Methode unterstützt kreisförmige Strings und Kurven.

Diese Funktion unterstützt polyedrische Flächen.

Diese Funktion unterstützt Dreiecke und dreieckige unregelmäßige Netzoberflächen (TIN).

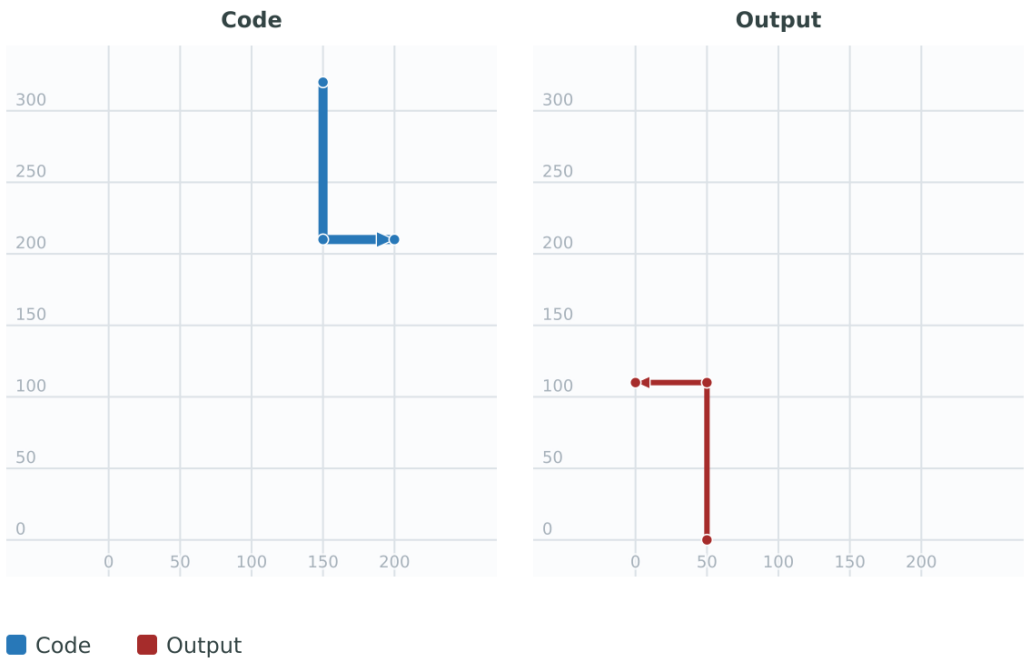
Beispiele

This example rotates a geometry by 180 degrees.

Code

```
SELECT ST_Rotate('LINESTRING (50 160,50 50,100 50)'::geometry, pi());
```

Figure

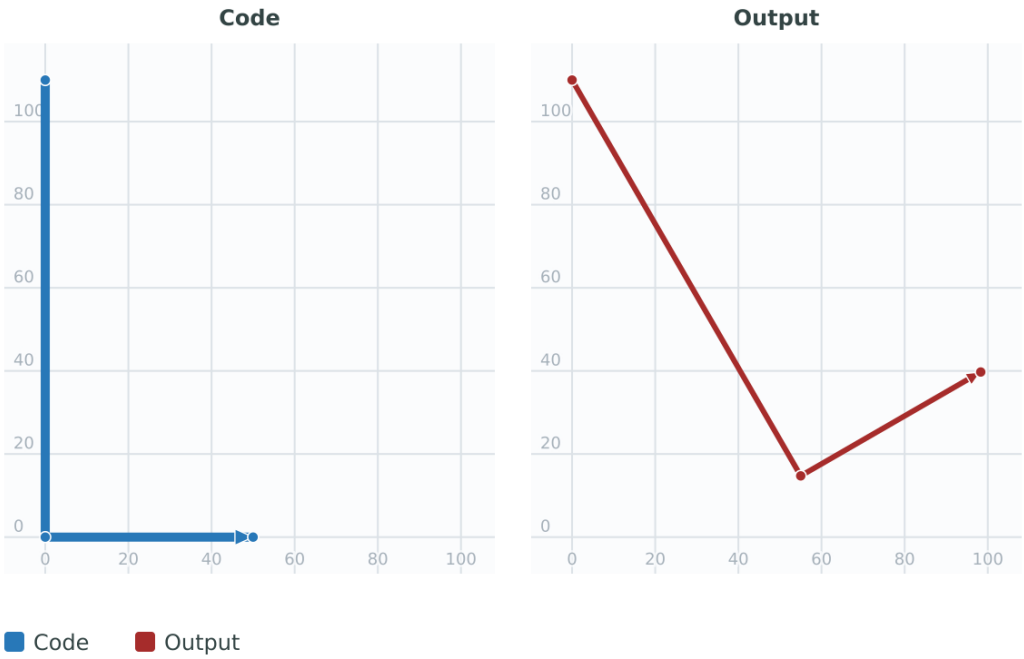


This example rotates a geometry 30 degrees counter-clockwise around x=50, y=160.

Code

```
SELECT ST_Rotate('LINESTRING (50 160,50 50,100 50)'::geometry, pi()/6, 50, 160);
```

Figure

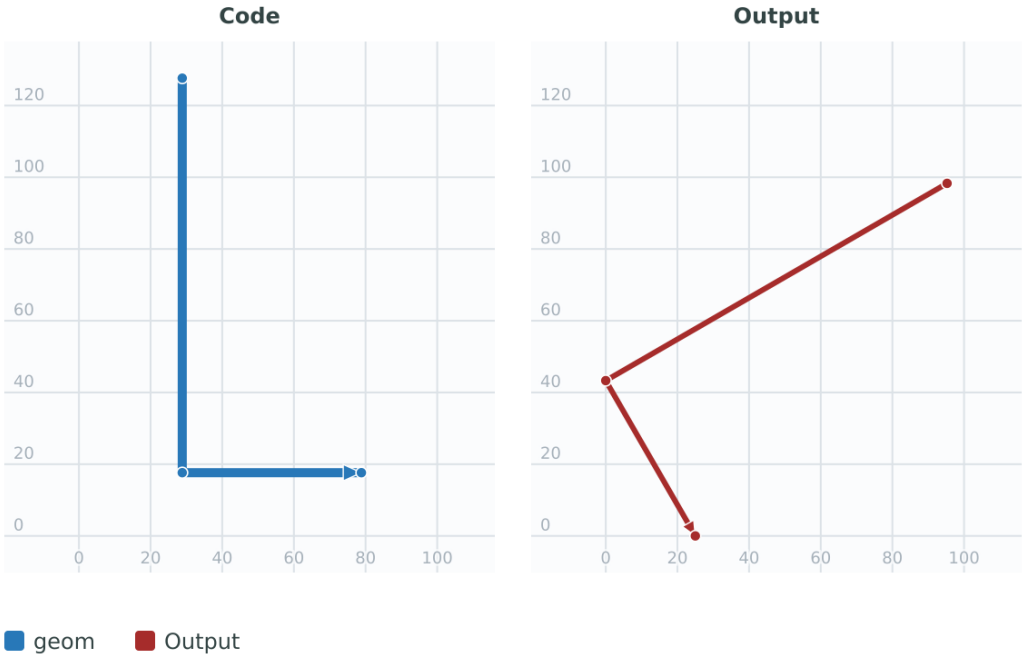


This example rotates a geometry 60 degrees clockwise around its centroid.

Code

```
SELECT ST_Rotate(geom, -pi()/3, ST_Centroid(geom))
FROM (SELECT 'LINESTRING (50 160,50 50,100 50)'::geometry AS geom) AS foo;
```

Figure



Siehe auch

ST_Affine, ST_RotateX, ST_RotateY, ST_RotateZ

7.16.3 ST_RotateX

ST_RotateX — Dreht eine Geometrie um die X-Achse.

Synopsis

geometry **ST_RotateX**(geometry geomA, float rotRadians);

Beschreibung

Dreht eine Geometrie geomA - rotRadians um die X-Achse.



Anmerkung
`ST_RotateX(geomA, rotRadians)` ist die Kurzform für `ST_Affine(geomA, 1, 0, 0, 0, cos(rotRadians), -sin(rotRadians), 0, sin(rotRadians), cos(rotRadians), 0, 0, 0, 0).`

Erweiterung: Mit 2.0.0 wurde die Unterstützung für polyedrische Oberflächen, Dreiecke und TIN eingeführt.

Verfügbarkeit: 1.1.2. Name geändert von RotateX zu ST_RotateX in 1.2.2

 Diese Funktion unterstützt polyedrische Flächen.

 Diese Funktion unterstützt 3d und lässt den Z-Index nicht fallen.

 Diese Funktion unterstützt Dreiecke und dreieckige unregelmäßige Netzoberflächen (TIN).

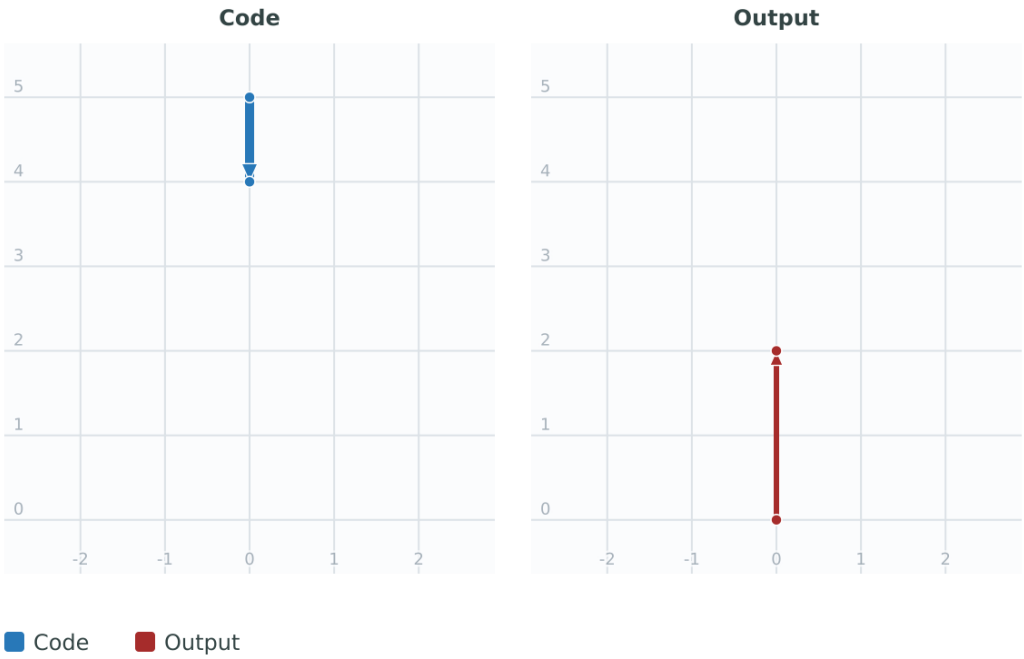
Beispiele

This example rotates a line 90 degrees around the X axis.

Code

```
SELECT ST_RotateX('LINESTRING(1 2 3,1 1 1)', pi()/2);
```

Figure



Siehe auch

ST_Affine, ST_RotateY, ST_RotateZ

7.16.4 ST_RotateY

ST_RotateY — Dreht eine Geometrie um die Y-Achse.

Synopsis

geometry **ST_RotateY**(geometry geomA, float rotRadians);

Beschreibung

Dreht eine Geometrie geomA - rotRadians um die y-Achse.

> **Anmerkung**
> `ST_RotateY(geomA, rotRadians)` ist die Kurzform für `ST_Affine(geomA, cos(rotRadians), 0, sin(rotRadians), 0, 1, 0, -sin(rotRadians), 0, cos(rotRadians), 0, 0, 0, 0)`.

Verfügbarkeit: 1.1.2. Name geändert von RotateY zu ST_RotateY in 1.2.2

Erweiterung: Mit 2.0.0 wurde die Unterstützung für polyedrische Oberflächen, Dreiecke und TIN eingeführt.

Diese Funktion unterstützt polyedrische Flächen.

Diese Funktion unterstützt 3d und lässt den Z-Index nicht fallen.

Diese Funktion unterstützt Dreiecke und dreieckige unregelmäßige Netzoberflächen (TIN).

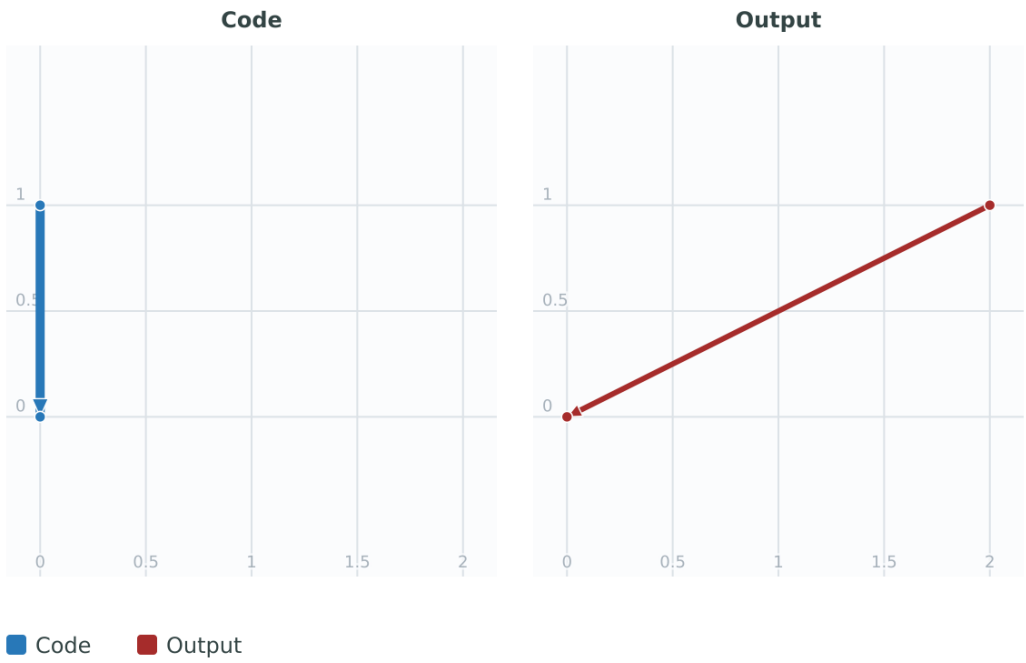
Beispiele

This example rotates a line 90 degrees around the Y axis.

Code

```
SELECT ST_RotateY('LINESTRING(1 2 3,1 1 1)', pi()/2);
```

Figure



Siehe auch

ST_Affine, ST_RotateY, ST_RotateZ

7.16.5 ST_RotateZ

ST_RotateZ — Dreht eine Geometrie um die Z-Achse.

Synopsis

geometry **ST_RotateZ**(geometry geomA, float rotRadians);

Beschreibung

Dreht eine Geometrie geomA - rotRadian um die Z-Achse.



Anmerkung
Dies ist ein Synonym für ST_Rotate



Anmerkung
`ST_RotateZ(geomA, rotRadians)` ist die Kurzform für `SELECT ST_Affine(geomA, cos(rotRadians), -sin(rotRadians), 0, sin(rotRadians), cos(rotRadians), 0, 0, 0, 1, 0, 0, 0, 0)`.

Erweiterung: Mit 2.0.0 wurde die Unterstützung für polyedrische Oberflächen, Dreiecke und TIN eingeführt.

Verfügbarkeit: 1.1.2. Name geändert von RotateZ zu ST_RotateZ in 1.2.2



Anmerkung
Vor 1.3.4 ist diese Funktion abgestürzt, wenn die Geometrien CURVES enthalten. Dies wurde mit 1.3.4+ behoben

 Diese Funktion unterstützt 3d und lässt den Z-Index nicht fallen.

 Diese Methode unterstützt kreisförmige Strings und Kurven.

 Diese Funktion unterstützt polyedrische Flächen.

 Diese Funktion unterstützt Dreiecke und dreieckige unregelmäßige Netzoberflächen (TIN).

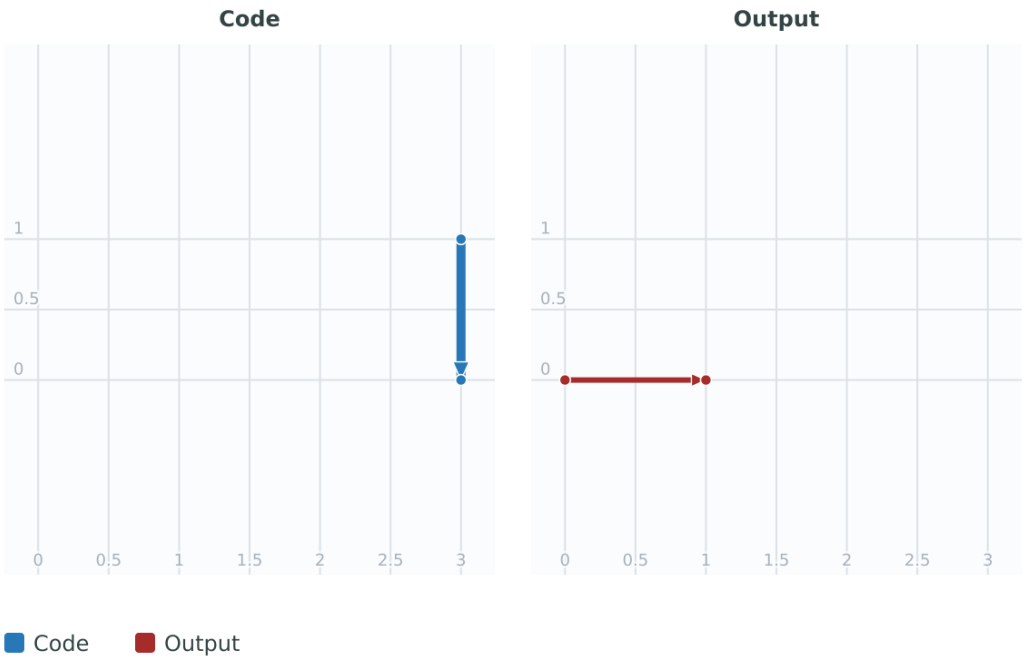
Beispiele

This example rotates a line 90 degrees around the Z axis.

Code

```
SELECT ST_RotateZ('LINESTRING(1 2 3,1 1 1)', pi()/2);
```

Figure

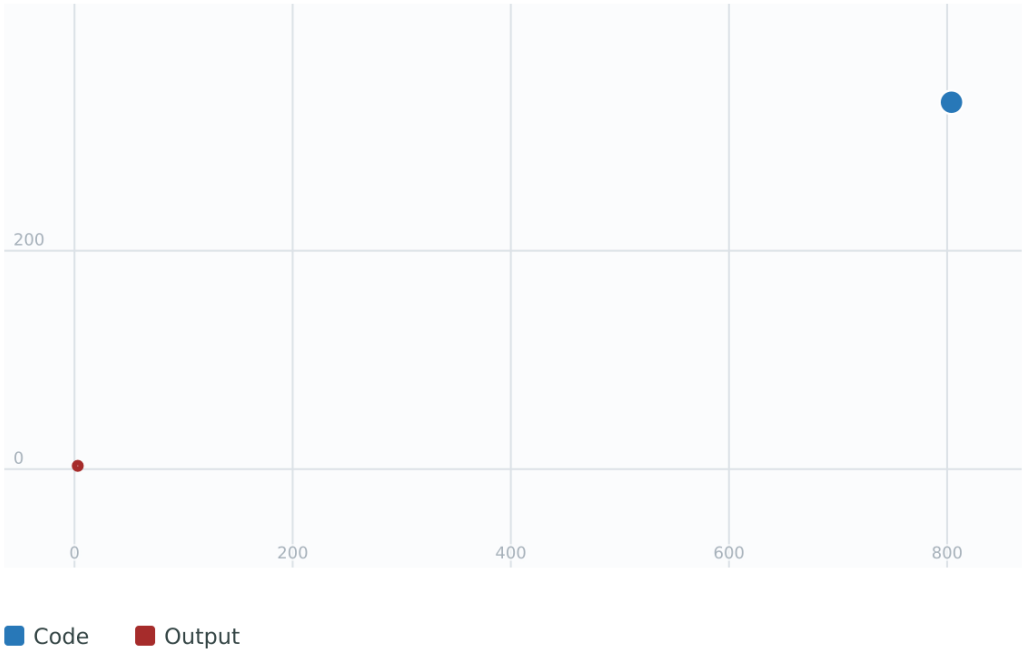


This example rotates a curved circle around the Z axis.

Code

```
SELECT ST_RotateZ(geom, pi()/2)
FROM (SELECT ST_LineToCurve(ST_Buffer('POINT(234 567)'::geometry, 3)) As geom) As foo;
```

Figure



Siehe auch

ST_Affine, ST_RotateX, ST_RotateY

7.16.6 ST_Scale

ST_Scale — Skaliert eine Geometrie um bestimmte Faktoren.

Synopsis

geometry **ST_Scale**(geometry geomA, float XFactor, float YFactor, float ZFactor);
geometry **ST_Scale**(geometry geomA, float XFactor, float YFactor);
geometry **ST_Scale**(geometry geom, geometry factor);
geometry **ST_Scale**(geometry geom, geometry factor, geometry origin);

Beschreibung

Skaliert die Geometrie auf eine neue Größe, indem die Ordinaten mit den entsprechenden Faktorparametern multipliziert werden.

Die Version, die eine Geometrie als `factor` Parameter nimmt, erlaubt die Übergabe eines 2d, 3dm, 3dz oder 4d Punktes, um den Skalierungsfaktor für alle unterstützten Dimensionen zu setzen. Fehlende Dimensionen im `factor` Punkt sind gleichbedeutend mit keiner Skalierung der entsprechenden Dimension.

Bei der Variante mit drei Geometrien kann ein "falscher Ursprung" für die Skalierung übergeben werden. Dies ermöglicht eine "Skalierung an Ort und Stelle", z. B. unter Verwendung des Schwerpunkts der Geometrie als falscher Ursprung. Ohne einen falschen Ursprung erfolgt die Skalierung relativ zum tatsächlichen Ursprung, so dass alle Koordinaten einfach mit dem Skalierungsfaktor multipliziert werden.

Anmerkung
Vor 1.3.4 ist diese Funktion abgestürzt, wenn die Geometrien CURVES enthalten. Dies wurde mit 1.3.4+ behoben

Verfügbarkeit: 1.1.0.

Erweiterung: Mit 2.0.0 wurde die Unterstützung für polyedrische Oberflächen, Dreiecke und TIN eingeführt.

Verbessert: In Version 2.2.0 wurde die Unterstützung für die Skalierung aller Dimensionen (Parameter`factor`) eingeführt.

Verbessert: In Version 2.5.0 wurde die Unterstützung für die Skalierung relativ zu einem lokalen Ursprung (Parameter`origin`) eingeführt.

Diese Funktion unterstützt polyedrische Flächen.

Diese Funktion unterstützt 3d und lässt den Z-Index nicht fallen.

Diese Methode unterstützt kreisförmige Strings und Kurven.

Diese Funktion unterstützt Dreiecke und dreieckige unregelmäßige Netzoberflächen (TIN).

Diese Funktion unterstützt M-Koordinaten.

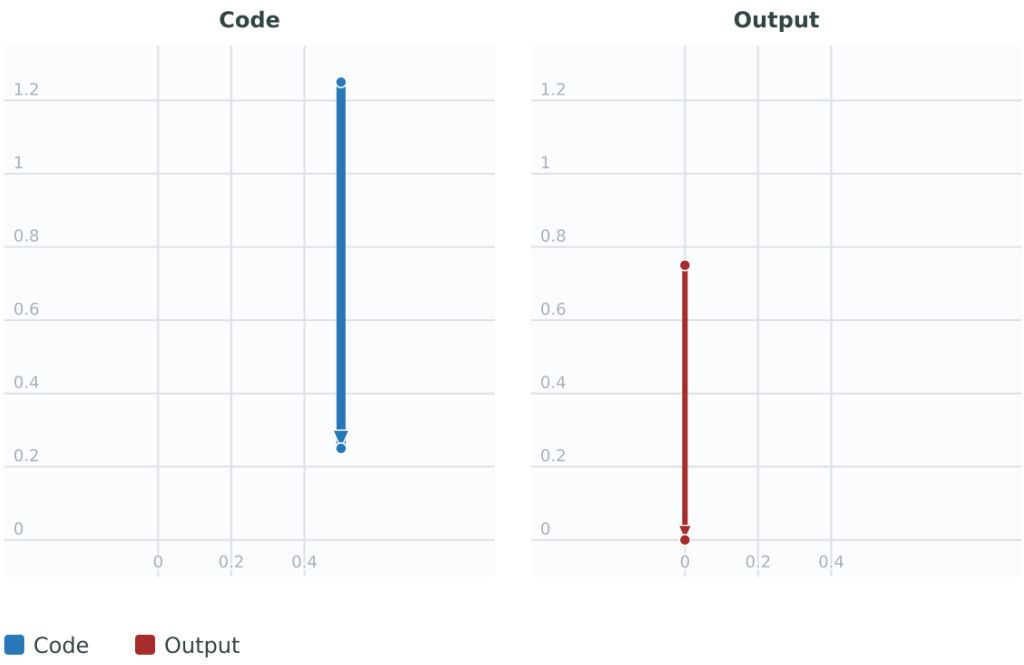
Beispiele

This example scales X, Y, and Z values.

Code

```
SELECT ST_Scale('LINESTRING(1 2 3,1 1 1)', 0.5, 0.75, 0.8);
```

Figure

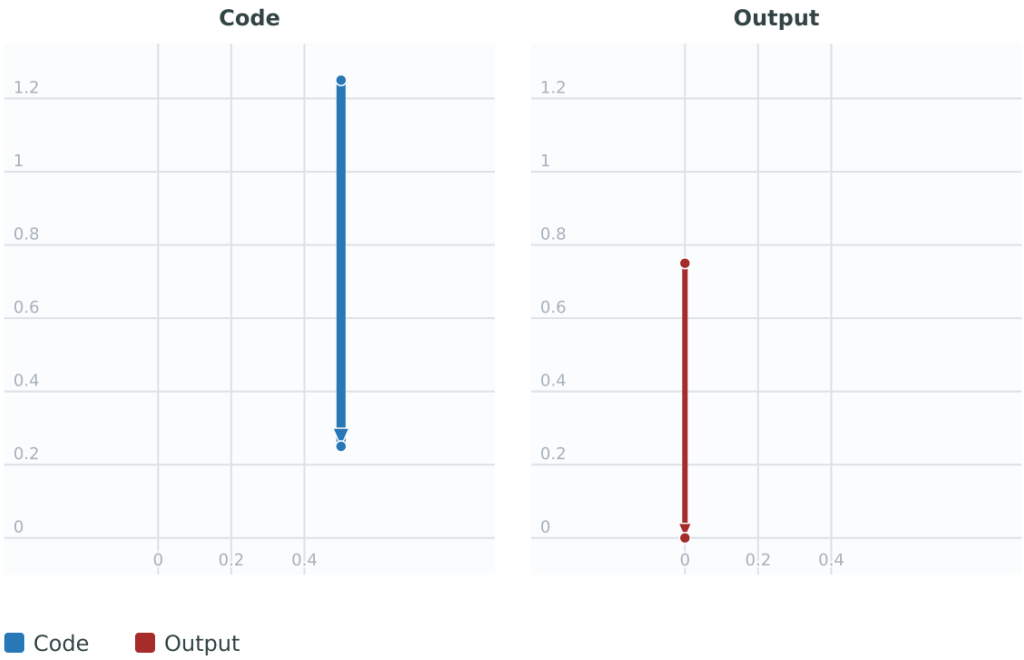


This example scales X and Y values.

Code

```
SELECT ST_Scale('LINESTRING(1 2 3,1 1 1)', 0.5, 0.75);
```

Figure

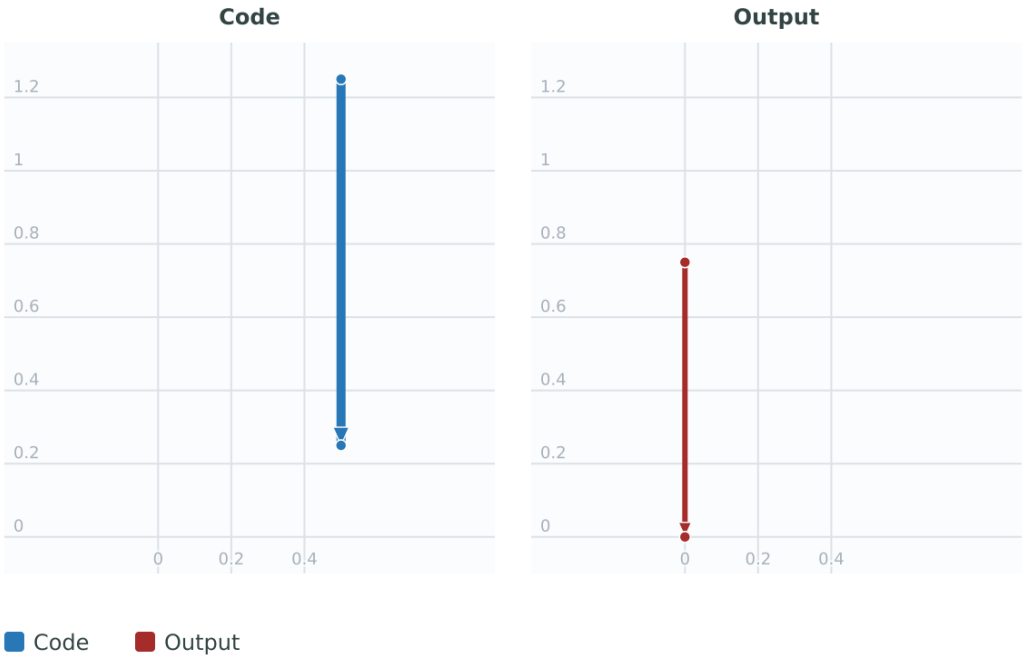


This example scales X, Y, Z, and M values.

Code

```
SELECT ST_Scale('LINESTRING(1 2 3 4,1 1 1 1)',
    ST_MakePoint(0.5, 0.75, 2, -1));
```

Figure

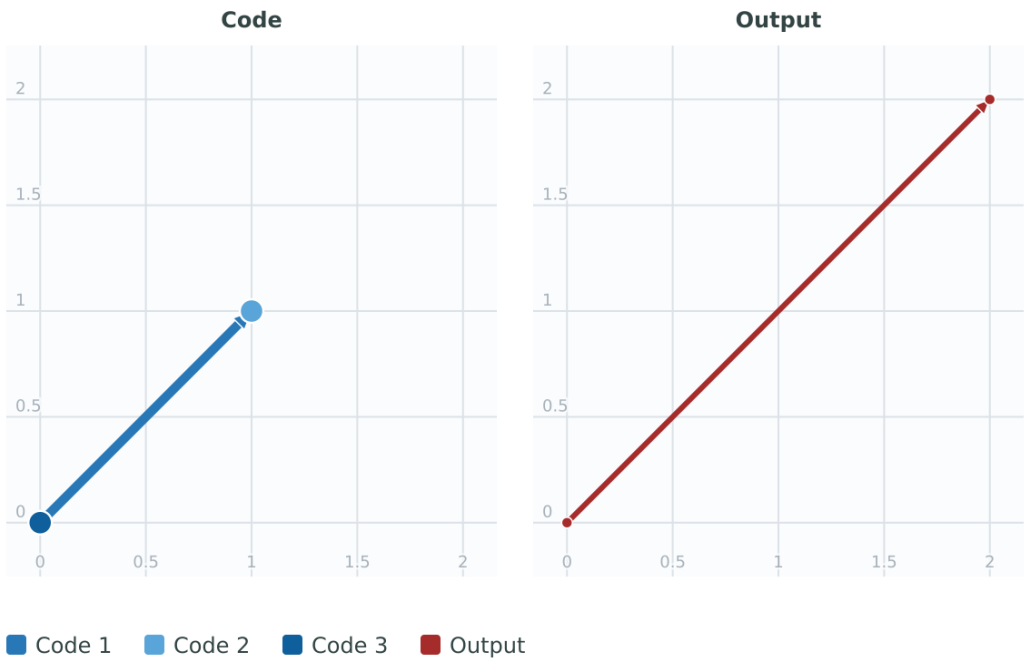


This example scales X and Y values using a false origin.

Code

```
SELECT ST_Scale('LINESTRING(1 1,2 2)', 'POINT(2 2)', 'POINT(1 1)'::geometry);
```

Figure



This example constructs an ellipse centered at (10, 20), with X radius 8, Y radius 4, and rotated 30 degrees counter-clockwise. Scaling and rotation use the center as their origin, so the ellipse keeps both its position and SRID.

Code

```
WITH params AS (
  SELECT
    ST_SetSRID(ST_Point(10, 20), 3857) AS center,
    8.0 AS x_radius,
    4.0 AS y_radius,
    radians(30) AS angle
)
SELECT ST_Rotate(
         ST_Scale(
           ST_Buffer(center, 1, 'quad_segs=32'),
           ST_MakePoint(x_radius, y_radius),
           center
         ),
         angle,
         center
       ) AS ellipse
FROM params;
```

Siehe auch

ST_Affine, ST_Buffer, ST_MakePoint, ST_Rotate, ST_TransScale

7.16.7 ST_Translate

ST_Translate — Verschiebt eine Geometrie um vorgegebene Offsets.

Synopsis

geometry **ST_Translate**(geometry g1, float deltax, float deltay);
geometry **ST_Translate**(geometry g1, float deltax, float deltay, float deltaz);

Beschreibung

Returns a new geometry whose coordinates are translated delta x,delta y,delta z units. Units are based on the units defined in spatial reference (SRID) for this geometry. M coordinates are preserved.



Anmerkung
Vor 1.3.4 ist diese Funktion abgestürzt, wenn die Geometrien CURVES enthalten. Dies wurde mit 1.3.4+ behoben

Verfügbarkeit: 1.2.2

 Diese Funktion unterstützt 3d und lässt den Z-Index nicht fallen.

 Diese Methode unterstützt kreisförmige Strings und Kurven.

 Diese Funktion unterstützt M-Koordinaten.

Beispiele

Einen Punkt um 1 Grad Länge verschieben

Code

```
SELECT ST_Translate(ST_GeomFromText('POINT(-71.01 42.37)', 4326), 1, 0) AS wgs_transgeomtxt;
```

Output

```
POINT(-70.01 42.37)
```

Verschieben eines Linienzugs um 1 Grad Länge und 1/2 Grad Breite

Code

```
SELECT ST_Translate(ST_GeomFromText('LINESTRING(-71.01 42.37,-71.11 42.38)', 4326), 1, 0.5)  ↩
    AS wgs_transgeomtxt;
```

Figure



Verschieben eines 3D-Punktes

Code

```
SELECT ST_Translate(CAST('POINT(0 0 0)' AS geometry), 5, 12, 3);
```

Output

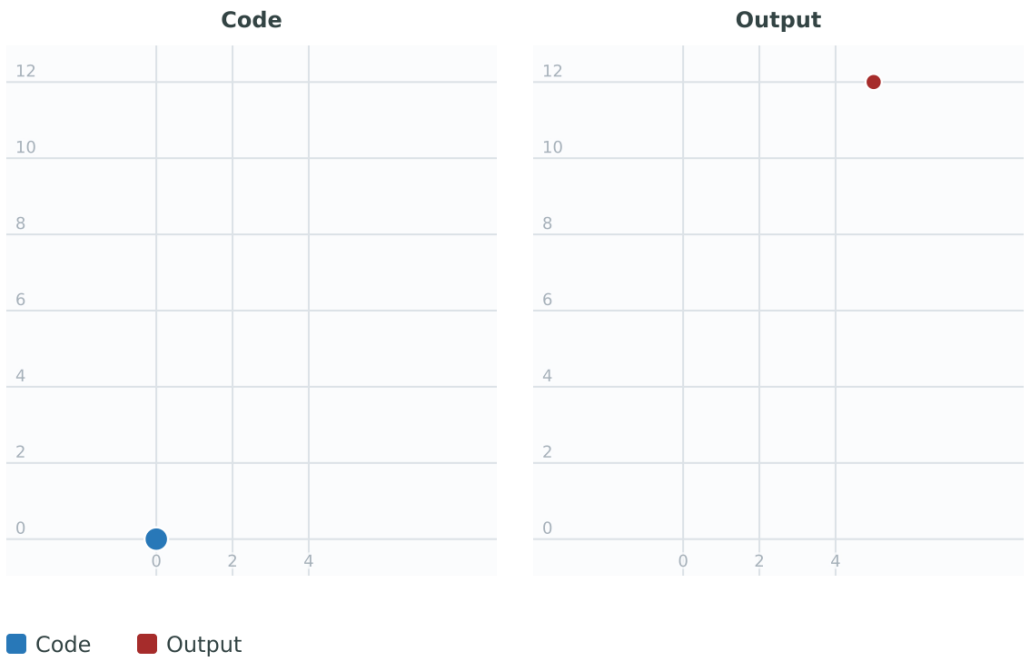
```
POINT(5 12 3)
```

Move points with a measure coordinate

Code

```
SELECT ST_Translate('POINTM(0 0 11)', 5, 12);
```

Figure



Code

```
SELECT ST_Translate('POINT(0 0 7 11)', 5, 12, 3);
```

Output

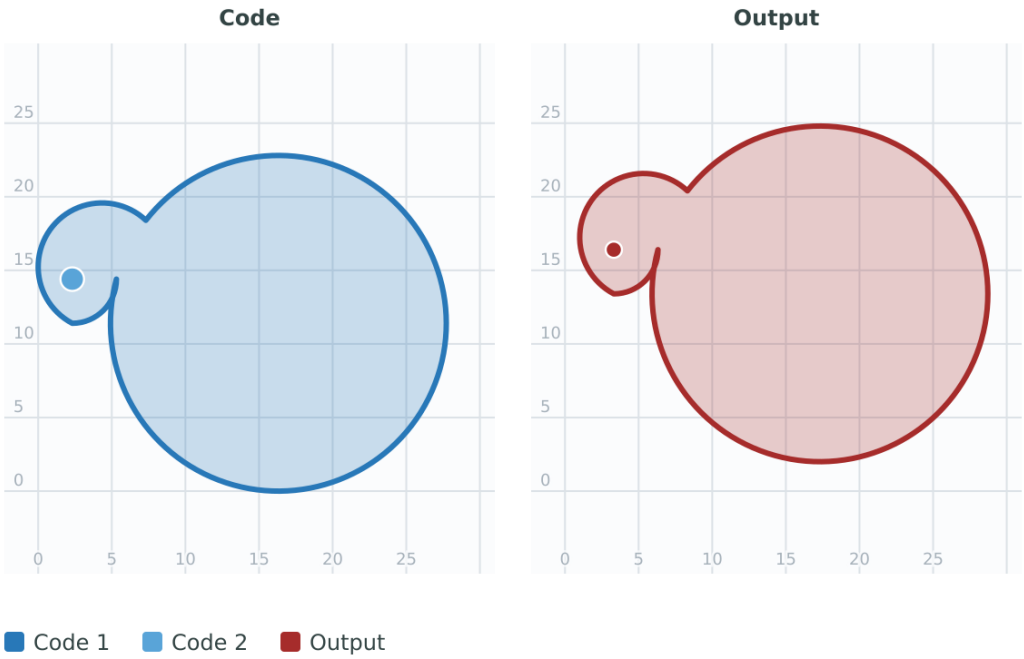
```
POINT(5 12 10 11)
```

Verschieben einer Kurve und eines Punktes

Code

```
SELECT ST_Translate(ST_Collect('CURVEPOLYGON(CIRCULARSTRING(4 3,3.12 0.878,1 0,-1.121 ↩
    5.1213,6 7,8 9,4 3))', 'POINT(1 3)'), 1, 2);
```

Figure



Siehe auch

ST_Affine, ST_AsText, ST_GeomFromText

7.16.8 ST_TransScale

ST_TransScale — Verschiebt und skaliert eine Geometrie mit vorgegebenen Offsets und Faktoren.

Synopsis

geometry **ST_TransScale**(geometry geomA, float deltaX, float deltaY, float XFactor, float YFactor);

Beschreibung

Verschiebt die Geometrie mit Hilfe der Args deltaX und deltaY und skaliert sie dann mit Hilfe der Args XFactor und YFactor, wobei nur in 2D gearbeitet wird.



Anmerkung
`ST_TransScale(geomA, deltaX, deltaY, XFactor, YFactor)` ist die Kurzform für `ST_Affine(geomA, XFactor, 0, 0, 0, YFactor, 0, 0, 0, 1, deltaX*XFactor, deltaY*YFactor, 0)`.



Anmerkung
Vor 1.3.4 ist diese Funktion abgestürzt, wenn die Geometrien CURVES enthalten. Dies wurde mit 1.3.4+ behoben

Verfügbarkeit: 1.1.0.

Diese Funktion unterstützt 3d und lässt den Z-Index nicht fallen.

Diese Methode unterstützt kreisförmige Strings und Kurven.

Beispiele

Code

```
SELECT ST_TransScale('LINESTRING(1 2 3,1 1 1)', 0.5, 1, 1, 2);
```

Figure

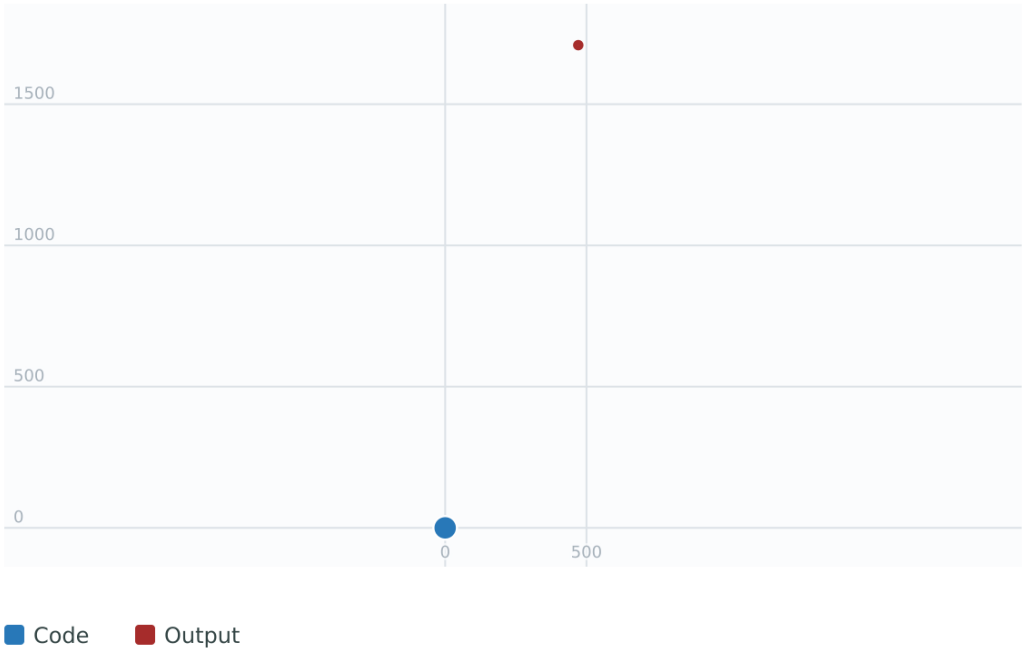


This example buffers a point to approximate a circle, converts it to a curve, translates it by 1,2, and scales it by 3,4.

Code

```
SELECT ST_TransScale(ST_LineToCurve(ST_Buffer('POINT(234 567)', 3)), 1, 2, 3, 4);
```

Figure



Siehe auch

ST_Affine, ST_Translate

7.17 Clustering-Funktionen

7.17.1 ST_ClusterDBSCAN

ST_ClusterDBSCAN — Fensterfunktion, die eine Cluster-ID für jede Eingabegeometrie unter Verwendung des DBSCAN-Algorithmus zurückgibt.

Synopsis

integer **ST_ClusterDBSCAN**(geometry winset geom, float8 eps, integer minpoints);

Beschreibung

Eine Fensterfunktion, die eine Clusternummer für jede Eingabegeometrie zurückgibt, unter Verwendung des 2D Dichte-basierten räumlichen Clustering von Anwendungen mit Rauschen (DBSCAN) Algorithmus. Im Gegensatz zu ST_ClusterKMeans muss die Anzahl der Cluster nicht angegeben werden, sondern es wird der gewünschte Distanz (`eps`) und Dichte (`minpoints`) zur Bestimmung der einzelnen Cluster.

Eine Eingangsgeometrie wird zu einem Cluster hinzugefügt, wenn sie entweder:

- Eine "Kern"-Geometrie, die innerhalb `eps` Abstand von mindestens `minpoints` Eingabegeometrien (einschließlich sich selbst) liegt; oder
- Eine "Rand"-Geometrie, die sich in `eps` Entfernung einer Kerngeometrie befindet.

Beachten Sie, dass Randgeometrien in `eps` Entfernung von Kerngeometrien in mehr als einem Cluster liegen können. Jede der beiden Zuordnungen wäre korrekt, so dass die Randgeometrie willkürlich einem der verfügbaren Cluster zugeordnet wird. In dieser Situation ist es möglich, dass ein korrekter Cluster mit weniger als `minpoints` Geometrien erzeugt wird. Um eine deterministische Zuordnung der Randgeometrien zu gewährleisten (so dass wiederholte Aufrufe von ST_ClusterDBSCAN identische Ergebnisse liefern), verwenden Sie eine `ORDER BY` Klausel in der Fensterdefinition. Mehrdeutige Clusterzuweisungen können sich von anderen DBSCAN-Implementierungen unterscheiden.



Anmerkung
Geometrien, die die Kriterien für die Zugehörigkeit zu einem Cluster nicht erfüllen, erhalten eine Clusternummer von NULL.

Verfügbarkeit: 2.3.0

 Diese Methode unterstützt kreisförmige Strings und Kurven.

Beispiele

Cluster polygons within 50 units of each other, requiring at least two polygons per cluster. The isolated polygon is reported as noise.

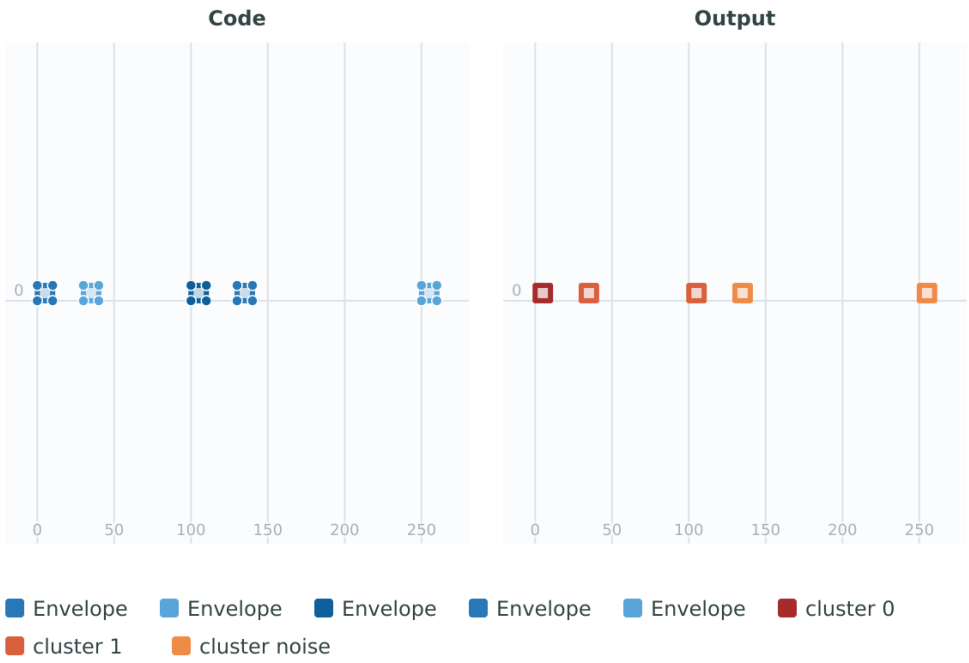
Code

```
WITH input(name, geom) AS (
  VALUES
    ('A1', ST_MakeEnvelope(0, 0, 10, 10)),
    ('A2', ST_MakeEnvelope(30, 0, 40, 10)),
    ('B1', ST_MakeEnvelope(100, 0, 110, 10)),
    ('B2', ST_MakeEnvelope(130, 0, 140, 10)),
    ('noise', ST_MakeEnvelope(250, 0, 260, 10))
), clustered AS (
  SELECT name,
         geom,
         ST_ClusterDBSCAN(geom, eps => 50, minpoints => 2)
           OVER (ORDER BY name) AS cid
  FROM input
)
SELECT COALESCE(cid::text, 'noise') AS cluster,
       string_agg(name, ',' ORDER BY name) AS members,
       ST_Collect(geom ORDER BY name) AS cluster_geom
FROM clustered
GROUP BY cid
ORDER BY cid NULLS LAST;
```

Output

```
0 | A1,A2 | MULTIPOLYGON(((0 0,0 10,10 10,10 0,0 0)),((30 0,30 10,40 10,40 0,30 0)))
1 | B1,B2 | MULTIPOLYGON(((100 0,100 10,110 10,110 0,100 0)),((130 0,130 10,140 10,140 0,130 0)))
noise | noise | MULTIPOLYGON(((250 0,250 10,260 10,260 0,250 0)))
```

Figure



Ein Beispiel für die Zusammenfassung von Flurstücken mit derselben Clusternummer zu geometrischen Sammlungen.

Code

```
SELECT cid, ST_Collect(geom) AS cluster_geom, array_agg(parcel_id) AS ids_in_cluster FROM (
    SELECT parcel_id, ST_ClusterDBSCAN(geom, eps => 0.5, minpoints => 5) over () AS cid,  ↩
    geom
    FROM parcels) sq
GROUP BY cid;
```

Siehe auch

ST_DWithin, ST_ClusterKMeans, ST_ClusterIntersecting, ST_ClusterIntersectingWin, ST_ClusterWithin, ST_ClusterWithinWin

7.17.2 ST_ClusterIntersecting

ST_ClusterIntersecting — Aggregatfunktion, die Eingabegeometrien zu zusammenhängenden Mengen clustert.

Synopsis

geometry[] **ST_ClusterIntersecting**(geometry set g);

Beschreibung

Eine Aggregatfunktion, die ein Array von GeometryCollections zurückgibt, das die Eingabegeometrien in zusammenhängende, disjunkte Cluster partitioniert. Jede Geometrie in einem Cluster schneidet mindestens eine andere Geometrie in diesem Cluster und schneidet keine Geometrie in anderen Clustern.

Verfügbarkeit: 2.2.0

Beispiele

Code

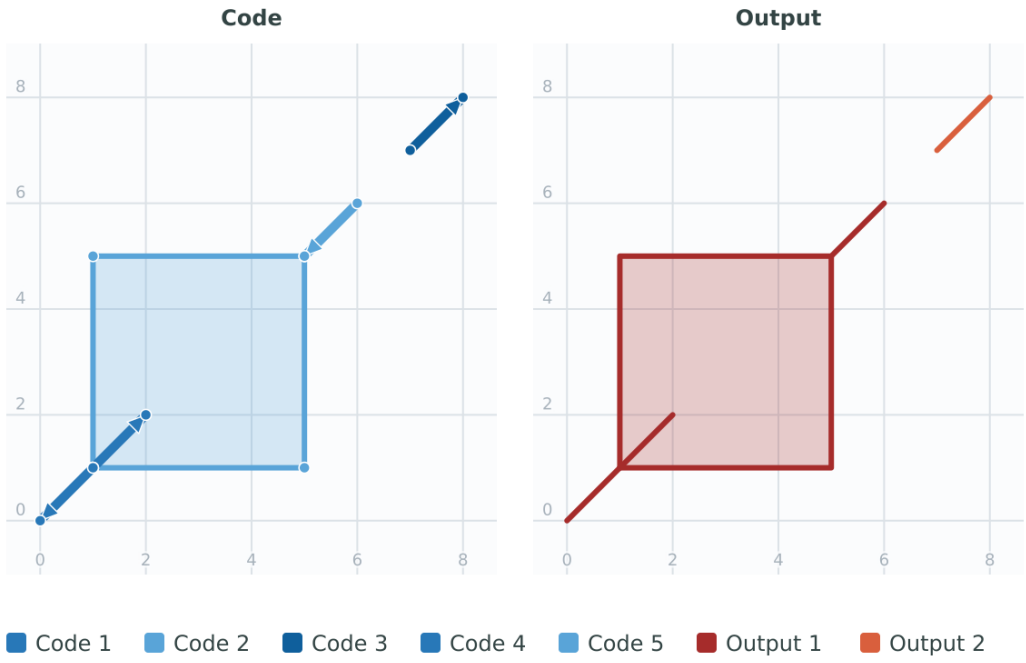
```
WITH testdata AS
  (SELECT unnest(
            ARRAY['LINESTRING (0 0,1 1)'::geometry,
            'LINESTRING (5 5,4 4)'::geometry,
            'LINESTRING (6 6,7 7)'::geometry,
            'LINESTRING (0 0,-1 -1)'::geometry,
            'POLYGON ((0 0,4 0,4 4,0 4,0 0))'::geometry]) AS geom)

SELECT unnest(ST_ClusterIntersecting(geom)) FROM testdata;
```

Output

```
GEOMETRYCOLLECTION(LINESTRING(0 0,1 1),LINESTRING(5 5,4 4),LINESTRING(0 0,-1 -1),POLYGON((0 0,4 0,4 4,0 4,0 0)))
GEOMETRYCOLLECTION(LINESTRING(6 6,7 7))
```

Figure



Siehe auch

ST_ClusterIntersectingWin, ST_ClusterWithin, ST_ClusterWithinWin

7.17.3 ST_ClusterIntersectingWin

ST_ClusterIntersectingWin — Fensterfunktion, die für jede Eingabegeometrie eine Cluster-ID zurückgibt und die Eingabegeometrien in zusammenhängende Gruppen clustert.

Synopsis

integer **ST_ClusterIntersectingWin**(geometry winset geom);

Beschreibung

Eine Fensterfunktion, die zusammenhängende Cluster von sich schneidenden Geometrien bildet. Es ist möglich, alle Geometrien in einem Cluster zu durchlaufen, ohne den Cluster zu verlassen. Der Rückgabewert ist die Clusternummer, an der das Geometrieargument beteiligt ist, oder null für Nulleingaben.

Verfügbarkeit: 3.4.0

Beispiele

Return an explicit cluster id for each input geometry. Using the conventional `cid` alias keeps the geometry column ready for color-by-cluster rendering.

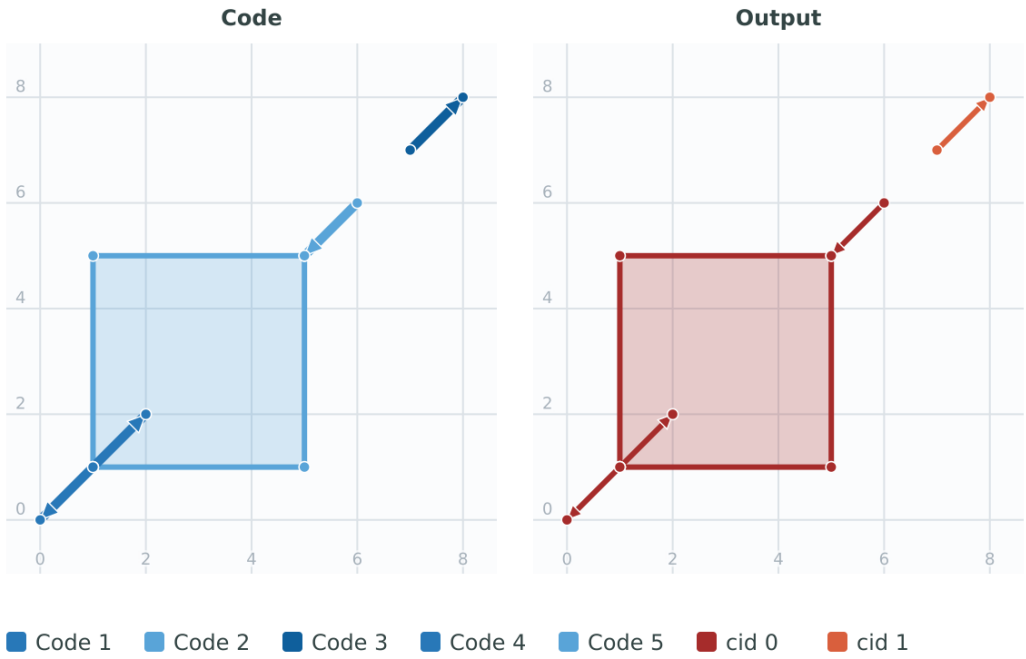
Code

```
WITH testdata(id, geom) AS (
  VALUES (1, 'LINESTRING (0 0,1 1)'::geometry),
         (2, 'LINESTRING (5 5,4 4)'::geometry),
         (3, 'LINESTRING (6 6,7 7)'::geometry),
         (4, 'LINESTRING (0 0,-1 -1)'::geometry),
         (5, 'POLYGON ((0 0,4 0,4 4,0 4,0 0))'::geometry)
), clustered AS (
  SELECT id,
         geom,
         ST_ClusterIntersectingWin(geom) OVER () AS cid
  FROM testdata
)
SELECT id,
       geom AS geom_wkt,
       cid
FROM clustered
ORDER BY id;
```

Output

```
id |            geom_wkt             | cid
----+---------------------------------+-----
  1 | LINESTRING(0 0,1 1)             |   0
  2 | LINESTRING(5 5,4 4)             |   0
  3 | LINESTRING(6 6,7 7)             |   1
  4 | LINESTRING(0 0,-1 -1)           |   0
  5 | POLYGON((0 0,4 0,4 4,0 4,0 0)) |   0
```

Figure



Siehe auch

ST_ClusterIntersecting, ST_ClusterWithin, ST_ClusterWithinWin

7.17.4 ST_ClusterRelateWin

ST_ClusterRelateWin — Window function that returns a cluster id for each input geometry, clustering input geometries into connected sets using the relate pattern to determine whether the geometries are connected.

Synopsis

integer **ST_ClusterRelateWin**(geometry winset geom, text relate_matrix);

Beschreibung

A window function that builds connected clusters of geometries that intersect. Geometries are added to a cluster if they share a pairwise DE9IM relationship with another member of the cluster. With this function it is possible to build a cluster of all objects that touch at boundaries, but exclude those that merely overlap.

Availability: 3.7.0

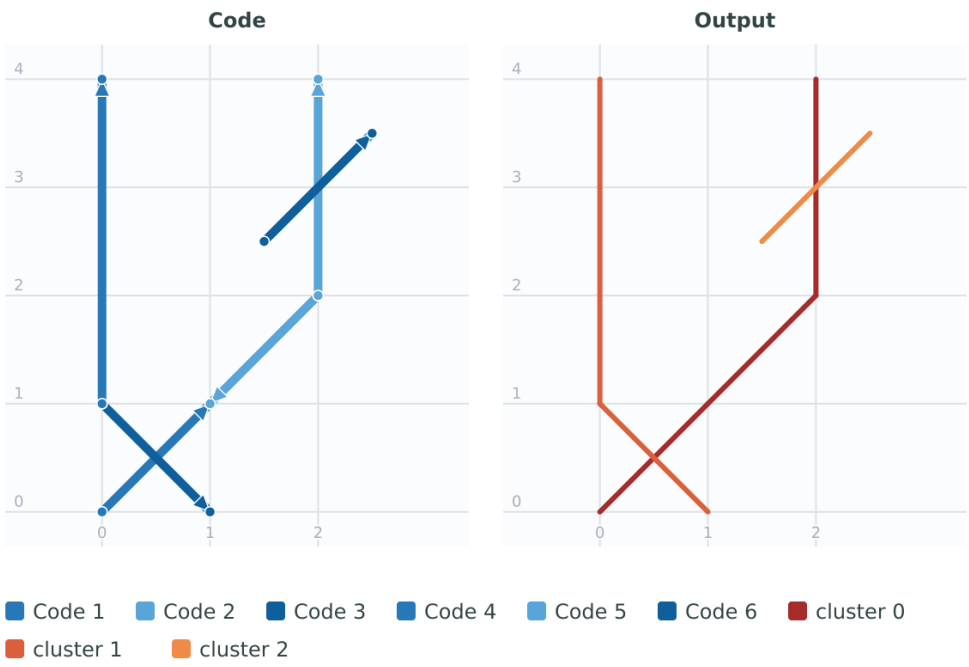
Beispiele

This collection of line strings would form a single cluster using ST_ClusterIntersectingWin, but using ST_ClusterRelateWin can be clustered into three groups that connect only via their end points.

Code

```
WITH data(id, geom) AS (
  VALUES (1, 'LINESTRING(0 0,1 1)'::geometry),
         (2, 'LINESTRING(2 2,1 1)'::geometry),
         (3, 'LINESTRING(0 1,1 0)'::geometry),
         (4, 'LINESTRING(0 1,0 4)'::geometry),
         (5, 'LINESTRING(2 2,2 4)'::geometry),
         (6, 'LINESTRING(1.5 2.5,2.5 3.5)'::geometry)
), clustered AS (
  SELECT id, geom,
         ST_ClusterRelateWin(geom, '****0****') OVER () AS cluster
  FROM data
)
SELECT cluster,
       string_agg(id::text, ',' ORDER BY id) AS members,
       ST_Collect(geom ORDER BY id) AS cluster_geom
FROM clustered
GROUP BY cluster
ORDER BY cluster;
```

Figure



Siehe auch

ST_Relate, ST_ClusterIntersectingWin, ST_ClusterWithinWin

7.17.5 ST_MinimumSpanningTree

ST_MinimumSpanningTree — Window function that returns a tree id for each input geometry, clustering input geometries into connected trees using the Minimum Spanning Tree algorithm.

Synopsis

integer **ST_MinimumSpanningTree**(geometry winset geom);

Beschreibung

A window function that builds connected graphs of line strings based on the Minimum Spanning Tree (MST) of the input geometries. The return value is the cluster number that the geometry argument participates in, or zero if it is not part of the minimum tree.

The Minimum Spanning Tree connects all geometries in the window partition such that the total length of the connecting lines is minimized. If the graph is not fully connected (e.g. infinite distance between some geometries), it produces a Minimum Spanning Forest, and each connected component is assigned a unique tree ID.

Availability: 3.7.0

Requires GEOS >= 3.15.0

Beispiele

These six edges describe a square with both diagonals. The minimum spanning tree keeps three edges and excludes the rest.

Code

```
WITH edges(id, geom) AS (
  VALUES
    (1, 'LINESTRING(0 0,1 0)'::geometry),
    (2, 'LINESTRING(0 0,0 1)'::geometry),
    (3, 'LINESTRING(1 1,0 1)'::geometry),
    (4, 'LINESTRING(1 1,1 0)'::geometry),
    (5, 'LINESTRING(0 0,1 1)'::geometry),
    (6, 'LINESTRING(1 0,0 1)'::geometry)
), marked AS (
  SELECT id,
         geom,
         ST_MinimumSpanningTree(geom) OVER (ORDER BY id) AS tree_id
  FROM edges
)
SELECT string_agg(id::text, ',' ORDER BY id)
         FILTER (WHERE tree_id > 0) AS tree_edge_ids,
       ST_Collect(geom ORDER BY id)
         FILTER (WHERE tree_id > 0) AS tree_edges,
       string_agg(id::text, ',' ORDER BY id)
         FILTER (WHERE tree_id = 0) AS excluded_edge_ids,
       ST_Collect(geom ORDER BY id)
         FILTER (WHERE tree_id = 0) AS excluded_edges
FROM marked;
```

Output

```
-[ RECORD 1 ]-----
tree_edge_ids     | 1,2,3
tree_edges        | MULTILINESTRING((0 0,1 0),(0 0,0 1),(1 1,0 1))
excluded_edge_ids | 4,5,6
excluded_edges    | MULTILINESTRING((1 1,1 0),(0 0,1 1),(1 0,0 1))
```

Siehe auch

ST_ClusterDBSCAN, ST_ClusterKMeans, ST_ClusterIntersectingWin

7.17.6 ST_ClusterKMeans

ST_ClusterKMeans — Fensterfunktion, die eine Cluster-ID für jede Eingabegeometrie unter Verwendung des K-Means-Algorithmus zurückgibt.

Synopsis

integer **ST_ClusterKMeans**(geometry winset geom , integer k , float8 max_radius);

Beschreibung

Gibt K-means Clusternummer für jede Eingabegeometrie zurück. Der für das Clustering verwendete Abstand ist der Abstand zwischen den Zentren für 2D-Geometrien und der Abstand zwischen den Bounding-Box-Zentren für 3D-Geometrien. Bei POINT-Eingaben wird die Koordinate M als Gewicht der Eingabe behandelt und muss größer als 0 sein.

`max_radius`, falls gesetzt, veranlasst ST_ClusterKMeans, mehr Cluster als `k` zu erzeugen, um sicherzustellen, dass kein Cluster in der Ausgabe einen größeren Radius als `max_radius` hat. Dies ist bei der Erreichbarkeitsanalyse nützlich.

Verbessert: 3.2.0 Unterstützung für `max_radius`

Verbessert: 3.1.0 Unterstützung für 3D-Geometrien und Gewichte

Verfügbarkeit: 2.3.0

Beispiele

Define the parcel geometries once in a CTE and reuse them across the executable examples.

Parcels color-coded by cluster number (`cid`).

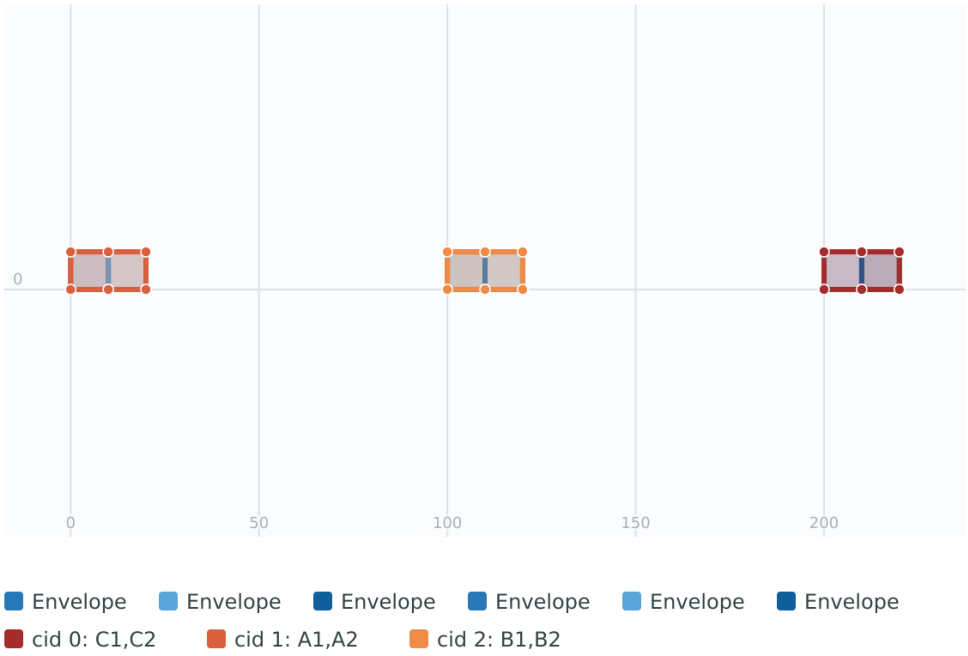
Code

```
WITH parcels(parcel_id, geom) AS (
  VALUES
    ('A1', ST_MakeEnvelope(0, 0, 10, 10)),
    ('A2', ST_MakeEnvelope(10, 0, 20, 10)),
    ('B1', ST_MakeEnvelope(100, 0, 110, 10)),
    ('B2', ST_MakeEnvelope(110, 0, 120, 10)),
    ('C1', ST_MakeEnvelope(200, 0, 210, 10)),
    ('C2', ST_MakeEnvelope(210, 0, 220, 10))
), clustered AS (
  SELECT parcel_id,
         geom,
         ST_ClusterKMeans(geom, 3) OVER (ORDER BY parcel_id) AS cid
  FROM parcels
)
SELECT cid,
       string_agg(parcel_id, ',' ORDER BY parcel_id) AS parcels,
       ST_Union(geom) AS cluster_geom
FROM clustered
GROUP BY cid
ORDER BY cid;
```

Output

```
0 | C1,C2 | POLYGON((200 10,210 10,220 10,220 0,210 0,200 0,200 10))
1 | A1,A2 | POLYGON((0 10,10 10,20 10,20 0,10 0,0 0,0 10))
2 | B1,B2 | POLYGON((100 10,110 10,120 10,120 0,110 0,100 0,100 10))
```

Figure



Unterteilung von Parzellenclustern nach Typ:

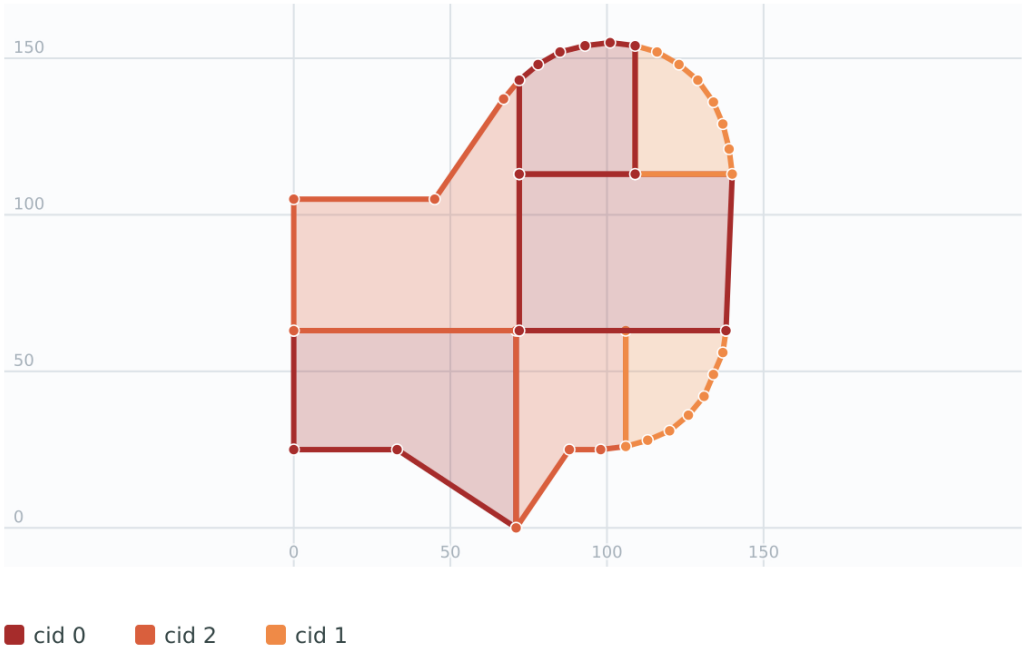
Code

```
WITH parcel_geoms AS (
  SELECT geom
  FROM ST_Subdivide(
    ST_Buffer('SRID=3857;LINESTRING(40 100,98 100,100 150,60 90)'::geometry,
              40, 'endcap=square'),
    12) AS geom
), parcels AS (
  SELECT lpad(row_number() OVER (ORDER BY ST_YMin(geom), ST_XMin(geom))::text, 3, '0') AS
    parcel_id,
         geom,
         ('{residential,commercial}'::text[])[1 + mod(row_number() OVER (ORDER BY ST_YMin(
    geom), ST_XMin(geom)), 2)] AS type
  FROM parcel_geoms
)
SELECT ST_ClusterKMeans(geom, 3) over (PARTITION BY type) AS cid,
       parcel_id,
       type,
       ST_SnapToGrid(geom, 1) AS parcel_geom
FROM parcels
ORDER BY type, parcel_id;
```

Output

```
cid | parcel_id |    type     | parcel_geom
----+-----------+-------------+-----------------------------------------------------------
  0 | 001       | commercial  | POLYGON((33 60,0 60,0 98,71 98,71 35,33 60))
  2 | 003       | commercial  | POLYGON((106 98,138 98,137 91,134 84,131 77,126 71,120 66,113 63,106 61,106 98))
  2 | 005       | commercial  | POLYGON((140 148,138 98,72 98,72 148,140 148))
  1 | 007       | commercial  | POLYGON((109 189,116 187,123 183,129 178,134 171,137 164,139 156,140 148,109 148,109
    189))
  0 | 002       | residential | POLYGON((71 98,106 98,106 61,98 60,88 60,71 35,71 98))
  2 | 004       | residential | POLYGON((0 98,0 140,45 140,67 172,72 178,72 98,0 98))
  1 | 006       | residential | POLYGON((72 178,78 183,85 187,93 189,101 190,109 189,109 148,72 148,72 178))
```

Figure



Example: Clustering a preaggregated planetary-scale data population dataset using 3D clustering and weighting. Identify at least 20 regions based on Kontur Population Data that do not span more than 3000 km from their center:

Code

```
CREATE TABLE kontur_population_3000km_clusters AS
SELECT
    geom,
    ST_ClusterKMeans(
        ST_Force4D(
            ST_Transform(ST_Force3D(geom), 4978),
            mvalue => population
        ),
        20,
        max_radius => 3000000
    ) OVER () AS cid
FROM kontur_population;
```

The clustering produces 46 regions. Clusters are centered at well-populated regions such as New York and Moscow. Greenland forms one cluster, several island clusters span the antimeridian, and cluster edges follow the Earth's curvature.



Kontur population clustered with 3000 km maximum radius.

Cluster weighted city populations on the Earth in geocentric coordinates. The maximum radius is specified in meters, while the returned geometries stay in WGS 84 for display.

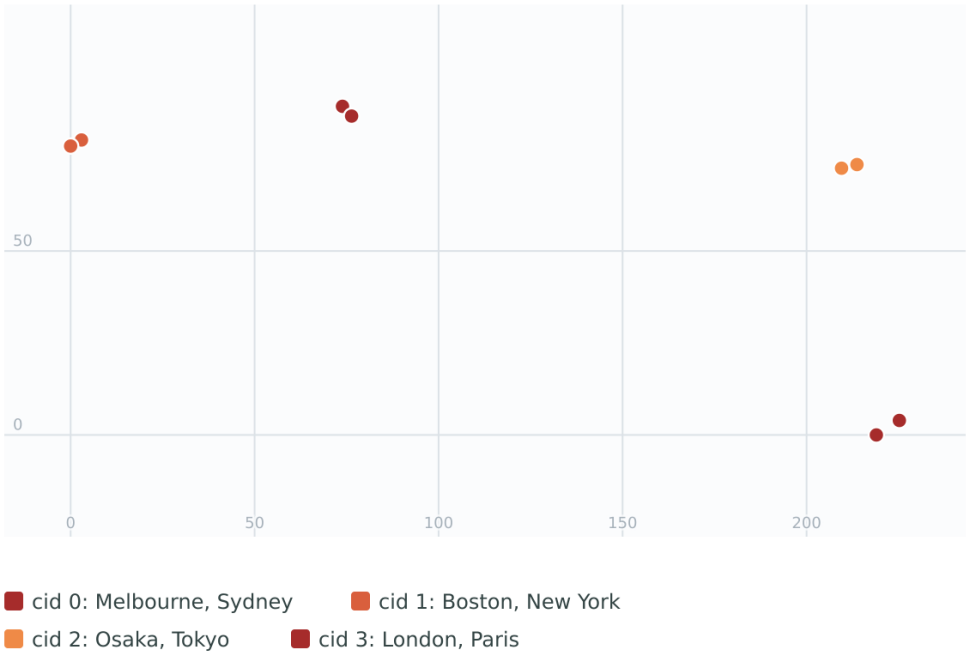
Code

```
WITH population(place, population, geom) AS (
  VALUES
    ('Boston', 5.0, ST_SetSRID(ST_Point(-71.06, 42.36), 4326)),
    ('New York', 20.0, ST_SetSRID(ST_Point(-74.01, 40.71), 4326)),
    ('London', 15.0, ST_SetSRID(ST_Point(-0.13, 51.51), 4326)),
    ('Paris', 11.0, ST_SetSRID(ST_Point(2.35, 48.86), 4326)),
    ('Osaka', 19.0, ST_SetSRID(ST_Point(135.50, 34.69), 4326)),
    ('Tokyo', 37.0, ST_SetSRID(ST_Point(139.69, 35.68), 4326)),
    ('Melbourne', 5.0, ST_SetSRID(ST_Point(144.96, -37.81), 4326)),
    ('Sydney', 5.0, ST_SetSRID(ST_Point(151.21, -33.87), 4326))
), clustered AS (
  SELECT place,
         geom,
         ST_ClusterKMeans(
           ST_Force4D(
             ST_Transform(ST_Force3D(geom), 4978),
             mvalue => population
           ),
           4,
           max_radius => 3000000
         ) OVER (ORDER BY place) AS cid
  FROM population
)
SELECT cid,
       string_agg(place, ', ' ORDER BY place) AS places,
       ST_Collect(geom ORDER BY place) AS cluster_geom
FROM clustered
GROUP BY cid
ORDER BY cid;
```

Output

```
0 | Melbourne, Sydney | MULTIPOINT((144.96 -37.81),(151.21 -33.87))
1 | Boston, New York | MULTIPOINT((-71.06 42.36),(-74.01 40.71))
2 | Osaka, Tokyo | MULTIPOINT((135.5 34.69),(139.69 35.68))
3 | London, Paris | MULTIPOINT((-0.13 51.51),(2.35 48.86))
```

Figure



Siehe auch

ST_ClusterDBSCAN, ST_ClusterIntersectingWin, ST_ClusterWithinWin, ST_ClusterIntersecting, ST_ClusterWithin, ST_Subdivide, ST_Force3D, ST_Force4D,

7.17.7 ST_ClusterWithin

ST_ClusterWithin — Aggregatfunktion, die Geometrien nach Trennungsabstand gruppiert.

Synopsis

geometry[] **ST_ClusterWithin**(geometry set g, float8 distance);

Beschreibung

Eine Aggregatfunktion, die ein Array von GeometryCollections zurückgibt, wobei jede Collection ein Cluster ist, das einige Eingabegeometrien enthält. Bei der Clusterbildung werden die Eingabegeometrien in Gruppen unterteilt, in denen jede Geometrie innerhalb des angegebenen Abstands von mindestens einer anderen Geometrie im selben Cluster liegt. Die Abstände sind kartesische Abstände in den Einheiten des SRID.

ST_ClusterWithin ist gleichbedeutend mit der Ausführung von ST_ClusterDBSCAN mit `minpoints => 0`.

Verfügbarkeit: 2.2.0

 Diese Methode unterstützt kreisförmige Strings und Kurven.

Beispiele

Code

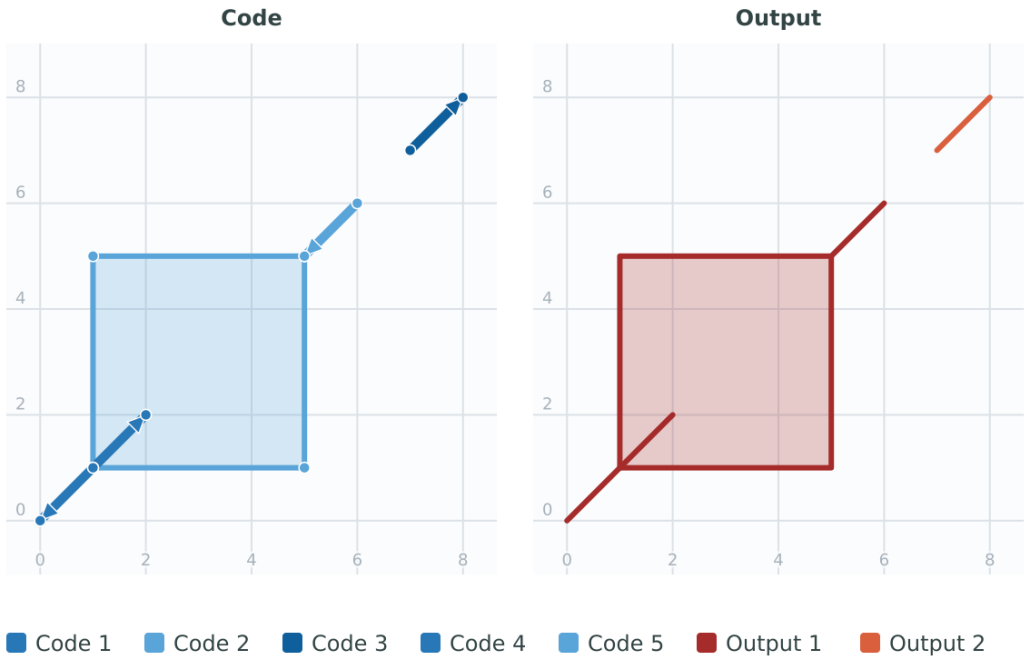
```
WITH testdata AS
  (SELECT unnest(ARRAY['LINESTRING (0 0,1 1)'::geometry,
                'LINESTRING (5 5,4 4)'::geometry,
                'LINESTRING (6 6,7 7)'::geometry,
                'LINESTRING (0 0,-1 -1)'::geometry,
                'POLYGON ((0 0,4 0,4 4,0 4,0 0))'::geometry]) AS geom)

SELECT unnest(ST_ClusterWithin(geom, 1.4)) FROM testdata;
```

Output

```
GEOMETRYCOLLECTION(LINESTRING(0 0,1 1),LINESTRING(5 5,4 4),LINESTRING(0 0,-1 -1),POLYGON((0 0,4 0,4 4,0 4,0 0)))
GEOMETRYCOLLECTION(LINESTRING(6 6,7 7))
```

Figure



Siehe auch

ST_ClusterWithinWin, ST_ClusterDBSCAN, ST_ClusterIntersecting, ST_ClusterIntersectingWin

7.17.8 ST_ClusterWithinWin

ST_ClusterWithinWin — Fensterfunktion, die eine Cluster-ID für jede Eingabegeometrie zurückgibt, Clustering anhand des Trennungsabstands.

Synopsis

integer **ST_ClusterWithinWin**(geometry winset geom, float8 distance);

Beschreibung

Eine Fensterfunktion, die für jede Eingabegeometrie eine Clusternummer zurückgibt. Bei der Clusterbildung werden die Geometrien in Gruppen unterteilt, in denen jede Geometrie innerhalb des angegebenen Abstands von mindestens einer anderen Geometrie im selben Cluster liegt. Die Abstände sind kartesische Abstände in den Einheiten des SRID.

ST_ClusterWithinWin ist gleichbedeutend mit der Ausführung von ST_ClusterDBSCAN mit `minpoints => 0`.

Verfügbarkeit: 3.4.0

Diese Methode unterstützt kreisförmige Strings und Kurven.

Beispiele

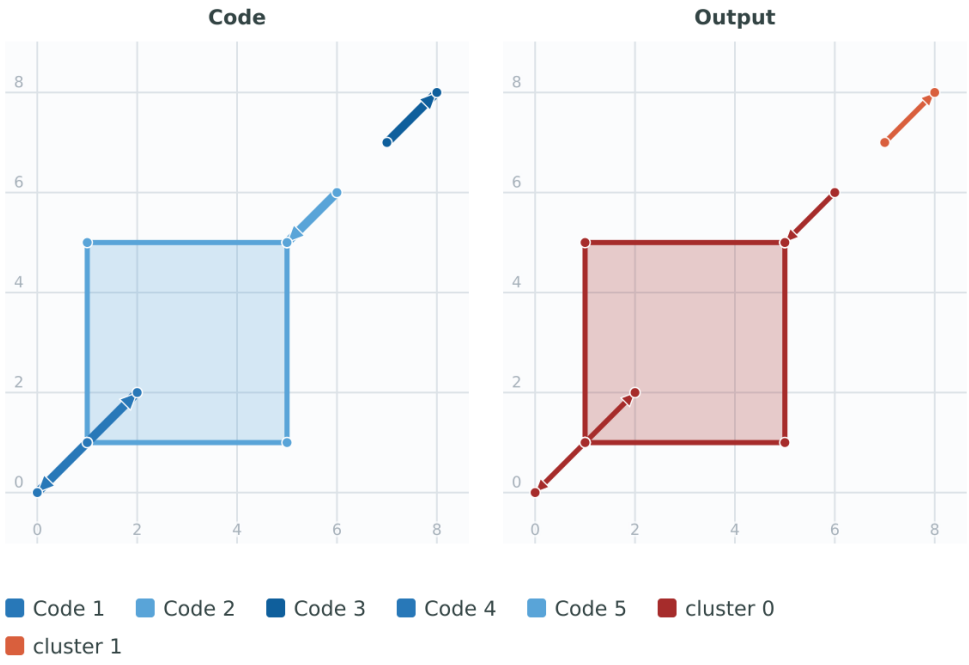
Code

```
WITH testdata AS (
  SELECT id, geom::geometry FROM (
  VALUES  (1, 'LINESTRING (0 0,1 1)'),
          (2, 'LINESTRING (5 5,4 4)'),
          (3, 'LINESTRING (6 6,7 7)'),
          (4, 'LINESTRING (0 0,-1 -1)'),
          (5, 'POLYGON ((0 0,4 0,4 4,0 4,0 0))')) AS t(id, geom)
)
SELECT id,
  geom,
  ST_ClusterWithinWin(geom, 1.4) OVER () AS cluster
FROM testdata;
```

Output

```
id |           st_astext             | cluster
---+---------------------------------+---------
  1 | LINESTRING(0 0,1 1)             |       0
  2 | LINESTRING(5 5,4 4)             |       0
  3 | LINESTRING(6 6,7 7)             |       1
  4 | LINESTRING(0 0,-1 -1)           |       0
  5 | POLYGON((0 0,4 0,4 4,0 4,0 0)) |       0
```

Figure



Siehe auch

ST_ClusterWithin, ST_ClusterDBSCAN, ST_ClusterIntersecting, ST_ClusterIntersectingWin,

7.18 Bounding Box-Funktionen

7.18.1 Box2D

Box2D — Gibt ein BOX2D zurück, das die 2D-Ausdehnung einer Geometrie darstellt.

Synopsis

box2d **Box2D**(geometry geom);

Beschreibung

Gibt eine box2d zurück, die die 2D-Ausdehnung der Geometrie darstellt.

Erweiterung: Mit 2.0.0 wurde die Unterstützung für polyedrische Oberflächen, Dreiecke und TIN eingeführt.

 Diese Methode unterstützt kreisförmige Strings und Kurven.

 Diese Funktion unterstützt polyedrische Flächen.

 Diese Funktion unterstützt Dreiecke und dreieckige unregelmäßige Netzoberflächen (TIN).

Beispiele

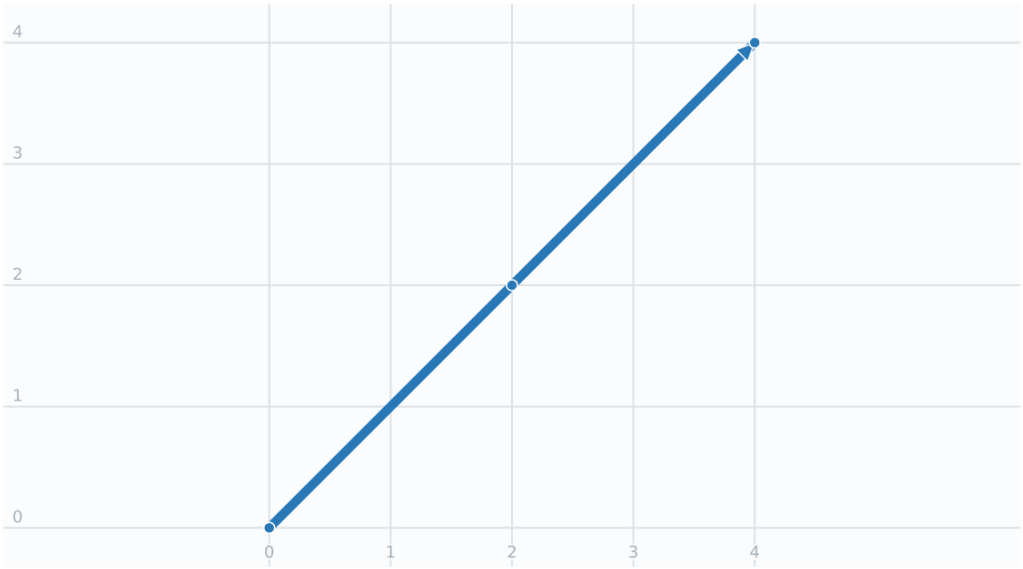
Code

```
SELECT Box2D('LINESTRING(1 2,3 4,5 6)'::geometry);
```

Output

```
BOX(1 2,5 6)
```

Figure



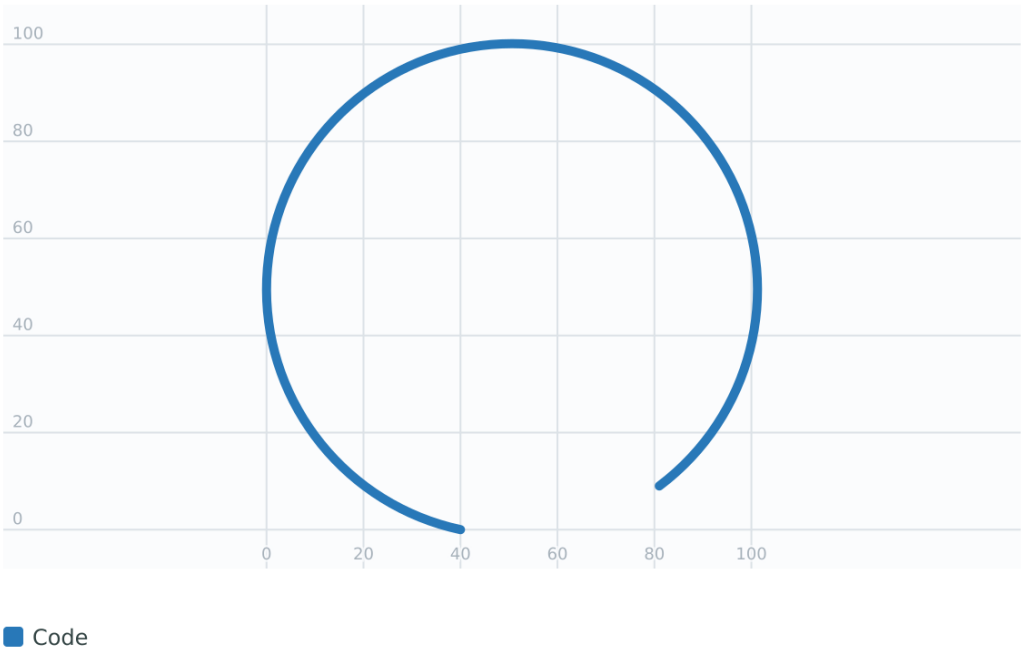
Code

```
SELECT Box2D('CIRCULARSTRING(220268 150415,220227 150505,220227 150406)'::geometry);
```

Output

```
BOX(220186.99512189245 150406,220288.24878054656 150506.12682932706)
```

Figure



Siehe auch

Box3D, ST_GeomFromText

7.18.2 Box3D

Box3D — Gibt ein BOX3D zurück, das die 3D-Ausdehnung einer Geometrie darstellt.

Synopsis

box3d **Box3D**(geometry geom);

Beschreibung

Gibt eine box3d zurück, die die 3D-Ausdehnung der Geometrie darstellt.

Erweiterung: Mit 2.0.0 wurde die Unterstützung für polyedrische Oberflächen, Dreiecke und TIN eingeführt.

Diese Methode unterstützt kreisförmige Strings und Kurven.

Diese Funktion unterstützt polyedrische Flächen.

Diese Funktion unterstützt Dreiecke und dreieckige unregelmäßige Netzoberflächen (TIN).

Diese Funktion unterstützt 3d und lässt den Z-Index nicht fallen.

Beispiele

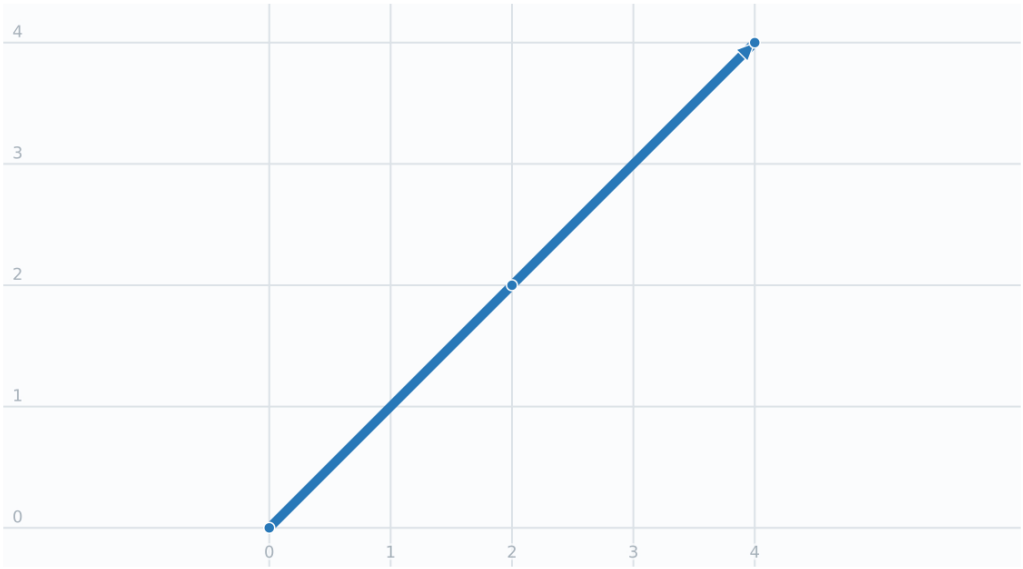
Code

```
SELECT Box3D('LINESTRING(1 2 3,3 4 5,5 6 5)'::geometry);
```

Output

```
BOX3D(1 2 3,5 6 5)
```

Figure



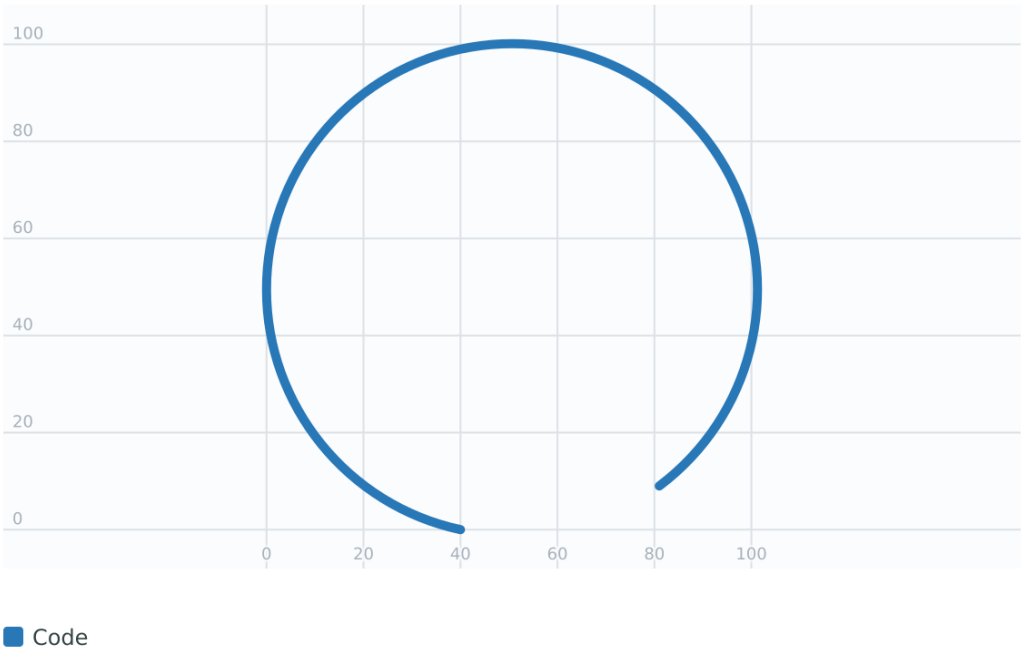
Code

```
SELECT Box3D('CIRCULARSTRING(220268 150415 1,220227 150505 1,220227 150406 1)'::geometry);
```

Output

```
BOX3D(220186.99512189245 150406 1,220288.24878054656 150506.12682932706 1)
```

Figure



Siehe auch

Box2D, ST_GeomFromEWKT

7.18.3 ST_EstimatedExtent

ST_EstimatedExtent — Gibt die geschätzte Ausdehnung einer räumlichen Tabelle zurück.

Synopsis

box2d **ST_EstimatedExtent**(text schema_name, text table_name, text geocolumn_name, boolean parent_only);
box2d **ST_EstimatedExtent**(text schema_name, text table_name, text geocolumn_name);
box2d **ST_EstimatedExtent**(text table_name, text geocolumn_name);

Beschreibung

Gibt die geschätzte Ausdehnung einer räumlichen Tabelle als box2d zurück. Falls nicht angegeben, wird das aktuelle Schema verwendet. Die geschätzte Ausdehnung wird aus den Statistiken der Geometriespalte entnommen. Dies ist in der Regel viel schneller als die Berechnung der genauen Ausdehnung der Tabelle mit ST_Extent oder ST_3DExtent.

Standardmäßig werden auch Statistiken aus Kindtabellen (Tabellen mit INHERITS) verwendet, sofern diese verfügbar sind. Wenn `parent_only` auf TRUE gesetzt ist, werden nur Statistiken für die angegebene Tabelle verwendet und untergeordnete Tabellen ignoriert.

Für PostgreSQL >= 8.0.0 werden die Statistiken durch VACUUM ANALYZE erfasst, und der Ergebnisumfang entspricht etwa 95% des tatsächlichen Umfangs. Für PostgreSQL < 8.0.0 werden die Statistiken durch Ausführen von `update_geometry_stats()` erfasst und der Ergebnisumfang ist exakt.



Anmerkung
Wenn keine Statistiken vorhanden sind (leere Tabelle oder kein ANALYZE-Aufruf), gibt diese Funktion NULL zurück. Vor Version 1.5.4 wurde stattdessen eine Ausnahme ausgelöst.



Anmerkung
Namen für Tabellen und/oder Namensräume, die Sonderzeichen und Anführungszeichen enthalten, erfordern möglicherweise eine besondere Behandlung. Anmerkung eines Benutzers: "Für die Bezeichnung von Schemata und Tabellen sollten Sie Escape-Regeln verwenden, um einen doppelt in Anführungszeichen gesetzten String zu erzeugen, und anschließend das erste und letzte Anführungszeichen entfernen. Für Geometriespalten können Sie den Namen unverändert angeben."

Verfügbarkeit: 1.0.0

Geändert: 2.1.0. Bis zu 2.0.x hieß dies ST_Estimated_Extent.

 Diese Methode unterstützt kreisförmige Strings und Kurven.

Beispiele

Code

```
SELECT ST_EstimatedExtent('ny', 'edges', 'geom');
```

Output

```
st_estimatedextent
--------------------------------------------------
 BOX(-8877653 4912316,-8010225.5 5589284)
(1 row)
```

Code

```
SELECT ST_EstimatedExtent('feature_poly', 'geom');
```

Output

```
st_estimatedextent
----------------------------------------------------------------------
 BOX(-124.659652709961 24.6830825805664,-67.7798080444336 49.0012092590332)
(1 row)
```

Siehe auch

ST_Extent, ST_GeomFromText

7.18.4 ST_Expand

ST_Expand — Gibt einen Begrenzungsrahmen zurück, der aus einem anderen Begrenzungsrahmen oder einer Geometrie erweitert wurde.

Synopsis

geometry **ST_Expand**(geometry geom, float units_to_expand);
geometry **ST_Expand**(geometry geom, float dx, float dy, float dz=0, float dm=0);
box2d **ST_Expand**(box2d box, float units_to_expand);
box2d **ST_Expand**(box2d box, float dx, float dy);
box3d **ST_Expand**(box3d box, float units_to_expand);
box3d **ST_Expand**(box3d box, float dx, float dy, float dz=0);

Beschreibung

Gibt einen Begrenzungsrahmen zurück, der aus dem Begrenzungsrahmen der Eingabe expandiert wurde, indem entweder ein einzelner Abstand angegeben wird, mit dem der Rahmen auf beiden Achsen expandiert werden soll, oder indem ein Expansionsabstand für jede Achse angegeben wird. Verwendet Doppelpräzision. Kann für Abstandsabfragen verwendet werden oder um einen Bounding-Box-Filter zu einer Abfrage hinzuzufügen, um einen räumlichen Index zu nutzen.

Zusätzlich zu der Version von ST_Expand, die eine Geometrie akzeptiert und zurückgibt, werden Varianten bereitgestellt, die die Datentypen box2d und box3d akzeptieren und zurückgeben.

Die Entfernungen sind in den Einheiten des räumlichen Bezugssystems der Eingabe angegeben.

ST_Expand ist ähnlich wie ST_Buffer, mit dem Unterschied, dass ST_Expand das Begrenzungsrechteck entlang jeder Achse erweitert, während Pufferung eine Geometrie in alle Richtungen erweitert.



Anmerkung
Vor Version 1.3 wurde ST_Expand in Verbindung mit ST_Distance verwendet, um indizierbare Abstandsabfragen durchzuführen. Zum Beispiel: `geom && ST_Expand('POINT(10 20)', 10) AND ST_Distance(geom, 'POINT(10 20)') < 10`. Diese Funktion wurde durch die einfachere und effizientere Funktion ST_DWithin ersetzt.

Verfügbarkeit: 1.5.0 Verhalten geändert, um double precision statt float4 Koordinaten auszugeben.

Erweiterung: Mit 2.0.0 wurde die Unterstützung für polyedrische Oberflächen, Dreiecke und TIN eingeführt.

Verbessert: In Version 2.3.0 wurde die Möglichkeit hinzugefügt, eine Box um unterschiedliche Beträge in verschiedenen Dimensionen zu erweitern.

 Diese Funktion unterstützt polyedrische Flächen.

 Diese Funktion unterstützt Dreiecke und dreieckige unregelmäßige Netzoberflächen (TIN).

Beispiele



Anmerkung
Die folgenden Beispiele verwenden den US National Atlas Equal Area (SRID=2163), der eine Meterprojektion ist

10-meter expanded box around bbox of a linestring.

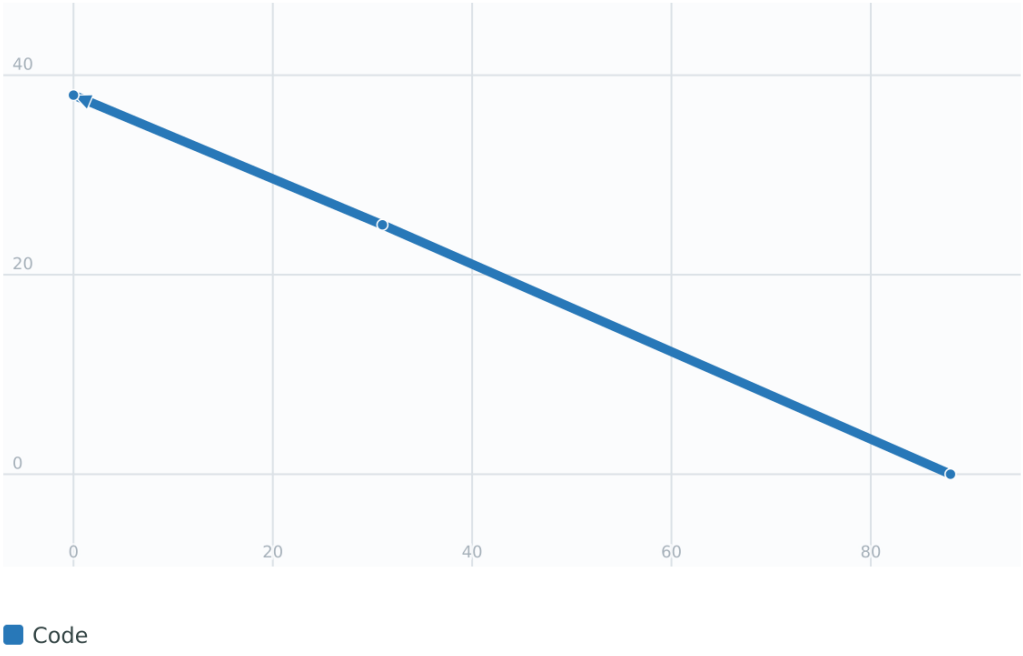
Code

```
SELECT CAST(ST_Expand(ST_GeomFromText('LINESTRING(2312980 110676,2312923 110701,2312892 ↩
    110714)', 2163), 10) As box2d);
```

Output

```
BOX(2312882 110666,2312990 110724)
```

Figure



This example expands a 3D box by 10 meters.

Code

```
SELECT ST_Expand(CAST('BOX3D(778783 2951741 1,794875 2970042.61545891 10)' AS box3d), 10)
```

Output

```
BOX3D(778773 2951731 -9,794885 2970052.61545891 20)
```

This example renders the text representation of a 10-meter expanded box around a point geometry.

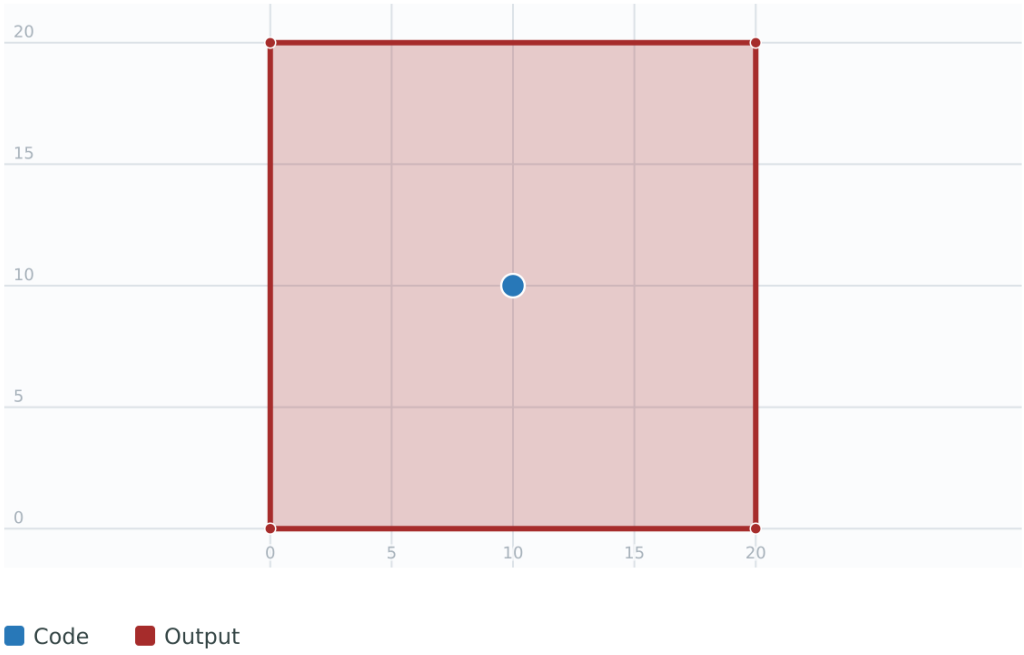
Code

```
SELECT ST_Expand('SRID=2163;POINT(2312980 110676)'::geometry, 10);
```

Output

```
SRID=2163;POLYGON((2312970 110666,2312970 110686,2312990 110686,2312990 110666,2312970 110666))
```

Figure



Siehe auch

ST_Buffer, ST_DWithin, ST_SRID

7.18.5 ST_Extent

ST_Extent — Aggregatfunktion, die den Begrenzungsrahmen von Geometrien zurückgibt.

Synopsis

box2d **ST_Extent**(geometry set geomfield);

Beschreibung

Eine aggregierte Funktion, die eine box2d Bounding Box zurückgibt, die einen Satz von Geometrien begrenzt.

Die Bounding-Box-Koordinaten sind im räumlichen Bezugssystem der Eingabegeometrien.

ST_Extent ist vom Konzept her ähnlich dem SDO_AGGR_MBR von Oracle Spatial/Locator.



Anmerkung
ST_Extent gibt auch bei 3D-Geometrien nur Boxen mit X- und Y-Ordinaten zurück. Um XYZ-Ordinaten zurückzugeben, verwenden Sie ST_3DExtent.



Anmerkung
Der zurückgegebene Wert `box3d` enthält keinen SRID. Verwenden Sie ST_SetSRID, um ihn in eine Geometrie mit SRID-Metadaten umzuwandeln. Der SRID ist derselbe wie bei den Eingabegeometrien.

Erweiterung: Mit 2.0.0 wurde die Unterstützung für polyedrische Oberflächen, Dreiecke und TIN eingeführt.


Diese Funktion unterstützt polyedrische Flächen.


Diese Funktion unterstützt Dreiecke und dreieckige unregelmäßige Netzoberflächen (TIN).

Beispiele



Anmerkung
Die folgenden Beispiele verwenden Massachusetts State Plane ft (SRID=2249)

Code

```
SELECT ST_Extent(geom) AS bextent FROM sometable;
```

Output

```
BOX(739651.875 2908247.25,794875.8125 2970042.75)
```

Return extent of each geometry category.

Code

```
SELECT ST_Extent(geom) as bextent
FROM sometable
GROUP BY category ORDER BY category;
```

Output

```
bextent                        |         name
-----------------------------------------------------+-----------------
 BOX(778783.5625 2951741.25,794875.8125 2970042.75) | A
 BOX(751315.8125 2919164.75,765202.6875 2935417.25) | B
 BOX(739651.875 2917394.75,756688.375 2935866)      | C
```

This example converts the extent back into a geometry and renders its extended text representation.

Code

```
SELECT ST_SetSRID(ST_Extent(geom), 2249) AS bextent FROM sometable;
```

Output

```
SRID=2249;POLYGON((739651.875 2908247.25,739651.875 2970042.75,794875.8125 2970042.75,
794875.8125 2908247.25,739651.875 2908247.25))
```

Siehe auch

ST_EstimatedExtent, ST_3DExtent, ST_SetSRID

7.18.6 ST_3DExtent

ST_3DExtent — Aggregatfunktion, die den 3D-Begrenzungsrahmen von Geometrien zurückgibt.

Synopsis

box3d **ST_3DExtent**(geometry set geomfield);

Beschreibung

Eine Aggregatfunktion, die eine box3d (einschließlich Z-Ordinate) Bounding Box zurückgibt, die einen Satz von Geometrien begrenzt.

Die Bounding-Box-Koordinaten sind im räumlichen Bezugssystem der Eingabegeometrien.



Anmerkung
Der zurückgegebene Wert `box3d` enthält keinen SRID. Verwenden Sie ST_SetSRID, um ihn in eine Geometrie mit SRID-Metadaten umzuwandeln. Der SRID ist derselbe wie bei den Eingabegeometrien.

Erweiterung: Mit 2.0.0 wurde die Unterstützung für polyedrische Oberflächen, Dreiecke und TIN eingeführt.

Geändert: 2.0.0 In früheren Versionen wurde dies als ST_Extent3D bezeichnet.

 Diese Funktion unterstützt 3d und lässt den Z-Index nicht fallen.

 Diese Methode unterstützt kreisförmige Strings und Kurven.

 Diese Funktion unterstützt polyedrische Flächen.

 Diese Funktion unterstützt Dreiecke und dreieckige unregelmäßige Netzoberflächen (TIN).

Beispiele

Code

```
SELECT ST_3DExtent(foo.geom) As b3extent
FROM (SELECT ST_MakePoint(x, y, z) As geom
    FROM generate_series(1, 3) As x
        CROSS JOIN generate_series(1, 2) As y
        CROSS JOIN generate_series(0, 2) As Z) As foo;
```

Output

```
BOX3D(1 1 0,3 2 2)
```

This example returns the extent of several elevated CircularStrings.

Code

```
SELECT ST_3DExtent(foo.geom) As b3extent
FROM (SELECT ST_Translate(ST_Force3DZ(ST_LineToCurve(ST_Buffer(ST_Point(x, y), 1))), 0, 0, z ↩
    ) As geom
    FROM generate_series(1, 3) As x
        CROSS JOIN generate_series(1, 2) As y
        CROSS JOIN generate_series(0, 2) As Z) As foo;
```

Output

```
BOX3D(0 0 0,4 3 2)
```

Siehe auch

ST_Extent, ST_Force3DZ, ST_SetSRID

7.18.7 ST_MakeBox2D

ST_MakeBox2D — Erzeugt ein BOX2D, das durch zwei 2D-Punktgeometrien definiert ist.

Synopsis

box2d **ST_MakeBox2D**(geometry pointLowLeft, geometry pointUpRight);

Beschreibung

Erzeugt eine box2d, die durch zwei Punktgeometrien definiert ist. Dies ist nützlich für Bereichsabfragen.

Beispiele

Return all features that fall within or partly reside in a US National Atlas coordinate bounding box. It is assumed here that the geometries are stored with SRID=2163 (US National Atlas equal area).

Code

```
SELECT feature_id, feature_name, geom
FROM features
WHERE geom && ST_SetSRID(
  ST_MakeBox2D(
    ST_Point(-989502.1875, 528439.5625),
    ST_Point(-987121.375, 529933.1875)
  ),
  2163
);
```

Siehe auch

ST_Point, ST_SetSRID, ST_SRID

7.18.8 ST_3DMakeBox

ST_3DMakeBox — Erzeugt einen BOX3D, der durch zwei 3D-Punktgeometrien definiert ist.

Synopsis

box3d **ST_3DMakeBox**(geometry point3DLowLeftBottom, geometry point3DUpRightTop);

Beschreibung

Erzeugt eine box3d, die durch zwei 3D-Punktgeometrien definiert ist.

Diese Funktion unterstützt 3D und lässt den Z-Index nicht fallen.

Geändert: 2.0.0 In früheren Versionen hieß diese Funktion ST_MakeBox3D

Beispiele

Code

```
SELECT ST_3DMakeBox(ST_MakePoint(-989502.1875, 528439.5625, 10),
    ST_MakePoint(-987121.375 , 529933.1875, 10))
```

Output

```
BOX3D(-989502.1875 528439.5625 10,-987121.375 529933.1875 10)
```

Siehe auch

ST_MakePoint, ST_SetSRID, ST_SRID

7.18.9 ST_XMax

ST_XMax — Gibt die X-Maxima eines 2D- oder 3D-Begrenzungsrahmens oder einer Geometrie zurück.

Synopsis

float **ST_XMax**(box3d aGeomorBox2DorBox3D);

Beschreibung

Gibt die X-Maxima eines 2D- oder 3D-Begrenzungsrahmens oder einer Geometrie zurück.



Anmerkung
Obwohl diese Funktion nur für box3d definiert ist, funktioniert sie aufgrund des automatischen Castings auch für box2d- und Geometriewerte. Sie akzeptiert jedoch keine Geometrie- oder box2d-Textdarstellung, da diese nicht automatisch gecastet werden.

Diese Funktion unterstützt 3d und lässt den Z-Index nicht fallen.

Diese Methode unterstützt kreisförmige Strings und Kurven.

Beispiele

Code

```
SELECT ST_XMax('BOX3D(1 2 3,4 5 6)');
```

Output

```
4
```

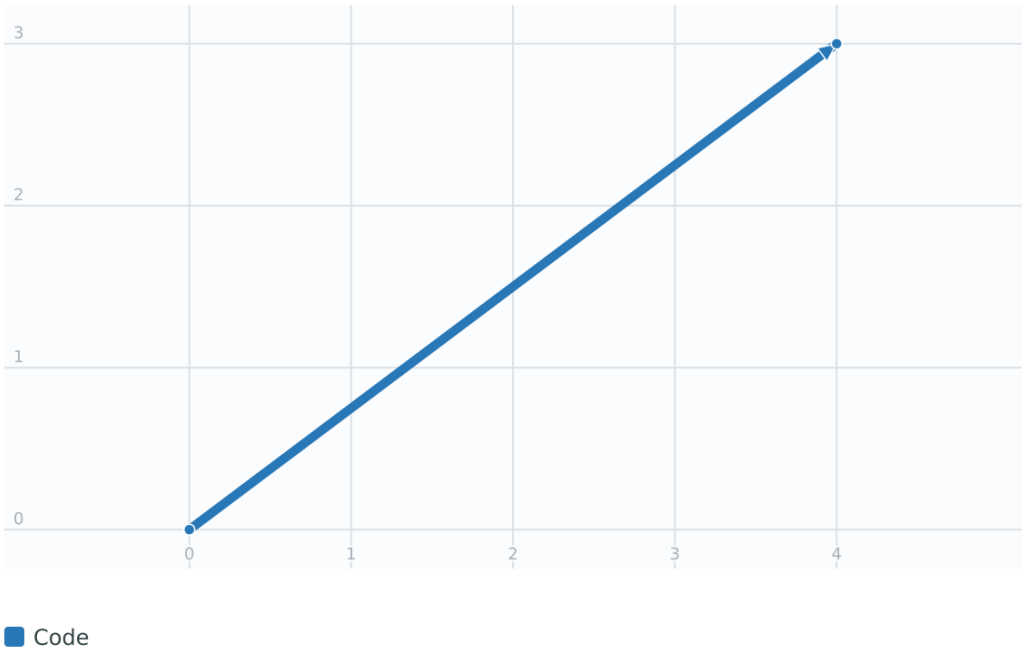
Code

```
SELECT ST_XMax('LINESTRING(1 3 4,5 6 7)'::geometry);
```

Output

```
5
```

Figure



Code

```
SELECT ST_XMax(CAST('BOX(-3 2,3 4)' AS box2d));
```

Output

```
3
```

Observe THIS DOES NOT WORK because it will try to auto-cast the string representation to a BOX3D.

Code

```
SELECT ST_XMax('LINESTRING(1 3,5 6)');
```

Output

```
ERROR: BOX3D parser - doesn't start with BOX3D(
```

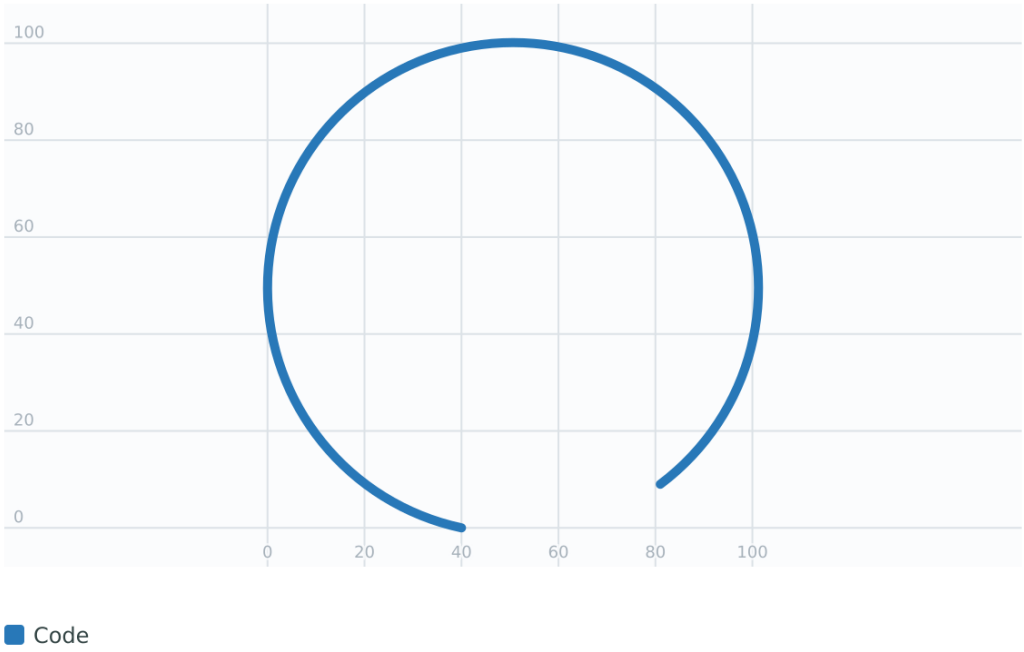
Code

```
SELECT ST_XMax('CIRCULARSTRING(220268 150415 1,220227 150505 2,220227 150406 3)'::geometry);
```

Output

```
220288.248780547
```

Figure



Siehe auch

ST_XMin, ST_YMin, ST_YMax, ST_ZMin, ST_ZMax, ST_MMin, ST_MMax

7.18.10 ST_XMin

ST_XMin — Gibt die X-Minima eines 2D- oder 3D-Begrenzungsrahmens oder einer Geometrie zurück.

Synopsis

float **ST_XMin**(box3d aGeomorBox2DorBox3D);

Beschreibung

Gibt die X-Minima eines 2D- oder 3D-Begrenzungsrahmens oder einer Geometrie zurück.



Anmerkung
Obwohl diese Funktion nur für box3d definiert ist, funktioniert sie aufgrund des automatischen Castings auch für box2d- und Geometriewerte. Sie akzeptiert jedoch keine Geometrie- oder box2d-Textdarstellung, da diese nicht automatisch gecastet werden.

 Diese Funktion unterstützt 3d und lässt den Z-Index nicht fallen.

 Diese Methode unterstützt kreisförmige Strings und Kurven.

Beispiele

Code

```
SELECT ST_XMin('BOX3D(1 2 3,4 5 6)');
```

Output

```
1
```

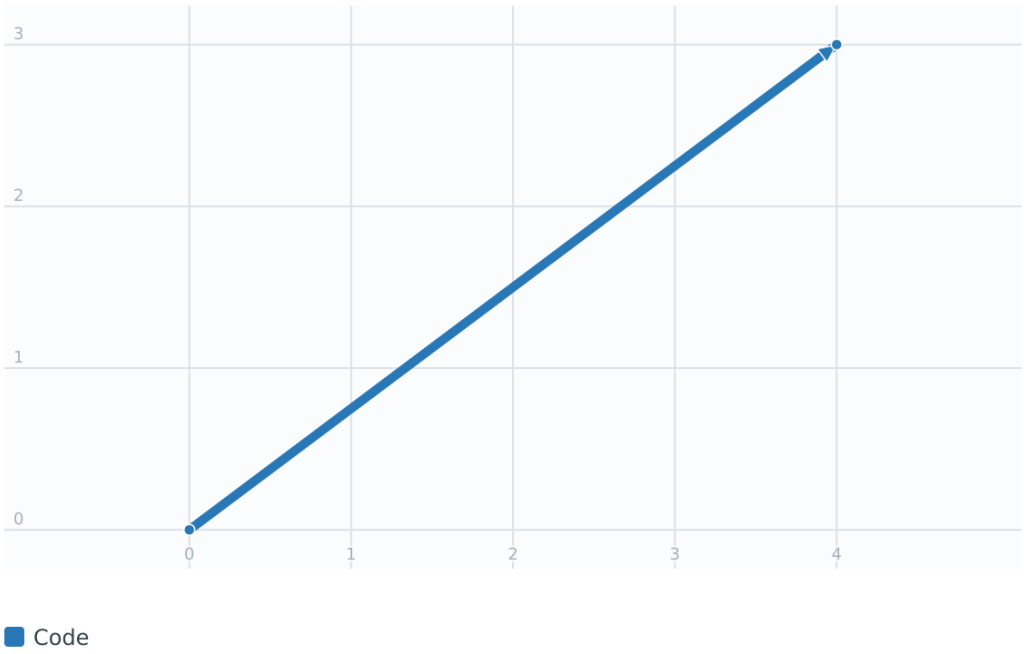
Code

```
SELECT ST_XMin('LINESTRING(1 3 4,5 6 7)'::geometry);
```

Output

```
1
```

Figure



Code

```
SELECT ST_XMin(CAST('BOX(-3 2,3 4)' AS box2d));
```

Output

```
-3
```

Observe THIS DOES NOT WORK because it will try to auto-cast the string representation to a BOX3D.

Code

```
SELECT ST_XMin('LINESTRING(1 3,5 6)');
```

Output

```
ERROR: BOX3D parser - doesn't start with BOX3D(
```

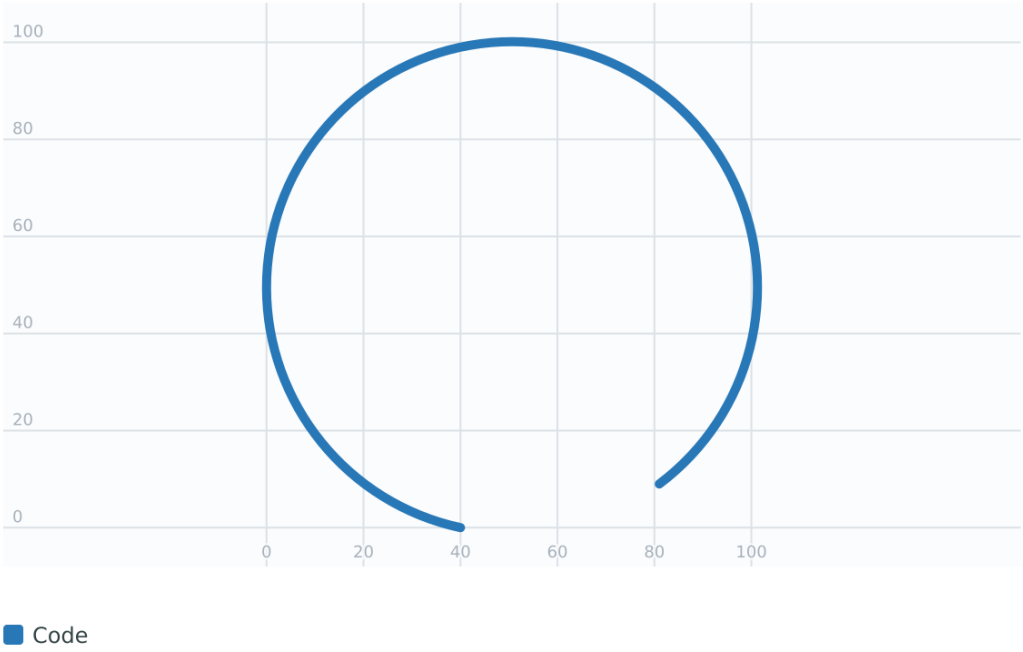
Code

```
SELECT ST_XMin('CIRCULARSTRING(220268 150415 1,220227 150505 2,220227 150406 3)'::geometry);
```

Output

```
220186.995121892
```

Figure



Siehe auch

ST_XMax, ST_YMin, ST_YMax, ST_ZMin, ST_ZMax, ST_MMin, ST_MMax

7.18.11 ST_YMax

ST_YMax — Gibt die Y-Maxima eines 2D- oder 3D-Begrenzungsrahmens oder einer Geometrie zurück.

Synopsis

float **ST_YMax**(box3d aGeomorBox2DorBox3D);

Beschreibung

Gibt die Y-Maxima eines 2D- oder 3D-Begrenzungsrahmens oder einer Geometrie zurück.



Anmerkung
Obwohl diese Funktion nur für box3d definiert ist, funktioniert sie aufgrund des automatischen Castings auch für box2d- und Geometriewerte. Sie akzeptiert jedoch keine Geometrie- oder box2d-Textdarstellung, da diese nicht automatisch gecastet werden.

 Diese Funktion unterstützt 3d und lässt den Z-Index nicht fallen.

 Diese Methode unterstützt kreisförmige Strings und Kurven.

Beispiele

Code

```
SELECT ST_YMax('BOX3D(1 2 3,4 5 6)');
```

Output

```
5
```

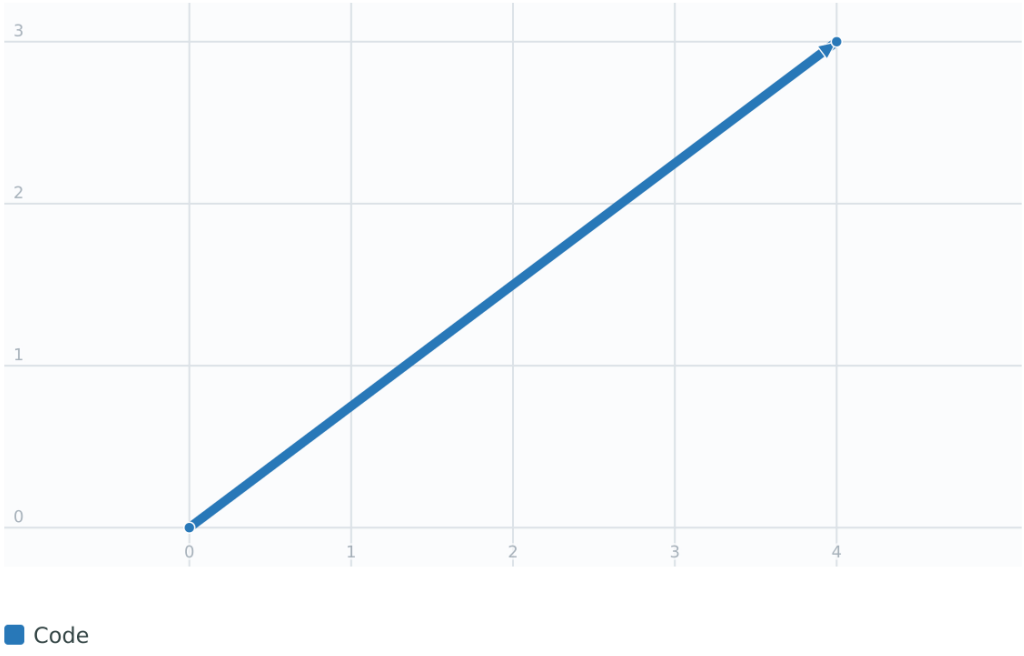
Code

```
SELECT ST_YMax('LINESTRING(1 3 4,5 6 7)'::geometry);
```

Output

```
6
```

Figure



Code

```
SELECT ST_YMax(CAST('BOX(-3 2,3 4)' AS box2d));
```

Output

```
4
```

Observe THIS DOES NOT WORK because it will try to auto-cast the string representation to a BOX3D.

Code

```
SELECT ST_YMax('LINESTRING(1 3,5 6)');
```

Output

```
ERROR: BOX3D parser - doesn't start with BOX3D(
```

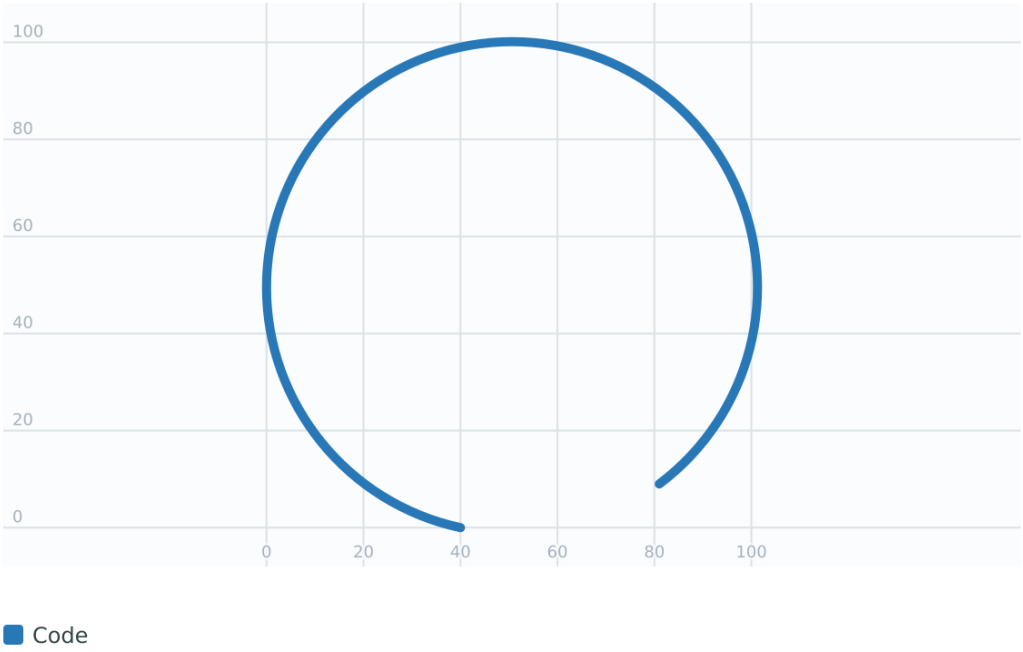
Code

```
SELECT ST_YMax('CIRCULARSTRING(220268 150415 1,220227 150505 2,220227 150406 3)'::geometry);
```

Output

```
150506.126829327
```

Figure



Siehe auch

ST_XMin, ST_XMax, ST_YMin, ST_ZMin, ST_ZMax, ST_MMin, ST_MMax

7.18.12 ST_YMin

ST_YMin — Gibt die Y-Minima eines 2D- oder 3D-Begrenzungsrahmens oder einer Geometrie zurück.

Synopsis

float **ST_YMin**(box3d aGeomorBox2DorBox3D);

Beschreibung

Gibt die Y-Minima eines 2D- oder 3D-Begrenzungsrahmens oder einer Geometrie zurück.

Anmerkung
Obwohl diese Funktion nur für box3d definiert ist, funktioniert sie aufgrund des automatischen Castings auch für box2d- und Geometriewerte. Sie akzeptiert jedoch keine Geometrie- oder box2d-Textdarstellung, da diese nicht automatisch gecastet werden.

 Diese Funktion unterstützt 3d und lässt den Z-Index nicht fallen.

 Diese Methode unterstützt kreisförmige Strings und Kurven.

Beispiele

Code

```
SELECT ST_YMin('BOX3D(1 2 3,4 5 6)');
```

Output

```
2
```

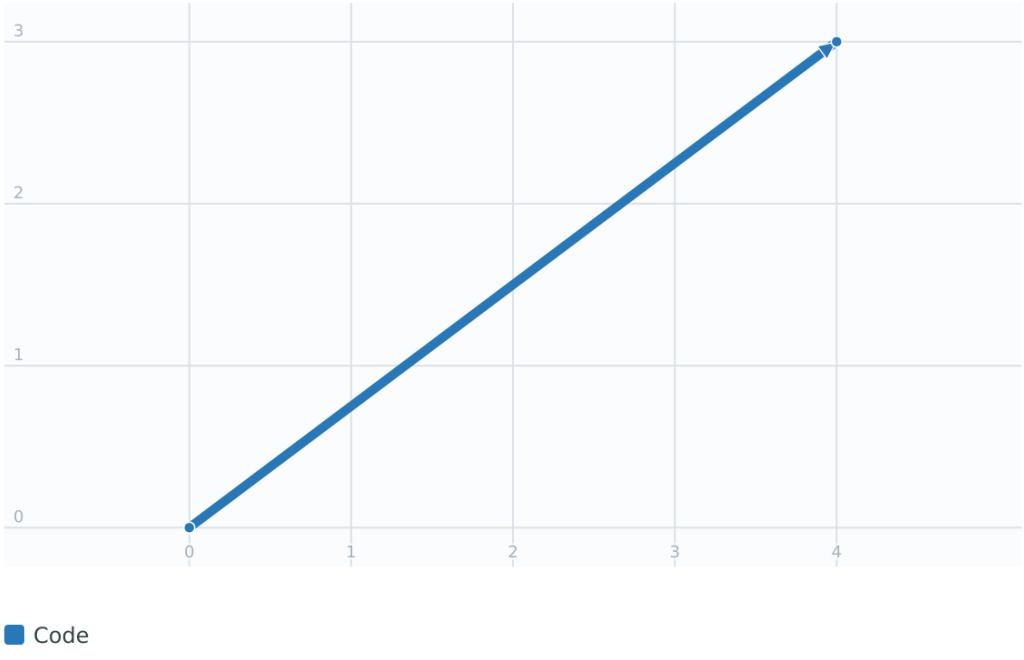
Code

```
SELECT ST_YMin('LINESTRING(1 3 4,5 6 7)'::geometry);
```

Output

```
3
```

Figure



Code

```
SELECT ST_YMin(CAST('BOX(-3 2,3 4)' AS box2d));
```

Output

```
2
```

Observe THIS DOES NOT WORK because it will try to auto-cast the string representation to a BOX3D.

Code

```
SELECT ST_YMin('LINESTRING(1 3,5 6)');
```

Output

```
ERROR: BOX3D parser - doesn't start with BOX3D(
```

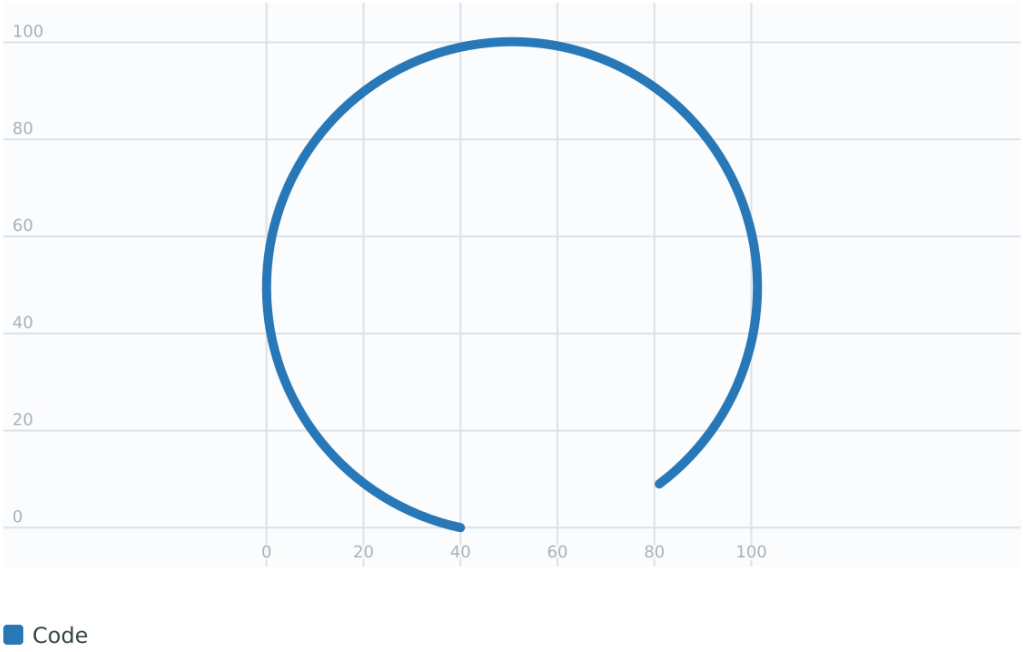
Code

```
SELECT ST_YMin('CIRCULARSTRING(220268 150415 1,220227 150505 2,220227 150406 3)'::geometry);
```

Output

```
150406
```

Figure



Siehe auch

ST_GeomFromEWKT, ST_XMin, ST_XMax, ST_YMax, ST_ZMin, ST_ZMax, ST_MMin, ST_MMax

7.18.13 ST_ZMax

ST_ZMax — Gibt die Z-Maxima eines 2D- oder 3D-Begrenzungsrahmens oder einer Geometrie zurück.

Synopsis

float **ST_ZMax**(box3d aGeomorBox2DorBox3D);

Beschreibung

Gibt die Z-Maxima eines 2D- oder 3D-Begrenzungsrahmens oder einer Geometrie zurück.



Anmerkung
Obwohl diese Funktion nur für box3d definiert ist, funktioniert sie aufgrund des automatischen Castings auch für box2d- und Geometriewerte. Sie akzeptiert jedoch keine Geometrie- oder box2d-Textdarstellung, da diese nicht automatisch gecastet werden.

 Diese Funktion unterstützt 3d und lässt den Z-Index nicht fallen.

 Diese Methode unterstützt kreisförmige Strings und Kurven.

Beispiele

Code

```
SELECT ST_ZMax('BOX3D(1 2 3,4 5 6)');
```

Output

```
6
```

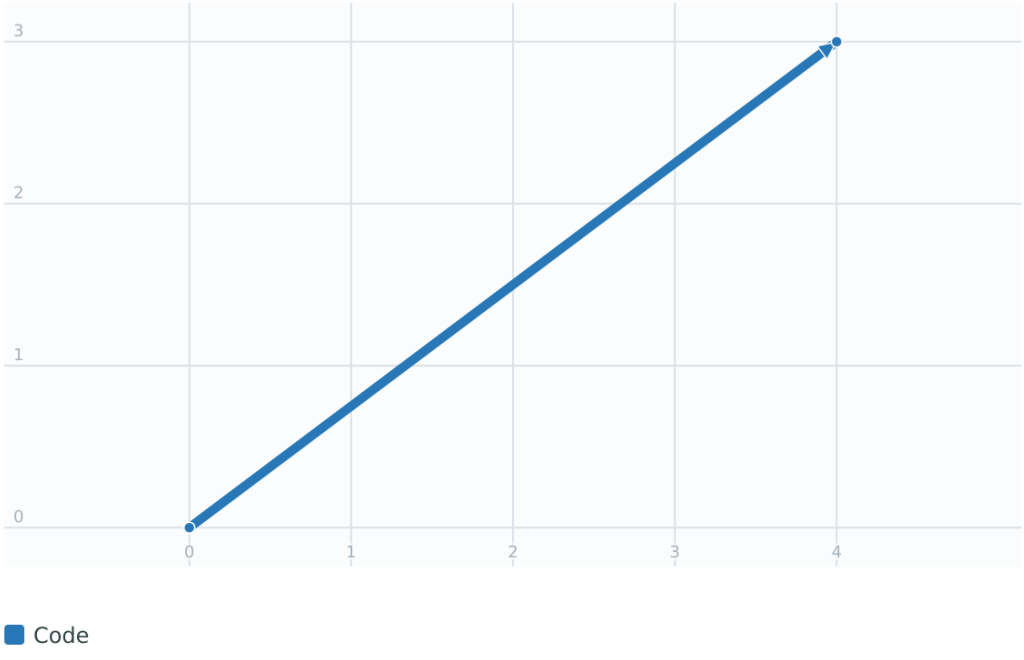
Code

```
SELECT ST_ZMax('LINESTRING(1 3 4,5 6 7)'::geometry);
```

Output

```
7
```

Figure



Code

```
SELECT ST_ZMax('BOX3D(-3 2 1,3 4 1)');
```

Output

```
1
```

Observe THIS DOES NOT WORK because it will try to auto-cast the string representation to a BOX3D.

Code

```
SELECT ST_ZMax('LINESTRING(1 3 4,5 6 7)');
```

Output

```
ERROR: BOX3D parser - doesn't start with BOX3D(
```

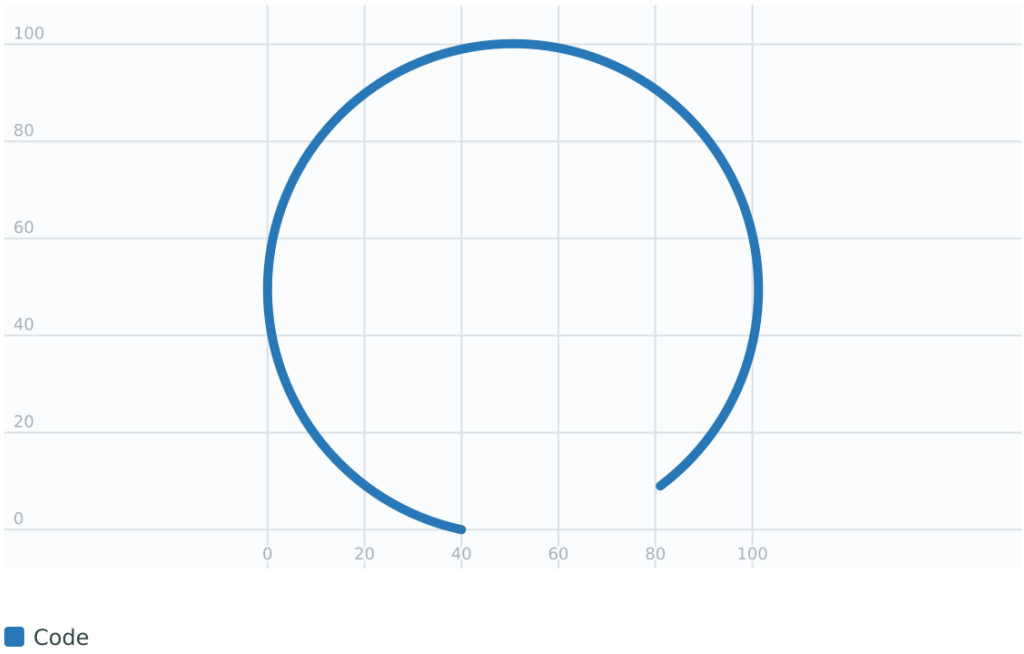
Code

```
SELECT ST_ZMax('CIRCULARSTRING(220268 150415 1,220227 150505 2,220227 150406 3)'::geometry);
```

Output

```
3
```

Figure



Siehe auch

ST_GeomFromEWKT, ST_XMin, ST_XMax, ST_YMin, ST_YMax, ST_ZMin, ST_MMin, ST_MMax

7.18.14 ST_ZMin

ST_ZMin — Gibt die Z-Minima eines 2D- oder 3D-Begrenzungsrahmens oder einer Geometrie zurück.

Synopsis

float **ST_ZMin**(box3d aGeomorBox2DorBox3D);

Beschreibung

Gibt die Z-Minima eines 2D- oder 3D-Begrenzungsrahmens oder einer Geometrie zurück.



Anmerkung
Obwohl diese Funktion nur für box3d definiert ist, funktioniert sie aufgrund des automatischen Castings auch für box2d- und Geometriewerte. Sie akzeptiert jedoch keine Geometrie- oder box2d-Textdarstellung, da diese nicht automatisch gecastet werden.

 Diese Funktion unterstützt 3d und lässt den Z-Index nicht fallen.

 Diese Methode unterstützt kreisförmige Strings und Kurven.

Beispiele

Code

```
SELECT ST_ZMin('BOX3D(1 2 3,4 5 6)');
```

Output

```
3
```

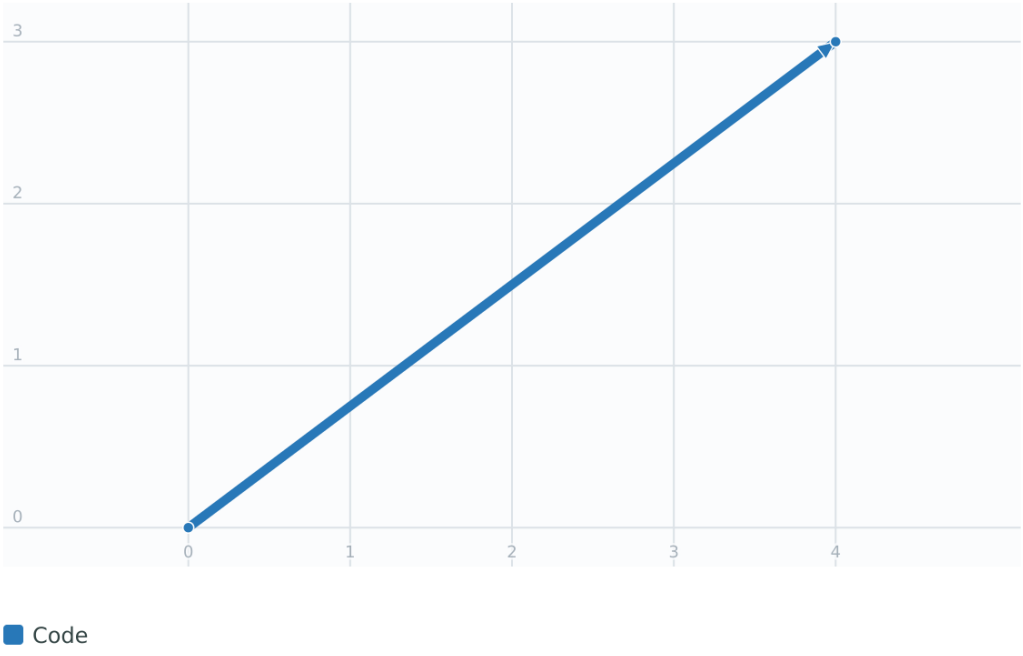
Code

```
SELECT ST_ZMin('LINESTRING(1 3 4,5 6 7)'::geometry);
```

Output

```
4
```

Figure



Code

```
SELECT ST_ZMin('BOX3D(-3 2 1,3 4 1)');
```

Output

```
1
```

Observe THIS DOES NOT WORK because it will try to auto-cast the string representation to a BOX3D.

Code

```
SELECT ST_ZMin('LINESTRING(1 3 4,5 6 7)');
```

Output

```
ERROR: BOX3D parser - doesn't start with BOX3D(
```

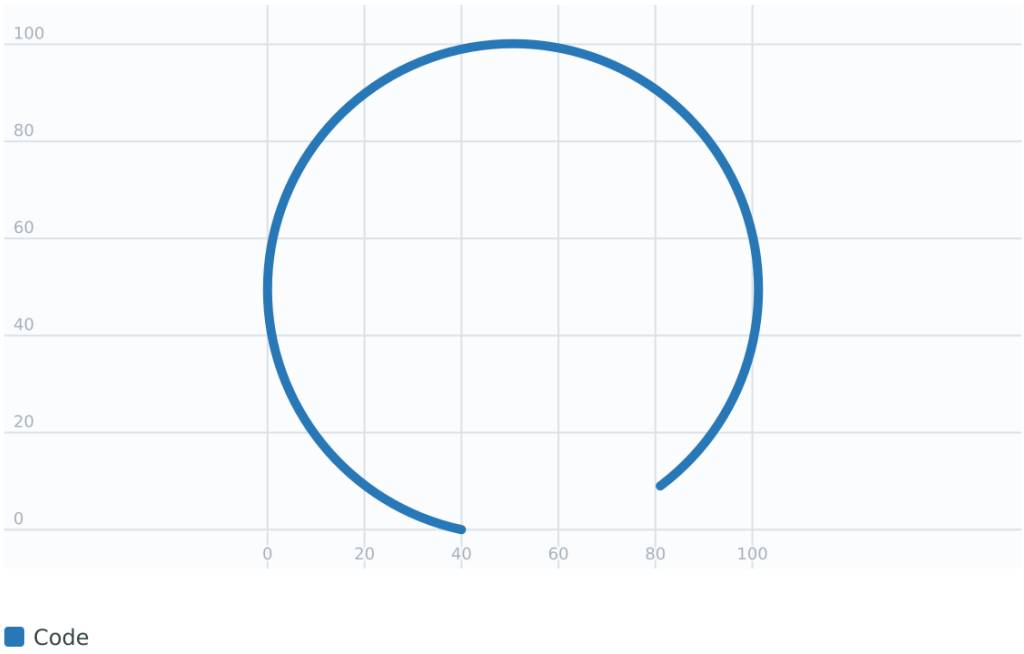
Code

```
SELECT ST_ZMin('CIRCULARSTRING(220268 150415 1,220227 150505 2,220227 150406 3)'::geometry);
```

Output

```
1
```

Figure



Siehe auch

ST_GeomFromEWKT, ST_GeomFromText, ST_XMin, ST_XMax, ST_YMin, ST_YMax, ST_ZMax, ST_MMin, ST_MMax

7.18.15 ST_XSize

ST_XSize — Returns the X size of a 2D or 3D bounding box or a geometry.

Synopsis

float **ST_XSize**(box3d aGeomorBox2DorBox3D);

Beschreibung

Returns the X size of a 2D or 3D bounding box or a geometry, computed as `ST_XMax(aGeomorBox2DorBox3D) - ST_XMin(aGeomorBox2DorBox3D)`.



Anmerkung
Obwohl diese Funktion nur für box3d definiert ist, funktioniert sie aufgrund des automatischen Castings auch für box2d- und Geometriewerte. Sie akzeptiert jedoch keine Geometrie- oder box2d-Textdarstellung, da diese nicht automatisch gecastet werden.

 Diese Funktion unterstützt 3d und lässt den Z-Index nicht fallen.

 Diese Methode unterstützt kreisförmige Strings und Kurven.

Availability: 3.7.0

Beispiele

Code

```
SELECT ST_XSize('BOX3D(1 2 3,4 5 6)');
```

Output

```
3
```

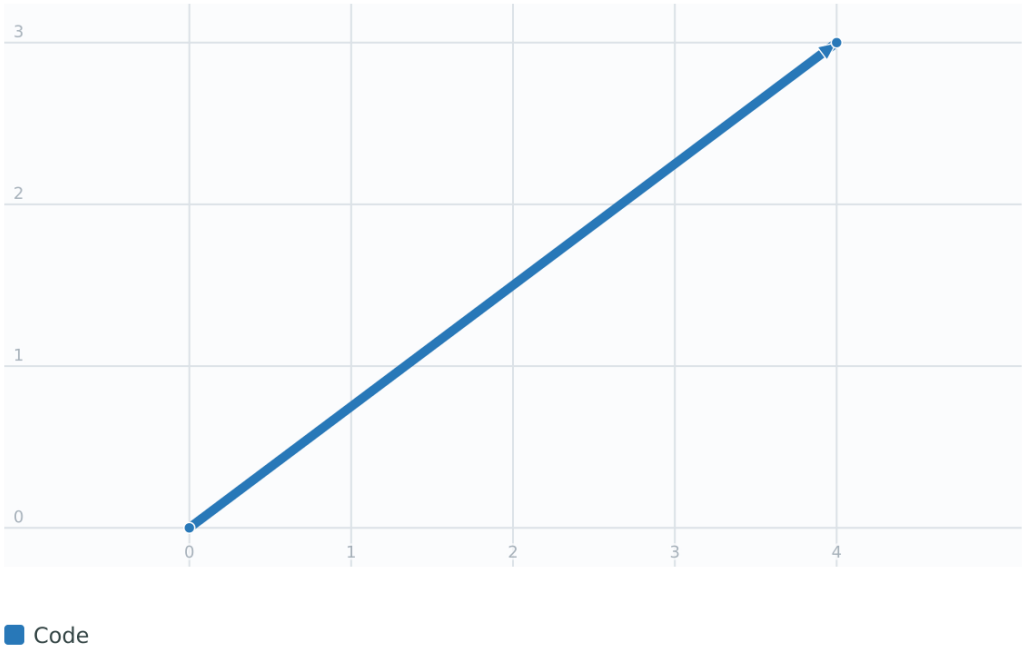
Code

```
SELECT ST_XSize('LINESTRING(1 3,5 6)'::geometry);
```

Output

```
4
```

Figure



Siehe auch

ST_XMin, ST_XMax, ST_YSize, ST_ZSize, ST_MSize

7.18.16 ST_YSize

ST_YSize — Returns the Y size of a 2D or 3D bounding box or a geometry.

Synopsis

float **ST_YSize**(box3d aGeomorBox2DorBox3D);

Beschreibung

Returns the Y size of a 2D or 3D bounding box or a geometry, computed as `ST_YMax(aGeomorBox2DorBox3D) - ST_YMin(aGeomorBox2DorBox3D)`.



Anmerkung
Obwohl diese Funktion nur für box3d definiert ist, funktioniert sie aufgrund des automatischen Castings auch für box2d- und Geometriewerte. Sie akzeptiert jedoch keine Geometrie- oder box2d-Textdarstellung, da diese nicht automatisch gecastet werden.

 Diese Funktion unterstützt 3d und lässt den Z-Index nicht fallen.

 Diese Methode unterstützt kreisförmige Strings und Kurven.

Availability: 3.7.0

Beispiele

Code

```
SELECT ST_YSize('BOX3D(1 2 3,4 5 6)');
```

Output

```
3
```

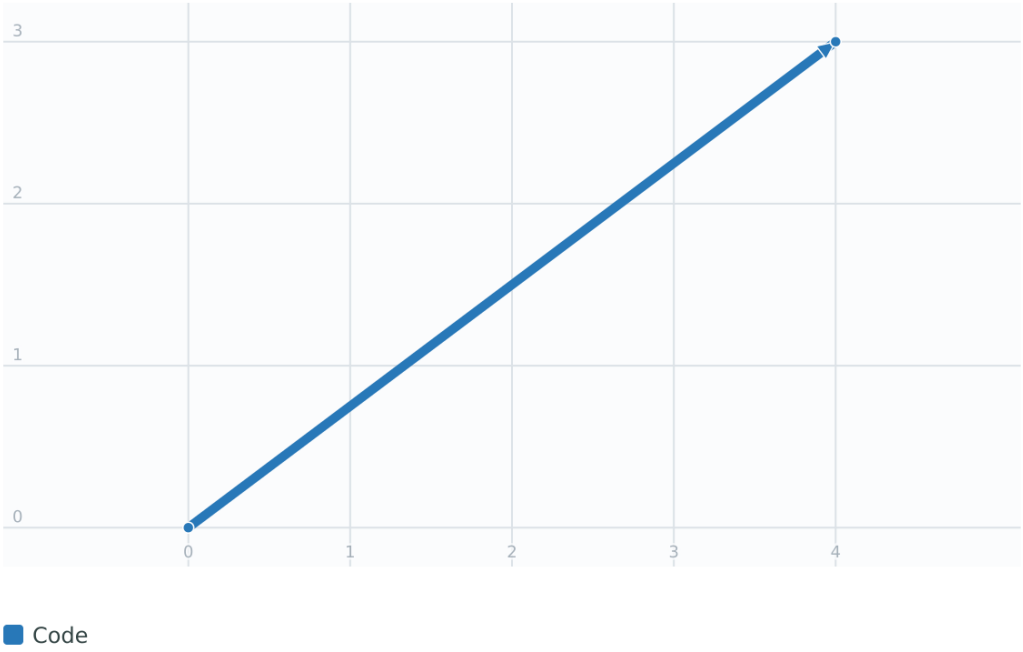
Code

```
SELECT ST_YSize('LINESTRING(1 3,5 6)'::geometry);
```

Output

```
3
```

Figure



Siehe auch

ST_YMin, ST_YMax, ST_XSize, ST_ZSize, ST_MSize

7.18.17 ST_ZSize

ST_ZSize — Returns the Z size of a 2D or 3D bounding box or a geometry.

Synopsis

float **ST_ZSize**(box3d aGeomorBox2DorBox3D);

Beschreibung

Returns the Z size of a 2D or 3D bounding box or a geometry, computed as `ST_ZMax(aGeomorBox2DorBox3D) - ST_ZMin(aGeomorBox2DorBox3D)`.



Anmerkung
Obwohl diese Funktion nur für box3d definiert ist, funktioniert sie aufgrund des automatischen Castings auch für box2d- und Geometriewerte. Sie akzeptiert jedoch keine Geometrie- oder box2d-Textdarstellung, da diese nicht automatisch gecastet werden.

 Diese Funktion unterstützt 3d und lässt den Z-Index nicht fallen.

 Diese Methode unterstützt kreisförmige Strings und Kurven.

Availability: 3.7.0

Beispiele

Code

```
SELECT ST_ZSize('BOX3D(1 2 3,4 5 6)');
```

Output

```
3
```

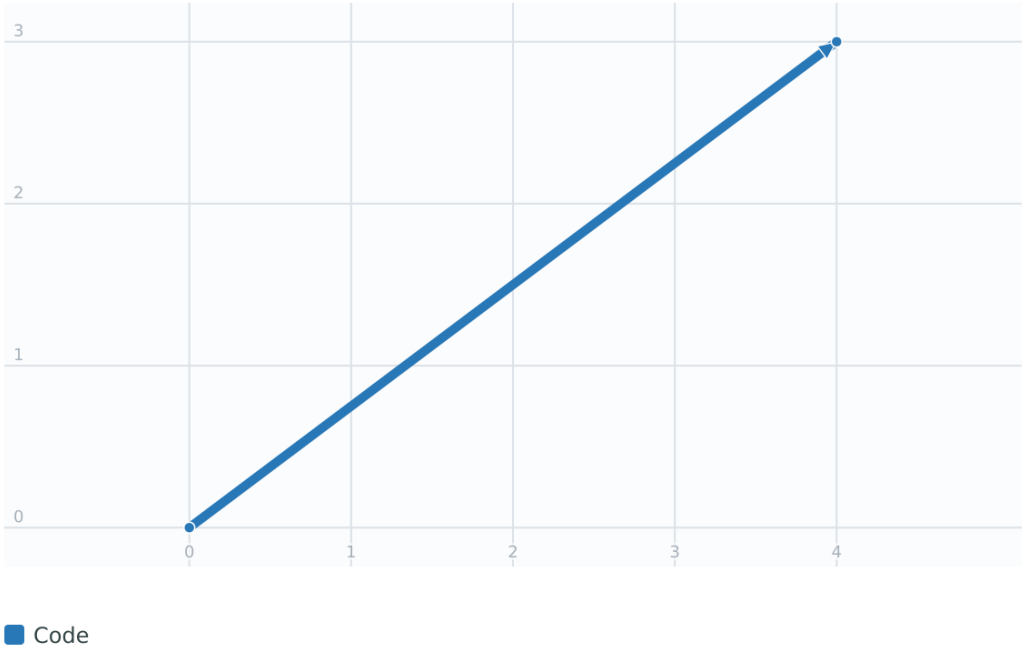
Code

```
SELECT ST_ZSize('LINESTRING(1 3 4,5 6 7)'::geometry);
```

Output

```
3
```

Figure



Siehe auch

ST_ZMin, ST_ZMax, ST_XSize, ST_YSize, ST_MSize

7.18.18 ST_MMin

ST_MMin — Returns the M minima of a geometry.

Synopsis

float **ST_MMin**(geometry geom);

Beschreibung

Returns the M minima of a geometry, or null if the geometry lacks M values.

Diese Funktion unterstützt 3d und lässt den Z-Index nicht fallen.

Diese Methode unterstützt kreisförmige Strings und Kurven.

Beispiele

Code

```
SELECT ST_MMin('POINT M (1 2 3)');
```

Output

```
3
```

Siehe auch

ST_XMin, ST_XMax, ST_YMin, ST_YMax, ST_ZMin, ST_ZMax, ST_MMax, ST_MSize

7.18.19 ST_MMax

ST_MMax — Returns the M maxima of a geometry.

Synopsis

float **ST_MMax**(geometry geom);

Beschreibung

Returns the M maxima of a geometry, or null if the geometry lacks M values.

Diese Funktion unterstützt 3d und lässt den Z-Index nicht fallen.

Diese Methode unterstützt kreisförmige Strings und Kurven.

Beispiele

Code

```
SELECT ST_MMax('POINT M (1 2 3)');
```

Output

```
3
```

Siehe auch

ST_XMin, ST_XMax, ST_YMin, ST_YMax, ST_ZMin, ST_ZMax, ST_MMin, ST_MSize

7.18.20 ST_MSize

ST_MSize — Returns the M size of a geometry.

Synopsis

float **ST_MSize**(geometry geom);

Beschreibung

Returns the M size of a geometry, computed as `ST_MMax(geom) - ST_MMin(geom)`, or null if the geometry lacks M values.

 Diese Funktion unterstützt 3d und lässt den Z-Index nicht fallen.

 Diese Methode unterstützt kreisförmige Strings und Kurven.

Availability: 3.7.0

Beispiele

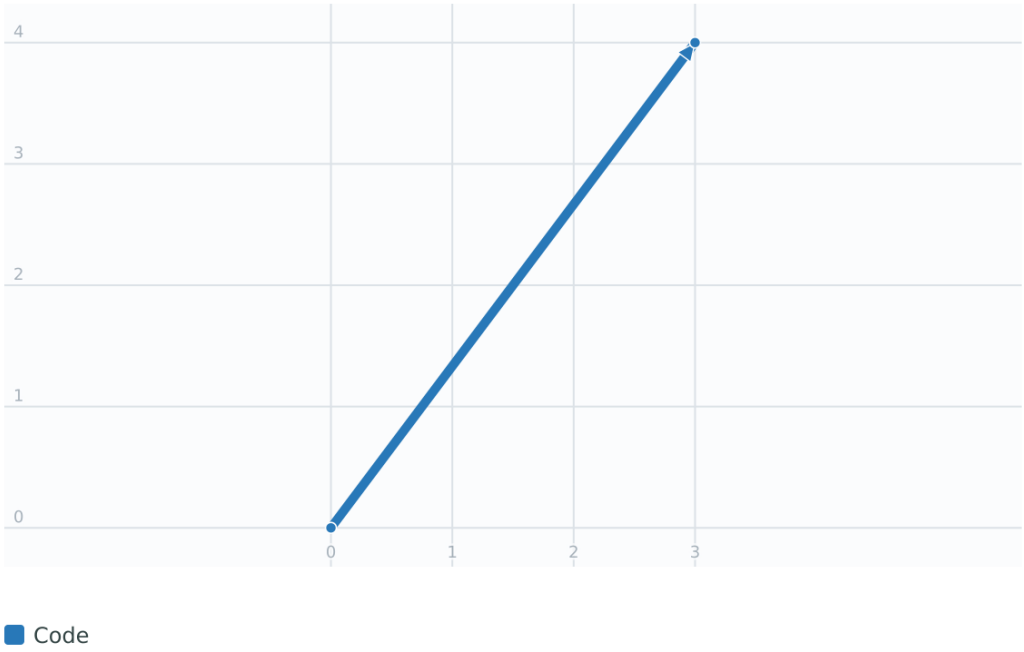
Code

```
SELECT ST_MSize('LINESTRING M (1 2 3,4 6 8)');
```

Output

```
5
```

Figure



Siehe auch

ST_MMin, ST_MMax, ST_XSize, ST_YSize, ST_ZSize

7.19 Kilometrierung

7.19.1 ST_LineInterpolatePoint

ST_LineInterpolatePoint — Gibt einen oder mehrere, entlang einer Linie interpolierte Punkte zurück.

Synopsis

geometry **ST_LineInterpolatePoint**(geometry a_linestring, float8 a_fraction);
geography **ST_LineInterpolatePoint**(geography a_linestring, float8 a_fraction, boolean use_spheroid = true);

Beschreibung

Fügt einen Punkt entlang einer Linie ein. Der erste Parameter muss einen Linienzug beschreiben. Der zweite Parameter, in Float8-Darstellung mit den Werten von 0 bis 1, gibt jenen Bruchteil der Gesamtlänge des Linienzuges an, wo der Punkt liegen soll.

Siehe ST_LineLocatePoint um die nächstliegende Linie zu einem Punkt zu berechnen.



Anmerkung
Diese Funktion berechnet Punkte in 2D und interpoliert dann Werte für Z und M, während ST_3DLineInterpolatePoint Punkte in 3D berechnet und nur den M-Wert interpoliert.



Anmerkung
Ab Version 1.1.1 interpoliert diese Funktion auch M- und Z-Werte (falls vorhanden), während frühere Versionen diese Werte auf 0.0 setzten.

Verfügbarkeit: 0.8.2, Z und M Unterstützung wurde mit 1.1.1 hinzugefügt

Änderung: 2.1.0. Bis zu 2.0.x wurde diese Funktion mit ST_Line_Interpolate_Point bezeichnet.

 Diese Funktion unterstützt 3d und lässt den Z-Index nicht fallen.

Beispiele

This example returns the point 20 percent along a line.

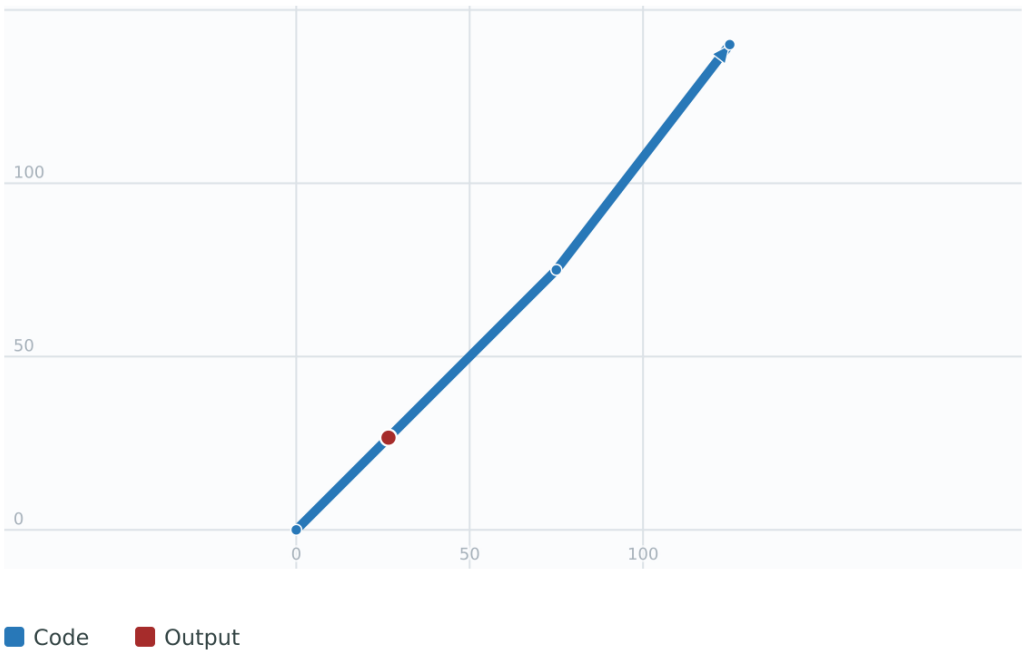
Code

```
SELECT ST_LineInterpolatePoint(
'LINESTRING(25 50,100 125,150 190)',
0.2);
```

Output

```
POINT(51.5974135047432 76.5974135047432)
```

Figure



Gibt einen oder mehrere, entlang einer Linie interpolierte Punkte zurück.

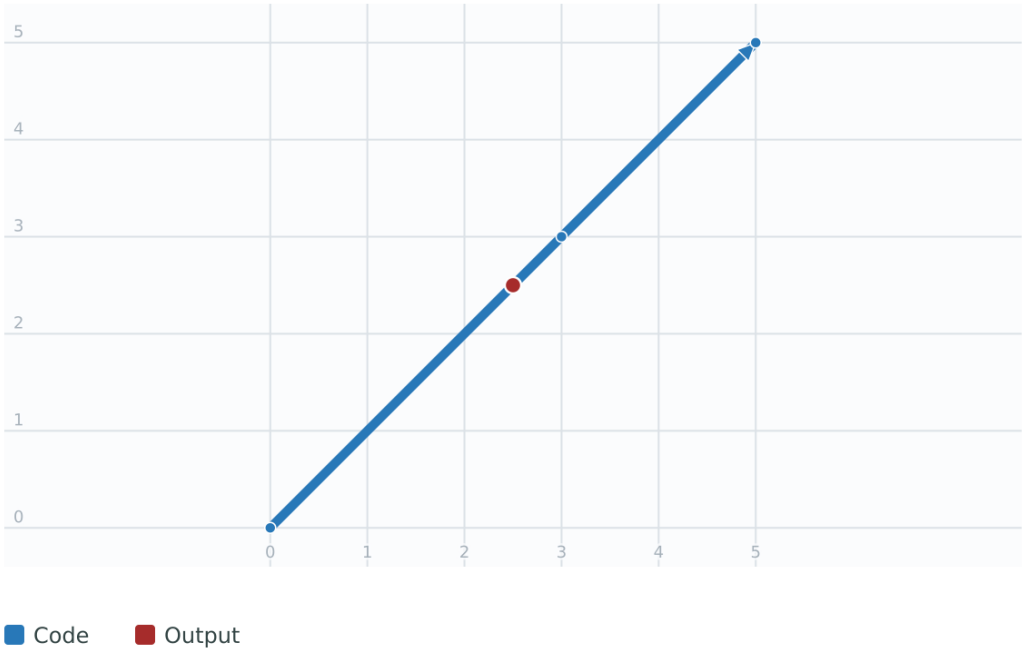
Code

```
SELECT ST_LineInterpolatePoint(
'LINESTRING(1 2 3,4 5 6,6 7 8)',
0.5);
```

Output

```
POINT(3.5 4.5 5.5)
```

Figure



Der Punkt auf einer Linie, der einem Punkt am nächsten liegt:

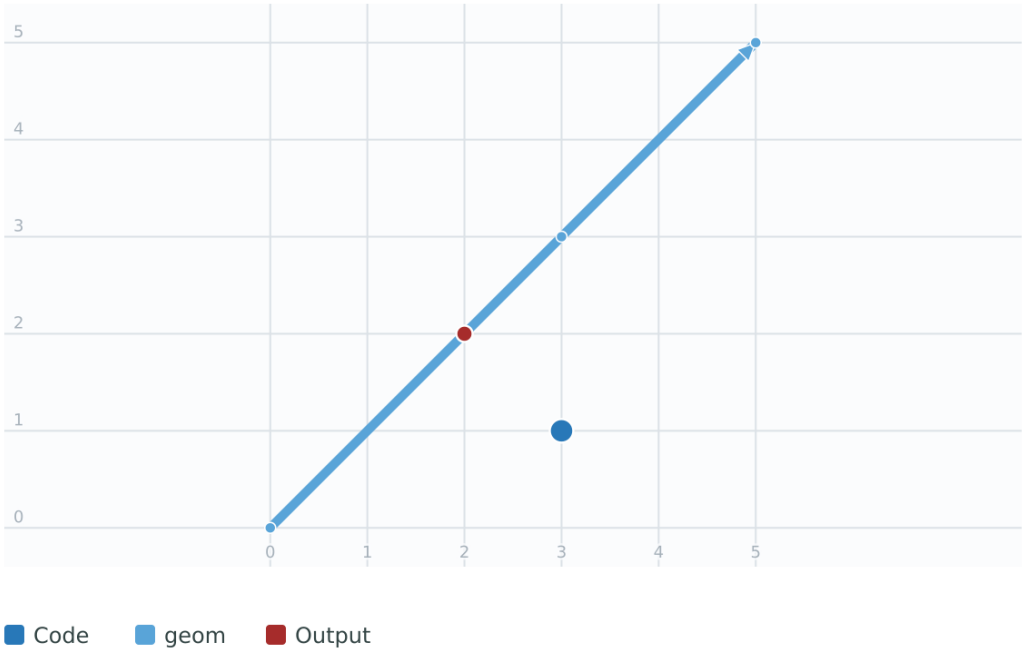
Code

```
SELECT ST_LineInterpolatePoint(
    line.geom,
    ST_LineLocatePoint(line.geom, 'POINT(4 3)'))
FROM (SELECT 'LINESTRING(1 2,4 5,6 7)'::geometry As geom) AS line;
```

Output

```
POINT(3 4)
```

Figure

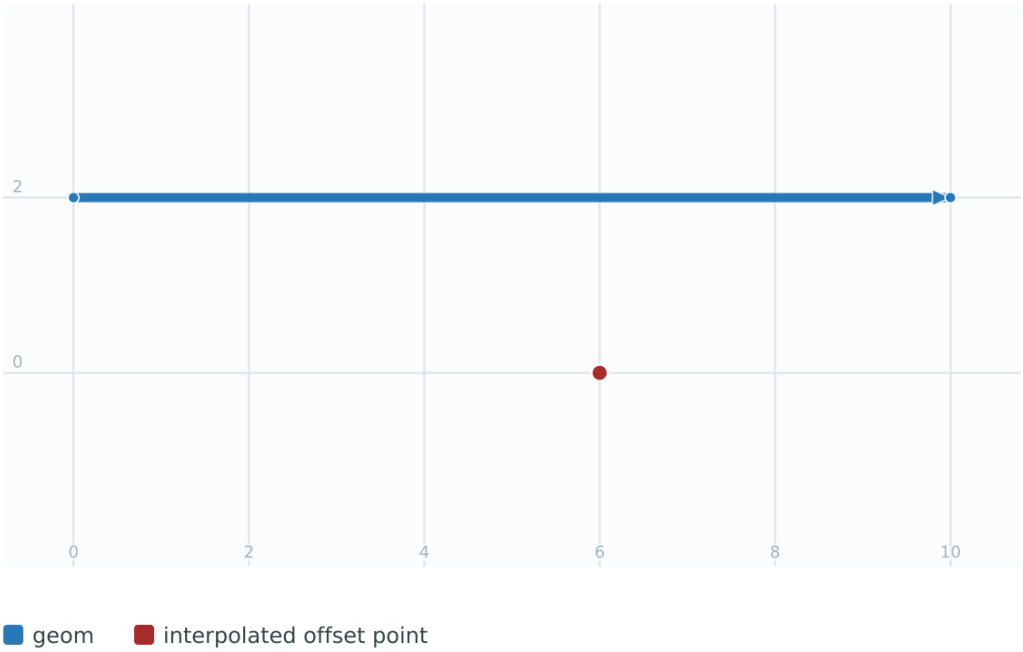


An interpolated point can be offset to the left or right of a line by using ST_OffsetCurve before interpolation. This is useful for locating an address point beside a street centerline.

Code

```
WITH street AS (
    SELECT 'LINESTRING(0 0, 10 0)'::geometry AS geom,
           100 AS from_num,
           200 AS to_num
)
SELECT
    ST_LineInterpolatePoint(
        ST_OffsetCurve(geom, -2),
        (160 - from_num)::float8 / (to_num - from_num)
    ) AS interpolated_offset_point
FROM street;
```

Figure



Siehe auch

ST_LineInterpolatePoints, ST_3DLineInterpolatePoint, ST_LineLocatePoint, ST_OffsetCurve

7.19.2 ST_3DLineInterpolatePoint

ST_3DLineInterpolatePoint — Gibt einen oder mehrere, entlang einer Linie interpolierte Punkte zurück.

Synopsis

geometry **ST_3DLineInterpolatePoint**(geometry a_linestring, float8 a_fraction);

Beschreibung

Fügt einen Punkt entlang einer Linie ein. Der erste Parameter muss einen Linienzug beschreiben. Der zweite Parameter, in Float8-Darstellung mit den Werten von 0 bis 1, gibt jenen Bruchteil der Gesamtlänge des Linienzuges an, wo der Punkt liegen soll.



Anmerkung
ST_LineInterpolatePoint berechnet Punkte in 2D und interpoliert dann die Werte für Z und M, während diese Funktion Punkte in 3D berechnet und nur den M-Wert interpoliert.

Verfügbarkeit: 2.1.0


Diese Funktion unterstützt 3d und lässt den Z-Index nicht fallen.

Beispiele

Gibt einen oder mehrere, entlang einer Linie interpolierte Punkte zurück.

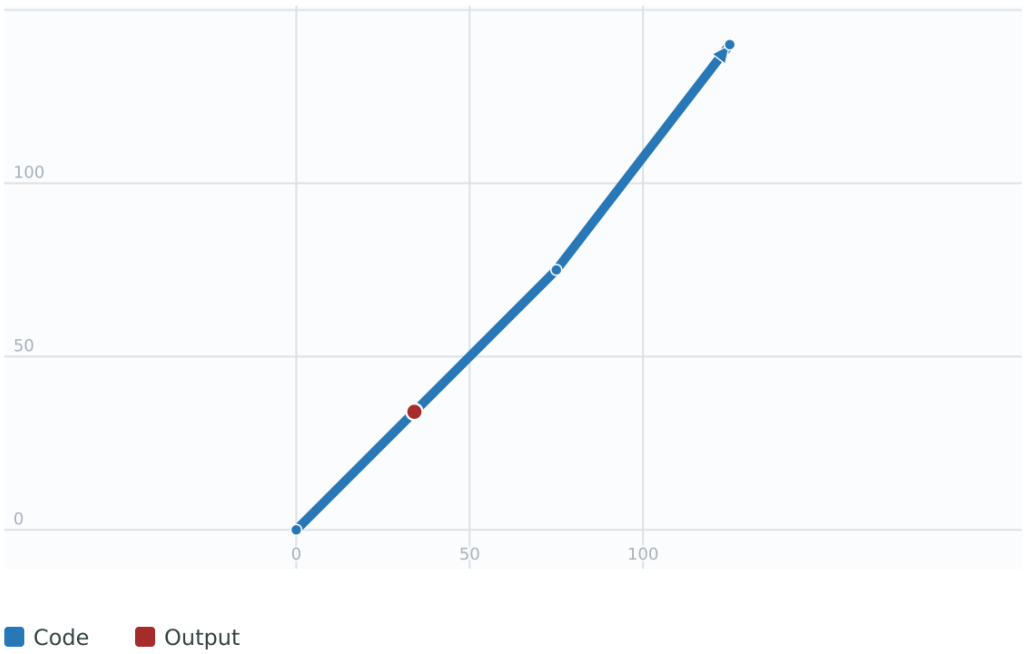
Code

```
SELECT ST_3DLineInterpolatePoint(
'LINESTRING(25 50 70,100 125 90,150 190 200)',
0.20);
```

Output

```
POINT Z (59.06758929108215 84.06758929108216 79.0846904776219)
```

Figure



Siehe auch

ST_LineInterpolatePoint, ST_LineInterpolatePoint, ST_LineMerge

7.19.3 ST_LineInterpolatePoints

ST_LineInterpolatePoints — Gibt einen oder mehrere, entlang einer Linie interpolierte Punkte zurück.

Synopsis

geometry **ST_LineInterpolatePoints**(geometry a_linestring, float8 a_fraction, boolean repeat);
geography **ST_LineInterpolatePoints**(geography a_linestring, float8 a_fraction, boolean use_spheroid = true, boolean repeat = true);

Beschreibung

Gibt einen oder mehrere Punkte zurück, die entlang einer Linie in einem Bruchteilintervall interpoliert wurden. Das erste Argument muss ein LINESTRING sein. Das zweite Argument ist ein float8 zwischen 0 und 1, der den Abstand zwischen den Punkten als Bruchteil der Linienlänge angibt. Wenn das dritte Argument false ist, wird höchstens ein Punkt konstruiert (was ST_LineInterpolatePoint entspricht).

Wenn das Ergebnis null oder einen Punkt hat, wird es als POINT zurückgegeben. Wenn es zwei oder mehr Punkte hat, wird es als MULTIPOINT zurückgegeben.

Verfügbarkeit: 2.5.0

Diese Funktion unterstützt 3d und lässt den Z-Index nicht fallen.

Diese Funktion unterstützt M-Koordinaten.

Beispiele

This example returns points every 20 percent along a two-dimensional line.

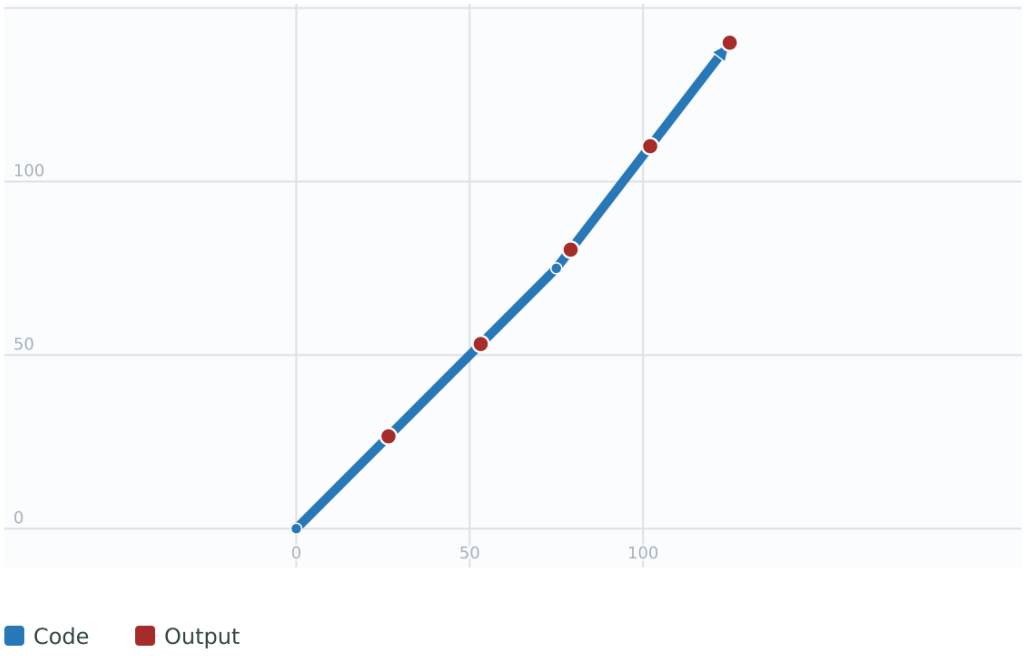
Code

```
SELECT ST_LineInterpolatePoints(
'LINESTRING(25 50,100 125,150 190)',
0.20);
```

Output

```
MULTIPOINT((51.5974135047432 76.5974135047432),(78.1948270094864 103.194827009486),(104.132163186446 130.37181214238) ↩
    ,(127.066081593223 160.18590607119),(150 190))
```

Figure



Siehe auch

ST_LineInterpolatePointDiese Funktion ändert den Namen einer bestehenden TopoGeometry-Spalte und stellt sicher, dass die Metadateninformationen entsprechend aktualisiert werden.

7.19.4 ST_LineLocatePoint

ST_LineLocatePoint — Liefert die gebrochene Position des Punktes auf einer Linie, der einem Punkt am nächsten liegt.

Synopsis

float8 **ST_LineLocatePoint**(geometry a_linestring, geometry a_point);
float8 **ST_LineLocatePoint**(geography a_linestring, geography a_point, boolean use_spheroid = true);

Beschreibung

Gibt eine Gleitpunktzahl zwischen 0 und 1 zurück, welche die Lage des Punktes auf einer Linie angibt, der zu einem gegebenen Punkt am nächsten liegt. Die Lage wird als Anteil an der Gesamtlänge der 2D Linie angegeben.

Sie können die zurückgegebene Lage nutzen, um einen Punkt (ST_LineInterpolatePoint) oder eine Teilzeichenfolge (ST_LineSubstring) zu extrahieren.

Nützlich, um die Hausnummern von Adressen anzunähern

Verfügbarkeit: 1.1.0

Änderung: 2.1.0. Bis zu 2.0.x wurde diese Funktion mit ST_Line_Locate_Point bezeichnet.

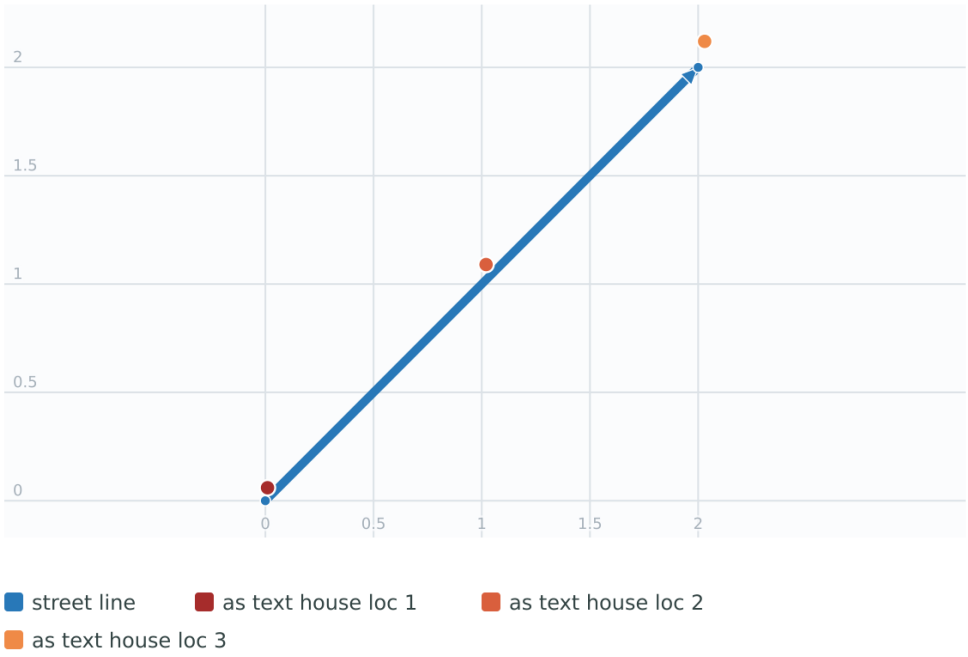
Beispiele

This example approximates the street number of a house by locating the closest point on a street line. The subquery generates sample house and street data, and `ST_DWithin` excludes houses that are too far from the street.

Code

```
SELECT house_loc As as_text_house_loc,
    start_house_num +
        CAST( (end_house_num - start_house_num)
            * ST_LineLocatePoint(street_line, house_loc) As integer) As house_num
FROM
(SELECT 'LINESTRING(1 2,3 4)'::geometry As street_line,
    ST_Point(x*1.01, y*1.03) As house_loc, 10 As start_house_num,
        20 As end_house_num
FROM generate_series(1, 3) x CROSS JOIN generate_series(2, 4) As y)
As foo
WHERE ST_DWithin(street_line, house_loc, 0.2);
```

Figure



This example finds the closest point on a line to a point or other geometry.

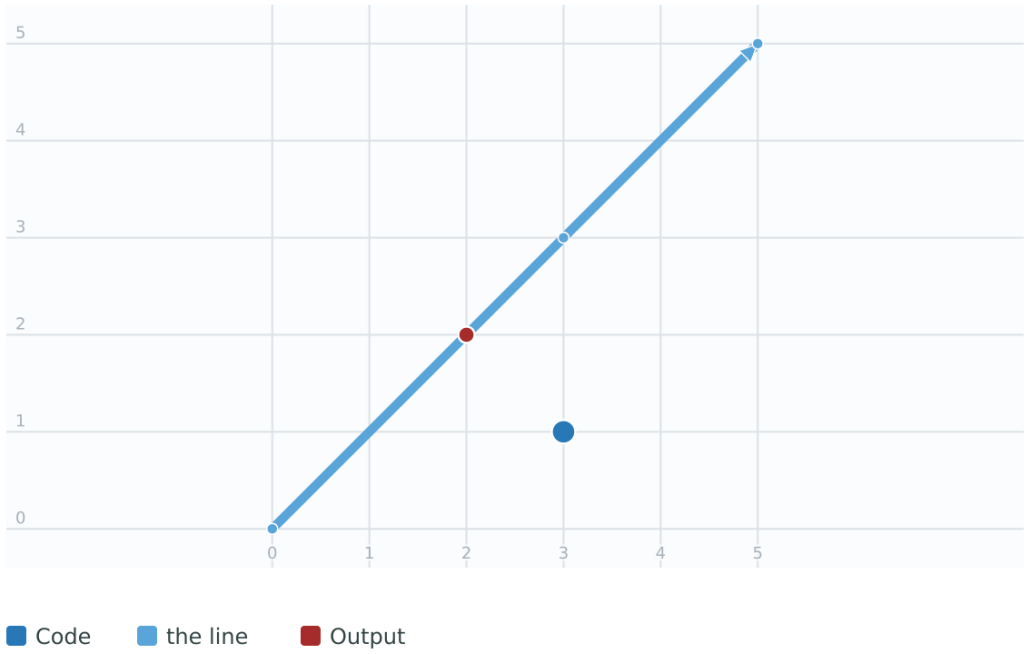
Code

```
SELECT ST_LineInterpolatePoint(
    foo.the_line,
    ST_LineLocatePoint(foo.the_line, 'POINT(4 3)'::geometry))
FROM (SELECT 'LINESTRING(1 2,4 5,6 7)'::geometry As the_line) As foo;
```

Output

```
POINT(3 4)
```

Figure



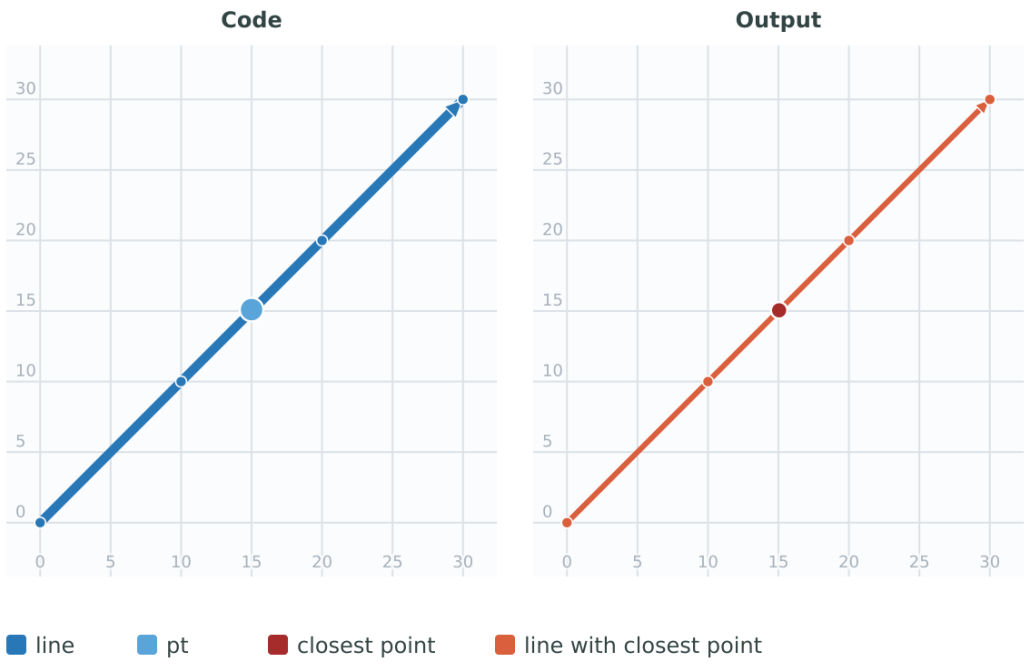
This example locates the closest segment and point on a LineString.

Code

```
WITH data AS (
  SELECT
    'LINESTRING (0 0, 10 10, 20 20, 30 30)'::geometry AS line,
    'POINT(15 15.1)'::geometry AS pt
),
segments AS (
  SELECT
    s.i,
    d.line,
    d.pt,
    ST_MakeLine(ST_PointN(d.line, s.i), ST_PointN(d.line, s.i + 1)) AS segment
  FROM data AS d
  CROSS JOIN LATERAL generate_series(1, ST_NumPoints(d.line) - 1) AS s(i)
)
SELECT
  i AS segment_number,
  round(ST_Distance(segment, pt)::numeric, 3) AS distance,
  ST_ClosestPoint(segment, pt) AS closest_point,
  ST_AddPoint(line, ST_ClosestPoint(segment, pt), i) AS line_with_closest_point
```

```
FROM segments
ORDER BY ST_Distance(segment, pt)
LIMIT 1;
```

Figure



Siehe auch

ST_AddPoint, ST_ClosestPoint, ST_DWithin, ST_Length2D, ST_LineInterpolatePoint, ST_LineSubstring

7.19.5 ST_LineSubstring

ST_LineSubstring — Gibt den Teil einer Linie zwischen zwei gebrochenen Stellen zurück.

Synopsis

geometry **ST_LineSubstring**(geometry a_linestring, float8 startfraction, float8 endfraction);
geography **ST_LineSubstring**(geography a_linestring, float8 startfraction, float8 endfraction);

Beschreibung

Computes the line which is the section of the input line starting and ending at the given fractional locations. The first argument must be a LINESTRING. The second and third arguments are values in the range [0, 1] representing the start and end locations. For geometry inputs, the fractions are measured in 2D line length. The Z and M values are interpolated for added endpoints if present.

Gleichbedeutend mit ST_LineInterpolatePoint, wenn Anfangswert und Endwert ident sind.



Anmerkung
Dies funktioniert nur bei LINESTRINGs. Zur Verwendung bei zusammenhängenden MULTILINESTRINGs müssen diese zunächst mit ST_LineMerge verbunden werden.



Anmerkung
Ab Version 1.1.1 interpoliert diese Funktion auch M- und Z-Werte (falls vorhanden), während frühere Versionen unbestimmte Werte setzten.

Verbessert: 3.4.0 - Unterstützung für Geographie wurde eingeführt.

Änderung: 2.1.0. Bis zu 2.0.x wurde diese Funktion mit ST_Line_Substring bezeichnet.

Verfügbarkeit: 1.1.0, mit 1.1.1 wurde die Unterstützung für Z und M hinzugefügt

 Diese Funktion unterstützt 3d und lässt den Z-Index nicht fallen.

Beispiele

Extract the middle third of a LineString, from 0.333 to 0.666.

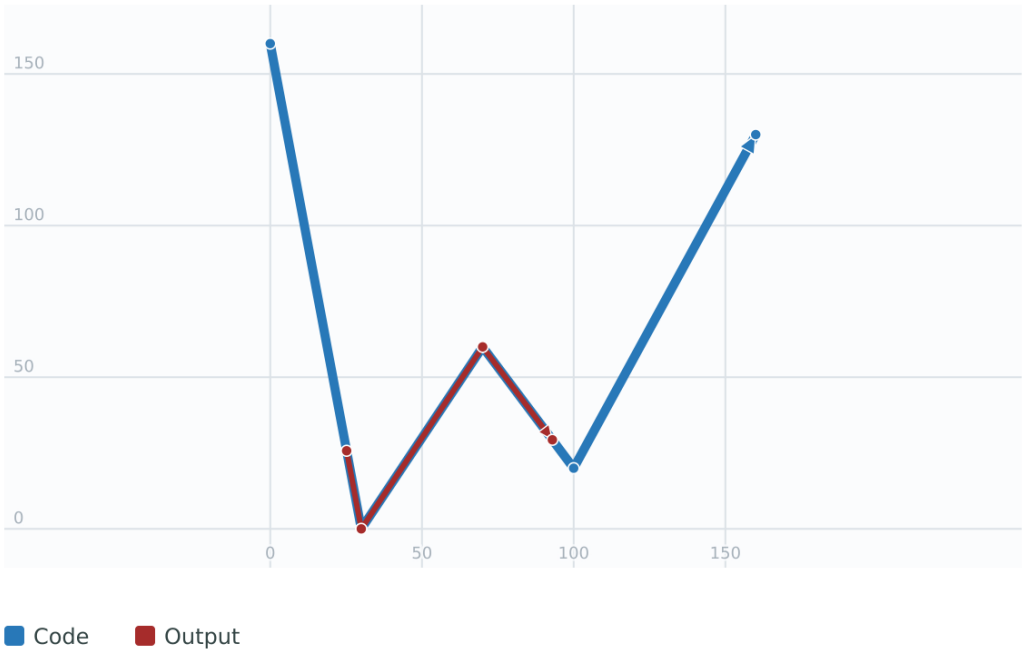
Code

```
SELECT ST_LineSubstring(
'LINESTRING (20 180,50 20,90 80,120 40,180 150)',
0.333,
0.666);
```

Output

```
LINESTRING (45.17311810399485 45.74337011202746, 50 20, 90 80, 112.97593050157862 49.36542599789519)
```

Figure



Wenn Anfangs- und Endpunkt gleich sind, ist das Ergebnis ein PUNKT.

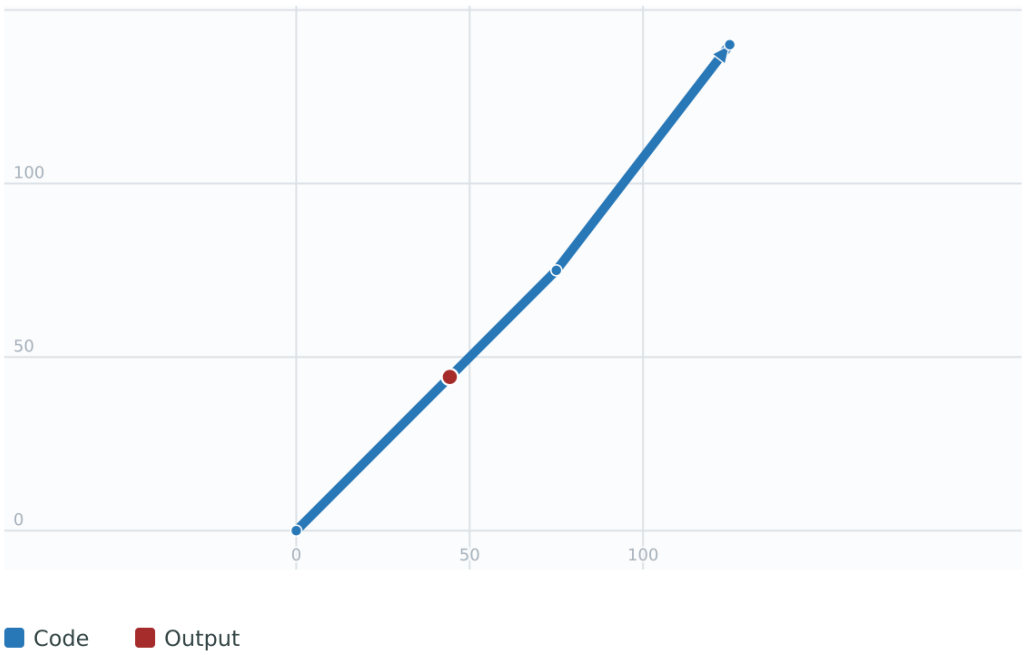
Code

```
SELECT ST_LineSubstring(
'LINESTRING(25 50,100 125,150 190)',
0.333,
0.333);
```

Output

```
POINT(69.28469348539744 94.28469348539744)
```

Figure



Eine Abfrage, um einen LineString in Abschnitte der Länge 100 oder kürzer zu schneiden. Sie verwendet `generate_series()` mit einem CROSS JOIN LATERAL, um das Äquivalent einer FOR-Schleife zu erzeugen.

The final generated step is filtered out when the line length is an exact multiple of the requested section length.

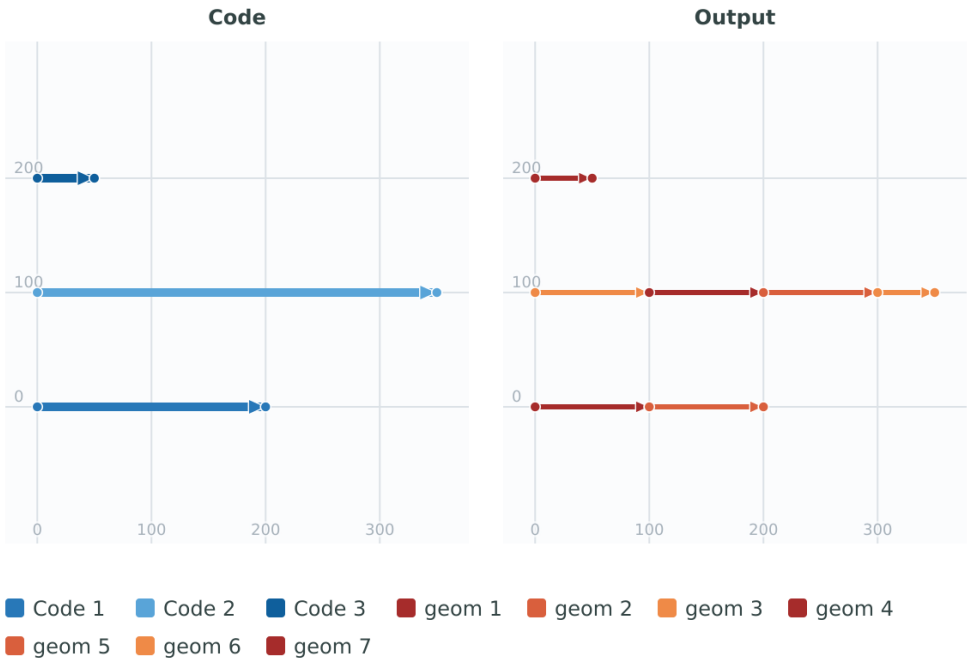
Code

```
WITH data(id, geom) AS (VALUES
        ( 'A', 'LINESTRING( 0 0, 200 0)'::geometry ),
        ( 'B', 'LINESTRING( 0 100, 350 100)'::geometry ),
        ( 'C', 'LINESTRING( 0 200, 50 200)'::geometry )
    )
SELECT id, i,
       ST_LineSubstring(geom, startfrac, LEAST( endfrac, 1 )) AS geom
FROM (
    SELECT id, geom, ST_Length(geom) len, 100 sublen FROM data
    ) AS d
CROSS JOIN LATERAL (
    SELECT i, (sublen * i) / len AS startfrac,
              (sublen * (i+1)) / len AS endfrac
    FROM generate_series(0, floor( len / sublen )::integer ) AS t(i)
    WHERE (sublen * i) / len <
> 1.0
    ) AS d2;
```

Output

```
id | i |            geom
----+---+-----------------------------
 A  | 0 | LINESTRING(0 0,100 0)
 A  | 1 | LINESTRING(100 0,200 0)
 B  | 0 | LINESTRING(0 100,100 100)
 B  | 1 | LINESTRING(100 100,200 100)
 B  | 2 | LINESTRING(200 100,300 100)
 B  | 3 | LINESTRING(300 100,350 100)
 C  | 0 | LINESTRING(0 200,50 200)
```

Figure



Split a LineString by points by locating the points along the line, adding the 0 and 1 endpoints, and extracting substrings between adjacent locations.

Code

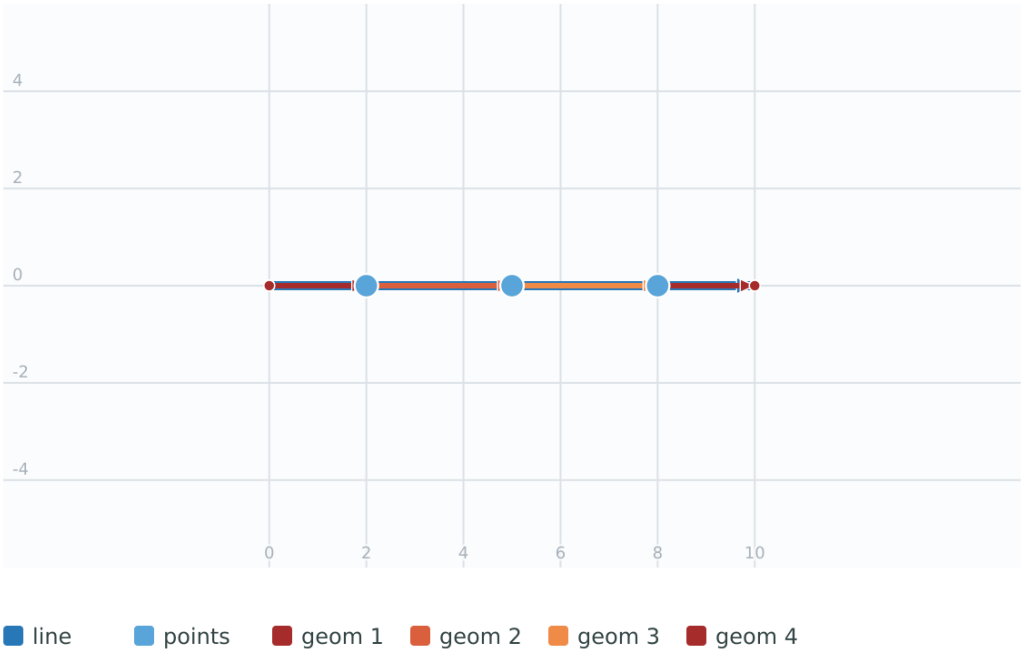
```
WITH data AS (
  SELECT 'LINESTRING(0 0, 10 0)'::geometry AS line,
         'MULTIPOINT((2 0),(5 0),(8 0))'::geometry AS points
),
fractions AS (
  SELECT 0.0 AS fraction
  UNION ALL
  SELECT 1.0
  UNION ALL
  SELECT ST_LineLocatePoint(line, (dp).geom)
  FROM data
  CROSS JOIN LATERAL ST_Dump(points) AS dp
),
ordered AS (
  SELECT DISTINCT fraction
  FROM fractions
),
segments AS (
  SELECT fraction AS start_fraction,
         lead(fraction) OVER (ORDER BY fraction) AS end_fraction
  FROM ordered
)
```

```
SELECT row_number() OVER (ORDER BY start_fraction) AS segment_id,
       ST_LineSubstring(line, start_fraction, end_fraction) AS geom
FROM data
CROSS JOIN segments
WHERE end_fraction IS NOT NULL
  AND start_fraction <
> end_fraction
ORDER BY start_fraction;
```

Output

```
segment_id |         geom
-----------+----------------------
         1 | LINESTRING(0 0,2 0)
         2 | LINESTRING(2 0,5 0)
         3 | LINESTRING(5 0,8 0)
         4 | LINESTRING(8 0,10 0)
```

Figure



Geografische Umsetzungsmaßnahmen entlang eines Sphäroids, Geometrie entlang einer Linie

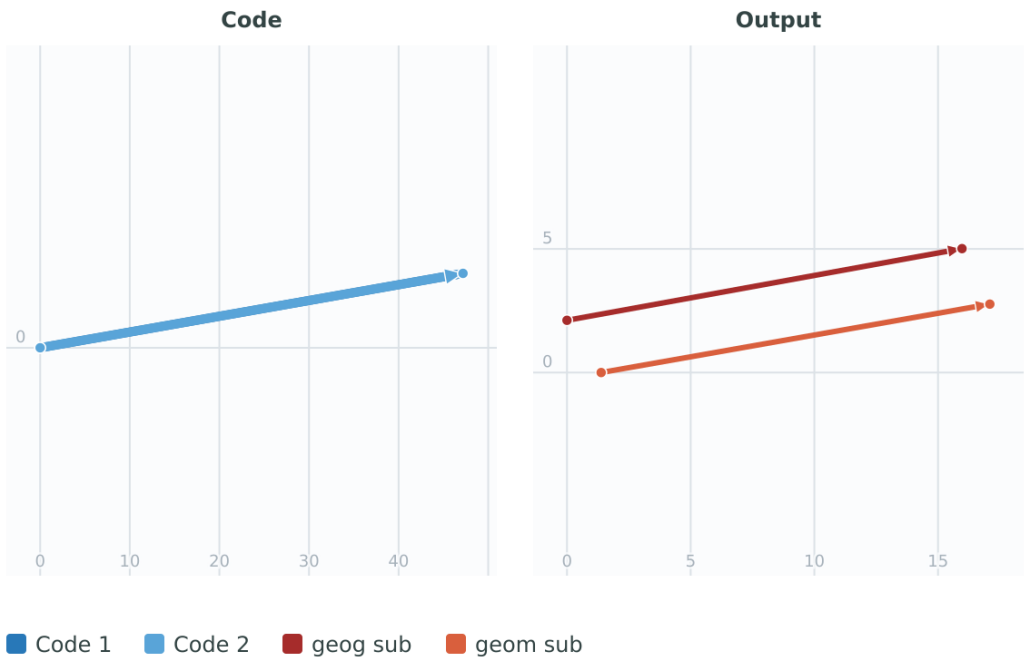
Code

```
SELECT ST_LineSubstring(
'LINESTRING(-118.2436 34.0522,-71.0570 42.3611)'::geography,
0.333,
0.666) AS geog_sub,
ST_LineSubstring(
    'LINESTRING(-118.2436 34.0522,-71.0570 42.3611)'::geometry,
    0.333,
    0.666) AS geom_sub;
```

Output

```
-[ RECORD 1 ]----------
geog_sub | LINESTRING(-103.911641 38.931128,-87.941787 41.831072)
geom_sub | LINESTRING(-102.530462 36.819064,-86.817324 39.585927)
```

Figure



For geometry inputs, the fractional locations are based on the 2D length of the line, even when the input has Z values. The resulting Z values are interpolated along the selected 2D location.

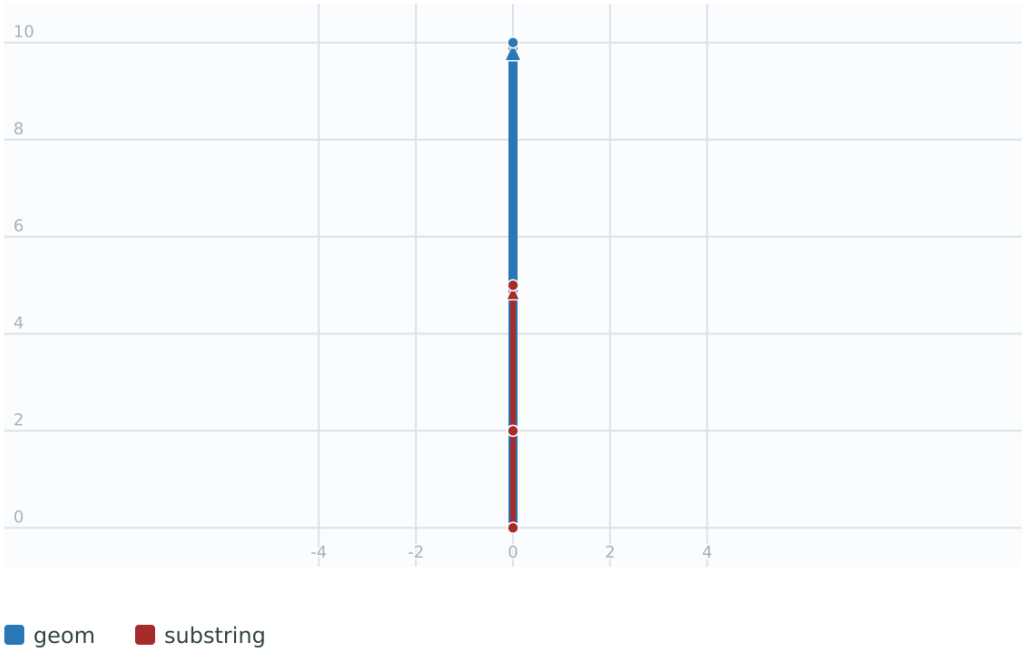
Code

```
WITH data AS (
  SELECT 'LINESTRING Z (0 0 0,0 2 5,0 10 10)'::geometry AS geom
)
SELECT ST_Length(geom) AS length_2d,
       ST_3DLength(geom) AS length_3d,
       ST_LineSubstring(geom, 0, 0.5) AS substring
FROM data;
```

Output

```
length_2d |    length_3d     |              substring
----------+------------------+-------------------------------------
        10 | 14.819145939191106 | LINESTRING Z (0 0 0,0 2 5,0 5 6.875)
```

Figure



Siehe auch

ST_Length, ST_LineExtend, ST_LineInterpolatePoint, ST_LineLocatePoint, ST_LineMerge, ST_Split

7.19.6 ST_LocateAlong

ST_LocateAlong — Gibt die Punkte auf einer Geometrie zurück, die einem Messwert entsprechen.

Synopsis

geometry **ST_LocateAlong**(geometry geom_with_measure, float8 measure, float8 offset = 0);

Beschreibung

Gibt die Position(en) entlang einer gemessenen Geometrie zurück, die die angegebenen Messwerte aufweisen. Das Ergebnis ist ein Point oder MultiPoint. Polygonale Eingaben werden nicht unterstützt.

Wenn `offset` angegeben ist, wird das Ergebnis um den angegebenen Abstand nach links oder rechts von der Eingabezeile verschoben. Ein positiver Versatz geht nach links, ein negativer nach rechts.



Anmerkung
Verwenden Sie diese Funktion nur für lineare Geometrien mit einer M-Komponente

Die Semantik ist in der Norm *ISO/IEC 13249-3 SQL/MM Spatial* festgelegt.

Verfügbarkeit: 1.1.0 über die alte Bezeichnung ST_Locate_Along_Measure.

Änderung: 2.0.0 In Vorgängerversionen als ST_Locate_Along_Measure bezeichet. Der alte Name ist überholt und wird in der Zukunft entfernt ist aber noch verfügbar.

Diese Funktion unterstützt M-Koordinaten.

Diese Methode setzt die SQL/MM-Spezifikation um. SQL-MM IEC 13249-3: 5.1.13

Beispiele

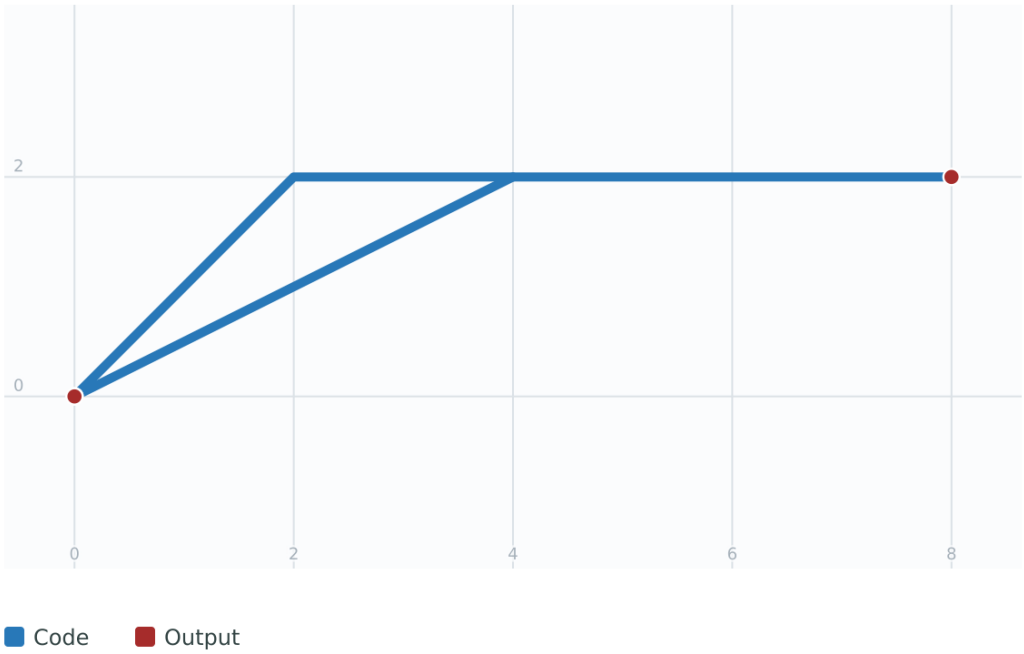
Code

```
SELECT ST_LocateAlong(
'MULTILINESTRINGM((1 2 3, 3 4 2, 9 4 3),(1 2 3, 5 4 5))'::geometry,
3);
```

Output

```
MULTIPOINT M ((1 2 3),(9 4 3),(1 2 3))
```

Figure



Siehe auch

ST_LocateBetween, ST_LocateBetweenElevations, ST_InterpolatePoint

7.19.7 ST_LocateBetween

ST_LocateBetween — Gibt die Teile einer Geometrie zurück, die einem Messbereich entsprechen.

Synopsis

geometry **ST_LocateBetween**(geometry geom, float8 measure_start, float8 measure_end, float8 offset = 0);

Beschreibung

Gibt eine abgeleitete Sammelgeometrie mit jenen Elementen zurück, die in dem gegebenen Kilometrierungsintervall liegen; das Intervall ist unbeschränkt. Polygonale Elemente werden nicht untersützt.

Wenn ein Versatz angegeben ist, werden die Resultierenden um diese Anzahl an Einheiten nach links oder rechts von der gegebenen Linie versetzt. Ein positiver Versatz geschieht nach links, ein negativer nach rechts.

Das Beschneiden eines nicht konvexen POLYGONs kann zu einer ungültigen Geometrie führen.

Die Semantik ist in der Norm *ISO/IEC 13249-3 SQL/MM Spatial* festgelegt.

Verfügbarkeit: 1.1.0 über die alte Bezeichnung ST_Locate_Between_Measures.

Änderung: 2.0.0 In Vorgängerversionen als ST_Locate_Along_Measure bezeichet. Der alte Name ist überholt und wird in der Zukunft entfernt ist aber noch verfügbar.

Verbessert: 3.0.0 - Unterstützung für POLYGON, TIN, TRIANGLE hinzugefügt.

Diese Funktion unterstützt M-Koordinaten.

Diese Methode setzt die SQL/MM-Spezifikation um. SQL-MM IEC 13249-3: 5.1

Beispiele

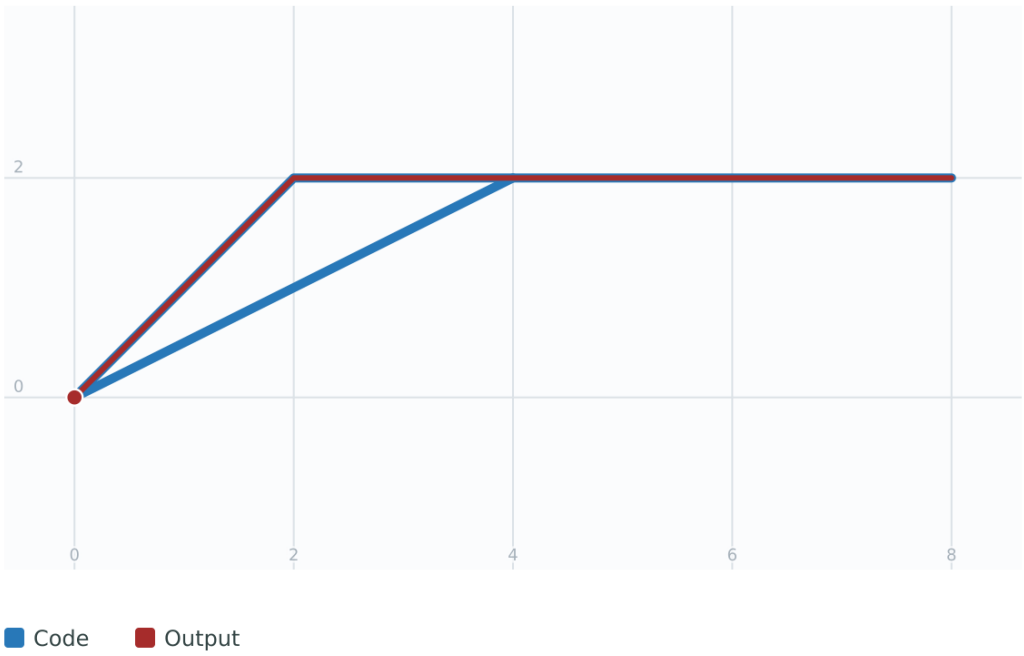
Code

```
SELECT ST_LocateBetween(
'MULTILINESTRING M ((1 2 3,3 4 2,9 4 3),(1 2 3,5 4 5))'::geometry,
1.5,
3);
```

Output

```
GEOMETRYCOLLECTION M (LINESTRING M (1 2 3,3 4 2,9 4 3),POINT M (1 2 3))
```

Figure



Extract the section between measures 2 and 8, offset to the left. Because the clipped line and the offset result share the same coordinate space, this example is intended for a single overlay panel rather than separate side-by-side figures.

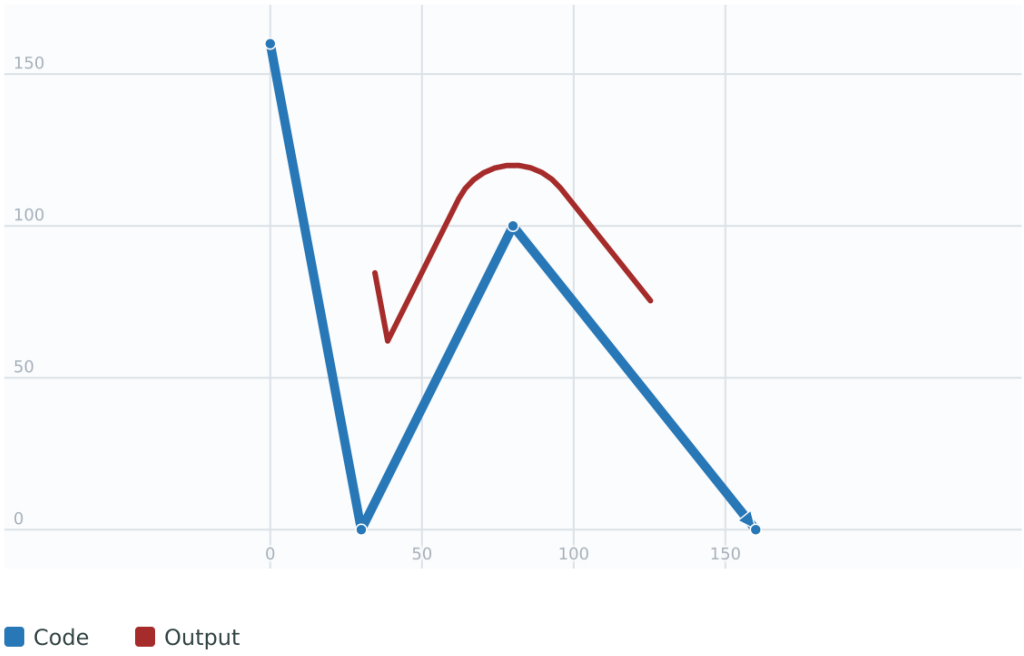
Code

```
SELECT ST_LocateBetween(
ST_AddMeasure(
    'LINESTRING (20 180,50 20,100 120,180 20)',
    0,
    10),
2,
8,
20);
```

Output

```
MULTILINESTRING((54.49835 104.53427,58.700561 82.122481,82.111456 128.944272,84.248937 132.324934,87.01637 ↩
    135.21267,90.303073 137.491987,93.977598 139.071724,97.892984 139.888702,101.892639 139.910247,105.816599 ↩
    139.135495,109.507928 137.595434,112.818995 135.351657,115.617376 132.493901,145.310173 95.377905))
```

Figure



Siehe auch

ST_LocateAlong, ST_LocateAlong, ST_LocateBetween

7.19.8 ST_LocateBetweenElevations

ST_LocateBetweenElevations — Gibt die Teile einer Geometrie zurück, die in einem Höhenbereich (Z) liegen.

Synopsis

geometry **ST_LocateBetweenElevations**(geometry geom, float8 elevation_start, float8 elevation_end);

Beschreibung

Gibt eine Geometrie (Sammlung) mit den Teilen einer Geometrie zurück, die in einem Höhenbereich (Z) liegen.

Das Beschneiden eines nicht konvexen POLYGONs kann zu einer ungültigen Geometrie führen.

Verfügbarkeit: 1.4.0

Verbessert: 3.0.0 - Unterstützung für POLYGON, TIN, TRIANGLE hinzugefügt.

Diese Funktion unterstützt 3d und lässt den Z-Index nicht fallen.

Beispiele

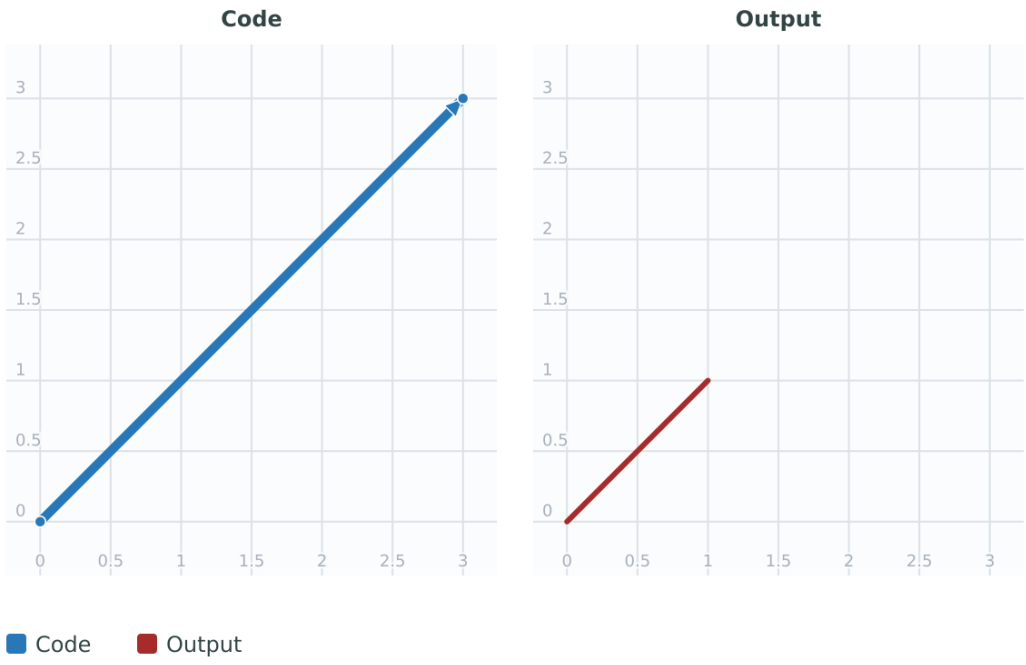
Code

```
SELECT ST_LocateBetweenElevations(
'LINESTRING(1 2 3,4 5 6)'::geometry,
2,
4);
```

Output

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

Figure



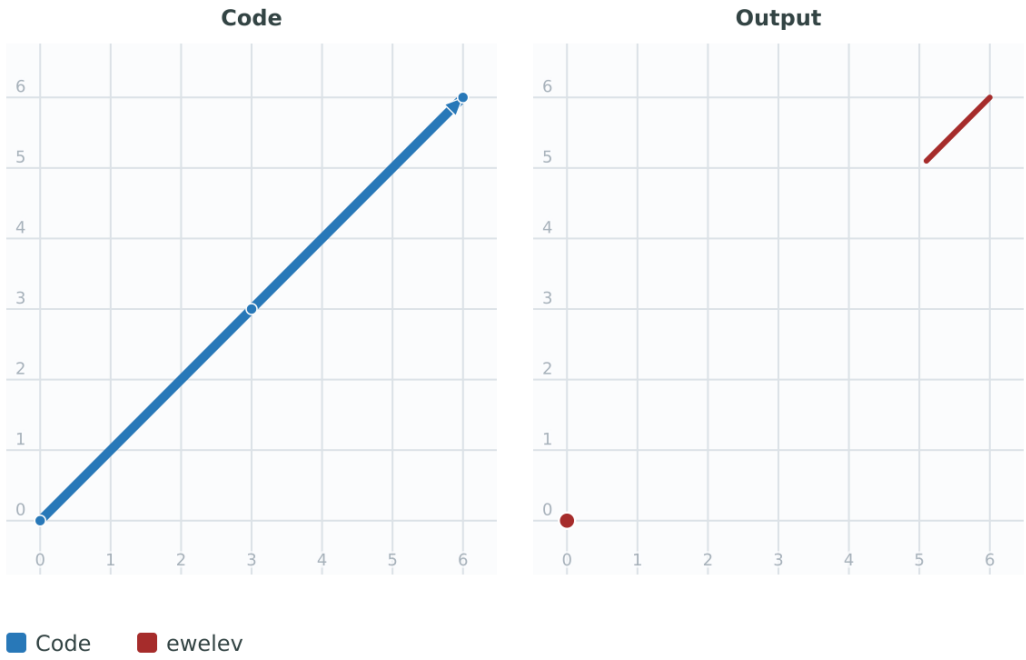
Code

```
SELECT ST_LocateBetweenElevations(
'LINESTRING(1 2 6,4 5 -1,7 8 9)',
6,
9) As ewelev;
```

Output

```
GEOMETRYCOLLECTION Z (POINT Z (1 2 6),LINESTRING Z (6.1 7.1 6,7 8 9))
```

Figure



Siehe auch

ST_Dump, ST_LocateAlong, ST_LocateBetween

7.19.9 ST_InterpolatePoint

ST_InterpolatePoint — Für einen gegebenen Punkt wird die Kilometrierung auf dem nächstliegenden Punkt einer Geometrie zurück.

Synopsis

float8 **ST_InterpolatePoint**(geometry linear_geom_with_measure, geometry point);

Beschreibung

Gibt einen interpolierten Messwert einer linear gemessenen Geometrie an der Stelle zurück, die dem angegebenen Punkt am nächsten liegt.



Anmerkung
Verwenden Sie diese Funktion nur für lineare Geometrien mit einer M-Komponente

Verfügbarkeit: 2.0.0

 Diese Funktion unterstützt 3d und lässt den Z-Index nicht fallen.

Beispiele

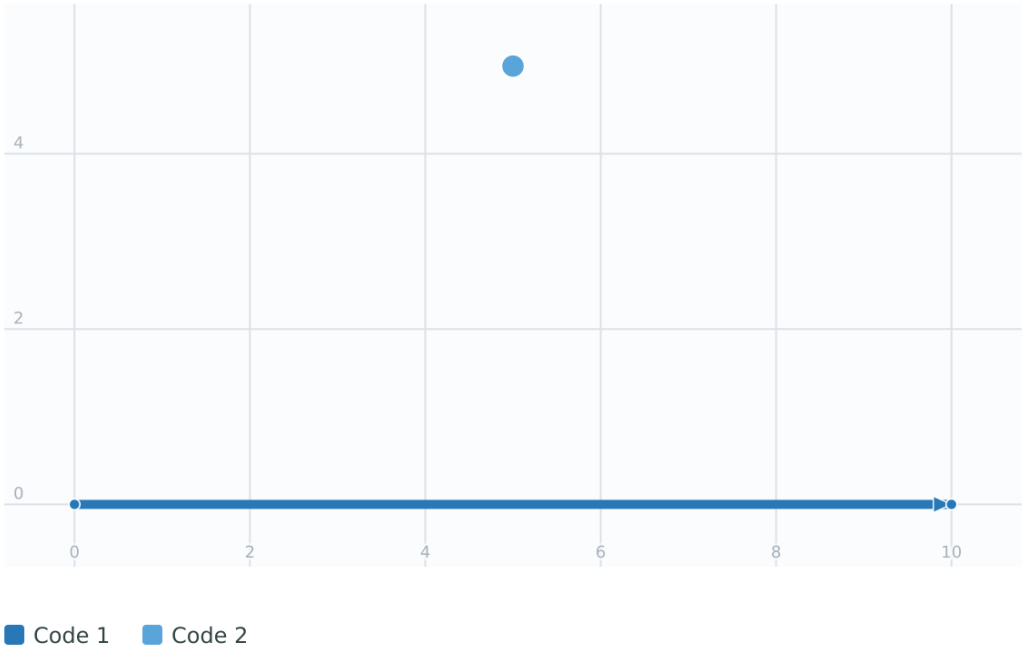
Code

```
SELECT ST_InterpolatePoint(
'LINESTRING M (0 0 0,10 0 20)',
'POINT(5 5)');
```

Output

```
10
```

Figure



Siehe auch

ST_AddMeasure, ST_LocateAlong, ST_LocateBetween

7.19.10 ST_3DInterpolatePoint

ST_3DInterpolatePoint — Returns the interpolated measure of a geometry closest to a point in 3D.

Synopsis

float8 **ST_3DInterpolatePoint**(geometry linear_geom_with_measure, geometry point);

Beschreibung

Returns the interpolated measure value of a linear ZM geometry at the location closest to the given point, using 3D (XYZ) distance for the projection. Use this function when the geometry has significant Z variation, such as flight trajectories, where ST_InterpolatePoint would give incorrect results by ignoring the Z dimension.

Anmerkung
The line must have both Z and M dimensions. The point should have a Z dimension.

Availability: 3.7.0

Diese Funktion unterstützt 3d und lässt den Z-Index nicht fallen.

Beispiele

This example interpolates on a line rising diagonally in Z, using a point at the 3D midpoint.

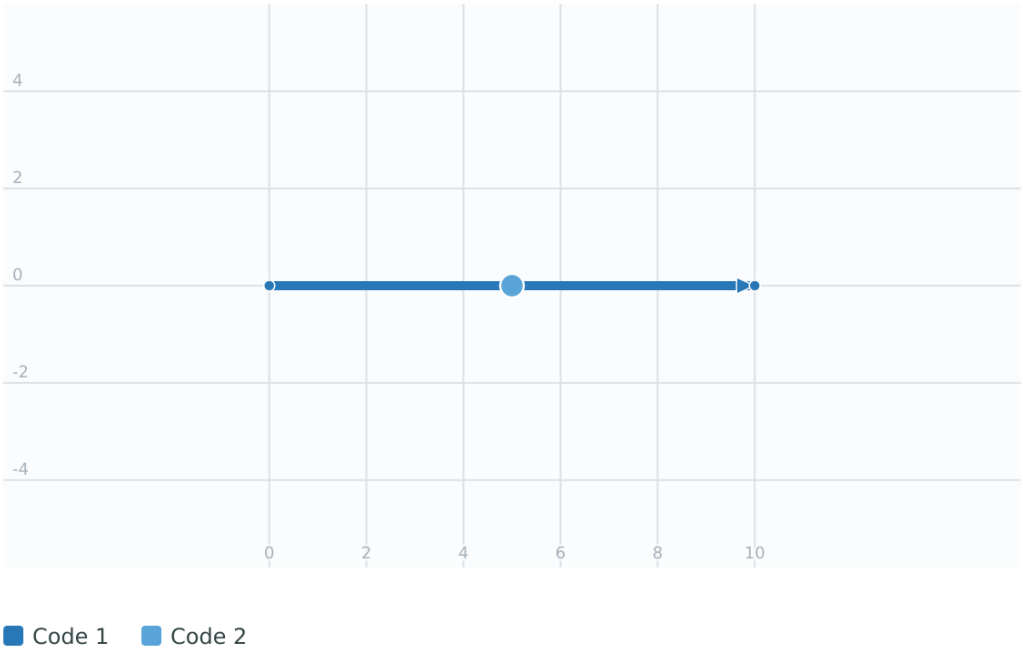
Code

```
SELECT ST_3DInterpolatePoint(
'LINESTRING ZM (0 0 0 0,10 0 10 20)',
'POINT Z (5 0 5)');
```

Output

```
10
```

Figure



This example uses a vertical line with zero XY extent, where 3D projection works but 2D projection does not.

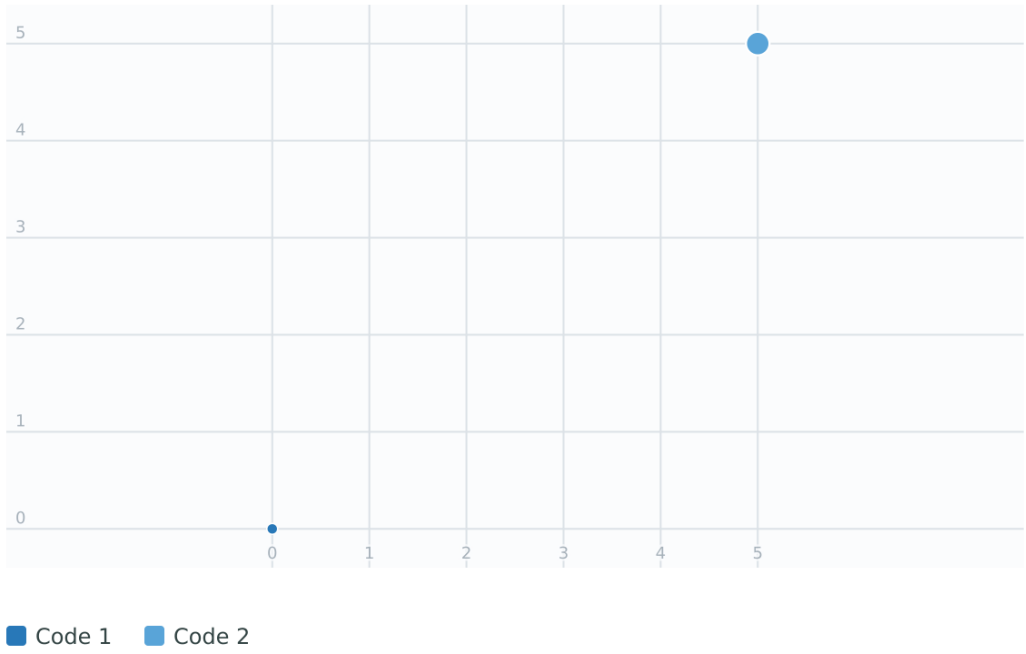
Code

```
SELECT ST_3DInterpolatePoint(
'LINESTRING ZM (0 0 0 0,0 0 10 100)',
'POINT Z (5 5 5)');
```

Output

```
50
```

Figure



Siehe auch

ST_InterpolatePoint, ST_AddMeasure, ST_LocateAlong, ST_LocateBetween

7.19.11 ST_AddMeasure

ST_AddMeasure — Interpoliert Maße entlang einer linearen Geometrie.

Synopsis

geometry **ST_AddMeasure**(geometry geom_mline, float8 measure_start, float8 measure_end);

Beschreibung

Gibt eine abgeleitete Geometrie mit einer zwischen Anfangs- und Endpunkt linear interpolierten Kilometrierung zurück. Wenn die Geometrie keine Dimension für die Kilometrierung aufweist, wird diese hinzugefügt. Wenn die Geometrie eine Dimension für die Kilometrierung hat, wird diese mit den neuen Werten überschrieben. Es werden nur LINESTRINGs und MULTILINESTRINGs unterstützt.

Verfügbarkeit: 1.5.0

Diese Funktion unterstützt 3d und lässt den Z-Index nicht fallen.

Beispiele

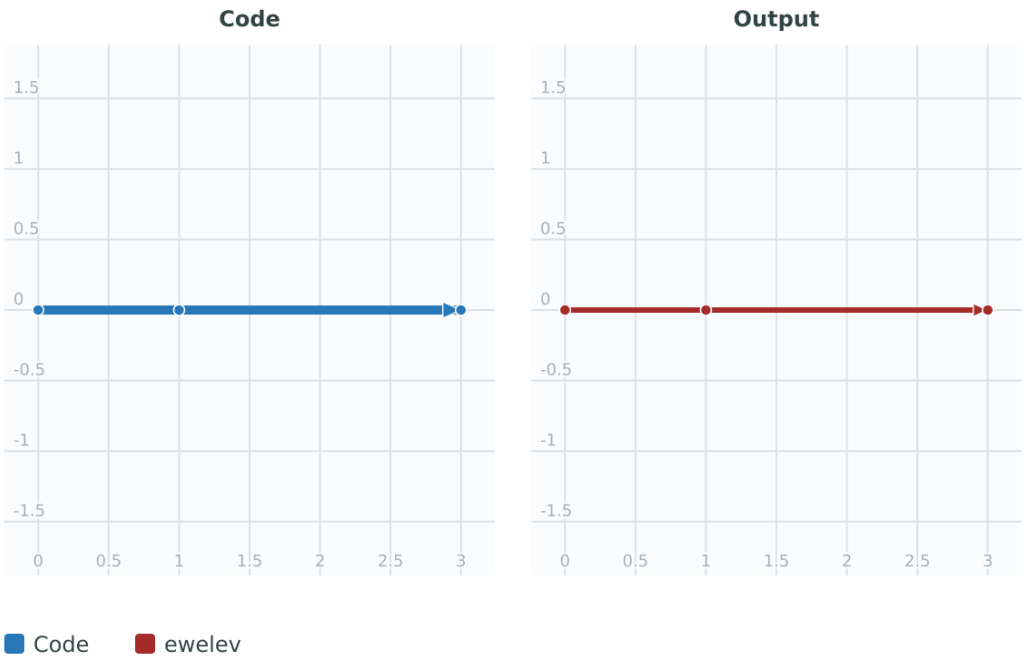
Code

```
SELECT ST_AddMeasure(
'LINESTRING(1 0,2 0,4 0)',
1,
4) As ewelev;
```

Output

```
LINESTRING M (1 0 1,2 0 2,4 0 4)
```

Figure



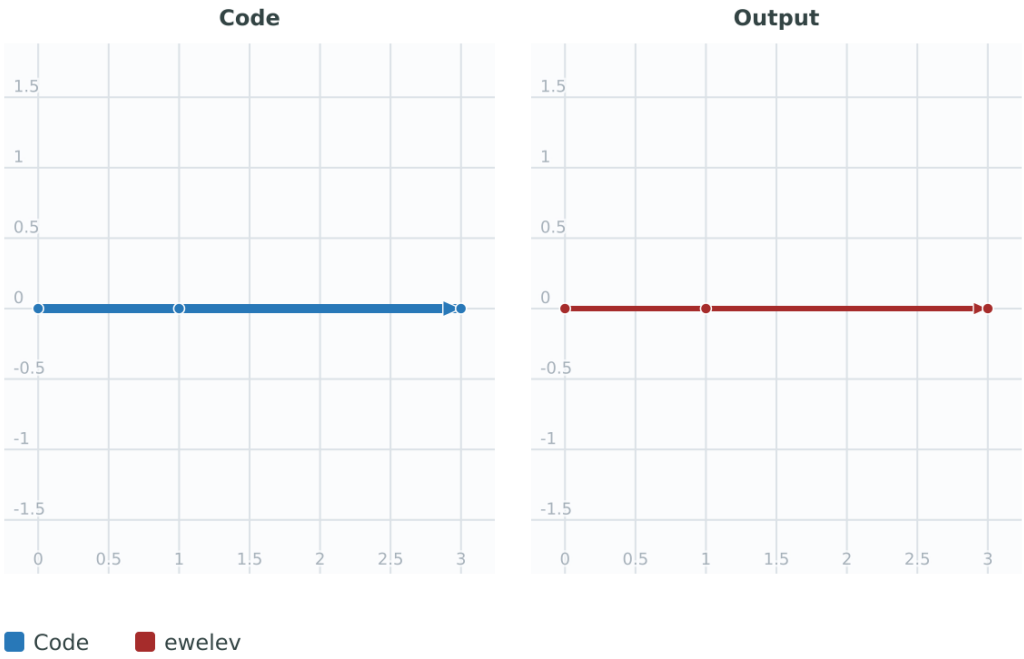
Code

```
SELECT ST_AddMeasure(
'LINESTRING(1 0 4,2 0 4,4 0 4)',
10,
40) As ewelev;
```

Output

```
LINESTRING ZM (1 0 4 10,2 0 4 20,4 0 4 40)
```

Figure



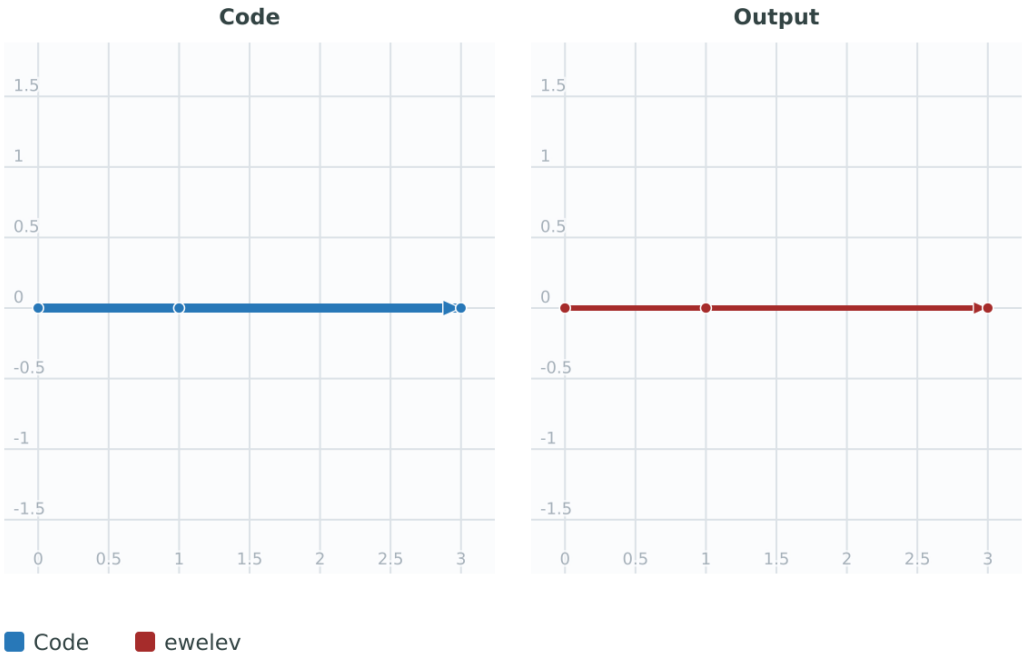
Code

```
SELECT ST_AddMeasure(
'LINESTRINGM(1 0 4, 2 0 4, 4 0 4)',
10,
40) As ewelev;
```

Output

```
LINESTRING M (1 0 10,2 0 20,4 0 40)
```

Figure



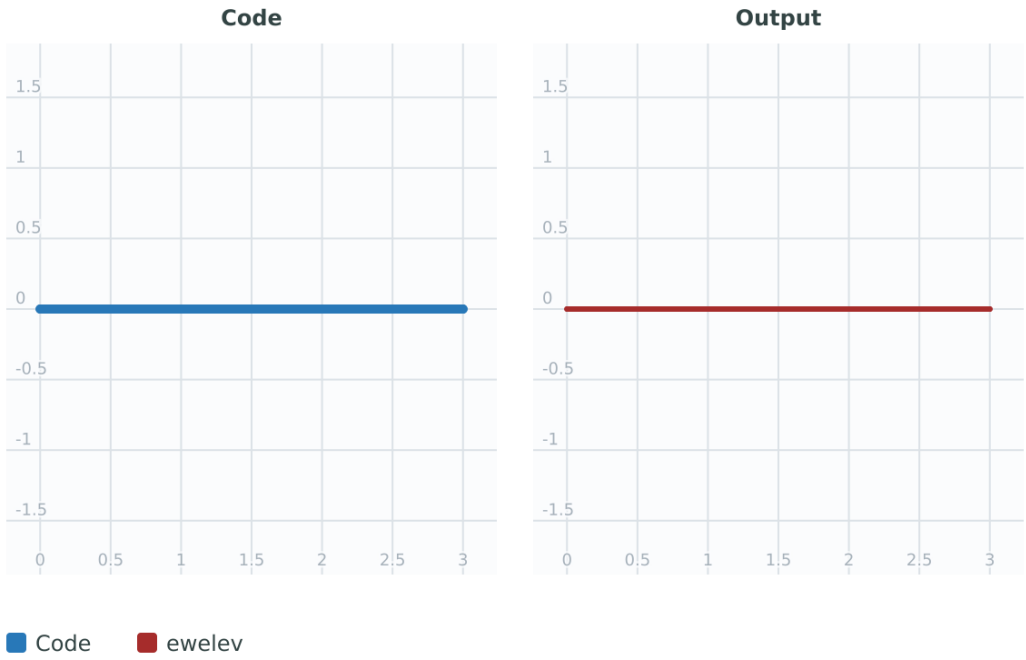
Code

```
SELECT ST_AddMeasure(
'MULTILINESTRINGM((1 0 4, 2 0 4, 4 0 4),(1 0 4, 2 0 4, 4 0 4))',
10,
70) As ewelev;
```

Output

```
MULTILINESTRING M ((1 0 10,2 0 20,4 0 40),(1 0 40,2 0 50,4 0 70))
```

Figure



7.20 Trajektorie-Funktionen

7.20.1 ST_IsValidTrajectory

ST_IsValidTrajectory — Prüft, ob die Geometrie eine gültige Flugbahn ist.

Synopsis

boolean **ST_IsValidTrajectory**(geometry line);

Beschreibung

Prüft, ob eine Geometrie eine gültige Trajektorie kodiert. Eine gültige Flugbahn wird als `LINESTRING` mit Maßen (M-Werten) dargestellt. Die Messwerte müssen von jedem Scheitelpunkt zum nächsten ansteigen.

Gültige Trajektorien werden als Eingabe für räumlich-zeitliche Funktionen wie ST_ClosestPointOfApproach

Verfügbarkeit: 2.2.0

Diese Funktion unterstützt 3d und lässt den Z-Index nicht fallen.

Beispiele

This example constructs a valid trajectory.

Code

```
SELECT ST_IsValidTrajectory(ST_MakeLine(ST_MakePointM(0, 0, 1),
ST_MakePointM(0, 1, 2)));
```

Output

```
t
```

This example constructs an invalid trajectory.

Code

```
SELECT ST_IsValidTrajectory(ST_MakeLine(ST_MakePointM(0, 0, 1), ST_MakePointM(0, 1, 0)));
```

Output

```
NOTICE:  Measure of vertex 1 (0) not bigger than measure of vertex 0 (1)
f
```

Siehe auch

ST_ClosestPointOfApproach

7.20.2 ST_ClosestPointOfApproach

ST_ClosestPointOfApproach — Liefert ein Maß für den nächstgelegenen Punkt der Annäherung von zwei Flugbahnen.

Synopsis

float8 **ST_ClosestPointOfApproach**(geometry track1, geometry track2);

Beschreibung

Gibt das kleinste Maß zurück, bei dem die entlang der angegebenen Trajektorien interpolierten Punkte den geringsten Abstand zueinander haben.

Die Eingaben müssen gültige Flugbahnen sein, wie von ST_IsValidTrajectory geprüft. Null wird zurückgegeben, wenn sich die Trajektorien in ihren M-Bereichen nicht überschneiden.

Um die tatsächlichen Punkte am berechneten Maß zu erhalten, verwenden Sie ST_LocateAlong.

Verfügbarkeit: 2.2.0

 Diese Funktion unterstützt 3d und lässt den Z-Index nicht fallen.

Beispiele

Return the time in which two objects moving between 10:00 and 11:00 are closest to each other and their distance at that point.

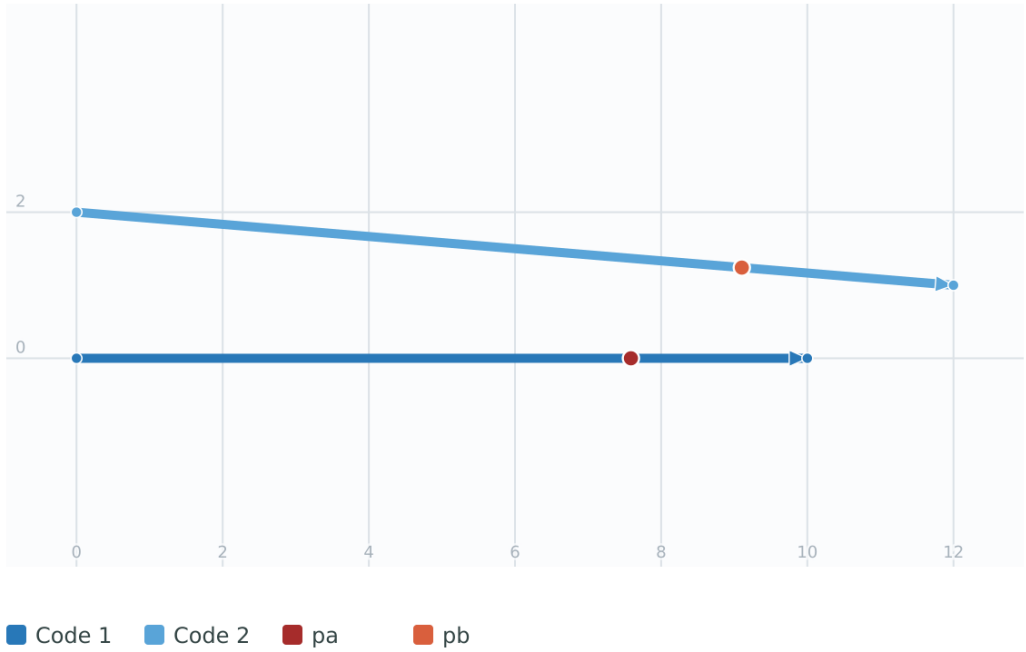
Code

```
WITH inp AS ( SELECT
  ST_AddMeasure('LINESTRING Z (0 0 0,10 0 5)'::geometry,
    extract(epoch from timestamp '2015-05-26 10:00' AT TIME ZONE 'UTC'),
    extract(epoch from timestamp '2015-05-26 11:00' AT TIME ZONE 'UTC')
  ) a,
  ST_AddMeasure('LINESTRING Z (0 2 10,12 1 2)'::geometry,
    extract(epoch from timestamp '2015-05-26 10:00' AT TIME ZONE 'UTC'),
    extract(epoch from timestamp '2015-05-26 11:00' AT TIME ZONE 'UTC')
  ) b
), cpa AS (
  SELECT ST_ClosestPointOfApproach(a, b) m FROM inp
), points AS (
  SELECT ST_GeometryN(ST_LocateAlong(a, m), 1) pa,
         ST_GeometryN(ST_LocateAlong(b, m), 1) pb
  FROM inp, cpa
)
SELECT to_timestamp(m) AT TIME ZONE 'UTC' t,
       ST_3DDistance(pa, pb) distance,
       pa AS pa, pb AS pb
FROM points, cpa;
```

Output

```
-[ RECORD 1 ]----------------------------
t        | 2015-05-26 10:45:31.034483
distance | 1.9652147377620688
pa       | POINT ZM (7.59 0 3.79 1432637131.03)
pb       | POINT ZM (9.1 1.24 3.93 1432637131.03)
```

Figure



Siehe auch

ST_IsValidTrajectory, ST_DistanceCPA, ST_LocateAlong, ST_AddMeasure

7.20.3 ST_DistanceCPA

ST_DistanceCPA — Liefert den Abstand zwischen dem nächstgelegenen Punkt der Annäherung zweier Flugbahnen.

Synopsis

float8 **ST_DistanceCPA**(geometry track1, geometry track2);

Beschreibung

Liefert den Abstand (in 2D) zwischen zwei Flugbahnen an ihrem engsten Annäherungspunkt.

Die Eingaben müssen gültige Flugbahnen sein, wie von ST_IsValidTrajectory geprüft. Null wird zurückgegeben, wenn sich die Trajektorien in ihren M-Bereichen nicht überschneiden.

Verfügbarkeit: 2.2.0

 Diese Funktion unterstützt 3d und lässt den Z-Index nicht fallen.

Beispiele

Return the minimum distance of two objects moving between 10:00 and 11:00.

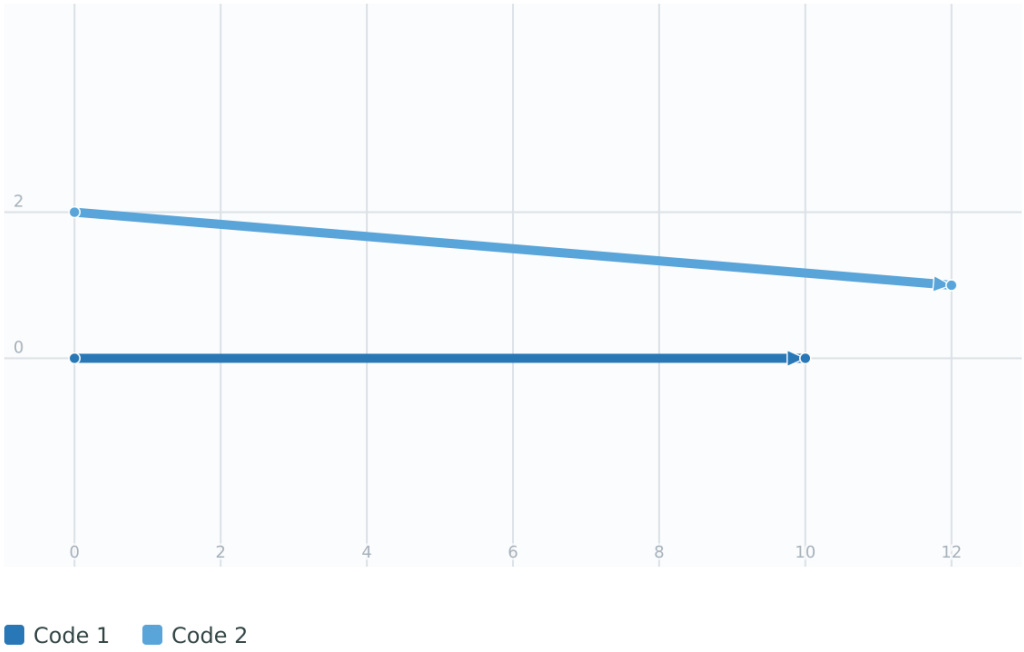
Code

```
WITH inp AS ( SELECT
  ST_AddMeasure('LINESTRING Z (0 0 0,10 0 5)'::geometry,
    extract(epoch from timestamp '2015-05-26 10:00' AT TIME ZONE 'UTC'),
    extract(epoch from timestamp '2015-05-26 11:00' AT TIME ZONE 'UTC')
  ) a,
  ST_AddMeasure('LINESTRING Z (0 2 10,12 1 2)'::geometry,
    extract(epoch from timestamp '2015-05-26 10:00' AT TIME ZONE 'UTC'),
    extract(epoch from timestamp '2015-05-26 11:00' AT TIME ZONE 'UTC')
  ) b
)
SELECT ST_DistanceCPA(a, b) distance FROM inp;
```

Output

```
1.9652147377620688
```

Figure



Siehe auch

ST_IsValidTrajectory, ST_ClosestPointOfApproach, ST_AddMeasure, |=|

7.20.4 ST_CPAWithin

ST_CPAWithin — Prüft, ob der nächstgelegene Punkt der Annäherung zweier Flugbahnen innerhalb der angegebenen Entfernung liegt.

Synopsis

boolean **ST_CPAWithin**(geometry track1, geometry track2, float8 dist);

Beschreibung

Prüft, ob sich zwei bewegte Objekte jemals näher als die angegebene Entfernung befunden haben.

Die Eingaben müssen gültige Flugbahnen sein, wie von ST_IsValidTrajectory geprüft. False wird zurückgegeben, wenn sich die Trajektorien in ihren M-Bereichen nicht überschneiden.

Verfügbarkeit: 2.2.0

Diese Funktion unterstützt 3d und lässt den Z-Index nicht fallen.

Beispiele

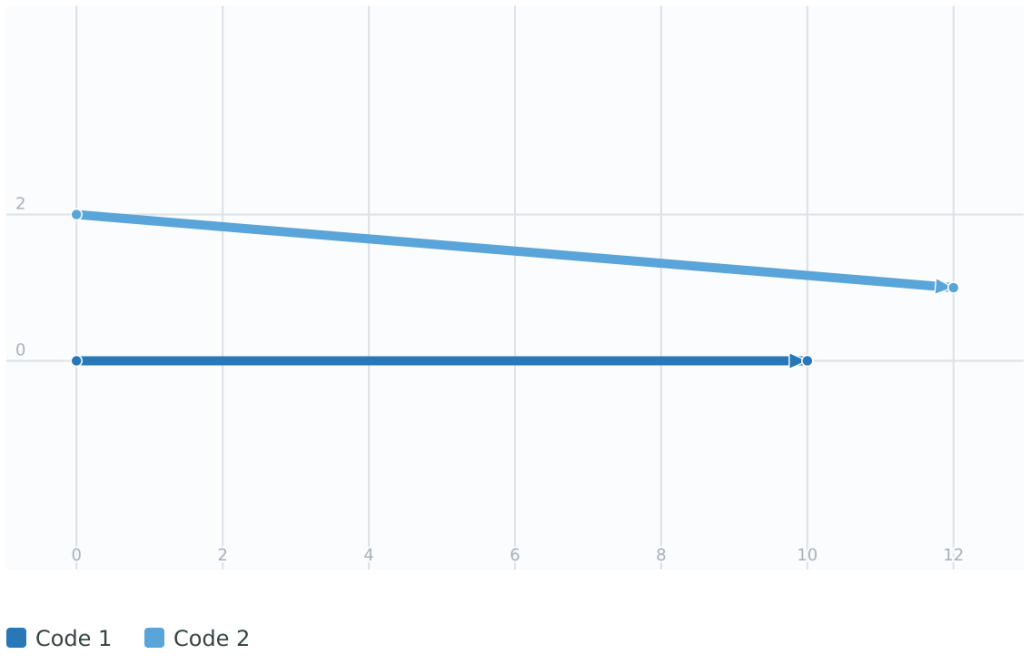
Code

```
WITH inp AS ( SELECT
  ST_AddMeasure('LINESTRING Z (0 0 0,10 0 5)'::geometry,
    extract(epoch from timestamp '2015-05-26 10:00' AT TIME ZONE 'UTC'),
    extract(epoch from timestamp '2015-05-26 11:00' AT TIME ZONE 'UTC')
  ) a,
  ST_AddMeasure('LINESTRING Z (0 2 10,12 1 2)'::geometry,
    extract(epoch from timestamp '2015-05-26 10:00' AT TIME ZONE 'UTC'),
    extract(epoch from timestamp '2015-05-26 11:00' AT TIME ZONE 'UTC')
  ) b
)
SELECT ST_CPAWithin(a, b, 2), ST_DistanceCPA(a, b) distance FROM inp;
```

Output

```
st_cpawithin |     distance
-------------+------------------
 t           | 1.9652147377620688
```

Figure



Siehe auch

ST_IsValidTrajectory, ST_ClosestPointOfApproach, ST_DistanceCPA, |=|

7.21 Version Funktionen

7.21.1 PostGIS_Extensions_Upgrade

PostGIS_Extensions_Upgrade — Packt und aktualisiert PostGIS-Erweiterungen (z.B. postgis_raster, postgis_topology, postgis_sfcgal) auf die angegebene oder neueste Version.

Synopsis

text **PostGIS_Extensions_Upgrade**(text target_version=null);

Beschreibung

Packages and upgrades PostGIS extensions to given or latest version. Only extensions you have installed in the database will be packaged and upgraded if needed. Reports full PostGIS version and build configuration infos after. This is short-hand for doing multiple CREATE EXTENSION .. FROM unpackaged and ALTER EXTENSION .. UPDATE for each PostGIS extension. Currently only tries to upgrade extensions postgis, postgis_raster, postgis_sfcgal, and postgis_topology.

Verfügbarkeit: 2.5.0

Geändert: 3.4.0 um das Argument target_version hinzuzufügen.

Geändert: 3.3.0 Unterstützung für Upgrades von jeder PostGIS-Version. Funktioniert nicht auf allen Systemen.

Geändert: 3.0.0, um lose Erweiterungen neu zu packen und postgis_raster zu unterstützen.

Beispiele

Code

```
SELECT PostGIS_Extensions_Upgrade();
```

Output

```
NOTICE:  Packaging extension postgis
NOTICE:  Packaging extension postgis_raster
NOTICE:  Packaging extension postgis_sfcgal
NOTICE:  Extension postgis_topology is not available or not packagable for some reason
```

Siehe auch

Abschnitt 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 — Meldet die vollständige PostGIS-Version und Informationen zur Build-Konfiguration.

Synopsis

text **PostGIS_Full_Version**();

Beschreibung

Meldet die vollständige PostGIS-Version und Informationen zur Build-Konfiguration. Informiert auch über die Synchronisation zwischen Bibliotheken und Skripten und schlägt bei Bedarf Upgrades vor.

Verbessert: 3.4.0 enthält jetzt zusätzliche PROJ-Konfigurationen NETWORK_ENABLED, URL_ENDPOINT und DATABASE_PATH des proj.db-Speicherorts

Beispiele

Code

```
SELECT PostGIS_Full_Version();
```

Output

```
POSTGIS="3.7.0dev 3.7.0alpha1-477-ga9004e778" [EXTENSION] PGSQL="180" GEOS="3.15.0beta1" SFCGAL="2.3.0" PROJ="9.8.1 ↩
     NETWORK_ENABLED=OFF URL_ENDPOINT=https://cdn.proj.org DATABASE_PATH=/usr/share/proj/proj.db" GDAL="GDAL 3.13.1, ↩
     released 2026/06/01" LIBXML="2.15.3"
```

Siehe auch

Abschnitt 3.4, PostGIS_GEOS_Version, PostGIS_Lib_Version, PostGIS_LibXML_Version, PostGIS_PROJ_Version, PostGIS_Wagyu_
PostGIS_Version

7.21.3 PostGIS_GEOS_Version

PostGIS_GEOS_Version — Gibt die Versionsnummer der GEOS-Bibliothek zurück.

Synopsis

text **PostGIS_GEOS_Version**();

Beschreibung

Gibt die Versionsnummer der GEOS-Bibliothek zurück, oder `NULL` wenn die GEOS-Unterstützung nicht aktiviert ist.

Beispiele

Code

```
SELECT PostGIS_GEOS_Version();
```

Output

```
3.12.0dev-CAPI-1.18.0
```

Siehe auch

PostGIS_Full_Version, PostGIS_Lib_Version, PostGIS_LibXML_Version, PostGIS_PROJ_Version, PostGIS_Version

7.21.4 PostGIS_GEOS_Compiled_Version

PostGIS_GEOS_Compiled_Version — Gibt die Versionsnummer der GEOS-Bibliothek zurück, mit der PostGIS erstellt wurde.

Synopsis

text **PostGIS_GEOS_Compiled_Version**();

Beschreibung

Gibt die Versionsnummer der GEOS-Bibliothek zurück, mit der PostGIS erstellt wurde.

Verfügbarkeit: 3.4.0

Beispiele

Code

```
SELECT PostGIS_GEOS_Compiled_Version();
```

Output

```
3.12.0
```

Siehe auch

PostGIS_GEOS_Version, PostGIS_Full_Version

7.21.5 PostGIS_Liblwgeom_Version

PostGIS_Liblwgeom_Version — Gibt die Versionsnummer der liblwgeom-Bibliothek zurück. Diese sollte mit der Version von PostGIS übereinstimmen.

Synopsis

text **PostGIS_Liblwgeom_Version**();

Beschreibung

Liefert die Versionsnummer der Bibliothek liblwgeom/

Beispiele

Code

```
SELECT PostGIS_Liblwgeom_Version();
```

Output

```
3.4.0dev 3.3.0rc2-993-g61bdf43a7
```

Siehe auch

PostGIS_Full_Version, PostGIS_Lib_Version, PostGIS_LibXML_Version, PostGIS_PROJ_Version, PostGIS_Version

7.21.6 PostGIS_LibXML_Version

PostGIS_LibXML_Version — Gibt die Versionsnummer der libxml2-Bibliothek zurück.

Synopsis

text **PostGIS_LibXML_Version**();

Beschreibung

Gibt die Versionsnummer der LibXML2-Bibliothek zurück.

Verfügbarkeit: 1.5

Beispiele

Code

```
SELECT PostGIS_LibXML_Version();
```

Output

```
2.9.10
```

Siehe auch

PostGIS_Full_Version, PostGIS_Lib_Version, PostGIS_PROJ_Version, PostGIS_GEOS_Version, PostGIS_Version

7.21.7 PostGIS_LibJSON_Version

PostGIS_LibJSON_Version — Returns the version number of the libjson-c library.

Synopsis

text **PostGIS_LibJSON_Version**();

Beschreibung

Returns the version number of the LibJSON-C library.

Beispiele

Code

```
SELECT PostGIS_LibJSON_Version();
```

Output

```
0.17
```

Siehe auch

PostGIS_Full_Version, PostGIS_Lib_Version, PostGIS_PROJ_Version, PostGIS_GEOS_Version, PostGIS_Version

7.21.8 PostGIS_Lib_Build_Date

PostGIS_Lib_Build_Date — Gibt das Erstellungsdatum der PostGIS-Bibliothek zurück.

Synopsis

text **PostGIS_Lib_Build_Date**();

Beschreibung

Gibt das Erstellungsdatum der PostGIS-Bibliothek zurück.

Beispiele

Code

```
SELECT PostGIS_Lib_Build_Date();
```

Output

```
2023-06-22 03:56:11
```

7.21.9 PostGIS_Lib_Version

PostGIS_Lib_Version — Gibt die Versionsnummer der PostGIS-Bibliothek zurück.

Synopsis

text **PostGIS_Lib_Version**();

Beschreibung

Gibt die Versionsnummer der PostGIS-Bibliothek zurück.

Beispiele

Code

```
SELECT PostGIS_Lib_Version();
```

Output

```
3.4.0dev
```

Siehe auch

PostGIS_Full_Version, PostGIS_GEOS_Version, PostGIS_LibXML_Version, PostGIS_PROJ_Version, PostGIS_Version

7.21.10 PostGIS_PROJ_Version

PostGIS_PROJ_Version — Gibt die Versionsnummer der PROJ4-Bibliothek zurück.

Synopsis

text **PostGIS_PROJ_Version**();

Beschreibung

Gibt die Versionsnummer der PROJ-Bibliothek und einige Konfigurationsoptionen von proj zurück.

Verbessert: 3.4.0 enthält jetzt NETWORK_ENABLED, URL_ENDPOINT und DATABASE_PATH des proj.db-Speicherorts

Beispiele

Code

```
SELECT PostGIS_PROJ_Version();
```

Output

```
7.2.1 NETWORK_ENABLED=OFF URL_ENDPOINT=https://cdn.proj.org USER_WRITABLE_DIRECTORY=/tmp/ ↩
    proj DATABASE_PATH=/usr/share/proj/proj.db
```

Siehe auch

PostGIS_PROJ_Compiled_Version, PostGIS_Full_Version, PostGIS_GEOS_Version, PostGIS_Lib_Version, PostGIS_LibXML_Versio
PostGIS_Version

7.21.11 PostGIS_PROJ_Compiled_Version

PostGIS_PROJ_Compiled_Version — Returns the version number of the PROJ library against which PostGIS was built.

Synopsis

text **PostGIS_PROJ_Compiled_Version**();

Beschreibung

Returns the version number of the PROJ library, or against which PostGIS was built.

Verfügbarkeit: 3.5.0

Beispiele

Code

```
SELECT PostGIS_PROJ_Compiled_Version();
```

Output

```
9.1.1
```

Siehe auch

PostGIS_PROJ_Version, PostGIS_Full_Version

7.21.12 PostGIS_Wagyu_Version

PostGIS_Wagyu_Version — Gibt die Versionsnummer der internen Wagyu-Bibliothek zurück.

Synopsis

text **PostGIS_Wagyu_Version**();

Beschreibung

Gibt die Versionsnummer der internen Wagyu-Bibliothek zurück, oder `NULL` wenn die Wagyu-Unterstützung nicht aktiviert ist.

Beispiele

Code

```
SELECT PostGIS_Wagyu_Version();
```

Output

```
0.5.0 (Internal)
```

Siehe auch

PostGIS_Full_Version, PostGIS_GEOS_Version, PostGIS_PROJ_Version, PostGIS_Lib_Version, PostGIS_LibXML_Version, PostGIS_Version

7.21.13 PostGIS_Scripts_Build_Date

PostGIS_Scripts_Build_Date — Gibt das Erstellungsdatum der PostGIS-Skripte zurück.

Synopsis

text **PostGIS_Scripts_Build_Date**();

Beschreibung

Gibt das Erstellungsdatum der PostGIS-Skripte zurück.

Verfügbarkeit: 1.0.0RC1

Beispiele

Code

```
SELECT PostGIS_Scripts_Build_Date();
```

Output

```
2023-06-22 03:56:11
```

Siehe auch

PostGIS_Full_Version, PostGIS_GEOS_Version, PostGIS_Lib_Version, PostGIS_LibXML_Version, PostGIS_Version

7.21.14 PostGIS_Scripts_Installed

PostGIS_Scripts_Installed — Gibt die Version der in dieser Datenbank installierten PostGIS-Skripte zurück.

Synopsis

text **PostGIS_Scripts_Installed**();

Beschreibung

Gibt die Version der in dieser Datenbank installierten PostGIS-Skripte zurück.



Anmerkung
Wenn die Ausgabe dieser Funktion nicht mit der Ausgabe von PostGIS_Scripts_Released übereinstimmt, haben Sie wahrscheinlich versäumt, eine bestehende Datenbank ordnungsgemäß zu aktualisieren. Weitere Informationen finden Sie im Abschnitt Upgrading.

Verfügbarkeit: 0.9.0

Beispiele

Code

```
SELECT PostGIS_Scripts_Installed();
```

Output

```
3.4.0dev 3.3.0rc2-993-g61bdf43a7
```

Siehe auch

PostGIS_Full_Version, PostGIS_Scripts_Released, PostGIS_Version

7.21.15 PostGIS_Scripts_Released

PostGIS_Scripts_Released — Gibt die Versionsnummer des Skripts postgis.sql zurück, das mit der installierten PostGIS-Bibliothek veröffentlicht wurde.

Synopsis

text **PostGIS_Scripts_Released**();

Beschreibung

Gibt die Versionsnummer des Skripts postgis.sql zurück, das mit der installierten PostGIS-Bibliothek veröffentlicht wurde.



Anmerkung
Ab Version 1.1.0 gibt diese Funktion denselben Wert zurück wie PostGIS_Lib_Version. Aus Gründen der Abwärtskompatibilität wurde diese Funktion beibehalten.

Verfügbarkeit: 0.9.0

Beispiele

Code

```
SELECT PostGIS_Scripts_Released();
```

Output

```
3.4.0dev 3.3.0rc2-993-g61bdf43a7
```

Siehe auch

PostGIS_Full_Version, PostGIS_Scripts_Installed, PostGIS_Lib_Version

7.21.16 PostGIS_Version

PostGIS_Version — Gibt die PostGIS-Versionsnummer und die Kompilieroptionen zurück.

Synopsis

text **PostGIS_Version**();

Beschreibung

Gibt die PostGIS-Versionsnummer und die Kompilieroptionen zurück.

Beispiele

Code

```
SELECT PostGIS_Version();
```

Output

```
3.4 USE_GEOS=1 USE_PROJ=1 USE_STATS=1
```

Siehe auch

PostGIS_Full_Version, PostGIS_GEOS_Version, PostGIS_Lib_Version, PostGIS_LibXML_Version, PostGIS_PROJ_Version

7.22 PostGIS Grand Unified Custom Variables (GUCs)

7.22.1 postgis.gdal_datapath

postgis.gdal_datapath — Eine Konfigurationsmöglichkeit um den Wert von GDAL's GDAL_DATA Option zu setzen. Wenn sie nicht gesetzt ist, wird die Umgebungsvariable GDAL_DATA verwendet.

Beschreibung

Eine PostgreSQL GUC Variable zum setzten von GDAL's GDAL_DATA Option. Der `postgis.gdal_datapath` Wert sollte dem gesamten physischen Pfad zu den Datendateien von GDAL entsprechen.

Diese Konfigurationsmöglichkeit ist am nützlichsten auf Windows Plattformen, wo der Dateipfad von "data" nicht fest kodiert ist. Diese Option sollte auch gesetzt werden, wenn sich die Datendateien nicht in dem von GDAL erwarteten Pfad befinden.



Anmerkung
Diese Option kann in der Konfigurationsdatei "postgresql.conf" gesetzt werden. Sie kann auch pro Verbindung oder pro Transaktion gesetzt werden.

Verfügbarkeit: 2.2.0



Anmerkung
Zusätzliche Informationen über GDAL_DATA ist unter den Konfigurationsmöglichkeiten für GDAL zu finden.

Beispiele

Den `postgis.gdal_datapath` setzen oder zurücksetzen

Code

```
SET postgis.gdal_datapath TO '/usr/local/share/gdal.hidden';
SET postgis.gdal_datapath TO default;
```

Auf Windows für eine bestimmte Datenbank setzen

Code

```
ALTER DATABASE gisdb
SET postgis.gdal_datapath = 'C:/Program Files/PostgreSQL/9.3/gdal-data';
```

Siehe auch

PostGIS_GDAL_Version, ST_Transform

7.22.2 postgis.gdal_enabled_drivers

postgis.gdal_enabled_drivers — Eine Konfigurationsmöglichkeit um einen GDAL Treiber in der PostGIS Umgebung zu aktivieren. Beeinflusst die Konfigurationsvariable GDAL_SKIP von GDAL.

Beschreibung

Eine Konfigurationsmöglichkeit um einen GDAL Treiber in der PostGIS Umgebung zu aktivieren. Beeinflusst die Konfigurationsvariable GDAL_SKIP von GDAL. Diese Option kann in der PostgreSQL Konfigurationsdatei "postgresql.conf" gesetzt werden. Sie kann aber auch pro Verbindung oder pro Transaktion gesetzt werden.

Der Ausgangswert von `postgis.gdal_enabled_drivers` kann auch beim Startprozess von PostgreSQL gesetzt werden, nämlich durch die Übergabe der Umgebungsvariablen `POSTGIS_GDAL_ENABLED_DRIVERS`, welche die Liste der aktivierten Treiber enthält.

Aktivierte GDAL Treiber können auch über die Kurzbezeichnung oder den Code des Treibers bestimmt werden. Kurzbezeichnungen und Codes für die Treiber finden sich unter GDAL Raster Formate Es können mehrere, durch Leerzeichen getrennte Treiber angegeben werden.



Anmerkung
Für `postgis.gdal_enabled_drivers` sind drei spezielle, case-sensitive Codes verfügbar.

- `DISABLE_ALL` deaktiviert alle GDAL-Treiber. Falls vorhanden, überschreibt `DISABLE_ALL` alle anderen Werte in `postgis.gdal_enabled_drivers`.
- `ENABLE_ALL` aktiviert alle GDAL-Treiber.
- `VSICURL` aktiviert GDAL's `/vsicurl/` virtuelles Dateisystem.

Falls `postgis.gdal_enabled_drivers`auf DISABLE_ALL gesetzt ist, kommt es bei der Anwendung von out-db Rastern, ST_FromGDALRaster(), ST_AsGDALRaster(), ST_AsTIFF(), ST_AsJPEG() und ST_AsPNG() zu Fehlermeldungen.



Anmerkung
`postgis.gdal_enabled_drivers` wird bei der Standardinstallation von PostGIS auf DISABLE_ALL gesetzt.



Anmerkung
Weiterführende Informationen über GDAL_SKIP ist auf GDAL's Configuration Options zu finden.

Verfügbarkeit: 2.2.0

Beispiele

Set and reset `postgis.gdal_enabled_drivers` for the current session.

Code

```
SET postgis.gdal_enabled_drivers = 'ENABLE_ALL';
SET postgis.gdal_enabled_drivers = default;
```

Set specific drivers for all new connections to a database.

Code

```
ALTER DATABASE mygisdb SET postgis.gdal_enabled_drivers TO 'GTiff PNG JPEG';
```

Set the default for the whole database cluster to enable all drivers. This requires superuser access. Also note that database, session, and user settings override this.

This example writes the setting to postgres.auto.conf.

Code

```
ALTER SYSTEM SET postgis.gdal_enabled_drivers TO 'ENABLE_ALL';
```

Reload the PostgreSQL configuration so the changed setting is applied.

Code

```
SELECT pg_reload_conf();
```

Siehe auch

ST_FromGDALRaster, ST_AsGDALRaster, ST_AsTIFF, ST_AsPNG, ST_AsJPEG, postgis.enable_outdb_rasters

7.22.3 postgis.enable_outdb_rasters

postgis.enable_outdb_rasters — Eine boolesche Konfigurationsmöglichkeit um den Zugriff auf out-db Rasterbänder zu ermöglichen

Beschreibung

Eine boolesche Konfigurationsmöglichkeit um den Zugriff auf out-db Rasterbänder zu ermöglichen. Diese Option kann in der PostgreSQL Konfigurationsdatei "postgresql.conf" gesetzt werden. Kann aber auch pro Verbindung oder pro Transaktion gesetzt werden.

Der Ausgangswert von `postgis.enable_outdb_rasters` kann auch beim Startprozess von PostgreSQL gesetzt werden, nämlich durch die Übergabe der Umgebungsvariablen `POSTGIS_ENABLE_OUTDB_RASTERS`, welche ungleich null sein muss.



Anmerkung
Auch wenn `postgis.enable_outdb_rasters` True ist, bestimmt die GUC `postgis.enable_outdb_rasters` die zugänglichen Rasterformate.



Anmerkung
Bei der Standardinstallation von PostGIS ist `postgis.enable_outdb_rasters` auf False gesetzt.

Verfügbarkeit: 2.2.0

Beispiele

Set and reset `postgis.enable_outdb_rasters` for the current session.

Code

```
SET postgis.enable_outdb_rasters TO True;
SET postgis.enable_outdb_rasters = default;
SET postgis.enable_outdb_rasters = True;
SET postgis.enable_outdb_rasters = False;
```

Set the default for all new connections to a database.

Code

```
ALTER DATABASE gisdb SET postgis.enable_outdb_rasters = true;
```

Set the default for the whole database cluster. This requires superuser access. Also note that database, session, and user settings override this.

This example writes the setting to postgres.auto.conf.

Code

```
ALTER SYSTEM SET postgis.enable_outdb_rasters = true;
```

Reload the PostgreSQL configuration so the changed setting is applied.

Code

```
SELECT pg_reload_conf();
```

Siehe auch

postgis.gdal_enabled_drivers postgis.gdal_vsi_options

7.22.4 postgis.gdal_vsi_options

postgis.gdal_vsi_options — Eine boolesche Konfigurationsmöglichkeit um den Zugriff auf out-db Rasterbänder zu ermöglichen

Beschreibung

Konfigurationsoptionen steuern z.B. wie viel Platz GDAL dem lokalen Datencache zuweist, ob Übersichten gelesen werden sollen und welche Zugriffsschlüssel für entfernte Out-DB-Datenquellen verwendet werden sollen.

Verfügbarkeit: 3.2.0

Beispiele

`postgis.enable_outdb_rasters` setzen oder zurücksetzen

Code

```
SET postgis.gdal_vsi_options = 'AWS_ACCESS_KEY_ID=xxxxxxxxxxxxxxx AWS_SECRET_ACCESS_KEY= ↩
    yyyyyyyyyyyyyyyyyyyyyyyyyy';
```

Setzen Sie `postgis.gdal_vsi_options` nur für die aktuelle Transaktion mit dem Schlüsselwort `LOCAL`:

Code

```
SET LOCAL postgis.gdal_vsi_options = 'AWS_ACCESS_KEY_ID=xxxxxxxxxxxxxxx ↩
    AWS_SECRET_ACCESS_KEY=yyyyyyyyyyyyyyyyyyyyyyyyyy';
```

Siehe auch

postgis.enable_outdb_rasters postgis.gdal_enabled_drivers

7.22.5 postgis.gdal_cpl_debug

postgis.gdal_cpl_debug — A boolean configuration to turn logging of GDAL debug messages on and off.

Beschreibung

By default, GDAL logging is printed to stderr, and lower level debug messages are not printed at all. Turning this GUC to true will cause detailed GDAL logging to be sent into the PostgreSQL server log. PostgreSQL clients receive only generic GDAL notice, debug, warning, or error messages.

Verfügbarkeit: 3.6.0

Siehe auch

postgis.enable_outdb_rasters postgis.gdal_enabled_drivers

7.23 Funktionen zur Fehlersuche

7.23.1 PostGIS_AddBBox

PostGIS_AddBBox — Fügt der Geometrie ein umschreibendes Rechteck bei.

Synopsis

geometry **PostGIS_AddBBox**(geometry geomA);

Beschreibung

Fügt der Geometrie ein umschreibendes Rechteck bei. Dies beschleunigt Abfragen, die sich auf die umschreibenden Rechtecke beziehen, erhöht aber die Größe der Geometrie.



Anmerkung
Die umschreibenden Rechtecke werden üblicherweise automatisch der Geometrie beigefügt, so dass es nicht nötig ist dies händisch zu tun, außer das umschreibende Rechteck wurde irgendwie beschädigt, oder Sie verwenden eine alte Installation bei der die umschreibenden Rechtecke noch fehlten. Falls dies der Fall ist, müssen Sie die alten löschen und neu hinzufügen.

 Diese Methode unterstützt kreisförmige Strings und Kurven.

Beispiele

Code

```
UPDATE sometable
SET geom =  PostGIS_AddBBox(geom)
WHERE PostGIS_HasBBox(geom) = false;
```

Siehe auch

PostGIS_DropBBox, PostGIS_HasBBox

7.23.2 PostGIS_DropBBox

PostGIS_DropBBox — Löscht die umschreibenden Rechtecke der Geometrie.

Synopsis

geometry **PostGIS_DropBBox**(geometry geomA);

Beschreibung

Löscht die umschreibenden Rechtecke der Geometrie. Dies verringert die Größe der Geometrie, macht aber die auf umschreibenden Rechtecken aufbauenden Abfragen langsamer. Wird auch zum Löschen eines beschädigten umschreibenden Rechtecks benutzt. Ein guter Hinweis auf ein beschädigtes umschreibendes Rechteck ist, wenn ST_Intersects und andere Abfragen über räumliche Beziehungen, Geometrien auslassen die eigentlich TRUE zurückgeben sollten.



Anmerkung
Umschreibende Rechtecke werden üblicherweise automatisch der Geometrie beigefügt, um die Geschwindigkeit von Abfragen zu steigern. Somit ist es nicht nötig ist dies händisch zu tun, außer das Umgebungsrechteck wurde irgendwie beschädigt, oder Sie verwenden eine alte Installation bei der die umschreibenden Rechtecke noch fehlten. . Falls dies der Fall ist, müssen Sie die alten löschen und neu hinzufügen. Diese Art der Beschädigung wurde in den Versionen 8.3-8.3.6 beobachtet. Der Grund war, dass die zwischengespeicherten umschreibenden Rechtecke nicht immer neu berechnet wurden, wenn sich die Geometrie geändert hat und das Upgraden auf eine neuere Version ohne einen PostgreSQL-Dump erfolgte. Dies kann händisch korrigiert werden, indem man die umschreibenden Rechtecke (BBox) wie unten angeführt neu hinzufügt, oder einen PostgreSQL-Dump einspielt.

 Diese Methode unterstützt kreisförmige Strings und Kurven.

Beispiele

This example drops cached bounding boxes where the stored box is wrong. Applying `Box2D` to `ST_AsBinary` forces a fresh box calculation, while `Box2D` on the table geometry reads the cached box.

Code

```
UPDATE sometable
SET geom =  PostGIS_DropBBox(geom)
WHERE Not (Box2D(ST_AsBinary(geom)) = Box2D(geom));
```

Output

```
UPDATE ...
```

After dropping corrupt cached boxes, re-add them where needed.

Code

```
UPDATE sometable
SET geom =  PostGIS_AddBBox(geom)
WHERE Not PostGIS_HasBBOX(geom);
```

Output

```
UPDATE ...
```

Siehe auch

PostGIS_AddBBox, PostGIS_HasBBox, Box2D

7.23.3 PostGIS_HasBBox

PostGIS_HasBBox — Gibt TRUE zurück, wenn die BBox der Geometrie zwischengespeichert ist, andernfalls wird FALSE zurückgegeben.

Synopsis

boolean **PostGIS_HasBBox**(geometry geomA);

Beschreibung

Gibt TRUE zurück, wenn die BBox der Geometrie zwischengespeichert ist, sonst FALSE. Benutzen Sie bitte PostGIS_AddBBox und PostGIS_DropBBox um das Zwischenspeichern zu steuern.

 Diese Methode unterstützt kreisförmige Strings und Kurven.

Beispiele

Code

```
SELECT geom
FROM sometable WHERE PostGIS_HasBBox(geom) = false;
```

Siehe auch

PostGIS_AddBBox, PostGIS_DropBBox

Kapitel 8

Referenz der SFCGAL-Funktionen

SFCGAL ist eine C++-Wrapper-Bibliothek für CGAL, die erweiterte räumliche 2D- und 3D-Funktionen bietet. Aus Gründen der Robustheit haben die Geometriekoordinaten eine exakte rationale Zahlendarstellung.

Installationsanweisungen für die Bibliothek sind auf der SFCGAL-Homepage zu finden (http://www.sfcgal.org). Um die Funktionen zu aktivieren, verwenden Sie `Erstellen der Erweiterung postgis_sfcgal`.

8.1 Verwaltungsfunktionen der SFCGAL

8.1.1 postgis_sfcgal_version

postgis_sfcgal_version — Gibt die verwendete Version von SFCGAL zurück

Synopsis

text **postgis_sfcgal_version**(void);

Beschreibung

Gibt die verwendete Version von SFCGAL zurück

Verfügbarkeit: 2.1.0

 Diese Methode benötigt ein SFCGAL-Backend.

Siehe auch

postgis_sfcgal_full_version

8.1.2 postgis_sfcgal_full_version

postgis_sfcgal_full_version — Liefert die vollständige Version von SFCGAL, einschließlich der CGAL- und Boost-Versionen

Synopsis

text **postgis_sfcgal_full_version**(void);

Beschreibung

Liefert die vollständige Version von SFCGAL, einschließlich der CGAL- und Boost-Versionen

Verfügbarkeit: 3.3.0

Diese Methode benötigt ein SFCGAL-Backend.

Siehe auch

postgis_sfcgal_version

8.2 SFCGAL-Accessoren und -Setzer

8.2.1 CG_ForceLHR

CG_ForceLHR — LHR-Ausrichtung erzwingen

Synopsis

geometry **CG_ForceLHR**(geometry geom);

Beschreibung

Returns an equivalent geometry whose polygonal components follow the left-hand rule: exterior rings are counter-clockwise and interior rings are clockwise. Rings are reversed as needed; coordinates and dimensionality are preserved. The input must be valid.

Verfügbarkeit: 3.5.0

Diese Methode benötigt ein SFCGAL-Backend.

Diese Funktion unterstützt 3d und lässt den Z-Index nicht fallen.

Diese Funktion unterstützt polyedrische Flächen.

Diese Funktion unterstützt Dreiecke und dreieckige unregelmäßige Netzoberflächen (TIN).

Beispiele

Reorient a polygon and its hole to the left-hand rule.

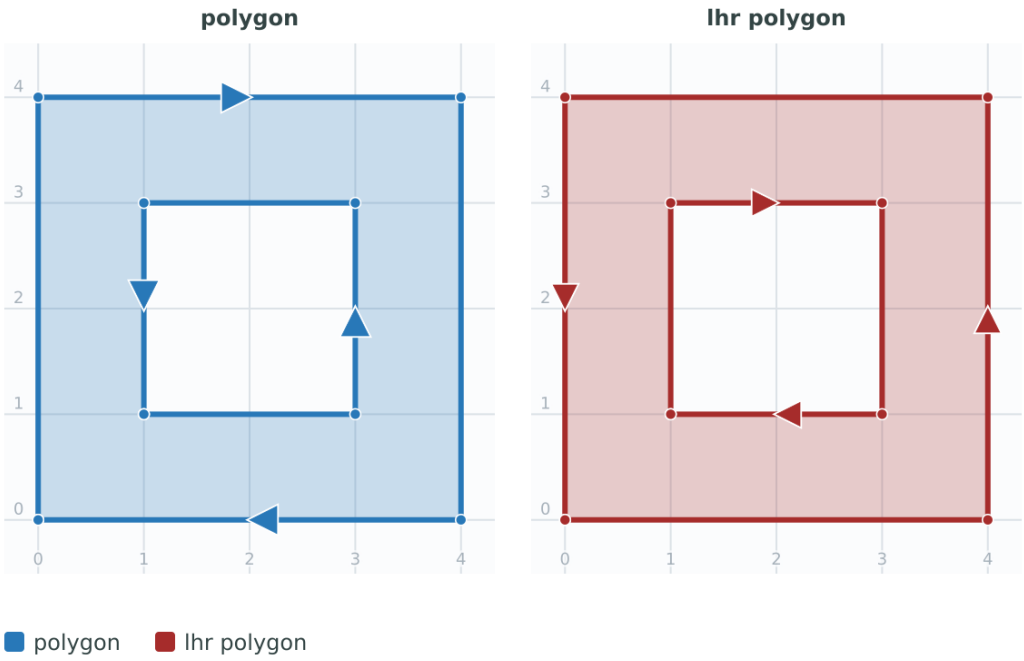
Code

```
WITH data AS (
  SELECT 'POLYGON((0 0,0 4,4 4,4 0,0 0),
                  (1 1,3 1,3 3,1 3,1 1))'::geometry AS geom
)
SELECT geom AS input_polygon,
       CG_ForceLHR(geom) AS lhr_polygon,
       ST_IsPolygonCCW(geom) AS input_is_lhr,
       ST_IsPolygonCCW(CG_ForceLHR(geom)) AS output_is_lhr
FROM data;
```

Output

```
POLYGON((0 0,0 4,4 4,4 0,0 0),(1 1,3 1,3 3,1 3,1 1)) | POLYGON((0 0,4 0,4 4,0 4,0 0),(1 1,1 3,3 3,3 1,1 1)) | f | t
```

Figure



8.2.2 CG_IsPlanar

CG_IsPlanar — Prüfen, ob eine Fläche planar ist oder nicht

Synopsis

boolean **CG_IsPlanar**(geometry geom);

Beschreibung

Tests whether the input geometry lies in a single plane in 3D.

Verfügbarkeit: 3.5.0

Diese Methode benötigt ein SFCGAL-Backend.

Diese Funktion unterstützt 3d und lässt den Z-Index nicht fallen.

Diese Funktion unterstützt polyedrische Flächen.

Diese Funktion unterstützt Dreiecke und dreieckige unregelmäßige Netzoberflächen (TIN).

8.2.3 CG_IsValid

CG_IsValid — Tests whether a geometry is valid according to SFCGAL

Synopsis

boolean **CG_IsValid**(geometry geom);

Beschreibung

Returns true if the input geometry is valid according to SFCGAL.

This differs from ST_IsValid, which checks OGC validity through GEOS.

Availability: 3.8.0

Diese Methode benötigt ein SFCGAL-Backend.

Diese Funktion unterstützt 3d und lässt den Z-Index nicht fallen.

Diese Funktion unterstützt polyedrische Flächen.

Diese Funktion unterstützt Dreiecke und dreieckige unregelmäßige Netzoberflächen (TIN).

Beispiele

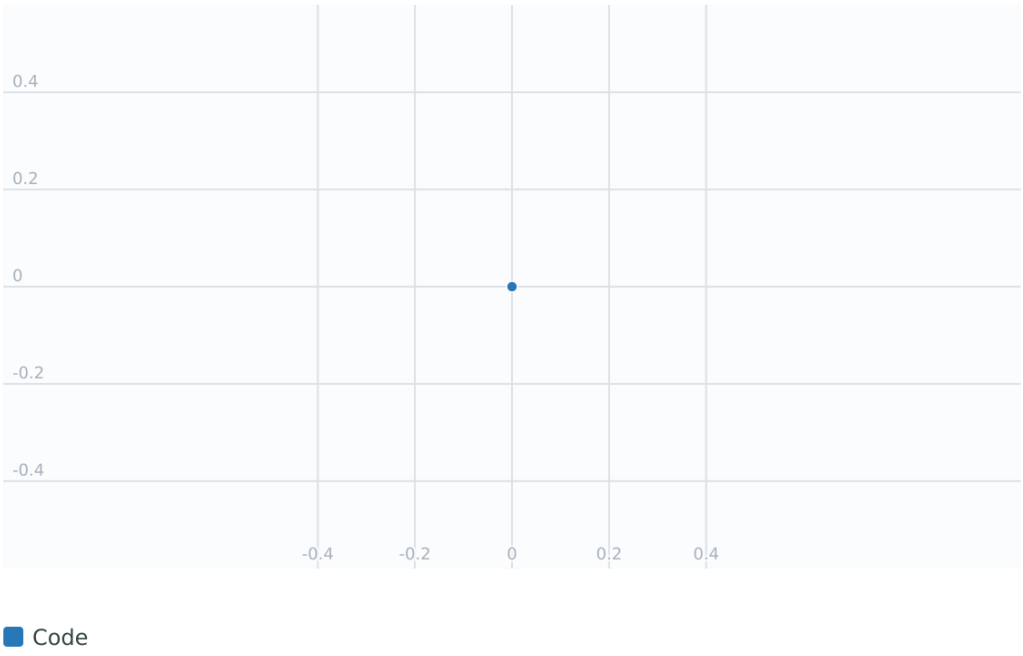
Code

```
SELECT CG_IsValid('LINESTRING Z (0 0 0,0 0 1)'::geometry);
```

Output

```
t
```

Figure



Siehe auch

CG_IsValidDetail, CG_IsSimple, ST_IsValid, CG_IsSolid

8.2.4 CG_IsValidDetail

CG_IsValidDetail — Returns a `valid_detail` row stating whether a geometry is valid according to SFCGAL, and if not a reason and a location.

Synopsis

valid_detail **CG_IsValidDetail**(geometry geom);

Beschreibung

Returns a `valid_detail` row with a boolean validity flag, a reason for invalidity, and the invalidity location when SFCGAL reports one.

This differs from ST_IsValidDetail, which checks OGC validity through GEOS.

Availability: 3.8.0

Diese Methode benötigt ein SFCGAL-Backend.

Diese Funktion unterstützt 3d und lässt den Z-Index nicht fallen.

Diese Funktion unterstützt polyedrische Flächen.

Diese Funktion unterstützt Dreiecke und dreieckige unregelmäßige Netzoberflächen (TIN).

Beispiele

Code

```
SELECT valid, reason, ST_AsText(location) AS location
FROM CG_IsValidDetail('POLYGON((0 0,2 2,2 0,0 2,0 0))'::geometry);
```

Output

```
f | ring 0 self intersects |
```

Siehe auch

CG_IsValid, CG_IsSimpleDetail, ST_IsValidDetail

8.2.5 CG_IsSimple

CG_IsSimple — Tests whether a geometry is simple according to SFCGAL

Synopsis

boolean **CG_IsSimple**(geometry geom);

Beschreibung

Returns true if the input geometry is simple according to SFCGAL.

This differs from ST_IsSimple, which checks simplicity through liblwgeom.

Availability: 3.8.0 - requires SFCGAL >= 2.3.0.

Diese Methode benötigt ein SFCGAL-Backend.

Diese Funktion unterstützt 3d und lässt den Z-Index nicht fallen.

Diese Funktion unterstützt polyedrische Flächen.

Diese Funktion unterstützt Dreiecke und dreieckige unregelmäßige Netzoberflächen (TIN).

Beispiele

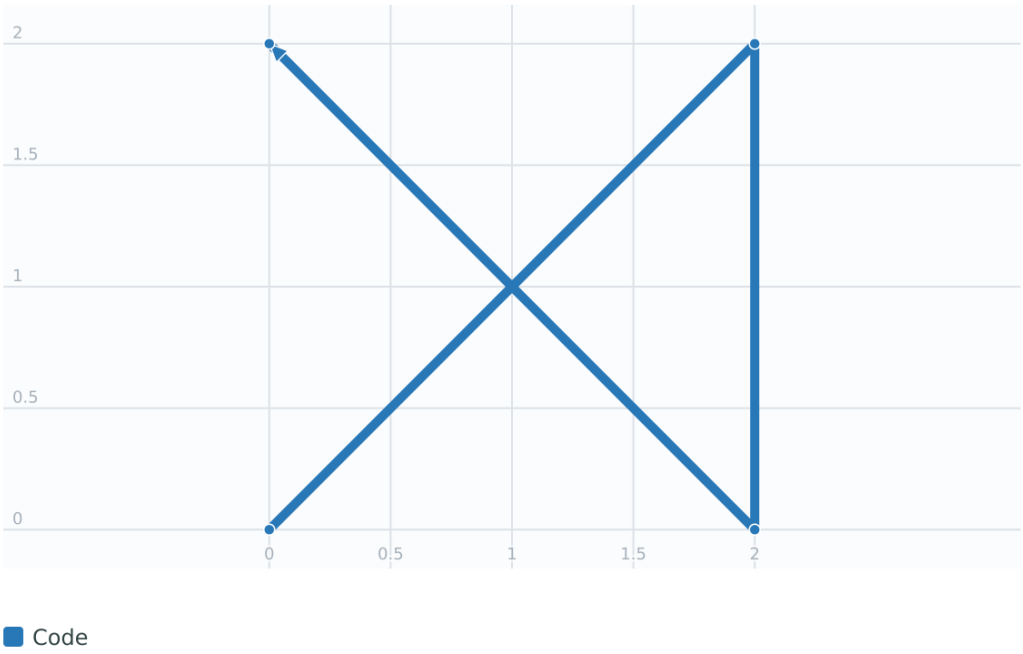
Code

```
SELECT CG_IsSimple('LINESTRING(0 0,2 2,2 0,0 2)'::geometry);
```

Output

```
f
```

Figure



Siehe auch

CG_IsSimpleDetail, CG_IsValid, ST_IsSimple

8.2.6 CG_IsSimpleDetail

CG_IsSimpleDetail — Returns a reason when a geometry is not simple according to SFCGAL

Synopsis

text **CG_IsSimpleDetail**(geometry geom);

Beschreibung

Returns SFCGAL's reason when the input geometry is not simple. Returns `NULL` for simple geometries.

Availability: 3.8.0 - requires SFCGAL >= 2.3.0.

Diese Methode benötigt ein SFCGAL-Backend.

Diese Funktion unterstützt 3d und lässt den Z-Index nicht fallen.

Diese Funktion unterstützt polyedrische Flächen.

Diese Funktion unterstützt Dreiecke und dreieckige unregelmäßige Netzoberflächen (TIN).

Beispiele

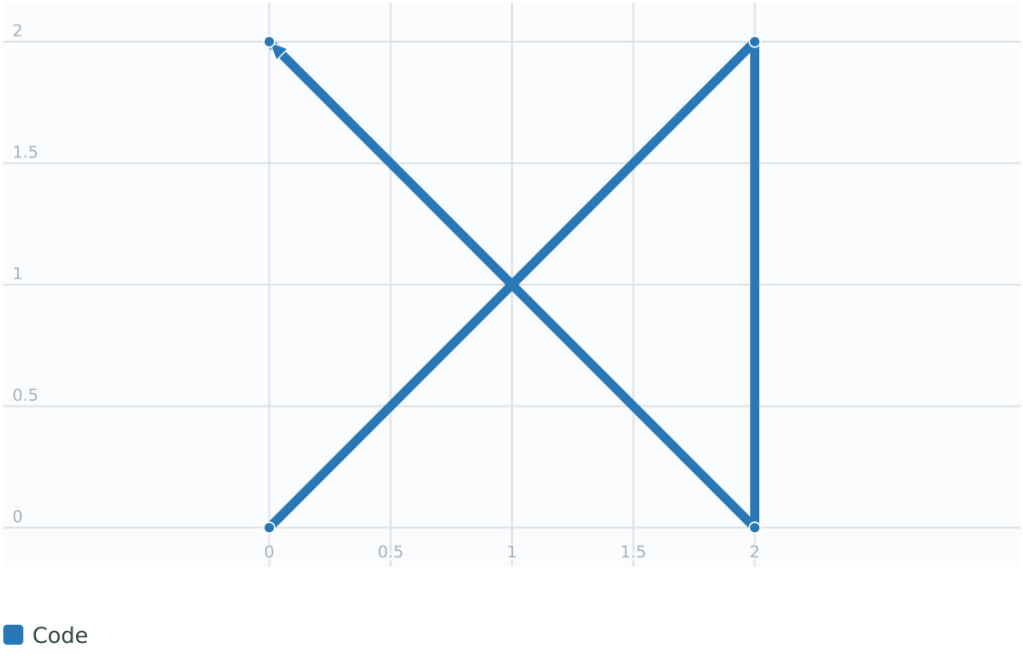
Code

```
SELECT CG_IsSimpleDetail('LINESTRING(0 0,2 2,2 0,0 2)'::geometry);
```

Output

```
linestring self intersects
```

Figure



Siehe auch

CG_IsSimple, CG_IsValidDetail, ST_IsSimple

8.2.7 CG_IsSolid

CG_IsSolid — Prüfen, ob die Geometrie ein Solid ist. Es wird keine Gültigkeitsprüfung durchgeführt.

Synopsis

boolean **CG_IsSolid**(geometry geom1);

Beschreibung

Returns `true` when the input carries the solid flag, such as a geometry produced by CG_MakeSolid. A closed `PolyhedralSurface` remains a surface until it is explicitly converted. This tests the representation, not shell closure or validity; no validity check is performed.

Verfügbarkeit: 3.5.0

Diese Methode benötigt ein SFCGAL-Backend.

Diese Funktion unterstützt 3d und lässt den Z-Index nicht fallen.

Diese Funktion unterstützt polyedrische Flächen.

Diese Funktion unterstützt Dreiecke und dreieckige unregelmäßige Netzoberflächen (TIN).

Beispiele

Compare the same closed shell before and after converting it to a solid.

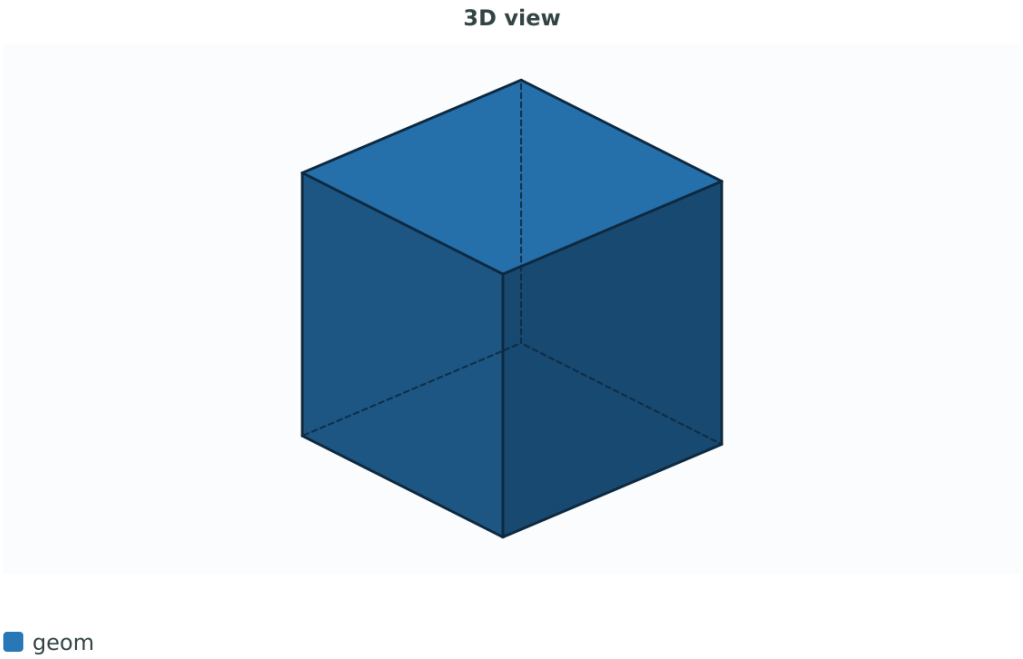
Code

```
WITH data AS (
  SELECT 'POLYHEDRALSURFACE Z (
    ((0 0 0,0 0 1,0 1 1,0 1 0,0 0 0)),
    ((0 0 0,0 1 0,1 1 0,1 0 0,0 0 0)),
    ((0 0 0,1 0 0,1 0 1,0 0 1,0 0 0)),
    ((1 1 0,1 1 1,1 0 1,1 0 0,1 1 0)),
    ((0 1 0,0 1 1,1 1 1,1 1 0,0 1 0)),
    ((0 0 1,1 0 1,1 1 1,0 1 1,0 0 1))
  )'::geometry AS geom
)
SELECT CG_IsSolid(geom) AS shell_is_solid,
       CG_IsSolid(CG_MakeSolid(geom)) AS solid_is_solid
FROM data;
```

Output

```
f | t
```

Figure



8.2.8 CG_MakeSolid

CG_MakeSolid — Gießen Sie die Geometrie in einen Körper. Es wird keine Prüfung durchgeführt. Um ein gültiges Solid zu erhalten, muss die Eingabegeometrie eine geschlossene polyedrische Fläche oder ein geschlossenes TIN sein.

Synopsis

geometry **CG_MakeSolid**(geometry geom1);

Beschreibung

Marks a PolyhedralSurface or TIN as a solid. The input is expected to describe a closed shell; the function sets the solid representation and does not validate the shell.

Verfügbarkeit: 3.5.0

Diese Methode benötigt ein SFCGAL-Backend.

Diese Funktion unterstützt 3d und lässt den Z-Index nicht fallen.

Diese Funktion unterstützt polyedrische Flächen.

Diese Funktion unterstützt Dreiecke und dreieckige unregelmäßige Netzoberflächen (TIN).

8.2.9 CG_Orientation

CG_Orientation — Bestimmung der Oberflächenausrichtung

Synopsis

integer **CG_Orientation**(geometry geom);

Beschreibung

Die Funktion gilt nur für Polygone. Sie gibt -1 zurück, wenn das Polygon gegen den Uhrzeigersinn orientiert ist, und 1, wenn das Polygon im Uhrzeigersinn orientiert ist.

Verfügbarkeit: 3.5.0

 Diese Methode benötigt ein SFCGAL-Backend.

 Diese Funktion unterstützt 3d und lässt den Z-Index nicht fallen.

8.2.10 CG_Area

CG_Area — Calculates the area of a geometry

Synopsis

double precision **CG_Area**(geometry geom);

Beschreibung

Calculates the area of a geometry.

Performed by the SFCGAL module



Anmerkung
NOTE: this function returns a double precision value representing the area.

Verfügbarkeit: 3.5.0

 Diese Methode benötigt ein SFCGAL-Backend.

Beispiele mit dem geometrischen Datentyp

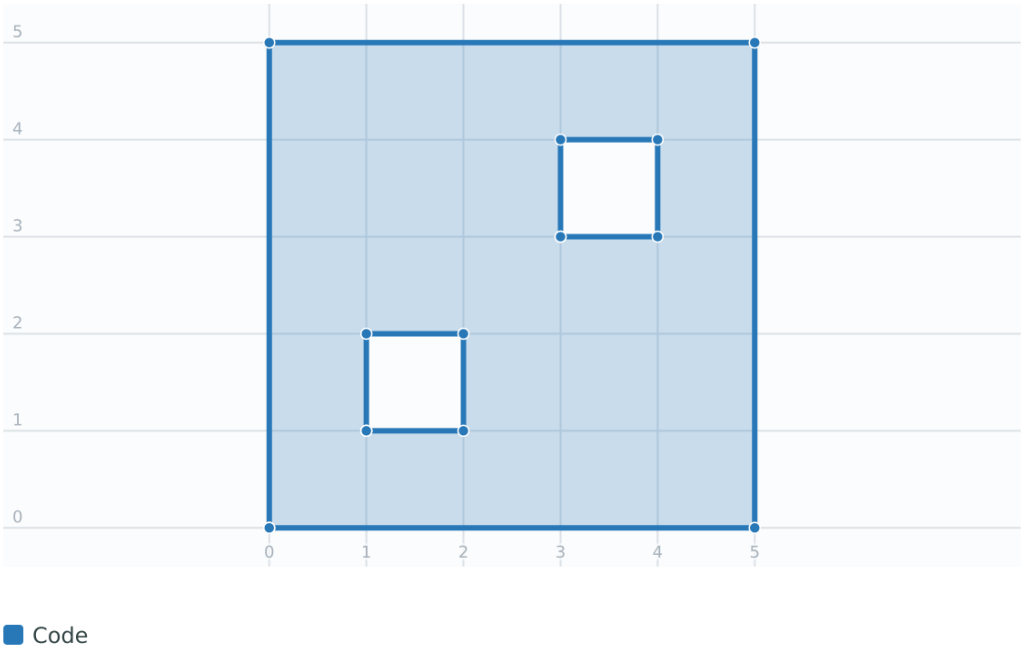
Code

```
SELECT CG_Area('Polygon ((0 0,0 5,5 5,5 0,0 0),(1 1,2 1,2 2,1 2,1 1),(3 3,4 3,4 4,3 4,3 3))' ↩
    );
```

Output

```
23
```

Figure



Siehe auch

CG_3DArea, ST_Area

8.2.11 CG_3DArea

CG_3DArea — Berechnet die Fläche von 3D-Oberflächengeometrien. Gibt 0 für Solids zurück.

Synopsis

float**CG_3DArea**(geometry geom1);

Beschreibung

Verfügbarkeit: 3.5.0

Diese Methode benötigt ein SFCGAL-Backend.

Diese Methode setzt die SQL/MM-Spezifikation um. SQL-MM IEC 13249-3: 8.1, 10.5

Diese Funktion unterstützt 3d und lässt den Z-Index nicht fallen.

Diese Funktion unterstützt polyedrische Flächen.

Diese Funktion unterstützt Dreiecke und dreieckige unregelmäßige Netzoberflächen (TIN).

Beispiele

Hinweis: Standardmäßig ist eine aus WKT erstellte PolyhedralSurface eine Flächengeometrie, kein Solid. Sie hat daher einen Oberflächenbereich. Nach der Umwandlung in einen Festkörper gibt es keine Fläche mehr.

Code

```
SELECT CG_3DArea(geom) As cube_surface_area,
CG_3DArea(CG_MakeSolid(geom)) As solid_surface_area
FROM (SELECT 'POLYHEDRALSURFACE( ((0 0 0,0 0 1,0 1 1,0 1 0,0 0 0)),
((0 0 0,0 1 0,1 1 0,1 0 0,0 0 0)),
((0 0 0,1 0 0,1 0 1,0 0 1,0 0 0)),
((1 1 0,1 1 1,1 0 1,1 0 0,1 1 0)),
((0 1 0,0 1 1,1 1 1,1 1 0,0 1 0)),
((0 0 1,1 0 1,1 1 1,0 1 1,0 0 1)) )'::geometry) As f(geom);
```

Output

```
cube_surface_area | solid_surface_area
------------------+--------------------
6                 | 0
```

Siehe auch

CG_Area, CG_MakeSolid, CG_IsSolid, CG_Area

8.2.12 CG_Volume

CG_Volume — Computes the volume of a 3D geometry. Closed surfaces can have non-zero volume.

Synopsis

float **CG_Volume**(geometry geom1);

Beschreibung

Verfügbarkeit: 3.5.0

Diese Methode benötigt ein SFCGAL-Backend.

Diese Funktion unterstützt 3d und lässt den Z-Index nicht fallen.

Diese Funktion unterstützt polyedrische Flächen.

Diese Funktion unterstützt Dreiecke und dreieckige unregelmäßige Netzoberflächen (TIN).

Diese Methode setzt die SQL/MM-Spezifikation um. SQL-MM IEC 13249-3: 9.1 (same as CG_3DVolume)

Closed surface geometries can have non-zero volume. Use CG_MakeSolid when the geometry value itself must carry the solid flag.

Beispiel

A closed polyhedral surface has volume. CG_MakeSolid makes the solid representation explicit.

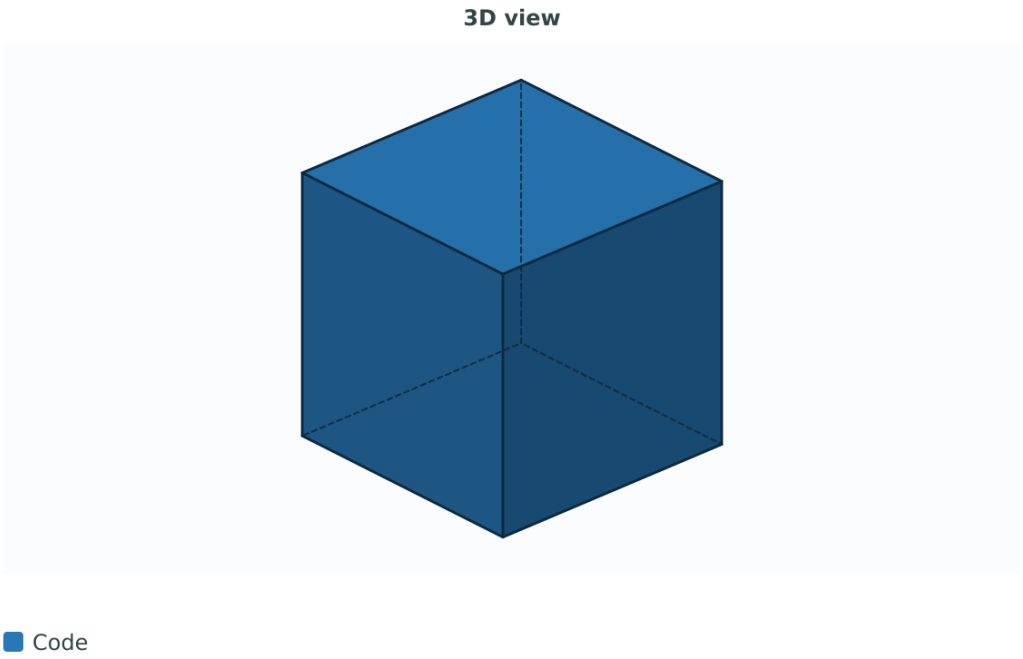
Code

```
SELECT CG_Volume(geom) As cube_surface_vol,
CG_Volume(CG_MakeSolid(geom)) As solid_surface_vol
FROM (SELECT 'POLYHEDRALSURFACE( ((0 0 0,0 0 1,0 1 1,0 1 0,0 0 0)),
((0 0 0,0 1 0,1 1 0,1 0 0,0 0 0)),
((0 0 0,1 0 0,1 0 1,0 0 1,0 0 0)),
((1 1 0,1 1 1,1 0 1,1 0 0,1 1 0)),
((0 1 0,0 1 1,1 1 1,1 1 0,0 1 0)),
((0 0 1,1 0 1,1 1 1,0 1 1,0 0 1)) )'::geometry) As f(geom);
```

Output

```
cube_surface_vol | solid_surface_vol
-----------------+------------------
1                | 1
```

Figure



Siehe auch

CG_3DArea, CG_MakeSolid, CG_IsSolid

8.3 SFCGAL-Verarbeitung und Beziehungsfunktionen

8.3.1 CG_Intersection

CG_Intersection — Computes the intersection of two geometries

Synopsis

geometry **CG_Intersection**(geometry geomA , geometry geomB);

Beschreibung

Computes the intersection of two geometries.

Performed by the SFCGAL module



Anmerkung
NOTE: this function returns a geometry representing the intersection.

Verfügbarkeit: 3.5.0

 Diese Methode benötigt ein SFCGAL-Backend.

Beispiele mit dem geometrischen Datentyp

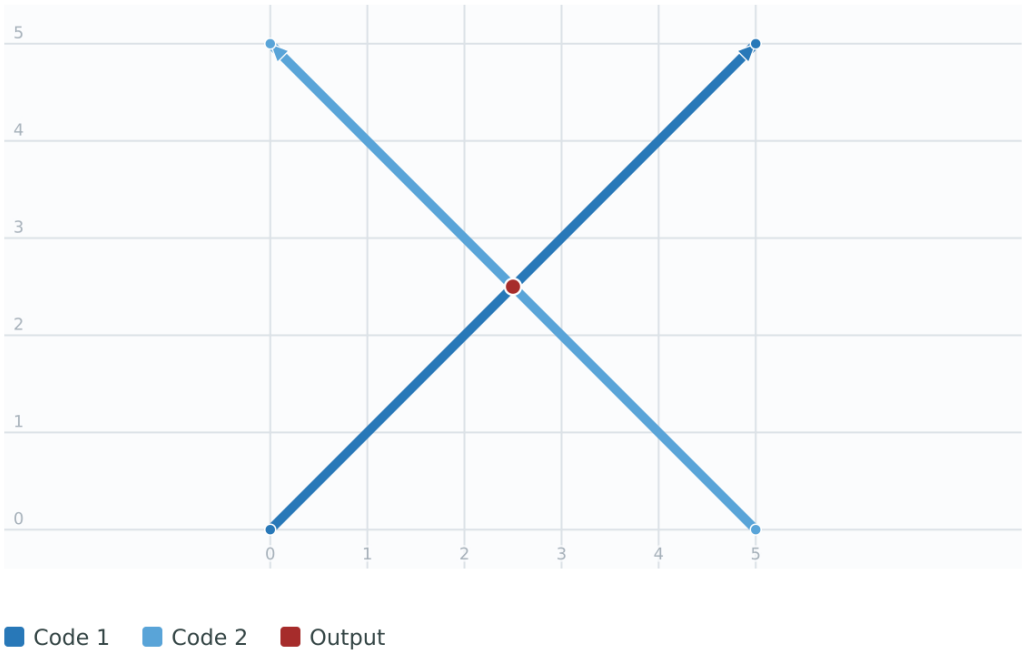
Code

```
SELECT CG_Intersection('LINESTRING(0 0,5 5)', 'LINESTRING(5 0,0 5)');
```

Output

```
POINT(2.5 2.5)
```

Figure



Siehe auch

CG_3DIntersection, ST_Intersection

8.3.2 CG_Intersects

CG_Intersects — Prüft, ob sich zwei Geometrien schneiden (sie haben mindestens einen Punkt gemeinsam)

Synopsis

boolean **CG_Intersects**(geometry geomA , geometry geomB);

Beschreibung

Gibt `true` zurück, wenn sich zwei Geometrien überschneiden. Geometrien überschneiden sich, wenn sie einen Punkt gemeinsam haben.

Performed by the SFCGAL module



Anmerkung
HINWEIS: Dies ist die "zulässige" Version, die einen booleschen Wert und keine ganze Zahl zurückgibt.

Verfügbarkeit: 3.5.0

 Diese Methode benötigt ein SFCGAL-Backend.

 Diese Funktion unterstützt Dreiecke und dreieckige unregelmäßige Netzoberflächen (TIN).

Beispiele mit dem geometrischen Datentyp

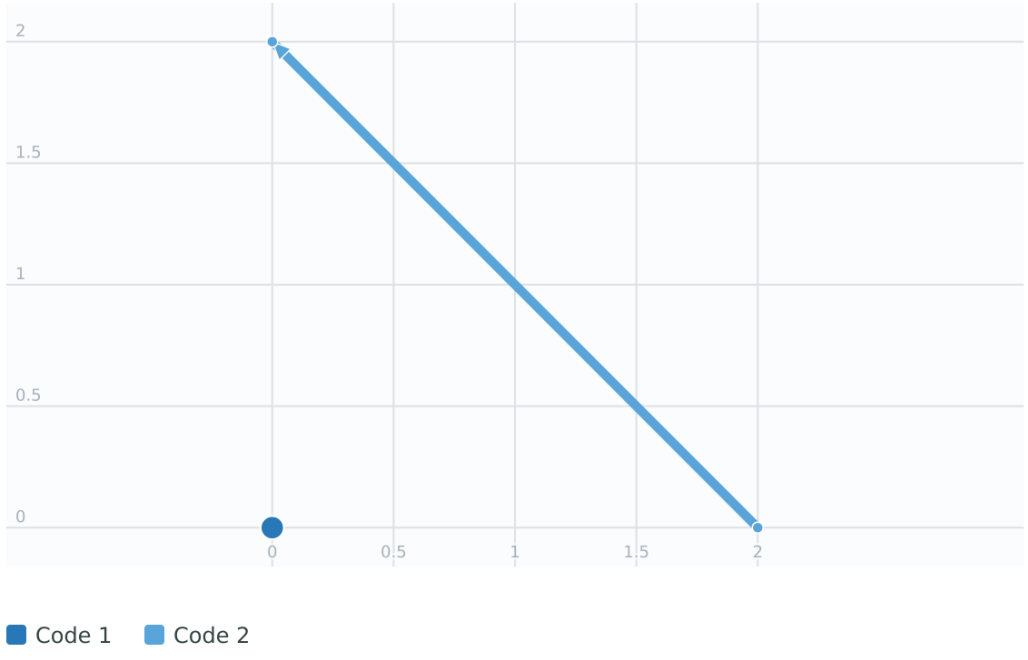
Code

```
SELECT CG_Intersects('POINT(0 0)'::geometry, 'LINESTRING ( 2 0,0 2 )'::geometry);
```

Output

```
f
```

Figure



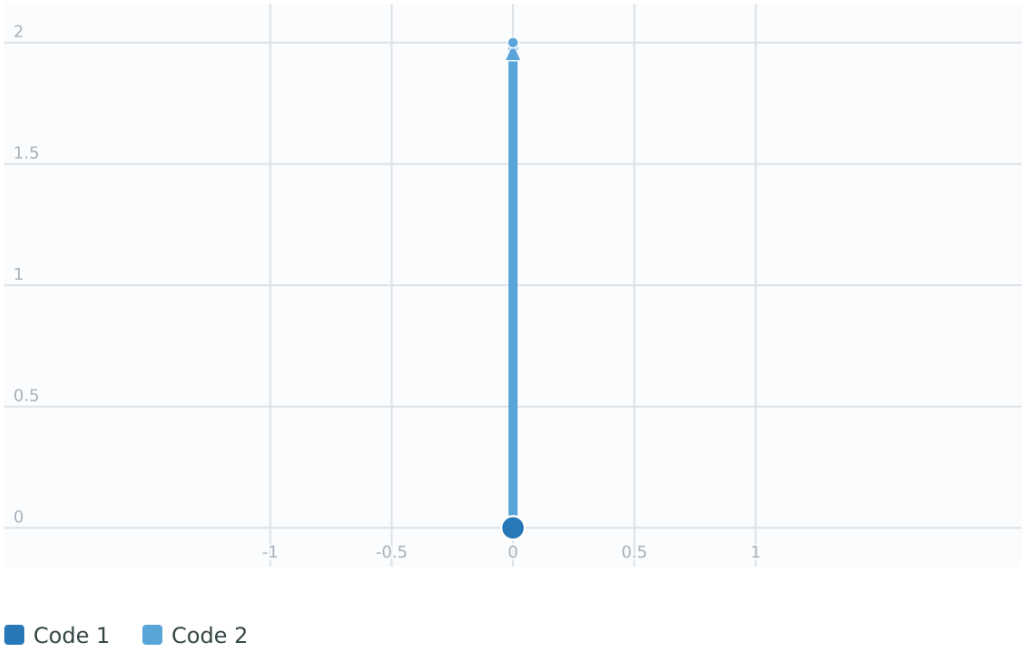
Code

```
SELECT CG_Intersects('POINT(0 0)'::geometry, 'LINESTRING ( 0 0,0 2 )'::geometry);
```

Output

```
t
```

Figure



Siehe auch

CG_3DIntersects, ST_3DIntersects, ST_Intersects, ST_Disjoint

8.3.3 CG_3DIntersects

CG_3DIntersects — Tests if two 3D geometries intersect

Synopsis

boolean **CG_3DIntersects**(geometry geomA , geometry geomB);

Beschreibung

Tests if two 3D geometries intersect. 3D geometries intersect if they have any point in common in the three-dimensional space.

Performed by the SFCGAL module



Anmerkung
HINWEIS: Dies ist die "zulässige" Version, die einen booleschen Wert und keine ganze Zahl zurückgibt.

Verfügbarkeit: 3.5.0

Diese Methode benötigt ein SFCGAL-Backend.

Diese Funktion unterstützt Dreiecke und dreieckige unregelmäßige Netzoberflächen (TIN).

Beispiele mit dem geometrischen Datentyp

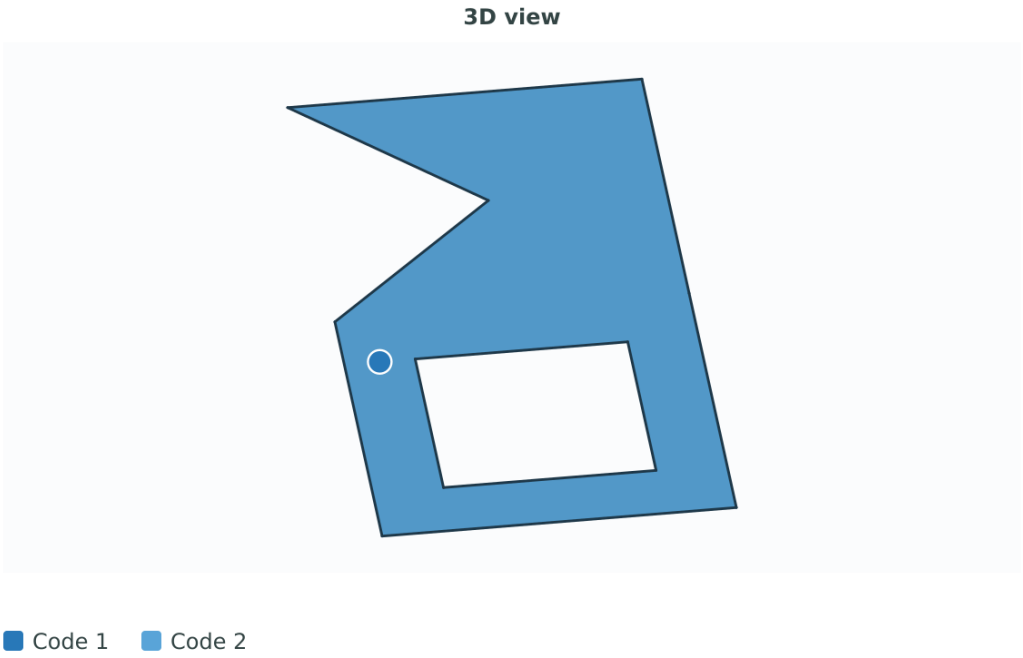
Code

```
SELECT CG_3DIntersects('POINT(1.2 0.1 0)', 'POLYHEDRALSURFACE(((0 0 0,0.5 0.5 0,1 0 0,1 1 0,0 1 0,0 0 0)),((1 0 0,2 0 0,2 1 0,1 1 0,1 0 0),(1.2 0.2 0,1.2 0.8 0,1.8 0.8 0,1.8 0.2 0,1.2 0.2 0)))');
```

Output

```
t
```

Figure



Siehe auch

CG_Intersects, ST_3DIntersects, ST_Intersects, ST_Disjoint

8.3.4 CG_Difference

CG_Difference — Computes the geometric difference between two geometries

Synopsis

geometry **CG_Difference**(geometry geomA , geometry geomB);

Beschreibung

Computes the geometric difference between two geometries. The resulting geometry is a set of points that are present in geomA but not in geomB.

Performed by the SFCGAL module



Anmerkung
NOTE: this function returns a geometry.

Verfügbarkeit: 3.5.0

Diese Methode benötigt ein SFCGAL-Backend.

Diese Funktion unterstützt Dreiecke und dreieckige unregelmäßige Netzoberflächen (TIN).

Beispiele mit dem geometrischen Datentyp

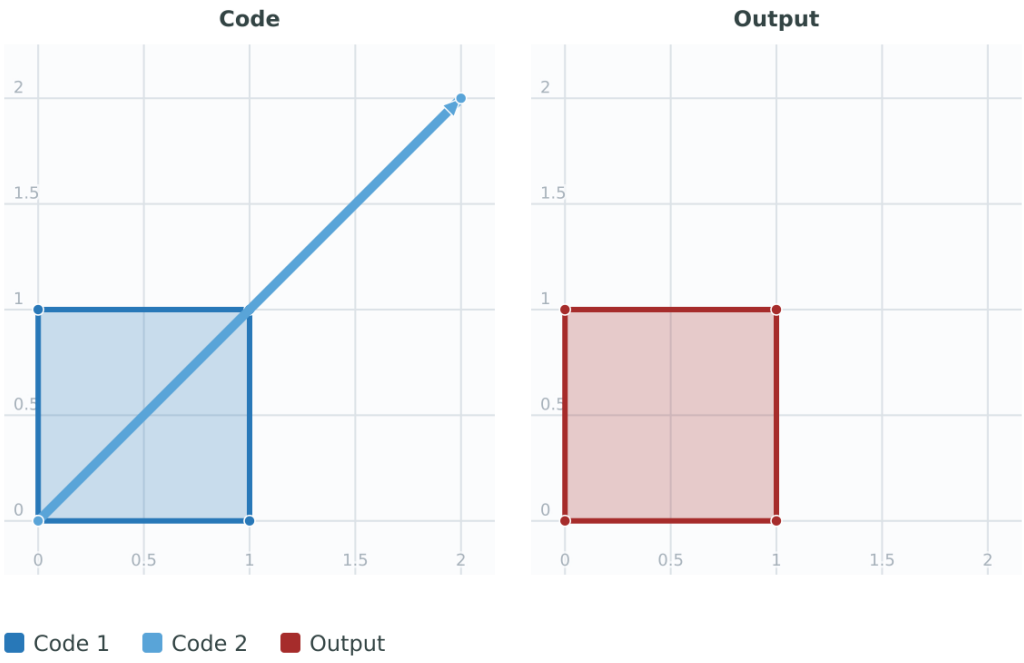
Code

```
SELECT CG_Difference('POLYGON((0 0,0 1,1 1,1 0,0 0))'::geometry, 'LINESTRING(0 0,2 2)':: ↩
    geometry);
```

Output

```
POLYGON((0 0,1 0,1 1,0 1,0 0))
```

Figure



Siehe auch

CG_3DDifference, ST_Difference

8.3.5 CG_3DDifference

CG_3DDifference — 3D-Differenz durchführen

Synopsis

geometry **CG_3DDifference**(geometry geom1, geometry geom2);

Beschreibung

Gibt den geometrischen Schwerpunkt einer Geometrie zurück.

Verfügbarkeit: 3.5.0

Diese Methode benötigt ein SFCGAL-Backend.

Diese Methode setzt die SQL/MM-Spezifikation um. SQL-MM IEC 13249-3: 5.1

Diese Funktion unterstützt 3d und lässt den Z-Index nicht fallen.

Diese Funktion unterstützt polyedrische Flächen.

Diese Funktion unterstützt Dreiecke und dreieckige unregelmäßige Netzoberflächen (TIN).

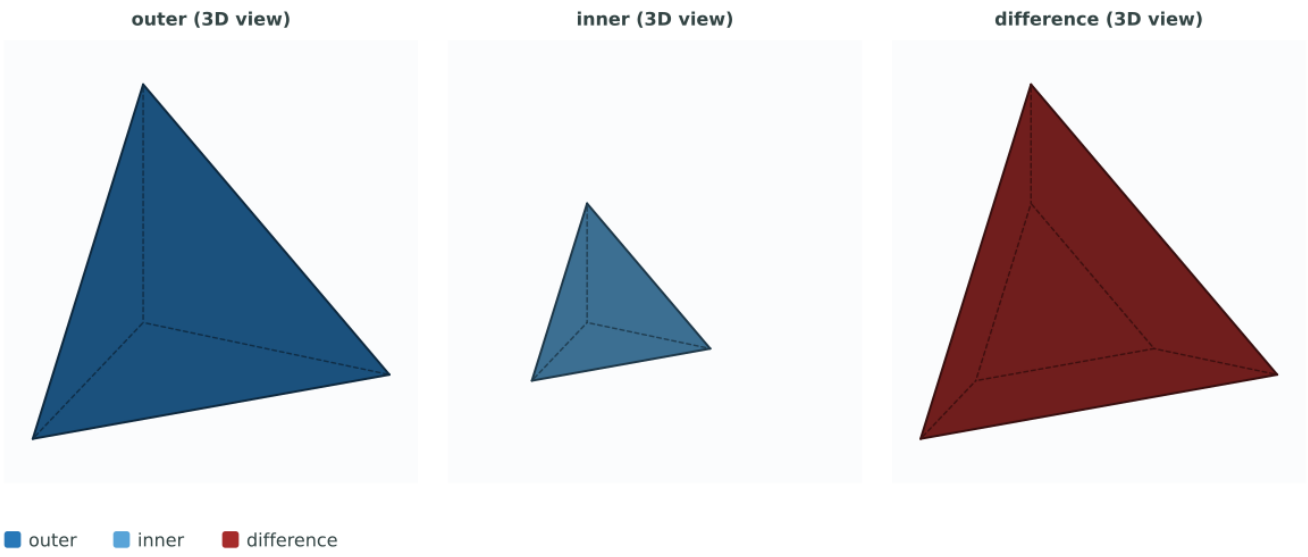
Beispiele

Subtract a smaller tetrahedral solid and render the two inputs and the actual remaining solid separately.

Code

```
WITH data AS (
  SELECT CG_MakeSolid('POLYHEDRALSURFACE Z (
    ((0 0 0,0 4 0,4 0 0,0 0 0)),
    ((0 0 0,4 0 0,0 0 4,0 0 0)),
    ((0 0 0,0 0 4,0 4 0,0 0 0)),
    ((4 0 0,0 4 0,0 0 4,4 0 0))
  )'::geometry) AS outer_solid
), solids AS (
  SELECT outer_solid,
         ST_Scale(outer_solid, 0.5, 0.5, 0.5) AS inner_solid
  FROM data
)
SELECT outer_solid AS input_outer,
       inner_solid AS input_inner,
       CG_3DDifference(outer_solid, inner_solid) AS difference
FROM solids;
```

Figure



Siehe auch

CG_Extrude, ST_AsX3D, CG_3DIntersection CG_3DUnion

8.3.6 CG_Distance

CG_Distance — Computes the minimum distance between two geometries

Synopsis

double precision **CG_Distance**(geometry geomA , geometry geomB);

Beschreibung

Computes the minimum distance between two geometries.

Performed by the SFCGAL module

Anmerkung
NOTE: this function returns a double precision value representing the distance.

Verfügbarkeit: 3.5.0

Diese Methode benötigt ein SFCGAL-Backend.

Diese Funktion unterstützt Dreiecke und dreieckige unregelmäßige Netzoberflächen (TIN).

Beispiele mit dem geometrischen Datentyp

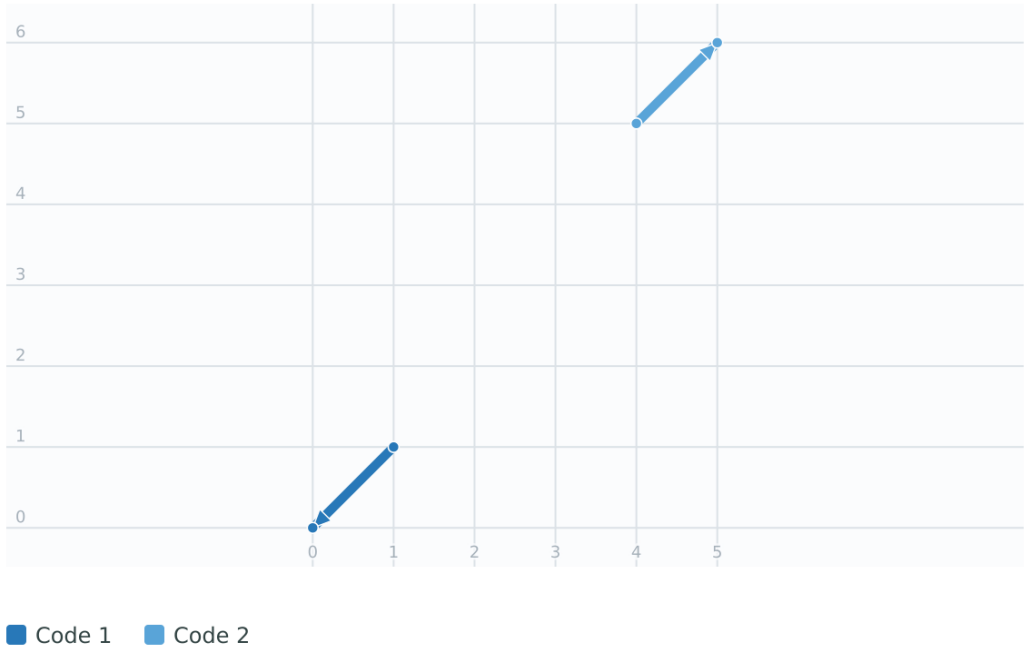
Code

```
SELECT CG_Distance('LINESTRING(0.0 0.0,-1.0 -1.0)', 'LINESTRING(3.0 4.0,4.0 5.0)');
```

Output

```
5
```

Figure



Siehe auch

CG_3DDistance, CG_Distance

8.3.7 CG_3DDistance

CG_3DDistance — Computes the minimum 3D distance between two geometries

Synopsis

double precision **CG_3DDistance**(geometry geomA , geometry geomB);

Beschreibung

Computes the minimum 3D distance between two geometries.

Performed by the SFCGAL module



Anmerkung
NOTE: this function returns a double precision value representing the 3D distance.

Verfügbarkeit: 3.5.0

Diese Methode benötigt ein SFCGAL-Backend.

Diese Funktion unterstützt Dreiecke und dreieckige unregelmäßige Netzoberflächen (TIN).

Beispiele mit dem geometrischen Datentyp

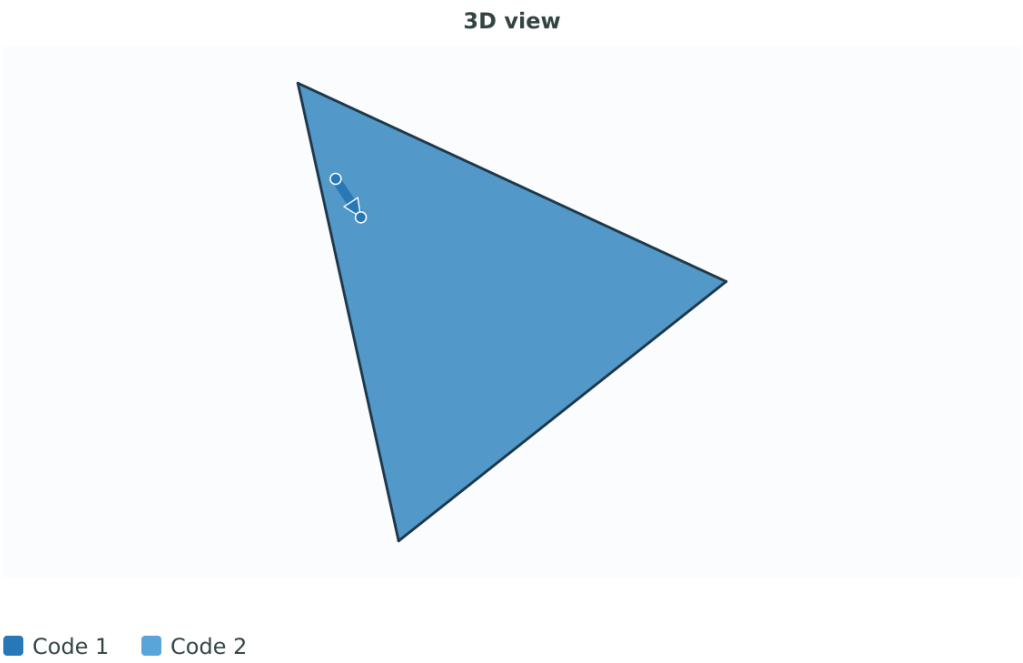
Code

```
SELECT CG_3DDistance('LINESTRING(-1.0 0.0 2.0,1.0 0.0 3.0)', 'TRIANGLE((-4.0 0.0 1.0,4.0 0.0 ↩
    1.0,0.0 4.0 1.0,-4.0 0.0 1.0))');
```

Output

```
1
```

Figure



Siehe auch

CG_Distance, ST_3DDistance

8.3.8 CG_3DConvexHull

CG_3DConvexHull — Berechnet die konvexe Hülle einer Geometrie.

Synopsis

geometry **CG_3DConvexHull**(geometry geom1);

Beschreibung

Computes the smallest convex three-dimensional geometry that contains the input. Unlike ST_ConvexHull, this function uses Z coordinates when constructing the hull.

Verfügbarkeit: 3.5.0

Diese Methode benötigt ein SFCGAL-Backend.

Diese Funktion unterstützt 3d und lässt den Z-Index nicht fallen.

Diese Funktion unterstützt polyedrische Flächen.

Diese Funktion unterstützt Dreiecke und dreieckige unregelmäßige Netzoberflächen (TIN).

Beispiele

Compute the 3D convex hull of a 3D LineString.

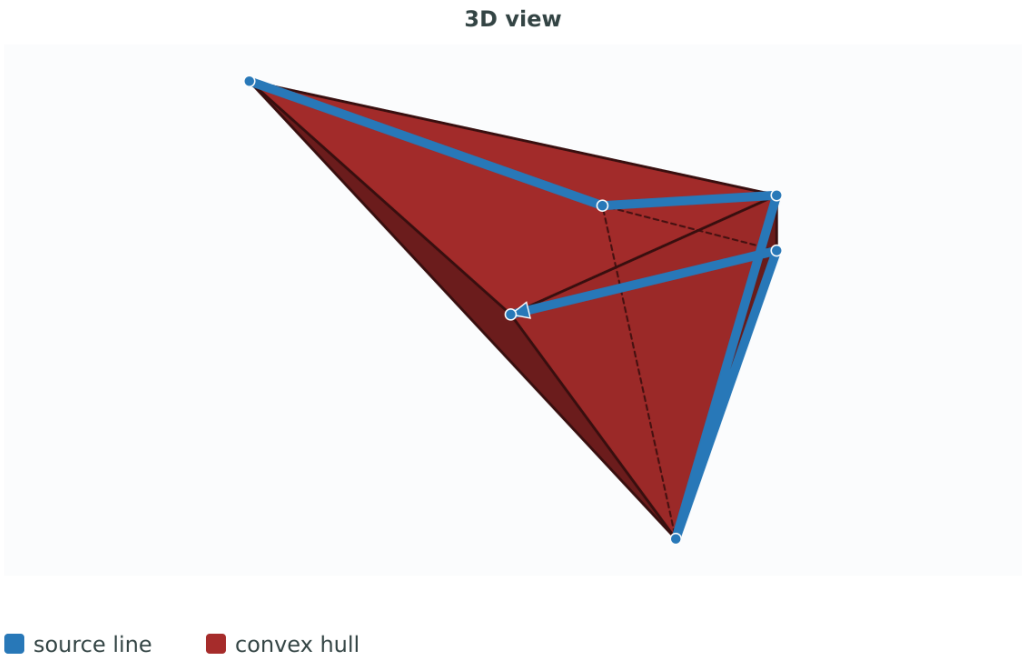
Code

```
WITH data AS (
  SELECT 'LINESTRING Z(0 0 5,1 5 3,5 7 6,9 5 3,5 7 5,6 3 5)'::geometry AS source_line
)
SELECT CG_3DConvexHull(source_line) AS convex_hull
FROM data;
```

Output

```
POLYHEDRALSURFACE Z (((1 5 3,9 5 3,0 0 5,1 5 3)),((1 5 3,0 0 5,5 7 6,1 5 3)),((5 7 6,5 7 5,1 5 3,5 7 6)),((0 0 5,6 3 ↩
    5,5 7 6,0 0 5)),((6 3 5,9 5 3,5 7 6,6 3 5)),((0 0 5,9 5 3,6 3 5,0 0 5)),((9 5 3,5 7 5,5 7 6,9 5 3)),((1 5 3,5 7 ↩
    5,9 5 3,1 5 3)))
```

Figure



Compare the extruded source geometry with its 3D convex hull.

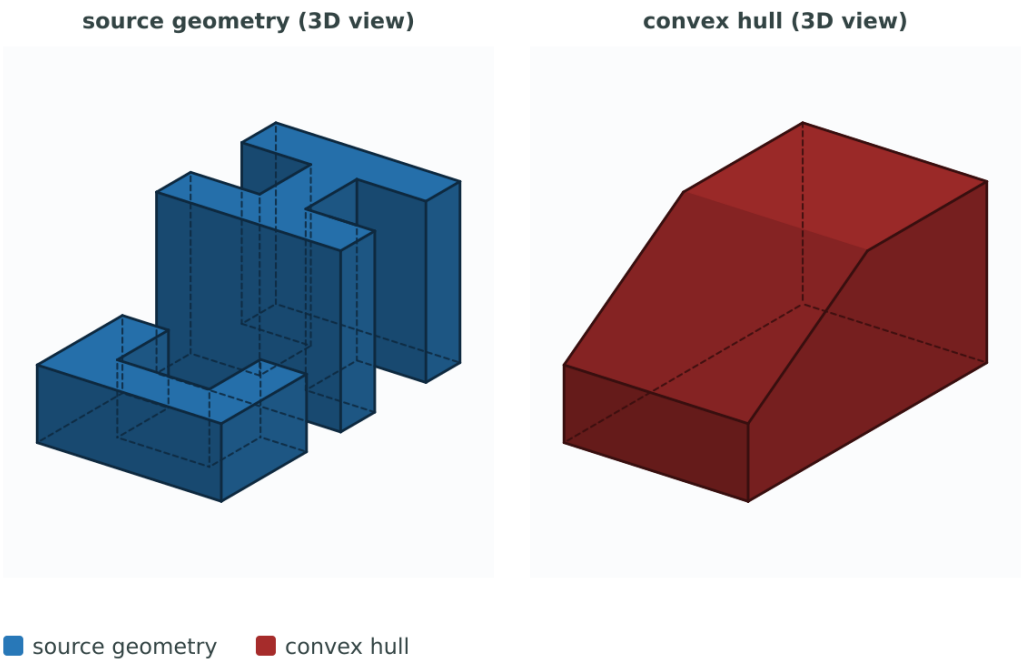
Code

```
WITH footprints(label, geom, height) AS (VALUES
  ('C', 'POLYGON((0 0,0 8,5 8,5 6,2 6,2 2,5 2,5 0,0 0))'::geometry, 3),
  ('H', 'POLYGON((7 0,9 0,9 3,12 3,12 0,14 0,14 8,12 8,12 5,9 5,9 8,7 8,7 0))'::geometry, 7)
), parts AS (
  SELECT label, CG_Extrude(geom, 0, 0, height) AS geom
  FROM footprints
), shapes AS (
  SELECT ST_Collect(geom ORDER BY label) AS source_geometry
  FROM parts
)
SELECT
  source_geometry AS input_source_geometry,
  CG_3DConvexHull(source_geometry) AS convex_hull
FROM shapes;
```

Output

```
-[ RECORD 1 ]----------
input_source_geometry | GEOMETRYCOLLECTION(POLYHEDRALSURFACE(((0 0 0,0 8 0,5 8 0,5 6 0,2 6 0,2 2 0,5 2 0,5 0 0,0 0 0)) ↩
     ,((0 0 3,5 0 3,5 2 3,2 2 3,2 6 3,5 6 3,5 8 3,0 8 3,0 0 3)),((0 0 0,0 0 3,0 8 3,0 8 0,0 0 0)),((0 8 0,0 8 3,5 8 3,5 ↩
      8 0,0 8 0)),((5 8 0,5 8 3,5 6 3,5 6 0,5 8 0)),((5 6 0,5 6 3,2 6 3,2 6 0,5 6 0)),((2 6 0,2 6 3,2 2 3,2 2 0,2 6 0)) ↩
     ,((2 2 0,2 2 3,5 2 3,5 2 0,2 2 0)),((5 2 0,5 2 3,5 0 3,5 0 0,5 2 0)),((5 0 0,5 0 3,0 0 3,0 0 0,5 0 0))),  ↩
     POLYHEDRALSURFACE(((7 0 0,7 8 0,9 8 0,9 5 0,12 5 0,12 8 0,14 8 0,14 0 0,12 0 0,12 3 0,9 3 0,9 0 0,7 0 0)),((7 0  ↩
     7,9 0 7,9 3 7,12 3 7,12 0 7,14 0 7,14 8 7,12 8 7,12 5 7,9 5 7,9 8 7,7 8 7,7 0 7)),((7 0 0,7 0 7,7 8 7,7 8 0,7 0 0) ↩
     ),((7 8 0,7 8 7,9 8 7,9 8 0,7 8 0)),((9 8 0,9 8 7,9 5 7,9 5 0,9 8 0)),((9 5 0,9 5 7,12 5 7,12 5 0,9 5 0)),((12 5  ↩
     0,12 5 7,12 8 7,12 8 0,12 5 0)),((12 8 0,12 8 7,14 8 7,14 8 0,12 8 0)),((14 8 0,14 8 7,14 0 7,14 0 0,14 8 0)),((14 ↩
      0 0,14 0 7,12 0 7,12 0 0,14 0 0)),((12 0 0,12 0 7,12 3 7,12 3 0,12 0 0)),((12 3 0,12 3 7,9 3 7,9 3 0,12 3 0)),((9 ↩
      3 0,9 3 7,9 0 7,9 0 0,9 3 0)),((9 0 0,9 0 7,7 0 7,7 0 0,9 0 0))))
convex_hull           | POLYHEDRALSURFACE(((14 8 7,14 0 0,14 8 0,14 8 7)),((14 0 7,7 8 7,7 0 7,14 0 7)),((0 8 3,7 8 7,0 ↩
      8 0,0 8 3)),((0 0 3,7 8 7,0 8 3,0 0 3)),((0 0 3,0 8 3,0 0 0,0 0 3)),((7 0 7,7 8 7,0 0 3,7 0 7)),((14 0 7,14 0  ↩
     0,14 8 7,14 0 7)),((7 0 7,14 0 0,14 0 7,7 0 7)),((14 8 7,7 8 7,14 0 7,14 8 7)),((0 0 0,0 8 3,0 8 0,0 0 0)),((0 0  ↩
     0,14 0 0,0 0 3,0 0 0)),((0 0 3,14 0 0,7 0 7,0 0 3)),((0 8 0,14 0 0,0 0 0,0 8 0)),((14 8 0,14 0 0,0 8 0,14 8 0)) ↩
     ,((14 8 0,7 8 7,14 8 7,14 8 0)),((0 8 0,7 8 7,14 8 0,0 8 0)))
```

Figure



Siehe auch

ST_Letters, ST_AsX3D

8.3.9 CG_3DIntersection

CG_3DIntersection — 3D-Schnitte durchführen

Synopsis

geometry **CG_3DIntersection**(geometry geom1, geometry geom2);

Beschreibung

Gibt eine Geometrie zurück, die der gemeinsame Teil von geom1 und geom2 ist.

Verfügbarkeit: 3.5.0

 Diese Methode benötigt ein SFCGAL-Backend.

 Diese Methode setzt die SQL/MM-Spezifikation um. SQL-MM IEC 13249-3: 5.1

 Diese Funktion unterstützt 3d und lässt den Z-Index nicht fallen.

 Diese Funktion unterstützt polyedrische Flächen.

 Diese Funktion unterstützt Dreiecke und dreieckige unregelmäßige Netzoberflächen (TIN).

Beispiele

3D linestrings and polygons can intersect in a 3D segment.

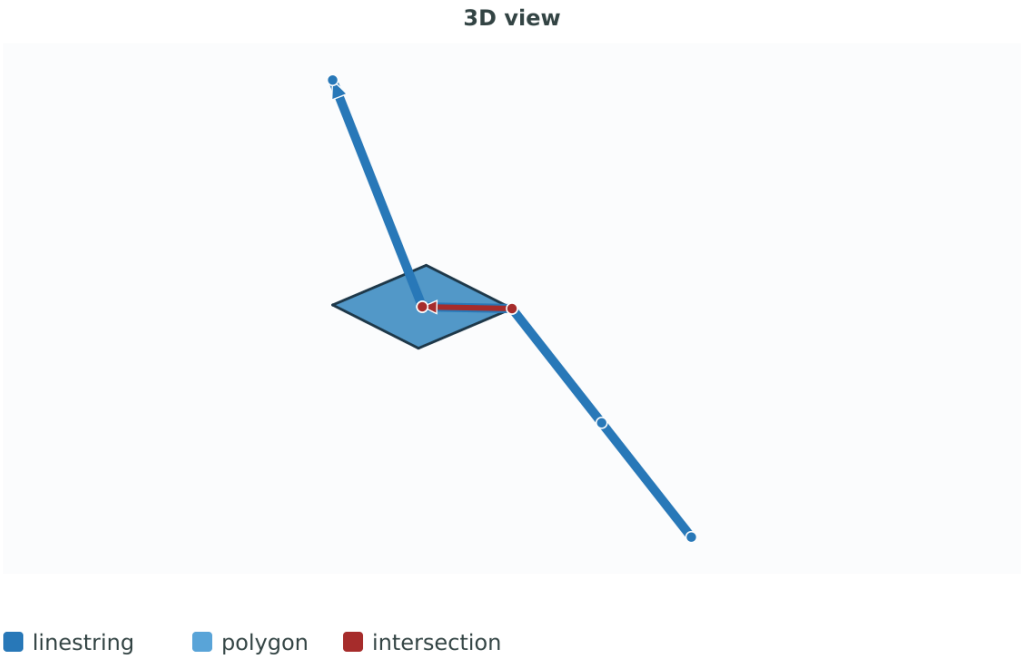
Code

```
WITH data AS (
  SELECT 'LINESTRING Z (2 2 6,1.5 1.5 7,1 1 8,0.5 0.5 8,0 0 10)'::geometry AS ↩
    input_linestring,
         'POLYGON Z ((0 0 8,0 1 8,1 1 8,1 0 8,0 0 8))'::geometry AS input_polygon
)
SELECT input_linestring AS input_linestring,
       input_polygon AS input_polygon,
       CG_3DIntersection(input_linestring, input_polygon) AS intersection
FROM data;
```

Output

```
LINESTRING Z (2 2 6,1.5 1.5 7,1 1 8,0.5 0.5 8,0 0 10) | POLYGON Z ((0 0 8,0 1 8,1 1 8,1 0 8,0 0 8)) | LINESTRING Z (1 1 ↩
     8,0.5 0.5 8)
```

Figure



Intersect a cube shell with a vertical polygon.

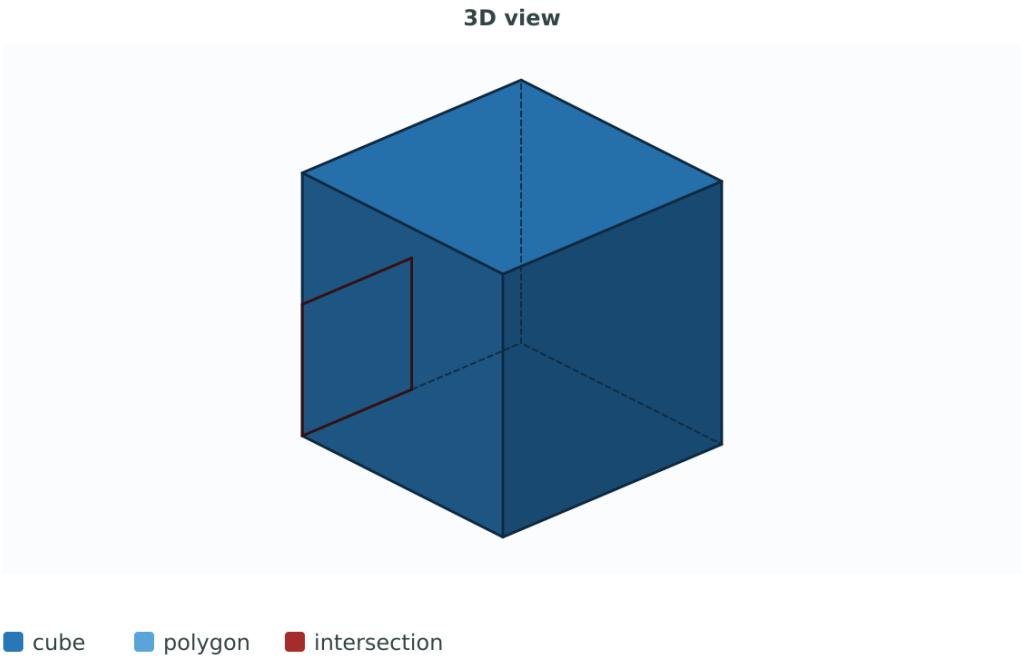
Code

```
WITH data AS (
  SELECT 'POLYHEDRALSURFACE Z(
    ((0 0 0,0 0 1,0 1 1,0 1 0,0 0 0)),
    ((0 0 0,0 1 0,1 1 0,1 0 0,0 0 0)),
    ((0 0 0,1 0 0,1 0 1,0 0 1,0 0 0)),
    ((1 1 0,1 1 1,1 0 1,1 0 0,1 1 0)),
    ((0 1 0,0 1 1,1 1 1,1 1 0,0 1 0)),
    ((0 0 1,1 0 1,1 1 1,0 1 1,0 0 1))
  )'::geometry AS input_cube,
  'POLYGON Z ((0 0 0,0 0 0.5,0 0.5 0.5,0 0.5 0,0 0 0))'::geometry AS input_polygon
)
SELECT input_cube AS input_cube,
       input_polygon AS input_polygon,
       CG_3DIntersection(input_cube, input_polygon) AS intersection
FROM data;
```

Output

```
POLYHEDRALSURFACE Z (((0 0 0,0 0 1,0 1 1,0 1 0,0 0 0)),((0 0 0,0 1 0,1 1 0,1 0 0,0 0 0)),((0 0 0,1 0 0,1 0 1,0 0 1,0 0 ↩
    0)),((1 1 0,1 1 1,1 0 1,1 0 0,1 1 0)),((0 1 0,0 1 1,1 1 1,1 1 0,0 1 0)),((0 0 1,1 0 1,1 1 1,0 1 1,0 0 1))) | ↩
    POLYGON Z ((0 0 0,0 0 0.5,0 0.5 0.5,0 0.5 0,0 0 0)) | TIN Z (((0 0 0,0 0 0.5,0 0.5 0.5,0 0 0)),((0 0.5 0,0 0 0,0 ↩
    0.5 0.5,0 0.5 0)))
```

Figure



A smaller solid contained by a larger solid remains volumetric after intersection.

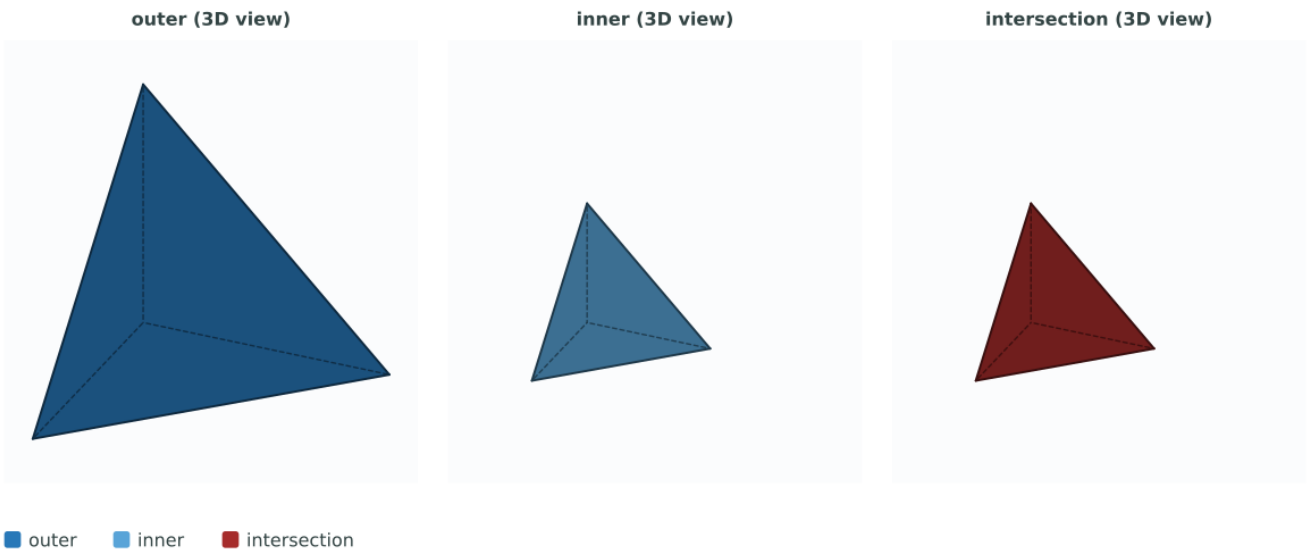
Code

```
WITH data AS (
  SELECT CG_MakeSolid('POLYHEDRALSURFACE Z (
    ((0 0 0,0 4 0,4 0 0,0 0 0)),
    ((0 0 0,4 0 0,0 0 4,0 0 0)),
    ((0 0 0,0 0 4,0 4 0,0 0 0)),
    ((4 0 0,0 4 0,0 0 4,4 0 0))
  )'::geometry) AS outer_solid
), solids AS (
  SELECT outer_solid,
         ST_Scale(outer_solid, 0.5, 0.5, 0.5) AS inner_solid
  FROM data
)
SELECT outer_solid AS input_outer,
       inner_solid AS input_inner,
       CG_3DIntersection(outer_solid, inner_solid) AS intersection
FROM solids;
```

Output

```
POLYHEDRALSURFACE Z (((0 0 0,0 4 0,4 0 0,0 0 0)),((0 0 0,4 0 0,0 0 4,0 0 0)),((0 0 0,0 0 4,0 4 0,0 0 0)),((4 0 0,0 4 ↩
    0,0 0 4,4 0 0))) | POLYHEDRALSURFACE Z (((0 0 0,0 2 0,2 0 0,0 0 0)),((0 0 0,2 0 0,0 0 2,0 0 0)),((0 0 0,0 0 2,0 2 ↩
    0,0 0 0)),((2 0 0,0 2 0,0 0 2,2 0 0))) | POLYHEDRALSURFACE Z (((0 0 0,0 2 0,2 0 0,0 0 0)),((0 0 0,0 0 2,0 2 0,0 0 ↩
    0)),((2 0 0,0 0 2,0 0 0,2 0 0)),((2 0 0,0 2 0,0 0 2,2 0 0)))
```

Figure



8.3.10 CG_Union

CG_Union — Computes the union of two geometries

Synopsis

geometry **CG_Union**(geometry geomA , geometry geomB);

Beschreibung

Computes the union of two geometries.

Performed by the SFCGAL module



Anmerkung
NOTE: this function returns a geometry representing the union.

Verfügbarkeit: 3.5.0


Diese Methode benötigt ein SFCGAL-Backend.

Beispiele mit dem geometrischen Datentyp

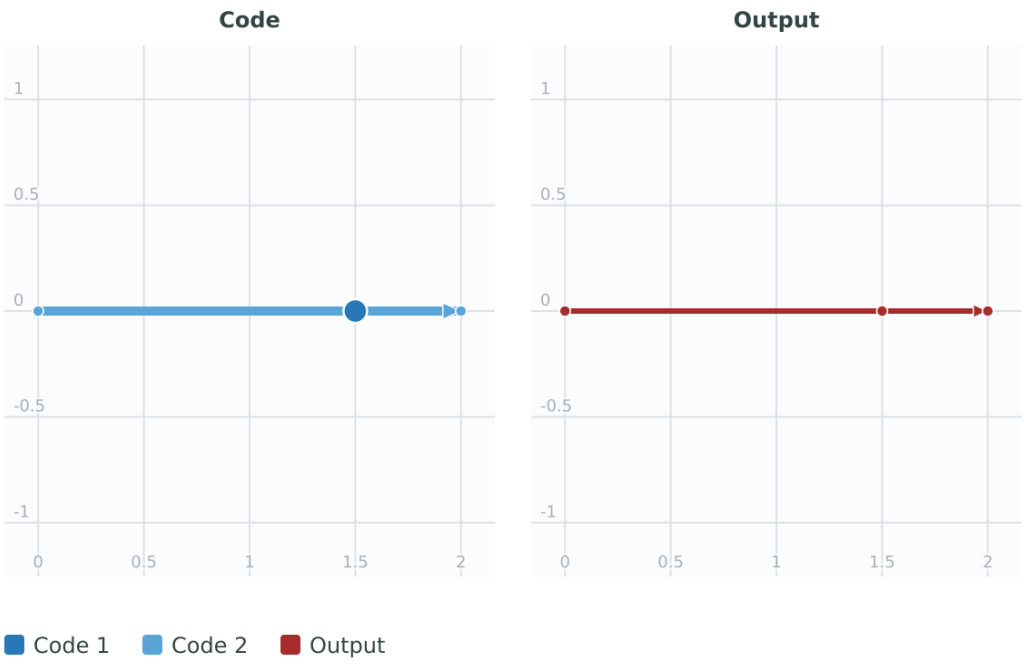
Code

```
SELECT CG_Union('POINT(.5 0)', 'LINESTRING(-1 0,1 0)');
```

Output

```
LINESTRING(-1 0,0.5 0,1 0)
```

Figure



Siehe auch

CG_3DUnion, ST_AsBinary

8.3.11 CG_3DUnion

CG_3DUnion — Perform 3D union using postgis_sfcgal.

Synopsis

geometry **CG_3DUnion**(geometry geom1, geometry geom2);
geometry **CG_3DUnion**(geometry set g1field);

Beschreibung

Verfügbarkeit: 3.5.0

Diese Methode benötigt ein SFCGAL-Backend.

Diese Methode setzt die SQL/MM-Spezifikation um. SQL-MM IEC 13249-3: 5.1

Diese Funktion unterstützt 3d und lässt den Z-Index nicht fallen.

Diese Funktion unterstützt polyedrische Flächen.

Diese Funktion unterstützt Dreiecke und dreieckige unregelmäßige Netzoberflächen (TIN).

Aggregate variant: returns a geometry that is the 3D union of a rowset of geometries. The CG_3DUnion() function is an "aggregate" function in the terminology of PostgreSQL. That means that it operates on rows of data, in the same way the SUM() and AVG() functions do and like most aggregates, it also ignores NULL geometries.

Beispiele

Union two overlapping tetrahedral solids.

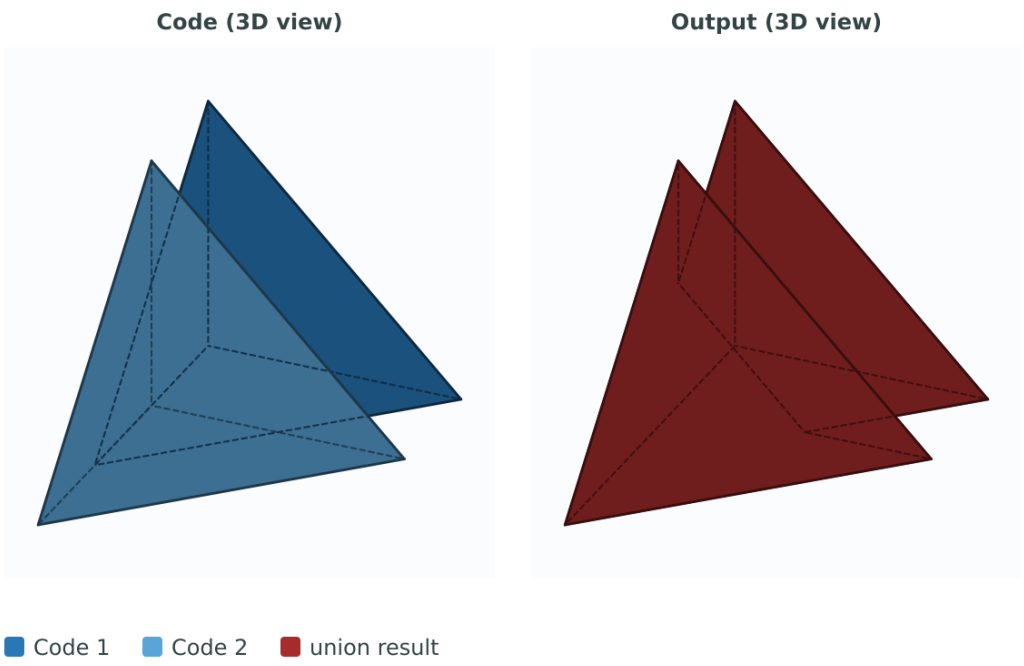
Code

```
WITH solids AS (
  SELECT CG_MakeSolid('POLYHEDRALSURFACE Z (
            ((0 0 0,4 0 0,0 4 0,0 0 0)),
            ((0 0 0,0 0 4,4 0 0,0 0 0)),
            ((0 0 0,0 4 0,0 0 4,0 0 0)),
            ((4 0 0,0 0 4,0 4 0,4 0 0)))'::geometry) AS geom1,
         CG_MakeSolid('POLYHEDRALSURFACE Z (
            ((2 0 0,6 0 0,2 4 0,2 0 0)),
            ((2 0 0,2 0 4,6 0 0,2 0 0)),
            ((2 0 0,2 4 0,2 0 4,2 0 0)),
            ((6 0 0,2 0 4,2 4 0,6 0 0)))'::geometry) AS geom2
)
SELECT CG_3DUnion(geom1, geom2) AS union_result
FROM solids;
```

Output

```
POLYHEDRALSURFACE Z (((2 0 0,0 0 0,2 2 0,2 0 0)),((2 0 2,0 0 0,2 0 0,2 0 2)),((0 0 4,0 4 0,0 0 0,0 0 4)),((2 0 2,0 4 ↩
    0,0 0 4,2 0 2)),((0 4 0,2 2 0,0 0 0,0 4 0)),((2 2 0,0 4 0,2 0 2,2 2 0)),((0 0 0,2 0 2,0 0 4,0 0 0)),((4 0 0,2 0 ↩
    0,2 2 0,4 0 0)),((4 0 0,2 0 2,2 0 0,4 0 0)),((4 0 0,2 2 0,6 0 0,4 0 0)),((2 0 2,4 0 0,2 0 4,2 0 2)),((2 0 2,2 4 ↩
    0,2 2 0,2 0 2)),((2 4 0,2 0 4,6 0 0,2 4 0)),((2 2 0,2 4 0,6 0 0,2 2 0)),((2 4 0,2 0 2,2 0 4,2 4 0)),((4 0 0,6 0 ↩
    0,2 0 4,4 0 0)))
```

Figure



Siehe auch

CG_Extrude, ST_AsX3D, CG_3DIntersection CG_3DDifference

8.3.12 CG_AlphaShape

CG_AlphaShape — Berechnet eine Alpha-Form, die eine Geometrie umschließt

Synopsis

geometry **CG_AlphaShape**(geometry geom, float alpha, boolean allow_holes = false);

Beschreibung

Berechnet den Alpha-Shape der Punkte in einer Geometrie. Eine Alpha-Form ist eine (normalerweise) konkave polygonale Geometrie, die alle Scheitelpunkte der Eingabe enthält und deren Scheitelpunkte eine Teilmenge der Eingabe-Scheitelpunkte sind. Eine Alpha-Form passt sich besser an die Form der Eingabe an als die Form, die durch die konvexe Hülle erzeugt wird.

Die "Passgenauigkeit" wird durch den Parameter `alpha` gesteuert, der Werte zwischen 0 und unendlich annehmen kann. Kleinere Alpha-Werte erzeugen konkavere Ergebnisse. Alpha-Werte, die größer als ein datenabhängiger Wert sind, erzeugen die konvexe Hülle der Eingabe.



Anmerkung
Nach der CGAL-Implementierung ist der Alpha-Wert das *Quadrat* des Radius der Scheibe, die im Alpha-Shape-Algorithmus verwendet wird, um die Delaunay-Triangulation der Eingabepunkte zu "erodieren". Siehe CGAL Alpha-Shapes für weitere Informationen. Dies unterscheidet sich von der ursprünglichen Definition von Alpha-Formen, die Alpha als den Radius der erodierenden Scheibe definiert.

Die berechnete Form enthält keine Löcher, es sei denn, das optionale Argument `allow_holes` wird als true angegeben.

Diese Funktion berechnet eine konkave Hülle einer Geometrie auf ähnliche Weise wie ST_ConcaveHull, verwendet aber CGAL und einen anderen Algorithmus.

Availability: 3.5.0 - requires SFCGAL >= 1.4.1.

 Diese Methode benötigt ein SFCGAL-Backend.

Beispiele

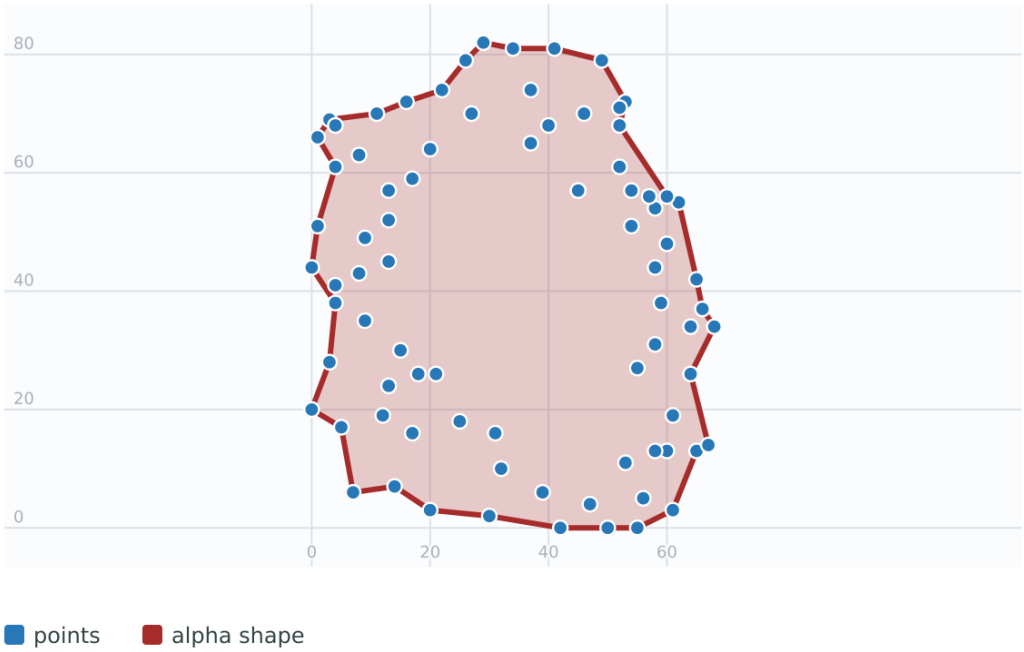
The original point cloud and the alpha-shape share one comparison frame.

Code

```
WITH data AS (
  SELECT 'MULTIPOINT((63 84),(76 88),(68 73),(53 18),(91 50),(81 70),
(88 29),(24 82),(32 51),(37 23),(27 54),(84 19),(75 87),(44 42),(77 67),(90 30),(36 61),(32 ↩
    65),
(81 47),(88 58),(68 73),(49 95),(81 60),(87 50),
(78 16),(79 21),(30 22),(78 43),(26 85),(48 34),(35 35),(36 40),(31 79),(83 29),(27 84),(52 ↩
    98),(72 95),(85 71),
(75 84),(75 77),(81 29),(77 73),(41 42),(83 72),(23 36),(89 53),(27 57),(57 97),(27 77),(39 ↩
    88),(60 81),
(80 72),(54 32),(55 26),(62 22),(70 20),(76 27),(84 35),(87 42),(82 54),(83 64),(69 86),(60 ↩
    90),(50 86),(43 80),(36 73),
(36 68),(40 75),(24 67),(23 60),(26 44),(28 33),(40 32),(43 19),(65 16),(73 16),(38 46),(31 ↩
    59),(34 86),(45 90),(64 97))'::geometry AS input_points
)
SELECT input_points AS input_points,

         CG_AlphaShape(input_points, 80.2) AS alpha_shape
FROM data;
```

Figure



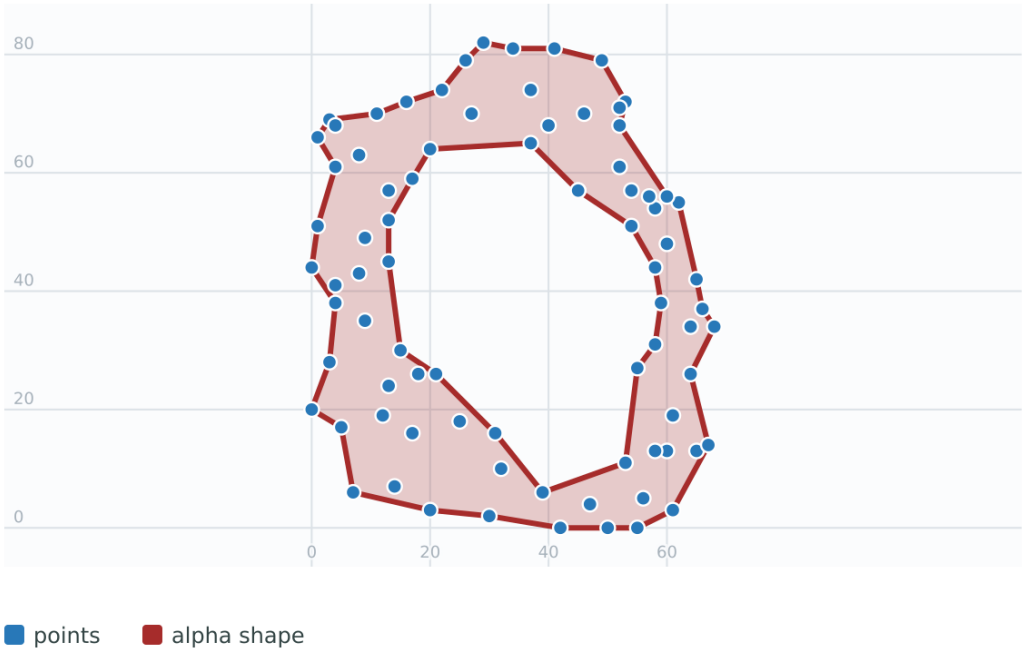
The same point cloud can produce a shape with a hole when holes are allowed.

Code

```
WITH data AS (
  SELECT 'MULTIPOINT((63 84),(76 88),(68 73),(53 18),(91 50),(81 70),(88 29),(24 82),(32 51) ↩
    ,(37 23),(27 54),(84 19),(75 87),(44 42),(77 67),(90 30),(36 61),(32 65),(81 47),(88 58) ↩
    ,(68 73),(49 95),(81 60),(87 50),
(78 16),(79 21),(30 22),(78 43),(26 85),(48 34),(35 35),(36 40),(31 79),(83 29),(27 84),(52  ↩
    98),(72 95),(85 71),
(75 84),(75 77),(81 29),(77 73),(41 42),(83 72),(23 36),(89 53),(27 57),(57 97),(27 77),(39  ↩
    88),(60 81),
(80 72),(54 32),(55 26),(62 22),(70 20),(76 27),(84 35),(87 42),(82 54),(83 64),(69 86),(60  ↩
    90),(50 86),(43 80),(36 73),
(36 68),(40 75),(24 67),(23 60),(26 44),(28 33),(40 32),(43 19),(65 16),(73 16),(38 46),(31  ↩
    59),(34 86),(45 90),(64 97))'::geometry AS input_points
)
SELECT input_points AS input_points,

         CG_AlphaShape(input_points, 100.1, true) AS alpha_shape
FROM data;
```

Figure



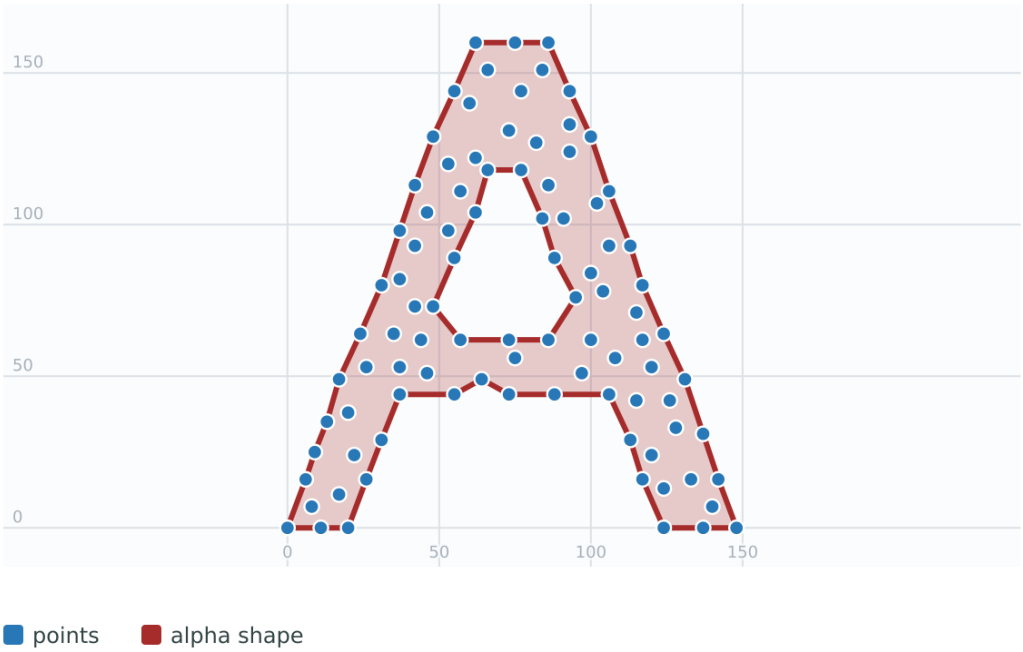
The second documented point cloud also produces a holed alpha-shape.

Code

```
WITH data AS (
  SELECT 'MULTIPOINT ((132 64),(114 64),(99 64),(81 64),(63 64),(57 49),(52 36),(46 20),(37
    20),(26 20),(32 36),(39 55),(43 69),(50 84),(57 100),(63 118),(68 133),(74 149),(81 164)
    ,(88 180),(101 180),(112 180),(119 164),(126 149),(132 131),(139 113),(143 100),(150 84)
    ,(157 69),(163 51),(168 36),(174 20),(163 20),(150 20),(143 36),(139 49),(132 64),(99
    151),(92 138),(88 124),(81 109),(74 93),(70 82),(83 82),(99 82),(112 82),(126 82),(121
    96),(114 109),(110 122),(103 138),(99 151),(34 27),(43 31),(48 44),(46 58),(52 73),(63
    73),(61 84),(72 71),(90 69),(101 76),(123 71),(141 62),(166 27),(150 33),(159 36),(146
    44),(154 53),(152 62),(146 73),(134 76),(143 82),(141 91),(130 98),(126 104),(132 113)
    ,(128 127),(117 122),(112 133),(119 144),(108 147),(119 153),(110 171),(103 164),(92
    171),(86 160),(88 142),(79 140),(72 124),(83 131),(79 118),(68 113),(63 102),(68 93),(35
     45))'::geometry AS input_points
)
SELECT input_points AS input_points,

         CG_AlphaShape(input_points, 102.2, true) AS alpha_shape
FROM data;
```

Figure



Siehe auch

ST_ConcaveHull, CG_OptimalAlphaShape

8.3.13 CG_ApproxConvexPartition

CG_ApproxConvexPartition — Berechnet die approximale konvexe Partition der Polygongeometrie

Synopsis

geometry **CG_ApproxConvexPartition**(geometry geom);

Beschreibung

Berechnet die approximale konvexe Partition der Polygongeometrie (unter Verwendung einer Triangulation).



Anmerkung
Eine Partition eines Polygons P ist eine Menge von Polygonen, bei der sich die Innenräume der Polygone nicht schneiden und die Vereinigung der Polygone gleich dem Innenraum des ursprünglichen Polygons P ist. Die Funktionen CG_ApproxConvexPartition und CG_GreeneApproxConvexPartition erzeugen annähernd optimale konvexe Partitionen. Beide Funktionen erzeugen konvexe Zerlegungen, indem sie das Polygon zunächst in einfachere Polygone zerlegen; CG_ApproxConvexPartition verwendet eine Triangulation und CG_GreeneApproxConvexPartition eine monotone Partition. Diese beiden Funktionen garantieren, dass sie nicht mehr als das Vierfache der optimalen Anzahl konvexer Teile erzeugen, unterscheiden sich aber in ihrer Laufzeitkomplexität. Obwohl der auf Triangulation basierende Approximationsalgorithmus oft zu weniger konvexen Teilen führt, ist dies nicht immer der Fall.

Verfügbarkeit: 3.5.0 - erfordert SFCGAL >= 1.5.0.

Erfordert SFCGAL >= 1.5.0

 Diese Methode benötigt ein SFCGAL-Backend.

Beispiele

Approximate convex partition of the same polygon used by the other partition examples.

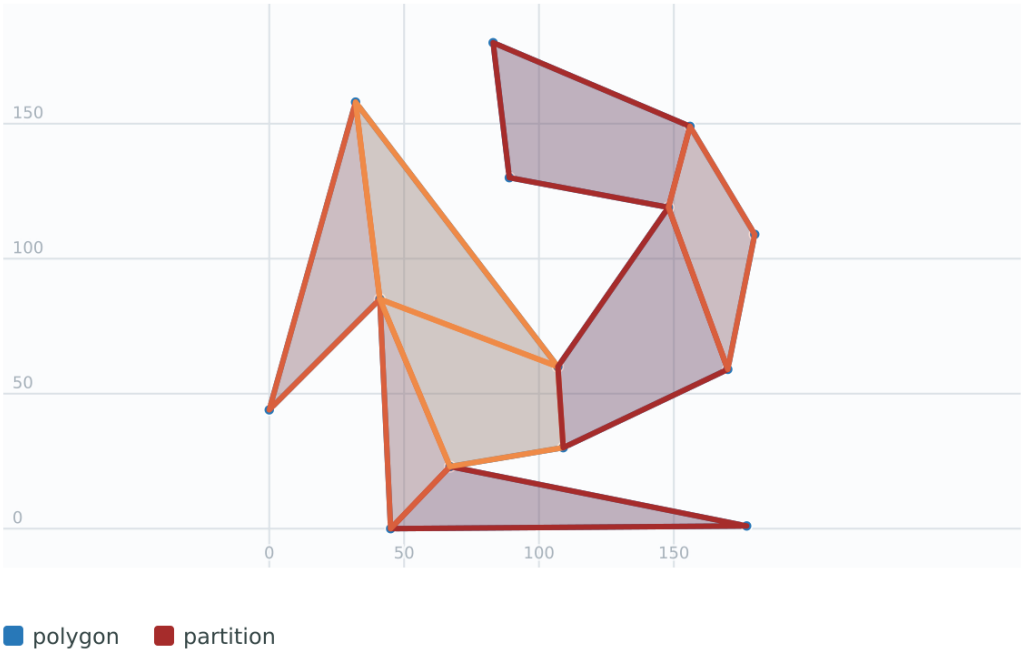
Code

```
WITH data AS (
  SELECT 'POLYGON((156 150,83 181,89 131,148 120,107 61,32 159,0 45,41 86,45 1,177 2,67 ↩
    24,109 31,170 60,180 110,156 150))'::geometry AS input_polygon
)
SELECT input_polygon AS input_polygon,
       CG_ApproxConvexPartition(input_polygon) AS partition
FROM data;
```

Output

```
POLYGON((156 150,83 181,89 131,148 120,107 61,32 159,0 45,41 86,45 1,177 2,67 24,109 31,170 60,180 110,156 150)) | ↩
    GEOMETRYCOLLECTION(POLYGON((156 150,83 181,89 131,148 120,156 150)), POLYGON((32 159,0 45,41 86,32 159)), POLYGON ↩
    ((107 61,32 159,41 86,107 61)), POLYGON((45 1,177 2,67 24,45 1)), POLYGON((41 86,45 1,67 24,41 86)), POLYGON((107 ↩
    61,41 86,67 24,109 31,107 61)), POLYGON((148 120,107 61,109 31,170 60,148 120)), POLYGON((156 150,148 120,170 ↩
    60,180 110,156 150)))
```

Figure



Siehe auch

CG_YMonotonePartition, CG_GreeneApproxConvexPartition, CG_OptimalConvexPartition

8.3.14 CG_ApproximateMedialAxis

CG_ApproximateMedialAxis — Berechnet die konvexe Hülle einer Geometrie.

Synopsis

geometry **CG_ApproximateMedialAxis**(geometry geom);
geometry **CG_ApproximateMedialAxis**(geometry geom, boolean projected);

Beschreibung

Return an approximate medial axis for the areal input based on its straight skeleton. Uses an SFCGAL specific API when built against a capable version (1.2.0+). Otherwise the function is just a wrapper around CG_StraightSkeleton (slower case).

When `projected` is `true`, free endpoints of the medial axis are extended to reach the polygon boundary (projected medial axis). Requires SFCGAL 2.3.0+. When built against an older SFCGAL version, a notice is emitted and the non-projected result is returned instead.

Verfügbarkeit: 3.5.0

Availability: 3.7.0 - `projected` parameter. Requires SFCGAL >= 2.3.0 for projected result; falls back to non-projected with a notice on older versions.



Anmerkung
Diese Funktion ignoriert die Z-Dimension. Sie liefert immer ein 2D-Ergebnis, auch wenn sie auf eine 3D-Geometrie angewendet wird.

 Diese Methode benötigt ein SFCGAL-Backend.

 Diese Funktion unterstützt polyedrische Flächen.

 Diese Funktion unterstützt Dreiecke und dreieckige unregelmäßige Netzoberflächen (TIN).

Beispiele

A polygon and its approximate medial axis.

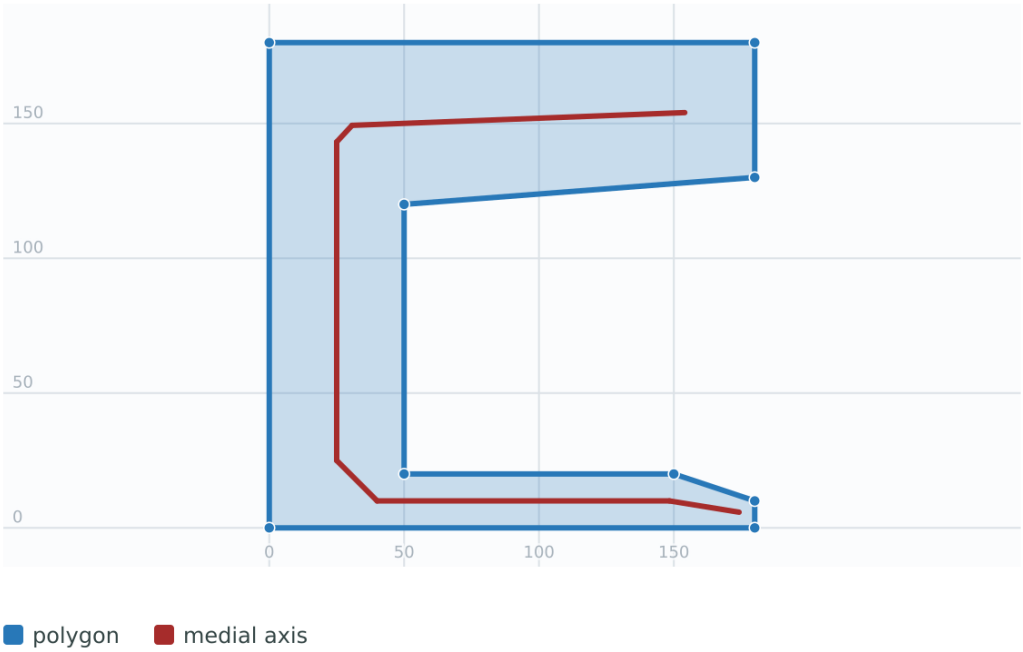
Code

```
WITH data AS (
  SELECT 'POLYGON((190 190,10 190,10 10,190 10,190 20,160 30,60 30,60 130,190 140,190 190))' ↩
    ::geometry AS polygon
)
SELECT polygon AS input_polygon,
       CG_ApproximateMedialAxis(polygon) AS medial_axis
FROM data;
```

Output

```
POLYGON((190 190,10 190,10 10,190 10,190 20,160 30,60 30,60 130,190 140,190 190)) | MULTILINESTRING((184.189 ↩
    15.811,158.377 20),(50 20,158.377 20),(50 20,35 35),(35 153.151,35 35),(35 153.151,40.697 159.303),(164.04 ↩
    164.04,40.697 159.303))
```

Figure



This example computes the projected medial axis with free endpoints extended to the polygon boundary.

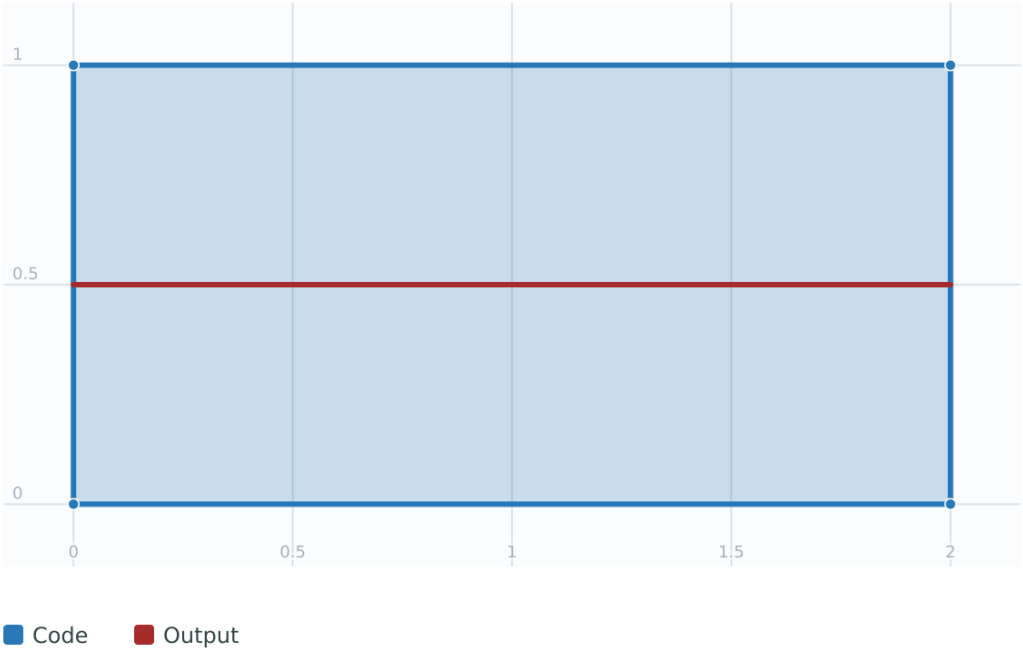
Code

```
SELECT CG_ApproximateMedialAxis('POLYGON((0 0,2 0,2 1,0 1,0 0))', true);
```

Output

```
MULTILINESTRING((0 0.5,0.5 0.5,1.5 0.5,2 0.5))
```

Figure



Siehe auch

CG_StraightSkeleton, CG_StraightSkeletonPartition

8.3.15 CG_GenerateFlatRoof

CG_GenerateFlatRoof — Generate a flat (box) roof from a footprint polygon.

Synopsis

geometry **CG_GenerateFlatRoof**(geometry geom, float8 height);

Beschreibung

Generate a flat (box-shaped) roof as a 3D `PolyhedralSurface Z` from a 2D footprint polygon. The roof height above the ground plane is given by `height` (default 3.0).

Availability: 3.7.0 - requires SFCGAL >= 2.3.0.

Diese Methode benötigt ein SFCGAL-Backend.

Diese Funktion unterstützt polyedrische Flächen.

Beispiele

Generate a flat roof two units above a rectangular footprint.

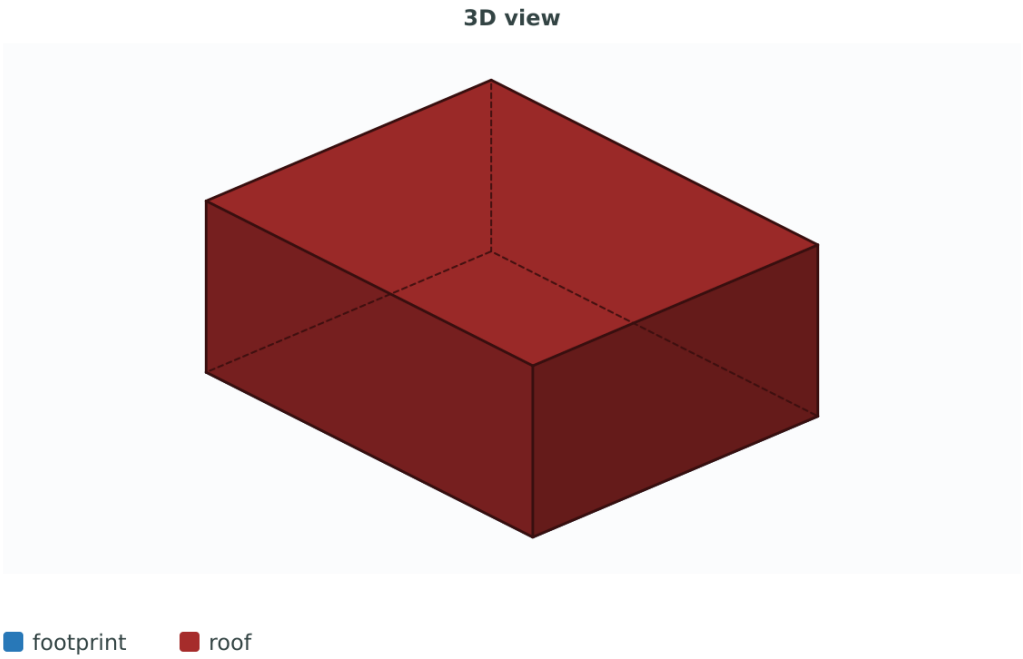
Code

```
WITH data AS (
  SELECT 'POLYGON((0 0,5 0,5 4,0 4,0 0))'::geometry AS footprint
)
SELECT CG_GenerateFlatRoof(footprint, 2.0) AS roof
FROM data;
```

Output

```
POLYHEDRALSURFACE Z (((0 0 0,0 4 0,5 4 0,5 0 0,0 0 0)),((0 0 2,5 0 2,5 4 2,0 4 2,0 0 2)),((0 0 0,0 0 2,0 4 2,0 4 0,0 0 ↩
    0)),((0 4 0,0 4 2,5 4 2,5 4 0,0 4 0)),((5 4 0,5 4 2,5 0 2,5 0 0,5 4 0)),((5 0 0,5 0 2,0 0 2,0 0 0,5 0 0)))
```

Figure



Siehe auch

CG_GenerateHippedRoof, CG_GenerateGableRoof, CG_GenerateSkillionRoof, CG_GenerateRoof

8.3.16 CG_GenerateHippedRoof

CG_GenerateHippedRoof — Generate a hipped roof from a footprint polygon.

Synopsis

geometry **CG_GenerateHippedRoof**(geometry geom, float8 height);

Beschreibung

Generate a hipped roof as a 3D `PolyhedralSurface Z` from a 2D footprint polygon. All sides slope upward to meet at a ridge or apex. The roof height is given by `height` (default 3.0).

Availability: 3.7.0 - requires SFCGAL >= 2.3.0.

Diese Methode benötigt ein SFCGAL-Backend.

Diese Funktion unterstützt polyedrische Flächen.

Beispiele

The footprint and hipped roof are shown side-by-side so the generated 3D faces do not hide behind the input footprint.

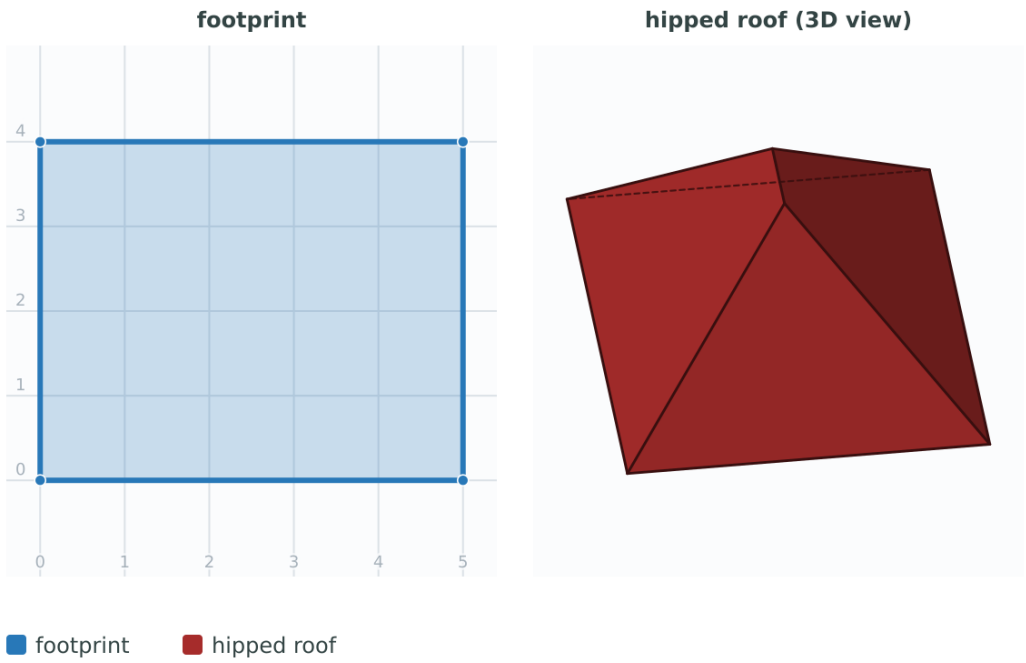
Code

```
WITH data AS (
  SELECT 'POLYGON((0 0,5 0,5 4,0 4,0 0))'::geometry AS input_footprint
)
SELECT input_footprint AS input_footprint,
       CG_GenerateHippedRoof(input_footprint, 2.0) AS hipped_roof
FROM data;
```

Output

```
POLYGON((0 0,5 0,5 4,0 4,0 0)) | POLYHEDRALSURFACE Z (((0 4 0,5 4 0,5 0 0,0 0 0,0 4 0)),((0 4 0,0 0 0,2 2 2,0 4 0)),((0 ↩
      0 0,5 0 0,3 2 2,2 2 2,0 0 0)),((5 0 0,5 4 0,3 2 2,5 0 0)),((5 4 0,0 4 0,2 2 2,3 2 2,5 4 0)))
```

Figure



Siehe auch

CG_GenerateFlatRoof, CG_GenerateGableRoof, CG_GenerateSkillionRoof, CG_GenerateRoof

8.3.17 CG_GenerateGableRoof

CG_GenerateGableRoof — Generate a gable roof from a footprint polygon.

Synopsis

geometry **CG_GenerateGableRoof**(geometry geom, float8 height, float8 slope_angle);

Beschreibung

Generate a gable roof as a 3D `PolyhedralSurface Z` from a 2D footprint polygon. Two opposite sides are vertical gable ends; the other two sides slope to a horizontal ridge. `height` sets the ridge height above the ground (default 3.0) and `slope_angle` controls the pitch in degrees (default 30.0). The SRID of the input geometry is preserved.

Availability: 3.7.0 - requires SFCGAL >= 2.3.0.

Diese Methode benötigt ein SFCGAL-Backend.

Diese Funktion unterstützt polyedrische Flächen.

Beispiele

Generate a gable roof using a height of 2 and a 30 degree slope.

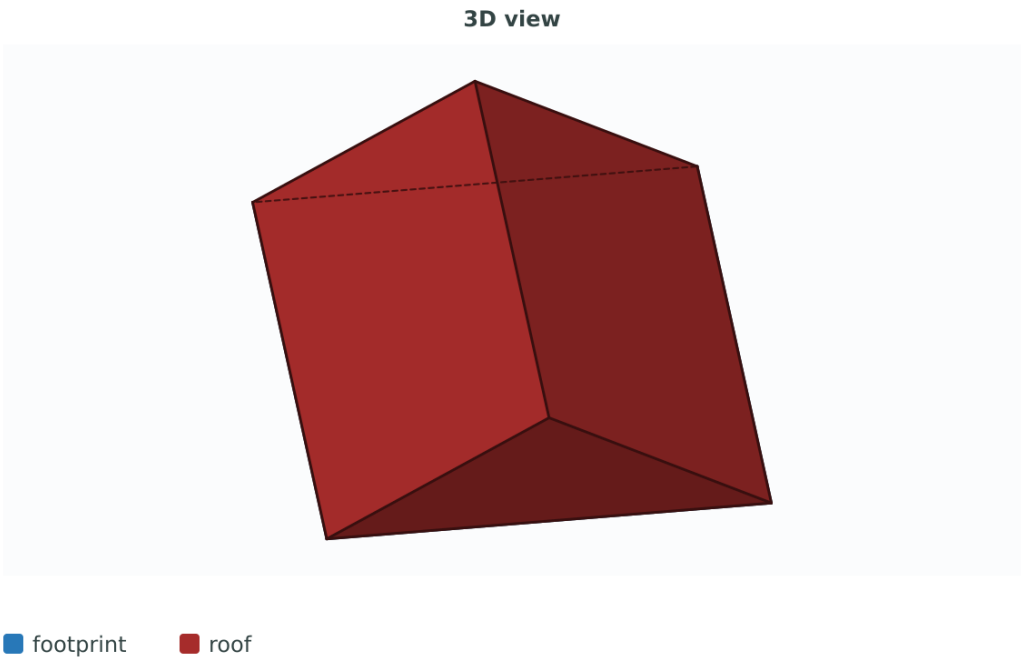
Code

```
WITH data AS (
  SELECT 'POLYGON((0 0,5 0,5 4,0 4,0 0))'::geometry AS footprint
)
SELECT CG_GenerateGableRoof(footprint, 2.0, 30.0) AS roof
FROM data;
```

Output

```
POLYHEDRALSURFACE Z (((5 4 0,5 0 0,0 0 0,0 4 0,5 4 0)),((0 4 0,0 0 0,0 2 1.155,0 4 0)),((0 0 0,5 0 0,5 2 1.155,0 2 ↩
    1.155,0 0 0)),((5 0 0,5 4 0,5 2 1.155,5 0 0)),((5 4 0,0 4 0,0 2 1.155,5 2 1.155,5 4 0)))
```

Figure



Siehe auch

CG_GenerateFlatRoof, CG_GenerateHippedRoof, CG_GenerateSkillionRoof, CG_GenerateRoof

8.3.18 CG_GenerateSkillionRoof

CG_GenerateSkillionRoof — Generate a skillion (single-slope) roof from a footprint polygon.

Synopsis

geometry **CG_GenerateSkillionRoof**(geometry geom, float8 height, float8 slope_angle, integer primary_edge_index);

Beschreibung

Generate a skillion (single-slope, shed-style) roof as a 3D `PolyhedralSurface Z` from a 2D footprint polygon. The entire roof surface slopes in one direction. `height` sets the maximum ridge height (default 3.0), `slope_angle` sets the pitch in degrees (default 30.0), and `primary_edge_index` selects which polygon edge defines the high side (0-based, default 0).

When the combination of `height` and `slope_angle` would cause a side wall to vanish (the slope reaches the far edge before the full height is achieved), that face is omitted from the result. This means the number of output patches can vary depending on the parameters — a lower height relative to the footprint depth produces more faces than a taller, steeper configuration.

Availability: 3.7.0 - requires SFCGAL >= 2.3.0.

Diese Methode benötigt ein SFCGAL-Backend.

Diese Funktion unterstützt polyedrische Flächen.

Beispiele

Height 2.0 produces 6 faces (all walls present); height 4.0 produces 5 faces (one side wall is filtered out because the slope reaches the far edge before the full height).

Code

```
SELECT
  ST_NumPatches(CG_GenerateSkillionRoof(
    'POLYGON((0 0,5 0,5 4,0 4,0 0))', 2.0)) AS "height 2",
  ST_NumPatches(CG_GenerateSkillionRoof(
    'POLYGON((0 0,5 0,5 4,0 4,0 0))', 4.0)) AS "height 4";
```

Output

```
6 | 5
```

The height 2.0 result is shown with its footprint.

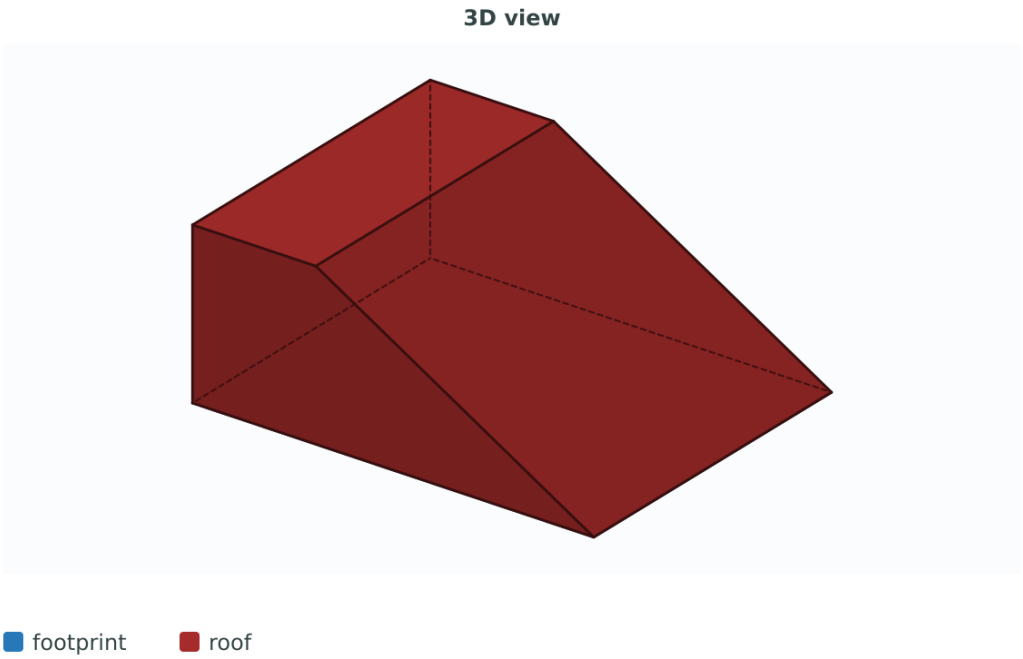
Code

```
WITH data AS (
  SELECT 'POLYGON((0 0,5 0,5 4,0 4,0 0))'::geometry AS footprint
)
SELECT CG_GenerateSkillionRoof(footprint, 2.0) AS roof
FROM data;
```

Output

```
POLYHEDRALSURFACE Z (((0 4 0,5 4 0,5 0 0,0 0 0,0 4 0)),((5 0 2,5 4 2,3.464 4 2,3.464 0 2,5 0 2)),((0 4 0,0 0 0,3.464 0 ↩
    2,3.464 4 2,0 4 0)),((0 0 0,5 0 0,5 0 2,3.464 0 2,0 0 0)),((5 0 0,5 4 0,5 4 2,5 0 2,5 0 0)),((5 4 0,0 4 0,3.464 4 ↩
    2,5 4 2,5 4 0)))
```

Figure



Siehe auch

CG_GenerateFlatRoof, CG_GenerateHippedRoof, CG_GenerateGableRoof, CG_GenerateRoof

8.3.19 CG_GenerateRoof

CG_GenerateRoof — Generate a roof of the requested type from a footprint polygon.

Synopsis

geometry **CG_GenerateRoof**(geometry geom, text roof_type, float8 height, float8 slope_angle, integer primary_edge_index);

Beschreibung

Generate a roof as a 3D `PolyhedralSurface Z` from a 2D footprint polygon. The `roof_type` parameter selects the roof style (default `'HIPPED'`):

- `FLAT` — flat box roof.
- `HIPPED` — hipped roof (all sides slope to apex).
- `GABLE` — gable roof (two sloping sides, two vertical ends).
- `SKILLION` — single-slope (shed) roof.

`height` sets the roof height (default 3.0), `slope_angle` sets the pitch in degrees (default 30.0), and `primary_edge_index` selects the reference edge for skillion roofs (default 0).

Availability: 3.7.0 - requires SFCGAL >= 2.3.0.

Diese Methode benötigt ein SFCGAL-Backend.

Diese Funktion unterstützt polyedrische Flächen.

Beispiele

The generic dispatcher returns the same 3D roofs as the specialized helpers for the same footprint.

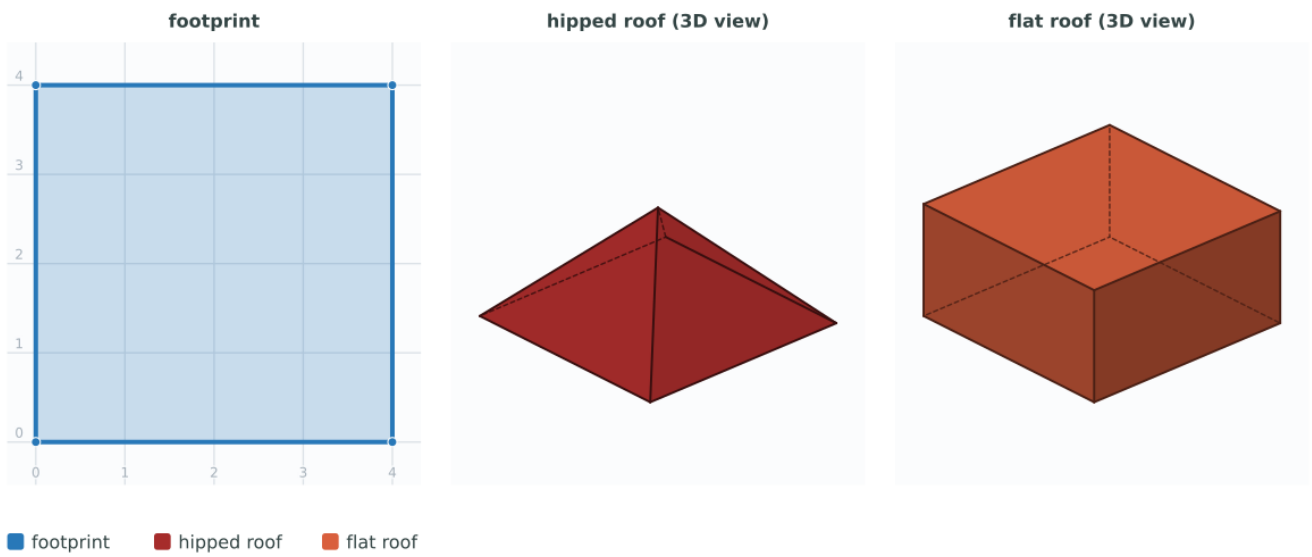
Code

```
WITH data AS (
  SELECT 'POLYGON((0 0,4 0,4 4,0 4,0 0))'::geometry AS input_footprint
)
SELECT input_footprint AS input_footprint,
       CG_GenerateRoof(input_footprint, 'HIPPED', 2.0, 30.0, 0) AS hipped_roof,
       CG_GenerateRoof(input_footprint, 'FLAT', 2.0, 30.0, 0) AS flat_roof
FROM data;
```

Output

```
POLYGON((0 0,4 0,4 4,0 4,0 0)) | POLYHEDRALSURFACE Z (((0 4 0,4 4 0,4 0 0,0 0 0,0 4 0)),((0 4 0,0 0 0,2 2 2,0 4 0)),((0 ↩
      0 0,4 0 0,2 2 2,0 0 0)),((4 0 0,4 4 0,2 2 2,4 0 0)),((4 4 0,0 4 0,2 2 2,4 4 0))) | POLYHEDRALSURFACE Z (((0 0 0,0 ↩
      4 0,4 4 0,4 0 0,0 0 0)),((0 0 2,4 0 2,4 4 2,0 4 2,0 0 2)),((0 0 0,0 0 2,0 4 2,0 4 0,0 0 0)),((0 4 0,0 4 2,4 4 2,4 ↩
      4 0,0 4 0)),((4 4 0,4 4 2,4 0 2,4 0 0,4 4 0)),((4 0 0,4 0 2,0 0 2,0 0 0,4 0 0)))
```

Figure



Siehe auch

CG_GenerateFlatRoof, CG_GenerateHippedRoof, CG_GenerateGableRoof, CG_GenerateSkillionRoof

8.3.20 CG_PolygonRepair

CG_PolygonRepair — Repair an invalid polygon or multipolygon.

Synopsis

geometry **CG_PolygonRepair**(geometry geom, text rule);

Beschreibung

Repair an invalid polygon or multipolygon using CGAL's 2D Polygon Repair algorithm. Returns a `MultiPolygon`.

The `rule` parameter controls the repair strategy:

- `EVEN_ODD` (default) — areas covered an odd number of times are kept. Available with CGAL 6.0+.
- `NON_ZERO` — areas with non-zero winding number are kept. Requires CGAL 6.1+.
- `UNION` — union of all input polygons. Requires CGAL 6.1+.
- `INTERSECTION` — intersection of all input polygons. Requires CGAL 6.1+.

Availability: 3.7.0 - requires SFCGAL >= 2.3.0 and CGAL >= 6.0.

Diese Methode benötigt ein SFCGAL-Backend.

Diese Funktion unterstützt polyedrische Flächen.

Beispiele

Repair a self-intersecting bowtie polygon with the default `EVEN_ODD` rule.

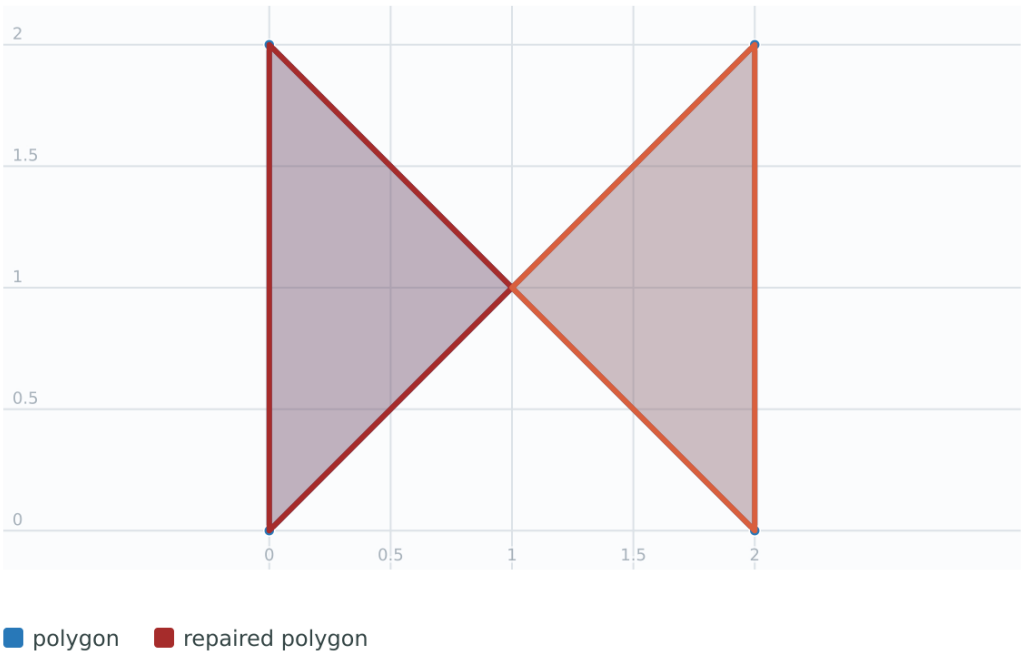
Code

```
WITH data AS (
  SELECT 'POLYGON((0 0,2 2,2 0,0 2,0 0))'::geometry AS input_polygon
)
SELECT input_polygon AS input_polygon,
       CG_PolygonRepair(input_polygon) AS repaired_polygon
FROM data;
```

Output

```
POLYGON((0 0,2 2,2 0,0 2,0 0)) | MULTIPOLYGON(((0 0,1 1,0 2,0 0)),((1 1,2 0,2 2,1 1)))
```

Figure



With the `UNION` rule, all winding area is kept. This rule requires CGAL 6.1 or later.

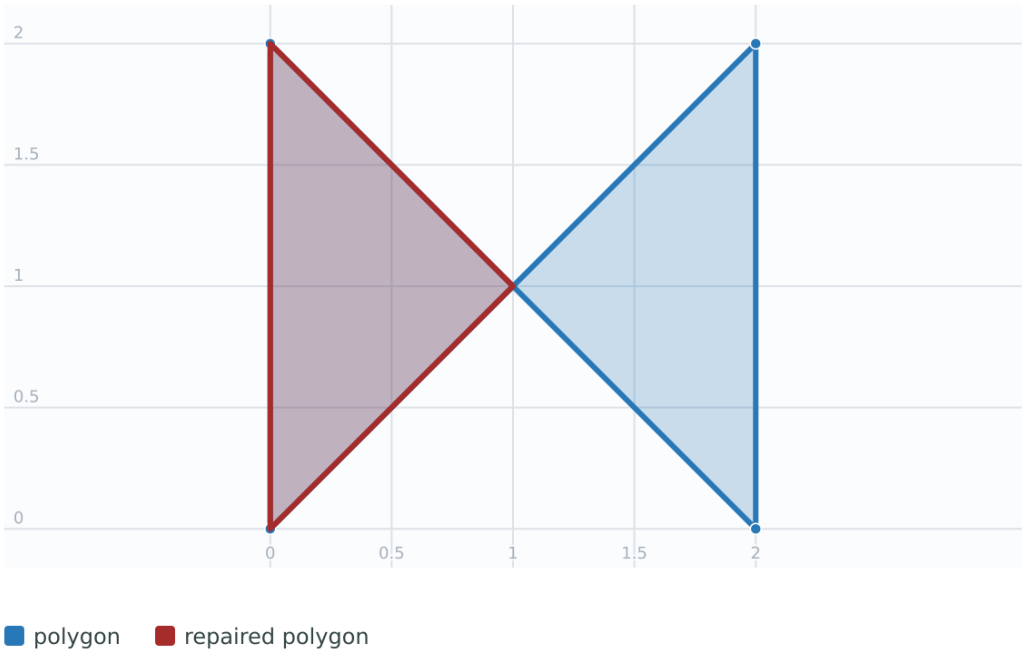
Code

```
WITH data AS (
  SELECT 'POLYGON((0 0,2 2,2 0,0 2,0 0))'::geometry AS input_polygon
)
SELECT input_polygon AS input_polygon,
       CG_PolygonRepair(input_polygon, 'UNION') AS repaired_polygon
FROM data;
```

Output

```
POLYGON((0 0,2 2,2 0,0 2,0 0)) | MULTIPOLYGON(((0 0,1 1,0 2,0 0)))
```

Figure



Siehe auch

ST_IsValid, ST_MakeValid

8.3.21 CG_ConstrainedDelaunayTriangles

CG_ConstrainedDelaunayTriangles — Gibt eine eingeschränkte Delaunay-Triangulation um die angegebene Eingabegeometrie zurück.

Synopsis

geometry **CG_ConstrainedDelaunayTriangles**(geometry g1);

Beschreibung

Gibt eine Constrained Delaunay Triangulation um die Eckpunkte der Eingabegeometrie zurück. Die Ausgabe ist ein TIN.

Diese Methode benötigt ein SFCGAL-Backend.

Verfügbarkeit: 2.1.0

Diese Funktion unterstützt 3d und lässt den Z-Index nicht fallen.

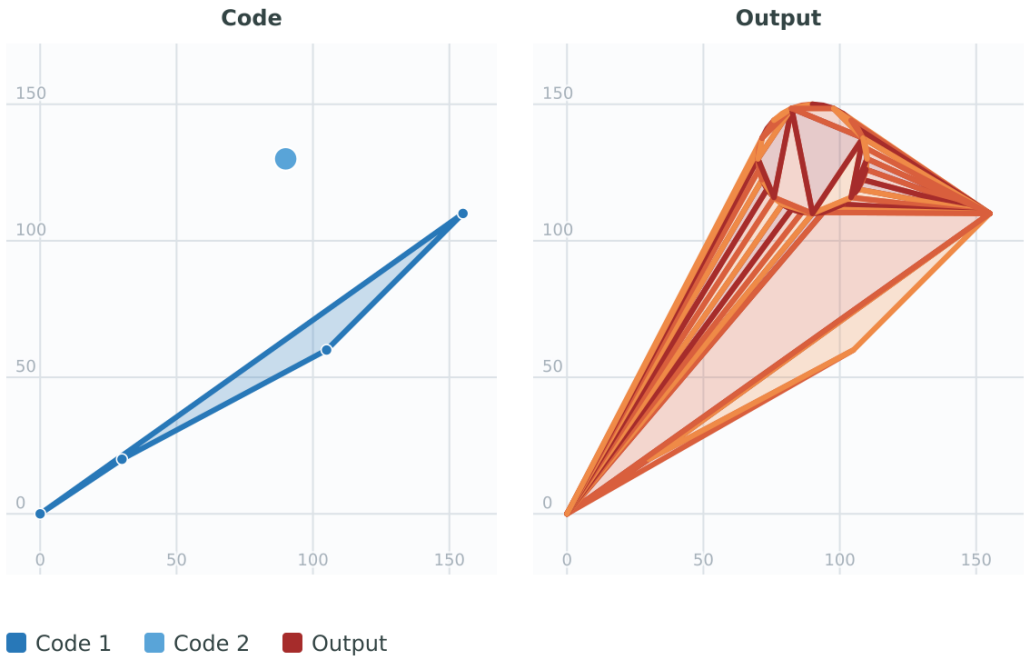
Beispiele

Constrained Delaunay triangulation of the polygon and buffered point.

Code

```
SELECT
  CG_ConstrainedDelaunayTriangles(
    ST_Union(
      'POLYGON((175 150,20 40,50 60,125 100,175 150))'::geometry,
      ST_Buffer('POINT(110 170)'::geometry, 20)
    )
  );
```

Figure

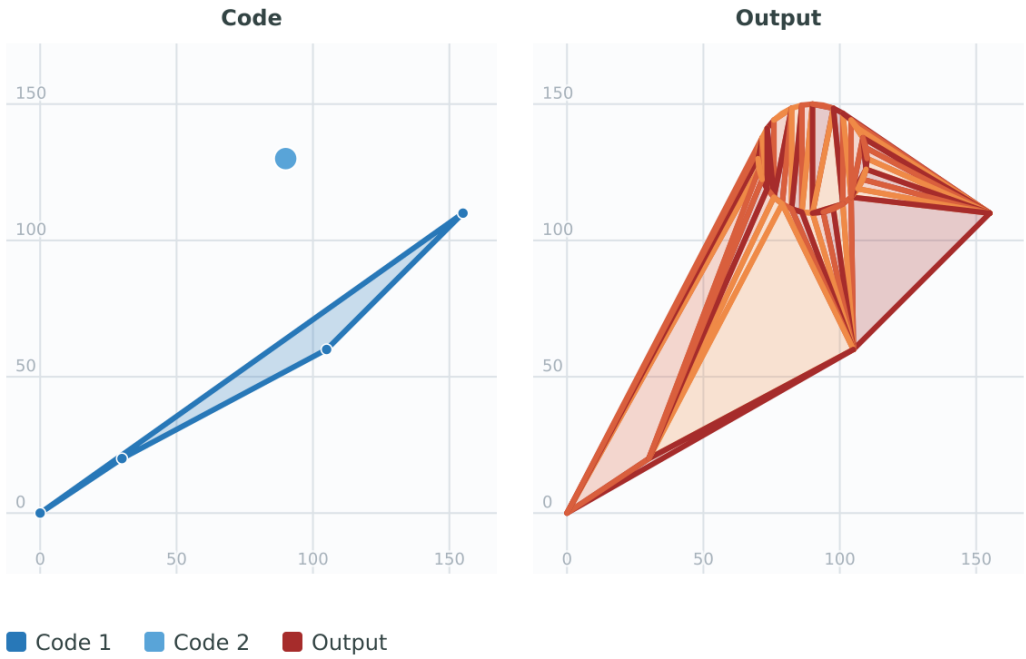


ST_DelaunayTriangles applied to the same input. Triangle edges cross polygon boundaries.

Code

```
SELECT
  ST_DelaunayTriangles(
    ST_Union(
      'POLYGON((175 150,20 40,50 60,125 100,175 150))'::geometry,
      ST_Buffer('POINT(110 170)'::geometry, 20)
    )
  );
```

Figure



Siehe auch

ST_DelaunayTriangles, ST_TriangulatePolygon, CG_Tesselate, ST_ConcaveHull, ST_Dump

8.3.22 CG_Extrude

CG_Extrude — Extrudieren einer Oberfläche in ein zugehöriges Volumen

Synopsis

geometry **CG_Extrude**(geometry geom, float x, float y, float z);

Beschreibung

Extrudes the input along the vector (`x`, `y`, `z`) and connects it to its translated copy. Extrusion can raise the dimensionality of the result, for example from a surface to a volume.

Verfügbarkeit: 3.5.0

Diese Methode benötigt ein SFCGAL-Backend.

Diese Funktion unterstützt 3d und lässt den Z-Index nicht fallen.

Diese Funktion unterstützt polyedrische Flächen.

Diese Funktion unterstützt Dreiecke und dreieckige unregelmäßige Netzoberflächen (TIN).

Beispiele

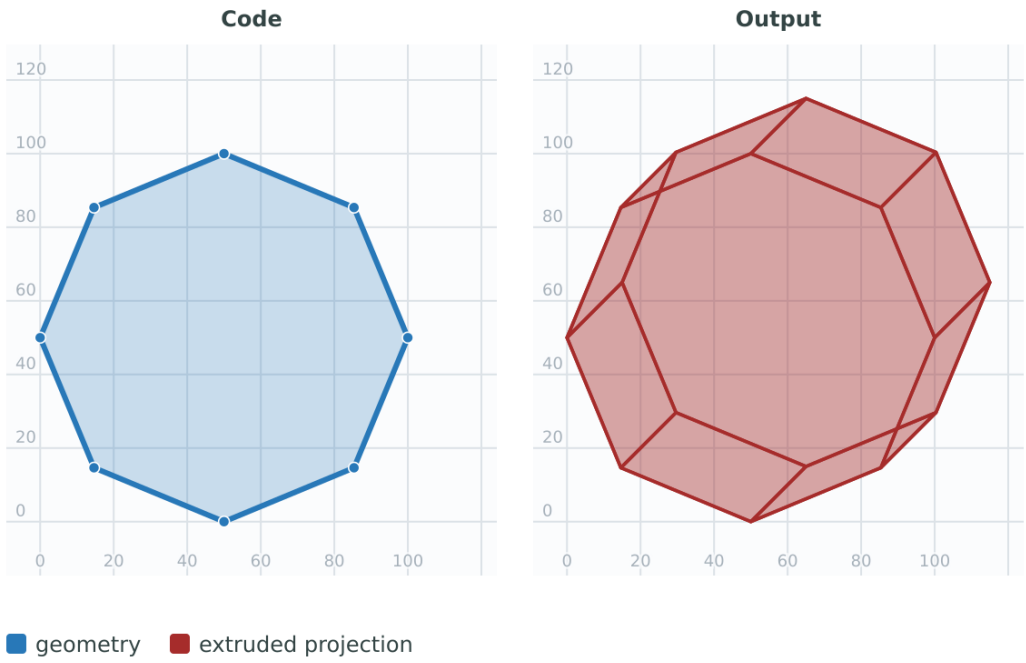
The figures use ST_Affine to project Z diagonally into XY after extrusion. The returned geometry from `CG_Extrude` remains a 3D PolyhedralSurface.

Extruding an octagonal Polygon 30 units along Z.

Code

```
WITH data AS (
  SELECT ST_Buffer('POINT(100 90)'::geometry, 50, 'quad_segs=2') AS geom
), projected AS (
  SELECT ST_Force2D(ST_Affine(geom,
           1, 0, 0.5, 0, 1, 0.5, 0, 0, 0, 0, 0, 0)) AS input_geom,
         ST_Force2D(ST_Affine(CG_Extrude(geom, 0, 0, 30),
           1, 0, 0.5, 0, 1, 0.5, 0, 0, 0, 0, 0, 0)) AS output_geom
  FROM data
)
SELECT input_geom AS input_geometry,
       output_geom AS extruded_projection
FROM projected;
```

Figure



Extruding a LineString 10 units along Z.

Code

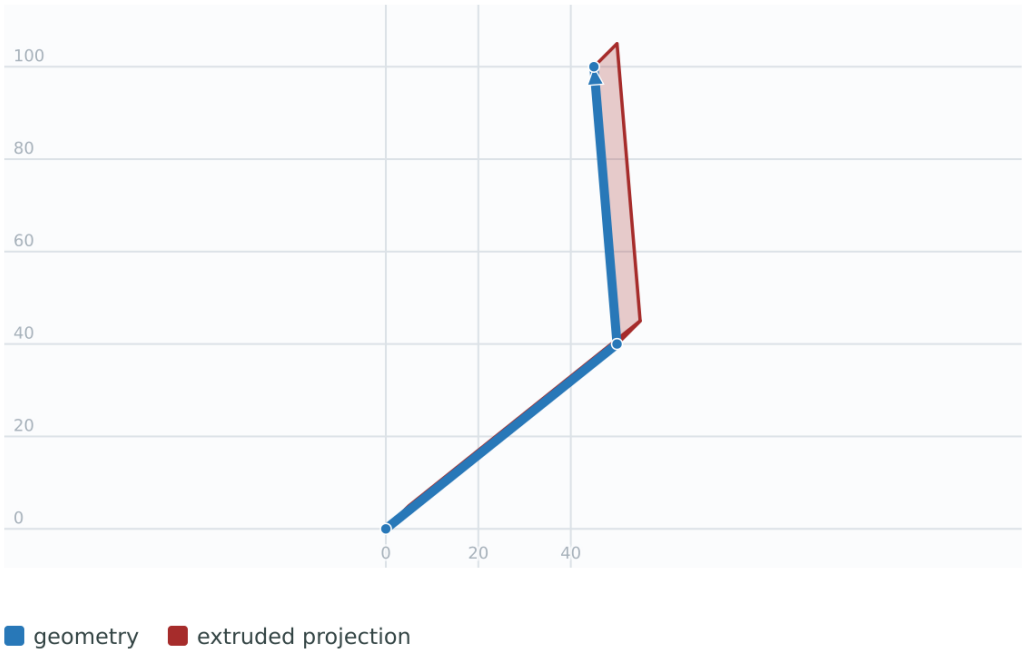
```
WITH data AS (
  SELECT 'LINESTRING(50 50,100 90,95 150)'::geometry AS geom
), projected AS (
  SELECT ST_Force2D(ST_Affine(geom,
           1, 0, 0.5, 0, 1, 0.5, 0, 0, 0, 0, 0, 0)) AS input_geom,
         ST_Force2D(ST_Affine(CG_Extrude(geom, 0, 0, 10),
           1, 0, 0.5, 0, 1, 0.5, 0, 0, 0, 0, 0, 0)) AS output_geom
  FROM data
)
SELECT input_geom AS input_geometry,
       output_geom AS extruded_projection
```

```
FROM projected;
```

Output

```
LINESTRING(50 50,100 90,95 150) | POLYHEDRALSURFACE(((50 50,100 90,105 95,55 55,50 50)),((100 90,95 150,100 155,105 ↩
    95,100 90)))
```

Figure



Siehe auch

ST_AsX3D, CG_ExtrudeStraightSkeleton

8.3.23 CG_ExtrudeStraightSkeleton

CG_ExtrudeStraightSkeleton — Gerade Skelett-Extrusion

Synopsis

geometry **CG_ExtrudeStraightSkeleton**(geometry geom, float roof_height, float body_height = 0);

Beschreibung

Berechnet eine Extrusion mit einer maximalen Höhe der Polygongeometrie.



Anmerkung
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.

Verfügbarkeit: 3.5.0 - erfordert SFCGAL >= 1.5.0.

Erfordert SFCGAL >= 1.5.0

Diese Methode benötigt ein SFCGAL-Backend.

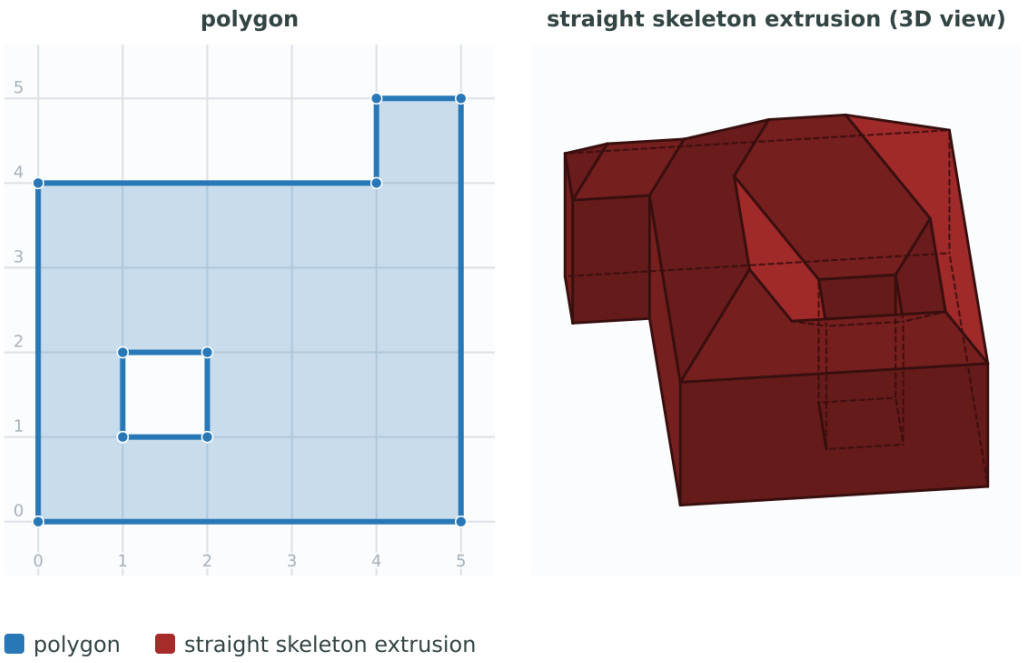
Beispiele

Extrude a polygon with a hole into a body and straight-skeleton roof.

Code

```
WITH data AS (
  SELECT 'POLYGON ((0 0,5 0,5 5,4 5,4 4,0 4,0 0),(1 1,1 2,2 2,2 1,1 1))'::geometry AS ↩
    input_polygon
)
SELECT input_polygon AS input_polygon,
       CG_ExtrudeStraightSkeleton(input_polygon, 3.0, 2.0) AS straight_skeleton_extrusion
FROM data;
```

Figure



Siehe auch

CG_Extrude, CG_ExtrudeStraightSkeleton, CG_StraightSkeleton, CG_StraightSkeletonPartition

8.3.24 CG_GreeneApproxConvexPartition

CG_GreeneApproxConvexPartition — Berechnet die approximale konvexe Partition der Polygongeometrie

Synopsis

geometry **CG_GreeneApproxConvexPartition**(geometry geom);

Beschreibung

Berechnet approximale monotone konvexe Partition der Polygongeometrie.



Anmerkung
Eine Partition eines Polygons P ist eine Menge von Polygonen, bei der sich die Innenräume der Polygone nicht schneiden und die Vereinigung der Polygone gleich dem Innenraum des ursprünglichen Polygons P ist. Die Funktionen CG_ApproxConvexPartition und CG_GreeneApproxConvexPartition erzeugen annähernd optimale konvexe Partitionen. Beide Funktionen erzeugen konvexe Zerlegungen, indem sie das Polygon zunächst in einfachere Polygone zerlegen; CG_ApproxConvexPartition verwendet eine Triangulation und CG_GreeneApproxConvexPartition eine monotone Partition. Diese beiden Funktionen garantieren, dass sie nicht mehr als das Vierfache der optimalen Anzahl konvexer Teile erzeugen, unterscheiden sich aber in ihrer Laufzeitkomplexität. Obwohl der auf Triangulation basierende Approximationsalgorithmus oft zu weniger konvexen Teilen führt, ist dies nicht immer der Fall.

Verfügbarkeit: 3.5.0 - erfordert SFCGAL >= 1.5.0.

Erfordert SFCGAL >= 1.5.0

 Diese Methode benötigt ein SFCGAL-Backend.

Beispiele

Greene's monotone-based approximation uses the same polygon and stays directly comparable in one panel.

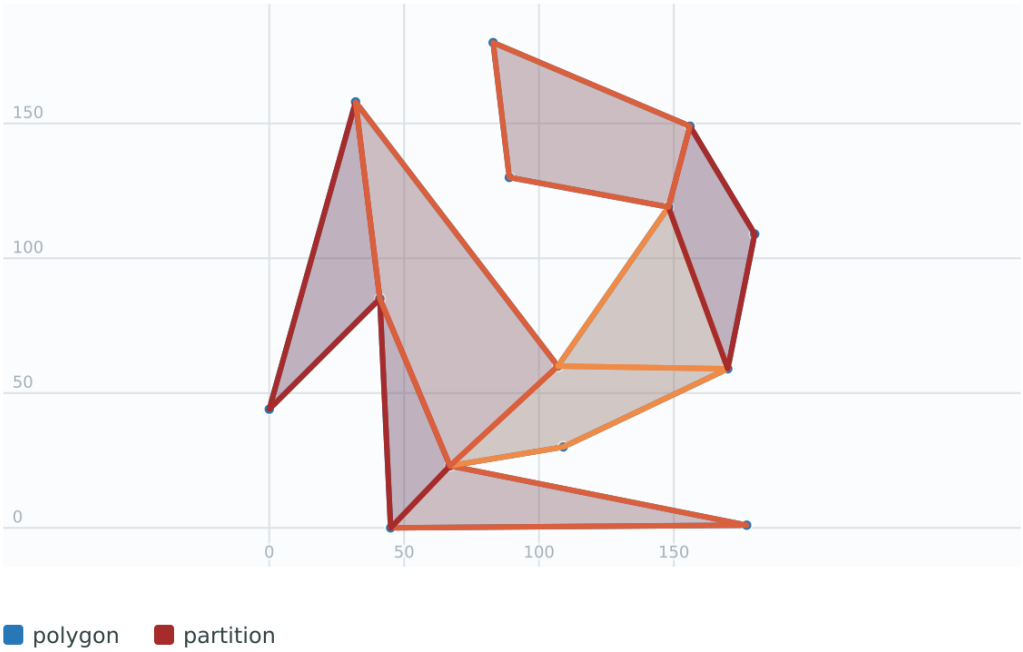
Code

```
WITH data AS (
  SELECT 'POLYGON((156 150,83 181,89 131,148 120,107 61,32 159,0 45,41 86,45 1,177 2,67  ↩
    24,109 31,170 60,180 110,156 150))'::geometry AS input_polygon
)
SELECT input_polygon AS input_polygon,
       CG_GreeneApproxConvexPartition(input_polygon) AS partition
FROM data;
```

Output

```
POLYGON((156 150,83 181,89 131,148 120,107 61,32 159,0 45,41 86,45 1,177 2,67 24,109 31,170 60,180 110,156 150)) | ↩
    GEOMETRYCOLLECTION(POLYGON((32 159,0 45,41 86,32 159)), POLYGON((45 1,177 2,67 24,45 1)), POLYGON((67 24,109 ↩
    31,170 60,107 61,67 24)), POLYGON((41 86,45 1,67 24,41 86)), POLYGON((107 61,32 159,41 86,67 24,107 61)), POLYGON ↩
    ((148 120,107 61,170 60,148 120)), POLYGON((148 120,170 60,180 110,156 150,148 120)), POLYGON((156 150,83 181,89 ↩
    131,148 120,156 150)))
```

Figure



Siehe auch

CG_YMonotonePartition, CG_ApproxConvexPartition, CG_OptimalConvexPartition

8.3.25 CG_MinkowskiSum

CG_MinkowskiSum — Führt die Minkowski-Summe aus

Synopsis

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

Beschreibung

Diese Funktion führt eine 2D-Minkowski-Summe eines Punktes, einer Linie oder eines Polygons mit einem Polygon durch.

Eine Minkowski-Summe zweier Geometrien A und B ist die Menge aller Punkte, die die Summe aller Punkte in A und B sind. Minkowski-Summen werden häufig in der Bewegungsplanung und im computergestützten Design verwendet. Mehr Details auf Wikipedia Minkowski Addition.

Der erste Parameter kann eine beliebige 2D-Geometrie sein (Punkt, Linienstring, Polygon). Wird eine 3D-Geometrie übergeben, so wird sie in 2D konvertiert, indem Z auf 0 gesetzt wird, was zu möglichen Ungültigkeitsfällen führen kann. Der zweite Parameter muss ein 2D-Polygon sein.

Die Implementierung verwendet CGAL 2D Minkowskisum.

Verfügbarkeit: 3.5.0

Diese Methode benötigt ein SFCGAL-Backend.

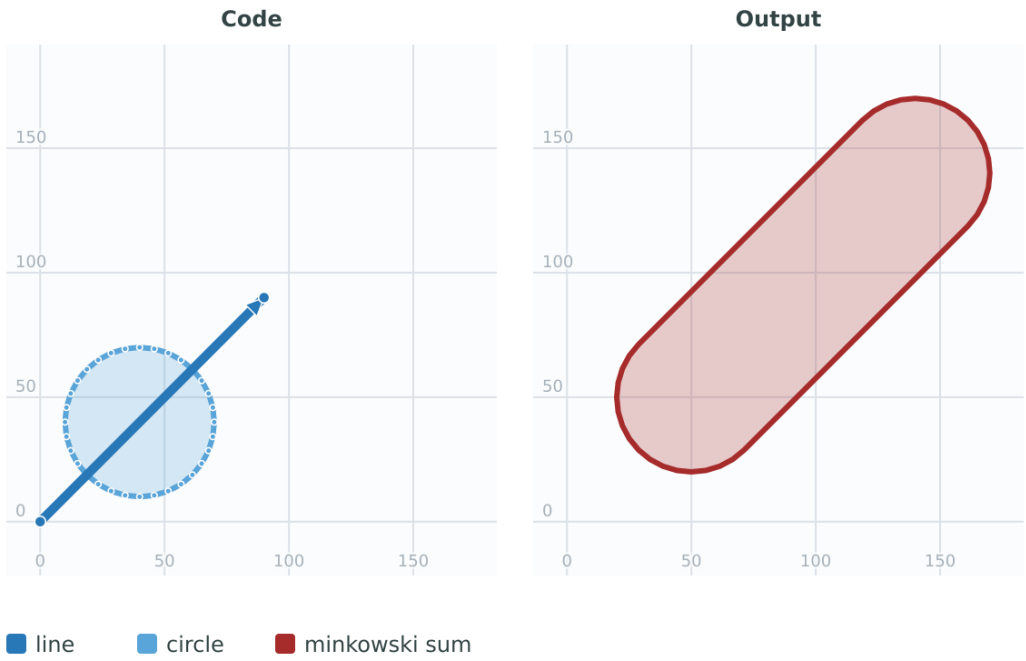
Beispiele

Minkowski sum of a LineString and a circular polygon. The line crosses the polygon, and the result sweeps the polygon along the line.

Code

```
WITH data AS (
  SELECT ST_MakeLine(ST_Point(10, 10), ST_Point(100, 100)) AS line,
         ST_Buffer('POINT(50 50)'::geometry, 30) AS circle
)
SELECT line AS input_line,
       circle AS input_circle,
       CG_MinkowskiSum(line, circle) AS minkowski_sum
FROM data;
```

Figure



Minkowski sum of a MultiPoint and a polygon. The result contains one translated copy of the polygon for each point.

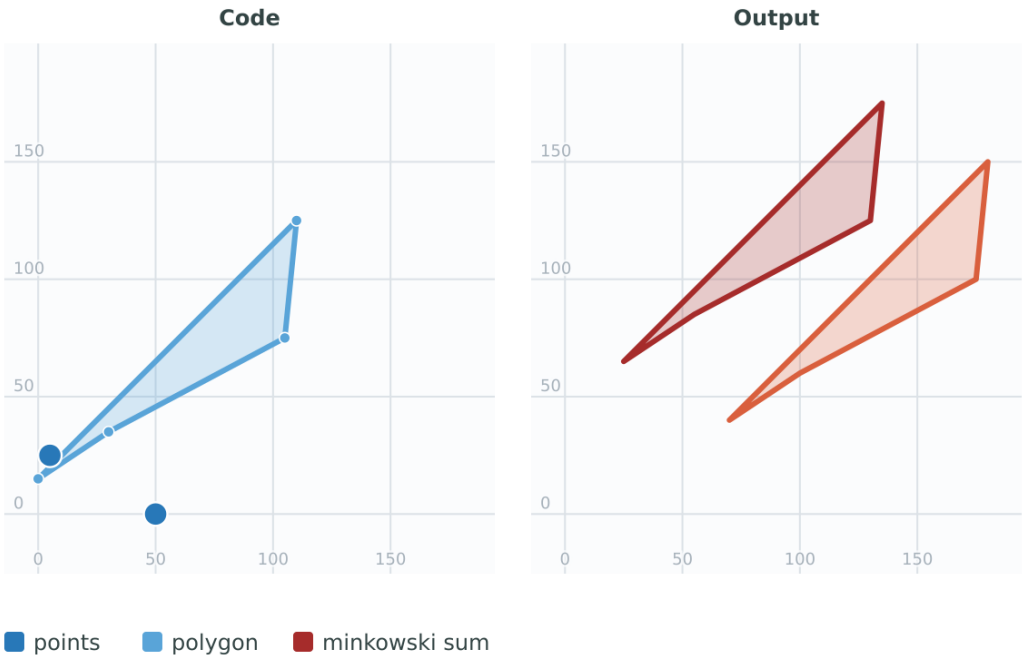
Code

```
WITH data AS (
  SELECT 'MULTIPOINT(25 50,70 25)'::geometry AS points,
         'POLYGON((130 150,20 40,50 60,125 100,130 150))'::geometry AS polygon
)
SELECT points AS input_points,
       polygon AS input_polygon,
       CG_MinkowskiSum(points, polygon) AS minkowski_sum
FROM data;
```

Output

```
MULTIPOINT((25 50),(70 25)) | POLYGON((130 150,20 40,50 60,125 100,130 150)) | MULTIPOLYGON(((45 90,75 110,150 150,155 ↩
    200,45 90)),((90 65,120 85,195 125,200 175,90 65)))
```

Figure



8.3.26 CG_OptimalAlphaShape

CG_OptimalAlphaShape — Berechnet eine Alpha-Form, die eine Geometrie umschließt, unter Verwendung eines "optimalen" Alpha-Wertes.

Synopsis

geometry **CG_OptimalAlphaShape**(geometry geom, boolean allow_holes = false, integer nb_components = 1);

Beschreibung

Berechnet die "optimale" Alpha-Form der Punkte in einer Geometrie. Die Alpha-Form wird mit einem Wert von α berechnet, der so gewählt wird, dass:

1. die Anzahl der Polygonelemente ist gleich oder kleiner als `nb_components` (Standardwert: 1)
2. alle Eingabepunkte sind in der Form enthalten

Das Ergebnis wird keine Löcher enthalten, es sei denn, das optionale Argument `allow_holes` wird als true angegeben.

Availability: 3.5.0 - requires SFCGAL >= 1.4.1.

Diese Methode benötigt ein SFCGAL-Backend.

Beispiele

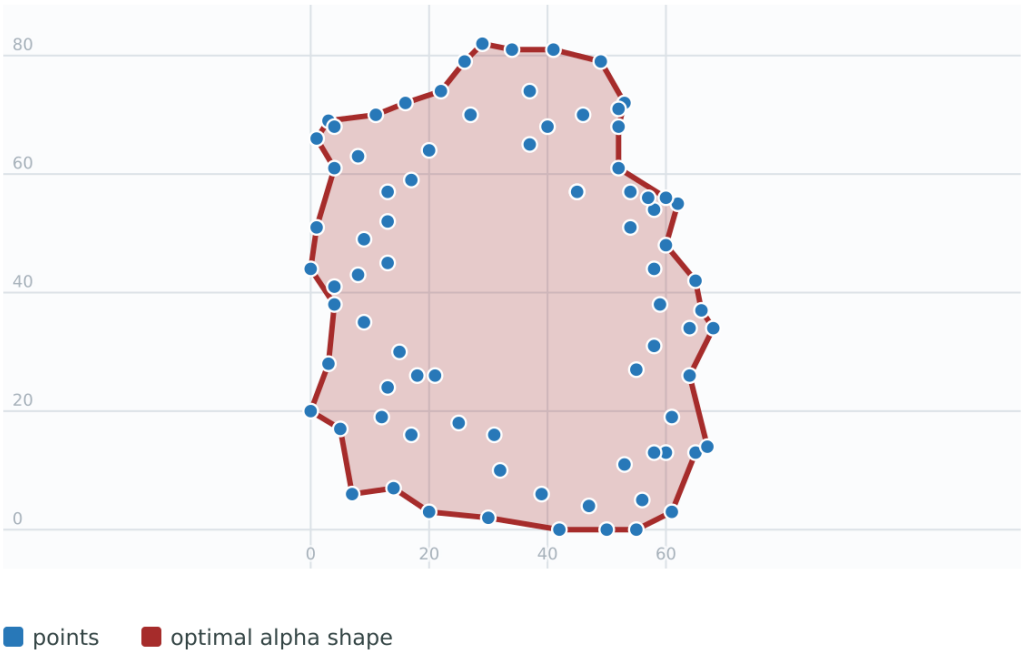
The original point cloud and the optimal alpha-shape share one comparison frame.

Code

```
WITH data AS (
  SELECT 'MULTIPOINT((63 84),(76 88),(68 73),(53 18),(91 50),(81 70),
(88 29),(24 82),(32 51),(37 23),(27 54),(84 19),(75 87),(44 42),(77 67),(90 30),(36 61),(32 ↩
    65),
(81 47),(88 58),(68 73),(49 95),(81 60),(87 50),
(78 16),(79 21),(30 22),(78 43),(26 85),(48 34),(35 35),(36 40),(31 79),(83 29),(27 84),(52 ↩
    98),(72 95),(85 71),
(75 84),(75 77),(81 29),(77 73),(41 42),(83 72),(23 36),(89 53),(27 57),(57 97),(27 77),(39 ↩
    88),(60 81),
(80 72),(54 32),(55 26),(62 22),(70 20),(76 27),(84 35),(87 42),(82 54),(83 64),(69 86),(60 ↩
    90),(50 86),(43 80),(36 73),
(36 68),(40 75),(24 67),(23 60),(26 44),(28 33),(40 32),(43 19),(65 16),(73 16),(38 46),(31 ↩
    59),(34 86),(45 90),(64 97))'::geometry AS input_points
)
SELECT input_points AS input_points,

         CG_OptimalAlphaShape(input_points) AS optimal_alpha_shape
FROM data;
```

Figure



Allowing holes keeps the same outer shell but carves a central hole.

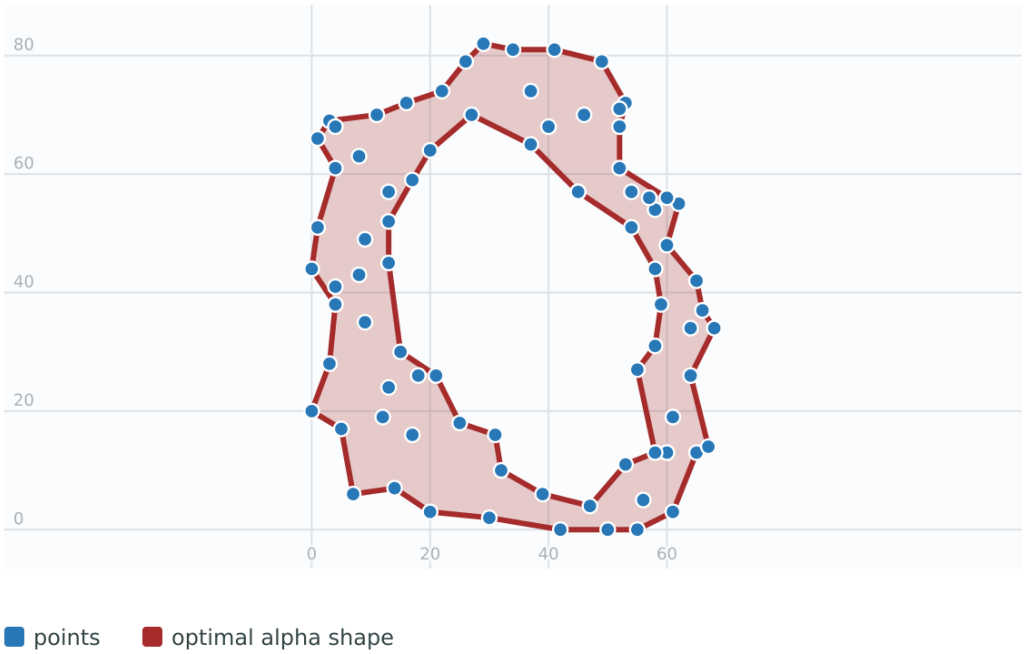
Code

```
WITH data AS (
  SELECT 'MULTIPOINT((63 84),(76 88),(68 73),(53 18),(91 50),(81 70),(88 29),(24 82),(32 51) ↩
    ,(37 23),(27 54),(84 19),(75 87),(44 42),(77 67),(90 30),(36 61),(32 65),(81 47),(88 58) ↩
    ,(68 73),(49 95),(81 60),(87 50),
(78 16),(79 21),(30 22),(78 43),(26 85),(48 34),(35 35),(36 40),(31 79),(83 29),(27 84),(52 ↩
    98),(72 95),(85 71),
(75 84),(75 77),(81 29),(77 73),(41 42),(83 72),(23 36),(89 53),(27 57),(57 97),(27 77),(39 ↩
    88),(60 81),
(80 72),(54 32),(55 26),(62 22),(70 20),(76 27),(84 35),(87 42),(82 54),(83 64),(69 86),(60 ↩
    90),(50 86),(43 80),(36 73),
(36 68),(40 75),(24 67),(23 60),(26 44),(28 33),(40 32),(43 19),(65 16),(73 16),(38 46),(31 ↩
    59),(34 86),(45 90),(64 97))'::geometry AS input_points
```

```
)
SELECT input_points AS input_points,

         CG_OptimalAlphaShape(input_points, allow_holes => true) AS optimal_alpha_shape
FROM data;
```

Figure



Siehe auch

ST_ConcaveHull, CG_AlphaShape

8.3.27 CG_OptimalConvexPartition

CG_OptimalConvexPartition — Berechnet eine optimale konvexe Partition der Polygongeometrie

Synopsis

geometry **CG_OptimalConvexPartition**(geometry geom);

Beschreibung

Berechnet eine optimale konvexe Partition der Polygongeometrie.



Anmerkung
Eine Partition eines Polygons P ist eine Menge von Polygonen, die so beschaffen ist, dass sich die Innenräume der Polygone nicht schneiden und die Vereinigung der Polygone gleich dem Innenraum des ursprünglichen Polygons P ist. CG_OptimalConvexPartition erzeugt eine Partition, die hinsichtlich der Anzahl der Teile optimal ist.

Verfügbarkeit: 3.5.0 - erfordert SFCGAL >= 1.5.0.

Erfordert SFCGAL >= 1.5.0

Diese Methode benötigt ein SFCGAL-Backend.

Beispiele

The optimal convex partition uses the same polygon and produces the smallest documented piece count.

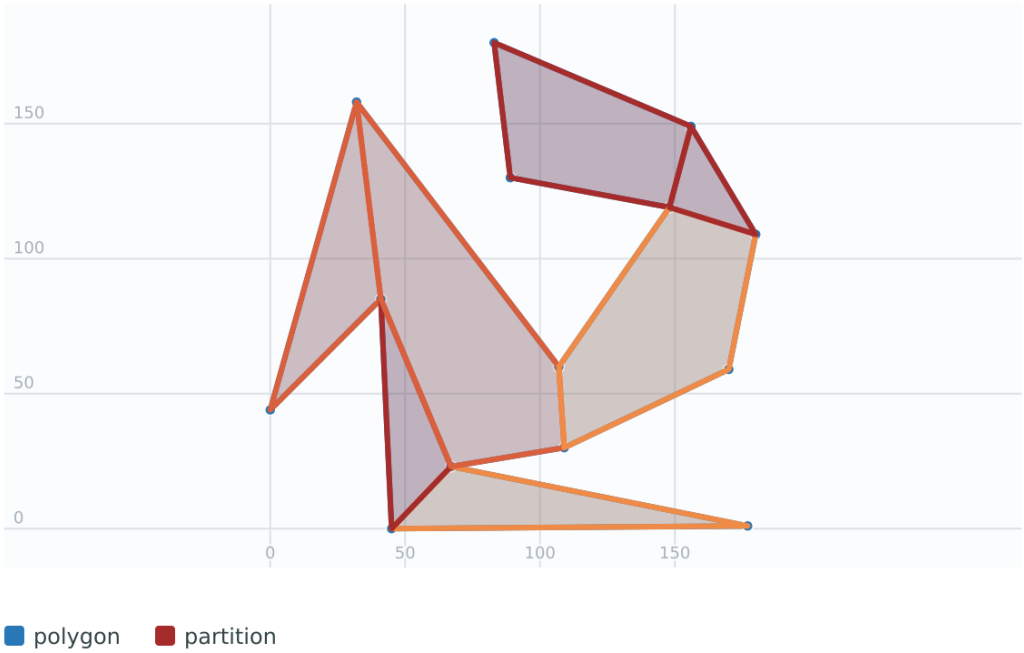
Code

```
WITH data AS (
  SELECT 'POLYGON((156 150,83 181,89 131,148 120,107 61,32 159,0 45,41 86,45 1,177 2,67 ↩
    24,109 31,170 60,180 110,156 150))'::geometry AS input_polygon
)
SELECT input_polygon AS input_polygon,
       CG_OptimalConvexPartition(input_polygon) AS partition
FROM data;
```

Output

```
POLYGON((156 150,83 181,89 131,148 120,107 61,32 159,0 45,41 86,45 1,177 2,67 24,109 31,170 60,180 110,156 150)) | ↩
    GEOMETRYCOLLECTION(POLYGON((156 150,83 181,89 131,148 120,156 150)), POLYGON((32 159,0 45,41 86,32 159)), POLYGON ↩
    ((45 1,177 2,67 24,45 1)), POLYGON((41 86,45 1,67 24,41 86)), POLYGON((107 61,32 159,41 86,67 24,109 31,107 61)), ↩
    POLYGON((148 120,107 61,109 31,170 60,180 110,148 120)), POLYGON((156 150,148 120,180 110,156 150)))
```

Figure



Siehe auch

CG_YMonotonePartition, CG_ApproxConvexPartition, CG_GreeneApproxConvexPartition

8.3.28 CG_StraightSkeleton

CG_StraightSkeleton — Berechnet die konvexe Hülle einer Geometrie.

Synopsis

geometry **CG_StraightSkeleton**(geometry geom, boolean use_distance_as_m = false);

Beschreibung

Computes the straight skeleton of a polygon by tracing the inward propagation of its edges. When *use_distance_as_m* is `true`, the M coordinate of each output point stores its distance from the input boundary.

Verfügbarkeit: 3.5.0

Requires SFCGAL >= 1.3.8 for option use_distance_as_m



Anmerkung
Diese Funktion ignoriert die Z-Dimension. Sie liefert immer ein 2D-Ergebnis, auch wenn sie auf eine 3D-Geometrie angewendet wird.

 Diese Methode benötigt ein SFCGAL-Backend.

 Diese Funktion unterstützt polyedrische Flächen.

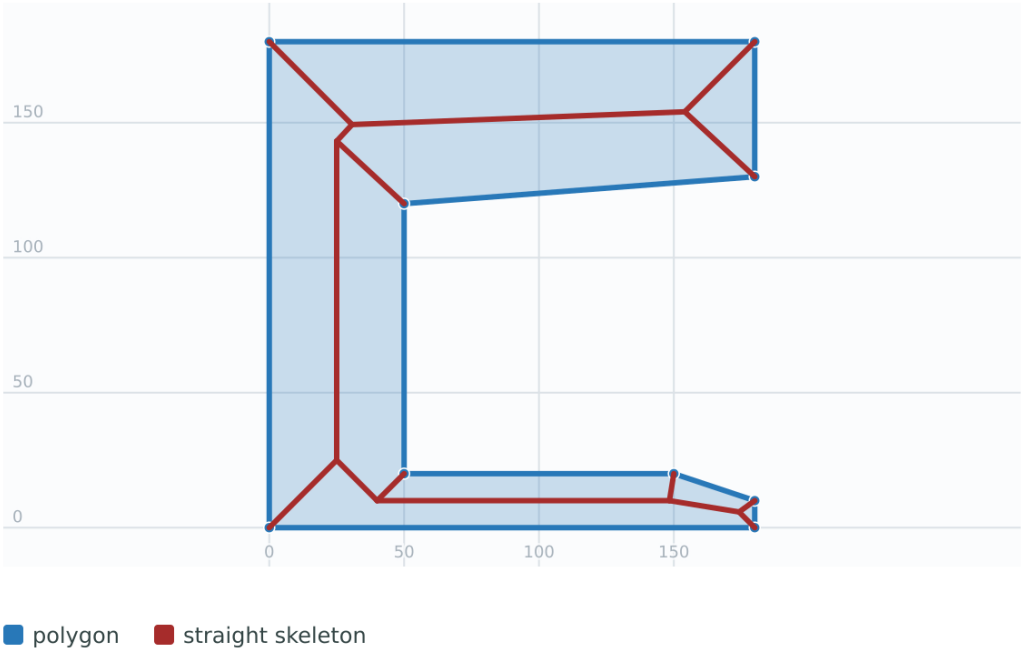
 Diese Funktion unterstützt Dreiecke und dreieckige unregelmäßige Netzoberflächen (TIN).

Beispiele

Code

```
WITH data AS (
  SELECT 'POLYGON((190 190,10 190,10 10,190 10,190 20,160 30,60 30,60 130,190 140,190 190))' ↩
    ::geometry AS polygon
)
SELECT CG_StraightSkeleton(polygon) AS straight_skeleton
FROM data;
```

Figure



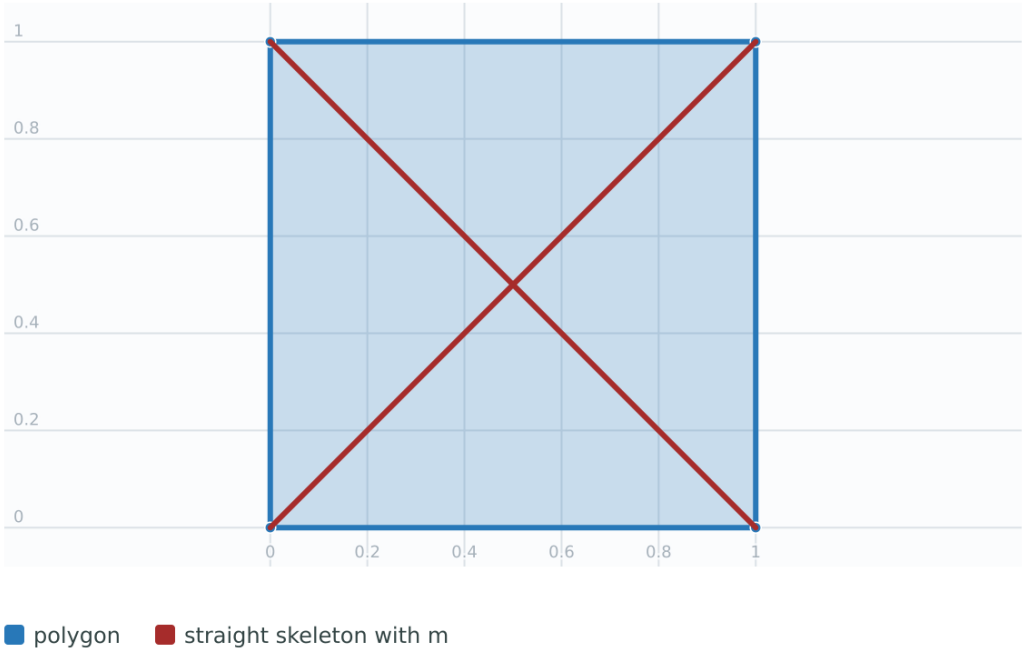
Code

```
WITH data AS (
  SELECT 'POLYGON((0 0,1 0,1 1,0 1,0 0))'::geometry AS polygon
)
SELECT CG_StraightSkeleton(polygon, true) AS straight_skeleton_with_m
FROM data;
```

Output

```
MULTILINESTRING M ((0 0 0,0.5 0.5 0.5),(1 0 0,0.5 0.5 0.5),(1 1 0,0.5 0.5 0.5),(0 1 0,0.5 0.5 0.5))
```

Figure



Note that valid inputs with rings that touch at a single point will raise an error.

Code

```
SELECT CG_StraightSkeleton('POLYGON((0 0,3 0,3 3,0 3,0 0),(0 0,1 2,2 1,0 0))'));
```

Output

```
NOTICE:  During straight_skeleton(A) :
NOTICE:    with A: POLYGON((0/1 0/1,3/1 0/1,3/1 3/1,0/1 3/1,0/1 0/1),(0/1 0/1,1/1 2/1,2/1 1/1,0/1 0/1))
ERROR:  straight skeleton of Polygon with point touching rings is not implemented.
```

Siehe auch

CG_StraightSkeletonPartition, CG_ExtrudeStraightSkeleton

8.3.29 CG_Tesselate

CG_Tesselate — Führt eine Oberflächentesselierung eines Polygons oder einer Polyederfläche durch und gibt diese als TIN oder Sammlung von TINS zurück

Synopsis

geometry **CG_Tesselate**(geometry geom);

Beschreibung

Nimmt als Eingabe eine Oberfläche wie MULTI(POLYGON) oder POLYHEDRALSURFACE und gibt eine TIN-Darstellung über den Prozess der Tesselierung mit Dreiecken zurück.



Anmerkung
ST_TriangulatePolygon funktioniert ähnlich wie diese Funktion, mit dem Unterschied, dass sie eine Geometriesammlung von Polygonen anstelle eines TINs zurückgibt und auch nur mit 2D-Geometrien funktioniert.

Verfügbarkeit: 3.5.0

 Diese Methode benötigt ein SFCGAL-Backend.

 Diese Funktion unterstützt 3d und lässt den Z-Index nicht fallen.

 Diese Funktion unterstützt polyedrische Flächen.

 Diese Funktion unterstützt Dreiecke und dreieckige unregelmäßige Netzoberflächen (TIN).

Beispiele

Tessellation divides each face of a cube into triangles.

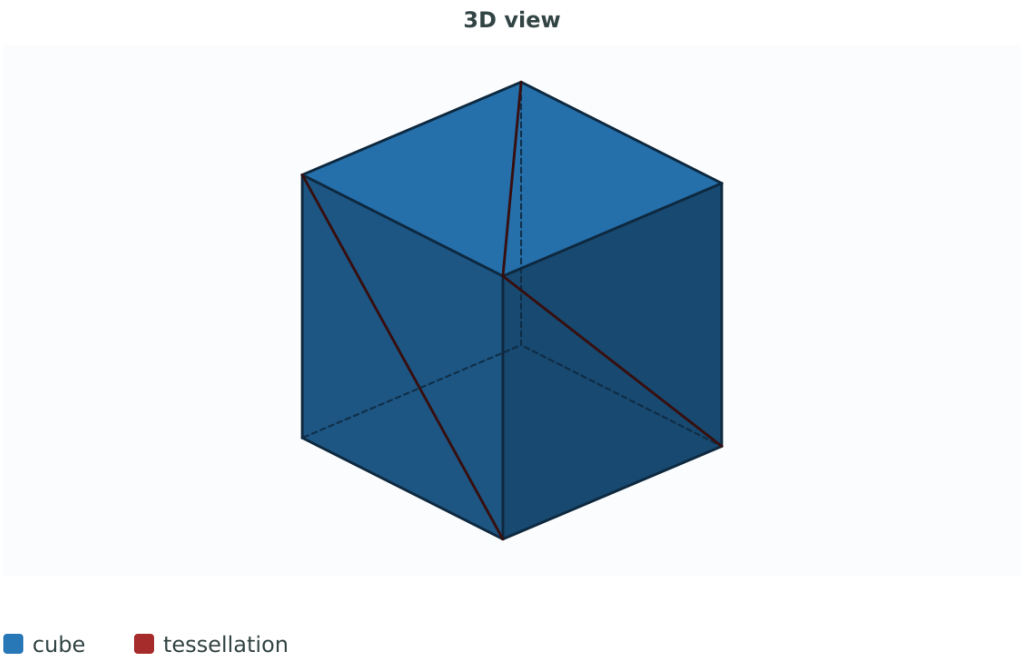
Code

```
WITH data AS (
  SELECT 'POLYHEDRALSURFACE Z( ((0 0 0,0 0 1,0 1 1,0 1 0,0 0 0)),
((0 0 0,0 1 0,1 1 0,1 0 0,0 0 0)),((0 0 0,1 0 0,1 0 1,0 0 1,0 0 0)),
((1 1 0,1 1 1,1 0 1,1 0 0,1 1 0)),
((0 1 0,0 1 1,1 1 1,1 1 0,0 1 0)),((0 0 1,1 0 1,1 1 1,0 1 1,0 0 1)) )'::geometry AS cube
)
SELECT CG_Tesselate(cube) AS tessellation
FROM data;
```

Output

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

Figure



Tessellation of a Polygon.

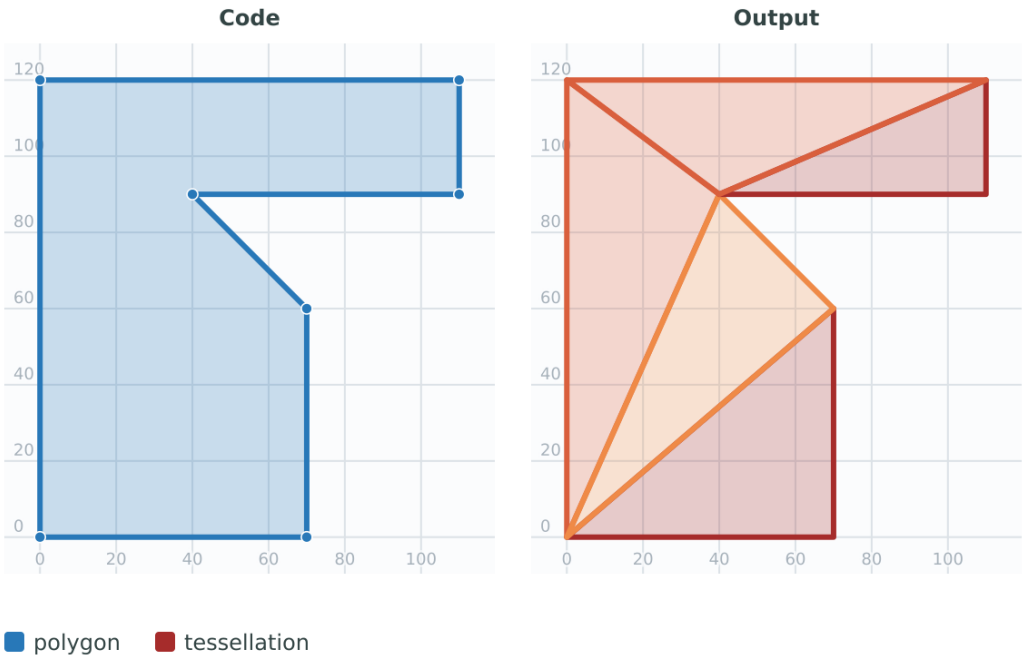
Code

```
WITH data AS (
  SELECT 'POLYGON((10 190,10 70,80 70,80 130,50 160,120 160,120 190,10 190))'::geometry AS
    polygon
)
SELECT polygon AS input_polygon,
       CG_Tesselate(polygon) AS tessellation
FROM data;
```

Output

```
POLYGON((10 190,10 70,80 70,80 130,50 160,120 160,120 190,10 190)) | TIN(((80 130,10 70,80 70,80 130)),((50 160,10
    190,10 70,50 160)),((80 130,50 160,10 70,80 130)),((120 160,120 190,50 160,120 160)),((120 190,10 190,50 160,120
    190)))
```

Figure



Siehe auch

CG_ConstrainedDelaunayTriangles, ST_DelaunayTriangles, ST_TriangulatePolygon

8.3.30 CG_Triangulate

CG_Triangulate — Triangulates a polygonal geometry

Synopsis

geometry **CG_Triangulate**(geometry geom);

Beschreibung

Triangulates a polygonal geometry.

Performed by the SFCGAL module



Anmerkung
NOTE: this function returns a geometry representing the triangulated result.

Verfügbarkeit: 3.5.0


Diese Methode benötigt ein SFCGAL-Backend.

Beispiele mit dem geometrischen Datentyp

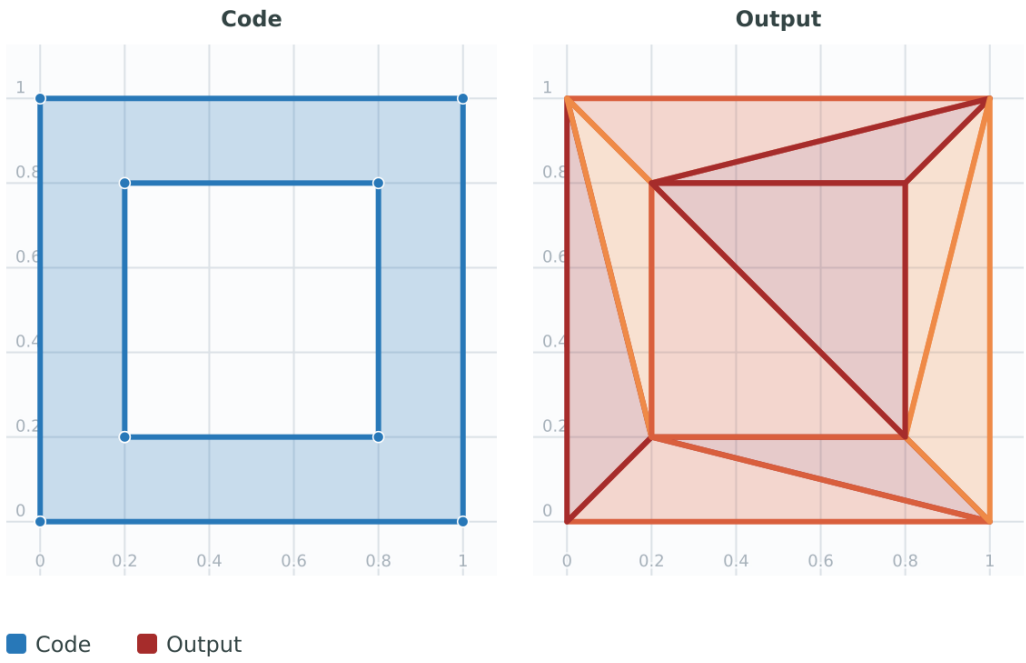
Code

```
SELECT CG_Triangulate(
  'POLYGON((0 0,1 0,1 1,0 1,0 0),
           (0.2 0.2,0.2 0.8,0.8 0.8,0.8 0.2,0.2 0.2))'
);
```

Output

```
TIN(((0.8 0.2,0.2 0.2,1 0,0.8 0.2)),((0.2 0.2,0 0,1 0,0.2 0.2)),((1 1,0.8 0.8,0.8 0.2,1 1)),((0 1,0 0,0.2 0.2,0 1)),((0 ↩
     1,0.2 0.8,1 1,0 1)),((0 1,0.2 0.2,0.2 0.8,0 1)),((0.2 0.8,0.8 0.8,1 1,0.2 0.8)),((0.2 0.8,0.2 0.2,0.8 0.2,0.2  ↩
    0.8)),((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)))
```

Figure



Siehe auch

CG_ConstrainedDelaunayTriangles, ST_DelaunayTriangles, ST_TriangulatePolygon

8.3.31 CG_Visibility

CG_Visibility — Berechnen eines Sichtbarkeitspolygons aus einem Punkt oder einem Segment in einer Polygongeometrie

Synopsis

geometry **CG_Visibility**(geometry polygon, geometry point);
geometry **CG_Visibility**(geometry polygon, geometry pointA, geometry pointB);

Beschreibung

Returns the portion of a polygon visible from an observer point, or from the observer segment defined by `pointA` and `pointB`.

Verfügbarkeit: 3.5.0 - erfordert SFCGAL >= 1.5.0.

Erfordert SFCGAL >= 1.5.0

Diese Methode benötigt ein SFCGAL-Backend.

Diese Funktion unterstützt 3d und lässt den Z-Index nicht fallen.

Diese Funktion unterstützt polyedrische Flächen.

Diese Funktion unterstützt Dreiecke und dreieckige unregelmäßige Netzoberflächen (TIN).

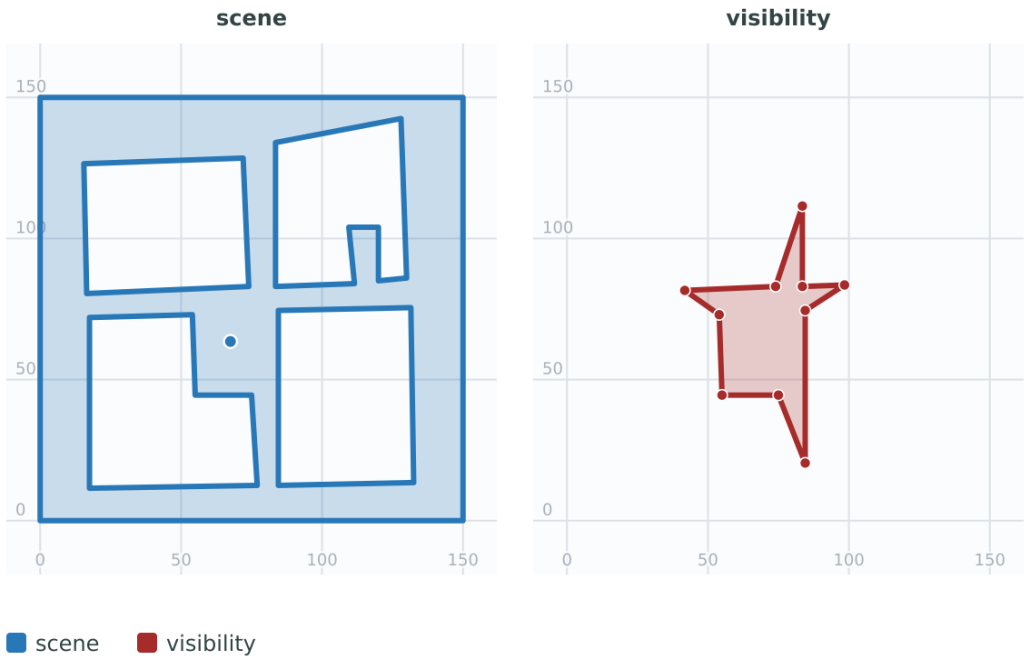
Beispiele

Visibility from a point inside a polygon with holes.

Code

```
WITH data AS (
  SELECT 'POLYGON((23.5 23.5,23.5 173.5,173.5 173.5,173.5 23.5,23.5 23.5),(108 98,108 36,156 ↩
    37,155 99,108 98),(107 157.5,107 106.5,135 107.5,133 127.5,143.5 127.5,143.5 ↩
   108.5,153.5 109.5,151.5 166,107 157.5),(41 95.5,41 35,100.5 36,98.5 68,78.5 68,77.5 ↩
   96.5,41 95.5),(39 150,40 104,97.5 106.5,95.5 152,39 150))'::geometry AS polygon,
        'POINT(91 87)'::geometry AS observer
)
SELECT ST_Collect(polygon, observer) AS input_scene,
       CG_Visibility(polygon, observer) AS visibility
FROM data;
```

Figure



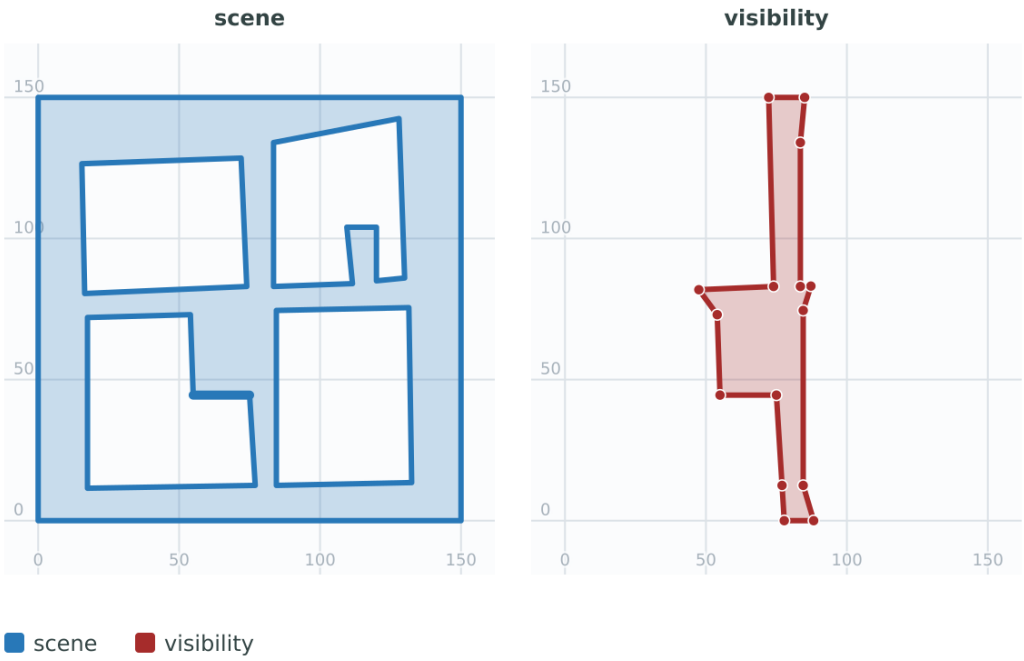
Visibility from a segment inside the same polygon.

Code

```
WITH data AS (
  SELECT 'POLYGON((23.5 23.5,23.5 173.5,173.5 173.5,173.5 23.5,23.5 23.5),(108 98,108 36,156 ↩
    37,155 99,108 98),(107 157.5,107 106.5,135 107.5,133 127.5,143.5 127.5,143.5 ↩
   108.5,153.5 109.5,151.5 166,107 157.5),(41 95.5,41 35,100.5 36,98.5 68,78.5 68,77.5 ↩
   96.5,41 95.5),(39 150,40 104,97.5 106.5,95.5 152,39 150))'::geometry AS polygon,
```

```
        'LINESTRING(78.5 68,98.5 68)'::geometry AS observer
)
SELECT ST_Collect(polygon, observer) AS input_scene,
       CG_Visibility(polygon,
                     ST_StartPoint(observer),
                     ST_EndPoint(observer)) AS visibility
FROM data;
```

Figure



8.3.32 CG_YMonotonePartition

CG_YMonotonePartition — Berechnet die y-monotone Partition der Polygongeometrie

Synopsis

geometry **CG_YMonotonePartition**(geometry geom);

Beschreibung

Berechnet die y-monotone Partition der Polygongeometrie.



Anmerkung
Eine Partition eines Polygons P ist eine Menge von Polygonen, die so beschaffen ist, dass sich die Innenräume der Polygone nicht schneiden und die Vereinigung der Polygone gleich dem Innenraum des ursprünglichen Polygons P ist. Ein y-monotones Polygon ist ein Polygon, dessen Scheitelpunkte v1,...,vn in zwei Ketten v1,...,vk und vk,...,vn,v1 unterteilt werden können, so dass jede horizontale Linie eine der beiden Ketten höchstens einmal schneidet. Dieser Algorithmus garantiert keine Begrenzung der Anzahl der erzeugten Polygone in Bezug auf die optimale Anzahl.

Verfügbarkeit: 3.5.0 - erfordert SFCGAL >= 1.5.0.

Erfordert SFCGAL >= 1.5.0

 Diese Methode benötigt ein SFCGAL-Backend.

Beispiele

The same polygon can also be partitioned into y-monotone pieces for downstream processing.

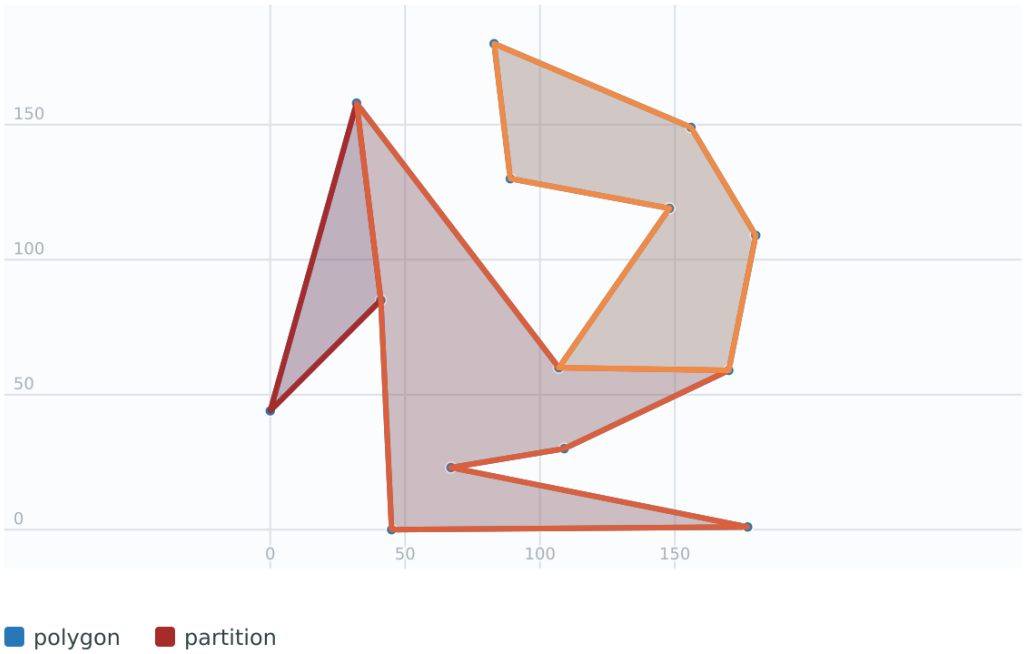
Code

```
WITH data AS (
  SELECT 'POLYGON((156 150,83 181,89 131,148 120,107 61,32 159,0 45,41 86,45 1,177 2,67 ↩
    24,109 31,170 60,180 110,156 150))'::geometry AS input_polygon
)
SELECT input_polygon AS input_polygon,
       CG_YMonotonePartition(input_polygon) AS partition
FROM data;
```

Output

```
POLYGON((156 150,83 181,89 131,148 120,107 61,32 159,0 45,41 86,45 1,177 2,67 24,109 31,170 60,180 110,156 150)) | ↩
    GEOMETRYCOLLECTION(POLYGON((32 159,0 45,41 86,32 159)), POLYGON((107 61,32 159,41 86,45 1,177 2,67 24,109 31,170 ↩
    60,107 61)), POLYGON((156 150,83 181,89 131,148 120,107 61,170 60,180 110,156 150)))
```

Figure



Siehe auch

CG_ApproxConvexPartition, CG_GreeneApproxConvexPartition, CG_OptimalConvexPartition

8.3.33 CG_StraightSkeletonPartition

CG_StraightSkeletonPartition — Computes the straight skeleton partition of a polygon.

Synopsis

geometry **CG_StraightSkeletonPartition**(geometry geom, boolean auto_orientation);

Beschreibung

Computes the straight skeleton partition of the input polygon geometry *geom*. The straight skeleton is a partitioning of the polygon into faces formed by tracing the collapse of its edges. If *auto_orientation* is set to true, the function will automatically adjust the orientation of the input polygon to ensure correct results.

Availability: 3.6.0 - requires SFCGAL >= 2.0.0.

Diese Methode benötigt ein SFCGAL-Backend.

Beispiele

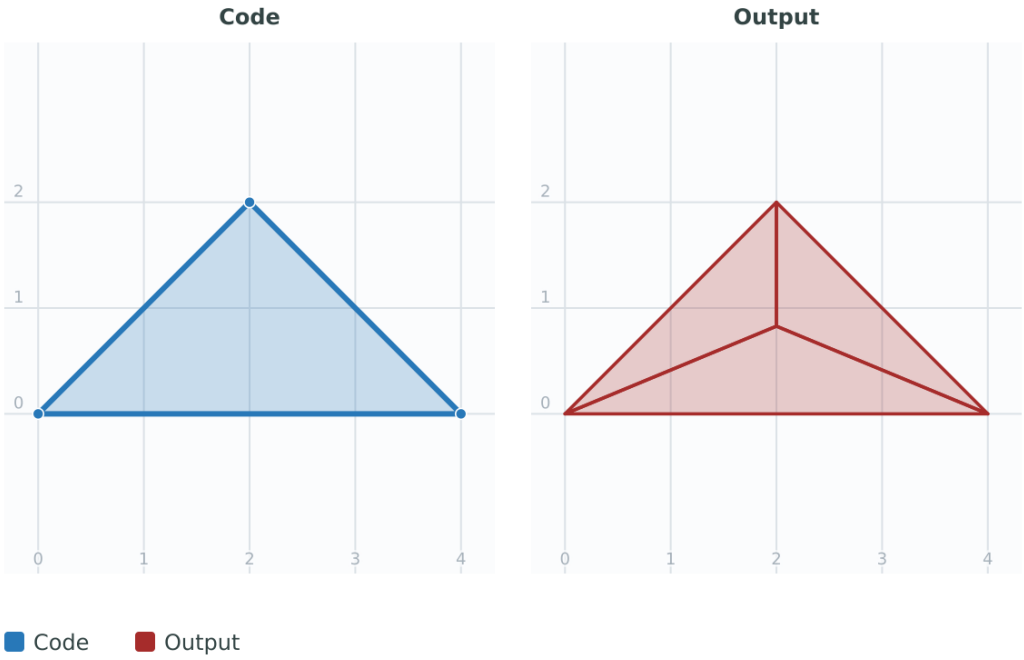
Code

```
SELECT CG_StraightSkeletonPartition('POLYGON((0 0,4 0,2 2,0 0))', true);
```

Output

```
POLYHEDRALSURFACE(((0 0,2 0.82842712474619,2 2,0 0)),((4 0,2 0.82842712474619,0 0,4 0)),((2 2,2 0.82842712474619,4 0,2 ↩
    2)))
```

Figure

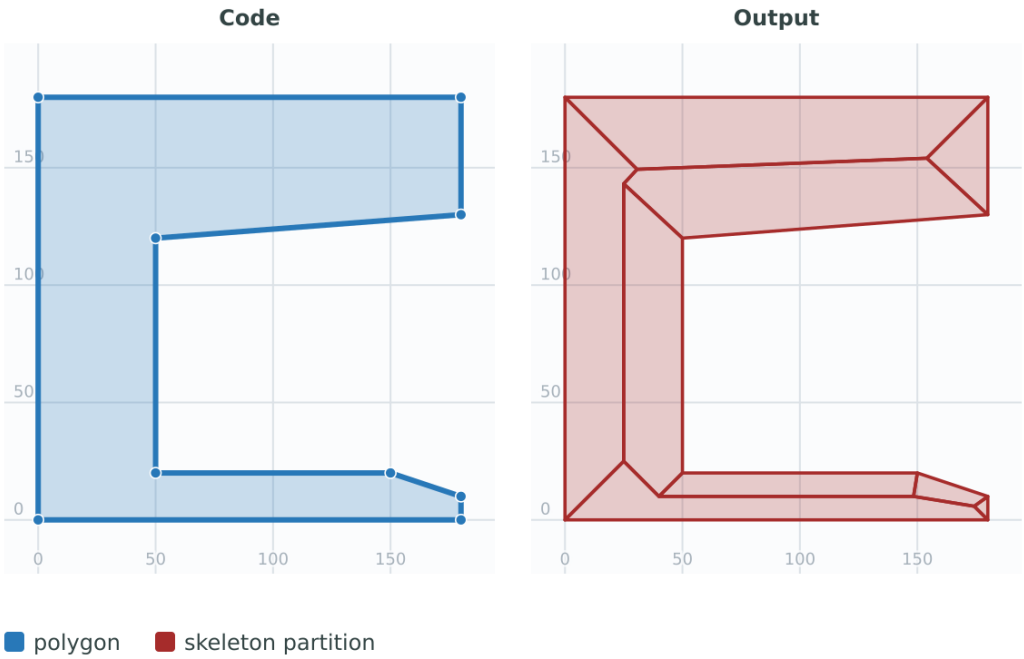


Straight skeleton partition of a polygon.

Code

```
WITH data AS (
  SELECT 'POLYGON((190 190,10 190,10 10,190 10,190 20,160 30,60 30,60 130,190 140,190 190))' ↩
    ::geometry AS polygon
)
SELECT polygon AS input_polygon,
       CG_StraightSkeletonPartition(polygon, true) AS skeleton_partition
FROM data;
```

Figure



Siehe auch

CG_StraightSkeleton, ST_Polygonize

8.3.34 CG_3DBuffer

CG_3DBuffer — Computes a 3D buffer around a geometry.

Synopsis

geometry **CG_3DBuffer**(geometry geom, float8 radius, integer segments, integer buffer_type);

Beschreibung

Generates a 3D buffer around the input geometry *geom* with a specified *radius*. The buffer is constructed in 3D space, creating a volumetric representation of the geometry's surroundings. The *segments* parameter defines the number of segments used to approximate the curved sections of the buffer, with a minimum value of 4 segments required. The *buffer_type* specifies the type of buffer to create: 0: Rounded buffer (default) 1: Flat buffer 2: Square buffer

Input geometry must be a Point or LineString.

Availability: 3.6.0 - requires SFCGAL >= 2.0.0

Diese Methode benötigt ein SFCGAL-Backend.

Beispiele

Code

```
SELECT CG_3DBuffer('POINT(0 0 0)', 1, 8, 0);
```

Output

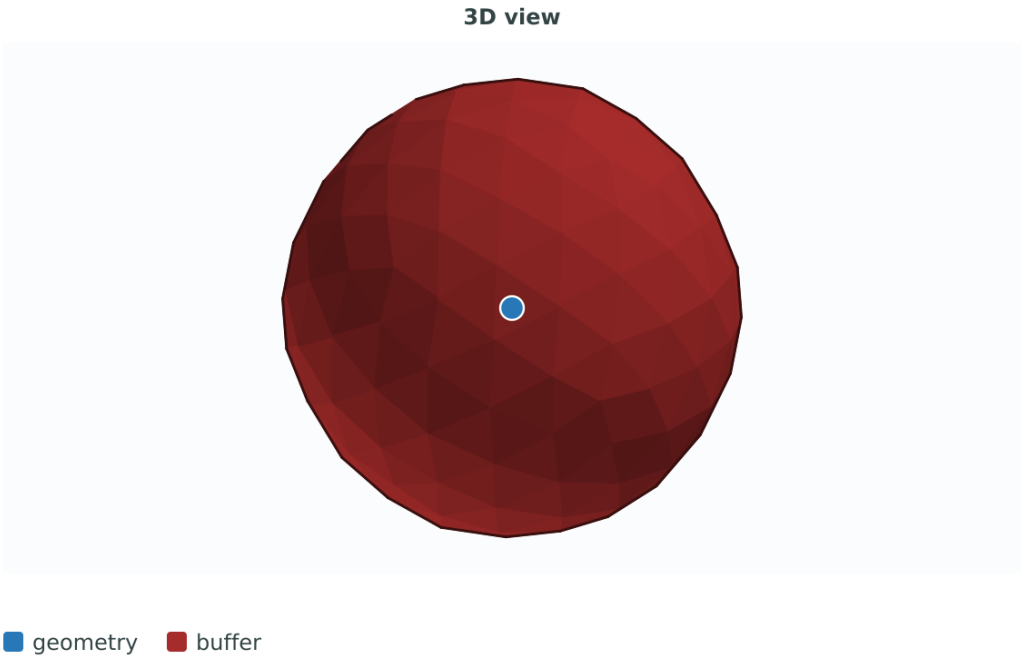
```
POLYHEDRALSURFACE Z (((0 0 1, 0.5 -0.5 0.71, 0 -0.71 0.71, 0 0 1)), ... )
```

A rounded point buffer using 32 segments.

Code

```
SELECT
  'POINT(100 90)'::geometry AS input_geometry,
  CG_3DBuffer('POINT(100 90)'::geometry, 50, 32, 0) AS buffer;
```

Figure

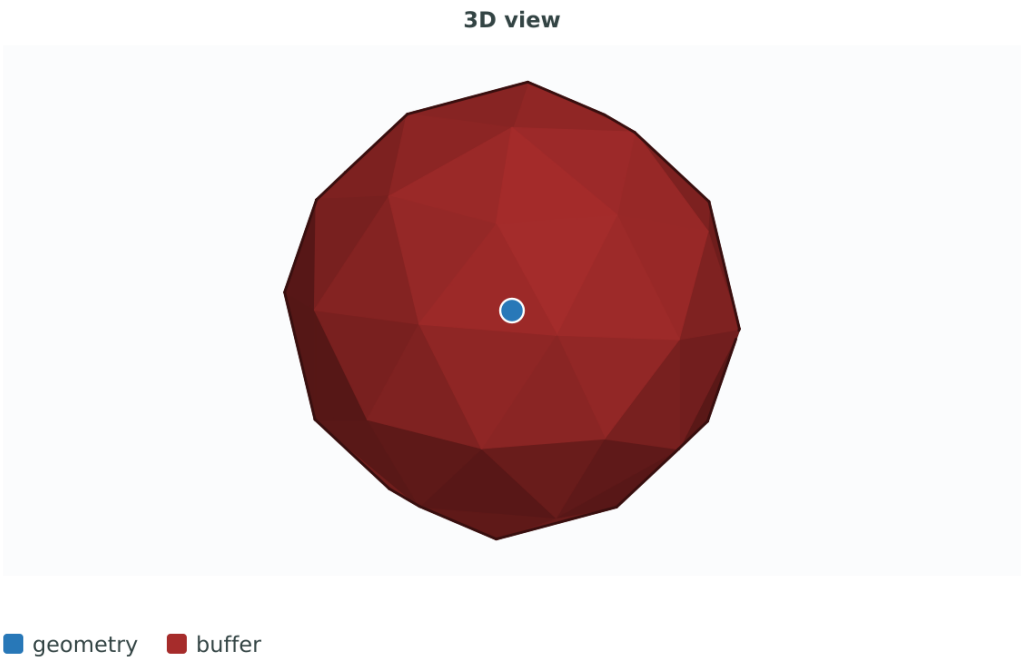


A coarser rounded point buffer using 5 segments.

Code

```
SELECT
  'POINT(100 90)'::geometry AS input_geometry,
  CG_3DBuffer('POINT(100 90)'::geometry, 50, 5, 0) AS buffer;
```

Figure

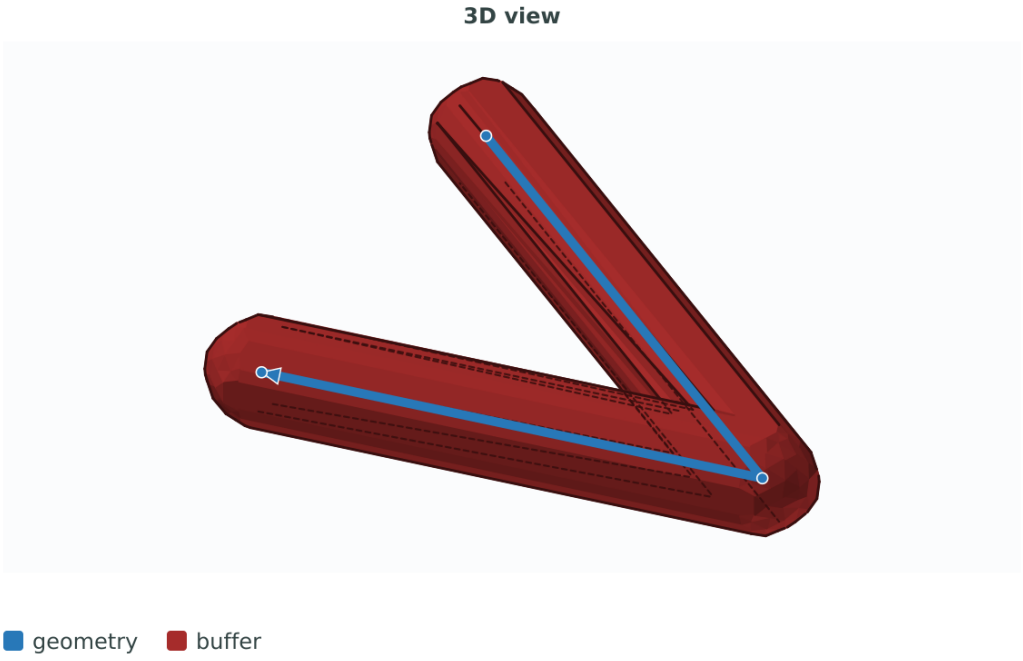


A rounded buffer around a line using 32 segments.

Code

```
SELECT
  'LINESTRING(50 50,150 150,150 50)'::geometry AS input_geometry,
  CG_3DBuffer('LINESTRING(50 50,150 150,150 50)'::geometry, 10, 32, 0) AS buffer;
```

Figure

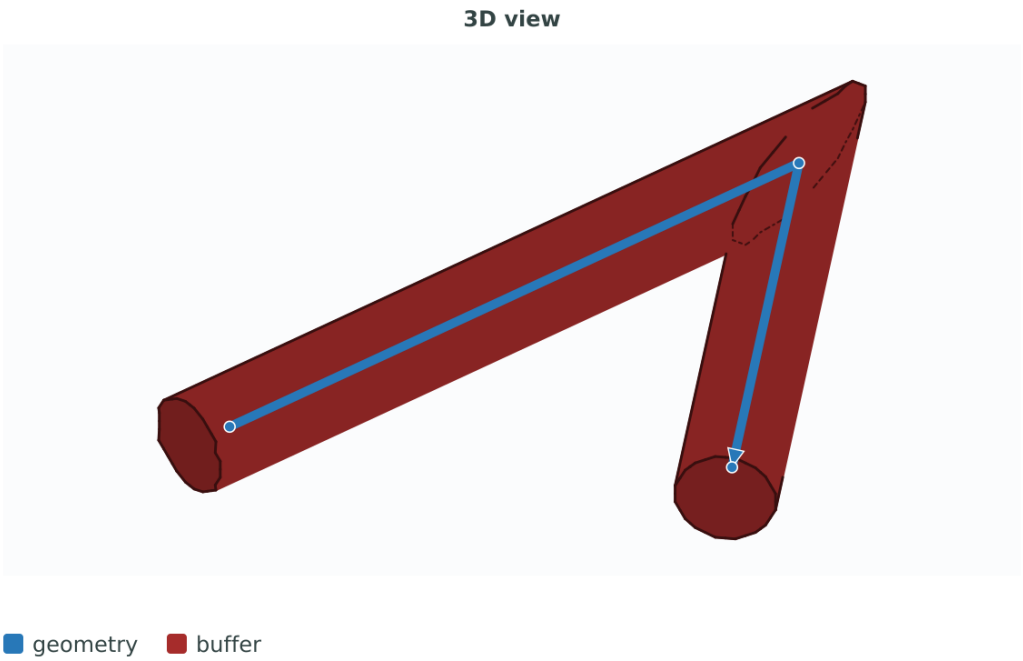


A square buffer around the same line using 32 segments.

Code

```
SELECT
  'LINESTRING(50 50,150 150,150 50)'::geometry AS input_geometry,
  CG_3DBuffer('LINESTRING(50 50,150 150,150 50)'::geometry, 10, 32, 2) AS buffer;
```

Figure



Siehe auch

ST_Buffer, CG_3DConvexHull, ST_AsX3D

8.3.35 CG_Rotate

CG_Rotate — Rotates a geometry by a given angle around the origin (0,0).

Synopsis

geometry **CG_Rotate**(geometry geom, float8 angle);

Beschreibung

Rotates the input geometry *geom* by *angle* radians around the origin point (0,0). The rotation is performed in 2D space; Z coordinates are not modified. Positive angles rotate the geometry counter-clockwise.

Availability: 3.6.0 - requires SFCGAL >= 2.0.0

Diese Methode benötigt ein SFCGAL-Backend.

Beispiele

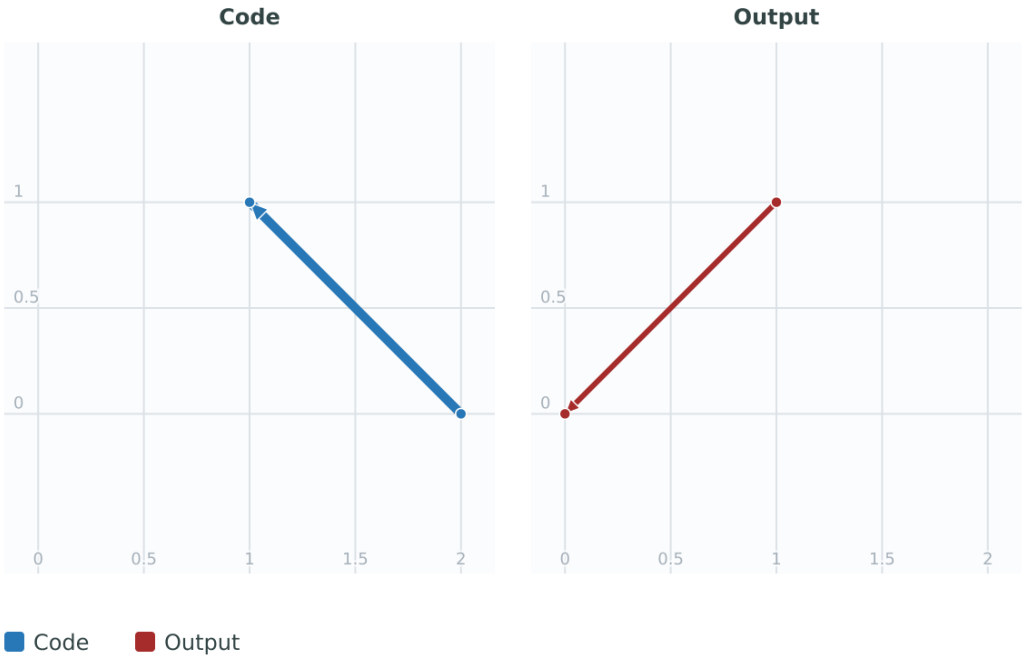
Code

```
SELECT CG_Rotate('LINESTRING(1 0,0 1)', pi()/2);
```

Output

```
LINESTRING(6.123233995736766e-17 1,-1 6.123233995736766e-17)
```

Figure



Siehe auch

CG_2DRotate, ST_Rotate

8.3.36 CG_2DRotate

CG_2DRotate — Rotates a geometry by a given angle around a specified point in 2D.

Synopsis

geometry **CG_2DRotate**(geometry geom, float8 angle, float8 cx, float8 cy);

Beschreibung

Rotates the input geometry *geom* by *angle* radians around the point (*cx*, *cy*). The rotation is performed in 2D space; Z coordinates are dropped. Positive angles rotate the geometry counter-clockwise.

Availability: 3.6.0 - requires SFCGAL >= 2.0.0

Diese Methode benötigt ein SFCGAL-Backend.

Beispiele

Code

```
SELECT CG_2DRotate('POINT(1 0)', pi()/2, 1, 1);
```

Output

```
POINT(2 1)
```

Figure



Siehe auch

CG_Rotate, CG_3DRotate

8.3.37 CG_3DRotate

CG_3DRotate — Rotates a geometry in 3D space around an axis vector.

Synopsis

geometry **CG_3DRotate**(geometry geom, float8 angle, float8 ax, float8 ay, float8 az);

Beschreibung

Rotates the input geometry *geom* by *angle* radians around an axis defined by the vector (*ax*, *ay*, *az*) passing through the origin (0,0,0).

Availability: 3.6.0 - requires SFCGAL >= 2.0.0

Diese Methode benötigt ein SFCGAL-Backend.

Diese Funktion unterstützt 3d und lässt den Z-Index nicht fallen.

Beispiele

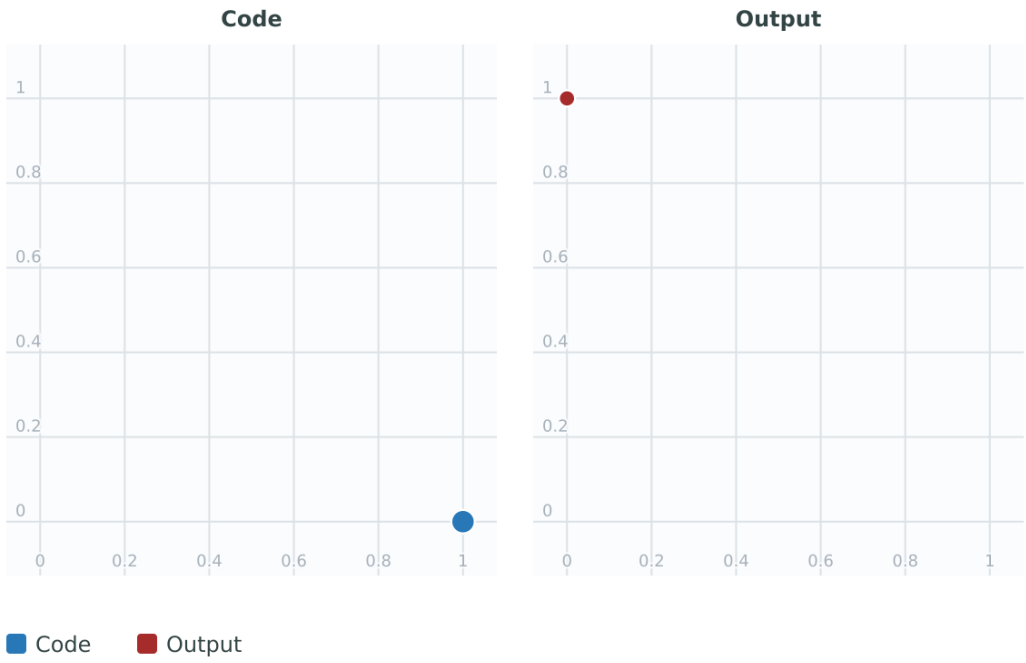
Code

```
SELECT CG_3DRotate('POINT(1 0 0)', pi()/2, 0, 0, 1);
```

Output

```
POINT Z (6.123233995736766e-17 1 0)
```

Figure



Siehe auch

CG_RotateX, CG_RotateY, CG_RotateZ

8.3.38 CG_RotateX

CG_RotateX — Rotates a geometry around the X-axis by a given angle.

Synopsis

geometry **CG_RotateX**(geometry geom, float8 angle);

Beschreibung

Rotates the input geometry *geom* by *angle* radians around the X-axis.

Availability: 3.6.0 - requires SFCGAL >= 2.0.0

Diese Methode benötigt ein SFCGAL-Backend.

Diese Funktion unterstützt 3d und lässt den Z-Index nicht fallen.

Beispiele

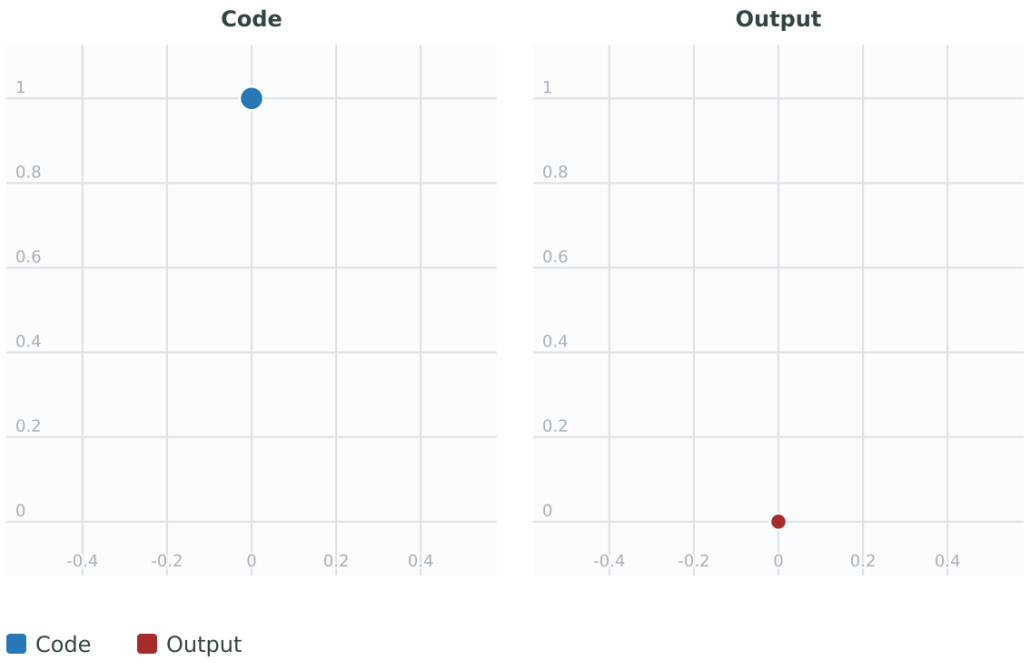
Code

```
SELECT CG_RotateX('POINT(0 1 0)', pi()/2);
```

Output

```
POINT Z (0 6.123233995736766e-17 1)
```

Figure



Siehe auch

CG_RotateY, CG_RotateZ, CG_3DRotate

8.3.39 CG_RotateY

CG_RotateY — Rotates a geometry around the Y-axis by a given angle.

Synopsis

geometry **CG_RotateY**(geometry geom, float8 angle);

Beschreibung

Rotates the input geometry *geom* by *angle* radians around the Y-axis passing.

Availability: 3.6.0 - requires SFCGAL >= 2.0.0

Diese Methode benötigt ein SFCGAL-Backend.

Diese Funktion unterstützt 3d und lässt den Z-Index nicht fallen.

Beispiele

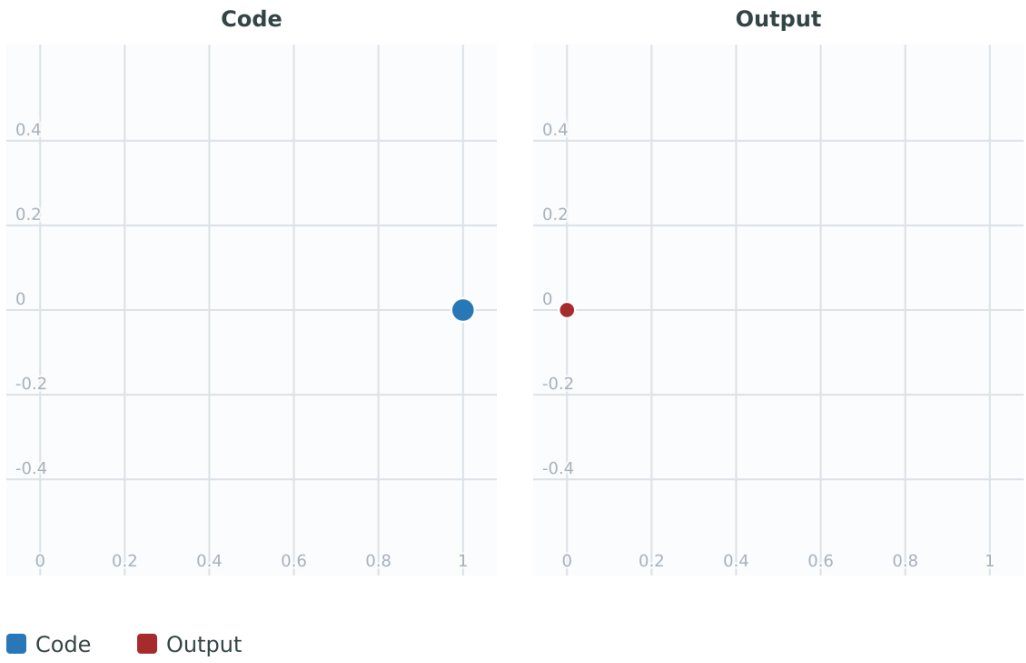
Code

```
SELECT CG_RotateY('POINT(1 0 0)', pi()/2);
```

Output

```
POINT Z (6.123233995736766e-17 0 -1)
```

Figure



Siehe auch

CG_RotateX, CG_RotateZ, CG_3DRotate

8.3.40 CG_RotateZ

CG_RotateZ — Rotates a geometry around the Z-axis by a given angle.

Synopsis

geometry **CG_RotateZ**(geometry geom, float8 angle);

Beschreibung

Rotates the input geometry *geom* by *angle* radians around the Z-axis.

Availability: 3.6.0 - requires SFCGAL >= 2.0.0

Diese Methode benötigt ein SFCGAL-Backend.

Diese Funktion unterstützt 3d und lässt den Z-Index nicht fallen.

Beispiele

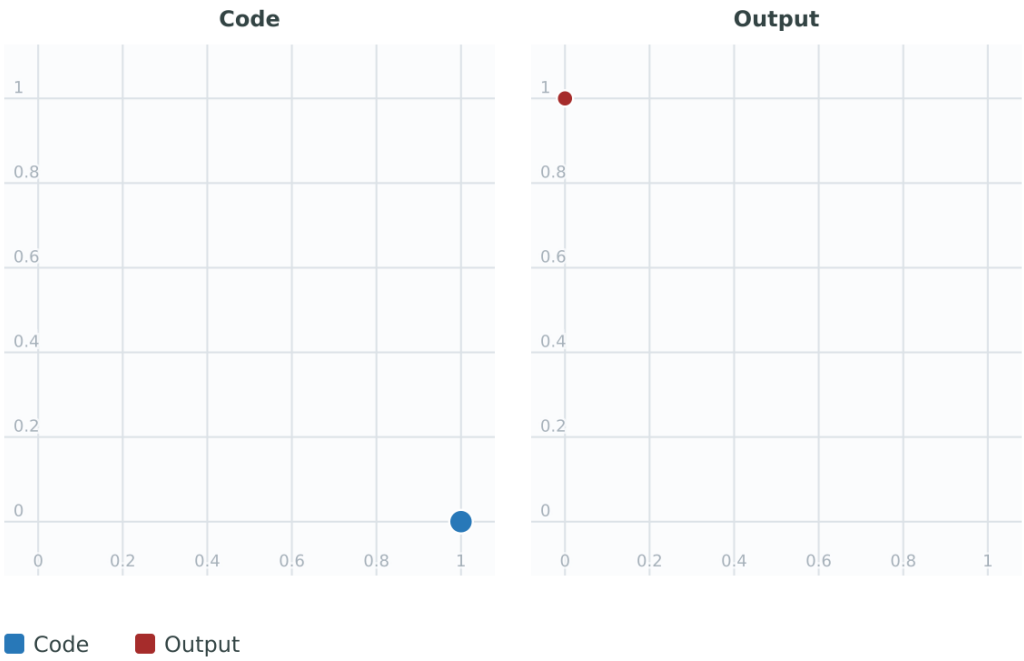
Code

```
SELECT CG_RotateZ('POINT(1 0 0)', pi()/2);
```

Output

```
POINT Z (6.123233995736766e-17 1 0)
```

Figure



Siehe auch

CG_RotateX, CG_RotateY, CG_3DRotate

8.3.41 CG_Scale

CG_Scale — Scales a geometry uniformly in all dimensions by a given factor.

Synopsis

geometry **CG_Scale**(geometry geom, float8 factor);

Beschreibung

Scales the input geometry *geom* by a uniform scale *factor* in all dimensions (X, Y, and Z). The scaling is performed relative to the origin point (0,0,0).

Availability: 3.6.0 - requires SFCGAL >= 2.0.0

Diese Methode benötigt ein SFCGAL-Backend.

Beispiele

Code

```
SELECT CG_Scale('LINESTRING(1 1,2 2)', 2);
```

Output

```
LINESTRING(2 2,4 4)
```

Figure



Siehe auch

CG_3DScale, CG_3DScaleAroundCenter, ST_Scale

8.3.42 CG_3DScale

CG_3DScale — Scales a geometry by separate factors along X, Y, and Z axes.

Synopsis

geometry **CG_3DScale**(geometry geom, float8 factorX, float8 factorY, float8 factorZ);

Beschreibung

Scales the input geometry *geom* by different factors along the X, Y, and Z axes. The scaling is performed relative to the origin point (0,0,0).

Availability: 3.6.0 - requires SFCGAL >= 2.0.0

Diese Methode benötigt ein SFCGAL-Backend.

Diese Funktion unterstützt 3d und lässt den Z-Index nicht fallen.

Beispiele

Code

```
SELECT CG_3DScale('POINT(1 1 1)', 2, 3, 4);
```

Output

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

Figure



Siehe auch

CG_Scale, CG_3DScaleAroundCenter

8.3.43 CG_3DScaleAroundCenter

CG_3DScaleAroundCenter — Scales a geometry in 3D space around a specified center point.

Synopsis

geometry **CG_3DScaleAroundCenter**(geometry geom, float8 factorX, float8 factorY, float8 factorZ, float8 centerX, float8 centerY, float8 centerZ);

Beschreibung

Scales the input geometry *geom* by different factors along the X, Y, and Z axes, relative to a specified center point (*centerX*, *centerY*, *centerZ*).

Availability: 3.6.0 - requires SFCGAL >= 2.0.0

Diese Methode benötigt ein SFCGAL-Backend.

Diese Funktion unterstützt 3d und lässt den Z-Index nicht fallen.

Beispiele

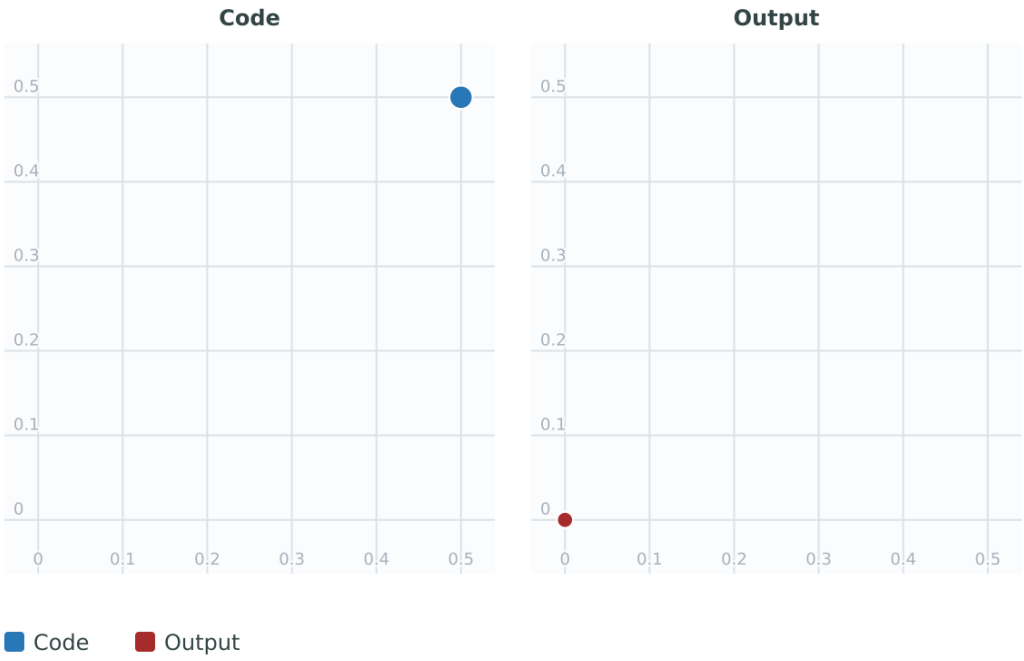
Code

```
SELECT CG_3DScaleAroundCenter('POINT(2 2 2)', 0.5, 0.5, 0.5, 1, 1, 1);
```

Output

```
POINT Z (1.5 1.5 1.5)
```

Figure



Siehe auch

CG_Scale, CG_3DScale

8.3.44 CG_Translate

CG_Translate — Translates (moves) a geometry by given offsets in 2D space.

Synopsis

geometry **CG_Translate**(geometry geom, float8 deltaX, float8 deltaY);

Beschreibung

Translates the input geometry *geom* by adding *deltaX* to the X coordinates and *deltaY* to the Y coordinates. Z coordinates are dropped.

Availability: 3.6.0 - requires SFCGAL >= 2.0.0

Diese Methode benötigt ein SFCGAL-Backend.

Beispiele

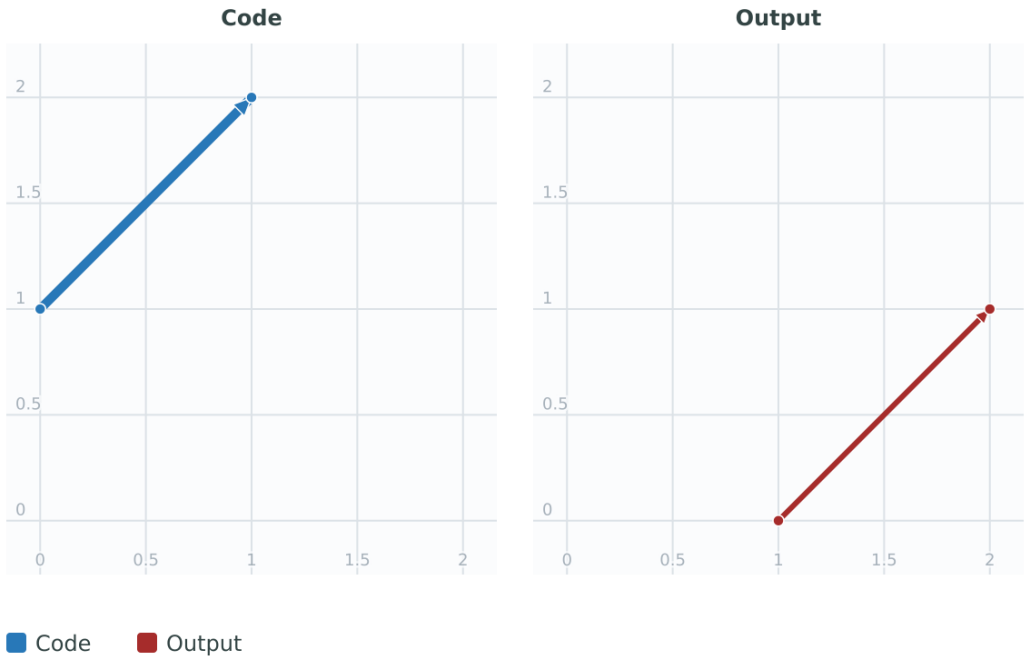
Code

```
SELECT CG_Translate('LINESTRING(1 1,2 2)', 1, -1);
```

Output

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

Figure



Siehe auch

CG_3DTranslate, ST_Translate

8.3.45 CG_3DTranslate

CG_3DTranslate — Translates (moves) a geometry by given offsets in 3D space.

Synopsis

geometry **CG_3DTranslate**(geometry geom, float8 deltaX, float8 deltaY, float8 deltaZ);

Beschreibung

Translates the input geometry *geom* by adding *deltaX* to the X coordinates, *deltaY* to the Y coordinates, and *deltaZ* to the Z coordinates.

Availability: 3.6.0 - requires SFCGAL >= 2.0.0

Diese Methode benötigt ein SFCGAL-Backend.

Diese Funktion unterstützt 3d und lässt den Z-Index nicht fallen.

Beispiele

Code

```
SELECT CG_3DTranslate('POINT(1 1 1)', 1, -1, 2);
```

Output

```
POINT Z (2 0 3)
```

Figure



Siehe auch

CG_Translate, ST_Translate

8.3.46 CG_Simplify

CG_Simplify — Reduces the complexity of a geometry while preserving essential features and Z/M values.

Synopsis

geometry **CG_Simplify**(geometry geom, double precision threshold, boolean preserveTopology = false);

Beschreibung

Simplifies a geometry using SFCGAL's simplification algorithm, which reduces the number of points or vertices while preserving the essential features of the geometry. This function preserves Z and M values during simplification.

The algorithm is based on constrained triangulation and uses the CGAL Polyline Simplification 2 library with additional handling to preserve Z and M coordinates. When topology is preserved and geometries intersect, Z and M values are interpolated at intersection points.

This function works well with 3D terrain-like geometries (2.5D) but is not designed for vertical surfaces like walls.

Availability: 3.6.0 - requires SFCGAL >= 2.1.0

Diese Methode benötigt ein SFCGAL-Backend.

Diese Funktion unterstützt 3d und lässt den Z-Index nicht fallen.

Diese Funktion unterstützt M-Koordinaten.

Parameters

geom Input geometry

threshold Maximum distance threshold (in geometry unit) for simplification. The higher this value, the more simplified the resulting geometry will be.

preserveTopology If set to true, the function ensures that the topology of the geometry is preserved. When geometries intersect in this mode, Z and M values at intersection points are interpolated. The default value is false.

Return Value

Returns a simplified geometry with preserved Z and M values.

Beispiele

This example simplifies a polygon with a threshold of 0.5.

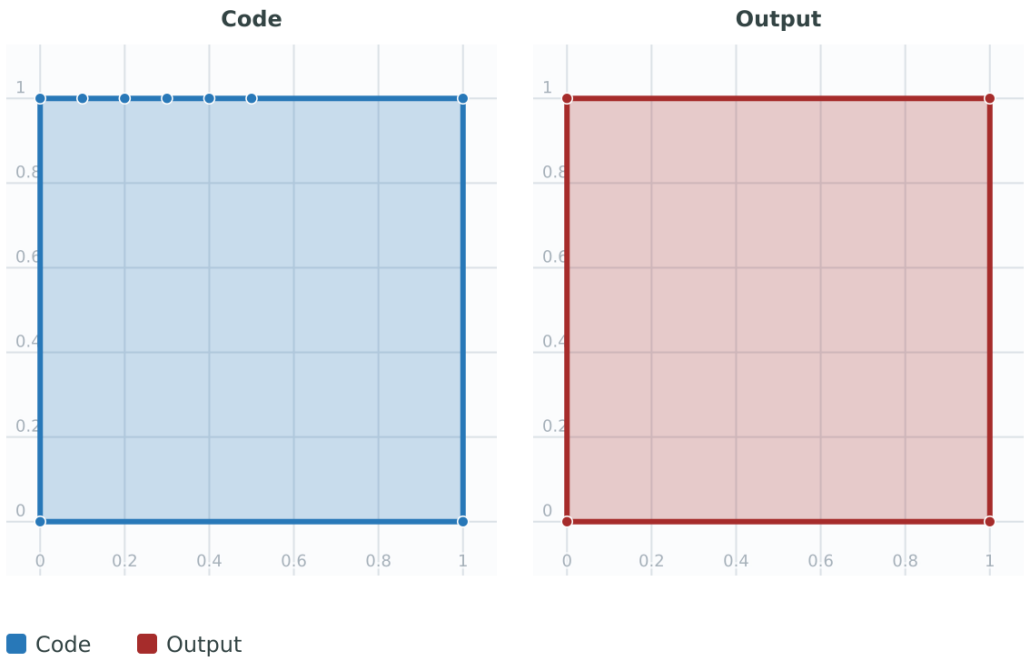
Code

```
SELECT CG_Simplify(
    'POLYGON((0 0,0 1,0.1 1,0.2 1,0.3 1,0.4 1,0.5 1,1 1,1 0,0 0))',
    0.5
  );
```

Output

```
POLYGON((0 0,0 1,1 1,1 0,0 0))
```

Figure



Preserving topology can keep additional support points in a mixed ZM collection.

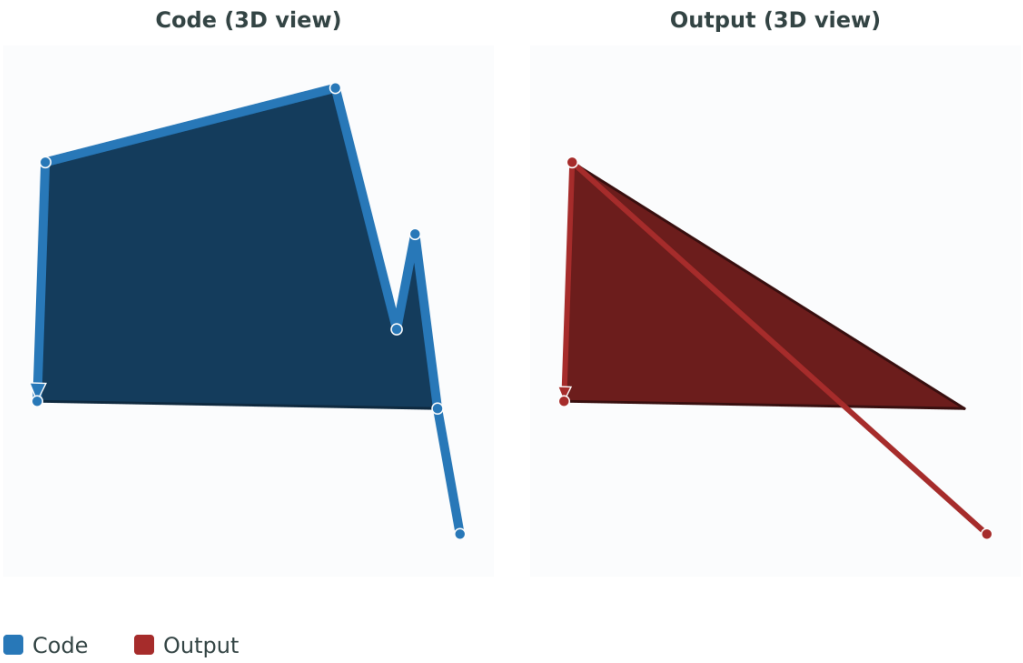
Code

```
SELECT CG_Simplify(
    'GEOMETRYCOLLECTION ZM(
      LINESTRING ZM(-1 -1 3 4,0 0 10 100,1 1 20 200,0 2 15 150,0 5 30 300,2 19 25 250,-4 20 15 150),
      POLYGON ZM((0 0 10 100,1 1 20 200,0 2 15 150,0 5 30 300,2 19 25 250,-4 20 15 150,0 0 10 100))
    )',
    2,
    false
  );
```

Output

```
GEOMETRYCOLLECTION ZM (LINESTRING ZM (-1 -1 3 4,2 19 25 250,-4 20 15 150),POLYGON ZM ((0 0 10 100,2 19 25 250,-4 20 15 150,0 0 10 100)))
```

Figure



The topology-preserving variant keeps the shared anchor point at the collection intersection.

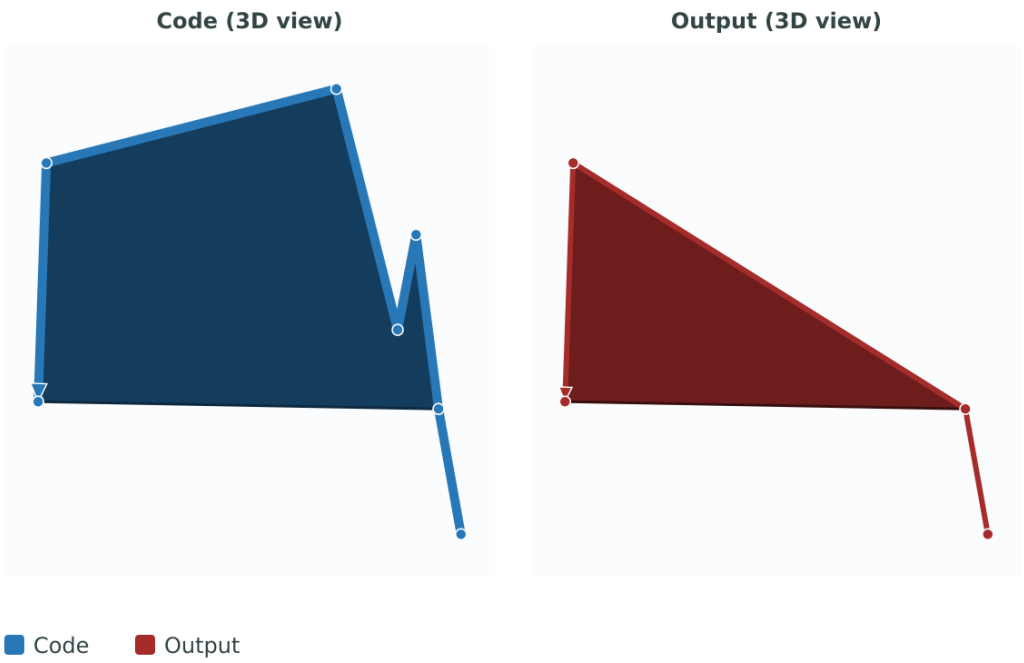
Code

```
SELECT CG_Simplify(
    'GEOMETRYCOLLECTION ZM(
      LINESTRING ZM(-1 -1 3 4,0 0 10 100,1 1 20 200,0 2 15 150,0 5 30 300,2 19 25 250,-4 20 15 150),
      POLYGON ZM((0 0 10 100,1 1 20 200,0 2 15 150,0 5 30 300,2 19 25 250,-4 20 15 150,0 0 10 100))
    )',
    2,
    true
  );
```

Output

```
GEOMETRYCOLLECTION ZM (LINESTRING ZM (-1 -1 3 4,0 0 10 100,2 19 25 250,-4 20 15 150),POLYGON ZM ((0 0 10 100,2 19 25 ↩
    250,-4 20 15 150,0 0 10 100)))
```

Figure

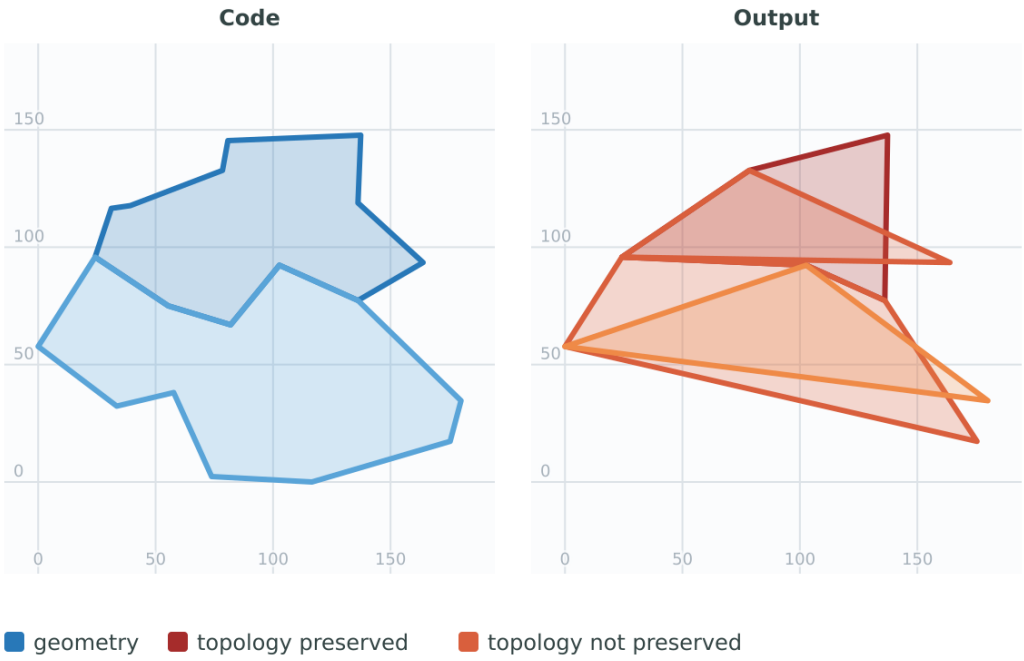


Compare simplification with and without topology preservation, using a threshold of 50.

Code

```
WITH data AS (
  SELECT ST_GeomFromText('GEOMETRYCOLLECTION(
    POLYGON((88.46 158.85,90.77 171.54,147.31 173.85,146.15 145,173.85 119.62,146.15 ↩
    103.46,112.69 118.46,91.92 93.08,65.38 101.15,34.23 121.92,41.15 142.69,49.23 ↩
    143.85,88.46 158.85)),
    POLYGON((112.69 118.46,146.15 103.46,190 60.77,185.38 43.46,126.54 26.15,83.85 ↩
    28.46,67.69 64.23,43.46 58.46,10 83.85,34.23 121.92,65.38 101.15,91.92 93.08,112.69 ↩
    118.46)))
  ') AS geom
)
SELECT geom AS input_geometry,
       CG_Simplify(geom, 50, true) AS topology_preserved,
       CG_Simplify(geom, 50, false) AS topology_not_preserved
FROM data;
```

Figure



Siehe auch

ST_Simplify, ST_SimplifyPreserveTopology

8.3.47 CG_3DAlphaWrapping

CG_3DAlphaWrapping — Computes a 3D Alpha-wrapping strictly enclosing a geometry.

Synopsis

geometry **CG_3DAlphaWrapping**(geometry geom, integer relative_alpha, integer relative_offset);

Beschreibung

Computes the 3D Alpha Wrapping of the points in a geometry. An alpha wrapping is a watertight and orientable surface mesh that strictly encloses the input. It can be seen as an extension or refinement of an alpha-shape.

The `relative_alpha` parameter controls which features will appear in the output. It can have values from 0 to infinity. Higher relative_alpha values produce coarser wraps; lower values retain more detail.

The `relative_offset` parameter controls the tightness of the result. It can have values from 0 to infinity. If this parameter is set to 0, its value is automatically determined based on the relative_alpha parameter.

Availability: 3.6.0 - requires SFCGAL >= 2.1.0

Diese Methode benötigt ein SFCGAL-Backend.

Diese Funktion unterstützt 3d und lässt den Z-Index nicht fallen.

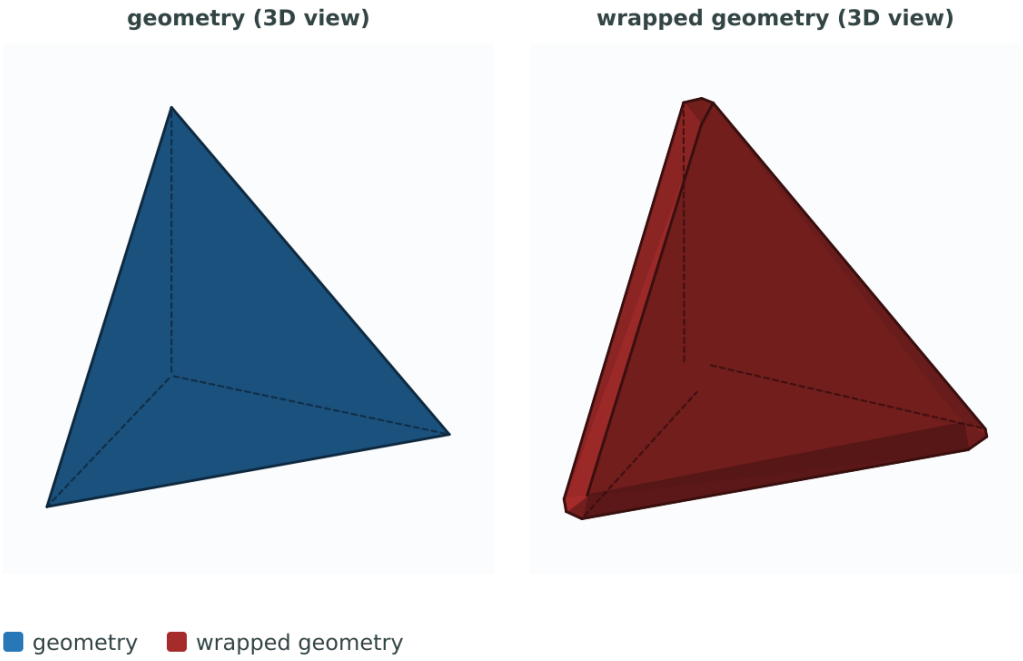
Beispiele

Wrap a tetrahedral surface and show the native 3D input beside the actual enclosing mesh.

Code

```
WITH data AS (
  SELECT 'POLYHEDRALSURFACE Z (
    ((0 0 0,4 0 0,0 4 0,0 0 0)),
    ((0 0 0,0 0 4,4 0 0,0 0 0)),
    ((0 0 0,0 4 0,0 0 4,0 0 0)),
    ((4 0 0,0 0 4,0 4 0,4 0 0))
  )'::geometry AS input_geometry
)
SELECT input_geometry AS input_geometry,
       CG_3DAlphaWrapping(input_geometry, 1)
         AS wrapped_geometry
FROM data;
```

Figure



Siehe auch

CG_AlphaShape

8.3.48 CG_NurbsCurveInterpolate

CG_NurbsCurveInterpolate — Creates an interpolating NURBS curve passing through all given data points

Synopsis

geometry **CG_NurbsCurveInterpolate**(geometry data_points, integer degree);

Beschreibung

Creates a NURBS curve that interpolates through all provided data points. The resulting curve passes exactly through each point in the input geometry.

Parameters:

- `data_points` - A LINESTRING geometry containing the points to interpolate through.
- `degree` - Polynomial degree of the NURBS curve (typically 2 or 3).

Availability: 3.7.0 - requires SFCGAL >= 2.3.0.

Diese Methode benötigt ein SFCGAL-Backend.

Diese Funktion unterstützt 3d und lässt den Z-Index nicht fallen.

Beispiele

Interpolate a degree-3 NURBS curve through four 2D points.

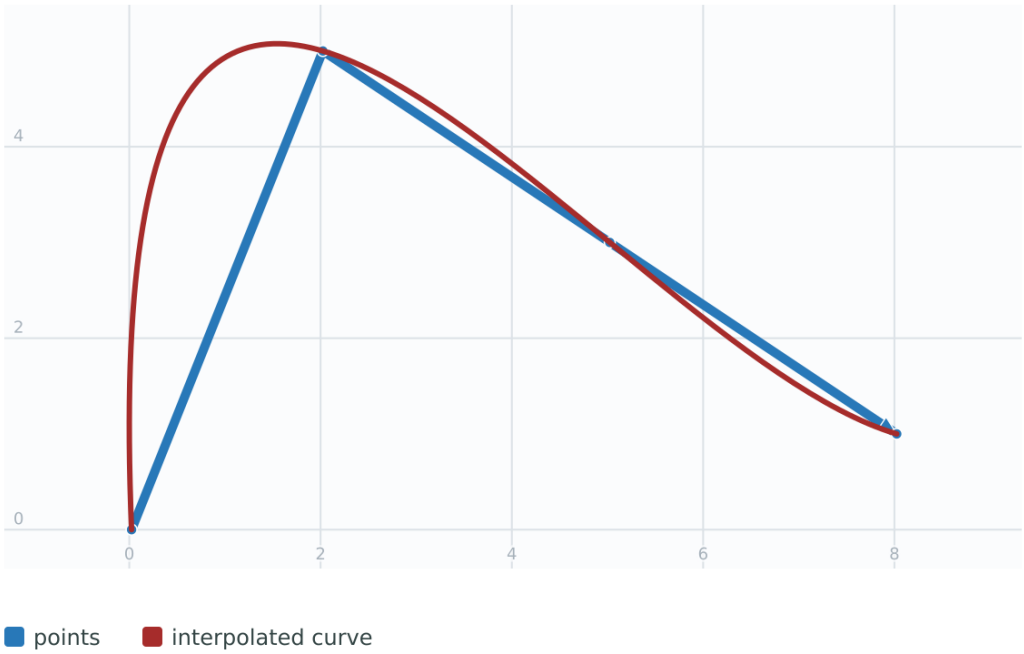
Code

```
WITH data AS (
  SELECT 'LINESTRING(0 0,2 5,5 3,8 1)'::geometry AS input_points
)
SELECT input_points AS input_points,
       CG_NurbsCurveInterpolate(input_points, 3) AS interpolated_curve
FROM data;
```

Output

```
LINESTRING(0 0,2 5,5 3,8 1) | NURBSCURVE(DEGREE 3, CONTROLPOINTS(NURBSPOINT(WEIGHTEDPOINT(0 0),WEIGHT 1), NURBSPOINT( ↩
    WEIGHTEDPOINT(-0.43 10.29),WEIGHT 1), NURBSPOINT(WEIGHTEDPOINT(4.95 1.9),WEIGHT 1), NURBSPOINT(WEIGHTEDPOINT(8 1), ↩
    WEIGHT 1)),KNOTS (KNOT(0,4), KNOT(1,4)))
```

Figure



Interpolate a 3D degree-3 NURBS curve through the documented control points.

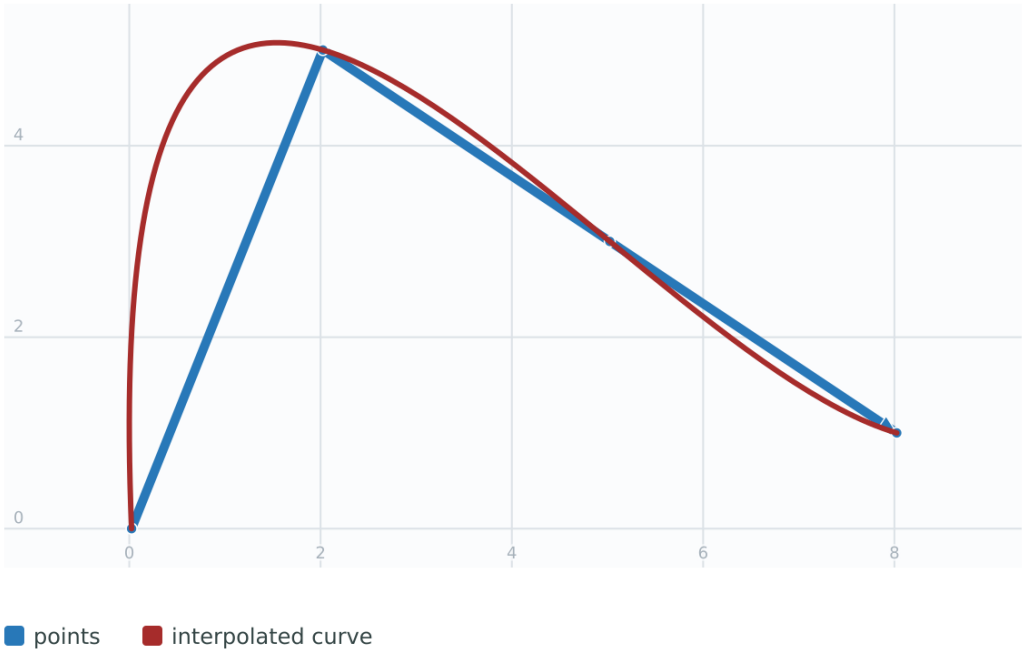
Code

```
WITH data AS (
  SELECT 'LINESTRING Z (0 0 0,2 5 2,5 3 1,8 1 0)'::geometry AS input_points
)
SELECT input_points AS input_points,
       CG_NurbsCurveInterpolate(input_points, 3) AS interpolated_curve
FROM data;
```

Output

```
LINESTRING Z (0 0 0,2 5 2,5 3 1,8 1 0) | NURBSCURVE Z (DEGREE 3,CONTROLPOINTS Z (NURBSPOINT(WEIGHTEDPOINT Z (0 0 0), ↩
    WEIGHT 1), NURBSPOINT(WEIGHTEDPOINT Z (-0.43 10.29 4.36),WEIGHT 1), NURBSPOINT(WEIGHTEDPOINT Z (4.95 1.9 0.54), ↩
    WEIGHT 1), NURBSPOINT(WEIGHTEDPOINT Z (8 1 0),WEIGHT 1)),KNOTS (KNOT(0,4), KNOT(1,4)))
```

Figure



Siehe auch

CG_NurbsCurveApproximate, ST_MakeNurbsCurve

8.3.49 CG_NurbsCurveApproximate

CG_NurbsCurveApproximate — Creates an approximating NURBS curve fitting data points within a tolerance

Synopsis

geometry **CG_NurbsCurveApproximate**(geometry data_points, integer degree, float8 tolerance, integer max_control_points=100);

Beschreibung

Creates a NURBS curve that approximates the given data points within the specified tolerance. Unlike interpolation, the curve does not necessarily pass through all points but provides a smooth fit with fewer control points.

Parameters:

- `data_points` - A LINESTRING geometry containing the points to approximate.
- `degree` - Polynomial degree of the NURBS curve.
- `tolerance` - Maximum allowed distance between the curve and data points.
- `max_control_points` - Optional maximum number of control points (default: 100). Limits curve complexity.

Availability: 3.7.0 - requires SFCGAL >= 2.3.0.

Diese Methode benötigt ein SFCGAL-Backend.

Diese Funktion unterstützt 3d und lässt den Z-Index nicht fallen.

Beispiele

Approximate noisy 2D data with a degree-2 curve and a tolerance of 0.5.

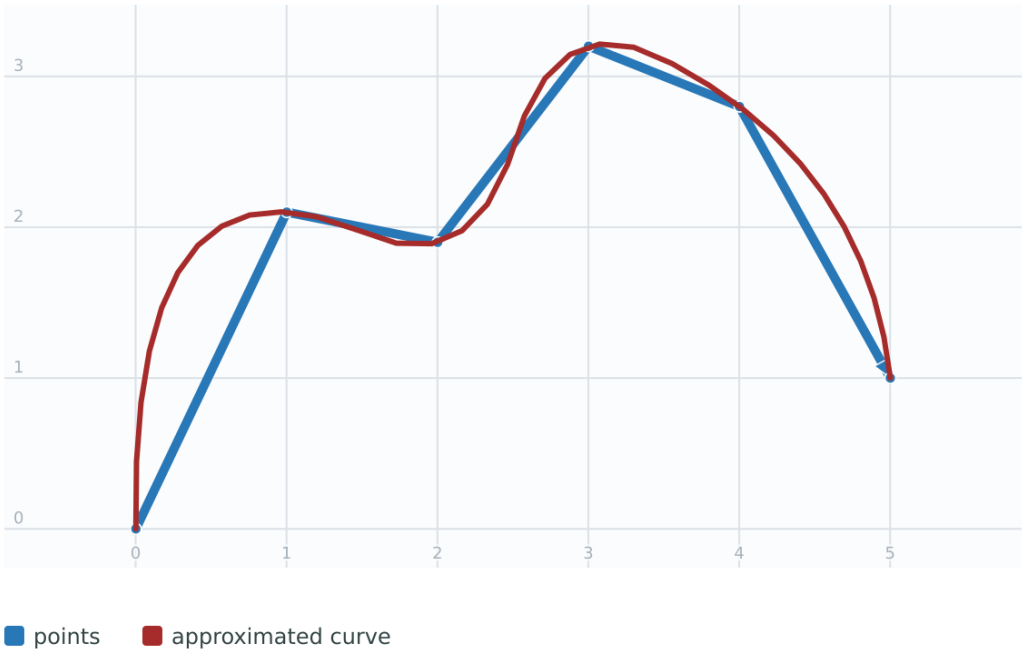
Code

```
WITH data AS (
  SELECT 'LINESTRING(0 0,1 2.1,2 1.9,3 3.2,4 2.8,5 1)'::geometry AS input_points
)
SELECT input_points AS input_points,
       CG_NurbsCurveApproximate(input_points, 2, 0.5) AS approximated_curve
FROM data;
```

Output

```
LINESTRING(0 0,1 2.1,2 1.9,3 3.2,4 2.8,5 1) | NURBSCURVE(DEGREE 2, CONTROLPOINTS(NURBSPOINT(WEIGHTEDPOINT(0 0),WEIGHT ↩
    1), NURBSPOINT(WEIGHTEDPOINT(-0.05 2.64),WEIGHT 1), NURBSPOINT(WEIGHTEDPOINT(2.24 1.65),WEIGHT 1), NURBSPOINT( ↩
    WEIGHTEDPOINT(2.79 3.48),WEIGHT 1), NURBSPOINT(WEIGHTEDPOINT(4.84 2.43),WEIGHT 1), NURBSPOINT(WEIGHTEDPOINT(5 1), ↩
    WEIGHT 1)),KNOTS (KNOT(0,3), KNOT(0.35,1), KNOT(0.51,1), KNOT(0.68,1), KNOT(1,3)))
```

Figure



Approximate a 3D point set while preserving the Z dimension in the fitted curve.

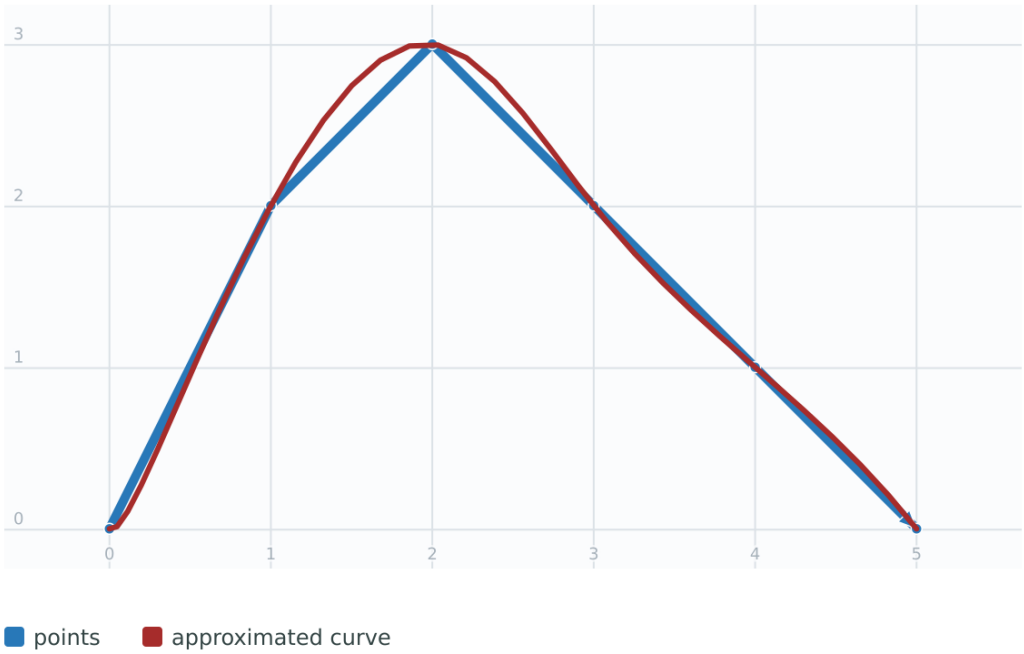
Code

```
WITH data AS (
  SELECT 'LINESTRING Z (0 0 0,1 2 1,2 3 2,3 2 1,4 1 0,5 0 0)'::geometry AS input_points
)
SELECT input_points AS input_points,
       CG_NurbsCurveApproximate(input_points, 3, 0.1) AS approximated_curve
FROM data;
```

Output

```
LINESTRING Z (0 0 0,1 2 1,2 3 2,3 2 1,4 1 0,5 0 0) | NURBSCURVE Z (DEGREE 3,CONTROLPOINTS Z (NURBSPOINT(WEIGHTEDPOINT Z ↩
      (0 0 0),WEIGHT 1), NURBSPOINT(WEIGHTEDPOINT Z (0.18 -0.15 -1.29),WEIGHT 1), NURBSPOINT(WEIGHTEDPOINT Z (1.49 4.36 ↩
      3.35),WEIGHT 1), NURBSPOINT(WEIGHTEDPOINT Z (3.33 1.33 0.38),WEIGHT 1), NURBSPOINT(WEIGHTEDPOINT Z (4.34 0.83  ↩
     -0.44),WEIGHT 1), NURBSPOINT(WEIGHTEDPOINT Z (5 0 0),WEIGHT 1)),KNOTS (KNOT(0,4), KNOT(0.46,1), KNOT(0.64,1), KNOT ↩
     (1,4)))
```

Figure



Siehe auch

CG_NurbsCurveInterpolate, ST_MakeNurbsCurve

8.3.50 CG_NurbsCurveDerivative

CG_NurbsCurveDerivative — Computes the derivative (tangent, curvature) of a NURBS curve at a given parameter

Synopsis

geometry **CG_NurbsCurveDerivative**(geometry nurbscurve, float8 parameter, integer derivative_order);

Beschreibung

Computes the derivative of a NURBS curve at a specific parameter value. The derivative represents the rate of change at that point:

- Order 1: Tangent vector (direction and speed)
- Order 2: Curvature vector (how the tangent changes)
- Order 3: Third-order derivative (supported maximum)

Parameters:

- `nurbscurve` - A NURBS curve geometry.
- `parameter` - Parameter value for evaluation (typically 0.0 to 1.0).
- `derivative_order` - Order of derivative to compute (1, 2, or 3).

Returns a POINT geometry representing the derivative vector at the specified parameter.

Availability: 3.7.0 - requires SFCGAL >= 2.3.0.

 Diese Methode benötigt ein SFCGAL-Backend.

 Diese Funktion unterstützt 3d und lässt den Z-Index nicht fallen.

Beispiele

The first derivative gives the tangent vector at the evaluation parameter.

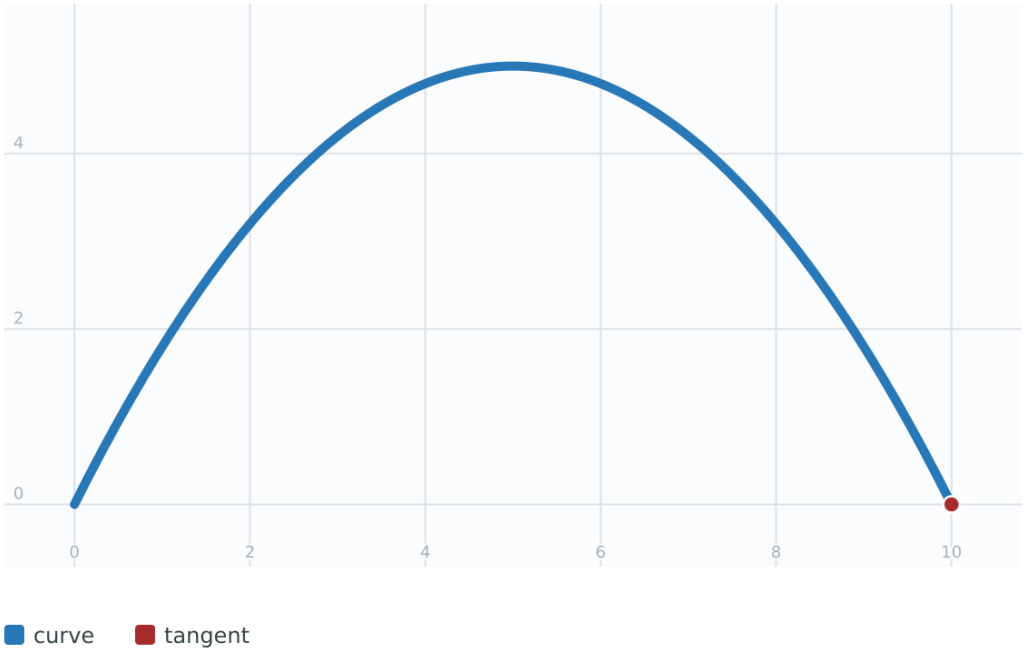
Code

```
WITH data AS (
  SELECT 'NURBSCURVE (2, (0.00 0.00, 5.00 10.00, 10.00 0.00))'::geometry AS input_curve
)
SELECT input_curve AS input_curve,
       CG_NurbsCurveDerivative(input_curve, 0.5, 1) AS tangent
FROM data;
```

Output

```
NURBSCURVE(DEGREE 2, CONTROLPOINTS(NURBSPOINT(WEIGHTEDPOINT(0 0),WEIGHT 1), NURBSPOINT(WEIGHTEDPOINT(5 10),WEIGHT 1), ↩
    NURBSPOINT(WEIGHTEDPOINT(10 0),WEIGHT 1)),KNOTS (KNOT(0,3), KNOT(1,3))) | POINT Z (10 0 0)
```

Figure



The second derivative exposes curvature change for a higher-degree curve.

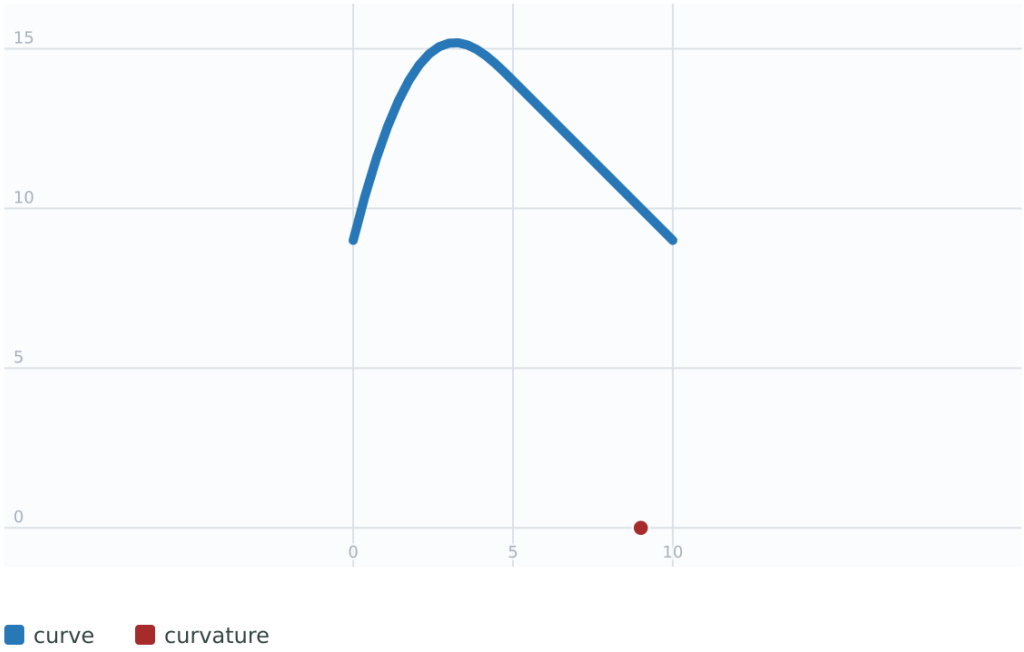
Code

```
WITH data AS (
  SELECT 'NURBSCURVE (3, (0.00 0.00, 2.00 8.00, 5.00 5.00, 8.00 2.00, 10.00 0.00))':: ↩
    geometry AS input_curve
)
SELECT input_curve AS input_curve,
       CG_NurbsCurveDerivative(input_curve, 0.5, 2) AS curvature
FROM data;
```

Output

```
NURBSCURVE(DEGREE 3, CONTROLPOINTS(NURBSPOINT(WEIGHTEDPOINT(0 0),WEIGHT 1), NURBSPOINT(WEIGHTEDPOINT(2 8),WEIGHT 1),  ↩
    NURBSPOINT(WEIGHTEDPOINT(5 5),WEIGHT 1), NURBSPOINT(WEIGHTEDPOINT(8 2),WEIGHT 1), NURBSPOINT(WEIGHTEDPOINT(10 0), ↩
    WEIGHT 1)),KNOTS (KNOT(0,4), KNOT(0.5,1), KNOT(1,4))) | POINT Z (9 -9 0)
```

Figure



The same derivative call also works for 3D NURBS curves.

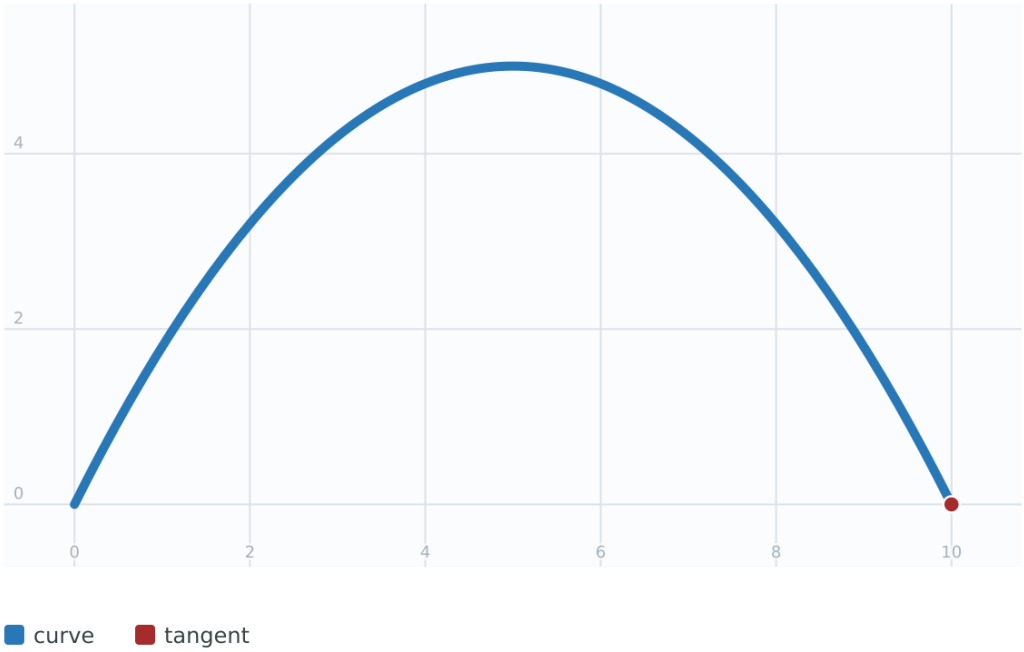
Code

```
WITH data AS (
  SELECT 'NURBSCURVE Z (2, (0.00 0.00 0.00, 5.00 10.00 5.00, 10.00 0.00 0.00))'::geometry AS ↩
     input_curve
)
SELECT input_curve AS input_curve,
       CG_NurbsCurveDerivative(input_curve, 0.5, 1) AS tangent
FROM data;
```

Output

```
NURBSCURVE Z (DEGREE 2,CONTROLPOINTS Z (NURBSPOINT(WEIGHTEDPOINT Z (0 0 0),WEIGHT 1), NURBSPOINT(WEIGHTEDPOINT Z (5 10 ↩
    5),WEIGHT 1), NURBSPOINT(WEIGHTEDPOINT Z (10 0 0),WEIGHT 1)),KNOTS (KNOT(0,3), KNOT(1,3))) | POINT Z (10 0 0)
```

Figure



Siehe auch

ST_Evaluate, ST_MakeNurbsCurve

8.4 Deprecated SFCGAL Function Aliases

These `ST_` names remain available for compatibility and emit deprecation notices. Their entries contain only the legacy signature and migration link. Use the corresponding `CG_` pages for behavior, examples, and geometry support details.

8.4.1 ST_3DIntersection

ST_3DIntersection — Deprecated alias for `CG_3DIntersection`.

Synopsis

geometry **ST_3DIntersection**(geometry geom1, geometry geom2);

Description

Warnung
`ST_3DIntersection` is deprecated as of 3.5.0. Use CG_3DIntersection instead.

Availability: 2.1.0

Diese Methode benötigt ein SFCGAL-Backend.

8.4.2 ST_3DDifference

ST_3DDifference — Deprecated alias for `CG_3DDifference`.

Synopsis

geometry **ST_3DDifference**(geometry geom1, geometry geom2);

Description



Warnung
`ST_3DDifference` is deprecated as of 3.5.0. Use CG_3DDifference instead.

Availability: 2.2.0


Diese Methode benötigt ein SFCGAL-Backend.

8.4.3 ST_3DUnion

ST_3DUnion — Deprecated alias for `CG_3DUnion`.

Synopsis

geometry **ST_3DUnion**(geometry geom1, geometry geom2);
geometry **ST_3DUnion**(geometry set g1field);

Description



Warnung
`ST_3DUnion` is deprecated as of 3.5.0. Use CG_3DUnion instead.

Availability: 2.2.0


Diese Methode benötigt ein SFCGAL-Backend.

8.4.4 ST_Tesselate

ST_Tesselate — Deprecated alias for `CG_Tesselate`.

Synopsis

geometry **ST_Tesselate**(geometry geom);

Description



Warnung
`ST_Tesselate` is deprecated as of 3.5.0. Use CG_Tesselate instead.

Availability: 2.1.0


Diese Methode benötigt ein SFCGAL-Backend.

8.4.5 ST_3DArea

ST_3DArea — Deprecated alias for `CG_3DArea`.

Synopsis

float**ST_3DArea**(geometry geom1);

Description



Warnung
`ST_3DArea` is deprecated as of 3.5.0. Use CG_3DArea instead.

Availability: 2.1.0


Diese Methode benötigt ein SFCGAL-Backend.

8.4.6 ST_Extrude

ST_Extrude — Deprecated alias for `CG_Extrude`.

Synopsis

geometry **ST_Extrude**(geometry geom, float x, float y, float z);

Description



Warnung
`ST_Extrude` is deprecated as of 3.5.0. Use CG_Extrude instead.

Availability: 2.1.0


Diese Methode benötigt ein SFCGAL-Backend.

8.4.7 ST_ForceLHR

ST_ForceLHR — Deprecated alias for `CG_ForceLHR`.

Synopsis

geometry **ST_ForceLHR**(geometry geom);

Description



Warnung
`ST_ForceLHR` is deprecated as of 3.5.0. Use CG_ForceLHR instead.

Availability: 2.1.0


Diese Methode benötigt ein SFCGAL-Backend.

8.4.8 ST_Orientation

ST_Orientation — Deprecated alias for `CG_Orientation`.

Synopsis

integer **ST_Orientation**(geometry geom);

Description



Warnung
`ST_Orientation` is deprecated as of 3.5.0. Use CG_Orientation instead.

Availability: 2.1.0


Diese Methode benötigt ein SFCGAL-Backend.

8.4.9 ST_MinkowskiSum

ST_MinkowskiSum — Deprecated alias for `CG_MinkowskiSum`.

Synopsis

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

Description



Warnung
`ST_MinkowskiSum` is deprecated as of 3.5.0. Use CG_MinkowskiSum instead.

Availability: 2.1.0


Diese Methode benötigt ein SFCGAL-Backend.

8.4.10 ST_StraightSkeleton

ST_StraightSkeleton — Deprecated alias for `CG_StraightSkeleton`.

Synopsis

geometry **ST_StraightSkeleton**(geometry geom);

Description



Warnung
`ST_StraightSkeleton` is deprecated as of 3.1.0. Use CG_StraightSkeleton instead.



Anmerkung
Like `CG_StraightSkeleton`, this function ignores the Z dimension and always gives a 2D result, even when used on a 3D geometry.

Availability: 2.1.0


Diese Methode benötigt ein SFCGAL-Backend.

8.4.11 ST_ApproximateMedialAxis

ST_ApproximateMedialAxis — Deprecated alias for `CG_ApproximateMedialAxis`.

Synopsis

geometry **ST_ApproximateMedialAxis**(geometry geom);

Description



Warnung
`ST_ApproximateMedialAxis` is deprecated as of 3.5.0. Use CG_ApproximateMedialAxis instead.



Anmerkung
Like `CG_ApproximateMedialAxis`, this function ignores the Z dimension and always gives a 2D result, even when used on a 3D geometry.

Availability: 2.2.0

 Diese Methode benötigt ein SFCGAL-Backend.

8.4.12 ST_IsPlanar

ST_IsPlanar — Deprecated alias for `CG_IsPlanar`.

Synopsis

boolean **ST_IsPlanar**(geometry geom);

Description



Warnung
`ST_IsPlanar` is deprecated as of 3.5.0. Use CG_IsPlanar instead.

Availability: 2.2.0: This was documented in 2.1.0 but got accidentally left out in 2.1 release.

 Diese Methode benötigt ein SFCGAL-Backend.

8.4.13 ST_Volume

ST_Volume — Deprecated alias for `CG_Volume`.

Synopsis

float **ST_Volume**(geometry geom1);

Description



Warnung
`ST_Volume` is deprecated as of 3.5.0. Use CG_Volume instead.

Availability: 2.2.0

 Diese Methode benötigt ein SFCGAL-Backend.

8.4.14 ST_MakeSolid

ST_MakeSolid — Deprecated alias for `CG_MakeSolid`.

Synopsis

geometry **ST_MakeSolid**(geometry geom1);

Description



Warnung
`ST_MakeSolid` is deprecated as of 3.5.0. Use CG_MakeSolid instead.

Availability: 2.2.0

 Diese Methode benötigt ein SFCGAL-Backend.

8.4.15 ST_IsSolid

ST_IsSolid — Deprecated alias for `CG_IsSolid`.

Synopsis

boolean **ST_IsSolid**(geometry geom1);

Description



Warnung
`ST_IsSolid` is deprecated as of 3.5.0. Use CG_IsSolid instead.

Availability: 2.2.0

 Diese Methode benötigt ein SFCGAL-Backend.

8.4.16 ST_ConstrainedDelaunayTriangles

ST_ConstrainedDelaunayTriangles — Deprecated alias for `CG_ConstrainedDelaunayTriangles`.

Synopsis

geometry **ST_ConstrainedDelaunayTriangles**(geometry g1);

Description



Warnung
`ST_ConstrainedDelaunayTriangles` is deprecated as of 3.5.0. Use CG_ConstrainedDelaunayTriangles instead.

Availability: 3.0.0


Diese Methode benötigt ein SFCGAL-Backend.

8.4.17 ST_3DConvexHull

ST_3DConvexHull — Deprecated alias for `CG_3DConvexHull`.

Synopsis

geometry **ST_3DConvexHull**(geometry geom1);

Description



Warnung
`ST_3DConvexHull` is deprecated as of 3.5.0. Use CG_3DConvexHull instead.

Availability: 3.3.0


Diese Methode benötigt ein SFCGAL-Backend.

8.4.18 ST_AlphaShape

ST_AlphaShape — Deprecated alias for `CG_AlphaShape`.

Synopsis

geometry **ST_AlphaShape**(geometry geom, float alpha , boolean allow_holes = false);

Description



Warnung
`ST_AlphaShape` is deprecated as of 3.5.0. Use CG_AlphaShape instead.


Diese Methode benötigt ein SFCGAL-Backend.

8.4.19 ST_OptimalAlphaShape

ST_OptimalAlphaShape — Deprecated alias for `CG_OptimalAlphaShape`.

Synopsis

geometry **ST_OptimalAlphaShape**(geometry geom, boolean allow_holes = false , integer nb_components = 1);

Description



Warnung
`ST_OptimalAlphaShape` is deprecated as of 3.5.0. Use CG_OptimalAlphaShape instead.

Availability: 3.3.0 - requires SFCGAL >= 1.4.1.


Diese Methode benötigt ein SFCGAL-Backend.

Kapitel 9

Topologie

Die topologischen Datentypen und Funktionen von PostGIS werden für die Verwaltung von topologischen Objekten wie Maschen, Kanten und Knoten verwendet.

Sandro Santilli's Vortrag auf der Tagung "PostGIS Day Paris 2011" liefert eine gute Übersicht über die PostGIS Topologie und deren Perspektiven Topology with PostGIS 2.0 slide deck.

Vincent Picavet gibt in PostGIS Topology PGConf EU 2012 einen guten Überblick darüber, was Topologie ist, wie sie verwendet wird und welche FOSS4G-Tools sie unterstützen.

An example of a topologically based GIS database is the US Census Topologically Integrated Geographic Encoding and Referencing System (TIGER) database. If you want to experiment with PostGIS topology and need some data.

The PostGIS topology module has existed for a long time but was not always part of the official documentation. Extensive cleanup removed deprecated functions, fixed known usability issues, documented the features and functions, added new functionality, and improved SQL-MM compliance.

This chapter is the maintained reference for the topology module, including its primitive tables, TopoGeometry type, topology management functions, editing functions, validation functions, and conversion helpers.

Alle Funktionen und Tabellen, die zu diesem Modul gehören, sind im Schema mit der Bezeichnung `topology` installiert.

Funktionen die im SQL/MM Standard definiert sind erhalten das Präfix ST_, PostGIS eigene Funktionen erhalten kein Präfix.

Topology support is built by default and can be disabled by specifying the --without-topology configure option at build time as described in Kapitel 2

9.1 Building Faces from Linework

This example builds topology faces from noded linework and associates point attributes with the resulting faces. It uses ST_CreateTopoGe
to populate an empty topology, ValidateTopology to check the result, GetFaceByPoint to locate each attribute point, and ST_GetFaceGeor
to materialize ordinary polygon geometries.

Create an empty topology and populate it with linework that encloses two faces:

Code

```
CREATE EXTENSION IF NOT EXISTS postgis_topology;

SELECT topology.CreateTopology('topo_example', 3857);

WITH lines(geom) AS (VALUES
    ('SRID=3857;LINESTRING(0 0, 2 0)'::geometry),
    ('SRID=3857;LINESTRING(0 1, 2 1)'::geometry),
    ('SRID=3857;LINESTRING(0 0, 0 1)'::geometry),
    ('SRID=3857;LINESTRING(1 0, 1 1)'::geometry),
```

```
    ('SRID=3857;LINESTRING(2 0, 2 1)'::geometry)
)
SELECT topology.ST_CreateTopoGeo(
    'topo_example',
    ST_Collect(geom)
)
FROM lines;

SELECT * FROM topology.ValidateTopology('topo_example');
```

An empty result from `ValidateTopology` means that no topology errors were found. Attribute points can now be matched to faces. A face identifier of `0` denotes the universal face, so the point is outside every bounded face.

Code

```
WITH labels(name, geom) AS (VALUES
    ('west', 'SRID=3857;POINT(0.5 0.5)'::geometry),
    ('east', 'SRID=3857;POINT(1.5 0.5)'::geometry),
    ('outside', 'SRID=3857;POINT(3 0.5)'::geometry)
),
located AS (
    SELECT name,
           topology.GetFaceByPoint('topo_example', geom, 0) AS face_id
    FROM labels
)
SELECT name,
       face_id,
       CASE WHEN face_id = 0 THEN NULL
            ELSE topology.ST_GetFaceGeometry('topo_example', face_id)
       END AS geom
FROM located
ORDER BY name;
```

Before copying attributes into a face table, check both for points outside bounded faces and for multiple source points assigned to the same face:

Code

```
WITH labels(name, geom) AS (VALUES
    ('west', 'SRID=3857;POINT(0.5 0.5)'::geometry),
    ('east', 'SRID=3857;POINT(1.5 0.5)'::geometry),
    ('outside', 'SRID=3857;POINT(3 0.5)'::geometry)
),
located AS (
    SELECT name,
           topology.GetFaceByPoint('topo_example', geom, 0) AS face_id
    FROM labels
)
SELECT 'outside face' AS problem,
       array_agg(name ORDER BY name) AS labels
FROM located
WHERE face_id = 0
HAVING count(*) > 0
UNION ALL
SELECT 'duplicate face ' || face_id,
       array_agg(name ORDER BY name)
FROM located
WHERE face_id != 0
GROUP BY face_id
HAVING count(*) > 1;
```

Use DropTopology to remove the example topology when it is no longer needed:

Code

```
SELECT topology.DropTopology('topo_example');
```

9.2 Topology Primitive Tables

The core primitives of any topology are stored in the `edge_data`, `node`, and `face` tables that live in the schema created by CreateTopology. Each row of `edge_data` represents an oriented edge: it records a directed curve from `start_node` to `end_node` together with the identifier of the face encountered on the left of that direction (`left_face`) and the face encountered on the right (`right_face`). The same geometric segment may therefore appear twice—once for each orientation—when it belongs to two faces.

The `next_left_edge` and `next_right_edge` columns complete this orientation information by encoding how to keep walking around a face. They store signed integers whose absolute value is the identifier of the next oriented edge and whose sign determines whether the stored orientation has to be followed as-is or reversed when traversing. Formally, the following rules hold for every edge `e`:

- `abs(next_left_edge)` is the identifier of the edge reached by continuing around the face that lies to the left of `e`. If the value is positive the walk continues from the end node of `e` along the stored orientation of the referenced edge; if it is negative the referenced edge must be followed backwards so that the shared face remains on the walker's left.
- `abs(next_right_edge)` analogously follows the boundary of the face located on the right of `e`. A positive value means that the next edge is taken with its recorded orientation starting from the end node of `e`, whereas a negative value instructs to traverse the referenced edge in reverse, starting from its end node, so that the right-hand face is preserved.
- A zero value indicates that the edge is dangling on the corresponding side (for example an isolated edge whose incident face is the universal face `0`). The `abs_next_left_edge` and `abs_next_right_edge` columns exposed by the `edge` view are convenience projections of these absolute values.

This representation is a variant of a doubly connected edge list and is exploited by many topology routines. Functions such as GetRingEdges and ValidateTopology rely on it to reconstruct face boundaries and to diagnose inconsistencies—hence the "invalid next_left_edge" and "invalid next_right_edge" diagnostics reported during validation. Constructors like AddEdge initialise the `next_*` attributes with trivial self references, while editing routines including ST_AddEdgeModFace and ST_RemEdgeModFace update the links as edges are inserted or removed. Other bulk operations (for example Polygonize) may intentionally leave the fields unset, which is why the documentation flags their behaviour explicitly.

9.3 Topologische Datentypen

9.3.1 getfaceedges_returntype

getfaceedges_returntype — Ein zusammengesetzter Typ, der aus einer Sequenznummer und einer Randnummer besteht.

Description

Ein zusammengesetzter Typ, der aus einer Sequenznummer und einer Kantennummer besteht. Dies ist der Rückgabetyp für die Funktionen `ST_GetFaceEdges` und `GetNodeEdges`.

1. `sequence` ist eine Ganzzahl: Sie verweist auf eine Topologie, die in der Tabelle topology.topology definiert ist. In dieser Tabelle sind das Schema, das die Topologie enthält, und die SRID verzeichnet.
2. `edge` ist eine Ganzzahl: Der Identifikator einer Kante.

9.3.2 TopoGeometry

TopoGeometry — Ein zusammengesetzter Typ, der eine topologisch festgelegte Geometrie darstellt.

Description

Ein zusammengesetzter Datentyp, der auf eine topologische Geometrie in einem bestimmten topologischen Layer verweist und einen spezifischen Datentyp und eine eindeutige ID hat. Folgende Bestandteile bilden die Elemente einer TopoGeometry: topology_id, layer_id, id Ganzzahl, type Ganzzahl.

1. `topology_id` ist eine Ganzzahl: Sie verweist auf eine Topologie, die in der Tabelle topology.topology definiert ist. In dieser Tabelle sind das Schema, das die Topologie enthält, und die SRID verzeichnet.
2. `layer_id` ist eine Ganzzahl: Die layer_id in der Tabelle topology.layer zu der die TopoGeometry gehört. Die Kombination aus topology_id und layer_id liefert eine eindeutige Referenz in der Tabelle topology.layer.
3. `id` ist eine Ganzzahl: Die id ist eine automatisch erzeugte Sequenznummer, welche die TopoGeometry in dem jeweiligen topologischen Layer eindeutig ausweist.
4. `type` ist eine Ganzzahl zwischen 1 und 4, welche den geometrischen Datentyp festlegt: 1:[Multi]Point, 2:[Multi]Line, 3:[Multi]Polygon, 4:GeometryCollection

Verhaltensweise bei der Typumwandlung

In diesem Abschnitt sind die für diesen Datentyp erlaubten impliziten und expliziten Typumwandlungen beschrieben.

Typumwandlung nach	Verhaltensweise
Geometrie	implizit

Siehe auch

CreateTopoGeom

9.3.3 validatetopology_returntype

validatetopology_returntype — Ein zusammengesetzter Datentyp, der aus einer Fehlermeldung und id1 und id2 besteht. id1 und id2 deuten auf die Stelle hin, an der der Fehler auftrat. Dies ist der von `ValidateTopology` zurückgegebene Datentyp.

Description

Ein zusammengesetzter Datentyp, der aus einer Fehlermeldung und zwei Ganzzahlen besteht. Die Funktion ValidateTopology gibt eine Menge dieser Datentypen zurück, um bei der Validierung gefundene Fehler zu beschreiben. Unter id1 und id2 sind die ids der topologischen Objekte verzeichnet, die an dem Fehler beteiligt sind.

Field	Type	Description
`error`	varchar	Denotes the type of error. Current error descriptors are: coincident nodes, edge crosses node, edge not simple, edge end node geometry mismatch, edge start node geometry mismatch, face overlaps face, face within face.
`id1`	integer	Denotes the identifier of the edge, face, or node in error.

Field	Type	Description
id2	integer	For errors that involve two objects, denotes the secondary edge or node.

Siehe auch

ValidateTopology

9.4 Topologische Domänen

9.4.1 TopoElement

TopoElement — Ein Feld mit 2 Ganzzahlen, welches in der Regel für die Auffindung einer Komponente einer TopoGeometry dient.

Description

Ein Feld mit 2 Ganzzahlen, welches einen Bestandteil einer einfachen oder hierarchischen TopoGeometry abbildet.

Im Falle einer einfachen TopoGeometry ist das erste Element des Feldes der Identifikator einer topologischen Elementarstruktur und das zweite Element der Typ (1:Knoten, 2:Kante, 3:Masche). Im Falle einer hierarchischen TopoGeometry ist das erste Element des Feldes der Identifikator der Kind-TopoGeometry und das zweite Element der Identifikator des Layers.



Anmerkung
Bei jeder gegebenen hierarchischen TopoGeometry werden alle Kindklassen der TopoGeometrie vom selben Kindlayer abgeleitet, so wie dies in dem Datensatz von "topology.layer" für den Layer der TopoGeometry definiert ist.

Beispiele

Code

```
SELECT te[1] AS id, te[2] AS type FROM
( SELECT ARRAY[1, 2]::topology.topoelement AS te ) f;
```

Output

```
id | type
----+------
  1 |    2
```

Code

```
SELECT ARRAY[1, 2]::topology.topoelement;
```

Output

```
te
-------
 {1,2}
```

This example shows what happens when casting a three-element array to topology.topoelement. The domain requires a two-element array, so the dimension check fails.

Code

```
SELECT ARRAY[1, 2, 3]::topology.topoelement;
```

Output

```
ERROR:  value for domain topology.topoelement violates check constraint "dimensions"
```

Siehe auch

GetTopoGeomElements, TopoElementArray, TopoGeometry, TopoGeom_addElement, TopoGeom_remElement

9.4.2 TopoElementArray

TopoElementArray — Ein Feld mit TopoElement Objekten.

Description

Ein Feld mit 1 oder mehreren TopoElement Objekten; wird hauptsächlich verwendet, um Bestandteile einer TopoGeometry herumzureichen.

Beispiele

Code

```
SELECT '{{1,2},{4,3}}'::topology.topoelementarray AS tea;
```

Output

```
{{1,2},{4,3}}
```

This is the more verbose equivalent.

Code

```
SELECT ARRAY[ARRAY[1, 2], ARRAY[4, 3]]::topology.topoelementarray AS tea;
```

Output

```
{{1,2},{4,3}}
```

This example uses the array aggregate function packaged with topology.

Code

```
SELECT topology.TopoElementArray_Agg(ARRAY[e, t] ORDER BY e, t) As tea
FROM generate_series(1, 4) As e CROSS JOIN generate_series(1, 3) As t;
```

Output

```
{{1,1},{1,2},{1,3},{2,1},{2,2},{2,3},{3,1},{3,2},{3,3},{4,1},{4,2},{4,3}}
```

Code

```
SELECT '{{1,2,4},{3,4,5}}'::topology.topoelementarray AS tea;
```

Output

```
ERROR:  value for domain topology.topoelementarray violates check constraint "dimensions"
```

Siehe auch

TopoElement, GetTopoGeomElementArray, TopoElementArray_Agg

9.5 Verwaltung von Topologie und TopoGeometry

9.5.1 AddTopoGeometryColumn

AddTopoGeometryColumn — Fügt ein TopoGeometry Attribut an eine bestehende Tabelle an, registriert dieses neue Attribut als einen Layer in topology.layer und gibt die neue layer_id zurück.

Synopsis

integer **AddTopoGeometryColumn**(name topology_name, name schema_name, name table_name, name column_name, varchar feature_type, integer child_layer);
integer **AddTopoGeometryColumn**(name topology_name, regclass tab, name column_name, integer layer_id, varchar feature_type, integer child_layer);

Description

Jedes TopoGeometry Objekt gehört zu einem bestimmten Layer einer bestimmten Topologie. Bevor Sie ein TopoGeometry Objekt erzeugen, müssen Sie dessen topologischen Layer erzeugen. Ein topologischer Layer ist eine Assoziation einer Featuretabelle mit der Topologie. Er enthält auch Angaben zum Typ und zur Hierarchie. Man erzeugt einen Layer mittels der Funktion AddTopoGeometryColumn():

Diese Funktion fügt sowohl das angeforderte Attribut an die Tabelle als auch einen Datensatz mit der angegebenen Information in die Tabelle topology.layer.

Wenn Sie den [child_layer] nicht angeben (oder auf NULL setzen), dann enthält dieser Layer elementare TopoGeometries (aus topologischen Elementarstrukturen zusammengesetzt). Andernfalls enthält dieser Layer hierarchische TopoGeometries (aus den TopoGeometries des child_layer zusammengesetzt).

Sobald der Layer erstellt wurde (seine id von der Funktion AddTopoGeometryColumn zurückgegeben wurde), können Sie TopoGeometry Objekte in ihm erzeugen

Gültige `feature_types` sind: PUNKT, MULTIPUNKT, LINIE, MULTILINIE, POLYGON, MULTIPOLYGON, SAMMLUNG

Verfügbarkeit: 1.1

Beispiele

This example creates the new table in the `ma_topo` schema. The table could be created in a different schema, in which case `topology_name` and `schema_name` would be different.

Code

```
CREATE SCHEMA ma;
CREATE TABLE ma.parcels(gid serial, parcel_id varchar(20) PRIMARY KEY, address text);
SELECT topology.AddTopoGeometryColumn('ma_topo', 'ma', 'parcels', 'topo', 'POLYGON');
```

Code

```
CREATE SCHEMA ri;
CREATE TABLE ri.roads(gid serial PRIMARY KEY, road_name text);
SELECT topology.AddTopoGeometryColumn('ri_topo', 'ri', 'roads', 'topo', 'LINE');
```

Siehe auch

DropTopoGeometryColumn, toTopoGeom, CreateTopology, CreateTopoGeom

9.5.2 RenameTopoGeometryColumn

RenameTopoGeometryColumn — Benennt eine topogeometrische Spalte um

Synopsis

topology.layer **RenameTopoGeometryColumn**(regclass layer_table, name feature_column, name new_name);

Description

Diese Funktion ändert den Namen einer bestehenden TopoGeometry-Spalte und stellt sicher, dass die Metadateninformationen entsprechend aktualisiert werden.

Verfügbarkeit: 3.4.0

Beispiele

Code

```
SELECT topology.RenameTopoGeometryColumn('public.parcels', 'topogeom', 'tgeom');
```

Siehe auch

AddTopoGeometryColumn, RenameTopology

9.5.3 DropTopology

DropTopology — Bitte mit Vorsicht verwenden: Löscht ein topologisches Schema und dessen Referenz in der Tabelle topology.topology, sowie die Referenzen zu den Tabellen in diesem Schema aus der Tabelle geometry_columns.

Synopsis

integer **DropTopology**(varchar topology_schema_name);

Description

Löscht ein topologisches Schema und dessen Referenz in der Tabelle topology.topology, sowie die Referenzen zu den Tabellen in diesem Schema aus der Tabelle geometry_columns. Diese Funktion sollte MIT VORSICHT BENUTZT werden, da damit unabsichtlich Daten zerstört werden können. Falls das Schema nicht existiert, werden nur die Referenzeinträge des bezeichneten Schemas gelöscht.

Verfügbarkeit: 1.1

Beispiele

Löscht das Schema "ma_topo" kaskadierend und entfernt alle Referenzen in topology.topology und in geometry_columns.

Code

```
SELECT topology.DropTopology('ma_topo');
```

Siehe auch

DropTopoGeometryColumn

9.5.4 RenameTopology

RenameTopology — Benennt eine Topologie um

Synopsis

varchar **RenameTopology**(varchar old_name, varchar new_name);

Description

Benennt ein Topologieschema um und aktualisiert seinen Metadatensatz in der Tabelle `topology.topology`.

Verfügbarkeit: 3.4.0

Beispiele

Umbenennen einer Topologie von `topo_stage` in `topo_prod`.

Code

```
SELECT topology.RenameTopology('topo_stage', 'topo_prod');
```

Siehe auch

CopyTopology, RenameTopoGeometryColumn

9.5.5 DropTopoGeometryColumn

DropTopoGeometryColumn — Entfernt ein TopoGeometry-Attribut aus der Tabelle mit der Bezeichnung `table_name` im Schema `schema_name` und entfernt die Registrierung der Attribute aus der Tabelle "topology.layer".

Synopsis

text **DropTopoGeometryColumn**(varchar schema_name, varchar table_name, varchar column_name);

Description

Entfernt ein TopoGeometry-Attribut aus der Tabelle mit der Bezeichnung `table_name` im Schema `schema_name` und entfernt die Registrierung der Attribute aus der Tabelle "topology.layer". Gibt eine Zusammenfassung des Löschstatus aus. ANMERKUNG: vor dem Löschen werden alle Werte auf NULL gesetzt, um die Überprüfung der referenziellen Integrität zu umgehen.

Verfügbarkeit: 1.1

Beispiele

Code

```
SELECT topology.DropTopoGeometryColumn('ma_topo', 'parcel_topo', 'topo');
```

Siehe auch

AddTopoGeometryColumn

9.5.6 FixCorruptTopoGeometryColumn

FixCorruptTopoGeometryColumn — Fixes topogeometry corruption caused by upgrade to postgis_topology 3.6.0 and higher

Synopsis

text **FixCorruptTopoGeometryColumn**(name layerSchema, name layerTable, name layerColumn);

Description

When upgrading from PostGIS topology <3.6.0 to version >3.6.0+, the topogeometry column definition was changed. This caused corruption in topogeometries created before the upgrade. This function fixes this corruption in affected tables.

The function refuses to process tables with expression or partial indexes; drop or rebuild those indexes manually around the repair.

Availability: 3.6.1

Beispiele

Fix all topology columns

Code

```
SELECT topology.FixCorruptTopoGeometryColumn(schema_name, table_name, feature_column)
    FROM topology.layer;
```

Siehe auch

UpgradeTopology

9.5.7 Populate_Topology_Layer

Populate_Topology_Layer — Fügt fehlende Einträge zu der Tabelle topology.layer hinzu, indem Metadaten aus den topologischen Tabellen ausgelesen werden.

Synopsis

table(schema_name text, table_name text, feature_column text) **Populate_Topology_Layer**();

Description

Trägt fehlende Einträge in der Tabelle `topology.layer` ein, indem die topologischen Constraints und Tabellen inspiziert werden. Diese Funktion ist nach der Wiederherstellung von Schemata mit topologischen Daten sinnvoll, um Einträge im Topologie-Katalog zu reparieren.

Wirft eine Liste der erzeugten Einträge aus. Die zurückgegebenen Attribute sind `schema_name`, `table_name` und `feature_colur`

Verfügbarkeit: 2.3.0

Beispiele

Code

```
SELECT CreateTopology('strk_topo');
CREATE SCHEMA strk;
CREATE TABLE strk.parcels(gid serial, parcel_id varchar(20) PRIMARY KEY, address text);
SELECT topology.AddTopoGeometryColumn('strk_topo', 'strk', 'parcels', 'topo', 'POLYGON');
```

Output

```
createtopology
----------------
              1
(1 row)

CREATE SCHEMA
CREATE TABLE
 addtopogeometrycolumn
-----------------------
                     1
(1 row)
```

This query returns no records because the feature is already registered.

Code

```
SELECT *
FROM topology.Populate_Topology_Layer();
```

Output

```
(0 rows)
```

Rebuild the topology layer catalog.

Code

```
TRUNCATE TABLE topology.layer;
```

Output

```
TRUNCATE TABLE
```

Code

```
SELECT *
FROM topology.Populate_Topology_Layer();
```

Output

```
schema_name | table_name | feature_column
-------------+------------+----------------
 strk        | parcels    | topo
(1 row)
```

Code

```
SELECT topology_id, layer_id,
       schema_name AS sn, table_name AS tn, feature_column AS fc
FROM topology.layer;
```

Output

```
topology_id | layer_id |  sn  |   tn    |  fc
-------------+----------+------+---------+------
           1 |        1 | strk | parcels | topo
(1 row)
```

Siehe auch

AddTopoGeometryColumn

9.5.8 TopologySummary

TopologySummary — Nimmt den Namen einer Topologie und liefert eine Zusammenfassung der Gesamtsummen der Typen und Objekte in der Topologie.

Synopsis

text **TopologySummary**(varchar topology_schema_name);

Description

Nimmt den Namen einer Topologie und liefert eine Zusammenfassung der Gesamtsummen der Typen und Objekte in der Topologie.

Verfügbarkeit: 2.0.0

Beispiele

Code

```
SELECT topology.TopologySummary('city_data');
```

Output

```
Topology city_data (329), SRID 4326, precision: 0
22 nodes, 24 edges, 10 faces, 29 topogeoms in 5 layers
Layer 1, type Polygonal (3), 9 topogeoms
 Deploy: features.land_parcels.feature
Layer 2, type Puntal (1), 8 topogeoms
 Deploy: features.traffic_signs.feature
Layer 3, type Lineal (2), 8 topogeoms
 Deploy: features.city_streets.feature
Layer 4, type Polygonal (3), 3 topogeoms
 Hierarchy level 1, child layer 1
 Deploy: features.big_parcels.feature
Layer 5, type Puntal (1), 1 topogeoms
 Hierarchy level 1, child layer 2
 Deploy: features.big_signs.feature
```

9.5.9 ValidateTopology

ValidateTopology — Liefert eine Menge validatetopology_returntype Objekte, die Probleme mit der Topologie beschreiben.

Synopsis

setof validatetopology_returntype **ValidateTopology**(varchar toponame, geometry bbox);

Description

Gibt eine Reihe von validatetopology_returntype Objekten zurück, die Probleme mit der Topologie aufzeigen, wobei die Prüfung optional auf den durch den Parameter `bbox` angegebenen Bereich beschränkt werden kann.

Die Liste der möglichen Fehler, ihre Bedeutung und die zurückgegebenen IDs sind unten aufgeführt:

Error	id1	id2	Bedeutung
übereinstimmende Knotenpunkte	Kennung des ersten Knotens.	Kennung des zweiten Knotens.	Zwei Knoten haben die gleiche Geometrie.
edge crosses node	Kennung der Kante.	Bezeichner des Knotens.	Eine Kante hat einen Knoten in ihrem Inneren. Siehe ST_Relate.
invalid edge	Kennung der Kante.		Eine Kantengeometrie ist ungültig. Siehe ST_IsValid.
edge not simple	Kennung der Kante.		Eine Kantengeometrie hat Selbstschnittpunkte. Siehe ST_IsSimple.
edge crosses edge	Kennung der ersten Kante.	Kennung der zweiten Kante.	Zwei Kanten haben einen inneren Schnittpunkt. Siehe ST_Relate.

Error	id1	id2	Bedeutung
edge start node geometry mismatch	Kennung der Kante.	Kennung des angegebenen Startknotens.	Die Geometrie des Knotens, der als Startknoten für eine Kante angegeben ist, stimmt nicht mit dem ersten Punkt der Kantengeometrie überein. Siehe ST_StartPoint.
edge end node geometry mismatch	Kennung der Kante.	Kennung des angegebenen Endknotens.	Die Geometrie des Knotens, der als Endknoten für eine Kante angegeben ist, stimmt nicht mit dem letzten Punkt der Kantengeometrie überein. Siehe ST_EndPoint.
face without edges	Kennung des verwaisten Gesichts.		Keine Kante meldet eine vorhandene Fläche auf einer ihrer Seiten (left_face, right_face).
face has no rings	Kennung der teilweise definierten Fläche.		Kanten, die an ihren Seiten eine Fläche aufweisen, bilden keinen Ring.
Gesicht hat falsches mbr	Kennung des Gesichts mit falschem mbr-Cache.		Das minimale Begrenzungsrechteck einer Fläche stimmt nicht mit dem minimalen Begrenzungsrahmen der Sammlung von Kanten überein, die die Fläche an ihren Seiten darstellen.
Loch nicht im beworbenen Gesicht	Signierter Bezeichner einer Kante, der den Ring identifiziert. Siehe GetRingEdges.		Ein Ring von Kanten, der eine Fläche auf seiner Außenseite meldet, ist in einer anderen Fläche enthalten.
nicht-isolierter Knoten hat nicht-enthaltende_Fläche	Kennung des schlecht definierten Knotens.		Ein Knoten, der als an der Grenze einer oder mehrerer Kanten liegend gemeldet wird, zeigt eine enthaltende Fläche an.
isolierter Knoten hat containing_face	Kennung des schlecht definierten Knotens.		Einem Knoten, der nicht als auf der Grenze einer Kante liegend gemeldet wird, fehlt der Hinweis auf eine enthaltende Fläche.
isolierter Knoten hat falsche containing_face	Kennung des falsch dargestellten Knotens.		Ein Knoten, der nicht als auf dem Rand einer Kante liegend gemeldet wird, weist auf eine enthaltende Fläche hin, die nicht die eigentliche Fläche ist, die ihn enthält. Siehe GetFaceContainingPoint.
ungültig next_right_edge	Kennung der falsch dargestellten Kante.	Vorzeichenlose ID der Kante, die als nächste rechte Kante angezeigt werden soll.	Die Kante, die als nächste Kante angegeben ist, wenn man auf der rechten Seite einer Kante geht, ist falsch.

Error	id1	id2	Bedeutung
ungültig nächster_linker_Rand	Kennung der falsch dargestellten Kante.	Signierte ID der Kante, die als nächste linke Kante angezeigt werden soll.	Die Kante, die als nächste Kante angegeben ist, wenn man auf der linken Seite einer Kante geht, ist falsch.
gemischte Gesichtsauszeichnung im Ring	Signierter Bezeichner einer Kante, der den Ring identifiziert. Siehe GetRingEdges.		Kanten in einem Ring weisen auf kollidierende Flächen auf der gehenden Seite hin. Dies wird auch als "Seitenlagekonflikt" bezeichnet.
nicht geschlossener Ring	Signierter Bezeichner einer Kante, der den Ring identifiziert. Siehe GetRingEdges.		Ein Ring von Kanten, der durch die Attribute next_left_edge/next_right_edge gebildet wird, beginnt und endet an verschiedenen Knoten.
Gesicht hat mehrere Schalen	Kennung der umstrittenen Fläche.	Signierter Bezeichner einer Kante, der den Ring identifiziert. Siehe GetRingEdges.	Mehr als ein Kranz von Kanten weist auf dieselbe Fläche im Inneren hin.

Verfügbarkeit: 1.0.0

Erweiterung: 2.0.0 effizientere Ermittlung sich überkreuzender Kanten. Falsch positive Fehlmeldungen von früheren Versionen fixiert.

Änderung: 2.2.0 Bei 'edge crosses node' wurden die Werte für id1 und id2 vertauscht, um mit der Fehlerbeschreibung konsistent zu sein.

Geändert: 3.2.0 fügte den optionalen bbox-Parameter hinzu und führte Prüfungen der Flächenbeschriftung und der Kantenverknüpfung durch.

Beispiele

Code

```
SELECT * FROM topology.ValidateTopology('ma_topo');
```

Output

```
error         | id1 | id2
------------------+-----+-----
face without edges |   1 |
```

Siehe auch

validatetopology_returntype

9.5.10 ValidateTopologyRelation

ValidateTopologyRelation — Gibt Informationen über ungültige Topologiebeziehungssätze zurück

Synopsis

table(error text, layer_id int, topogeo_id bigint, element_id bigint) **ValidateTopologyRelation**(varchar toponame);

Description

Gibt eine Reihe von Datensätzen mit Informationen über Ungültigkeiten in der Beziehungstabelle der Topologie zurück.

Verfügbarkeit: 3.2.0

Siehe auch

ValidateTopology

9.5.11 ValidateTopologyPrecision

ValidateTopologyPrecision — Returns non-precise vertices in the topology.

Synopsis

geometry **ValidateTopologyPrecision**(name toponame, geometry bbox, float8 gridSize);

Description

Returns all vertices that are not rounded to the topology or given `gridSize` as a puntal geometry, optionally limiting the check to the area specified by the `bbox` parameter.

Verfügbarkeit: 3.6.0

Beispiele

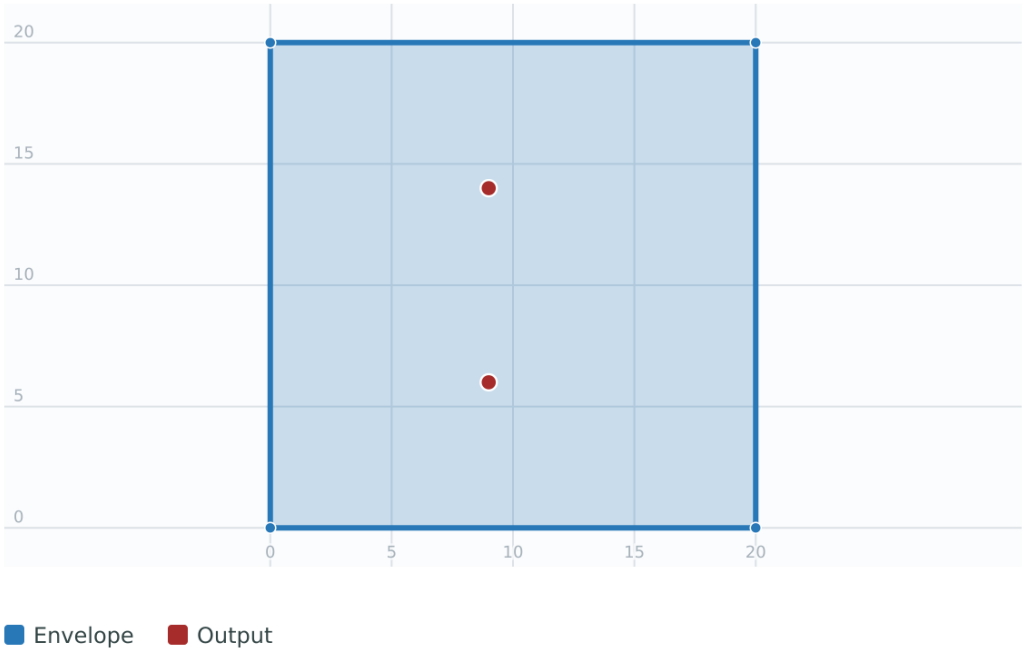
Code

```
SELECT g
FROM topology.ValidateTopologyPrecision(
    'city_data',
    gridSize =
> 2,
    bbox =
> ST_MakeEnvelope(0, 0, 20, 20)
) g;
```

Output

```
MULTIPOINT(9 6,9 14)
```

Figure



Siehe auch

MakeTopologyPrecise

9.5.12 MakeTopologyPrecise

MakeTopologyPrecise — Snap topology vertices to precision grid.

Synopsis

void **MakeTopologyPrecise**(name toponame, geometry bbox, float8 gridSize);

Description

Snaps all vertices of a topology to the topology precision grid or to the grid whose size is specified with the `gridSize` parameter, optionally limiting the operation to the objects intersecting the area specified by the `bbox` parameter.

Using a `gridSize` larger than the smallest edge extent raises an exception rather than collapsing the edge into an empty geometry. This safeguards adjacent topological relationships by preventing precision snapping from removing primitives altogether.



Anmerkung
Snapping could make the topology invalid, so it is recommended to check the outcome of operation with ValidateTopology.

Verfügbarkeit: 3.6.0

Beispiele

Code

```
SELECT topology.MakeTopologyPrecise(
    'city_data',
    gridSize =
> 2
);
```

Siehe auch

ValidateTopologyPrecision, ValidateTopology

9.5.13 FindTopology

FindTopology — Gibt einen Topologie-Datensatz mit anderen Mitteln zurück.

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

Nimmt einen Topologiebezeichner oder den Bezeichner eines topologiebezogenen Objekts und gibt einen topology.topology-Datensatz zurück.

Verfügbarkeit: 3.2.0

Beispiele

Code

```
SELECT name(FindTopology('features.land_parcels', 'feature'));
```

Output

```
city_data
```

Siehe auch

FindLayer

9.5.14 FindLayer

FindLayer — Gibt einen topology.layer-Datensatz mit anderen Mitteln zurück.

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

Nimmt einen Layer-Identifikator oder den Identifikator eines topologiebezogenen Objekts und gibt einen topology.layer-Datensatz zurück.

Verfügbarkeit: 3.2.0

Beispiele

Code

```
SELECT layer_id(FindLayer('features.land_parcels', 'feature'));
```

Output

```
1
```

Siehe auch

FindTopology

9.5.15 TotalTopologySize

TotalTopologySize — Total disk space used by the specified topology, including all indexes and TOAST data.

Synopsis

int8 **TotalTopologySize**(name toponame);

Description

Takes a topology name and provides the total disk space used by all its tables, including indexes and TOAST data.

Verfügbarkeit: 3.6.0

Beispiele

Code

```
SELECT topology.TopologySummary('city_data');
```

Output

```
Topology city_data (329), SRID 4326, precision: 0
22 nodes, 24 edges, 10 faces, 29 topogeoms in 5 layers
Layer 1, type Polygonal (3), 9 topogeoms
 Deploy: features.land_parcels.feature
Layer 2, type Puntal (1), 8 topogeoms
 Deploy: features.traffic_signs.feature
Layer 3, type Lineal (2), 8 topogeoms
 Deploy: features.city_streets.feature
Layer 4, type Polygonal (3), 3 topogeoms
 Hierarchy level 1, child layer 1
 Deploy: features.big_parcels.feature
Layer 5, type Puntal (1), 1 topogeoms
 Hierarchy level 1, child layer 2
 Deploy: features.big_signs.feature
```

9.5.16 UpgradeTopology

UpgradeTopology — Upgrades the specified topology to support large ids (int8) for topology and primitive ids.

Synopsis

void **UpgradeTopology**(name toponame);

Description

Takes a topology name and upgrades it to support large ids (int8) for topology and primitive ids. The function upgrades the following: - face (face_id column from int4 to int8, face_id_seq from int4 to int8) - node (node_id column from int4 to int8, containing_face column from int4 to int8, node_id_seq from int4 to int8) - edge_data (edge_id column from int4 to int8, edge_data_edge_id_seq from int4 to int8, left_face and right_face columns from int4 to int8, start_node and end_node columns from int4 to int8, next_left_edge and next_right_edge columns from int4 to int8) - relation (topogeo_id column from int4 to int8, element_id from int4 to int8) - topology (useslargeids column set to true)

Verfügbarkeit: 3.6.0

Beispiele

Code

```
SELECT topology.UpgradeTopology('city_data');
```

9.6 Verwaltung der Topologie-Statistiken

Das Hinzufügen von Elementen zu einer Topologie löst viele Datenbankabfragen aus, um bestehende Kanten zu finden, die geteilt werden, Knoten hinzuzufügen und Kanten zu aktualisieren, die mit dem neuen Liniennetz verknotet werden. Aus diesem Grund ist es sinnvoll, dass die Statistiken über die Daten in den Topologietabellen aktuell sind.

Die PostGIS-Topologiebevölkerungs- und -bearbeitungsfunktionen aktualisieren die Statistiken nicht automatisch, da eine Aktualisierung der Statistiken nach jeder einzelnen Änderung in einer Topologie zu viel Aufwand wäre.



Anmerkung
Die von Autovacuum aktualisierten Statistiken sind für Transaktionen, die vor Beendigung des Autovacuum-Prozesses gestartet wurden, NICHT sichtbar, so dass lang laufende Transaktionen selbst ANALYZE ausführen müssen, um aktualisierte Statistiken zu verwenden.

9.7 Topologie Konstruktoren

9.7.1 CreateTopology

CreateTopology — Erstellt ein neues Topologie-Schema und trägt es in die Tabelle topology.topology ein.

Synopsis

integer **CreateTopology**(name topology_schema_name, integer srid, double precision prec=0, boolean hasz=false, integer topoid=0, boolean useslargeids=false);

Description

Erstellt ein neues Topologie-Schema mit dem Namen `topology_name` und registriert es in der Tabelle `topology.topology`. Topologien müssen eindeutig benannt sein. Die Topologietabellen (`edge_data`, `face`, `node`, und `relation` werden im Schema erstellt. Sie gibt die ID der Topologie zurück.

The `srid` is the spatial reference system SRID for the topology. The SRID defaults to -1 (unknown) if not specified.

Die Toleranz `prec` wird in den Einheiten des räumlichen Bezugssystems gemessen. Der Standardwert für die Toleranz ist 0.

`hasz` ist standardmäßig false, wenn nicht angegeben.

`topoid` optional explicit identifier (allows deterministic topology id assignment, needs to be unique)

`useslargeids` optional, defaults to false. If true, the topology will be created to support large ids (int8) for topology and primitive ids.

Sie ähnelt der SQL/MM ST_InitTopoGeo, hat aber mehr Funktionen.

Verfügbarkeit: 1.1

Enhanced: 2.0 added support for `hasZ`.

Enhanced: 3.6.0 added `topoid` and `usesLargeIDs` parameters.

Beispiele

Erstellen Sie ein Topologieschema mit der Bezeichnung `ma_topo`, das Kanten und Knoten in Massachusetts State Plane-Metern (SRID = 26986) speichert. Die Toleranz beträgt 0,5 Meter, da das räumliche Bezugssystem meterbasiert ist.

Code

```
SELECT topology.CreateTopology('ma_topo', 26986, 0.5);
```

Erstellen einer Topologie für Rhode Island mit der Bezeichnung `ri_topo` im Raumbezugssystem State Plane-feet (SRID = 3438)

Code

```
SELECT topology.CreateTopology('ri_topo', 3438) AS topoid;
```

Output

```
2
```

Siehe auch

Abschnitt 4.5, ST_InitTopoGeo

9.7.2 CopyTopology

CopyTopology — Erstellt eine Kopie einer Topologie (Knoten, Kanten, Flächen, Ebenen und TopoGeometrien) in ein neues Schema

Synopsis

integer **CopyTopology**(varchar existing_topology_name, varchar new_name);

Description

Erstellt eine neue Topologie mit dem Namen `new_name`, wobei SRID und Präzision von `existing_topology_name` kopiert werden. Die Knoten, Kanten und Flächen in `existing_topology_name` werden in die neue Topologie kopiert, ebenso wie die Layer und ihre zugehörigen TopoGeometrien.



Anmerkung
Die neuen Zeilen in der Tabelle `topology.layer` enthalten synthetische Werte für schema_name, table_name und feature_column. Dies liegt daran, dass die TopoGeometry-Objekte nur als Definition existieren und noch nicht in einer benutzerdefinierten Tabelle verfügbar sind.

Verfügbarkeit: 2.0.0

Beispiele

Erstellen Sie ein Backup einer Topologie mit dem Namen `ma_topo`.

Code

```
SELECT topology.CopyTopology('ma_topo', 'ma_topo_backup');
```

Siehe auch

Abschnitt 4.5, CreateTopology, RenameTopology

9.7.3 ST_InitTopoGeo

ST_InitTopoGeo — Erstellt ein neues Topologie-Schema und trägt es in die Tabelle topology.topology ein.

Synopsis

text **ST_InitTopoGeo**(varchar topology_schema_name);

Description

Dies ist das SQL-MM-Äquivalent zu CreateTopology. Es fehlen die Optionen für das räumliche Bezugssystem und die Toleranz. Es liefert eine Textbeschreibung der Topologieerstellung anstelle der Topologie-ID.

Verfügbarkeit: 1.1



Diese Methode setzt die SQL/MM-Spezifikation um. SQL-MM 3 Topo-Geo and Topo-Net 3: Routine Details: X.3.17

Beispiele

Code

```
SELECT topology.ST_InitTopoGeo('topo_schema_to_create') AS topocreation;
```

Output

```
Topology-Geometry 'topo_schema_to_create' (id:7) created.
```

Siehe auch

CreateTopology

9.7.4 ST_CreateTopoGeo

ST_CreateTopoGeo — Fügt eine Sammlung von Geometrien an eine leere Topologie an und gibt eine Bestätigungsmeldung aus.

Synopsis

text **ST_CreateTopoGeo**(varchar atopology, geometry acollection);

Description

Fügt eine Sammelgeometrie einer leeren Topologie hinzu und gibt eine Bestätigungsmeldung aus.

Nützlich um eine leere Topologie zu befüllen.

Verfügbarkeit: 2.0

Diese Methode setzt die SQL/MM-Spezifikation um. SQL-MM: Topo-Geo and Topo-Net 3: Routine Details -- X.3.18

Beispiele

Populate the topology.

Code

```
SELECT topology.ST_CreateTopoGeo(
'ri_topo',
ST_GeomFromText(
 'MULTILINESTRING((384744 236928,384750 236923,384769 236911,384799 236895,384811 ↩
    236890,384833 236884,
 384844 236882,384866 236881,384879 236883,384954 236898,385087 236932,385117 236938,
 385167 236938,385203 236941,385224 236946,385233 236950,385241 236956,385254 236971,
 385260 236979,385268 236999,385273 237018,385273 237037,385271 237047,385267 237057,
 385225 237125,385210 237144,385192 237161,385167 237192,385162 237202,385159 237214,
 385159 237227,385162 237241,385166 237256,385196 237324,385209 237345,385234 237375,
 385237 237383,385238 237399,385236 237407,385227 237419,385213 237430,385193 237439,
 385174 237451,385170 237455,385169 237460,385171 237475,385181 237503,385190 237521,
 385200 237533,385206 237538,385213 237541,385221 237542,385235 237540,385242 237541,
 385249 237544,385260 237555,385270 237570,385289 237584,385292 237589,385291 237596,385284 ↩
    237630))',
 3438)
 );
```

Output

```
Topology ri_topo populated
```

Create the tables and topogeometries.

Code

```
CREATE TABLE ri.roads(gid serial PRIMARY KEY, road_name text);

SELECT topology.AddTopoGeometryColumn('ri_topo', 'ri', 'roads', 'topo', 'LINE');
```

Siehe auch

TopoGeo_LoadGeometry, AddTopoGeometryColumn, CreateTopology, DropTopology

9.7.5 TopoGeo_AddPoint

TopoGeo_AddPoint — Fügt einen Punkt, unter Berücksichtigung einer Toleranz, an eine bestehende Topologie an. Existierende Kanten werden eventuell aufgetrennt.

Synopsis

bigint **TopoGeo_AddPoint**(varchar atopology, geometry apoint, float8 tolerance);

Description

Fügt einen Punkt an eine bestehende Topologie an und gibt dessen Identifikator zurück. Der vorgegebene Punkt wird von bestehenden Knoten oder Kanten, innerhalb einer gegebenen Toleranz, gefangen. Eine existierende Kante kann durch den gefangenen Punkt aufgetrennt werden.

Verfügbarkeit: 2.0.0

Siehe auch

TopoGeo_AddLineString, TopoGeo_AddPolygon, TopoGeo_LoadGeometry, AddNode, CreateTopology

9.7.6 TopoGeo_AddLineString

TopoGeo_AddLineString — Fügt einen Linienzug unter Berücksichtigung eines Toleranzwerts zu einer Topologie hinzu , wobei Kanten/Maschen dabei aufgetrennt werden können.

Synopsis

SETOF bigint **TopoGeo_AddLineString**(varchar atopology, geometry aline, float8 tolerance, int max_edges);

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, in addition to new edges.

The returned edge identifiers may be either existing edges or newly created edges as needed to fully represent the input line as closely as possible.

The number of newly created edges (either covering space previously uncovered or resulting from split of existing edges) may be limited by the `max_edges` parameter.



Anmerkung
When adding many lines, the load order can have a large impact on performance because each call checks the line against topology elements already loaded. In bulk workflows, adding large closed or enclosing lines before smaller contained lines can avoid expensive later splits. For loading a collection in one call, see TopoGeo_LoadGeometry.



Anmerkung
Die Aktualisierung von Statistiken über Topologien, die über diese Funktion geladen werden, obliegt dem Aufrufer, siehe maintaining statistics during topology editing and population.

Verfügbarkeit: 2.0.0

Erweitert: In Version 3.2.0 wurde Unterstützung für die Rückgabe signierter Identifikatoren hinzugefügt.

Enhanced: 3.7.0 added support for limiting the number of new edges created in the topology.

Siehe auch

TopoGeo_AddPoint, TopoGeo_AddPolygon, TopoGeo_LoadGeometry, AddEdge, CreateTopology

9.7.7 TopoGeo_AddPolygon

TopoGeo_AddPolygon — Fügt ein Polygon unter Berücksichtigung eines Toleranzwertes zu einer bestehenden Topologie hinzu, wobei bestehende Kanten/Maschen aufgetrennt werden können. Gibt Maschenidentifikatoren zurück.

Synopsis

SETOF bigint **TopoGeo_AddPolygon**(varchar atopology, geometry apoly, float8 tolerance);

Description

Fügt ein Polygon zu einer existierenden Topologie hinzu und gibt die Maschenidentifikatoren zurück, aus denen dieses aufgebaut ist. Die Begrenzung des gegebenen Polygons wird innerhalb einer gegebenen Toleranz an bestehenden Knoten und Kanten gefangen. Existierende Kanten und Maschen können entlang der Begrenzung des neuen Polygons aufgetrennt werden.



Anmerkung
Die Aktualisierung von Statistiken über Topologien, die über diese Funktion geladen werden, obliegt dem Aufrufer, siehe maintaining statistics during topology editing and population.

Verfügbarkeit: 2.0.0

Siehe auch

TopoGeo_AddPoint, TopoGeo_AddLineString, TopoGeo_LoadGeometry, AddFace, CreateTopology

9.7.8 TopoGeo_LoadGeometry

TopoGeo_LoadGeometry — Laden Sie eine Geometrie in eine bestehende Topologie und fangen und teilen Sie sie nach Bedarf.

Synopsis

void **TopoGeo_LoadGeometry**(varchar atopology, geometry ageom, float8 tolerance);

Description

Lädt eine Geometrie in eine bestehende Topologie. Die gegebene Geometrie rastet an vorhandenen Knoten oder Kanten innerhalb der gegebenen Toleranz ein. Vorhandene Kanten und Flächen können als Folge des Ladevorgangs aufgetrennt werden.



Anmerkung
Die Aktualisierung von Statistiken über Topologien, die über diese Funktion geladen werden, obliegt dem Aufrufer, siehe maintaining statistics during topology editing and population.

Verfügbarkeit: 3.5.0

Siehe auch

TopoGeo_AddPoint, TopoGeo_AddLineString, TopoGeo_AddPolygon, CreateTopology

9.8 Topologie Editoren

9.8.1 ST_AddIsoNode

ST_AddIsoNode — Fügt einen isolierten Knoten zu einer Masche in einer Topologie hinzu und gibt die "nodeid" des neuen Knotens aus. Falls die Masche NULL ist, wird der Knoten dennoch erstellt.

Synopsis

bigint **ST_AddIsoNode**(varchar atopology, bigint aface, geometry apoint);

Description

Fügt einen isolierten Knoten mit der Punktlage `apoint` zu einer bestehenden Masche mit der "faceid" `aface` zu einer Topologie `atopology` und gibt die "nodeid" des neuen Knoten aus.

Wenn das Koordinatenreferenzsystem (SRID) der Punktgeometrie nicht mit dem der Topologie übereinstimmt, `apoint` keine Punktgeometrie ist, der Punkt NULL ist, oder der Punkt eine bestehende Kante (auch an den Begrenzungen) schneidet, wird eine Fehlermeldung ausgegeben. Falls der Punkt bereits als Knoten existiert, wird ebenfalls eine Fehlermeldung ausgegeben.

Wenn `aface` nicht NULL ist und `apoint` nicht innerhalb der Masche liegt, wird eine Fehlermeldung ausgegeben.

Verfügbarkeit: 1.1

 Diese Methode setzt die SQL/MM-Spezifikation um. SQL-MM: Topo-Net Routines: X+1.3.1

Beispiele

Siehe auch

AddNode, CreateTopology, DropTopology, ST_Intersects

9.8.2 ST_AddIsoEdge

ST_AddIsoEdge — Fügt eine isolierte Kante, die durch die Geometrie `alinestring` festgelegt wird zu einer Topologie hinzu, indem zwei bestehende isolierte Knoten `anode` und `anothernode` verbunden werden. Gibt die "edgeid" der neuen Kante aus.

Synopsis

bigint **ST_AddIsoEdge**(varchar atopology, bigint anode, bigint anothernode, geometry alinestring);

Description

Fügt eine isolierte Kante, die durch die Geometrie `alinestring` festgelegt wird zu einer Topologie hinzu, indem zwei bestehende isolierte Knoten `anode` und `anothernode` verbunden werden. Gibt die "edgeid" der neuen Kante aus.

Wenn das Koordinatenreferenzsystem (SRID) der Geometrie `alinestring` nicht mit dem der Topologie übereinstimmt, irgendein Eingabewert NULL ist, die Knoten in mehreren Maschen enthalten sind, oder die Knoten Anfangs- oder Endknoten einer bestehenden Kante darstellen, wird eine Fehlermeldung ausgegeben.

Wenn `alinestring` nicht innerhalb der Masche liegt zu der `anode` und `anothernode` gehören, dann wird eine Fehlermeldung ausgegeben.

Wenn `anode` und `anothernode` nicht Anfangs- und Endpunkt von `alinestring` sind, wird eine Fehlermeldung ausgegeben.

Verfügbarkeit: 1.1

 Diese Methode setzt die SQL/MM-Spezifikation um. SQL-MM: Topo-Geo and Topo-Net 3: Routine Details: X.3.4

Beispiele

Siehe auch

ST_AddIsoNode, ST_IsSimple, ST_Within

9.8.3 ST_AddEdgeNewFaces

ST_AddEdgeNewFaces — Fügt eine Kante hinzu. Falls dabei eine Masche aufgetrennt wird, so wird die ursprüngliche Masche gelöscht und durch zwei neue Maschen ersetzt.

Synopsis

bigint **ST_AddEdgeNewFaces**(varchar atopology, bigint anode, bigint anothernode, geometry acurve);

Description

Fügt eine Kante hinzu. Falls dabei eine Masche aufgetrennt wird, so wird die ursprüngliche Masche gelöscht und durch zwei neue Maschen ersetzt. Gibt die ID der hinzugefügten Kante aus.

Führt ein entsprechendes Update auf alle verbundenen Kanten und Beziehungen durch.

Wenn irgendwelche Übergabewerte NULL sind, die angegebenen Knoten unbekannt sind (müssen bereits in der `node` Tabelle des Schemas "topology" existieren), `acurve` kein `LINESTRING` ist, oder `anode` und `anothernode` nicht die Anfangs- und Endpunkte von `acurve` sind, dann wird eine Fehlermeldung ausgegeben.

Wenn das Koordinatenreferenzsystem (SRID) der Geometrie `acurve` nicht mit jener der Topologie übereinstimmt, wird eine Fehlermeldung ausgegeben.

Verfügbarkeit: 2.0

Diese Methode setzt die SQL/MM-Spezifikation um. SQL-MM: Topo-Geo and Topo-Net 3: Routine Details: X.3.12

Beispiele

Siehe auch

ST_RemEdgeNewFace

ST_AddEdgeModFace

9.8.4 ST_AddEdgeModFace

ST_AddEdgeModFace — Fügt eine Kante hinzu. Falls dabei eine Masche aufgetrennt wird, so wird die ursprüngliche Masche angepasst und eine weitere Masche hinzugefügt.

Synopsis

bigint **ST_AddEdgeModFace**(varchar atopology, bigint anode, bigint anothernode, geometry acurve);

Description

Fügt eine Kante hinzu. Falls dabei eine Masche aufgetrennt wird, so wird die ursprüngliche Masche angepasst und eine weitere Masche hinzugefügt.



Anmerkung
Wenn möglich, wird die neue Masche auf der linken Seite der neuen Kante erstellt. Dies ist jedoch nicht möglich, wenn die Masche auf der linken Seite die (unbegrenzte) Grundmenge der Maschen darstellt.

Gibt die id der hinzugefügten Kante zurück.

Führt ein entsprechendes Update auf alle verbundenen Kanten und Beziehungen durch.

Wenn irgendwelche Übergabewerte NULL sind, die angegebenen Knoten unbekannt sind (müssen bereits in der `node` Tabelle des Schemas "topology" existieren), `acurve` kein `LINESTRING` ist, oder `anode` und `anothernode` nicht die Anfangs- und Endpunkte von `acurve` sind, dann wird eine Fehlermeldung ausgegeben.

Wenn das Koordinatenreferenzsystem (SRID) der Geometrie `acurve` nicht mit jener der Topologie übereinstimmt, wird eine Fehlermeldung ausgegeben.

Verfügbarkeit: 2.0

Diese Methode setzt die SQL/MM-Spezifikation um. SQL-MM: Topo-Geo and Topo-Net 3: Routine Details: X.3.13

Beispiele

Siehe auch

ST_RemEdgeModFace

ST_AddEdgeNewFaces

9.8.5 ST_RemEdgeNewFace

ST_RemEdgeNewFace — Entfernt eine Kante. Falls die gelöschte Kante zwei Maschen voneinander getrennt hat, werden die ursprünglichen Maschen gelöscht und durch einer neuen Masche ersetzt.

Synopsis

bigint **ST_RemEdgeNewFace**(varchar atopology, bigint anedge);

Description

Entfernt eine Kante. Falls die gelöschte Kante zwei Maschen voneinander getrennt hat, werden die ursprünglichen Maschen gelöscht und durch einer neuen Masche ersetzt.

Gibt die ID der neu erzeugten Masche aus; oder NULL wenn keine Masche erstellt wurde. Es wird keine neue Masche erstellt, wenn die gelöschte Kante defekt oder isoliert ist, oder wenn sie die Begrenzung der Maschengrundmenge darstellt (wodurch die Grundmenge womöglich von der anderen Seite in die Masche fliessen könnte).

Führt ein entsprechendes Update auf alle verbundenen Kanten und Beziehungen durch.

Wenn eine Kante an der Definition einer bestehenden TopoGeometry beteiligt ist, kann sie nicht gelöscht werden. Wenn irgendeine TopoGeometry nur durch eine der beiden Maschen definiert ist (und nicht auch durch die andere), können die beiden Maschen nicht "geheilt" werden.

Wenn irgendwelche Übergabewerte NULL sind, die angegebene Kante unbekannt ist (muss bereits in der `edge` Tabelle des Schemas "topology" existieren), oder die Bezeichnung der Topologie ungültig ist, dann wird eine Fehlermeldung ausgegeben.

Verfügbarkeit: 2.0

 Diese Methode setzt die SQL/MM-Spezifikation um. SQL-MM: Topo-Geo and Topo-Net 3: Routine Details: X.3.14

Beispiele

Siehe auch

ST_RemEdgeModFace

ST_AddEdgeNewFaces

9.8.6 ST_RemEdgeModFace

ST_RemEdgeModFace — Entfernt eine Kante, und wenn die Kante zwei Flächen trennt, wird eine Fläche gelöscht und die andere Fläche so verändert, dass sie den Raum beider Flächen abdeckt.

Synopsis

bigint **ST_RemEdgeModFace**(varchar atopology, bigint anedge);

Description

Entfernt eine Kante, und wenn die entfernte Kante zwei Flächen trennt, wird eine Fläche gelöscht und die andere Fläche so verändert, dass sie den Raum beider Flächen abdeckt. Bevorzugt wird die Fläche auf der rechten Seite beibehalten, um mit ST_AddEdgeModFace konsistent zu sein. Gibt die ID der Fläche zurück, die erhalten bleibt.

Führt ein entsprechendes Update auf alle verbundenen Kanten und Beziehungen durch.

Wenn eine Kante an der Definition einer bestehenden TopoGeometry beteiligt ist, kann sie nicht gelöscht werden. Wenn irgendeine TopoGeometry nur durch eine der beiden Maschen definiert ist (und nicht auch durch die andere), können die beiden Maschen nicht "geheilt" werden.

Wenn irgendwelche Übergabewerte NULL sind, die angegebene Kante unbekannt ist (muss bereits in der `edge` Tabelle des Schemas "topology" existieren), oder die Bezeichnung der Topologie ungültig ist, dann wird eine Fehlermeldung ausgegeben.

Verfügbarkeit: 2.0

 Diese Methode setzt die SQL/MM-Spezifikation um. SQL-MM: Topo-Geo and Topo-Net 3: Routine Details: X.3.15

Beispiele

Siehe auch

ST_AddEdgeModFace

ST_RemEdgeNewFace

9.8.7 ST_ChangeEdgeGeom

ST_ChangeEdgeGeom — Ändert die geometrische Form einer Kante, ohne sich auf die topologische Struktur auszuwirken.

Synopsis

text **ST_ChangeEdgeGeom**(varchar atopology, bigint anedge, geometry acurve);

Description

Ändert die geometrische Form der Kante, ohne sich auf topologische Struktur auszuwirken.

Wenn eines der Argumente null ist, die angegebene Kante nicht in der Tabelle `edge` des Topologie-Schemas existiert, die `acurve` keine `LINESTRING` ist oder die Änderung die zugrunde liegende Topologie verändern würde, wird ein Fehler ausgelöst.

Wenn das Koordinatenreferenzsystem (SRID) der Geometrie `acurve` nicht mit jener der Topologie übereinstimmt, wird eine Fehlermeldung ausgegeben.

Wenn die neue `acurve` nicht "simple" ist, wird eine Fehlermeldung ausgegeben.

Wenn beim Verschieben der Kante von der alten auf die neue Position ein Hindernis auftritt, wird eine Fehlermeldung ausgegeben.

Verfügbarkeit: 1.1.0

Erweiterung: 2.0.0 Erzwingung topologischer Konsistenz hinzugefügt

 Diese Methode setzt die SQL/MM-Spezifikation um. SQL-MM: Topo-Geo and Topo-Net 3: Routine Details X.3.6

Beispiele

Code

```
SELECT topology.ST_ChangeEdgeGeom(
    'ma_topo',
    1,
        ST_GeomFromText(
            'LINESTRING(227591.9 893900.4,227622.6 893844.3,227641.6 893816.6,227704.5 ↩
    893778.5)',
            26986)
);
```

Output

```
Edge 1 changed
```

Siehe auch

ST_AddEdgeModFace

ST_RemEdgeModFace

ST_ModEdgeSplit

9.8.8 ST_ModEdgeSplit

ST_ModEdgeSplit — Trennt eine Kante auf, indem ein neuer Knoten entlang einer bestehenden Kante erstellt wird. Ändert die ursprüngliche Kante und fügt eine neue Kante hinzu.

Synopsis

bigint **ST_ModEdgeSplit**(varchar atopology, bigint anedge, geometry apoint);

Description

Trennt eine Kante auf, indem ein neuer Knoten entlang einer bestehenden Kante erstellt wird. Ändert die ursprüngliche Kante und fügt eine neue Kante hinzu. Alle bestehenden Kanten und Beziehungen werden entsprechend aktualisiert. Gibt den Identifikator des neu hinzugefügten Knotens aus.

Verfügbarkeit: 1.1

Änderung: 2.0 - In Vorgängerversionen fälschlicherweise als ST_ModEdgesSplit bezeichnet

 Diese Methode setzt die SQL/MM-Spezifikation um. SQL-MM: Topo-Geo and Topo-Net 3: Routine Details: X.3.9

Beispiele

Add an edge to split.

Code

```
SELECT topology.AddEdge('ma_topo', ST_GeomFromText('LINESTRING(227592 893910,227600 893910) ↩
    ', 26986)) AS edgeid;
```

Output

```
3
```

Split the edge with a new node.

Code

```
SELECT topology.ST_ModEdgeSplit('ma_topo', 3, ST_SetSRID(ST_Point(227594, 893910), 26986)) ↩
    AS node_id;
```

Output

```
7
```

Siehe auch

ST_NewEdgesSplit, ST_ModEdgeHeal, ST_NewEdgeHeal, AddEdge

9.8.9 ST_ModEdgeHeal

ST_ModEdgeHeal — "Heilt" zwei Kanten, indem der verbindende Knoten gelöscht wird, die erste Kante modifiziert und die zweite Kante gelöscht wird. Gibt die ID des gelöschten Knoten zurück.

Synopsis

bigint **ST_ModEdgeHeal**(varchar atopology, bigint anedge, bigint anotheredge);

Description

"Heilt" zwei Kanten, indem der verbindende Knoten gelöscht wird, die erste Kante modifiziert und die zweite Kante gelöscht wird. Gibt die ID des gelöschten Knoten zurück. Aktualisiert alle verbundenen Kanten und Beziehungen dementsprechend.

Verfügbarkeit: 2.0

 Diese Methode setzt die SQL/MM-Spezifikation um. SQL-MM: Topo-Geo and Topo-Net 3: Routine Details: X.3.9

Siehe auch

ST_ModEdgeSplit ST_NewEdgesSplit

9.8.10 ST_NewEdgeHeal

ST_NewEdgeHeal — "Heilt" zwei Kanten, indem der verbindende Knoten und beide Kanten gelöscht werden. Die beiden Kanten werden durch eine Kante ersetzt, welche dieselbe Ausichtung wie die erste Kante hat.

Synopsis

bigint **ST_NewEdgeHeal**(varchar atopology, bigint anedge, bigint anotheredge);

Description

"Heilt" zwei Kanten, indem der verbindende Knoten und beide Kanten gelöscht werden. Die beiden Kanten werden durch eine Kante ersetzt, welche dieselbe Ausichtung wie die erste Kante hat. Gibt die ID der neuen Kante aus. Aktualisiert alle verbundenen Kanten und Beziehungen dementsprechend.

Verfügbarkeit: 2.0

 Diese Methode setzt die SQL/MM-Spezifikation um. SQL-MM: Topo-Geo and Topo-Net 3: Routine Details: X.3.9

Siehe auch

ST_ModEdgeHeal ST_ModEdgeSplit ST_NewEdgesSplit

9.8.11 ST_MoveIsoNode

ST_MoveIsoNode — Verschiebt einen isolierten Knoten in einer Topologie von einer Stelle an eine andere. Falls die neue Geometrie `apoint` bereits als Knoten existiert, wird eine Fehlermeldung ausgegeben. Gibt eine Beschreibung der Verschiebung aus.

Synopsis

text **ST_MoveIsoNode**(varchar atopology, bigint anode, geometry apoint);

Description

Verschiebt einen isolierten Knoten in einer Topologie von einer Stelle an eine andere. Falls die neue Geometrie `apoint` bereits als Knoten existiert, wird eine Fehlermeldung ausgegeben.

Wenn eines der Argumente null ist, der `apoint` kein Punkt ist, der vorhandene Knoten nicht isoliert ist (ein Start- oder Endpunkt einer vorhandenen Kante ist), die neue Knotenposition eine vorhandene Kante schneidet (sogar an den Endpunkten) oder die neue Position in einer anderen Fläche liegt (seit 3.2.0), wird eine Ausnahme ausgelöst.

Wenn das Koordinatenreferenzsystem (SRID) der Punktgeometrie nicht mit jener der Topologie übereinstimmt, wird eine Fehlermeldung ausgegeben.

Verfügbarkeit: 2.0.0

Verbessert: 3.2.0 stellt sicher, dass der Knoten nicht in ein anderes Gesicht verschoben werden kann

 Diese Methode setzt die SQL/MM-Spezifikation um. SQL-MM: Topo-Net Routines: X.3.2

Beispiele

Add an isolated node with no face.

Code

```
SELECT topology.ST_AddIsoNode('ma_topo', NULL, ST_GeomFromText('POINT(227579 893916)', ↩
    26986)) AS nodeid;
```

Output

```
7
```

Move the new node.

Code

```
SELECT topology.ST_MoveIsoNode('ma_topo', 7, ST_GeomFromText('POINT(227579.5 893916.5)', ↩
    26986)) AS descrip;
```

Output

```
Isolated Node 7 moved to location 227579.5,893916.5
```

Siehe auch

ST_AddIsoNode

9.8.12 ST_NewEdgesSplit

ST_NewEdgesSplit — Trennt eine Kante auf, indem ein neuer Knoten entlang einer bestehenden Kante erstellt, die ursprüngliche Kante gelöscht und durch zwei neue Kanten ersetzt wird. Gibt die ID des neu erstellten Knotens aus, der die neuen Kanten verbindet.

Synopsis

bigint **ST_NewEdgesSplit**(varchar atopology, bigint anedge, geometry apoint);

Description

Trennt eine Kante mit der Kanten-ID `anedge`auf, indem ein neuer Knoten mit der Punktlage `apoint` entlang der aktuellen Kante erstellt, die ursprüngliche Kante gelöscht und durch zwei neue Kanten ersetzt wird. Gibt die ID des neu erstellten Knotens aus, der die neuen Kanten verbindet. Aktualisiert alle verbundenen Kanten und Beziehungen dementsprechend.

Wenn das Koordinatenreferenzsystem (SRID) der Punktgeometrie nicht mit dem der Topologie übereinstimmt, `apoint` keine Punktgeometrie ist, der Punkt NULL ist, der Punkt bereits als Knoten existiert, die Kante mit einer bestehenden Kante nicht zusammenpasst, oder der Punkt nicht innerhalb der Kante liegt, dann wird eine Fehlermeldung ausgegeben.

Verfügbarkeit: 1.1

 Diese Methode setzt die SQL/MM-Spezifikation um. SQL-MM: Topo-Net Routines: X.3.8

Beispiele

Add an edge to split.

Code

```
SELECT topology.AddEdge('ma_topo', ST_GeomFromText('LINESTRING(227575 893917,227592 893900) ↩
    ', 26986)) AS edgeid;
```

Output

```
2
```

Split the new edge.

Code

```
SELECT topology.ST_NewEdgesSplit('ma_topo', 2, ST_GeomFromText('POINT(227578.5 893913.5)', ↩
    26986)) AS newnodeid;
```

Output

```
6
```

Siehe auch

ST_ModEdgeSplit ST_ModEdgeHeal ST_NewEdgeHeal AddEdge

9.8.13 ST_RemoveIsoNode

ST_RemoveIsoNode — Löscht einen isolierten Knoten und gibt eine Beschreibung der getroffenen Maßnahmen aus. Falls der Knoten nicht isoliert ist (ist der Anfangs- oder der Endpunkt einer Kante), wird eine Fehlermeldung ausgegeben.

Synopsis

text **ST_RemoveIsoNode**(varchar atopology, bigint anode);

Description

Löscht einen isolierten Knoten und gibt eine Beschreibung der getroffenen Maßnahmen aus. Falls der Knoten nicht isoliert ist (ist der Anfangs- oder der Endpunkt einer Kante), wird eine Fehlermeldung ausgegeben.

Verfügbarkeit: 1.1

 Diese Methode setzt die SQL/MM-Spezifikation um. SQL-MM: Topo-Geo and Topo-Net 3: Routine Details: X+1.3.3

Beispiele

Remove an isolated node with no face.

Code

```
SELECT topology.ST_RemoveIsoNode('ma_topo', 7) AS result;
```

Output

```
Isolated node 7 removed
```

Siehe auch

ST_AddIsoNode

9.8.14 ST_RemoveIsoEdge

ST_RemoveIsoEdge — Löscht einen isolierten Knoten und gibt eine Beschreibung der getroffenen Maßnahmen aus. Falls der Knoten nicht isoliert ist, wird eine Fehlermeldung ausgegeben.

Synopsis

text **ST_RemoveIsoEdge**(varchar atopology, bigint anedge);

Description

Löscht einen isolierten Knoten und gibt eine Beschreibung der getroffenen Maßnahmen aus. Falls der Knoten nicht isoliert ist, wird eine Fehlermeldung ausgegeben.

Verfügbarkeit: 1.1

 Diese Methode setzt die SQL/MM-Spezifikation um. SQL-MM: Topo-Geo and Topo-Net 3: Routine Details: X+1.3.3

Beispiele

Remove an isolated edge with no face.

Code

```
SELECT topology.ST_RemoveIsoEdge('ma_topo', 7) AS result;
```

Output

```
Isolated edge 7 removed
```

Siehe auch

ST_AddIsoNode

9.9 Zugriffsfunktionen zur Topologie

9.9.1 GetEdgeByPoint

GetEdgeByPoint — Findet die edge-id einer Kante die einen gegebenen Punkt schneidet.

Synopsis

bigint **GetEdgeByPoint**(varchar atopology, geometry apoint, float8 tol1);

Description

Erfasst die ID einer Kante, die einen Punkt schneidet

Die Funktion gibt eine Ganzzahl (id-edge) für eine Topologie, einen POINT und eine Toleranz aus. Wenn tolerance = 0, dann muss der Punkt die Kante schneiden.

Wenn `apoint` keine Kante schneidet, wird 0 (Null) zurückgegeben.

Wenn use tolerance > 0 und es gibt mehr als eine Kante in der Nähe des Punktes dann eine Ausnahme ausgelöst wird.



Anmerkung
Wenn tolerance = 0, wird von der Funktion ST_Intersects angewendet, ansonsten ST_DWithin.

Wird vom GEOS Modul ausgeführt

Verfügbarkeit: 2.0.0

Beispiele

Die folgenden Beispiele benutzen die Kanten, die wir in AddEdge erzeugt haben

Code

```
SELECT topology.GetEdgeByPoint('ma_topo', geom, 1) As with1mtol, topology.GetEdgeByPoint(' ↩
    ma_topo', geom, 0) As withnotol
FROM ST_GeomFromEWKT('SRID=26986;POINT(227622.6 893843)') As geom;
```

Output

```
with1mtol | withnotol
-----------+-----------
         2 |          0
```

This query errors because two or more edges are found.

Code

```
SELECT topology.GetEdgeByPoint('ma_topo', geom, 1) As nearnode
FROM ST_GeomFromEWKT('SRID=26986;POINT(227591.9 893900.4)') As geom;
```

Output

```
ERROR:  Two or more edges found
```

Siehe auch

AddEdge, GetNodeByPoint, GetFaceByPoint

9.9.2 GetFaceByPoint

GetFaceByPoint — Findet eine Fläche, die einen bestimmten Punkt schneidet.

Synopsis

bigint **GetFaceByPoint**(varchar atopology, geometry apoint, float8 tol1);

Description

Findet eine Fläche, die von einem Punkt referenziert wird, mit der angegebenen Toleranz.

Die Funktion sucht tatsächlich nach einer Fläche, die einen Kreis mit dem Punkt als Mittelpunkt und der Toleranz als Radius schneidet.

Wenn keine Fläche die angegebene Abfrageposition schneidet, wird 0 zurückgegeben (Universalfläche).

Wenn mehr als eine Fläche den Abfrageort schneidet, wird eine Ausnahme ausgelöst.

Verfügbarkeit: 2.0.0

Verbessert: 3.2.0 effizientere Implementierung und klarerer Vertrag, funktioniert nicht mehr mit ungültigen Topologien.

Beispiele

Code

```
SELECT topology.GetFaceByPoint('ma_topo', geom, 10) As with1mtol, topology.GetFaceByPoint(' ↩
    ma_topo', geom, 0) As withnotol
FROM ST_GeomFromEWKT('POINT(234604.6 899382.0)') As geom;
```

Output

```
with1mtol | withnotol
-----------+-----------
         1 |         0
```

This query errors because two or more faces are found.

Code

```
SELECT topology.GetFaceByPoint('ma_topo', geom, 1) As nearnode
FROM ST_GeomFromEWKT('POINT(227591.9 893900.4)') As geom;
```

Output

```
ERROR:  Two or more faces found
```

Siehe auch

GetFaceContainingPoint, AddFace, GetNodeByPoint, GetEdgeByPoint

9.9.3 GetFaceContainingPoint

GetFaceContainingPoint — Findet die Fläche, die einen Punkt enthält.

Synopsis

bigint **GetFaceContainingPoint**(text atopology, geometry apoint);

Description

Gibt die ID der Fläche zurück, die einen Punkt enthält.

Eine Ausnahme wird ausgelöst, wenn der Punkt auf eine Flächenbegrenzung fällt.



Anmerkung
Die Funktion setzt eine gültige Topologie voraus, die Kantenverknüpfungen und Flächenbeschriftungen verwendet.

Verfügbarkeit: 3.2.0

Siehe auch

ST_GetFaceGeometry

9.9.4 GetNodeByPoint

GetNodeByPoint — Findet zu der Lage eines Punktes die node-id eines Knotens.

Synopsis

bigint **GetNodeByPoint**(varchar atopology, geometry apoint, float8 tol1);

Description

Erfasst zu der Lage eines Punktes die ID eines Knotens.

Diese Funktion gibt für eine Topologie, einen POINT und eine Toleranz eine Ganzzahl (id-node) aus. Tolerance = 0 bedeutet exakte Überschneidung, ansonsten wird der Knoten in einem bestimmten Abstand gesucht.

Wenn `apoint` keinen Knoten schneidet, wird 0 (Null) zurückgegeben.

Wenn use tolerance > 0 und es gibt mehr als einen Knoten in der Nähe des Punktes dann eine Ausnahme geworfen wird.



Anmerkung
Wenn tolerance = 0, wird von der Funktion ST_Intersects angewendet, ansonsten ST_DWithin.

Wird vom GEOS Modul ausgeführt

Verfügbarkeit: 2.0.0

Beispiele

Die folgenden Beispiele benutzen die Kanten, die wir in AddEdge erzeugt haben

Code

```
SELECT topology.GetNodeByPoint('ma_topo', geom, 1) As nearnode
FROM ST_GeomFromEWKT('SRID=26986;POINT(227591.9 893900.4)') As geom;
```

Output

```
2
```

This query errors because the tolerance matches two or more nodes.

Code

```
SELECT topology.GetNodeByPoint('ma_topo', geom, 1000) As too_much_tolerance
FROM ST_GeomFromEWKT('SRID=26986;POINT(227591.9 893900.4)') As geom;
```

Output

```
ERROR:  Two or more nodes found
```

Siehe auch

AddEdge, GetEdgeByPoint, GetFaceByPoint

9.9.5 GetTopologyID

GetTopologyID — Gibt für den Namen einer Topologie die ID der Topologie in der Tabelle "topology.topology" aus.

Synopsis

integer **GetTopologyID**(varchar toponame);

Description

Gibt für den Namen einer Topologie, die ID der Topologie in der Tabelle "topology.topology" aus.

Verfügbarkeit: 1.1

Beispiele

Code

```
SELECT topology.GetTopologyID('ma_topo') AS topo_id;
```

Output

```
1
```

Siehe auch

CreateTopology, DropTopology, GetTopologyName, GetTopologySRID

9.9.6 GetTopologySRID

GetTopologySRID — Gibt für den Namen einer Topologie, die SRID der Topologie in der Tabelle "topology.topology" aus.

Synopsis

integer **GetTopologyID**(varchar toponame);

Description

Gibt für den Namen einer Topologie, die ID des Koordinatenreferenzsystems in der Tabelle "topology.topology" aus.

Verfügbarkeit: 2.0.0

Beispiele

Code

```
SELECT topology.GetTopologySRID('ma_topo') AS srid;
```

Output

```
4326
```

Siehe auch

CreateTopology, DropTopology, GetTopologyName, GetTopologyID

9.9.7 GetTopologyName

GetTopologyName — Gibt für die ID der Topologie, den Namen der Topologie (Schema) zurück.

Synopsis

varchar **GetTopologyName**(integer topology_id);

Description

Gibt für die ID einer Topologie, den Namen (Schema) der Topologie in der Tabelle "topology.topology" aus.

Verfügbarkeit: 1.1

Beispiele

Code

```
SELECT topology.GetTopologyName(1) AS topo_name;
```

Output

```
ma_topo
```

Siehe auch

CreateTopology, DropTopology, GetTopologyID, GetTopologySRID

9.9.8 ST_GetFaceEdges

ST_GetFaceEdges — Gibt die Kanten, die `aface` begrenzen, sortiert aus.

Synopsis

getfaceedges_returntype **ST_GetFaceEdges**(varchar atopology, bigint aface);

Description

Gibt die Kanten, die `aface` begrenzen, sortiert aus. Jede Ausgabe besteht aus einer Sequenz und einer "edgeid". Die Sequenzzahlen beginnen mit dem Wert 1.

Die Aufzählung der Kanten des Rings beginnt mit der Kante mit dem niedrigsten Identifikator. Die Reihenfolge der Kanten folgt der Drei-Finger-Regel (die begrenzte Masche liegt links von den gerichteten Kanten).

Verfügbarkeit: 2.0

 Diese Methode setzt die SQL/MM-Spezifikation um. SQL-MM 3 Topo-Geo and Topo-Net 3: Routine Details: X.3.5

Beispiele

This query returns the edges that bound face 1.

Code

```
SELECT (topology.ST_GetFaceEdges('tt', 1)).*;
```

Output

```
sequence | edge
----------+------
         1 |   -4
         2 |    5
         3 |    7
         4 |   -6
         5 |    1
         6 |    2
         7 |    3
```

This query returns the sequence, edge identifier, and geometry of the edges that bound face 1. Use ST_GetFaceGeometry if only the geometry and sequence are needed.

Code

```
SELECT t.seq, t.edge, geom
FROM topology.ST_GetFaceEdges('tt', 1) As t(seq, edge)
    INNER JOIN tt.edge AS e ON abs(t.edge) = e.edge_id;
```

Siehe auch

GetRingEdges, AddFace, ST_GetFaceGeometry

9.9.9 ST_GetFaceGeometry

ST_GetFaceGeometry — Gibt für eine Topologie und eine bestimmte Maschen-ID das Polygon zurück.

Synopsis

geometry **ST_GetFaceGeometry**(varchar atopology, bigint aface);

Description

Gibt für eine Topologie und eine bestimmte Maschen-ID das Polygon zurück. Erstellt das Polygon aus den Kanten, die die Masche aufbauen.

Verfügbarkeit: 1.1


Diese Methode setzt die SQL/MM-Spezifikation um. SQL-MM 3 Topo-Geo and Topo-Net 3: Routine Details: X.3.16

Beispiele

This query returns the WKT of the polygon added with AddFace.

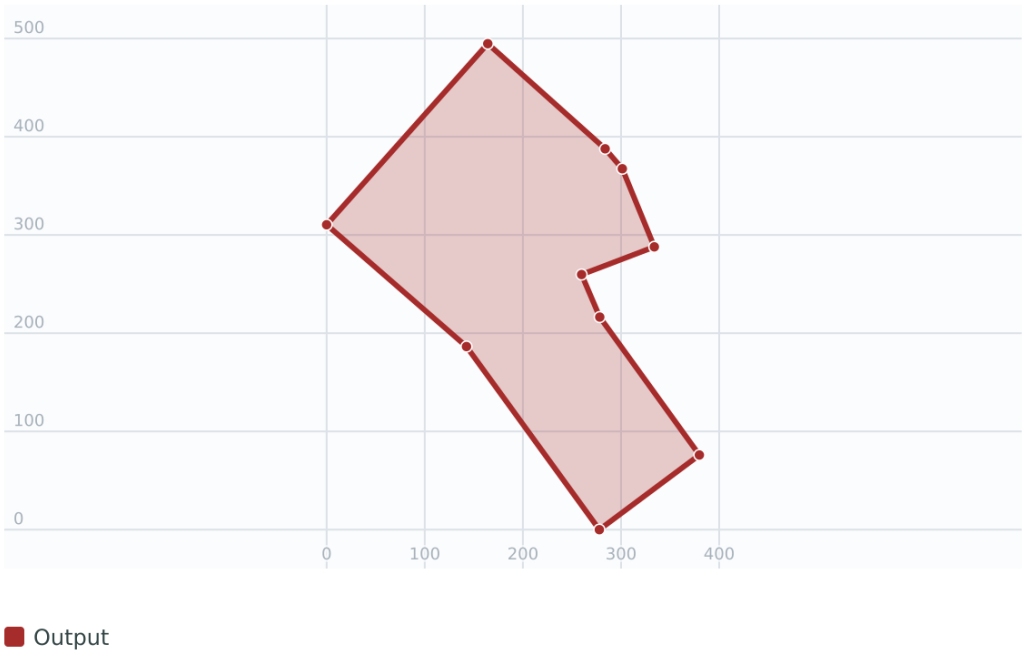
Code

```
SELECT topology.ST_GetFaceGeometry('ma_topo', 1) AS facegeomwkt;
```

Output

```
POLYGON((234776.9 899563.7,234896.5 899456.7,234914 899436.4,234946.6 899356.9,
234872.5 899328.7,234891 899285.4,234992.5 899145,234890.6 899069,
234755.2 899255.4,234612.7 899379.4,234776.9 899563.7))
```

Figure



Siehe auch

AddFace

9.9.10 GetRingEdges

GetRingEdges — Gibt eine sortierte Liste von mit Vorzeichen versehenen Identifikatoren der Kanten zurück, die angetroffen werden, wenn man an der Seite der gegebenen Kante entlangwandert.

Synopsis

getfaceedges_returntype **GetRingEdges**(varchar atopology, bigint aring, integer max_edges=null);

Description

Gibt eine sortierte Liste von mit Vorzeichen versehenen Identifikatoren der Kanten zurück, die angetroffen werden, wenn man an der Seite der gegebenen Kante entlangwandert. Jede Ausgabe besteht aus einer Sequenz und einer mit einem Vorzeichen versehenen ID der Kante. Die Sequenz beginnt mit dem Wert 1.

Wenn Sie eine positive ID für die Kante übergeben, beginnt der Weg auf der linken Seite der entsprechenden Kante und folgt der Ausrichtung der Kante. Wenn Sie eine negative ID für die Kante übergeben, beginnt der Weg auf der rechten Seite der Kante und verläuft rückwärts.

Wenn `max_edges` nicht NULL ist, so beschränkt dieser Parameter die Anzahl der von dieser Funktion ausgegebenen Datensätze. Ist als Sicherheitsparameter für den Umgang mit möglicherweise invaliden Topologien gedacht.



Anmerkung
Diese Funktion verwendet Metadaten um die Kanten eines Ringes zu verbinden.

Verfügbarkeit: 2.0.0

Siehe auch

ST_GetFaceEdges, GetNodeEdges getfaceedges_returntype

9.9.11 GetNodeEdges

GetNodeEdges — Gibt für einen Knoten die sortierte Menge der einfallenden Kanten aus.

Synopsis

getfaceedges_returntype **GetNodeEdges**(varchar atopology, bigint anode);

Description

Gibt für einen Knoten die sortierte Menge der einfallenden Kanten aus. Jede Ausgabe besteht aus einer Sequenz und einer mit Vorzeichen versehenen ID für die Kante. Die Sequenz beginnt mit dem Wert 1. Eine Kante mit positiver ID beginnt an dem gegebenen Knoten. Eine negative Kante endet in dem gegebenen Knoten. Geschlossene Kanten kommen zweimal vor (mit beiden Vorzeichen). Die Sortierung geschieht von Norden ausgehend im Uhrzeigersinn.



Anmerkung
Diese Funktion errechnet die Reihenfolge, anstatt sie aus den Metadaten abzuleiten und kann daher verwendet werden, um die Kanten eines Ringes zu verbinden.

Verfügbarkeit: 2.0

Siehe auch

getfaceedges_returntype, GetRingEdges, ST_Azimuth

9.9.12 FindVertexSegmentPairsBelowDistance

FindVertexSegmentPairsBelowDistance — Find pairs of topology vertex/segment that are closer than tolerated distance

Synopsis

table(seg_edge bigint, segno int, vert_edge bigint, vertno int, vert_geom geometry) **FindVertexSegmentPairsBelowDistance**(varchar atopology, float8 tolerance);

Description

Returns a relation in which each tuple is a pair of vertex/segment where the vertex is NOT an endpoint of the segment but the distance between them is below the given tolerance.

Availability: 3.7

Siehe auch

ValidateTopologyPrecision

9.10 Topologie Verarbeitung

9.10.1 Polygonize

Polygonize — Findet und registriert alle Maschen, die durch die Kanten der Topologie festgelegt sind.

Synopsis

text **Polygonize**(varchar toponame);

Description

Registriert alle Maschen, die aus den Kanten der topologischen Elementarstrukturen erstellt werden können

Von der Zieltopologie wird angenommen, dass sie keine sich selbst überschneidenden Kanten enthält.



Anmerkung
Da bereits bekannte Maschen erkannt werden, kann Polygonize gefahrlos mehrere Male auf die selbe Topologie angewendet werden.



Anmerkung
Von dieser Funktion werden die Attribute "next_left_edge" und "next_right_edge" der Tabelle "edge" weder verwendet noch gesetzt.

Verfügbarkeit: 2.0.0

Siehe auch

AddFace, ST_Polygonize

9.10.2 AddNode

AddNode — Fügt einen Knotenpunkt zu der Tabelle "node" in dem vorgegebenen topologischen Schema hinzu und gibt die "nodeid" des neuen Knotens aus. Falls der Punkt bereits als Knoten existiert, wird die vorhandene nodeid zurückgegben.

Synopsis

bigint **AddNode**(varchar toponame, geometry apoint, boolean allowEdgeSplitting=false, boolean computeContainingFace=false);

Description

Fügt einen Knotenpunkt zu der Tabelle "node" in dem vorgegebenen topologischen Schema hinzu. Die Funktion AddEdge fügt die Anfangs- und Endpunkte einer Kante automatisch hinzu, wenn sie aufgerufen wird. Deshalb ist es nicht notwendig die Knoten einer Kante explizit anzufügen.

Falls eine Kante aufgefunden wird, die den Knoten kreuzt, dann wird entweder eine Fehlermeldung ausgegeben, oder die Kante aufgetrennt. Dieses Verhalten hängt vom Wert des Parameters `allowEdgeSplitting` ab.

Wenn `computeContainingFace` TRUE ist, dann wird für einen neu hinzugefügten Knoten die berichtigte Begrenzung der Masche berechnet.



Anmerkung
Wenn die Geometrie `apoint` bereits als Knoten existiert, dann wird der Knoten nicht hinzugefügt und der bestehende Knoten ausgegeben.

Verfügbarkeit: 2.0.0

Beispiele

Code

```
SELECT topology.AddNode('ma_topo', ST_GeomFromText('POINT(227641.6 893816.5)', 26986)) AS ↩
    nodeid;
```

Output

```
4
```

Siehe auch

AddEdge, CreateTopology

9.10.3 AddEdge

AddEdge — Fügt die Kante eines Linienzugs in der Tabelle "edge", und die zugehörigen Anfangs- und Endpunkte in die Knotenpunkttabelle, des jeweiligen topologischen Schemas ein. Dabei wird die übergebene Linienzuggeometrie verwendet und die edgeid der neuen (oder bestehenden) Kante ausgegeben.

Synopsis

bigint **AddEdge**(varchar toponame, geometry aline);

Description

Fügt eine Kante in der Tabelle "edge", und die zugehörigen Knoten in die Tabelle "node", des jeweiligen Schemas `toponame` ein. Dabei wird die übergebene Linienzuggeometrie verwendet und die edgeid des neuen oder des bestehenden Datensatzes ausgegeben. Die neu hinzugefügte Kante hat auf beiden Seiten die Masche für die Grundmenge/"Universum" und verweist auf sich selbst.



Anmerkung
Wenn die Geometrie `aline` eine bestehende Kante kreuzt, überlagert, beinhaltet oder in ihr enthalten ist, dann wird eine Fehlermeldung ausgegeben und die Kante wird nicht hinzugefügt.



Anmerkung
Die Geometrie von `aline` muss dieselbe `SRID` aufweisen wie die Topologie, ansonsten wird die Fehlermeldung "invalid spatial reference sys error" ausgegeben.

Wird vom GEOS Modul ausgeführt



Warnung
AddEdge Veraltet seit 3.5.0. Bitte verwenden Sie TopoGeo_AddLineString stattdessen.

Verfügbarkeit: 2.0.0

Beispiele

Code

```
SELECT topology.AddEdge('ma_topo', ST_GeomFromText('LINESTRING(227575.8 893917.2,227591.9 ↩
    893900.4)', 26986)) AS edgeid;
```

Output

```
1
```

Code

```
SELECT topology.AddEdge('ma_topo', ST_GeomFromText('LINESTRING(227591.9 893900.4,227622.6 ↩
    893844.2,227641.6 893816.5,227704.5 893778.5)', 26986)) AS edgeid;
```

Output

```
2
```

This query errors because the edge intersects an existing edge away from the endpoints.

Code

```
SELECT topology.AddEdge('ma_topo', ST_GeomFromText('LINESTRING(227591.2 893900,227591.9 ↩
    893900.4,227704.5 893778.5)', 26986)) AS edgeid;
```

Output

```
ERROR:  Edge intersects (not on endpoints) with existing edge 1
```

Siehe auch

TopoGeo_AddLineString, CreateTopology, Abschnitt 4.5

9.10.4 AddFace

AddFace — Registriert die Elementarstruktur einer Masche in einer Topologie und gibt den Identifikator der Masche aus.

Synopsis

bigint **AddFace**(varchar toponame, geometry apolygon, boolean force_new=false);

Description

Registriert die Elementarstruktur einer Masche in einer Topologie und gibt den Identifikator der Masche aus.

Bei einer neu hinzugefügten Masche werden die Kanten die ihre Begrenzung bilden und jene die innerhalb der Masche liegen aktualisiert, damit diese die richtigen Werte in den Attributen "left_face" und "right_face" aufweisen. Isolierte Knoten innerhalb der Masche werden ebenfalls aktualisiert, damit das Attribut "containing_face" die richtigen Werte aufweist.



Anmerkung
Von dieser Funktion werden die Attribute "next_left_edge" und "next_right_edge" der Tabelle "edge" weder verwendet noch gesetzt.

Es wird angenommen, dass die Zieltopologie valide ist (keine sich selbst überschneidenden Kanten enthält). Eine Fehlermeldung wird ausgegeben, wenn: Die Begrenzung des Polygons nicht vollständig durch bestehende Kanten festgelegt ist oder das Polygon eine bereits bestehende Masche überlappt.

Falls die Geometrie `apolygon` bereits als Masche existiert, dann: wenn `force_new` FALSE (der Standardwert) ist, dann wird die bestehende Masche zurückgegeben; wenn `force_new` TRUE ist, dann wird der neu registrierten Masche eine neue ID zugewiesen.



Anmerkung
Wenn eine bestehende Masche neu registriert wird (force_new=true), werden keine Maßnahmen durchgeführt um hängende Verweise auf eine bestehende Masche in den Tabellen "edge", "node" und "relation" zu bereinigen, noch wird der das Attribut MBR der bestehenden Masche aktualisert. Es ist die Aufgabe des Aufrufers sich um dies zu kümmern.



Anmerkung
Die Geometrie von `apolygon` muss dieselbe `SRID` aufweisen wie für die Topologie festgelegt, ansonsten wird die Fehlermeldung "invalid spatial reference sys error" ausgegeben.

Verfügbarkeit: 2.0.0

Beispiele

First add the edges using `generate_series` as an iterator. This query only works for polygons with fewer than 10000 points because of the maximum value in `generate_series`.

Code

```
SELECT topology.AddEdge('ma_topo', ST_MakeLine(ST_PointN(geom,i), ST_PointN(geom, i + 1) ))  ↩
    As edgeid
FROM (
  SELECT ST_NPoints(geom) AS npt, geom
```

```
  FROM (
    SELECT ST_Boundary(ST_GeomFromText('POLYGON((234896.5 899456.7,234914 899436.4,234946.6 ↩
    899356.9,234872.5 899328.7,
                234891 899285.4,234992.5 899145, 234890.6 899069,234755.2 899255.4,
                234612.7 899379.4,234776.9 899563.7,234896.5 899456.7))', 26986) )  As geom
  ) AS geoms
) AS facen
CROSS JOIN generate_series(1,10000) AS i
WHERE i < npt;
```

Output

```
3
4
5
6
7
8
9
10
11
12
```

Then add the face.

Code

```
SELECT topology.AddFace(
    'ma_topo',
    ST_GeomFromText(
    'POLYGON((234896.5 899456.7,234914 899436.4,234946.6 899356.9,234872.5 899328.7,
    234891 899285.4,234992.5 899145, 234890.6 899069,234755.2 899255.4,
    234612.7 899379.4,234776.9 899563.7,234896.5 899456.7))',
    26986) ) As faceid;
```

Output

```
1
```

Siehe auch

AddEdge, CreateTopology, Abschnitt 4.5

9.10.5 ST_Simplify

ST_Simplify — Gibt für eine TopoGeometry eine "vereinfachte" geometrische Version zurück. Verwendet den Douglas-Peucker Algorithmus.

Synopsis

geometry **ST_Simplify**(topogeometry tg, float8 tolerance);

Description

Gibt für eine TopoGeometry eine "vereinfachte" geometrische Version zurück. Wendet den Douglas-Peucker Algorithmus auf jede Kante an.



Anmerkung
Es kann vorkommen, dass die zurückgegebene Geometrie weder "simple" noch valide ist.
Das Auftrennen der Kanten kann helfen, die Simplizität/Validität zu erhalten.

Wird vom GEOS Modul ausgeführt

Verfügbarkeit: 2.1.0

Siehe auch

Geometrie ST_Simplify, ST_IsSimple, ST_IsValid, ST_ModEdgeSplit

9.10.6 RemoveUnusedPrimitives

RemoveUnusedPrimitives — Entfernt Topologieprimitive, die zur Definition bestehender TopoGeometry-Objekte nicht benötigt werden.

Synopsis

bigint **RemoveUnusedPrimitives**(text topology_name, geometry bbox);

Description

Findet alle Primitive (Knoten, Kanten, Flächen), die nicht unbedingt zur Darstellung vorhandener TopoGeometry-Objekte benötigt werden, und entfernt sie, wobei die Gültigkeit der Topologie (Kantenverknüpfung, Flächenbeschriftung) und die Belegung des TopoGeometry-Raums erhalten bleiben.

Es werden keine neuen Primitivbezeichner erstellt, sondern die vorhandenen Primitive werden erweitert, um verschmolzene Flächen (beim Entfernen von Kanten) oder geheilte Kanten (beim Entfernen von Knoten) aufzunehmen.

Verfügbarkeit: 3.3.0

Siehe auch

ST_ModEdgeHeal, ST_RemEdgeModFace

9.11 TopoGeometry Konstruktoren

9.11.1 CreateTopoGeom

CreateTopoGeom — Erzeugt ein neues topologisch geometrisches Objekt aus einem Feld mit topologischen Elementen - tg_type: 1:[multi]point, 2:[multi]line, 3:[multi]poly, 4:collection

Synopsis

topogeometry **CreateTopoGeom**(varchar toponame, integer tg_type, integer layer_id, topoelementarray tg_objs, bigint tg_id);
topogeometry **CreateTopoGeom**(varchar toponame, integer tg_type, integer layer_id);

Description

Erstellt ein TopoGeometry Objekt für den Layer, der über die `layer_id` angegeben wird, und registriert es in der Tabelle "relation" in dem Schema `toponame`.

`tg_type` ist eine Ganzzahl: 1:[multi]point (punktförmig), 2:[multi]line (geradlinig), 3:[multi]poly (flächenhaft), 4:collection. `layer_id` ist der Identifikator des Layers in der Tabelle "topology.layer".

Punktförmige Layer werden aus Knoten gebildet, linienförmige Layer aus Kanten, flächige Layer aus Maschen und Kollektionen können aus einer Mischung von Knoten, Kanten und Maschen gebildet werden.

Wird das Feld mit den Komponenten weggelassen, wird ein leeres TopoGeometrie Objekt erstellt.

Verfügbarkeit: 1.1

Beispiele

Form from existing edges.

Erstellt eine TopoGeometry im Schema "ri_topo" für den Layer 2 (ri_roads), vom Datentyp (2) LINE, für die erste Kante (die wir unter `ST_CreateTopoGeo` geladen haben).

Code

```
INSERT INTO ri.ri_roads(road_name, topo)
VALUES (
    'Unknown',
    topology.CreateTopoGeom(
        'ri_topo',
        2,
        2,
        '{{1,2}}'::topology.topoelementarray
    )
);
```

Convert an areal geometry to best guess topogeometry.

Angenommen wir wollen eine Geometrie aus einer Kollektion von Maschen bilden. Wir haben zum Beispiel die Tabelle "blockgroups" und wollen die TopoGeometry von jeder "blockgroup" wissen. Falls unsere Daten perfekt ausgerichtet sind, können wir folgendes ausführen:

Create the topogeometry column.

Code

```
SELECT topology.AddTopoGeometryColumn(
'topo_boston',
'boston', 'blockgroups', 'topo', 'POLYGON');
```

Output

```
1
```

Update the column assuming everything is perfectly aligned with the edges.

Code

```
UPDATE boston.blockgroups AS bg
SET topo = topology.CreateTopoGeom(
    'topo_boston',
    3,
    1,
```

```
    foo.bfaces
)
FROM (
    SELECT
        b.gid,
        topology.TopoElementArray_Agg(ARRAY[f.face_id, 3]) AS bfaces
    FROM boston.blockgroups AS b
    INNER JOIN topo_boston.face AS f
        ON b.geom && f.mbr
    WHERE ST_Covers(
        b.geom,
        topology.ST_GetFaceGeometry('topo_boston', f.face_id)
    )
    GROUP BY b.gid
) AS foo
WHERE foo.gid = bg.gid;
```

The world is rarely perfect, so this version allows some error. It counts a face if 50 percent of it falls within the expected block group boundary.

Code

```
WITH candidate_faces AS (
    SELECT
        b.gid,
        b.geom AS block_geom,
        f.face_id,
        topology.ST_GetFaceGeometry('topo_boston', f.face_id) AS face_geom
    FROM boston.blockgroups AS b
    INNER JOIN topo_boston.face AS f
        ON b.geom && f.mbr
),
block_faces AS (
    SELECT
        gid,
        topology.TopoElementArray_Agg(ARRAY[face_id, 3]) AS bfaces
    FROM candidate_faces
    WHERE ST_Covers(block_geom, face_geom)
        OR (
            ST_Intersects(block_geom, face_geom)
            AND ST_Area(ST_Intersection(block_geom, face_geom))

> ST_Area(face_geom) * 0.5
        )
    GROUP BY gid
)
UPDATE boston.blockgroups AS bg
SET topo = topology.CreateTopoGeom(
    'topo_boston',
    3,
    1,
    block_faces.bfaces
)
FROM block_faces
WHERE block_faces.gid = bg.gid;
```

To convert the topogeometry back to a denormalized geometry aligned with faces and edges, cast the topogeometry to a geometry. The resulting geometries are aligned with the TIGER street centerlines.

Code

```
UPDATE boston.blockgroups SET new_geom = topo::geometry;
```

Siehe auch

AddTopoGeometryColumn, toTopoGeom ST_CreateTopoGeo, ST_GetFaceGeometry, TopoElementArray, TopoElementArray_Agg

9.11.2 toTopoGeom

toTopoGeom — Wandelt eine einfache Geometrie in eine TopoGeometry um.

Synopsis

topogeometry **toTopoGeom**(geometry geom, varchar toponame, integer layer_id, float8 tolerance=-1);
topogeometry **toTopoGeom**(geometry geom, topogeometry topogeom, float8 tolerance=-1);

Description

Wandelt eine einfache Geometrie in eine TopoGeometry um.

Die topologischen Elementarstrukturen, die benötigt werden um die Übergabegeometrie darzustellen, werden der zugrunde liegenden Topologie hinzugefügt. Dabei können bestehende Strukturen aufgetrennt werden, die dann mit der ausgegebenen TopoGeometry in der Tabelle `relation` zusammengeführt werden.

Bestehende Objekte einer TopoGeometry (mit der möglichen Ausnahme von `topogeom`, falls angegeben) behalten ihre geometrische Gestalt.

When `tolerance` is given it will be used to snap the input geometry to existing primitives. The special value -1 (default) means use the topology precision as tolerance.

Bei der ersten Form wird eine neue TopoGeometry für den Layer (`layer_id`) einer Topologie (`toponame`) erstellt.

Bei der zweiten Form werden die aus der Konvertierung enstehenden Elementarstrukturen zu der bestehenden TopoGeometry (`topogeom`) hinzugefügt. Dabei wird möglicherweise zusätzlicher Raum aufgefüllt, um die endgültige geometrische Gestalt zu erreichen. Um die alte geometrische Gestalt zur Gänze durch eine neue zu ersetzen, siehe clearTopoGeom.

Verfügbarkeit: 2.0

Erweiterung: 2.1.0 die Version, welche eine bestehende TopoGeometry entgegennimmt, wurde hinzugefügt.

Changed: 3.7.0 changes meaning of tolerance 0 to really mean 0 and -1 (new default) to request using the topology precision.

Beispiele

Dies ist ein in sich selbst vollkommen abgeschlossener Arbeitsablauf

Create the topology without allowing any tolerance.

Code

```
SELECT topology.CreateTopology('topo_boston_test', 2249);
```

Create a new table.

Code

```
CREATE TABLE nei_topo(gid serial primary key, nei varchar(30));
```

Add a topogeometry column to the new table.

Code

```
SELECT topology.AddTopoGeometryColumn('topo_boston_test', 'public', 'nei_topo', 'topo', ' ↩
    MULTIPOLYGON') AS new_layer_id;
```

Output

```
1
```

Use the new layer identifier to populate the topogeometry column.

Code

```
INSERT INTO nei_topo(nei, topo)
SELECT nei, topology.toTopoGeom(geom, 'topo_boston_test', 1)
FROM neighborhoods
WHERE gid BETWEEN 1 and 15;
```

Verify what happened.

Code

```
SELECT * FROM
    topology.TopologySummary('topo_boston_test');
```

Output

```
Topology topo_boston_test (5), SRID 2249, precision 0
61 nodes, 87 edges, 35 faces, 15 topogeoms in 1 layers
Layer 1, type Polygonal (3), 15 topogeoms
 Deploy: public.nei_topo.topo
```

Shrink all TopoGeometry polygons by 10 meters.

Code

```
UPDATE nei_topo SET topo = toTopoGeom(ST_Buffer(topo, -10), clearTopoGeom(topo), 0);
```

Find faces that are left without any TopoGeometry reference after the shrink operation.

Code

```
SELECT ST_GetFaceGeometry('topo_boston_test', f.face_id)
  FROM topo_boston_test.face f
  WHERE f.face_id
> 0
  AND NOT EXISTS (
    SELECT * FROM topo_boston_test.relation
    WHERE layer_id = 1 AND element_id = f.face_id
  );
```

Siehe auch

CreateTopology, AddTopoGeometryColumn, CreateTopoGeom, TopologySummary, clearTopoGeom

9.11.3 TopoElementArray_Agg

TopoElementArray_Agg — Gibt für eine Menge an element_id, type Feldern (topoelements) ein `topoelementarray` zurück.

Synopsis

topoelementarray **TopoElementArray_Agg**(topoelement set tefield);

Description

Verwendet um ein TopoElementArray aus einer Menge an TopoElement zu erstellen.

Verfügbarkeit: 2.0.0

Beispiele

Code

```
SELECT topology.TopoElementArray_Agg(ARRAY[e, t] ORDER BY e, t) As tea
FROM generate_series(1, 3) As e CROSS JOIN generate_series(1, 4) As t;
```

Output

```
{{1,1},{1,2},{1,3},{1,4},{2,1},{2,2},{2,3},{2,4},{3,1},{3,2},{3,3},{3,4}}
```

Siehe auch

TopoElement, TopoElementArray

9.11.4 TopoElement

TopoElement — Konvertiert eine Topogeometrie in ein Topoelement.

Synopsis

topoelement **TopoElement**(topogeometry topo);

Description

Konvertiert eine TopoGeometry in eine TopoElement.

Verfügbarkeit: 3.4.0

Beispiele

Dies ist ein in sich selbst vollkommen abgeschlossener Arbeitsablauf

Return the TopoElement representation of each topogeometry.

Code

```
SELECT TopoElement(topo)
FROM neighborhoods;
```

Use a cast to aggregate topogeometries into a TopoElementArray.

Code

```
SELECT topology.TopoElementArray_Agg(topo::topoelement)
FROM neighborhoods
GROUP BY city;
```

Siehe auch

TopoElementArray_Agg, TopoGeometry, TopoElement

9.12 TopoGeometry Editoren

9.12.1 clearTopoGeom

clearTopoGeom — Löscht den Inhalt einer TopoGeometry.

Synopsis

topogeometry **clearTopoGeom**(topogeometry topogeom);

Description

Löscht den Inhalt einer TopoGeometry und wandelt sie in eine leere um. Am nützlichsten in Verbindung mit toTopoGeom, um die geometrische Gestalt bestehende Objekte und alle abhängigen Objekte in höheren hierarchischen Ebenen zu ersetzen.

Verfügbarkeit: 2.1

Beispiele

Shrink all TopoGeometry polygons by 10 meters.

Code

```
UPDATE nei_topo SET topo = toTopoGeom(ST_Buffer(topo, -10), clearTopoGeom(topo), 0);
```

Siehe auch

toTopoGeom

9.12.2 TopoGeom_addElement

TopoGeom_addElement — Fügt ein Element zu der Definition einer TopoGeometry hinzu.

Synopsis

topogeometry **TopoGeom_addElement**(topogeometry tg, topoelement el);

Description

Fügt ein TopoElement zur Definition eines TopoGeometry-Objekts hinzu. Wenn das Element bereits Teil der Definition ist, führt dies zu keinem Fehler.

Verfügbarkeit: 2.3

Beispiele

Add edge 5 to TopoGeometry `tg`.

Code

```
UPDATE mylayer SET tg = TopoGeom_addElement(tg, '{5,2}');
```

Siehe auch

TopoGeom_remElement, CreateTopoGeom

9.12.3 TopoGeom_remElement

TopoGeom_remElement — Entfernt ein Element aus der Definition einer TopoGeometry.

Synopsis

topogeometry **TopoGeom_remElement**(topogeometry tg, topoelement el);

Description

Entfernt ein TopoElement aus der Ausgestaltung des TopoGeometry Objekts.

Verfügbarkeit: 2.3

Beispiele

Remove face 43 from TopoGeometry `tg`.

Code

```
UPDATE mylayer SET tg = TopoGeom_remElement(tg, '{43,3}');
```

Siehe auch

TopoGeom_addElement, CreateTopoGeom

9.12.4 TopoGeom_addTopoGeom

TopoGeom_addTopoGeom — Fügt Element einer TopoGeometry zur Definition einer anderen TopoGeometry hinzu.

Synopsis

topogeometry **TopoGeom_addTopoGeom**(topogeometry tgt, topogeometry src);

Description

Fügt die Elemente einer TopoGeometry zur Definition einer anderen TopoGeometry hinzu, wobei der zwischengespeicherte Typ (type-Attribut) gegebenenfalls in eine Sammlung geändert wird, um alle Elemente des Quellobjekts aufzunehmen.

Die beiden TopoGeometry-Objekte müssen für die *gleiche* Topologie definiert sein und, falls sie hierarchisch definiert sind, aus Elementen derselben untergeordneten Ebene bestehen.

Verfügbarkeit: 3.2

Beispiele

Set an overall TopoGeometry value to be composed of all elements from specific TopoGeometry values.

Code

```
UPDATE mylayer SET tg_overall = TopoGeom_addTopoGeom(
    TopoGeom_addTopoGeom(
        clearTopoGeom(tg_overall),
        tg_specific1
    ),
    tg_specific2
);
```

Siehe auch

TopoGeom_addElement, clearTopoGeom, CreateTopoGeom

9.12.5 toTopoGeom

toTopoGeom — Fügt eine Geometrie zu einer bestehenden TopoGeometry hinzu.

Description

Siehe toTopoGeom.

9.13 TopoGeometry Accessors

9.13.1 GetTopoGeomElementArray

GetTopoGeomElementArray — Gibt ein `topoelementarray` (ein Feld von topoelements) zurück, das die topologischen Elemente und den Datentyp der gegebenen TopoGeometry (die Elementarstrukturen) enthält.

Synopsis

topoelementarray **GetTopoGeomElementArray**(varchar toponame, integer layer_id, bigint tg_id);

topoelementarray **GetTopoGeomElementArray**(topogeometry tg);

Description

Gibt ein TopoElementArray zurück, das die topologischen Elemente und den Datentyp der gegebenen TopoGeometry (die Elementarstrukturen) enthält. Dies ist ähnlich dem GetTopoGeomElements, ausser dass die Elemente als Feld statt als Datensatz ausgegeben werden.

tg_id steht für die ID des TopoGeometry Objekts der Topologie eines Layers, der durch die `layer_id` der Tabelle "topology.layer" angegeben wird.

Verfügbarkeit: 1.1

Beispiele

Siehe auch

GetTopoGeomElements, TopoElementArray

9.13.2 GetTopoGeomElements

GetTopoGeomElements — Gibt für eine TopoGeometry (Elementarstrukturen) einen Satz an `topoelement` Objekten zurück, welche die topologische element_id und den element_type beinhalten.

Synopsis

setof topoelement **GetTopoGeomElements**(varchar toponame, integer layer_id, bigint tg_id);

setof topoelement **GetTopoGeomElements**(topogeometry tg);

Description

Gibt eine Menge von element_id,element_type (topoelements) zurück, die den primitiven Topologieelementen TopoElement (1: Knoten, 2: Kanten, 3: Flächen) entsprechen, aus denen ein bestimmtes topogeometrisches Objekt im Schema `toponame` besteht.

tg_id steht für die ID des TopoGeometry Objekts der Topologie eines Layers, der durch die `layer_id` der Tabelle "topology.layer" angegeben wird.

Verfügbarkeit: 2.0.0

Beispiele

Siehe auch

GetTopoGeomElementArray, TopoElement, TopoGeom_addElement, TopoGeom_remElement

9.13.3 ST_SRID

ST_SRID — Gibt den räumlichen Referenzbezeichner für eine Topogeometrie zurück.

Synopsis

integer **ST_SRID**(topogeometry tg);

Description

Returns the spatial reference identifier of the topology containing the TopoGeometry.

The identifier is recorded for the topology when it is created and refers to an entry in Abschnitt 4.5.

Verfügbarkeit: 3.2.0

Beispiele

Return the SRID of a TopoGeometry stored in the `ri_topo` topology.

Code

```
SELECT topology.ST_SRID(topo) AS srid
FROM ri.roads
WHERE road_name = 'Unknown';
```

Output

```
srid
------
 3438
```

Siehe auch

Abschnitt 4.5, ST_SetSRID, ST_Transform, ST_SRID

9.14 TopoGeometry Ausgabe

9.14.1 AsGML

AsGML — Gibt die GML-Darstellung einer TopoGeometry zurück.

Synopsis

text **AsGML**(topogeometry tg);
text **AsGML**(topogeometry tg, text nsprefix_in);
text **AsGML**(topogeometry tg, regclass visitedTable);
text **AsGML**(topogeometry tg, regclass visitedTable, text nsprefix);
text **AsGML**(topogeometry tg, text nsprefix_in, integer precision, integer options);
text **AsGML**(topogeometry tg, text nsprefix_in, integer precision, integer options, regclass visitedTable);
text **AsGML**(topogeometry tg, text nsprefix_in, integer precision, integer options, regclass visitedTable, text idprefix);
text **AsGML**(topogeometry tg, text nsprefix_in, integer precision, integer options, regclass visitedTable, text idprefix, int gml-version);

Description

Gibt die GML-Darstellung einer TopoGeometry im GML3-Format aus. Wenn kein `nsprefix_in` angegeben ist, dann wird `gml` verwendet. Übergeben sie eine leere Zeichenfolge für "nsprefix" um keinen bestimmten Namensraum festzulegen. Wenn die Parameter "precision" (Standardwert: 15) und "options" (Standardwert: 1) angegeben sind, werden diese unangetastet an den zugrunde liegenden Aufruf von ST_AsGML übergeben.

Wenn der Parameter `visitedTable` angegeben ist, dann wird dieser verwendet um die bereits besuchten Knoten und Kanten über Querverweise (xlink:xref) zu verfolgen, anstatt Definitionen zu vervielfältigen. Die Tabelle muss (zumindest) zwei Integerfelder enthalten: 'element_type' und 'element_id'. Für den Aufruf muss der Anwender sowohl Lese- als auch Schreibrechte auf die Tabelle besitzen. Um die maximale Rechenleistung zu erreichen, sollte ein Index für die Attribute `element_type` und `element_id` - in dieser Reihenfolge - festgelegt werden. Dieser Index wird automatisch erstellt, wenn auf die Attribute ein Unique Constraint gelegt wird. Beispiel:

Code

```
CREATE TABLE visited (
  element_type integer, element_id integer,
  unique(element_type, element_id)
);
```

Wird der Parameter `idprefix` angegeben, so wird dieser den Identifikatoren der Tags von Kanten und Knoten vorangestellt.

Wird der Parameter `gmlver` angegeben, so wird dieser and das zugrunde liegende ST_AsGML übergeben. Standardmäßig wird 3 angenommen.

Verfügbarkeit: 2.0.0

Beispiele

Hier wird die TopoGeometry verwendet, die wir unter CreateTopoGeom erstellt haben

Code

```
SELECT topology.AsGML(topo) As rdgml
  FROM ri.roads
  WHERE road_name = 'Unknown';
```

Output

```
<gml:TopoCurve>
    <gml:directedEdge>
        <gml:Edge gml:id="E1">
            <gml:directedNode orientation="-">
                <gml:Node gml:id="N1"/>
            </gml:directedNode>
            <gml:directedNode
></gml:directedNode>
            <gml:curveProperty>
                <gml:Curve srsName="urn:ogc:def:crs:EPSG::3438">
                    <gml:segments>
                        <gml:LineStringSegment>
                            <gml:posList srsDimension="2"
>384744 236928 384750 236923 384769 236911 384799 236895 384811 236890
                            384833 236884 384844 236882 384866 236881 384879 236883 384954 236898 385087 236932 385117 ↩
     236938
                            385167 236938 385203 236941 385224 236946 385233 236950 385241 236956 385254 236971
                            385260 236979 385268 236999 385273 237018 385273 237037 385271 237047 385267 237057 385225 ↩
     237125
                            385210 237144 385192 237161 385167 237192 385162 237202 385159 237214 385159 237227 385162 ↩
     237241
                            385166 237256 385196 237324 385209 237345 385234 237375 385237 237383 385238 237399 385236 ↩
     237407
                            385227 237419 385213 237430 385193 237439 385174 237451 385170 237455 385169 237460 385171 ↩
     237475
                            385181 237503 385190 237521 385200 237533 385206 237538 385213 237541 385221 237542 385235 ↩
     237540 385242 237541
                            385249 237544 385260 237555 385270 237570 385289 237584 385292 237589 385291 237596 385284 ↩
     237630</gml:posList>
                        </gml:LineStringSegment>
                    </gml:segments>
                </gml:Curve>
            </gml:curveProperty>
        </gml:Edge>
    </gml:directedEdge>
</gml:TopoCurve>
```

Selbes Beispiel wie das Vorige, aber ohne Namensraum

Code

```
SELECT topology.AsGML(topo,'') As rdgml
  FROM ri.roads
  WHERE road_name = 'Unknown';
```

Output

```
<TopoCurve>
    <directedEdge>
        <Edge id="E1">
            <directedNode orientation="-">
                <Node id="N1"/>
            </directedNode>
            <directedNode
></directedNode>
            <curveProperty>
                <Curve srsName="urn:ogc:def:crs:EPSG::3438">
                    <segments>
                        <LineStringSegment>
                            <posList srsDimension="2"
>384744 236928 384750 236923 384769 236911 384799 236895 384811 236890
                            384833 236884 384844 236882 384866 236881 384879 236883 384954 236898 385087 236932 385117 ↩
    236938
                            385167 236938 385203 236941 385224 236946 385233 236950 385241 236956 385254 236971
                            385260 236979 385268 236999 385273 237018 385273 237037 385271 237047 385267 237057 385225 ↩
    237125
                            385210 237144 385192 237161 385167 237192 385162 237202 385159 237214 385159 237227 385162 ↩
    237241
                            385166 237256 385196 237324 385209 237345 385234 237375 385237 237383 385238 237399 385236 ↩
    237407
                            385227 237419 385213 237430 385193 237439 385174 237451 385170 237455 385169 237460 385171 ↩
    237475
                            385181 237503 385190 237521 385200 237533 385206 237538 385213 237541 385221 237542 385235 ↩
    237540 385242 237541
                            385249 237544 385260 237555 385270 237570 385289 237584 385292 237589 385291 237596 385284 ↩
    237630</posList>
                        </LineStringSegment>
                    </segments>
                </Curve>
            </curveProperty>
        </Edge>
    </directedEdge>
</TopoCurve>
```

Siehe auch

CreateTopoGeom, ST_CreateTopoGeo

9.14.2 AsTopoJSON

AsTopoJSON — Gibt die TopoJSON-Darstellung einer TopoGeometry zurück.

Synopsis

text **AsTopoJSON**(topogeometry tg, regclass edgeMapTable);

Description

Gibt eine TopoGeometry in der TopoJSON-Darstellung zurück. Wenn `edgeMapTable` nicht NULL ist, wird diese als Lookup/Speicher für die Abbildung der Identifikatoren der Kanten auf die Indizes der Kreisbögen verwendet. Dadurch wird ein kompaktes Feld "arcs" im endgültigen Dokument ermöglicht.

Wenn die Tabelle angegeben ist, wird die Existenz der Attribute "arc_id" vom Datentyp "serial" und "edge_id" vom Typ "integer" vorausgesetzt; da der Code die Tabelle nach der "edge_id" abfragt, sollte ein Index für dieses Attribut erstellt werden.



Anmerkung
Die Kreisbögen in der TopoJSON Ausgabe sind von 0 weg indiziert, während sie in der Tabelle "edgeMapTable" 1-basiert sind.

Ein vollständiges TopoJson Dokument benötigt zusätzlich zu den von dieser Funktion ausgegebenen Schnipseln, die tatsächlichen Bögen und einige Header. SieheTopoJSON specification.

Verfügbarkeit: 2.1.0

Erweiterung: 2.2.1 Unterstützung für punktförmige Eingabewerte hinzugefügt

Siehe auch

ST_AsGeoJSON

Beispiele

Create a temporary edge map for the TopoJSON output.

Code

```
CREATE TEMP TABLE edgemap(arc_id serial, edge_id int unique);
```

Build the TopoJSON header, objects, arcs, and footer.

Code

```
SELECT '{ "type": "Topology", "transform": { "scale": [1,1], "translate": [0,0] }, "objects ↩
    ": {'

UNION ALL SELECT '"' || feature_name || '": ' || AsTopoJSON(feature, 'edgemap')
FROM features.big_parcels WHERE feature_name = 'P3P4';

WITH edges AS (
  SELECT m.arc_id, e.geom FROM edgemap m, city_data.edge e
  WHERE e.edge_id = m.edge_id
), points AS (
  SELECT arc_id, (ST_DumpPoints(geom)).* FROM edges
), compare AS (
  SELECT p2.arc_id,
         CASE WHEN p1.path IS NULL THEN p2.geom
              ELSE ST_Translate(p2.geom, -ST_X(p1.geom), -ST_Y(p1.geom))
         END AS geom
  FROM points p2 LEFT OUTER JOIN points p1
  ON ( p1.arc_id = p2.arc_id AND p2.path[1] = p1.path[1]+1 )
  ORDER BY arc_id, p2.path
), arcsdump AS (
  SELECT arc_id, (regexp_matches(ST_AsGeoJSON(geom), '\[.*\]'))[1] as t
  FROM compare
), arcs AS (
  SELECT arc_id, '[' || array_to_string(array_agg(t), ',') || ']' as a FROM arcsdump
  GROUP BY arc_id
  ORDER BY arc_id
)
SELECT '}, "arcs": [' UNION ALL
SELECT array_to_string(array_agg(a), E',\n') from arcs

UNION ALL SELECT ']}'::text as t;
```

Output

```
{ "type": "Topology", "transform": { "scale": [1,1], "translate": [0,0] }, "objects": {
"P3P4": { "type": "MultiPolygon", "arcs": [[[-1]],[[6,5,-5,-4,-3,1]]]}
}, "arcs": [
 [[25,30],[6,0],[0,10],[-14,0],[0,-10],[8,0]],
 [[35,6],[0,8]],
 [[35,6],[12,0]],
 [[47,6],[0,8]],
 [[47,14],[0,8]],
 [[35,22],[12,0]],
 [[35,14],[0,8]]
 ]}
```

9.15 Räumliche Beziehungen einer Topologie

9.15.1 Equals

Equals — Gibt TRUE zurück, wenn zwei TopoGeometry Objekte aus denselben topologischen Elementarstrukturen bestehen.

Synopsis

boolean **Equals**(topogeometry tg1, topogeometry tg2);

Description

Gibt TRUE zurück, wenn zwei TopoGeometry Objekte aus denselben topologischen Elementarstrukturen: Maschen, Kanten, Knoten, bestehen.



Anmerkung
Diese Funktion unterstützt keine TopoGeometry aus einer Sammelgeometrie. Es kann auch keine TopoGeometry Objekte unterschiedlicher Topologien vergleichen

Verfügbarkeit: 1.1.0

 Diese Funktion unterstützt 3d und lässt den Z-Index nicht fallen.

Beispiele

The following example uses the `nei_topo` table created in the toTopoGeom examples.

Code

```
SELECT topology.Equals(a.topo, b.topo)
FROM nei_topo AS a
JOIN nei_topo AS b ON b.gid = a.gid
WHERE a.gid = 1;
```

Output

```
t
```

Siehe auch

GetTopoGeomElements, ST_Equals

9.15.2 Intersects

Intersects — Gibt TRUE zurück, wenn sich kein beliebiges Paar von Elemtarstrukturen zweier TopoGeometry Objekte überschneidet.

Synopsis

boolean **Intersects**(topogeometry tg1, topogeometry tg2);

Description

Gibt TRUE zurück, wenn sich kein beliebiges Paar von Elemtarstrukturen zweier TopoGeometry Objekte überschneidet.



Anmerkung
Diese Funktion wird nicht für Topogeometrien unterstützt, die Geometriesammlungen sind. Sie kann auch keine Topogeometrien aus verschiedenen Topologien vergleichen. Außerdem wird sie derzeit nicht für hierarchische Topogeometrien (Topogeometrien, die aus anderen Topogeometrien bestehen) unterstützt.

Verfügbarkeit: 1.1.0

 Diese Funktion unterstützt 3d und lässt den Z-Index nicht fallen.

Beispiele

The following example uses the `nei_topo` table created in the toTopoGeom examples.

Code

```
SELECT topology.Intersects(a.topo, b.topo)
FROM nei_topo AS a
JOIN nei_topo AS b ON b.gid = a.gid
WHERE a.gid = 1;
```

Output

```
t
```

Siehe auch

ST_Intersects

9.16 Importieren und Exportieren von Topologien

Nachdem Sie Topologien und eventuell zugehörige topologische Ebenen erstellt haben, möchten Sie diese vielleicht in ein dateibasiertes Format exportieren, um sie zu sichern oder in eine andere Datenbank zu übertragen.

Die Verwendung der Standard-Dump/Restore-Tools von PostgreSQL ist problematisch, da Topologien aus einer Reihe von Tabellen (4 für Primitive, eine beliebige Anzahl für Layer) und Datensätzen in Metadatentabellen (topology.topology und topology.layer) bestehen. Außerdem sind die Topologiebezeichner nicht datenbankübergreifend einheitlich, so dass die Parameter Ihrer Topologie bei der Wiederherstellung geändert werden müssen.

Um das Exportieren/Wiederherstellen von Topologien zu vereinfachen, werden zwei ausführbare Dateien bereitgestellt: `pgtopo_export` und `pgtopo_import`. Beispiel für die Verwendung:

Code

```
pgtopo_export dev_db topo1 | pgtopo_import topo1 | psql staging_db
```

9.16.1 Verwendung des Topologie-Exporters

Das Skript `pgtopo_export` nimmt den Namen einer Datenbank und eine Topologie und gibt eine Dump-Datei aus, die zum Importieren der Topologie (und der zugehörigen Ebenen) in eine neue Datenbank verwendet werden kann.

Standardmäßig schreibt `pgtopo_export` die Speicherauszugsdatei in die Standardausgabe, so dass sie an `pgtopo_import` weitergeleitet oder in eine Datei umgeleitet werden kann (Ablehnung des Schreibens ins Terminal). Sie können optional einen Ausgabedateinamen mit der Befehlszeilenoption `-f` angeben.

Standardmäßig enthält `pgtopo_export` einen Speicherauszug aller für die angegebene Topologie definierten Ebenen. Dies können mehr Daten sein, als Sie benötigen, oder sie können nicht funktionieren (falls Ihre Ebenentabellen komplexe Abhängigkeiten haben). In diesem Fall können Sie mit dem Schalter `--skip-layers` verlangen, dass die Ebenen übersprungen werden, und diese separat behandeln.

Der Aufruf von `pgtopo_export` mit dem Schalter `--help` (oder kurz `-h`) gibt immer einen kurzen Verwendungsstring aus.

Das Format der Dump-Datei ist ein komprimiertes tar-Archiv eines `pgtopo_export` Verzeichnisses, das mindestens eine `pgtopo_dump_version` Datei mit Informationen zur Formatversion enthält. Ab Version `1` enthält das Verzeichnis tabulatorgetrennte CSV-Dateien mit den Daten der Topologie-Primitivtabellen (node, edge_data, face, relation), den damit verbundenen Topologie- und Ebenendatensätzen und (sofern nicht `--skip-layers` angegeben ist) einem PostgreSQL-Dump im benutzerdefinierten Format der Tabellen, die als Ebenen der angegebenen Topologie gemeldet werden.

9.16.2 Verwendung des Topologie-Importers

Das Skript `pgtopo_import` nimmt einen Topologie-Dump im Format `pgtopo_export` und einen Namen für die zu erstellende Topologie und gibt ein SQL-Skript aus, das die Topologie und die zugehörigen Ebenen rekonstruiert.

Die generierte SQL-Datei enthält Anweisungen, die eine Topologie mit dem angegebenen Namen erstellen, Primitivdaten in sie laden, alle Topologieebenen wiederherstellen und registrieren, indem sie alle TopoGeometry-Werte ordnungsgemäß mit der richtigen Topologie verknüpfen.

Standardmäßig liest `pgtopo_import` den Speicherauszug von der Standardeingabe, so dass er in Verbindung mit `pgtopo_export` in einer Pipeline verwendet werden kann. Sie können optional einen Eingabedateinamen mit der Befehlszeilenoption `-f` angeben.

Standardmäßig enthält `pgtopo_import` in der SQL-Ausgabedatei den Code zur Wiederherstellung aller im Dump gefundenen Ebenen.

Dies kann unerwünscht sein oder nicht funktionieren, wenn Ihre Zieldatenbank bereits Tabellen mit dem gleichen Namen wie die im Dump hat. In diesem Fall können Sie das Überspringen der Ebenen mit dem Schalter `--skip-layers` anfordern und diese separat (oder später) bearbeiten.

SQL, um nur Ebenen zu laden und mit einer benannten Topologie zu verknüpfen, kann mit dem Schalter `--only-layers` erzeugt werden. Dies kann nützlich sein, um Ebenen zu laden, NACHDEM die Benennungskonflikte gelöst wurden, oder um Ebenen mit einer anderen Topologie zu verknüpfen (z. B. einer räumlich vereinfachten Version der Ausgangstopologie).

If the target topology already exists and you want it dropped upfront you can pass the `--drop-topology` switch (since PostGIS-3.6.0).

Kapitel 10

Rasterdatenverwaltung, -abfrage und Anwendungen

10.1 Laden und Erstellen von Rastertabellen

In den häufigsten Anwendungsfällen werden Sie einen PostGIS-Raster durch das Laden einer bestehenden Rasterdatei, mit Hilfe des Rasterladers `raster2pgsql`, erstellen.

10.1.1 Verwendung von raster2pgsql zum Laden von Rastern

Das `raster2pgsql` ist ein Rasterladeprogramm, das GDAL-unterstützte Rasterformate in eine SQL-Datei lädt, die für das Laden in eine PostGIS-Rastertabelle geeignet ist. Es ist in der Lage, Ordner mit Rasterdateien zu laden und Übersichten über Raster zu erstellen.

Da raster2pgsql in den meisten Fällen als Teil von PostGIS kompiliert wird (es sei denn, Sie kompilieren Ihre eigene GDAL-Bibliothek), werden die von der ausführbaren Datei unterstützten Rastertypen dieselben sein wie die in der GDAL-Abhängigkeitsbibliothe kompilierten. Um eine Liste der Rastertypen zu erhalten, die Ihr spezielles `raster2pgsql` unterstützt, verwenden Sie den Schalter `-G`.



Anmerkung
Bei einem bestimmten Faktor kann es vorkommen, dass die Raster in der Übersicht/Overview nicht bündig angeordnet sind, obwohl sie die Raster selbst dies sind. Siehe http://trac.osgeo.org/postgis/ticket/1764 für ein solches Beispiel.

10.1.1.1 Beispiel für die Verwendung

Eine Beispielssitzung, wo mit dem Lader eine Eingabedatei erstellt und stückchenweise als 100x100 Kacheln hochgeladen wird, könnte so aussehen:

The loader assigns SRID 4326, creates an index and standard constraints, analyzes the loaded table, records source filenames, and stores the input rasters in 100 by 100 pixel tiles.

Code

```
raster2pgsql -s 4326 -I -C -M -F -t 100x100 *.tif public.demelevation
> elev.sql
psql -d gisdb -f elev.sql
```



Anmerkung
Wenn Sie das Schema nicht als Teil des Namens der Zieltabelle angeben, wird die Tabelle im Standardschema der Datenbank oder des Benutzers, mit dem Sie sich verbinden, erstellt.

Durch die Verwendung von UNIX-Pipes kann die Konvertierung und der Upload in einem Schritt vollzogen werden:

Code

```
raster2pgsql -s 4326 -I -C -M -F -t 100x100 \
  *.tif public.demelevation |
  psql -d gisdb
```

Luftbildkacheln in "Massachusetts State Plane Meters" in das Schema `aerial` laden. Einen vollständigen View und Übersichtstabellen mit Faktor 2 und 4 erstellen. Verwendet den Modus "copy" für das Insert (keine dazwischengeschaltete Datei, sondern direkt in die Datenbank). Die Option -e bedingt, dass nicht alles innerhalb einer Transaktion abläuft (nützlich, wenn Sie sofort Daten sehen wollen, ohne zu warten). Die Raster werden in 128x128 Pixel große Kacheln zerlegt und Constraints auf die Raster gesetzt. Verwendet den Modus "copy" anstelle eines Tabellen-Inserts. (-F) Erzeugt das Attribut "filename", welches die Bezeichnung der Ausgangsdateien enthält, aus denen die Rasterkacheln ausgeschnitten wurden.

Code

```
raster2pgsql -I -C -e -Y -F -s 26986 -t 128x128 -l 2,4 \
  bostonaerials2008/*.jpg aerials.boston |
  psql -U postgres -d gisdb -h localhost -p 5432
```

List the raster formats supported by this build:

Code

```
raster2pgsql -G
```

The `-G` command outputs a list something like:

Output

```
Available GDAL raster formats:
  Virtual Raster
  GeoTIFF
  National Imagery Transmission Format
  Raster Product Format TOC format
  ECRG TOC format
  Erdas Imagine Images (.img)
  CEOS SAR Image
  CEOS Image
  ...
  Arc/Info Export E00 GRID
  ZMap Plus Grid
  NOAA NGS Geoid Height Grids
```

10.1.1.2 raster2pgsql-Optionen

Long options that take values also accept the `--option=value` spelling.

`-?, --help` Zeigt die Hilfe an, auch dann, wenn keine Argumente übergeben werden.

`-G, --gdal-formats` Gibt die unterstützten Rasterformate aus.

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

`-c` Eine neue Tabelle anlegen und mit Raster(n) befüllen, *this is the default mode*

`-a` Raster zu einer bestehende Tabelle hinzufügen.

-d Tabelle löschen, eine Neu erzeugen und mit einem oder mehreren Raster befüllen

-p Beim vorbereitenden Modus wird ledigliche eine Tabelle erstellt.

--if-not-exists Use `IF NOT EXISTS` for table creation in `-c` and `-p` modes. When `-I` or `--create-index` is also specified, use `IF NOT EXISTS` for index creation too. This option cannot be used with `-d`. Append mode requires an explicit creation action.

Atomic loader actions The short modes are presets over explicit actions. The equivalent long options are `--drop-table`, `--create-table`, `--load-data`, `--create-index`, `--add-constraints`, `--vacuum`, `--analyze`, and `--no-transaction`. Add `--if-not-exists` to make active creation actions use `IF NOT EXISTS`. `--drop-table` emits `DROP TABLE IF EXISTS` before the selected actions. With no mode specified, the default create/load actions still apply.

Raster-Verarbeitung: Anwendung von Constraint's zur ordnungsgemäßen Registrierung im Rasterkatalog

-C, --add-constraints Anwendung von Raster-Constraints, wie SRID, Zellgröße etc., um die ordnungsgemäße Registrierung des Rasters in der `raster_columns` View sicherzustellen.

-x, --no-extent Disable setting the max extent constraint. Only applied if -C or --add-constraints is also used.

-r, --regular-blocking Set the constraints (spatially unique and coverage tile) for regular blocking. Only applied if -C or --add-constraints is also used.

Rasterdaten-Verarbeitung: Optionale Parameter zur Manipulation von Input Raster Datensätzen

-s, --srid [<FROM_SRID>:]<SRID> Assign the output raster to the specified SRID. When FROM_SRID is also provided, raster2pgsql marks the input raster with FROM_SRID and emits SQL that reprojects the raster to the target SRID. If the source SRID is omitted or is zero, the raster metadata will be checked to determine an appropriate source SRID. Reprojection cannot be used with copy mode.

-b, --band BAND Die Kennung (1-basiert) des Bandes, das aus dem Raster entnommen werden soll. Um mehrere Bänder anzugeben, trennen Sie die Kennungen bitte durch ein Komma (,).

-t, --tile-size TILE_SIZE Zerlegt den Raster in Kacheln, um eine Kachel pro Tabellenzeile einzufügen. `TILE_SIZE` wird entweder in BREITExHöhe ausgedrückt, oder auf den Wert "auto" gesetzt, wodurch der Raster-Lader eine passende Kachelgröße an Hand des ersten Raster's ermittelt und diese dann auf die anderen Raster anwendet.

-P, --pad Die ganz rechts und ganz unten liegenden Kacheln aufstocken, damit für alle Kacheln gleiche Breite und Höhe sichergestellt ist.

-R, --register Einen im Dateisystem vorliegenden Raster als (out-db) Raster registrieren.
Es werden nur die Metadaten und der Dateipfad des Rasters abgespeichert (nicht die Rasterzellen).

-l, --overview-factor 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. If a source raster already has an overview with matching dimensions, that overview is reused. Created overview is stored in the database and is not affected by -R. Note that your generated sql file will contain both the main table and overview tables.

-N, --nodata NODATA Der NODATA-Wert, der für Bänder verwendet wird, die keinen NODATA-Wert definiert haben.

Optionale Parameter zur Manipulation von Datenbankobjekten

-f, --raster-column COLUMN Gibt den Spaltennamen des Zielrasters an; standardmäßig wird er 'rast' benannt.

-F, --filename Eine Spalte mit dem Dateinamen hinzufügen

-n, --filename-column COLUMN Gibt die Bezeichnung für die Spalte mit dem Dateinamen an. Schließt -F" mit ein.

-q, --quote Setzt die PostgreSQL-Identifikatoren unter Anführungszeichen.

-I Alias for `--create-index`.

--create-index Create a GiST index on the raster column at the end of this raster2pgsql run. With repeated `-a` append runs, create the index on the final run or after loading; add `--if-not-exists` to make reruns tolerate an existing index.

-M VACUUM ANALYZE auf die Rastertabelle.

-k, --skip-nodata-check Behält leere Kacheln bei und überspringt die Überprüfung der NODATA-Werte für jedes Rasterband. Beachten Sie, dass Sie bei der Überprüfung Zeit sparen, aber am Ende viel mehr Junk-Zeilen in Ihrer Datenbank haben können, die nicht als leere Kacheln markiert sind.

-T, --tablespace tablespace Bestimmt den Tablespace für die neue Tabelle. Beachten Sie bitte, dass Indizes (einschließlich des Primärschlüssels) weiterhin den standardmäßigen Tablespace nutzen, solange nicht die -X Flag benutzt wird.

-X, --index-tablespace tablespace Bestimmt den Tablespace für den neuen Index der Tabelle. Dieser gilt sowohl für den Primärschlüssel als auch für den räumlichen Index, falls die -I Flag gesetzt ist.

-Y, --copy [MAX_ROWS_PER_COPY] Verwenden Sie Kopieranweisungen anstelle von Einfügeanweisungen. Optional können Sie `max_rows_per_copy` angeben; Standardwert 50, wenn nicht angegeben.

-e, --no-transaction Keine Transaktion verwenden, sondern jede Anweisung einzeln ausführen.

-E, --endian ENDIAN Legt die Byte-Reihenfolge des binär erstellten Rasters fest; geben Sie für XDR 0 und für NDR (Standardwert) 1 an; zurzeit wird nur die Ausgabe von NDR unterstützt.

-V, --wkb-version version Bestimmt die Version des Ausgabeformats. Voreingestellt ist 0. Zur Zeit wird auch nur 0 unterstützt.

10.1.2 Erzeugung von Rastern mit den PostGIS Rasterfunktionen

Oftmals werden Sie die Raster und die Rastertabellen direkt in der Datenbank erzeugen wollen. Dafür existieren eine Unmenge an Funktionen. Dies verlangt im Allgemeinen die folgende Schritte.

1. Erstellung einer Tabelle mit einer Rasterspalte für die neuen Rasterdatensätze:

   Code

   ```
   CREATE TABLE myrasters(rid serial primary key, rast raster);
   ```

2. Es existieren viele Funktionen die Ihnen helfen dieses Ziel zu erreichen. Wenn Sie einen Raster nicht von anderen Rastern ableiten, sondern selbst erzeugen, können Sie mit ST_MakeEmptyRaster beginnen, gefolgt von ST_AddBand

   Sie können Raster auch aus Geometrien erzeugen. Hierzu können Sie ST_AsRaster verwenden, möglicherweise in Verbindung mit anderen Funktionen, wie ST_Union, ST_MapAlgebraFct oder irgendeiner anderen Map Algebra Funktion.

   Es gibt sogar noch viele andere Möglichkeiten, um eine neue Rastertabelle aus bestehenden Tabellen zu erzeugen. Sie können zum Beispiel mit ST_Transform einen Raster in eine andere Projektion transformieren und so eine neue Rastertabelle erstellen.

3. Wenn Sie mit der Erstbefüllung der Tabelle fertig sind, werden Sie einen räumlichen Index auf die Rasterspalte setzen wollen:

   Code

   ```
   CREATE INDEX myrasters_rast_st_convexhull_idx ON myrasters USING gist(ST_ConvexHull( ↩
       rast));
   ```

   Beachten Sie bitte die Verwendung von ST_ConvexHull; der Grund dafür ist, dass die meisten Rasteroperatoren auf der konvexen Hülle des Rasters beruhen.

   Anmerkung
   Vor der Version 2.0 von PostGIS, basierten die Raster auf der Einhüllenden, anstatt auf der konvexen Hülle. Damit die räumlichen Indizes korrekt funktionieren, müssen Sie diese löschen und mit einem auf der konvexen Hülle basierenden Index ersetzen.

4. Mittels AddRasterConstraints Bedingungen auf den Raster legen.

10.1.3 Verwendung von "out db"-Wolkenrastern

Das Tool `raster2pgsql` verwendet GDAL, um auf Rasterdaten zuzugreifen, und kann eine wichtige GDAL-Funktion nutzen: die Fähigkeit, von Rastern zu lesen, die entfernt in Cloud-"Objektspeichern" (z. B. AWS S3, Google Cloud Storage) gespeichert sind.

Eine effiziente Nutzung von in der Cloud gespeicherten Rastern erfordert die Verwendung eines "cloudoptimierten" Formats. Das bekannteste und am weitesten verbreitete Format ist das "cloud optimized GeoTIFF". Die Verwendung eines nicht cloud-optimierten Formats wie JPEG oder TIFF ohne Kacheln führt zu einer sehr schlechten Leistung, da das System jedes Mal das gesamte Raster herunterladen muss, wenn es auf eine Teilmenge zugreifen will.

Laden Sie zunächst Ihr Raster in den Cloud-Speicher Ihrer Wahl. Sobald es geladen ist, haben Sie eine URI, mit der Sie darauf zugreifen können, entweder eine "http"-URI oder manchmal eine für den Dienst spezifische URI. (z. B. "s3://bucket/object"). Um auf nicht-öffentliche Buckets zuzugreifen, müssen Sie GDAL-Konfigurationsoptionen angeben, um Ihre Verbindung zu authentifizieren. Beachten Sie, dass dieser Befehl aus dem Cloud-Raster liest und in die Datenbank schreibt.

Code

```
AWS_ACCESS_KEY_ID=xxxxxxxxxxxxxxxxxxx \
AWS_SECRET_ACCESS_KEY=xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx \
raster2pgsql \
  -s 990000 \
  -t 256x256 \
  -I \
  -R \
  /vsis3/your.bucket.com/your_file.tif \
  your_table \
  | psql your_db
```

Sobald die Tabelle geladen ist, müssen Sie der Datenbank die Berechtigung geben, von entfernten Rastern zu lesen, indem Sie zwei Berechtigungen festlegen: postgis.enable_outdb_rasters und postgis.gdal_enabled_drivers.

Code

```
SET postgis.enable_outdb_rasters = true;
SET postgis.gdal_enabled_drivers TO 'ENABLE_ALL';
```

Um die Änderungen dauerhaft zu machen, legen Sie sie direkt in Ihrer Datenbank fest. Sie müssen die Verbindung erneut herstellen, damit die neuen Einstellungen wirksam werden.

Code

```
ALTER DATABASE your_db SET postgis.enable_outdb_rasters = true;
ALTER DATABASE your_db SET postgis.gdal_enabled_drivers TO 'ENABLE_ALL';
```

Für nicht-öffentliche Raster müssen Sie möglicherweise Zugriffsschlüssel bereitstellen, um aus den Cloud-Rastern zu lesen. Dieselben Schlüssel, die Sie für den Aufruf `raster2pgsql` verwendet haben, können für die Verwendung innerhalb der Datenbank mit der Konfiguration postgis.gdal_vsi_options festgelegt werden. Beachten Sie, dass mehrere Optionen gesetzt werden können, indem die `Schlüssel=Wert-Paare` durch Leerzeichen getrennt werden.

Code

```
SET postgis.gdal_vsi_options = 'AWS_ACCESS_KEY_ID=xxxxxxxxxxxxxxxxxxx
AWS_SECRET_ACCESS_KEY=xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx';
```

Sobald Sie die Daten geladen und die Berechtigungen festgelegt haben, können Sie mit der Rastertabelle wie mit jeder anderen Rastertabelle interagieren und dieselben Funktionen verwenden. Die Datenbank kümmert sich um die Verbindung zu den Cloud-Daten, wenn sie Pixeldaten lesen muss.

10.2 Raster Coverages and Raster/Vector Analysis

A coverage is a way to interpret spatial data as a mapping from a *domain* of locations to a *range* of values. Evaluating a coverage means asking for the value at a location; an inverse query asks for the locations whose values match a condition. PostGIS does not expose a generic ISO 19123 coverage SQL type, but geometry tables and raster values provide the building blocks for both directions.

A coverage can be continuous, with values calculated or interpolated for arbitrary locations, or discrete, with values supplied by stored spatial objects or grid cells. The coverage model describes how locations map to values, independently of whether those values come from an equation, a table, or one or more raster values. A time column or function argument can add a temporal dimension without changing that distinction.

In a vector coverage, a geometry column supplies the domain and other columns supply attributes in the range. A vector range can contain text, numbers, or another spatial value, provided its meaning and type are consistent wherever the coverage is defined. In a raster, each cell location is part of the domain and the band values at that cell form the range. Other columns in the same table row describe the raster value or tile as a whole; they do not attach separate values to individual cells.

A logical raster coverage may be stored as one raster value or as many tiles in a table. Tiles that belong to one coverage should have compatible SRIDs, scales, alignment, band counts, pixel types, and NODATA conventions. The constraints exposed by Abschnitt 10.3.1 record and validate those properties.

Raster table arrangements. A raster column can represent an image warehouse, a tiled coverage, or a collection of rasterized objects. These arrangements have different assumptions about tile size, alignment, gaps, and overlaps. PostGIS does not infer the intended arrangement from the column type alone.

Arrangement	Row relationship	Typical use
Image warehouse	Independent rasters; sizes, alignment, and extents may differ or overlap	Cataloging unrelated source images
Irregular tiled coverage	One logical coverage with gaps allowed, variable tile sizes, and normally no overlap	Incremental or source-driven mosaics
Regular tiled coverage	One alignment and tile size with no overlap; gaps may remain at the boundary or inside the coverage	Indexed analysis over a large coverage
Rectangular regular coverage	Equal aligned tiles fill a rectangular extent without gaps or overlaps	Predictable block processing and overview generation
Separate tiled images	Each image is internally regular but stored in its own table or partition	Keeping source-image identity and lifecycle separate
Raster-object coverage	Each row is a rasterized feature; extents and sizes may differ and overlap	Object models where cells replace or supplement vector geometry

Use AddRasterConstraints to record common SRID, scale, alignment, band, extent, and regular-blocking properties, and use spatial indexes and application constraints for relationships that the catalog cannot express.

Queries against a tiled coverage normally have two stages:

1. Use the raster spatial index and ST_Intersects to select candidate tiles.
2. Read, clip, aggregate, or vectorize values inside those tiles with functions such as ST_Value, ST_Clip, ST_SummaryStatsAgg, and ST_DumpAsPolygons.

ST_DumpAsPolygons returns geomval rows that pair geometries with cell values. This allows ordinary geometry overlay and filtering to answer inverse coverage queries, but polygonizing many cells is usually more expensive than keeping the operation in raster space.

Choose boundary semantics deliberately. Clipping and aggregating raster pixels is usually the shorter and faster path. If a partial boundary cell must contribute in proportion to its covered area, intersect the geometry with the geomval rows and calculate

an area-weighted result in an appropriate projected coordinate system. That exact partial-cell method costs more because it vectorizes the selected cells.

Tile size affects both sides of the indexed query: smaller tiles reduce the amount of raster data inspected for each candidate row, while increasing row and index overhead. Benchmark representative queries instead of treating one tile size as a universal default.

For a current end-to-end raster/vector pattern, load and tile data with Abschnitt 10.1.1, then see the building example in ST_SummaryStats It selects intersecting tiles, clips them to each polygon, and combines per-tile statistics into one result per feature. For point observations such as animal locations, first buffer the points in an appropriate projected coordinate system, then apply the same tile selection, clipping, and aggregation pattern to an elevation or other continuous raster coverage.

10.3 Raster Katalog

Mit PostGIS kommen zwei Views des Rasterkatalogs. Beide Views nützen die Information, welche in den Bedingungen/Constraints der Rastertabellen festgelegt ist. Da die Bedingungen zwingend sind, sind die Views des Rasterkatalogs immer konsistent mit den Daten in den Rastertabellen.

1. `raster_columns` diese View/gespeicherte Abfrage katalogisiert alle Rastertabellenspalten Ihrer Datenbank.
2. `raster_overviews` Dieser View katalogisiert all jene Spalten einer Rastertabelle in Ihrer Datenbank, die als Übersicht für Rastertabellen mit höherer Auflösung dienen. Tabellen dieses Typs werden mit der `-l` Option beim Laden erstellt.

10.3.1 Rasterspalten Katalog

`raster_columns` ist ein Katalog mit allen Rasterspalten Ihrer Datenbanktabellen. Es handelt sich dabei um einen View, der die Constraints auf die Tabellen ausnutzt, um so immer konsistent mit dem aktuellen Stand der Datenbank zu bleiben; sogar dann, wenn Sie den Raster aus einem Backup oder einer anderen Datenbank wiederherstellen. Der `raster_columns` Katalog beinhaltet die folgenden Spalten.

Falls Sie Ihre Tabellen nicht mit dem Loader erstellt haben, oder vergessen haben, die `-C` Option während des Ladens anzugeben, können Sie die Constraints auch anschließend erzwingen, indem Sie AddRasterConstraints verwenden, wodurch der `raster_columns` Katalog die Information über Ihre Rasterkacheln, wie üblich abspeichert.

- `r_table_catalog` Die Datenbank, in der sich die Tabelle befindet. Greift immer auf die aktuelle Datenbank zu.
- `r_table_schema` Das Datenbankschema in dem sich die Rastertabelle befindet.
- `r_table_name` Rastertabelle
- `r_raster_column` Die Spalte, in der Tabelle `r_table_name`, die den Datentyp Raster aufweist. In PostGIS gibt es nichts, was Sie daran hindert, mehrere Rasterspalten in einer Tabelle zu haben. Somit ist es möglich auf unterschiedliche Raster(spalten) in einer einzigen Rastertabelle zuzugreifen.
- `srid` Der Identifikator für das Koordinatensystem in dem der Raster vorliegt. Sollte in Abschnitt 4.5 eingetragen sein.
- `scale_x` Der Skalierungsfaktor zwischen den Koordinaten der Vektoren und den Pixeln. Dieser steht nur dann zur Verfügung, wenn alle Kacheln der Rasterspalte denselben `scale_x` aufweisen und dieser Constraint auch gesetzt ist. Siehe ST_ScaleX für genauere Angaben.
- `scale_y` Der Skalierungsfaktor zwischen den Koordinaten der Vektoren und den Pixeln. Dieser steht nur dann zur Verfügung, wenn alle Kacheln der Rasterspalte denselben `scale_y` aufweisen und der Constraint `scale_y` auch gesetzt ist. Siehe ST_ScaleY für genauere Angaben.
- `blocksize_x` Die Breite (Anzahl der waagrechten Zellen) einer Rasterkachel. Siehe ST_Width für weitere Details.
- `blocksize_y` Die Höhe (Anzahl der senkrechten Zellen) einer Rasterkachel. Siehe ST_Height für weitere Details.
- `same_alignment` Eine boolesche Variable, die TRUE ist, wenn alle Rasterkacheln dieselbe Ausrichtung haben. Siehe ST_SameAlignment für genauere Angaben.

- `regular_blocking` Wenn auf die Rasterspalte die Constraints für die räumliche Eindeutigkeit und für die Coveragekachel gesetzt sind, ist der Wert TRUE, ansonsten FALSE..
- `num_bands` Die Anzahl der Bänder, die jede Kachel des Rasters aufweist. ST_NumBands gibt die gleiche Information aus. ST_NumBands
- `pixel_types` Ein Feld das den Pixeltyp für die Bänder festlegt. Die Anzahl der Elemente in diesem Feld entspricht der Anzahl der Rasterbänder. Die "pixel_types" sind unter ST_BandPixelType definiert.
- `nodata_values` Ein Feld mit Double Precision Zahlen, welche den `nodata_value` für jedes Band festlegen. Die Anzahl der Elemente in diesem Feld entspricht der Anzahl der Rasterbänder. Diese Zahlen legen den Pixelwert für jedes Rasterband fest, der bei den meisten Operationen ignoriert wir. Eine ähnliche Information erhalten Sie durch ST_BandNoDataValue.
- `out_db` Ein Feld mit booleschen Flags, das anzeigt, ob die Rasterbanddaten außerhalb der Datenbank gehalten werden. Die Anzahl der Elemente in diesem Feld entspricht der Anzahl der Rasterbänder.
- `extent` Die Ausdehnung aller Rasterspalten in Ihrem Rasterdatensatz. Falls Sie vor haben Daten zu laden, welche die Ausdehnung des Datensatzes ändern, sollten Sie die Funktion DropRasterConstraints ausführen, bevor Sie die Daten laden und nach dem Laden die Constraints mit der Funktion AddRasterConstraints erneut setzen.
- `spatial_index` Eine Boolesche Variable, die TRUE anzeigt, wenn ein räumlicher Index auf das Rasterattribut gelegt ist.

10.3.2 Raster Übersicht/Raster Overviews

`raster_overviews` Katalogisiert Information über die Rastertabellenspalten die für die Übersichten/Overviews herangezogen wurden, sowie weitere zusätzliche Information bezüglich Overviews. Die Übersichtstabellen werden sowohl in `raster_columns` als auch in `raster_overviews` registriert, da sie sowohl eigene Raster darstellen, als auch, als niedriger aufgelöstes Zerrbild einer höher aufgelösten Tabelle, einem bestimmten Zweck dienen. Wenn Sie den `-l` Switch beim Laden des Rasters angeben, werden diese gemeinsam mit der Rasterhaupttabelle erstellt; sie können aber auch händisch über AddOverviewConstraints erstellt werden.

Übersichtstabellen enthalten dieselben Constraints wie andere Rastertabellen und zusätzliche informative Constraints, spezifisch für die Übersichten.



Anmerkung
Die Information in `raster_overviews` befindet sich nicht in `raster_columns`. Falls Sie die Information der Übersichtstabelle und der `raster_columns` zusammen benötigen, können Sie einen Join auf `raster_overviews` und `raster_columns` ausführen, um die gesamte Information zu erhalten.

Die zwei Hauptgründe für Übersichtsraster sind:

1. Eine niedrig aufgelöste Darstellung der Basistabellen; wird im Allgemeinen zum schnellen Hinauszoomen verwendet.
2. Die Berechnungen laufen grundsätzlich schneller ab, als bei den Stammdaten mit höherer Auflösung, da weniger Datensätze vorhanden sind und die Pixel eine größere Fläche abdecken. Obwohl diese Berechnungen nicht so exakt sind, wie jene auf die hochauflösenden Stammtabellen, sind sie doch für viele Überschlagsrechnunen ausreichend.

Der `raster_overviews` Katalog enthält folgende Attribute an Information.

- `o_table_catalog` Die Datenbank, in der sich die Übersichtstabelle befindet. Liest immer die aktuelle Datenbank.
- `o_table_schema` Das Datenbankschema dem die Rasterübersichtstabelle angehört.
- `o_table_name` Der Tabellenname der Rasterübersicht
- `o_raster_column` das Rasterattribut in der Übersichtstabelle.
- `r_table_catalog` Die Datenbank, in der sich die Rastertabelle befindet, für die diese Übersicht gilt. Greift immer auf die aktuelle Datenbank zu.

- `r_table_schema` Das Datenbankschema, in dem sich die Rastertabelle befindet, zu der der Übersichtsdienst gehört.
- `r_table_name` Die Rastertabelle, welche von dieser Übersicht bedient wird.
- `r_raster_column` Die Rasterspalte, die diese Overviewspalte bedient.
- `overview_factor` - der Pyramidenlevel der Übersichtstabelle. Umso größer die Zahl ist, desto geringer ist die Auflösung. Wenn ein Ordner für die Bilder angegeben ist, rechnet raster2pgsql eine Übersicht für jede Bilddatei und ladet diese einzeln. Es wird Level 1 und die Ursprungsdatei angenommen. Beim Level 2 repräsentiert jede Kachel 4 Originalkacheln. Angenommen Sie haben einen Ordner mit Bilddateien in einer Auflösung von 5000x5000 Pixel, die Sie auf 125x125 große Kacheln zerlegen wollen. Für jede Bilddattei enthält die Basistabelle (5000*5000)/(125*125) Datensätze = 1600, Ihre (l=2) `o_2` Tabelle hat dann eine Obergrenze von (1600/Power(2,2)) = 400 Zeilen, Ihre (l=3) `o_3` (1600/Power(2,3)) = 200 Zeilen. Wenn sich die Pixel nicht durch die Größe Ihrer Kacheln teilen lassen, erhalten Sie einige Ausschusskacheln (Kacheln die nicht zur Gänze gefüllt sind). Beachten Sie bitte, dass jede durch raster2pgsql erzeugte Übersichtskachel dieselbe Pixelanzahl hat wie die ursprüngliche Kachel, aber eine geringere Auflösung, wo ein Pixel (Power(2,overview_factor) Pixel der Ursprungsdatei) repräsentiert.

10.4 Eigene Anwendungen mit PostGIS Raster erstellen

Die Tatsache, dass PostGIS Raster Ihnen SQL-Funktionen zum Rendern von Rastern in bekannten Bildformaten zur Verfügung stellt, gibt Ihnen eine Vielzahl von Möglichkeiten, diese zu rendern. Zum Beispiel können Sie OpenOffice / LibreOffice für das Rendering verwenden, wie in Rendering PostGIS Raster graphics with LibreOffice Base Reports gezeigt wird. Darüber hinaus können Sie eine Vielzahl von Sprachen verwenden, wie in diesem Abschnitt gezeigt wird.

10.4.1 Rendering Map Algebra and Reclass Results

ST_MapAlgebra (callback function version) and ST_Reclass can be used together to turn analysis rasters into viewable images. A common pattern is to build a derived raster with map algebra, convert numeric ranges into display classes with reclassification, and render the result with ST_AsPNG or another ST_AsGDALRaster output format.

The raster reference includes executable visual examples for these workflows:

- ST_MapAlgebraExpr compares two-raster intersection and union results.
- ST_MapAlgebra (callback function version) demonstrates neighborhood processing and renders the source and result rasters.

For a classification-style output, reclassify a continuous band into a small display palette and render it directly.

Code

```
WITH classified AS (
    SELECT ST_Reclass(
        rast,
        1,
        '[0-50]:1, (50-100]:80, (100-150]:160, (150-255]:255',
        '8BUI',
        0
    ) AS rast
    FROM my_analysis_rasters
    WHERE rid = 1
)
SELECT ST_AsPNG(rast) AS png
FROM classified;
```

10.4.2 PL/pgSQL Map Algebra Callback Example

Map algebra callbacks are ordinary PostgreSQL functions. The following example uses PL/pgSQL to scale one raster band while keeping the callback argument types compatible with the double precision[][][] value matrix passed by ST_MapAlgebra (callback function version).

Code

```
CREATE OR REPLACE FUNCTION plpgsql_scale_pixel(
    value double precision[][][],
    positions integer[][],
    VARIADIC userargs text[]
)
RETURNS double precision
AS $$
DECLARE
    pixel double precision := value[1][1][1];
    factor double precision := COALESCE(userargs[1]::double precision, 1);
BEGIN
    IF pixel IS NULL THEN
        RETURN NULL;
    END IF;

    RETURN greatest(0, least(255, pixel * factor));
END;
$$ LANGUAGE plpgsql IMMUTABLE;

SELECT ST_MapAlgebra(
    rast,
    1,
    'plpgsql_scale_pixel(double precision[], integer[], text[])'::regprocedure,
    '8BUI',
    'FIRST',
    NULL,
    0,
    0,
    '1.5'
) AS rast
FROM my_analysis_rasters
WHERE rid = 1;
```

The first callback argument is a PostgreSQL double precision[][][] value matrix. With no neighborhood distance, the current pixel value for the first raster is available as `value[1][1][1]`.

10.4.3 PHP Beispiel: Ausgabe mittels ST_AsPNG in Verbindung mit anderen Rasterfunktionen

In diesem Abschnitt zeigen wir die Anwendung des PHP PostgreSQL Treibers und der Funktion ST_AsGDALRaster, um die Bänder 1,2,3 eines Rasters an einen PHP Request-Stream zu übergeben. Dieser kann dann in einen "img src" HTML Tag eingebunden werden.

Dieses Beispiel zeigt, wie Sie ein ganzes Bündel von Rasterfunktionen kombinieren können, um jene Kacheln zu erhalten, die ein bestimmtes WGS84 Umgebungsrechteck schneiden. Anschließend werden alle Bänder dieser Kacheln mit ST_Union vereinigt, mit ST_Transform in die vom Benutzer vorgegebene Projektion transformiert und das Ergebnis mit ST_AsPNG als PNG ausgegeben.

You would call the script with `http://mywebserver/test_raster.php?srid=2249` to get the raster image in Massachusetts state plane feet.

Code

```
<?php
$conn_str = 'dbname=mydb host=localhost port=5432 user=myuser password=mypwd';
$dbconn = pg_connect($conn_str);
header('Content-Type: image/png');

if (!empty($_REQUEST['srid']) &amp;&amp; is_numeric($_REQUEST['srid'])) {
    $input_srid = intval($_REQUEST['srid']);
```

```
} else {
    $input_srid = 26986;
}

$sql = "SET bytea_output = 'escape';
SELECT ST_AsPNG(
    ST_Transform(
        ST_AddBand(
            ST_Union(rast, 1),
            ARRAY[ST_Union(rast, 2), ST_Union(rast, 3)]
        ),
        $input_srid
    )
) AS new_rast
FROM aerials.boston
WHERE ST_Intersects(
    rast,
    ST_Transform(
        ST_MakeEnvelope(-71.1217, 42.227, -71.1210, 42.218, 4326),
        26986
    )
)";

$result = pg_query($sql);
$row = pg_fetch_row($result);
pg_free_result($result);
if ($row === false) return;
echo pg_unescape_bytea($row[0]);
?>
```

10.4.4 ASP.NET C# Beispiel: Ausgabe mittels ST_AsPNG in Verbindung mit anderen Rasterfunktionen

In this section, we'll demonstrate how to use Npgsql PostgreSQL .NET driver and the ST_AsGDALRaster family of functions to output band 1,2,3 of a raster to the HTTP response stream so it can be embedded in an `img` tag.

Für dieses Beispiel benötigen Sie den npgsql .NET PostgreSQL Treiber. Um loslegen zu können, reicht es aus, dass Sie die neueste Version von http://npgsql.projects.postgresql.org/ in Ihren ASP.NET Ordner laden.

Dieses Beispiel zeigt, wie Sie ein ganzes Bündel von Rasterfunktionen kombinieren können, um jene Kacheln zu erhalten, die ein bestimmtes WGS84 Umgebungsrechteck schneiden. Anschließend werden alle Bänder dieser Kacheln mit ST_Union vereinigt, mit ST_Transform in die vom Benutzer vorgegebene Projektion transformiert und das Ergebnis mit ST_AsPNG als PNG ausgegeben.

Dasselbe Beispiel wie Abschnitt 10.4.3 nur in C# implementiert.

You would call the handler with `http://mywebserver/TestRaster.ashx?srid=2249` to get the raster image in Massachusetts state plane feet.

web.config connection string section:

Code

```
<connectionStrings>
    <add name="DSN"
        connectionString="server=localhost;database=mydb;Port=5432;User Id=myuser;password= ↩
    mypwd"/>
</connectionStrings>
```

Code for `TestRaster.ashx`:

Code

```
<%@ WebHandler Language="C#" Class="TestRaster" %>
using System;
using System.Data;
using System.Web;
using Npgsql;

public class TestRaster : IHttpHandler
{
    public void ProcessRequest(HttpContext context)
    {
        context.Response.ContentType = "image/png";
        context.Response.BinaryWrite(GetResults(context));
    }

    public bool IsReusable {
        get { return false; }
    }

    public byte[] GetResults(HttpContext context)
    {
        byte[] result = null;
        NpgsqlCommand command;
        string sql = null;
        int input_srid = 26986;
        try {
            string connectionString = System.Configuration.ConfigurationManager
                .ConnectionStrings["DSN"].ConnectionString;
            using (NpgsqlConnection conn = new NpgsqlConnection(connectionString)) {
                conn.Open();

                if (context.Request["srid"] != null)
                {
                    input_srid = Convert.ToInt32(context.Request["srid"]);
                }
                sql = @"SELECT ST_AsPNG(
                            ST_Transform(
                                ST_AddBand(
                                    ST_Union(rast, 1),
                                    ARRAY[ST_Union(rast, 2), ST_Union(rast, 3)]
                                ),
                                :input_srid
                            )
                        ) 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;
    }
}
```

10.4.5 Applikation für die Java-Konsole, welche eine Rasterabfrage als Bilddatei ausgibt

Eine einfache Java Applikation, die eine Abfrage entgegennimmt, ein Bild erzeugt und in eine bestimmte Datei ausgibt.

Sie können die neuesten PostgreSQL JDBC Treiber unter http://jdbc.postgresql.org/download.html herunterladen.

You can compile the following code with commands like:

Code

```
CLASSPATH=".:../postgresql.jar" javac SaveQueryImage.java
jar cfm SaveQueryImage.jar Manifest.txt *.class
```

And call it from the command line with something like:

Code

```
java -jar SaveQueryImage.jar \
  "SELECT ST_AsPNG(ST_AsRaster(ST_Buffer(ST_Point(1, 5), 10, 'quad_segs=2'), 150, 150, '8BUI ↩
    ', 100));" \
  "test.png"
```

`Manifest.txt`:

Code

```
Class-Path: postgresql-9.0-801.jdbc4.jar
Main-Class: SaveQueryImage
```

Code for `SaveQueryImage.java`:

Code

```
import java.sql.Connection;
import java.sql.SQLException;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.io.*;

public class SaveQueryImage {
  public static void main(String[] argv) {
      System.out.println("Checking if Driver is registered with DriverManager.");

      try {
        //java.sql.DriverManager.registerDriver (new org.postgresql.Driver());
        Class.forName("org.postgresql.Driver");
      }
      catch (ClassNotFoundException cnfe) {
```

```
        System.out.println("Couldn't find the driver!");
        cnfe.printStackTrace();
        System.exit(1);
      }

      Connection conn = null;

      try {
        conn = DriverManager.getConnection("jdbc:postgresql://localhost:5432/mydb", "myuser" ↩
    , "mypwd");
        conn.setAutoCommit(false);

        PreparedStatement sGetImg = conn.prepareStatement(argv[0]);

        ResultSet rs = sGetImg.executeQuery();

        FileOutputStream fout;
        try
        {
            rs.next();
            /** Output to file name requested by user **/
            fout = new FileOutputStream(new File(argv[1]) );
            fout.write(rs.getBytes(1));
            fout.close();
        }
        catch(Exception e)
        {
            System.out.println("Can't create file");
            e.printStackTrace();
        }

        rs.close();
        sGetImg.close();
        conn.close();
      }
      catch (SQLException se) {
        System.out.println("Couldn't connect: print out a stack trace and exit.");
        se.printStackTrace();
        System.exit(1);
      }
  }
}
```

10.4.6 Verwenden Sie PLPython um Bilder via SQL herauszuschreiben

Diese als plpython gespeicherte Prozedur erzeugt eine Datei pro Datensatz im Serververzeichnis. Benötigt die Installation von plpython. Funktioniert sowohl mit plpythonu als auch mit plpython3u.

Code

```
CREATE OR REPLACE FUNCTION write_file (param_bytes bytea, param_filepath text)
RETURNS text
AS $$
f = open(param_filepath, 'wb+')
f.write(param_bytes)
return param_filepath
$$ LANGUAGE plpythonu;
```

Write out 5 images to the PostgreSQL server in varying sizes. The PostgreSQL daemon account needs write access to the folder; this echoes back the file names created.

Code

```
SELECT write_file(
    ST_AsPNG(
        ST_AsRaster(
            ST_Buffer(ST_Point(1, 5), j * 5, 'quad_segs=2'),
            150 * j,
            150 * j,
            '8BUI',
            100
        )
    ),
    'C:/temp/slices' || j || '.png'
)
FROM generate_series(1, 5) AS j;
```

Output

```
write_file
--------------------
 C:/temp/slices1.png
 C:/temp/slices2.png
 C:/temp/slices3.png
 C:/temp/slices4.png
 C:/temp/slices5.png
```

10.4.7 Faster mit PSQL ausgeben

Sadly PSQL doesn't have easy to use built-in functionality for outputting binaries. This is a bit of a hack that piggy backs on PostgreSQL somewhat legacy large object support. To use it, first launch your **psql** command line connected to your database.

Anders als beim Ansatz mit Python, wird die Datei bei diesem Ansatz auf Ihrem lokalen Rechner erzeugt.

Code

```
SELECT
    oid,
    lowrite(lo_open(oid, 131072), png) AS num_bytes
FROM (VALUES (
    lo_create(0),
    ST_AsPNG((SELECT rast FROM aerials.boston WHERE rid = 1))
)) AS v(oid, png);
```

Output

```
oid   | num_bytes
---------+-----------
 2630819 |     74860
```

Note the large object OID and export it to a file on your local computer, replacing the example path with the desired file path.

Code

```
\lo_export 2630819 'C:/temp/aerial_samp.png'
```

This deletes the file from large object storage in the database.

Code

```
SELECT lo_unlink(2630819);
```

Kapitel 11

Referenz Raster

`raster` is a PostGIS type for storing and analyzing raster data.

Um Raster aus Rasterdateien zu laden, siehe Abschnitt 10.1

Some examples in this reference use a raster table of dummy data, created with the following code:

Code

```
CREATE TABLE dummy_rast(rid integer, rast raster);
INSERT INTO dummy_rast(rid, rast)
VALUES (1,
('01' -- little endian (uint8 ndr)
||
'0000' -- version (uint16 0)
||
'0000' -- nBands (uint16 0)
||
'0000000000000040' -- scaleX (float64 2)
||
'0000000000000840' -- scaleY (float64 3)
||
'000000000000E03F' -- ipX (float64 0.5)
||
'000000000000E03F' -- ipY (float64 0.5)
||
'0000000000000000' -- skewX (float64 0)
||
'0000000000000000' -- skewY (float64 0)
||
'00000000' -- SRID (int32 0)
||
'0A00' -- width (uint16 10)
||
'1400' -- height (uint16 20)
)::raster
),
-- Raster: 5 x 5 pixels, 3 bands, PT_8BUI pixel type, NODATA = 0
(2, ('01000003009A9999999999A93F9A9999999999A9BF000000E02B274A' ||
'4100000000771956410000000000000000000000000000000000 ↩
    FFFFFFFF050005000400FDFEFDFEFEFDFEFEFDF9FAFEF' ||
' ↩
    EFCF9FBFDFEFEFDFCFAFEFEFE04004E627AADD16076B4F9FE6370A9F5FE59637AB0E54F5861708704004656648 7A1506
    ')::raster);
```

The functions below are ones which a user of PostGIS Raster is likely to need. There are other raster support functions which are not of interest to a general user.

11.1 Datentypen zur Unterstützung von Rastern.

11.1.1 geomval

geomval — Ein räumlicher Datatyp mit zwei Feldern - geom (enthält das geometrische Element) und val (enthält den Zellwert eines Rasterbandes in Doppelter Genauigkeit).

Description

geomval ist ein a zusammengesetzter Datentyp, der aus einem geometrischen Objekt, auf welches vom Attribut ".geom" her verwiesen wird, und "val" besteht. "val" hält einen Wert in Double Precision, der dem Pixelwert an einer bestimmten geometrischen Stelle in einem Rasterband entspricht. Verwendung findet dieser Datentyp in ST_DumpAsPolygon und als Ausgabetyp in der Familie der Rasterüberlagerungsfunktionen um ein Rasterband in Polygone zu zerlegen.

Siehe auch

Abschnitt 13.6

11.1.2 addbandarg

addbandarg — Ein zusammengesetzter Datentyp, der als Eingabewert für die Funktion "ST_AddBand" verwendet wird und sowohl Attribute als auch Initialwert des neuen Bandes festlegt.

Description

Ein zusammengesetzter Datentyp, der als Eingabewert für die Funktion "ST_AddBand" verwendet wird und sowohl Attribute als auch Initialwert des neuen Bandes festlegt.

Field	Type	Description
`index`	integer	Ein von 1 aufwärts zählender Wert, der die Positionunter den Rasterbänder angibt, an der das neue Band eingefügt werden soll. Wenn er NULL ist wird das neue Band am Ende der Rasterbänder hinzugefügt.
`pixeltype`	Text	Pixel type of the new band. One of the pixel types described in ST_BandPixelType.
`initialvalue`	double precision	Initial value that all pixels of the new band will be set to.
`nodataval`	double precision	Der NODATA-Wert des neuen Bandes. Wenn NULL, dann wird für das neue Band kein NODATA-Wert vergeben.

Siehe auch

ST_AddBand

11.1.3 rastbandarg

rastbandarg — Ein zusammengesetzter Datentyp, der verwendet wird um den Raster und, über einen Index, das Band des Rasters anzugeben.

Description

Ein zusammengesetzter Datentyp, der verwendet wird um den Raster und, über einen Index, das Band des Rasters anzugeben.

Field	Type	Description
rast	raster	The raster in question.
nband	integer	1-based value indicating the raster band.

Siehe auch

ST_MapAlgebra (callback function version)

11.1.4 raster

raster — Der räumliche Datentyp Raster

Description

Raster ist ein Geodatentyp, der verwendet wird um digitale Bilder darzustellen. Diese Daten können zum Beispiel von JPEGs, TIFFs, PNGs oder digitalen Höhenmodellen stammen. Jeder Raster kann aus 1 oder mehreren Bänder bestehen, diese wiederum bestehen aus einzelnen Zellen denen Werte zugewiesen werden können. Raster können georeferenziert werden.



Anmerkung
Verlangt, dass PostGIS mit GDAL-Unterstützung kompiliert wurde. Gegenwärtig können Raster implizit in den Geometry Datentyp umgewandelt werden, allerdings gibt diese Umwandlung die ST_ConvexHull des Rasters zurück. Diese automatische Typumwandlung kann möglicherweise in der nahen Zukunft entfernt werden; verlassen Sie sich daher bitte nicht darauf.

Verhaltensweise bei der Typumwandlung

In diesem Abschnitt sind die für diesen Datentyp erlaubten impliziten und expliziten Typumwandlungen beschrieben.

Typumwandlung nach	Verhaltensweise
Geometrie	implizit

Siehe auch

Kapitel 11

11.1.5 reclassarg

reclassarg — A composite type used as input into the ST_Reclass function defining the behavior of reclassification.

Description

A composite type used as input into the ST_Reclass function defining the behavior of reclassification.

Field	Type	Description
nband	integer	Die Nummer des Bandes das neu klasssifiziert werden soll.
reclassexpr	Text	Comma-delimited range:map_range mappings that define how old band values map to new band values.
pixeltype	Text	One of defined pixel types as described in ST_BandPixelType.
nodataval	double precision	Der Zellwert, der als NODATA/NULL betrachtet werden soll. Bei der Ausgabe von Bildern, die Transparenz unterstützen, bleibt dieser leer.

The reclassexpr range delimiters mean: (greater than,) less than,] less than or equal, and [greater than or equal.

- [a-b] means a <= x <= b.
- (a-b] means a < x <= b.
- [a-b) means a <= x < b.
- (a-b) means a < x < b.

Opening-parenthesis notation is optional, so a-b means the same as (a-b).

Beispiele

Reclassify band 2 as an 8BUI where 255 is nodata value.

Code

```
SELECT ROW(2, '0-100:1-10, 101-500:11-150,501 - 10000: 151-254', '8BUI', 255)::reclassarg;
```

Reclassify band 1 as an 1BB and no nodata value defined.

Code

```
SELECT ROW(1, '0-100]:0, (100-255:1', '1BB', NULL)::reclassarg;
```

Siehe auch

ST_Reclass

11.1.6 summarystats

summarystats — Ein zusammgesetzter Datentyp, welcher von den Funktionen ST_SummaryStats und ST_SummaryStatsAgg zurückgegeben wird.

Description

Ein zusammgesetzter Datentyp, welcher von ST_SummaryStats und ST_SummaryStatsAgg zurückgegeben wird.

Zählung ganzzahlig Die Summe aller Zellen, die für eine zusammenfassende Statistik abgezählt wurden.

Summe doppelte Genauigkeit Die Summe aller abgezählten Zellwerte.

mittlere doppelte Genauigkeit Der arithmetische Mittelwert aller abgezählten Zellwerte.

stddev doppelte Genauigkeit Die Standardabweichung aller abgezählten Zellwerte.

min doppelte Genauigkeit Der Minimalwert aller abgezählten Zellwerte.

max doppelte Genauigkeit Der Maximalwert aller abgezählten Zellwerte.

Siehe auch

ST_SummaryStats, ST_SummaryStatsAgg

11.1.7 unionarg

unionarg — Ein zusammengesetzter Datentyp, der als Eingabewert für die Funktion "ST_Union" dient. Dieser Datentyp bestimmt die zu behandelnden Bänder, sowie die Verhaltensweise des UNION Operators.

Description

Ein zusammengesetzter Datentyp, der als Eingabewert für die Funktion "ST_Union" dient. Dieser Datentyp bestimmt die zu behandelnden Bänder, sowie die Verhaltensweise des UNION Operators.

Field	Type	Description
`nband`	integer	Der 1-basierte Wert, welcher das Band aller Input-Rraster kennzeichnet, die bearbeitet werden sollen.
`uniontype`	Text	Type of union operation. One of the types described in ST_Union.

Siehe auch

ST_Union

11.2 Rastermanagement

11.2.1 AddRasterConstraints

AddRasterConstraints — Fügt die Raster-Constraints zu einer bestimmten Spalte einer bereits geladenen Rastertabelle hinzu. Diese Constraints beschränken das Koordinatentransformationssystem, den Maßstab, die Blockgröße, die Ausrichtung, die Bänder, den Bandtyp und eine Flag, die anzeigt ob die Rasterspalte regelmäßig geblockt ist. Es müssen bereits Daten in die Tabelle geladen sein, damit die Constraints abgeleitet werden können. Gibt TRUE zurück, wenn das Setzen der Constraints ausgeführt wurde; bei Problemen wird eine Meldung angezeigt.

Synopsis

boolean **AddRasterConstraints**(name rasttable, name rastcolumn, boolean srid=true, boolean scale_x=true, boolean scale_y=true, boolean blocksize_x=true, boolean blocksize_y=true, boolean same_alignment=true, boolean regular_blocking=false, boolean num_bands=true , boolean pixel_types=true , boolean nodata_values=true , boolean out_db=true , boolean extent=true);
boolean **AddRasterConstraints**(name rasttable, name rastcolumn, text[] VARIADIC constraints);
boolean **AddRasterConstraints**(name rastschema, name rasttable, name rastcolumn, text[] VARIADIC constraints);
boolean **AddRasterConstraints**(name rastschema, name rasttable, name rastcolumn, boolean srid=true, boolean scale_x=true, boolean scale_y=true, boolean blocksize_x=true, boolean blocksize_y=true, boolean same_alignment=true, boolean regular_blocking=fa
boolean num_bands=true, boolean pixel_types=true, boolean nodata_values=true , boolean out_db=true , boolean extent=true);

Description

Setzt die Constraints auf eine Rasterspalte Diese werden verwendet um die Information in dem Rasterkatalog `raster_columns` anzuzeigeen. Der Parameter `rastschema` bezeichnet das Tabellenschema in dem die Tabelle liegt. Die Ganzzahl `srid` verweist auf einen Eintrag in der Tabelle "spatial_ref_sys".

Der `raster2pgsql` Lader verwendet diese Funktion um Rastertabellen zu registrieren.

Gültige Bezeichnungen für Constraints: siehe Abschnitt 10.3.1 für weitere Details.

- `blocksize` bestimmt die Datenblockgröße von X und Y
- `blocksize_x` setzt die X-Kachel (die Breite der Kacheln in Pixel)
- `blocksize_y` setzt die Y-Kachel (die Höhe der Kacheln in Pixel)
- `extent` berechnet die räumliche Ausdehnung der ganzen Tabelle und setzt einen Constraint, der alle Raster auf diesen Ausschnitt beschränkt.
- `num_bands` Anzahl der Bänder
- `pixel_types` liest ein Feld mit Pixeltypen für jedes Band ein, und stellt sicher, dass alle Bänder denselben Pixeltyp haben
- `regular_blocking` setzt die Constraints für die räumliche Eindeutigkeit (keine zwei Raster dürfen räumlich ident sein) und für die Coverage-Kachel (der Raster wird an einem Coverage ausgerichtet)
- `same_alignment` stellt sicher, dass alle die selbe Ausrichtung haben, d.h. der Vergleich von zwei beliebigen Kacheln gibt TRUE zurück. Siehe ST_SameAlignment.
- `srid` stellt sicher, dass alle die selbe SRID aufweisen
- Mehr -- alles was als Eingabe in die obere Funktion aufgeführt ist



Anmerkung
Diese Funktion leitet die Constraints von den Daten ab, die bereits in der Tabelle vorliegen. Damit Sie die Funktion anwenden können, müssen Sie zuerst die Rasterspalte erzeugen in die Sie anschließend die Daten laden.



Anmerkung
Falls Sie weitere Daten in Ihre Tabellen laden müssen, nachdem Sie bereits die Constraints gesetzt haben, können Sie DropRasterConstraints ausführen, wenn sich die räumliche Ausdehnung Ihrer Daten geändert hat.

Verfügbarkeit: 2.0.0

Beispiele

Apply all possible constraints on column based on data.

Verify the raster table is registered correctly in the raster_columns view.

Code

```
CREATE TABLE myrasters(rid SERIAL primary key, rast raster);
INSERT INTO myrasters(rast)
SELECT ST_AddBand(ST_MakeEmptyRaster(1000, 1000, 0.3, -0.3, 2, 2, 0, 0, 4326), 1, '8BSI':: ↩
    text, -129, NULL);

SELECT AddRasterConstraints('myrasters'::name, 'rast'::name);
SELECT srid, scale_x, scale_y, blocksize_x, blocksize_y, num_bands, pixel_types,  ↩
    nodata_values
    FROM raster_columns
    WHERE r_table_name = 'myrasters';
```

Output

```
srid | scale_x | scale_y | blocksize_x | blocksize_y | num_bands | pixel_types| nodata_values
------+---------+---------+-------------+-------------+-----------+-------------+---------------
 4326 |       2 |       2 |        1000 |        1000 |         1 | {8BSI}      | {0}
```

Apply single constraint.

Code

```
CREATE TABLE public.myrasters2(rid SERIAL primary key, rast raster);
INSERT INTO myrasters2(rast)
SELECT ST_AddBand(ST_MakeEmptyRaster(1000, 1000, 0.3, -0.3, 2, 2, 0, 0, 4326), 1, '8BSI':: ↩
    text, -129, NULL);

SELECT AddRasterConstraints('public'::name, 'myrasters2'::name, 'rast'::name, ' ↩
    regular_blocking', 'blocksize');
```

Output

```
NOTICE:  Adding regular blocking constraint
NOTICE:  Adding blocksize-X constraint
NOTICE:  Adding blocksize-Y constraint
```

Siehe auch

Abschnitt 10.3.1, ST_AddBand, ST_MakeEmptyRaster, DropRasterConstraints, ST_BandPixelType, ST_SRID

11.2.2 DropRasterConstraints

DropRasterConstraints — Löscht die Constraints eines PostGIS Rasters die sich auf eine Rastertabellenspalte beziehen. Nützlich um Daten erneut zu laden oder um die Daten einer Rasterspalte zu aktualisieren.

Synopsis

boolean **DropRasterConstraints**(name rasttable, name rastcolumn, boolean srid, boolean scale_x, boolean scale_y, boolean blocksize_x, boolean blocksize_y, boolean same_alignment, boolean regular_blocking, boolean num_bands=true, boolean pixel_types=true, boolean nodata_values=true, boolean out_db=true , boolean extent=true);
boolean **DropRasterConstraints**(name rastschema, name rasttable, name rastcolumn, boolean srid=true, boolean scale_x=true, boolean scale_y=true, boolean blocksize_x=true, boolean blocksize_y=true, boolean same_alignment=true, boolean regular_blocking=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

Löscht die Constraints eines PostGIS Rasters die sich auf eine Rastertabellenspalte beziehen und mit AddRasterConstraints hinzugefügt wurden. Nützlich um weitere Daten zu laden oder um die Daten einer Rasterspalte zu aktualisieren. Dies ist nicht notwendig, wenn Sie eine Rastertabelle oder eine Rasterspalte löschen wollen.

To drop a raster table use standard SQL: `DROP TABLE mytable`.

To drop just a raster column and leave the rest of the table, use standard SQL: `ALTER TABLE mytable DROP COLUMN rast`.

Die Tabelle verschwindet aus dem `raster_columns` Katalog, wenn die Tabelle oder die Spalte gelöscht wurde. Wenn allerdings nur die Constraints gelöscht werden, so wird die Rasterspalte weiterhin im `raster_columns` Katalog aufgeführt, aber es ist keine zusätzliche Information mehr vorhanden - abgesehen vom Spalten- und Tabellennamen.

Verfügbarkeit: 2.0.0

Beispiele

Code

```
SELECT DropRasterConstraints ('myrasters', 'rast');
```

Output

```
droprasterconstraints
----------------------
 t
(1 row)
```

Verify change in raster_columns.

Code

```
SELECT srid, scale_x, scale_y, blocksize_x, blocksize_y, num_bands, pixel_types,  ↩
    nodata_values
    FROM raster_columns
    WHERE r_table_name = 'myrasters';
```

Output

```
srid | scale_x | scale_y | blocksize_x | blocksize_y | num_bands | pixel_types| nodata_values
------+---------+---------+-------------+-------------+-----------+-------------+---------------
    0 |         |         |             |             |           |             |
```

Siehe auch

AddRasterConstraints

11.2.3 AddOverviewConstraints

AddOverviewConstraints — Eine Rasterspalte als Übersicht für eine andere Rasterspalte kennzeichnen.

Synopsis

boolean **AddOverviewConstraints**(name ovschema, name ovtable, name ovcolumn, name refschema, name reftable, name refcolumn, int ovfactor);
boolean **AddOverviewConstraints**(name ovtable, name ovcolumn, name reftable, name refcolumn, int ovfactor);

Description

Fügt Constraints zu der Rasterspalte hinzu. Diese werden verwendet um die Information im `raster_overviews` Katalog anzuzeigen.

Der Parameter `ovfactorr` gibt den Multiplikator für den Maßstab der Übersichtsspalte an: ein höherer "overview factor" bedingt eine niedrigere Auflösung.

Falls die Parameter `ovschema` und `refschema` weggelassen werden, so wird die erste so benannte Tabelle verwendet, die beim Durchlaufen des `search_path` gefunden wird.

Verfügbarkeit: 2.0.0

Beispiele

Verify the raster table is registered correctly in the raster_overviews view.

Code

```
CREATE TABLE res1 AS SELECT
ST_AddBand(ST_MakeEmptyRaster(1000, 1000, 0, 0, 2),
  1, '8BSI'::text, -129, NULL
) r1;

CREATE TABLE res2 AS SELECT
ST_AddBand(ST_MakeEmptyRaster(500, 500, 0, 0, 4),
  1, '8BSI'::text, -129, NULL
) r2;

SELECT AddOverviewConstraints('res2', 'r2', 'res1', 'r1', 2);
SELECT o_table_name ot, o_raster_column oc,
       r_table_name rt, r_raster_column rc,
       overview_factor f
FROM raster_overviews WHERE o_table_name = 'res2';
```

Output

```
ot  | oc |  rt  | rc | f
------+----+------+----+---
 res2 | r2 | res1 | r1 | 2
(1 row)
```

Siehe auch

Abschnitt 10.3.2, DropOverviewConstraints, ST_CreateOverview, AddRasterConstraints

11.2.4 DropOverviewConstraints

DropOverviewConstraints — Löscht die Markierung einer Rasterspalte, die festlegt dass sie als Übersicht für eine andere Spalte dient.

Synopsis

boolean **DropOverviewConstraints**(name ovschema, name ovtable, name ovcolumn);
boolean **DropOverviewConstraints**(name ovtable, name ovcolumn);

Description

Jene Constraints einer Rasterspalte löschen, die sie als Übersicht einer anderen Rasterspalte in dem Rasterkatalog `raster_overviews` ausweisen.

Falls der Parameter `ovschema` weggelassen wird, so wird die erste so benannte Tabelle verwendet, die beim Durchlaufen des `search_path` gefunden wird.

Verfügbarkeit: 2.0.0

Siehe auch

Abschnitt 10.3.2, AddOverviewConstraints, DropRasterConstraints

11.2.5 PostGIS_GDAL_Version

PostGIS_GDAL_Version — Gibt die von PostGIS verwendete Version der GDAL-Bibliothek aus.

Synopsis

text **PostGIS_GDAL_Version**();

Description

Gibt die von PostGIS verwendete Version der GDAL-Bibliothek aus. Überprüft und meldet, ob GDAL die zugehörigen Dateien findet.

Beispiele

Code

```
SELECT PostGIS_GDAL_Version();
```

Output

```
GDAL 1.11dev, released 2013/04/13
```

Siehe auch

postgis.gdal_datapath

11.2.6 PostGIS_Raster_Lib_Build_Date

PostGIS_Raster_Lib_Build_Date — Gibt einen vollständigen Bericht aus, wann die Rasterbibliothek kompiliert wurde.

Synopsis

text **PostGIS_Raster_Lib_Build_Date**();

Description

Gibt einen Bericht aus, wann die Rasterbibliothek kompiliert wurde.

Beispiele

Code

```
SELECT PostGIS_Raster_Lib_Build_Date();
```

Output

```
2010-04-28 21:15:10
```

Siehe auch

PostGIS_Raster_Lib_Version

11.2.7 PostGIS_Raster_Lib_Version

PostGIS_Raster_Lib_Version — Gibt einen vollständigen Bericht über die Version und das Kompilationsdatum der Rasterbibliothek aus.

Synopsis

text **PostGIS_Raster_Lib_Version**();

Description

Gibt einen vollständigen Bericht über die Version und das Kompilationsdatum der Rasterbibliothek aus.

Beispiele

Code

```
SELECT PostGIS_Raster_Lib_Version();
```

Output

```
2.0.0
```

Siehe auch

PostGIS_Lib_Version

11.2.8 ST_GDALDrivers

ST_GDALDrivers — Gibt eine Liste der Rasterformate aus, die von PostGIS über die Bibliothek GDAL unterstützt werden. Nur die Formate mit can_write=True können von ST_AsGDALRaster verwendet werden.

Synopsis

setof record **ST_GDALDrivers**(integer OUT idx, text OUT short_name, text OUT long_name, text OUT can_read, text OUT can_write, text OUT create_options);

Description

Gibt eine Liste der Rasterformate aus - short_name, long_name und die Optionen des Urhebers - die von GDAL unterstützt werden. Verwenden Sie den "short_name" als Übergabewert für den Parameter `format` vonST_AsGDALRaster. Die Optionen variieren, je nach dem mit welchen Treibern Ihre "libgdal" kompiliert wurde. `create_options`gibt einen XML-formatierten Satz an CreationOptionList/Option zurück, der aus dem Namen und optional aus `type`, `description`und `VALUE` für jede Option eines bestimmten Treibers besteht.

Änderung: 2.5.0 - die Spalten can_read und can_write hinzugefügt.

Änderung: 2.0.6, 2.1.3 - standardmäßig ist kein Treiber aktiviert, solange die GUC oder die Umgebungsvariable "gdal_enabled_drivers" nicht gesetzt sind.

Verfügbarkeit: 2.0.0 - GDAL >= 1.6.0.

Beispiele

List of Drivers.

Code

```
SET postgis.gdal_enabled_drivers = 'ENABLE_ALL';
SELECT short_name, long_name, can_write
FROM ST_GDALDrivers()
ORDER BY short_name;
```

Output

```
short_name    |                          long_name                          | can_write
--------------+-------------------------------------------------------------+-----------
 AAIGrid      | Arc/Info ASCII Grid                                         | t
 ACE2         | ACE2                                                        | f
 ADRG         | ARC Digitized Raster Graphics                               | f
 AIG          | Arc/Info Binary Grid                                        | f
 AirSAR       | AirSAR Polarimetric Image                                   | f
 ARG          | Azavea Raster Grid format                                   | t
 BAG          | Bathymetry Attributed Grid                                  | f
 BIGGIF       | Graphics Interchange Format (.gif)                          | f
 BLX          | Magellan topo (.blx)                                        | t
 BMP          | MS Windows Device Independent Bitmap                        | f
 BSB          | Maptech BSB Nautical Charts                                 | f
 PAux         | PCI .aux Labelled                                           | f
 PCIDSK       | PCIDSK Database File                                        | f
 PCRaster     | PCRaster Raster File                                        | f
 PDF          | Geospatial PDF                                              | f
 PDS          | NASA Planetary Data System                                  | f
 PDS4         | NASA Planetary Data System 4                                | t
 PLMOSAIC     | Planet Labs Mosaics API                                     | f
 PLSCENES     | Planet Labs Scenes API                                      | f
 PNG          | Portable Network Graphics                                   | t
 PNM          | Portable Pixmap Format (netpbm)                             | f
 PRF          | Racurs PHOTOMOD PRF                                         | f
 R            | R Object Data Store                                         | t
 Rasterlite   | Rasterlite                                                  | t
 RDA          | DigitalGlobe Raster Data Access driver                      | f
 RIK          | Swedish Grid RIK (.rik)                                     | f
 RMF          | Raster Matrix Format                                        | f
 ROI_PAC      | ROI_PAC raster                                              | f
 RPFTOC       | Raster Product Format TOC format                            | f
 RRASTER      | R Raster                                                    | f
 RS2          | RadarSat 2 XML Product                                      | f
 RST          | Idrisi Raster A.1                                           | t
 SAFE         | Sentinel-1 SAR SAFE Product                                 | f
 SAGA         | SAGA GIS Binary Grid (.sdat, .sg-grd-z)                     | t
 SAR_CEOS     | CEOS SAR Image                                              | f
 SDTS         | SDTS Raster                                                 | f
 SENTINEL2    | Sentinel 2                                                  | f
 SGI          | SGI Image File Format 1.0                                   | f
 SNODAS       | Snow Data Assimilation System                               | f
 SRP          | Standard Raster Product (ASRP/USRP)                         | f
 SRTMHGT      | SRTMHGT File Format                                         | t
 Terragen     | Terragen heightfield                                        | f
 TIL          | EarthWatch .TIL                                             | f
 TSX          | TerraSAR-X Product                                          | f
```

```
 USGSDEM         | USGS Optional ASCII DEM (and CDED)                          | t
 VICAR           | MIPL VICAR file                                             | f
 VRT             | Virtual Raster                                              | t
 WCS             | OGC Web Coverage Service                                    | f
 WMS             | OGC Web Map Service                                         | t
 WMTS            | OGC Web Map Tile Service                                    | t
 XPM             | X11 PixMap Format                                           | t
 XYZ             | ASCII Gridded XYZ                                           | t
 ZMap            | ZMap Plus Grid                                              | t
```

List of options for each driver.

Output the create options XML column of JPEG as a table. These creator options can be used in the ST_AsGDALRaster options argument.

Code

```
SELECT (xpath('@name', g.opt))[1]::text As oname,
       (xpath('@type', g.opt))[1]::text As otype,
       (xpath('@description', g.opt))[1]::text As descrip
FROM (SELECT unnest(xpath('/CreationOptionList/Option', create_options::xml)) As opt
FROM  ST_GDALDrivers()
WHERE short_name = 'JPEG') As g;
```

Output

```
oname        |  otype  |      descrip
-------------------+---------+--------------------------------------------------
 PROGRESSIVE       | boolean | whether to generate a progressive JPEG
 QUALITY           | int     | good=100, bad=0, default=75
 WORLDFILE         | boolean | whether to generate a worldfile
 INTERNAL_MASK     | boolean | whether to generate a validity mask
 COMMENT           | string  | Comment
 SOURCE_ICC_PROFILE | string  | ICC profile encoded in Base64
 EXIF_THUMBNAIL    | boolean | whether to generate an EXIF thumbnail(overview).
                             By default its max dimension will be 128
 THUMBNAIL_WIDTH   | int     | Forced thumbnail width
 THUMBNAIL_HEIGHT  | int     | Forced thumbnail height
(9 rows)
```

This example turns a few common GeoTIFF creator options into a compact table.

Code

```
WITH gtiff_options AS (
    SELECT
        option_name,
        option_type,
        default_value
    FROM ST_GDALDrivers() AS driver
    CROSS JOIN LATERAL XMLTABLE(
        '/CreationOptionList/Option'
        PASSING (driver.create_options::xml)
        COLUMNS
            option_name text PATH '@name',
            option_type text PATH '@type',
            default_value text PATH '@default'
    ) AS options
    WHERE driver.short_name = 'GTiff'
)
SELECT option_name AS name, option_type AS type, coalesce(default_value, '') AS ↩
    default_value
FROM gtiff_options
WHERE option_name = ANY (ARRAY[
    'COMPRESS',
    'INTERLEAVE',
    'TILED',
    'BLOCKXSIZE',
```

```
    'BLOCKYSIZE',
    'JPEG_QUALITY'
])
ORDER BY 1;
```

Output

```
name      |     type      | default_value
--------------+---------------+---------------
 BLOCKXSIZE    | int           |
 BLOCKYSIZE    | int           |
 COMPRESS      | string-select |
 INTERLEAVE    | string-select | PIXEL
 JPEG_QUALITY  | int           | 75
 TILED         | boolean       |
(6 rows)
```

Use XMLTABLE to expose selected GTiff creation options as rows. The same pattern can be used without the name filter to inspect the full XML document.

Code

```
SET postgis.gdal_enabled_drivers = 'ENABLE_ALL';

SELECT option_name, option_type
FROM ST_GDALDrivers() AS driver
CROSS JOIN LATERAL XMLTABLE(
    '/CreationOptionList/Option'
    PASSING (driver.create_options::xml)
    COLUMNS
        option_name text PATH '@name',
        option_type text PATH '@type'
) AS options
WHERE driver.short_name = 'GTiff'
  AND option_name IN ('COMPRESS', 'TILED', 'BIGTIFF')
ORDER BY option_name;
```

Output

```
option_name |  option_type
------------+---------------
 BIGTIFF     | string-select
 COMPRESS    | string-select
 TILED       | boolean
(3 rows)
```

Siehe auch

ST_AsGDALRaster, ST_SRID, postgis.gdal_enabled_drivers

11.2.9 UpdateRasterSRID

UpdateRasterSRID — Änderung der SRID aller Raster in der vom Anwender angegebenen Spalte und Tabelle.

Synopsis

raster **UpdateRasterSRID**(name schema_name, name table_name, name column_name, integer new_srid);
raster **UpdateRasterSRID**(name table_name, name column_name, integer new_srid);

Description

Änderung der SRID aller Raster in der vom Anwender angegebenen Spalte und Tabelle. Diese Funktion löscht alle entsprechenden Constraints (extent, alignment und die SRID) der Spalte bevor die SRID der Rasterspalte geändert wird.



Anmerkung
Die Daten des Rasters (Pixelwerte der Bänder) bleiben von dieser Funktion unangetastet. Es werden lediglich die Metadaten des Rasters geändert.

Verfügbarkeit: 2.1.0

Siehe auch

UpdateGeometrySRID

11.2.10 ST_CreateOverview

ST_CreateOverview — Erzeugt eine Version des gegebenen Raster-Coverage mit geringerer Auflösung.

Synopsis

regclass **ST_CreateOverview**(regclass tab, name col, int factor, text algo='NearestNeighbor');

Description

Erzeugt eine Übersichtstabelle mit skalierten Kacheln der Ursprungstabelle. Die Ausgabekacheln haben dieselbe Größe und haben die gleiche räumliche Ausdehnung wie die Eingabekacheln mit einer niedrigeren Auflösung (die Pixelgröße ist in beiden Richtungen 1/`factor` der Ursprünglichen).

Auf die Übersichtstabelle werden die Constraints gesetzt und sie steht über den Katalog `raster_overviews` zur Verfügung.

Die Optionen für den Algorithmus sind: 'NearestNeighbor', 'Bilinear', 'Cubic', 'CubicSpline' und 'Lanczos'. Siehe GDAL Warp resampling methods für weitere Details.

Verfügbarkeit: 2.2.0

Beispiel

Ausgabe mit besserer Qualität, aber langsamerer Formaterstellung

Code

```
SELECT ST_CreateOverview('mydata.mytable'::regclass, 'rast', 2, 'Lanczos');
```

Schnellere Berechnung der Ausgabe mittels "Nearest Neighbor" (Standardeinstellung)

Code

```
SELECT ST_CreateOverview('mydata.mytable'::regclass, 'rast', 2);
```

Siehe auch

ST_Retile, AddOverviewConstraints, AddRasterConstraints, Abschnitt 10.3.2

11.3 Raster Constructors

11.3.1 ST_AddBand

ST_AddBand — Gibt einen Raster mit den neu hinzugefügten Band(Bändern) aus. Der Typ , der Ausgangswert und der Index für den Speicherort des Bandes kann angegeben werden. Wenn kein Index angegeben ist, wird das Band am Ende hinzugefügt.

Synopsis

(1) raster **ST_AddBand**(raster rast, addbandarg[] addbandargset);
(2) raster **ST_AddBand**(raster rast, integer index, text pixeltype, double precision initialvalue=0, double precision nodataval=NULL);
(3) raster **ST_AddBand**(raster rast, text pixeltype, double precision initialvalue=0, double precision nodataval=NULL);
(4) raster **ST_AddBand**(raster torast, raster fromrast, integer fromband=1, integer torastindex=at_end);
(5) raster **ST_AddBand**(raster torast, raster[] fromrasts, integer fromband=1, integer torastindex=at_end);
(6) raster **ST_AddBand**(raster rast, integer index, text outdbfile, integer[] outdbindex, double precision nodataval=NULL);
(7) raster **ST_AddBand**(raster rast, text outdbfile, integer[] outdbindex, integer index=at_end, double precision nodataval=NULL);

Description

Gibt ein Raster zurück, dem an der angegebenen Position (Index) ein neues Band mit dem angegebenen Typ, dem angegebenen Anfangswert und dem angegebenen Nodata-Wert hinzugefügt wurde. Wenn kein Index angegeben ist, wird das Band am Ende hinzugefügt. Wenn kein `vonband` angegeben ist, wird Band 1 angenommen. Der Pixeltyp ist eine String-Darstellung eines der in ST_BandPixelType angegebenen Pixeltypen. Wird ein vorhandener Index angegeben, werden alle nachfolgenden Bänder >= dieser Index um 1 erhöht. Wird ein Anfangswert angegeben, der größer ist als der Maximalwert des Pixeltyps, wird der Anfangswert auf den höchsten vom Pixeltyp zugelassenen Wert gesetzt.

Bei der Variante, welche ein Feld von addbandarg entgegennimmt (Variante 1), ist ein bestimmter Indexwert von "addbandarg" auf den Raster zu dem Zeitpunkt bezogen, als das Band mit diesem "addbandarg" zum Raster hinzugefügt wurde. Siehe das Beispiel "Mehrere neue Bänder" unterhalb.

Bei der Variante, die ein Feld an Rastern (Variante 5) entgegennimmt, wird wenn `torast` NULL ist, das `fromband` Band eines jeden Rasters in diesem Feld, in einem neuen Raster akkumuliert.

Bei den Varianten, die ein `outdbfile` (Varianten 6 und 7) entgegennehmen, muss der Wert den vollständigen Pfad der Rasterdatei enthalten. Die Datei muss auch für die PostgreSQL-Instanz zugänglich sein.

Erweiterung: 2.1.0 - Unterstützung für "addbandarg" hinzugefügt.

Erweiterung: 2.1.0 Unterstützung für die neuen "out-db" Bänder hinzugefügt.

Beispiele

Single New Band.

This example adds another 8-bit unsigned integer band with pixels initialized to 200.

Code

```
UPDATE dummy_rast
    SET rast = ST_AddBand(rast, '8BUI'::text, 200)
WHERE rid = 1;
```

Create an empty raster 100x100 units, with upper left right at 0, add 2 bands (band 1 is 0/1 boolean bit switch, band2 allows values 0-15) Uses addbandargs.

Output metadata for the raster bands to verify they are correct.

Code

```
INSERT INTO dummy_rast(rid, rast)
    VALUES(10, ST_AddBand(ST_MakeEmptyRaster(100, 100, 0, 0, 1, -1, 0, 0, 0),
    ARRAY[
        ROW(1, '1BB'::text, 0, NULL),
        ROW(2, '4BUI'::text, 0, NULL)
            ]::addbandarg[]
     )
    );

SELECT  (bmd).*
FROM (SELECT ST_BandMetaData(rast, generate_series(1, 2)) As bmd
    FROM dummy_rast WHERE rid = 10) AS foo;
```

Output

```
pixeltype | nodatavalue | isoutdb | path
----------+-------------+---------+------
 1BB      |             | f       |
 4BUI     |             | f       |
```

This example outputs metadata for the raster.

Code

```
SELECT  (rmd).width, (rmd).height, (rmd).numbands
FROM (SELECT ST_MetaData(rast) As rmd
    FROM dummy_rast WHERE rid = 10) AS foo;
```

Output

```
width | height | numbands
------+--------+----------
   100 |    100 |        2
```

Multiple New Bands.

Code

```
SELECT
    *
FROM ST_BandMetaData(ST_AddBand(ST_MakeEmptyRaster(10, 10, 0, 0, 1, -1, 0, 0, 0),
        ARRAY[
            ROW(NULL, '8BUI', 255, 0),
            ROW(NULL, '16BUI', 1, 2),
            ROW(2, '32BUI', 100, 12),
            ROW(2, '32BF', 3.14, -1)
        ]::addbandarg[]
    ),
    ARRAY[]::integer[]
);
```

Output

```
bandnum | pixeltype | nodatavalue | isoutdb | path | outdbbandnum | filesize | filetimestamp
--------+-----------+-------------+---------+------+--------------+----------+---------------
      1 | 8BUI      |           0 | f       | null |         null |     null |          null
      2 | 32BF      |          -1 | f       | null |         null |     null |          null
      3 | 32BUI     |          12 | f       | null |         null |     null |          null
      4 | 16BUI     |           2 | f       | null |         null |     null |          null
```

Aggregate the 1st band of a table of like rasters into a single raster. With as many bands as there are test_types and as many rows (new rasters) as there are mice. Note that The ORDER BY test_type clause relies on aggregate ORDER BY support. The resulting raster will have a band for each test_type alphabetical by test_type. For mouse lovers: No mice were harmed in this exercise.

Code

```
SELECT
    mouse,
    ST_AddBand(NULL, array_agg(rast ORDER BY test_type), 1) As rast
FROM mice_studies
GROUP BY mouse;
```

New Out-db band.

Code

```
SELECT
    *
FROM ST_BandMetaData(ST_AddBand(ST_MakeEmptyRaster(10, 10, 0, 0, 1, -1, 0, 0, 0),
        '/home/raster/mytestraster.tif'::text, NULL::int[]
    ),
    ARRAY[]::integer[]
);
```

Output

```
bandnum | pixeltype | nodatavalue | isoutdb | path
---------+-----------+-------------+---------+------
       1 | 8BUI      |             | t       | /home/raster/mytestraster.tif
       2 | 8BUI      |             | t       | /home/raster/mytestraster.tif
       3 | 8BUI      |             | t       | /home/raster/mytestraster.tif
```

Siehe auch

ST_BandMetaData, ST_BandPixelType, ST_MakeEmptyRaster, ST_MetaData, ST_NumBands, ST_Reclass

11.3.2 ST_AsRaster

ST_AsRaster — Konvertiert den geometrischen Datentyp von PostGIS in einen PostGIS Raster.

Synopsis

raster **ST_AsRaster**(geometry geom, raster ref, text pixeltype, double precision value=1, double precision nodataval=0, boolean touched=false);
raster **ST_AsRaster**(geometry geom, raster ref, text[] pixeltype=ARRAY['8BUI'], double precision[] value=ARRAY[1], double precision[] nodataval=ARRAY[0], boolean touched=false);
raster **ST_AsRaster**(geometry geom, double precision scalex, double precision scaley, double precision gridx, double precision gridy, text pixeltype, double precision value=1, double precision nodataval=0, double precision skewx=0, double precision skewy=0, boolean touched=false);
raster **ST_AsRaster**(geometry geom, double precision scalex, double precision scaley, double precision gridx=NULL, double precision gridy=NULL, text[] pixeltype=ARRAY['8BUI'], double precision[] value=ARRAY[1], double precision[] nodataval=ARRAY[0], double precision skewx=0, double precision skewy=0, boolean touched=false);
raster **ST_AsRaster**(geometry geom, double precision scalex, double precision scaley, text pixeltype, double precision value=1, double precision nodataval=0, double precision upperleftx=NULL, double precision upperlefty=NULL, double precision skewx=0, double precision skewy=0, boolean touched=false);
raster **ST_AsRaster**(geometry geom, double precision scalex, double precision scaley, text[] pixeltype, double precision[] value=ARRAY[1], double precision[] nodataval=ARRAY[0], double precision upperleftx=NULL, double precision upperlefty=NULL, double precision skewx=0, double precision skewy=0, boolean touched=false);
raster **ST_AsRaster**(geometry geom, integer width, integer height, double precision gridx, double precision gridy, text pixeltype, double precision value=1, double precision nodataval=0, double precision skewx=0, double precision skewy=0, boolean touched=false);

raster **ST_AsRaster**(geometry geom, integer width, integer height, double precision gridx=NULL, double precision gridy=NULL, text[] pixeltype=ARRAY['8BUI'], double precision[] value=ARRAY[1], double precision[] nodataval=ARRAY[0], double precision skewx=0, double precision skewy=0, boolean touched=false);
raster **ST_AsRaster**(geometry geom, integer width, integer height, text pixeltype, double precision value=1, double precision nodataval=0, double precision upperleftx=NULL, double precision upperlefty=NULL, double precision skewx=0, double precision skewy=0, boolean touched=false);
raster **ST_AsRaster**(geometry geom, integer width, integer height, text[] pixeltype, double precision[] value=ARRAY[1], double precision[] nodataval=ARRAY[0], double precision upperleftx=NULL, double precision upperlefty=NULL, double precision skewx=0, double precision skewy=0, boolean touched=false);

Description

Konvertiert den geometrischen Datentyp von PostGIS in einen PostGIS Raster. Es gibt viele Varianten, die jeweils drei Gruppen von Möglichkeiten bieten um die Ausrichtung/alignment und die Pixelgröße des resultierenden Rasters zu bestimmen.

Die erste Gruppe besteht aus den ersten zwei Varianten und erzeugt einen Raster mit derselben Ausrichtung (`scalex`, `scaley`, `gridx` und `gridy`), dem selben Pixeltyp und NODATA Wert wie der gegebene Referenzraster. Üblicherweise wird der Referenzraster über einen Join übergeben, der aus der Tabelle mit der Geometrie und der Tabelle mit dem Referenzraster gebildet wird.

Die zweite Gruppe setzt sich aus vier Varianten zusammen und erlaubt Ihnen, die Dimensionen des Rasters über die Parameter der Pixelgröße festzulegen (`scalex` & `scaley` und `skewx` & `skewy`). Die `width` & `height` des resultierenden Rasters wird an die Ausdehnung der Geometrie angepasst. In den meisten Fällen müssen Sie eine Typumwandlung der Eingabewerte `scalex` & `scaley` von Integer nach Double Precision vornehmen, damit PostgreSQL die richtige Variante auswählt.

Die zweite Gruppe setzt sich aus vier Varianten zusammen und erlaubt Ihnen, die Dimensionen des Rasters über die Dimensionen des Rasters zu fixieren (`width` & `height`). Die Parameter der Pixelgröße (`scalex` & `scaley` und `skewx` & `skewy`). des resultierenden Rasters werden an die Ausdehnung der Geometrie angepasst.

Die ersten zwei Varianten der beiden letzten Gruppen erlauben es Ihnen, an einer beliebigen Ecke des Führungsgitters (`gridx`& `gridy`) auszurichten; und die letzten zwei Varianten nehmen die obere linke Ecke (`upperleftx` & `upperlefty`) entgegen.

Jede Variantengruppe erlaubt die Erstellung eines Rasters sowohl mit einem als auch mit mehreren Bändern. Um einen Raster mit mehreren Bändern zu erstellen, können Sie ein Feld mit Pixeltypen (`pixeltype[]`), ein Feld mit Ausgangswerten (`value`) und ein Feld mit NODATA Werten (`nodataval`) bereitstellen. Falls nicht, werden die Standardwerte - 8BUI für den Pixeltyp, 1 für den Ausgangswert und 0 für NODATA - angenommen.

Der Ausgaberaster hat dieselbe Koordinatenreferenz wie die Ausgangsgeometrie. Die einzige Ausnahme bilden Varianten mit einem Referenzraster. In diesem Fall erhält der resultierende Raster dieselbe SRID wie der Referenzraster.

Der optionale Parameter `touched` ist standardmäßig FALSE und wird auf die GDAL Rasterungsoption ALL_TOUCHED abgebildet, welche bestimmt, ob Pixel die von Linien oder Polygonen nur berührt werden, ebenfalls gerastert werden sollen; und nicht nur die Pixel auf einem Linienzug oder mit dem Mittelpunkt innerhalb des Polygons.

Dies ist insbesondere in Verbindung mit ST_AsPNG und der Funktionsfamilie ST_AsGDALRaster, für die Erstellung von JPEGs oder PNGs aus einer Geometrie direkt von der Datenbank heraus, nützlich.

Verfügbarkeit: 2.0.0 - GDAL >= 1.6.0.

Curved geometry inputs are linearized before rasterization, including NURBSCurve inputs handled by PostGIS. TIN and PolyhedralSurface inputs are not supported.

Beispiele: Ausgabe der Geometrien als PNG-Datei

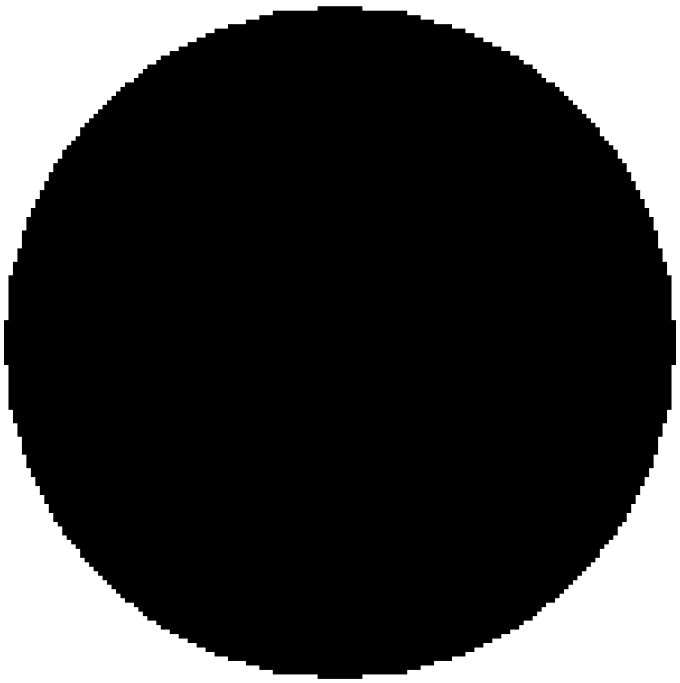
This example outputs a black circle on a white background, occupying 150 by 150 pixels.

Code

```
SELECT ST_AsPNG(
    ST_AsRaster(
        ST_Buffer(ST_Point(1, 5), 10),
        150, 150,
```

```
        ARRAY['8BUI', '8BUI', '8BUI'],
        ARRAY[0, 0, 0],
        ARRAY[255, 255, 255]
    )
) AS image;
```

Figure



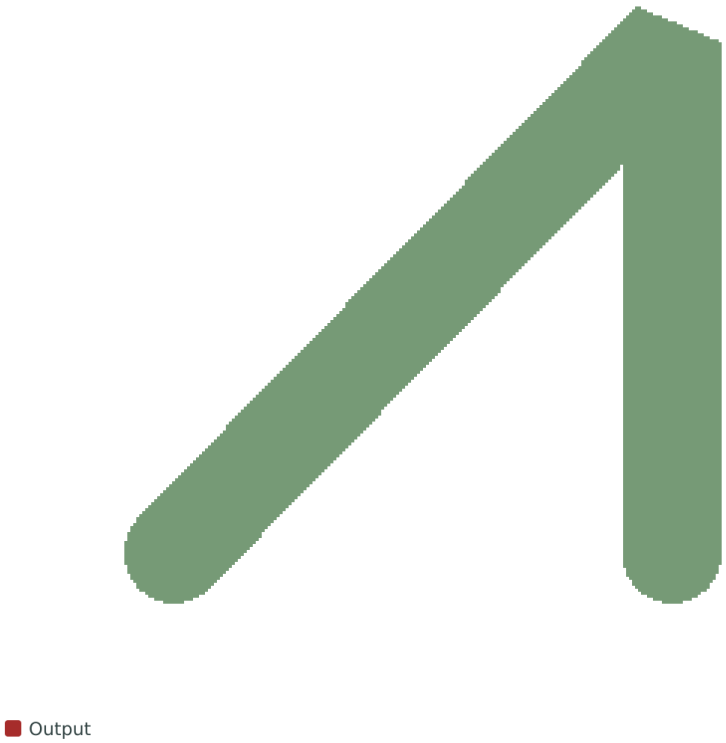
Output

This example maps bands to RGB channels, producing the teal value (118,154,118).

Code

```
SELECT ST_AsPNG(
    ST_AsRaster(
        ST_Buffer('LINESTRING(50 50,150 150,150 50)'::geometry, 10, 'join=bevel'),
        200, 200,
        ARRAY['8BUI', '8BUI', '8BUI'],
        ARRAY[118, 154, 118],
        ARRAY[255, 255, 255]
    )
) AS image;
```

Figure



Siehe auch

ST_BandPixelType, ST_Buffer, ST_GDALDrivers, ST_AsGDALRaster, ST_AsPNG, ST_AsJPEG, ST_SRID

11.3.3 ST_AsRasterAgg

ST_AsRasterAgg — Aggregiert. Wandelt PostGIS-Geometrien in einen neuen Raster um.

Synopsis

raster **ST_AsRasterAgg**(geometry geom, double precision val, raster ref, text pixeltype, double precision nodataval, text uniontype, boolean touched);

Description

Gibt einen einbandigen Raster zurück, der die gerasterte Version aller eingehenden Geometrien und den jeweils zugehörigen Wert enthält.

Verfügbarkeit: 3.6.0

Beispiele

Code

```
WITH inp(g, v) AS (
    VALUES
        ( ST_Buffer(ST_MakePoint(10, 0), 10), 1 ),
        ( ST_Buffer(ST_MakePoint(20, 0), 10), 2 )
),
agg AS (
    SELECT ST_AsRasterAgg(
        g,
        v,
        ST_MakeEmptyRaster(0, 0, 0, 0, 1.0),
        '8BUI',
        99,
        'SUM',
        true
    ) r
    FROM inp
)
SELECT
    ST_Width(r) w,
    ST_Height(r) h,
    ST_Value(r, 'POINT(5 0)') v5_0,
    ST_Value(r, 'POINT(15 0)') v15_0,
    ST_Value(r, 'POINT(25 0)') v25_0
FROM agg;
```

Output

```
w  | h  | v5_0 | v15_0 | v25_0
----+----+------+-------+-------
 30 | 20 |    1 |     3 |     2
(1 row)
```

Siehe auch

ST_AsRaster, ST_DumpAsPolygons, ST_Union

11.3.4 ST_Band

ST_Band — Gibt einen oder mehrere Bänder eines bestehenden Rasters als neuen Raster aus. Nützlich um neue Raster aus bestehenden Rastern abzuleiten.

Synopsis

raster **ST_Band**(raster rast, integer[] nbands = ARRAY[1]);
raster **ST_Band**(raster rast, integer nband);
raster **ST_Band**(raster rast, text nbands, character delimiter=,);

Description

Gibt einen oder mehrere Bänder eines bestehenden Rasters in Form eines neuen Raster aus. Nützlich um neue Raster aus bestehenden Rastern abzuleiten, um einen Export auf ausgewählte Bänder einzuschränken, oder um die Reihenfolge der Rasterbänder neu zu ordnen. Wenn kein Band angegeben ist, oder keines der spezifizierten Bänder im Raster existiert, dann werden alle Bänder zurückgegeben. Dient auch als Hilfsfunktion für verschiedene Funktionen, wie das Löschen eines Bandes.



Warnung
Bei der Funktionsvariante mit `nbands` als Text, ist das Standardtrennzeichen `,` ; d.h.: Sie können `'1,2,3'` abfragen und falls Sie ein anderes Trennzeichen verwenden wollen `ST_Band(rast, '1@2@3', '@')`. Wenn Sie mehrere Bänder abfragen, empfehlen wir Ihnen unbedingt die Feldvariante der Funktion zu verwenden, z.B. `ST_Band(rast, '{1,2,3}'::int[]);`, da die Variante mit der `text` Liste der Bänder in zukünftigen Versionen von PostGIS entfernt werden könnte.

Verfügbarkeit: 2.0.0

Beispiele

This example creates two rasters: one containing band 1 of dummy_rast, and one containing band 2 reclassified as 2BUI.

Code

```
SELECT ST_NumBands(rast1) As numb1, ST_BandPixelType(rast1) As pix1,
 ST_NumBands(rast2) As numb2, ST_BandPixelType(rast2) As pix2
FROM (
    SELECT ST_Band(rast) As rast1, ST_Reclass(ST_Band(rast, 3), '100-200):1, [200-254:2', '2 ↩
    BUI') As rast2
        FROM dummy_rast
        WHERE rid = 2) As foo;
```

Output

```
numb1 | pix1 | numb2 | pix2
------+------+-------+------
     1 | 8BUI |     1 | 2BUI
```

Return bands 2 and 3 using array cast syntax.

Code

```
SELECT ST_NumBands(ST_Band(rast, '{2,3}'::int[])) As num_bands
FROM dummy_rast WHERE rid=2;
```

Output

```
2
```

Return bands 2 and 3 using array syntax.

Code

```
SELECT ST_NumBands(ST_Band(rast, ARRAY[2, 3])) As num_bands
    FROM dummy_rast
WHERE rid=2;
```

Output

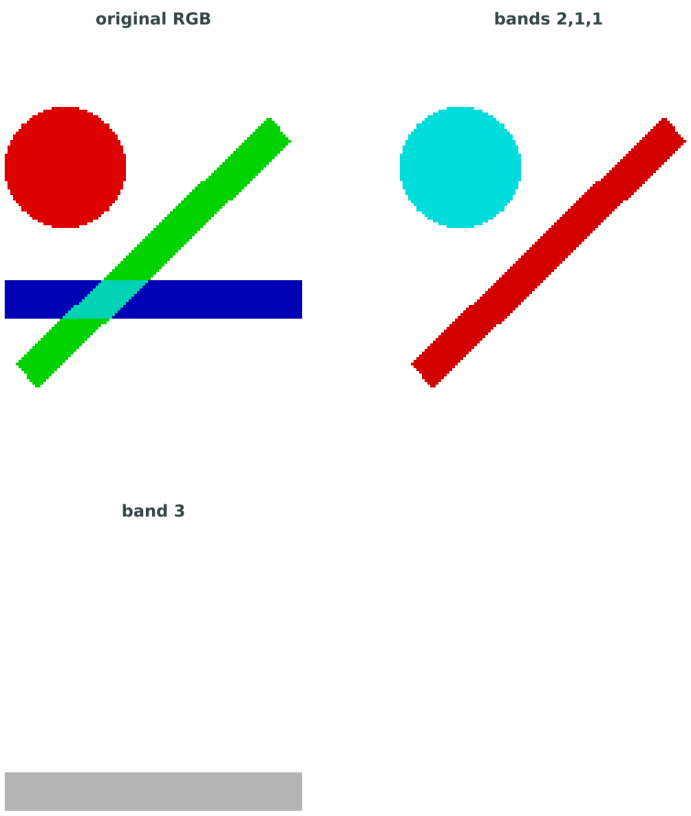
```
2
```

This example uses three distinct shapes as the red, green, and blue bands of a synthetic raster. It compares the original RGB raster with band 2 followed by band 1 twice, and with band 3 alone.

Code

```
WITH canvas AS (
    SELECT ST_AddBand(
        ST_AddBand(
            ST_AddBand(
                ST_MakeEmptyRaster(128, 128, 0, 128, 1, -1, 0, 0, 0),
                '8BUI'::text, 0::double precision, 0::double precision
            ),
            '8BUI'::text, 0::double precision, 0::double precision
        ),
        '8BUI'::text, 0::double precision, 0::double precision
    ) AS rast
), bands AS (
    SELECT ST_SetValue(
        ST_SetValue(
            ST_SetValue(rast, 1, ST_Buffer('POINT(32 96)'::geometry, 22), 220),
            2, ST_Buffer('LINESTRING(18 20,110 110)'::geometry, 6,
                         'endcap=flat join=bevel'), 210
        ),
        3, ST_Buffer('LINESTRING(10 48,118 48)'::geometry, 7,
                     'endcap=flat'), 180
    ) AS rast
    FROM canvas
)
SELECT ST_AsPNG(rast) AS "original RGB",
    ST_AsPNG(ST_Band(rast, ARRAY[2, 1, 1])) AS "bands 2,1,1",
    ST_AsPNG(ST_Band(rast, 3)) AS "band 3"
FROM bands;
```

Figure



Band selection from a multi-band raster.



Original (COLUMN rast)



dupe_band



sing_band

Siehe auch

ST_AddBand, ST_NumBands, ST_Reclass, Kapitel 11

11.3.5 ST_MakeEmptyCoverage

ST_MakeEmptyCoverage — Bedeckt die georeferenzierte Fläche mit einem Gitter aus leeren Rasterkacheln.

Synopsis

raster **ST_MakeEmptyCoverage**(integer tilewidth, integer tileheight, integer width, integer height, double precision upperleftx, double precision upperlefty, double precision scalex, double precision scaley, double precision skewx, double precision skewy, integer srid=unknown);

Description

Erzeugt Rasterkacheln mit ST_MakeEmptyRaster. Die Größe des Gitters wird über `width` & `height` angegeben. Die Kachelgröße über `tilewidth` & `tileheight`. Die abgedeckte georeferenzierte Fläche reicht vom oberen linken Eck (`upperleftx`, `upperlefty`) bis zum unteren rechten Eck (`upperleftx` + `width` * `scalex`, `upperlefty` + `height` * `scaley`).



Anmerkung
Beachten Sie bitte, dass bei Rastern "scaley" im Allgemeinen negativ und "scalex" positiv ist. Dadurch hat das untere rechte Eck einen niedrigeren Y-Wert und einen höheren X-Wert als die obere rechte Ecke.

Verfügbarkeit: 2.4.0

Grundlegende Beispiele

Erzeugt ein 4x4 Gitter aus 16 Kacheln, um die WGS84 Fläche vom linken oberen Eck (22, 77) zum rechten unteren Eck (55, 33) abzudecken.

Code

```
SELECT (ST_MetaData(tile)).* FROM ST_MakeEmptyCoverage(1, 1, 4, 4, 22, 33, (55 - 22)/(4):: ↩
    float, (33 - 77)/(4)::float, 0., 0., 4326) tile;
```

Output

```
upperleftx | upperlefty | width | height | scalex | scaley | skewx | skewy | srid | numbands
-----------------------------------------------------------------------------------------
         22 |         33 |     1 |      1 |   8.25 |    -11 |     0 |     0 | 4326 |        0
      30.25 |         33 |     1 |      1 |   8.25 |    -11 |     0 |     0 | 4326 |        0
       38.5 |         33 |     1 |      1 |   8.25 |    -11 |     0 |     0 | 4326 |        0
      46.75 |         33 |     1 |      1 |   8.25 |    -11 |     0 |     0 | 4326 |        0
         22 |         22 |     1 |      1 |   8.25 |    -11 |     0 |     0 | 4326 |        0
      30.25 |         22 |     1 |      1 |   8.25 |    -11 |     0 |     0 | 4326 |        0
       38.5 |         22 |     1 |      1 |   8.25 |    -11 |     0 |     0 | 4326 |        0
      46.75 |         22 |     1 |      1 |   8.25 |    -11 |     0 |     0 | 4326 |        0
         22 |         11 |     1 |      1 |   8.25 |    -11 |     0 |     0 | 4326 |        0
      30.25 |         11 |     1 |      1 |   8.25 |    -11 |     0 |     0 | 4326 |        0
       38.5 |         11 |     1 |      1 |   8.25 |    -11 |     0 |     0 | 4326 |        0
      46.75 |         11 |     1 |      1 |   8.25 |    -11 |     0 |     0 | 4326 |        0
         22 |          0 |     1 |      1 |   8.25 |    -11 |     0 |     0 | 4326 |        0
      30.25 |          0 |     1 |      1 |   8.25 |    -11 |     0 |     0 | 4326 |        0
       38.5 |          0 |     1 |      1 |   8.25 |    -11 |     0 |     0 | 4326 |        0
      46.75 |          0 |     1 |      1 |   8.25 |    -11 |     0 |     0 | 4326 |        0
```

Siehe auch

ST_MakeEmptyRaster

11.3.6 ST_MakeEmptyRaster

ST_MakeEmptyRaster — Gibt einen leeren Raster (ohne Bänder), mit den gegebenen Dimensionen (width & height), upperleft X und Y, Pixelgröße, Rotation (scalex, scaley, skewx & skewy) und Koordinatenreferenzsystem (SRID), zurück. Wenn ein Raster übergeben wird, dann wird ein neuer Raster mit der selben Größe, Ausrichtung und SRID zurückgegeben. Wenn SRID nicht angegeben ist, wird das Koordinatenreferenzsystem auf "unknown" (0) gesetzt.

Synopsis

raster **ST_MakeEmptyRaster**(raster rast);
raster **ST_MakeEmptyRaster**(integer width, integer height, float8 upperleftx, float8 upperlefty, float8 scalex, float8 scaley, float8 skewx, float8 skewy, integer srid=unknown);
raster **ST_MakeEmptyRaster**(integer width, integer height, float8 upperleftx, float8 upperlefty, float8 pixelsize);

Description

Gibt einen leeren Raster (ohne Bänder), mit den gegebenen Dimensionen (width & height), georeferenziert in geodätischen (oder geographischen) Koordinaten, oberes linkes X (upperleftx), oberes linkes Y (upperlefty), Pixelgröße, Rotation (scalex, scaley, skewx & skewy) und Koordinatenreferenzsystem (SRID), zurück.

Die letzte Version verwendet einen einzelnen Parameter um die Pixelgröße festzulegen. "scalex" wird auf diesen Übergabewert und "scaley" auf den negativen Wert davon gesetzt. "skewx" und "skewy" werden auf 0 gesetzt.

Wird ein bestehender Raster übergeben, so wird ein neuer Raster mit den gleichen Metadateneinstellungen erstellt (ohne die Bänder).

Wenn die SRID nicht angegeben ist, wird sie standardmäßig auf 0 gesetzt. Nachdem Sie einen leeren Raster erzeugt haben, werden Sie vermutlich Bänder hinzufügen und eventuell auch bearbeiten wollen. Siehe ST_AddBand um die Bänder festzulegen und ST_SetValue um Ausgangswerte für die Pixel zu setzen.

Beispiele

Use an existing raster as a template and output metadata of the rasters we added.

Code

```
INSERT INTO dummy_rast(rid, rast)
VALUES(3, ST_MakeEmptyRaster(100, 100, 0.0005, 0.0005, 1, 1, 0, 0, 4326) );
INSERT INTO dummy_rast(rid, rast)
SELECT 4, ST_MakeEmptyRaster(rast)
FROM dummy_rast WHERE rid = 3;
SELECT
    rid,
    (md).width,
    (md).height,
    (md).scalex,
    (md).scaley,
    (md).srid
FROM (
    SELECT rid, ST_MetaData(rast) AS md
    FROM dummy_rast
    WHERE rid IN (3, 4)
) AS foo
ORDER BY rid;
```

Output

```
rid | width | height | scalex | scaley | srid
-----+-------+--------+--------+--------+------
   3 |   100 |    100 |      1 |      1 | 4326
   4 |   100 |    100 |      1 |      1 | 4326
```

Siehe auch

ST_AddBand, ST_MetaData, ST_ScaleX, ST_ScaleY, ST_SetValue, ST_SkewX, , ST_SkewY

11.3.7 ST_Tile

ST_Tile — Gibt Raster, die aus einer Teilungsoperation des Eingaberasters resultieren, mit den gewünschten Dimensionen aus.

Synopsis

setof raster **ST_Tile**(raster rast, int[] nband, integer width, integer height, boolean padwithnodata=FALSE, double precision nodataval=NULL);
setof raster **ST_Tile**(raster rast, integer nband, integer width, integer height, boolean padwithnodata=FALSE, double precision nodataval=NULL);
setof raster **ST_Tile**(raster rast, integer width, integer height, boolean padwithnodata=FALSE, double precision nodataval=NULL);

Description

Gibt Raster, die aus einer Teilungsoperation des Eingaberasters resultieren, mit den gewünschten Dimensionen aus.

Wenn `padwithnodata` = FALSE, dann können die Eckkacheln auf der rechten Seite und auf der unteren Seite andere Dimensionen als die übrigen Kacheln aufweisen. Wenn `padwithnodata` = TRUE, dann haben alle Kacheln dieselbe Dimension und die Eckkacheln werden eventuell mit NODATA Werten aufgefüllt. Wenn für die Rasterbänder keine NODATA Werte festgelegt sind, können Sie diese mit `nodataval` spezifizieren.



Anmerkung
Wenn das Band des Eingaberasters "out-of-db" ist, dann ist das entsprechende Band des Ausgaberasters auch "out-of-db".

Verfügbarkeit: 2.1.0

Beispiele

Code

```
WITH foo AS (
    SELECT ST_AddBand(ST_AddBand(ST_MakeEmptyRaster(3, 3, 0, 0, 1, -1, 0, 0, 0), 1, '8BUI',  ↩
    1, 0), 2, '8BUI', 10, 0) AS rast UNION ALL
    SELECT ST_AddBand(ST_AddBand(ST_MakeEmptyRaster(3, 3, 3, 0, 1, -1, 0, 0, 0), 1, '8BUI',  ↩
    2, 0), 2, '8BUI', 20, 0) AS rast UNION ALL
    SELECT ST_AddBand(ST_AddBand(ST_MakeEmptyRaster(3, 3, 6, 0, 1, -1, 0, 0, 0), 1, '8BUI',  ↩
    3, 0), 2, '8BUI', 30, 0) AS rast UNION ALL

    SELECT ST_AddBand(ST_AddBand(ST_MakeEmptyRaster(3, 3, 0, -3, 1, -1, 0, 0, 0), 1, '8BUI', ↩
     4, 0), 2, '8BUI', 40, 0) AS rast UNION ALL
    SELECT ST_AddBand(ST_AddBand(ST_MakeEmptyRaster(3, 3, 3, -3, 1, -1, 0, 0, 0), 1, '8BUI', ↩
     5, 0), 2, '8BUI', 50, 0) AS rast UNION ALL
    SELECT ST_AddBand(ST_AddBand(ST_MakeEmptyRaster(3, 3, 6, -3, 1, -1, 0, 0, 0), 1, '8BUI', ↩
     6, 0), 2, '8BUI', 60, 0) AS rast UNION ALL

    SELECT ST_AddBand(ST_AddBand(ST_MakeEmptyRaster(3, 3, 0, -6, 1, -1, 0, 0, 0), 1, '8BUI', ↩
     7, 0), 2, '8BUI', 70, 0) AS rast UNION ALL
    SELECT ST_AddBand(ST_AddBand(ST_MakeEmptyRaster(3, 3, 3, -6, 1, -1, 0, 0, 0), 1, '8BUI', ↩
     8, 0), 2, '8BUI', 80, 0) AS rast UNION ALL
    SELECT ST_AddBand(ST_AddBand(ST_MakeEmptyRaster(3, 3, 6, -6, 1, -1, 0, 0, 0), 1, '8BUI', ↩
     9, 0), 2, '8BUI', 90, 0) AS rast
), bar AS (
    SELECT ST_Union(rast) AS rast FROM foo
), baz AS (
```

```
    SELECT
        row_number() OVER (
            ORDER BY ST_UpperLeftY(rast) DESC, ST_UpperLeftX(rast)
        ) AS tile_id,
        rast
    FROM (
        SELECT ST_Tile(rast, 3, 3, TRUE) AS rast
        FROM bar
    ) AS tiled
)
SELECT
    tile_id,
    ST_UpperLeftX(rast) AS ulx,
    ST_UpperLeftY(rast) AS uly,
    ST_Value(rast, 1, 1, 1) AS band1,
    ST_Value(rast, 2, 1, 1) AS band2
FROM baz;
```

Output

```
tile_id | ulx | uly | band1 | band2
---------+-----+-----+-------+-------
       1 |   0 |   0 |     1 |    10
       2 |   3 |   0 |     2 |    20
       3 |   6 |   0 |     3 |    30
       4 |   0 |  -3 |     4 |    40
       5 |   3 |  -3 |     5 |    50
       6 |   6 |  -3 |     6 |    60
       7 |   0 |  -6 |     7 |    70
       8 |   3 |  -6 |     8 |    80
       9 |   6 |  -6 |     9 |    90
(9 rows)
```

Code

```
WITH foo AS (
    SELECT ST_AddBand(ST_AddBand(ST_MakeEmptyRaster(3, 3, 0, 0, 1, -1, 0, 0, 0), 1, '8BUI',  ↩
    1, 0), 2, '8BUI', 10, 0) AS rast UNION ALL
    SELECT ST_AddBand(ST_AddBand(ST_MakeEmptyRaster(3, 3, 3, 0, 1, -1, 0, 0, 0), 1, '8BUI',  ↩
    2, 0), 2, '8BUI', 20, 0) AS rast UNION ALL
    SELECT ST_AddBand(ST_AddBand(ST_MakeEmptyRaster(3, 3, 6, 0, 1, -1, 0, 0, 0), 1, '8BUI',  ↩
    3, 0), 2, '8BUI', 30, 0) AS rast UNION ALL

    SELECT ST_AddBand(ST_AddBand(ST_MakeEmptyRaster(3, 3, 0, -3, 1, -1, 0, 0, 0), 1, '8BUI', ↩
     4, 0), 2, '8BUI', 40, 0) AS rast UNION ALL
    SELECT ST_AddBand(ST_AddBand(ST_MakeEmptyRaster(3, 3, 3, -3, 1, -1, 0, 0, 0), 1, '8BUI', ↩
     5, 0), 2, '8BUI', 50, 0) AS rast UNION ALL
    SELECT ST_AddBand(ST_AddBand(ST_MakeEmptyRaster(3, 3, 6, -3, 1, -1, 0, 0, 0), 1, '8BUI', ↩
     6, 0), 2, '8BUI', 60, 0) AS rast UNION ALL

    SELECT ST_AddBand(ST_AddBand(ST_MakeEmptyRaster(3, 3, 0, -6, 1, -1, 0, 0, 0), 1, '8BUI', ↩
     7, 0), 2, '8BUI', 70, 0) AS rast UNION ALL
    SELECT ST_AddBand(ST_AddBand(ST_MakeEmptyRaster(3, 3, 3, -6, 1, -1, 0, 0, 0), 1, '8BUI', ↩
     8, 0), 2, '8BUI', 80, 0) AS rast UNION ALL
    SELECT ST_AddBand(ST_AddBand(ST_MakeEmptyRaster(3, 3, 6, -6, 1, -1, 0, 0, 0), 1, '8BUI', ↩
     9, 0), 2, '8BUI', 90, 0) AS rast
), bar AS (
    SELECT ST_Union(rast) AS rast FROM foo
), baz AS (
    SELECT
        row_number() OVER (
            ORDER BY ST_UpperLeftY(rast) DESC, ST_UpperLeftX(rast)
        ) AS tile_id,
        rast
    FROM (
```

```
        SELECT ST_Tile(rast, ARRAY[2], 3, 3) AS rast
        FROM bar
    ) AS tiled
)
SELECT
    tile_id,
    ST_UpperLeftX(rast) AS ulx,
    ST_UpperLeftY(rast) AS uly,
    ST_Value(rast, 1, 1, 1) AS band_value
FROM baz;
```

Output

```
tile_id | ulx | uly | band_value
---------+-----+-----+------------
       1 |   0 |   0 |         10
       2 |   3 |   0 |         20
       3 |   6 |   0 |         30
       4 |   0 |  -3 |         40
       5 |   3 |  -3 |         50
       6 |   6 |  -3 |         60
       7 |   0 |  -6 |         70
       8 |   3 |  -6 |         80
       9 |   6 |  -6 |         90
(9 rows)
```

Siehe auch

ST_Union, ST_Retile

11.3.8 ST_Retile

ST_Retile — Gibt konfigurierte Kacheln eines beliebig gekachelten Rastercoverage aus.

Synopsis

SETOF raster **ST_Retile**(regclass tab, name col, geometry ext, float8 sfx, float8 sfy, int tw, int th, text algo='NearestNeighbor');

Description

Gibt die Kacheln in einem bestimmten Maßstab (`sfx`, `sfy`), maximaler Größe (`tw`, `th`) und räumlicher Ausdehnung (`ext`) mit den Daten des angegebenen Raster-Coverages (`tab`, `col`) zurück.

Die Optionen für den Algorithmus sind: 'NearestNeighbor', 'Bilinear', 'Cubic', 'CubicSpline' und 'Lanczos'. Siehe GDAL Warp resampling methods für weitere Details.

Verfügbarkeit: 2.2.0

Siehe auch

ST_CreateOverview

11.3.9 ST_FromGDALRaster

ST_FromGDALRaster — Erzeugt einen Raster aus einer von GDAL unterstützten Rasterdatei.

Synopsis

raster **ST_FromGDALRaster**(bytea gdaldata, integer srid=NULL);

Description

Erzeugt einen Raster aus einer von GDAL unterstützten Rasterdatei. `gdaldata` hat den Datentyp BYTEA und den Inhalt der GDAL Rasterdatei.

Wenn `srid` NULL ist, dann versucht die Funktion die SRID aus dem GDAL Raster automatisch zuzuweisen. Wenn `srid` angegeben ist, überschreibt dieser Werrt jede automatisch zugewiesene SRID.

Verfügbarkeit: 2.1.0

Beispiele

Code

```
WITH foo AS (
    SELECT ST_AsPNG(ST_AddBand(ST_AddBand(ST_AddBand(ST_MakeEmptyRaster(2, 2, 0, 0, 0.1,  ↩
    -0.1, 0, 0, 4326), 1, '8BUI', 1, 0), 2, '8BUI', 2, 0), 3, '8BUI', 3, 0)) AS png
),
bar AS (
    SELECT 1 AS rid, ST_FromGDALRaster(png) AS rast FROM foo
    UNION ALL
    SELECT 2 AS rid, ST_FromGDALRaster(png, 3310) AS rast FROM foo
)
SELECT
    rid,
    ST_MetaData(rast) AS metadata,
    ST_SummaryStats(rast, 1) AS stats1,
    ST_SummaryStats(rast, 2) AS stats2,
    ST_SummaryStats(rast, 3) AS stats3
FROM bar
ORDER BY rid;
```

Output

```
rid |         metadata          |    stats1     |    stats2     |     stats3
-----+---------------------------+---------------+---------------+----------------
   1 | (0,0,2,2,1,-1,0,0,0,3)    | (4,4,1,0,1,1) | (4,8,2,0,2,2) | (4,12,3,0,3,3)
   2 | (0,0,2,2,1,-1,0,0,3310,3) | (4,4,1,0,1,1) | (4,8,2,0,2,2) | (4,12,3,0,3,3)
(2 rows)
```

Siehe auch

ST_AsGDALRaster

11.4 Zugriffsfunktionen auf Raster

11.4.1 ST_GeoReference

ST_GeoReference — Gibt die Metadaten der Georeferenzierung, die sich üblicherweise in einem sogenannten "World File befinden, im GDAL oder ESRI Format aus. Die Standardeinstellung ist GDAL.

Synopsis

text **ST_GeoReference**(raster rast, text format=GDAL);

Description

Gibt die Metadaten der Georeferenzierung, die sich üblicherweise in einem sogenannten "World File befinden, inklusive "Carriage Return" im GDAL oder ESRI Format aus. Die Standardeinstellung ist GDAL. Der Datentyp ist die Zeichenfolge 'GDAL' oder 'ESRI'.

Die Formate unterscheiden sich wie folgt:

GDAL:

Output

```
scalex
skewy
skewx
scaley
upperleftx
upperlefty
```

ESRI:

Output

```
scalex
skewy
skewx
scaley
upperleftx + scalex*0.5
upperlefty + scaley*0.5
```

Beispiele

Code

```
SELECT
    format,
    replace(ST_GeoReference(rast, format), E'\n', ' | ') AS georef
FROM dummy_rast
CROSS JOIN (VALUES ('ESRI'), ('GDAL')) AS formats(format)
WHERE rid = 1;
```

Output

```
format | georef
--------+-------------------------------------------------------------------------
 ESRI   | 2.0000000000 | 0.0000000000 | 0.0000000000 | 3.0000000000 | 1.5000000000 | 2.0000000000
 GDAL   | 2.0000000000 | 0.0000000000 | 0.0000000000 | 3.0000000000 | 0.5000000000 | 0.5000000000
(2 rows)
```

Siehe auch

ST_SetGeoReference, ST_ScaleX, ST_ScaleY

11.4.2 ST_Height

ST_Height — Gibt die Höhe des Rasters in Pixel aus.

Synopsis

integer **ST_Height**(raster rast);

Description

Gibt die Höhe des Rasters aus.

Beispiele

Code

```
SELECT rid, ST_Height(rast) As rastheight
FROM dummy_rast;
```

Output

```
rid | rastheight
-----+------------
   1 |         20
   2 |          5
```

Siehe auch

ST_Width

11.4.3 ST_IsEmpty

ST_IsEmpty — Gibt TRUE zurück, wenn der Raster leer ist (width = 0 and height = 0). Andernfalls wird FALSE zurückgegeben.

Synopsis

boolean **ST_IsEmpty**(raster rast);

Description

Gibt TRUE zurück, wenn der Raster leer ist (width = 0 and height = 0). Andernfalls wird FALSE zurückgegeben.

Verfügbarkeit: 2.0.0

Beispiele

Code

```
SELECT ST_IsEmpty(ST_MakeEmptyRaster(100, 100, 0, 0, 0, 0, 0, 0))
```

Output

```
f
```

Code

```
SELECT ST_IsEmpty(ST_MakeEmptyRaster(0, 0, 0, 0, 0, 0, 0, 0))
```

Output

```
t
```

Siehe auch

ST_HasNoBand

11.4.4 ST_MemSize

ST_MemSize — Gibt den Platzbedarf des Rasters (in Byte) aus.

Synopsis

integer **ST_MemSize**(raster rast);

Description

Gibt den Platzbedarf des Rasters (in Byte) aus.

Dies Funktion ist eine schöne Ergänzung zu den in PostgreSQL eingebauten Funktionen pg_column_size, pg_size_pretty, pg_relation_siz
und pg_total_relation_size.



Anmerkung
pg_relation_size, das die Größe einer Tabelle in Byte angibt kann niedrigere Werte liefern als ST_MemSize. Dies kann vorkommen, da pg_relation_size den TOAST-Speicher der Tabellen nicht mitrechnet und eine große Geometrie in TOAST-Tabellen gespeichert wird. pg_column_size kann einen niedrigeren Wert anzeigen, da es die komprimierte Dateigröße ausgibt.
pg_total_relation_size - schließt die Tabelle, die TOAST-Tabellen und di Indizes mit ein.

Verfügbarkeit: 2.2.0

Beispiele

Code

```
SELECT ST_MemSize(ST_AsRaster(ST_Buffer(ST_Point(1, 5), 10, 1000), 150, 150, '8BUI')) AS ↩
    rast_mem;
```

Output

```
22568
```

Siehe auch

11.4.5 ST_MetaData

ST_MetaData — Gibt die wesentlichen Metadaten eines Rasterobjektes, wie Zellgröße, Rotation (Versatz) etc. aus

Synopsis

record **ST_MetaData**(raster rast);

Description

Gibt die grundlegenden Metadaten eines Rasters aus, wie Pixelgröße, Rotation (skew), obere, untere linke, etc.. Die zurückgegebenen Spalten sind: upperleftx | upperlefty | width | height | scalex | scaley | skewx | skewy | srid | numbands

Beispiele

Code

```
SELECT
    rid,
    format('(%s, %s)', (md).upperleftx, (md).upperlefty) AS upper_left,
    format('%sx%s', (md).width, (md).height) AS size,
    format('(%s, %s)', (md).scalex, (md).scaley) AS scale,
    format('(%s, %s)', (md).skewx, (md).skewy) AS skew,
    (md).srid AS srid,
    (md).numbands AS numbands
FROM (
    SELECT rid, ST_MetaData(rast) AS md
    FROM dummy_rast
) AS foo
ORDER BY rid;
```

Output

```
rid |        upper_left         | size  |      scale      |  skew  | srid | numbands
-----+---------------------------+-------+-----------------+--------+------+----------
   1 | (0.5, 0.5)                | 10x20 | (2, 3)          | (0, 0) |    0 |        0
   2 | (3427927.75, 5793244)     | 5x5   | (0.05, -0.05)   | (0, 0) |    0 |        3
(2 rows)
```

Siehe auch

ST_BandMetaData, ST_NumBands

11.4.6 ST_NumBands

ST_NumBands — Gibt die Anzahl der Bänder des Rasters aus.

Synopsis

integer **ST_NumBands**(raster rast);

Description

Gibt die Anzahl der Bänder des Rasters aus.

Beispiele

Code

```
SELECT rid, ST_NumBands(rast) As numbands
FROM dummy_rast;
```

Output

```
rid | numbands
----+----------
  1 |        0
  2 |        3
```

Siehe auch

ST_Value

11.4.7 ST_PixelHeight

ST_PixelHeight — Gibt die Pixelhöhe in den Einheiten des Koordinatenreferenzsystem aus.

Synopsis

double precision **ST_PixelHeight**(raster rast);

Description

Gibt die Höhe eines Pixels in den Einheiten des Koordinatenreferenzsystem aus. Beim üblichen Fall ohne Versatz/skew, entspricht die Pixelhöhe dem Maßstabsverhältnis zwischen den geometrischen Koordinaten und den Rasterpixeln.

Siehe ST_PixelWidth für eine schematische Darstellung der Verhältnisse.

Beispiele

Rasters with no skew.

Code

```
SELECT ST_Height(rast) As rastheight, ST_PixelHeight(rast) As pixheight,
 ST_ScaleX(rast) As scalex, ST_ScaleY(rast) As scaley, ST_SkewX(rast) As skewx,
        ST_SkewY(rast) As skewy
FROM dummy_rast;
```

Output

```
rastheight | pixheight | scalex | scaley | skewx | skewy
-----------+-----------+--------+--------+-------+----------
        20 |         3 |      2 |      3 |     0 |        0
         5 |      0.05 |   0.05 |  -0.05 |     0 |        0
```

Rasters with skew different than 0.

Code

```
SELECT ST_Height(rast) As rastheight, ST_PixelHeight(rast) As pixheight,
 ST_ScaleX(rast) As scalex, ST_ScaleY(rast) As scaley, ST_SkewX(rast) As skewx,
        ST_SkewY(rast) As skewy
FROM (SELECT ST_SetSkew(rast, 0.5, 0.5) As rast
        FROM dummy_rast) As skewed;
```

Output

```
rastheight |     pixheight     | scalex | scaley | skewx | skewy
-----------+-------------------+--------+--------+-------+-------
        20 |  3.04138126514911 |      2 |      3 |   0.5 |   0.5
         5 | 0.502493781056044 |   0.05 |  -0.05 |   0.5 |   0.5
```

Siehe auch

ST_PixelWidth, ST_ScaleX, ST_ScaleY, ST_SkewX, ST_SkewY

11.4.8 ST_PixelWidth

ST_PixelWidth — Gibt die Pixelbreite in den Einheiten des Koordinatenreferenzsystems aus.

Synopsis

double precision **ST_PixelWidth**(raster rast);

Description

Gibt die Pixelbreite in den Einheiten des Koordinatenreferenzsystems aus. Beim üblichen Fall ohne Versatz entspricht die Pixelbreite dem Maßstabsverhältnis zwischen den geometrischen Koordinaten und den Rasterpixel.

The following executable example draws the two pixel basis vectors on the same coordinate frame as a skewed raster footprint. The `i direction` vector is `(scaleX, skewY)`; the `j direction` vector is `(skewX, scaleY)`.

Beispiele

Pixel basis vectors and footprint for a raster with non-zero skew.

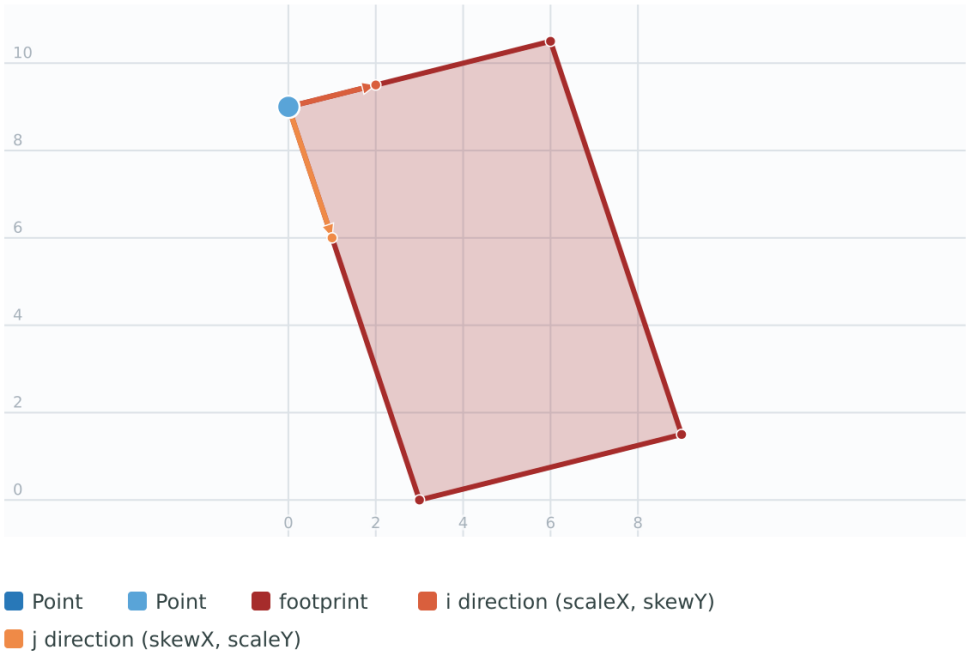
Code

```
WITH raster AS (
    SELECT ST_MakeEmptyRaster(
        3, 3, 0, 0,
        2, -3, 1, 0.5,
        0
    ) AS rast
)
SELECT
    ST_ConvexHull(rast) AS footprint,
    ST_MakeLine(
        ST_Point(0, 0),
        ST_Point(ST_ScaleX(rast), ST_SkewY(rast))
    ) AS "i direction (scaleX, skewY)",
    ST_MakeLine(
        ST_Point(0, 0),
        ST_Point(ST_SkewX(rast), ST_ScaleY(rast))
    ) AS "j direction (skewX, scaleY)"
FROM raster;
```

Output

```
POLYGON((0 0,6 1.5,9 -7.5,3 -9,0 0)) | LINESTRING(0 0,2 0.5) | LINESTRING(0 0,1 -3)
```

Figure



Rasters with no skew.

Code

```
SELECT ST_Width(rast) As rastwidth, ST_PixelWidth(rast) As pixwidth,
ST_ScaleX(rast) As scalex, ST_ScaleY(rast) As scaley, ST_SkewX(rast) As skewx,
ST_SkewY(rast) As skewy
FROM dummy_rast;
```

Output

```
rastwidth | pixwidth | scalex | scaley | skewx | skewy
-----------+----------+--------+--------+-------+-----------
10 |        2 |      2 |      3 |     0 |        0
 5 |     0.05 |   0.05 |  -0.05 |     0 |        0
```

Rasters with skew different than 0.

Code

```
SELECT ST_Width(rast) As rastwidth, ST_PixelWidth(rast) As pixwidth,
ST_ScaleX(rast) As scalex, ST_ScaleY(rast) As scaley, ST_SkewX(rast) As skewx,
ST_SkewY(rast) As skewy
FROM (SELECT ST_SetSkew(rast, 0.5, 0.5) As rast
FROM dummy_rast) As skewed;
```

Output

```
rastwidth |     pixwidth      | scalex | scaley | skewx | skewy
-----------+-------------------+--------+--------+-------+-----------
10 |  2.06155281280883 |      2 |      3 |   0.5 |       0.5
 5 | 0.502493781056044 |   0.05 |  -0.05 |   0.5 |       0.5
```

Siehe auch

ST_PixelHeight, ST_ScaleX, ST_ScaleY, ST_SkewX, ST_SkewY

11.4.9 ST_ScaleX

ST_ScaleX — Gibt die X-Komponente der Pixelbreite in den Einheiten des Koordinatenreferenzsystems aus.

Synopsis

float8 **ST_ScaleX**(raster rast);

Description

Gibt die X-Komponente der Pixelbreite in den Einheiten des Koordinatenreferenzsystems aus. Siehe World File für weitere Details.

Änderung: 2.0.0 In WKTRaster Versionen wurde dies als ST_PixelSizeX bezeichnet.

Beispiele

Code

```
SELECT rid, ST_ScaleX(rast) As rastpixwidth
FROM dummy_rast;
```

Output

```
rid | rastpixwidth
-----+--------------
   1 |            2
   2 |         0.05
```

Siehe auch

ST_Width

11.4.10 ST_ScaleY

ST_ScaleY — Gibt die Y-Komponente der Pixelhöhe in den Einheiten des Koordinatenreferenzsystems aus.

Synopsis

float8 **ST_ScaleY**(raster rast);

Description

Gibt die Y-Komponente der Pixelhöhe in den Einheiten des Koordinatenreferenzsystems aus. Kann negative Werte annehmen. Siehe World File für weitere Details.

Änderung: 2.0.0. Versionen von WKTRaster haben dies als "ST_PixelSizeY" bezeichnet.

Beispiele

Code

```
SELECT rid, ST_ScaleY(rast) As rastpixheight
FROM dummy_rast;
```

Output

```
rid | rastpixheight
-----+--------------
   1 |             3
   2 |         -0.05
```

Siehe auch

ST_Height

11.4.11 ST_RasterToWorldCoord

ST_RasterToWorldCoord — Gibt die obere linke Ecke des Rasters in geodätischem X und Y (Länge und Breite) für eine gegebene Spalte und Zeile aus. Spalte und Zeile wird von 1 aufwärts gezählt.

Synopsis

record **ST_RasterToWorldCoord**(raster rast, integer xcolumn, integer yrow);

Description

Gibt die obere linke Ecke einer Spalte und Zeile als geometrisches X und Y (als geographische Länge und Breite) aus. Die zurückgegebenen X und Y sind in den Einheiten des georeferenzierten Rasters. Die Nummerierung der Spalten und Zeilen beginnt mit 1. Allerdings, wenn für einen der Parameter 0, eine negative Zahl, oder eine höhere Zahl als die betreffende Größe des Rasters übergeben wird, dann werden Koordinaten außerhalb des Rasters ausgegeben; dabei wird angenommen, dass das Gitter des Rasters auch außerhalb der Begrenzung des Rasters angewendet werden kann.

Verfügbarkeit: 2.1.0

Beispiele

This example uses a non-skewed raster.

Code

```
SELECT
    rid,
    (ST_RasterToWorldCoord(rast, 1, 1)).*,
    (ST_RasterToWorldCoord(rast, 2, 2)).*
FROM dummy_rast
```

Output

```
rid | longitude  | latitude | longitude |  latitude
-----+------------+----------+-----------+------------
   1 |        0.5 |      0.5 |       2.5 |        3.5
   2 | 3427927.75 |  5793244 | 3427927.8 | 5793243.95
```

This example uses a skewed raster.

Code

```
SELECT
    rid,
    (ST_RasterToWorldCoord(rast, 1, 1)).*,
    (ST_RasterToWorldCoord(rast, 2, 3)).*
FROM (
    SELECT
        rid,
        ST_SetSkew(rast, 100.5, 0) As rast
    FROM dummy_rast
) As foo
```

Output

```
rid | longitude  | latitude | longitude | latitude
----+------------+----------+-----------+----------
  1 |        0.5 |      0.5 |     203.5 |       6.5
  2 | 3427927.75 |  5793244 | 3428128.8 | 5793243.9
```

Siehe auch

ST_RasterToWorldCoordX, ST_RasterToWorldCoordY, ST_SetSkew

11.4.12 ST_RasterToWorldCoordX

ST_RasterToWorldCoordX — Gibt die geodätische X Koordinate links oberhalb des Rasters, der Spalte und der Zeile aus. Die Nummerierung der Spalten und Zeilen beginnt mit 1.

Synopsis

float8 **ST_RasterToWorldCoordX**(raster rast, integer xcolumn);
float8 **ST_RasterToWorldCoordX**(raster rast, integer xcolumn, integer yrow);

Description

Gibt die obere linke X-Koordinate einer Rasterzeile in den geometrischen Einheiten des georeferenzierten Rasters aus. Die Nummerierung der Spalten und Zeilen beginnt mit 1. Wenn Sie eine negative Zahl oder eine höhere Zahl als die Anzahl der Spalten des Rasters angeben, dann bekommen Sie die Koordinaten links außerhalb und rechts außerhalb des Rasters zurück; dabei wird angenommen, dass der Versatz und die Pixelgröße gleich wie bei dem ausgewählten Raster sind.



Anmerkung
Bei Rastern ohne Versatz, ist die Angabe der X-Spalte ausreichend. Bei Rastern mit Versatz ist die georeferenzierte Koordinate eine Funktion von "ST_ScaleX", "ST_SkewX", der Zeile und der Spalte. Wenn Sie bei einem Raster mit Versatz nur die X-Spalte angeben, dann wird eine Fehlermeldung ausgegeben.

Änderung: 2.1.0 Vorgängerversionen haben dies als "ST_Raster2WorldCoordX" bezeichnet.

Beispiele

This example shows that providing a column is sufficient for a non-skewed raster.

Code

```
SELECT rid, ST_RasterToWorldCoordX(rast, 1) As x1coord,
    ST_RasterToWorldCoordX(rast, 2) As x2coord,
    ST_ScaleX(rast) As pixelx
FROM dummy_rast;
```

Output

```
rid |  x1coord   |  x2coord  | pixelx
----+------------+-----------+--------
  1 |        0.5 |       2.5 |      2
  2 | 3427927.75 | 3427927.8 |   0.05
```

For fun let's skew it.

Code

```
SELECT rid, ST_RasterToWorldCoordX(rast, 1, 1) As x1coord,
    ST_RasterToWorldCoordX(rast, 2, 3) As x2coord,
    ST_ScaleX(rast) As pixelx
FROM (SELECT rid, ST_SetSkew(rast, 100.5, 0) As rast FROM dummy_rast) As foo;
```

Output

```
rid |  x1coord   |  x2coord  | pixelx
----+------------+-----------+--------
  1 |        0.5 |     203.5 |      2
  2 | 3427927.75 | 3428128.8 |   0.05
```

Siehe auch

ST_ScaleX, ST_RasterToWorldCoordY, ST_SetSkew, ST_SkewX

11.4.13 ST_RasterToWorldCoordY

ST_RasterToWorldCoordY — Gibt die geodätische Y Koordinate links oberhalb des Rasters, der Spalte und der Zeile aus. Die Nummerierung der Spalten und Zeilen beginnt mit 1.

Synopsis

float8 **ST_RasterToWorldCoordY**(raster rast, integer yrow);
float8 **ST_RasterToWorldCoordY**(raster rast, integer xcolumn, integer yrow);

Description

Gibt die obere linke Y-Koordinate einer Rasterspalte in den Einheiten des georeferenzierten Rasters aus. Die Nummerierung der Spalten und Zeilen beginnt mit 1. Wenn Sie eine negative Zahl oder eine höhere Zahl als die Anzahl der Spalten/Zeilen des Rasters angeben, dann bekommen Sie die Koordinaten außerhalb des Rasters zurück; dabei wird angenommen, dass Versatz und Pixelgröße gleich wie bei der ausgewählten Rasterkachel sind.



Anmerkung
Bei Rastern ohne Versatz ist die Angabe der Y-Spalte ausreichend. Bei Rastern mit Versatz entspricht die georeferenzierte Koordinate einer Funktion von ST_ScaleY, ST_SkewY, Zeile und Spalte. Wenn Sie bei einem Raster mit Versatz nur die Y-Spalte angeben, wird eine Fehlermeldung ausgegeben.

Änderung: 2.1.0 In Vorgängerversionen wurde dies als ST_Raster2WorldCoordY bezeichnet

Beispiele

Non-skewed raster providing row is sufficient.

Code

```
SELECT rid, ST_RasterToWorldCoordY(rast, 1) As y1coord,
    ST_RasterToWorldCoordY(rast, 3) As y2coord,
    ST_ScaleY(rast) As pixely
FROM dummy_rast;
```

Output

```
rid | y1coord |  y2coord  | pixely
----+---------+-----------+--------
  1 |     0.5 |       6.5 |      3
  2 | 5793244 | 5793243.9 |  -0.05
```

For fun let's skew it.

Code

```
SELECT rid, ST_RasterToWorldCoordY(rast, 1, 1) As y1coord,
    ST_RasterToWorldCoordY(rast, 2, 3) As y2coord,
    ST_ScaleY(rast) As pixely
FROM (SELECT rid, ST_SetSkew(rast, 0, 100.5) As rast FROM dummy_rast) As foo;
```

Output

```
rid | y1coord |  y2coord  | pixely
----+---------+-----------+--------
  1 |     0.5 |       107 |      3
  2 | 5793244 | 5793344.4 |  -0.05
```

Siehe auch

ST_ScaleY, ST_RasterToWorldCoordX, ST_SetSkew, ST_SkewY

11.4.14 ST_Rotation

ST_Rotation — Gibt die Rotation des Rasters im Bogenmaß aus.

Synopsis

float8 **ST_Rotation**(raster rast);

Description

Gibt die einheitliche Rotation des Rasters in Radiant aus. Wenn der Raster keine einheitliche Rotation aufweist wird NaN zurückgegeben. Siehe World File für weitere Details.

Beispiele

Code

```
SELECT rid, ST_Rotation(ST_SetScale(ST_SetSkew(rast, sqrt(2)), sqrt(2))) AS rot FROM ↩
    dummy_rast;
```

Output

```
rid |        rot
-----+-------------------
   1 | 0.785398163397448
   2 | 0.785398163397448
```

Siehe auch

ST_SetRotation, ST_SetScale, ST_SetSkew

11.4.15 ST_SkewX

ST_SkewX — Gibt den georeferenzierten Versatz in X-Richtung (oder den Rotationsparameter) aus.

Synopsis

float8 **ST_SkewX**(raster rast);

Description

Gibt den georeferenzierten Versatz in X-Richtung (oder den Rotationsparameter) aus. Siehe World File für weitere Details.

Beispiele

Code

```
WITH refs AS (
    SELECT
        rid,
        format('(%s, %s)', ST_SkewX(rast), ST_SkewY(rast)) AS skew,
        string_to_array(ST_GeoReference(rast), E'\n') AS georef
    FROM dummy_rast
)
SELECT
    rid,
    skew,
    term,
    georef[idx] AS value
FROM refs
CROSS JOIN (VALUES
    (1, 'scaleX'),
    (2, 'skewY'),
    (3, 'skewX'),
    (4, 'scaleY'),
    (5, 'originX'),
    (6, 'originY')
) AS parts(idx, term)
ORDER BY rid, idx;
```

Output

```
rid |  skew  |  term   |       value
----+--------+---------+--------------------
  1 | (0, 0) | scaleX  | 2.0000000000
  1 | (0, 0) | skewY   | 0.0000000000
  1 | (0, 0) | skewX   | 0.0000000000
  1 | (0, 0) | scaleY  | 3.0000000000
  1 | (0, 0) | originX | 0.5000000000
  1 | (0, 0) | originY | 0.5000000000
  2 | (0, 0) | scaleX  | 0.0500000000
  2 | (0, 0) | skewY   | 0.0000000000
  2 | (0, 0) | skewX   | 0.0000000000
  2 | (0, 0) | scaleY  | -0.0500000000
  2 | (0, 0) | originX | 3427927.7500000000
  2 | (0, 0) | originY | 5793244.0000000000
(12 rows)
```

Siehe auch

ST_GeoReference, ST_SkewY, ST_SetSkew

11.4.16 ST_SkewY

ST_SkewY — Gibt den georeferenzierten Versatz in Y-Richtung (oder den Rotationsparameter) aus.

Synopsis

float8 **ST_SkewY**(raster rast);

Description

Gibt den georeferenzierten Versatz in Y-Richtung (oder den Rotationsparameter) aus. Siehe World File für weitere Details.

Beispiele

Code

```
WITH refs AS (
    SELECT
        rid,
        format('(%s, %s)', ST_SkewX(rast), ST_SkewY(rast)) AS skew,
        string_to_array(ST_GeoReference(rast), E'\n') AS georef
    FROM dummy_rast
)
SELECT
    rid,
    skew,
    term,
    georef[idx] AS value
FROM refs
CROSS JOIN (VALUES
    (1, 'scaleX'),
    (2, 'skewY'),
    (3, 'skewX'),
    (4, 'scaleY'),
    (5, 'originX'),
    (6, 'originY')
) AS parts(idx, term)
ORDER BY rid, idx;
```

Output

```
rid |  skew  |  term   |       value
-----+--------+---------+--------------------
   1 | (0, 0) | scaleX  | 2.0000000000
   1 | (0, 0) | skewY   | 0.0000000000
   1 | (0, 0) | skewX   | 0.0000000000
   1 | (0, 0) | scaleY  | 3.0000000000
   1 | (0, 0) | originX | 0.5000000000
   1 | (0, 0) | originY | 0.5000000000
   2 | (0, 0) | scaleX  | 0.0500000000
   2 | (0, 0) | skewY   | 0.0000000000
   2 | (0, 0) | skewX   | 0.0000000000
   2 | (0, 0) | scaleY  | -0.0500000000
   2 | (0, 0) | originX | 3427927.7500000000
   2 | (0, 0) | originY | 5793244.0000000000
(12 rows)
```

Siehe auch

ST_GeoReference, ST_SkewX, ST_SetSkew

11.4.17 ST_SRID

ST_SRID — Gibt den Identifikator des Koordinatenreferenzsystems des Rasters aus, das in der Tabelle "spatial_ref_sys" definiert ist.

Synopsis

integer **ST_SRID**(raster rast);

Description

Gibt den Identifikator des Koordinatenreferenzsystems des Rasters aus, das in der Tabelle "spatial_ref_sys" definiert ist.



Anmerkung
Non-georeferenced rasters and geometries use SRID 0.

Beispiele

Code

```
SELECT ST_SRID(rast) As srid
FROM dummy_rast WHERE rid=1;
```

Output

```
0
```

Siehe auch

Abschnitt 4.5, ST_SRID

11.4.18 ST_Summary

ST_Summary — Gibt eine textliche Zusammenfassung des Rasterinhalts zurück.

Synopsis

text **ST_Summary**(raster rast);

Description

Gibt eine textliche Zusammenfassung des Rasterinhalts zurück

Verfügbarkeit: 2.1.0

Beispiele

Code

```
SELECT ST_Summary(ST_AddBand(ST_AddBand(ST_AddBand(ST_MakeEmptyRaster(10, 10, 0, 0, 1, -1,  ↩
    0, 0, 0),
                1, '8BUI', 1, 0
            ),
            2, '32BF', 0, -9999
        ),
        3, '16BSI', 0, NULL
    )
);
```

Output

```
Raster of 10x10 pixels has 3 bands and extent of BOX(0 -10,10 0)+
     band 1 of pixtype 8BUI is in-db with NODATA value of 0      +
     band 2 of pixtype 32BF is in-db with NODATA value of -9999  +
     band 3 of pixtype 16BSI is in-db with no NODATA value
(1 row)
```

Siehe auch

ST_MetaData, ST_BandMetaData, ST_Summary ST_Extent

11.4.19 ST_UpperLeftX

ST_UpperLeftX — Gibt die obere linke X-Koordinate des Rasters im Koordinatenprojektionssystem aus.

Synopsis

float8 **ST_UpperLeftX**(raster rast);

Description

Gibt die obere linke X-Koordinate des Rasters im Koordinatenprojektionssystem aus.

Beispiele

Code

```
SELECt rid, ST_UpperLeftX(rast) As ulx
FROM dummy_rast;
```

Output

```
rid |    ulx
-----+-----------
   1 |        0.5
   2 | 3427927.75
```

Siehe auch

ST_UpperLeftY, ST_GeoReference, Box3D

11.4.20 ST_UpperLeftY

ST_UpperLeftY — Gibt die obere linke Y-Koordinate des Rasters im Koordinatenprojektionssystem aus.

Synopsis

float8 **ST_UpperLeftY**(raster rast);

Description

Gibt die obere linke Y-Koordinate des Rasters im Koordinatenprojektionssystem aus.

Beispiele

Code

```
SELECT rid, ST_UpperLeftY(rast) As uly
FROM dummy_rast;
```

Output

```
rid |   uly
-----+---------
   1 |     0.5
   2 | 5793244
```

Siehe auch

ST_UpperLeftX, ST_GeoReference, Box3D

11.4.21 ST_Width

ST_Width — Gibt die Breite des Rasters in Pixel aus.

Synopsis

integer **ST_Width**(raster rast);

Description

Gibt die Breite des Rasters in Pixel aus.

Beispiele

Code

```
SELECT ST_Width(rast) As rastwidth
FROM dummy_rast WHERE rid=1;
```

Output

```
10
```

Siehe auch

ST_Height

11.4.22 ST_WorldToRasterCoord

ST_WorldToRasterCoord — Gibt für ein geometrisches X und Y (geographische Länge und Breite) oder für eine Punktgeometrie im Koordinatenreferenzsystem des Rasters, die obere linke Ecke als Spalte und Zeile aus.

Synopsis

record **ST_WorldToRasterCoord**(raster rast, geometry pt);
record **ST_WorldToRasterCoord**(raster rast, double precision longitude, double precision latitude);

Description

Gibt für ein geometrisches X und Y (geographische Länge und Breite), oder für eine Punktgeometrie, die obere linke Ecke als Spalte und Zeile aus. Diese Funktion funktioniert auch dann, wenn sich X und Y, bzw. die Punktgeometrie außerhalb der Ausdehnung des Rasters befindet. X und Y müssen im Koordinatenreferenzsystem des Rasters angegeben werden.

Verfügbarkeit: 2.1.0

Beispiele

Code

```
SELECT
    rid,
    (ST_WorldToRasterCoord(rast, 3427927.8, 20.5)).*,
    (ST_WorldToRasterCoord(rast, ST_GeomFromText('POINT(3427927.8 20.5)', ST_SRID(rast)))).*
FROM dummy_rast;
```

Output

```
rid | columnx |   rowy    | columnx |   rowy
-----+---------+-----------+---------+-----------
   1 | 1713964 |         7 | 1713964 |         7
   2 |       2 | 115864471 |       2 | 115864471
```

Siehe auch

ST_WorldToRasterCoordX, ST_WorldToRasterCoordY, ST_RasterToWorldCoordX, ST_RasterToWorldCoordY, ST_SRID

11.4.23 ST_WorldToRasterCoordX

ST_WorldToRasterCoordX — Gibt für eine Punktgeometrie (pt) oder eine globale X- und Y-Koordinate (xw, yw) die Rasterspalte im globalen Koordinatenreferenzsystem des Rasters aus.

Synopsis

integer **ST_WorldToRasterCoordX**(raster rast, geometry pt);
integer **ST_WorldToRasterCoordX**(raster rast, double precision xw);
integer **ST_WorldToRasterCoordX**(raster rast, double precision xw, double precision yw);

Description

Gibt für eine Punktgeometrie (pt) oder eine globale X- und Y-Koordinate (xw, yw) die Rasterspalte aus. Es werden ein Punkt oder sowohl "xw" als auch "yw" in globalen Koordinaten benötigt, wenn der Raster einen Versatz aufweist. Wenn der Raster keinen Versatz aufweist, ist "xw" ausreichend. Die globalen Koordinaten sind im Koordinatenreferenzsystem des Rasters.

Änderung: 2.1.0 In Vorgängerversionen wurde dies als ST_World2RasterCoordX bezeichnet

Beispiele

Code

```
SELECT rid, ST_WorldToRasterCoordX(rast, 3427927.8) As xcoord,
        ST_WorldToRasterCoordX(rast, 3427927.8, 20.5) As xcoord_xwyw,
        ST_WorldToRasterCoordX(rast, ST_GeomFromText('POINT(3427927.8 20.5)', ST_SRID(rast)) ↩
    ) As ptxcoord
FROM dummy_rast;
```

Output

```
rid | xcoord  |  xcoord_xwyw   | ptxcoord
-----+---------+---------+----------
   1 | 1713964 | 1713964 |  1713964
   2 |       1 |       1 |        1
```

Siehe auch

ST_RasterToWorldCoordX, ST_RasterToWorldCoordY, ST_SRID

11.4.24 ST_WorldToRasterCoordY

ST_WorldToRasterCoordY — Gibt für eine Punktgeometrie (pt) oder eine globale X- und Y-Koordinate (xw, yw) die Rasterzeile im globalen Koordinatenreferenzsystem des Rasters aus.

Synopsis

integer **ST_WorldToRasterCoordY**(raster rast, geometry pt);
integer **ST_WorldToRasterCoordY**(raster rast, double precision xw);
integer **ST_WorldToRasterCoordY**(raster rast, double precision xw, double precision yw);

Description

Gibt für eine Punktgeometrie (pt) oder eine globale X- und Y-Koordinate (xw, yw) die Rasterzeile aus. Es werden ein Punkt oder sowohl "xw" als auch "yw" in globalen Koordinaten benötigt, wenn der Raster einen Versatz aufweist. Wenn der Raster keinen Versatz aufweist, ist "xw" ausreichend. Die globalen Koordinaten sind im Koordinatenreferenzsystem des Rasters.

Änderung: 2.1.0 In Vorgängerversionen wurde dies als ST_World2RasterCoordY bezeichnet

Beispiele

Code

```
SELECT rid, ST_WorldToRasterCoordY(rast, 20.5) As ycoord,
        ST_WorldToRasterCoordY(rast, 3427927.8, 20.5) As ycoord_xwyw,
        ST_WorldToRasterCoordY(rast, ST_GeomFromText('POINT(3427927.8 20.5)', ST_SRID(rast)) ↩
    ) As ptycoord
FROM dummy_rast;
```

Output

```
rid |  ycoord   | ycoord_xwyw | ptycoord
-----+-----------+-------------+-----------
   1 |         7 |           7 |         7
   2 | 115864471 |   115864471 | 115864471
```

Siehe auch

ST_RasterToWorldCoordX, ST_RasterToWorldCoordY, ST_SRID

11.5 Zugriffsfunktionen auf Rasterbänder

11.5.1 ST_BandMetaData

ST_BandMetaData — Gibt die grundlegenden Metadaten eines bestimmten Rasterbandes aus. Wenn der Parameter "bandnum" nicht angegeben ist, wird das 1ste Band angenommen.

Synopsis

table(pixeltype text, nodatavalue double precision, isoutdb boolean, path text, outdbbandnum integer, filesize bigint, filetimestamp bigint) **ST_BandMetaData**(raster rast, integer band=1);
table(bandnum integer, pixeltype text, nodatavalue double precision, isoutdb boolean, path text, outdbbandnum integer, filesize bigint, filetimestamp bigint) **ST_BandMetaData**(raster rast, integer[] band);

Description

Gibt die grundlegenden Metadaten eines Rasterbandes aus. Die zurückgegebenen Spalten sind pixeltype, nodatavalue, isoutdb, path, filesize und filetimestamp.



Anmerkung
Wenn der Raster keine Bänder beinhaltet wird ein Fehler gemeldet.



Anmerkung
Wenn für das Band kein NODATA-Wert vergeben wurde, wird der NODATA-Wert auf NULL gesetzt.



Anmerkung
Wenn isoutdb False ist, dann sind path, outdbbandnum, filesize und filetimestamp NULL. Wenn der Zugriff auf otdb-Raster deaktiviert ist, dann sind filesize und filetimestamp ebenfalls NULL.

Erweiterung: 2.5.0 inkludiert jetzt*outdbbandnum*, *filesize* und *filetimestamp* für outdb Raster.

Beispiele

Variant 1.

Code

```
SELECT
    rid,
    (foo.md).*
FROM (
    SELECT
        rid,
        ST_BandMetaData(rast, 1) AS md
    FROM dummy_rast
    WHERE rid=2
) As foo;
```

Output

```
rid | pixeltype | nodatavalue | isoutdb | path | outdbbandnum
-----+-----------+-------------+---------+------+--------------
   2 | 8BUI      |           0 | f       |      |
(1 row)
```

Variant 2.

Code

```
WITH foo AS (
    SELECT
        ST_AddBand(NULL::raster, '/home/pele/devel/geo/postgis-git/raster/test/regress/ ↩
    loader/Projected.tif', NULL::int[]) AS rast
)
SELECT
    bandnum,
    pixeltype,
    isoutdb,
    regexp_replace(path, '^.*/', '') AS file,
    outdbbandnum
FROM ST_BandMetaData((SELECT rast FROM foo),
    ARRAY[1, 3, 2]::int[]
);
```

Output

```
bandnum | pixeltype | isoutdb |     file      | outdbbandnum
---------+-----------+---------+---------------+--------------
       1 | 8BUI      | t       | Projected.tif |            1
       3 | 8BUI      | t       | Projected.tif |            3
       2 | 8BUI      | t       | Projected.tif |            2
```

Siehe auch

ST_MetaData, ST_BandPixelType

11.5.2 ST_BandNoDataValue

ST_BandNoDataValue — Gibt den NODATA Wert des gegebenen Bandes aus. Wenn der Parameter "bandnum" nicht angegeben ist, wird das 1ste Band angenommen.

Synopsis

double precision **ST_BandNoDataValue**(raster rast, integer bandnum=1);

Description

Gibt den NODATA Wert des Bandes aus.

Beispiele

Code

```
SELECT ST_BandNoDataValue(rast, 1) As bnval1,
    ST_BandNoDataValue(rast, 2) As bnval2, ST_BandNoDataValue(rast, 3) As bnval3
FROM dummy_rast
WHERE rid = 2;
```

Output

```
bnval1 | bnval2 | bnval3
--------+--------+--------
      0 |      0 |      0
```

Siehe auch

ST_NumBands

11.5.3 ST_BandIsNoData

ST_BandIsNoData — Gibt TRUE aus, wenn das Band ausschließlich aus NODATA Werten besteht.

Synopsis

boolean **ST_BandIsNoData**(raster rast, integer band, boolean forceChecking=false);
boolean **ST_BandIsNoData**(raster rast, boolean forceChecking=false);

Description

Gibt TRUE aus, wenn das Band ausschließlich aus NODATA Werten besteht. Wenn kein Band angegeben ist, wird Band 1 angenommen. Wenn der letzte Übergabewert TRUE ist, dann wird das gesamte Band, Pixel für Pixel, überprüft. Sonst gibt die Funktion nur den Wert der Flag "isnodata" für das Band aus. Wenn dieser Parameter nicht gesetzt wurde, wird der Standardwert "FALSE" angenommen.

Verfügbarkeit: 2.0.0



Anmerkung
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.

Beispiele

This example creates a dummy table with one raster column. The raster has two bands and one pixel per band. In the first band, the NODATA value and pixel value are both 3. In the second band, the NODATA value is 13 and the pixel value is 4.

Code

```
create table dummy_rast (rid integer, rast raster);

insert into dummy_rast values(
1,
(
'01' -- little endian (uint8 ndr)
||
'0000' -- version (uint16 0)
||
'0200' -- nBands (uint16 0)
||
'17263529ED684A3F' -- scaleX (float64 0.000805965234044584)
||
'F9253529ED684ABF' -- scaleY (float64 -0.000805965234044584)
||
'1C9F33CE69E352C0' -- ipX (float64 -75.5533328537098)
||
'718F0E9A27A44840' -- ipY (float64 49.2824585505576)
||
'ED50EB853EC32B3F' -- skewX (float64 0.000211812383858707)
||
'7550EB853EC32B3F' -- skewY (float64 0.000211812383858704)
||
'E6100000' -- SRID (int32 4326)
||
'0100' -- width (uint16 1)
||
'0100' -- height (uint16 1)
||
'6' -- hasnodatavalue and isnodata value set to true.
||
'2' -- first band type (4BUI)
||
'03' -- novalue==3
||
'03' -- pixel(0,0)==3 (same that nodata)
||
'0' -- hasnodatavalue set to false
```

```
||
'5' -- second band type (16BSI)
||
'0D00' -- novalue==13
||
'0400' -- pixel(0,0)==4
)::raster
);

select ST_BandIsNoData(rast, 1) from dummy_rast where rid = 1; -- Expected true
select ST_BandIsNoData(rast, 2) from dummy_rast where rid = 1; -- Expected false
```

Siehe auch

ST_BandNoDataValue, ST_NumBands, ST_SetBandNoDataValue, ST_SetBandIsNoData

11.5.4 ST_BandPath

ST_BandPath — Gibt den Dateipfad aus, unter dem das Band im Dateisystem gespeichert ist. Wenn "bandnum" nicht angegeben ist, wird 1 angenommen.

Synopsis

text **ST_BandPath**(raster rast, integer bandnum=1);

Description

Gibt den Dateipfad des Bandes im System aus. Wenn man die Funktion mit einem "in-db"-Rasterband aufruft, wird eine Fehlermeldung ausgegeben.

Beispiele

Code

```
WITH foo AS (
    SELECT ST_AddBand(
        NULL::raster,
        '/home/pele/devel/geo/postgis-git/raster/test/regress/loader/Projected.tif',
        NULL::int[]
    ) AS rast
)
SELECT bandnum, ST_BandPath(rast, bandnum) AS path
FROM foo
CROSS JOIN generate_series(1, 3) AS bandnum;
```

Output

```
bandnum |                                  path
---------+-----------------------------------------------------------------------------
       1 | /home/pele/devel/geo/postgis-git/raster/test/regress/loader/Projected.tif
       2 | /home/pele/devel/geo/postgis-git/raster/test/regress/loader/Projected.tif
       3 | /home/pele/devel/geo/postgis-git/raster/test/regress/loader/Projected.tif
```

Siehe auch

ST_BandMetaData, ST_SetBandPath, ST_SetBandIndex

11.5.5 ST_BandFileSize

ST_BandFileSize — Gibt die Dateigröße eines im Dateisystem gespeicherten Bandes aus. Wenn "bandnum" nicht angegeben ist, wird 1 angenommen.

Synopsis

bigint **ST_BandFileSize**(raster rast, integer bandnum=1);

Description

Gibt die Dateigröße eines im Dateisystem gespeicherten Bandes aus. Wenn die Funktion mit einem indb-Band aufgerufen wird oder der outdb-Zugriff deaktiviert ist, wird eine Fehlermeldung ausgegeben.

Diese Funkction wird üblicherweise gemeinsam imit ST_BandPath() und ST_BandFileTimestamp() verwendet. Auf diese Weise kann ein Client feststellen, ob er die selbe Sicht auf den Dateinamen eines outdb-Rasters hat wie der Server.

Verfügbarkeit: 2.5.0

Beispiele

Code

```
SELECT ST_BandFileSize(rast, 1) FROM dummy_rast WHERE rid = 1;
```

Output

```
240574
```

11.5.6 ST_BandFileTimestamp

ST_BandFileTimestamp — Gibt den Zeitstempel eines im Dateisystem gespeicherten Bandes aus. Wenn "bandnum" nicht angegeben ist, wird 1 angenommen.

Synopsis

bigint **ST_BandFileTimestamp**(raster rast, integer bandnum=1);

Description

Gibt den Zeitstempel (Anzahl der Sekunden seit Jan 1st 1970 00:00:00 UTC) eines im Dateisystem gespeicherten Bandes aus. Wenn die Funktion mit einem indb-Band aufgerufen wird oder der outdb-Zugriff deaktiviert ist, wird eine Fehlermeldung ausgegeben.

Diese Funkction wird üblicherweise gemeinsam imit ST_BandPath() und ST_BandFileSize() verwendet. Auf diese Weise kann ein Client feststellen, ob er die selbe Sicht auf den Dateinamen eines outdb-Rasters hat wie der Server.

Verfügbarkeit: 2.5.0

Beispiele

Code

```
SELECT ST_BandFileTimestamp(rast, 1) FROM dummy_rast WHERE rid = 1;
```

Output

```
1521807257
```

11.5.7 ST_BandPixelType

ST_BandPixelType — Gibt den Pixeltyp des angegebenen Bandes aus. Wenn der Parameter "bandnum" nicht angegeben ist, wird das 1ste Band angenommen.

Synopsis

text **ST_BandPixelType**(raster rast, integer bandnum=1);

Description

Gibt eine Beschreibung des Datentyps und der Größe der Zellwerte in dem gegebenen Band zurück.

Im Folgenden die 11 unterstützten Pixeltypen

- 1BB - 1-Bit Boolean
- 2BUI - 2-Bit vorzeichenlose Ganzzahl
- 4BUI - 4-Bit vorzeichenlose Ganzzahl
- 8BSI - 8-Bit Ganzzahl
- 8BUI - 8-Bit vorzeichenlose Ganzzahl
- 16BSI - 16-Bit Ganzzahl
- 16BUI - 16-bit vorzeichenlose Ganzzahl
- 32BSI - 32-Bit Ganzzahl
- 32BUI - 32-Bit vorzeichenlose Ganzzahl
- 32BF - 32-Bit Float
- 64BF - 64-Bit Float

Beispiele

Code

```
SELECT ST_BandPixelType(rast, 1) As btype1,
    ST_BandPixelType(rast, 2) As btype2, ST_BandPixelType(rast, 3) As btype3
FROM dummy_rast
WHERE rid = 2;
```

Output

```
btype1 | btype2 | btype3
--------+---------+--------
 8BUI   | 8BUI    | 8BUI
```

Siehe auch

ST_NumBands

11.5.8 ST_MinPossibleValue

ST_MinPossibleValue — Gibt den Mindestwert zurück, den dieser Pixeltyp speichern kann.

Synopsis

integer **ST_MinPossibleValue**(text pixeltype);

Description

Gibt den Mindestwert zurück, den dieser Pixeltyp speichern kann.

Beispiele

Code

```
SELECT ST_MinPossibleValue('16BSI');
```

Output

```
-32768
```

Code

```
SELECT ST_MinPossibleValue('8BUI');
```

Output

```
0
```

Siehe auch

ST_BandPixelType

11.5.9 ST_HasNoBand

ST_HasNoBand — Gibt TRUE aus, wenn kein Band mit der angegebenen Bandnummer existiert. Gibt den Pixeltyp des angegebenen Bandes aus. Wenn keine Bandnummer angegeben ist, wird das 1ste Band angenommen.

Synopsis

boolean **ST_HasNoBand**(raster rast, integer bandnum=1);

Description

Gibt TRUE aus, wenn kein Band mit der angegebenen Bandnummer existiert. Gibt den Pixeltyp des angegebenen Bandes aus. Wenn keine Bandnummer angegeben ist, wird das 1ste Band angenommen.

Verfügbarkeit: 2.0.0

Beispiele

Code

```
SELECT rid, ST_HasNoBand(rast) As hb1, ST_HasNoBand(rast, 2) as hb2,
ST_HasNoBand(rast, 4) as hb4, ST_NumBands(rast) As numbands
FROM dummy_rast;
```

Output

```
rid | hb1 | hb2 | hb4 | numbands
-----+-----+-----+-----+----------
1 | t   | t   | t   |        0
2 | f   | f   | t   |        3
```

Siehe auch

ST_NumBands

11.6 Zugriffsfunktionen und Änderungsmethoden für Rasterpixel

11.6.1 ST_PixelAsPolygon

ST_PixelAsPolygon — Gibt die Polygongeometrie aus, die das Pixel einer bestimmten Zeile und Spalte begrenzt.

Synopsis

geometry **ST_PixelAsPolygon**(raster rast, integer columnx, integer rowy);

Description

Gibt die Polygongeometrie aus, die das Pixel einer bestimmten Zeile und Spalte begrenzt.

Verfügbarkeit: 2.0.0

Beispiele

This example returns a raster pixel polygon.

Code

```
SELECT i, j, ST_PixelAsPolygon(foo.rast, i, j) As b1pgeom
FROM dummy_rast As foo
    CROSS JOIN generate_series(1, 2) As i
    CROSS JOIN generate_series(1, 1) As j
WHERE rid=2;
```

Output

```
i | j |                                                  blpgeom
---+---+-------------------------------------------------------------------------------
 1 | 1 | POLYGON((3427927.75 5793244,3427927.8 5793244,3427927.8 5793243.95,...
 2 | 1 | POLYGON((3427927.8 5793244,3427927.85 5793244,3427927.85 5793243.95, ..
```

Siehe auch

ST_DumpAsPolygons, ST_PixelAsPolygons, ST_PixelAsPoint, ST_PixelAsPoints, ST_PixelAsCentroid, ST_PixelAsCentroids, ST_Intersection, ST_AsText

11.6.2 ST_PixelAsPolygons

ST_PixelAsPolygons — Gibt die umhüllende Polygongeometrie, den Zellwert, sowie die X- und Y-Rasterkoordinate für jedes Pixel aus.

Synopsis

table(geom geometry, val double precision, x integer, y integer) **ST_PixelAsPolygons**(raster rast, integer band=1, boolean exclude_nodata_value=TRUE);

Description

Gibt die umhüllende Polygongeometrie, den Zellwert (double precision), sowie die X- und Y-Rasterkoordinate (Ganzzahl) für jedes Pixel aus.

Datensatzformatausgabe: *geom* geometry, *val* double precision, *x* integer, *y* integers.



Anmerkung
When *exclude_nodata_value* is TRUE, only pixels whose values are not NODATA are returned. Unlike ST_DumpAsPolygons, this function returns one polygon per pixel rather than merging adjacent pixels with the same value.

Verfügbarkeit: 2.0.0

Erweiterung: 2.1.0 Der optionale Übergabewert "exclude_nodata_value" wurde hinzugefügt.

Änderung: 2.1.1 Die Verhaltensweise von "exclude_nodata_value" wurde geändert.

Beispiele

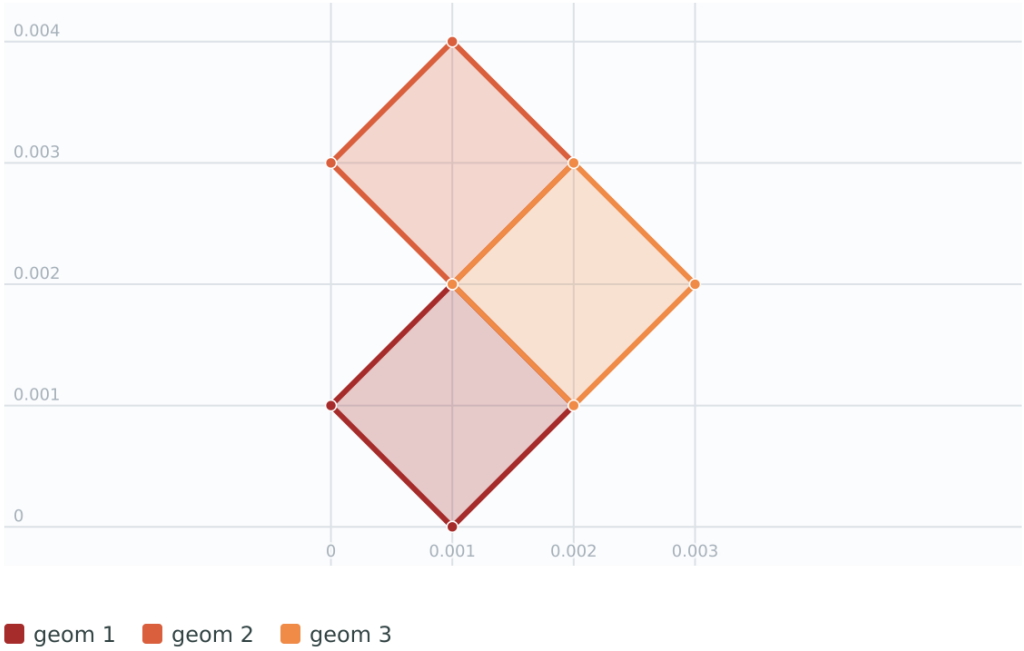
This example returns a raster pixel polygon.

Code

```
WITH raster AS (
    SELECT ST_SetValue(
        ST_SetValue(
            ST_AddBand(
                ST_MakeEmptyRaster(
                    2, 2, 0, 0, 0.001, -0.001, 0.001, 0.001, 4269
                ),
                1, '8BUI', 1, 0
            ),
            2, 2, 10
```

```
        ),
        1, 1, NULL
    ) AS rast
), pixels AS (
    SELECT (ST_PixelAsPolygons(rast)).*
    FROM raster
)
SELECT x, y, val, geom AS geom
FROM pixels
ORDER BY x, y;
```

Figure



Siehe auch

ST_DumpAsPolygons, ST_PixelAsPolygon, ST_PixelAsPoint, ST_PixelAsPoints, ST_PixelAsCentroid, ST_PixelAsCentroids, ST_AsText

11.6.3 ST_PixelAsPoint

ST_PixelAsPoint — Gibt eine Punktgeometrie der oberen linken Ecke des Rasters zurück.

Synopsis

geometry **ST_PixelAsPoint**(raster rast, integer columnx, integer rowy);

Description

Gibt eine Punktgeometrie der oberen linken Ecke des Rasters zurück.

Verfügbarkeit: 2.1.0

Beispiele

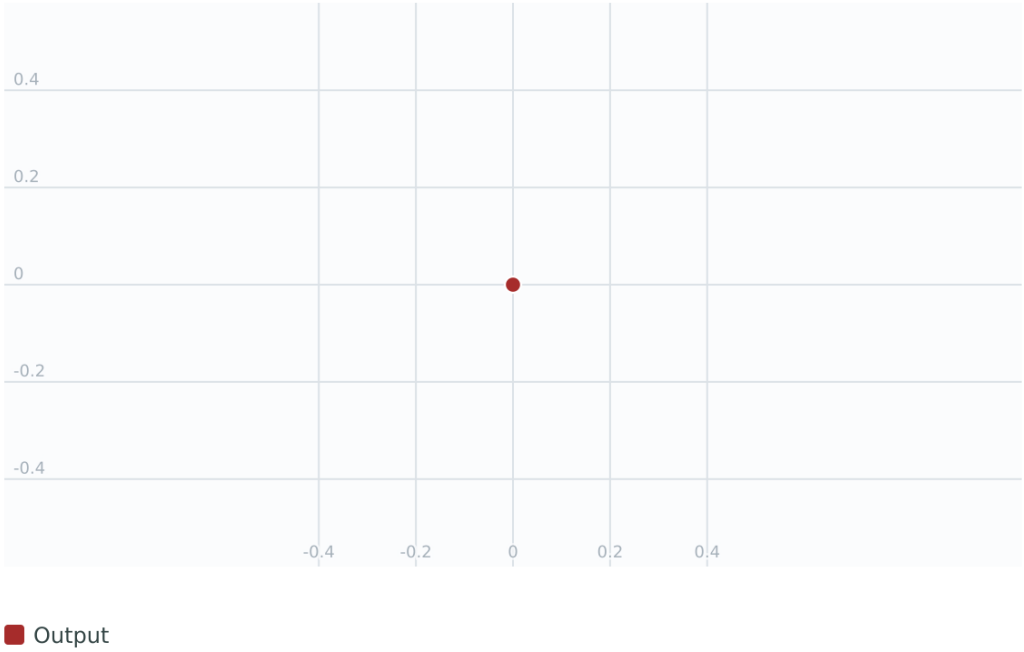
Code

```
SELECT ST_PixelAsPoint(rast, 1, 1) FROM dummy_rast WHERE rid = 1;
```

Output

```
POINT(0.5 0.5)
```

Figure



Siehe auch

ST_DumpAsPolygons, ST_PixelAsPolygon, ST_PixelAsPolygons, ST_PixelAsPoints, ST_PixelAsCentroid, ST_PixelAsCentroids

11.6.4 ST_PixelAsPoints

ST_PixelAsPoints — Gibt eine Punktgeometrie für jedes Pixel des Rasterbandes zurück, zusammen mit dem Zellwert und den X- und Y-Rasterkoordinaten eines jeden Pixels. Die Koordinaten der Punkte entsprechen dem oberen linken Eck der Pixel.

Synopsis

table(geom geometry, val double precision, x integer, y integer) **ST_PixelAsPoints**(raster rast, integer band=1, boolean exclude_nodata_value=TRUE);

Description

Gibt eine Punktgeometrie für jedes Pixel des Rasterbandes zurück, zusammen mit dem Zellwert und den X- und Y-Rasterkoordinaten eines jeden Pixels. Die Koordinaten der Punkte entsprechen dem oberen linken Eck der Pixel.

Datensatzformatausgabe: *geom* geometry, *val* double precision, *x* integer, *y* integers.



Anmerkung
Wenn *exclude_nodata_value* = TRUE, dann werden nur jene Pixel als Punkte zurückgegeben deren Zellwerte nicht NODATA sind.

Verfügbarkeit: 2.1.0

Änderung: 2.1.1 Die Verhaltensweise von "exclude_nodata_value" wurde geändert.

Beispiele

This example returns point features together with a derived marker size that a client can use for value-scaled point rendering.

Code

```
SELECT
    x,
    y,
    val,
    round((val / 32.0)::numeric, 2) AS marker_scale,geom AS geom
FROM (
    SELECT (ST_PixelAsPoints(rast, 1)).*
    FROM dummy_rast
    WHERE rid = 2
) AS foo
ORDER BY y, x;
```

Output

```
x | y | val | marker_scale |          geom
---+---+-----+--------------+------------------------------
 1 | 1 | 253 |         7.91 | POINT(3427927.75 5793244)
 2 | 1 | 254 |         7.94 | POINT(3427927.8 5793244)
 3 | 1 | 253 |         7.91 | POINT(3427927.85 5793244)
 4 | 1 | 254 |         7.94 | POINT(3427927.9 5793244)
 5 | 1 | 254 |         7.94 | POINT(3427927.95 5793244)
 1 | 2 | 253 |         7.91 | POINT(3427927.75 5793243.95)
 2 | 2 | 254 |         7.94 | POINT(3427927.8 5793243.95)
 3 | 2 | 254 |         7.94 | POINT(3427927.85 5793243.95)
 4 | 2 | 253 |         7.91 | POINT(3427927.9 5793243.95)
 5 | 2 | 249 |         7.78 | POINT(3427927.95 5793243.95)
 1 | 3 | 250 |         7.81 | POINT(3427927.75 5793243.9)
 2 | 3 | 254 |         7.94 | POINT(3427927.8 5793243.9)
 3 | 3 | 254 |         7.94 | POINT(3427927.85 5793243.9)
 4 | 3 | 252 |         7.88 | POINT(3427927.9 5793243.9)
 5 | 3 | 249 |         7.78 | POINT(3427927.95 5793243.9)
 1 | 4 | 251 |         7.84 | POINT(3427927.75 5793243.85)
 2 | 4 | 253 |         7.91 | POINT(3427927.8 5793243.85)
 3 | 4 | 254 |         7.94 | POINT(3427927.85 5793243.85)
 4 | 4 | 254 |         7.94 | POINT(3427927.9 5793243.85)
 5 | 4 | 253 |         7.91 | POINT(3427927.95 5793243.85)
 1 | 5 | 252 |         7.88 | POINT(3427927.75 5793243.8)
 2 | 5 | 250 |         7.81 | POINT(3427927.8 5793243.8)
 3 | 5 | 254 |         7.94 | POINT(3427927.85 5793243.8)
 4 | 5 | 254 |         7.94 | POINT(3427927.9 5793243.8)
 5 | 5 | 254 |         7.94 | POINT(3427927.95 5793243.8)
```

Figure



Siehe auch

ST_DumpAsPolygons, ST_PixelAsPolygon, ST_PixelAsPolygons, ST_PixelAsPoint, ST_PixelAsCentroid, ST_PixelAsCentroids

11.6.5 ST_PixelAsCentroid

ST_PixelAsCentroid — Gibt den geometrischen Schwerpunkt (Punktgeometrie) der Fläche aus, die durch das Pixel repräsentiert wird.

Synopsis

geometry **ST_PixelAsCentroid**(raster rast, integer x, integer y);

Description

Gibt den geometrischen Schwerpunkt (Punktgeometrie) der Fläche aus, die durch das Pixel repräsentiert wird.

Verbessert: 3.2.0 Schneller jetzt in C implementiert.

Verfügbarkeit: 2.1.0

Beispiele

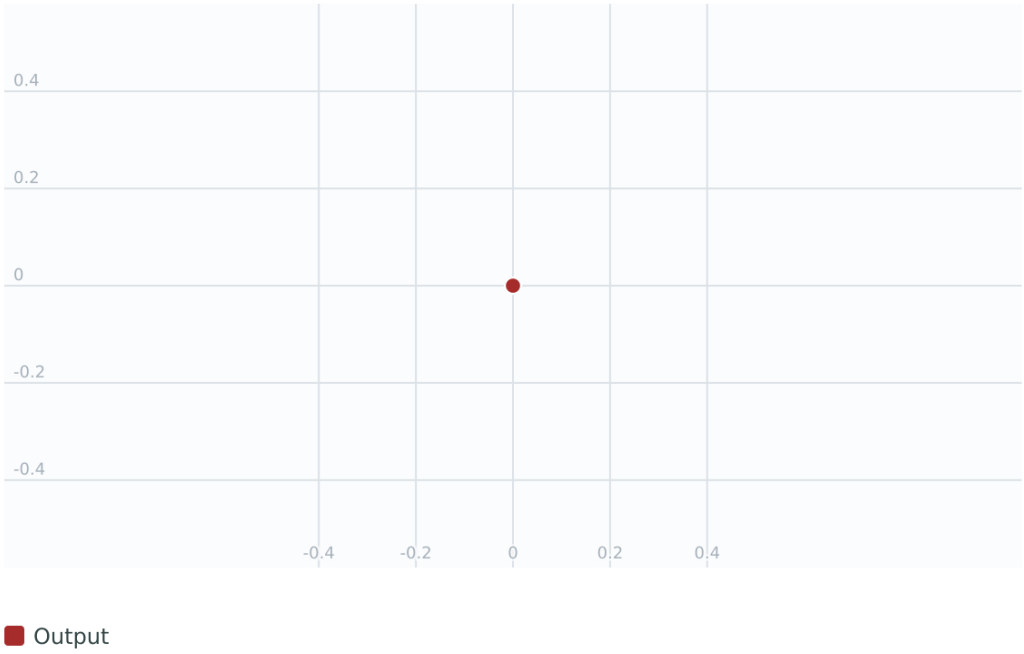
Code

```
SELECT ST_PixelAsCentroid(rast, 1, 1) FROM dummy_rast WHERE rid = 1;
```

Output

```
POINT(1.5 2)
```

Figure



Siehe auch

ST_DumpAsPolygons, ST_PixelAsPolygon, ST_PixelAsPolygons, ST_PixelAsPoint, ST_PixelAsPoints, ST_PixelAsCentroids

11.6.6 ST_PixelAsCentroids

ST_PixelAsCentroids — Gibt den geometrischen Schwerpunkt (Punktgeometrie) für jedes Pixel des Rasterbandes zurück, zusammen mit dem Zellwert und den X- und Y-Rasterkoordinaten eines jeden Pixels. Die Koordinaten der Punkte entsprechen dem geometrischen Schwerpunkt der Pixel.

Synopsis

table(geom geometry, val double precision, x integer, y integer) **ST_PixelAsCentroids**(raster rast, integer band=1, boolean exclude_nodata_value=TRUE);

Description

Gibt den geometrischen Schwerpunkt (Punktgeometrie) für jedes Pixel des Rasterbandes zurück, zusammen mit dem Zellwert und den X- und Y-Rasterkoordinaten eines jeden Pixels. Die Koordinaten der Punkte entsprechen dem geometrischen Schwerpunkt der Pixel.

Datensatzformatausgabe: `geom` geometry, `val` double precision, `x` integer, `y` integers.



Anmerkung
Wenn `exclude_nodata_value` = TRUE, dann werden nur jene Pixel als Punkte zurückgegeben deren Zellwerte nicht NODATA sind.

Verbessert: 3.2.0 Schneller jetzt in C implementiert.

Änderung: 2.1.1 Die Verhaltensweise von "exclude_nodata_value" wurde geändert.

Verfügbarkeit: 2.1.0

Beispiele

This example uses LATERAL syntax.

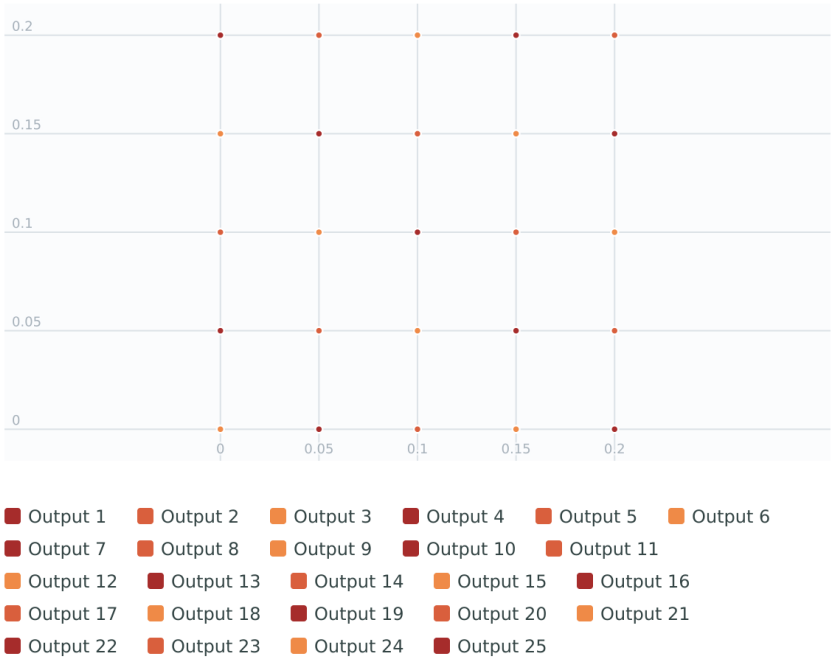
Code

```
SELECT x, y, val,geom
    FROM (SELECT dp.* FROM dummy_rast, LATERAL ST_PixelAsCentroids(rast, 1) AS dp WHERE rid ↩
    = 2) foo;
```

Output

```
x | y | val |           st_astext
---+---+-----+-------------------------------
 1 | 1 | 253 | POINT(3427927.775 5793243.975)
 2 | 1 | 254 | POINT(3427927.825 5793243.975)
 3 | 1 | 253 | POINT(3427927.875 5793243.975)
 4 | 1 | 254 | POINT(3427927.925 5793243.975)
 5 | 1 | 254 | POINT(3427927.975 5793243.975)
 1 | 2 | 253 | POINT(3427927.775 5793243.925)
 2 | 2 | 254 | POINT(3427927.825 5793243.925)
 3 | 2 | 254 | POINT(3427927.875 5793243.925)
 4 | 2 | 253 | POINT(3427927.925 5793243.925)
 5 | 2 | 249 | POINT(3427927.975 5793243.925)
 1 | 3 | 250 | POINT(3427927.775 5793243.875)
 2 | 3 | 254 | POINT(3427927.825 5793243.875)
 3 | 3 | 254 | POINT(3427927.875 5793243.875)
 4 | 3 | 252 | POINT(3427927.925 5793243.875)
 5 | 3 | 249 | POINT(3427927.975 5793243.875)
 1 | 4 | 251 | POINT(3427927.775 5793243.825)
 2 | 4 | 253 | POINT(3427927.825 5793243.825)
 3 | 4 | 254 | POINT(3427927.875 5793243.825)
 4 | 4 | 254 | POINT(3427927.925 5793243.825)
 5 | 4 | 253 | POINT(3427927.975 5793243.825)
 1 | 5 | 252 | POINT(3427927.775 5793243.775)
 2 | 5 | 250 | POINT(3427927.825 5793243.775)
 3 | 5 | 254 | POINT(3427927.875 5793243.775)
 4 | 5 | 254 | POINT(3427927.925 5793243.775)
 5 | 5 | 254 | POINT(3427927.975 5793243.775)
```

Figure



Siehe auch

ST_DumpAsPolygons, ST_PixelAsPolygon, ST_PixelAsPolygons, ST_PixelAsPoint, ST_PixelAsPoints, ST_PixelAsCentroid

11.6.7 ST_Value

ST_Value — Gibt den Zellwert eines Pixels aus, das über columnx und rowy oder durch einen bestimmten geometrischen Punkt angegeben wird. Die Bandnummern beginnen mit 1 und wenn keine Bandnummer angegeben ist, dann wird Band 1 angenommen. Wenn `exclude_nodata_value` auf FALSE gesetzt ist, werden auch die Pixel mit einem `nodata` Wert mit einbezogen. Wenn `exclude_nodata_value` nicht übergeben wird, dann wird er über die Metadaten des Rasters ausgelesen.

Synopsis

double precision **ST_Value**(raster rast, geometry pt, boolean exclude_nodata_value=true);
double precision **ST_Value**(raster rast, integer band, geometry pt, boolean exclude_nodata_value=true, text resample='nearest');
double precision **ST_Value**(raster rast, integer x, integer y, boolean exclude_nodata_value=true);
double precision **ST_Value**(raster rast, integer band, integer x, integer y, boolean exclude_nodata_value=true);

Description

Gibt den Wert eines bestimmten Bandes in einer bestimmten Spaltex, Zeiley Pixel oder an einem bestimmten Geometriepunkt zurück. Die Bandnummern beginnen bei 1 und es wird angenommen, dass Band 1 ist, wenn nicht angegeben.

Wenn `exclude_nodata_value` auf true gesetzt ist, dann werden nur Pixel berücksichtigt, die nicht `nodata` sind. Wenn `exclude_nodata_value` auf false gesetzt ist, werden alle Pixel berücksichtigt.

The allowed values of the `resample` parameter are "nearest" which performs the default nearest-neighbor resampling, "bilinear" which performs a bilinear interpolation to estimate the value between pixel centers, and the nearest-neighbor boundary options "nearest-UL", "nearest-UR", "nearest-LL", and "nearest-LR". Boundary options choose which pixel is returned when the point lies on a horizontal or vertical pixel boundary, including corner intersections.

Enhanced: 3.7.0 resample accepts nearest-neighbor boundary options "nearest-UL", "nearest-UR", "nearest-LL", and "nearest-LR".

Verbessert: 3.2.0 Das optionale Argument resample wurde hinzugefügt.

Erweiterung: 2.0.0 Der optionale Übergabewert "exclude_nodata_value" wurde hinzugefügt.

Beispiele

Get raster values at particular postgis geometry points. The srid of your geometry should be same as for your raster.

Code

```
SELECT rid, ST_Value(rast, foo.pt_geom) As b1pval, ST_Value(rast, 2, foo.pt_geom) As b2pval
FROM dummy_rast CROSS JOIN (SELECT ST_SetSRID(ST_Point(3427927.77, 5793243.76), 0) As  ↩
    pt_geom) As foo
WHERE rid=2;
```

Output

```
rid | b1pval | b2pval
-----+--------+--------
   2 |    252 |     79
```

General fictitious example using a real table.

Code

```
SELECT rid, ST_Value(rast, 3, sometable.geom) As b3pval
FROM sometable
WHERE ST_Intersects(rast, sometable.geom);
```

Code

```
SELECT rid, ST_Value(rast, 1, 1, 1) As b1pval,
    ST_Value(rast, 2, 1, 1) As b2pval, ST_Value(rast, 3, 1, 1) As b3pval
FROM dummy_rast
WHERE rid=2;
```

Output

```
rid | b1pval | b2pval | b3pval
-----+--------+--------+--------
   2 |    253 |     78 |     70
```

Get all values in bands 1,2,3 of each pixel.

Code

```
SELECT x, y, ST_Value(rast, 1, x, y) As b1val,
    ST_Value(rast, 2, x, y) As b2val, ST_Value(rast, 3, x, y) As b3val
FROM dummy_rast CROSS JOIN
generate_series(1, 1000) As x CROSS JOIN generate_series(1, 1000) As y
WHERE rid =  2 AND x <= ST_Width(rast) AND y <= ST_Height(rast);
```

Output

```
 x | y | b1val | b2val | b3val
---+---+-------+-------+-------
 1 | 1 |   253 |    78 |    70
 1 | 2 |   253 |    96 |    80
 1 | 3 |   250 |    99 |    90
 1 | 4 |   251 |    89 |    77
 1 | 5 |   252 |    79 |    62
 2 | 1 |   254 |    98 |    86
 2 | 2 |   254 |   118 |   108
 :
 :
```

Get all values in bands 1,2,3 of each pixel same as above but returning the upper left point point of each pixel.

Code

```
SELECT ST_SetSRID(ST_Point(ST_UpperLeftX(rast) + ST_ScaleX(rast)*x,
        ST_UpperLeftY(rast) + ST_ScaleY(rast)*y),
        ST_SRID(rast)) As uplpt,
    ST_Value(rast, 1, x, y) As b1val,
    ST_Value(rast, 2, x, y) As b2val, ST_Value(rast, 3, x, y) As b3val
FROM dummy_rast CROSS JOIN
generate_series(1,1000) As x CROSS JOIN generate_series(1,1000) As y
WHERE rid =  2 AND x <= ST_Width(rast) AND y <= ST_Height(rast);
```

Output

```
           uplpt            | b1val | b2val | b3val
----------------------------+-------+-------+-------
 POINT(3427929.25 5793245.5) |   253 |    78 |    70
 POINT(3427929.25 5793247)   |   253 |    96 |    80
 POINT(3427929.25 5793248.5) |   250 |    99 |    90
:
```

Get a polygon formed by union of all pixels that fall in a particular value range and intersect a particular polygon.

Code

```
SELECT ST_Union(pixpolyg) As shadow
FROM (SELECT ST_Translate(ST_MakeEnvelope(ST_UpperLeftX(rast), ST_UpperLeftY(rast),
            ST_UpperLeftX(rast) + ST_ScaleX(rast),
            ST_UpperLeftY(rast) + ST_ScaleY(rast), 0
            ), ST_ScaleX(rast)*x, ST_ScaleY(rast)*y
        ) As pixpolyg, ST_Value(rast, 2, x, y) As b2val
    FROM dummy_rast CROSS JOIN
generate_series(1,1000) As x CROSS JOIN generate_series(1,1000) As y
WHERE rid =  2
    AND x <= ST_Width(rast) AND y <= ST_Height(rast)) As foo
WHERE
    ST_Intersects(
        pixpolyg,
        ST_GeomFromText('POLYGON((3427928 5793244,3427927.75 5793243.75,3427928  ↩
    5793243.75,3427928 5793244))',0)
        ) AND b2val != 254;
```

Output

```
        shadow
-------------------------------------------------------------------------------
 MULTIPOLYGON(((3427928 5793243.9,3427928 5793243.85,3427927.95 5793243.85,3427927.95 5793243.9,
 3427927.95 5793243.95,3427928 5793243.95,3427928.05 5793243.95,3427928.05 5793243.9,3427928 5793243.9)),((3427927.95  ↩
     5793243.9,3427927.95 579324
3.85,3427927.9 5793243.85,3427927.85 5793243.85,3427927.85 5793243.9,3427927.9 5793243.9,3427927.9 5793243.95,
3427927.95 5793243.95,3427927.95 5793243.9)),((3427927.85 5793243.75,3427927.85 5793243.7,3427927.8 5793243.7,3427927.8 ↩
      5793243.75,
3427927.8 5793243.8,3427927.8 5793243.85,3427927.85 5793243.85,3427927.85 5793243.8,3427927.85 5793243.75)),
((3427928.05 5793243.75,3427928.05 5793243.7,3427928 5793243.7,3427927.95 5793243.7,3427927.95 5793243.75,3427927.95  ↩
     5793243.8,3427
927.95 5793243.85,3427928 5793243.85,3427928 5793243.8,3427928.05 5793243.8,
3427928.05 5793243.75)),((3427927.95 5793243.75,3427927.95 5793243.7,3427927.9 5793243.7,3427927.85 5793243.7,
3427927.85 5793243.75,3427927.85 5793243.8,3427927.85 5793243.85,3427927.9 5793243.85,
3427927.95 5793243.85,3427927.95 5793243.8,3427927.95 5793243.75)))
```

Checking all the pixels of a large raster tile can take a long time. You can dramatically improve speed at some 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.

Code

```
SELECT ST_Union(pixpolyg) As shadow
FROM (SELECT ST_Translate(ST_MakeEnvelope(ST_UpperLeftX(rast), ST_UpperLeftY(rast),
            ST_UpperLeftX(rast) + ST_ScaleX(rast)*2,
            ST_UpperLeftY(rast) + ST_ScaleY(rast)*2, 0
            ), ST_ScaleX(rast)*x, ST_ScaleY(rast)*y
        ) As pixpolyg, ST_Value(rast, 2, x, y) As b2val
    FROM dummy_rast CROSS JOIN
generate_series(1,1000,2) As x CROSS JOIN generate_series(1,1000,2) As y
WHERE rid =  2
    AND x <= ST_Width(rast)  AND y <= ST_Height(rast)  ) As foo
WHERE
    ST_Intersects(
        pixpolyg,
        ST_GeomFromText('POLYGON((3427928 5793244,3427927.75 5793243.75,3427928 ↩
    5793243.75,3427928 5793244))',0)
        ) AND b2val != 254;
```

Output

```
        shadow
-----------------------------------------------------------------------------------------
 MULTIPOLYGON(((3427927.9 5793243.85,3427927.8 5793243.85,3427927.8 5793243.95,
 3427927.9 5793243.95,3427928 5793243.95,3427928.1 5793243.95,3427928.1 5793243.85,3427928 ↩
    5793243.85,3427927.9 5793243.85)),
 ((3427927.9 5793243.65,3427927.8 5793243.65,3427927.8 5793243.75,3427927.8 ↩
    5793243.85,3427927.9 5793243.85,
 3427928 5793243.85,3427928 5793243.75,3427928.1 5793243.75,3427928.1 5793243.65,3427928 ↩
    5793243.65,3427927.9 5793243.65)))
```

Siehe auch

ST_DumpValues, ST_SetValue, ST_DumpAsPolygons, ST_NumBands, ST_PixelAsPolygon, ST_ScaleX, ST_ScaleY, ST_UpperLeftX ST_UpperLeftY, ST_SRID, ST_AsText, ST_Point, ST_MakeEnvelope, ST_Intersects, ST_Intersection

11.6.8 ST_NearestValue

ST_NearestValue — Gibt den nächstgelegenen nicht NODATA Wert eines bestimmten Pixels aus, das über "columnx" und "rowy" oder durch eine Punktgeometrie - im gleichen Koordinatenreferenzsystem wie der Raster - ausgewählt wird.

Synopsis

double precision **ST_NearestValue**(raster rast, integer bandnum, geometry pt, boolean exclude_nodata_value=true);
double precision **ST_NearestValue**(raster rast, geometry pt, boolean exclude_nodata_value=true);
double precision **ST_NearestValue**(raster rast, integer bandnum, integer columnx, integer rowy, boolean exclude_nodata_value=true);
double precision **ST_NearestValue**(raster rast, integer columnx, integer rowy, boolean exclude_nodata_value=true);

Description

Gibt den nächstgelegenen, nicht-NODATA Wert eines bestimmten Pixels aus, das über "columnx" und "rowy", oder durch einen geometrischen Punkt angegeben wird. Falls das angegebene Pixel den Wert NODATA hat, findet die Funktion das nächstgelegene Pixel, das nicht den Wert NODATA hat.

Die Bandnummern beginnen mit 1 und wenn keine Bandnummer angegeben ist, dann wird für `bandnum` 1 angenommen. Wenn `exclude_nodata_value` auf FALSE gesetzt ist, werden auch die Pixel mit einem `nodata` Wert einbezogen. Wenn `exclude_nodata_value` nicht übergeben wird, dann wird er über die Metadaten des Rasters ausgelesen.

Verfügbarkeit: 2.1.0



Anmerkung
ST_NearestValue ist eine Alternative zu ST_Value.

Beispiele

This example reads a value from pixel 2x2.

Code

```
WITH source AS (
    SELECT ST_SetValues(
        ST_AddBand(
            ST_MakeEmptyRaster(5, 5, -2, 2, 1, -1, 0, 0, 0),
            1, '8BUI', 1, 0
        ),
        1, 1, 1,
        ARRAY[
            [0, 1, 1, 1, 1],
            [1, 1, 1, 0, 1],
            [1, 0, 1, 1, 1],
            [1, 1, 1, 1, 0],
            [1, 1, 0, 1, 1]
        ]::double precision[][]
    ) AS rast
)
SELECT
    ST_Value(rast, 2, 2) AS value,
    ST_NearestValue(rast, 2, 2) AS nearestvalue
FROM source
```

Output

```
value | nearestvalue
-------+--------------
     1 |            1
```

This example reads a NODATA value from pixel 2x3.

Code

```
WITH source AS (
    SELECT ST_SetValues(
        ST_AddBand(
            ST_MakeEmptyRaster(5, 5, -2, 2, 1, -1, 0, 0, 0),
            1, '8BUI', 1, 0
        ),
        1, 1, 1,
        ARRAY[
            [0, 1, 1, 1, 1],
            [1, 1, 1, 0, 1],
            [1, 0, 1, 1, 1],
            [1, 1, 1, 1, 0],
            [1, 1, 0, 1, 1]
```

```
        ]::double precision[][]
    ) AS rast
)
SELECT
    ST_Value(rast, 2, 3) AS value,
    ST_NearestValue(rast, 2, 3) AS nearestvalue
FROM source
```

Output

```
value | nearestvalue
------+--------------
      |            1
```

Siehe auch

ST_Neighborhood, ST_Value

11.6.9 ST_SetZ

ST_SetZ — Gibt eine Geometrie mit denselben X/Y-Koordinaten wie die Eingabegeometrie zurück, wobei die Werte aus dem Raster mit dem gewünschten Resample-Algorithmus in die Z-Dimension kopiert werden.

Synopsis

geometry **ST_SetZ**(raster rast, geometry geom, text resample=nearest, integer band=1);

Description

Gibt eine Geometrie mit denselben X/Y-Koordinaten wie die Eingabegeometrie zurück, wobei die Werte aus dem Raster mit dem gewünschten Resample-Algorithmus in die Z-Dimensionen kopiert werden.

Der Parameter `resample` kann auf "nearest" gesetzt werden, um die Werte aus der Zelle zu kopieren, in die jeder Scheitelpunkt fällt, oder auf "bilinear", um bilineare Interpolation zu verwenden, um einen Wert zu berechnen, der auch die benachbarten Zellen berücksichtigt.

Verfügbarkeit: 3.2.0

Beispiele

2x2 test raster with values. 10 50. 40 20.

Code

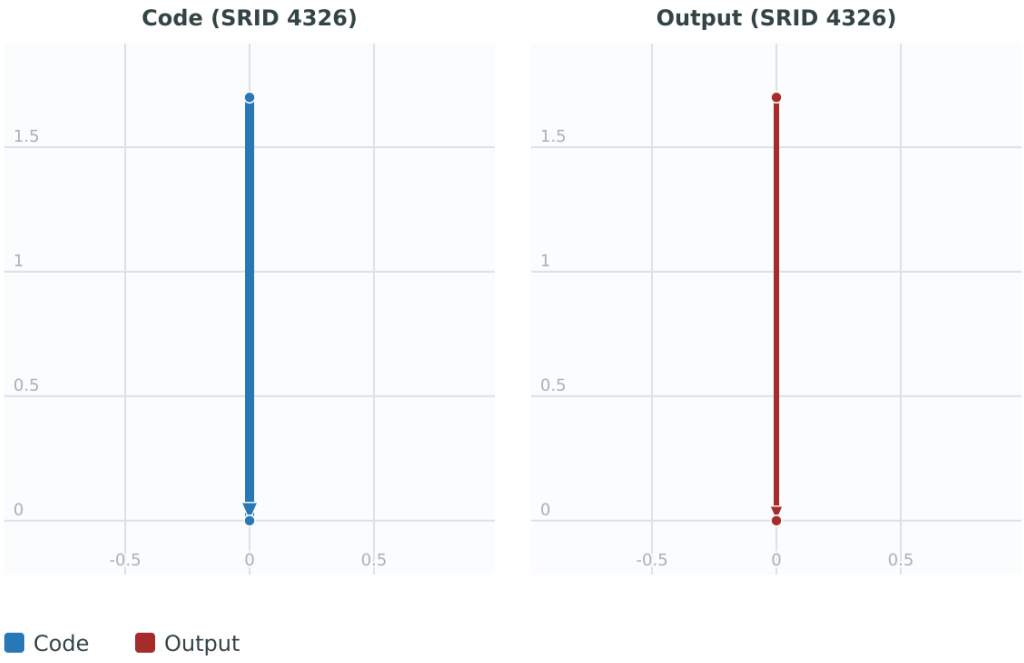
```
WITH test_raster AS (
SELECT
ST_SetValues(
ST_AddBand(
ST_MakeEmptyRaster(
      width => 2,
      height => 2,
      upperleftx => 0,
      upperlefty => 2,
      scalex => 1.0,
      scaley => -1.0,
      skewx => 0,
      skewy => 0,
```

```
      srid => 4326),
    index => 1,
    pixeltype => '16BSI',
    initialvalue => 0,
    nodataval => -999
),
  1,
  1,
  1,
  newvalueset => ARRAY[ARRAY[10.0::float8, 50.0::float8], ARRAY[40.0::float8, 20.0::float8 ↩
    ]]) AS rast
)
SELECT
ST_SetZ(
    rast,
    band => 1,
    geom => 'SRID=4326;LINESTRING(1.0 1.9,1.0 0.2)'::geometry,
    resample => 'bilinear'
)
FROM test_raster
```

Output

```
LINESTRING Z (1 1.9 38,1 0.2 27)
```

Figure



Siehe auch

ST_Value, ST_SetM

11.6.10 ST_SetM

ST_SetM — Gibt eine Geometrie mit denselben X/Y-Koordinaten wie die Eingabegeometrie zurück, wobei die Werte aus dem Raster mit dem gewünschten Resample-Algorithmus in die Dimension M kopiert werden.

Synopsis

geometry **ST_SetM**(raster rast, geometry geom, text resample=nearest, integer band=1);

Description

Liefert eine Geometrie mit denselben X/Y-Koordinaten wie die Eingabegeometrie und mit Werten aus dem Raster, die unter Verwendung des gewünschten Resample-Algorithmus in die M-Dimensionen kopiert wurden.

Der Parameter `resample` kann auf "nearest" gesetzt werden, um die Werte aus der Zelle zu kopieren, in die jeder Scheitelpunkt fällt, oder auf "bilinear", um bilineare Interpolation zu verwenden, um einen Wert zu berechnen, der auch die benachbarten Zellen berücksichtigt.

Verfügbarkeit: 3.2.0

Beispiele

2x2 test raster with values. 10 50. 40 20.

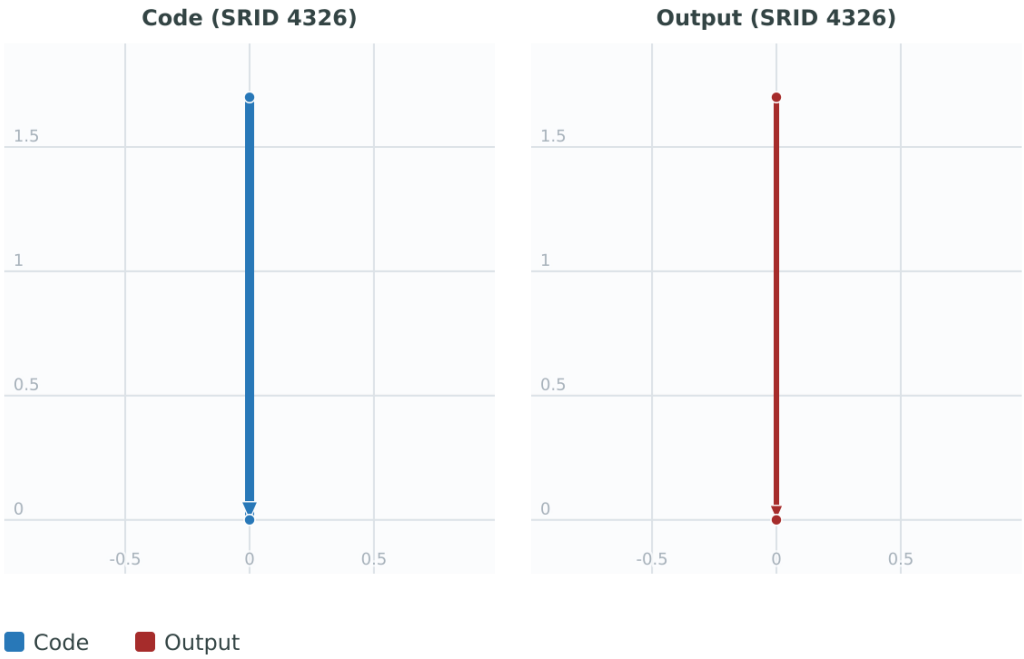
Code

```
WITH test_raster AS (
SELECT
ST_SetValues(
ST_AddBand(
ST_MakeEmptyRaster(
      width => 2,
      height => 2,
      upperleftx => 0,
      upperlefty => 2,
      scalex => 1.0,
      scaley => -1.0,
      skewx => 0,
      skewy => 0,
      srid => 4326),
    index => 1,
    pixeltype => '16BSI',
    initialvalue => 0,
    nodataval => -999
),
  1,
  1,
  1,
  newvalueset => ARRAY[ARRAY[10.0::float8, 50.0::float8], ARRAY[40.0::float8, 20.0::float8 ↩
    ]]) AS rast
)
SELECT
ST_SetM(
    rast,
    band => 1,
    geom => 'SRID=4326;LINESTRING(1.0 1.9,1.0 0.2)'::geometry,
    resample => 'bilinear'
)
FROM test_raster
```

Output

```
LINESTRING M (1 1.9 38,1 0.2 27)
```

Figure



Siehe auch

ST_Value, ST_SetZ

11.6.11 ST_Neighborhood

ST_Neighborhood — Gibt ein 2-D Feld in "Double Precision" aus, das sich aus nicht `NODATA` Werten um ein bestimmtes Pixel herum zusammensetzt. Das Pixel Kann über "columnx" und "rowy" oder über eine Punktgeometrie - im gleichen Koordinatenreferenzsystem wie der Raster - ausgewählt werden.

Synopsis

double precision[][] **ST_Neighborhood**(raster rast, integer bandnum, integer columnX, integer rowY, integer distanceX, integer distanceY, boolean exclude_nodata_value=true);
double precision[][] **ST_Neighborhood**(raster rast, integer columnX, integer rowY, integer distanceX, integer distanceY, boolean exclude_nodata_value=true);
double precision[][] **ST_Neighborhood**(raster rast, integer bandnum, geometry pt, integer distanceX, integer distanceY, boolean exclude_nodata_value=true);
double precision[][] **ST_Neighborhood**(raster rast, geometry pt, integer distanceX, integer distanceY, boolean exclude_nodata_value=tru

Description

Gibt für ein Pixel eines Bandes die ringsherum liegenden nicht-`NODATA` Werte in einem 2D-Feld mit Double Precision zurück. Das Pixel kann entweder über columnX und rowY, oder über einen geometrischen Punkt der im selben Koordinatenreferenzsystem wie der Raster vorliegt übergeben werden. Die Parameter `distanceX` und `distanceY` bestimmen die Anzahl der Pixel auf der X- und Y-Achse, um das festgelegte Pixel herum; z.B.: Sie möchten alle Werte innerhalb einer Entfernung von 3 Pixel entlang der X-Achse und einer Entfernung von 2 Pixel entlang der Y-Achse, um das Pixel von Interesse herum. Der Wert im Zentrum des 2-D Feldes ist der Wert jenes Pixels, das über columnX und rowY oder über einen geometrischen Punkt übergeben wurde.

Die Bandnummern beginnen mit 1 und wenn keine Bandnummer angegeben ist, dann wird für `bandnum` 1 angenommen. Wenn `exclude_nodata_value` auf FALSE gesetzt ist, werden auch die Pixel mit einem `nodata` Wert einbezogen. Wenn `exclude_nodata_value` nicht übergeben wird, dann wird er über die Metadaten des Rasters ausgelesen.



Anmerkung
Die Anzahl der Elemente entlang jeder Achse des zurückgegebenen 2D-Feldes ergibt sich aus 2 * (`distanceX|distanceY`) + 1. So ergibt sich bei einer `distanceX` und einer `distanceY` von je 1, ein Feld von 3x3.



Anmerkung
Das 2-D Feld kann einer beliebigen, integrierten Funktion zur Weiterverarbeitung übergeben werden, wie z.B. an ST_Min4ma, ST_Sum4ma, ST_Mean4ma.

Verfügbarkeit: 2.1.0

Beispiele

This example reads a value from pixel 2x2.

Code

```
SELECT
    ST_Neighborhood(rast, 2, 2, 1, 1)
FROM (
    SELECT
        ST_SetValues(ST_AddBand(ST_MakeEmptyRaster(5, 5, -2, 2, 1, -1, 0, 0, 0),
                '8BUI'::text, 1, 0
            ),
            1, 1, 1, ARRAY[
                [0, 1, 1, 1, 1],
                [1, 1, 1, 0, 1],
                [1, 0, 1, 1, 1],
                [1, 1, 1, 1, 0],
                [1, 1, 0, 1, 1]
            ]::double precision[],
            1
        ) AS rast
) AS foo
```

Output

```
{{NULL,1,1},{1,1,1},{1,NULL,1}}
```

This example reads a NODATA value from pixel 2x3.

Code

```
SELECT
    ST_Neighborhood(rast, 2, 3, 1, 1)
FROM (
    SELECT
        ST_SetValues(ST_AddBand(ST_MakeEmptyRaster(5, 5, -2, 2, 1, -1, 0, 0, 0),
                '8BUI'::text, 1, 0
            ),
            1, 1, 1, ARRAY[
                [0, 1, 1, 1, 1],
```

```
                [1, 1, 1, 0, 1],
                [1, 0, 1, 1, 1],
                [1, 1, 1, 1, 0],
                [1, 1, 0, 1, 1]
            ]::double precision[],
            1
        ) AS rast
) AS foo
```

Output

```
{{1,1,1},{1,NULL,1},{1,1,1}}
```

Pixel 3x3 has value. Exclude_nodata_value = FALSE.

Code

```
SELECT
    ST_Neighborhood(rast, 3, 3, 1, 1, false)
FROM ST_SetValues(ST_AddBand(ST_MakeEmptyRaster(5, 5, -2, 2, 1, -1, 0, 0, 0),
                '8BUI'::text, 1, 0
            ),
            1, 1, 1, ARRAY[
                [0, 1, 1, 1, 1],
                [1, 1, 1, 0, 1],
                [1, 0, 1, 1, 1],
                [1, 1, 1, 1, 0],
                [1, 1, 0, 1, 1]
            ]::double precision[],
            1
        ) AS rast
```

Output

```
{{1,1,0},{0,1,1},{1,1,1}}
```

Siehe auch

ST_NearestValue, ST_Min4ma, ST_Max4ma, ST_Sum4ma, ST_Mean4ma, ST_Range4ma, ST_Distinct4ma, ST_StdDev4ma

11.6.12 ST_SetValue

ST_SetValue — Setzt den Wert für ein Pixel eines Bandes, das über columnx und rowy festgelegt wird, oder für die Pixel die eine bestimmte Geometrie schneiden, und gibt den veränderten Raster zurück. Die Bandnummerierung beginnt mit 1; wenn die Bandnummer nicht angegeben ist, wird 1 angenommen.

Synopsis

raster **ST_SetValue**(raster rast, integer bandnum, geometry geom, double precision newvalue);
raster **ST_SetValue**(raster rast, geometry geom, double precision newvalue);
raster **ST_SetValue**(raster rast, integer bandnum, integer columnx, integer rowy, double precision newvalue);
raster **ST_SetValue**(raster rast, integer columnx, integer rowy, double precision newvalue);

Description

Setzt die Werte bestimmter Pixel eines Bandes auf einen neuen Wert und gibt den veränderten Raster zurück. Die Pixel können über die Rasterzeile und die Rasterspalte, oder über eine Geometrie festgelegt werden. Wenn die Bandnummer nicht angegeben ist, wird 1 angenommen.

Erweiterung: 2.1.0 Die geometrische Variante von ST_SetValue() unterstützt nun jeden geometrischen Datentyp, nicht nur POINT. Die geometrische Variante ist ein Adapter für die geomval[] Variante von ST_SetValues()

Beispiele

This example uses geometry input.

Code

```
                SELECT (foo.geomval).val, ST_Union((foo.geomval).geom)
FROM (SELECT ST_DumpAsPolygons(ST_SetValue(
            rast,
            1,
            ST_Point(3427927.75, 5793243.95),
            50)
            ) As geomval
FROM dummy_rast
where rid = 2) As foo
WHERE (foo.geomval).val < 250
GROUP BY (foo.geomval).val;
```

Output

```
 val |                                                              st_astext
-----+------------------------------------------------------------------------
  50 | POLYGON((3427927.75 5793244,3427927.75 5793243.95,3427927.8 579324 ...
 249 | POLYGON((3427927.95 5793243.95,3427927.95 5793243.85,3427928 57932 ...
```

This example stores the changed raster.

Code

```
UPDATE dummy_rast SET rast = ST_SetValue(rast, 1, ST_Point(3427927.75, 5793243.95), 100)
    WHERE rid = 2   ;
```

Siehe auch

ST_Value, ST_DumpAsPolygons

11.6.13 ST_SetValues

ST_SetValues — Gibt einen Raster zurück, der durch das Setzen der Werte eines bestimmten Bandes verändert wurde.

Synopsis

raster **ST_SetValues**(raster rast, integer nband, integer columnx, integer rowy, double precision[][] newvalueset, boolean[][] noset=NULL, boolean keepnodata=FALSE);
raster **ST_SetValues**(raster rast, integer nband, integer columnx, integer rowy, double precision[][] newvalueset, double precision nosetvalue, boolean keepnodata=FALSE);

raster **ST_SetValues**(raster rast, integer nband, integer columnx, integer rowy, integer width, integer height, double precision newvalue, boolean keepnodata=FALSE);
raster **ST_SetValues**(raster rast, integer columnx, integer rowy, integer width, integer height, double precision newvalue, boolean keepnodata=FALSE);
raster **ST_SetValues**(raster rast, integer nband, geomval[] geomvalset, boolean keepnodata=FALSE);

Description

Gibt einen veränderten Raster zurück, der durch die Zuweisung neuer Zellwerte auf festgelegte Pixel des ausgewiesenen Rasterbandes entsteht. `columnx` und `rowy` sind bei 1 beginnend indiziert.

Wenn `keepnodata` TRUE ist, dann werden die Pixel mit dem Wert NODATA nicht mit dem entsprechenden Wert in `newvalueset` belegt.

Bei der Variante 1 werden die betreffenden Pixel über die Pixelkoordinaten `columnx` und `rowy`, und durch die Größe des Feldes `newvalueset` festgelegt. Über den Parameter `noset` kann verhindert werden, dass bestimmte, in `newvalueset` auftretende Pixelwerte gesetzt werden (da PostgreSQL keine unregelmäßigen Felder/"ragged arrays" zulässt). Siehe das Beispiel mit der Variante 1.

Variante 2 gleicht Variante 1, aber mit einem einfachen `nosetvalue` in Double Precision anstelle des booleschen Feldes `noset`. Elemente in `newvalueset` mit dem Wert `nosetvalue` werden übersprungen. Siehe das Beispiel mit der Variante 2.

Bei der Variante 3 werden die betreffenden Pixel über die Pixelkoordinaten `columnx` und `rowy`, und durch `width` und `height` festgelegt. Siehe das Beispiel mit der Variante 3.

Variante 4 entspricht Variante3 mit der Ausnahme, dass die Pixel des ersten Bandes von `rast` gesetzt werden.

Bei der Variante 5 wird ein Feld von geomval verwendet um die Pixel zu bestimmen. Wenn die gesamte Geometrie des Feldes vom Datentyp POINT oder MULTIPOINT ist, verwendet die Funktion Länge und Breite eines jeden Punktes um den Wert eines Pixels direkt zu setzen. Andernfalls wird die Geometrie in Raster umgewandelt über die dann in einem Schritt iteriert wird. Siehe das Beispiel mit der Variante 5.

Verfügbarkeit: 2.1.0

Beispiele

Variant 1.

These examples pass `FALSE` for *`exclude_nodata_value`* so that every raster cell remains visible. NODATA cells display the band's NODATA value, which is `0` here.

Set the lower-right 2 by 2 block of a 3 by 3 raster to `9`.

Code

```
SELECT
    (poly).x,
    (poly).y,
    (poly).val
FROM (
SELECT
    ST_PixelAsPolygons(ST_SetValues(ST_AddBand(ST_MakeEmptyRaster(3, 3, 0, 0, 1, -1, 0, 0,  ↩
    0),
                1, '8BUI', 1, 0
            ),
            1, 2, 2, ARRAY[[9, 9], [9, 9]]::double precision[][]
        ),
        1, FALSE
    ) AS poly
) foo
ORDER BY 1, 2;
```

Output

```
 x | y | val
---+---+-----
 1 | 1 |   1
 1 | 2 |   1
 1 | 3 |   1
 2 | 1 |   1
 2 | 2 |   9
 2 | 3 |   9
 3 | 1 |   1
 3 | 2 |   9
 3 | 3 |   9
```

Replace the complete raster from an array; the central `NULL` becomes NODATA.

Code

```
SELECT
    (poly).x,
    (poly).y,
    (poly).val
FROM (
SELECT
    ST_PixelAsPolygons(ST_SetValues(ST_AddBand(ST_MakeEmptyRaster(3, 3, 0, 0, 1, -1, 0, 0,  ↩
    0),
                1, '8BUI', 1, 0
            ),
            1, 1, 1, ARRAY[[9, 9, 9], [9, NULL, 9], [9, 9, 9]]::double precision[][]
        ),
        1, FALSE
    ) AS poly
) foo
ORDER BY 1, 2;
```

Output

```
 x | y | val
---+---+-----
 1 | 1 |   9
 1 | 2 |   9
 1 | 3 |   9
 2 | 1 |   9
 2 | 2 |   0
 2 | 3 |   9
 3 | 1 |   9
 3 | 2 |   9
 3 | 3 |   9
```

Use the *noset* mask to retain the left-middle source pixel while replacing the other cells.

Code

```
SELECT
    (poly).x,
    (poly).y,
    (poly).val
FROM (
SELECT
    ST_PixelAsPolygons(ST_SetValues(ST_AddBand(ST_MakeEmptyRaster(3, 3, 0, 0, 1, -1, 0, 0,  ↩
    0),
                1, '8BUI', 1, 0
            ),
            1, 1, 1,
                ARRAY[[9, 9, 9], [9, NULL, 9], [9, 9, 9]]::double precision[][],
                ARRAY[[false], [true]]::boolean[][]
        ),
        1, FALSE
```

```
    ) AS poly
) foo
ORDER BY 1, 2;
```

Output

```
x | y | val
---+---+-----
 1 | 1 |   9
 1 | 2 |   1
 1 | 3 |   9
 2 | 1 |   9
 2 | 2 |   0
 2 | 3 |   9
 3 | 1 |   9
 3 | 2 |   9
 3 | 3 |   9
```

With *keepnodata* enabled, an existing NODATA pixel is not overwritten.

Code

```
SELECT
    (poly).x,
    (poly).y,
    (poly).val
FROM (
SELECT
    ST_PixelAsPolygons(ST_SetValues(ST_SetValue(ST_AddBand(ST_MakeEmptyRaster(3, 3, 0, 0, 1, ↩
     -1, 0, 0, 0),
                    1, '8BUI', 1, 0
                ),
                1, 1, 1, NULL
            ),
            1, 1, 1,
                ARRAY[[9, 9, 9], [9, NULL, 9], [9, 9, 9]]::double precision[][],
                ARRAY[[false], [true]]::boolean[][],
                TRUE
        ),
        1, FALSE
    ) AS poly
) foo
ORDER BY 1, 2;
```

Output

```
x | y | val
---+---+-----
 1 | 1 |   0
 1 | 2 |   1
 1 | 3 |   9
 2 | 1 |   9
 2 | 2 |   0
 2 | 3 |   9
 3 | 1 |   9
 3 | 2 |   9
 3 | 3 |   9
```

Variant 2.

Use `-1` as the value that should not be written; only the lower-right 2 by 2 block changes.

Code

```
SELECT
    (poly).x,
    (poly).y,
    (poly).val
```

```
FROM (
SELECT
    ST_PixelAsPolygons(ST_SetValues(ST_AddBand(ST_MakeEmptyRaster(3, 3, 0, 0, 1, -1, 0, 0,  ↩
    0),
                1, '8BUI', 1, 0
            ),
            1, 1, 1, ARRAY[[-1, -1, -1], [-1, 9, 9], [-1, 9, 9]]::double precision[][], -1
        ),
        1, FALSE
    ) AS poly
) foo
ORDER BY 1, 2;
```

Output

```
x | y | val
---+---+-----
 1 | 1 |   1
 1 | 2 |   1
 1 | 3 |   1
 2 | 1 |   1
 2 | 2 |   9
 2 | 3 |   9
 3 | 1 |   1
 3 | 2 |   9
 3 | 3 |   9
```

The same update can use `NULL` as the value that should not be written.

Code

```
SELECT
    (poly).x,
    (poly).y,
    (poly).val
FROM (
SELECT
    ST_PixelAsPolygons(ST_SetValues(ST_AddBand(ST_MakeEmptyRaster(3, 3, 0, 0, 1, -1, 0, 0,  ↩
    0),
                1, '8BUI', 1, 0
            ),
            1, 1, 1, ARRAY[[NULL, NULL, NULL], [NULL, 9, 9], [NULL, 9, 9]]::double precision ↩
    [][], NULL::double precision
        ),
        1, FALSE
    ) AS poly
) foo
ORDER BY 1, 2;
```

Output

```
x | y | val
---+---+-----
 1 | 1 |   1
 1 | 2 |   1
 1 | 3 |   1
 2 | 1 |   1
 2 | 2 |   9
 2 | 3 |   9
 3 | 1 |   1
 3 | 2 |   9
 3 | 3 |   9
```

Variant 3.

Set a 2 by 2 rectangle starting at column 2, row 2.

Code

```
SELECT
    (poly).x,
    (poly).y,
    (poly).val
FROM (
SELECT
    ST_PixelAsPolygons(ST_SetValues(ST_AddBand(ST_MakeEmptyRaster(3, 3, 0, 0, 1, -1, 0, 0,  ↩
    0),
                1, '8BUI', 1, 0
            ),
            1, 2, 2, 2, 2, 9
        ),
        1, FALSE
    ) AS poly
) foo
ORDER BY 1, 2;
```

Output

```
x | y | val
---+---+-----
 1 | 1 |   1
 1 | 2 |   1
 1 | 3 |   1
 2 | 1 |   1
 2 | 2 |   9
 2 | 3 |   9
 3 | 1 |   1
 3 | 2 |   9
 3 | 3 |   9
```

When *keepnodata* is true, the NODATA cell inside the rectangle remains unchanged.

Code

```
SELECT
    (poly).x,
    (poly).y,
    (poly).val
FROM (
SELECT
    ST_PixelAsPolygons(ST_SetValues(ST_SetValue(ST_AddBand(ST_MakeEmptyRaster(3, 3, 0, 0, 1, ↩
     -1, 0, 0, 0),
                    1, '8BUI', 1, 0
                ),
                1, 2, 2, NULL
            ),
            1, 2, 2, 2, 2, 9, TRUE
        ),
        1, FALSE
    ) AS poly
) foo
ORDER BY 1, 2;
```

Output

```
x | y | val
---+---+-----
 1 | 1 |   1
 1 | 2 |   1
 1 | 3 |   1
 2 | 1 |   1
 2 | 2 |   0
 2 | 3 |   9
 3 | 1 |   1
 3 | 2 |   9
 3 | 3 |   9
```

Variant 5.

Code

```
WITH foo AS (
    SELECT 1 AS rid, ST_AddBand(ST_MakeEmptyRaster(5, 5, 0, 0, 1, -1, 0, 0, 0), 1, '8BUI', ↩
    0, 0) AS rast
), bar AS (
    SELECT 1 AS gid, 'SRID=0;POINT(2.5 -2.5)'::geometry geom UNION ALL
    SELECT 2 AS gid, 'SRID=0;POLYGON((1 -1,4 -1,4 -4,1 -4,1 -1))'::geometry geom UNION ALL
    SELECT 3 AS gid, 'SRID=0;POLYGON((0 0,5 0,5 -1,1 -1,1 -4,0 -4,0 0))'::geometry geom ↩
    UNION ALL
    SELECT 4 AS gid, 'SRID=0;MULTIPOINT(0 0,4 4,4 -4)'::geometry
), changed AS (
    SELECT rid, gid, ST_SetValue(rast, 1, geom, gid) AS rast
    FROM foo
    CROSS JOIN bar
)
SELECT
    rid,
    gid,
    y,
    array_agg(ST_Value(rast, 1, x, y) ORDER BY x) AS values
FROM changed
CROSS JOIN generate_series(1, 5) AS x
CROSS JOIN generate_series(1, 5) AS y
GROUP BY rid, gid, y
ORDER BY rid, gid, y;
```

Output

```
rid | gid | y |           values
-----+-----+---+----------------------------
   1 |   1 | 1 | {NULL,NULL,NULL,NULL,NULL}
   1 |   1 | 2 | {NULL,NULL,NULL,NULL,NULL}
   1 |   1 | 3 | {NULL,NULL,1,NULL,NULL}
   1 |   1 | 4 | {NULL,NULL,NULL,NULL,NULL}
   1 |   1 | 5 | {NULL,NULL,NULL,NULL,NULL}
   1 |   2 | 1 | {NULL,NULL,NULL,NULL,NULL}
   1 |   2 | 2 | {NULL,2,2,2,NULL}
   1 |   2 | 3 | {NULL,2,2,2,NULL}
   1 |   2 | 4 | {NULL,2,2,2,NULL}
   1 |   2 | 5 | {NULL,NULL,NULL,NULL,NULL}
   1 |   3 | 1 | {3,3,3,3,3}
   1 |   3 | 2 | {3,NULL,NULL,NULL,NULL}
   1 |   3 | 3 | {3,NULL,NULL,NULL,NULL}
   1 |   3 | 4 | {3,NULL,NULL,NULL,NULL}
   1 |   3 | 5 | {NULL,NULL,NULL,NULL,NULL}
   1 |   4 | 1 | {4,NULL,NULL,NULL,NULL}
   1 |   4 | 2 | {NULL,NULL,NULL,NULL,NULL}
   1 |   4 | 3 | {NULL,NULL,NULL,NULL,NULL}
   1 |   4 | 4 | {NULL,NULL,NULL,NULL,NULL}
   1 |   4 | 5 | {NULL,NULL,NULL,NULL,4}
(20 rows)
```

Das folgende Beispiel zeigt, dass bestehende "geomvals" durch ein Feld mit "geomvals" später überschrieben werden können

Code

```
WITH foo AS (
    SELECT 1 AS rid, ST_AddBand(ST_MakeEmptyRaster(5, 5, 0, 0, 1, -1, 0, 0, 0), 1, '8BUI', ↩
    0, 0) AS rast
), bar AS (
    SELECT 1 AS gid, 'SRID=0;POINT(2.5 -2.5)'::geometry geom UNION ALL
    SELECT 2 AS gid, 'SRID=0;POLYGON((1 -1,4 -1,4 -4,1 -4,1 -1))'::geometry geom UNION ALL
    SELECT 3 AS gid, 'SRID=0;POLYGON((0 0,5 0,5 -1,1 -1,1 -4,0 -4,0 0))'::geometry geom ↩
    UNION ALL
    SELECT 4 AS gid, 'SRID=0;MULTIPOINT(0 0,4 4,4 -4)'::geometry
), changed AS (
```

```
    SELECT ST_SetValues(
        rast,
        1,
        ARRAY[
            ROW((SELECT geom FROM bar WHERE gid = 1), 1),
            ROW((SELECT geom FROM bar WHERE gid = 2), 2)
        ]::geomval[]
    ) AS rast
    FROM foo
)
SELECT
    y,
    array_agg(ST_Value(rast, 1, x, y) ORDER BY x) AS values
FROM changed
CROSS JOIN generate_series(1, 5) AS x
CROSS JOIN generate_series(1, 5) AS y
GROUP BY y
ORDER BY y;
```

Output

```
y |          values
---+---------------------------
 1 | {NULL,NULL,NULL,NULL,NULL}
 2 | {NULL,2,2,2,NULL}
 3 | {NULL,2,2,2,NULL}
 4 | {NULL,2,2,2,NULL}
 5 | {NULL,NULL,NULL,NULL,NULL}
(5 rows)
```

Dieses Beispiel ist das Gegenteil des vorigen Beispiels

Code

```
WITH foo AS (
    SELECT 1 AS rid, ST_AddBand(ST_MakeEmptyRaster(5, 5, 0, 0, 1, -1, 0, 0, 0), 1, '8BUI', ↩
    0, 0) AS rast
), bar AS (
    SELECT 1 AS gid, 'SRID=0;POINT(2.5 -2.5)'::geometry geom UNION ALL
    SELECT 2 AS gid, 'SRID=0;POLYGON((1 -1,4 -1,4 -4,1 -4,1 -1))'::geometry geom UNION ALL
    SELECT 3 AS gid, 'SRID=0;POLYGON((0 0,5 0,5 -1,1 -1,1 -4,0 -4,0 0))'::geometry geom ↩
    UNION ALL
    SELECT 4 AS gid, 'SRID=0;MULTIPOINT(0 0,4 4,4 -4)'::geometry
), changed AS (
    SELECT ST_SetValues(
        rast,
        1,
        ARRAY[
            ROW((SELECT geom FROM bar WHERE gid = 2), 2),
            ROW((SELECT geom FROM bar WHERE gid = 1), 1)
        ]::geomval[]
    ) AS rast
    FROM foo
)
SELECT
    y,
    array_agg(ST_Value(rast, 1, x, y) ORDER BY x) AS values
FROM changed
CROSS JOIN generate_series(1, 5) AS x
CROSS JOIN generate_series(1, 5) AS y
GROUP BY y
ORDER BY y;
```

Output

```
y |           values
---+----------------------------
 1 | {NULL,NULL,NULL,NULL,NULL}
 2 | {NULL,2,2,2,NULL}
 3 | {NULL,2,1,2,NULL}
 4 | {NULL,2,2,2,NULL}
 5 | {NULL,NULL,NULL,NULL,NULL}
(5 rows)
```

Siehe auch

ST_Value, ST_SetValue, ST_PixelAsPolygons

11.6.14 ST_DumpValues

ST_DumpValues — Gibt die Werte eines bestimmten Bandes als 2-dimensionales Feld aus.

Synopsis

table(nband integer, valarray double precision[][]) **ST_DumpValues**(raster rast , integer[] nband=NULL , boolean exclude_nodata_value
);
double precision[][] **ST_DumpValues**(raster rast , integer nband , boolean exclude_nodata_value=true);

Description

Gibt die Werte eines bestimmten Bandes als 2-dimensionales Feld (der erste Index entspricht der Zeile, der zweite der Spalte) aus. Wenn `nband` NULL oder nicht gegeben ist, werden alle Rasterbänder abgearbeitet.

Verfügbarkeit: 2.1.0

Beispiele

Code

```
WITH foo AS (
    SELECT ST_AddBand(ST_AddBand(ST_AddBand(ST_MakeEmptyRaster(3, 3, 0, 0, 1, -1, 0, 0, 0),  ↩
    1, '8BUI'::text, 1, 0), 2, '32BF'::text, 3, -9999), 3, '16BSI', 0, 0) AS rast
)
SELECT
    (ST_DumpValues(rast)).*
FROM foo;
```

Output

```
nband |                      valarray
------+-----------------------------------------------------
    1 | {{1,1,1},{1,1,1},{1,1,1}}
    2 | {{3,3,3},{3,3,3},{3,3,3}}
    3 | {{NULL,NULL,NULL},{NULL,NULL,NULL},{NULL,NULL,NULL}}
(3 rows)
```

Code

```
WITH foo AS (
    SELECT ST_AddBand(ST_AddBand(ST_AddBand(ST_MakeEmptyRaster(3, 3, 0, 0, 1, -1, 0, 0, 0),  ↩
    1, '8BUI'::text, 1, 0), 2, '32BF'::text, 3, -9999), 3, '16BSI', 0, 0) AS rast
)
SELECT
    (ST_DumpValues(rast, ARRAY[3, 1])).*
FROM foo;
```

Output

```
nband |                       valarray
-------+-----------------------------------------------------
     3 | {{NULL,NULL,NULL},{NULL,NULL,NULL},{NULL,NULL,NULL}}
     1 | {{1,1,1},{1,1,1},{1,1,1}}
(2 rows)
```

Code

```
WITH foo AS (
    SELECT ST_SetValue(ST_AddBand(ST_MakeEmptyRaster(3, 3, 0, 0, 1, -1, 0, 0, 0), 1, '8BUI', ↩
     1, 0), 1, 2, 5) AS rast
)
SELECT
    (ST_DumpValues(rast, 1))[2][1]
FROM foo;
```

Output

```
5
```

Siehe auch

ST_Value, ST_SetValue, ST_SetValues

11.6.15 ST_PixelOfValue

ST_PixelOfValue — Gibt die columnx- und rowy-Koordinaten jener Pixel aus, deren Zellwert gleich dem gesuchten Wert ist.

Synopsis

table(val double precision, x integer, y integer) **ST_PixelOfValue**(raster rast , integer nband , double precision[] search , boolean exclude_nodata_value=true);
table(val double precision, x integer, y integer) **ST_PixelOfValue**(raster rast , double precision[] search , boolean exclude_nodata_value=true);
table(x integer, y integer) **ST_PixelOfValue**(raster rast , integer nband , double precision search , boolean exclude_nodata_value=true);
table(x integer, y integer) **ST_PixelOfValue**(raster rast , double precision search , boolean exclude_nodata_value=true);

Description

Gibt die columnx- und rowy-Koordinaten jener Pixel aus, deren Zellwert gleich dem gesuchten Wert ist. Wenn kein Band angegeben ist, wird Band 1 angenommen.

Verfügbarkeit: 2.1.0

Beispiele

Code

```
WITH source AS (
    SELECT ST_SetValues(
        ST_AddBand(
            ST_MakeEmptyRaster(5, 5, -2, 2, 1, -1, 0, 0, 0),
            1, '8BUI', 1, 0
        ),
```

```
        1, 1, 1,
        ARRAY[
            [0, 1, 1, 1, 1],
            [1, 1, 1, 0, 1],
            [1, 0, 1, 1, 1],
            [1, 1, 1, 1, 255],
            [1, 1, 0, 1, 1]
        ]::double precision[][]
    ) AS rast
)
SELECT (ST_PixelOfValue(rast, 1, ARRAY[1, 255])).*
FROM source
```

Output

```
val | x | y
-----+---+---
   1 | 1 | 2
   1 | 1 | 3
   1 | 1 | 4
   1 | 1 | 5
   1 | 2 | 1
   1 | 2 | 2
   1 | 2 | 4
   1 | 2 | 5
   1 | 3 | 1
   1 | 3 | 2
   1 | 3 | 3
   1 | 3 | 4
   1 | 4 | 1
   1 | 4 | 3
   1 | 4 | 4
   1 | 4 | 5
   1 | 5 | 1
   1 | 5 | 2
   1 | 5 | 3
 255 | 5 | 4
   1 | 5 | 5
```

11.7 Raster Editoren

11.7.1 ST_SetGeoReference

ST_SetGeoReference — Georeferenziert einen Raster über 6 Parameter in einem einzigen Aufruf. Die Zahlen müssen durch Leerzeichen getrennt sein. Die Funktion akzeptiert die Eingabe im Format von 'GDAL' und von 'ESRI'. Der Standardwert ist GDAL.

Synopsis

raster **ST_SetGeoReference**(raster rast, text georefcoords, text format=GDAL);
raster **ST_SetGeoReference**(raster rast, double precision upperleftx, double precision upperlefty, double precision scalex, double precision scaley, double precision skewx, double precision skewy);

Description

Georeferenziert einen Raster über 6 Parameter in einem einzigen Aufruf. Die Funktion akzeptiert die Eingabe im Format von 'GDAL' und von 'ESRI'. Der Standardwert ist GDAL. Wenn die 6 Parameter nicht angegeben sind, wird NULL zurückgegeben.

Die Formate unterscheiden sich wie folgt:

`GDAL` uses the value order shown below.

Code

```
scalex skewy skewx scaley upperleftx upperlefty
```

`ESRI` uses the value order shown below.

Code

```
scalex skewy skewx scaley upperleftx + scalex*0.5 upperlefty + scaley*0.5
```



Anmerkung
Wenn der Raster out-db Bänder aufweist, dann kann eine Änderung der Georeferenzierung zu einem unkorrekten Zugriff auf die extern gespeicherten Daten des Bandes führen.

Erweiterung: 2.1.0 ST_SetGeoReference(raster, double precision, ...) Variante hinzugefügt

Beispiele

Code

```
WITH source AS (
    SELECT ST_MakeEmptyRaster(5, 5, 0, 0, 1, -1, 0, 0, 0) AS rast
), variants AS (
    SELECT 0 AS rid, rast FROM source
    UNION ALL
    SELECT 1, ST_SetGeoReference(rast, '10 0 0 -10 0.1 0.1', 'GDAL')
    FROM source
    UNION ALL
    SELECT 2, ST_SetGeoReference(rast, '10 0 0 -10 5.1 -4.9', 'ESRI')
    FROM source
    UNION ALL
    SELECT 3, ST_SetGeoReference(rast, 1, 1, 10, -10, 0.001, 0.001)
    FROM source
)
SELECT
    rid,
    round(ST_UpperLeftX(rast)::numeric, 3) AS upperleftx,
    round(ST_UpperLeftY(rast)::numeric, 3) AS upperlefty,
    ST_ScaleX(rast) AS scalex,
    ST_ScaleY(rast) AS scaley,
    ST_SkewX(rast) AS skewx,
    ST_SkewY(rast) AS skewy
FROM variants
ORDER BY rid;
```

Output

```
rid | upperleftx | upperlefty | scalex | scaley | skewx | skewy
-----+------------+------------+--------+--------+-------+-------
   0 |      0.000 |      0.000 |      1 |     -1 |     0 |     0
   1 |      0.100 |      0.100 |     10 |    -10 |     0 |     0
   2 |      0.100 |      0.100 |     10 |    -10 |     0 |     0
   3 |      1.000 |      1.000 |     10 |    -10 | 0.001 | 0.001
(4 rows)
```

Siehe auch

ST_GeoReference, ST_ScaleX, ST_ScaleY, ST_UpperLeftX, ST_UpperLeftY

11.7.2 ST_SetRotation

ST_SetRotation — Bestimmt die Rotation des Rasters in Radiant.

Synopsis

raster **ST_SetRotation**(raster rast, float8 rotation);

Description

Einheitliche Rotation des Rasters. Die Rotation wird in Radiant angegeben. Siehe World File für mehr Details.

Beispiele

Code

```
WITH variants AS (
    SELECT rid, 'rotated' AS kind, ST_SetRotation(rast, 15) AS rast
    FROM dummy_rast
    UNION ALL
    SELECT rid, 'original', rast
    FROM dummy_rast
)
SELECT
    rid,
    kind,
    round(ST_ScaleX(rast)::numeric, 4) AS scale_x,
    round(ST_ScaleY(rast)::numeric, 4) AS scale_y,
    round(ST_SkewX(rast)::numeric, 4) AS skew_x,
    round(ST_SkewY(rast)::numeric, 4) AS skew_y
FROM variants
ORDER BY rid, kind;
```

Output

```
rid |   kind   | scale_x | scale_y | skew_x | skew_y
-----+----------+---------+---------+--------+--------
   1 | original |  2.0000 |  3.0000 | 0.0000 | 0.0000
   1 | rotated  | -1.5194 | -2.2791 | 1.9509 | 1.3006
   2 | original |  0.0500 | -0.0500 | 0.0000 | 0.0000
   2 | rotated  | -0.0380 | -0.0380 | 0.0325 | 0.0325
(4 rows)
```

Siehe auch

ST_Rotation, ST_ScaleX, ST_ScaleY, ST_SkewX, ST_SkewY

11.7.3 ST_SetScale

ST_SetScale — Setzt die X- und Y-Größe der Pixel in den Einheiten des Koordinatenreferenzsystems. Entweder eine Zahl pro Pixel oder Breite und Höhe.

Synopsis

raster **ST_SetScale**(raster rast, float8 xy);
raster **ST_SetScale**(raster rast, float8 x, float8 y);

Description

Setzt die X- und Y-Größe der Pixel in den Einheiten des Koordinatenreferenzsystems. Entweder eine Zahl pro Pixel oder Breite und Höhe. X und Y werden als gleich groß angenommen, wenn nur eine Zahl übergeben wird.



Anmerkung
ST_SetScale unterscheidet sich von ST_Rescale dadurch, dass bei ST_SetScale der Raster nicht skaliert wird um mit der Rasterausdehnung übereinzustimmen. Es werden nur die Metadaten (oder die Georeferenz) des Rasters geändert, um eine ursprünglich falsch angegebene Skalierung zu korrigieren. ST_Rescale berechnet die Breite und Höhe eines Rasters so, dass er mit der geographischen Ausdehnung des Rasters übereinstimmt. ST_SetScale verändert weder die Breite noch die Höhe des Rasters.

Änderung: 2.0.0. Versionen von WKTRaster haben dies als "ST_SetPixelSizeY" bezeichnet. Dies wurde mit 2.0.0 geändert.

Beispiele

Code

```
UPDATE dummy_rast
    SET rast = ST_SetScale(rast, 1.5)
WHERE rid = 2;

SELECT ST_ScaleX(rast) As pixx, ST_ScaleY(rast) As pixy, Box3D(rast) As newbox
FROM dummy_rast
WHERE rid = 2;
```

Output

```
pixx | pixy |                    newbox
------+------+-------------------------------------------------
  1.5 |  1.5 | BOX(3427927.75 5793244 0, 3427935.25 5793251.5 0)
```

Code

```
UPDATE dummy_rast
    SET rast = ST_SetScale(rast, 1.5, 0.55)
WHERE rid = 2;

SELECT ST_ScaleX(rast) As pixx, ST_ScaleY(rast) As pixy, Box3D(rast) As newbox
FROM dummy_rast
WHERE rid = 2;
```

Output

```
pixx | pixy |                    newbox
------+------+-----------------------------------------------
  1.5 | 0.55 | BOX(3427927.75 5793244 0,3427935.25 5793247 0)
```

Siehe auch

ST_ScaleX, ST_ScaleY, Box3D

11.7.4 ST_SetSkew

ST_SetSkew — Setzt den georeferenzierten X- und Y-Versatz (oder den Rotationsparameter). Wenn nur ein Wert übergeben wird, werden X und Y auf den selben Wert gesetzt.

Synopsis

raster **ST_SetSkew**(raster rast, float8 skewxy);
raster **ST_SetSkew**(raster rast, float8 skewx, float8 skewy);

Description

Setzt den georeferenzierten X- und Y-Versatz (oder den Rotationsparameter). Wenn nur ein Wert übergeben wird, werden X und Y auf den selben Wert gesetzt. Siehe World File für weitere Details.

Beispiele

This is the first example.

Code

```
UPDATE dummy_rast SET rast = ST_SetSkew(rast, 1, 2) WHERE rid = 1;
SELECT
    rid,
    format('(%s, %s)', skewx, skewy) AS skew,
    term,
    georef[idx] AS value
FROM (
    SELECT
        rid,
        ST_SkewX(rast) AS skewx,
        ST_SkewY(rast) AS skewy,
        string_to_array(ST_GeoReference(rast), E'\n') AS georef
    FROM dummy_rast
    WHERE rid = 1
) AS refs
CROSS JOIN (VALUES
    (1, 'scaleX'),
    (2, 'skewY'),
    (3, 'skewX'),
    (4, 'scaleY'),
    (5, 'originX'),
    (6, 'originY')
) AS parts(idx, term)
ORDER BY idx;
```

Output

```
rid |  skew  |  term   |    value
-----+--------+---------+--------------
   1 | (1, 2) | scaleX  | 2.0000000000
   1 | (1, 2) | skewY   | 2.0000000000
   1 | (1, 2) | skewX   | 1.0000000000
   1 | (1, 2) | scaleY  | 3.0000000000
   1 | (1, 2) | originX | 0.5000000000
   1 | (1, 2) | originY | 0.5000000000
(6 rows)
```

This second example sets both values to the same number.

Code

```
UPDATE dummy_rast SET rast = ST_SetSkew(rast, 0) WHERE rid = 1;
SELECT
    rid,
    format('(%s, %s)', skewx, skewy) AS skew,
    term,
```

```
    georef[idx] AS value
FROM (
    SELECT
        rid,
        ST_SkewX(rast) AS skewx,
        ST_SkewY(rast) AS skewy,
        string_to_array(ST_GeoReference(rast), E'\n') AS georef
    FROM dummy_rast
    WHERE rid = 1
) AS refs
CROSS JOIN (VALUES
    (1, 'scaleX'),
    (2, 'skewY'),
    (3, 'skewX'),
    (4, 'scaleY'),
    (5, 'originX'),
    (6, 'originY')
) AS parts(idx, term)
ORDER BY idx;
```

Output

```
rid |  skew  |  term   |    value
-----+--------+---------+-------------
   1 | (0, 0) | scaleX  | 2.0000000000
   1 | (0, 0) | skewY   | 0.0000000000
   1 | (0, 0) | skewX   | 0.0000000000
   1 | (0, 0) | scaleY  | 3.0000000000
   1 | (0, 0) | originX | 0.5000000000
   1 | (0, 0) | originY | 0.5000000000
(6 rows)
```

Siehe auch

ST_GeoReference, ST_SetGeoReference, ST_SkewX, ST_SkewY

11.7.5 ST_SetSRID

ST_SetSRID — Setzt die SRID eines Rasters auf einen bestimmten Ganzzahlwert. Die SRID wird in der Tabelle "spatial_ref_sys" definiert.

Synopsis

raster **ST_SetSRID**(raster rast, integer srid);

Description

Weist der SRID des Rasters einen bestimmten Ganzzahlwert zu.



Anmerkung
Diese Funktion führt keine Koordinatentransformation des Rasters durch - sie setzt nur die Metadaten, welche das Koordinatenreferenzsystem definieren, in dem der Raster vorliegt. Nützlich für spätere Koordinatentransformationen.

Siehe auch

Abschnitt 4.5, ST_SRID

11.7.6 ST_SetUpperLeft

ST_SetUpperLeft — Setzt den Wert der oberen linke Ecke des Rasters auf die projizierten X- und Y-Koordinaten.

Synopsis

raster **ST_SetUpperLeft**(raster rast, double precision x, double precision y);

Description

Setzt den Wert der oberen linke Ecke des Rasters auf die projizierten X- und Y-Koordinaten.

Beispiele

Code

```
SELECT ST_SetUpperLeft(rast, -71.01, 42.37)
FROM dummy_rast
WHERE rid = 2;
```

Siehe auch

ST_UpperLeftX, ST_UpperLeftY

11.7.7 ST_Resample

ST_Resample — Skaliert einen Raster mit einem bestimmten Algorithmus, neuen Dimensionen, einer beliebigen Gitterecke und über Parameter zur Georeferenzierung des Rasters, die angegeben oder von einem anderen Raster übernommen werden können.

Synopsis

raster **ST_Resample**(raster rast, integer width, integer height, double precision gridx=NULL, double precision gridy=NULL, double precision skewx=0, double precision skewy=0, text algorithm=NearestNeighbor, double precision maxerr=0.125);
raster **ST_Resample**(raster rast, double precision scalex=0, double precision scaley=0, double precision gridx=NULL, double precision gridy=NULL, double precision skewx=0, double precision skewy=0, text algorithm=NearestNeighbor, double precision maxerr=0.125);
raster **ST_Resample**(raster rast, raster ref, text algorithm=NearestNeighbor, double precision maxerr=0.125, boolean usescale=true);
raster **ST_Resample**(raster rast, raster ref, boolean usescale, text algorithm=NearestNeighbor, double precision maxerr=0.125);

Description

Skaliert einen Raster mit einem bestimmten Algorithmus, neuen Dimensionen (width & height), einer Gitterecke (gridx & gridy) und über Parameter zur Georeferenzierung des Rasters (scalex, scaley, skewx & skewy), die angegeben oder von einem anderen Raster übernommen werden können. Wenn Sie einen Referenzraster verwenden, müssen beide Raster die selbe SRID haben.

Die neuen Pixelwerte werden mit einem der folgenden Resampling-Algorithmen berechnet:

- NearestNeighbor (englische oder amerikanische Schreibweise)
- Biliniear

- Kubisch
- CubicSpline
- Lanczos
- Max
- Min

Die Standardeinstellung ist NearestNeighbor, die am schnellsten ist, aber die schlechteste Interpolation ergibt.

Wenn `maxerr` nicht angegeben ist, wird der maximale Fehler mit 0.125 Prozent angesetzt.



Anmerkung
Siehe GDAL Warp resampling methods für mehr Details.

Verfügbarkeit: 2.0.0 benötigt GDAL 1.6.1+

Verbessert: 3.4.0 Max und Min Resampling Optionen hinzugefügt

Beispiele

Code

```
SELECT
    ST_Width(orig) AS orig_width,
    ST_Width(reduce_100) AS new_width
FROM (
    SELECT
        rast AS orig,
        ST_Resample(rast, 100, 100) AS reduce_100
    FROM aerials.boston
    WHERE ST_Intersects(
        rast,
        ST_Transform(ST_MakeEnvelope(-71.128, 42.2392, -71.1277, 42.2397, 4326),
            26986)
    )
    LIMIT 1
) AS foo;
```

Output

```
orig_width | new_width
-----------+-------------
        200 |         100
```

Siehe auch

ST_Rescale, ST_Resize, ST_Transform

11.7.8 ST_Rescale

ST_Rescale — Neuabtastung eines Rasters, indem nur die Skala (oder Pixelgröße) angepasst wird. Die neuen Pixelwerte werden mit den Algorithmen NearestNeighbor (englische oder amerikanische Schreibweise), Bilinear, Cubic, CubicSpline, Lanczos, Max oder Min resampling berechnet. Die Voreinstellung ist 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

Sie können ein Raster neu abtasten, indem Sie nur den Maßstab (oder die Pixelgröße) anpassen. Die neuen Pixelwerte werden mit einem der folgenden Resampling-Algorithmen berechnet:

- NearestNeighbor (englische oder amerikanische Schreibweise)
- Biliniear
- Kubisch
- CubicSpline
- Lanczos
- Max
- Min

Die Standardeinstellung ist NearestNeighbor, die am schnellsten ist, aber die schlechteste Interpolation ergibt.

`scalex` und `scaley` bestimmen die neue Pixelgröße. Der Wert von "scaley" muss oftmals negativ sein, um einen ordnungsgemäß ausgerichteten Raster zu erhalten.

Wenn "scalex" oder "scaley" teilerfremd zur Breite oder Höhe des Rasters sind, dann wird der Zielraster auf die Ausdehnung des Ausgangsrasters erweitert. Um die exakte Ausdehnung des Ausgangsrasters sicher zu erhalten, siehe ST_Resize

`maxerr` ist der Schwellenwert bei der Näherungstransformation des Skalierungsalgorithmus (in Pixeleinheiten). Ein Standardwert von 0.125 wird verwendet, wenn kein `maxerr` angegeben wird. Dies ist dergleiche Wert, welcher von gdalwarp verwendet wird. Wenn der Wert auf Null gesetzt ist, wird keine Annährung ausgeführt.



Anmerkung
Siehe GDAL Warp resampling methods für mehr Details.



Anmerkung
ST_Rescale unterscheidet sich von ST_SetScale darin, dass ST_SetScale den Raster nicht skaliert um mit der Ausdehnung des Ausgangsrasters übereinzustimmen. ST_SetScale ändert lediglich die Metadaten (oder die Georeferenz) des Rasters, um eine ursprünglich falsch angegebene Skalierung zu korrigieren. ST_Rescale berechnet die Breite und Höhe eines Rasters so, dass er mit der geographischen Ausdehnung des Ausgangsraster übereinstimmt. ST_SetScale verändert weder die Breite noch die Höhe des Rasters.

Verfügbarkeit: 2.0.0 benötigt GDAL 1.6.1+

Verbessert: 3.4.0 Max und Min Resampling Optionen hinzugefügt

Änderung: 2.1.0 Funktioniert jetzt auch mit Raster ohne SRID

Beispiele

Ein einfaches Beispiel, das die Pixelgröße eines Raster von 0.001 Grad auf 0.0015 Grad ändert.

This example returns the original raster pixel size.

Code

```
SELECT ST_PixelWidth(ST_AddBand(ST_MakeEmptyRaster(100, 100, 0, 0, 0.001, -0.001, 0, 0,  ↩
    4269), '8BUI'::text, 1, 0)) width
```

Output

```
0.001
```

This example returns the pixel size of the rescaled raster.

Code

```
SELECT ST_PixelWidth(ST_Rescale(ST_AddBand(ST_MakeEmptyRaster(100, 100, 0, 0, 0.001, -0.001, ↩
     0, 0, 4269), '8BUI'::text, 1, 0), 0.0015)) width
```

Output

```
0.0015
```

Siehe auch

ST_Resize, ST_Resample, ST_SetScale, ST_ScaleX, ST_ScaleY, ST_Transform

11.7.9 ST_Reskew

ST_Reskew — Skaliert einen Raster, indem lediglich der Versatz (oder Rotationsparameter) angepasst wird. Neue Pixelwerte werden über NearestNeighbor, bilinear, kubisch, CubicSpline oder mit dem Lanczos-Filter errechnet. Die Standardeinstellung ist NearestNeighbor.

Synopsis

raster **ST_Reskew**(raster rast, double precision skewxy, text algorithm=NearestNeighbor, double precision maxerr=0.125);
raster **ST_Reskew**(raster rast, double precision skewx, double precision skewy, text algorithm=NearestNeighbor, double precision maxerr=0.125);

Description

Skaliert einen Raster, indem lediglich der Versatz (oder Rotationsparameter) angepasst wird. Neue Pixelwerte werden über NearestNeighbor, bilinear, kubisch, CubicSpline oder mit dem Lanczos-Filter errechnet. Die Standardeinstellung "NearestNeighbor" ist am schnellsten, ergibt aber die schlechteste Interpolation.

`skewx` und `skewy` legen den neuen Versatz fest.

Die Ausdehnung des Zielrasters erfasst die Ausdehnung des Ausgangsrasters.

Wenn `maxerr` nicht angegeben ist, wird der maximale Fehler mit 0.125 Prozent angesetzt.



Anmerkung
Siehe GDAL Warp resampling methods für mehr Details.



Anmerkung
ST_Reskew unterscheidet sich von ST_SetSkew darin, dass ST_SetSkew den Raster nicht skaliert um mit der Ausdehnung des Ausgangsrasters übereinzustimmen. ST_SetSkew ändert lediglich die Metadaten (oder die Georeferenz) des Rasters, um einen ursprünglich falsch angegebenen Versatz zu korrigieren. ST_Reskew ergibt einen Raster mit geänderter Breite und Höhe, da die Berechnung so durchgeführt wird, dass die geographischen Ausdehnung des Zielrasters mit der des Ausgangsrasters übereinstimmt. ST_SetSkew verändert weder die Breite noch die Höhe des Rasters.

Verfügbarkeit: 2.0.0 benötigt GDAL 1.6.1+

Änderung: 2.1.0 Funktioniert jetzt auch mit Raster ohne SRID

Beispiele

Ein einfaches Beispiel, das den Versatz eines Rasters von 0.0 auf 0.0015 ändert.

This example uses the original non-rotated raster.

Code

```
SELECT ST_Rotation(ST_AddBand(ST_MakeEmptyRaster(100, 100, 0, 0, 0.001, -0.001, 0, 0, 4269), ↩
    '8BUI'::text, 1, 0));
```

Output

```
st_rotation
-------------
           0
(1 row)
```

The reskewed raster rotation.

Code

```
SELECT ST_Rotation(ST_Reskew(ST_AddBand(ST_MakeEmptyRaster(100, 100, 0, 0, 0.001, -0.001, 0, ↩
    0, 4269), '8BUI'::text, 1, 0), 0.0015));
```

Output

```
st_rotation
------------------
 -0.982793723247329
(1 row)
```

Siehe auch

ST_Resample, ST_Rescale, ST_SetSkew, ST_SetRotation, ST_SkewX, ST_SkewY, ST_Transform

11.7.10 ST_SnapToGrid

ST_SnapToGrid — Skaliert einen Raster durch Fangen an einem Führungsgitter. Neue Pixelwerte werden über NearestNeighbor, bilinear, kubisch, CubicSpline oder mit dem Lanczos-Filter errechnet. Die Standardeinstellung ist NearestNeighbor.

Synopsis

raster **ST_SnapToGrid**(raster rast, double precision gridx, double precision gridy, text algorithm=NearestNeighbor, double precision maxerr=0.125, double precision scalex=DEFAULT 0, double precision scaley=DEFAULT 0);
raster **ST_SnapToGrid**(raster rast, double precision gridx, double precision gridy, double precision scalex, double precision scaley, text algorithm=NearestNeighbor, double precision maxerr=0.125);
raster **ST_SnapToGrid**(raster rast, double precision gridx, double precision gridy, double precision scalexy, text algorithm=NearestNeigh
double precision maxerr=0.125);

Description

Skaliert einen Raster durch Fangen an einem Führungsgitter, welches durch einen beliebige Pixeleckpunkt (gridx & gridy) und eine optionale Pixelgröße (scalex & scaley) definiert ist. Neue Pixelwerte werden über NearestNeighbor, bilinear, kubisch, CubicSpline oder mit dem Lanczos-Filter errechnet. Die Standardeinstellung "NearestNeighbor" ist am schnellsten, ergibt aber die schlechteste Interpolation.

`gridx` und `gridy` bestimmen einen beliebigen Pixeleckpunkt des neuen Gitters. Dies muss nicht unbedingt die obere linke Ecke des Zielrasters sein, darf aber nicht innerhalb oder am Rand der Ausdehnung des Zielrasters liegen.

Optional können Sie die Pixelgröße des neuen Gitters mit `scalex` und `scaley` festlegen.

Die Ausdehnung des Zielrasters erfasst die Ausdehnung des Ausgangsrasters.

Wenn `maxerr` nicht angegeben ist, wird der maximale Fehler mit 0.125 Prozent angesetzt.



Anmerkung
Siehe GDAL Warp resampling methods für mehr Details.



Anmerkung
Verwenden Sie bitte ST_Resample, wenn Sie eine bessere Kontrolle über die Gitterparameter benötigen.

Verfügbarkeit: 2.0.0 benötigt GDAL 1.6.1+

Änderung: 2.1.0 Funktioniert jetzt auch mit Raster ohne SRID

Beispiele

Ein einfaches Beispiel, bei dem ein Raster an einem sich geringfügig unterscheidenden Gitter gefangen wird.

This example returns the upper-left X coordinate of the original raster.

Code

```
SELECT ST_UpperLeftX(ST_AddBand(ST_MakeEmptyRaster(10, 10, 0, 0, 0.001, -0.001, 0, 0, 4269), ↩
     '8BUI'::text, 1, 0));
```

Output

```
st_upperleftx
--------------
             0
(1 row)
```

The upper left of raster after snapping.

Code

```
SELECT ST_UpperLeftX(ST_SnapToGrid(ST_AddBand(ST_MakeEmptyRaster(10, 10, 0, 0, 0.001,  ↩
    -0.001, 0, 0, 4269), '8BUI'::text, 1, 0), 0.0002, 0.0002));
```

Output

```
st_upperleftx
---------------
       -0.0008
(1 row)
```

Siehe auch

ST_Resample, ST_Rescale, ST_UpperLeftX, ST_UpperLeftY

11.7.11 ST_Resize

ST_Resize — Ändert die Zellgröße - width/height - eines Rasters

Synopsis

raster **ST_Resize**(raster rast, integer width, integer height, text algorithm=NearestNeighbor, double precision maxerr=0.125);
raster **ST_Resize**(raster rast, double precision percentwidth, double precision percentheight, text algorithm=NearestNeighbor, double precision maxerr=0.125);
raster **ST_Resize**(raster rast, text width, text height, text algorithm=NearestNeighbor, double precision maxerr=0.125);

Description

Passt die Größe des Rasters an eine neue Breite/Höhe an. Die neue Breite/Höhe kann durch die genaue Anzahl der Pixel, oder durch einen Prozentsatz der Breite/Höhe des Rasters, angegeben werden. Die Ausdehnung des Zielrasters ist mit der Ausdehnung des Ausgangsrasters ident.

Neue Pixelwerte werden über NearestNeighbor, bilinear, kubisch, CubicSpline oder mit dem Lanczos-Filter errechnet. Die Standardeinstellung "NearestNeighbor" ist am schnellsten, erzeugt aber auch die schlechteste Interpolation.

Variante 1 erwartet die tatsächliche width/height des Ausgaberasters.

Variante 2 erwartet Dezimalwerte zwischen null (0) und eins (1), welche das Verhältnis zur Pixelbreite und zur Pixelhöhe des Eingaberasters angeben.

Variante 3 nimmt entweder die tatsächliche Breite/Höhe des Zielrasters oder einen Prozentsatz der Breite/Höhe des Ausgangsrasters als Zeichenfolge ("20%") entgegen.

Verfügbarkeit: 2.1.0 benötigt GDAL 1.6.1+

Beispiele

Code

```
WITH foo AS(
SELECT
    1 AS rid,
    ST_Resize(ST_AddBand(ST_MakeEmptyRaster(1000, 1000, 0, 0, 1, -1, 0, 0, 0),
            1, '8BUI', 255, 0
        ),
    '50%', '500') AS rast
UNION ALL
SELECT
    2 AS rid,
    ST_Resize(ST_AddBand(ST_MakeEmptyRaster(1000, 1000, 0, 0, 1, -1, 0, 0, 0),
            1, '8BUI', 255, 0
        ),
```

```
    500, 100) AS rast
UNION ALL
SELECT
    3 AS rid,
    ST_Resize(ST_AddBand(ST_MakeEmptyRaster(1000, 1000, 0, 0, 1, -1, 0, 0, 0),
            1, '8BUI', 255, 0
        ),
    0.25, 0.9) AS rast
)
SELECT
    rid,
    ST_Width(rast) AS width,
    ST_Height(rast) AS height,
    ST_ScaleX(rast) AS scale_x,
    round(ST_ScaleY(rast)::numeric, 4) AS scale_y
FROM foo
ORDER BY rid
```

Output

```
rid | width | height | scale_x | scale_y
-----+-------+--------+---------+---------
   1 |   500 |    500 |       2 | -2.0000
   2 |   500 |    100 |       2 | -10.0000
   3 |   250 |    900 |       4 | -1.1111
(3 rows)
```

Siehe auch

ST_Resample, ST_Rescale, ST_Reskew, ST_SnapToGrid

11.7.12 ST_Transform

ST_Transform — Projiziert einen Raster von einem bekannten Koordinatenreferenzsystem in ein anderes bekanntes Koordinatenreferenzsystem um. Die Optionen für die Skalierung sind NearestNeighbor, Bilinear, Cubisch, CubicSpline und der Lanczos-Filter, die Standardeinstellung ist NearestNeighbor.

Synopsis

raster **ST_Transform**(raster rast, integer srid, text algorithm=NearestNeighbor, double precision maxerr=0.125, double precision scalex, double precision scaley);
raster **ST_Transform**(raster rast, integer srid, double precision scalex, double precision scaley, text algorithm=NearestNeighbor, double precision maxerr=0.125);
raster **ST_Transform**(raster rast, raster alignto, text algorithm=NearestNeighbor, double precision maxerr=0.125);

Description

Projiziert einen Raster von einem bekannten Koordinatenreferenzsystem in ein anderes bekanntes Koordinatenreferenzsystem um. Verwendet den angegebenen Algorithmus für die Interpolation der Pixelwerte. Wenn kein Algorithmus angegeben ist, wird 'NearestNeighbor' verwendet. Wenn "maxerr" nicht angegeben ist, wird ein Prozentsatz von 0.125 angenommen.

Die Optionen für den Algorithmus sind: 'NearestNeighbor', 'Bilinear', 'Cubic', 'CubicSpline' und 'Lanczos'. Siehe GDAL Warp resampling methods für weitere Details.

ST_Transform wird oft mit ST_SetSRID() verwechselt. ST_Transform ändert die Koordinaten der Geometrie tatsächlich (und die Pixelwerte mittels "resampling") von einem Koordinatenreferenzsystem auf ein anderes, während ST_SetSRID() nur den Identifikator "SRID" des Rasters ändert.

Anders als die anderen Varianten, benötigt Variante 3 einen Referenzraster für den Parameter `alignto`. Der Raster wird in das Koordinatenreferenzsystem (SRID) des Referenzrasters transformiert und an dem Referenzraster ausgerichtet (ST_SameAlignment = TRUE).



Anmerkung
If you find your transformation support is not working right, you may need to set the environment variable PROJSO to the projection library your PostGIS build 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`. You'll have to restart your PostgreSQL service/daemon after this change.



Warnung
Bei der Umwandlung einer Abdeckung von Kacheln sollten Sie fast immer ein Referenzraster verwenden, um sicherzustellen, dass die Kacheln gleich ausgerichtet sind und keine Lücken aufweisen, wie im Beispiel gezeigt: Variante 3.

Verfügbarkeit: 2.0.0 benötigt GDAL 1.6.1+

Erweiterung: 2.1.0 Variante ST_Transform(rast, alignto) hinzugefügt

Beispiele

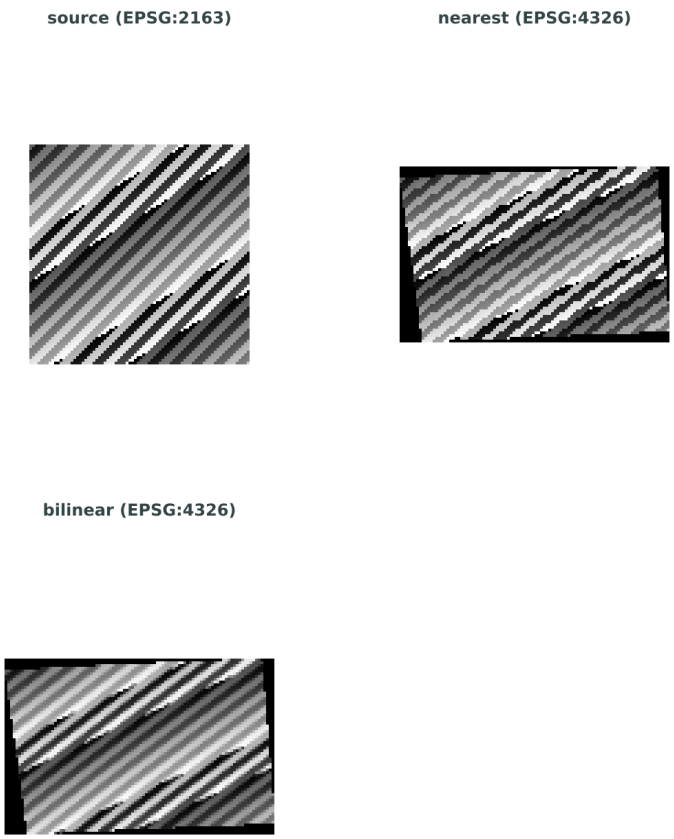
Compare a source raster with nearest-neighbor and bilinear reprojections. The diagonal bands make the resampling behavior visible: nearest-neighbor keeps hard pixel steps, while bilinear smooths values between pixels. All three figures are generated from this query at manual build time.

Code

```
WITH rows AS (
    SELECT
        y,
        array_agg(
            ((
                    x * 3 + y * 5
                    + CASE WHEN (x + y) % 8 < 4 THEN 80 ELSE 0 END
                ) % 255)::double precision
            ORDER BY x
        ) AS row_values
    FROM generate_series(1, 80) AS y
    CROSS JOIN generate_series(1, 80) AS x
    GROUP BY y
), source AS (
    SELECT ST_SetValues(
        ST_AddBand(
            ST_MakeEmptyRaster(
                80, 80,
                -500000, 600000,
                2500, -2500,
                0, 0,
                2163
            ),
            1, '8BUI', 0, NULL
        ),
        1, 1, 1,
        array_agg(row_values ORDER BY y)::double precision[][]
    ) AS rast
    FROM rows
), transformed AS (
```

```
    SELECT
        rast,
        ST_Transform(rast, 4326, 'NearestNeighbor') AS nearest,
        ST_Transform(rast, 4326, 'Bilinear') AS bilinear
    FROM source
)
SELECT
    ST_AsPNG(rast) AS "source (EPSG:2163)",
    ST_AsPNG(nearest) AS "nearest (EPSG:4326)",
    ST_AsPNG(bilinear) AS "bilinear (EPSG:4326)"
FROM transformed;
```

Figure



Raster reprojection and resampling.



Massachusetts State Plane source



WGS 84 nearest-neighbor



WGS 84 bilinear

Variant 3.

Im Folgenden wird der Unterschied zwischen der Verwendung von ST_Transform(raster, srid) und ST_Transform(raster, alignto) gezeigt

Code

```
WITH foo AS (
    SELECT 0 AS rid, ST_AddBand(ST_MakeEmptyRaster(2, 2, -500000, 600000, 100, -100, 0, 0,  ↩
    2163), 1, '16BUI', 1, 0) AS rast UNION ALL
    SELECT 1, ST_AddBand(ST_MakeEmptyRaster(2, 2, -499800, 600000, 100, -100, 0, 0, 2163),  ↩
    1, '16BUI', 2, 0) AS rast UNION ALL
    SELECT 2, ST_AddBand(ST_MakeEmptyRaster(2, 2, -499600, 600000, 100, -100, 0, 0, 2163),  ↩
    1, '16BUI', 3, 0) AS rast UNION ALL

    SELECT 3, ST_AddBand(ST_MakeEmptyRaster(2, 2, -500000, 599800, 100, -100, 0, 0, 2163),  ↩
    1, '16BUI', 10, 0) AS rast UNION ALL
    SELECT 4, ST_AddBand(ST_MakeEmptyRaster(2, 2, -499800, 599800, 100, -100, 0, 0, 2163),  ↩
    1, '16BUI', 20, 0) AS rast UNION ALL
    SELECT 5, ST_AddBand(ST_MakeEmptyRaster(2, 2, -499600, 599800, 100, -100, 0, 0, 2163),  ↩
    1, '16BUI', 30, 0) AS rast UNION ALL

    SELECT 6, ST_AddBand(ST_MakeEmptyRaster(2, 2, -500000, 599600, 100, -100, 0, 0, 2163),  ↩
    1, '16BUI', 100, 0) AS rast UNION ALL
    SELECT 7, ST_AddBand(ST_MakeEmptyRaster(2, 2, -499800, 599600, 100, -100, 0, 0, 2163),  ↩
    1, '16BUI', 200, 0) AS rast UNION ALL
    SELECT 8, ST_AddBand(ST_MakeEmptyRaster(2, 2, -499600, 599600, 100, -100, 0, 0, 2163),  ↩
    1, '16BUI', 300, 0) AS rast
), bar AS (
    SELECT
        ST_Transform(rast, 4269) AS alignto
    FROM foo
    LIMIT 1
), baz AS (
    SELECT
        rid,
        rast,
        ST_Transform(rast, 4269) AS not_aligned,
        ST_Transform(rast, alignto) AS aligned
    FROM foo
    CROSS JOIN bar
)
SELECT
    ST_Collect(ST_ConvexHull(not_aligned) ORDER BY rid) AS not_aligned,
```

```
    ST_Collect(ST_ConvexHull(aligned) ORDER BY rid) AS aligned
FROM baz
```

Output

```
MULTIPOLYGON(((-107.007 50.2,-107.005 50.2,-107.005 50.198,-107.007 50.198,-107.007 50.2)),((-107.004 50.201,-107.002 ↩
    50.201,-107.002 50.198,-107.004 50.198,-107.004 50.201)),((-107.001 50.201,-106.999 50.201,-106.999 ↩
    50.198,-107.001 50.198,-107.001 50.201)),((-107.007 50.199,-107.004 50.199,-107.004 50.196,-107.007 ↩
    50.196,-107.007 50.199)),((-107.004 50.199,-107.001 50.199,-107.001 50.196,-107.004 50.196,-107.004 50.199)) ↩
    ,((-107.001 50.199,-106.999 50.199,-106.999 50.196,-107.001 50.196,-107.001 50.199)),((-107.006 50.197,-107.004 ↩
    50.197,-107.004 50.194,-107.006 50.194,-107.006 50.197)),((-107.004 50.197,-107.001 50.197,-107.001 ↩
    50.194,-107.004 50.194,-107.004 50.197)),((-107.001 50.197,-106.998 50.197,-106.998 50.195,-107.001 ↩
    50.195,-107.001 50.197))) | MULTIPOLYGON(((-107.007 50.2,-107.005 50.2,-107.005 50.198,-107.007 50.198,-107.007 ↩
    50.2)),((-107.005 50.202,-107.001 50.202,-107.001 50.198,-107.005 50.198,-107.005 50.202)),((-107.002 ↩
    50.202,-106.998 50.202,-106.998 50.198,-107.002 50.198,-107.002 50.202)),((-107.007 50.199,-107.003 ↩
    50.199,-107.003 50.195,-107.007 50.195,-107.007 50.199)),((-107.005 50.199,-107.001 50.199,-107.001 ↩
    50.195,-107.005 50.195,-107.005 50.199)),((-107.002 50.199,-106.998 50.199,-106.998 50.195,-107.002 ↩
    50.195,-107.002 50.199)),((-107.007 50.198,-107.003 50.198,-107.003 50.194,-107.007 50.194,-107.007 50.198)) ↩
    ,((-107.005 50.198,-107.001 50.198,-107.001 50.194,-107.005 50.194,-107.005 50.198)),((-107.002 50.198,-106.998 ↩
    50.198,-106.998 50.194,-107.002 50.194,-107.002 50.198)))
```

Figure



Siehe auch

ST_Transform, ST_SetSRID

11.8 Editoren für Rasterbänder

11.8.1 ST_SetBandNoDataValue

ST_SetBandNoDataValue — Setzt den NODATA Wert eines Bandes. Wenn kein Band angegeben ist, wird Band 1 angenommen. Falls ein Band keinen NODATA Wert aufweisen soll, übergeben Sie bitte für den Parameter "nodatavalue" NULL.

Synopsis

raster **ST_SetBandNoDataValue**(raster rast, double precision nodatavalue);
raster **ST_SetBandNoDataValue**(raster rast, integer band, double precision nodatavalue, boolean forcechecking=false);

Description

Setzt den NODATA Wert eines Bandes. Wenn kein Band angegeben ist, wird Band 1 angenommen. Dies beeinflusst die Ergebnisse von ST_Polygon, ST_DumpAsPolygons und die Funktionen ST_PixelAs...().

Beispiele

This example changes only the first band NODATA value.

Code

```
UPDATE dummy_rast
    SET rast = ST_SetBandNoDataValue(rast, 1, 254)
WHERE rid = 2;
```

This example changes the NODATA value for bands 1, 2, and 3.

Code

```
UPDATE dummy_rast
    SET rast =
        ST_SetBandNoDataValue(ST_SetBandNoDataValue(ST_SetBandNoDataValue(rast, 1, 254),
                2, 99),
                3, 108)
        WHERE rid = 2;
```

This example removes the NODATA value so that all pixels are considered by processing functions.

Code

```
UPDATE dummy_rast
    SET rast = ST_SetBandNoDataValue(rast, 1, NULL)
WHERE rid = 2;
```

Siehe auch

ST_BandNoDataValue, ST_NumBands

11.8.2 ST_SetBandIsNoData

ST_SetBandIsNoData — Setzt die Flag "isnodata" für das Band auf TRUE.

Synopsis

raster **ST_SetBandIsNoData**(raster rast, integer band=1);

Description

Setzt die Flag "isnodata" des Bandes auf TRUE. Wenn kein Band angegeben ist, wird Band 1 angenommen. Diese Funktion sollte nur aufgerufen werden, wenn sich die Flag verändert hat. Dies ist der Fall, wenn sich die Ergebnisse von ST_BandIsNoData unterscheiden, da einmal mit TRUE als letzten Übergabewert und ein anderes mal ohne diesen aufgerufen wurde.

Verfügbarkeit: 2.0.0

Beispiele

This example creates a dummy table with one raster column. The raster has two bands and one pixel per band. In the first band, the NODATA value and pixel value are both 3. In the second band, the NODATA value is 13 and the pixel value is 4.

Code

```
create table dummy_rast (rid integer, rast raster);

insert into dummy_rast values(
1,
(
'01' -- little endian (uint8 ndr)
||
'0000' -- version (uint16 0)
||
'0200' -- nBands (uint16 0)
||
'17263529ED684A3F' -- scaleX (float64 0.000805965234044584)
||
'F9253529ED684ABF' -- scaleY (float64 -0.00080596523404458)
||
'1C9F33CE69E352C0' -- ipX (float64 -75.5533328537098)
||
'718F0E9A27A44840' -- ipY (float64 49.2824585505576)
||
'ED50EB853EC32B3F' -- skewX (float64 0.000211812383858707)
||
'7550EB853EC32B3F' -- skewY (float64 0.000211812383858704)
||
'E6100000' -- SRID (int32 4326)
||
'0100' -- width (uint16 1)
||
'0100' -- height (uint16 1)
||
'4' -- hasnodatavalue set to true, isnodata value set to false (when it should be true)
||
'2' -- first band type (4BUI)
||
'03' -- novalue==3
||
'03' -- pixel(0,0)==3 (same that nodata)
||
'0' -- hasnodatavalue set to false
||
'5' -- second band type (16BSI)
||
'0D00' -- novalue==13
||
'0400' -- pixel(0,0)==4
)::raster
);

select ST_BandIsNoData(rast, 1) from dummy_rast where rid = 1; -- Expected false
select ST_BandIsNoData(rast, 1, TRUE) from dummy_rast where rid = 1; -- Expected true
```

The `isnodata` flag is dirty, so set it to true.

Code

```
update dummy_rast set rast = ST_SetBandIsNoData(rast, 1) where rid = 1;


select ST_BandIsNoData(rast, 1) from dummy_rast where rid = 1; -- Expected true
```

Siehe auch

ST_BandNoDataValue, ST_NumBands, ST_SetBandNoDataValue, ST_BandIsNoData

11.8.3 ST_SetBandPath

ST_SetBandPath — Aktualisiert den externen Dateipfad und die Bandnummer eines out-db Bandes.

Synopsis

raster **ST_SetBandPath**(raster rast, integer band, text outdbpath, integer outdbindex, boolean force=false);

Description

Aktualisiert den externen Rasterdateipfad und die externe Bandnummer eines out-db Bandes.



Anmerkung
Wenn `force` auf TRUE gesetzt ist, wird die Kompatibilität (z.B. Ausrichtung, Pixelunterstützung) zwischen der externen Rasterdatei und dem PostGIS Raster nicht überprüft. Dieser Modus ist für Dateisystemänderungen vorgesehen, bei denen der externe Raster bestehen bleibt.

Verfügbarkeit: 2.5.0

Beispiele

Code

```
WITH foo AS (
    SELECT ST_AddBand(
        NULL::raster,
        '/home/pele/devel/geo/postgis-git/raster/test/regress/loader/Projected.tif',
        NULL::int[]
    ) AS rast
)
SELECT
    1 AS query,
    bandnum,
    regexp_replace(path, '^.*/', '') AS path,
    outdbbandnum
FROM ST_BandMetaData(
    (SELECT rast FROM foo),
    ARRAY[1, 3, 2]::int[]
) AS metadata
UNION ALL
SELECT
    2, bandnum,
    regexp_replace(path, '^.*/', ''),
    outdbbandnum
FROM ST_BandMetaData((
        SELECT
            ST_SetBandPath(rast,
                2,
                '/home/pele/devel/geo/postgis-git/raster/test/regress/loader/Projected2.tif' ↩
    ,
                1
```

```
            ) AS rast
        FROM foo
    ),
    ARRAY[1, 3, 2]::int[]
) AS metadata
ORDER BY 1, 2;
```

In the second result set, band 2 now points at `Projected2.tif` and uses out-db band 1.

Output

```
query | bandnum | path           | outdbbandnum
------+---------+----------------+--------------
    1 |       1 | Projected.tif  |            1
    1 |       2 | Projected.tif  |            2
    1 |       3 | Projected.tif  |            3
    2 |       1 | Projected.tif  |            1
    2 |       2 | Projected2.tif |            1
    2 |       3 | Projected.tif  |            3
```

Siehe auch

ST_BandMetaData, ST_SetBandIndex

11.8.4 ST_SetBandIndex

ST_SetBandIndex — Aktualisiert die externe Bandnummer eines out-db Bandes.

Synopsis

raster **ST_SetBandIndex**(raster rast, integer band, integer outdbindex, boolean force=false);

Description

Aktualisiert die externe Bandnummer eines out-db Bandes. Die mit dem out-db Band assoziierte externe Rasterdatei wird davon nicht betroffen



Anmerkung
Wenn `force` auf TRUE gesetzt ist, wird die Kompatibilität (z.B. Ausrichtung, Pixelunterstützung) zwischen der externen Rasterdatei und dem PostGIS Raster nicht überprüft. Dieser Modus ist für das Verschieben von Bändern in einer externen Rasterdatei vorgesehen.



Anmerkung
Intern wird bei dieser Methode das PostGIS Raster Band mit dem Index `band` durch ein neues Band ersetzt, ohne dass die existierende Pfadinformation aktualisiert wird.

Verfügbarkeit: 2.5.0

Beispiele

Code

```
WITH foo AS (
    SELECT ST_AddBand(
        NULL::raster,
        '/home/pele/devel/geo/postgis-git/raster/test/regress/loader/Projected.tif',
        NULL::int[]
    ) AS rast
)
SELECT
    1 AS query,
    bandnum,
    regexp_replace(path, '^.*/', '') AS path,
    outdbbandnum
FROM ST_BandMetaData(
    (SELECT rast FROM foo),
    ARRAY[1, 3, 2]::int[]
) AS metadata
UNION ALL
SELECT
    2, bandnum,
    regexp_replace(path, '^.*/', ''),
    outdbbandnum
FROM ST_BandMetaData((
        SELECT
            ST_SetBandIndex(rast,
                2,
                1
            ) AS rast
        FROM foo
    ),
    ARRAY[1, 3, 2]::int[]
) AS metadata
ORDER BY 1, 2;
```

In the second result set, band 2 still points at the same file but now reads external band 1.

Output

```
query | bandnum | path          | outdbbandnum
------+---------+---------------+-------------
    1 |       1 | Projected.tif |            1
    1 |       2 | Projected.tif |            2
    1 |       3 | Projected.tif |            3
    2 |       1 | Projected.tif |            1
    2 |       2 | Projected.tif |            1
    2 |       3 | Projected.tif |            3
```

Siehe auch

ST_BandMetaData, ST_SetBandPath

11.9 Rasterband Statistik und Analytik

11.9.1 ST_Count

ST_Count — Gibt die Anzahl der Pixel für ein Band eines Rasters oder eines Raster-Coverage zurück. Wenn kein Band angegeben ist, wird standardmäßig Band 1 gewählt. Wenn der Parameter "exclude_nodata_value" auf TRUE gesetzt ist, werden nur Pixel mit Werten ungleich NODATA gezählt.

Synopsis

bigint **ST_Count**(raster rast, integer nband=1, boolean exclude_nodata_value=true);
bigint **ST_Count**(raster rast, boolean exclude_nodata_value);

Description

Gibt die Anzahl der Pixel für ein Band eines Rasters oder eines Raster-Coverage zurück. Wenn kein Band angegeben ist, wird für `nband` 1 vorausgesetzt.



Anmerkung
Wenn der Parameter `exclude_nodata_value` auf TRUE gesetzt ist, werden nur die Pixel des Rasters gezählt, deren Werte ungleich `nodata` sind. Setzen Sie bitte `exclude_nodata_value` auf FALSE um die Anzahl aller Pixel zu erhalten

Geändert: 3.1.0 - Die ST_Count(rastertable, rastercolumn, ...) Varianten wurden entfernt. Verwenden Sie stattdessen ST_CountAgg.

Verfügbarkeit: 2.0.0

Beispiele

Example will count all pixels not 249 and one will count all pixels.

Code

```
SELECT rid, ST_Count(ST_SetBandNoDataValue(rast, 249)) As exclude_nodata,
        ST_Count(ST_SetBandNoDataValue(rast, 249), false) As include_nodata
    FROM dummy_rast WHERE rid=2;
```

Output

```
rid | exclude_nodata | include_nodata
-----+----------------+----------------
   2 |             23 |             25
```

Siehe auch

ST_CountAgg, ST_SummaryStats, ST_SetBandNoDataValue

11.9.2 ST_CountAgg

ST_CountAgg — Aggregatfunktion. Gibt die Anzahl der Pixel in einem bestimmten Band der Raster aus. Wenn kein Band angegeben ist, wird Band 1 angenommen. Wenn "exclude_nodata_value" TRUE ist, werden nur die Pixel ohne NODATA Werte gezählt.

Synopsis

bigint **ST_CountAgg**(raster rast, integer nband, boolean exclude_nodata_value, double precision sample_percent);
bigint **ST_CountAgg**(raster rast, integer nband, boolean exclude_nodata_value);
bigint **ST_CountAgg**(raster rast, boolean exclude_nodata_value);

Description

Gibt die Anzahl der Pixel in einem bestimmten Band der Raster aus. Wenn kein Band angegeben ist, wird `nband` standardmäßig auf 1 gesetzt.

Wenn der Parameter `exclude_nodata_value` auf TRUE gesetzt ist, werden nur die Pixel des Rasters gezählt, deren Werte ungleich `NODATA` sind. Setzen Sie bitte `exclude_nodata_value` auf FALSE um die Anzahl aller Pixel zu erhalten

Standardmäßig werden alle Pixel abgetastet. Um eine schnellere Rückmeldung zu bekommen, können Sie den Parameter `sample_perc` auf einen Wert zwischen null (0) und eins (1) setzen.

Verfügbarkeit: 2.2.0

Beispiele

Code

```
WITH foo AS (
    SELECT
        rast.rast
    FROM (
        SELECT ST_SetValue(ST_SetValue(ST_SetValue(ST_AddBand(ST_MakeEmptyRaster(10, 10, 10, ↩
     10, 2, 2, 0, 0, 0),
                        1, '64BF', 0, 0
                    ),
                    1, 1, 1, -10
                ),
                1, 5, 4, 0
            ),
            1, 5, 5, 3.14159
        ) AS rast
    ) AS rast
    FULL JOIN (
        SELECT generate_series(1, 10) AS id
    ) AS id
        ON 1 = 1
)
SELECT
    ST_CountAgg(rast, 1, TRUE) AS non_nodata_pixels
FROM foo;
```

Output

```
non_nodata_pixels
------------------
                20
(1 row)
```

Siehe auch

ST_Count, ST_SummaryStats, ST_SetBandNoDataValue

11.9.3 ST_Histogram

ST_Histogram — Gibt Datensätze aus, welche die Verteilung der Daten eines Rasters oder eines Rastercoverage darstellen. Dabei wird die Wertemenge in Klassen aufgeteilt und für jede Klasse zusammengefasst. Wenn die Anzahl der Klassen nicht angegeben ist, wird sie automatisch berechnet.

Synopsis

SETOF record **ST_Histogram**(raster rast, integer nband=1, boolean exclude_nodata_value=true, integer bins=autocomputed, double precision[] width=NULL, boolean right=false);
SETOF record **ST_Histogram**(raster rast, integer nband, integer bins, double precision[] width=NULL, boolean right=false);
SETOF record **ST_Histogram**(raster rast, integer nband, boolean exclude_nodata_value, integer bins, boolean right);
SETOF record **ST_Histogram**(raster rast, integer nband, integer bins, boolean right);

Description

Gibt Datensätze aus, die das Minimum, das Maximum, die Anzahl und die Prozent der Werte des Rasterbandes für jede Klasse enthalten. Wenn kein Band angegeben ist, wird `nband` standardmäßig auf 1 gesetzt.



Anmerkung
Standardmäßig werden nur jene Pixelwerte berücksichtigt, die nicht den Wert `NODATA` haben. Setzen Sie bitte `exclude_nodata_value` auf FALSE um die Anzahl sämtlicher Pixel zu erhalten.

`width` Breite: ein Feld das die Breite für jede Kategorie/Klasse angibt. Wenn die Anzahl der Klassen größer ist als die Anzahl der Breiten, werden die Breiten wiederholt.

Beispiel: 9 Klassen, die Breiten sind [a, b, c] und werden als [a, b, c, a, b, c, a, b, c] übergegeben.

`bins` Anzahl der Klassen - die Anzahl der Datensätze die von der Funktion ausgegeben werden. Wenn die Anzahl der Klassen nicht angegeben ist, wird sie automatisch berechnet.

`right` Berechnet das Histogramm von rechts anstatt von links (Standardwert). Dies ändert das Auswahlkriterium für einen Wert x von [a,b) auf (a,b].

Geändert: 3.1.0 Die Variante ST_Histogram(table_name, column_name) wurde entfernt.

Verfügbarkeit: 2.0.0

Beispiele

Single raster tile - compute histograms for bands 1, 2, 3 and autocompute bins.

Code

```
SELECT band, (stats).*
FROM (SELECT rid, band, ST_Histogram(rast, band) As stats
    FROM dummy_rast CROSS JOIN generate_series(1, 3) As band
     WHERE rid=2) As foo;
```

Output

```
band |  min  |  max  | count | percent
------+-------+-------+-------+---------
   1 |   249 |   250 |     2 |    0.08
   1 |   250 |   251 |     2 |    0.08
   1 |   251 |   252 |     1 |    0.04
   1 |   252 |   253 |     2 |    0.08
   1 |   253 |   254 |    18 |    0.72
   2 |    78 | 113.2 |    11 |    0.44
   2 | 113.2 | 148.4 |     4 |    0.16
   2 | 148.4 | 183.6 |     4 |    0.16
   2 | 183.6 | 218.8 |     1 |    0.04
   2 | 218.8 |   254 |     5 |     0.2
   3 |    62 | 100.4 |    11 |    0.44
   3 | 100.4 | 138.8 |     5 |     0.2
   3 | 138.8 | 177.2 |     4 |    0.16
   3 | 177.2 | 215.6 |     1 |    0.04
   3 | 215.6 |   254 |     4 |    0.16
```

Just band 2 but for 6 bins.

Code

```
SELECT (stats).*
FROM (SELECT rid, ST_Histogram(rast, 2, 6) As stats
    FROM dummy_rast
     WHERE rid=2) As foo;
```

Output

```
min     |    max     | count | percent
------------+------------+-------+---------
         78 | 107.333333 |     9 |    0.36
 107.333333 | 136.666667 |     6 |    0.24
 136.666667 |        166 |     0 |       0
        166 | 195.333333 |     4 |    0.16
 195.333333 | 224.666667 |     1 |    0.04
 224.666667 |        254 |     5 |     0.2
(6 rows)
```

Same as previous but we explicitly control the pixel value range of each bin.

Code

```
SELECT (stats).*
FROM (SELECT rid, ST_Histogram(rast, 2, 6, ARRAY[0.5, 1, 4, 100, 5]) As stats
    FROM dummy_rast
     WHERE rid=2) As foo;
```

Output

```
min  |  max  | count | percent
-------+-------+-------+----------
    78 |  78.5 |     1 |     0.08
  78.5 |  79.5 |     1 |     0.04
  79.5 |  83.5 |     0 |        0
  83.5 | 183.5 |    17 |   0.0068
 183.5 | 188.5 |     0 |        0
 188.5 |   254 |     6 | 0.003664
(6 rows)
```

Siehe auch

ST_Count, ST_SummaryStats, ST_SummaryStatsAgg

11.9.4 ST_Quantile

ST_Quantile — Berechnet die Quantile eines Rasters oder einer Rastercoverage Tabelle im Kontext von Stichproben oder Bevölkerung. Dadurch kann untersucht werden, ob ein Wert bei 25%, 50% oder 75% Perzentil des Rasters liegt.

Synopsis

SETOF record **ST_Quantile**(raster rast, integer nband=1, boolean exclude_nodata_value=true, double precision[] quantiles=NULL);
SETOF record **ST_Quantile**(raster rast, double precision[] quantiles);
SETOF record **ST_Quantile**(raster rast, integer nband, double precision[] quantiles);
double precision **ST_Quantile**(raster rast, double precision quantile);
double precision **ST_Quantile**(raster rast, boolean exclude_nodata_value, double precision quantile=NULL);
double precision **ST_Quantile**(raster rast, integer nband, double precision quantile);
double precision **ST_Quantile**(raster rast, integer nband, boolean exclude_nodata_value, double precision quantile);
double precision **ST_Quantile**(raster rast, integer nband, double precision quantile);

Description

Berechnet die Quantile eines Rasters oder einer Rastercoverage Tabelle im Kontext von Stichproben oder Bevölkerung. Dadurch kann untersucht werden, ob ein Wert bei 25%, 50% oder 75% Perzentil des Rasters liegt.



Anmerkung
Wenn `exclude_nodata_value` auf FALSE gesetzt ist, werden auch die Pixel mit NODATA Werten gezählt.

Geändert: 3.1.0 Die Variante ST_Quantile(table_name, column_name) wurde entfernt.

Verfügbarkeit: 2.0.0

Beispiele

Example will consider only pixels of band 1 that are not 249 and in named quantiles.

Code

```
UPDATE dummy_rast SET rast = ST_SetBandNoDataValue(rast, 249) WHERE rid=2;

SELECT (pvq).*
FROM (SELECT ST_Quantile(rast, ARRAY[0.25, 0.75]) As pvq
    FROM dummy_rast WHERE rid=2) As foo
    ORDER BY (pvq).quantile;
```

Output

```
quantile | value
----------+-------
    0.25 |   253
    0.75 |   254
```

Code

```
SELECT ST_Quantile(rast, 0.75) As value
FROM dummy_rast WHERE rid=2;
```

Output

```
254
```

Real live example. Quantile of all pixels in band 2 intersecting a geometry.

Code

```
SELECT rid, (ST_Quantile(rast, 2)).* As pvc
    FROM o_4_boston
        WHERE ST_Intersects(
            rast,
            ST_GeomFromText('POLYGON((224486 892151,224486 892200,224706 892200,224706 ↩
    892151,224486 892151))', 26986)
            )
ORDER BY value, quantile, rid
;
```

Output

```
rid | quantile | value
-----+----------+-------
   1 |        0 |     0
   2 |        0 |     0
  14 |        0 |     1
  15 |        0 |     2
  14 |     0.25 |    37
   1 |     0.25 |    42
  15 |     0.25 |    47
   2 |     0.25 |    50
  14 |      0.5 |    56
   1 |      0.5 |    64
  15 |      0.5 |    66
   2 |      0.5 |    77
  14 |     0.75 |    81
  15 |     0.75 |    87
   1 |     0.75 |    94
   2 |     0.75 |   106
  14 |        1 |   199
   1 |        1 |   244
   2 |        1 |   255
  15 |        1 |   255
```

Siehe auch

ST_Count, ST_SummaryStats, ST_SummaryStatsAgg, ST_SetBandNoDataValue

11.9.5 ST_SummaryStats

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 if no band is specified.

Synopsis

summarystats **ST_SummaryStats**(raster rast, boolean exclude_nodata_value);
summarystats **ST_SummaryStats**(raster rast, integer nband, boolean exclude_nodata_value);

Description

Gibt summarystats aus, bestehend aus der Anzahl, der Summe, dem arithmetischen Mittel, der Standardabweichung, dem Minimum und dem Maximum der Werte eines Rasterbandes oder eines Rastercoverage. Wenn kein Band angegeben ist, wird Band 1 angenommen.



Anmerkung
Standardmäßig werden nur jene Pixelwerte berücksichtigt, die nicht den Wert `NODATA` haben. Setzen Sie bitte `exclude_nodata_value` auf FALSE um die Anzahl sämtlicher Pixel zu erhalten.



Anmerkung
Standardmäßig werden alle Pixel abgetastet. Um eine schnellere Rückmeldung zu bekommen, können Sie den Parameter `sample_percent` auf einen Wert kleiner als 1 setzen.

Geändert: 3.1.0 ST_SummaryStats(rastertable, rastercolumn, ...) Varianten wurden entfernt. Verwenden Sie stattdessen ST_SummaryStat

Verfügbarkeit: 2.0.0

Beispiele

Single raster tile.

Code

```
SELECT rid, band, (stats).*
FROM (SELECT rid, band, ST_SummaryStats(rast, band) As stats
    FROM dummy_rast CROSS JOIN generate_series(1, 3) As band
     WHERE rid=2) As foo;
```

Output

```
rid | band | count | sum  |    mean    |  stddev   | min | max
----+------+-------+------+------------+-----------+-----+-----
  2 |    1 |    23 | 5821 | 253.086957 |  1.248061 | 250 | 254
  2 |    2 |    25 | 3682 |     147.28 | 59.862188 |  78 | 254
  2 |    3 |    25 | 3290 |      131.6 | 61.647384 |  62 | 254
```

Summarize pixels that intersect buildings of interest.

This example selects the target buildings, clips band 2 of the raster tiles to each building boundary, computes statistics for each clipped region, and then summarizes those statistics by building. It took 574ms on PostGIS Windows 64-bit with all of Boston Buildings and aerial Tiles (tiles each 150x150 pixels ~ 134,000 tiles), ~102,000 building records.

Code

```
WITH
-- our features of interest
feat AS (
    SELECT gid AS building_id, geom_26986 AS geom
    FROM buildings AS b
    WHERE gid IN (100, 103, 150)
),
-- clip band 2 of raster tiles to boundaries of builds
-- then get stats for these clipped regions
b_stats AS (
    SELECT building_id, (stats).*
    FROM (
        SELECT
            building_id,
            ST_SummaryStats(ST_Clip(rast, 2, geom)) AS stats
        FROM 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;
```

Output

```
building_id | num_pixels | min_pval | max_pval |     avg_pval
------------+------------+----------+----------+------------------
        100 |       1090 |        1 |      255 | 61.0697247706422
        103 |        655 |        7 |      182 | 70.5038167938931
        150 |        895 |        2 |      252 | 185.642458100559
```

Raster coverage.

This example returns statistics for each band.

Code

```
SELECT band, (stats).*
FROM (SELECT band, ST_SummaryStats('o_4_boston', 'rast', band) As stats
    FROM generate_series(1, 3) As band) As foo;
```

Output

```
band |  count  |  sum   |       mean       |      stddev      | min | max
------+---------+--------+------------------+------------------+-----+-----
    1 | 8450000 | 725799 | 82.7064349112426 | 45.6800222638537 |   0 | 255
    2 | 8450000 | 700487 | 81.4197705325444 | 44.2161184161765 |   0 | 255
    3 | 8450000 | 575943 |  74.682739408284 | 44.2143885481407 |   0 | 255
```

For a table, sampling less than 100% can improve speed. Here the sampling is set to 25%.

Code

```
SELECT band, (stats).*
FROM (SELECT band, ST_SummaryStats('o_4_boston', 'rast', band, true, 0.25) As stats
    FROM generate_series(1, 3) As band) As foo;
```

Output

```
band |  count  |  sum   |       mean       |      stddev      | min | max
------+---------+--------+------------------+------------------+-----+-----
    1 | 2112500 | 180686 | 82.6890480473373 | 45.6961043857248 |   0 | 255
    2 | 2112500 | 174571 |  81.448503668639 | 44.2252623171821 |   0 | 255
    3 | 2112500 | 144364 | 74.6765884023669 | 44.2014869384578 |   0 | 255
```

Siehe auch

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 if 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

Gibt summarystats aus, bestehend aus der Anzahl, der Summe, dem arithmetischen Mittel, der Standardabweichung, dem Minimum und dem Maximum der Werte eines Rasterbandes oder eines Rastercoverage. Wenn kein Band angegeben ist, wird Band 1 angenommen.



Anmerkung
Standardmäßig werden nur jene Pixelwerte berücksichtigt, die nicht `NODATA` sind. Setzen Sie bitte `exclude_nodata_value` auf FALSE um die Anzahl sämtlicher Pixel zu erhalten.



Anmerkung
Standardmäßig werden alle Pixel abgetastet. Um eine schnellere Rückmeldung zu bekommen, können Sie den Parameter `sample_percent` auf einen Wert zwischen 0 und 1 setzen.

Verfügbarkeit: 2.2.0

Beispiele

Code

```
WITH foo AS (
    SELECT
        rast.rast
    FROM (
        SELECT ST_SetValue(ST_SetValue(ST_SetValue(ST_AddBand(ST_MakeEmptyRaster(10, 10, 10, ↩
     10, 2, 2, 0, 0, 0),
                        1, '64BF', 0, 0
                    ),
                    1, 1, 1, -10
                ),
                1, 5, 4, 0
            ),
            1, 5, 5, 3.14159
        ) AS rast
    ) AS rast
    FULL JOIN (
        SELECT generate_series(1, 10) AS id
    ) AS id
        ON 1 = 1
)
SELECT
    (stats).count AS count,
    round((stats).sum::numeric, 3) AS sum,
    round((stats).mean::numeric, 3) AS mean,
    round((stats).stddev::numeric, 3) AS stddev,
    round((stats).min::numeric, 3) AS min,
    round((stats).max::numeric, 3) AS max
FROM (
    SELECT
        ST_SummaryStatsAgg(rast, 1, TRUE, 1) AS stats
    FROM foo
) bar;
```

Output

```
count |   sum    |  mean  | stddev |   min   |  max
-------+----------+--------+--------+---------+-------
    20 | -68.584 | -3.429 |  6.571 | -10.000 | 3.142
(1 row)
```

Siehe auch

summarystats, ST_SummaryStats, ST_Count, ST_Clip

11.9.7 ST_ValueCount

ST_ValueCount — Gibt Datensätze aus, die den Zellwert und die Anzahl der Pixel eines Rasterbandes (oder Rastercoverageban-des) für gegebene Werte enthalten. Wenn kein Band angegeben ist, wird Band 1 angenommen. Pixel mit dem Wert NODATA werden standardmäßig nicht gezählt; alle anderen Pixelwerte des Bandes werden ausgegeben und auf die nächste Ganzzahl gerundet.

Synopsis

SETOF record **ST_ValueCount**(raster rast, integer nband=1, boolean exclude_nodata_value=true, double precision[] searchvalues=NULL, double precision roundto=0, double precision OUT value, integer OUT count);
SETOF record **ST_ValueCount**(raster rast, integer nband, double precision[] searchvalues, double precision roundto=0, double precision OUT value, integer OUT count);
SETOF record **ST_ValueCount**(raster rast, double precision[] searchvalues, double precision roundto=0, double precision OUT value, integer OUT count);
bigint **ST_ValueCount**(raster rast, double precision searchvalue, double precision roundto=0);
bigint **ST_ValueCount**(raster rast, integer nband, boolean exclude_nodata_value, double precision searchvalue, double precision roundto=0);
bigint **ST_ValueCount**(raster rast, integer nband, double precision searchvalue, double precision roundto=0);
SETOF record **ST_ValueCount**(text rastertable, text rastercolumn, integer nband=1, boolean exclude_nodata_value=true, double precision[] searchvalues=NULL, double precision roundto=0, double precision OUT value, integer OUT count);
SETOF record **ST_ValueCount**(text rastertable, text rastercolumn, double precision[] searchvalues, double precision roundto=0, double precision OUT value, integer OUT count);
SETOF record **ST_ValueCount**(text rastertable, text rastercolumn, integer nband, double precision[] searchvalues, double precision roundto=0, double precision OUT value, integer OUT count);
bigint**ST_ValueCount**(text rastertable, text rastercolumn, integer nband, boolean exclude_nodata_value, double precision searchvalue, double precision roundto=0);
bigint **ST_ValueCount**(text rastertable, text rastercolumn, double precision searchvalue, double precision roundto=0);
bigint **ST_ValueCount**(text rastertable, text rastercolumn, integer nband, double precision searchvalue, double precision roundto=0);

Description

Gibt Datensätze mit den Spalten `value` und `count` aus, welche die Pixelwerte und die Anzahl der Pixel im angegebenen Band der Rasterkachel oder des Rastercoverage enthalten.

Wenn kein Band angegeben ist, wird `nband` standardmäßig auf 1 gesetzt. Wenn keine `searchvalues` angegeben sind, werden alle Pixelwerte des Rasters oder des Rastercoverage ausgeben. Wenn nur ein Suchwert angegeben ist, wird eine Ganzzahl ausgegeben - anstelle von Datensätzen mit der Pixelanzahl eines jeden Pixelwertes für das Bandes.



Anmerkung
Wenn `exclude_nodata_value` auf FALSE gesetzt ist, werden auch die Pixel mit NODATA Werten gezählt.

Verfügbarkeit: 2.0.0

Beispiele

Example will count only pixels of band 1 that are not 249.

Code

```
UPDATE dummy_rast SET rast = ST_SetBandNoDataValue(rast, 249) WHERE rid=2;

SELECT (pvc).*
FROM (SELECT ST_ValueCount(rast) As pvc
    FROM dummy_rast WHERE rid=2) As foo
    ORDER BY (pvc).value;
```

Output

```
value | count
-------+-------
   250 |     2
   251 |     1
   252 |     2
   253 |     6
   254 |    12
```

Example will coount all pixels of band 1 including 249.

Code

```
SELECT (pvc).*
FROM (SELECT ST_ValueCount(rast, 1, false) As pvc
    FROM dummy_rast WHERE rid=2) As foo
    ORDER BY (pvc).value;
```

Output

```
value | count
-------+-------
   249 |     2
   250 |     2
   251 |     1
   252 |     2
   253 |     6
   254 |    12
```

Example will count only non-nodata value pixels of band 2.

Code

```
SELECT (pvc).*
FROM (SELECT ST_ValueCount(rast, 2) As pvc
    FROM dummy_rast WHERE rid=2) As foo
    ORDER BY (pvc).value;
```

Output

```
value | count
-------+-------
    78 |     1
    79 |     1
    88 |     1
    89 |     1
    96 |     1
    97 |     1
    98 |     1
    99 |     2
   112 |     2
:
```

Real live example. Count all the pixels in an aerial raster tile band 2 intersecting a geometry. And return only the pixel band values that have a count > 500.

Code

```
SELECT (pvc).value, SUM((pvc).count) As total
FROM (SELECT ST_ValueCount(rast, 2) As pvc
    FROM o_4_boston
        WHERE ST_Intersects(
            rast,
            ST_GeomFromText('POLYGON((224486 892151,224486 892200,224706 892200,224706 ↩
    892151,224486 892151))', 26986)
             )
        ) As foo
    GROUP BY (pvc).value
    HAVING SUM((pvc).count) > 500
    ORDER BY (pvc).value;
```

Output

```
value | total
-------+-----
    51 | 502
    54 | 521
```

Just return count of pixels in each raster tile that have value of 100 of tiles that intersect a specific geometry.

Code

```
SELECT rid, ST_ValueCount(rast, 2, 100) As count
    FROM o_4_boston
        WHERE ST_Intersects(
            rast,
            ST_GeomFromText('POLYGON((224486 892151,224486 892200,224706 892200,224706 ↩
    892151,224486 892151))', 26986)
             ) ;
```

Output

```
rid | count
-----+-------
   1 |    56
   2 |    95
  14 |    37
  15 |    64
```

Siehe auch

ST_Count, ST_SetBandNoDataValue

11.10 Rastereingabe

11.10.1 ST_RastFromWKB

ST_RastFromWKB — Gibt einen Rasterwert von einer Well-known-Binary (WKB) Darstellung eines Rasters zurück.

Synopsis

raster **ST_RastFromWKB**(bytea wkb);

Description

Gibt einen Raster von einem Raster in der Well-known-Binary (WKB) Darstellung zurück.

Verfügbarkeit: 2.5.0

Beispiele

Code

```
SELECT (ST_MetaData(ST_RastFromWKB('\001\000\000\000\000\000\000\000\000\000\000\000@ ↩
    \000\000\000\000\000\000\010@ ↩
    \000\000\000\000\000\000\340?\000\000\000\000\000\000\340?\000\000\000\000\000\000\000\000\00
    '::bytea
    )
)).* AS metadata;
```

Output

```
upperleftx | upperlefty | width | height | scalex | scaley | skewx | skewy | srid | numbands
-----------+------------+-------+--------+--------+--------+-------+-------+------+----------
        0.5 |        0.5 |    10 |     20 |      2 |      3 |     0 |     0 |   10 |        0
```

Siehe auch

ST_MetaData, ST_RastFromHexWKB, ST_AsBinary/ST_AsWKB, ST_AsHexWKB

11.10.2 ST_RastFromHexWKB

ST_RastFromHexWKB — Gibt einen Rasterwert von einer Well-known-Binary (WKB) Hex-Darstellung eines Rasters zurück.

Synopsis

raster **ST_RastFromHexWKB**(text wkb);

Description

Gibt einen Raster von einem Raster in der Well-known-Binary (WKB) Hex-Darstellung zurück.

Verfügbarkeit: 2.5.0

Beispiele

Code

```
SELECT (ST_MetaData(ST_RastFromHexWKB(' ↩
    01000000000000000000000040000000000000084000000000000000 ↩
    E03F000000000000E03F0000000000000000000000000000000A0000000A001400'
    )
)).* AS metadata;
```

Output

```
upperleftx | upperlefty | width | height | scalex | scaley | skewx | skewy | srid | numbands
-----------+------------+-------+--------+--------+--------+-------+-------+------+----------
        0.5 |        0.5 |    10 |     20 |      2 |      3 |     0 |     0 |   10 |        0
```

Siehe auch

ST_MetaData, ST_RastFromWKB, ST_AsBinary/ST_AsWKB, ST_AsHexWKB

11.11 Ausgabe von Rastern

11.11.1 ST_AsBinary/ST_AsWKB

ST_AsBinary/ST_AsWKB — Gibt die Well-known-Binary (WKB) Darstellung eines Rasters zurück.

Synopsis

bytea **ST_AsBinary**(raster rast, boolean outasin=FALSE);
bytea **ST_AsWKB**(raster rast, boolean outasin=FALSE);

Description

Gibt den Raster in binärer Darstellung zurück. Wenn `outasin` TRUE ist, werden Bänder die außerhalb der Datenbank liegen (out-db) gleich behandelt wie Bänder, die direkt in der Datenbank gespeichert (in-db) sind. Siehe "raster/doc/RFC2-WellKnownBinaryFo im Quellverzeichnis von PostGIS für Details zu dieser Darstellung.

Dies ist nützlich, um mit einem "binary" Cursor die Daten direkt als Binärdatei von der Datenbank abzurufen - ohne Konvertierung in eine Textdarstellung.



Anmerkung
Standardmäßig ist in der WKB-Ausgabe der Pfad der out-db Bänder enthalten. Wenn ein Client keinen Zugriff auf die Rasterdatei eines "out-db" Bandes hat, können Sie `outasin` auf TRUE setzen.

Erweiterung: 2.1.0 `outasin` hinzugefügt

Erweiterung: 2.5.0 `ST_AsWKB` hinzugefügt

Beispiele

Code

```
SELECT ST_AsBinary(rast) AS rastbin FROM dummy_rast WHERE rid=1;
```

Output

```
\001\000\000\000\000\000\000\000\000\000\000\000@\000\000\000\000\000\000\010@ ↩
     \000\000\000\000\000\000\340?\000\000\000\000\000\000\340?\000\000\000\000\000\000\000\000\000\000\000\000\000\000\000\0
```

Siehe auch

ST_RastFromWKB, ST_AsHexWKB

11.11.2 ST_AsHexWKB

ST_AsHexWKB — Gibt die Well-known-Binary (WKB) Hex-Darstellung eines Rasters zurück.

Synopsis

bytea **ST_AsHexWKB**(raster rast, boolean outasin=FALSE);

Description

Gibt den Raster in binärer Hex-Darstellung zurück. Wenn `outasin` TRUE ist, werden Bänder die außerhalb der Datenbank liegen (out-db) gleich behandelt wie Bänder, die direkt in der Datenbank gespeichert (in-db) sind. Siehe "raster/doc/RFC2-WellKnownBinaryFormat" im Quellverzeichnis von PostGIS für Details zu dieser Darstellung.



Anmerkung
Standardmäßig ist in der Hex-WKB-Ausgabe der Pfad der out-db Bänder enthalten. Wenn ein Client keinen Zugriff auf die Rasterdatei eines "out-db" Bandes hat, können Sie `outasin` auf TRUE setzen.

Verfügbarkeit: 2.5.0

Beispiele

Code

```
SELECT ST_AsHexWKB(rast) AS rastbin FROM dummy_rast WHERE rid=1;
```

Output

```
01000000000000000000000040000000000000084000000000000000 ↩
    E03F000000000000E03F0000000000000000000000000000000000A0000000A001400
```

Siehe auch

ST_RastFromHexWKB, ST_AsBinary/ST_AsWKB

11.11.3 ST_AsGDALRaster

ST_AsGDALRaster — Gibt die Rasterkachel in dem ausgewiesenen Rasterformat von GDAL aus. Sie können jedes Rasterformat angeben, das von Ihrer Bibliothek unterstützt wird. Um eine Liste mit den unterstützten Formaten auszugeben, verwenden Sie bitte ST_GDALDrivers().

Synopsis

bytea **ST_AsGDALRaster**(raster rast, text format, text[] options=NULL, integer srid=sameassource);

Description

Gibt die Rasterkachel im dem ausgewiesenen Format zurück. Die Übergabewerte sind:

- `format` das Format das abgerufen wird. Dies ist abhängig von den Treibern die mit Ihrer Bibliothek "libgdal" kompiliert wurden. Üblicherweise stehen 'JPEG', 'GTIff' und 'PNG' zur Verfügung. Verwenden Sie bitte ST_GDALDrivers, um eine Liste der von Ihrer Bibliothek unterstützten Formate zu erhalten.
- `options` ein Textfeld mit Optionen für GDAL. Gültige Optionen sind formatabhängig. Siehe GDAL Raster format options für weitere Details.
- `srid` the spatial reference identifier to embed in the output raster

Verfügbarkeit: 2.0.0 - GDAL >= 1.6.0.

Beispiele

Export multiple raster tiles as a single JPEG image with 50 percent quality.

Code

```
SELECT ST_AsGDALRaster(ST_Union(rast), 'JPEG', ARRAY['QUALITY=50']) As rastjpg
FROM dummy_rast
WHERE rast && ST_MakeEnvelope(10, 10, 11, 11);
```

One way to export raster into another format is using PostgreSQL large object export functions. The following repeats the JPEG example and exports the result. Note that this requires superuser access because it uses server-side large object functions. It will also export to path on server network. If you need export locally, use the psql equivalent lo_ functions which export to the local file system instead of the server file system.

Code

```
DROP TABLE IF EXISTS tmp_out ;

CREATE TABLE tmp_out AS
SELECT lo_from_bytea(
       0,
       ST_AsGDALRaster(ST_Union(rast), 'JPEG', ARRAY['QUALITY=50'])
        ) AS loid
  FROM dummy_rast
WHERE rast && ST_MakeEnvelope(10, 10, 11, 11);

SELECT lo_export(loid, '/tmp/dummy.jpg')
   FROM tmp_out;

SELECT lo_unlink(loid)
  FROM tmp_out;
```

Export a raster as a GeoTIFF using the default creation options.

Code

```
SELECT ST_AsGDALRaster(rast, 'GTiff') AS rasttiff
FROM dummy_rast WHERE rid = 2;
```

Export a GeoTIFF using JPEG compression at 90 percent quality and embed SRID 4269.

Code

```
SELECT ST_AsGDALRaster(
  rast,
  'GTiff',
  ARRAY['COMPRESS=JPEG', 'JPEG_QUALITY=90'],
  4269) AS rasttiff
FROM dummy_rast WHERE rid = 2;
```

Siehe auch

Abschnitt 10.4, ST_GDALDrivers, ST_SRID

11.11.4 ST_AsJPEG

ST_AsJPEG — Gibt die ausgewählten Bänder der Rasterkachel als einzelnes Bild (Byte-Array) im Format "Joint Photographic Exports Group" (JPEG) aus. Wenn kein Band angegeben ist und 1 oder mehr als 3 Bänder ausgewählt wurden, dann wird nur das erste Band verwendet. Wenn 3 Bänder ausgewählt wurden, werden alle 3 Bänder verwendet und auf RGB abgebildet.

Synopsis

bytea **ST_AsJPEG**(raster rast, text[] options=NULL);
bytea **ST_AsJPEG**(raster rast, integer nband, integer quality);
bytea **ST_AsJPEG**(raster rast, integer nband, text[] options=NULL);
bytea **ST_AsJPEG**(raster rast, integer[] nbands, text[] options=NULL);
bytea **ST_AsJPEG**(raster rast, integer[] nbands, integer quality);

Description

Gibt die ausgewählten Bänder der Rasterkachel als einzelnes Bild im Format "Joint Photographic Exports Group" (JPEG) aus. Sie können ST_AsGDALRaster verwenden, wenn Sie in weniger gebräuchliche Rasterformate exportieren wollen. Wenn kein Band angegeben ist und 1 oder mehr als 3 Bänder ausgewählt wurden, dann wird nur das erste Band verwendet. Wenn 3 Bänder ausgewählt wurden, werden alle 3 Bänder verwendet. Die Funktion hat viele Varianten mit vielen Optionen. Diese sind unterhalb angeführt:

- `nband` für den Export einzelner Bänder.
- `nbands` Ein Feld mit den Bändern die exportiert werden sollen (bei JPEG maximal 3). Die Reihenfolge der Bänder ist RBG; z.B. ARRAY[3,2,1] bedutet, dass Band 3 auf Rot, Band 2 auf Grün und Band 1 auf Blau abgebildet wird.
- `quality` Eine Zahl zwischen 0 und 100. Umso höher die Zahl ist, umso schärfer ist das Bild.
- `options` Ein Textfeld mit GDAL Optionen für JPEG (siehe "create_options" für JPEG unter ST_GDALDrivers). Gültige Optionen für JPEG sind `PROGRESSIVE` ON oder OFF und `QUALITY` zwischen 0 und 100 mit dem Standardwert 75. Siehe GDAL Raster format options für weitere Details.

Verfügbarkeit: 2.0.0 - GDAL >= 1.6.0.

Beispiele: Ausgabe

Output the first three bands at 75 percent quality.

Code

```
SELECT ST_AsJPEG(rast) As rastjpg
FROM dummy_rast WHERE rid=2;
```

Output only the first band at 90 percent quality.

Code

```
SELECT ST_AsJPEG(rast, 1, 90) As rastjpg
FROM dummy_rast WHERE rid=2;
```

Output the first three bands, mapping band 2 to red, band 1 to green, and band 3 to blue, using progressive encoding at 90 percent quality.

Code

```
SELECT ST_AsJPEG(rast, ARRAY[2, 1, 3], ARRAY['QUALITY=90', 'PROGRESSIVE=ON']) As rastjpg
FROM dummy_rast WHERE rid=2;
```

Siehe auch

Abschnitt 10.4, ST_GDALDrivers, ST_AsGDALRaster, ST_AsPNG, ST_AsTIFF

11.11.5 ST_AsPNG

ST_AsPNG — Gibt die ausgewählten Bänder der Rasterkachel als einzelnes, übertragbares Netzwerkgraphik (PNG) Bild (Byte-Feld) aus. Wenn der Raster 1,3 oder 4 Bänder hat und keine Bänder angegeben sind, dann werden alle Bänder verwendet. Wenn der Raster 2 oder mehr als 4 Bänder hat und keine Bänder angegeben sind, dann wird nur Band 1 verwendet. Die Bänder werden in den RGB- oder den RGBA-Raum abgebildet.

Synopsis

bytea **ST_AsPNG**(raster rast, text[] options=NULL);
bytea **ST_AsPNG**(raster rast, integer nband, integer compression);
bytea **ST_AsPNG**(raster rast, integer nband, text[] options=NULL);
bytea **ST_AsPNG**(raster rast, integer[] nbands, integer compression);
bytea **ST_AsPNG**(raster rast, integer[] nbands, text[] options=NULL);

Description

Gibt die ausgewählten Bänder des Raster als einzelnes Bild im Format "Portable Network Graphics Image" (PNG) aus. Sie können ST_AsGDALRaster verwenden, wenn Sie in weniger gebräuchliche Rasterformate exportieren wollen. Wenn kein Band angegeben ist, werden die ersten 3 Bänder exportiert. Wenn `SRID` nicht angegeben ist, wird die SRID des Ausgangsraster verwendet. Die Funktion hat viele Varianten mit vielen Optionen. Diese sind unterhalb angeführt:

- `nband` für den Export einzelner Bänder.
- `nbands` Ein Feld mit den Bändern die exportiert werden sollen (bei PNG maximal 4). Die Reihenfolge der Bänder ist RBGA; z.B. ARRAY[3,2,1] bedutet, dass Band 3 auf Rot, Band 2 auf Grün und Band 1 auf Blau abgebildet wird.
- `compression` Eine Zahl zwischen 1 und 9. Umso höher die Zahl umso größer die Komprimierung.
- `options` Ein Textfeld mit GDAL Optionen für PNG (siehe "create_options" für PNG unter ST_GDALDrivers). Für PNG gibt es nur eine gültige Option "ZLEVEL" (wieviel Zeit für die Komprimierung aufgewendet werden soll - die Standardeinstellung ist 6); z.B.: ARRAY['ZLEVEL=9']. Ein WORLDFILE ist nicht erlaubt, da die Funktion sonst zwei Ausgaben machen müsste. Siehe GDAL Raster format options für weitere Details.

Verfügbarkeit: 2.0.0 - GDAL >= 1.6.0.

Beispiele

Export the first 3 bands and map band 3 to red, band 1 to green, and band 2 to blue.

Code

```
SELECT ST_AsPNG(rast) As rastpng
FROM dummy_rast WHERE rid=2;
SELECT ST_AsPNG(rast, ARRAY[3, 1, 2]) As rastpng
FROM dummy_rast WHERE rid=2;
```

Siehe auch

ST_AsGDALRaster, ST_ColorMap, ST_GDALDrivers, Abschnitt 10.4

11.11.6 ST_AsTIFF

ST_AsTIFF — Gibt die ausgewählten Bänder des Raster als einzelnes TIFF Bild (Byte-Feld) zurück. Wenn kein Band angegeben ist oder keines der angegebenen Bänder im Raster existiert, werden alle Bänder verwendet.

Synopsis

bytea **ST_AsTIFF**(raster rast, text[] options='', integer srid=sameassource);
bytea **ST_AsTIFF**(raster rast, text compression='', integer srid=sameassource);
bytea **ST_AsTIFF**(raster rast, integer[] nbands, text compression='', integer srid=sameassource);
bytea **ST_AsTIFF**(raster rast, integer[] nbands, text[] options, integer srid=sameassource);

Description

Gibt die ausgewählten Bänder des Raster als einzelnes Bild im Format " Tagged Image File Format" (TIFF) aus. Wenn kein Band angegeben ist, wird versucht alle Bänder zu verwenden. Diese Funktion ist ein Adapter für ST_AsGDALRaster. Verwenden Sie bitte ST_AsGDALRaster, wenn Sie in weniger gebräuchliche Rasterformate exportieren wollen. Wenn kein SRS-Text für die Georeferenz vorhanden ist, wird das Koordinatenreferenzsystem des Ausgangsraster verwendet. Die Funktion hat viele Varianten mit vielen Optionen. Diese sind unterhalb angeführt:

- `nbands` Ein Feld mit den Bändern die exportiert werden sollen (bei PNG maximal 3). Die Reihenfolge der Bänder ist RBG; z.B. ARRAY[3,2,1] bedutet, dass Band 3 auf Rot, Band 2 auf Grün und Band 1 auf Blau abgebildet wird.
- `compression` Ein Ausdruck für die Art der Datenkompression -- JPEG90 (oder eine andere Prozentangabe), LZW, JPEG, DEFLATE9.
- `options` Ein Textfeld mit GDAL Optionen für die Erstellung eines GTiff (siehe "create_options" für GTiff unter ST_GDALDrivers oder GDAL Raster format options für weitere Details.
- `srid` Die SRID des Koordinatenreferenzsystems des Raster. Wird als Information zur Georeferenzierung verwendet.

Verfügbarkeit: 2.0.0 - GDAL >= 1.6.0.

Beispiele: 90%ige JPEG Komprimierung

Code

```
SELECT ST_AsTIFF(rast, 'JPEG90') As rasttiff
FROM dummy_rast WHERE rid=2;
```

Siehe auch

ST_GDALDrivers, ST_AsGDALRaster, ST_SRID

11.12 Rasterverarbeitung: Kartenalgebra

11.12.1 ST_Clip

ST_Clip — Gibt den nach der Eingabegeometrie zugeschnittenen Raster zurück. Falls die Bandnummer nicht angegeben ist, werden alle Bänder verarbeitet. Wenn `crop` nicht angegeben oder auf TRUE gesetzt ist, wird der Ausgaberaster entsprechend zugeschnitten. Wenn `touched` auf TRUE gesetzt ist, werden auch berührte Pixel einbezogen, andernfalls nur Pixel, bei denen sich der Mittelpunkt im Inneren der Geometrie befindet.

Synopsis

raster **ST_Clip**(raster rast, integer[] nband, geometry geom, double precision[] nodataval=NULL, boolean crop=TRUE, boolean touched=FALSE);
raster **ST_Clip**(raster rast, integer nband, geometry geom, double precision nodataval, boolean crop=TRUE, boolean touched=FALSE);
raster **ST_Clip**(raster rast, integer nband, geometry geom, boolean crop, boolean touched=FALSE);
raster **ST_Clip**(raster rast, geometry geom, double precision[] nodataval=NULL, boolean crop=TRUE, boolean touched=FALSE);
raster **ST_Clip**(raster rast, geometry geom, double precision nodataval, boolean crop=TRUE, boolean touched=FALSE);
raster **ST_Clip**(raster rast, geometry geom, boolean crop, boolean touched=FALSE);

Description

Gibt einen Raster aus, der nach der Eingabegeometrie `geom` ausgeschnitten wird. Wird kein Band angegeben, so werden alle Bänder bearbeitet.

Raster die aus einer Operation mit ST_Clip resultieren, müssen in den Bereichen wo sie ausgeschnitten werden, einen NODATA-Wert für jedes Band aufweisen. Wenn keine Werte übergeben werden und für den Ausgangsraster kein NODATA-Wert festgelegt wurde, dann werden die NODATA-Werte des Zielrasters auf ST_MinPossibleValue(ST_BandPixelType(rast, band)) gesetzt. Wenn die Anzahl der NODATA-Werte in dem übergebenen Feld kleiner als die Anzahl der Bänder ist, wird für die restlichen Bänder der letzte Wert des Feldes verwendet. Wenn die Anzahl der NODATA-Werte größer als die Anzahl der Bänder ist, so werden die zusätzlichen NODATA-Werte übergangen. Alle Varianten, die ein Feld mit NODATA-Werten akzeptieren, nehmen auch einen einzelnen Wert für alle Bänder entgegen.

Wenn `crop` nicht angegeben ist, wird true angenommen, was bedeutet, dass das Ausgaberaster auf den Schnittpunkt der Ausdehnungen `geom`und `rast` beschnitten wird. Wenn `crop` auf false gesetzt ist, erhält das neue Raster die gleiche Ausdehnung wie `rast`. Wenn `touched` auf true gesetzt ist, werden alle Pixel im `rast`, die die Geometrie schneiden, ausgewählt.



Anmerkung
Das Standardverhalten ist touched=false, wodurch nur Pixel ausgewählt werden, deren Mittelpunkt von der Geometrie abgedeckt ist.

Verbessert: 3.5.0 - berührtes Argument hinzugefügt.

Verfügbarkeit: 2.0.0

Erweiterung: 2.1.0 neu geschrieben in C

Die Beispiele verwenden Luftbilddaten aus Massachusetts, die auf der MassGIS-Website MassGIS Aerial Orthos verfügbar sind.

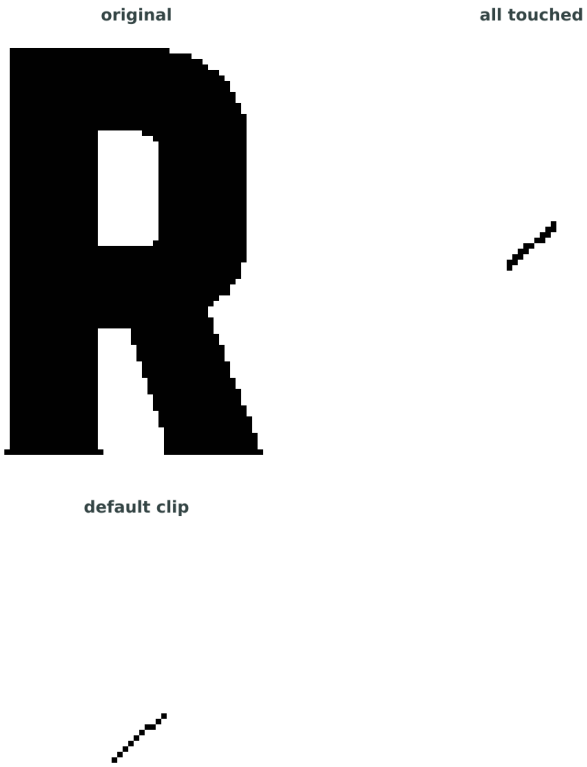
Beispiele

Comparing selecting all touched vs. not all touched.

Code

```
WITH source AS (
    SELECT rast, geom
    FROM ST_AsRaster(ST_Letters('R'), scalex =
> 1.0, scaley =
> -1.0) AS r(rast)
    INNER JOIN ST_GeomFromText('LINESTRING(0 1,5 6,10 10)') AS g(geom)
        ON ST_Intersects(r.rast, g.geom)
), clipped AS (
    SELECT rast,
        ST_Clip(rast, geom, touched =
> true) AS rast_touched,
        ST_Clip(rast, geom, touched =
> false) AS rast_not_touched
    FROM source
), variants AS (
    SELECT 'original' AS title, ST_Count(rast) AS pixel_count, rast AS rendered
    FROM clipped
    UNION ALL
    SELECT 'all touched', ST_Count(rast_touched), rast_touched
    FROM clipped
    UNION ALL
    SELECT 'default clip', ST_Count(rast_not_touched), rast_not_touched
    FROM clipped
)
SELECT title, pixel_count, ST_AsPNG(rendered) AS image
FROM variants
ORDER BY CASE title
    WHEN 'original' THEN 1
    WHEN 'all touched' THEN 2
    ELSE 3
END;
```

Figure



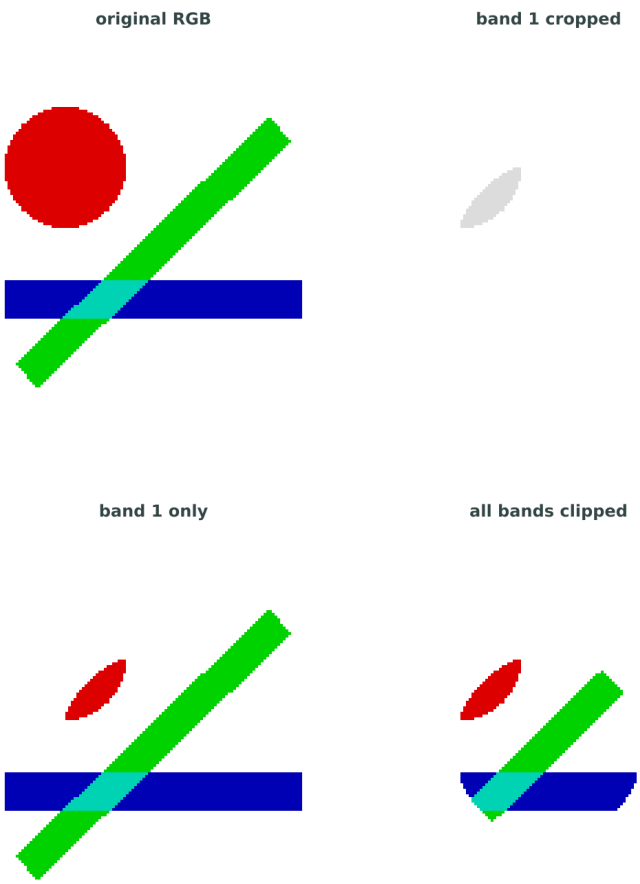
The following self-contained example uses three simple shapes as RGB bands. It compares cropping band 1, clipping band 1 without cropping and then restoring bands 2 and 3, and clipping every band without cropping.

Code

```
WITH canvas AS (
    SELECT ST_AddBand(
        ST_AddBand(
            ST_AddBand(
                ST_MakeEmptyRaster(128, 128, 0, 128, 1, -1, 0, 0, 0),
                '8BUI'::text, 0::double precision, 0::double precision
            ),
            '8BUI'::text, 0::double precision, 0::double precision
        ),
        '8BUI'::text, 0::double precision, 0::double precision
    ) AS rast
), source AS (
    SELECT ST_SetValue(
        ST_SetValue(
            ST_SetValue(rast, 1, ST_Buffer('POINT(32 96)'::geometry, 22), 220),
            2, ST_Buffer('LINESTRING(18 20,110 110)'::geometry, 6,
                         'endcap=flat join=bevel'), 210
        ),
        3, ST_Buffer('LINESTRING(10 48,118 48)'::geometry, 7,
```

```
                    'endcap=flat'), 180
    ) AS rast,
    ST_Buffer('POINT(64 64)'::geometry, 34) AS clipper
    FROM canvas
), variants AS (
    SELECT 'original RGB' AS title, rast AS rendered FROM source
    UNION ALL
    SELECT 'band 1 cropped', ST_Clip(rast, 1, clipper, true) FROM source
    UNION ALL
    SELECT
        'band 1 only',
        ST_AddBand(
            ST_Clip(rast, 1, clipper, false),
            ARRAY[ST_Band(rast, 2), ST_Band(rast, 3)]
        )
    FROM source
    UNION ALL
    SELECT 'all bands clipped', ST_Clip(rast, clipper, false) FROM source
)
SELECT title, ST_AsPNG(rendered) AS image
FROM variants
ORDER BY CASE title
    WHEN 'original RGB' THEN 1
    WHEN 'band 1 cropped' THEN 2
    WHEN 'band 1 only' THEN 3
    ELSE 4
END;
```

Figure



Crop, single-band clipping, and all-band clipping.



Source tile



Cropped



One-band source



One-band clip



All-band source



All-band clip

Siehe auch

ST_AddBand, ST_Count, ST_MapAlgebra (callback function version), ST_Intersection

11.12.2 ST_ColorMap

ST_ColorMap — Erzeugt aus einem bestimmten Band des Ausgangsrasters einen neuen Raster mit bis zu vier 8BUI-Bändern (Grauwert, RGB, RGBA). Wenn kein Band angegeben ist, wird Band 1 angenommen.

Synopsis

raster **ST_ColorMap**(raster rast, integer nband=1, text colormap=grayscale, text method=INTERPOLATE);

raster **ST_ColorMap**(raster rast, text colormap, text method=INTERPOLATE);

Description

Wendet eine `colormap` auf das Band `nband` von `rast` an, wodurch ein neuer Raster aus bis zu vier 8BUI-Bändern erstellt wird. Die Anzahl der 8BUI-Bänder des neuen Raster wird durch die Anzahl der Farbkomponenten bestimmt, die in der `colormap` definiert sind.

Wenn `nband` nicht angegeben ist, wird Band 1 angenommen.

`colormap` kann ein Schlüsselwort, ein vordefiniertes Farbschema, oder Zeilen in denen der Zellwert und die Farbkomponenten festgelegt sind.

Gültige, vordefinierte Schlüsselwörter von `colormap`:

- `grayscale` oder `greyscale` für Grauabstufungen in einem 8BUI-Rasterband.
- `pseudocolor` für vier 8BUI-Rasterbänder (RGBA) mit Farbverläufen von Blau zu Grün zu Rot.
- `fire` für vier 8BUI-Rasterbänder (RGBA) mit Farbverläufen von Blau zu Rot zu blassem Gelb.
- `bluered` für vier 8BUI-Rasterbänder (RGBA) mit Farbverläufen von Blau zu blassem Weiß zu Rot.

Anwender können mehrere Einträge (einen pro Zeile) an `colormap` übergeben, um bestimmte Farbschemata zu spezifizieren. Üblicherweise besteht jeder Eintrag aus fünf Werten: der Pixelwert und die zugehörigen Rot-, Grün-, Blau- und Alpha-Komponenten (Farbkomponenten zwischen 0 und 255). Anstelle der Pixelwerte können auch Prozentwerte verwendet werden, wobei 0% und 100% die minimalen und maximalen Werte des Rasterbandes sind. Die Werte können durch Beistriche (','), Tabulatoren, Doppelpunkte (':') und/oder durch Leerzeichen getrennt werden. Für die NODATA-Werte kann der Pixelwert mit *nv*, *NULL* oder auf *NODATA* angegeben werden. Ein Beispiel finden Sie unterhalb.

Code

```
5 0 0 0 255
4 100:50 55 255
1 150, 100 150 255
0% 255 255 255 255
nv 0 0 0 0
```

Die Syntax von `colormap` ist ähnlich wie bei dem Modus "color-relief" bei gdaldem von GDAL.

Gültige Schlüsselwörter für `method`:

- `INTERPOLATE` verwendet eine lineare Interpolation um einen kontinuierlichen Übergang der Farben zwischen den Pixelwerten zu erhalten
- `EXACT` genaue Entsprechung der Pixelwerte mit der "colormap". Pixel, deren Werte keinen Eintrag in "colormap" haben, werden mit "0 0 0 0" (RGBA) eingefärbt
- `NEAREST` verwendet die Einträge der "colormap", deren Wert dem Pixelwert am nächsten kommt



Anmerkung
Eine großartige Hilfestellung für "colormaps" bietet ColorBrewer



Warnung
Bei den resultierenden Bänder des neuen Rasters sind keine NODATA-Werte gesetzt. Verwenden Sie bitte ST_SetBandNoDataValue um einen NODATA-Wert zu setzen, falls dieser benötigt wird.

Verfügbarkeit: 2.1.0

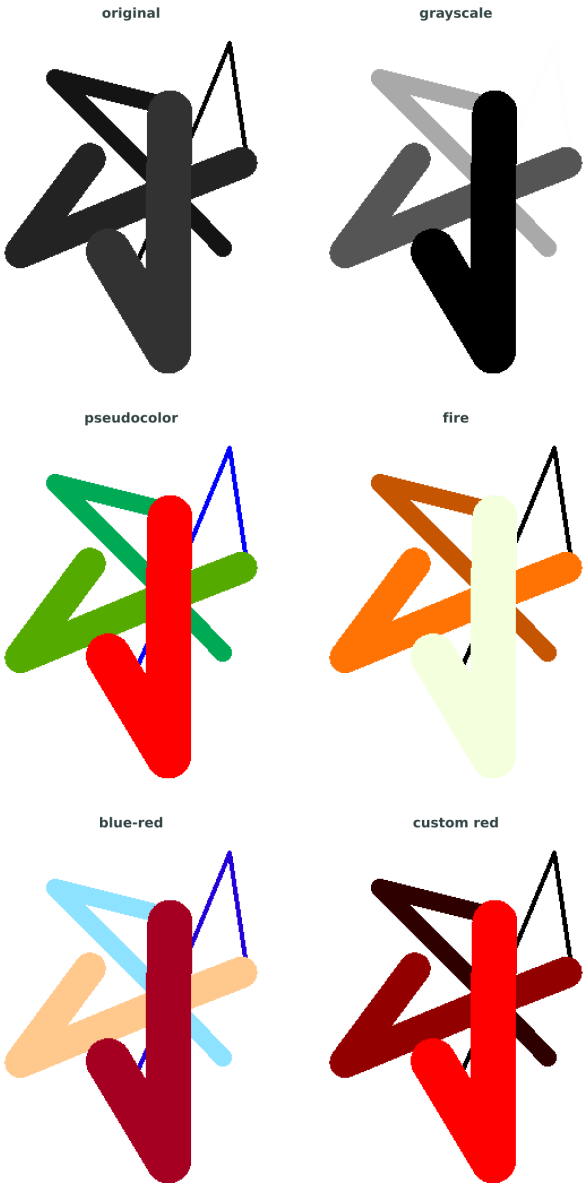
Beispiele

This example builds a raster from rotated buffered lines and compares the original values with predefined and custom color maps.

Code

```
WITH ref AS (
    SELECT ST_MakeEmptyRaster(200, 200, 0, 200, 1, -1, 0, 0) AS rast
), source AS (
    SELECT ST_Union(rast) AS rast
    FROM (
    SELECT
        ST_AsRaster(ST_Rotate(ST_Buffer('LINESTRING(0 2,50 50,150 150,125 50)'::geometry,
                    i*2
                ),
                pi() * i * 0.125, ST_Point(50, 50)
            ),
            ref.rast, '8BUI'::text, i * 5
        ) AS rast
    FROM ref
    CROSS JOIN generate_series(1, 10, 3) AS i
) AS shapes
)
SELECT title, ST_AsPNG(rendered) AS image
FROM source
CROSS JOIN LATERAL (VALUES
    ('original', rast),
    ('grayscale', ST_ColorMap(rast, 1, 'greyscale')),
    ('pseudocolor', ST_ColorMap(rast, 1, 'pseudocolor')),
    ('fire', ST_ColorMap(rast, 1, 'fire')),
    ('blue-red', ST_ColorMap(rast, 1, 'bluered')),
    ('custom red', ST_ColorMap(
        rast,
        1,
        E'100% 255 0 0 255\n80% 160 0 0 255\n50% 130 0 0 255\n' ||
        E'30% 30 0 0 255\n20% 60 0 0 255\n0% 0 0 0 255\n' ||
        E'nv 255 255 255 0'
    ))
) AS variants(title, rendered);
```

Figure



Siehe auch

ST_AsPNG, ST_AsRaster ST_MapAlgebra (callback function version), ST_Grayscale ST_NumBands, ST_Reclass, ST_SetBandNoDat ST_Union

11.12.3 ST_Grayscale

ST_Grayscale — Erzeugt einen neuen Raster mit einem 8BUI-Band aus dem Ausgangsrasters und den angegebenen Bändern für Rot, Grün und Blau

Synopsis

(1) raster **ST_Grayscale**(raster rast, integer redband=1, integer greenband=2, integer blueband=3, text extenttype=INTERSECTION);

(2) raster **ST_Grayscale**(rastbandarg[] rastbandargset, text extenttype=INTERSECTION);

Description

Erzeugt einen Raster mit einem 8BUI-Band aus drei Eingabebändern (von einem oder mehreren Raster). Bänder die nicht den Pixeltyp 8BUI haben werden mit ST_Reclass neu klassifiziert.



Anmerkung
Diese Funktion unterscheidet sich insofern von ST_ColorMap mit dem Schlüsselwort `grayscale`, da ST_ColorMap lediglich ein Band bearbeitet, während diese Funktion drei Bänder für RGB erwartet. Diese Funktion verwendet folgende Gleichung zur Konvertierung von RGB auf Graustufen: 0.2989 * RED + 0.5870 * GREEN + 0.1140 * BLUE

Verfügbarkeit: 2.5.0

Beispiele

This self-contained example paints an apple with gradients and highlights into three raster bands. It compares the original RGB raster with the equivalent grayscale results produced by both function variants.

Code

```
WITH pixels AS (
    SELECT
        col,
        row,
        x,
        y,
        (((x - 48.0)^2 / (25.0^2) + (y - 54.0)^2 / (29.0^2) < 1)
         OR ((x - 72.0)^2 / (25.0^2) + (y - 54.0)^2 / (29.0^2) < 1))
         AND NOT ((x - 60.0)^2 / (8.0^2) + (y - 26.0)^2 / (5.0^2) < 1) AS fruit,
        ((x - 82.0) * 0.84 + (y - 28.0) * 0.54)^2 / (22.0^2)
          + (-(x - 82.0) * 0.54 + (y - 28.0) * 0.84)^2 / (8.0^2) < 1 AS leaf,
        abs((x - 62.0) * 0.96 - (y - 29.0) * 0.28) < 3
          AND x BETWEEN 60 AND 69 AND y BETWEEN 21 AND 45 AS stem
    FROM (
        SELECT
            col,
            row,
            col / 2.0 AS x,
            row / 2.0 AS y
        FROM generate_series(1, 180) AS row
        CROSS JOIN generate_series(1, 240) AS col
    ) AS source_pixels
), shaded AS (
    SELECT
        col,
        row,
        x,
        y,
        CASE
            WHEN stem THEN 95 + y * 0.7 + (x % 3) * 8
            WHEN leaf THEN 28 + (90 - x) * 0.5
            WHEN fruit THEN LEAST(255, GREATEST(0,
                150
                + (85 - sqrt((x - 39.0)^2 + (y - 37.0)^2)) * 1.1
                + CASE
                    WHEN (x - 43.0)^2 / (10.0^2) + (y - 43.0)^2 / (7.0^2) < 1
                    THEN 35
```

```
                    ELSE 0
                END
                - CASE
                    WHEN (x - 77.0)^2 / (20.0^2) + (y - 67.0)^2 / (13.0^2) < 1
                    THEN 35
                    ELSE 0
                END
                + ((x * 13 + y * 7) % 19 - 9)
            ))
            ELSE 245
        END AS red,
        CASE
            WHEN stem THEN 50 + y * 0.35
            WHEN leaf THEN LEAST(190, GREATEST(40,
                105 + (95 - x) * 1.2 + (y - 20) * 1.5
                + CASE
                    WHEN (x - 76.0)^2 / (12.0^2) + (y - 24.0)^2 / (4.0^2) < 1
                    THEN 45
                    ELSE 0
                END
            ))
            WHEN fruit THEN LEAST(95, GREATEST(15,
                24 + (68 - y) * 0.35
                + CASE
                    WHEN (x - 43.0)^2 / (10.0^2) + (y - 43.0)^2 / (7.0^2) < 1
                    THEN 30
                    ELSE 0
                END
                + ((x * 5 + y * 11) % 13 - 6)
            ))
            ELSE 245
        END AS green,
        CASE
            WHEN stem THEN 22 + (x % 4) * 3
            WHEN leaf THEN 42 + (100 - x) * 0.4
            WHEN fruit THEN LEAST(75, GREATEST(10,
                18
                + CASE
                    WHEN (x - 43.0)^2 / (10.0^2) + (y - 43.0)^2 / (7.0^2) < 1
                    THEN 25
                    ELSE 0
                END
                + ((x * 3 + y * 5) % 11 - 5)
            ))
            ELSE 245
        END AS blue
    FROM pixels
), rows AS (
    SELECT
        row,
        array_agg(red::double precision ORDER BY col) AS red_values,
        array_agg(green::double precision ORDER BY col) AS green_values,
        array_agg(blue::double precision ORDER BY col) AS blue_values
    FROM shaded
    GROUP BY row
), source AS (
    SELECT ST_SetValues(ST_SetValues(ST_SetValues(
        ST_AddBand(ST_AddBand(ST_AddBand(
            ST_MakeEmptyRaster(240, 180, 0, 180, 1, -1, 0, 0, 0),
            '8BUI'::text, 245::double precision, 0::double precision),
            '8BUI'::text, 245::double precision, 0::double precision),
            '8BUI'::text, 245::double precision, 0::double precision),
```

```
        1, 1, 1, array_agg(red_values ORDER BY row)::double precision[][]),
        2, 1, 1, array_agg(green_values ORDER BY row)::double precision[][]),
        3, 1, 1, array_agg(blue_values ORDER BY row)::double precision[][]) AS rast
    FROM rows
)
SELECT ST_AsPNG(rast) AS "original RGB",
    ST_AsPNG(ST_Grayscale(rast)) AS "grayscale (variant 1)",
    ST_AsPNG(ST_Grayscale(ARRAY[
        ROW(rast, 1)::rastbandarg,
        ROW(rast, 2)::rastbandarg,
        ROW(rast, 3)::rastbandarg
    ]::rastbandarg[])) AS "grayscale (variant 2)"
FROM source;
```

Figure



Original photographic illustration:



original apple



grayscale apple

Siehe auch

ST_AsPNG, ST_Reclass, ST_ColorMap

11.12.4 ST_Intersection

ST_Intersection — Gibt Geometry-PixelValue Paare, oder einen Raster aus, der durch die Schnittmenge der beiden Raster bestimmt wird, oder durch die geometrische Verschneidung einer Vektorisierung des Rasters mit einem geometrischen Datentyp.

Synopsis

setof geomval **ST_Intersection**(geometry geom, raster rast, integer band_num=1);
setof geomval **ST_Intersection**(raster rast, geometry geom);
setof geomval **ST_Intersection**(raster rast, integer band, geometry geomin);
raster **ST_Intersection**(raster rast1, raster rast2, double precision[] nodataval);
raster **ST_Intersection**(raster rast1, raster rast2, text returnband, double precision[] nodataval);
raster **ST_Intersection**(raster rast1, integer band1, raster rast2, integer band2, double precision[] nodataval);
raster **ST_Intersection**(raster rast1, integer band1, raster rast2, integer band2, text returnband, double precision[] nodataval);

Description

Gibt Geometry-PixelValue Paare, oder einen Raster aus, der durch die Schnittmenge der beiden Raster bestimmt wird, oder durch die geometrische Verschneidung einer Vektorisierung des Rasters mit einem geometrischen Datentyp.

Die ersten drei Varianten, die ein "setof geomval" zurückgeben, führen die Berechnungen im Vektorraum aus. Der Raster wird zuerst in eine Menge von "geomval"-Zeilen vektorisiert (mit ST_DumpAsPolygons) und anschließend mit der Geometrie über die PostGIS Funktion ST_Intersection(geometry, geometry) verschnitten. Wenn die verschnittene Geometrie nur aus NODATA Werten besteht, wird eine leere Geometrie zurückgegeben. Diese wird üblicherweise durch die richtige Verwendung von ST_Intersects in der WHERE-Klausel ausgeschlossen.

Sie können auf die Geometriebestandteile und die Werte der erzeugten "geomvals" zugreifen, indem Sie diese mit Klammern versehen und '.geom' oder '.val' am Ende des Ausdrucks hinzufügen; z.B. (ST_Intersection(rast, geom)).geom

Die anderen Varianten, welche einen Raster zurückgeben, führen die Berechnungen im Rasterraum aus. Sie verwenden die Version mit den zwei Rastern von ST_MapAlgebraExpr um die Verschneidung durchzuführen.

Die Ausdehnung des resultierenden Raster entspricht der Ausdehnung des geometrischen Durchschnitts der beiden Raster. Der resultierende Raster enthält die Bänder 'BAND1', 'BAND2' und 'BOTH', gefolgt von dem Parameter `returnband`. Wenn

irgendein Band Bereiche mit NODATA-Werten enthält, so werden diese Bereiche in allen Bändern des resultierenden Raster zu NODATA. Anders ausgedückt, jedes Pixel, das ein Pixel mit NODATA-Wert schneidet wird im Ergebnis selbst zu einem Pixel mit NODATA-Wert.

Raster die aus einer Operation mit ST_Intersection resultieren, müssen in den Bereichen wo sie sich nicht schneiden, einen NODATA-Wert aufweisen. Sie können den NODATA Wert eines jeden resultierenden Bandes festlegen oder ersetzen, indem Sie ein Feld `nodataval[]` übergeben, das einen oder zwei NODATA Werte - 'BAND1', 'BAND2' oder 'BOTH' - enthält. Der erste Wert in dem Feld ersetzt den NODATA Wert im ersten Band, der zweite Wert ersetzt den NODATA Wert im zweiten Band. Wenn für ein übergebenes Band kein NODATA Wert festgelegt wurde und auch keiner als Feld übergeben wurde, dann wird der Wert mit der Funktion ST_MinPossibleValue ausgewählt. Alle Varianten, die ein Feld mit NODATA-Werten akzeptieren, nehmen auch einen einzelnen Wert entgegen, welcher dann auf alle verlangten Bänder übertragen wird.

Bei sämtlichen Varianten wird Band 1 angenommen, wenn keine Bandnummer angegeben ist. Wenn Sie die Verschneidung zwischen einem Raster und einer Geometrie als Raster ausgegeben haben wollen, sehen Sie bitte ST_Clip.



Anmerkung
For more control over the result extent or NODATA handling, use the two-raster variant of ST_MapAlgebraExpr. To compute a raster-space intersection with a geometry, use ST_Clip.
Use this function together with ST_Intersects and an index on the raster or geometry column when filtering table data.

Enhanced: 2.0.0 - Verschneidungsoperation im Rasterraum eingeführt. In Vorgängerversionen von 2.0.0 wurde lediglich die Verschneidung im Vektorraum unterstützt.

Beispiele: Geometrie, Raster -- das Ergebnis sind "geomvals"

Code

```
SELECT
    foo.rid,
    foo.gid,
    GeometryType((foo.geomval).geom) AS geom_type,
    (foo.geomval).val AS value
FROM (
    SELECT
        A.rid,
        g.gid,
        ST_Intersection(A.rast, g.geom) As geomval
    FROM dummy_rast AS A
    CROSS JOIN (
        VALUES
            (1, ST_Point(3427928, 5793243.85) ),
            (2, 'LINESTRING(3427927.85 5793243.75,3427927.8 5793243.75,3427927.8 5793243.8)' ↩
    ::geometry),
            (3, 'LINESTRING(1 2,3 4)'::geometry)
    ) As g(gid, geom)
    WHERE A.rid = 2
) AS foo;
```

Output

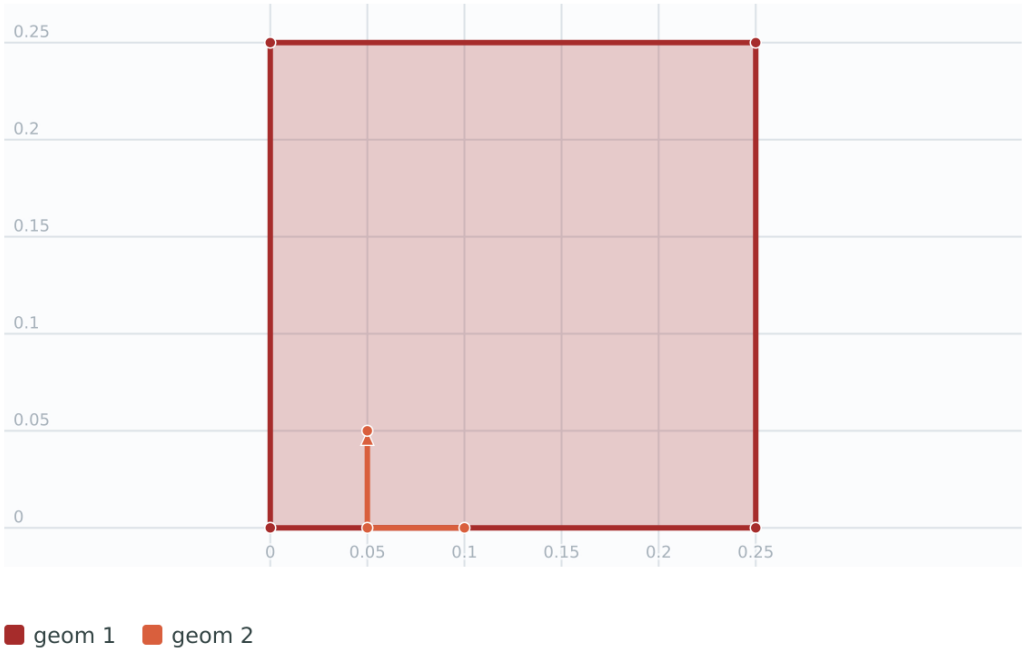
```
rid | gid |      geom_type       | value
-----+-----+----------------------+-------
   2 |   1 | POINT                |   249
   2 |   1 | POINT                |   253
   2 |   2 | POINT                |   254
   2 |   2 | POINT                |   251
   2 |   2 | POINT                |   253
   2 |   2 | LINESTRING           |   252
   2 |   2 | MULTILINESTRING      |   250
   2 |   3 | GEOMETRYCOLLECTION   |
```

The same line example rendered in a common coordinate frame. The raster footprint and complete input line remain visible behind the returned intersection pieces.

Code

```
WITH source AS (
    SELECT
        rast,
        ST_GeomFromText(
            'LINESTRING(3427927.85 5793243.75,'
            '3427927.8 5793243.75,3427927.8 5793243.8)'
        ) AS input_geometry
    FROM dummy_rast
    WHERE rid = 2
), intersection AS (
SELECT
    rast,
    input_geometry,
    ST_Collect(gv.geom) AS result_geometry
FROM source
CROSS JOIN LATERAL ST_Intersection(rast, input_geometry) AS gv
GROUP BY rast, input_geometry
), layers AS (
    SELECT
        'raster' AS title,
        ST_Translate(ST_Envelope(rast), -3427927.75, -5793243.75) AS geom
    FROM intersection
    UNION ALL
    SELECT
        'input',
        ST_Translate(input_geometry, -3427927.75, -5793243.75)
    FROM intersection
    UNION ALL
    SELECT
        'intersection',
        ST_Translate(result_geometry, -3427927.75, -5793243.75)
    FROM intersection
)
SELECT title, geom AS geom
FROM layers
ORDER BY CASE title
    WHEN 'raster' THEN 1
    WHEN 'input' THEN 2
    ELSE 3
END;
```

Figure



Siehe auch

geomval, ST_Intersects, ST_MapAlgebraExpr, ST_Clip, ST_AsText

11.12.5 ST_MapAlgebra (callback function version)

ST_MapAlgebra (callback function version) — Die Version mit der Rückruffunktion - Gibt für einen oder mehrere Eingaberaster einen Raster mit einem Band, den Bandindizes und einer vom Anwender vorgegebenen Rückruffunktion zurück.

Synopsis

raster **ST_MapAlgebra**(rastbandarg[] rastbandargset, regprocedure callbackfunc, text pixeltype=NULL, text extenttype=INTERSECTIO
raster customextent=NULL, integer distancex=0, integer distancey=0, text[] VARIADIC userargs=NULL);
raster **ST_MapAlgebra**(raster rast, integer[] nband, regprocedure callbackfunc, text pixeltype=NULL, text extenttype=FIRST,
raster customextent=NULL, integer distancex=0, integer distancey=0, text[] VARIADIC userargs=NULL);
raster **ST_MapAlgebra**(raster rast, integer nband, regprocedure callbackfunc, text pixeltype=NULL, text extenttype=FIRST,
raster customextent=NULL, integer distancex=0, integer distancey=0, text[] VARIADIC userargs=NULL);
raster **ST_MapAlgebra**(raster rast1, integer nband1, raster rast2, integer nband2, regprocedure callbackfunc, text pixeltype=NULL,
text extenttype=INTERSECTION, raster customextent=NULL, integer distancex=0, integer distancey=0, text[] VARIADIC userargs=NU
raster **ST_MapAlgebra**(raster rast, integer nband, regprocedure callbackfunc, float8[] mask, boolean weighted, text pixelty-
pe=NULL, text extenttype=INTERSECTION, raster customextent=NULL, text[] VARIADIC userargs=NULL);

Description

Gibt für einen oder mehrere Eingaberaster einen Raster mit einem Band, den Bandindizes und einer vom Anwender vorgegebenen Rückruffunktion zurück.

`rast,rast1,rast2, rastbandargset` Raster die mit der Map Algebra Operation ausgewertet werden.

`rastbandargset` ermöglicht die Anwendung einer Map Algebra Operation auf viele Raster und/oder viele Bänder. Siehe das Beispiel zu Variante 1.

nband, nband1, nband2 Die Nummern der Rasterbänder, die ausgewertet werden sollen. Die Bänder können über "nband" als Integer oder Integer[] angegeben werden. Bei der Variante mit 2 Raster steht "nband1" für die Bänder von "rast1" und nband2 für die Bänder von rast2.

callbackfunc Der Parameter `callbackfunc` muss der Name und die Signatur einer SQL- oder PL/pgSQL-Funktion sein, die in eine regprocedure umgewandelt wird. Ein Beispiel für eine PL/pgSQL-Funktion ist:

Code

```
CREATE OR REPLACE FUNCTION sample_callbackfunc(value double precision[][][], pos ↩
    integer[][], VARIADIC userargs text[])
    RETURNS double precision
    AS $$
    BEGIN
        RETURN 0;
    END;
    $$ LANGUAGE 'plpgsql' IMMUTABLE;
```

The `callbackfunc` must have two or three arguments: a 3-dimension double precision array, a 2-dimension integer array and, optionally, a variadic 1-dimension text array. The first argument `value` is the set of values (as double precision) from all input rasters. The three dimensions (where indexes are 1-based) are: raster #, row y, column x. The second argument `position` is the set of pixel positions from the output raster and input rasters. The outer dimension (where indexes are 0-based) is the raster #. The position at outer dimension index 0 is the output raster's pixel position. For each outer dimension, there are two elements in the inner dimension for X and Y. The optional third argument `userargs` is for passing through any user-specified arguments.

The callback return value is used as the value of the output pixel. If the callback returns `NULL`, the output pixel is set to NODATA when the output band has a NODATA value. If the output band has no NODATA value, a `NULL` callback result raises an error.

Die Übergabe eines regprocedure Arguments an eine SQL-Funktion erfordert die Übergabe der vollständigen Funktionssignatur und die anschließende Umwandlung in einen regprocedure Typ. Um die obige Beispiel-PL/pgSQL-Funktion als Argument zu übergeben, lautet das SQL für das Argument:

Code

```
'sample_callbackfunc(double precision[], integer[], text[])'::regprocedure
```

Note that PostgreSQL array types do not encode dimensions in function signatures, so multidimensional callback arguments are written as double precision[] and integer[] in the regprocedure signature.

mask Ein n-dimensionales Feld (Matrix) von Zahlen, das verwendet wird um Zellen für die "Map Algebra"-Rückruffunktion zu filtern. 0 bedeutet, dass ein Nachbarzellwert wie NODATA behandelt werden soll. 1 bedeutet, dass dieser Wert als Datum behandelt werden soll. Wenn der Parameter "weighted" (gewichtet) TRUE ist, dann werden diese Werte als Multiplikatoren für die Pixelwerte in der Nachbarschaft verwendet.

weighted Boolesche Variable (TRUE/FALSE), die angibt ob ein Wert der Maske gewichtet (mit dem ursprünglichen Wert multipliziert) werden soll oder nicht (gilt nur für den ersten, der die Maske übernimmt).

pixeltype Wenn `pixeltype` angegeben ist, dann hat das Band des neuen Rasters diesen Pixeltyp. Wenn für den Pixeltyp NULL oder kein Wert übergeben wird, dann hat das neue Rasterband denselben Pixeltyp wie das angegebene Band des ersten Raster (für die Lagevergleiche: INTERSECTION, UNION, FIRST, CUSTOM), oder wie das spezifizierte Band des entsprechenden Rasters (für die Lagevergleiche:SECOND, LAST). Im Zweifelsfall sollten Sie den `pixeltype` immer angeben.

Der resultierende Pixeltyp des Zielraster muss entweder aus ST_BandPixelType sein, weggelassen oder auf NULL gesetzt werden.

extenttype Mögliche Werte sind INTERSECTION (standardmäßig), UNION, FIRST (standardmäßig für Einzelraster-Varianten), SECOND, LAST, CUSTOM.

customextent Wenn `extentype` CUSTOM ist, dann muss ein Raster für `customextent` übergeben werden. Siehe Beispiel 4 von Variante 1.

distancex Der Abstand in Pixel von der Referenzzelle in X-Richtung. Die Breite der resultierenden Matrix ergibt sich aus `2*distancex + 1`. Wenn nicht angegeben, dann wird nur die Referenzzelle berücksichtigt (keine Nachbarschaft/"neighborhoo of 0").

distancey Die Entfernung der Pixel zur Referenzzelle in Y-Richtung. Die Höhe der resultierenden Matrix ergibt sich aus `2*distancey + 1`. Wenn nicht angegeben, dann wird nur die Referenzzelle berücksichtigt (keine Nachbarschaft/"neighborhoo of 0").

userargs If the `callbackfunc` declares a third argument, that argument is a variadic text array. All trailing text arguments are passed through to the specified `callbackfunc`, and are contained in the `userargs` argument.



Anmerkung
Für nähere Information zu dem Schlüsselwort VARIADIC, sehen Sie bitte die PostgreSQL Dokumentation und den Abschnitt "SQL Functions with Variable Numbers of Arguments" unter Query Language (SQL) Functions.

Variante 1 nimmt ein Feld an `rastbandarg` entgegen, wodurch die Anwendung einer Map Algebra Operation auf mehrere Raster und/oder mehrere Bänder ermöglicht wird. Siehe das Beispiel zu Variante 1.

Die Varianten 2 und 3 führen die Operation auf einem oder mehreren Bändern eines Raster aus. Siehe die Beispiele mit Variante 2 und 3.

Die Variante 4 führt die Operationen an zwei Raster mit einem Band pro Raster aus. Siehe das Beispiel mit Variante 4.

Verfügbarkeit: 2.2.0: Möglichkeit eine Maske hinzuzufügen

Verfügbarkeit: 2.1.0

Beispiele

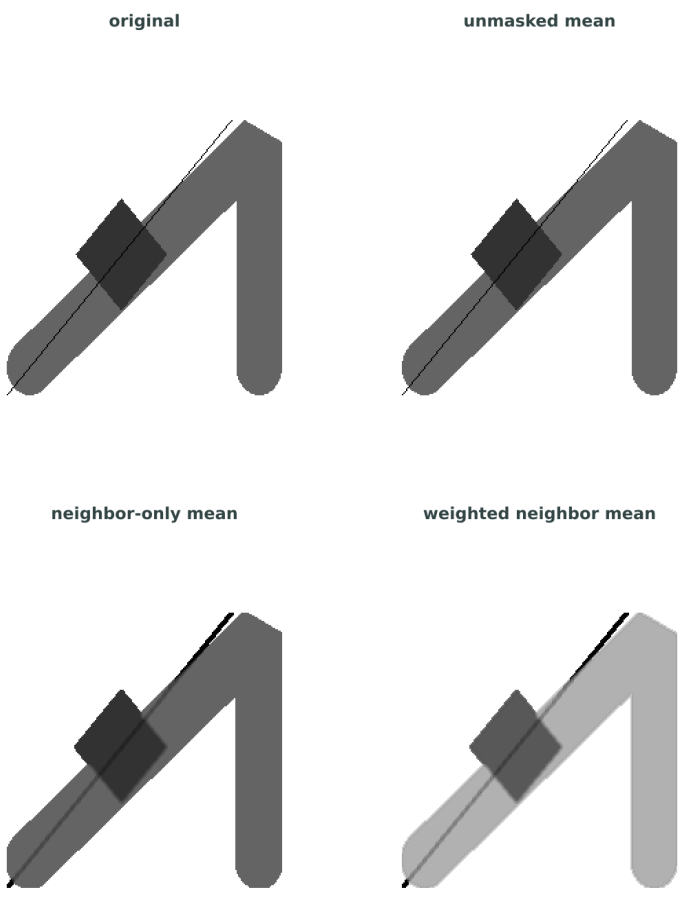
Compare an unmasked mean with neighbor-only and weighted-neighbor masks. The weighted mask multiplies neighboring values by 2 before computing the result.

Code

```
WITH foo AS (
    SELECT ST_SetBandNoDataValue(
        ST_SetValue(
            ST_SetValue(
                ST_AsRaster(
                    ST_Buffer('LINESTRING(50 50,100 90,100 50)'::geometry, 5, 'join=bevel'),
                    200, 200, ARRAY['8BUI'], ARRAY[100], ARRAY[0]
                ),
                ST_Buffer('POINT(70 70)'::geometry, 10, 'quad_segs=1'),
                50
            ),
            'LINESTRING(20 20,100 100,150 98)'::geometry,
            1
        ),
        0
    ) AS rast
), variants AS (
    SELECT 'original' AS title, rast
    FROM foo
    UNION ALL
    SELECT 'unmasked mean' AS title,
        ST_MapAlgebra(rast, 1, 'ST_Mean4ma(double precision[], int[], text[])'::regprocedure ↩
    ) AS rast
    FROM foo
    UNION ALL
```

```
    SELECT 'neighbor-only mean' AS title,
        ST_MapAlgebra(
            rast,
            1,
            'ST_Mean4ma(double precision[], int[], text[])'::regprocedure,
            '{{1,1,1}, {1,0,1}, {1,1,1}}'::double precision[],
            false
        ) AS rast
    FROM foo
    UNION ALL
    SELECT 'weighted neighbor mean' AS title,
        ST_MapAlgebra(
            rast,
            1,
            'ST_Mean4ma(double precision[], int[], text[])'::regprocedure,
            '{{2,2,2}, {2,0,2}, {2,2,2}}'::double precision[],
            true
        ) AS rast
    FROM foo
)
SELECT title, ST_AsPNG(rast) AS image
FROM variants;
```

Figure



Variant 1.

Ein Raster, ein Band

Code

```
WITH foo AS (
    SELECT 1 AS rid, ST_AddBand(ST_MakeEmptyRaster(2, 2, 0, 0, 1, -1, 0, 0, 0), 1, '16BUI', ↩
    1, 0) AS rast
)
SELECT
    ST_MapAlgebra(ARRAY[ROW(rast, 1)]::rastbandarg[],
        'sample_callbackfunc(double precision[], int[], text[])'::regprocedure
    ) AS rast
FROM foo
```

Ein Raster, mehrere Bänder

Code

```
WITH foo AS (
    SELECT 1 AS rid, ST_AddBand(ST_AddBand(ST_AddBand(ST_MakeEmptyRaster(2, 2, 0, 0, 1, -1, ↩
    0, 0, 0), 1, '16BUI', 1, 0), 2, '8BUI', 10, 0), 3, '32BUI', 100, 0) AS rast
)
SELECT
    ST_MapAlgebra(ARRAY[ROW(rast, 3), ROW(rast, 1), ROW(rast, 3), ROW(rast, 2)]::rastbandarg ↩
    [],
        'sample_callbackfunc(double precision[], int[], text[])'::regprocedure
    ) AS rast
FROM foo
```

Mehrere Raster, mehrere Bänder

Code

```
WITH foo AS (
    SELECT 1 AS rid, ST_AddBand(ST_AddBand(ST_AddBand(ST_MakeEmptyRaster(2, 2, 0, 0, 1, -1, ↩
    0, 0, 0), 1, '16BUI', 1, 0), 2, '8BUI', 10, 0), 3, '32BUI', 100, 0) AS rast UNION ALL
    SELECT 2 AS rid, ST_AddBand(ST_AddBand(ST_AddBand(ST_MakeEmptyRaster(2, 2, 0, 1, 1, -1, ↩
    0, 0, 0), 1, '16BUI', 2, 0), 2, '8BUI', 20, 0), 3, '32BUI', 300, 0) AS rast
)
SELECT
    ST_MapAlgebra(ARRAY[ROW(t1.rast, 3), ROW(t2.rast, 1), ROW(t2.rast, 3), ROW(t1.rast, 2) ↩
    ]::rastbandarg[],
        'sample_callbackfunc(double precision[], int[], text[])'::regprocedure
    ) AS rast
FROM foo t1
CROSS JOIN foo t2
WHERE t1.rid = 1
    AND t2.rid = 2
```

Complete example of tiles of a coverage with neighborhood.

Code

```
WITH foo AS (
    SELECT 0 AS rid, ST_AddBand(ST_MakeEmptyRaster(2, 2, 0, 0, 1, -1, 0, 0, 0), 1, '16BUI', ↩
    1, 0) AS rast UNION ALL
    SELECT 1, ST_AddBand(ST_MakeEmptyRaster(2, 2, 2, 0, 1, -1, 0, 0, 0), 1, '16BUI', 2, 0) ↩
    AS rast UNION ALL
```

```
    SELECT 2, ST_AddBand(ST_MakeEmptyRaster(2, 2, 4, 0, 1, -1, 0, 0, 0), 1, '16BUI', 3, 0) ↩
    AS rast UNION ALL

    SELECT 3, ST_AddBand(ST_MakeEmptyRaster(2, 2, 0, -2, 1, -1, 0, 0, 0), 1, '16BUI', 10, 0) ↩
     AS rast UNION ALL
    SELECT 4, ST_AddBand(ST_MakeEmptyRaster(2, 2, 2, -2, 1, -1, 0, 0, 0), 1, '16BUI', 20, 0) ↩
     AS rast UNION ALL
    SELECT 5, ST_AddBand(ST_MakeEmptyRaster(2, 2, 4, -2, 1, -1, 0, 0, 0), 1, '16BUI', 30, 0) ↩
     AS rast UNION ALL

    SELECT 6, ST_AddBand(ST_MakeEmptyRaster(2, 2, 0, -4, 1, -1, 0, 0, 0), 1, '16BUI', 100, ↩
    0) AS rast UNION ALL
    SELECT 7, ST_AddBand(ST_MakeEmptyRaster(2, 2, 2, -4, 1, -1, 0, 0, 0), 1, '16BUI', 200, ↩
    0) AS rast UNION ALL
    SELECT 8, ST_AddBand(ST_MakeEmptyRaster(2, 2, 4, -4, 1, -1, 0, 0, 0), 1, '16BUI', 300, ↩
    0) AS rast
)
SELECT
    t1.rid,
    ST_MapAlgebra(ARRAY[ROW(ST_Union(t2.rast), 1)]::rastbandarg[],
        'sample_callbackfunc(double precision[], int[], text[])'::regprocedure,
        '32BUI',
        'CUSTOM', t1.rast,
        1, 1
    ) AS rast
FROM foo t1
CROSS JOIN foo t2
WHERE t1.rid = 4
    AND t2.rid BETWEEN 0 AND 8
    AND ST_Intersects(t1.rast, t2.rast)
GROUP BY t1.rid, t1.rast
```

Example like the prior one for tiles of a coverage with neighborhood but using only basic SQL features.

Code

```
WITH src AS (
    SELECT 0 AS rid, ST_AddBand(ST_MakeEmptyRaster(2, 2, 0, 0, 1, -1, 0, 0, 0), 1, '16BUI', ↩
    1, 0) AS rast UNION ALL
    SELECT 1, ST_AddBand(ST_MakeEmptyRaster(2, 2, 2, 0, 1, -1, 0, 0, 0), 1, '16BUI', 2, 0) ↩
    AS rast UNION ALL
    SELECT 2, ST_AddBand(ST_MakeEmptyRaster(2, 2, 4, 0, 1, -1, 0, 0, 0), 1, '16BUI', 3, 0) ↩
    AS rast UNION ALL

    SELECT 3, ST_AddBand(ST_MakeEmptyRaster(2, 2, 0, -2, 1, -1, 0, 0, 0), 1, '16BUI', 10, 0) ↩
     AS rast UNION ALL
    SELECT 4, ST_AddBand(ST_MakeEmptyRaster(2, 2, 2, -2, 1, -1, 0, 0, 0), 1, '16BUI', 20, 0) ↩
     AS rast UNION ALL
    SELECT 5, ST_AddBand(ST_MakeEmptyRaster(2, 2, 4, -2, 1, -1, 0, 0, 0), 1, '16BUI', 30, 0) ↩
     AS rast UNION ALL

    SELECT 6, ST_AddBand(ST_MakeEmptyRaster(2, 2, 0, -4, 1, -1, 0, 0, 0), 1, '16BUI', 100, ↩
    0) AS rast UNION ALL
    SELECT 7, ST_AddBand(ST_MakeEmptyRaster(2, 2, 2, -4, 1, -1, 0, 0, 0), 1, '16BUI', 200, ↩
    0) AS rast UNION ALL
    SELECT 8, ST_AddBand(ST_MakeEmptyRaster(2, 2, 4, -4, 1, -1, 0, 0, 0), 1, '16BUI', 300, ↩
    0) AS rast
), foo AS (
    SELECT
        t1.rid,
        ST_Union(t2.rast) AS rast
```

```
    FROM src t1
    JOIN src t2
        ON ST_Intersects(t1.rast, t2.rast)
        AND t2.rid BETWEEN 0 AND 8
    WHERE t1.rid = 4
    GROUP BY t1.rid
), bar AS (
    SELECT
        t1.rid,
        ST_MapAlgebra(
            ARRAY[ROW(t2.rast, 1)]::rastbandarg[],
            'sample_callbackfunc(double precision[], int[], text[])'::regprocedure,
            '32BUI',
            'CUSTOM', t1.rast,
            1, 1
        ) AS rast
    FROM src t1
    JOIN foo t2
        ON t1.rid = t2.rid
)
SELECT
    rid,
    (ST_MetaData(rast)),
    (ST_BandMetaData(rast, 1)),
    ST_Value(rast, 1, 1, 1)
FROM bar;
```

Variants 2 and 3.

Ein Raster, mehrere Bänder

Code

```
WITH foo AS (
    SELECT 1 AS rid, ST_AddBand(ST_AddBand(ST_AddBand(ST_MakeEmptyRaster(2, 2, 0, 0, 1, -1, ↩
    0, 0, 0), 1, '16BUI', 1, 0), 2, '8BUI', 10, 0), 3, '32BUI', 100, 0) AS rast
)
SELECT
    ST_MapAlgebra(
        rast,
        ARRAY[3, 1, 3, 2]::integer[],
        'sample_callbackfunc(double precision[], int[], text[])'::regprocedure
    ) AS rast
FROM foo
```

Ein Raster, ein Band

Code

```
WITH foo AS (
    SELECT 1 AS rid, ST_AddBand(ST_AddBand(ST_AddBand(ST_MakeEmptyRaster(2, 2, 0, 0, 1, -1, ↩
    0, 0, 0), 1, '16BUI', 1, 0), 2, '8BUI', 10, 0), 3, '32BUI', 100, 0) AS rast
)
SELECT
    ST_MapAlgebra(
        rast,
        2,
        'sample_callbackfunc(double precision[], int[], text[])'::regprocedure
    ) AS rast
FROM foo
```

Variant 4.

Zwei Raster, zwei Bänder

Code

```
WITH foo AS (
    SELECT 1 AS rid, ST_AddBand(ST_AddBand(ST_AddBand(ST_MakeEmptyRaster(2, 2, 0, 0, 1, -1,  ↩
    0, 0, 0), 1, '16BUI', 1, 0), 2, '8BUI', 10, 0), 3, '32BUI', 100, 0) AS rast UNION ALL
    SELECT 2 AS rid, ST_AddBand(ST_AddBand(ST_AddBand(ST_MakeEmptyRaster(2, 2, 0, 1, 1, -1,  ↩
    0, 0, 0), 1, '16BUI', 2, 0), 2, '8BUI', 20, 0), 3, '32BUI', 300, 0) AS rast
)
SELECT
    ST_MapAlgebra(
        t1.rast,
        2,
        t2.rast,
        1,
        'sample_callbackfunc(double precision[], int[], text[])'::regprocedure
    ) AS rast
FROM foo t1
CROSS JOIN foo t2
WHERE t1.rid = 1
    AND t2.rid = 2
```

Siehe auch

rastbandarg, ST_Union, ST_MapAlgebra (expression version)

11.12.6 ST_MapAlgebra (expression version)

ST_MapAlgebra (expression version) — Version mit Ausdrücken - Gibt für einen oder zwei Ausgangsraster, Bandindizes und einer oder mehreren vom Anwender vorgegebenen SQL-Ausdrücken, einen Raster mit einem Band zurück.

Synopsis

raster **ST_MapAlgebra**(raster rast, integer nband, text pixeltype, text expression, double precision nodataval=NULL);
raster **ST_MapAlgebra**(raster rast, text pixeltype, text expression, double precision nodataval=NULL);
raster **ST_MapAlgebra**(raster rast1, integer nband1, raster rast2, integer nband2, text expression, text pixeltype=NULL, text extenttype=INTERSECTION, text nodata1expr=NULL, text nodata2expr=NULL, double precision nodatanodataval=NULL);
raster **ST_MapAlgebra**(raster rast1, raster rast2, text expression, text pixeltype=NULL, text extenttype=INTERSECTION, text nodata1expr=NULL, text nodata2expr=NULL, double precision nodatanodataval=NULL);

Description

Version mit Ausdrücken - Gibt für einen oder zwei Ausgangsraster, Bandindizes und einer oder mehreren vom Anwender vorgegebenen SQL-Ausdrücken, einen Raster mit einem Band zurück.

Verfügbarkeit: 2.1.0

Beschreibung: Variante 1 und 2 (ein Raster)

Erstellt ein neues Rasterband indem eine gültige algebraische PostgreSQL Operation (`expression`) auf den Ausgangsraster (`rast`) angewendet wird. Wenn `nband` nicht angegeben ist, wird Band 1 angenommen. Der Zielraster hat die selbe Georeferenzierung, Breite und Höhe wie der ursprüngliche Raster, hat aber nur ein Band.

Wenn `pixeltype` angegeben ist, dann hat das Band des neuen Rasters diesen Pixeltyp. Wenn für den Pixeltyp NULL übergeben wird, dann hat das neue Rasterband denselben Pixeltyp wie das gegebene Band von `rast`

- Erlaubte Schlüsselwörter für `expression`
  1. `[rast]` - Zellwert der Pixel von Interesse
  2. `[rast.val]` - Zellwert der Pixel von Interesse
  3. `[rast.x]` - Rasterspalte (von 1 wegzählend) der Pixel von Interesse
  4. `[rast.y]` - Rasterszeile (von 1 wegzählend) der Pixel von Interesse

Beschreibung: Variante 3 und 4 (zwei Raster)

Erstellt einen neuen Raster mit einem Band, indem eine gültige algebraische PostgreSQL Operation auf die beiden, durch den Ausdruck `expression` bestimmten Ausgangsrasterbänder `rast1` und `rast2`) angewendet wird. Wenn `band1` oder `band2` nicht angegeben ist, wird Band 1 angenommen. Der Zielraster wird an dem Gitter des ersten Raster ausgerichtet (Größe, Versatz und Eckpunkte der Pixel). Die Ausdehnung des Zielrasters wird durch den Parameter `extenttype` bestimmt.

`expression` Ein algebraischer PostgreSQL Ausdruck, der zwei Raster und in PostgreSQL definierte Funktionen/Operatoren einbezieht, die den Pixelwert für sich schneidende Pixel festlegt. z.B. (([rast1] + [rast2])/2.0)::integer

`pixeltype` Der resultierende Pixeltyp des Zielraster muss entweder aus ST_BandPixelType sein, weggelassen oder auf NULL gesetzt werden. Wenn er nicht übergeben wird oder auf NULL gesetzt ist, wird er standardmäßig auf den Pixeltyp des ersten Raster gesetzt.

`extenttype` Bestimmt die Ausdehnung des resultierenden Raster

1. `INTERSECTION` - Die Ausdehnung des neuen Raster entspricht der Schnittmenge der beiden Raster. Die Standardeinstellung.
2. `UNION` - Die Ausdehnung des neuen Raster entspricht der Vereinigungsmenge der beiden Raster.
3. `FIRST` - Die Ausdehnung des neuen Raster entspricht jener des ersten Raster.
4. `SECOND` - Die Ausdehnung des neuen Raster entspricht jender des zweiten Raster.

`nodata1expr` Ein algebraischer Ausdruck der nur `rast2` oder eine Konstante einbezieht. Dieser bestimmt was zurückgegeben wird, wenn die Pixel von `rast1` NODATA Werte sind und die räumlich übereinstimmenden Pixel von rast2 Werte aufweisen.

`nodata2expr` Ein algebraischer Ausdruck der nur `rast1` oder eine Konstante einbezieht. Dieser bestimmt was zurückgegeben wird, wenn die Pixel von `rast2` NODATA Werte haben und die räumlich übereinstimmenden Pixel von rast1 Werte aufweisen.

`nodatanodataval` Eine numerische Konstante die ausgegeben wird, wenn die übereinstimmenden Pixel der beiden Raster "rast1" und "rast2" nur NODATA Werte enthalten.

- Zugelassene Schlüsselwörter in `expression`, `nodata1expr` und `nodata2expr`
  1. `[rast1]` - Zellwert der Pixel von Interesse von `rast1`
  2. `[rast1.val]` - Zellwert der Pixel von Interesse von `rast1`
  3. `[rast1.x]` - Rasterspalte (von 1 wegzählend) der Pixel von Interesse von `rast1`
  4. `[rast1.y]` - Rasterzeile (von 1 wegzählend) der Pixel von Interesse von `rast1`
  5. `[rast2]` - Zellwert der Pixel von Interesse von `rast2`
  6. `[rast2.val]` - Zellwert der Pixel von Interesse von `rast2`
  7. `[rast2.x]` - Rasterspalte (von 1 wegzählend) der Pixel von Interesse von `rast2`
  8. `[rast2.y]` - Rasterzeile (von 1 wegzählend) der Pixel von Interesse von `rast2`

Beispiele

Variants 1 and 2.

Code

```
WITH foo AS (
    SELECT ST_AddBand(ST_MakeEmptyRaster(10, 10, 0, 0, 1, 1, 0, 0, 0), '32BF'::text, 1, -1) ↩
    AS rast
)
SELECT
    ST_MapAlgebra(rast, 1, NULL, 'ceil([rast]*[rast.x]/[rast.y]+[rast.val])')
FROM foo;
```

Variant 3 and 4.

Code

```
WITH foo AS (
    SELECT 1 AS rid, ST_AddBand(ST_AddBand(ST_AddBand(ST_MakeEmptyRaster(2, 2, 0, 0, 1, -1, ↩
    0, 0, 0), 1, '16BUI', 1, 0), 2, '8BUI', 10, 0), 3, '32BUI'::text, 100, 0) AS rast UNION ↩
    ALL
    SELECT 2 AS rid, ST_AddBand(ST_AddBand(ST_AddBand(ST_MakeEmptyRaster(2, 2, 0, 1, 1, -1, ↩
    0, 0, 0), 1, '16BUI', 2, 0), 2, '8BUI', 20, 0), 3, '32BUI'::text, 300, 0) AS rast
)
SELECT
    ST_MapAlgebra(
        t1.rast,
        2,
        t2.rast,
        1,
        '([rast2] + [rast1.val]) / 2'
    ) AS rast
FROM foo t1
CROSS JOIN foo t2
WHERE t1.rid = 1
    AND t2.rid = 2;
```

Siehe auch

rastbandarg, ST_Union, ST_MapAlgebra (callback function version), Abschnitt 10.4.1, Abschnitt 10.4.2

11.12.7 ST_MapAlgebraExpr

ST_MapAlgebraExpr — Version mit 1 Rasterband: Erzeugt ein neues Rasterband, dass über eine gültige, algebraische PostgreSQL Operation für ein Rasterband mit gegebenen Pixeltyp erstellt wird. Wenn kein Band bestimmt ist, wird Band 1 angenommen.

Synopsis

raster **ST_MapAlgebraExpr**(raster rast, integer band, text pixeltype, text expression, double precision nodataval=NULL);
raster **ST_MapAlgebraExpr**(raster rast, text pixeltype, text expression, double precision nodataval=NULL);

Description



Warnung
ST_MapAlgebraExpr Seit der Version 2.1.0 überholt. Benutzen Sie stattdessen bitte ST_MapAlgebra (expression version) .

Erstellt ein neues Rasterband indem eine gültige algebraische PostgreSQL Operation (`expression`) auf den Ausgangsraster (`rast`) angewendet wird. Wenn kein `band` festgelegt ist, wird Band 1 angenommen. Der Zielraster hat die selbe Georeferenzierung, Breite und Höhe wie der ursprüngliche Raster, hat aber nur ein Band.

Wenn `pixeltype` angegeben ist, dann hat das Band des neuen Rasters diesen Pixeltyp. Wenn für den Pixeltyp NULL übergeben wird, dann hat das neue Rasterband denselben Pixeltyp wie das gegebene Band von `rast`

Im Ausdruck können Sie folgende Terme verwenden: `[rast]` für den Zellwert des ursprünglichen Bandes, `[rast.x]` für den von 1 wegzählenden Index der Rasterspalten, `[rast.y]` für den von 1 wegzählenden Index der Rasterzeilen.

Verfügbarkeit: 2.0.0

Beispiele

Erzeugt einen Einzelbandraster, der sich durch die Anwendung der Funktion "modulo 2" auf das ursprüngliche Rasterband ergibt.

Code

```
ALTER TABLE dummy_rast ADD COLUMN map_rast raster;
UPDATE dummy_rast SET map_rast = ST_MapAlgebraExpr(rast, NULL, 'mod([rast]::numeric,2)')  ↩
    WHERE rid = 2;

SELECT
    ST_Value(rast, 1, i, j) As origval,
    ST_Value(map_rast, 1, i, j) As mapval
FROM dummy_rast
CROSS JOIN generate_series(1, 3) AS i
CROSS JOIN generate_series(1, 3) AS j
WHERE rid = 2;
```

Output

```
origval | mapval
---------+--------
     253 |      1
     254 |      0
     253 |      1
     253 |      1
     254 |      0
     254 |      0
     250 |      0
     254 |      0
     254 |      0
```

Erzeugt einen neuen Einzelbandraster mit dem Pixeltyp 2BUI. Der ursprüngliche Raster wird dabei neu klassifiziert und mit einem NODATA Wert von 0 versehen.

Code

```
ALTER TABLE dummy_rast ADD COLUMN map_rast2 raster;
UPDATE dummy_rast SET
    map_rast2 = ST_MapAlgebraExpr(rast, '2BUI'::text, 'CASE WHEN [rast] BETWEEN 100 and 250  ↩
    THEN 1 WHEN [rast] = 252 THEN 2 WHEN [rast] BETWEEN 253 and 254 THEN 3 ELSE 0 END'::text ↩
    , '0')
WHERE rid = 2;

SELECT DISTINCT
    ST_Value(rast, 1, i, j) As origval,
    ST_Value(map_rast2, 1, i, j) As mapval
FROM dummy_rast
CROSS JOIN generate_series(1, 5) AS i
CROSS JOIN generate_series(1, 5) AS j
WHERE rid = 2;
```

Output

```
origval | mapval
---------+--------
     249 |      1
     250 |      1
     251 |
     252 |      2
     253 |      3
     254 |      3
```

Code

```
SELECT
    ST_BandPixelType(map_rast2) As b1pixtyp
FROM dummy_rast
WHERE rid = 2;
```

Output

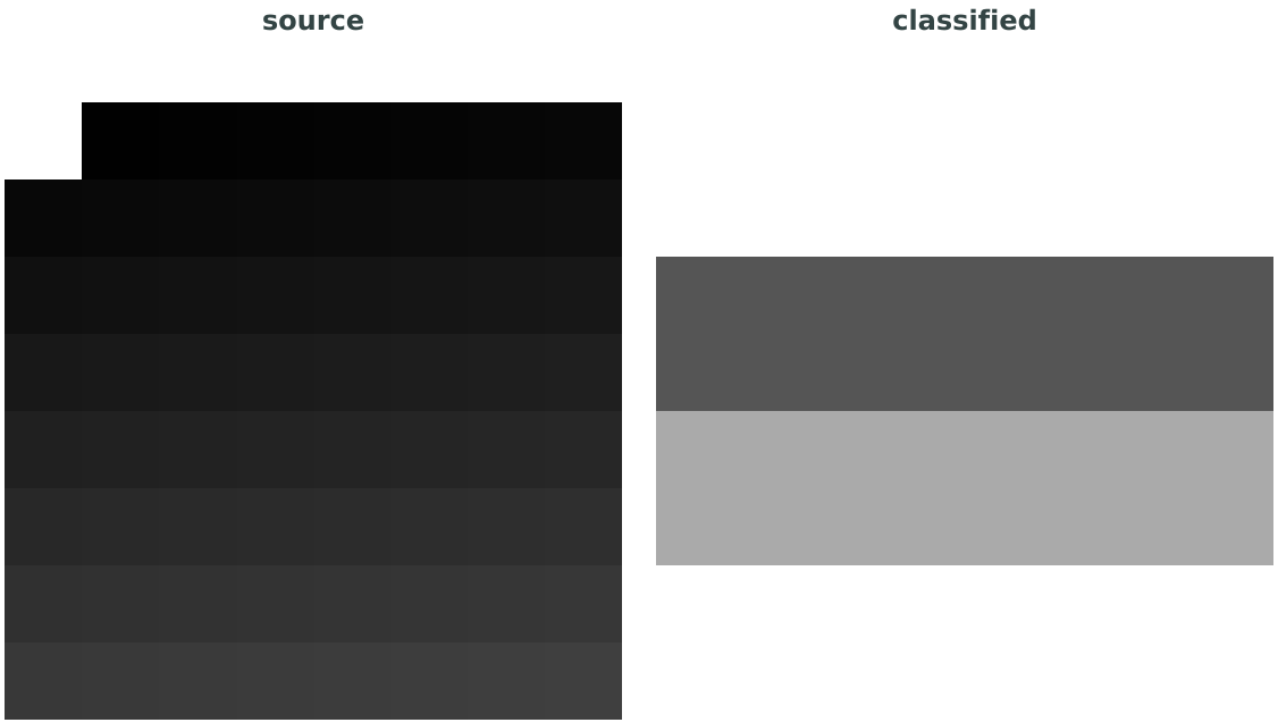
```
2BUI
```

A self-contained source raster and the raster produced by classifying its values into four display levels.

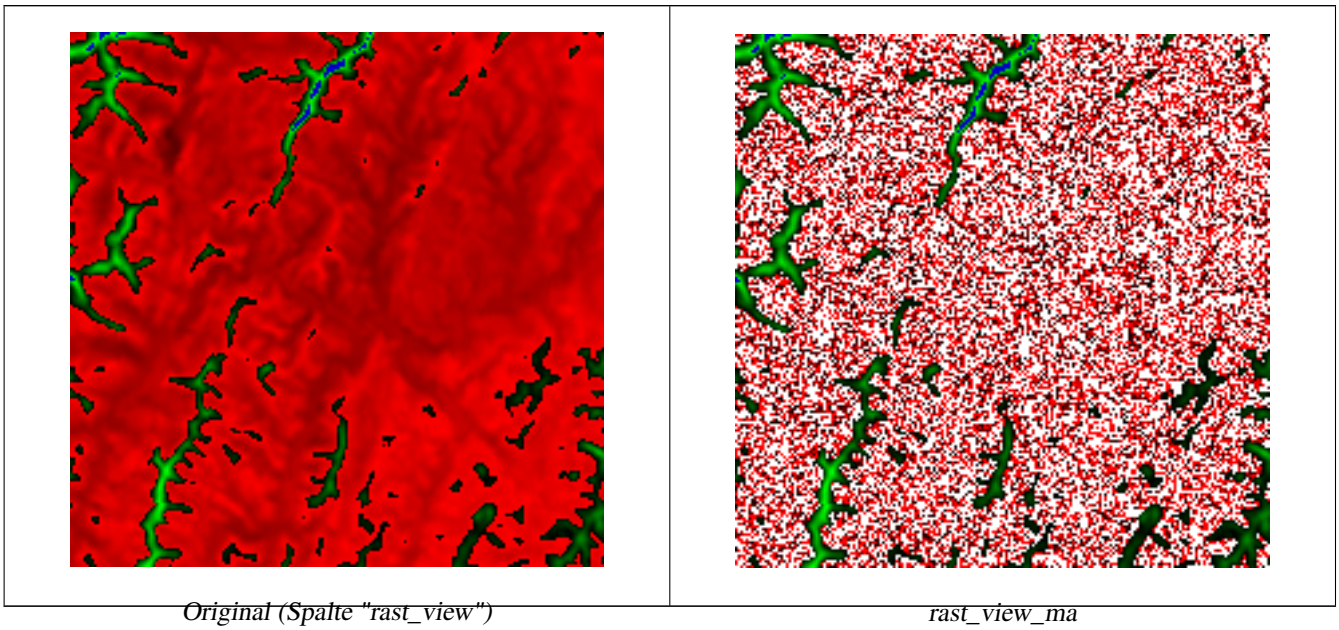
Code

```
WITH source AS (
    SELECT ST_SetValues(
        ST_AddBand(
            ST_MakeEmptyRaster(8, 8, 0, 8, 1, -1, 0, 0, 0),
            1, '8BUI', 0, 0
        ),
        1, 1, 1,
        ARRAY[
            [0, 1, 2, 3, 4, 5, 6, 7],
            [8, 9, 10, 11, 12, 13, 14, 15],
            [16, 17, 18, 19, 20, 21, 22, 23],
            [24, 25, 26, 27, 28, 29, 30, 31],
            [32, 33, 34, 35, 36, 37, 38, 39],
            [40, 41, 42, 43, 44, 45, 46, 47],
            [48, 49, 50, 51, 52, 53, 54, 55],
            [56, 57, 58, 59, 60, 61, 62, 63]
        ]::double precision[][]
    ) AS rast
), variants AS (
    SELECT 'source' AS title, rast AS rendered
    FROM source
    UNION ALL
    SELECT
        'classified',
        ST_MapAlgebraExpr(rast, '8BUI', 'floor([rast] / 16) * 85')
    FROM source
)
SELECT title, ST_AsPNG(rendered) AS image
FROM variants
ORDER BY title DESC;
```

Figure



Map algebra altering the first band of a three-band raster.



Original (Spalte "rast_view") *rast_view_ma*

Erzeugt einen neuen Raster mit 3 Bändern und demselben Pixeltyp als unser ursprünglicher Raster mit 3 Bändern. Das erste Band wird über einen Map Algebra Ausdruck geändert, die verbleibenden 2 Bänder sind unverändert.

Code

```
SELECT
    ST_AddBand(ST_AddBand(ST_AddBand(ST_MakeEmptyRaster(rast_view),
                ST_MapAlgebraExpr(rast_view, 1, NULL, 'tan([rast])*[rast]')
            ),
            ST_Band(rast_view, 2)
        ),
        ST_Band(rast_view, 3)
    )  As rast_view_ma
FROM wind
WHERE rid=167;
```

Siehe auch

ST_MapAlgebraExpr, ST_MapAlgebraFct, ST_BandPixelType, ST_GeoReference, ST_Value

11.12.8 ST_MapAlgebraExpr

ST_MapAlgebraExpr — Version mit 2 Rasterbändern: Erstellt einen neuen Einzelbandraster, indem eine gültige algebraische PostgreSQL Funktion auf die zwei Ausgangsrasterbänder und den entsprechenden Pixeltyp angewendet wird. Wenn keine Bandnummern angegeben sind, wird von jedem Raste Band 1 angenommen. Der Ergebnisraster wird nach dem Gitter des ersten Raster ausgerichtet (Skalierung, Versatz und Eckpunkte der Pixel) und hat die Ausdehnung, welche durch den Parameter "extenttype" definiert ist. Der Parameter "extenttype" kann die Werte INTERSECTION, UNION, FIRST, SECOND annehmen.

Synopsis

raster **ST_MapAlgebraExpr**(raster rast1, raster rast2, text expression, text pixeltype=same_as_rast1_band, text extenttype=INTERSEC
text nodata1expr=NULL, text nodata2expr=NULL, double precision nodatanodataval=NULL);
raster **ST_MapAlgebraExpr**(raster rast1, integer band1, raster rast2, integer band2, text expression, text pixeltype=same_as_rast1_band,
text extenttype=INTERSECTION, text nodata1expr=NULL, text nodata2expr=NULL, double precision nodatanodataval=NULL);

Description



Warnung
ST_MapAlgebraExpr Seit der Version 2.1.0 überholt. Benutzen Sie stattdessen bitte ST_MapAlgebra (expression version) .

Erstellt einen neuen Raster mit einem Band, indem eine gültige algebraische PostgreSQL Operation auf die beiden, durch den Ausdruck `expression` bestimmten Ausgangsrasterbänder `rast1` und `rast2`) angewendet wird. Wenn `band1` oder `band2` nicht angegeben ist, wird Band 1 angenommen. Der Zielraster wird an dem Gitter des ersten Raster ausgerichtet (Größe, Versatz und Eckpunkte der Pixel). Die Ausdehnung des Zielrasters wird durch den Parameter `extenttype` bestimmt.

`expression` Ein algebraischer PostgreSQL Ausdruck, der zwei Raster und in PostgreSQL definierte Funktionen/Operatoren einbezieht, die den Pixelwert für sich schneidende Pixel festlegt. z.B. (([rast1] + [rast2])/2.0)::integer

`pixeltype` Der resultierende Pixeltyp des Zielraster muss entweder aus ST_BandPixelType sein, weggelassen oder auf NULL gesetzt werden. Wenn er nicht übergeben wird oder auf NULL gesetzt ist, wird er standardmäßig auf den Pixeltyp des ersten Raster gesetzt.

`extenttype` Bestimmt die Ausdehnung des resultierenden Raster

1. INTERSECTION - Die Ausdehnung des neuen Raster entspricht der Schnittmenge der beiden Raster. Die Standardeinstellung.
2. UNION - Die Ausdehnung des neuen Raster entspricht der Vereinigungsmenge der beiden Raster.
3. FIRST - Die Ausdehnung des neuen Raster entspricht jener des ersten Raster.
4. SECOND - Die Ausdehnung des neuen Raster entspricht jender des zweiten Raster.

nodata1expr Ein algebraischer Ausdruck der nur rast2 oder eine Konstante einbezieht. Dieser bestimmt was zurückgegeben wird, wenn die Pixel von rast1 NODATA Werte sind und die räumlich übereinstimmenden Pixel von rast2 Werte aufweisen.

nodata2expr Ein algebraischer Ausdruck der nur rast1 oder eine Konstante einbezieht. Dieser bestimmt was zurückgegeben wird, wenn die Pixel von rast2 NODATA Werte haben und die räumlich übereinstimmenden Pixel von rast1 Werte aufweisen.

nodatanodataval Eine numerische Konstante die ausgegeben wird, wenn die übereinstimmenden Pixel der beiden Raster "rast1" und "rast2" nur NODATA Werte enthalten.

Wenn pixeltype angegeben ist, dann hat das Band des neuen Rasters diesen Pixeltyp. Wenn für den Pixeltyp NULL übergeben wird oder der Pixeltyp nicht festgelegt ist, dann hat das neue Rasterband denselben Pixeltyp wie das angegebene Band von rast1

Sie können den Ausdruck [rast1.val] [rast2.val] verwenden, um auf die Pixelwerte der ursprünglichen Rasterbänder zu verweisen, und [rast1.x], [rast1.y] etc. um auf die Positionen der Pixel über die Rasterspalten / Rasterzeilen zu verweisen.

Verfügbarkeit: 2.0.0

Beispiele

2 Band Intersection and Union.

Erzeugt einen Einzelbandraster, der sich durch die Anwendung der Funktion "modulo 2" auf das ursprüngliche Rasterband ergibt.

Create a cool set of rasters.

Insert some cool shapes around Boston in Massachusetts state plane meters, then map their intersection and union.

Code

```
DROP TABLE IF EXISTS fun_shapes;
CREATE TABLE fun_shapes(rid serial PRIMARY KEY, fun_name text, rast raster);

INSERT INTO fun_shapes(fun_name, rast)
VALUES ('ref', ST_AsRaster(ST_MakeEnvelope(235229, 899970, 237229, 901930, 26986), 200, 200, ↩
     '8BUI', 0, 0));

INSERT INTO fun_shapes(fun_name, rast)
WITH ref(rast) AS (SELECT rast FROM fun_shapes WHERE fun_name = 'ref' )
SELECT 'area' AS fun_name, ST_AsRaster(ST_Buffer(ST_SetSRID(ST_Point(236229, 900930), 26986) ↩
    , 1000),
            ref.rast, '8BUI', 10, 0) As rast
FROM ref
UNION ALL
SELECT 'rand bubbles',
       ST_AsRaster(
           (
               SELECT ST_Collect(geom)
               FROM (
                   SELECT ST_Buffer(
                       ST_SetSRID(ST_Point(236229 + i*random()*100, 900930 + j*random()*100) ↩
    , 26986),
                       random()*20
```

```
                    ) AS geom
                    FROM generate_series(1, 10) AS i,
                         generate_series(1, 10) AS j
                ) AS foo
            ),
            ref.rast, '8BUI', 200, 0
        )
FROM ref;

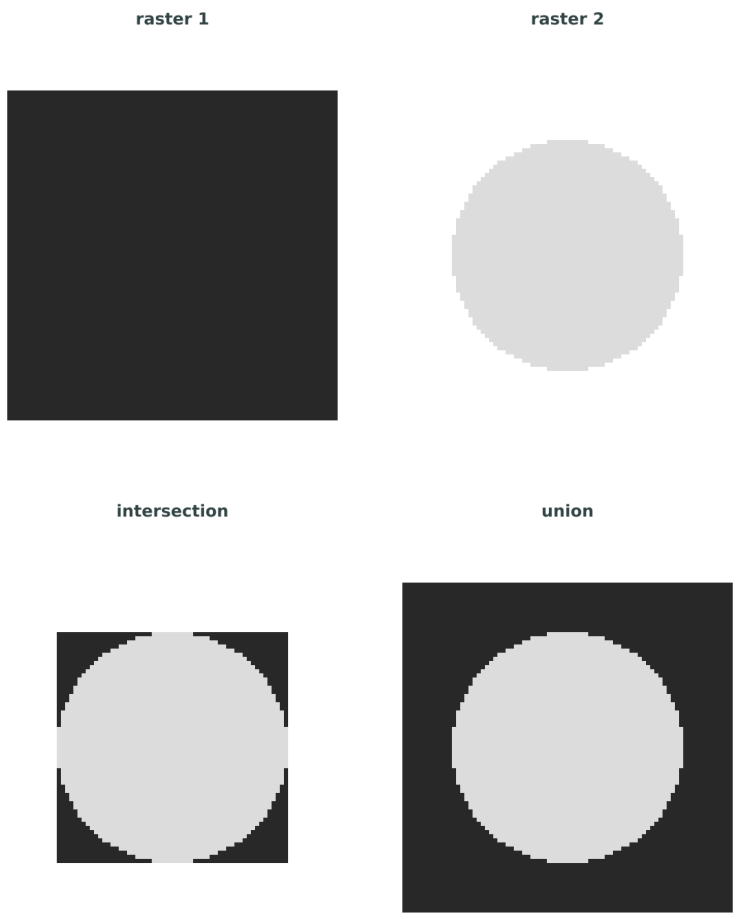
SELECT  ST_MapAlgebraExpr(area.rast, bub.rast, '[rast2.val]', '8BUI', 'INTERSECTION', '[ ↩
    rast2.val]', '[rast1.val]') As interrast,
        ST_MapAlgebraExpr(area.rast, bub.rast, '[rast2.val]', '8BUI', 'UNION', '[rast2.val]' ↩
    , '[rast1.val]') As unionrast
FROM
  (SELECT rast FROM fun_shapes WHERE fun_name = 'area') As area
CROSS JOIN
  (SELECT rast FROM fun_shapes WHERE fun_name = 'rand bubbles') As bub
```

The same intersection and union operations using two deterministic, self-contained rasters.

Code

```
WITH canvas AS (
    SELECT ST_AddBand(
        ST_MakeEmptyRaster(80, 80, 0, 80, 1, -1, 0, 0, 0),
        1, '8BUI', 40, 0
    ) AS rast
), inputs AS (
    SELECT
        rast AS raster_1,
        ST_AsRaster(
            ST_Buffer(ST_Point(40, 40), 28),
            rast,
            '8BUI', 220, 0
        ) AS raster_2
    FROM canvas
), variants AS (
    SELECT 'raster 1' AS title, raster_1 AS rendered FROM inputs
    UNION ALL
    SELECT 'raster 2', raster_2 FROM inputs
    UNION ALL
    SELECT 'intersection', ST_MapAlgebraExpr(
        raster_1, raster_2,
        '[rast2.val]', '8BUI', 'INTERSECTION',
        '[rast2.val]', '[rast1.val]'
    ) FROM inputs
    UNION ALL
    SELECT 'union', ST_MapAlgebraExpr(
        raster_1, raster_2,
        '[rast2.val]', '8BUI', 'UNION',
        '[rast2.val]', '[rast1.val]'
    ) FROM inputs
)
SELECT title, ST_AsPNG(rendered) AS image
FROM variants
ORDER BY CASE title
    WHEN 'raster 1' THEN 1
    WHEN 'raster 2' THEN 2
    WHEN 'intersection' THEN 3
    ELSE 4
END;
```

Figure



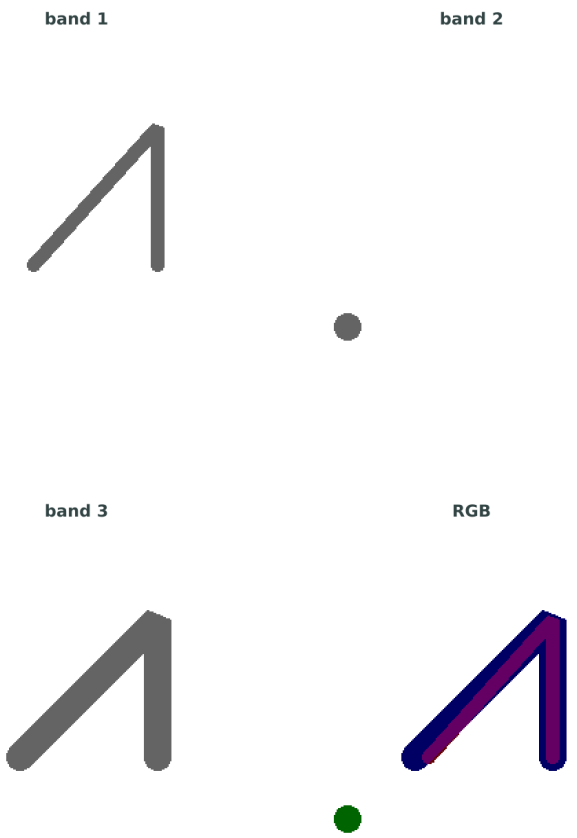
Overlaying rasters on a one-unit-per-pixel canvas as separate bands. We use ST_AsPNG to render the image so all single band ones look grey.

Code

```
WITH mygeoms AS (
    SELECT 2 AS bnum, ST_Buffer(ST_Point(1, 5), 10) AS geom
    UNION ALL
    SELECT 3,
        ST_Buffer(
            ST_GeomFromText('LINESTRING(50 50,150 150,150 50)'),
            10,
            'join=bevel'
        )
    UNION ALL
    SELECT 1,
        ST_Buffer(
            ST_GeomFromText('LINESTRING(60 50,150 150,150 50)'),
            5,
            'join=bevel'
        )
), canvas AS (
    SELECT ST_AddBand(
        ST_MakeEmptyRaster(
```

```
            200,
            200,
            ST_XMin(e)::integer,
            ST_YMax(e)::integer,
            1, -1, 0, 0),
        '8BUI'::text,
        0
    ) AS rast
    FROM (
      SELECT ST_Extent(geom) AS e
      FROM mygeoms
    ) AS bounds
), rbands AS (
    SELECT ARRAY(
        SELECT ST_MapAlgebraExpr(
            canvas.rast,
            ST_AsRaster(m.geom, canvas.rast, '8BUI', 100),
            '[rast2.val]', '8BUI', 'FIRST',
            '[rast2.val]', '[rast1.val]'
        )
        FROM mygeoms AS m
        CROSS JOIN canvas
        ORDER BY m.bnum
    ) AS rasts
)
SELECT title, ST_AsPNG(rendered) AS image
FROM rbands
CROSS JOIN LATERAL (VALUES
    ('band 1', rasts[1]),
    ('band 2', rasts[2]),
    ('band 3', rasts[3]),
    ('RGB', ST_AddBand(ST_AddBand(rasts[1], rasts[2]), rasts[3]))
) AS variants(title, rendered);
```

Figure



Overlay a two-meter boundary of selected parcels over aerial imagery. The query first unions the parcel geometries, clips the source raster shards to the region, unions those smaller shards, and finally adds the first two bands plus a third-band map algebra overlay.

Create a new three-band raster composed of the first two clipped bands and the third band overlaid with the geometry. This query took 3.6 seconds on a 64-bit PostGIS Windows installation.

Code

```
WITH pr AS (
    SELECT ST_Clip(rast, ST_Expand(geom, 50)) AS rast, g.geom
    FROM aerials.o_2_boston AS r
    INNER JOIN (
        SELECT ST_Union(ST_Transform(geom, 26986)) AS geom
        FROM landparcels
        WHERE pid IN ('0303890000', '0303900000')
    ) AS g
        ON ST_Intersects(rast::geometry, ST_Expand(g.geom, 50))
), prunion AS (
    SELECT ST_AddBand(
        NULL,
        ARRAY[
            ST_Union(rast, 1),
            ST_Union(rast, 2),
```

```
            ST_Union(rast, 3)
        ]
    ) AS clipped,
    geom
    FROM pr
    GROUP BY geom
)
SELECT ST_AddBand(
    ST_Band(clipped, ARRAY[1, 2]),
    ST_MapAlgebraExpr(
     ST_Band(clipped, 3),
     ST_AsRaster(ST_Buffer(ST_Boundary(geom), 2), clipped, '8BUI', 250),
     '[rast2.val]',
     '8BUI',
     'FIRST',
     '[rast2.val]',
     '[rast1.val]') ) As rast
FROM prunion;
```

Parcel-boundary overlay.



The blue lines are the boundaries of selected parcels

The generated four-panel canvas example above shows the same band-overlay technique without requiring an external aerial-imagery or parcel table.

Siehe auch

ST_MapAlgebraExpr, ST_AddBand, ST_AsPNG, ST_AsRaster, ST_MapAlgebraFct, ST_BandPixelType, ST_GeoReference, ST_Value, ST_Union, ST_Union

11.12.9 ST_MapAlgebraFct

ST_MapAlgebraFct — Version mit 1 Rasterband: Erzeugt ein neues Rasterband, dass über eine gültige PostgreSQL Funktion für ein gegebenes Rasterband und Pixeltyp erstellt wird. Wenn kein Band bestimmt ist, wird Band 1 angenommen.

Synopsis

raster **ST_MapAlgebraFct**(raster rast, regprocedure onerasteruserfunc);
raster **ST_MapAlgebraFct**(raster rast, regprocedure onerasteruserfunc, text[] VARIADIC args);
raster **ST_MapAlgebraFct**(raster rast, text pixeltype, regprocedure onerasteruserfunc);
raster **ST_MapAlgebraFct**(raster rast, text pixeltype, regprocedure onerasteruserfunc, text[] VARIADIC args);
raster **ST_MapAlgebraFct**(raster rast, integer band, regprocedure onerasteruserfunc);
raster **ST_MapAlgebraFct**(raster rast, integer band, regprocedure onerasteruserfunc, text[] VARIADIC args);
raster **ST_MapAlgebraFct**(raster rast, integer band, text pixeltype, regprocedure onerasteruserfunc);
raster **ST_MapAlgebraFct**(raster rast, integer band, text pixeltype, regprocedure onerasteruserfunc, text[] VARIADIC args);

Description



Warnung
ST_MapAlgebraFct Seit der Version 2.1.0 überholt. Benutzen Sie stattdessen bitte ST_MapAlgebra (callback function version)

Erstellt einen neuen Raster mit einem Band, indem eine gültige PostgreSQL Funktion durch `tworastuserfunc` spezifiziert und auf den gegebenen Raster (`rast`) angewendet wird. Wenn kein `Band` angegeben ist, wird Band 1 angenommen. Der Zielraster hat die selbe Georeferenzierung, Breite und Höhe wie der ursprüngliche Raster, hat aber nur ein Band.

Wenn `pixeltype` angegeben ist, dann hat das Band des neuen Rasters diesen Pixeltyp. Wenn für den Pixeltyp NULL übergeben wird, dann hat das neue Rasterband denselben Pixeltyp wie das gegebene Band von `rast`

Der Parameter `onerasteruserfunc` muss der Name und die Signatur einer SQL- oder PL/pgSQL-Funktion sein, die in eine regprocedure umgewandelt wird. Ein sehr einfaches und ziemlich nutzloses PL/pgSQL-Funktionsbeispiel ist:

Code

```
CREATE OR REPLACE FUNCTION simple_function(pixel FLOAT, pos INTEGER[], VARIADIC args TEXT[])
RETURNS FLOAT
AS $$ BEGIN
    RETURN 0.0;
END; $$
LANGUAGE 'plpgsql' IMMUTABLE;
```

Die Benutzerfunktion kann zwei oder drei Argumente akzeptieren: einen Float-Wert, ein optionales Integer-Array und ein variables Text-Array. Das erste Argument ist der Wert einer einzelnen Rasterzelle (unabhängig vom Rasterdatentyp). Das zweite Argument ist die Position der aktuellen Verarbeitungszelle in der Form '{x,y}'. Das dritte Argument gibt an, dass alle übrigen Parameter an ST_MapAlgebraFct an die Benutzerfunktion weitergegeben werden sollen.

Die Übergabe eines regprodedure Arguments an eine SQL-Funktion erfordert die Übergabe der vollständigen Funktionssignatur und die anschließende Umwandlung in einen regprocedure Typ. Um die obige Beispiel-PL/pgSQL-Funktion als Argument zu übergeben, lautet das SQL für das Argument:

Code

```
'simple_function(float,integer[],text[])'::regprocedure
```

Beachten Sie, dass das Argument den Namen der Funktion, die Typen der Funktionsargumente, Anführungszeichen um den Namen und die Argumenttypen sowie einen Cast in eine regprocedure enthält.

Der dritte Übergabewert an die Funktion `userfunction` ist ein Feld vom Datentyp variadic text. Alle an einen Funktionsaufruf von ST_MapAlgebraFct angehängten Parameter werden an die spezifizierte `userfunction` durchgereicht und sind in dem Übergabewert `args` enthalten.



Anmerkung
Für nähere Information zu dem Schlüsselwort VARIADIC, sehen Sie bitte die PostgreSQL Dokumentation und den Abschnitt "SQL Functions with Variable Numbers of Arguments" unter Query Language (SQL) Functions.



Anmerkung
Der Übergabewert text[] an die `userfunction` ist auch dann erforderlich, wenn Sie sonst keine Argumente für die Auswertung an an "userfunction" übergeben.

Verfügbarkeit: 2.0.0

Beispiele

Erzeugt einen Einzelbandraster, der sich durch die Anwendung der Funktion "modulo 2" auf das ursprüngliche Rasterband ergibt.

Code

```
ALTER TABLE dummy_rast ADD COLUMN map_rast raster;
CREATE FUNCTION mod_fct(pixel float, pos integer[], variadic args text[])
RETURNS float
AS $$
BEGIN
    RETURN pixel::integer % 2;
END;
$$
LANGUAGE 'plpgsql' IMMUTABLE;

UPDATE dummy_rast SET map_rast = ST_MapAlgebraFct(rast, NULL, 'mod_fct(float,integer[],text ↩
    [])'::regprocedure) WHERE rid = 2;

SELECT ST_Value(rast, 1, i, j) As origval, ST_Value(map_rast, 1, i, j) As mapval
FROM dummy_rast CROSS JOIN generate_series(1, 3) AS i CROSS JOIN generate_series(1, 3) AS j
WHERE rid = 2;
```

Output

```
origval | mapval
---------+--------
     253 |      1
     254 |      0
     253 |      1
     253 |      1
     254 |      0
     254 |      0
     250 |      0
     254 |      0
     254 |      0
```

Erzeugt einen neuen Raster mit einem Band und mit dem Pixeltyp 2BUI. Der ursprüngliche Raster wird dabei neu klassifiziert und der NODATA Wert wird an die "user function" (0) übergeben.

Code

```
ALTER TABLE dummy_rast ADD COLUMN map_rast2 raster;
CREATE FUNCTION classify_fct(pixel float, pos integer[], variadic args text[])
RETURNS float
AS
$$
DECLARE
    nodata float := 0;
```

```
BEGIN
    IF NOT args[1] IS NULL THEN
        nodata := args[1];
    END IF;
    IF pixel < 251 THEN
        RETURN 1;
    ELSIF pixel = 252 THEN
        RETURN 2;
    ELSIF pixel
> 252 THEN
        RETURN 3;
    ELSE
        RETURN nodata;
    END IF;
END;
$$
LANGUAGE 'plpgsql';
UPDATE dummy_rast SET map_rast2 = ST_MapAlgebraFct(rast,'2BUI','classify_fct(float,integer ↩
    [],text[])'::regprocedure, '0') WHERE rid = 2;

SELECT DISTINCT ST_Value(rast,1,i,j) As origval, ST_Value(map_rast2, 1, i, j) As mapval
FROM dummy_rast CROSS JOIN generate_series(1, 5) AS i CROSS JOIN generate_series(1,5) AS j
WHERE rid = 2;
```

Output

```
 origval | mapval
---------+--------
     249 |      1
     250 |      1
     251 |
     252 |      2
     253 |      3
     254 |      3
```

Code

```
SELECT ST_BandPixelType(map_rast2) As b1pixtyp
FROM dummy_rast WHERE rid = 2;
```

Output

```
 b1pixtyp
----------
 2BUI
```

Erzeugt einen neuen Raster mit 3 Bändern und demselben Pixeltyp als unser ursprünglicher Raster mit 3 Bändern. Das erste Band wird über einen Map Algebra Ausdruck geändert, die verbleibenden 2 Bänder sind unverändert.

Code

```
CREATE FUNCTION rast_plus_tan(pixel float, pos integer[], variadic args text[])
RETURNS float
AS
$$
BEGIN
    RETURN tan(pixel) * pixel;
END;
$$
LANGUAGE 'plpgsql';

SELECT ST_AddBand(ST_AddBand(ST_AddBand(ST_MakeEmptyRaster(rast_view),
            ST_MapAlgebraFct(rast_view, 1, NULL, 'rast_plus_tan(float,integer[],text[])':: ↩
    regprocedure)
```

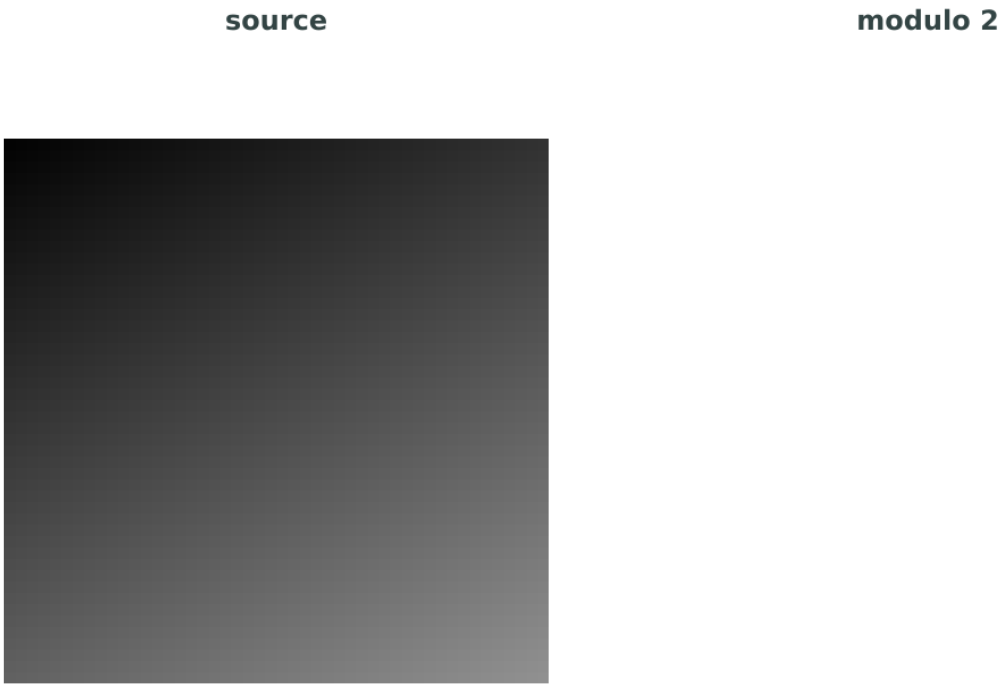
```
        ),
        ST_Band(rast_view, 2)
    ),
    ST_Band(rast_view, 3) As rast_view_ma
)
FROM wind
WHERE rid=167;
```

The deprecated callback form above is equivalent to a modern ST_MapAlgebra (callback function version) expression for simple per-pixel transforms. This build-time example renders the original raster and a modulo-2 classification.

Code

```
WITH pixels AS (
    SELECT x, y, (x + y * 2)::double precision AS value
    FROM generate_series(1, 48) AS x
    CROSS JOIN generate_series(1, 48) AS y
), rows AS (
    SELECT y, array_agg(value ORDER BY x) AS values
    FROM pixels
    GROUP BY y
), source AS (
    SELECT ST_SetValues(
        ST_AddBand(
            ST_MakeEmptyRaster(48, 48, 0, 48, 1, -1, 0, 0, 0),
            1, '8BUI', 0, 0
        ),
        1, 1, 1, array_agg(values ORDER BY y)::double precision[][]
    ) AS rast
    FROM rows
), variants AS (
    SELECT 'source' AS title, rast AS rendered FROM source
    UNION ALL
    SELECT 'modulo 2', ST_MapAlgebra(
        rast, 1,
        '8BUI',
        'mod([rast]::integer,2) * 255'
    )
    FROM source
)
SELECT title, ST_AsPNG(rendered) AS image
FROM variants
ORDER BY title DESC;
```

Figure



Siehe auch

ST_MapAlgebraExpr, ST_BandPixelType, ST_GeoReference, ST_SetValue

11.12.10 ST_MapAlgebraFct

ST_MapAlgebraFct — Version mit 2 Rasterbändern: Erstellt einen neuen Einzelbandraster, indem eine gültige PostgreSQL Funktion auf die 2 gegebenen Rasterbänder und den entsprechenden Pixeltyp angewendet wird. Wenn kein Band bestimmt ist, wird Band 1 angenommen. Wenn der "Extent"-Typ nicht angegeben ist, wird standardmäßig INTERSECTION angenommen.

Synopsis

raster **ST_MapAlgebraFct**(raster rast1, raster rast2, regprocedure tworastuserfunc, text pixeltype=same_as_rast1, text extenttype=INTERSECTION, text[] VARIADIC userargs);
raster **ST_MapAlgebraFct**(raster rast1, integer band1, raster rast2, integer band2, regprocedure tworastuserfunc, text pixeltype=same_as_rast1, text extenttype=INTERSECTION, text[] VARIADIC userargs);

Description



Warnung
ST_MapAlgebraFct Seit der Version 2.1.0 überholt. Benutzen Sie stattdessen bitte ST_MapAlgebra (callback function version)

Erstellt einen neuen Raster mit einem Band, indem eine gültige PostgreSQL Funktion (`tworastuserfunc`) auf die Ausgangsraster (`rast1`, `rast2`) angewendet wird. Wenn `band1` oder `band2` nicht angegeben ist, wird Band 1 angenommen. Der Zielraster hat die selbe Georeferenzierung, Breite und Höhe wie der ursprüngliche Raster, hat aber nur ein Band.

Wenn `pixeltype` angegeben ist, dann hat das Band des neuen Rasters diesen Pixeltyp. Wenn für den Pixeltyp NULL oder kein Wert übergeben wird, dann hat das neue Rasterband denselben Pixeltyp wie das angegebene Band von `rast1`

Der Parameter `tworastuserfunc` muss der Name und die Signatur einer SQL- oder PL/pgSQL-Funktion sein, die in eine regprocedure umgewandelt wird. Ein Beispiel für eine PL/pgSQL-Funktion ist:

Code

```
CREATE OR REPLACE FUNCTION simple_function_for_two_rasters(pixel1 FLOAT, pixel2 FLOAT, pos ↩
    INTEGER[], VARIADIC args TEXT[])
RETURNS FLOAT
AS $$ BEGIN
    RETURN 0.0;
END; $$
LANGUAGE 'plpgsql' IMMUTABLE;
```

Die Funktion `tworastuserfunc` kann drei oder vier Argumente akzeptieren: einen Wert mit doppelter Genauigkeit, einen Wert mit doppelter Genauigkeit, ein optionales Integer-Array und ein variables Text-Array. Das erste Argument ist der Wert einer einzelnen Rasterzelle in `rast1` (unabhängig vom Rasterdatentyp). Das zweite Argument ist ein individueller Rasterzellenwert in `rast2`. Das dritte Argument ist die Position der aktuellen Verarbeitungszelle in der Form '{x,y}'. Das vierte Argument gibt an, dass alle übrigen Parameter von ST_MapAlgebraFct an die `tworastuserfunc` weitergereicht werden sollen.

Die Übergabe eines regprodedure Arguments an eine SQL-Funktion erfordert die Übergabe der vollständigen Funktionssignatur und die anschließende Umwandlung in einen regprocedure Typ. Um die obige Beispiel-PL/pgSQL-Funktion als Argument zu übergeben, lautet das SQL für das Argument:

Code

```
'simple_function(double precision, double precision, integer[], text[])'::regprocedure
```

Beachten Sie, dass das Argument den Namen der Funktion, die Typen der Funktionsargumente, Anführungszeichen um den Namen und die Argumenttypen sowie einen Cast in eine regprocedure enthält.

Der vierte Übergabewert an die Funktion `tworastuserfunc` ist ein Feld mit dem Datentyp variadic text. Alle an eine Funktion ST_MapAlgebraFct angehängten Parameter werden an die `tworastuserfunc` durchgereicht und sind in dem Übergabewert `userargs` enthalten.



Anmerkung
Für nähere Information zu dem Schlüsselwort VARIADIC, sehen Sie bitte die PostgreSQL Dokumentation und den Abschnitt "SQL Functions with Variable Numbers of Arguments" unter Query Language (SQL) Functions.



Anmerkung
Der Übergabewert text[] an die `tworastuserfunc` ist auch dann erforderlich, wenn Sie sonst keine Argumente für die Auswertung an an "userfunction" übergeben.

Verfügbarkeit: 2.0.0

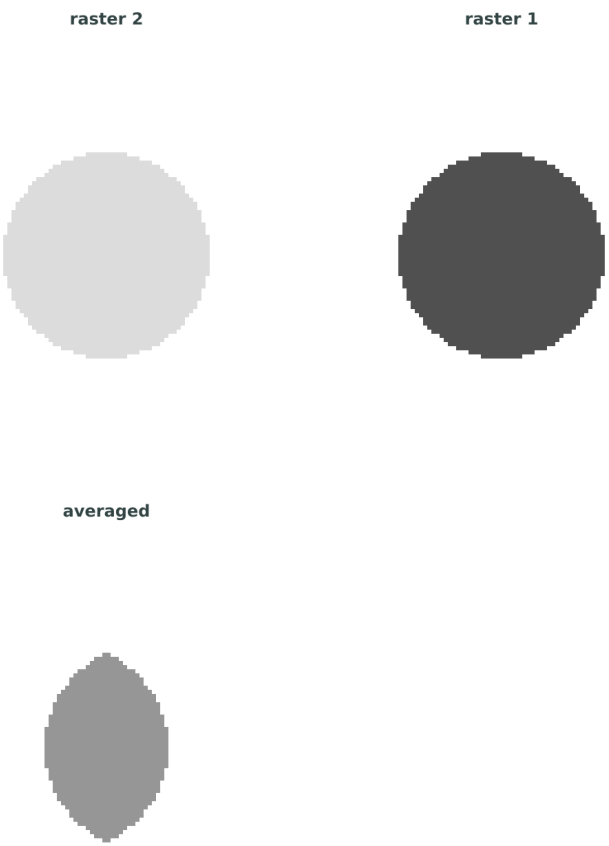
Beispiel: Überlagerung von Rastern als Einzelbänder am Canvas

The deprecated two-raster callback form can often be expressed with the modern two-raster ST_MapAlgebra (callback function version) expression variant. This build-time example renders two input rasters and their averaged overlap.

Code

```
WITH canvas AS (
    SELECT ST_AddBand(
        ST_MakeEmptyRaster(80, 80, 0, 80, 1, -1, 0, 0, 0),
        1, '8BUI', 0, 0
    ) AS rast
), inputs AS (
    SELECT
        ST_AsRaster(ST_Buffer(ST_Point(30, 40), 25), rast, '8BUI', 80, 0)
            AS raster_1,
        ST_AsRaster(ST_Buffer(ST_Point(50, 40), 25), rast, '8BUI', 220, 0)
            AS raster_2
    FROM canvas
), variants AS (
    SELECT 'raster 1' AS title, raster_1 AS rendered FROM inputs
    UNION ALL
    SELECT 'raster 2', raster_2 FROM inputs
    UNION ALL
    SELECT 'averaged', ST_MapAlgebra(
        raster_1, 1,
        raster_2, 1,
        '([rast1] + [rast2]) / 2',
        '8BUI',
        'UNION'
    ) FROM inputs
)
SELECT title, ST_AsPNG(rendered) AS image
FROM variants
ORDER BY title DESC;
```

Figure



Define our user defined function.

Code

```
CREATE OR REPLACE FUNCTION raster_mapalgebra_union(
    rast1 double precision,
    rast2 double precision,
    pos integer[],
    VARIADIC userargs text[]
)
    RETURNS double precision
    AS $$
    DECLARE
    BEGIN
        CASE
            WHEN rast1 IS NOT NULL AND rast2 IS NOT NULL THEN
                RETURN ((rast1 + rast2)/2.);
            WHEN rast1 IS NULL AND rast2 IS NULL THEN
                RETURN NULL;
            WHEN rast1 IS NULL THEN
                RETURN rast2;
            ELSE
                RETURN rast1;
```

```
        END CASE;

        RETURN NULL;
    END;
    $$ LANGUAGE 'plpgsql' IMMUTABLE COST 1000;
```

Prepare a test table of rasters.

Code

```
DROP TABLE IF EXISTS map_shapes;
CREATE TABLE map_shapes(rid serial PRIMARY KEY, rast raster, bnum integer, descrip text);
INSERT INTO map_shapes(rast,bnum, descrip)
WITH mygeoms
    AS ( SELECT 2 As bnum, ST_Buffer(ST_Point(90,90),30) As geom, 'circle' As descrip
            UNION ALL
            SELECT 3 AS bnum,
                ST_Buffer(ST_GeomFromText('LINESTRING(50 50,150 150,150 50)'), 15) As geom,
    'big road' As descrip
            UNION ALL
            SELECT 1 As bnum,
                ST_Translate(ST_Buffer(ST_GeomFromText('LINESTRING(60 50,150 150,150 50)'),
    8,'join=bevel'), 10,-6) As geom, 'small road' As descrip
            ),
   -- define our canvas to be 1 to 1 pixel to geometry
   canvas
    AS ( SELECT ST_AddBand(
        ST_MakeEmptyRaster(
            250,
            250,
            ST_XMin(e)::integer,
            ST_YMax(e)::integer,
            1, -1, 0, 0),
        '8BUI'::text,
        0) As rast
        FROM (SELECT ST_Extent(geom) As e,
                    Max(ST_SRID(geom)) As srid
                    from mygeoms
                    ) As foo
            )
-- return our rasters aligned with our canvas
SELECT ST_AsRaster(m.geom, canvas.rast, '8BUI', 240) As rast, bnum, descrip
                FROM mygeoms AS m CROSS JOIN canvas
UNION ALL
SELECT canvas.rast, 4, 'canvas'
FROM canvas;
```

Map algebra on single-band rasters and then collect the result with ST_AddBand.

Code

```
INSERT INTO map_shapes(rast,bnum,descrip)
SELECT ST_AddBand(ST_AddBand(rasts[1], rasts[2]),rasts[3]), 4, 'map bands overlay fct union
    (canvas)'
    FROM (SELECT ARRAY(SELECT ST_MapAlgebraFct(
            m1.rast,
            m2.rast,
            'raster_mapalgebra_union(double precision, double precision, integer[], text[])'
    ::regprocedure,
            '8BUI',
            'FIRST')
                FROM map_shapes As m1 CROSS JOIN map_shapes As m2
```

```
    WHERE m1.descrip = 'canvas' AND m2.descrip <
> 'canvas' ORDER BY m2.bnum) As rasts) As foo;
```

A compact, executable callback example. The source rasters and their averaged overlap are rendered at build time.

Code

```
CREATE OR REPLACE FUNCTION pg_temp.raster_average(
    value_1 double precision,
    value_2 double precision,
    position integer[],
    VARIADIC userargs text[]
) RETURNS double precision
LANGUAGE SQL IMMUTABLE
AS 'SELECT COALESCE(($1 + $2) / 2.0, $1, $2)';

WITH canvas AS (
    SELECT ST_AddBand(
        ST_MakeEmptyRaster(80, 80, 0, 80, 1, -1, 0, 0, 0),
        1, '8BUI', 0, 0
    ) AS rast
), inputs AS (
    SELECT
        ST_AsRaster(ST_Buffer(ST_Point(30, 40), 25), rast, '8BUI', 80, 0)
            AS raster_1,
        ST_AsRaster(ST_Buffer(ST_Point(50, 40), 25), rast, '8BUI', 220, 0)
            AS raster_2
    FROM canvas
), variants AS (
    SELECT 'raster 1' AS title, raster_1 AS rendered FROM inputs
    UNION ALL
    SELECT 'raster 2', raster_2 FROM inputs
    UNION ALL
    SELECT 'averaged', ST_MapAlgebraFct(
        raster_1,
        raster_2,
        'pg_temp.raster_average(double precision,double precision,integer[],text[])':: ↩
    regprocedure,
        '8BUI',
        'UNION'
    ) FROM inputs
)
SELECT title, ST_AsPNG(rendered) AS image
FROM variants
ORDER BY title DESC;
```

Output

```
title    | image
---------+---------------------------
 raster 2 | PNG image, 50 x 50 pixels
 raster 1 | PNG image, 50 x 50 pixels
 averaged | PNG image, 70 x 50 pixels
```

Eine benutzerdefinierte Funktion, die zusätzliche Übergabewerte entgegennimmt

Code

```
CREATE OR REPLACE FUNCTION raster_mapalgebra_userargs(
    rast1 double precision,
    rast2 double precision,
    pos integer[],
    VARIADIC userargs text[]
)
    RETURNS double precision
    AS $$
    DECLARE
    BEGIN
        CASE
            WHEN rast1 IS NOT NULL AND rast2 IS NOT NULL THEN
                RETURN least(userargs[1]::integer,(rast1 + rast2)/2.);
            WHEN rast1 IS NULL AND rast2 IS NULL THEN
                RETURN userargs[2]::integer;
            WHEN rast1 IS NULL THEN
                RETURN greatest(rast2, random()*userargs[3]::integer)::integer;
            ELSE
                RETURN greatest(rast1, random()*userargs[4]::integer)::integer;
        END CASE;

        RETURN NULL;
    END;
    $$ LANGUAGE 'plpgsql' VOLATILE COST 1000;

SELECT ST_MapAlgebraFct(
    m1.rast,
    1,
    m1.rast,
    3,
    'raster_mapalgebra_userargs(double precision, double precision, integer[], text[])':: ↩
    regprocedure,
    '8BUI',
    'INTERSECT',
    '100', '200', '200', '0')
                FROM map_shapes As m1
    WHERE m1.descrip = 'map bands overlay fct union (canvas)';
```

The generated callback figure above demonstrates the raster alignment and overlap used by this variant; the exact callback result remains available as text.

Siehe auch

ST_MapAlgebraExpr, ST_BandPixelType, ST_GeoReference, ST_SetValue

11.12.11 ST_MapAlgebraFctNgb

ST_MapAlgebraFctNgb — Version mit 1em Band: Map Algebra Nearest Neighbor mit einer benutzerdefinierten PostgreSQL Funktion. Gibt einen Raster zurück, dessen Werte sich aus einer benutzerdefinierte PL/pgsql Funktion ergeben, welche die Nachbarschaftswerte des Ausgangsrasterbandes einbezieht.

Synopsis

raster **ST_MapAlgebraFctNgb**(raster rast, integer band, text pixeltype, integer ngbwidth, integer ngbheight, regprocedure onerastngbuserfunc, text nodatamode, text[] VARIADIC args);

Description



Warnung
ST_MapAlgebraFctNgb Seit der Version 2.1.0 überholt. Benutzen Sie stattdessen bitte ST_MapAlgebra (callback function version)

(Version mit einem Raster) Gibt einen Raster zurück, dessen Werte sich aus einer benutzerdefinierte PL/pgSQL Funktion errechnen, welche die Nachbarschaftswerte des Ausgangsrasterbandes mit einbezieht. Die benutzerdefinierte Funktion nimmt die Werte der Nachbarschaftspixel als Zahlenfeld entgegen und gibt für jedes Pixel das Ergebnis der Funktion zurück, indem die aktuellen Werte der betrachteten Pixel mit dem Ergebnis der benutzerdefinierten Funktion ersetzt werden.

rast Raster der durch die benutzerdefinierte Funktion ausgewertet werden soll.

band Die Bandnummer des Raster, die ausgewertet werden soll. Die Standardeinstellung ist 1.

pixeltype Der resultierende Pixeltyp des Zielraster muss entweder aus ST_BandPixelType sein, weggelassen oder auf NULL gesetzt werden. Wenn er nicht übergeben wird oder auf NULL gesetzt ist, wird er standardmäßig auf den Pixeltyp von `rast` gesetzt. Die Ergebnisse werden abgeschnitten, wenn sie größer als für den Pixeltyp erlaubt sind.

ngbwidth Die Breite der Nachbarschaft in Zellen.

ngbheight Die Höhe der Nachbarschaft in Zellen.

onerastngbuserfunc Eine benutzerdefinierte Funktion in PLPGSQL/psql, die auf die Nachbarschaftspixel in einem einzelnen Rasterband angewandt wird. Das erste Element ist ein 2-dimensionales Feld mit Zahlen, welche das Rechteck mit den Nachbarschaftspixel beschreiben

nodatamode Bestimmt welcher Wert an die Funktion übergeben werden soll, wenn ein Nachbarschaftspixel NODATA oder NULL ist

'ignore': NODATA Werte in der Nachbarschaft werden bei der Berechnung ignoriert -- diese Flag muss an die Rückruffunktion gesendet werden und die benutzerdefinierte Funktion entscheidet dann, wie diese Werte ignoriert werden sollen.

'NULL': NODATA Werte in der Nachbarschaft bedingen, dass die resultierenden Pixel ebenfalls NULL annehmen - die benutzerdefinierte Rückruffunktion wird bei diesem Fall übersprungen.

'value': NODATA Werte in der Nachbarschaft werden durch den Wert des Referenzpixel (das Pixel im Zentrum der Nachbarschaft) ersetzt. Anmerkung: Wenn dieser Wert NODATA ist, dann ist die Verhaltensweise gleich wie bei 'NULL' (für die betroffene Nachbarschaft)

args Übergabewerte für die benutzerdefinierte Funktion.

Verfügbarkeit: 2.0.0

Beispiele

Die Beispiele nutzen den Raster "Katrina", der als einzelne Kachel geladen wird, wie unter https://gdal.org/user/drivers/raster/-postgisraster.html beschrieben; und in den Beispielen von ST_Rescale aufbereitet wurde.

A simple 'callback' user function that averages up all the values in a neighborhood.

Code

```
CREATE OR REPLACE FUNCTION rast_avg(matrix float[][], nodatamode text, variadic args text[])
    RETURNS float AS
    $$
    DECLARE
        _matrix float[][];
        x1 integer;
```

```
        x2 integer;
        y1 integer;
        y2 integer;
        sum float;
    BEGIN
        _matrix := matrix;
        sum := 0;
        FOR x in array_lower(matrix, 1)..array_upper(matrix, 1) LOOP
            FOR y in array_lower(matrix, 2)..array_upper(matrix, 2) LOOP
                sum := sum + _matrix[x][y];
            END LOOP;
        END LOOP;
        RETURN (sum*1.0/(array_upper(matrix,1)*array_upper(matrix,2) ))::integer ;
    END;
    $$
LANGUAGE 'plpgsql' IMMUTABLE COST 1000;
```

Apply the raster averaging callback within two pixels of each input pixel in the X and Y directions.

Code

```
SELECT ST_MapAlgebraFctNgb(
        rast,
        1,
        '8BUI',
        4,
        4,
        'rast_avg(float[][], text, text[])'::regprocedure,
        'NULL',
        NULL) As nn_with_border
    FROM katrinas_rescaled
    limit 1;
```

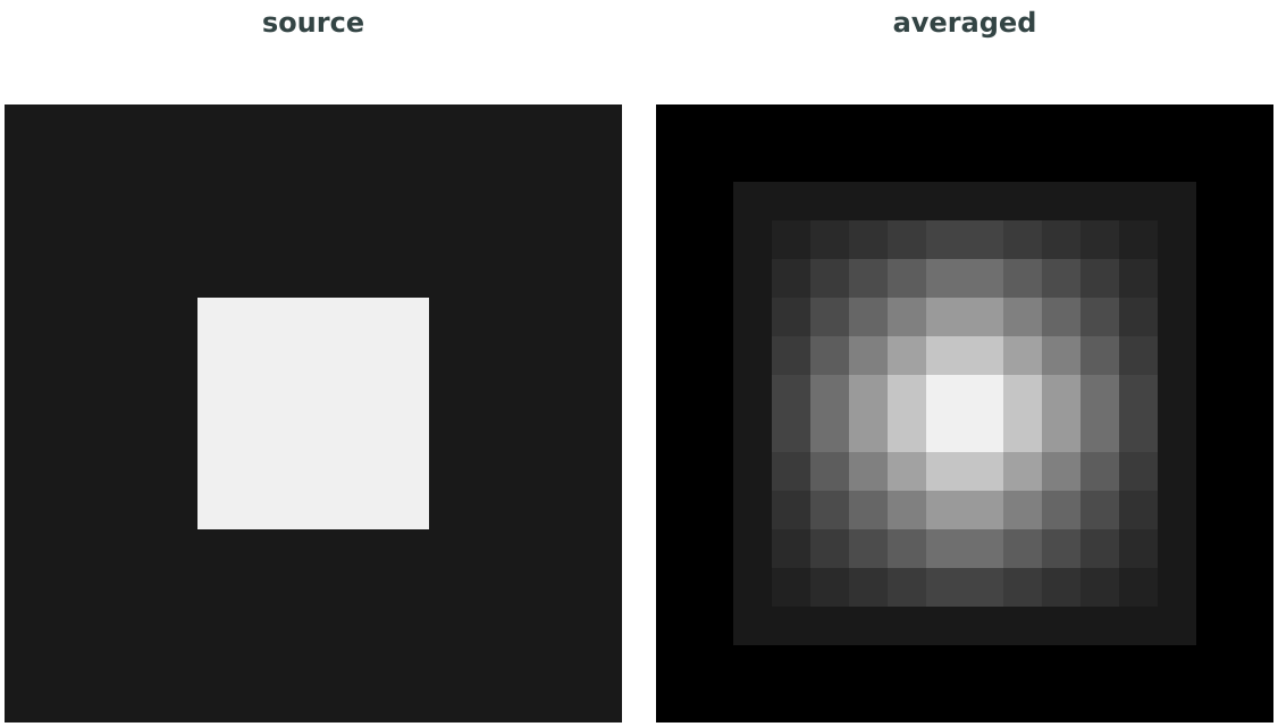
A self-contained neighborhood average. The source raster has a sharp central square, while the result spreads those values across neighboring pixels. The source uses a non-NODATA background so `nodatamode ignore` averages background values with the square values, instead of treating the background as absent.

Code

```
WITH source AS (
    SELECT ST_SetValues(
        ST_AddBand(
            ST_MakeEmptyRaster(16, 16, 0, 16, 1, -1, 0, 0, 0),
            1, '8BUI', 25, NULL
        ),
        1, 6, 6,
        ARRAY[
            [240, 240, 240, 240, 240, 240],
            [240, 240, 240, 240, 240, 240],
            [240, 240, 240, 240, 240, 240],
            [240, 240, 240, 240, 240, 240],
            [240, 240, 240, 240, 240, 240],
            [240, 240, 240, 240, 240, 240]
        ]::double precision[][]
    ) AS rast
), variants AS (
    SELECT 'source' AS title, rast AS rendered FROM source
    UNION ALL
    SELECT 'averaged', ST_SetBandNoDataValue(ST_MapAlgebraFctNgb(
        rast,
```

```
        1,
        '8BUI',
        2,
        2,
        'ST_Mean4ma(float[][],text,text[])'::regprocedure,
        'ignore',
        NULL
    ), NULL) FROM source
)
SELECT title, ST_AsPNG(rendered) AS image
FROM variants
ORDER BY title DESC;
```

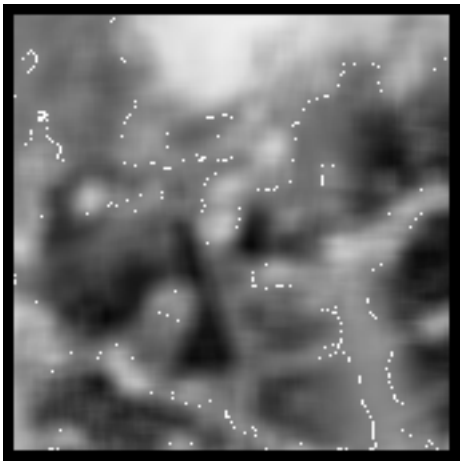
Figure



Neighborhood averaging.



Das erste Band Unseres Rasters



Averaged result

Siehe auch

ST_MapAlgebraFct, ST_MapAlgebraExpr, ST_Rescale

11.12.12 ST_Reclass

ST_Reclass — Erstellt einen neuen Raster, der aus neu klassifizierten Bändern des Originalraster besteht. Das Band "nband" ist jenes das verändert werden soll. Wenn "nband" nicht angegeben ist, wird "Band 1" angenommen. Alle anderen Bänder bleiben unverändert. Anwendungsfall: zwecks einfacherer Visualisierung ein 16BUI-Band in ein 8BUI-Band konvertieren und so weiter.

Synopsis

raster **ST_Reclass**(raster rast, integer nband, text reclassexpr, text pixeltype, double precision nodataval=NULL);
raster **ST_Reclass**(raster rast, reclassarg[] VARIADIC reclassargset);
raster **ST_Reclass**(raster rast, text reclassexpr, text pixeltype);

Description

Creates a new raster formed by applying a reclassification operation defined by the `reclassexpr` on the input raster (`rast`). Refer to reclassarg for the description of reclassification expressions. If no `band` is specified band 1 is assumed.

The new raster will have the same georeference, width, and height as the original raster. The bands of the new raster have pixel type of `pixeltype`. If `reclassargset` is specified then each `reclassarg` defines the type of the target band. Bands not designated are returned unchanged.

Verfügbarkeit: 2.0.0

Beispiele

Basic.

Erzeugt einen neuen Raster aus dem Original, indem Band 2 von 8BUI auf 4BUI und alle Werte von 101-254 auf NODATA gesetzt werden.

Code

```
ALTER TABLE dummy_rast ADD COLUMN reclass_rast raster;
UPDATE dummy_rast SET reclass_rast = ST_Reclass(rast, 2, '0-87:1-10, 88-100:11-15,  ↩
    101-254:0-0', '4BUI', 0) WHERE rid = 2;

SELECT i as col, j as row, ST_Value(rast, 2, i, j) As origval,
    ST_Value(reclass_rast, 2, i, j) As reclassval,
    ST_Value(reclass_rast, 2, i, j, false) As reclassval_include_nodata
FROM dummy_rast CROSS JOIN generate_series(1, 3) AS i CROSS JOIN generate_series(1, 3) AS j
WHERE rid = 2;
```

Output

```
col | row | origval | reclassval | reclassval_include_nodata
-----+-----+---------+------------+--------------------------
   1 |   1 |      78 |          9 |                         9
   2 |   1 |      98 |         14 |                        14
   3 |   1 |     122 |            |                         0
   1 |   2 |      96 |         14 |                        14
   2 |   2 |     118 |            |                         0
   3 |   2 |     180 |            |                         0
   1 |   3 |      99 |         15 |                        15
   2 |   3 |     112 |            |                         0
   3 |   3 |     169 |            |                         0
```

Advanced using multiple reclassargs.

Erstellt einen neuen Raster aus dem Ursprungsraster, wobei die Bänder 1, 2 und 3 in 1BB, 4BUI und 4BUI konvertiert und neu klassifiziert werden. Verwendet das variadische Argument `reclassarg`, welches eine unbegrenzte Anzahl von "reclassargs" entgegennehmen kann (theoretisch so viele wie Bänder vorhanden sind)

Code

```
UPDATE dummy_rast SET reclass_rast =
    ST_Reclass(
        rast,
        ROW(2, '0-87]:1-10, (87-100]:11-15, (101-254]:0-0', '4BUI', NULL)::reclassarg,
        ROW(1, '0-253]:1, 254:0', '1BB', NULL)::reclassarg,
        ROW(3, '0-70]:1, (70-86:2, [86-150):3, [150-255:4', '4BUI', NULL)::reclassarg
        ) WHERE rid = 2;

SELECT i as col, j as row, ST_Value(rast, 1, i, j) As ov1, ST_Value(reclass_rast, 1, i, j)  ↩
    As rv1,
    ST_Value(rast, 2, i, j) As ov2, ST_Value(reclass_rast, 2, i, j) As rv2,
    ST_Value(rast, 3, i, j) As ov3, ST_Value(reclass_rast, 3, i, j) As rv3
FROM dummy_rast CROSS JOIN generate_series(1, 3) AS i CROSS JOIN generate_series(1, 3) AS j
WHERE rid = 2;
```

Output

```
col | row | ov1 | rv1 | ov2 | rv2 | ov3 | rv3
----+-----+-----+-----+-----+-----+-----+-----
  1 |   1 | 253 |   1 |  78 |   9 |  70 |   1
  2 |   1 | 254 |   0 |  98 |  14 |  86 |   3
  3 |   1 | 253 |   1 | 122 |   0 | 100 |   3
  1 |   2 | 253 |   1 |  96 |  14 |  80 |   2
  2 |   2 | 254 |   0 | 118 |   0 | 108 |   3
  3 |   2 | 254 |   0 | 180 |   0 | 162 |   4
  1 |   3 | 250 |   1 |  99 |  15 |  90 |   3
  2 |   3 | 254 |   0 | 112 |   0 | 108 |   3
  3 |   3 | 254 |   0 | 169 |   0 | 175 |   4
```

Advanced Map a single band 32BF raster to multiple viewable bands.

Erstellung eines neuen Raster mit 3 Bändern (8BUI,8BUI,8BUI darstellbarer Raster) aus einem Raster mit nur einem 32bf-Band

Code

```
ALTER TABLE wind ADD COLUMN rast_view raster;
UPDATE wind
    set rast_view = ST_AddBand(
        NULL,
        ARRAY[
    ST_Reclass(rast, 1, '0.1-10]:1-10,9-10]:11,(11-33:0'::text, '8BUI'::text, 0),
    ST_Reclass(rast, 1, '11-33):0-255,[0-32:0,(34-1000:0'::text, '8BUI'::text, 0),
    ST_Reclass(rast, 1, '0-32]:0,(32-100:100-255'::text, '8BUI'::text, 0)
    ]
    );
```

Siehe auch

ST_AddBand, ST_Band, ST_BandPixelType, ST_MakeEmptyRaster, reclassarg, Abschnitt 10.4.1, ST_Value

11.12.13 ST_ReclassExact

ST_ReclassExact — Creates a new raster composed of bands reclassified from original, using a 1:1 mapping from values in the original band to new values in the destination band.

Synopsis

raster **ST_ReclassExact**(raster rast, double precision[] inputvalues, double precision[] outputvalues, integer bandnumber=1, text pixeltype=32BF, double precision nodatavalue=NULL);

Description

Creates a new raster formed by applying a reclassification operation defined by the `inputvalues` and `outputvalues` arrays. Pixel values found in the input array are mapped to the corresponding value in the output array. All other pixel values are mapped to the `nodatavalue`.

The output pixel type defaults to float, but can be specified using the `pixeltype` parameter. If no `bandnumber` is specified band 1 is assumed.

The new raster will have the same georeference, width, and height as the original raster. Bands not designated are returned unchanged.

Verfügbarkeit: 3.6.0

Example: Basic

Create a small raster and map its pixels to new values.

Create a raster with four pixels: first row 1 2, second row 3 4.

Code

```
CREATE TABLE reclassexact (
    id integer,
    rast raster
);
INSERT INTO reclassexact (id, rast)
SELECT 1, ST_SetValues(ST_AddBand(ST_MakeEmptyRaster(2, -- width in pixels
        2, -- height in pixels
```

```
        0, -- upper-left x-coordinate
        0, -- upper-left y-coordinate
        1, -- pixel size in x-direction
        -1, -- pixel size in y-direction (negative for north-up)
        0, -- skew in x-direction
        0, -- skew in y-direction
        4326  -- SRID (e.g., WGS 84)
      ),
      '32BUI'::text, -- pixel type (e.g., '32BF' for float, '8BUI' for unsigned 8-bit int)
      0.0, -- initial value for the band (e.g., 0.0 or a no-data value)
      -99            -- nodatavalue
    ),
    1, -- band number (usually 1 for single-band rasters)
    1, -- x origin for setting values (usually 1)
    1, -- y origin for setting values (usually 1)
    ARRAY[
      ARRAY[1, 2],
      ARRAY[3, 4]
    ]::double precision[][] -- 2D array of values
  );
```

Reclass the values to new values and dump the values of the new raster for display.

Code

```
WITH rc AS (
  SELECT ST_ReclassExact(rast, -- input raster
    ARRAY[4, 3, 2, 1], -- input map
    ARRAY[14, 13, 12, 11], -- output map
    1, -- band number to remap
    '32BUI'              -- output raster pixel type
    ) AS rast
  FROM reclassexact
  WHERE id = 1
  )
SELECT 'rce-1', (ST_DumpValues(rc.rast)).*
FROM rc;
```

Siehe auch

ST_Reclass, ST_AddBand, ST_Band, ST_MakeEmptyRaster

11.12.14 ST_Union

ST_Union — Gibt die Vereinigung mehrerer Rasterkacheln in einem einzelnen Raster mit mehreren Bändern zurück.

Synopsis

raster **ST_Union**(setof raster rast);
raster **ST_Union**(setof raster rast, unionarg[] unionargset);
raster **ST_Union**(setof raster rast, integer nband);
raster **ST_Union**(setof raster rast, text uniontype);
raster **ST_Union**(setof raster rast, integer nband, text uniontype);

Description

Gibt die Vereinigung von Rasterkacheln in einem einzelnen Raster zurück, der mindestens aus einem Band besteht. Die räumliche Ausdehnung des Zielraster entspricht der Gesamtausdehnung. Im Fall von Überschneidungen wird der resultierende Wert durch `uniontype` festgelegt, welcher folgende Werte annehmen kann: LAST (Standardwert), FIRST, MIN, MAX, COUNT, SUM, MEAN, RANGE.



Anmerkung
Damit Raster vereinigt werden können, müssen sie alle gleich ausgerichtet sein. Siehe ST_SameAlignment und ST_NotSameAlignmentReason für weitere Details und Hilfe. Eine Möglichkeit, um Probleme mit der Ausrichtung zu beheben ist ST_Resample und die Verwendung eines gemeinsamen Referenzraster zur Anpassung.

Verfügbarkeit: 2.0.0

Erweiterung: 2.1.0 Geschwindigkeit verbessert (zur Gänze C-basiert)

Verfügbarkeit: 2.1.0 ST_Union(rast, unionarg) Variante wurde eingeführt.

Erweiterung: 2.1.0 ST_Union(rast) (Variante 1) vereinigt alle Bänder aller Ausgangsraster. Vorherige Versionen von PostGIS setzten das erste Band voraus.

Erweiterung: 2.1.0 ST_Union(rast, uniontype) (Variante 4) vereinigt alle Bänder aller Ausgangsraster.

Beispiele

Reconstitute a single band chunked raster tile.

This creates a single band from first band of raster tiles. That form the original file system tile.

Code

```
SELECT filename, ST_Union(rast, 1) As file_rast
FROM sometable WHERE filename IN('dem01', 'dem02') GROUP BY filename;
```

Return a multi-band raster that is the union of tiles intersecting geometry.

This creates a multi band raster collecting all the tiles that intersect a line. Note that In 2.0, this would have just returned a single band raster. , new union works on all bands by default. This is equivalent to unionarg: ARRAY[ROW(1, 'LAST'), ROW(2, 'LAST'), ROW(3, 'LAST')]::unionarg[].

Code

```
SELECT ST_Union(rast)
FROM aerials.boston
WHERE ST_Intersects(rast, ST_GeomFromText('LINESTRING(230486 887771,230500 88772)', 26986) ) ↩
    ;
```

Select and reorder bands while unioning tiles that intersect a geometry.

Um nur eine Teilmenge der Bänder zu erhalten, oder die Reihenfolge der Bänder zu ändern, können wir die längere Syntax verwenden

This creates a multi band raster collecting all the tiles that intersect a line.

Code

```
SELECT ST_Union(rast, ARRAY[ROW(2, 'LAST'), ROW(1, 'LAST'), ROW(3, 'LAST')]::unionarg[])
FROM aerials.boston
WHERE ST_Intersects(rast, ST_GeomFromText('LINESTRING(230486 887771,230500 88772)', 26986) ) ↩
    ;
```

Siehe auch

unionarg, ST_Envelope, ST_ConvexHull, ST_Clip, ST_Union

11.13 Integrierte Map Algebra Callback Funktionen

11.13.1 ST_Distinct4ma

ST_Distinct4ma — Funktion zur Rasterdatenverarbeitung, welche die Anzahl der einzelnen Pixelwerte in der Nachbarschaft errechnet.

Synopsis

float8 **ST_Distinct4ma**(float8[][] matrix, text nodatamode, text[] VARIADIC args);
double precision **ST_Distinct4ma**(double precision[][][] value, integer[][] pos, text[] VARIADIC userargs);

Description

Berechnet die Anzahl der einzelnen Pixelwerte in der Nachbarschaft von Pixel.



Anmerkung
Variante 1 ist eine spezialisierte Rückruffunktion, die als Rückrufparameter für ST_MapAlgebraFctNgb verwendet werden kann.



Anmerkung
Variante 2 ist eine spezialisierte Rückruffunktion, die als Rückrufparameter für ST_MapAlgebra (callback function version) verwendet werden kann.



Warnung
Von der Verwendung der Variante 1 wird abgeraten, da ST_MapAlgebraFctNgb seit 2.1.0 überholt ist.

Verfügbarkeit: 2.0.0

Erweiterung: 2.1.0 Variante 2 wurde hinzugefügt

Beispiele

Code

```
SELECT
    rid,
    ST_Value(ST_MapAlgebraFctNgb(rast, 1, NULL, 1, 1, 'ST_Distinct4ma(float[][],text,text[]) ↩
    '::regprocedure, 'ignore', NULL), 2, 2
    )
FROM dummy_rast
WHERE rid = 2;
```

Output

```
rid | st_value
-----+----------
   2 |        3
(1 row)
```

Siehe auch

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 — Funktion zur Rasterdatenverarbeitung, die den Wert eines Pixel aus den Pixel der Nachbarschaft interpoliert.

Synopsis

double precision **ST_InvDistWeight4ma**(double precision[][][] value, integer[][] pos, text[] VARIADIC userargs);

Description

Interpoliert den Wert eines Pixel über die Methode der inversen Distanzwichtung.

Es gibt zwei optionale Parameter, die über `userargs` übergeben werden können. Der erste Parameter ist der Exponent (die Variable "k" in unterer Gleichung); dieser liegt zwischen 0 und 1 und wird in der Gleichung für die Inverse Distanzwichtung verwendet. Wenn er nicht angegeben ist, wird standardmäßig 1 angenommen. Der zweite Parameter entspricht der Gewichtung in Prozent und wird nur verwendet, wenn der Wert der betrachteten Pixel einem aus der Nachbarschaft interpolierten Wert entspricht. Wenn er nicht angegeben ist und das betrachtete Pixel einen Wert hat, so wird dieser Wert zurückgegeben.

The basic inverse-distance weighted estimate is: z = (Σᵢ zᵢ / dᵢᵏ) / (Σᵢ 1 / dᵢ&#x1d4
Here `zᵢ` is a neighboring pixel value, `dᵢ` is its distance from the pixel being estimated, and `k` is a real-valued power factor between 0 and 1.



Anmerkung
Diese Funktion ist eine spezialisierte Rückruffunktion, die als Rückrufparameter für ST_MapAlgebra (callback function version) verwendet werden kann.

Verfügbarkeit: 2.1.0

Beispiele

Code

```
WITH foo AS (
    SELECT ST_SetValues(ST_AddBand(ST_MakeEmptyRaster(3, 3, 0, 0, 1, -1, 0, 0, 0), 1, '32BF' ↩
    , NULL::double precision, -9999),
        1, 1, 1, ARRAY[
            [10, 10, 10],
            [10, NULL, 20],
            [20, 20, 20]
        ]::double precision[][]
    ) AS rast
```

```
)
SELECT ST_Value(
    ST_MapAlgebra(
        rast,
        1,
        'ST_InvDistWeight4ma(double precision[][][],integer[][],text[])'::regprocedure,
        '32BF',
        'FIRST',
        NULL,
        1,
        1
    ),
    2, 2
) AS interpolated_value
FROM foo;
```

Output

```
15
```

Siehe auch

ST_MapAlgebra (callback function version), ST_MinDist4ma

11.13.3 ST_Max4ma

ST_Max4ma — Funktion zur Rasterdatenverarbeitung, die den maximalen Zellwert in der Nachbarschaft eines Pixel errechnet.

Synopsis

float8 **ST_Max4ma**(float8[][] matrix, text nodatamode, text[] VARIADIC args);
double precision **ST_Max4ma**(double precision[][][] value, integer[][] pos, text[] VARIADIC userargs);

Description

Berechnet den maximalen Zellwert in der Nachbarschaft von Pixel.

Bei Variante 2 kann ein Substitutionswert für NODATA an "userargs" übergeben werden.



Anmerkung
Variante 1 ist eine spezialisierte Rückruffunktion, die als Rückrufparameter für ST_MapAlgebraFctNgb verwendet werden kann.



Anmerkung
Variante 2 ist eine spezialisierte Rückruffunktion, die als Rückrufparameter für ST_MapAlgebra (callback function version) verwendet werden kann.



Warnung
Von der Verwendung der Variante 1 wird abgeraten, da ST_MapAlgebraFctNgb seit 2.1.0 überholt ist.

Verfügbarkeit: 2.0.0

Erweiterung: 2.1.0 Variante 2 wurde hinzugefügt

Beispiele

Code

```
SELECT
    rid,
    ST_Value(ST_MapAlgebraFctNgb(rast, 1, NULL, 1, 1, 'ST_Max4ma(float[][],text,text[])':: ↩
    regprocedure, 'ignore', NULL), 2, 2
    )
FROM dummy_rast
WHERE rid = 2;
```

Output

```
rid | st_value
-----+----------
   2 |      254
(1 row)
```

Siehe auch

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 — Funktion zur Rasterdatenverarbeitung, die den mittleren Zellwert in der Nachbarschaft von Pixel errechnet.

Synopsis

float8 **ST_Mean4ma**(float8[][] matrix, text nodatamode, text[] VARIADIC args);
double precision **ST_Mean4ma**(double precision[][][] value, integer[][] pos, text[] VARIADIC userargs);

Description

Berechnet den mittleren Zellwert in der Nachbarschaft von Pixel.

Bei Variante 2 kann ein Substitutionswert für NODATA an "userargs" übergeben werden.



Anmerkung
Variante 1 ist eine spezialisierte Rückruffunktion, die als Rückrufparameter für ST_MapAlgebraFctNgb verwendet werden kann.



Anmerkung
Variante 2 ist eine spezialisierte Rückruffunktion, die als Rückrufparameter für ST_MapAlgebra (callback function version) verwendet werden kann.



Warnung
Von der Verwendung der Variante 1 wird abgeraten, da ST_MapAlgebraFctNgb seit 2.1.0 überholt ist.

Verfügbarkeit: 2.0.0

Erweiterung: 2.1.0 Variante 2 wurde hinzugefügt

Beispiele

Variant 1.

Code

```
SELECT
    rid,
    ST_Value(ST_MapAlgebraFctNgb(rast, 1, '32BF', 1, 1, 'ST_Mean4ma(float[][],text,text[])' ↩
    ::regprocedure, 'ignore', NULL), 2, 2
    )
FROM dummy_rast
WHERE rid = 2;
```

Output

```
rid |     st_value
-----+------------------
   2 | 253.222229003906
(1 row)
```

Variant 2.

Code

```
SELECT
  rid,
  ST_Value(
      ST_MapAlgebra(rast, 1, 'ST_Mean4ma(double precision[][][], integer[][], text[])':: ↩
    regprocedure, '32BF', 'FIRST', NULL, 1, 1),
      2,
      2)
FROM dummy_rast
 WHERE rid = 2;
```

Output

```
rid |     st_value
-----+------------------
   2 | 253.222229003906
(1 row)
```

Siehe auch

ST_MapAlgebraFctNgb, ST_MapAlgebra (callback function version), ST_Min4ma, ST_Max4ma, ST_Sum4ma, ST_Range4ma, ST_StdDev4ma

11.13.5 ST_Min4ma

ST_Min4ma — Funktion zur Rasterdatenverarbeitung, die den minimalen Zellwert in der Nachbarschaft von Pixel errechnet.

Synopsis

float8 **ST_Min4ma**(float8[][] matrix, text nodatamode, text[] VARIADIC args);
double precision **ST_Min4ma**(double precision[][][] value, integer[][] pos, text[] VARIADIC userargs);

Description

Berechnet den minimalen Zellwert in der Nachbarschaft von Pixel.

Bei Variante 2 kann ein Substitutionswert für NODATA an "userargs" übergeben werden.



Anmerkung
Variante 1 ist eine spezialisierte Rückruffunktion, die als Rückrufparameter für ST_MapAlgebraFctNgb verwendet werden kann.



Anmerkung
Variante 2 ist eine spezialisierte Rückruffunktion, die als Rückrufparameter für ST_MapAlgebra (callback function version) verwendet werden kann.



Warnung
Von der Verwendung der Variante 1 wird abgeraten, da ST_MapAlgebraFctNgb seit 2.1.0 überholt ist.

Verfügbarkeit: 2.0.0

Erweiterung: 2.1.0 Variante 2 wurde hinzugefügt

Beispiele

Code

```
SELECT
    rid,
    ST_Value(ST_MapAlgebraFctNgb(rast, 1, NULL, 1, 1, 'ST_Min4ma(float[][],text,text[])':: ↩
    regprocedure, 'ignore', NULL), 2, 2
    )
FROM dummy_rast
WHERE rid = 2;
```

Output

```
rid | st_value
-----+----------
   2 |      250
(1 row)
```

Siehe auch

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 — Funktion zur Rasterdatenverarbeitung, welche die kürzeste Entfernung (in Pixel) zwischen dem Pixel von Interesse und einem benachbarten Pixel mit Zellwert zurückgibt.

Synopsis

double precision **ST_MinDist4ma**(double precision[][][] value, integer[][] pos, text[] VARIADIC userargs);

Description

Gibt die kürzeste Entfernung (Pixelanzahl) zwischen dem Pixel von Interesse und dem nächstgelegenen Pixel mit Zellwert in der Nachbarschaft zurück.



Anmerkung
Diese Funktion soll einen Anhaltspunkt liefern, um die Sinnhaftigkeit des mittels ST_InvDistWeight4ma interpolierten Wertes für das betrachtete Pixel abzuleiten. Diese Funktion ist besonders nützlich wenn die Nachbarschaft des Pixel nur spärlich besetzt ist.



Anmerkung
Diese Funktion ist eine spezialisierte Rückruffunktion, die als Rückrufparameter für ST_MapAlgebra (callback function version) verwendet werden kann.

Verfügbarkeit: 2.1.0

Beispiele

Code

```
WITH foo AS (
    SELECT ST_SetValues(ST_AddBand(ST_MakeEmptyRaster(3, 3, 0, 0, 1, -1, 0, 0, 0), 1, '32BF' ↩
    , NULL::double precision, -9999),
        1, 1, 1, ARRAY[
            [10, 10, 10],
            [10, NULL, 20],
            [20, 20, 20]
        ]::double precision[][]
    ) AS rast
)
SELECT ST_Value(
    ST_MapAlgebra(
        rast,
        1,
        'ST_MinDist4ma(double precision[][][],integer[][],text[])'::regprocedure,
        '32BF',
        'FIRST',
        NULL,
        1,
        1
    ),
    2, 2
) AS min_distance
FROM foo;
```

Output

```
1
```

Siehe auch

ST_MapAlgebra (callback function version), ST_InvDistWeight4ma

11.13.7 ST_Range4ma

ST_Range4ma — Funktion zur Rasterdatenverarbeitung, die den Wertebereich der Pixel in einer Nachbarschaft errechnet.

Synopsis

float8 **ST_Range4ma**(float8[][] matrix, text nodatamode, text[] VARIADIC args);
double precision **ST_Range4ma**(double precision[][][] value, integer[][] pos, text[] VARIADIC userargs);

Description

Berechnet den Wertebereich der Pixel in einer Nachbarschaft von Pixel.

Bei Variante 2 kann ein Substitutionswert für NODATA an "userargs" übergeben werden.



Anmerkung
Variante 1 ist eine spezialisierte Rückruffunktion, die als Rückrufparameter für ST_MapAlgebraFctNgb verwendet werden kann.



Anmerkung
Variante 2 ist eine spezialisierte Rückruffunktion, die als Rückrufparameter für ST_MapAlgebra (callback function version) verwendet werden kann.



Warnung
Von der Verwendung der Variante 1 wird abgeraten, da ST_MapAlgebraFctNgb seit 2.1.0 überholt ist.

Verfügbarkeit: 2.0.0

Erweiterung: 2.1.0 Variante 2 wurde hinzugefügt

Beispiele

Code

```
SELECT
    rid,
    ST_Value(ST_MapAlgebraFctNgb(rast, 1, NULL, 1, 1, 'ST_Range4ma(float[][],text,text[])':: ↩
    regprocedure, 'ignore', NULL), 2, 2
    )
FROM dummy_rast
WHERE rid = 2;
```

Output

```
rid | st_value
-----+----------
   2 |        4
(1 row)
```

Siehe auch

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 — Funktion zur Rasterdatenverarbeitung, welche die Standardabweichung der Zellwerte in der Nachbarschaft von Pixel errechnet.

Synopsis

float8 **ST_StdDev4ma**(float8[][] matrix, text nodatamode, text[] VARIADIC args);
double precision **ST_StdDev4ma**(double precision[][][] value, integer[][] pos, text[] VARIADIC userargs);

Description

Berechnet die Standardabweichung der Zellwerte in der Nachbarschaft von Pixel.



Anmerkung
Variante 1 ist eine spezialisierte Rückruffunktion, die als Rückrufparameter für ST_MapAlgebraFctNgb verwendet werden kann.



Anmerkung
Variante 2 ist eine spezialisierte Rückruffunktion, die als Rückrufparameter für ST_MapAlgebra (callback function version) verwendet werden kann.



Warnung
Von der Verwendung der Variante 1 wird abgeraten, da ST_MapAlgebraFctNgb seit 2.1.0 überholt ist.

Verfügbarkeit: 2.0.0

Erweiterung: 2.1.0 Variante 2 wurde hinzugefügt

Beispiele

Code

```
SELECT
    rid,
    ST_Value(ST_MapAlgebraFctNgb(rast, 1, '32BF', 1, 1, 'ST_StdDev4ma(float[][],text,text[]) ↩
    '::regprocedure, 'ignore', NULL), 2, 2
    )
FROM dummy_rast
WHERE rid = 2;
```

Output

```
rid |     st_value
-----+------------------
   2 | 1.30170822143555
(1 row)
```

Siehe auch

ST_MapAlgebraFctNgb, ST_MapAlgebra (callback function version), ST_Min4ma, ST_Max4ma, ST_Sum4ma, ST_Mean4ma, ST_Distinct4ma, ST_StdDev4ma

11.13.9 ST_Sum4ma

ST_Sum4ma — Funktion zur Rasterdatenverarbeitung, die die Summe aller Zellwerte in der Nachbarschaft von Pixel errechnet.

Synopsis

float8 **ST_Sum4ma**(float8[][] matrix, text nodatamode, text[] VARIADIC args);
double precision **ST_Sum4ma**(double precision[][][] value, integer[][] pos, text[] VARIADIC userargs);

Description

Berechnet die Summe aller Zellwerte in der Nachbarschaft von Pixel.

Bei Variante 2 kann ein Substitutionswert für NODATA an "userargs" übergeben werden.



Anmerkung
Variante 1 ist eine spezialisierte Rückruffunktion, die als Rückrufparameter für ST_MapAlgebraFctNgb verwendet werden kann.



Anmerkung
Variante 2 ist eine spezialisierte Rückruffunktion, die als Rückrufparameter für ST_MapAlgebra (callback function version) verwendet werden kann.



Warnung
Von der Verwendung der Variante 1 wird abgeraten, da ST_MapAlgebraFctNgb seit 2.1.0 überholt ist.

Verfügbarkeit: 2.0.0

Erweiterung: 2.1.0 Variante 2 wurde hinzugefügt

Beispiele

Code

```
SELECT
    rid,
    ST_Value(ST_MapAlgebraFctNgb(rast, 1, '32BF', 1, 1, 'ST_Sum4ma(float[][],text,text[])':: ↩
    regprocedure, 'ignore', NULL), 2, 2
    )
FROM dummy_rast
WHERE rid = 2;
```

Output

```
rid | st_value
-----+----------
   2 |     2279
(1 row)
```

Siehe auch

ST_MapAlgebraFctNgb, ST_MapAlgebra (callback function version), ST_Min4ma, ST_Max4ma, ST_Mean4ma, ST_Range4ma, ST_Distinct4ma, ST_StdDev4ma

11.14 Rasterverarbeitung: DEM (Elevation)

11.14.1 ST_Aspect

ST_Aspect — Gibt die Exposition (standardmäßig in Grad) eines Rasterbandes mit Höhen aus. Nützlich für Terrain-Analysen.

Synopsis

raster **ST_Aspect**(raster rast, integer band=1, text pixeltype=32BF, text units=DEGREES, boolean interpolate_nodata=FALSE);
raster **ST_Aspect**(raster rast, integer band, raster customextent, text pixeltype=32BF, text units=DEGREES, boolean interpolate_nodata=FALSE);

Description

Gibt die Exposition (standardmäßig in Grad) eines Höhenrasterbandes zurück. Verwendet Map Algebra und wendet die Expositionsgleichung auf benachbarte Pixel an.

`units` die Einheiten für die Exposition. Mögliche Werte sind: RADIANS, DEGREES (Standardeinstellung).

Wenn `units` = RADIANS, dann liegen die Werte zwischen 0 und 2 * pi Radiant und werden im Uhrzeigersinn von Norden her angegeben.

Wenn `units` = DEGREES, dann liegen die Werte zwischen 0 und 360 Grad und sind im Uhrzeigersinn von Norden her angegeben.

Wenn die Neigung einer Zelle null ist, dann ist die Exposition des Pixel -1.



Anmerkung
Für weitere Information über Neigung, Exposition und Schummerung siehe ESRI - How hillshade works und ERDAS Field Guide - Aspect Images.

Verfügbarkeit: 2.0.0

Erweiterung: 2.1.0 Verwendet ST_MapAlgebra() und der optionale Funktionsparameter `interpolate_nodata` wurde hinzugefügt

Änderung: 2.1.0 In Vorgängerversionen wurden die Werte in Radiant ausgegeben. Nun werden die Werte standardmäßig in Grad ausgegeben.

Beispiele

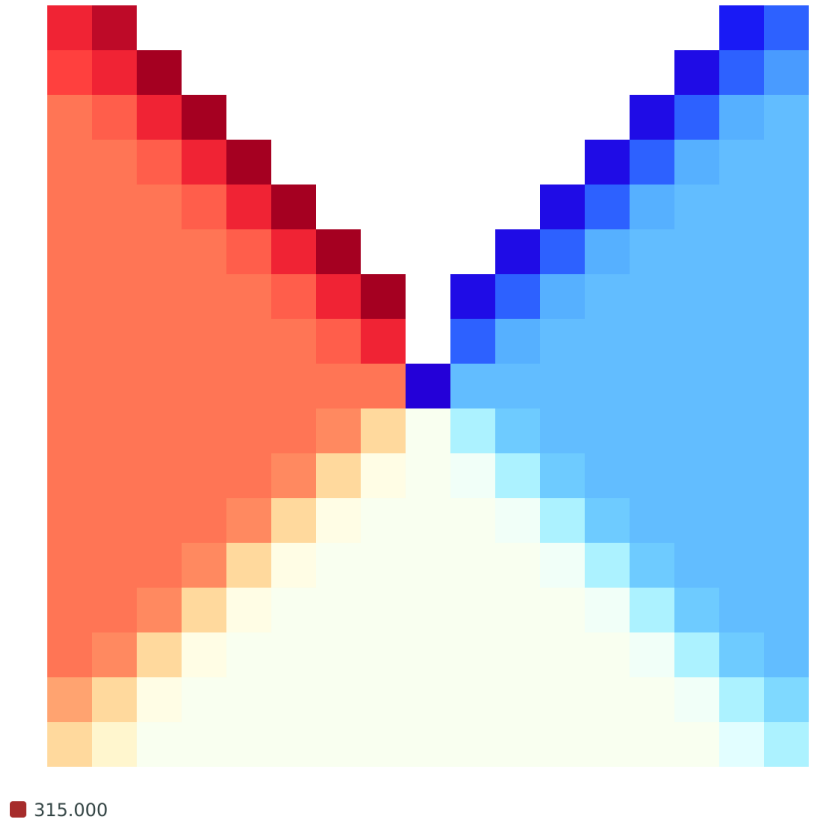
Variant 1.

This symmetric peak highlights the expected compass directions at the corners and the `-1` aspect at the summit. The generated PNG shows the full aspect surface without relying on a manual illustration. The generated source raster is large enough for the PNG to show the directional field around the summit.

Code

```
WITH pixels AS (
    SELECT
        x,
        y,
        (20 - greatest(abs(x - 9), abs(y - 9)))::double precision AS value
    FROM generate_series(1, 17) AS y
    CROSS JOIN generate_series(1, 17) AS x
), rows AS (
    SELECT
        y,
        array_agg(value ORDER BY x) AS row_values
    FROM pixels
    GROUP BY y
), source AS (
    SELECT ST_SetValues(
        ST_AddBand(
            ST_MakeEmptyRaster(17, 17, 0, 0, 1, -1, 0, 0, 0),
            1, '32BF', 0, -9999
        ),
        1, 1, 1,
        array_agg(row_values ORDER BY y)::double precision[][]
    ) AS rast
    FROM rows
), aspect AS (
    SELECT ST_Aspect(rast, 1, '32BF') AS rast
    FROM source
)
SELECT
    round(ST_Value(rast, 1, 1, 1)::numeric, 3) AS northwest_degrees,
    round(ST_Value(rast, 1, 9, 9)::numeric, 3) AS summit_degrees,
    round(ST_Value(rast, 1, 17, 17)::numeric, 3) AS southeast_degrees,
    ST_AsPNG(ST_ColorMap(rast, 1, 'bluered')) AS image
FROM aspect;
```

Figure



Variant 2.

Complete example of tiles of a coverage.

Code

```
WITH foo AS (
    SELECT ST_Tile(ST_SetValues(ST_AddBand(ST_MakeEmptyRaster(6, 6, 0, 0, 1, -1, 0, 0, 0),
                1, '32BF', 0, -9999
            ),
            1, 1, 1, ARRAY[
                [1, 1, 1, 1, 1, 1],
                [1, 1, 1, 1, 2, 1],
                [1, 2, 2, 3, 3, 1],
                [1, 1, 3, 2, 1, 1],
                [1, 2, 2, 1, 2, 1],
                [1, 1, 1, 1, 1, 1]
            ]::double precision[]
        ),
        2, 2
    ) AS rast
)
SELECT
    t1.rast,
    ST_Aspect(ST_Union(t2.rast), 1, t1.rast)
FROM foo t1
CROSS JOIN foo t2
WHERE ST_Intersects(t1.rast, t2.rast)
GROUP BY t1.rast;
```

Siehe auch

ST_MapAlgebra (callback function version), ST_TRI, ST_TPI, ST_Roughness, ST_HillShade, ST_Slope

11.14.2 ST_HillShade

ST_HillShade — Gibt für gegebenen Horizontalwinkel, Höhenwinkel, Helligkeit und Maßstabsverhältnis die hypothetische Beleuchtung eines Höhenrasterbandes zurück.

Synopsis

raster **ST_HillShade**(raster rast, integer band=1, text pixeltype=32BF, double precision azimuth=315, double precision altitude=45, double precision max_bright=255, double precision scale=1.0, boolean interpolate_nodata=FALSE);
raster **ST_HillShade**(raster rast, integer band, raster customextent, text pixeltype=32BF, double precision azimuth=315, double precision altitude=45, double precision max_bright=255, double precision scale=1.0, boolean interpolate_nodata=FALSE);

Description

Gibt für gegebenen Horizontalwinkel, Höhenwinkel, Helligkeit und Maßstabsverhältnis die hypothetische Beleuchtung eines Höhenrasterbandes zurück. Verwendet Map Algebra und wendet die Schummerungsgleichung auf die Nachbarpixel an. Die zurückgegebenen Pixelwerte liegen zwischen 0 und 255.

`azimuth` der Richtungswinkel zwischen 0 und 360 Grad, im Uhrzeigersinn von Norden gemessen.

`altitude` der Höhenwinkel zwischen 0 und 90 Grad, wobei 0 Grad am Horizont und 90 Grad direkt über Kopf liegt.

`max_bright` ein Wert zwischen 0 und 255, wobei 0 keine Helligkeit und 255 maximale Helligkeit bedeutet.

`scale` ist das Verhältnis zwischen vertikalen und horizontalen Einheiten. Für Feet:LatLon verwenden Sie bitte scale=370400, für Meter:LatLon scale=111120.

Wenn `interpolate_nodata` TRUE ist, dann werden die NODATA Pixel des Ausgangsraster mit ST_InvDistWeight4ma interpoliert, bevor die Schummerung berechnet wird.



Anmerkung
Für weitere Information zur Schummerung siehe How hillshade works.

Verfügbarkeit: 2.0.0

Erweiterung: 2.1.0 Verwendet ST_MapAlgebra() und der optionale Funktionsparameter `interpolate_nodata` wurde hinzugefügt

Änderung: 2.1.0 In Vorgängerversionen wurden Richtungswinkel und Höhenwinkel in Radiant angegeben. Nun werden die Werte in Grad ausgedrückt.

Beispiele

Variant 1.

Code

```
WITH foo AS (
    SELECT ST_SetValues(ST_AddBand(ST_MakeEmptyRaster(5, 5, 0, 0, 1, -1, 0, 0, 0), 1, '32BF' ↩
    , 0, -9999),
        1, 1, 1, ARRAY[
            [1, 1, 1, 1, 1],
```

```
            [1, 2, 2, 2, 1],
            [1, 2, 3, 2, 1],
            [1, 2, 2, 2, 1],
            [1, 1, 1, 1, 1]
        ]::double precision[][]
    ) AS rast
)
SELECT
    ST_DumpValues(ST_HillShade(rast, 1, '32BF'))
FROM foo
```

```
                                                
```

Synopsis

raster **ST_Roughness**(raster rast, integer nband, raster customextent, text pixeltype="32BF" , boolean interpolate_nodata=FALSE);

Description

Berechnet die "Rauigkeit" eines DHM, indem für ein bestimmtes Gebiet das Maximum vom Minimum abgezogen wird..

Verfügbarkeit: 2.1.0

Beispiele

Code

```
WITH foo AS (
    SELECT ST_SetValues(ST_AddBand(ST_MakeEmptyRaster(5, 5, 0, 0, 1, -1, 0, 0, 0), 1, '32BF' ↩
    , 0, -9999),
        1, 1, 1, ARRAY[
            [1, 1, 2, 3, 3],
            [1, 2, 4, 15, 3],
            [2, 4, 10, 7, 5],
            [3, 5, 7, 6, 4],
            [3, 3, 5, 4, 4]
        ]::double precision[][]
    ) AS rast
)
SELECT ST_Value(ST_Roughness(rast, 1, '32BF'), 3, 3) AS roughness
FROM foo;
```

Output

```
13
```

Siehe auch

ST_MapAlgebra (callback function version), ST_TRI, ST_TPI, ST_Slope, ST_HillShade, ST_Aspect

11.14.4 ST_Slope

ST_Slope — Gibt die Neigung (standardmäßig in Grad) eines Höhenrasterbandes zurück. Nützlich für Terrain-Analysen.

Synopsis

raster **ST_Slope**(raster rast, integer nband=1, text pixeltype=32BF, text units=DEGREES, double precision scale=1.0, boolean interpolate_nodata=FALSE);
raster **ST_Slope**(raster rast, integer nband, raster customextent, text pixeltype=32BF, text units=DEGREES, double precision scale=1.0, boolean interpolate_nodata=FALSE);

Description

Gibt die Neigung (standardmäßig in Grad) eines Höhenrasterbandes zurück. Verwendet Map Algebra und wendet die Neigungsgleichung auf benachbarte Pixel an.

`units` die Einheiten für die Neigung. Mögliche Werte sind: RADIANS, DEGREES (Standardeinstellung) und Prozent.

`scale` ist das Verhältnis zwischen vertikalen und horizontalen Einheiten. Für Feet:LatLon verwenden Sie bitte scale=370400, für Meter:LatLon scale=111120.

Wenn `interpolate_nodata` TRUE ist, dann werden die NODATA Pixel des Ausgangsraster mit ST_InvDistWeight4ma interpoliert, bevor die Neigung der Oberfläche berechnet wird.



Anmerkung
Für weitere Information über Neigung, Exposition und Schummerung siehe ESRI - How hillshade works und ERDAS Field Guide - Slope Images.

Verfügbarkeit: 2.0.0

Erweiterung: 2.1.0 Verwendet ST_MapAlgebra() und es wurden die optionalen Funktionsparameter `units`, `scale` und `interpolate` hinzugefügt

Änderung: 2.1.0 In Vorgängerversionen wurden die Werte in Radiant ausgegeben. Nun werden die Werte standardmäßig in Grad ausgegeben.

Beispiele

Variant 1.

Code

```
WITH foo AS (
    SELECT ST_SetValues(ST_AddBand(ST_MakeEmptyRaster(5, 5, 0, 0, 1, -1, 0, 0, 0), 1, '32BF' ↩
    , 0, -9999),
        1, 1, 1, ARRAY[
            [1, 1, 1, 1, 1],
            [1, 2, 2, 2, 1],
            [1, 2, 3, 2, 1],
            [1, 2, 2, 2, 1],
            [1, 1, 1, 1, 1]
        ]::double precision[][]
    ) AS rast
)
SELECT
    ST_DumpValues(ST_Slope(rast, 1, '32BF'))
FROM foo

                            st_dumpvalues
```

Output

```
----------------------------------------------------------------------------------------------------------------------------------

----------------------------------------------------------------------------------------------------------------------------------

-------------------------------------------------------------------
 ↩
    (1,"{{10.0249881744385,21.5681285858154,26.5650520324707,21.5681285858154,10.0249881744385},{21.5681285858154,35.26438903808

{26.5650520324707,36.8698959350586,0,36.8698959350586,26.5650520324707},{21.5681285858154,35.2643890380859,36.8698959350586,35.26

5681285858154,26.5650520324707,21.5681285858154,10.0249881744385}}")
(1 row)
```

Variant 2.

Complete example of tiles of a coverage.

Code

```
WITH foo AS (
    SELECT ST_Tile(ST_SetValues(ST_AddBand(ST_MakeEmptyRaster(6, 6, 0, 0, 1, -1, 0, 0, 0),
                1, '32BF', 0, -9999
            ),
            1, 1, 1, ARRAY[
                [1, 1, 1, 1, 1, 1],
                [1, 1, 1, 1, 2, 1],
                [1, 2, 2, 3, 3, 1],
                [1, 1, 3, 2, 1, 1],
                [1, 2, 2, 1, 2, 1],
                [1, 1, 1, 1, 1, 1]
            ]::double precision[]
        ),
        2, 2
    ) AS rast
)
SELECT
    t1.rast,
    ST_Slope(ST_Union(t2.rast), 1, t1.rast)
FROM foo t1
CROSS JOIN foo t2
WHERE ST_Intersects(t1.rast, t2.rast)
GROUP BY t1.rast;
```

Siehe auch

ST_MapAlgebra (callback function version), ST_TRI, ST_TPI, ST_Roughness, ST_HillShade, ST_Aspect

11.14.5 ST_TPI

ST_TPI — Berechnet den "Topographic Position Index" eines Raster.

Synopsis

raster **ST_TPI**(raster rast, integer nband, raster customextent, text pixeltype="32BF" , boolean interpolate_nodata=FALSE);

Description

Berechnet den Topographischen Lageindex, welcher als fokaler Mittelwert mit dem Radius eins, abzüglich der mittigen Zelle, definiert ist.



Anmerkung
Diese Funktion unterstützt lediglich einen "focalmean"-Radius von Eins.

Verfügbarkeit: 2.1.0

Beispiele

Code

```
WITH foo AS (
    SELECT ST_SetValues(ST_AddBand(ST_MakeEmptyRaster(5, 5, 0, 0, 1, -1, 0, 0, 0), 1, '32BF' ↩
    , 0, -9999),
        1, 1, 1, ARRAY[
            [1, 1, 2, 3, 3],
            [1, 2, 4, 15, 3],
            [2, 4, 10, 7, 5],
            [3, 5, 7, 6, 4],
            [3, 3, 5, 4, 4]
        ]::double precision[][]
    ) AS rast
)
SELECT ST_Value(ST_TPI(rast, 1, '32BF'), 3, 3) AS tpi
FROM foo;
```

Output

```
3.75
```

Siehe auch

ST_MapAlgebra (callback function version), ST_TRI, ST_Roughness, ST_Slope, ST_HillShade, ST_Aspect

11.14.6 ST_TRI

ST_TRI — Gibt einen Raster mit errechneten Geländerauheitsindex aus.

Synopsis

raster **ST_TRI**(raster rast, integer nband, raster customextent, text pixeltype="32BF" , boolean interpolate_nodata=FALSE);

Description

Der Geländerauheitsindex wird durch den Vergleich eines zentralen Pixel mit seinen Nachbarn berechnet. Dabei werden die Differenzen der absoluten Werte (Beträge) genommen und für das Ergebnis gemittelt.



Anmerkung
Diese Funktion unterstützt lediglich einen "focalmean"-Radius von Eins.

Verfügbarkeit: 2.1.0

Beispiele

Code

```
WITH foo AS (
    SELECT ST_SetValues(ST_AddBand(ST_MakeEmptyRaster(5, 5, 0, 0, 1, -1, 0, 0, 0), 1, '32BF' ↩
    , 0, -9999),
        1, 1, 1, ARRAY[
            [1, 1, 2, 3, 3],
```

```
            [1, 2, 4, 15, 3],
            [2, 4, 10, 7, 5],
            [3, 5, 7, 6, 4],
            [3, 3, 5, 4, 4]
        ]::double precision[][]
    ) AS rast
)
SELECT ST_Value(ST_TRI(rast, 1, '32BF'), 3, 3) AS tri
FROM foo;
```

Output

```
5
```

Siehe auch

ST_MapAlgebra (callback function version), ST_Roughness, ST_TPI, ST_Slope, ST_HillShade, ST_Aspect

11.14.7 ST_InterpolateRaster

ST_InterpolateRaster — Interpoliert eine gerasterte Oberfläche auf der Grundlage eines Eingabesatzes von 3D-Punkten, wobei die X- und Y-Werte zur Positionierung der Punkte auf dem Gitter und der Z-Wert der Punkte als Oberflächenhöhe verwendet werden.

Synopsis

raster **ST_InterpolateRaster**(geometry input_points, text algorithm_options, raster template, integer template_band_num=1);

Description

Interpoliert eine gerasterte Oberfläche auf der Grundlage eines Eingabesatzes von 3D-Punkten, wobei die X- und Y-Werte zur Positionierung der Punkte auf dem Gitter und der Z-Wert der Punkte als Oberflächenhöhe verwendet werden. Es stehen fünf Interpolationsalgorithmen zur Verfügung: inverser Abstand, inverser Abstand nächster Nachbar, gleitender Durchschnitt, nächster Nachbar und lineare Interpolation. Weitere Einzelheiten zu den Algorithmen und ihren Parametern finden Sie in der gdal_grid Dokumentation. Weitere Informationen darüber, wie Interpolationen berechnet werden, finden Sie im GDAL-Grid-Tutorial.

Die Eingabeparameter sind:

`input_points` Die Punkte, die die Interpolation steuern. Jede Geometrie mit Z-Werten ist akzeptabel, alle Punkte in der Eingabe werden verwendet.

`algorithm_options` Eine Zeichenkette, die den Algorithmus und die Algorithmusoptionen definiert, im Format von gdal_grid. Für eine Interpolation mit inversem Abstand und einer Glättung von 2 würde man zum Beispiel "invdist:smoothing=2.0" verwenden.

`template` Eine Rastervorlage zur Steuerung der Geometrie des Ausgaberasters. Die Breite, Höhe, Pixelgröße, räumliche Ausdehnung und der Pixeltyp werden aus dieser Vorlage gelesen.

`template_band_num` Standardmäßig wird das erste Band im Vorlagenraster für die Steuerung des Ausgaberasters verwendet, aber das kann mit diesem Parameter angepasst werden.

Verfügbarkeit: 3.2.0

Beispiel

Code

```
SELECT ST_InterpolateRaster('MULTIPOINT(10.5 9.5 1000,11.5 8.5 1000,10.5 8.5 500,11.5 9.5 ↩
    500)'::geometry,
    'invdist:smoothing:2.0',
    ST_AddBand(ST_MakeEmptyRaster(200, 400, 10, 10, 0.01, -0.005, 0, 0), '16BSI')
)
```

Siehe auch

ST_Contour

11.14.8 ST_Contour

ST_Contour — Erzeugt einen Satz von Vektorkonturen aus dem angegebenen Rasterband unter Verwendung des GDAL-Konturierungsal

Synopsis

table(geom geometry, id integer, value float8) **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

Erzeugt eine Reihe von Vektorkonturen aus dem angegebenen Rasterband, unter Verwendung des GDAL Konturierungsalgorithmus.

Wenn der Parameter `fixed_levels` ein nicht-leeres Array ist, werden die Parameter `level_interval` und `level_base` ignoriert.

Die Eingabeparameter sind:

`rast` Das Raster zur Erzeugung der Kontur von

`bandnumber` Das Band zur Erzeugung der Kontur von

`level_interval` Der Höhenabstand zwischen den erzeugten Konturen

`level_base` Die "Basis", auf die sich die Konturintervalle beziehen. Normalerweise ist dies Null, kann aber auch anders sein. Um 10m-Konturen bei 5, 15, 25, ... zu erzeugen, wäre die LEVEL_BASE 5.

`fixed_levels` Der Höhenabstand zwischen den erzeugten Konturen

`polygonize` Wenn `true`, werden Konturpolygone anstelle von Polygonlinien erstellt.

Die Rückgabewerte sind eine Reihe von Datensätzen mit den folgenden Attributen:

`geom` Die Geometrie der Konturlinie.

`id` Ein eindeutiger Bezeichner, der der Höhenlinie von GDAL zugewiesen wird.

`value` Der Rasterwert, den die Linie darstellt. Bei einer DEM-Höheneingabe wäre dies die Höhe der Ausgangskontur.

Verfügbarkeit: 3.2.0

Beispiel

Code

```
WITH c AS (
SELECT (ST_Contour(rast, 1, fixed_levels => ARRAY[100.0, 200.0, 300.0])).*
FROM dem_grid WHERE rid = 1
)
SELECT geom, id, value
FROM c;
```

Siehe auch

ST_InterpolateRaster

11.15 Rasterverarbeitung: Raster zu Geometrie

11.15.1 Box3D

Box3D — Stellt das umschreibende Rechteck eines Raster als Box3D dar.

Synopsis

box3d **Box3D**(raster rast);

Description

Gibt ein Rechteck mit der Ausdehnung des Raster zurück.

Das Polygon ist durch die Eckpunkte des umschreibenden Rechtecks ((MINX, MINY), (MAXX, MAXY)) bestimmt

Änderung: 2.0.0 In Versionen vor 2.0 war dies üblicherweise Box2D anstelle von Box3D. Da Box2D überholt ist, wurde dies zu Box3D geändert.

Beispiele

Code

```
SELECT
    rid,
    Box3D(rast) AS rastbox
FROM dummy_rast;
```

Output

```
rid |        rastbox
----+-------------------------------------------------
1   | BOX3D(0.5 0.5 0,20.5 60.5 0)
2   | BOX3D(3427927.75 5793243.5 0,3427928 5793244 0)
```

Siehe auch

ST_Envelope

11.15.2 ST_ConvexHull

ST_ConvexHull — Gibt die Geometrie der konvexen Hülle des Raster, inklusive der Pixel deren Werte gleich BandNoDataValue sind. Bei regelmäßig geformten und nicht rotierten Raster ist das Ergebnis ident mit ST_Envelope. Diese Funktion ist deshalb nur bei unregelmäßig geformten oder rotierten Raster nützlich.

Synopsis

geometry **ST_ConvexHull**(raster rast);

Description

Gibt die Geometrie der konvexen Hülle des Raster, inklusive der Pixel NoDataBandValue des Bandes. Bei regelmäßig geformten und nicht rotierten Raster ist das Ergebnis mehr oder weniger das von ST_Envelope. Diese Funktion ist deshalb nur bei unregelmäßig geformten oder rotierten Raster nützlich.



Anmerkung
ST_Envelope rundet die Koordinaten und fügt so einen kleinen Puffer um den Raster hinzu. Dadurch unterscheidet sich das Ergebnis gering von ST_ConvexHull, das nicht rundet.

Beispiele

See `doc/development/internals/raster-affine.md` for the scale, skew, and rotation terms that make the convex hull and envelope differ for skewed rasters.

For regular non-skewed rasters, the envelope and convex hull are more or less the same.

Code

```
SELECT ST_ConvexHull(rast) As convhull, ST_Envelope(rast) As env
FROM dummy_rast WHERE rid=1;
```

Output

```
convhull                        |               env
--------------------------------------------------------+-----------------------------------
 POLYGON((0.5 0.5,20.5 0.5,20.5 60.5,0.5 60.5,0.5 0.5)) | 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.

Code

```
SELECT ST_ConvexHull(rast) As convhull, ST_Envelope(rast) As env
FROM (SELECT ST_SetRotation(rast, 0.1, 0.1) As rast
    FROM dummy_rast WHERE rid=1) As foo;
```

Output

```
convhull                        |               env
--------------------------------------------------------+-----------------------------------
 POLYGON((0.5 0.5,20.5 1.5,22.5 61.5,2.5 60.5,0.5 0.5)) | POLYGON((0 0,22 0,22 61,0 61,0 0))
```

Siehe auch

ST_Envelope, ST_MinConvexHull, ST_ConvexHull, ST_AsText

11.15.3 ST_DumpAsPolygons

ST_DumpAsPolygons — Gibt geomval (geom,val) Zeilen eines Rasterbandes zurück. Wenn kein Band angegeben ist, wird die Bandnummer standardmäßig auf 1 gesetzt.

Synopsis

setof geomval **ST_DumpAsPolygons**(raster rast, integer band_num=1, boolean exclude_nodata_value=TRUE);

Description

Dies ist eine Funktion mit Ergebnismenge (SRF von set-returning function). Sie gibt eine Menge an geomval Zeilen für das Band zurück, die aus einer Geometrie (geom) und einem Pixelwert (val) gebildet werden. Ein Polygon entspricht der Vereinigung der Pixel in dem Band, die denselben Pixelwert "val" aufweisen.

ST_DumpAsPolygon ist nützlich um Raster in Polygone zu vektorisieren. Im Gegensatz zu GROUP BY werden hier neue Zeilen erzeugt. Die Funktion kann zum Beispiel verwendet werden, um einen einzelnen Raster in mehrere Polygone/Mehrfach-Polygone zu expandieren.

Geändert 3.3.0, Validierung und Fixierung ist deaktiviert, um die Leistung zu verbessern. Kann zu ungültigen Geometrien führen.

Changed 3.7.0, the polygonization honours PostgreSQL interrupts so cancellations and statement timeouts halt processing promptly.

Verfügbarkeit: Benötigt GDAL 1.7+



Anmerkung
Wenn das Band einen NODATA-Wert aufweist, dann werden die Pixel mit diesem Wert nicht zurückgegeben, außer wenn exclude_nodata_value=false.



Anmerkung
Wenn Sie nur die Anzahl der Pixel des Raster benötigen, die einen bestimmten Zellwert haben, dann ist ST_ValueCount schneller.



Anmerkung
Dies unterscheidet sich von ST_PixelAsPolygons, wo eine Geometrie für jedes Pixel zurückgegeben wird, unabhängig vom Pixelwert.

Beispiele

This syntax uses a LATERAL join.

Code

```
SELECT val,geom As geomwkt
FROM (
SELECT dp.*
FROM dummy_rast, LATERAL ST_DumpAsPolygons(rast) AS dp
WHERE rid = 2
) As foo
WHERE val BETWEEN 249 and 251
ORDER BY val;
```

Output

```
val |                                                                        geomwkt
-----+-------------------------------------------------------------------------------
 249 | POLYGON((3427927.95 5793243.95,3427927.95 5793243.85,3427928 5793243.85,
        3427928 5793243.95,3427927.95 5793243.95))
 250 | POLYGON((3427927.75 5793243.9,3427927.75 5793243.85,3427927.8 5793243.85,
        3427927.8 5793243.9,3427927.75 5793243.9))
 250 | POLYGON((3427927.8 5793243.8,3427927.8 5793243.75,3427927.85 5793243.75,
        3427927.85 5793243.8, 3427927.8 5793243.8))
 251 | POLYGON((3427927.75 5793243.85,3427927.75 5793243.8,3427927.8 5793243.8,
        3427927.8 5793243.85,3427927.75 5793243.85))
```

Siehe auch

geomval, ST_AsRasterAgg, ST_Value, ST_Polygon, ST_ValueCount

11.15.4 ST_Envelope

ST_Envelope — Stellt die Ausdehnung des Raster als Polygon dar.

Synopsis

geometry **ST_Envelope**(raster rast);

Description

Gibt eine Polygondarstellung der Rasterausdehnung zurück. Dies ist das minimale umschreibendes Rechteck in Float8, das in dem durch die SRID festgelegten Koordinatenreferenzsystem als Polygon dargestellt wird.

Das Polygon wird durch die Eckpunkte des umschreibenden Rechtecks ((`MINX`, `MINY`), (`MINX`, `MAXY`), (`MAXX`, `MAXY`), (`MAXX`, `MINY`), (`MINX`, `MINY`)) festgelegt.

Beispiele

Code

```
SELECT rid, ST_Envelope(rast) As envgeomwkt
FROM dummy_rast;
```

Output

```
rid |                                    envgeomwkt
-----+---------------------------------------------------------------------
   1 | POLYGON((0 0,20 0,20 60,0 60,0 0))
   2 | POLYGON((3427927 5793243,3427928 5793243,
        3427928 5793244,3427927 5793244, 3427927 5793243))
```

Siehe auch

ST_Envelope, ST_AsText, ST_SRID

11.15.5 ST_MinConvexHull

ST_MinConvexHull — Gibt die Geometrie der konvexen Hülle des Raster aus, wobei Pixel mit NODATA ausgenommen werden.

Synopsis

geometry **ST_MinConvexHull**(raster rast, integer nband=NULL);

Description

Gibt die Geometrie der konvexen Hülle des Raster aus, wobei Pixel mit NODATA ausgenommen werden. Wenn `nband` NULL ist, dann werden alle Bänder des Raster berücksichtigt.

Verfügbarkeit: 2.1.0

Beispiele

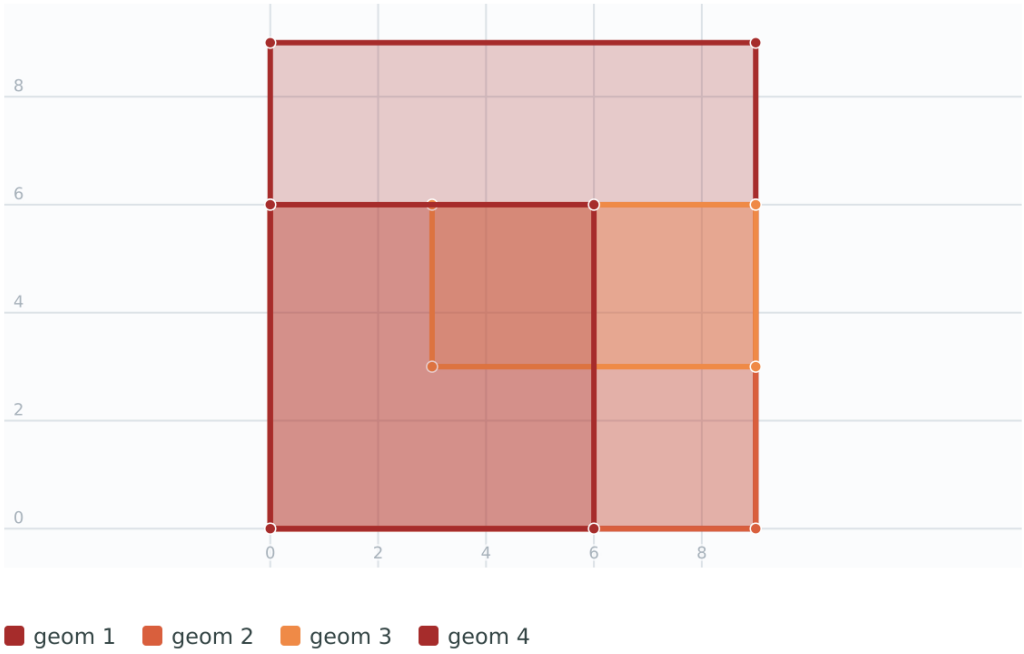
Code

```
WITH foo AS (
    SELECT
        ST_SetValues(ST_SetValues(ST_AddBand(ST_AddBand(ST_MakeEmptyRaster(9, 9, 0, 0, 1,  ↩
    -1, 0, 0, 0), 1, '8BUI', 0, 0), 2, '8BUI', 1, 0),
                1, 1, 1,
                ARRAY[
                    [0, 0, 0, 0, 0, 0, 0, 0, 0],
                    [0, 0, 0, 0, 0, 0, 0, 0, 0],
                    [0, 0, 0, 0, 0, 0, 0, 0, 0],
                    [0, 0, 0, 1, 0, 0, 0, 0, 1],
                    [0, 0, 0, 1, 1, 0, 0, 0, 0],
                    [0, 0, 0, 1, 0, 0, 0, 0, 0],
                    [0, 0, 0, 0, 0, 0, 0, 0, 0],
                    [0, 0, 0, 0, 0, 0, 0, 0, 0],
                    [0, 0, 0, 0, 0, 0, 0, 0, 0]
                ]::double precision[][]
            ),
            2, 1, 1,
            ARRAY[
                [0, 0, 0, 0, 0, 0, 0, 0, 0],
                [0, 0, 0, 0, 0, 0, 0, 0, 0],
                [0, 0, 0, 0, 0, 0, 0, 0, 0],
                [1, 0, 0, 0, 0, 1, 0, 0, 0],
                [0, 0, 0, 0, 1, 1, 0, 0, 0],
                [0, 0, 0, 0, 0, 1, 0, 0, 0],
                [0, 0, 0, 0, 0, 0, 0, 0, 0],
                [0, 0, 0, 0, 0, 0, 0, 0, 0],
                [0, 0, 1, 0, 0, 0, 0, 0, 0]
            ]::double precision[][]
        ) AS rast
)
SELECT title, geom AS geom
FROM foo
CROSS JOIN LATERAL (VALUES
    ('raster hull', ST_ConvexHull(rast)),
    ('all bands', ST_MinConvexHull(rast)),
    ('band 1', ST_MinConvexHull(rast, 1)),
    ('band 2', ST_MinConvexHull(rast, 2))
) AS variants(title, geom);
```

Output

```
title    |                 geom
---------+--------------------------------------
 raster hull | POLYGON((0 0,9 0,9 -9,0 -9,0 0))
 all bands   | POLYGON((0 -3,9 -3,9 -9,0 -9,0 -3))
 band 1      | POLYGON((3 -3,9 -3,9 -6,3 -6,3 -3))
 band 2      | POLYGON((0 -3,6 -3,6 -9,0 -9,0 -3))
```

Figure



Siehe auch

ST_Envelope, ST_ConvexHull, ST_ConvexHull, ST_AsText

11.15.6 ST_Polygon

ST_Polygon — Gibt eine Geometrie mit Mehrfachpolygonen zurück, die aus der Vereinigung von Pixel mit demselben Zellwert gebildet werden. Pixel mit NODATA Werten werden nicht berücksichtigt. Wenn keine Band angegeben ist, wird die Bandnummer standardmäßig auf 1 gesetzt.

Synopsis

geometry **ST_Polygon**(raster rast, integer band_num=1);

Description

Geändert 3.3.0, Validierung und Fixierung ist deaktiviert, um die Leistung zu verbessern. Kann zu ungültigen Geometrien führen.

Verfügbarkeit: 0.1.6 - benötigt GDAL 1.7+

Erweiterung: 2.1.0 Geschwindigkeit verbessert (zur Gänze C-basiert) und es wird sichergestellt, dass das zurückgegebenen Mehrfachpolygon valide ist.

Änderung: 2.1.0 In Vorgängerversionen wurde manchmal ein Polygon zurückgegeben; dies wurde geändert so dass jetzt immer ein Mehrfachpolygon zurückgegeben wird.

Beispiele

Compare polygonization of every pixel with polygonization after zero is assigned as the NODATA value. The latter preserves the gap formed by zero-valued cells.

Code

```
WITH source AS (
    SELECT ST_SetValues(
        ST_AddBand(
            ST_MakeEmptyRaster(4, 4, 0, 4, 1, -1, 0, 0, 0),
            1, '8BUI', 1, NULL
        ),
        1, 1, 1,
        ARRAY[
            [1, 1, 1, 1],
            [1, 0, 0, 1],
            [1, 1, 0, 1],
            [1, 1, 1, 1]
        ]::double precision[][]
    ) AS rast
), variants AS (
    SELECT 'all pixels' AS title, ST_Polygon(rast) AS geom
    FROM source
    UNION ALL
    SELECT
        'zero excluded',
        ST_Polygon(ST_SetBandNoDataValue(rast, 1, 0))
    FROM source
)
SELECT title, geom AS geom
FROM variants
ORDER BY CASE title
    WHEN 'all pixels' THEN 1
    ELSE 2
END;
```

Figure



Siehe auch

ST_Value, ST_DumpAsPolygons

11.15.7 ST_IntersectionFractions

ST_IntersectionFractions — Calculates the fraction of each raster cell that is covered by a given geometry.

Synopsis

raster **ST_IntersectionFractions**(raster rast, geometry geom);

Description

Calculates the fraction of each raster cell that is covered by a given geometry. The first argument is a raster, which defines the grid geometry to use for the calculation. The extent and cell size are read from the raster parameter. The second argument is a geometry, which is overlaid with the grid, and each grid populated based on overlaying the geometry on the grid. For polygons, the value returned for each cell is the proportion of its area that is covered by the geometry. For linestrings, the value returned for each cell is the length contained in the cell.

Availability: 3.6.0 Requires GEOS 3.14 or higher.

Beispiele

This rotated square polygon covers half of each cell in the raster.

Code

```
CREATE TABLE raster_proportions_rast (
    name text,
    rast raster
);

INSERT INTO raster_proportions_rast (name, rast) VALUES (
  '2x2 raster covering 0,0 to 10,10',
  ST_MakeEmptyRaster(2, 2, -- raster width/height in pixels
    0, 10, -- upper-left corner x/y coordinates
    5, -5, -- pixel width/height in ground units
    0, 0, -- skew x/y
    0        -- SRID
  ));

SELECT name, ST_DumpValues(
    ST_IntersectionFractions(
        rast,
        'POLYGON((5 0,0 5,5 10,10 5,5 0))'::geometry),
    1)
FROM raster_proportions_rast;
```

Output

```
2x2 raster covering 0,0 to 10,10
--------------------------------
 {{0.5,0.5},{0.5,0.5}}
```

Siehe auch

ST_MakeEmptyRaster

11.16 Rasteroperatoren

11.16.1 &&

&& — Gibt TRUE zurück, wenn das umschreibende Rechteck von A das umschreibende Rechteck von B schneidet.

Synopsis

boolean **&&**(raster A , raster B);
boolean **&&**(raster A , geometry B);
boolean **&&**(geometry B , raster A);

Description

Der Operator && gibt TRUE zurück, wenn das umschreibende Rechteck von Raster/Geometrie A das umschreibende Rechteck von Raster/Geometrie B schneidet.



Anmerkung
Dieser Operand benützt jeden Index, der für den Raster zur Verfügung stellt.

Verfügbarkeit: 2.0.0

Beispiele

Code

```
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;
```

Output

```
a_rid | b_rid | intersect
-------+-------+---------
     2 |     2 | t
     2 |     3 | f
     2 |     1 | f
```

11.16.2 &<

&< — Gibt TRUE zurück, wenn das umschreibende Rechteck von A links von dem von B liegt.

Synopsis

boolean **&<**(raster A , raster B);

Description

Der &< Operator gibt TRUE zurück, wenn das umschreibende Rechteck von Raster A das umschreibende Rechteck von Raster B überlagert oder links davon liegt, oder präziser, überlagert und NICHT rechts des umschreibenden Rechtecks von Raster B liegt.



Anmerkung
Dieser Operand benützt jeden Index, der für den Raster zur Verfügung stellt.

Beispiele

Code

```
SELECT A.rid As a_rid, B.rid As b_rid, A.rast &< B.rast As overleft
 FROM dummy_rast AS A CROSS JOIN dummy_rast AS B;
```

Output

```
a_rid | b_rid | overleft
------+-------+----------
    2 |     2 | t
    2 |     3 | f
    2 |     1 | f
    3 |     2 | t
    3 |     3 | t
    3 |     1 | f
    1 |     2 | t
    1 |     3 | t
    1 |     1 | t
```

11.16.3 &>

&> — Gibt TRUE zurück, wenn das umschreibende Rechteck von A rechts von dem von B liegt.

Synopsis

boolean **&>**(raster A , raster B);

Description

Der &> Operator gibt TRUE zurück, wenn das umschreibende Rechteck von Raster A das umschreibende Rechteck von Raster B überlagert oder rechts davon liegt, oder präziser, überlagert und NICHT links des umschreibenden Rechtecks von Raster B liegt.



Anmerkung
Dieser Operand benützt jeden Index, der für die Geometrie zur Verfügung stellt.

Beispiele

Code

```
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;
```

Output

```
 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 =

= — Gibt TRUE zurück, wenn die umschreibenden Rechtecke von A und B ident sind. Das umschreibende Rechteck ist in Double Precision.

Synopsis

boolean =(raster A , raster B);

Description

Der = Operator gibt TRUE zurück, wenn das umschreibende Rechteck von Raster A ident mit dem umschreibenden Rechteck von Raster B ist. PostgreSQL verwendet die =, <, und > Operatoren um die interne Sortierung und den Vergleich von Raster durchzuführen (z.B.: in einer GROUP BY oder ORDER BY Klausel).



Achtung
Dieser Operator verwendet NICHT die für Raster verfügbaren Indizes. Verwenden Sie bitte ~= stattdessen. Dieser Operator existiert meistens, so dass man nach der Rasterspalte gruppieren kann.

Verfügbarkeit: 2.1.0

Siehe auch

~=

11.16.5 @

@ — Gibt TRUE zurück, wenn das umschreibende Rechteck von A in jenem von B enthalten ist. Das umschreibende Rechteck ist in Double Precision.

Synopsis

boolean @(raster A , raster B);
boolean @(geometry A , raster B);
boolean @(raster B , geometry A);

Description

Der @ Operator gibt TRUE zurück, wenn das umschreibende Rechteck von Raster/Geometrie A in dem umschreibenden Rechteck von Raster/Geometrie B enthalten ist.



Anmerkung
Dieser Operand verwendet die räumichen Indizes der Raster.

Verfügbarkeit: 2.0.0 raster @ raster, und raster @ geometry eingeführt

Verfügbarkeit: 2.0.5 "geometry @ raster" eingeführt

Siehe auch

~

11.16.6 ~=

~= — Gibt TRUE zurück, wenn die bounding box von A ident mit jener von B ist.

Synopsis

boolean **~=**(raster A , raster B);

Description

Der Operator ~= gibt TRUE zurück, wenn die Umgebungsrechtecke der Raster "A" und "B" ident sind.



Anmerkung
Dieser Operand benützt jeden Index, der für den Raster zur Verfügung stellt.

Verfügbarkeit: 2.0.0

Beispiele

Ein sinnvoller Anwendungsfall wäre, aus zwei Einzelbandraster mit den gleichen Eigenschaften aber unterschiedlicher Thematik, einen Multi-Bandraster zu erstellen.

Code

```
SELECT ST_AddBand(prec.rast, alt.rast) As new_rast
FROM prec INNER JOIN alt ON (prec.rast ~= alt.rast);
```

Siehe auch

ST_AddBand, =

11.16.7 ~

~ — Gibt TRUE zurück, wenn das umschreibende Rechteck von A jenes von B enthält. Das umschreibende Rechteck ist in Double Precision.

Synopsis

boolean ~(raster A , raster B);
boolean ~(geometry A , raster B);
boolean ~(raster B , geometry A);

Description

Der Operator ~ gibt `TRUE` zurück, wenn das umschreibende Rechteck von Raster/Geometrie A das umschreibende Rechteck von Raster/Geometrie B enthält.



Anmerkung
Dieser Operand verwendet die räumichen Indizes der Raster.

Verfügbarkeit: 2.0.0

Siehe auch

@

11.17 Räumliche Beziehungen von Rastern und Rasterbändern

11.17.1 ST_Contains

ST_Contains — Gibt TRUE zurück, wenn kein Punkt des Rasters "rastB" im Äußeren des Rasters "rastA" liegt und zumindest ein Punkt im Inneren von "rastB" auch im Inneren von "rastA" liegt.

Synopsis

boolean **ST_Contains**(raster rastA , integer nbandA , raster rastB , integer nbandB);
boolean **ST_Contains**(raster rastA , raster rastB);

Description

Raster "rastA" enthält dann und nur dann "rastB", wenn sich kein Punkt von "rastB" im Äußeren des Rasters "rastA" befindet und zumindest ein Punkt im Inneren von "rastB" auch im Inneren von "rastA" liegt. Wenn die Bandnummer nicht angegeben ist (oder auf NULL gesetzt), dann wird nur die konvexe Hülle des Rasters zur Überprüfung herangezogen. Wenn die Bandnummer angegeben ist, werden nur die Pixel mit einem Wert (nicht NODATA) bei der Überprüfung herangezogen.



Anmerkung
Diese Funktion verwendet die für Raster verfügbaren Indizes.



Anmerkung
Um die räumliche Beziehung zwischen einem Raster und einem geometrischen Datentyp zu überprüfen, verwenden Sie bitte ST_Polygon für den Raster, z.B. ST_Contains(ST_Polygon(raster), geometry) or ST_Contains(geometry, ST_Polygon(raster)).



Anmerkung
ST_Contains() ist das Gegenstück zuf ST_Within(). Daher impliziert ST_Contains(rastA, rastB) ST_Within(rastB, rastA).

Verfügbarkeit: 2.1.0

Beispiele

Specified band numbers.

Code

```
SELECT r1.rid, r2.rid, ST_Contains(r1.rast, 1, r2.rast, 1) FROM dummy_rast r1 CROSS JOIN ↩
    dummy_rast r2 WHERE r1.rid = 1;
```

Output

```
NOTICE:  The first raster provided has no bands
 rid | rid | st_contains
-----+-----+-------------
   1 |   1 |
   1 |   2 | f
```

No band numbers specified.

Code

```
SELECT r1.rid, r2.rid, ST_Contains(r1.rast, r2.rast) FROM dummy_rast r1 CROSS JOIN ↩
    dummy_rast r2 WHERE r1.rid = 1;
```

Output

```
rid | rid | st_contains
-----+-----+-------------
   1 |   1 | t
   1 |   2 | f
```

Siehe auch

ST_Intersects, ST_Within

11.17.2 ST_ContainsProperly

ST_ContainsProperly — Gibt TRUE zurück, wenn "rastB" das Innere von "rastA" schneidet, aber nicht die Begrenzung oder das Äußere von "rastA".

Synopsis

boolean **ST_ContainsProperly**(raster rastA , integer nbandA , raster rastB , integer nbandB);
boolean **ST_ContainsProperly**(raster rastA , raster rastB);

Description

Raster "rastA" enthält "rastB" vollständig, wenn "rastB" nur das Innere von "rastA" schneidet, die Begrenzung und das Äußere von "rastA" jedoch nicht. Wenn die Bandnummer nicht angegeben ist (oder auf NULL gesetzt), dann wird nur die konvexe Hülle des Rasters zur Überprüfung herangezogen. Wenn die Bandnummer angegeben ist, werden nur die Pixel mit einem Wert (nicht NODATA) bei der Überprüfung herangezogen.

Raster "rastA" enthält sich selbst, aber enthält sich nicht zur Gänze selbst.



Anmerkung
Diese Funktion verwendet die für Raster verfügbaren Indizes.



Anmerkung
Um die räumliche Beziehung zwischen einem Raster und einem geometrischen Datentyp zu überprüfen, verwenden Sie bitte ST_Polygon für den Raster, z.B. ST_ContainsProperly(ST_Polygon(raster), geometry) or ST_ContainsProperly(geometry, ST_Polygon(raster)).

Verfügbarkeit: 2.1.0

Beispiele

Code

```
SELECT r1.rid, r2.rid, ST_ContainsProperly(r1.rast, 1, r2.rast, 1) FROM dummy_rast r1 CROSS ↩
    JOIN dummy_rast r2 WHERE r1.rid = 2;
```

Output

```
rid | rid | st_containsproperly
-----+-----+---------------------
   2 |   1 | f
   2 |   2 | f
```

Siehe auch

ST_Intersects, ST_Contains

11.17.3 ST_Covers

ST_Covers — Gibt TRUE zurück, wenn kein Punkt des Rasters "rastB" außerhalb des Rasters "rastA" liegt.

Synopsis

boolean **ST_Covers**(raster rastA , integer nbandA , raster rastB , integer nbandB);
boolean **ST_Covers**(raster rastA , raster rastB);

Description

Raster "rastA" deckt "rastB" dann und nur dann ab, wenn kein Punkt von "rastB" im Äußeren von "rastA" liegt. Wenn die Bandnummer nicht angegeben ist (oder auf NULL gesetzt), dann wird nur die konvexe Hülle des Rasters zur Überprüfung herangezogen. Wenn die Bandnummer angegeben ist, werden nur die Pixel mit einem Wert (nicht NODATA) bei der Überprüfung herangezogen.



Anmerkung
Diese Funktion verwendet die für Raster verfügbaren Indizes.



Anmerkung
Um die räumliche Beziehung zwischen einem Raster und einem geometrischen Datentyp zu überprüfen, verwenden Sie bitte ST_Polygon für den Raster, z.B. ST_Covers(ST_Polygon(raster), geometry) or ST_Covers(geometry, ST_Polygon(raster)).

Verfügbarkeit: 2.1.0

Beispiele

Code

```
SELECT r1.rid, r2.rid, ST_Covers(r1.rast, 1, r2.rast, 1) FROM dummy_rast r1 CROSS JOIN ↩
    dummy_rast r2 WHERE r1.rid = 2;
```

Output

```
rid | rid | st_covers
-----+-----+-----------
   2 |   1 | f
   2 |   2 | t
```

Siehe auch

ST_Intersects, ST_CoveredBy

11.17.4 ST_CoveredBy

ST_CoveredBy — Gibt TRUE zurück, wenn kein Punkt des Rasters "rastA" außerhalb des Rasters "rastB" liegt.

Synopsis

boolean **ST_CoveredBy**(raster rastA , integer nbandA , raster rastB , integer nbandB);
boolean **ST_CoveredBy**(raster rastA , raster rastB);

Description

Raster "rastA" wird von "rastB" dann und nur dann abgedeckt, wenn kein Punkt von "rastA" im Äußeren von "rastB" liegt. Wenn die Bandnummer nicht angegeben ist (oder auf NULL gesetzt), dann wird nur die konvexe Hülle des Rasters zur Überprüfung herangezogen. Wenn die Bandnummer angegeben ist, werden nur die Pixel mit einem Wert (nicht NODATA) bei der Überprüfung herangezogen.



Anmerkung
Diese Funktion verwendet die für Raster verfügbaren Indizes.



Anmerkung
Um die räumliche Beziehung zwischen einem Raster und einem geometrischen Datentyp zu überprüfen, verwenden Sie bitte ST_Polygon für den Raster, z.B. ST_CoveredBy(ST_Polygon(raster), geometry) or ST_CoveredBy(geometry, ST_Polygon(raster)).

Verfügbarkeit: 2.1.0

Beispiele

Code

```
SELECT r1.rid, r2.rid, ST_CoveredBy(r1.rast, 1, r2.rast, 1) FROM dummy_rast r1 CROSS JOIN ↩
    dummy_rast r2 WHERE r1.rid = 2;
```

Output

```
rid | rid | st_coveredby
-----+-----+--------------
   2 |   1 | f
   2 |   2 | t
```

Siehe auch

ST_Intersects, ST_Covers

11.17.5 ST_Disjoint

ST_Disjoint — Gibt TRUE zurück, wenn sich die Raster "rastA" und "rastB" räumlich nicht überschneiden.

Synopsis

boolean **ST_Disjoint**(raster rastA , integer nbandA , raster rastB , integer nbandB);
boolean **ST_Disjoint**(raster rastA , raster rastB);

Description

Raster "rastA" und "rastB" sind getrennt voneinander (disjoint) wenn sie sich keinen gemeinsamen Raum teilen. Wenn die Bandnummer nicht angegeben ist (oder auf NULL gesetzt), dann wird nur die konvexe Hülle des Rasters zur Überprüfung herangezogen. Wenn die Bandnummer angegeben ist, werden nur die Pixel mit einem Wert (nicht NODATA) bei der Überprüfung herangezogen.



Anmerkung
Diese Funktion verwendet keine Indizes.



Anmerkung
Um die räumliche Beziehung zwischen einem Raster und einem geometrischen Datentyp zu überprüfen, verwenden Sie bitte ST_Polygon für den Raster, z.B. ST_Disjoint(ST_Polygon(raster), geometry).

Verfügbarkeit: 2.1.0

Beispiele

Rid = 1 has no bands, hence the NOTICE and the NULL value for st_disjoint.

Code

```
SELECT r1.rid, r2.rid, ST_Disjoint(r1.rast, 1, r2.rast, 1) FROM dummy_rast r1 CROSS JOIN ↩
    dummy_rast r2 WHERE r1.rid = 2;
```

Output

```
NOTICE:  The second raster provided has no bands
 rid | rid | st_disjoint
-----+-----+-------------
   2 |   1 |
   2 |   2 | f
```

This time, without specifying band numbers.

Code

```
SELECT r1.rid, r2.rid, ST_Disjoint(r1.rast, r2.rast) FROM dummy_rast r1 CROSS JOIN ↩
    dummy_rast r2 WHERE r1.rid = 2;
```

Output

```
rid | rid | st_disjoint
-----+-----+-------------
   2 |   1 | t
   2 |   2 | f
```

Siehe auch

ST_Intersects

11.17.6 ST_Intersects

ST_Intersects — Gibt TRUE zurück, wenn sich die Raster "rastA" und "rastB" nicht räumlich überschneiden.

Synopsis

boolean **ST_Intersects**(raster rastA , integer nbandA , raster rastB , integer nbandB);
boolean **ST_Intersects**(raster rastA , raster rastB);
boolean **ST_Intersects**(raster rast , integer nband , geometry geommin);
boolean **ST_Intersects**(raster rast , geometry geommin , integer nband=NULL);
boolean **ST_Intersects**(geometry geommin , raster rast , integer nband=NULL);

Description

Gibt TRUE zurück, wenn der Raster "rastA" den Raster "rastB" räumlich schneidet. Wenn die Bandnummer nicht angegeben ist (oder auf NULL gesetzt), dann wird nur die konvexe Hülle des Rasters zur Überprüfung herangezogen. Wenn die Bandnummer angegeben ist, werden nur die Pixel mit einem Wert (nicht NODATA) bei der Überprüfung herangezogen.



Anmerkung
Diese Funktion verwendet die für Raster verfügbaren Indizes.

Erweiterung: 2.0.0 Unterstützung für Raster/Raster Verschneidungen eingeführt.



Warnung
Änderung: 2.1.0 Die Verhaltensweise der Varianten von ST_Intersects(raster, geometry) wurde geändert um mit dem von ST_Intersects(geometry, raster) übereinzustimmen.

Beispiele

Different bands of same raster.

Code

```
SELECT ST_Intersects(rast, 2, rast, 3) FROM dummy_rast WHERE rid = 2;
```

Output

```
t
```

Siehe auch

ST_Intersection, ST_Disjoint

11.17.7 ST_Overlaps

ST_Overlaps — Gibt TRUE zurück, wenn sich die Raster "rastA" und "rastB" schneiden, aber ein Raster den anderen nicht zur Gänze enthält.

Synopsis

boolean **ST_Overlaps**(raster rastA , integer nbandA , raster rastB , integer nbandB);
boolean **ST_Overlaps**(raster rastA , raster rastB);

Description

Gibt TRUE zurück, wenn der Raster "rastA" den Raster "rastB" überlagert. Dies bedeutet, das rastA und rastB sich schneiden aber einer den anderen nicht zur Gänze enthält. Wenn die Bandnummer nicht angegeben ist (oder auf NULL gesetzt), dann wird nur die konvexe Hülle des Rasters zur Überprüfung herangezogen. Wenn die Bandnummer angegeben ist, werden nur die Pixel mit einem Wert (nicht NODATA) bei der Überprüfung herangezogen.



Anmerkung
Diese Funktion verwendet die für Raster verfügbaren Indizes.



Anmerkung
Um die räumliche Beziehung zwischen einem Raster und einem geometrischen Datentyp zu überprüfen, verwenden Sie bitte ST_Polygon für den Raster, z.B. ST_Overlaps(ST_Polygon(raster), geometry).

Verfügbarkeit: 2.1.0

Beispiele

Comparing different bands of same raster.

Code

```
SELECT ST_Overlaps(rast, 1, rast, 2) FROM dummy_rast WHERE rid = 2;
```

Output

```
f
```

Siehe auch

ST_Intersects

11.17.8 ST_Touches

ST_Touches — Gibt TRUE zurück, wenn rastA und rastB zumindest einen Punkt gemeinsam haben sich aber nicht überschneiden.

Synopsis

boolean **ST_Touches**(raster rastA , integer nbandA , raster rastB , integer nbandB);
boolean **ST_Touches**(raster rastA , raster rastB);

Description

Gibt TRUE zurück, wenn der Raster "rastA" den Raster "rastB" räumlich berührt. Dies bedeutet, das rastA und rastB zumindest einen Punkt gemeinsam haben, ihr Inneres sich aber nicht überschneidet. Wenn die Bandnummer nicht angegeben ist (oder auf NULL gesetzt), dann wird nur die konvexe Hülle des Rasters zur Überprüfung herangezogen. Wenn die Bandnummer angegeben ist, werden nur die Pixel mit einem Wert (nicht NODATA) bei der Überprüfung herangezogen.



Anmerkung
Diese Funktion verwendet die für Raster verfügbaren Indizes.



Anmerkung
Um die räumliche Beziehung zwischen einem Raster und einem geometrischen Datentyp zu überprüfen, verwenden Sie bitte ST_Polygon für den Raster, z.B. ST_Touches(ST_Polygon(raster), geometry).

Verfügbarkeit: 2.1.0

Beispiele

Code

```
SELECT r1.rid, r2.rid, ST_Touches(r1.rast, 1, r2.rast, 1) FROM dummy_rast r1 CROSS JOIN ↩
    dummy_rast r2 WHERE r1.rid = 2;
```

Output

```
rid | rid | st_touches
-----+-----+------------
   2 |   1 | f
   2 |   2 | f
```

Siehe auch

ST_Intersects

11.17.9 ST_SameAlignment

ST_SameAlignment — Gibt TRUE zurück, wenn die Raster die selbe Rotation, Skalierung, Koordinatenreferenzsystem und Versatz (Pixel können auf dasselbe Gitter gelegt werden, ohne dass die Gitterlinien durch die Pixel schneiden) aufweisen. Wenn nicht, wird FALSE und eine Bechreibung des Problems ausgegeben.

Synopsis

boolean **ST_SameAlignment**(raster rastA , raster rastB);
boolean **ST_SameAlignment**(double precision ulx1 , double precision uly1 , double precision scalex1 , double precision scaley1 , double precision skewx1 , double precision skewy1 , double precision ulx2 , double precision uly2 , double precision scalex2 , double precision scaley2 , double precision skewx2 , double precision skewy2);
boolean **ST_SameAlignment**(raster set rastfield);

Description

Nicht-Aggregat Version (Versionen 1 und 2): Gibt TRUE zurück, wenn die zwei Raster (die entweder direkt übergeben oder mit Werten für UpperLeft, Scale, Skew und SRID erstellt werden) dieselbe Skalierung, Rotation und SRID haben und zumindest eine der vier Ecken eines Pixel des einen Raster auf eine Ecke des anderen Rastergitters fällt. Wenn nicht, wird FALSE und eine Problembeschreibung ausgegeben.

Aggregat Version (Variante 3): Gibt TRUE zurück, wenn alle übergebenen Raster gleich ausgerichtet sind. Die Funktion ST_SameAlignm ist in der Terminologie von PostgreSQL eine Aggregatfunktion. Dies bedeutet, dass sie so wie die Funktionen SUM() und AVG() mit Datenzeilen arbeitet.

Verfügbarkeit: 2.0.0

Erweiterung: 2.1.0 die Variante mit der Aggregatfunktion hinzugefügt

Beispiele: Raster

Code

```
SELECT ST_SameAlignment(ST_MakeEmptyRaster(1, 1, 0, 0, 1, 1, 0, 0),
    ST_MakeEmptyRaster(1, 1, 0, 0, 1, 1, 0, 0)
) as sm;
```

Output

```
t
```

Code

```
SELECT ST_SameAlignment(A.rast, b.rast)
FROM dummy_rast AS A CROSS JOIN dummy_rast AS B;
```

Output

```
NOTICE:  The two rasters provided have different SRIDs
NOTICE:  The two rasters provided have different SRIDs
 t
 f
 f
 f
```

Siehe auch

Abschnitt 10.1, ST_NotSameAlignmentReason, ST_MakeEmptyRaster

11.17.10 ST_NotSameAlignmentReason

ST_NotSameAlignmentReason — Gibt eine Meldung aus, die angibt ob die Raster untereinander ausgerichtet sind oder nicht und warum wenn nicht.

Synopsis

text **ST_NotSameAlignmentReason**(raster rastA, raster rastB);

Description

Gibt eine Meldung aus, die angibt ob die Raster untereinander ausgerichtet sind oder nicht und warum wenn nicht.



Anmerkung
Falls es mehrere Gründe gibt, warum die Raster nicht untereinander ausgerichtet sind, so wird nur der erste Grund (die erste fehlgeschlagene Überprüfung) ausgegeben.

Verfügbarkeit: 2.1.0

Beispiele

Code

```
SELECT
    ST_SameAlignment(ST_MakeEmptyRaster(1, 1, 0, 0, 1, 1, 0, 0),
        ST_MakeEmptyRaster(1, 1, 0, 0, 1.1, 1.1, 0, 0)
    ),
    ST_NotSameAlignmentReason(ST_MakeEmptyRaster(1, 1, 0, 0, 1, 1, 0, 0),
        ST_MakeEmptyRaster(1, 1, 0, 0, 1.1, 1.1, 0, 0)
    )
;
```

Output

```
st_samealignment |            st_notsamealignmentreason
-----------------+-----------------------------------------------------
 f               | The rasters have different scales on the X axis
(1 row)
```

Siehe auch

Abschnitt 10.1, ST_SameAlignment

11.17.11 ST_Within

ST_Within — Gibt TRUE zurück, wenn kein Punkt des Rasters "rastA" außerhalb des Rasters "rastB" liegt und zumindest ein Punkt im Inneren von "rastA" auch im Inneren von "rastB" liegt.

Synopsis

boolean **ST_Within**(raster rastA , integer nbandA , raster rastB , integer nbandB);
boolean **ST_Within**(raster rastA , raster rastB);

Description

Raster "rastA" liegt dann und nur dann "within"/innerhalb von "rastB", wenn sich kein Punkt von "rastA" im Äußeren des Rasters "rastB" befindet und zumindest ein Punkt im Inneren von "rastA" auch im Inneren von "rastB" liegt. Wenn die Bandnummer nicht angegeben ist (oder auf NULL gesetzt), dann wird nur die konvexe Hülle des Rasters zur Überprüfung herangezogen. Wenn die Bandnummer angegeben ist, werden nur die Pixel mit einem Wert (nicht NODATA) bei der Überprüfung herangezogen.



Anmerkung
Dieser Operand benützt jeden Index, der für den Raster zur Verfügung stellt.



Anmerkung
Um die räumliche Beziehung zwischen einem Raster und einer Geometrie zu überprüfen, verwenden Sie bitte ST_Polygon für den Raster, z.B. ST_Within(ST_Polygon(raster), geometry) or ST_Within(geometry, ST_Polygon(raster)).



Anmerkung
ST_Within() ist die Umkehrfunktion von ST_Contains(). Es gilt daher: ST_Within(rastA, rastB) impliziert ST_Contains(rastB, rastA).

Verfügbarkeit: 2.1.0

Beispiele

Code

```
SELECT r1.rid, r2.rid, ST_Within(r1.rast, 1, r2.rast, 1) FROM dummy_rast r1 CROSS JOIN ↩
    dummy_rast r2 WHERE r1.rid = 2;
```

Output

```
rid | rid | st_within
-----+-----+-----------
   2 |   1 | f
   2 |   2 | t
```

Siehe auch

ST_Intersects, ST_Contains, ST_DWithin, ST_DFullyWithin

11.17.12 ST_DWithin

ST_DWithin — Gibt TRUE zurück, wenn die Raster "rastA" und "rastB" innerhalb der angegebenen Entfernung voneinander liegen.

Synopsis

boolean **ST_DWithin**(raster rastA , integer nbandA , raster rastB , integer nbandB , double precision distance_of_srid);
boolean **ST_DWithin**(raster rastA , raster rastB , double precision distance_of_srid);

Description

Gibt TRUE zurück, wenn die Raster "rastA" und "rastB" innerhalb der angegebenen Distanz zueinander liegen. Wenn die Bandnummer nicht angegeben ist (oder auf NULL gesetzt), dann wird nur die konvexe Hülle des Rasters zur Überprüfung herangezogen. Wenn die Bandnummer angegeben ist, werden nur die Pixel mit einem Wert (nicht NODATA) bei der Überprüfung herangezogen.

Die Distanz wird in den Einheiten des Koordinatenreferenzsystems des Rasters angegeben. Damit diese Funktion Sinn hat, müssen die Quellraster dieselbe Projektion und die gleiche SRID haben.



Anmerkung
Dieser Operand benützt jeden Index, der für den Raster zur Verfügung stellt.



Anmerkung
Um die räumliche Beziehung zwischen einem Raster und einer Geometrie zu untersuchen, wenden Sie bitte ST_Polygon auf den Raster an, z.B.: ST_DWithin(ST_Polygon(raster), geometry).

Verfügbarkeit: 2.1.0

Beispiele

Code

```
SELECT r1.rid, r2.rid, ST_DWithin(r1.rast, 1, r2.rast, 1, 3.14) FROM dummy_rast r1 CROSS ↩
    JOIN dummy_rast r2 WHERE r1.rid = 2;
```

Output

```
rid | rid | st_dwithin
-----+-----+------------
   2 |   1 | f
   2 |   2 | t
```

Siehe auch

ST_Within, ST_DFullyWithin

11.17.13 ST_DFullyWithin

ST_DFullyWithin — Gibt TRUE zurück, wenn die Raster "rastA" und "rastB" zur Gänze innerhalb der angegebenen Distanz zueinander liegen.

Synopsis

boolean **ST_DFullyWithin**(raster rastA , integer nbandA , raster rastB , integer nbandB , double precision distance_of_srid);
boolean **ST_DFullyWithin**(raster rastA , raster rastB , double precision distance_of_srid);

Description

Gibt TRUE zurück, wenn die Raster "rastA" und "rastB" zur Gänze innerhalb der angegebenen Distanz zueinander liegen. Wenn die Bandnummer nicht angegeben ist (oder auf NULL gesetzt), dann wird nur die konvexe Hülle des Rasters zur Überprüfung herangezogen. Wenn die Bandnummer angegeben ist, werden nur die Pixel mit einem Wert (nicht NODATA) bei der Überprüfung herangezogen.

Die Distanz wird in den Einheiten des Koordinatenreferenzsystems des Rasters angegeben. Damit diese Funktion Sinn hat, müssen die Quellraster dieselbe Projektion und die gleiche SRID haben.



Anmerkung
Dieser Operand benützt jeden Index, der für den Raster zur Verfügung stellt.



Anmerkung
Um die räumliche Relation zwischen einem Raster und einer Geometrie zu untersuchen, wenden Sie bitte ST_Polygon auf den Raster an, z.B.: ST_DFullyWithin(ST_Polygon(raster), geometry).

Verfügbarkeit: 2.1.0

Beispiele

Code

```
SELECT r1.rid, r2.rid, ST_DFullyWithin(r1.rast, 1, r2.rast, 1, 3.14) FROM dummy_rast r1 ↩
    CROSS JOIN dummy_rast r2 WHERE r1.rid = 2;
```

Output

```
rid | rid | st_dfullywithin
-----+-----+-----------------
   2 |   1 | f
   2 |   2 | t
```

Siehe auch

ST_Within, ST_DWithin

11.18 Raster Tipps

11.18.1 Out-DB Raster

11.18.1.1 Das Verzeichnis enthält eine Vielzahl an Dateien

Wenn GDAL eine Datei öffnet, dann liest es eifrig das gesamte Verzeichnis in dem sich die Datei befindet um einen Katalog mit den weieren Dateien zu erstellen. Wenn dieses Verzeichnis viele Dateien (z.B.: Tausende, Millionen) enthält, kann das Öffnen dieser Datei extrem lange dauern (insbesondere wenn sich die Datei auf einem Netzlaufwerk, wie einem NFS befindet).

Dieses Verhalten kann durch folgende Umgebungsvariable von GDAL beeinflusst werden: GDAL_DISABLE_READDIR_ON_OPEN. Setzen Sie `GDAL_DISABLE_READDIR_ON_OPEN` auf `TRUE` um das Scannen von Verzeichnissen zu verhindern.

Auf Ubuntu (angenommen Sie verwenden ein PostgreSQL Paket für Ubuntu), kann `GDAL_DISABLE_READDIR_ON_OPEN` in */etc/postgresql/POSTGRESQL_VERSION/CLUSTER_NAME/environment* gesetzt werden (wobei POSTGRESQL_VERSION der Version von PostgreSQL entspricht, z.B. 9.6 und CLUSTER_NAME der Bezeichnung des Datenbankclusters, z.B. maindb). Sie können hier ebenso die Umgebungsvariablen von PostGIS setzen.

Code

```
# environment variables for postmaster process
# This file has the same syntax as postgresql.conf:
#  VARIABLE = simple_value
#  VARIABLE2 = 'any value!'
# I. e. you need to enclose any value which does not only consist of letters,
# numbers, and '-', '_', '.' in single quotes. Shell commands are not
# evaluated.
POSTGIS_GDAL_ENABLED_DRIVERS = 'ENABLE_ALL'

POSTGIS_ENABLE_OUTDB_RASTERS = 1

GDAL_DISABLE_READDIR_ON_OPEN = 'TRUE'
```

11.18.1.2 Die maximale Anzahl geöffneter Dateien

Die Einstellungen von Linux und PostgreSQL bezüglich der maximal erlaubten Anzahl von offenen Dateien sind üblicherweise sehr konservativ (normalerweise 1024 offene Dateien pro Prozess), da sie unter der Annahme getroffen wurden, dass das System von Menschen genutzt wird. Bei Out-DB Rastern kann eine einzelne Abfrage spielend dieses Limit überschreiten (z.B. ein Datensatz mit Rasterwerten über 10 Jahre, wobei ein Raster die Tageswerte der niedrigsten und höchsten Temperaturwerte enthält und wir den absoluten Mindest- und Höchstwert des Datensatzes abfragen wollen).

Am einfachsten kann dies über die PostgreSQL Einstellung max_files_per_process geändert werden. Der Standardwert von 1000 ist für Out-DB Raster viel zu niedrig. Ein zuverlässiger Anfangswert könnte 65536 sein, wobei dies jedoch sehr stark von den verwendeten Datensätzen abhängt und den Abfragen die Sie auf diese ausführen wollen. Diese Einstellung muss vor dem Starten des Servers gesetzt werden und kann vermutlich nur in der Konfigurationsdatei von PostgreSQL (z.B. /etc/postgresql/POSTGRESQL_VERSION/CLUSTER_NAME/postgresql.conf auf Ubuntu) vorgenommen werden.

Code

```
...
# - Kernel Resource Usage -

max_files_per_process = 65536           # min 25
                                        # (change requires restart)
...
```

Die wesentliche Änderung muss an den Limits des Linux Kernels für offene Dateien vorgenommen werden. Dies umfasst zwei Teile:

- Die maximale Anzahl geöffneter Dateien für das ganze System
- Die maximale Anzahl geöffneter Dateien pro Prozess

11.18.1.2.1 Die maximale Anzahl geöffneter Dateien für das ganze System

Das folgende Beispiel zeigt, wie Sie die aktuelle Einstellung zu der maximalen Anzahl geöffneter Dateien für das ganze System anzeigen können:

Code

```
$ sysctl -a | grep fs.file-max
fs.file-max = 131072
```

Wenn der ausgegebene Wert zu niedrig ist, können Sie wie im folgenden Beispiel gezeigt wird, eine Datei zu */etc/sysctl.d/* hinzufügen:

Code

```
$ echo "fs.file-max = 6145324" >> /etc/sysctl.d/fs.conf

$ cat /etc/sysctl.d/fs.conf
fs.file-max = 6145324

$ sysctl -p --system
* Applying /etc/sysctl.d/fs.conf ...
fs.file-max = 2097152
* Applying /etc/sysctl.conf ...

$ sysctl -a | grep fs.file-max
fs.file-max = 6145324
```

11.18.1.2.2 Die maximale Anzahl geöffneter Dateien pro Prozess

Die maximale Anzahl geöffneter Dateien pro Process für die Serverprozesse von PostgreSQL sollten geändert werden.

Um die maximale Anzahl geöffneter Dateien herauszufinden, welche von den Prozessen des PostgreSQL Dienstes genutzt werden, können Sie folgendes ausführen (stellen Sie sicher, dass PostgreSQL läuft):

Code

```
$ ps aux | grep postgres
```

Output

```
postgres 31713  0.0  0.4 179012 17564 pts/0    S    Dec26   0:03 /home/dustymugs/devel/postgresql/sandbox/10/usr/local/ ↩
     bin/postgres -D /home/dustymugs/devel/postgresql/sandbox/10/pgdata
postgres 31716  0.0  0.8 179776 33632 ?        Ss   Dec26   0:01 postgres: checkpointer process
postgres 31717  0.0  0.2 179144  9416 ?        Ss   Dec26   0:05 postgres: writer process
postgres 31718  0.0  0.2 179012  8708 ?        Ss   Dec26   0:06 postgres: wal writer process
postgres 31719  0.0  0.1 179568  7252 ?        Ss   Dec26   0:03 postgres: autovacuum launcher process
postgres 31720  0.0  0.1  34228  4124 ?        Ss   Dec26   0:09 postgres: stats collector process
postgres 31721  0.0  0.1 179308  6052 ?        Ss   Dec26   0:00 postgres: bgworker: logical replication launcher
```

Code

```
$ cat /proc/31718/limits
```

Output

```
Limit                     Soft Limit           Hard Limit           Units
Max cpu time              unlimited            unlimited            seconds
Max file size             unlimited            unlimited            bytes
Max data size             unlimited            unlimited            bytes
Max stack size            8388608              unlimited            bytes
Max core file size        0                    unlimited            bytes
Max resident set          unlimited            unlimited            bytes
Max processes             15738                15738                processes
Max open files            1024                 4096                 files
Max locked memory         65536                65536                bytes
Max address space         unlimited            unlimited            bytes
Max file locks            unlimited            unlimited            locks
Max pending signals       15738                15738                signals
Max msgqueue size         819200               819200               bytes
Max nice priority         0                    0
Max realtime priority     0                    0
Max realtime timeout      unlimited            unlimited            us
```

Im oberen Beispiel haben wir die Limits für geöffnete Dateien für den Prozess 31718 ausgelesen. Es spielt dabei keine Rolle welcher Prozess von PostgreSQL, es genügt jeder. Von Interesse ist dabei die Rückmeldung von *Max open files*.

Wir wollen das *Soft Limit* und das *Hard Limit* für die *Max open files* so erhöhen, dass sie über dem Wert liegen den wir in der Einstellung von PostgreSQL für `max_files_per_process` angegeben haben. In unserem Beispiel haben wir `max_files_per_proc` mit 65536 angegeben.

Auf Ubuntu (angenommen Sie verwenden ein PostgreSQL Paket für Ubuntu), kann das *Soft Limit* und das *Hard Limit* am einfachsten durch editieren von **/etc/init.d/postgresql** (SysV) oder **/lib/systemd/system/postgresql*.service** (systemd) geändert werden.

Befassen wir uns zuerst mit dem Fall SysV auf Ubuntu, bei dem wir **ulimit -H -n 262144** und **ulimit -n 131072** zu **/etc/init.d/postgresql** hinzufügen.

Code

```
...
case "$1" in
    start|stop|restart|reload)
        if [ "$1" = "start" ]; then
            create_socket_directory
        fi
    if [ -z "`pg_lsclusters -h`" ]; then
        log_warning_msg 'No PostgreSQL clusters exist; see "man pg_createcluster"'
        exit 0
    fi

    ulimit -H -n 262144
    ulimit -n 131072

    for v in $versions; do
        $1 $v || EXIT=$?
    done
    exit ${EXIT:-0}
        ;;
    status)
...
```

Nun der Fall mit systemd unter Ubuntu. Wir fügen **LimitNOFILE=131072** in jeder Datei **/lib/systemd/system/postgresql*.service** in dem Abschnitt **[Service]** ein.

Code

```
...
[Service]

LimitNOFILE=131072

...

[Install]
WantedBy=multi-user.target
...
```

Nach den erforderlichen systemd Änderungen müssen Sie den Dämon neu laden

Code

```
systemctl daemon-reload
```

Kapitel 12

PostGIS Extras

Dieses Kapitel beschreibt Funktionen, die sich in dem Verzeichnis "extras" des PostGIS Quellcodes (Tarball oder Repository) befinden. Diese sind nicht immer mit der binären PostGIS Release paketiert, es handelt sich dabei aber üblicherweise um Pl/Pgsql- oder Shell-Skripts, die direkt aufgerufen werden können.

Kapitel 13

PostGIS Spezialfunktionen Index

13.1 PostGIS-Aggregat-Funktionen

Bei den folgenden Funktionen handelt es sich um räumliche Aggregatfunktionen, die auf die gleiche Weise wie SQL-Aggregatfunktionen wie `sum` und `average` verwendet werden.

- CG_3DUnion - Perform 3D union using postgis_sfcgal.
- ST_3DExtent - Aggregatfunktion, die den 3D-Begrenzungsrahmen von Geometrien zurückgibt.
- ST_AsFlatGeobuf - Rückgabe einer FlatGeobuf-Darstellung einer Reihe von Zeilen.
- ST_AsGeobuf - Gibt eine Menge an Zeilen in der Geobuf Darstellung aus.
- ST_AsMVT - Aggregatfunktion, die eine MVT-Darstellung einer Reihe von Zeilen zurückgibt.
- ST_ClusterDBSCAN - Fensterfunktion, die eine Cluster-ID für jede Eingabegeometrie unter Verwendung des DBSCAN-Algorithmus zurückgibt.
- ST_ClusterIntersecting - Aggregatfunktion, die Eingabegeometrien zu zusammenhängenden Mengen clustert.
- ST_ClusterIntersectingWin - Fensterfunktion, die für jede Eingabegeometrie eine Cluster-ID zurückgibt und die Eingabegeometrien in zusammenhängende Gruppen clustert.
- ST_ClusterKMeans - Fensterfunktion, die eine Cluster-ID für jede Eingabegeometrie unter Verwendung des K-Means-Algorithmus zurückgibt.
- ST_ClusterRelateWin - Window function that returns a cluster id for each input geometry, clustering input geometries into connected sets using the relate pattern to determine whether the geometries are connected.
- ST_ClusterWithin - Aggregatfunktion, die Geometrien nach Trennungsabstand gruppiert.
- ST_ClusterWithinWin - Fensterfunktion, die eine Cluster-ID für jede Eingabegeometrie zurückgibt, Clustering anhand des Trennungsabstands.
- ST_Collect - Erzeugt eine GeometryCollection oder Multi*-Geometrie aus einer Reihe von Geometrien.
- ST_CoverageClean - Computes a clean (edge matched, non-overlapping, gap-cleared) polygonal coverage, given a non-clean input.
- ST_CoverageInvalidEdges - Fensterfunktion, die Stellen findet, an denen die Polygone keine gültige Abdeckung bilden.
- ST_CoverageSimplify - Fensterfunktion, die die Kanten einer polygonalen Abdeckung vereinfacht.
- ST_CoverageUnion - Berechnet die Vereinigung einer Menge von Polygonen, die eine Abdeckung bilden, indem gemeinsame Kanten entfernt werden.

- ST_Extent - Aggregatfunktion, die den Begrenzungsrahmen von Geometrien zurückgibt.
- ST_MakeLine - Erzeugt einen Linienzug aus einer Punkt-, Mehrfachpunkt- oder Liniengeometrie.
- ST_MemUnion - Aggregatfunktion, die Geometrien auf eine speichereffiziente, aber langsamere Weise zusammenfasst
- ST_MinimumSpanningTree - Window function that returns a tree id for each input geometry, clustering input geometries into connected trees using the Minimum Spanning Tree algorithm.
- ST_Polygonize - Berechnet eine Sammlung von Polygonen, die aus dem Linienwerk einer Reihe von Geometrien gebildet werden.
- ST_SameAlignment - Gibt TRUE zurück, wenn die Raster die selbe Rotation, Skalierung, Koordinatenreferenzsystem und Versatz (Pixel können auf dasselbe Gitter gelegt werden, ohne dass die Gitterlinien durch die Pixel schneiden) aufweisen. Wenn nicht, wird FALSE und eine Bechreibung des Problems ausgegeben.
- ST_Union - Berechnet eine Geometrie, die die Punktmengenvereinigung der Eingabegeometrien darstellt.
- ST_Union - Gibt die Vereinigung mehrerer Rasterkacheln in einem einzelnen Raster mit mehreren Bändern zurück.
- TopoElementArray_Agg - Gibt für eine Menge an element_id, type Feldern (topoelements) ein topoelementarray zurück.

13.2 PostGIS-Fenster-Funktionen

Bei den folgenden Funktionen handelt es sich um räumliche Fensterfunktionen, die auf die gleiche Weise wie SQL-Fensterfunktionen wie `row_number()`, `lead()` und `lag()` verwendet werden. Sie müssen von einer `OVER()` Klausel gefolgt werden.

- ST_ClusterDBSCAN - Fensterfunktion, die eine Cluster-ID für jede Eingabegeometrie unter Verwendung des DBSCAN-Algorithmus zurückgibt.
- ST_ClusterIntersectingWin - Fensterfunktion, die für jede Eingabegeometrie eine Cluster-ID zurückgibt und die Eingabegeometrien in zusammenhängende Gruppen clustert.
- ST_ClusterKMeans - Fensterfunktion, die eine Cluster-ID für jede Eingabegeometrie unter Verwendung des K-Means-Algorithmus zurückgibt.
- ST_ClusterRelateWin - Window function that returns a cluster id for each input geometry, clustering input geometries into connected sets using the relate pattern to determine whether the geometries are connected.
- ST_ClusterWithinWin - Fensterfunktion, die eine Cluster-ID für jede Eingabegeometrie zurückgibt, Clustering anhand des Trennungsabstands.
- ST_CoverageClean - Computes a clean (edge matched, non-overlapping, gap-cleared) polygonal coverage, given a non-clean input.
- ST_CoverageInvalidEdges - Fensterfunktion, die Stellen findet, an denen die Polygone keine gültige Abdeckung bilden.
- ST_CoverageSimplify - Fensterfunktion, die die Kanten einer polygonalen Abdeckung vereinfacht.
- ST_MinimumSpanningTree - Window function that returns a tree id for each input geometry, clustering input geometries into connected trees using the Minimum Spanning Tree algorithm.

13.3 PostGIS SQL-MM-kompatible Funktionen

Die folgenden Funktionen sind PostGIS-Funktionen, die dem SQL/MM 3-Standard entsprechen

- CG_3DArea - Berechnet die Fläche von 3D-Oberflächengeometrien. Gibt 0 für Solids zurück.
- CG_3DDifference - 3D-Differenz durchführen

- CG_3DIntersection - 3D-Schnitte durchführen
- CG_3DUnion - Perform 3D union using postgis_sfcgal.
- CG_Volume - Computes the volume of a 3D geometry. Closed surfaces can have non-zero volume.
- ST_3DDWithin - Prüft, ob zwei 3D-Geometrien innerhalb eines bestimmten 3D-Abstands liegen
- ST_3DDistance - Für den geometrischen Datentyp. Es wird der geringste 3-dimensionale kartesische Abstand (basierend auf dem Koordinatenreferenzsystem) zwischen zwei geometrischen Objekten in projizierten Einheiten zurückgegeben.
- ST_3DIntersects - Prüft, ob sich zwei Geometrien in 3D räumlich schneiden - nur für Punkte, Linienzüge, Polygone, polyedrische Flächen (Bereich)
- ST_3DLength - Gibt den geometrischen Schwerpunkt einer Geometrie zurück.
- ST_3DPerimeter - Gibt den geometrischen Schwerpunkt einer Geometrie zurück.
- ST_AddEdgeModFace - Fügt eine Kante hinzu. Falls dabei eine Masche aufgetrennt wird, so wird die ursprüngliche Masche angepasst und eine weitere Masche hinzugefügt.
- ST_AddEdgeNewFaces - Fügt eine Kante hinzu. Falls dabei eine Masche aufgetrennt wird, so wird die ursprüngliche Masche gelöscht und durch zwei neue Maschen ersetzt.
- ST_AddIsoEdge - Fügt eine isolierte Kante, die durch die Geometrie alinestring festgelegt wird zu einer Topologie hinzu, indem zwei bestehende isolierte Knoten anode und anothernode verbunden werden. Gibt die "edgeid" der neuen Kante aus.
- ST_AddIsoNode - Fügt einen isolierten Knoten zu einer Masche in einer Topologie hinzu und gibt die "nodeid" des neuen Knotens aus. Falls die Masche NULL ist, wird der Knoten dennoch erstellt.
- ST_Area - Gibt den geometrischen Schwerpunkt einer Geometrie zurück.
- ST_AsBinary - Rückgabe der OGC/ISO Well-Known Binary (WKB)-Darstellung der Geometrie/Geografie ohne SRID-Metadaten.
- ST_AsGML - Gibt die Geometrie als GML-Element - Version 2 oder 3 - zurück.
- ST_AsText - Gibt die Well-known-Text(WKT) Darstellung der Geometrie/Geographie ohne die SRID Metadaten zurück.
- ST_Boundary - Gibt die abgeschlossene Hülle aus der kombinierten Begrenzung der Geometrie zurück.
- ST_Buffer - Berechnet eine Geometrie, die alle Punkte innerhalb eines bestimmten Abstands zu einer Geometrie umfasst.
- ST_Centroid - Gibt den geometrischen Schwerpunkt einer Geometrie zurück.
- ST_ChangeEdgeGeom - Ändert die geometrische Form einer Kante, ohne sich auf die topologische Struktur auszuwirken.
- ST_Contains - Tests, wenn jeder Punkt von B in A liegt und ihre Innenräume einen gemeinsamen Punkt haben
- ST_ConvexHull - Berechnet die konvexe Hülle einer Geometrie.
- ST_CoordDim - Gibt die Dimension der Koordinaten für den Wert von ST_Geometry zurück.
- ST_CreateTopoGeo - Fügt eine Sammlung von Geometrien an eine leere Topologie an und gibt eine Bestätigungsmeldung aus.
- ST_Crosses - Prüft, ob zwei Geometrien einige, aber nicht alle, innere Punkte gemeinsam haben
- ST_CurveN - Gibt die N-te Geometrie einer Kurvenkomponente in einer CompoundCurve zurück.
- ST_CurveToLine - Konvertiert eine Geometrie mit Kurven in eine lineare Geometrie.
- ST_Difference - Berechnet eine Geometrie, die den Teil der Geometrie A darstellt, der die Geometrie B nicht schneidet.
- ST_Dimension - Gibt die Dimension der Koordinaten für den Wert von ST_Geometry zurück.
- ST_Disjoint - Prüft, ob zwei Geometrien keine gemeinsamen Punkte haben
- ST_Distance - Gibt die größte 3-dimensionale Distanz zwischen zwei geometrischen Objekten als Linie zurück

- ST_EndPoint - Returns the last point of a LineString, CircularLineString, or NURBSCurve.
- ST_Envelope - Gibt eine Geometrie in doppelter Genauigkeit (float8) zurück, welche das Umgebungsrechteck der beigestellten Geometrie darstellt.
- ST_Equals - Prüft, ob zwei Geometrien dieselbe Menge von Punkten enthalten
- ST_ExteriorRing - Gibt die Anzahl der inneren Ringe einer Polygongeometrie aus.
- ST_GMLToSQL - Returns a geometry from GML. Alias for ST_GeomFromGML.
- ST_GeomCollFromText - Erzeugt eine Sammelgeometrie mit der gegebenen SRID aus einer WKT-Kollektion. Wenn keine SRID angegeben ist, wird diese standardmäßig auf 0 gesetzt.
- ST_GeomFromText - Gibt einen spezifizierten ST_Geometry Wert aus einer Well-known-Text Darstellung (WKT) zurück.
- ST_GeomFromWKB - Erzeugt ein geometrisches Objekt aus der Well-known-Binary (WKB) Darstellung und einer optionalen SRID.
- ST_GeometryFromText - Returns a geometry from Well-Known Text (WKT). Alias for ST_GeomFromText.
- ST_GeometryN - Gibt den Geometrietyp des ST_Geometry Wertes zurück.
- ST_GeometryType - Gibt den Geometrietyp des ST_Geometry Wertes zurück.
- ST_GetFaceEdges - Gibt die Kanten, die aface begrenzen, sortiert aus.
- ST_GetFaceGeometry - Gibt für eine Topologie und eine bestimmte Maschen-ID das Polygon zurück.
- ST_InitTopoGeo - Erstellt ein neues Topologie-Schema und trägt es in die Tabelle topology.topology ein.
- ST_InteriorRingN - Gibt die Anzahl der inneren Ringe einer Polygongeometrie aus.
- ST_Intersection - Berechnet eine Geometrie, die den gemeinsamen Teil der Geometrien A und B darstellt.
- ST_Intersects - Prüft, ob sich zwei Geometrien schneiden (sie haben mindestens einen Punkt gemeinsam)
- ST_IsClosed - Gibt den Wert TRUE zurück, wenn die Anfangs- und Endpunkte des LINESTRING's zusammenfallen. Bei polyedrischen Oberflächen, wenn sie geschlossen (volumetrisch) sind.
- ST_IsEmpty - Prüft, ob eine Geometrie leer ist.
- ST_IsRing - Prüft, ob ein LineString geschlossen und einfach ist.
- ST_IsSimple - Gibt den Wert (TRUE) zurück, wenn die Geometrie keine irregulären Stellen, wie Selbstüberschneidungen oder Selbstberührungen, aufweist.
- ST_IsValid - Prüft, ob eine Geometrie in 2D wohlgeformt ist.
- ST_Length - Gibt den geometrischen Schwerpunkt einer Geometrie zurück.
- ST_LineFromText - Erzeugt eine Geometrie aus einer WKT Darstellung mit der angegebenen SRID. Wenn keine SRID angegeben wird, wird diese standardmäßig auf 0 gesetzt.
- ST_LineFromWKB - Creates a LINESTRING from a WKB representation, optionally using a given SRID.
- ST_LinestringFromWKB - Creates a LINESTRING from a WKB representation, optionally using a given SRID.
- ST_LocateAlong - Gibt die Punkte auf einer Geometrie zurück, die einem Messwert entsprechen.
- ST_LocateBetween - Gibt die Teile einer Geometrie zurück, die einem Messbereich entsprechen.
- ST_M - Gibt die M-Koordinate eines Punktes zurück.
- ST_MLineFromText - Liest einen festgelegten ST_MultiLineString Wert von einer WKT-Darstellung aus.
- ST_MPointFromText - Erzeugt eine Geometrie aus WKT mit der angegebenen SRID. Wenn keine SRID angegeben wird, wird diese standardmäßig auf 0 gesetzt.

- ST_MPolyFromText - Erzeugt eine MultiPolygon Geometrie aus WKT mit der angegebenen SRID. Wenn SRID nicht angegeben ist, wird sie standardmäßig auf 0 gesetzt.
- ST_ModEdgeHeal - "Heilt" zwei Kanten, indem der verbindende Knoten gelöscht wird, die erste Kante modifiziert und die zweite Kante gelöscht wird. Gibt die ID des gelöschten Knoten zurück.
- ST_ModEdgeSplit - Trennt eine Kante auf, indem ein neuer Knoten entlang einer bestehenden Kante erstellt wird. Ändert die ursprüngliche Kante und fügt eine neue Kante hinzu.
- ST_MoveIsoNode - Verschiebt einen isolierten Knoten in einer Topologie von einer Stelle an eine andere. Falls die neue Geometrie apoint bereits als Knoten existiert, wird eine Fehlermeldung ausgegeben. Gibt eine Beschreibung der Verschiebung aus.
- ST_NewEdgeHeal - "Heilt" zwei Kanten, indem der verbindende Knoten und beide Kanten gelöscht werden. Die beiden Kanten werden durch eine Kante ersetzt, welche dieselbe Ausichtung wie die erste Kante hat.
- ST_NewEdgesSplit - Trennt eine Kante auf, indem ein neuer Knoten entlang einer bestehenden Kante erstellt, die ursprüngliche Kante gelöscht und durch zwei neue Kanten ersetzt wird. Gibt die ID des neu erstellten Knotens aus, der die neuen Kanten verbindet.
- ST_NumCurves - Gibt die Anzahl der Kurvenkomponenten einer CompoundCurve zurück.
- ST_NumGeometries - Gibt die Anzahl der Punkte einer Geometrie zurück. Funktioniert für alle Geometrien.
- ST_NumInteriorRings - Gibt die Anzahl der inneren Ringe einer Polygongeometrie aus.
- ST_NumPatches - Returns the number of faces in a PolyhedralSurface or TIN.
- ST_NumPoints - Gibt die Anzahl der Stützpunkte eines ST_LineString oder eines ST_CircularString zurück.
- ST_OrderingEquals - Prüft, ob zwei Geometrien die gleiche Geometrie darstellen und Punkte in der gleichen Richtungsreihenfolge haben
- ST_Overlaps - Prüft, ob zwei Geometrien die gleiche Abmessung haben und sich schneiden, aber jede mindestens einen Punkt hat, der nicht in der anderen liegt
- ST_PatchN - Returns the Nth face of a PolyhedralSurface or TIN.
- ST_Perimeter - Gibt die Länge der Begrenzung einer polygonalen Geometrie oder Geografie zurück.
- ST_Point - Erzeugt einen Punkt mit X-, Y- und SRID-Werten.
- ST_PointFromText - Erzeugt eine Punktgeometrie mit gegebener SRID von WKT. Wenn SRID nicht angegeben ist, wird sie standardmäßig auf 0 gesetzt.
- ST_PointFromWKB - Erzeugt eine Geometrie mit gegebener SRID von WKB.
- ST_PointN - Gibt die Anzahl der Stützpunkte eines ST_LineString oder eines ST_CircularString zurück.
- ST_PointOnSurface - Berechnet einen Punkt, der garantiert in einem Polygon oder auf einer Geometrie liegt.
- ST_Polygon - Erzeugt ein Polygon aus einem LineString mit einem angegebenen SRID.
- ST_PolygonFromText - Erzeugt eine Geometrie aus WKT mit der angegebenen SRID. Wenn keine SRID angegeben wird, wird diese standardmäßig auf 0 gesetzt.
- ST_Relate - Prüft, ob zwei Geometrien eine topologische Beziehung haben, die einem Schnittpunktmatrixmuster entspricht, oder berechnet ihre Schnittpunktmatrix
- ST_RemEdgeModFace - Entfernt eine Kante, und wenn die Kante zwei Flächen trennt, wird eine Fläche gelöscht und die andere Fläche so verändert, dass sie den Raum beider Flächen abdeckt.
- ST_RemEdgeNewFace - Entfernt eine Kante. Falls die gelöschte Kante zwei Maschen voneinander getrennt hat, werden die ursprünglichen Maschen gelöscht und durch einer neuen Masche ersetzt.

- ST_RemoveIsoEdge - Löscht einen isolierten Knoten und gibt eine Beschreibung der getroffenen Maßnahmen aus. Falls der Knoten nicht isoliert ist, wird eine Fehlermeldung ausgegeben.
- ST_RemoveIsoNode - Löscht einen isolierten Knoten und gibt eine Beschreibung der getroffenen Maßnahmen aus. Falls der Knoten nicht isoliert ist (ist der Anfangs- oder der Endpunkt einer Kante), wird eine Fehlermeldung ausgegeben.
- ST_SRID - Gibt die Raumbezugskennung für eine Geometrie zurück.
- ST_StartPoint - Returns the first point of a LineString, CircularLineString, or NURBSCurve.
- ST_SymDifference - Berechnet eine Geometrie, die die Teile der Geometrien A und B darstellt, die sich nicht überschneiden.
- ST_Touches - Prüft, ob zwei Geometrien mindestens einen Punkt gemeinsam haben, aber ihre Innenräume sich nicht schneiden
- ST_Transform - Rückgabe einer neuen Geometrie mit in ein anderes räumliches Bezugssystem transformierten Koordinaten.
- ST_Union - Berechnet eine Geometrie, die die Punktmengenvereinigung der Eingabegeometrien darstellt.
- ST_WKBToSQL - Returns a geometry from Well-Known Binary (WKB). SQL/MM alias for ST_GeomFromWKB without an SRID argument.
- ST_WKTToSQL - Returns a geometry from Well-Known Text (WKT). SQL/MM alias for ST_GeomFromText.
- ST_Within - Tests, wenn jeder Punkt von A in B liegt und ihre Innenräume einen gemeinsamen Punkt haben
- ST_X - Gibt die X-Koordinate eines Punktes zurück.
- ST_Y - Gibt die Y-Koordinate eines Punktes zurück.
- ST_Z - Gibt die Z-Koordinate eines Punktes zurück.

13.4 PostGIS-Funktionen zur Unterstützung der Geografie

Die unten aufgeführten Funktionen und Operatoren sind PostGIS-Funktionen/Operatoren, die als Eingabe ein Objekt des Datentyps geography annehmen oder als Ausgabe zurückgeben.



Anmerkung

Funktionen mit einem (T) sind keine nativen geodätischen Funktionen und verwenden einen ST_Transform-Aufruf zu und von der Geometrie, um die Operation durchzuführen. Daher verhalten sie sich beim Überschreiten der Datumsgrenze, der Pole und bei großen Geometrien oder Geometriepaaren, die mehr als eine UTM-Zone abdecken, möglicherweise nicht wie erwartet. Basistransformation - (bevorzugt UTM, Lambert Azimutal (Nord/Süd), und im schlimmsten Fall Rückgriff auf Mercator)

- ST_Area - Gibt den geometrischen Schwerpunkt einer Geometrie zurück.
- ST_AsBinary - Rückgabe der OGC/ISO Well-Known Binary (WKB)-Darstellung der Geometrie/Geografie ohne SRID-Metadaten.
- ST_AsEWKT - Gibt die Well-known-Text(WKT) Darstellung der Geometrie mit den SRID-Metadaten zurück.
- ST_AsGML - Gibt die Geometrie als GML-Element - Version 2 oder 3 - zurück.
- ST_AsGeoJSON - Rückgabe einer Geometrie oder eines Merkmals im GeoJSON-Format.
- ST_AsKML - Gibt die Geometrie als GML-Element - Version 2 oder 3 - zurück.
- ST_AsSVG - Gibt eine Geometrie als SVG-Pfad aus.
- ST_AsText - Gibt die Well-known-Text(WKT) Darstellung der Geometrie/Geographie ohne die SRID Metadaten zurück.
- ST_Azimuth - Gibt die 2-dimenionale kürzeste Strecke zwischen zwei Geometrien als Linie zurück

- ST_Buffer - Berechnet eine Geometrie, die alle Punkte innerhalb eines bestimmten Abstands zu einer Geometrie umfasst.
- ST_Centroid - Gibt den geometrischen Schwerpunkt einer Geometrie zurück.
- ST_ClosestPoint - Gibt den 2D-Punkt auf g1 zurück, der g2 am nächsten ist. Dies ist der erste Punkt der kürzesten Linie von einer Geometrie zur anderen.
- ST_CoveredBy - Prüft, ob jeder Punkt von A in B liegt
- ST_Covers - Prüft, ob jeder Punkt von B in A liegt
- ST_DWithin - Prüft, ob zwei Geometrien innerhalb eines bestimmten Abstands liegen
- ST_Distance - Gibt die größte 3-dimensionale Distanz zwischen zwei geometrischen Objekten als Linie zurück
- ST_GeogFromText - Gibt einen geographischen Datentyp aus einer Well-known-Text (WKT), oder einer erweiterten WKT (EWKT), Darstellung zurück.
- ST_GeogFromWKB - Erzeugt ein geographisches Objekt aus der Well-known-Binary (WKB) oder der erweiterten Well-known-Binary (EWKB) Darstellung.
- ST_GeographyFromText - Gibt einen geographischen Datentyp aus einer Well-known-Text (WKT), oder einer erweiterten WKT (EWKT), Darstellung zurück.
- = - Gibt TRUE zurück, wenn die Koordinaten und die Reihenfolge der Koordinaten der Geometrie/Geographie A und der Geometrie/Geographie B ident sind.
- ST_Intersection - Berechnet eine Geometrie, die den gemeinsamen Teil der Geometrien A und B darstellt.
- ST_Intersects - Prüft, ob sich zwei Geometrien schneiden (sie haben mindestens einen Punkt gemeinsam)
- ST_Length - Gibt den geometrischen Schwerpunkt einer Geometrie zurück.
- ST_LineInterpolatePoint - Gibt einen oder mehrere, entlang einer Linie interpolierte Punkte zurück.
- ST_LineInterpolatePoints - Gibt einen oder mehrere, entlang einer Linie interpolierte Punkte zurück.
- ST_LineLocatePoint - Liefert die gebrochene Position des Punktes auf einer Linie, der einem Punkt am nächsten liegt.
- ST_LineSubstring - Gibt den Teil einer Linie zwischen zwei gebrochenen Stellen zurück.
- ST_Perimeter - Gibt die Länge der Begrenzung einer polygonalen Geometrie oder Geografie zurück.
- ST_Project - Gibt einen Punkt zurück, der von einem Startpunkt um eine bestimmte Entfernung und Peilung (Azimut) projiziert wird.
- ST_Segmentize - Gibt eine geänderte Geometrie/Geografie zurück, bei der kein Segment länger als eine bestimmte Entfernung ist.
- ST_ShortestLine - Gibt die 2-dimenionale kürzeste Strecke zwischen zwei Geometrien als Linie zurück
- ST_Summary - Gibt eine Zusammenfassung des Inhalts einer Geometrie wieder.
- <-> - Gibt die 2D Entfernung zwischen A und B zurück.
- && - Gibt TRUE zurück, wenn die 2D Bounding Box von A die 2D Bounding Box von B schneidet.

13.5 PostGIS-Funktionen zur Unterstützung von Rastern

Die unten aufgeführten Funktionen und Operatoren sind PostGIS-Funktionen/Operatoren, die als Eingabe ein Objekt des Datentyps raster annehmen oder als Ausgabe zurückgeben. Sie sind in alphabetischer Reihenfolge aufgelistet.

- Box3D - Stellt das umschreibende Rechteck eines Raster als Box3D dar.
- @ - Gibt TRUE zurück, wenn das umschreibende Rechteck von A in jenem von B enthalten ist. Das umschreibende Rechteck ist in Double Precision.
- ~ - Gibt TRUE zurück, wenn das umschreibende Rechteck von A jenes von B enthält. Das umschreibende Rechteck ist in Double Precision.
- = - Gibt TRUE zurück, wenn die umschreibenden Rechtecke von A und B ident sind. Das umschreibende Rechteck ist in Double Precision.
- && - Gibt TRUE zurück, wenn das umschreibende Rechteck von A das umschreibende Rechteck von B schneidet.
- &< - Gibt TRUE zurück, wenn das umschreibende Rechteck von A links von dem von B liegt.
- &> - Gibt TRUE zurück, wenn das umschreibende Rechteck von A rechts von dem von B liegt.
- ~= - Gibt TRUE zurück, wenn die bounding box von A ident mit jener von B ist.
- ST_Retile - Gibt konfigurierte Kacheln eines beliebig gekachelten Rastercoverage aus.
- ST_AddBand - Gibt einen Raster mit den neu hinzugefügten Band(Bändern) aus. Der Typ , der Ausgangswert und der Index für den Speicherort des Bandes kann angegeben werden. Wenn kein Index angegeben ist, wird das Band am Ende hinzugefügt.
- ST_AsBinary/ST_AsWKB - Gibt die Well-known-Binary (WKB) Darstellung eines Rasters zurück.
- ST_AsGDALRaster - Gibt die Rasterkachel in dem ausgewiesenen Rasterformat von GDAL aus. Sie können jedes Rasterformat angeben, das von Ihrer Bibliothek unterstützt wird. Um eine Liste mit den unterstützten Formaten auszugeben, verwenden Sie bitte ST_GDALDrivers().
- ST_AsHexWKB - Gibt die Well-known-Binary (WKB) Hex-Darstellung eines Rasters zurück.
- ST_AsJPEG - Gibt die ausgewählten Bänder der Rasterkachel als einzelnes Bild (Byte-Array) im Format "Joint Photographic Exports Group" (JPEG) aus. Wenn kein Band angegeben ist und 1 oder mehr als 3 Bänder ausgewählt wurden, dann wird nur das erste Band verwendet. Wenn 3 Bänder ausgewählt wurden, werden alle 3 Bänder verwendet und auf RGB abgebildet.
- ST_AsPNG - Gibt die ausgewählten Bänder der Rasterkachel als einzelnes, übertragbares Netzwerkgraphik (PNG) Bild (Byte-Feld) aus. Wenn der Raster 1,3 oder 4 Bänder hat und keine Bänder angegeben sind, dann werden alle Bänder verwendet. Wenn der Raster 2 oder mehr als 4 Bänder hat und keine Bänder angegeben sind, dann wird nur Band 1 verwendet. Die Bänder werden in den RGB- oder den RGBA-Raum abgebildet.
- ST_AsRaster - Konvertiert den geometrischen Datentyp von PostGIS in einen PostGIS Raster.
- ST_AsRasterAgg - Aggregiert. Wandelt PostGIS-Geometrien in einen neuen Raster um.
- ST_AsTIFF - Gibt die ausgewählten Bänder des Raster als einzelnes TIFF Bild (Byte-Feld) zurück. Wenn kein Band angegeben ist oder keines der angegebenen Bänder im Raster existiert, werden alle Bänder verwendet.
- ST_Aspect - Gibt die Exposition (standardmäßig in Grad) eines Rasterbandes mit Höhen aus. Nützlich für Terrain-Analysen.
- ST_Band - Gibt einen oder mehrere Bänder eines bestehenden Rasters als neuen Raster aus. Nützlich um neue Raster aus bestehenden Rastern abzuleiten.
- ST_BandFileSize - Gibt die Dateigröße eines im Dateisystem gespeicherten Bandes aus. Wenn "bandnum" nicht angegeben ist, wird 1 angenommen.
- ST_BandFileTimestamp - Gibt den Zeitstempel eines im Dateisystem gespeicherten Bandes aus. Wenn "bandnum" nicht angegeben ist, wird 1 angenommen.

- ST_BandIsNoData - Gibt TRUE aus, wenn das Band ausschließlich aus NODATA Werten besteht.
- ST_BandMetaData - Gibt die grundlegenden Metadaten eines bestimmten Rasterbandes aus. Wenn der Parameter "bandnum" nicht angegeben ist, wird das 1ste Band angenommen.
- ST_BandNoDataValue - Gibt den NODATA Wert des gegebenen Bandes aus. Wenn der Parameter "bandnum" nicht angegeben ist, wird das 1ste Band angenommen.
- ST_BandPath - Gibt den Dateipfad aus, unter dem das Band im Dateisystem gespeichert ist. Wenn "bandnum" nicht angegeben ist, wird 1 angenommen.
- ST_BandPixelType - Gibt den Pixeltyp des angegebenen Bandes aus. Wenn der Parameter "bandnum" nicht angegeben ist, wird das 1ste Band angenommen.
- ST_Clip - Gibt den nach der Eingabegeometrie zugeschnittenen Raster zurück. Falls die Bandnummer nicht angegeben ist, werden alle Bänder verarbeitet. Wenn crop nicht angegeben oder auf TRUE gesetzt ist, wird der Ausgaberaster entsprechend zugeschnitten. Wenn touched auf TRUE gesetzt ist, werden auch berührte Pixel einbezogen, andernfalls nur Pixel, bei denen sich der Mittelpunkt im Inneren der Geometrie befindet.
- ST_ColorMap - Erzeugt aus einem bestimmten Band des Ausgangsrasters einen neuen Raster mit bis zu vier 8BUI-Bändern (Grauwert, RGB, RGBA). Wenn kein Band angegeben ist, wird Band 1 angenommen.
- ST_Contains - Gibt TRUE zurück, wenn kein Punkt des Rasters "rastB" im Äußeren des Rasters "rastA" liegt und zumindest ein Punkt im Inneren von "rastB" auch im Inneren von "rastA" liegt.
- ST_ContainsProperly - Gibt TRUE zurück, wenn "rastB" das Innere von "rastA" schneidet, aber nicht die Begrenzung oder das Äußere von "rastA".
- ST_Contour - Erzeugt einen Satz von Vektorkonturen aus dem angegebenen Rasterband unter Verwendung des GDAL-Konturierungsalgorithmus.
- ST_ConvexHull - Gibt die Geometrie der konvexen Hülle des Raster, inklusive der Pixel deren Werte gleich BandNoDataValue sind. Bei regelmäßig geformten und nicht rotierten Raster ist das Ergebnis ident mit ST_Envelope. Diese Funktion ist deshalb nur bei unregelmäßig geformten oder rotierten Raster nützlich.
- ST_Count - Gibt die Anzahl der Pixel für ein Band eines Rasters oder eines Raster-Coverage zurück. Wenn kein Band angegeben ist, wird standardmäßig Band 1 gewählt. Wenn der Parameter "exclude_nodata_value" auf TRUE gesetzt ist, werden nur Pixel mit Werten ungleich NODATA gezählt.
- ST_CountAgg - Aggregatfunktion. Gibt die Anzahl der Pixel in einem bestimmten Band der Raster aus. Wenn kein Band angegeben ist, wird Band 1 angenommen. Wenn "exclude_nodata_value" TRUE ist, werden nur die Pixel ohne NODATA Werte gezählt.
- ST_CoveredBy - Gibt TRUE zurück, wenn kein Punkt des Rasters "rastA" außerhalb des Rasters "rastB" liegt.
- ST_Covers - Gibt TRUE zurück, wenn kein Punkt des Rasters "rastB" außerhalb des Rasters "rastA" liegt.
- ST_DFullyWithin - Gibt TRUE zurück, wenn die Raster "rastA" und "rastB" zur Gänze innerhalb der angegebenen Distanz zueinander liegen.
- ST_DWithin - Gibt TRUE zurück, wenn die Raster "rastA" und "rastB" innerhalb der angegebenen Entfernung voneinander liegen.
- ST_Disjoint - Gibt TRUE zurück, wenn sich die Raster "rastA" und "rastB" räumlich nicht überschneiden.
- ST_DumpAsPolygons - Gibt geomval (geom,val) Zeilen eines Rasterbandes zurück. Wenn kein Band angegeben ist, wird die Bandnummer standardmäßig auf 1 gesetzt.
- ST_DumpValues - Gibt die Werte eines bestimmten Bandes als 2-dimensionales Feld aus.
- ST_Envelope - Stellt die Ausdehnung des Raster als Polygon dar.
- ST_FromGDALRaster - Erzeugt einen Raster aus einer von GDAL unterstützten Rasterdatei.

- ST_GeoReference - Gibt die Metadaten der Georeferenzierung, die sich üblicherweise in einem sogenannten "World File befinden, im GDAL oder ESRI Format aus. Die Standardeinstellung ist GDAL.
- ST_Grayscale - Erzeugt einen neuen Raster mit einem 8BUI-Band aus dem Ausgangsrasters und den angegebenen Bändern für Rot, Grün und Blau
- ST_HasNoBand - Gibt TRUE aus, wenn kein Band mit der angegebenen Bandnummer existiert.Gibt den Pixeltyp des angegebenen Bandes aus. Wenn keine Bandnummer angegeben ist, wird das 1ste Band angenommen.
- ST_Height - Gibt die Höhe des Rasters in Pixel aus.
- ST_HillShade - Gibt für gegebenen Horizontalwinkel, Höhenwinkel, Helligkeit und Maßstabsverhältnis die hypothetische Beleuchtung eines Höhenrasterbandes zurück.
- ST_Histogram - Gibt Datensätze aus, welche die Verteilung der Daten eines Rasters oder eines Rastercoverage darstellen. Dabei wird die Wertemenge in Klassen aufgeteilt und für jede Klasse zusammengefasst. Wenn die Anzahl der Klassen nicht angegeben ist, wird sie automatisch berechnet.
- ST_InterpolateRaster - Interpoliert eine gerasterte Oberfläche auf der Grundlage eines Eingabesatzes von 3D-Punkten, wobei die X- und Y-Werte zur Positionierung der Punkte auf dem Gitter und der Z-Wert der Punkte als Oberflächenhöhe verwendet werden.
- ST_Intersection - Gibt Geometry-PixelValue Paare, oder einen Raster aus, der durch die Schnittmenge der beiden Raster bestimmt wird, oder durch die geometrische Verschneidung einer Vektorisierung des Rasters mit einem geometrischen Datentyp.
- ST_IntersectionFractions - Calculates the fraction of each raster cell that is covered by a given geometry.
- ST_Intersects - Gibt TRUE zurück, wenn sich die Raster "rastA" und "rastB" nicht räumlich überschneiden.
- ST_IsEmpty - Gibt TRUE zurück, wenn der Raster leer ist (width = 0 and height = 0). Andernfalls wird FALSE zurückgegeben.
- ST_MakeEmptyCoverage - Bedeckt die georeferenzierte Fläche mit einem Gitter aus leeren Rasterkacheln.
- ST_MakeEmptyRaster - Gibt einen leeren Raster (ohne Bänder), mit den gegebenen Dimensionen (width & height), upperleft X und Y, Pixelgröße, Rotation (scalex, scaley, skewx & skewy) und Koordinatenreferenzsystem (SRID), zurück. Wenn ein Raster übergeben wird, dann wird ein neuer Raster mit der selben Größe, Ausrichtung und SRID zurückgegeben. Wenn SRID nicht angegeben ist, wird das Koordinatenreferenzsystem auf "unknown" (0) gesetzt.
- ST_MapAlgebra (callback function version) - Die Version mit der Rückruffunktion - Gibt für einen oder mehrere Eingaberaster einen Raster mit einem Band, den Bandindizes und einer vom Anwender vorgegebenen Rückruffunktion zurück.
- ST_MapAlgebraExpr - Version mit 1 Rasterband: Erzeugt ein neues Rasterband, dass über eine gültige, algebraische PostgreSQL Operation für ein Rasterband mit gegebenen Pixeltyp erstellt wird. Wenn kein Band bestimmt ist, wird Band 1 angenommen.
- ST_MapAlgebraExpr - Version mit 2 Rasterbändern: Erstellt einen neuen Einzelbandraster, indem eine gültige algebraische PostgreSQL Funktion auf die zwei Ausgangsrasterbänder und den entsprechenden Pixeltyp angewendet wird. Wenn keine Bandnummern angegeben sind, wird von jedem Raste Band 1 angenommen. Der Ergebnisraster wird nach dem Gitter des ersten Raster ausgerichtet (Skalierung, Versatz und Eckpunkte der Pixel) und hat die Ausdehnung, welche durch den Parameter "extenttype" definiert ist. Der Parameter "extenttype" kann die Werte INTERSECTION, UNION, FIRST, SECOND annehmen.
- ST_MapAlgebraFct - Version mit 1 Rasterband: Erzeugt ein neues Rasterband, dass über eine gültige PostgreSQL Funktion für ein gegebenes Rasterband und Pixeltyp erstellt wird. Wenn kein Band bestimmt ist, wird Band 1 angenommen.
- ST_MapAlgebraFct - Version mit 2 Rasterbändern: Erstellt einen neuen Einzelbandraster, indem eine gültige PostgreSQL Funktion auf die 2 gegebenen Rasterbänder und den entsprechenden Pixeltyp angewendet wird. Wenn kein Band bestimmt ist, wird Band 1 angenommen. Wenn der "Extent"-Typ nicht angegeben ist, wird standardmäßig INTERSECTION angenommen.
- ST_MapAlgebraFctNgb - Version mit 1em Band: Map Algebra Nearest Neighbor mit einer benutzerdefinierten PostgreSQL Funktion. Gibt einen Raster zurück, dessen Werte sich aus einer benutzerdefinierte PL/pgsql Funktion ergeben, welche die Nachbarschaftswerte des Ausgangsrasterbandes einbezieht.
- ST_MapAlgebra (expression version) - Version mit Ausdrücken - Gibt für einen oder zwei Ausgangsraster, Bandindizes und einer oder mehreren vom Anwender vorgegebenen SQL-Ausdrücken, einen Raster mit einem Band zurück.

- ST_MemSize - Gibt den Platzbedarf des Rasters (in Byte) aus.
- ST_MetaData - Gibt die wesentlichen Metadaten eines Rasterobjektes, wie Zellgröße, Rotation (Versatz) etc. aus
- ST_MinConvexHull - Gibt die Geometrie der konvexen Hülle des Raster aus, wobei Pixel mit NODATA ausgenommen werden.
- ST_NearestValue - Gibt den nächstgelegenen nicht NODATA Wert eines bestimmten Pixels aus, das über "columnx" und "rowy" oder durch eine Punktgeometrie - im gleichen Koordinatenreferenzsystem wie der Raster - ausgewählt wird.
- ST_Neighborhood - Gibt ein 2-D Feld in "Double Precision" aus, das sich aus nicht NODATA Werten um ein bestimmtes Pixel herum zusammensetzt. Das Pixel Kann über "columnx" und "rowy" oder über eine Punktgeometrie - im gleichen Koordinatenreferenzsystem wie der Raster - ausgewählt werden.
- ST_NotSameAlignmentReason - Gibt eine Meldung aus, die angibt ob die Raster untereinander ausgerichtet sind oder nicht und warum wenn nicht.
- ST_NumBands - Gibt die Anzahl der Bänder des Rasters aus.
- ST_Overlaps - Gibt TRUE zurück, wenn sich die Raster "rastA" und "rastB" schneiden, aber ein Raster den anderen nicht zur Gänze enthält.
- ST_PixelAsCentroid - Gibt den geometrischen Schwerpunkt (Punktgeometrie) der Fläche aus, die durch das Pixel repräsentiert wird.
- ST_PixelAsCentroids - Gibt den geometrischen Schwerpunkt (Punktgeometrie) für jedes Pixel des Rasterbandes zurück, zusammen mit dem Zellwert und den X- und Y-Rasterkoordinaten eines jeden Pixels. Die Koordinaten der Punkte entsprechen dem geometrischen Schwerpunkt der Pixel.
- ST_PixelAsPoint - Gibt eine Punktgeometrie der oberen linken Ecke des Rasters zurück.
- ST_PixelAsPoints - Gibt eine Punktgeometrie für jedes Pixel des Rasterbandes zurück, zusammen mit dem Zellwert und den X- und Y-Rasterkoordinaten eines jeden Pixels. Die Koordinaten der Punkte entsprechen dem oberen linken Eck der Pixel.
- ST_PixelAsPolygon - Gibt die Polygongeometrie aus, die das Pixel einer bestimmten Zeile und Spalte begrenzt.
- ST_PixelAsPolygons - Gibt die umhüllende Polygongeometrie, den Zellwert, sowie die X- und Y-Rasterkoordinate für jedes Pixel aus.
- ST_PixelHeight - Gibt die Pixelhöhe in den Einheiten des Koordinatenreferenzsystem aus.
- ST_PixelOfValue - Gibt die columnx- und rowy-Koordinaten jener Pixel aus, deren Zellwert gleich dem gesuchten Wert ist.
- ST_PixelWidth - Gibt die Pixelbreite in den Einheiten des Koordinatenreferenzsystems aus.
- ST_Polygon - Gibt eine Geometrie mit Mehrfachpolygonen zurück, die aus der Vereinigung von Pixel mit demselben Zellwert gebildet werden. Pixel mit NODATA Werten werden nicht berücksichtigt. Wenn keine Band angegeben ist, wird die Bandnummer standardmäßig auf 1 gesetzt.
- ST_Quantile - Berechnet die Quantile eines Rasters oder einer Rastercoverage Tabelle im Kontext von Stichproben oder Bevölkerung. Dadurch kann untersucht werden, ob ein Wert bei 25%, 50% oder 75% Perzentil des Rasters liegt.
- ST_RastFromHexWKB - Gibt einen Rasterwert von einer Well-known-Binary (WKB) Hex-Darstellung eines Rasters zurück.
- ST_RastFromWKB - Gibt einen Rasterwert von einer Well-known-Binary (WKB) Darstellung eines Rasters zurück.
- ST_RasterToWorldCoord - Gibt die obere linke Ecke des Rasters in geodätischem X und Y (Länge und Breite) für eine gegebene Spalte und Zeile aus. Spalte und Zeile wird von 1 aufwärts gezählt.
- ST_RasterToWorldCoordX - Gibt die geodätische X Koordinate links oberhalb des Rasters, der Spalte und der Zeile aus. Die Nummerierung der Spalten und Zeilen beginnt mit 1.
- ST_RasterToWorldCoordY - Gibt die geodätische Y Koordinate links oberhalb des Rasters, der Spalte und der Zeile aus. Die Nummerierung der Spalten und Zeilen beginnt mit 1.

- ST_Reclass - Erstellt einen neuen Raster, der aus neu klassifizierten Bändern des Originalraster besteht. Das Band "nband" ist jenes das verändert werden soll. Wenn "nband" nicht angegeben ist, wird "Band 1" angenommen. Alle anderen Bänder bleiben unverändert. Anwendungsfall: zwecks einfacherer Visualisierung ein 16BUI-Band in ein 8BUI-Band konvertieren und so weiter.
- ST_ReclassExact - Creates a new raster composed of bands reclassified from original, using a 1:1 mapping from values in the original band to new values in the destination band.
- ST_Resample - Skaliert einen Raster mit einem bestimmten Algorithmus, neuen Dimensionen, einer beliebigen Gitterecke und über Parameter zur Georeferenzierung des Rasters, die angegeben oder von einem anderen Raster übernommen werden können.
- ST_Rescale - Neuabtastung eines Rasters, indem nur die Skala (oder Pixelgröße) angepasst wird. Die neuen Pixelwerte werden mit den Algorithmen NearestNeighbor (englische oder amerikanische Schreibweise), Bilinear, Cubic, CubicSpline, Lanczos, Max oder Min resampling berechnet. Die Voreinstellung ist NearestNeighbor.
- ST_Resize - Ändert die Zellgröße - width/height - eines Rasters
- ST_Reskew - Skaliert einen Raster, indem lediglich der Versatz (oder Rotationsparameter) angepasst wird. Neue Pixelwerte werden über NearestNeighbor, bilinear, kubisch, CubicSpline oder mit dem Lanczos-Filter errechnet. Die Standardeinstellung ist NearestNeighbor.
- ST_Rotation - Gibt die Rotation des Rasters im Bogenmaß aus.
- ST_Roughness - Gibt einen Raster mit der berechneten "Rauhigkeit" des DHM zurück.
- ST_SRID - Gibt den Identifikator des Koordinatenreferenzsystems des Rasters aus, das in der Tabelle "spatial_ref_sys" definiert ist.
- ST_SameAlignment - Gibt TRUE zurück, wenn die Raster die selbe Rotation, Skalierung, Koordinatenreferenzsystem und Versatz (Pixel können auf dasselbe Gitter gelegt werden, ohne dass die Gitterlinien durch die Pixel schneiden) aufweisen. Wenn nicht, wird FALSE und eine Bechreibung des Problems ausgegeben.
- ST_ScaleX - Gibt die X-Komponente der Pixelbreite in den Einheiten des Koordinatenreferenzsystems aus.
- ST_ScaleY - Gibt die Y-Komponente der Pixelhöhe in den Einheiten des Koordinatenreferenzsystems aus.
- ST_SetBandIndex - Aktualisiert die externe Bandnummer eines out-db Bandes.
- ST_SetBandIsNoData - Setzt die Flag "isnodata" für das Band auf TRUE.
- ST_SetBandNoDataValue - Setzt den NODATA Wert eines Bandes. Wenn kein Band angegeben ist, wird Band 1 angenommen. Falls ein Band keinen NODATA Wert aufweisen soll, übergeben Sie bitte für den Parameter "nodatavalue" NULL.
- ST_SetBandPath - Aktualisiert den externen Dateipfad und die Bandnummer eines out-db Bandes.
- ST_SetGeoReference - Georeferenziert einen Raster über 6 Parameter in einem einzigen Aufruf. Die Zahlen müssen durch Leerzeichen getrennt sein. Die Funktion akzeptiert die Eingabe im Format von 'GDAL' und von 'ESRI'. Der Standardwert ist GDAL.
- ST_SetM - Gibt eine Geometrie mit denselben X/Y-Koordinaten wie die Eingabegeometrie zurück, wobei die Werte aus dem Raster mit dem gewünschten Resample-Algorithmus in die Dimension M kopiert werden.
- ST_SetRotation - Bestimmt die Rotation des Rasters in Radiant.
- ST_SetSRID - Setzt die SRID eines Rasters auf einen bestimmten Ganzzahlwert. Die SRID wird in der Tabelle "spatial_ref_sys" definiert.
- ST_SetScale - Setzt die X- und Y-Größe der Pixel in den Einheiten des Koordinatenreferenzsystems. Entweder eine Zahl pro Pixel oder Breite und Höhe.
- ST_SetSkew - Setzt den georeferenzierten X- und Y-Versatz (oder den Rotationsparameter). Wenn nur ein Wert übergeben wird, werden X und Y auf den selben Wert gesetzt.
- ST_SetUpperLeft - Setzt den Wert der oberen linke Ecke des Rasters auf die projizierten X- und Y-Koordinaten.

- ST_SetValue - Setzt den Wert für ein Pixel eines Bandes, das über columnx und rowy festgelegt wird, oder für die Pixel die eine bestimmte Geometrie schneiden, und gibt den veränderten Raster zurück. Die Bandnummerierung beginnt mit 1; wenn die Bandnummer nicht angegeben ist, wird 1 angenommen.
- ST_SetValues - Gibt einen Raster zurück, der durch das Setzen der Werte eines bestimmten Bandes verändert wurde.
- ST_SetZ - Gibt eine Geometrie mit denselben X/Y-Koordinaten wie die Eingabegeometrie zurück, wobei die Werte aus dem Raster mit dem gewünschten Resample-Algorithmus in die Z-Dimension kopiert werden.
- ST_SkewX - Gibt den georeferenzierten Versatz in X-Richtung (oder den Rotationsparameter) aus.
- ST_SkewY - Gibt den georeferenzierten Versatz in Y-Richtung (oder den Rotationsparameter) aus.
- ST_Slope - Gibt die Neigung (standardmäßig in Grad) eines Höhenrasterbandes zurück. Nützlich für Terrain-Analysen.
- ST_SnapToGrid - Skaliert einen Raster durch Fangen an einem Führungsgitter. Neue Pixelwerte werden über NearestNeighbor, bilinear, kubisch, CubicSpline oder mit dem Lanczos-Filter errechnet. Die Standardeinstellung ist NearestNeighbor.
- ST_Summary - Gibt eine textliche Zusammenfassung des Rasterinhalts zurück.
- ST_SummaryStats - 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 if 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 if no band is specified.
- ST_TPI - Berechnet den "Topographic Position Index" eines Raster.
- ST_TRI - Gibt einen Raster mit errechneten Geländerauheitsindex aus.
- ST_Tile - Gibt Raster, die aus einer Teilungsoperation des Eingaberasters resultieren, mit den gewünschten Dimensionen aus.
- ST_Touches - Gibt TRUE zurück, wenn rastA und rastB zumindest einen Punkt gemeinsam haben sich aber nicht überschneiden.
- ST_Transform - Projiziert einen Raster von einem bekannten Koordinatenreferenzsystem in ein anderes bekanntes Koordinatenreferenzsystem um. Die Optionen für die Skalierung sind NearestNeighbor, Bilinear, Cubisch, CubicSpline und der Lanczos-Filter, die Standardeinstellung ist NearestNeighbor.
- ST_Union - Gibt die Vereinigung mehrerer Rasterkacheln in einem einzelnen Raster mit mehreren Bändern zurück.
- ST_UpperLeftX - Gibt die obere linke X-Koordinate des Rasters im Koordinatenprojektionssystem aus.
- ST_UpperLeftY - Gibt die obere linke Y-Koordinate des Rasters im Koordinatenprojektionssystem aus.
- ST_Value - Gibt den Zellwert eines Pixels aus, das über columnx und rowy oder durch einen bestimmten geometrischen Punkt angegeben wird. Die Bandnummern beginnen mit 1 und wenn keine Bandnummer angegeben ist, dann wird Band 1 angenommen. Wenn exclude_nodata_value auf FALSE gesetzt ist, werden auch die Pixel mit einem nodata Wert mit einbezogen. Wenn exclude_nodata_value nicht übergeben wird, dann wird er über die Metadaten des Rasters ausgelesen.
- ST_ValueCount - Gibt Datensätze aus, die den Zellwert und die Anzahl der Pixel eines Rasterbandes (oder Rastercoverageban-des) für gegebene Werte enthalten. Wenn kein Band angegeben ist, wird Band 1 angenommen. Pixel mit dem Wert NODATA werden standardmäßig nicht gezählt; alle anderen Pixelwerte des Bandes werden ausgegeben und auf die nächste Ganzzahl gerundet.
- ST_Width - Gibt die Breite des Rasters in Pixel aus.
- ST_Within - Gibt TRUE zurück, wenn kein Punkt des Rasters "rastA" außerhalb des Rasters "rastB" liegt und zumindest ein Punkt im Inneren von "rastA" auch im Inneren von "rastB" liegt.
- ST_WorldToRasterCoord - Gibt für ein geometrisches X und Y (geographische Länge und Breite) oder für eine Punktgeometrie im Koordinatenreferenzsystem des Rasters, die obere linke Ecke als Spalte und Zeile aus.
- ST_WorldToRasterCoordX - Gibt für eine Punktgeometrie (pt) oder eine globale X- und Y-Koordinate (xw, yw) die Rasterspalte im globalen Koordinatenreferenzsystem des Rasters aus.

- ST_WorldToRasterCoordY - Gibt für eine Punktgeometrie (pt) oder eine globale X- und Y-Koordinate (xw, yw) die Rasterzeile im globalen Koordinatenreferenzsystem des Rasters aus.
- UpdateRasterSRID - Änderung der SRID aller Raster in der vom Anwender angegebenen Spalte und Tabelle.

13.6 PostGIS Geometrie / Geographie / Raster Dump Funktionen

Die nachstehend aufgeführten Funktionen sind PostGIS-Funktionen, die als Eingabe einen Satz oder ein einzelnes geometry_dump oder geomval Objekt vom Datentyp annehmen oder als Ausgabe zurückgeben.

- ST_DumpAsPolygons - Gibt geomval (geom,val) Zeilen eines Rasterbandes zurück. Wenn kein Band angegeben ist, wird die Bandnummer standardmäßig auf 1 gesetzt.
- ST_Intersection - Gibt Geometry-PixelValue Paare, oder einen Raster aus, der durch die Schnittmenge der beiden Raster bestimmt wird, oder durch die geometrische Verschneidung einer Vektorisierung des Rasters mit einem geometrischen Datentyp.
- ST_Dump - Gibt einen Satz von geometry_dump Zeilen für die Komponenten einer Geometrie zurück.
- ST_DumpPoints - Gibt eine Zusammenfassung des Inhalts einer Geometrie wieder.
- ST_DumpRings - Gibt einen Satz von geometry_dump Zeilen für die äußeren und inneren Ringe eines Polygons zurück.
- ST_DumpSegments - Gibt eine Zusammenfassung des Inhalts einer Geometrie wieder.

13.7 PostGIS-Box-Funktionen

Bei den nachstehenden Funktionen handelt es sich um PostGIS-Funktionen, die als Eingabe die Box*-Familie von PostGIS-Geodatentypen verwenden oder als Ausgabe zurückgeben. Die Familie der Box-Typen besteht aus box2d und box3d

- Box2D - Gibt ein BOX2D zurück, das die 2D-Ausdehnung einer Geometrie darstellt.
- Box3D - Gibt ein BOX3D zurück, das die 3D-Ausdehnung einer Geometrie darstellt.
- MakeTopologyPrecise - Snap topology vertices to precision grid.
- Box3D - Stellt das umschreibende Rechteck eines Raster als Box3D dar.
- ST_3DExtent - Aggregatfunktion, die den 3D-Begrenzungsrahmen von Geometrien zurückgibt.
- ST_3DMakeBox - Erzeugt einen BOX3D, der durch zwei 3D-Punktgeometrien definiert ist.
- ST_AsMVTGeom - Transformiert eine Geometrie in den Koordinatenraum einer MVT-Kachel.
- ST_AsTWKB - Gibt die Geometrie als TWKB, aka "Tiny Well-known Binary" zurück
- ST_Box2dFromGeoHash - Gibt die BOX2D einer GeoHash Zeichenkette zurück.
- ST_ClipByBox2D - Berechnet den Teil einer Geometrie, der innerhalb eines Rechtecks liegt.
- ST_EstimatedExtent - Gibt die geschätzte Ausdehnung einer räumlichen Tabelle zurück.
- ST_Expand - Gibt einen Begrenzungsrahmen zurück, der aus einem anderen Begrenzungsrahmen oder einer Geometrie erweitert wurde.
- ST_Extent - Aggregatfunktion, die den Begrenzungsrahmen von Geometrien zurückgibt.
- ST_MakeBox2D - Erzeugt ein BOX2D, das durch zwei 2D-Punktgeometrien definiert ist.
- ST_RemoveIrrelevantPointsForView - Removes points that are irrelevant for rendering a specific rectangular view of a geometry.

- ST_XMax - Gibt die X-Maxima eines 2D- oder 3D-Begrenzungsrahmens oder einer Geometrie zurück.
- ST_XMin - Gibt die X-Minima eines 2D- oder 3D-Begrenzungsrahmens oder einer Geometrie zurück.
- ST_XSize - Returns the X size of a 2D or 3D bounding box or a geometry.
- ST_YMax - Gibt die Y-Maxima eines 2D- oder 3D-Begrenzungsrahmens oder einer Geometrie zurück.
- ST_YMin - Gibt die Y-Minima eines 2D- oder 3D-Begrenzungsrahmens oder einer Geometrie zurück.
- ST_YSize - Returns the Y size of a 2D or 3D bounding box or a geometry.
- ST_ZMax - Gibt die Z-Maxima eines 2D- oder 3D-Begrenzungsrahmens oder einer Geometrie zurück.
- ST_ZMin - Gibt die Z-Minima eines 2D- oder 3D-Begrenzungsrahmens oder einer Geometrie zurück.
- ST_ZSize - Returns the Z size of a 2D or 3D bounding box or a geometry.
- RemoveUnusedPrimitives - Entfernt Topologieprimitive, die zur Definition bestehender TopoGeometry-Objekte nicht benötigt werden.
- ValidateTopology - Liefert eine Menge validatetopology_returntype Objekte, die Probleme mit der Topologie beschreiben.
- ValidateTopologyPrecision - Returns non-precise vertices in the topology.
- ~(box2df,box2df) - Gibt TRUE zurück, wenn eine 2D float precision bounding box (BOX2DF) eine andere 2D float precision bounding box (BOX2DF) enthält.
- ~(box2df,geometry) - Gibt TRUE zurück, wenn eine 2D float precision bounding box (BOX2DF) die 2D Bounding Box einer Geometrie enthält.
- ~(geometry,box2df) - Gibt TRUE zurück, wenn die 2D bounding box einer Geometrie eine 2D float precision bounding box (GIDX) enthält.
- @(box2df,box2df) - Gibt TRUE zurück, wenn eine 2D float precision bounding box (BOX2DF) innerhalb einer anderen 2D float precision bounding box enthalten ist.
- @(box2df,geometry) - Gibt TRUE zurück, wenn eine 2D float precision bounding box (BOX2DF) in der 2D Bounding Box einer Geometrie enthalten ist..
- @(geometry,box2df) - Gibt TRUE zurück, wenn die 2D Bounding Box einer Geometrie in einer 2D float precision Bbounding Box (BOX2DF) enthalten ist.
- &&(box2df,box2df) - Gibt TRUE zurück, wenn sich zwei 2D float precision Bounding Boxes (BOX2DF) überschneiden.
- &&(box2df,geometry) - Gibt TRUE zurück, wenn eine 2D float precision bounding box (BOX2DF) eine Geometrie (cached) 2D bounding box schneidet.
- &&(geometry,box2df) - Gibt TRUE zurück, wenn sich die 2D Bounding Box (cached) einer Geometrie mit einer 2D Bounding Box mit Gleitpunktgenauigkeit (BOX2DF) überschneidet.

13.8 PostGIS-Funktionen, die 3D unterstützen

Die unten aufgeführten Funktionen sind PostGIS-Funktionen, die den Z-Index nicht wegwerfen.

- AddGeometryColumn - Entfernt eine Geometriespalte aus einer räumlichen Tabelle.
- Box3D - Gibt ein BOX3D zurück, das die 3D-Ausdehnung einer Geometrie darstellt.
- CG_3DAlphaWrapping - Computes a 3D Alpha-wrapping strictly enclosing a geometry.
- CG_3DArea - Berechnet die Fläche von 3D-Oberflächengeometrien. Gibt 0 für Solids zurück.
- CG_3DConvexHull - Berechnet die konvexe Hülle einer Geometrie.

- CG_3DDifference - 3D-Differenz durchführen
- CG_3DIntersection - 3D-Schnitte durchführen
- CG_3DRotate - Rotates a geometry in 3D space around an axis vector.
- CG_3DScale - Scales a geometry by separate factors along X, Y, and Z axes.
- CG_3DScaleAroundCenter - Scales a geometry in 3D space around a specified center point.
- CG_3DTranslate - Translates (moves) a geometry by given offsets in 3D space.
- CG_3DUnion - Perform 3D union using postgis_sfcgal.
- CG_ConstrainedDelaunayTriangles - Gibt eine eingeschränkte Delaunay-Triangulation um die angegebene Eingabegeometrie zurück.
- CG_Extrude - Extrudieren einer Oberfläche in ein zugehöriges Volumen
- CG_ForceLHR - LHR-Ausrichtung erzwingen
- CG_IsPlanar - Prüfen, ob eine Fläche planar ist oder nicht
- CG_IsSimple - Tests whether a geometry is simple according to SFCGAL
- CG_IsSimpleDetail - Returns a reason when a geometry is not simple according to SFCGAL
- CG_IsSolid - Prüfen, ob die Geometrie ein Solid ist. Es wird keine Gültigkeitsprüfung durchgeführt.
- CG_IsValid - Tests whether a geometry is valid according to SFCGAL
- CG_IsValidDetail - Returns a valid_detail row stating whether a geometry is valid according to SFCGAL, and if not a reason and a location.
- CG_MakeSolid - Gießen Sie die Geometrie in einen Körper. Es wird keine Prüfung durchgeführt. Um ein gültiges Solid zu erhalten, muss die Eingabegeometrie eine geschlossene polyedrische Fläche oder ein geschlossenes TIN sein.
- CG_NurbsCurveApproximate - Creates an approximating NURBS curve fitting data points within a tolerance
- CG_NurbsCurveDerivative - Computes the derivative (tangent, curvature) of a NURBS curve at a given parameter
- CG_NurbsCurveInterpolate - Creates an interpolating NURBS curve passing through all given data points
- CG_Orientation - Bestimmung der Oberflächenausrichtung
- CG_RotateX - Rotates a geometry around the X-axis by a given angle.
- CG_RotateY - Rotates a geometry around the Y-axis by a given angle.
- CG_RotateZ - Rotates a geometry around the Z-axis by a given angle.
- CG_Simplify - Reduces the complexity of a geometry while preserving essential features and Z/M values.
- CG_Tesselate - Führt eine Oberflächentesselierung eines Polygons oder einer Polyederfläche durch und gibt diese als TIN oder Sammlung von TINS zurück
- CG_Visibility - Berechnen eines Sichtbarkeitspolygons aus einem Punkt oder einem Segment in einer Polygongeometrie
- CG_Volume - Computes the volume of a 3D geometry. Closed surfaces can have non-zero volume.
- DropGeometryColumn - Entfernt eine Geometriespalte aus einer räumlichen Tabelle.
- GeometryType - Gibt den Geometrietyp des ST_Geometry Wertes zurück.
- ST_3DClosestPoint - Gibt den 3-dimensionalen Punkt auf g1 zurück, der den kürzesten Abstand zu g2 hat. Dies ist der Anfangspunkt des kürzesten Abstands in 3D.
- ST_3DDFullyWithin - Prüft, ob zwei 3D-Geometrien vollständig innerhalb eines bestimmten 3D-Abstands liegen

- ST_3DDWithin - Prüft, ob zwei 3D-Geometrien innerhalb eines bestimmten 3D-Abstands liegen
- ST_3DDistance - Für den geometrischen Datentyp. Es wird der geringste 3-dimensionale kartesische Abstand (basierend auf dem Koordinatenreferenzsystem) zwischen zwei geometrischen Objekten in projizierten Einheiten zurückgegeben.
- ST_3DExtent - Aggregatfunktion, die den 3D-Begrenzungsrahmen von Geometrien zurückgibt.
- ST_3DInterpolatePoint - Returns the interpolated measure of a geometry closest to a point in 3D.
- ST_3DIntersects - Prüft, ob sich zwei Geometrien in 3D räumlich schneiden - nur für Punkte, Linienzüge, Polygone, polyedrische Flächen (Bereich)
- ST_3DLength - Gibt den geometrischen Schwerpunkt einer Geometrie zurück.
- ST_3DLineInterpolatePoint - Gibt einen oder mehrere, entlang einer Linie interpolierte Punkte zurück.
- ST_3DLongestLine - Gibt die größte 3-dimensionale Distanz zwischen zwei geometrischen Objekten als Linie zurück
- ST_3DMaxDistance - Für den geometrischen Datentyp. Gibt die maximale 3-dimensionale kartesische Distanz (basierend auf dem Koordinatenreferenzsystem) zwischen zwei geometrischen Objekten in projizierten Einheiten zurück.
- ST_3DPerimeter - Gibt den geometrischen Schwerpunkt einer Geometrie zurück.
- ST_3DShortestLine - Gibt den kürzesten 3-dimensionalen Abstand zwischen zwei geometrischen Objekten als Linie zurück
- ST_AddMeasure - Interpoliert Maße entlang einer linearen Geometrie.
- ST_AddPoint - Fügt einem Linienzug einen Punkt hinzu.
- ST_Affine - Wenden Sie eine affine 3D-Transformation auf eine Geometrie an.
- ST_AsBinary - Rückgabe der OGC/ISO Well-Known Binary (WKB)-Darstellung der Geometrie/Geografie ohne SRID-Metadaten.
- ST_AsEWKB - Rückgabe der Extended Well-Known Binary (EWKB) Darstellung der Geometrie mit SRID-Metadaten.
- ST_AsEWKT - Gibt die Well-known-Text(WKT) Darstellung der Geometrie mit den SRID-Metadaten zurück.
- ST_AsGML - Gibt die Geometrie als GML-Element - Version 2 oder 3 - zurück.
- ST_AsGeoJSON - Rückgabe einer Geometrie oder eines Merkmals im GeoJSON-Format.
- ST_AsHEXEWKB - Gibt eine Geometrie im HEXEWKB Format (als Text) aus; verwendet entweder die Little-Endian (NDR) oder die Big-Endian (XDR) Zeichenkodierung.
- ST_AsKML - Gibt die Geometrie als GML-Element - Version 2 oder 3 - zurück.
- ST_AsX3D - Gibt eine Geometrie im X3D XML Knotenelement-Format zurück: ISO-IEC-19776-1.2-X3DEncodings-XML
- ST_Boundary - Gibt die abgeschlossene Hülle aus der kombinierten Begrenzung der Geometrie zurück.
- ST_BoundingDiagonal - Gibt die Diagonale des Umgebungsdreiecks der angegebenen Geometrie zurück.
- ST_CPAWithin - Prüft, ob der nächstgelegene Punkt der Annäherung zweier Flugbahnen innerhalb der angegebenen Entfernung liegt.
- ST_CatmullRomSmoothing - Returns a smoothed version of a geometry, using the Catmull-Rom spline algorithm
- ST_ChaikinSmoothing - Gibt eine geglättete Version einer Geometrie zurück, die den Chaikin-Algorithmus verwendet
- ST_ClosestPointOfApproach - Liefert ein Maß für den nächstgelegenen Punkt der Annäherung von zwei Flugbahnen.
- ST_Collect - Erzeugt eine GeometryCollection oder Multi*-Geometrie aus einer Reihe von Geometrien.
- ST_ConvexHull - Berechnet die konvexe Hülle einer Geometrie.
- ST_CoordDim - Gibt die Dimension der Koordinaten für den Wert von ST_Geometry zurück.
- ST_CurveN - Gibt die N-te Geometrie einer Kurvenkomponente in einer CompoundCurve zurück.

- ST_CurveToLine - Konvertiert eine Geometrie mit Kurven in eine lineare Geometrie.
- ST_DelaunayTriangles - Gibt die Delaunay-Triangulation der Scheitelpunkte einer Geometrie zurück.
- ST_Difference - Berechnet eine Geometrie, die den Teil der Geometrie A darstellt, der die Geometrie B nicht schneidet.
- ST_DistanceCPA - Liefert den Abstand zwischen dem nächstgelegenen Punkt der Annäherung zweier Flugbahnen.
- ST_Dump - Gibt einen Satz von geometry_dump Zeilen für die Komponenten einer Geometrie zurück.
- ST_DumpPoints - Gibt eine Zusammenfassung des Inhalts einer Geometrie wieder.
- ST_DumpRings - Gibt einen Satz von geometry_dump Zeilen für die äußeren und inneren Ringe eines Polygons zurück.
- ST_DumpSegments - Gibt eine Zusammenfassung des Inhalts einer Geometrie wieder.
- ST_EndPoint - Returns the last point of a LineString, CircularLineString, or NURBSCurve.
- ST_ExteriorRing - Gibt die Anzahl der inneren Ringe einer Polygongeometrie aus.
- ST_FlipCoordinates - Gibt eine Version einer Geometrie mit gespiegelter X- und Y-Achse zurück.
- ST_Force2D - Die Geometrien in einen "2-dimensionalen Modus" zwingen.
- ST_ForceCurve - Wandelt einen geometrischen in einen Kurven Datentyp um, soweit anwendbar.
- ST_ForcePolygonCCW - Richtet alle äußeren Ringe gegen den Uhrzeigersinn und alle inneren Ringe mit dem Uhrzeigersinn aus.
- ST_ForcePolygonCW - Richtet alle äußeren Ringe im Uhrzeigersinn und alle inneren Ringe gegen den Uhrzeigersinn aus.
- ST_ForceRHR - Orientiert die Knoten in einem Polygon so, dass sie der Drei-Finger-Regel folgen.
- ST_ForceSFS - Force geometries to use SFS 1.1 or 1.2 geometry types.
- ST_Force3D - Zwingt die Geometrien in einen XYZ Modus. Dies ist ein Alias für ST_Force3DZ.
- ST_Force3DZ - Zwingt die Geometrien in einen XYZ Modus.
- ST_Force4D - Zwingt die Geometrien in einen XYZM Modus.
- ST_ForceCollection - Wandelt eine Geometrie in eine GEOMETRYCOLLECTION um.
- ST_GeomFromEWKB - Gibt einen geometrischen Datentyp (ST_Geometry) aus einer Well-known-Binary (WKB) Darstellung zurück.
- ST_GeomFromEWKT - Gibt einen spezifizierten ST_Geometry-Wert von einer erweiterten Well-known-Text Darstellung (EWKT) zurück.
- ST_GeomFromGML - Nimmt als Eingabe eine GML-Darstellung der Geometrie und gibt ein geometrisches PostGIS-Objekt aus.
- ST_GeomFromGeoJSON - Nimmt als Eingabe eine GeoJSON-Darstellung der Geometrie und gibt ein geometrisches PostGIS-Objekt aus.
- ST_GeomFromKML - Nimmt als Eingabe eine KML-Darstellung der Geometrie und gibt ein geometrisches PostGIS-Objekt aus.
- ST_GeometricMedian - Gibt den geometrischen Median eines Mehrfachpunktes zurück.
- ST_GeometryN - Gibt den Geometrietyp des ST_Geometry Wertes zurück.
- ST_GeometryType - Gibt den Geometrietyp des ST_Geometry Wertes zurück.
- ST_HasArc - Prüft, ob eine Geometrie einen Kreisbogen enthält
- ST_HasM - Prüft, ob eine Geometrie eine M-Dimension (Maß) hat.

- ST_HasZ - Prüft, ob eine Geometrie eine Z-Dimension hat.
- ST_InteriorRingN - Gibt die Anzahl der inneren Ringe einer Polygongeometrie aus.
- ST_InterpolatePoint - Für einen gegebenen Punkt wird die Kilometrierung auf dem nächstliegenden Punkt einer Geometrie zurück.
- ST_Intersection - Berechnet eine Geometrie, die den gemeinsamen Teil der Geometrien A und B darstellt.
- ST_IsClosed - Gibt den Wert TRUE zurück, wenn die Anfangs- und Endpunkte des LINESTRING's zusammenfallen. Bei polyedrischen Oberflächen, wenn sie geschlossen (volumetrisch) sind.
- ST_IsCollection - Gibt den Wert TRUE zurück, falls es sich bei der Geometrie um eine leere GeometryCollection, Polygon, Point etc. handelt.
- ST_IsPolygonCCW - Gibt TRUE zurück, wenn alle äußeren Ringe gegen den Uhrzeigersinn orientiert sind und alle inneren Ringe im Uhrzeigersinn ausgerichtet sind.
- ST_IsPolygonCW - Gibt den Wert TRUE zurück, wenn alle äußeren Ringe im Uhrzeigersinn und alle inneren Ringe gegen den Uhrzeigersinn ausgerichtet sind.
- ST_IsSimple - Gibt den Wert (TRUE) zurück, wenn die Geometrie keine irregulären Stellen, wie Selbstüberschneidungen oder Selbstberührungen, aufweist.
- ST_IsValidTrajectory - Prüft, ob die Geometrie eine gültige Flugbahn ist.
- ST_LengthSpheroid - Gibt den geometrischen Schwerpunkt einer Geometrie zurück.
- ST_LineFromMultiPoint - Erzeugt einen LineString aus einer MultiPoint Geometrie.
- ST_LineInterpolatePoint - Gibt einen oder mehrere, entlang einer Linie interpolierte Punkte zurück.
- ST_LineInterpolatePoints - Gibt einen oder mehrere, entlang einer Linie interpolierte Punkte zurück.
- ST_LineSubstring - Gibt den Teil einer Linie zwischen zwei gebrochenen Stellen zurück.
- ST_LineToCurve - Konvertiert eine lineare Geometrie in eine gekrümmte Geometrie.
- ST_LocateBetweenElevations - Gibt die Teile einer Geometrie zurück, die in einem Höhenbereich (Z) liegen.
- ST_M - Gibt die M-Koordinate eines Punktes zurück.
- ST_MMax - Returns the M maxima of a geometry.
- ST_MMin - Returns the M minima of a geometry.
- ST_MSize - Returns the M size of a geometry.
- ST_MakeLine - Erzeugt einen Linienzug aus einer Punkt-, Mehrfachpunkt- oder Liniengeometrie.
- ST_MakePoint - Erzeugt eine 2D-, 3DZ- oder 4D-Punktgeometrie.
- ST_MakePolygon - Creates a Polygon or CurvePolygon from a shell and optional list of holes.
- ST_MakeValid - Versucht, eine ungültige Geometrie gültig zu machen, ohne dass Scheitelpunkte verloren gehen.
- ST_MemSize - Gibt den Geometrietyp des ST_Geometry Wertes zurück.
- ST_MemUnion - Aggregatfunktion, die Geometrien auf eine speichereffiziente, aber langsamere Weise zusammenfasst
- ST_NDims - Gibt die Dimension der Koordinaten für den Wert von ST_Geometry zurück.
- ST_NPoints - Gibt die Anzahl der Punkte (Knoten) einer Geometrie zurück.
- ST_NRings - Gibt die Anzahl der inneren Ringe einer Polygongeometrie aus.
- ST_Node - Knoten eine Sammlung von Linien.

- ST_NumCurves - Gibt die Anzahl der Kurvenkomponenten einer CompoundCurve zurück.
- ST_NumGeometries - Gibt die Anzahl der Punkte einer Geometrie zurück. Funktioniert für alle Geometrien.
- ST_NumPatches - Returns the number of faces in a PolyhedralSurface or TIN.
- ST_PatchN - Returns the Nth face of a PolyhedralSurface or TIN.
- ST_PointFromWKB - Erzeugt eine Geometrie mit gegebener SRID von WKB.
- ST_PointN - Gibt die Anzahl der Stützpunkte eines ST_LineString oder eines ST_CircularString zurück.
- ST_PointOnSurface - Berechnet einen Punkt, der garantiert in einem Polygon oder auf einer Geometrie liegt.
- ST_Points - Gibt einen MultiPoint zurück, welcher alle Koordinaten einer Geometrie enthält.
- ST_Polygon - Erzeugt ein Polygon aus einem LineString mit einem angegebenen SRID.
- ST_RemovePoint - Einen Punkt aus einem Linienzug entfernen.
- ST_RemoveRepeatedPoints - Gibt eine Version einer Geometrie zurück, bei der doppelte Punkte entfernt wurden.
- ST_Reverse - Gibt die Geometrie in umgekehrter Knotenreihenfolge zurück.
- ST_Rotate - Dreht eine Geometrie um einen Ursprungspunkt.
- ST_RotateX - Dreht eine Geometrie um die X-Achse.
- ST_RotateY - Dreht eine Geometrie um die Y-Achse.
- ST_RotateZ - Dreht eine Geometrie um die Z-Achse.
- ST_Scale - Skaliert eine Geometrie um bestimmte Faktoren.
- ST_Scroll - Startpunkt eines geschlossenen LineStrings ändern.
- ST_SetPoint - Einen Punkt eines Linienzuges durch einen gegebenen Punkt ersetzen.
- ST_ShiftLongitude - Verschiebt die Längenkoordinaten einer Geometrie zwischen -180..180 und 0..360.
- ST_SnapToGrid - Fängt alle Punkte der Eingabegeometrie auf einem regelmäßigen Gitter.
- ST_StartPoint - Returns the first point of a LineString, CircularLineString, or NURBSCurve.
- ST_SwapOrdinates - Gibt eine Version der Ausgangsgeometrie zurück, in der die angegebenen Ordinatenwerte ausgetauscht werden.
- ST_SymDifference - Berechnet eine Geometrie, die die Teile der Geometrien A und B darstellt, die sich nicht überschneiden.
- ST_TransScale - Verschiebt und skaliert eine Geometrie mit vorgegebenen Offsets und Faktoren.
- ST_Translate - Verschiebt eine Geometrie um vorgegebene Offsets.
- ST_UnaryUnion - Berechnet die Vereinigung der Komponenten einer einzelnen Geometrie.
- ST_Union - Berechnet eine Geometrie, die die Punktmengenvereinigung der Eingabegeometrien darstellt.
- ST_WrapX - Versammelt eine Geometrie um einen X-Wert
- ST_X - Gibt die X-Koordinate eines Punktes zurück.
- ST_XMax - Gibt die X-Maxima eines 2D- oder 3D-Begrenzungsrahmens oder einer Geometrie zurück.
- ST_XMin - Gibt die X-Minima eines 2D- oder 3D-Begrenzungsrahmens oder einer Geometrie zurück.
- ST_XSize - Returns the X size of a 2D or 3D bounding box or a geometry.
- ST_Y - Gibt die Y-Koordinate eines Punktes zurück.

- ST_YMax - Gibt die Y-Maxima eines 2D- oder 3D-Begrenzungsrahmens oder einer Geometrie zurück.
- ST_YMin - Gibt die Y-Minima eines 2D- oder 3D-Begrenzungsrahmens oder einer Geometrie zurück.
- ST_YSize - Returns the Y size of a 2D or 3D bounding box or a geometry.
- ST_Z - Gibt die Z-Koordinate eines Punktes zurück.
- ST_ZMFlag - Gibt die Dimension der Koordinaten von ST_Geometry zurück.
- ST_ZMax - Gibt die Z-Maxima eines 2D- oder 3D-Begrenzungsrahmens oder einer Geometrie zurück.
- ST_ZMin - Gibt die Z-Minima eines 2D- oder 3D-Begrenzungsrahmens oder einer Geometrie zurück.
- ST_ZSize - Returns the Z size of a 2D or 3D bounding box or a geometry.
- Equals - Gibt TRUE zurück, wenn zwei TopoGeometry Objekte aus denselben topologischen Elementarstrukturen bestehen.
- Intersects - Gibt TRUE zurück, wenn sich kein beliebiges Paar von Elemtarstrukturen zweier TopoGeometry Objekte überschneidet.
- UpdateGeometrySRID - Aktualisiert die SRID aller Features in einer Geometriespalte und die Metadaten der Tabelle.
- &&& - Gibt TRUE zurück, wenn A's n-D bounding box B's n-D bounding box schneidet.
- &&&(geometry,gidx) - Gibt TRUE zurück, wenn die (cached) n-D bounding box einer Geometrie eine n-D float precision bounding box (GIDX) schneidet.
- &&&(gidx,geometry) - Gibt TRUE zurück, wenn eine n-D float precision bounding box (GIDX) eine (cached) n-D bounding box einer Geometrie schneidet.
- &&&(gidx,gidx) - Gibt TRUE zurück, wenn sich zwei n-D float precision bounding boxes (GIDX) gegenseitig überschneiden.

13.9 PostGIS Funktionen zur Unterstützung gekrümmter Geometrien

Die folgenden Funktionen sind PostGIS-Funktionen, die CIRCULARSTRING, CURVEPOLYGON und andere gekrümmte Geometrietypen verwenden können

- AddGeometryColumn - Entfernt eine Geometriespalte aus einer räumlichen Tabelle.
- Box2D - Gibt ein BOX2D zurück, das die 2D-Ausdehnung einer Geometrie darstellt.
- Box3D - Gibt ein BOX3D zurück, das die 3D-Ausdehnung einer Geometrie darstellt.
- DropGeometryColumn - Entfernt eine Geometriespalte aus einer räumlichen Tabelle.
- GeometryType - Gibt den Geometrietyp des ST_Geometry Wertes zurück.
- PostGIS_AddBBox - Fügt der Geometrie ein umschreibendes Rechteck bei.
- PostGIS_DropBBox - Löscht die umschreibenden Rechtecke der Geometrie.
- PostGIS_HasBBox - Gibt TRUE zurück, wenn die BBox der Geometrie zwischengespeichert ist, andernfalls wird FALSE zurückgegeben.
- ST_3DExtent - Aggregatfunktion, die den 3D-Begrenzungsrahmen von Geometrien zurückgibt.
- ST_Affine - Wenden Sie eine affine 3D-Transformation auf eine Geometrie an.
- ST_AsBinary - Rückgabe der OGC/ISO Well-Known Binary (WKB)-Darstellung der Geometrie/Geografie ohne SRID-Metadaten.
- ST_AsEWKB - Rückgabe der Extended Well-Known Binary (EWKB) Darstellung der Geometrie mit SRID-Metadaten.
- ST_AsEWKT - Gibt die Well-known-Text(WKT) Darstellung der Geometrie mit den SRID-Metadaten zurück.

- ST_AsHEXEWKB - Gibt eine Geometrie im HEXEWKB Format (als Text) aus; verwendet entweder die Little-Endian (NDR) oder die Big-Endian (XDR) Zeichenkodierung.
- ST_AsSVG - Gibt eine Geometrie als SVG-Pfad aus.
- ST_AsText - Gibt die Well-known-Text(WKT) Darstellung der Geometrie/Geographie ohne die SRID Metadaten zurück.
- ST_ClusterDBSCAN - Fensterfunktion, die eine Cluster-ID für jede Eingabegeometrie unter Verwendung des DBSCAN-Algorithmus zurückgibt.
- ST_ClusterWithin - Aggregatfunktion, die Geometrien nach Trennungsabstand gruppiert.
- ST_ClusterWithinWin - Fensterfunktion, die eine Cluster-ID für jede Eingabegeometrie zurückgibt, Clustering anhand des Trennungsabstands.
- ST_Collect - Erzeugt eine GeometryCollection oder Multi*-Geometrie aus einer Reihe von Geometrien.
- ST_CoordDim - Gibt die Dimension der Koordinaten für den Wert von ST_Geometry zurück.
- ST_CurveToLine - Konvertiert eine Geometrie mit Kurven in eine lineare Geometrie.
- ST_Distance - Gibt die größte 3-dimensionale Distanz zwischen zwei geometrischen Objekten als Linie zurück
- ST_Dump - Gibt einen Satz von geometry_dump Zeilen für die Komponenten einer Geometrie zurück.
- ST_DumpPoints - Gibt eine Zusammenfassung des Inhalts einer Geometrie wieder.
- ST_EndM - Returns the M coordinate of the last point of a LineString, CircularLineString, or NURBSCurve.
- ST_EndPoint - Returns the last point of a LineString, CircularLineString, or NURBSCurve.
- ST_EstimatedExtent - Gibt die geschätzte Ausdehnung einer räumlichen Tabelle zurück.
- ST_FlipCoordinates - Gibt eine Version einer Geometrie mit gespiegelter X- und Y-Achse zurück.
- ST_Force2D - Die Geometrien in einen "2-dimensionalen Modus" zwingen.
- ST_ForceCurve - Wandelt einen geometrischen in einen Kurven Datentyp um, soweit anwendbar.
- ST_ForceSFS - Force geometries to use SFS 1.1 or 1.2 geometry types.
- ST_Force3D - Zwingt die Geometrien in einen XYZ Modus. Dies ist ein Alias für ST_Force3DZ.
- ST_Force3DM - Zwingt die Geometrien in einen XYM Modus.
- ST_Force3DZ - Zwingt die Geometrien in einen XYZ Modus.
- ST_Force4D - Zwingt die Geometrien in einen XYZM Modus.
- ST_ForceCollection - Wandelt eine Geometrie in eine GEOMETRYCOLLECTION um.
- ST_GeoHash - Gibt die Geometrie in der GeoHash Darstellung aus.
- ST_GeogFromWKB - Erzeugt ein geographisches Objekt aus der Well-known-Binary (WKB) oder der erweiterten Well-known-Binary (EWKB) Darstellung.
- ST_GeomFromEWKB - Gibt einen geometrischen Datentyp (ST_Geometry) aus einer Well-known-Binary (WKB) Darstellung zurück.
- ST_GeomFromEWKT - Gibt einen spezifizierten ST_Geometry-Wert von einer erweiterten Well-known-Text Darstellung (EWKT) zurück.
- ST_GeomFromText - Gibt einen spezifizierten ST_Geometry Wert aus einer Well-known-Text Darstellung (WKT) zurück.
- ST_GeomFromWKB - Erzeugt ein geometrisches Objekt aus der Well-known-Binary (WKB) Darstellung und einer optionalen SRID.

- ST_GeometryN - Gibt den Geometrietyp des ST_Geometry Wertes zurück.
- = - Gibt TRUE zurück, wenn die Koordinaten und die Reihenfolge der Koordinaten der Geometrie/Geographie A und der Geometrie/Geographie B ident sind.
- &<| - Gibt TRUE zurück, wenn die bounding box von A jene von B überlagert oder unterhalb liegt.
- ST_HasArc - Prüft, ob eine Geometrie einen Kreisbogen enthält
- ST_Intersects - Prüft, ob sich zwei Geometrien schneiden (sie haben mindestens einen Punkt gemeinsam)
- ST_IsClosed - Gibt den Wert TRUE zurück, wenn die Anfangs- und Endpunkte des LINESTRING's zusammenfallen. Bei polyedrischen Oberflächen, wenn sie geschlossen (volumetrisch) sind.
- ST_IsCollection - Gibt den Wert TRUE zurück, falls es sich bei der Geometrie um eine leere GeometryCollection, Polygon, Point etc. handelt.
- ST_IsEmpty - Prüft, ob eine Geometrie leer ist.
- ST_LineToCurve - Konvertiert eine lineare Geometrie in eine gekrümmte Geometrie.
- ST_MMax - Returns the M maxima of a geometry.
- ST_MMin - Returns the M minima of a geometry.
- ST_MSize - Returns the M size of a geometry.
- ST_MemSize - Gibt den Geometrietyp des ST_Geometry Wertes zurück.
- ST_NPoints - Gibt die Anzahl der Punkte (Knoten) einer Geometrie zurück.
- ST_NRings - Gibt die Anzahl der inneren Ringe einer Polygongeometrie aus.
- ST_PointFromWKB - Erzeugt eine Geometrie mit gegebener SRID von WKB.
- ST_PointN - Gibt die Anzahl der Stützpunkte eines ST_LineString oder eines ST_CircularString zurück.
- ST_Points - Gibt einen MultiPoint zurück, welcher alle Koordinaten einer Geometrie enthält.
- ST_Rotate - Dreht eine Geometrie um einen Ursprungspunkt.
- ST_RotateZ - Dreht eine Geometrie um die Z-Achse.
- ST_SRID - Gibt die Raumbezugskennung für eine Geometrie zurück.
- ST_Scale - Skaliert eine Geometrie um bestimmte Faktoren.
- ST_SetEndM - Sets the M coordinate of the last point of a LineString, CircularLineString, or NURBSCurve.
- ST_SetSRID - Legen Sie den SRID für eine Geometrie fest.
- ST_SetStartM - Sets the M coordinate of the first point of a LineString, CircularLineString, or NURBSCurve.
- ST_StartM - Returns the M coordinate of the first point of a LineString, CircularLineString, or NURBSCurve.
- ST_StartPoint - Returns the first point of a LineString, CircularLineString, or NURBSCurve.
- ST_Summary - Gibt eine Zusammenfassung des Inhalts einer Geometrie wieder.
- ST_SwapOrdinates - Gibt eine Version der Ausgangsgeometrie zurück, in der die angegebenen Ordinatenwerte ausgetauscht werden.
- ST_TransScale - Verschiebt und skaliert eine Geometrie mit vorgegebenen Offsets und Faktoren.
- ST_Transform - Rückgabe einer neuen Geometrie mit in ein anderes räumliches Bezugssystem transformierten Koordinaten.
- ST_Translate - Verschiebt eine Geometrie um vorgegebene Offsets.

- ST_XMax - Gibt die X-Maxima eines 2D- oder 3D-Begrenzungsrahmens oder einer Geometrie zurück.
- ST_XMin - Gibt die X-Minima eines 2D- oder 3D-Begrenzungsrahmens oder einer Geometrie zurück.
- ST_XSize - Returns the X size of a 2D or 3D bounding box or a geometry.
- ST_YMax - Gibt die Y-Maxima eines 2D- oder 3D-Begrenzungsrahmens oder einer Geometrie zurück.
- ST_YMin - Gibt die Y-Minima eines 2D- oder 3D-Begrenzungsrahmens oder einer Geometrie zurück.
- ST_YSize - Returns the Y size of a 2D or 3D bounding box or a geometry.
- ST_ZMFlag - Gibt die Dimension der Koordinaten von ST_Geometry zurück.
- ST_ZMax - Gibt die Z-Maxima eines 2D- oder 3D-Begrenzungsrahmens oder einer Geometrie zurück.
- ST_ZMin - Gibt die Z-Minima eines 2D- oder 3D-Begrenzungsrahmens oder einer Geometrie zurück.
- ST_ZSize - Returns the Z size of a 2D or 3D bounding box or a geometry.
- UpdateGeometrySRID - Aktualisiert die SRID aller Features in einer Geometriespalte und die Metadaten der Tabelle.
- ~(box2df,box2df) - Gibt TRUE zurück, wenn eine 2D float precision bounding box (BOX2DF) eine andere 2D float precision bounding box (BOX2DF) enthält.
- ~(box2df,geometry) - Gibt TRUE zurück, wenn eine 2D float precision bounding box (BOX2DF) die 2D Bounding Box einer Geometrie enthält.
- ~(geometry,box2df) - Gibt TRUE zurück, wenn die 2D bounding box einer Geometrie eine 2D float precision bounding box (GIDX) enthält.
- && - Gibt TRUE zurück, wenn die 2D Bounding Box von A die 2D Bounding Box von B schneidet.
- &&& - Gibt TRUE zurück, wenn A's n-D bounding box B's n-D bounding box schneidet.
- @(box2df,box2df) - Gibt TRUE zurück, wenn eine 2D float precision bounding box (BOX2DF) innerhalb einer anderen 2D float precision bounding box enthalten ist.
- @(box2df,geometry) - Gibt TRUE zurück, wenn eine 2D float precision bounding box (BOX2DF) in der 2D Bounding Box einer Geometrie enthalten ist..
- @(geometry,box2df) - Gibt TRUE zurück, wenn die 2D Bounding Box einer Geometrie in einer 2D float precision Bbounding Box (BOX2DF) enthalten ist.
- &&(box2df,box2df) - Gibt TRUE zurück, wenn sich zwei 2D float precision Bounding Boxes (BOX2DF) überschneiden.
- &&(box2df,geometry) - Gibt TRUE zurück, wenn eine 2D float precision bounding box (BOX2DF) eine Geometrie (cached) 2D bounding box schneidet.
- &&(geometry,box2df) - Gibt TRUE zurück, wenn sich die 2D Bounding Box (cached) einer Geometrie mit einer 2D Bounding Box mit Gleitpunktgenauigkeit (BOX2DF) überschneidet.
- &&&(geometry,gidx) - Gibt TRUE zurück, wenn die (cached) n-D bounding box einer Geometrie eine n-D float precision bounding box (GIDX) schneidet.
- &&&(gidx,geometry) - Gibt TRUE zurück, wenn eine n-D float precision bounding box (GIDX) eine (cached) n-D bounding box einer Geometrie schneidet.
- &&&(gidx,gidx) - Gibt TRUE zurück, wenn sich zwei n-D float precision bounding boxes (GIDX) gegenseitig überschneiden.

13.10 PostGIS-Funktionen zur Unterstützung polyedrischer Flächen

Die folgenden Funktionen sind PostGIS-Funktionen, die die Geometrien POLYHEDRALSURFACE und POLYHEDRALSURFACEM verwenden können

- Box2D - Gibt ein BOX2D zurück, das die 2D-Ausdehnung einer Geometrie darstellt.
- Box3D - Gibt ein BOX3D zurück, das die 3D-Ausdehnung einer Geometrie darstellt.
- CG_3DArea - Berechnet die Fläche von 3D-Oberflächengeometrien. Gibt 0 für Solids zurück.
- CG_3DConvexHull - Berechnet die konvexe Hülle einer Geometrie.
- CG_3DDifference - 3D-Differenz durchführen
- CG_3DIntersection - 3D-Schnitte durchführen
- CG_3DUnion - Perform 3D union using postgis_sfcgal.
- CG_ApproximateMedialAxis - Berechnet die konvexe Hülle einer Geometrie.
- CG_Extrude - Extrudieren einer Oberfläche in ein zugehöriges Volumen
- CG_ForceLHR - LHR-Ausrichtung erzwingen
- CG_GenerateFlatRoof - Generate a flat (box) roof from a footprint polygon.
- CG_GenerateGableRoof - Generate a gable roof from a footprint polygon.
- CG_GenerateHippedRoof - Generate a hipped roof from a footprint polygon.
- CG_GenerateRoof - Generate a roof of the requested type from a footprint polygon.
- CG_GenerateSkillionRoof - Generate a skillion (single-slope) roof from a footprint polygon.
- CG_IsPlanar - Prüfen, ob eine Fläche planar ist oder nicht
- CG_IsSimple - Tests whether a geometry is simple according to SFCGAL
- CG_IsSimpleDetail - Returns a reason when a geometry is not simple according to SFCGAL
- CG_IsSolid - Prüfen, ob die Geometrie ein Solid ist. Es wird keine Gültigkeitsprüfung durchgeführt.
- CG_IsValid - Tests whether a geometry is valid according to SFCGAL
- CG_IsValidDetail - Returns a valid_detail row stating whether a geometry is valid according to SFCGAL, and if not a reason and a location.
- CG_MakeSolid - Gießen Sie die Geometrie in einen Körper. Es wird keine Prüfung durchgeführt. Um ein gültiges Solid zu erhalten, muss die Eingabegeometrie eine geschlossene polyedrische Fläche oder ein geschlossenes TIN sein.
- CG_PolygonRepair - Repair an invalid polygon or multipolygon.
- CG_StraightSkeleton - Berechnet die konvexe Hülle einer Geometrie.
- CG_Tesselate - Führt eine Oberflächentesselierung eines Polygons oder einer Polyederfläche durch und gibt diese als TIN oder Sammlung von TINS zurück
- CG_Visibility - Berechnen eines Sichtbarkeitspolygons aus einem Punkt oder einem Segment in einer Polygongeometrie
- CG_Volume - Computes the volume of a 3D geometry. Closed surfaces can have non-zero volume.
- GeometryType - Gibt den Geometrietyp des ST_Geometry Wertes zurück.
- ST_3DClosestPoint - Gibt den 3-dimensionalen Punkt auf g1 zurück, der den kürzesten Abstand zu g2 hat. Dies ist der Anfangspunkt des kürzesten Abstands in 3D.

- ST_3DDFullyWithin - Prüft, ob zwei 3D-Geometrien vollständig innerhalb eines bestimmten 3D-Abstands liegen
- ST_3DDWithin - Prüft, ob zwei 3D-Geometrien innerhalb eines bestimmten 3D-Abstands liegen
- ST_3DDistance - Für den geometrischen Datentyp. Es wird der geringste 3-dimensionale kartesische Abstand (basierend auf dem Koordinatenreferenzsystem) zwischen zwei geometrischen Objekten in projizierten Einheiten zurückgegeben.
- ST_3DExtent - Aggregatfunktion, die den 3D-Begrenzungsrahmen von Geometrien zurückgibt.
- ST_3DIntersects - Prüft, ob sich zwei Geometrien in 3D räumlich schneiden - nur für Punkte, Linienzüge, Polygone, polyedrische Flächen (Bereich)
- ST_3DLongestLine - Gibt die größte 3-dimensionale Distanz zwischen zwei geometrischen Objekten als Linie zurück
- ST_3DMaxDistance - Für den geometrischen Datentyp. Gibt die maximale 3-dimensionale kartesische Distanz (basierend auf dem Koordinatenreferenzsystem) zwischen zwei geometrischen Objekten in projizierten Einheiten zurück.
- ST_3DShortestLine - Gibt den kürzesten 3-dimensionalen Abstand zwischen zwei geometrischen Objekten als Linie zurück
- ST_Affine - Wenden Sie eine affine 3D-Transformation auf eine Geometrie an.
- ST_Area - Gibt den geometrischen Schwerpunkt einer Geometrie zurück.
- ST_AsBinary - Rückgabe der OGC/ISO Well-Known Binary (WKB)-Darstellung der Geometrie/Geografie ohne SRID-Metadaten.
- ST_AsEWKB - Rückgabe der Extended Well-Known Binary (EWKB) Darstellung der Geometrie mit SRID-Metadaten.
- ST_AsEWKT - Gibt die Well-known-Text(WKT) Darstellung der Geometrie mit den SRID-Metadaten zurück.
- ST_AsGML - Gibt die Geometrie als GML-Element - Version 2 oder 3 - zurück.
- ST_AsX3D - Gibt eine Geometrie im X3D XML Knotenelement-Format zurück: ISO-IEC-19776-1.2-X3DEncodings-XML
- ST_CoordDim - Gibt die Dimension der Koordinaten für den Wert von ST_Geometry zurück.
- ST_Dimension - Gibt die Dimension der Koordinaten für den Wert von ST_Geometry zurück.
- ST_Dump - Gibt einen Satz von geometry_dump Zeilen für die Komponenten einer Geometrie zurück.
- ST_DumpPoints - Gibt eine Zusammenfassung des Inhalts einer Geometrie wieder.
- ST_Expand - Gibt einen Begrenzungsrahmen zurück, der aus einem anderen Begrenzungsrahmen oder einer Geometrie erweitert wurde.
- ST_Extent - Aggregatfunktion, die den Begrenzungsrahmen von Geometrien zurückgibt.
- ST_FlipCoordinates - Gibt eine Version einer Geometrie mit gespiegelter X- und Y-Achse zurück.
- ST_Force2D - Die Geometrien in einen "2-dimensionalen Modus" zwingen.
- ST_ForceRHR - Orientiert die Knoten in einem Polygon so, dass sie der Drei-Finger-Regel folgen.
- ST_ForceSFS - Force geometries to use SFS 1.1 or 1.2 geometry types.
- ST_Force3D - Zwingt die Geometrien in einen XYZ Modus. Dies ist ein Alias für ST_Force3DZ.
- ST_Force3DZ - Zwingt die Geometrien in einen XYZ Modus.
- ST_ForceCollection - Wandelt eine Geometrie in eine GEOMETRYCOLLECTION um.
- ST_GeomFromEWKB - Gibt einen geometrischen Datentyp (ST_Geometry) aus einer Well-known-Binary (WKB) Darstellung zurück.
- ST_GeomFromEWKT - Gibt einen spezifizierten ST_Geometry-Wert von einer erweiterten Well-known-Text Darstellung (EWKT) zurück.
- ST_GeomFromGML - Nimmt als Eingabe eine GML-Darstellung der Geometrie und gibt ein geometrisches PostGIS-Objekt aus.

- ST_GeometryN - Gibt den Geometrietyp des ST_Geometry Wertes zurück.
- ST_GeometryType - Gibt den Geometrietyp des ST_Geometry Wertes zurück.
- = - Gibt TRUE zurück, wenn die Koordinaten und die Reihenfolge der Koordinaten der Geometrie/Geographie A und der Geometrie/Geographie B ident sind.
- &<| - Gibt TRUE zurück, wenn die bounding box von A jene von B überlagert oder unterhalb liegt.
- ~= - Gibt TRUE zurück, wenn die bounding box von A ident mit jener von B ist.
- ST_IsClosed - Gibt den Wert TRUE zurück, wenn die Anfangs- und Endpunkte des LINESTRING's zusammenfallen. Bei polyedrischen Oberflächen, wenn sie geschlossen (volumetrisch) sind.
- ST_MemSize - Gibt den Geometrietyp des ST_Geometry Wertes zurück.
- ST_NPoints - Gibt die Anzahl der Punkte (Knoten) einer Geometrie zurück.
- ST_NumGeometries - Gibt die Anzahl der Punkte einer Geometrie zurück. Funktioniert für alle Geometrien.
- ST_NumPatches - Returns the number of faces in a PolyhedralSurface or TIN.
- ST_PatchN - Returns the Nth face of a PolyhedralSurface or TIN.
- ST_RemoveRepeatedPoints - Gibt eine Version einer Geometrie zurück, bei der doppelte Punkte entfernt wurden.
- ST_Reverse - Gibt die Geometrie in umgekehrter Knotenreihenfolge zurück.
- ST_Rotate - Dreht eine Geometrie um einen Ursprungspunkt.
- ST_RotateX - Dreht eine Geometrie um die X-Achse.
- ST_RotateY - Dreht eine Geometrie um die Y-Achse.
- ST_RotateZ - Dreht eine Geometrie um die Z-Achse.
- ST_Scale - Skaliert eine Geometrie um bestimmte Faktoren.
- ST_ShiftLongitude - Verschiebt die Längenkoordinaten einer Geometrie zwischen -180..180 und 0..360.
- ST_Summary - Gibt eine Zusammenfassung des Inhalts einer Geometrie wieder.
- ST_SwapOrdinates - Gibt eine Version der Ausgangsgeometrie zurück, in der die angegebenen Ordinatenwerte ausgetauscht werden.
- ST_Transform - Rückgabe einer neuen Geometrie mit in ein anderes räumliches Bezugssystem transformierten Koordinaten.
- ~(box2df,box2df) - Gibt TRUE zurück, wenn eine 2D float precision bounding box (BOX2DF) eine andere 2D float precision bounding box (BOX2DF) enthält.
- ~(box2df,geometry) - Gibt TRUE zurück, wenn eine 2D float precision bounding box (BOX2DF) die 2D Bounding Box einer Geometrie enthält.
- ~(geometry,box2df) - Gibt TRUE zurück, wenn die 2D bounding box einer Geometrie eine 2D float precision bounding box (GIDX) enthält.
- && - Gibt TRUE zurück, wenn die 2D Bounding Box von A die 2D Bounding Box von B schneidet.
- &&& - Gibt TRUE zurück, wenn A's n-D bounding box B's n-D bounding box schneidet.
- @(box2df,box2df) - Gibt TRUE zurück, wenn eine 2D float precision bounding box (BOX2DF) innerhalb einer anderen 2D float precision bounding box enthalten ist.
- @(box2df,geometry) - Gibt TRUE zurück, wenn eine 2D float precision bounding box (BOX2DF) in der 2D Bounding Box einer Geometrie enthalten ist..

- @(geometry,box2df) - Gibt TRUE zurück, wenn die 2D Bounding Box einer Geometrie in einer 2D float precision Bbounding Box (BOX2DF) enthalten ist.
- &&(box2df,box2df) - Gibt TRUE zurück, wenn sich zwei 2D float precision Bounding Boxes (BOX2DF) überschneiden.
- &&(box2df,geometry) - Gibt TRUE zurück, wenn eine 2D float precision bounding box (BOX2DF) eine Geometrie (cached) 2D bounding box schneidet.
- &&(geometry,box2df) - Gibt TRUE zurück, wenn sich die 2D Bounding Box (cached) einer Geometrie mit einer 2D Bounding Box mit Gleitpunktgenauigkeit (BOX2DF) überschneidet.
- &&&(geometry,gidx) - Gibt TRUE zurück, wenn die (cached) n-D bounding box einer Geometrie eine n-D float precision bounding box (GIDX) schneidet.
- &&&(gidx,geometry) - Gibt TRUE zurück, wenn eine n-D float precision bounding box (GIDX) eine (cached) n-D bounding box einer Geometrie schneidet.
- &&&(gidx,gidx) - Gibt TRUE zurück, wenn sich zwei n-D float precision bounding boxes (GIDX) gegenseitig überschneiden.

13.11 PostGIS Funktionsunterstützungsmatrix

Nachfolgend finden Sie eine alphabetische Auflistung der raumspezifischen Funktionen in PostGIS und die Arten von Raumtypen, mit denen sie arbeiten, bzw. die OGC/SQL-Konformität, die sie zu erfüllen versuchen.

- Ein ✓ bedeutet, dass die Funktion mit dem Typ oder Subtyp von Haus aus arbeitet.
- A bedeutet, dass es funktioniert, aber mit einem Transformations-Cast eingebaut mit Cast-to-Geometrie, transformieren, um eine "beste srid" räumliche ref und dann zurück zu werfen. Die Ergebnisse sind möglicherweise nicht wie erwartet für große Bereiche oder Bereiche an Polen und können Fließkomma-Müll ansammeln.
- Eine bedeutet, dass die Funktion mit dem Typ funktioniert, weil ein Auto-Cast auf einen anderen, wie z.B. auf box3d, anstatt einer direkten Typunterstützung.
- Eine bedeutet, dass die Funktion nur verfügbar ist, wenn PostGIS mit SFCGAL-Unterstützung kompiliert wurde.
- geom - Grundlegende 2D-Geometrieunterstützung (x,y).
- geog - Grundlegende 2D-Geografie-Unterstützung (x,y).
- 2.5D - grundlegende 2D-Geometrien im 3 D/4D-Raum (mit Z- oder M-Koordinaten).
- PS - Polyedrische Flächen
- T - Dreiecke und triangulierte unregelmäßige Netzflächen (TIN)

Funktion	geom	geog	2.5D	Kurven	SQL MM	PS	T
ST_Collect	✓		✓	✓			
ST_LineFromMulti	✓		✓				
ST_MakeEnvelope	✓						
ST_MakeLine	✓		✓				
ST_MakePoint	✓		✓				

Funktion	geom	geog	2.5D	Kurven	SQL MM	PS	T
ST_MakePointM	✓						
ST_MakePolygon	✓		✓				
ST_Point	✓				✓		
ST_PointZ	✓						
ST_PointM	✓						
ST_PointZM	✓						
ST_Polygon	✓		✓		✓		
ST_TileEnvelope	✓						
ST_HexagonGrid	✓						
ST_Hexagon	✓						
ST_SquareGrid	✓						
ST_Square	✓						
ST_Letters	✓						
ST_MakeNurbsCur	✓						
GeometryType	✓		✓	✓		✓	✓
ST_Boundary	✓		✓		✓		
ST_BoundingDiago	✓		✓				
ST_CoordDim	✓		✓	✓	✓	✓	✓
ST_Dimension	✓				✓	✓	✓
ST_Dump	✓		✓	✓		✓	✓
ST_DumpPoints	✓		✓	✓		✓	✓
ST_DumpSegments	✓		✓				✓
ST_DumpRings	✓		✓				
ST_EndPoint	✓		✓	✓	✓		
ST_Envelope	✓				✓		
ST_ExteriorRing	✓		✓		✓		
ST_GeometryN	✓		✓	✓	✓	✓	✓
ST_GeometryType	✓		✓		✓	✓	
ST_HasArc	✓		✓	✓			
ST_InteriorRingN	✓		✓		✓		
ST_NumCurves	✓		✓		✓		
ST_CurveN	✓		✓		✓		
ST_IsClosed	✓		✓	✓	✓	✓	
ST_IsCollection	✓		✓	✓			

Funktion	geom	geog	2.5D	Kurven	SQL MM	PS	T
ST_IsEmpty	✓			✓	✓		
ST_IsPolygonCCW	✓		✓				
ST_IsPolygonCW	✓		✓				
ST_IsRing	✓				✓		
ST_IsSimple	✓		✓		✓		
ST_IsSimpleDetail	✓						
ST_M	✓		✓		✓		
ST_MemSize	✓		✓	✓		✓	✓
ST_NDims	✓		✓				
ST_NPoints	✓		✓	✓		✓	
ST_NRings	✓		✓	✓			
ST_NumGeometries	✓		✓		✓	✓	✓
ST_NumInteriorRin	✓				✓		
ST_NumInteriorRin	✓						
ST_NumPatches	✓		✓		✓	✓	
ST_NumPoints	✓				✓		
ST_PatchN	✓		✓		✓	✓	✓
ST_PointN	✓		✓	✓	✓		
ST_Points	✓		✓	✓			
ST_StartPoint	✓		✓	✓	✓		
ST_StartM	✓			✓			
ST_EndM	✓			✓			
ST_SetStartM	✓			✓			
ST_SetEndM	✓			✓			
ST_Summary	✓	✓		✓		✓	✓
ST_X	✓		✓		✓		
ST_Y	✓		✓		✓		
ST_Z	✓		✓		✓		
ST_ZMFlag	✓		✓	✓			
ST_HasZ	✓		✓				
ST_HasM	✓		✓				
ST_ControlPoints	✓						
ST_Degree	✓						
ST_Weights	✓						

Funktion	geom	geog	2.5D	Kurven	SQL MM	PS	T
ST_Knots	✓						
ST_NumControlPoi	✓						
ST_NurbsCurveIsR	✓						
ST_IsNurbsCurve	✓						
ST_Evaluate	✓						
ST_NurbsToLineSt	✓						
ST_AddPoint	✓		✓				
ST_CollectionExtra	✓						
ST_CollectionHom	✓						
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	✓						
ST_RemovePoint	✓		✓				
ST_RemoveRepeate	✓		✓			✓	
ST_RemoveIrreleva	✓						
ST_RemoveSmallP	✓						
ST_Reverse	✓		✓			✓	

Funktion	geom	geog	2.5D	Kurven	SQL MM	PS	T
ST_Segmentize	✓	✓					
ST_SetPoint	✓		✓				
ST_ShiftLongitude	✓		✓			✓	✓
ST_WrapX	✓		✓				
ST_SnapToGrid	✓		✓				
ST_Snap	✓						
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_GeomFromEW	✓		✓	✓		✓	✓
ST_GeomFromMA	✓ 1						
ST_GeometryFrom	✓				✓		
ST_GeomFromText	✓			✓	✓		
ST_LineFromText	✓				✓		
ST_MLineFromTex	✓				✓		
ST_MPointFromTe	✓				✓		
ST_MPolyFromTex	✓				✓		
ST_PointFromText	✓				✓		

Funktion	geom	geog	2.5D	Kurven	SQL MM	PS	T
ST_PolygonFromTe	✓				✓		
ST_WKTToSQL	✓				✓		
ST_GeogFromWKB		✓		✓			
ST_GeomFromEWI	✓		✓	✓		✓	✓
ST_GeomFromWK	✓			✓	✓		
ST_LineFromWKB	✓				✓		
ST_LinestringFrom	✓ B				✓		
ST_PointFromWKE	✓		✓	✓	✓		
ST_WKBToSQL	✓				✓		
ST_Box2dFromGeo	✓ h						
ST_GeomFromGeo	✓ ı						
ST_GeomFromGML	✓		✓			✓	✓
ST_GeomFromGeo	✓ N		✓				
ST_GeomFromKML	✓		✓				
ST_GeomFromTWI	✓						
ST_GMLToSQL	✓				✓		
ST_LineFromEncoc	✓ olyline						
ST_PointFromGeoHash							
ST_FromFlatGeobufToTable							
ST_FromFlatGeobuf							
ST_AsEWKT	✓	✓	✓	✓		✓	✓
ST_AsText	✓	✓		✓	✓		
ST_AsBinary	✓	✓	✓	✓	✓	✓	✓
ST_AsEWKB	✓		✓	✓		✓	✓
ST_AsHEXEWKB	✓		✓	✓			
ST_AsEncodedPoly	✓						
ST_AsFlatGeobuf	✓						
ST_AsGeobuf	✓						
ST_AsGeoJSON	✓	✓	✓				
ST_AsGML	✓	✓	✓		✓	✓	✓
ST_AsKML	✓	✓	✓				
ST_AsLatLonText	✓						
ST_AsMARC21	✓						
ST_AsMVTGeom	✓						
ST_AsMVT	✓						

Funktion	geom	geog	2.5D	Kurven	SQL MM	PS	T
ST_AsSVG	✓	✓		✓			
ST_AsTWKB	✓						
ST_AsX3D	✓		✓			✓	✓
ST_GeoHash	✓			✓			
&&	✓	✓		✓		✓	
&&(geometry,box2d	✓			✓		✓	
&&(box2df,geomet	✓			✓		✓	
&&(box2df,box2df)	✓			✓		✓	
&&&	✓		✓	✓		✓	✓
&&&(geometry,gid	✓		✓	✓		✓	✓
&&&(gidx,geometr	✓		✓	✓		✓	✓
&&&(gidx,gidx)			✓	✓		✓	✓
&<	✓						
&<|	✓			✓		✓	
&>	✓						
<<	✓						
<<|	✓						
=	✓	✓		✓		✓	
>>	✓						
@	✓						
@(geometry,box2df	✓			✓		✓	
@(box2df,geometry	✓			✓		✓	
@(box2df,box2df)	✓			✓		✓	
|&>	✓						
|>>	✓						
~	✓						
~(geometry,box2df)	✓			✓		✓	
~(box2df,geometry)	✓			✓		✓	
~(box2df,box2df)	✓			✓		✓	
~=	✓					✓	
<->	✓	✓					
|=|	✓						
<#>	✓						
<<->>	✓						

Funktion	geom	geog	2.5D	Kurven	SQL MM	PS	T
ST_3DIntersects	✓		✓		✓	✓	✓
ST_Contains	✓				✓		
ST_ContainsProper	✓						
ST_CoveredBy	✓	✓					
ST_Covers	✓	✓					
ST_Crosses	✓				✓		
ST_Disjoint	✓				✓		
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	✓						

Funktion	geom	geog	2.5D	Kurven	SQL MM	PS	T
ST_3DLength	✓		✓		✓		
ST_LengthSpheroid	✓		✓				
ST_LongestLine	✓						
ST_3DLongestLine	✓		✓			✓	
ST_MaxDistance	✓						
ST_3DMaxDistance	✓		✓			✓	
ST_MinimumClear	✓						
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_CatmullRomSm	✓ ing		✓				
ST_ConcaveHull	✓						
ST_ConvexHull	✓		✓		✓		
ST_DelaunayTriang	✓		✓				✓
ST_FilterByM	✓						
ST_GeneratePoints	✓						
ST_GeometricMedi	✓		✓				

Funktion	geom	geog	2.5D	Kurven	SQL MM	PS	T
ST_LineMerge	✓						
ST_MaximumInscri	✓ Circle						
ST_LargestEmptyC	✓						
ST_MinimumBoun	✓ Circle						
ST_MinimumBoun	✓ Radius						
ST_OrientedEnvelo	✓						
ST_OffsetCurve	✓						
ST_PointOnSurface	✓		✓		✓		
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_CoverageClean	✓						
ST_CoverageSimpli	✓						
ST_CoverageEdges	✓						
ST_CoverageUnion	✓						
ST_Affine	✓		✓	✓		✓	✓
ST_Rotate	✓		✓	✓		✓	✓
ST_RotateX	✓		✓			✓	✓
ST_RotateY	✓		✓			✓	✓
ST_RotateZ	✓		✓	✓		✓	✓
ST_Scale	✓		✓	✓		✓	✓
ST_Translate	✓		✓	✓			
ST_TransScale	✓		✓	✓			
ST_ClusterDBSCA	✓			✓			
ST_ClusterIntersect	✓						

Funktion	geom	geog	2.5D	Kurven	SQL MM	PS	T
ST_ClusterIntersect	✓ Vin						
ST_ClusterRelateW	✓						
ST_MinimumSpanr	✓ Tree						
ST_ClusterKMeans	✓						
ST_ClusterWithin	✓			✓			
ST_ClusterWithinW	✓			✓			
Box2D	✓			✓		✓	✓
Box3D	✓		✓	✓		✓	✓
ST_EstimatedExten	☑			✓			
ST_Expand	✓					✓	✓
ST_Extent	✓					✓	✓
ST_3DExtent	✓		✓	✓		✓	✓
ST_MakeBox2D	✓						
ST_3DMakeBox	✓						
ST_XMax	☑		✓	✓			
ST_XMin	☑		✓	✓			
ST_YMax	☑		✓	✓			
ST_YMin	☑		✓	✓			
ST_ZMax	☑		✓	✓			
ST_ZMin	☑		✓	✓			
ST_XSize	☑		✓	✓			
ST_YSize	☑		✓	✓			
ST_ZSize	☑		✓	✓			
ST_MMin	✓		✓	✓			
ST_MMax	✓		✓	✓			
ST_MSize	✓		✓	✓			
ST_LineInterpolatel	✓ t	✓	✓				
ST_3DLineInterpol	✓ oint		✓				
ST_LineInterpolatel	✓ ts	✓	✓				
ST_LineLocatePoin	✓	✓					
ST_LineSubstring	✓	✓	✓				
ST_LocateAlong	✓				✓		
ST_LocateBetween	✓				✓		
ST_LocateBetweenI	✓ ations		✓				

Funktion	geom	geog	2.5D	Kurven	SQL MM	PS	T
ST_InterpolatePoint	✓		✓				
ST_3DInterpolatePo	✓		✓				
ST_AddMeasure	✓		✓				
ST_IsValidTrajecto	✓		✓				
ST_ClosestPointOfA	✓	roach	✓				
ST_DistanceCPA	✓		✓				
ST_CPAWithin	✓		✓				
postgis.gdal_datapath							
postgis.gdal_enabled_drivers							
postgis.enable_outdb_rasters							
postgis.gdal_vsi_options							
postgis.gdal_cpl_debug							
PostGIS_AddBBox	✓			✓			
PostGIS_DropBBox	✓			✓			
PostGIS_HasBBox	✓			✓			
postgis_sfcgal_version							
postgis_sfcgal_full_version							
CG_ForceLHR							
CG_IsPlanar							
CG_IsValid							
CG_IsValidDetail							
CG_IsSimple							
CG_IsSimpleDetai							
CG_IsSolid							
CG_MakeSolid							
CG_Orientation							
CG_Area							
CG_3DArea							
CG_Volume							
CG_Intersection							
CG_Intersects							

Funktion	geom	geog	2.5D	Kurven	SQL MM	PS	T
CG_3DIntersects							
CG_Difference							
CG_3DDifference							
CG_Distance							
CG_3DDistance							
CG_3DConvexHul							
CG_3DIntersectior							
CG_Union							
CG_3DUnion							
CG_AlphaShape							
CG_ApproxConve	 ion						
CG_ApproximateM	 Axis						
CG_GenerateFlatR							
CG_GenerateHipp	 f						
CG_GenerateGabl							
CG_GenerateSkilli	 of						
CG_GenerateRoof							
CG_PolygonRepai							
CG_ConstrainedD	yTriangles						
CG_Extrude							
CG_ExtrudeStraigl	eton						
CG_GreeneApprox	exPartition						
CG_MinkowskiSu							
CG_OptimalAlpha							

Funktion	geom	geog	2.5D	Kurven	SQL MM	PS	T
CG_OptimalConve	tion						
CG_StraightSkelet							
CG_Tesselate							
CG_Triangulate							
CG_Visibility							
CG_YMonotonePa							
CG_StraightSkelet	ition						
CG_3DBuffer							
CG_Rotate							
CG_2DRotate							
CG_3DRotate							
CG_RotateX							
CG_RotateY							
CG_RotateZ							
CG_Scale							
CG_3DScale							
CG_3DScaleArour	ter						
CG_Translate							
CG_3DTranslate							
CG_Simplify							
CG_3DAlphaWrap							
CG_NurbsCurveIn	ate						
CG_NurbsCurveA	mate						
CG_NurbsCurveD	ve						

Funktion	geom	geog	2.5D	Kurven	SQL MM	PS	T
getfaceedges_returntype							
TopoGeometry							
validatetopology_returntype							
TopoElement							
TopoElementArray							
AddTopoGeometryColumn							
RenameTopoGeometryColumn							
DropTopology							
RenameTopology							
DropTopoGeometryColumn							
FixCorruptTopoGeometryColumn							
Populate_Topology_Layer							
TopologySummary							
ValidateTopology	✓						
ValidateTopologyRelation							
ValidateTopologyPr ion	✓						
MakeTopologyPreci	✓						
FindTopology	✓						
FindLayer	✓						
TotalTopologySize							
UpgradeTopology							
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	✓				✓		

Funktion	geom	geog	2.5D	Kurven	SQL MM	PS	T
ST_RemoveIsoNode					✓		
ST_RemoveIsoEdge					✓		
GetEdgeByPoint	✓						
GetFaceByPoint	✓						
GetFaceContainingPoint	✓						
GetNodeByPoint	✓						
GetTopologyID							
GetTopologySRID							
GetTopologyName							
ST_GetFaceEdges					✓		
ST_GetFaceGeometry	✓				✓		
GetRingEdges							
GetNodeEdges							
FindVertexSegmentsBelowDistance	✓						
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							

Funktion	geom	geog	2.5D	Kurven	SQL MM	PS	T
raster							
reclassarg							
summarystats							
unionarg							
AddRasterConstraints							
DropRasterConstraints							
AddOverviewConstraints							
DropOverviewConstraints							
PostGIS_GDAL_Version							
PostGIS_Raster_Lib_Build_Date							
PostGIS_Raster_Lib_Version							
ST_GDALDrivers							
UpdateRasterSRID							
ST_CreateOverview							
ST_AddBand							
ST_AsRaster	✓						
ST_AsRasterAgg	✓						
ST_Band							
ST_MakeEmptyCoverage							
ST_MakeEmptyRaster							
ST_Tile							
ST_Retile	✓						
ST_FromGDALRaster							
ST_GeoReference							
ST_Height							
ST_IsEmpty							
ST_MemSize							
ST_MetaData							
ST_NumBands							
ST_PixelHeight							
ST_PixelWidth							
ST_ScaleX							
ST_ScaleY							
ST_RasterToWorldCoord							
ST_RasterToWorldCoordX							
ST_RasterToWorldCoordY							
ST_Rotation							
ST_SkewX							
ST_SkewY							
ST_SRID							
ST_Summary							
ST_UpperLeftX							
ST_UpperLeftY							
ST_Width							
ST_WorldToRasterCoord	✓						
ST_WorldToRasterCoordX	✓						
ST_WorldToRasterCoordY	✓						
ST_BandMetaData							
ST_BandNoDataValue							
ST_BandIsNoData							
ST_BandPath							
ST_BandFileSize							

Funktion	geom	geog	2.5D	Kurven	SQL MM	PS	T
ST_BandFileTimestamp							
ST_BandPixelType							
ST_MinPossibleValue							
ST_HasNoBand							
ST_PixelAsPolygon	✓						
ST_PixelAsPolygon	✓						
ST_PixelAsPoint	✓						
ST_PixelAsPoints	✓						
ST_PixelAsCentroid	✓						
ST_PixelAsCentroid	✓						
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							

Funktion	geom	geog	2.5D	Kurven	SQL MM	PS	T
ST_AsPNG							
ST_AsTIFF							
ST_Clip	✓						
ST_ColorMap							
ST_Grayscale							
ST_Intersection	✓						
ST_MapAlgebra (callback function version)							
ST_MapAlgebra (expression version)							
ST_MapAlgebraExpr							
ST_MapAlgebraExpr							
ST_MapAlgebraFct							
ST_MapAlgebraFct							
ST_MapAlgebraFctNgb							
ST_Reclass							
ST_ReclassExact							
ST_Union							
ST_Distinct4ma							
ST_InvDistWeight4ma							
ST_Max4ma							
ST_Mean4ma							
ST_Min4ma							
ST_MinDist4ma							
ST_Range4ma							
ST_StdDev4ma							
ST_Sum4ma							
ST_Aspect							
ST_HillShade							
ST_Roughness							
ST_Slope							
ST_TPI							
ST_TRI							
ST_InterpolateRaste	✓						
ST_Contour	✓						
Box3D	✓						
ST_ConvexHull	✓						
ST_DumpAsPolygons							
ST_Envelope	✓						
ST_MinConvexHull	✓						
ST_Polygon	✓						
ST_IntersectionFrac	✓ s						
&&	✓						
&<							
&>							
=							

Funktion	geom	geog	2.5D	Kurven	SQL MM	PS	T
@	✓						
~=							
~	✓						
ST_Contains							
ST_ContainsProperly							
ST_Covers							
ST_CoveredBy							
ST_Disjoint							
ST_Intersects	✓						
ST_Overlaps							
ST_Touches							
ST_SameAlignment							
ST_NotSameAlignmentReason							
ST_Within							
ST_DWithin							
ST_DFullyWithin							

13.12 Neue, erweiterte oder geänderte PostGIS-Funktionen

13.12.1 PostGIS-Funktionen neu oder erweitert in 3.7

Die unten aufgeführten Funktionen sind PostGIS-Funktionen, die hinzugefügt oder erweitert wurden.

Neue Funktionen in PostGIS 3.7

- CG_ApproximateMedialAxis - Availability: 3.7.0 - projected parameter. Requires SFCGAL >= 2.3.0 for projected result; falls back to non-projected with a notice on older versions. Berechnet die konvexe Hülle einer Geometrie.
- CG_GenerateFlatRoof - Availability: 3.7.0 - requires SFCGAL >= 2.3.0. Generate a flat (box) roof from a footprint polygon.
- CG_GenerateGableRoof - Availability: 3.7.0 - requires SFCGAL >= 2.3.0. Generate a gable roof from a footprint polygon.
- CG_GenerateHippedRoof - Availability: 3.7.0 - requires SFCGAL >= 2.3.0. Generate a hipped roof from a footprint polygon.
- CG_GenerateRoof - Availability: 3.7.0 - requires SFCGAL >= 2.3.0. Generate a roof of the requested type from a footprint polygon.
- CG_GenerateSkillionRoof - Availability: 3.7.0 - requires SFCGAL >= 2.3.0. Generate a skillion (single-slope) roof from a footprint polygon.
- CG_NurbsCurveApproximate - Availability: 3.7.0 - requires SFCGAL >= 2.3.0. Creates an approximating NURBS curve fitting data points within a tolerance
- CG_NurbsCurveDerivative - Availability: 3.7.0 - requires SFCGAL >= 2.3.0. Computes the derivative (tangent, curvature) of a NURBS curve at a given parameter
- CG_NurbsCurveInterpolate - Availability: 3.7.0 - requires SFCGAL >= 2.3.0. Creates an interpolating NURBS curve passing through all given data points
- CG_PolygonRepair - Availability: 3.7.0 - requires SFCGAL >= 2.3.0 and CGAL >= 6.0. Repair an invalid polygon or multi-polygon.
- FindVertexSegmentPairsBelowDistance - Availability: 3.7 Find pairs of topology vertex/segment that are closer than tolerated distance
- ST_3DInterpolatePoint - Availability: 3.7.0 Returns the interpolated measure of a geometry closest to a point in 3D.

- ST_CatmullRomSmoothing - Availability: 3.7.0 Returns a smoothed version of a geometry, using the Catmull-Rom spline algorithm
- ST_ClusterRelateWin - Availability: 3.7.0 Window function that returns a cluster id for each input geometry, clustering input geometries into connected sets using the relate pattern to determine whether the geometries are connected.
- ST_ControlPoints - Availability: 3.7.0 Returns the control points of a NURBS curve as a MULTIPOINT geometry.
- ST_CoverageEdges - Availability: 3.7.0 Computes the unique edges of a polygonal coverage.
- ST_Degree - Availability: 3.7.0 Returns the polynomial degree of a NURBS curve.
- ST_Evaluate - Availability: 3.7.0 Evaluates a NURBS curve at a specific parameter value and returns the resulting point.
- ST_IsNurbsCurve - Availability: 3.7.0 Checks if a geometry is a NURBS curve.
- ST_IsSimpleDetail - Availability: 3.7.0 Report locations of self-intersection or self-tangency, if any.
- ST_Knots - Availability: 3.7.0 Returns the knot vector of a NURBS curve.
- ST_MSize - Availability: 3.7.0 Returns the M size of a geometry.
- ST_MakeLine - Availability: 3.7.0 - Support for MultiLineString input elements was introduced Erzeugt einen Linienzug aus einer Punkt-, Mehrfachpunkt- oder Liniengeometrie.
- ST_MakeNurbsCurve - Availability: 3.7.0 Creates a NURBS (Non-Uniform Rational B-Spline) curve from control points, optional weights, and optional knot vector.
- ST_MinimumSpanningTree - Availability: 3.7.0 Window function that returns a tree id for each input geometry, clustering input geometries into connected trees using the Minimum Spanning Tree algorithm.
- ST_NumControlPoints - Availability: 3.7.0 Returns the number of control points in a NURBS curve.
- ST_NurbsCurveIsRational - Availability: 3.7.0 Checks if a NURBS curve is rational (has weights).
- ST_NurbsToLineString - Availability: 3.7.0 Converts a NURBS curve to a LINESTRING by uniform sampling.
- ST_Weights - Availability: 3.7.0 Returns the weight array of a rational NURBS curve.
- ST_XSize - Availability: 3.7.0 Returns the X size of a 2D or 3D bounding box or a geometry.
- ST_YSize - Availability: 3.7.0 Returns the Y size of a 2D or 3D bounding box or a geometry.
- ST_ZSize - Availability: 3.7.0 Returns the Z size of a 2D or 3D bounding box or a geometry.

Erweiterte Funktionen in PostGIS 3.7

- ST_AsX3D - Enhanced: 3.7.0: Support for curve geometries. NURBSCURVE is emitted as X3D NurbsCurve; other curve types are linearized before output. Gibt eine Geometrie im X3D XML Knotenelement-Format zurück: ISO-IEC-19776-1.2-X3DEncodings-XML
- ST_GMLToSQL - Enhanced: 3.7.0 support for GML ArcString, curved Ring, and CurvePolygon geometries was introduced. Returns a geometry from GML. Alias for ST_GeomFromGML.
- ST_GeomFromGML - Enhanced: 3.7.0 support for GML ArcString, curved Ring, and CurvePolygon geometries was introduced. Nimmt als Eingabe eine GML-Darstellung der Geometrie und gibt ein geometrisches PostGIS-Objekt aus.
- ST_LongestLine - Enhanced: 3.7.0 - support for a single geometry input. Gibt die größte 3-dimensionale Distanz zwischen zwei geometrischen Objekten als Linie zurück
- ST_MakePolygon - Enhanced: 3.7.0 - Support for curved input rings was introduced. Creates a Polygon or CurvePolygon from a shell and optional list of holes.
- ST_MaxDistance - Enhanced: 3.7.0 - support for a single geometry input. Gibt die größte 2-dimensionale Distanz zwischen zwei geometrischen Objekten in projizierten Einheiten zurück.

- ST_Value - Enhanced: 3.7.0 resample accepts nearest-neighbor boundary options "nearest-UL", "nearest-UR", "nearest-LL", and "nearest-LR". Gibt den Zellwert eines Pixels aus, das über columnx und rowy oder durch einen bestimmten geometrischen Punkt angegeben wird. Die Bandnummern beginnen mit 1 und wenn keine Bandnummer angegeben ist, dann wird Band 1 angenommen. Wenn exclude_nodata_value auf FALSE gesetzt ist, werden auch die Pixel mit einem nodata Wert mit einbezogen. Wenn exclude_nodata_value nicht übergeben wird, dann wird er über die Metadaten des Rasters ausgelesen.
- TopoGeo_AddLineString - Enhanced: 3.7.0 added support for limiting the number of new edges created in the topology. Fügt einen Linienzug unter Berücksichtigung eines Toleranzwerts zu einer Topologie hinzu , wobei Kanten/Maschen dabei aufgetrennt werden können.

Geänderte Funktionen in PostGIS 3.7

- ST_AsGeoJSON - Changed: 3.7.0 added warning about duplicate keys Rückgabe einer Geometrie oder eines Merkmals im GeoJSON-Format.
- toTopoGeom - Changed: 3.7.0 changes meaning of tolerance 0 to really mean 0 and -1 (new default) to request using the topology precision. Wandelt eine einfache Geometrie in eine TopoGeometry um.

13.12.2 PostGIS-Funktionen neu oder erweitert in 3.6

Die unten aufgeführten Funktionen sind PostGIS-Funktionen, die hinzugefügt oder erweitert wurden.

Neue Funktionen in PostGIS 3.6

- CG_2DRotate - Availability: 3.6.0 - requires SFCGAL >= 2.0.0 Rotates a geometry by a given angle around a specified point in 2D.
- CG_3DAlphaWrapping - Availability: 3.6.0 - requires SFCGAL >= 2.1.0 Computes a 3D Alpha-wrapping strictly enclosing a geometry.
- CG_3DBuffer - Availability: 3.6.0 - requires SFCGAL >= 2.0.0 Computes a 3D buffer around a geometry.
- CG_3DRotate - Availability: 3.6.0 - requires SFCGAL >= 2.0.0 Rotates a geometry in 3D space around an axis vector.
- CG_3DScale - Availability: 3.6.0 - requires SFCGAL >= 2.0.0 Scales a geometry by separate factors along X, Y, and Z axes.
- CG_3DScaleAroundCenter - Availability: 3.6.0 - requires SFCGAL >= 2.0.0 Scales a geometry in 3D space around a specified center point.
- CG_3DTranslate - Availability: 3.6.0 - requires SFCGAL >= 2.0.0 Translates (moves) a geometry by given offsets in 3D space.
- CG_Rotate - Availability: 3.6.0 - requires SFCGAL >= 2.0.0 Rotates a geometry by a given angle around the origin (0,0).
- CG_RotateX - Availability: 3.6.0 - requires SFCGAL >= 2.0.0 Rotates a geometry around the X-axis by a given angle.
- CG_RotateY - Availability: 3.6.0 - requires SFCGAL >= 2.0.0 Rotates a geometry around the Y-axis by a given angle.
- CG_RotateZ - Availability: 3.6.0 - requires SFCGAL >= 2.0.0 Rotates a geometry around the Z-axis by a given angle.
- CG_Scale - Availability: 3.6.0 - requires SFCGAL >= 2.0.0 Scales a geometry uniformly in all dimensions by a given factor.
- CG_Simplify - Availability: 3.6.0 - requires SFCGAL >= 2.1.0 Reduces the complexity of a geometry while preserving essential features and Z/M values.
- CG_StraightSkeletonPartition - Availability: 3.6.0 - requires SFCGAL >= 2.0.0. Computes the straight skeleton partition of a polygon.
- CG_Translate - Availability: 3.6.0 - requires SFCGAL >= 2.0.0 Translates (moves) a geometry by given offsets in 2D space.
- FixCorruptTopoGeometryColumn - Availability: 3.6.1 Fixes topogeometry corruption caused by upgrade to postgis_topology 3.6.0 and higher
- MakeTopologyPrecise - Verfügbarkeit: 3.6.0 Snap topology vertices to precision grid.

- ST_AsRasterAgg - Verfügbarkeit: 3.6.0 Aggregiert. Wandelt PostGIS-Geometrien in einen neuen Raster um.
- ST_CoverageClean - Availability: 3.6.0 - requires GEOS >= 3.14.0 Computes a clean (edge matched, non-overlapping, gap-cleared) polygonal coverage, given a non-clean input.
- ST_IntersectionFractions - Availability: 3.6.0 Requires GEOS 3.14 or higher. Calculates the fraction of each raster cell that is covered by a given geometry.
- ST_ReclassExact - Verfügbarkeit: 3.6.0 Creates a new raster composed of bands reclassified from original, using a 1:1 mapping from values in the original band to new values in the destination band.
- TotalTopologySize - Verfügbarkeit: 3.6.0 Total disk space used by the specified topology, including all indexes and TOAST data.
- UpgradeTopology - Verfügbarkeit: 3.6.0 Upgrades the specified topology to support large ids (int8) for topology and primitive ids.
- ValidateTopologyPrecision - Verfügbarkeit: 3.6.0 Returns non-precise vertices in the topology.
- postgis.gdal_cpl_debug - Verfügbarkeit: 3.6.0 A boolean configuration to turn logging of GDAL debug messages on and off.

Erweiterte Funktionen in PostGIS 3.6

- CreateTopology - Enhanced: 3.6.0 added topoid and usesLargeIDs parameters. Erstellt ein neues Topologie-Schema und trägt es in die Tabelle topology.topology ein.

Geänderte Funktionen in PostGIS 3.6

- ST_GeometryN - Changed: 3.6.0 TIN and PolyhedralSurface are treated as unitary geometries and are not decomposed into patches; use ST_PatchN to access faces. Gibt den Geometrietyp des ST_Geometry Wertes zurück.
- ST_NumGeometries - Changed: 3.6.0 TIN and PolyhedralSurface are treated as unitary geometries and return 1; use ST_NumPatches to count faces. Gibt die Anzahl der Punkte einer Geometrie zurück. Funktioniert für alle Geometrien.

13.12.3 PostGIS-Funktionen neu oder erweitert in 3.5

Die unten aufgeführten Funktionen sind PostGIS-Funktionen, die hinzugefügt oder erweitert wurden.

Neue Funktionen in PostGIS 3.5

- CG_3DArea - Verfügbarkeit: 3.5.0 Berechnet die Fläche von 3D-Oberflächengeometrien. Gibt 0 für Solids zurück.
- CG_3DConvexHull - Verfügbarkeit: 3.5.0 Berechnet die konvexe Hülle einer Geometrie.
- CG_3DDifference - Verfügbarkeit: 3.5.0 3D-Differenz durchführen
- CG_3DDistance - Verfügbarkeit: 3.5.0 Computes the minimum 3D distance between two geometries
- CG_3DIntersection - Verfügbarkeit: 3.5.0 3D-Schnitte durchführen
- CG_3DIntersects - Verfügbarkeit: 3.5.0 Tests if two 3D geometries intersect
- CG_3DUnion - Verfügbarkeit: 3.5.0 Perform 3D union using postgis_sfcgal.
- CG_AlphaShape - Availability: 3.5.0 - requires SFCGAL >= 1.4.1. Berechnet eine Alpha-Form, die eine Geometrie umschließt
- CG_ApproxConvexPartition - Verfügbarkeit: 3.5.0 - erfordert SFCGAL >= 1.5.0. Berechnet die approximale konvexe Partition der Polygongeometrie
- CG_ApproximateMedialAxis - Verfügbarkeit: 3.5.0 Berechnet die konvexe Hülle einer Geometrie.
- CG_Area - Verfügbarkeit: 3.5.0 Calculates the area of a geometry

- CG_Difference - Verfügbarkeit: 3.5.0 Computes the geometric difference between two geometries
- CG_Distance - Verfügbarkeit: 3.5.0 Computes the minimum distance between two geometries
- CG_Extrude - Verfügbarkeit: 3.5.0 Extrudieren einer Oberfläche in ein zugehöriges Volumen
- CG_ExtrudeStraightSkeleton - Verfügbarkeit: 3.5.0 - erfordert SFCGAL >= 1.5.0. Gerade Skelett-Extrusion
- CG_ForceLHR - Verfügbarkeit: 3.5.0 LHR-Ausrichtung erzwingen
- CG_GreeneApproxConvexPartition - Verfügbarkeit: 3.5.0 - erfordert SFCGAL >= 1.5.0. Berechnet die approximale konvexe Partition der Polygongeometrie
- CG_Intersection - Verfügbarkeit: 3.5.0 Computes the intersection of two geometries
- CG_Intersects - Verfügbarkeit: 3.5.0 Prüft, ob sich zwei Geometrien schneiden (sie haben mindestens einen Punkt gemeinsam)
- CG_IsPlanar - Verfügbarkeit: 3.5.0 Prüfen, ob eine Fläche planar ist oder nicht
- CG_IsSolid - Verfügbarkeit: 3.5.0 Prüfen, ob die Geometrie ein Solid ist. Es wird keine Gültigkeitsprüfung durchgeführt.
- CG_MakeSolid - Verfügbarkeit: 3.5.0 Gießen Sie die Geometrie in einen Körper. Es wird keine Prüfung durchgeführt. Um ein gültiges Solid zu erhalten, muss die Eingabegeometrie eine geschlossene polyedrische Fläche oder ein geschlossenes TIN sein.
- CG_MinkowskiSum - Verfügbarkeit: 3.5.0 Führt die Minkowski-Summe aus
- CG_OptimalAlphaShape - Availability: 3.5.0 - requires SFCGAL >= 1.4.1. Berechnet eine Alpha-Form, die eine Geometrie umschließt, unter Verwendung eines "optimalen" Alpha-Wertes.
- CG_OptimalConvexPartition - Verfügbarkeit: 3.5.0 - erfordert SFCGAL >= 1.5.0. Berechnet eine optimale konvexe Partition der Polygongeometrie
- CG_Orientation - Verfügbarkeit: 3.5.0 Bestimmung der Oberflächenausrichtung
- CG_StraightSkeleton - Verfügbarkeit: 3.5.0 Berechnet die konvexe Hülle einer Geometrie.
- CG_Tesselate - Verfügbarkeit: 3.5.0 Führt eine Oberflächentesselierung eines Polygons oder einer Polyederfläche durch und gibt diese als TIN oder Sammlung von TINS zurück
- CG_Triangulate - Verfügbarkeit: 3.5.0 Triangulates a polygonal geometry
- CG_Union - Verfügbarkeit: 3.5.0 Computes the union of two geometries
- CG_Visibility - Verfügbarkeit: 3.5.0 - erfordert SFCGAL >= 1.5.0. Berechnen eines Sichtbarkeitspolygons aus einem Punkt oder einem Segment in einer Polygongeometrie
- CG_Volume - Verfügbarkeit: 3.5.0 Computes the volume of a 3D geometry. Closed surfaces can have non-zero volume.
- CG_YMonotonePartition - Verfügbarkeit: 3.5.0 - erfordert SFCGAL >= 1.5.0. Berechnet die y-monotone Partition der Polygongeometrie
- ST_HasM - Verfügbarkeit: 3.5.0 Prüft, ob eine Geometrie eine M-Dimension (Maß) hat.
- ST_HasZ - Verfügbarkeit: 3.5.0 Prüft, ob eine Geometrie eine Z-Dimension hat.
- ST_RemoveIrrelevantPointsForView - Verfügbarkeit: 3.5.0 Removes points that are irrelevant for rendering a specific rectangular view of a geometry.
- ST_RemoveSmallParts - Verfügbarkeit: 3.5.0 Entfernt kleine Teile (Polygonringe oder Linienzüge) einer Geometrie.
- TopoGeo_LoadGeometry - Verfügbarkeit: 3.5.0 Laden Sie eine Geometrie in eine bestehende Topologie und fangen und teilen Sie sie nach Bedarf.

Erweiterte Funktionen in PostGIS 3.5

- ST_Clip - Verbessert: 3.5.0 - berührtes Argument hinzugefügt. Gibt den nach der Eingabegeometrie zugeschnittenen Raster zurück. Falls die Bandnummer nicht angegeben ist, werden alle Bänder verarbeitet. Wenn crop nicht angegeben oder auf TRUE gesetzt ist, wird der Ausgaberaster entsprechend zugeschnitten. Wenn touched auf TRUE gesetzt ist, werden auch berührte Pixel einbezogen, andernfalls nur Pixel, bei denen sich der Mittelpunkt im Inneren der Geometrie befindet.

Geänderte Funktionen in PostGIS 3.5

- ST_AsGeoJSON - Geändert: 3.5.0 erlaubt die Angabe der Spalte, die die Feature-ID enthält Rückgabe einer Geometrie oder eines Merkmals im 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

13.12.4 PostGIS-Funktionen neu oder erweitert in 3.4

Die unten aufgeführten Funktionen sind PostGIS-Funktionen, die hinzugefügt oder erweitert wurden.

Neue Funktionen in PostGIS 3.4

- PostGIS_GEOS_Compiled_Version - Verfügbarkeit: 3.4.0 Gibt die Versionsnummer der GEOS-Bibliothek zurück, mit der PostGIS erstellt wurde.
- PostGIS_PROJ_Compiled_Version - Verfügbarkeit: 3.5.0 Returns the version number of the PROJ library against which PostGIS was built.
- RenameTopoGeometryColumn - Verfügbarkeit: 3.4.0 Benennt eine topogeometrische Spalte um
- RenameTopology - Verfügbarkeit: 3.4.0 Benennt eine Topologie um
- ST_ClusterIntersectingWin - Verfügbarkeit: 3.4.0 Fensterfunktion, die für jede Eingabegeometrie eine Cluster-ID zurückgibt und die Eingabegeometrien in zusammenhängende Gruppen clustert.
- ST_ClusterWithinWin - Verfügbarkeit: 3.4.0 Fensterfunktion, die eine Cluster-ID für jede Eingabegeometrie zurückgibt, Clustering anhand des Trennungsabstands.
- ST_CoverageInvalidEdges - Verfügbarkeit: 3.4.0 Fensterfunktion, die Stellen findet, an denen die Polygone keine gültige Abdeckung bilden.
- ST_CoverageSimplify - Verfügbarkeit: 3.4.0 Fensterfunktion, die die Kanten einer polygonalen Abdeckung vereinfacht.
- ST_CoverageUnion - Verfügbarkeit: 3.4.0 - erfordert GEOS >= 3.8.0 Berechnet die Vereinigung einer Menge von Polygonen, die eine Abdeckung bilden, indem gemeinsame Kanten entfernt werden.
- ST_InverseTransformPipeline - Verfügbarkeit: 3.4.0 Rückgabe einer neuen Geometrie mit in ein anderes räumliches Bezugssystem transformierten Koordinaten unter Verwendung der Umkehrung einer definierten Koordinatentransformationspipeline.
- ST_LargestEmptyCircle - Verfügbarkeit: 3.4.0. Berechnet den größten Kreis, der eine Geometrie nicht überschneidet.
- ST_LineExtend - Verfügbarkeit: 3.4.0 Gibt eine Linie zurück, die um die angegebenen Abstände vorwärts und rückwärts verlängert wurde.
- ST_TransformPipeline - Verfügbarkeit: 3.4.0 Rückgabe einer neuen Geometrie mit in ein anderes räumliches Bezugssystem transformierten Koordinaten unter Verwendung einer definierten Koordinatentransformationspipeline.
- TopoElement - Verfügbarkeit: 3.4.0 Konvertiert eine Topogeometrie in ein Topoelement.
- postgis_srs - Verfügbarkeit: 3.4.0 Rückgabe eines Metadatensatzes für die angefragte Behörde und srid.
- postgis_srs_all - Verfügbarkeit: 3.4.0 Gibt Metadatensätze für jedes räumliche Bezugssystem in der zugrunde liegenden Proj-Datenbank zurück.

- postgis_srs_codes - Verfügbarkeit: 3.4.0 Gibt die Liste der SRS-Codes zurück, die mit der angegebenen Behörde verbunden sind.
- postgis_srs_search - Verfügbarkeit: 3.4.0 Gibt Metadatensätze für projizierte Koordinatensysteme zurück, die Nutzungsbereiche haben, die den Parameter bounds vollständig enthalten.

Erweiterte Funktionen in PostGIS 3.4

- PostGIS_Full_Version - Verbessert: 3.4.0 enthält jetzt zusätzliche PROJ-Konfigurationen NETWORK_ENABLED, URL_ENDPOINT und DATABASE_PATH des proj.db-Speicherorts Meldet die vollständige PostGIS-Version und Informationen zur Build-Konfiguration.
- PostGIS_PROJ_Version - Verbessert: 3.4.0 enthält jetzt NETWORK_ENABLED, URL_ENDPOINT und DATABASE_PATH des proj.db-Speicherorts Gibt die Versionsnummer der PROJ4-Bibliothek zurück.
- ST_AsSVG - Verbessert: 3.4.0 zur Unterstützung aller Kurventypen Gibt eine Geometrie als SVG-Pfad aus.
- ST_ClosestPoint - Verbessert: 3.4.0 - Unterstützung für Geographie. Gibt den 2D-Punkt auf g1 zurück, der g2 am nächsten ist. Dies ist der erste Punkt der kürzesten Linie von einer Geometrie zur anderen.
- ST_LineSubstring - Verbessert: 3.4.0 - Unterstützung für Geographie wurde eingeführt. Gibt den Teil einer Linie zwischen zwei gebrochenen Stellen zurück.
- ST_Project - Verbessert: 3.4.0 Erlaubt Geometrieargumente und Zweipunktform ohne Azimut. Gibt einen Punkt zurück, der von einem Startpunkt um eine bestimmte Entfernung und Peilung (Azimut) projiziert wird.
- ST_Resample - Verbessert: 3.4.0 Max und Min Resampling Optionen hinzugefügt Skaliert einen Raster mit einem bestimmten Algorithmus, neuen Dimensionen, einer beliebigen Gitterecke und über Parameter zur Georeferenzierung des Rasters, die angegeben oder von einem anderen Raster übernommen werden können.
- ST_Rescale - Verbessert: 3.4.0 Max und Min Resampling Optionen hinzugefügt Neuabtastung eines Rasters, indem nur die Skala (oder Pixelgröße) angepasst wird. Die neuen Pixelwerte werden mit den Algorithmen NearestNeighbor (englische oder amerikanische Schreibweise), Bilinear, Cubic, CubicSpline, Lanczos, Max oder Min resampling berechnet. Die Voreinstellung ist NearestNeighbor.
- ST_ShortestLine - Verbessert: 3.4.0 - Unterstützung für Geographie. Gibt die 2-dimenionale kürzeste Strecke zwischen zwei Geometrien als Linie zurück

Geänderte Funktionen in PostGIS 3.4

- PostGIS_Extensions_Upgrade - Geändert: 3.4.0 um das Argument target_version hinzuzufügen. Packt und aktualisiert PostGIS-Erweiterungen (z.B. postgis_raster, postgis_topology, postgis_sfcgal) auf die angegebene oder neueste Version.

13.12.5 PostGIS-Funktionen neu oder erweitert in 3.3

Die unten aufgeführten Funktionen sind PostGIS-Funktionen, die hinzugefügt oder erweitert wurden.

Neue Funktionen in PostGIS 3.3

- RemoveUnusedPrimitives - Verfügbarkeit: 3.3.0 Entfernt Topologieprimitive, die zur Definition bestehender TopoGeometry-Objekte nicht benötigt werden.
- ST_AsMARC21 - Verfügbarkeit: 3.3.0 Gibt die Geometrie als MARC21/XML-Datensatz mit einem geografischen Datenfeld (034) zurück.
- ST_GeomFromMARC21 - Verfügbarkeit: 3.3.0, erfordert libxml2 2.6+ Nimmt MARC21/XML-Geodaten als Eingabe und gibt ein PostGIS-Geometrieobjekt zurück.
- ST_Letters - Verfügbarkeit: 3.3.0 Gibt die eingegebenen Buchstaben als Geometrie mit einer Standardstartposition am Ursprung und einer Standardtexthöhe von 100 zurück.

- ST_SimplifyPolygonHull - Verfügbarkeit: 3.3.0. Berechnet eine vereinfachte topologieerhaltende äußere oder innere Hülle einer polygonalen Geometrie.
- ST_TriangulatePolygon - Verfügbarkeit: 3.3.0. Berechnet die eingeschränkte Delaunay-Triangulation von Polygonen
- postgis_sfcgal_full_version - Verfügbarkeit: 3.3.0 Liefert die vollständige Version von SFCGAL, einschließlich der CGAL- und Boost-Versionen

Erweiterte Funktionen in PostGIS 3.3

- ST_ConcaveHull - Verbessert: 3.3.0, native GEOS-Implementierung aktiviert für GEOS 3.11+ Berechnet eine möglicherweise konkave Geometrie, die alle Eckpunkte der Eingabegeometrie enthält
- ST_LineMerge - Verbessert: 3.3.0 akzeptiert einen gerichteten Parameter. Gibt die Linien zurück, die durch das Zusammenfügen eines MultiLineString gebildet werden.

Geänderte Funktionen in PostGIS 3.3

- PostGIS_Extensions_Upgrade - Geändert: 3.3.0 Unterstützung für Upgrades von jeder PostGIS-Version. Funktioniert nicht auf allen Systemen. Packt und aktualisiert PostGIS-Erweiterungen (z.B. postgis_raster, postgis_topology, postgis_sfcgal) auf die angegebene oder neueste Version.

13.12.6 PostGIS-Funktionen neu oder erweitert in 3.2

Die unten aufgeführten Funktionen sind PostGIS-Funktionen, die hinzugefügt oder erweitert wurden.

Neue Funktionen in PostGIS 3.2

- FindLayer - Verfügbarkeit: 3.2.0 Gibt einen topology.layer-Datensatz mit anderen Mitteln zurück.
- FindTopology - Verfügbarkeit: 3.2.0 Gibt einen Topologie-Datensatz mit anderen Mitteln zurück.
- GetFaceContainingPoint - Verfügbarkeit: 3.2.0 Findet die Fläche, die einen Punkt enthält.
- ST_AsFlatGeobuf - Verfügbarkeit: 3.2.0 Rückgabe einer FlatGeobuf-Darstellung einer Reihe von Zeilen.
- ST_Contour - Verfügbarkeit: 3.2.0 Erzeugt einen Satz von Vektorkonturen aus dem angegebenen Rasterband unter Verwendung des GDAL-Konturierungsalgorithmus.
- ST_DumpSegments - Verfügbarkeit: 3.2.0 Gibt eine Zusammenfassung des Inhalts einer Geometrie wieder.
- ST_FromFlatGeobuf - Verfügbarkeit: 3.2.0 Liest FlatGeobuf-Daten.
- ST_FromFlatGeobufToTable - Verfügbarkeit: 3.2.0 Erstellt eine Tabelle auf der Grundlage der Struktur der FlatGeobuf-Daten.
- ST_InterpolateRaster - Verfügbarkeit: 3.2.0 Interpoliert eine gerasterte Oberfläche auf der Grundlage eines Eingabesatzes von 3D-Punkten, wobei die X- und Y-Werte zur Positionierung der Punkte auf dem Gitter und der Z-Wert der Punkte als Oberflächenhöhe verwendet werden.
- ST_SRID - Verfügbarkeit: 3.2.0 Gibt den räumlichen Referenzbezeichner für eine Topogeometrie zurück.
- ST_Scroll - Verfügbarkeit: 3.2.0 Startpunkt eines geschlossenen LineStrings ändern.
- ST_SetM - Verfügbarkeit: 3.2.0 Gibt eine Geometrie mit denselben X/Y-Koordinaten wie die Eingabegeometrie zurück, wobei die Werte aus dem Raster mit dem gewünschten Resample-Algorithmus in die Dimension M kopiert werden.
- ST_SetZ - Verfügbarkeit: 3.2.0 Gibt eine Geometrie mit denselben X/Y-Koordinaten wie die Eingabegeometrie zurück, wobei die Werte aus dem Raster mit dem gewünschten Resample-Algorithmus in die Z-Dimension kopiert werden.
- TopoGeom_addTopoGeom - Verfügbarkeit: 3.2 Fügt Element einer TopoGeometry zur Definition einer anderen TopoGeometry hinzu.

- ValidateTopologyRelation - Verfügbarkeit: 3.2.0 Gibt Informationen über ungültige Topologiebeziehungssätze zurück
- postgis.gdal_vsi_options - Verfügbarkeit: 3.2.0 Eine boolesche Konfigurationsmöglichkeit um den Zugriff auf out-db Rasterbänder zu ermöglichen

Erweiterte Funktionen in PostGIS 3.2

- GetFaceByPoint - Verbessert: 3.2.0 effizientere Implementierung und klarerer Vertrag, funktioniert nicht mehr mit ungültigen Topologien. Findet eine Fläche, die einen bestimmten Punkt schneidet.
- ST_ClusterKMeans - Verbessert: 3.2.0 Unterstützung für max_radius Fensterfunktion, die eine Cluster-ID für jede Eingabegeometrie unter Verwendung des K-Means-Algorithmus zurückgibt.
- ST_MakeValid - Verbessert: 3.2.0, zusätzliche Algorithmus-Optionen, 'Linienwerk' und 'Struktur', die GEOS >= 3.10.0 erfordern. Versucht, eine ungültige Geometrie gültig zu machen, ohne dass Scheitelpunkte verloren gehen.
- ST_PixelAsCentroid - Verbessert: 3.2.0 Schneller jetzt in C implementiert. Gibt den geometrischen Schwerpunkt (Punktgeometrie) der Fläche aus, die durch das Pixel repräsentiert wird.
- ST_PixelAsCentroids - Verbessert: 3.2.0 Schneller jetzt in C implementiert. Gibt den geometrischen Schwerpunkt (Punktgeometrie) für jedes Pixel des Rasterbandes zurück, zusammen mit dem Zellwert und den X- und Y-Rasterkoordinaten eines jeden Pixels. Die Koordinaten der Punkte entsprechen dem geometrischen Schwerpunkt der Pixel.
- ST_Point - Verbessert: 3.2.0 srid wurde als zusätzliches optionales Argument hinzugefügt. Ältere Installationen erfordern die Kombination mit ST_SetSRID, um das Raster auf der Geometrie zu markieren. Erzeugt einen Punkt mit X-, Y- und SRID-Werten.
- ST_PointM - Verbessert: 3.2.0 srid wurde als zusätzliches optionales Argument hinzugefügt. Ältere Installationen erfordern die Kombination mit ST_SetSRID, um das Raster auf der Geometrie zu markieren. Erzeugt einen Punkt mit den Werten X, Y, M und SRID.
- ST_PointZ - Verbessert: 3.2.0 srid wurde als zusätzliches optionales Argument hinzugefügt. Ältere Installationen erfordern die Kombination mit ST_SetSRID, um das Raster auf der Geometrie zu markieren. Erzeugt einen Punkt mit X-, Y-, Z- und SRID-Werten.
- ST_PointZM - Verbessert: 3.2.0 srid wurde als zusätzliches optionales Argument hinzugefügt. Ältere Installationen erfordern die Kombination mit ST_SetSRID, um das Raster auf der Geometrie zu markieren. Erzeugt einen Punkt mit den Werten X, Y, Z, M und SRID.
- ST_RemovePoint - Verbessert: 3.2.0 Einen Punkt aus einem Linienzug entfernen.
- ST_RemoveRepeatedPoints - Verbessert: 3.2.0 Gibt eine Version einer Geometrie zurück, bei der doppelte Punkte entfernt wurden.
- ST_StartPoint - Verbessert: 3.2.0 gibt einen Punkt für alle Geometrien zurück. Vorheriges Verhalten gibt NULLs zurück, wenn die Eingabe kein LineString war. Returns the first point of a LineString, CircularLineString, or NURBSCurve.
- ST_Value - Verbessert: 3.2.0 Das optionale Argument resample wurde hinzugefügt. Gibt den Zellwert eines Pixels aus, das über columnx und rowy oder durch einen bestimmten geometrischen Punkt angegeben wird. Die Bandnummern beginnen mit 1 und wenn keine Bandnummer angegeben ist, dann wird Band 1 angenommen. Wenn exclude_nodata_value auf FALSE gesetzt ist, werden auch die Pixel mit einem nodata Wert mit einbezogen. Wenn exclude_nodata_value nicht übergeben wird, dann wird er über die Metadaten des Rasters ausgelesen.
- TopoGeo_AddLineString - Erweitert: In Version 3.2.0 wurde Unterstützung für die Rückgabe signierter Identifikatoren hinzugefügt. Fügt einen Linienzug unter Berücksichtigung eines Toleranzwerts zu einer Topologie hinzu , wobei Kanten/Maschen dabei aufgetrennt werden können.

Geänderte Funktionen in PostGIS 3.2

- ST_Boundary - Geändert: 3.2.0 Unterstützung für TIN, verwendet keine Geos, linearisiert keine Kurven Gibt die abgeschlossene Hülle aus der kombinierten Begrenzung der Geometrie zurück.
- ValidateTopology - Geändert: 3.2.0 fügte den optionalen bbox-Parameter hinzu und führte Prüfungen der Flächenbeschriftung und der Kantenverknüpfung durch. Liefert eine Menge validatetopology_returntype Objekte, die Probleme mit der Topologie beschreiben.

13.12.7 PostGIS-Funktionen neu oder erweitert in 3.1

Die unten aufgeführten Funktionen sind PostGIS-Funktionen, die hinzugefügt oder erweitert wurden.

Neue Funktionen in PostGIS 3.1

- ST_Hexagon - Verfügbarkeit: 2.1.0 Liefert ein einzelnes Sechseck unter Verwendung der angegebenen Kantengröße und Zellkoordinate innerhalb des Sechseck-Gitterraums.
- ST_HexagonGrid - Verfügbarkeit: 2.1.0 Gibt eine Menge von Sechsecken und Zellindizes zurück, die die Grenzen des Arguments Geometrie vollständig abdecken.
- ST_MaximumInscribedCircle - Verfügbarkeit: 3.1.0. Berechnet die konvexe Hülle einer Geometrie.
- ST_ReducePrecision - Verfügbarkeit: 3.1.0. Gibt eine gültige Geometrie mit auf eine Rastertoleranz gerundeten Punkten zurück.
- ST_Square - Verfügbarkeit: 2.1.0 Gibt ein einzelnes Quadrat mit der angegebenen Kantengröße und Zellkoordinate innerhalb des quadratischen Gitterraums zurück.
- ST_SquareGrid - Verfügbarkeit: 2.1.0 Gibt eine Menge von Gitterquadraten und Zellindizes zurück, die die Grenzen des Arguments Geometrie vollständig abdecken.

Erweiterte Funktionen in PostGIS 3.1

- ST_AsEWKT - Verbessert: 3.1.0 Unterstützung für optionale Präzisionsparameter. Gibt die Well-known-Text(WKT) Darstellung der Geometrie mit den SRID-Metadaten zurück.
- ST_ClusterKMeans - Verbessert: 3.1.0 Unterstützung für 3D-Geometrien und Gewichte Fensterfunktion, die eine Cluster-ID für jede Eingabegeometrie unter Verwendung des K-Means-Algorithmus zurückgibt.
- ST_Difference - Verbessert: 3.1.0 akzeptiert einen gridSize-Parameter. Berechnet eine Geometrie, die den Teil der Geometrie A darstellt, der die Geometrie B nicht schneidet.
- ST_Intersection - Verbessert: 3.1.0 akzeptiert einen gridSize Parameter Berechnet eine Geometrie, die den gemeinsamen Teil der Geometrien A und B darstellt.
- ST_MakeValid - Verbessert: 3.1.0, Entfernen von Koordinaten mit NaN-Werten hinzugefügt. Versucht, eine ungültige Geometrie gültig zu machen, ohne dass Scheitelpunkte verloren gehen.
- ST_Subdivide - Verbessert: 3.1.0 akzeptiert einen gridSize-Parameter. Berechnet eine geradlinige Unterteilung einer Geometrie.
- ST_SymDifference - Verbessert: 3.1.0 akzeptiert einen gridSize-Parameter. Berechnet eine Geometrie, die die Teile der Geometrien A und B darstellt, die sich nicht überschneiden.
- ST_TileEnvelope - Erweiterung: 2.0.0 Standardwert für den optionalen Parameter SRID eingefügt. Erzeugt ein rechteckiges Polygon in Web Mercator (SRID:3857) unter Verwendung des XYZ-Kachelsystems.
- ST_UnaryUnion - Verbessert: 3.1.0 akzeptiert einen gridSize-Parameter. Berechnet die Vereinigung der Komponenten einer einzelnen Geometrie.
- ST_Union - Verbessert: 3.1.0 akzeptiert einen gridSize-Parameter. Berechnet eine Geometrie, die die Punktmengenvereinigung der Eingabegeometrien darstellt.

Geänderte Funktionen in PostGIS 3.1

- ST_Count - Geändert: 3.1.0 - Die ST_Count(rastertable, rastercolumn, ...) Varianten wurden entfernt. Verwenden Sie stattdessen . Gibt die Anzahl der Pixel für ein Band eines Rasters oder eines Raster-Coverage zurück. Wenn kein Band angegeben ist, wird standardmäßig Band 1 gewählt. Wenn der Parameter "exclude_nodata_value" auf TRUE gesetzt ist, werden nur Pixel mit Werten ungleich NODATA gezählt.

- ST_Force3D - Geändert: 3.1.0. Unterstützung für die Angabe eines Z-Wertes ungleich Null wurde zugefügt. Zwingt die Geometrien in einen XYZ Modus. Dies ist ein Alias für ST_Force3DZ.
- ST_Force3DM - Geändert: 3.1.0. Unterstützung für die Angabe eines M-Wertes ungleich Null wurde hinzugefügt. Zwingt die Geometrien in einen XYM Modus.
- ST_Force3DZ - Geändert: 3.1.0. Unterstützung für die Angabe eines Z-Wertes ungleich Null wurde zugefügt. Zwingt die Geometrien in einen XYZ Modus.
- ST_Force4D - Geändert: 3.1.0. Unterstützung für die Angabe von Z- und M-Werten ungleich Null wurde hinzugefügt. Zwingt die Geometrien in einen XYZM Modus.
- ST_Histogram - Geändert: 3.1.0 Die Variante ST_Histogram(table_name, column_name) wurde entfernt. Gibt Datensätze aus, welche die Verteilung der Daten eines Rasters oder eines Rastercoverage darstellen. Dabei wird die Wertemenge in Klassen aufgeteilt und für jede Klasse zusammengefasst. Wenn die Anzahl der Klassen nicht angegeben ist, wird sie automatisch berechnet.
- ST_Quantile - Geändert: 3.1.0 Die Variante ST_Quantile(table_name, column_name) wurde entfernt. Berechnet die Quantile eines Rasters oder einer Rastercoverage Tabelle im Kontext von Stichproben oder Bevölkerung. Dadurch kann untersucht werden, ob ein Wert bei 25%, 50% oder 75% Perzentil des Rasters liegt.
- ST_SummaryStats - Geändert: 3.1.0 ST_SummaryStats(rastertable, rastercolumn, ...) Varianten wurden entfernt. Verwenden Sie stattdessen . 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 if no band is specified.

13.12.8 PostGIS-Funktionen neu oder erweitert in 3.0

Die unten aufgeführten Funktionen sind PostGIS-Funktionen, die hinzugefügt oder erweitert wurden.

Neue Funktionen in PostGIS 3.0

- CG_ConstrainedDelaunayTriangles - Verfügbarkeit: 2.1.0 Gibt eine eingeschränkte Delaunay-Triangulation um die angegebene Eingabegeometrie zurück.
- ST_3DLineInterpolatePoint - Verfügbarkeit: 2.1.0 Gibt einen oder mehrere, entlang einer Linie interpolierte Punkte zurück.
- ST_TileEnvelope - Verfügbarkeit: 2.1.0 Erzeugt ein rechteckiges Polygon in Web Mercator (SRID:3857) unter Verwendung des XYZ-Kachelsystems.

Erweiterte Funktionen in PostGIS 3.0

- ST_AsMVT - Erweiterung: 3.0 - Unterstützung für eine Feature-ID. Aggregatfunktion, die eine MVT-Darstellung einer Reihe von Zeilen zurückgibt.
- ST_Contains - Verbessert: 3.0.0 ermöglicht die Unterstützung von GEOMETRYCOLLECTION Tests, wenn jeder Punkt von B in A liegt und ihre Innenräume einen gemeinsamen Punkt haben
- ST_ContainsProperly - Verbessert: 3.0.0 ermöglicht die Unterstützung von GEOMETRYCOLLECTION Prüft, ob jeder Punkt von B im Inneren von A liegt
- ST_CoveredBy - Verbessert: 3.0.0 ermöglicht die Unterstützung von GEOMETRYCOLLECTION Prüft, ob jeder Punkt von A in B liegt
- ST_Covers - Verbessert: 3.0.0 ermöglicht die Unterstützung von GEOMETRYCOLLECTION Prüft, ob jeder Punkt von B in A liegt
- ST_Crosses - Verbessert: 3.0.0 ermöglicht die Unterstützung von GEOMETRYCOLLECTION Prüft, ob zwei Geometrien einige, aber nicht alle, innere Punkte gemeinsam haben
- ST_CurveToLine - Erweiterung: 3.0.0 führte eine minimale Anzahl an Segmenten pro linearisierten Bogen ein, um einem topologischen Kollaps vorzubeugen. Konvertiert eine Geometrie mit Kurven in eine lineare Geometrie.

- ST_Disjoint - Verbessert: 3.0.0 ermöglicht die Unterstützung von GEOMETRYCOLLECTION Prüft, ob zwei Geometrien keine gemeinsamen Punkte haben
- ST_Equals - Verbessert: 3.0.0 ermöglicht die Unterstützung von GEOMETRYCOLLECTION Prüft, ob zwei Geometrien dieselbe Menge von Punkten enthalten
- ST_GeneratePoints - Erweiterung: mit 3.0.0 wurde das Argument "seed" hinzugefügt Erzeugt einen Multipunkt aus zufälligen Punkten, die in einem Polygon oder MultiPolygon enthalten sind.
- ST_GeomFromGeoJSON - Verbessert: 3.0.0 Geometrie wird standardmäßig auf SRID=4326 gesetzt, wenn nicht anders angegeben. Nimmt als Eingabe eine GeoJSON-Darstellung der Geometrie und gibt ein geometrisches PostGIS-Objekt aus.
- ST_LocateBetween - Verbessert: 3.0.0 - Unterstützung für POLYGON, TIN, TRIANGLE hinzugefügt. Gibt die Teile einer Geometrie zurück, die einem Messbereich entsprechen.
- ST_LocateBetweenElevations - Verbessert: 3.0.0 - Unterstützung für POLYGON, TIN, TRIANGLE hinzugefügt. Gibt die Teile einer Geometrie zurück, die in einem Höhenbereich (Z) liegen.
- ST_Overlaps - Verbessert: 3.0.0 ermöglicht die Unterstützung von GEOMETRYCOLLECTION Prüft, ob zwei Geometrien die gleiche Abmessung haben und sich schneiden, aber jede mindestens einen Punkt hat, der nicht in der anderen liegt
- ST_Relate - Verbessert: 3.0.0 ermöglicht die Unterstützung von GEOMETRYCOLLECTION Prüft, ob zwei Geometrien eine topologische Beziehung haben, die einem Schnittpunktmatrixmuster entspricht, oder berechnet ihre Schnittpunktmatrix
- ST_Segmentize - Verbessert: 3.0.0 Segmentize-Geometrie erzeugt jetzt Teilsegmente gleicher Länge Gibt eine geänderte Geometrie/Geografie zurück, bei der kein Segment länger als eine bestimmte Entfernung ist.
- ST_Touches - Verbessert: 3.0.0 ermöglicht die Unterstützung von GEOMETRYCOLLECTION Prüft, ob zwei Geometrien mindestens einen Punkt gemeinsam haben, aber ihre Innenräume sich nicht schneiden
- ST_Within - Verbessert: 3.0.0 ermöglicht die Unterstützung von GEOMETRYCOLLECTION Tests, wenn jeder Punkt von A in B liegt und ihre Innenräume einen gemeinsamen Punkt haben

Geänderte Funktionen in PostGIS 3.0

- PostGIS_Extensions_Upgrade - Geändert: 3.0.0, um lose Erweiterungen neu zu packen und postgis_raster zu unterstützen. Packt und aktualisiert PostGIS-Erweiterungen (z.B. postgis_raster, postgis_topology, postgis_sfcgal) auf die angegebene oder neueste Version.
- ST_3DDistance - Geändert: 3.0.0 - SFCGAL-Version entfernt Für den geometrischen Datentyp. Es wird der geringste 3-dimensionale kartesische Abstand (basierend auf dem Koordinatenreferenzsystem) zwischen zwei geometrischen Objekten in projizierten Einheiten zurückgegeben.
- ST_3DIntersects - Änderung: 3.0.0 das SFCGAL Back-end wurde entfernt, das GEOS Back-end unterstützt TIN. Prüft, ob sich zwei Geometrien in 3D räumlich schneiden - nur für Punkte, Linienzüge, Polygone, polyedrische Flächen (Bereich)
- ST_Area - Geändert: 3.0.0 - hängt nicht mehr von SFCGAL ab. Gibt den geometrischen Schwerpunkt einer Geometrie zurück.
- ST_AsGeoJSON - Änderung: 3.0.0 Unterstützung von Datensätzen bei der Eingabe Rückgabe einer Geometrie oder eines Merkmals im GeoJSON-Format.
- ST_AsGeoJSON - Änderung: 3.0.0 Ausgabe der SRID wenn nicht EPSG:4326 Rückgabe einer Geometrie oder eines Merkmals im GeoJSON-Format.
- ST_AsKML - Geändert: 3.0.0 - Die Signatur der "versionierten" Variante wurde entfernt. Gibt die Geometrie als GML-Element - Version 2 oder 3 - zurück.
- ST_Distance - Geändert: 3.0.0 - hängt nicht mehr von SFCGAL ab. Gibt die größte 3-dimensionale Distanz zwischen zwei geometrischen Objekten als Linie zurück
- ST_Intersection - Geändert: 3.0.0 ist nicht von SFCGAL abhängig. Berechnet eine Geometrie, die den gemeinsamen Teil der Geometrien A und B darstellt.

- ST_Intersects - Geändert: 3.0.0 SFCGAL Version entfernt und native Unterstützung für 2D TINS hinzugefügt. Prüft, ob sich zwei Geometrien schneiden (sie haben mindestens einen Punkt gemeinsam)
- ST_Union - Geändert: 3.0.0 ist nicht von SFCGAL abhängig. Berechnet eine Geometrie, die die Punktmengenvereinigung der Eingabegeometrien darstellt.

13.12.9 PostGIS-Funktionen neu oder erweitert in 2.5

Die unten aufgeführten Funktionen sind PostGIS-Funktionen, die hinzugefügt oder erweitert wurden.

Neue Funktionen in PostGIS 2.5

- PostGIS_Extensions_Upgrade - Verfügbarkeit: 2.5.0 Packt und aktualisiert PostGIS-Erweiterungen (z.B. postgis_raster, postgis_topology, postgis_sfcgal) auf die angegebene oder neueste Version.
- ST_Angle - Verfügbarkeit: 2.5.0 Gibt den Winkel zwischen 3 Punkten oder zwischen 2 Vektoren (4 Punkte oder 2 Linien) zurück.
- ST_AsHexWKB - Verfügbarkeit: 2.5.0 Gibt die Well-known-Binary (WKB) Hex-Darstellung eines Rasters zurück.
- ST_BandFileSize - Verfügbarkeit: 2.5.0 Gibt die Dateigröße eines im Dateisystem gespeicherten Bandes aus. Wenn "bandnum" nicht angegeben ist, wird 1 angenommen.
- ST_BandFileTimestamp - Verfügbarkeit: 2.5.0 Gibt den Zeitstempel eines im Dateisystem gespeicherten Bandes aus. Wenn "bandnum" nicht angegeben ist, wird 1 angenommen.
- ST_ChaikinSmoothing - Verfügbarkeit: 2.5.0 Gibt eine geglättete Version einer Geometrie zurück, die den Chaikin-Algorithmus verwendet
- ST_FilterByM - Verfügbarkeit: 2.5.0 Entfernt Scheitelpunkte basierend auf ihrem M-Wert
- ST_Grayscale - Verfügbarkeit: 2.5.0 Erzeugt einen neuen Raster mit einem 8BUI-Band aus dem Ausgangsrasters und den angegebenen Bändern für Rot, Grün und Blau
- ST_LineInterpolatePoints - Verfügbarkeit: 2.5.0 Gibt einen oder mehrere, entlang einer Linie interpolierte Punkte zurück.
- ST_OrientedEnvelope - Verfügbarkeit: 2.5.0. Gibt ein Rechteck mit minimalem Flächeninhalt zurück, das eine Geometrie enthält.
- ST_QuantizeCoordinates - Verfügbarkeit: 2.5.0 Setzt die niedrigwertigsten Bits der Koordinaten auf Null
- ST_RastFromHexWKB - Verfügbarkeit: 2.5.0 Gibt einen Rasterwert von einer Well-known-Binary (WKB) Hex-Darstellung eines Rasters zurück.
- ST_RastFromWKB - Verfügbarkeit: 2.5.0 Gibt einen Rasterwert von einer Well-known-Binary (WKB) Darstellung eines Rasters zurück.
- ST_SetBandIndex - Verfügbarkeit: 2.5.0 Aktualisiert die externe Bandnummer eines out-db Bandes.
- ST_SetBandPath - Verfügbarkeit: 2.5.0 Aktualisiert den externen Dateipfad und die Bandnummer eines out-db Bandes.

Erweiterte Funktionen in PostGIS 2.5

- ST_AsBinary/ST_AsWKB - Erweiterung: 2.5.0 ST_AsWKB hinzugefügt Gibt die Well-known-Binary (WKB) Darstellung eines Rasters zurück.
- ST_AsMVT - Erweiterung: 2.5.0 - Unterstützung von nebenläufigen Abfragen. Aggregatfunktion, die eine MVT-Darstellung einer Reihe von Zeilen zurückgibt.
- ST_AsText - Erweiterung: 2.5 - der optionale Parameter "precision" wurde eingeführt. Gibt die Well-known-Text(WKT) Darstellung der Geometrie/Geographie ohne die SRID Metadaten zurück.

- ST_BandMetaData - Erweiterung: 2.5.0 inkludiert jetztoutdbbandnum, filesize und filetimestamp für outdb Raster. Gibt die grundlegenden Metadaten eines bestimmten Rasterbandes aus. Wenn der Parameter "bandnum" nicht angegeben ist, wird das 1ste Band angenommen.
- ST_Buffer - Erweiterung: 2.5.0 - ST_Buffer ermöglicht jetzt auch eine seitliche Pufferzonenberechnung über side=both|left|right. Berechnet eine Geometrie, die alle Punkte innerhalb eines bestimmten Abstands zu einer Geometrie umfasst.
- ST_GeomFromGeoJSON - Erweiterung: 2.5.0 unterstützt nun auch die Eingabe von json und jsonb. Nimmt als Eingabe eine GeoJSON-Darstellung der Geometrie und gibt ein geometrisches PostGIS-Objekt aus.
- ST_GeometricMedian - Erweiterung: 2.5.0 Unterstützung für M zur Gewichtung nach Punkten. Gibt den geometrischen Median eines Mehrfachpunktes zurück.
- ST_Intersects - Verbessert: 2.5.0 Unterstützt GEOMETRYCOLLECTION. Prüft, ob sich zwei Geometrien schneiden (sie haben mindestens einen Punkt gemeinsam)
- ST_OffsetCurve - Erweiterung: ab 2.5 wird auch GEOMETRYCOLLECTION und MULTILINESTRING unterstützt. Gibt eine versetzte Linie in einem bestimmten Abstand und einer bestimmten Seite von einer Eingabelinie zurück.
- ST_Scale - Verbessert: In Version 2.5.0 wurde die Unterstützung für die Skalierung relativ zu einem lokalen Ursprung (Parameterorigin) eingeführt. Skaliert eine Geometrie um bestimmte Faktoren.
- ST_Split - Verbessert: In Version 2.5.0 wurde die Unterstützung für die Aufteilung eines Polygons durch eine Mehrlinie eingeführt. Gibt eine Sammlung von Geometrien zurück, die durch Aufteilung einer Geometrie durch eine andere Geometrie entstanden sind.
- ST_Subdivide - Verbessert: 2.5.0 verwendet vorhandene Punkte bei der Polygonaufteilung wieder, die Anzahl der Scheitelpunkte wurde von 8 auf 5 gesenkt. Berechnet eine geradlinige Unterteilung einer Geometrie.

Geänderte Funktionen in PostGIS 2.5

- ST_GDALDrivers - Änderung: 2.5.0 - die Spalten can_read und can_write hinzugefügt. Gibt eine Liste der Rasterformate aus, die von PostGIS über die Bibliothek GDAL unterstützt werden. Nur die Formate mit can_write=True können von ST_AsGDALRaster verwendet werden.

13.12.10 PostGIS-Funktionen neu oder erweitert in 2.4

Die unten aufgeführten Funktionen sind PostGIS-Funktionen, die hinzugefügt oder erweitert wurden.

Neue Funktionen in PostGIS 2.4

- ST_AsGeobuf - Verfügbarkeit: 2.4.0 Gibt eine Menge an Zeilen in der Geobuf Darstellung aus.
- ST_AsMVT - Verfügbarkeit: 2.4.0 Aggregatfunktion, die eine MVT-Darstellung einer Reihe von Zeilen zurückgibt.
- ST_AsMVTGeom - Verfügbarkeit: 2.4.0 Transformiert eine Geometrie in den Koordinatenraum einer MVT-Kachel.
- ST_Centroid - Verfügbarkeit: Mit 2.4.0 wurde die Unterstützung für den geograpischen Datentyp eingeführt. Gibt den geometrischen Schwerpunkt einer Geometrie zurück.
- ST_ForcePolygonCCW - Verfügbarkeit: 2.4.0 Richtet alle äußeren Ringe gegen den Uhrzeigersinn und alle inneren Ringe mit dem Uhrzeigersinn aus.
- ST_ForcePolygonCW - Verfügbarkeit: 2.4.0 Richtet alle äußeren Ringe im Uhrzeigersinn und alle inneren Ringe gegen den Uhrzeigersinn aus.
- ST_FrechetDistance - Verfügbarkeit: 2.4.0 - benötigt GEOS >= 3.7.0 Gibt den kürzesten 3-dimensionalen Abstand zwischen zwei geometrischen Objekten als Linie zurück
- ST_IsPolygonCCW - Verfügbarkeit: 2.4.0 Gibt TRUE zurück, wenn alle äußeren Ringe gegen den Uhrzeigersinn orientiert sind und alle inneren Ringe im Uhrzeigersinn ausgerichtet sind.

- ST_IsPolygonCW - Verfügbarkeit: 2.4.0 Gibt den Wert TRUE zurück, wenn alle äußeren Ringe im Uhrzeigersinn und alle inneren Ringe gegen den Uhrzeigersinn ausgerichtet sind.
- ST_MakeEmptyCoverage - Verfügbarkeit: 2.4.0 Bedeckt die georeferenzierte Fläche mit einem Gitter aus leeren Rasterkacheln.

Erweiterte Funktionen in PostGIS 2.4

- ST_AsTWKB - Erweiterung: 2.4.0 Hauptspeicher- und Geschwindigkeitsverbesserungen. Gibt die Geometrie als TWKB, aka "Tiny Well-known Binary" zurück
- ST_Covers - Verbessert: 2.4.0 Unterstützung für Polygon in Polygon und Linie in Polygon für Geografietypen hinzugefügt Prüft, ob jeder Punkt von B in A liegt
- ST_CurveToLine - Erweiterung: ab 2.4.0 kann die Toleranz über die 'maximale Abweichung' und den 'maximalen Winkel' angegeben werden. Die symmetrische Ausgabe wurde hinzugefügt. Konvertiert eine Geometrie mit Kurven in eine lineare Geometrie.
- ST_Project - Verbessert: 2.4.0 Erlaubt negative Entfernungen und nicht-normierte Azimute. Gibt einen Punkt zurück, der von einem Startpunkt um eine bestimmte Entfernung und Peilung (Azimut) projiziert wird.
- ST_Reverse - Erweiterung: mit 2.4.0 wurde die Unterstützung für Kurven eingeführt. Gibt die Geometrie in umgekehrter Knotenreihenfolge zurück.

Geänderte Funktionen in PostGIS 2.4

- = - Änderung: 2.4.0, in Vorgängerversionen war dies die Gleichheit der umschreibenden Rechtecke, nicht die geometrische Gleichheit. Falls Sie auf Gleichheit der umschreibenden Rechtecke prüfen wollen, verwenden Sie stattdesse bitte . Gibt TRUE zurück, wenn die Koordinaten und die Reihenfolge der Koordinaten der Geometrie/Geographie A und der Geometrie/Geographie B ident sind.
- ST_Node - Geändert: 2.4.0 verwendet diese Funktion intern GEOSNode anstelle von GEOSUnaryUnion. Dies kann dazu führen, dass die resultierenden Linestrings eine andere Reihenfolge und Richtung haben als in PostGIS < 2.4. Knoten eine Sammlung von Linien.

13.12.11 PostGIS-Funktionen neu oder erweitert in 2.3

Die unten aufgeführten Funktionen sind PostGIS-Funktionen, die hinzugefügt oder erweitert wurden.

Neue Funktionen in PostGIS 2.3

- &&&(geometry,gidx) - Availability: 2.3.0 support for Block Range INdexes (BRIN) was introduced. Gibt TRUE zurück, wenn die (cached) n-D bounding box einer Geometrie eine n-D float precision bounding box (GIDX) schneidet.
- &&&(gidx,geometry) - Availability: 2.3.0 support for Block Range INdexes (BRIN) was introduced. Gibt TRUE zurück, wenn eine n-D float precision bounding box (GIDX) eine (cached) n-D bounding box einer Geometrie schneidet.
- &&&(gidx,gidx) - Availability: 2.3.0 support for Block Range INdexes (BRIN) was introduced. Gibt TRUE zurück, wenn sich zwei n-D float precision bounding boxes (GIDX) gegenseitig überschneiden.
- &&(box2df,box2df) - Availability: 2.3.0 support for Block Range INdexes (BRIN) was introduced. Gibt TRUE zurück, wenn sich zwei 2D float precision Bounding Boxes (BOX2DF) überschneiden.
- &&(box2df,geometry) - Availability: 2.3.0 support for Block Range INdexes (BRIN) was introduced. Gibt TRUE zurück, wenn eine 2D float precision bounding box (BOX2DF) eine Geometrie (cached) 2D bounding box schneidet.
- &&(geometry,box2df) - Availability: 2.3.0 support for Block Range INdexes (BRIN) was introduced. Gibt TRUE zurück, wenn sich die 2D Bounding Box (cached) einer Geometrie mit einer 2D Bounding Box mit Gleitpunktgenauigkeit (BOX2DF) überschneidet.

- @(box2df,box2df) - Availability: 2.3.0 support for Block Range INdexes (BRIN) was introduced. Gibt TRUE zurück, wenn eine 2D float precision bounding box (BOX2DF) innerhalb einer anderen 2D float precision bounding box enthalten ist.
- @(box2df,geometry) - Availability: 2.3.0 support for Block Range INdexes (BRIN) was introduced. Gibt TRUE zurück, wenn eine 2D float precision bounding box (BOX2DF) in der 2D Bounding Box einer Geometrie enthalten ist..
- @(geometry,box2df) - Availability: 2.3.0 support for Block Range INdexes (BRIN) was introduced. Gibt TRUE zurück, wenn die 2D Bounding Box einer Geometrie in einer 2D float precision Bbounding Box (BOX2DF) enthalten ist.
- Populate_Topology_Layer - Verfügbarkeit: 2.3.0 Fügt fehlende Einträge zu der Tabelle topology.layer hinzu, indem Metadaten aus den topologischen Tabellen ausgelesen werden.
- ST_ClusterDBSCAN - Verfügbarkeit: 2.3.0 Fensterfunktion, die eine Cluster-ID für jede Eingabegeometrie unter Verwendung des DBSCAN-Algorithmus zurückgibt.
- ST_ClusterKMeans - Verfügbarkeit: 2.3.0 Fensterfunktion, die eine Cluster-ID für jede Eingabegeometrie unter Verwendung des K-Means-Algorithmus zurückgibt.
- ST_GeneratePoints - Verfügbarkeit: 2.3.0 Erzeugt einen Multipunkt aus zufälligen Punkten, die in einem Polygon oder Multi-Polygon enthalten sind.
- ST_GeometricMedian - Verfügbarkeit: 2.3.0 Gibt den geometrischen Median eines Mehrfachpunktes zurück.
- ST_MakeLine - Verfügbarkeit: Mit 2.3.0 wurde die Unterstützung zur Eingabe von MultiPoint Elementen eingeführt Erzeugt einen Linienzug aus einer Punkt-, Mehrfachpunkt- oder Liniengeometrie.
- ST_MinimumBoundingRadius - Verfügbarkeit: 2.3.0 Gibt den Mittelpunkt und den Radius des kleinsten Kreises zurück, der eine Geometrie enthält.
- ST_MinimumClearance - Verfügbarkeit: 2.3.0 Gibt das Mindestabstandsmaß für eine Geometrie zurück; ein Maß für die Robustheit einer Geometrie.
- ST_MinimumClearanceLine - Verfügbarkeit: 2.3.0 - benötigt GEOS >= 3.6.0 Gibt ein Linienstück mit zwei Punkten zurück, welche sich über das Mindestabstandsmaß erstreckt.
- ST_Normalize - Verfügbarkeit: 2.3.0 Gibt die Geometrie in Normalform zurück.
- ST_Points - Verfügbarkeit: 2.3.0 Gibt einen MultiPoint zurück, welcher alle Koordinaten einer Geometrie enthält.
- ST_VoronoiLines - Verfügbarkeit: 2.3.0 Gibt die Grenzen des Voronoi-Diagramms der Eckpunkte einer Geometrie zurück.
- ST_VoronoiPolygons - Verfügbarkeit: 2.3.0 Gibt die Zellen des Voronoi-Diagramms der Scheitelpunkte einer Geometrie zurück.
- ST_WrapX - Verfügbarkeit: 2.3.0 erfordert GEOS Versammelt eine Geometrie um einen X-Wert
- TopoGeom_addElement - Verfügbarkeit: 2.3 Fügt ein Element zu der Definition einer TopoGeometry hinzu.
- TopoGeom_remElement - Verfügbarkeit: 2.3 Entfernt ein Element aus der Definition einer TopoGeometry.
- ~(box2df,box2df) - Availability: 2.3.0 support for Block Range INdexes (BRIN) was introduced. Gibt TRUE zurück, wenn eine 2D float precision bounding box (BOX2DF) eine andere 2D float precision bounding box (BOX2DF) enthält.
- ~(box2df,geometry) - Availability: 2.3.0 support for Block Range INdexes (BRIN) was introduced. Gibt TRUE zurück, wenn eine 2D float precision bounding box (BOX2DF) die 2D Bounding Box einer Geometrie enthält.
- ~(geometry,box2df) - Availability: 2.3.0 support for Block Range INdexes (BRIN) was introduced. Gibt TRUE zurück, wenn die 2D bounding box einer Geometrie eine 2D float precision bounding box (GIDX) enthält.

Erweiterte Funktionen in PostGIS 2.3

- ST_Contains - Verbessert: 2.3.0 Verbesserung des PIP-Kurzschlusses erweitert um die Unterstützung von MultiPoints mit wenigen Punkten. Frühere Versionen unterstützten nur Punkte in Polygonen. Tests, wenn jeder Punkt von B in A liegt und ihre Innenräume einen gemeinsamen Punkt haben

- ST_Covers - Verbessert: 2.3.0 Verbesserung des PIP-Kurzschlusses für Geometrien, erweitert um die Unterstützung von MultiPoints mit wenigen Punkten. Frühere Versionen unterstützten nur Punkte in Polygonen. Prüft, ob jeder Punkt von B in A liegt
- ST_Expand - Verbessert: In Version 2.3.0 wurde die Möglichkeit hinzugefügt, eine Box um unterschiedliche Beträge in verschiedenen Dimensionen zu erweitern. Gibt einen Begrenzungsrahmen zurück, der aus einem anderen Begrenzungsrahmen oder einer Geometrie erweitert wurde.
- ST_Intersects - Verbessert: 2.3.0 Verbesserung des PIP-Kurzschlusses erweitert um die Unterstützung von MultiPoints mit wenigen Punkten. Frühere Versionen unterstützten nur Punkte in Polygonen. Prüft, ob sich zwei Geometrien schneiden (sie haben mindestens einen Punkt gemeinsam)
- ST_Segmentize - Verbessert: 2.3.0 Die Segmentierung der Geografie erzeugt nun Teilsegmente gleicher Länge Gibt eine geänderte Geometrie/Geografie zurück, bei der kein Segment länger als eine bestimmte Entfernung ist.
- ST_Transform - Verbessert: In Version 2.3.0 wurde die Unterstützung für direkten PROJ.4 Text eingeführt. Rückgabe einer neuen Geometrie mit in ein anderes räumliches Bezugssystem transformierten Koordinaten.
- ST_Within - Verbessert: 2.3.0 Verbesserung des PIP-Kurzschlusses für Geometrien, erweitert um die Unterstützung von MultiPoints mit wenigen Punkten. Frühere Versionen unterstützten nur Punkte in Polygonen. Tests, wenn jeder Punkt von A in B liegt und ihre Innenräume einen gemeinsamen Punkt haben

Geänderte Funktionen in PostGIS 2.3

- ST_PointN - Änderung: 2.3.0 : negatives Indizieren verfügbar (-1 entspricht dem Endpunkt) Gibt die Anzahl der Stützpunkte eines ST_LineString oder eines ST_CircularString zurück.

13.12.12 PostGIS-Funktionen neu oder erweitert in 2.2

Die unten aufgeführten Funktionen sind PostGIS-Funktionen, die hinzugefügt oder erweitert wurden.

Neue Funktionen in PostGIS 2.2

- <<->> - Availability: 2.2.0. Gibt den n-D-Abstand zwischen den Geometrien oder Begrenzungsrahmen von A und B zurück
- ST_AsEncodedPolyline - Verfügbarkeit: 2.2.0 Erzeugt eine codierte Polylinie aus einer LineString Geometrie.
- ST_AsTWKB - Verfügbarkeit: 2.2.0 Gibt die Geometrie als TWKB, aka "Tiny Well-known Binary" zurück
- ST_BoundingDiagonal - Verfügbarkeit: 2.2.0 Gibt die Diagonale des Umgebungsdreiecks der angegebenen Geometrie zurück.
- ST_CPAWithin - Verfügbarkeit: 2.2.0 Prüft, ob der nächstgelegene Punkt der Annäherung zweier Flugbahnen innerhalb der angegebenen Entfernung liegt.
- ST_ClipByBox2D - Verfügbarkeit: 2.2.0 Berechnet den Teil einer Geometrie, der innerhalb eines Rechtecks liegt.
- ST_ClosestPointOfApproach - Verfügbarkeit: 2.2.0 Liefert ein Maß für den nächstgelegenen Punkt der Annäherung von zwei Flugbahnen.
- ST_ClusterIntersecting - Verfügbarkeit: 2.2.0 Aggregatfunktion, die Eingabegeometrien zu zusammenhängenden Mengen clustert.
- ST_ClusterWithin - Verfügbarkeit: 2.2.0 Aggregatfunktion, die Geometrien nach Trennungsabstand gruppiert.
- ST_CountAgg - Verfügbarkeit: 2.2.0 Aggregatfunktion. Gibt die Anzahl der Pixel in einem bestimmten Band der Raster aus. Wenn kein Band angegeben ist, wird Band 1 angenommen. Wenn "exclude_nodata_value" TRUE ist, werden nur die Pixel ohne NODATA Werte gezählt.
- ST_CreateOverview - Verfügbarkeit: 2.2.0 Erzeugt eine Version des gegebenen Raster-Coverage mit geringerer Auflösung.
- ST_DistanceCPA - Verfügbarkeit: 2.2.0 Liefert den Abstand zwischen dem nächstgelegenen Punkt der Annäherung zweier Flugbahnen.

- ST_ForceCurve - Verfügbarkeit: 2.2.0 Wandelt einen geometrischen in einen Kurven Datentyp um, soweit anwendbar.
- ST_IsValidTrajectory - Verfügbarkeit: 2.2.0 Prüft, ob die Geometrie eine gültige Flugbahn ist.
- ST_LineFromEncodedPolyline - Verfügbarkeit: 2.2.0 Erzeugt einen LineString aus einem codierten Linienzug.
- ST_MapAlgebra (callback function version) - Verfügbarkeit: 2.2.0: Möglichkeit eine Maske hinzuzufügen Die Version mit der Rückruffunktion - Gibt für einen oder mehrere Eingaberaster einen Raster mit einem Band, den Bandindizes und einer vom Anwender vorgegebenen Rückruffunktion zurück.
- ST_MemSize - Verfügbarkeit: 2.2.0 Gibt den Platzbedarf des Rasters (in Byte) aus.
- ST_RemoveRepeatedPoints - Verfügbarkeit: 2.2.0 Gibt eine Version einer Geometrie zurück, bei der doppelte Punkte entfernt wurden.
- ST_Retile - Verfügbarkeit: 2.2.0 Gibt konfigurierte Kacheln eines beliebig gekachelten Rastercoverage aus.
- ST_SetEffectiveArea - Verfügbarkeit: 2.2.0 Legt die effektive Fläche für jeden Scheitelpunkt unter Verwendung des Visvalingam-Whyatt-Algorithmus fest.
- ST_SimplifyVW - Verfügbarkeit: 2.2.0 Liefert eine vereinfachte Darstellung einer Geometrie unter Verwendung des Visvalingam-Whyatt-Algorithmus
- ST_Subdivide - Verfügbarkeit: 2.2.0 Berechnet eine geradlinige Unterteilung einer Geometrie.
- ST_SummaryStatsAgg - Verfügbarkeit: 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 if no band is specified.
- ST_SwapOrdinates - Verfügbarkeit: 2.2.0 Gibt eine Version der Ausgangsgeometrie zurück, in der die angegebenen Ordinatenwerte ausgetauscht werden.
- postgis.enable_outdb_rasters - Verfügbarkeit: 2.2.0 Eine boolesche Konfigurationsmöglichkeit um den Zugriff auf out-db Rasterbänder zu ermöglichen
- postgis.gdal_datapath - Verfügbarkeit: 2.2.0 Eine Konfigurationsmöglichkeit um den Wert von GDAL's GDAL_DATA Option zu setzen. Wenn sie nicht gesetzt ist, wird die Umgebungsvariable GDAL_DATA verwendet.
- postgis.gdal_enabled_drivers - Verfügbarkeit: 2.2.0 Eine Konfigurationsmöglichkeit um einen GDAL Treiber in der PostGIS Umgebung zu aktivieren. Beeinflusst die Konfigurationsvariable GDAL_SKIP von GDAL.
- |=| - Availability: 2.2.0. Gibt die Entfernung zwischen den Trajektorien A und B, am Ort der dichtesten Annäherung, an.

Erweiterte Funktionen in PostGIS 2.2

- <-> - Enhanced: 2.2.0 -- True KNN ("K nearest neighbor") behavior for geometry and geography. Note for geography KNN is based on sphere rather than spheroid. Gibt die 2D Entfernung zwischen A und B zurück.
- AsTopoJSON - Erweiterung: 2.2.1 Unterstützung für punktförmige Eingabewerte hinzugefügt Gibt die TopoJSON-Darstellung einer TopoGeometry zurück.
- ST_Area - Erweiterung: 2.2.0 - die Messung auf dem Referenzellipsoid wird mit der Bibliothek "GeographicLib" durchgeführt. Dadurch wurde die Genauigkeit und die Robustheit erhöht. Um die Vorteile dieser neuen Funktionalität zu nutzen, benötigen Sie Proj >= 4.9.0. Gibt den geometrischen Schwerpunkt einer Geometrie zurück.
- ST_AsX3D - Erweiterung: 2.2.0: Unterstützung für geographische Koordinaten und Vertauschen der Achsen (x/y, Länge/Breite). Für nähere Details siehe Optionen. Gibt eine Geometrie im X3D XML Knotenelement-Format zurück: ISO-IEC-19776-1.2-X3DEncodings-XML
- ST_Azimuth - Erweiterung: 2.2.0 die Messungen auf dem Referenzellipsoid werden mit der Bibliothek "GeographicLib" durchgeführt. Dadurch wurde die Genauigkeit und die Robustheit erhöht. Um die Vorteile dieser neuen Funktionalität zu nutzen, benötigen Sie Proj >= 4.9.0. Gibt die 2-dimenionale kürzeste Strecke zwischen zwei Geometrien als Linie zurück

- ST_Distance - Erweiterung: 2.2.0 - die Messung auf dem Referenzellipsoid wird mit der Bibliothek "GeographicLib" durchgeführt. Dadurch wurde die Genauigkeit und die Robustheit erhöht. Um die Vorteile dieser neuen Funktionalität zu nutzen, benötigen Sie Proj >= 4.9.0. Gibt die größte 3-dimensionale Distanz zwischen zwei geometrischen Objekten als Linie zurück
- ST_Scale - Verbessert: In Version 2.2.0 wurde die Unterstützung für die Skalierung aller Dimensionen (Parameterfactor) eingeführt. Skaliert eine Geometrie um bestimmte Faktoren.
- ST_Split - Verbessert: In Version 2.2.0 wurde die Unterstützung für die Aufteilung einer Linie durch eine Mehrlinien-, eine Mehrpunkt- oder eine (Mehr-)Polygonbegrenzung eingeführt. Gibt eine Sammlung von Geometrien zurück, die durch Aufteilung einer Geometrie durch eine andere Geometrie entstanden sind.
- ST_Summary - Erweiterung: 2.2.0 Unterstützung für TIN und Kurven Gibt eine Zusammenfassung des Inhalts einer Geometrie wieder.

Geänderte Funktionen in PostGIS 2.2

- <-> - Changed: 2.2.0 -- Old hybrid-syntax workarounds may be slower once true KNN is available. See examples below. Gibt die 2D Entfernung zwischen A und B zurück.
- ST_3DClosestPoint - Änderung: 2.2.0 - Wenn 2 geometrische Objekte in 2D übergegeben werden, wird ein 2D-Punkt zurückgegeben (anstelle wie früher 0 für ein fehlendes Z). Im Falle von 2D und 3D, wird für fehlende Z nicht länger 0 angenommen. Gibt den 3-dimensionalen Punkt auf g1 zurück, der den kürzesten Abstand zu g2 hat. Dies ist der Anfangspunkt des kürzesten Abstands in 3D.
- ST_3DDistance - Änderung: 2.2.0 - Im Falle von 2D und 3D wird für ein fehlendes Z nicht mehr 0 angenommen. Für den geometrischen Datentyp. Es wird der geringste 3-dimensionale kartesische Abstand (basierend auf dem Koordinatenreferenzsystem) zwischen zwei geometrischen Objekten in projizierten Einheiten zurückgegeben.
- ST_3DLongestLine - Änderung: 2.2.0 - Wenn 2 geometrische Objekte in 2D übergegeben werden, wird ein 2D-Punkt zurückgegeben (anstelle wie früher 0 für ein fehlendes Z). Im Falle von 2D und 3D, wird für fehlende Z nicht länger 0 angenommen. Gibt die größte 3-dimensionale Distanz zwischen zwei geometrischen Objekten als Linie zurück
- ST_3DMaxDistance - Änderung: 2.2.0 - Im Falle von 2D und 3D wird für ein fehlendes Z nicht mehr 0 angenommen. Für den geometrischen Datentyp. Gibt die maximale 3-dimensionale kartesische Distanz (basierend auf dem Koordinatenreferenzsystem) zwischen zwei geometrischen Objekten in projizierten Einheiten zurück.
- ST_3DShortestLine - Änderung: 2.2.0 - Wenn 2 geometrische Objekte in 2D übergegeben werden, wird ein 2D-Punkt zurückgegeben (anstelle wie früher 0 für ein fehlendes Z). Im Falle von 2D und 3D, wird für fehlende Z nicht länger 0 angenommen. Gibt den kürzesten 3-dimensionalen Abstand zwischen zwei geometrischen Objekten als Linie zurück
- ST_DistanceSphere - Änderung: 2.2.0 In Vorgängerversionen als ST_Distance_Sphere bezeichet. Gibt die kürzeste Distanz zwischen zwei geometrischen Objekten zurück, die über Länge, Breite und ein bestimmtes Referenzellipsoid gegeben sind. Vorgängerversionen von PostGIS 1.5 unterstützten nur Punkte.
- ST_DistanceSpheroid - Änderung: 2.2.0 In Vorgängerversionen als ST_Distance_Spheroid bezeichet. Gibt die kürzeste Distanz zwischen zwei geometrischen Objekten zurück, die über Länge, Breite und ein bestimmtes Referenzellipsoid gegeben sind. Vorgängerversionen von PostGIS 1.5 unterstützten nur Punkte.
- ST_Equals - Geändert: 2.2.0 Gibt auch bei ungültigen Geometrien true zurück, wenn sie binär gleich sind Prüft, ob zwei Geometrien dieselbe Menge von Punkten enthalten
- ST_LengthSpheroid - Änderung: 2.2.0 In Vorgängerversionen als ST_Length_Spheroid bezeichet.und mit dem Alias "ST_3DLength_S versehen Gibt den geometrischen Schwerpunkt einer Geometrie zurück.
- ST_MemSize - Geändert: 2.2.0 Name geändert in ST_MemSize, um der Namenskonvention zu folgen. Gibt den Geometrietyp des ST_Geometry Wertes zurück.
- ST_PointInsideCircle - Geändert: 2.2.0 In früheren Versionen hieß dies ST_Point_Inside_Circle Prüft, ob ein geometrischer Punkt innerhalb eines Kreises liegt, der durch einen Mittelpunkt und einen Radius definiert ist
- ValidateTopology - Änderung: 2.2.0 Bei 'edge crosses node' wurden die Werte für id1 und id2 vertauscht, um mit der Fehlerbeschreibung konsistent zu sein. Liefert eine Menge validatetopology_returntype Objekte, die Probleme mit der Topologie beschreiben.

13.12.13 PostGIS-Funktionen neu oder erweitert in 2.1

Die unten aufgeführten Funktionen sind PostGIS-Funktionen, die hinzugefügt oder erweitert wurden.

Neue Funktionen in PostGIS 2.1

- = - Verfügbarkeit: 2.1.0 Gibt TRUE zurück, wenn die umschreibenden Rechtecke von A und B ident sind. Das umschreibende Rechteck ist in Double Precision.
- AsTopoJSON - Verfügbarkeit: 2.1.0 Gibt die TopoJSON-Darstellung einer TopoGeometry zurück.
- ST_Box2dFromGeoHash - Verfügbarkeit: 2.1.0 Gibt die BOX2D einer GeoHash Zeichenkette zurück.
- ST_ColorMap - Verfügbarkeit: 2.1.0 Erzeugt aus einem bestimmten Band des Ausgangsrasters einen neuen Raster mit bis zu vier 8BUI-Bändern (Grauwert, RGB, RGBA). Wenn kein Band angegeben ist, wird Band 1 angenommen.
- ST_Contains - Verfügbarkeit: 2.1.0 Gibt TRUE zurück, wenn kein Punkt des Rasters "rastB" im Äußeren des Rasters "rastA" liegt und zumindest ein Punkt im Inneren von "rastB" auch im Inneren von "rastA" liegt.
- ST_ContainsProperly - Verfügbarkeit: 2.1.0 Gibt TRUE zurück, wenn "rastB" das Innere von "rastA" schneidet, aber nicht die Begrenzung oder das Äußere von "rastA".
- ST_CoveredBy - Verfügbarkeit: 2.1.0 Gibt TRUE zurück, wenn kein Punkt des Rasters "rastA" außerhalb des Rasters "rastB" liegt.
- ST_Covers - Verfügbarkeit: 2.1.0 Gibt TRUE zurück, wenn kein Punkt des Rasters "rastB" außerhalb des Rasters "rastA" liegt.
- ST_DFullyWithin - Verfügbarkeit: 2.1.0 Gibt TRUE zurück, wenn die Raster "rastA" und "rastB" zur Gänze innerhalb der angegebenen Distanz zueinander liegen.
- ST_DWithin - Verfügbarkeit: 2.1.0 Gibt TRUE zurück, wenn die Raster "rastA" und "rastB" innerhalb der angegebenen Entfernung voneinander liegen.
- ST_DelaunayTriangles - Verfügbarkeit: 2.1.0 Gibt die Delaunay-Triangulation der Scheitelpunkte einer Geometrie zurück.
- ST_Disjoint - Verfügbarkeit: 2.1.0 Gibt TRUE zurück, wenn sich die Raster "rastA" und "rastB" räumlich nicht überschneiden.
- ST_DumpValues - Verfügbarkeit: 2.1.0 Gibt die Werte eines bestimmten Bandes als 2-dimensionales Feld aus.
- ST_ForceSFS - Verfügbarkeit: 2.1.0 Force geometries to use SFS 1.1 or 1.2 geometry types.
- ST_FromGDALRaster - Verfügbarkeit: 2.1.0 Erzeugt einen Raster aus einer von GDAL unterstützten Rasterdatei.
- ST_GeomFromGeoHash - Verfügbarkeit: 2.1.0 Gibt die Geometrie einer GeoHash Zeichenfolge zurück.
- ST_InvDistWeight4ma - Verfügbarkeit: 2.1.0 Funktion zur Rasterdatenverarbeitung, die den Wert eines Pixel aus den Pixel der Nachbarschaft interpoliert.
- ST_MapAlgebra (callback function version) - Verfügbarkeit: 2.1.0 Die Version mit der Rückruffunktion - Gibt für einen oder mehrere Eingaberaster einen Raster mit einem Band, den Bandindizes und einer vom Anwender vorgegebenen Rückruffunktion zurück.
- ST_MapAlgebra (expression version) - Verfügbarkeit: 2.1.0 Version mit Ausdrücken - Gibt für einen oder zwei Ausgangsraster, Bandindizes und einer oder mehreren vom Anwender vorgegebenen SQL-Ausdrücken, einen Raster mit einem Band zurück.
- ST_MinConvexHull - Verfügbarkeit: 2.1.0 Gibt die Geometrie der konvexen Hülle des Raster aus, wobei Pixel mit NODATA ausgenommen werden.
- ST_MinDist4ma - Verfügbarkeit: 2.1.0 Funktion zur Rasterdatenverarbeitung, welche die kürzeste Entfernung (in Pixel) zwischen dem Pixel von Interesse und einem benachbarten Pixel mit Zellwert zurückgibt.
- ST_NearestValue - Verfügbarkeit: 2.1.0 Gibt den nächstgelegenen nicht NODATA Wert eines bestimmten Pixels aus, das über "columnx" und "rowy" oder durch eine Punktgeometrie - im gleichen Koordinatenreferenzsystem wie der Raster - ausgewählt wird.

- ST_Neighborhood - Verfügbarkeit: 2.1.0 Gibt ein 2-D Feld in "Double Precision" aus, das sich aus nicht NODATA Werten um ein bestimmtes Pixel herum zusammensetzt. Das Pixel Kann über "columnx" und "rowy" oder über eine Punktgeometrie - im gleichen Koordinatenreferenzsystem wie der Raster - ausgewählt werden.
- ST_NotSameAlignmentReason - Verfügbarkeit: 2.1.0 Gibt eine Meldung aus, die angibt ob die Raster untereinander ausgerichtet sind oder nicht und warum wenn nicht.
- ST_Overlaps - Verfügbarkeit: 2.1.0 Gibt TRUE zurück, wenn sich die Raster "rastA" und "rastB" schneiden, aber ein Raster den anderen nicht zur Gänze enthält.
- ST_PixelAsCentroid - Verfügbarkeit: 2.1.0 Gibt den geometrischen Schwerpunkt (Punktgeometrie) der Fläche aus, die durch das Pixel repräsentiert wird.
- ST_PixelAsCentroids - Verfügbarkeit: 2.1.0 Gibt den geometrischen Schwerpunkt (Punktgeometrie) für jedes Pixel des Rasterbandes zurück, zusammen mit dem Zellwert und den X- und Y-Rasterkoordinaten eines jeden Pixels. Die Koordinaten der Punkte entsprechen dem geometrischen Schwerpunkt der Pixel.
- ST_PixelAsPoint - Verfügbarkeit: 2.1.0 Gibt eine Punktgeometrie der oberen linken Ecke des Rasters zurück.
- ST_PixelAsPoints - Verfügbarkeit: 2.1.0 Gibt eine Punktgeometrie für jedes Pixel des Rasterbandes zurück, zusammen mit dem Zellwert und den X- und Y-Rasterkoordinaten eines jeden Pixels. Die Koordinaten der Punkte entsprechen dem oberen linken Eck der Pixel.
- ST_PixelOfValue - Verfügbarkeit: 2.1.0 Gibt die columnx- und rowy-Koordinaten jener Pixel aus, deren Zellwert gleich dem gesuchten Wert ist.
- ST_PointFromGeoHash - Verfügbarkeit: 2.1.0 Gibt einen Punkt von einer GeoHash Zeichenfolge zurück.
- ST_RasterToWorldCoord - Verfügbarkeit: 2.1.0 Gibt die obere linke Ecke des Rasters in geodätischem X und Y (Länge und Breite) für eine gegebene Spalte und Zeile aus. Spalte und Zeile wird von 1 aufwärts gezählt.
- ST_Resize - Verfügbarkeit: 2.1.0 benötigt GDAL 1.6.1+ Ändert die Zellgröße - width/height - eines Rasters
- ST_Roughness - Verfügbarkeit: 2.1.0 Gibt einen Raster mit der berechneten "Rauhigkeit" des DHM zurück.
- ST_SetValues - Verfügbarkeit: 2.1.0 Gibt einen Raster zurück, der durch das Setzen der Werte eines bestimmten Bandes verändert wurde.
- ST_Simplify - Verfügbarkeit: 2.1.0 Gibt für eine TopoGeometry eine "vereinfachte" geometrische Version zurück. Verwendet den Douglas-Peucker Algorithmus.
- ST_Summary - Verfügbarkeit: 2.1.0 Gibt eine textliche Zusammenfassung des Rasterinhalts zurück.
- ST_TPI - Verfügbarkeit: 2.1.0 Berechnet den "Topographic Position Index" eines Raster.
- ST_TRI - Verfügbarkeit: 2.1.0 Gibt einen Raster mit errechneten Geländerauheitsindex aus.
- ST_Tile - Verfügbarkeit: 2.1.0 Gibt Raster, die aus einer Teilungsoperation des Eingaberasters resultieren, mit den gewünschten Dimensionen aus.
- ST_Touches - Verfügbarkeit: 2.1.0 Gibt TRUE zurück, wenn rastA und rastB zumindest einen Punkt gemeinsam haben sich aber nicht überschneiden.
- ST_Union - Verfügbarkeit: 2.1.0 ST_Union(rast, unionarg) Variante wurde eingeführt. Gibt die Vereinigung mehrerer Rasterkacheln in einem einzelnen Raster mit mehreren Bändern zurück.
- ST_Within - Verfügbarkeit: 2.1.0 Gibt TRUE zurück, wenn kein Punkt des Rasters "rastA" außerhalb des Rasters "rastB" liegt und zumindest ein Punkt im Inneren von "rastA" auch im Inneren von "rastB" liegt.
- ST_WorldToRasterCoord - Verfügbarkeit: 2.1.0 Gibt für ein geometrisches X und Y (geographische Länge und Breite) oder für eine Punktgeometrie im Koordinatenreferenzsystem des Rasters, die obere linke Ecke als Spalte und Zeile aus.
- UpdateRasterSRID - Verfügbarkeit: 2.1.0 Änderung der SRID aller Raster in der vom Anwender angegebenen Spalte und Tabelle.

- clearTopoGeom - Verfügbarkeit: 2.1 Löscht den Inhalt einer TopoGeometry.
- postgis_sfcgal_version - Verfügbarkeit: 2.1.0 Gibt die verwendete Version von SFCGAL zurück

Erweiterte Funktionen in PostGIS 2.1

- ST_AddBand - Erweiterung: 2.1.0 - Unterstützung für "addbandarg" hinzugefügt. Gibt einen Raster mit den neu hinzugefügten Band(Bändern) aus. Der Typ , der Ausgangswert und der Index für den Speicherort des Bandes kann angegeben werden. Wenn kein Index angegeben ist, wird das Band am Ende hinzugefügt.
- ST_AddBand - Erweiterung: 2.1.0 Unterstützung für die neuen "out-db" Bänder hinzugefügt. Gibt einen Raster mit den neu hinzugefügten Band(Bändern) aus. Der Typ , der Ausgangswert und der Index für den Speicherort des Bandes kann angegeben werden. Wenn kein Index angegeben ist, wird das Band am Ende hinzugefügt.
- ST_AsBinary/ST_AsWKB - Erweiterung: 2.1.0 outasin hinzugefügt Gibt die Well-known-Binary (WKB) Darstellung eines Rasters zurück.
- ST_AsGML - Erweiterung: 2.1.0 Für GML 3 wurde die Unterstützung einer ID eingeführt. Gibt die Geometrie als GML-Element - Version 2 oder 3 - zurück.
- ST_Aspect - Erweiterung: 2.1.0 Verwendet ST_MapAlgebra() und der optionale Funktionsparameter interpolate_nodata wurde hinzugefügt Gibt die Exposition (standardmäßig in Grad) eines Rasterbandes mit Höhen aus. Nützlich für Terrain-Analysen.
- ST_Boundary - Erweiterung: mit 2.1.0 wurde die Unterstützung von Dreiecken eingeführt Gibt die abgeschlossene Hülle aus der kombinierten Begrenzung der Geometrie zurück.
- ST_Clip - Erweiterung: 2.1.0 neu geschrieben in C Gibt den nach der Eingabegeometrie zugeschnittenen Raster zurück. Falls die Bandnummer nicht angegeben ist, werden alle Bänder verarbeitet. Wenn crop nicht angegeben oder auf TRUE gesetzt ist, wird der Ausgaberaster entsprechend zugeschnitten. Wenn touched auf TRUE gesetzt ist, werden auch berührte Pixel einbezogen, andernfalls nur Pixel, bei denen sich der Mittelpunkt im Inneren der Geometrie befindet.
- ST_DWithin - Erweitert: 2.1.0 verbesserte die Geschwindigkeit für Geographie. Weitere Details finden Sie unter Making Geography faster. Prüft, ob zwei Geometrien innerhalb eines bestimmten Abstands liegen
- ST_DWithin - Verbessert: 2.1.0 Unterstützung für gekrümmte Geometrien wurde eingeführt. Prüft, ob zwei Geometrien innerhalb eines bestimmten Abstands liegen
- ST_Distance - Erweitert: 2.1.0 verbesserte die Geschwindigkeit für Geographie. Weitere Details finden Sie unter Making Geography faster. Gibt die größte 3-dimensionale Distanz zwischen zwei geometrischen Objekten als Linie zurück
- ST_Distance - Erweiterung: 2.1.0 - Unterstützung für Kurven beim geometrischen Datentyp eingeführt. Gibt die größte 3-dimensionale Distanz zwischen zwei geometrischen Objekten als Linie zurück
- ST_Distinct4ma - Erweiterung: 2.1.0 Variante 2 wurde hinzugefügt Funktion zur Rasterdatenverarbeitung, welche die Anzahl der einzelnen Pixelwerte in der Nachbarschaft errechnet.
- ST_DumpPoints - Verbessert: 2.1.0 Höhere Geschwindigkeit. Reimplementiert als natives C. Gibt eine Zusammenfassung des Inhalts einer Geometrie wieder.
- ST_HillShade - Erweiterung: 2.1.0 Verwendet ST_MapAlgebra() und der optionale Funktionsparameter interpolate_nodata wurde hinzugefügt Gibt für gegebenen Horizontalwinkel, Höhenwinkel, Helligkeit und Maßstabsverhältnis die hypothetische Beleuchtung eines Höhenrasterbandes zurück.
- ST_MakeValid - Verbessert: 2.1.0, Unterstützung für GEOMETRYCOLLECTION und MULTIPOINT hinzugefügt. Versucht, eine ungültige Geometrie gültig zu machen, ohne dass Scheitelpunkte verloren gehen.
- ST_Max4ma - Erweiterung: 2.1.0 Variante 2 wurde hinzugefügt Funktion zur Rasterdatenverarbeitung, die den maximalen Zellwert in der Nachbarschaft eines Pixel errechnet.
- ST_Mean4ma - Erweiterung: 2.1.0 Variante 2 wurde hinzugefügt Funktion zur Rasterdatenverarbeitung, die den mittleren Zellwert in der Nachbarschaft von Pixel errechnet.

- ST_Min4ma - Erweiterung: 2.1.0 Variante 2 wurde hinzugefügt Funktion zur Rasterdatenverarbeitung, die den minimalen Zellwert in der Nachbarschaft von Pixel errechnet.
- ST_PixelAsPolygons - Erweiterung: 2.1.0 Der optionale Übergabewert "exclude_nodata_value" wurde hinzugefügt. Gibt die umhüllende Polygongeometrie, den Zellwert, sowie die X- und Y-Rasterkoordinate für jedes Pixel aus.
- ST_Polygon - Erweiterung: 2.1.0 Geschwindigkeit verbessert (zur Gänze C-basiert) und es wird sichergestellt, dass das zurückgegebenen Mehrfachpolygon valide ist. Gibt eine Geometrie mit Mehrfachpolygonen zurück, die aus der Vereinigung von Pixel mit demselben Zellwert gebildet werden. Pixel mit NODATA Werten werden nicht berücksichtigt. Wenn keine Band angegeben ist, wird die Bandnummer standardmäßig auf 1 gesetzt.
- ST_Range4ma - Erweiterung: 2.1.0 Variante 2 wurde hinzugefügt Funktion zur Rasterdatenverarbeitung, die den Wertebereich der Pixel in einer Nachbarschaft errechnet.
- ST_SameAlignment - Erweiterung: 2.1.0 die Variante mit der Aggregatfunktion hinzugefügt Gibt TRUE zurück, wenn die Raster die selbe Rotation, Skalierung, Koordinatenreferenzsystem und Versatz (Pixel können auf dasselbe Gitter gelegt werden, ohne dass die Gitterlinien durch die Pixel schneiden) aufweisen. Wenn nicht, wird FALSE und eine Bechreibung des Problems ausgegeben.
- ST_Segmentize - Erweiterung: mit 2.1.0 wurde die Unterstützung des geographischen Datentyps eingeführt. Gibt eine geänderte Geometrie/Geografie zurück, bei der kein Segment länger als eine bestimmte Entfernung ist.
- ST_SetGeoReference - Erweiterung: 2.1.0 ST_SetGeoReference(raster, double precision, ...) Variante hinzugefügt Georeferenziert einen Raster über 6 Parameter in einem einzigen Aufruf. Die Zahlen müssen durch Leerzeichen getrennt sein. Die Funktion akzeptiert die Eingabe im Format von 'GDAL' und von 'ESRI'. Der Standardwert ist GDAL.
- ST_SetValue - Erweiterung: 2.1.0 Die geometrische Variante von ST_SetValue() unterstützt nun jeden geometrischen Datentyp, nicht nur POINT. Die geometrische Variante ist ein Adapter für die geomval[] Variante von ST_SetValues() Setzt den Wert für ein Pixel eines Bandes, das über columnx und rowy festgelegt wird, oder für die Pixel die eine bestimmte Geometrie schneiden, und gibt den veränderten Raster zurück. Die Bandnummerierung beginnt mit 1; wenn die Bandnummer nicht angegeben ist, wird 1 angenommen.
- ST_Slope - Erweiterung: 2.1.0 Verwendet ST_MapAlgebra() und es wurden die optionalen Funktionsparameter units, scale und interpolate_nodata hinzugefügt Gibt die Neigung (standardmäßig in Grad) eines Höhenrasterbandes zurück. Nützlich für Terrain-Analysen.
- ST_StdDev4ma - Erweiterung: 2.1.0 Variante 2 wurde hinzugefügt Funktion zur Rasterdatenverarbeitung, welche die Standardabweichung der Zellwerte in der Nachbarschaft von Pixel errechnet.
- ST_Sum4ma - Erweiterung: 2.1.0 Variante 2 wurde hinzugefügt Funktion zur Rasterdatenverarbeitung, die die Summe aller Zellwerte in der Nachbarschaft von Pixel errechnet.
- ST_Summary - Erweiterung: 2.1.0 S-Flag, diese zeigt an ob das Koordinatenreferenzsystem bekannt ist Gibt eine Zusammenfassung des Inhalts einer Geometrie wieder.
- ST_Transform - Erweiterung: 2.1.0 Variante ST_Transform(rast, alignto) hinzugefügt Projiziert einen Raster von einem bekannten Koordinatenreferenzsystem in ein anderes bekanntes Koordinatenreferenzsystem um. Die Optionen für die Skalierung sind NearestNeighbor, Bilinear, Cubisch, CubicSpline und der Lanczos-Filter, die Standardeinstellung ist NearestNeighbor.
- ST_Union - Erweiterung: 2.1.0 Geschwindigkeit verbessert (zur Gänze C-basiert) Gibt die Vereinigung mehrerer Rasterkacheln in einem einzelnen Raster mit mehreren Bändern zurück.
- ST_Union - Erweiterung: 2.1.0 ST_Union(rast) (Variante 1) vereinigt alle Bänder aller Ausgangsraster. Vorherige Versionen von PostGIS setzten das erste Band voraus. Gibt die Vereinigung mehrerer Rasterkacheln in einem einzelnen Raster mit mehreren Bändern zurück.
- ST_Union - Erweiterung: 2.1.0 ST_Union(rast, uniontype) (Variante 4) vereinigt alle Bänder aller Ausgangsraster. Gibt die Vereinigung mehrerer Rasterkacheln in einem einzelnen Raster mit mehreren Bändern zurück.
- toTopoGeom - Erweiterung: 2.1.0 die Version, welche eine bestehende TopoGeometry entgegennimmt, wurde hinzugefügt. Wandelt eine einfache Geometrie in eine TopoGeometry um.

Geänderte Funktionen in PostGIS 2.1

- ST_Aspect - Änderung: 2.1.0 In Vorgängerversionen wurden die Werte in Radiant ausgegeben. Nun werden die Werte standardmäßig in Grad ausgegeben. Gibt die Exposition (standardmäßig in Grad) eines Rasterbandes mit Höhen aus. Nützlich für Terrain-Analysen.
- ST_EstimatedExtent - Geändert: 2.1.0. Bis zu 2.0.x hieß dies ST_Estimated_Extent. Gibt die geschätzte Ausdehnung einer räumlichen Tabelle zurück.
- ST_Force2D - Änderung: 2.1.0. Bis zu 2.0.x wurde diese Funktion mit ST_Force_2D bezeichnet. Die Geometrien in einen "2-dimensionalen Modus" zwingen.
- ST_Force3D - Änderung: 2.1.0. Bis zu 2.0.x wurde diese Funktion mit ST_Force_3D bezeichnet. Zwingt die Geometrien in einen XYZ Modus. Dies ist ein Alias für ST_Force3DZ.
- ST_Force3DM - Änderung: 2.1.0. Bis zu 2.0.x wurde diese Funktion mit ST_Force_3DM bezeichnet. Zwingt die Geometrien in einen XYM Modus.
- ST_Force3DZ - Änderung: 2.1.0. Bis zu 2.0.x wurde diese Funktion mit ST_Force_3DZ bezeichnet. Zwingt die Geometrien in einen XYZ Modus.
- ST_Force4D - Änderung: 2.1.0. Bis zu 2.0.x wurde diese Funktion mit ST_Force_4D bezeichnet. Zwingt die Geometrien in einen XYZM Modus.
- ST_ForceCollection - Änderung: 2.1.0. Bis zu 2.0.x wurde diese Funktion mit ST_Force_Collection bezeichnet. Wandelt eine Geometrie in eine GEOMETRYCOLLECTION um.
- ST_HillShade - Änderung: 2.1.0 In Vorgängerversionen wurden Richtungswinkel und Höhenwinkel in Radiant angegeben. Nun werden die Werte in Grad ausgedrückt. Gibt für gegebenen Horizontalwinkel, Höhenwinkel, Helligkeit und Maßstabsverhältnis die hypothetische Beleuchtung eines Höhenrasterbandes zurück.
- ST_LineInterpolatePoint - Änderung: 2.1.0. Bis zu 2.0.x wurde diese Funktion mit ST_Line_Interpolate_Point bezeichnet. Gibt einen oder mehrere, entlang einer Linie interpolierte Punkte zurück.
- ST_LineLocatePoint - Änderung: 2.1.0. Bis zu 2.0.x wurde diese Funktion mit ST_Line_Locate_Point bezeichnet. Liefert die gebrochene Position des Punktes auf einer Linie, der einem Punkt am nächsten liegt.
- ST_LineSubstring - Änderung: 2.1.0. Bis zu 2.0.x wurde diese Funktion mit ST_Line_Substring bezeichnet. Gibt den Teil einer Linie zwischen zwei gebrochenen Stellen zurück.
- ST_PixelAsCentroids - Änderung: 2.1.1 Die Verhaltensweise von "exclude_nodata_value" wurde geändert. Gibt den geometrischen Schwerpunkt (Punktgeometrie) für jedes Pixel des Rasterbandes zurück, zusammen mit dem Zellwert und den X- und Y-Rasterkoordinaten eines jeden Pixels. Die Koordinaten der Punkte entsprechen dem geometrischen Schwerpunkt der Pixel.
- ST_PixelAsPoints - Änderung: 2.1.1 Die Verhaltensweise von "exclude_nodata_value" wurde geändert. Gibt eine Punktgeometrie für jedes Pixel des Rasterbandes zurück, zusammen mit dem Zellwert und den X- und Y-Rasterkoordinaten eines jeden Pixels. Die Koordinaten der Punkte entsprechen dem oberen linken Eck der Pixel.
- ST_PixelAsPolygons - Änderung: 2.1.1 Die Verhaltensweise von "exclude_nodata_value" wurde geändert. Gibt die umhüllende Polygongeometrie, den Zellwert, sowie die X- und Y-Rasterkoordinate für jedes Pixel aus.
- ST_Polygon - Änderung: 2.1.0 In Vorgängerversionen wurde manchmal ein Polygon zurückgegeben; dies wurde geändert so dass jetzt immer ein Mehrfachpolygon zurückgegeben wird. Gibt eine Geometrie mit Mehrfachpolygonen zurück, die aus der Vereinigung von Pixel mit demselben Zellwert gebildet werden. Pixel mit NODATA Werten werden nicht berücksichtigt. Wenn keine Band angegeben ist, wird die Bandnummer standardmäßig auf 1 gesetzt.
- ST_RasterToWorldCoordX - Änderung: 2.1.0 Vorgängerversionen haben dies als "ST_Raster2WorldCoordX" bezeichnet. Gibt die geodätische X Koordinate links oberhalb des Rasters, der Spalte und der Zeile aus. Die Nummerierung der Spalten und Zeilen beginnt mit 1.
- ST_RasterToWorldCoordY - Änderung: 2.1.0 In Vorgängerversionen wurde dies als ST_Raster2WorldCoordY bezeichnet Gibt die geodätische Y Koordinate links oberhalb des Rasters, der Spalte und der Zeile aus. Die Nummerierung der Spalten und Zeilen beginnt mit 1.

- ST_Rescale - Änderung: 2.1.0 Funktioniert jetzt auch mit Raster ohne SRID Neuabtastung eines Rasters, indem nur die Skala (oder Pixelgröße) angepasst wird. Die neuen Pixelwerte werden mit den Algorithmen NearestNeighbor (englische oder amerikanische Schreibweise), Bilinear, Cubic, CubicSpline, Lanczos, Max oder Min resampling berechnet. Die Voreinstellung ist NearestNeighbor.
- ST_Reskew - Änderung: 2.1.0 Funktioniert jetzt auch mit Raster ohne SRID Skaliert einen Raster, indem lediglich der Versatz (oder Rotationsparameter) angepasst wird. Neue Pixelwerte werden über NearestNeighbor, bilinear, kubisch, CubicSpline oder mit dem Lanczos-Filter errechnet. Die Standardeinstellung ist NearestNeighbor.
- ST_Segmentize - Geändert: 2.1.0 Infolge der Einführung der Geographie-Unterstützung verursacht die Verwendung ST_Segmentize('I 2, 3 4)', 0.5) einen mehrdeutigen Funktionsfehler. Die Eingabe muss korrekt als Geometrie oder Geografie eingegeben werden. Verwenden Sie ST_GeomFromText, ST_GeogFromText oder eine Umwandlung in den erforderlichen Typ (z.B. ST_Segmentize('LINE 2, 3 4)'::geometry, 0.5)) Gibt eine geänderte Geometrie/Geografie zurück, bei der kein Segment länger als eine bestimmte Entfernung ist.
- ST_Slope - Änderung: 2.1.0 In Vorgängerversionen wurden die Werte in Radiant ausgegeben. Nun werden die Werte standardmäßig in Grad ausgegeben. Gibt die Neigung (standardmäßig in Grad) eines Höhenrasterbandes zurück. Nützlich für Terrain-Analysen.
- ST_SnapToGrid - Änderung: 2.1.0 Funktioniert jetzt auch mit Raster ohne SRID Skaliert einen Raster durch Fangen an einem Führungsgitter. Neue Pixelwerte werden über NearestNeighbor, bilinear, kubisch, CubicSpline oder mit dem Lanczos-Filter errechnet. Die Standardeinstellung ist NearestNeighbor.
- ST_WorldToRasterCoordX - Änderung: 2.1.0 In Vorgängerversionen wurde dies als ST_World2RasterCoordX bezeichnet Gibt für eine Punktgeometrie (pt) oder eine globale X- und Y-Koordinate (xw, yw) die Rasterspalte im globalen Koordinatenreferenzsystem des Rasters aus.
- ST_WorldToRasterCoordY - Änderung: 2.1.0 In Vorgängerversionen wurde dies als ST_World2RasterCoordY bezeichnet Gibt für eine Punktgeometrie (pt) oder eine globale X- und Y-Koordinate (xw, yw) die Rasterzeile im globalen Koordinatenreferenzsystem des Rasters aus.

13.12.14 PostGIS-Funktionen neu oder erweitert in 2.0

Die unten aufgeführten Funktionen sind PostGIS-Funktionen, die hinzugefügt oder erweitert wurden.

Neue Funktionen in PostGIS 2.0

- && - Verfügbarkeit: 2.0.0 Gibt TRUE zurück, wenn das umschreibende Rechteck von A das umschreibende Rechteck von B schneidet.
- &&& - Verfügbarkeit: 2.0.0 Gibt TRUE zurück, wenn A's n-D bounding box B's n-D bounding box schneidet.
- <#> - Availability: 2.0.0. Gibt die 2D Entfernung zwischen den Bounding Boxes von A und B zurück
- <-> - 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. Gibt die 2D Entfernung zwischen A und B zurück.
- @ - Verfügbarkeit: 2.0.0 raster @ raster, und raster @ geometry eingeführt Gibt TRUE zurück, wenn das umschreibende Rechteck von A in jenem von B enthalten ist. Das umschreibende Rechteck ist in Double Precision.
- @ - Verfügbarkeit: 2.0.5 "geometry @ raster" eingeführt Gibt TRUE zurück, wenn das umschreibende Rechteck von A in jenem von B enthalten ist. Das umschreibende Rechteck ist in Double Precision.
- AddEdge - Verfügbarkeit: 2.0.0 Fügt die Kante eines Linienzugs in der Tabelle "edge", und die zugehörigen Anfangs- und Endpunkte in die Knotenpunkttabelle, des jeweiligen topologischen Schemas ein. Dabei wird die übergebene Linienzuggeometrie verwendet und die edgeid der neuen (oder bestehenden) Kante ausgegeben.
- AddFace - Verfügbarkeit: 2.0.0 Registriert die Elementarstruktur einer Masche in einer Topologie und gibt den Identifikator der Masche aus.

- AddNode - Verfügbarkeit: 2.0.0 Fügt einen Knotenpunkt zu der Tabelle "node" in dem vorgegebenen topologischen Schema hinzu und gibt die "nodeid" des neuen Knotens aus. Falls der Punkt bereits als Knoten existiert, wird die vorhandene nodeid zurückgegben.
- AddOverviewConstraints - Verfügbarkeit: 2.0.0 Eine Rasterspalte als Übersicht für eine andere Rasterspalte kennzeichnen.
- AddRasterConstraints - Verfügbarkeit: 2.0.0 Fügt die Raster-Constraints zu einer bestimmten Spalte einer bereits geladenen Rastertabelle hinzu. Diese Constraints beschränken das Koordinatentransformationssystem, den Maßstab, die Blockgröße, die Ausrichtung, die Bänder, den Bandtyp und eine Flag, die anzeigt ob die Rasterspalte regelmäßig geblockt ist. Es müssen bereits Daten in die Tabelle geladen sein, damit die Constraints abgeleitet werden können. Gibt TRUE zurück, wenn das Setzen der Constraints ausgeführt wurde; bei Problemen wird eine Meldung angezeigt.
- AsGML - Verfügbarkeit: 2.0.0 Gibt die GML-Darstellung einer TopoGeometry zurück.
- CopyTopology - Verfügbarkeit: 2.0.0 Erstellt eine Kopie einer Topologie (Knoten, Kanten, Flächen, Ebenen und TopoGeometrien) in ein neues Schema
- DropOverviewConstraints - Verfügbarkeit: 2.0.0 Löscht die Markierung einer Rasterspalte, die festlegt dass sie als Übersicht für eine andere Spalte dient.
- DropRasterConstraints - Verfügbarkeit: 2.0.0 Löscht die Constraints eines PostGIS Rasters die sich auf eine Rastertabellenspalte beziehen. Nützlich um Daten erneut zu laden oder um die Daten einer Rasterspalte zu aktualisieren.
- GetEdgeByPoint - Verfügbarkeit: 2.0.0 Findet die edge-id einer Kante die einen gegebenen Punkt schneidet.
- GetFaceByPoint - Verfügbarkeit: 2.0.0 Findet eine Fläche, die einen bestimmten Punkt schneidet.
- GetNodeByPoint - Verfügbarkeit: 2.0.0 Findet zu der Lage eines Punktes die node-id eines Knotens.
- GetNodeEdges - Verfügbarkeit: 2.0 Gibt für einen Knoten die sortierte Menge der einfallenden Kanten aus.
- GetRingEdges - Verfügbarkeit: 2.0.0 Gibt eine sortierte Liste von mit Vorzeichen versehenen Identifikatoren der Kanten zurück, die angetroffen werden, wenn man an der Seite der gegebenen Kante entlangwandert.
- GetTopoGeomElements - Verfügbarkeit: 2.0.0 Gibt für eine TopoGeometry (Elementarstrukturen) einen Satz an topoelement Objekten zurück, welche die topologische element_id und den element_type beinhalten.
- GetTopologySRID - Verfügbarkeit: 2.0.0 Gibt für den Namen einer Topologie, die SRID der Topologie in der Tabelle "topology.topology" aus.
- Polygonize - Verfügbarkeit: 2.0.0 Findet und registriert alle Maschen, die durch die Kanten der Topologie festgelegt sind.
- ST_3DClosestPoint - Verfügbarkeit: 2.0.0 Gibt den 3-dimensionalen Punkt auf g1 zurück, der den kürzesten Abstand zu g2 hat. Dies ist der Anfangspunkt des kürzesten Abstands in 3D.
- ST_3DDFullyWithin - Verfügbarkeit: 2.0.0 Prüft, ob zwei 3D-Geometrien vollständig innerhalb eines bestimmten 3D-Abstands liegen
- ST_3DDWithin - Verfügbarkeit: 2.0.0 Prüft, ob zwei 3D-Geometrien innerhalb eines bestimmten 3D-Abstands liegen
- ST_3DDistance - Verfügbarkeit: 2.0.0 Für den geometrischen Datentyp. Es wird der geringste 3-dimensionale kartesische Abstand (basierend auf dem Koordinatenreferenzsystem) zwischen zwei geometrischen Objekten in projizierten Einheiten zurückgegeben.
- ST_3DIntersects - Verfügbarkeit: 2.0.0 Prüft, ob sich zwei Geometrien in 3D räumlich schneiden - nur für Punkte, Linienzüge, Polygone, polyedrische Flächen (Bereich)
- ST_3DLongestLine - Verfügbarkeit: 2.0.0 Gibt die größte 3-dimensionale Distanz zwischen zwei geometrischen Objekten als Linie zurück
- ST_3DMaxDistance - Verfügbarkeit: 2.0.0 Für den geometrischen Datentyp. Gibt die maximale 3-dimensionale kartesische Distanz (basierend auf dem Koordinatenreferenzsystem) zwischen zwei geometrischen Objekten in projizierten Einheiten zurück.

- ST_3DShortestLine - Verfügbarkeit: 2.0.0 Gibt den kürzesten 3-dimensionalen Abstand zwischen zwei geometrischen Objekten als Linie zurück
- ST_AddEdgeModFace - Verfügbarkeit: 2.0 Fügt eine Kante hinzu. Falls dabei eine Masche aufgetrennt wird, so wird die ursprüngliche Masche angepasst und eine weitere Masche hinzugefügt.
- ST_AddEdgeNewFaces - Verfügbarkeit: 2.0 Fügt eine Kante hinzu. Falls dabei eine Masche aufgetrennt wird, so wird die ursprüngliche Masche gelöscht und durch zwei neue Maschen ersetzt.
- ST_AsGDALRaster - Verfügbarkeit: 2.0.0 - GDAL >= 1.6.0. Gibt die Rasterkachel in dem ausgewiesenen Rasterformat von GDAL aus. Sie können jedes Rasterformat angeben, das von Ihrer Bibliothek unterstützt wird. Um eine Liste mit den unterstützten Formaten auszugeben, verwenden Sie bitte ST_GDALDrivers().
- ST_AsJPEG - Verfügbarkeit: 2.0.0 - GDAL >= 1.6.0. Gibt die ausgewählten Bänder der Rasterkachel als einzelnes Bild (Byte-Array) im Format "Joint Photographic Exports Group" (JPEG) aus. Wenn kein Band angegeben ist und 1 oder mehr als 3 Bänder ausgewählt wurden, dann wird nur das erste Band verwendet. Wenn 3 Bänder ausgewählt wurden, werden alle 3 Bänder verwendet und auf RGB abgebildet.
- ST_AsLatLonText - Verfügbarkeit: 2.0 Gibt die "Grad, Minuten, Sekunden"-Darstellung für den angegebenen Punkt aus.
- ST_AsPNG - Verfügbarkeit: 2.0.0 - GDAL >= 1.6.0. Gibt die ausgewählten Bänder der Rasterkachel als einzelnes, übertragbares Netzwerkgraphik (PNG) Bild (Byte-Feld) aus. Wenn der Raster 1,3 oder 4 Bänder hat und keine Bänder angegeben sind, dann werden alle Bänder verwendet. Wenn der Raster 2 oder mehr als 4 Bänder hat und keine Bänder angegeben sind, dann wird nur Band 1 verwendet. Die Bänder werden in den RGB- oder den RGBA-Raum abgebildet.
- ST_AsRaster - Verfügbarkeit: 2.0.0 - GDAL >= 1.6.0. Konvertiert den geometrischen Datentyp von PostGIS in einen PostGIS Raster.
- ST_AsTIFF - Verfügbarkeit: 2.0.0 - GDAL >= 1.6.0. Gibt die ausgewählten Bänder des Raster als einzelnes TIFF Bild (Byte-Feld) zurück. Wenn kein Band angegeben ist oder keines der angegebenen Bänder im Raster existiert, werden alle Bänder verwendet.
- ST_AsX3D - Verfügbarkeit: 2.0.0: ISO-IEC-19776-1.2-X3DEncodings-XML Gibt eine Geometrie im X3D XML Knotenelement-Format zurück: ISO-IEC-19776-1.2-X3DEncodings-XML
- ST_Aspect - Verfügbarkeit: 2.0.0 Gibt die Exposition (standardmäßig in Grad) eines Rasterbandes mit Höhen aus. Nützlich für Terrain-Analysen.
- ST_Band - Verfügbarkeit: 2.0.0 Gibt einen oder mehrere Bänder eines bestehenden Rasters als neuen Raster aus. Nützlich um neue Raster aus bestehenden Rastern abzuleiten.
- ST_BandIsNoData - Verfügbarkeit: 2.0.0 Gibt TRUE aus, wenn das Band ausschließlich aus NODATA Werten besteht.
- ST_Clip - Verfügbarkeit: 2.0.0 Gibt den nach der Eingabegeometrie zugeschnittenen Raster zurück. Falls die Bandnummer nicht angegeben ist, werden alle Bänder verarbeitet. Wenn crop nicht angegeben oder auf TRUE gesetzt ist, wird der Ausgaberaster entsprechend zugeschnitten. Wenn touched auf TRUE gesetzt ist, werden auch berührte Pixel einbezogen, andernfalls nur Pixel, bei denen sich der Mittelpunkt im Inneren der Geometrie befindet.
- ST_CollectionHomogenize - Verfügbarkeit: 2.0.0 Gibt die einfachste Darstellung einer Geometriesammlung zurück.
- ST_ConcaveHull - Verfügbarkeit: 2.0.0 Berechnet eine möglicherweise konkave Geometrie, die alle Eckpunkte der Eingabegeometrie enthält
- ST_Count - Verfügbarkeit: 2.0.0 Gibt die Anzahl der Pixel für ein Band eines Rasters oder eines Raster-Coverage zurück. Wenn kein Band angegeben ist, wird standardmäßig Band 1 gewählt. Wenn der Parameter "exclude_nodata_value" auf TRUE gesetzt ist, werden nur Pixel mit Werten ungleich NODATA gezählt.
- ST_CreateTopoGeo - Verfügbarkeit: 2.0 Fügt eine Sammlung von Geometrien an eine leere Topologie an und gibt eine Bestätigungsmeldung aus.
- ST_Distinct4ma - Verfügbarkeit: 2.0.0 Funktion zur Rasterdatenverarbeitung, welche die Anzahl der einzelnen Pixelwerte in der Nachbarschaft errechnet.
- ST_FlipCoordinates - Verfügbarkeit: 2.0.0 Gibt eine Version einer Geometrie mit gespiegelter X- und Y-Achse zurück.

- ST_GDALDrivers - Verfügbarkeit: 2.0.0 - GDAL >= 1.6.0. Gibt eine Liste der Rasterformate aus, die von PostGIS über die Bibliothek GDAL unterstützt werden. Nur die Formate mit can_write=True können von ST_AsGDALRaster verwendet werden.
- ST_GeomFromGeoJSON - Verfügbarkeit: 2.0.0 benötigt - JSON-C >= 0.9 Nimmt als Eingabe eine GeoJSON-Darstellung der Geometrie und gibt ein geometrisches PostGIS-Objekt aus.
- ST_GetFaceEdges - Verfügbarkeit: 2.0 Gibt die Kanten, die aface begrenzen, sortiert aus.
- ST_HasNoBand - Verfügbarkeit: 2.0.0 Gibt TRUE aus, wenn kein Band mit der angegebenen Bandnummer existiert.Gibt den Pixeltyp des angegebenen Bandes aus. Wenn keine Bandnummer angegeben ist, wird das 1ste Band angenommen.
- ST_HillShade - Verfügbarkeit: 2.0.0 Gibt für gegebenen Horizontalwinkel, Höhenwinkel, Helligkeit und Maßstabsverhältnis die hypothetische Beleuchtung eines Höhenrasterbandes zurück.
- ST_Histogram - Verfügbarkeit: 2.0.0 Gibt Datensätze aus, welche die Verteilung der Daten eines Rasters oder eines Rastercoverage darstellen. Dabei wird die Wertemenge in Klassen aufgeteilt und für jede Klasse zusammengefasst. Wenn die Anzahl der Klassen nicht angegeben ist, wird sie automatisch berechnet.
- ST_InterpolatePoint - Verfügbarkeit: 2.0.0 Für einen gegebenen Punkt wird die Kilometrierung auf dem nächstliegenden Punkt einer Geometrie zurück.
- ST_IsEmpty - Verfügbarkeit: 2.0.0 Gibt TRUE zurück, wenn der Raster leer ist (width = 0 and height = 0). Andernfalls wird FALSE zurückgegeben.
- ST_IsValidDetail - Verfügbarkeit: 2.0.0 Gibt eine Zeile valid_detail zurück, die angibt, ob eine Geometrie gültig ist oder, falls nicht, einen Grund und einen Ort.
- ST_IsValidReason - Verfügbarkeit: Version 2.0 mit Flaggen. Gibt einen Text zurück, der angibt, ob eine Geometrie gültig ist, oder einen Grund für die Ungültigkeit.
- ST_MakeLine - Verfügbarkeit: 2.0.0 - Unterstützung zur Eingabe von LineString Elementen eingeführt Erzeugt einen Linienzug aus einer Punkt-, Mehrfachpunkt- oder Liniengeometrie.
- ST_MakeValid - Verfügbarkeit: 2.0.0 Versucht, eine ungültige Geometrie gültig zu machen, ohne dass Scheitelpunkte verloren gehen.
- ST_MapAlgebraExpr - Verfügbarkeit: 2.0.0 Version mit 1 Rasterband: Erzeugt ein neues Rasterband, dass über eine gültige, algebraische PostgreSQL Operation für ein Rasterband mit gegebenen Pixeltyp erstellt wird. Wenn kein Band bestimmt ist, wird Band 1 angenommen.
- ST_MapAlgebraExpr - Verfügbarkeit: 2.0.0 Version mit 2 Rasterbändern: Erstellt einen neuen Einzelbandraster, indem eine gültige algebraische PostgreSQL Funktion auf die zwei Ausgangsrasterbänder und den entsprechenden Pixeltyp angewendet wird. Wenn keine Bandnummern angegeben sind, wird von jedem Raste Band 1 angenommen. Der Ergebnisraster wird nach dem Gitter des ersten Raster ausgerichtet (Skalierung, Versatz und Eckpunkte der Pixel) und hat die Ausdehnung, welche durch den Parameter "extenttype" definiert ist. Der Parameter "extenttype" kann die Werte INTERSECTION, UNION, FIRST, SECOND annehmen.
- ST_MapAlgebraFct - Verfügbarkeit: 2.0.0 Version mit 1 Rasterband: Erzeugt ein neues Rasterband, dass über eine gültige PostgreSQL Funktion für ein gegebenes Rasterband und Pixeltyp erstellt wird. Wenn kein Band bestimmt ist, wird Band 1 angenommen.
- ST_MapAlgebraFct - Verfügbarkeit: 2.0.0 Version mit 2 Rasterbändern: Erstellt einen neuen Einzelbandraster, indem eine gültige PostgreSQL Funktion auf die 2 gegebenen Rasterbänder und den entsprechenden Pixeltyp angewendet wird. Wenn kein Band bestimmt ist, wird Band 1 angenommen. Wenn der "Extent"-Typ nicht angegeben ist, wird standardmäßig INTERSECTION angenommen.
- ST_MapAlgebraFctNgb - Verfügbarkeit: 2.0.0 Version mit 1em Band: Map Algebra Nearest Neighbor mit einer benutzerdefinierten PostgreSQL Funktion. Gibt einen Raster zurück, dessen Werte sich aus einer benutzerdefinierte PL/pgsql Funktion ergeben, welche die Nachbarschaftswerte des Ausgangsrasterbandes einbezieht.
- ST_Max4ma - Verfügbarkeit: 2.0.0 Funktion zur Rasterdatenverarbeitung, die den maximalen Zellwert in der Nachbarschaft eines Pixel errechnet.

- ST_Mean4ma - Verfügbarkeit: 2.0.0 Funktion zur Rasterdatenverarbeitung, die den mittleren Zellwert in der Nachbarschaft von Pixel errechnet.
- ST_Min4ma - Verfügbarkeit: 2.0.0 Funktion zur Rasterdatenverarbeitung, die den minimalen Zellwert in der Nachbarschaft von Pixel errechnet.
- ST_ModEdgeHeal - Verfügbarkeit: 2.0 "Heilt" zwei Kanten, indem der verbindende Knoten gelöscht wird, die erste Kante modifiziert und die zweite Kante gelöscht wird. Gibt die ID des gelöschten Knoten zurück.
- ST_MoveIsoNode - Verfügbarkeit: 2.0.0 Verschiebt einen isolierten Knoten in einer Topologie von einer Stelle an eine andere. Falls die neue Geometrie apoint bereits als Knoten existiert, wird eine Fehlermeldung ausgegeben. Gibt eine Beschreibung der Verschiebung aus.
- ST_NewEdgeHeal - Verfügbarkeit: 2.0 "Heilt" zwei Kanten, indem der verbindende Knoten und beide Kanten gelöscht werden. Die beiden Kanten werden durch eine Kante ersetzt, welche dieselbe Ausichtung wie die erste Kante hat.
- ST_Node - Verfügbarkeit: 2.0.0 Knoten eine Sammlung von Linien.
- ST_NumPatches - Verfügbarkeit: 2.0.0 Returns the number of faces in a PolyhedralSurface or TIN.
- ST_OffsetCurve - Verfügbarkeit: 2.0 Gibt eine versetzte Linie in einem bestimmten Abstand und einer bestimmten Seite von einer Eingabelinie zurück.
- ST_PatchN - Verfügbarkeit: 2.0.0 Returns the Nth face of a PolyhedralSurface or TIN.
- ST_Perimeter - Verfügbarkeit: Mit 2.0.0 wurde die Unterstützung für geograpischen Koordinaten eingeführt Gibt die Länge der Begrenzung einer polygonalen Geometrie oder Geografie zurück.
- ST_PixelAsPolygon - Verfügbarkeit: 2.0.0 Gibt die Polygongeometrie aus, die das Pixel einer bestimmten Zeile und Spalte begrenzt.
- ST_PixelAsPolygons - Verfügbarkeit: 2.0.0 Gibt die umhüllende Polygongeometrie, den Zellwert, sowie die X- und Y-Rasterkoordinate für jedes Pixel aus.
- ST_Project - Verfügbarkeit: 2.0.0 Gibt einen Punkt zurück, der von einem Startpunkt um eine bestimmte Entfernung und Peilung (Azimut) projiziert wird.
- ST_Quantile - Verfügbarkeit: 2.0.0 Berechnet die Quantile eines Rasters oder einer Rastercoverage Tabelle im Kontext von Stichproben oder Bevölkerung. Dadurch kann untersucht werden, ob ein Wert bei 25%, 50% oder 75% Perzentil des Rasters liegt.
- ST_Range4ma - Verfügbarkeit: 2.0.0 Funktion zur Rasterdatenverarbeitung, die den Wertebereich der Pixel in einer Nachbarschaft errechnet.
- ST_Reclass - Verfügbarkeit: 2.0.0 Erstellt einen neuen Raster, der aus neu klassifizierten Bändern des Originalraster besteht. Das Band "nband" ist jenes das verändert werden soll. Wenn "nband" nicht angegeben ist, wird "Band 1" angenommen. Alle anderen Bänder bleiben unverändert. Anwendungsfall: zwecks einfacherer Visualisierung ein 16BUI-Band in ein 8BUI-Band konvertieren und so weiter.
- ST_RelateMatch - Verfügbarkeit: 2.0.0 Testet, ob eine DE-9IM Schnittpunktmatrix mit einem Schnittpunktmuster übereinstimmt
- ST_RemEdgeModFace - Verfügbarkeit: 2.0 Entfernt eine Kante, und wenn die Kante zwei Flächen trennt, wird eine Fläche gelöscht und die andere Fläche so verändert, dass sie den Raum beider Flächen abdeckt.
- ST_RemEdgeNewFace - Verfügbarkeit: 2.0 Entfernt eine Kante. Falls die gelöschte Kante zwei Maschen voneinander getrennt hat, werden die ursprünglichen Maschen gelöscht und durch einer neuen Masche ersetzt.
- ST_Resample - Verfügbarkeit: 2.0.0 benötigt GDAL 1.6.1+ Skaliert einen Raster mit einem bestimmten Algorithmus, neuen Dimensionen, einer beliebigen Gitterecke und über Parameter zur Georeferenzierung des Rasters, die angegeben oder von einem anderen Raster übernommen werden können.
- ST_Rescale - Verfügbarkeit: 2.0.0 benötigt GDAL 1.6.1+ Neuabtastung eines Rasters, indem nur die Skala (oder Pixelgröße) angepasst wird. Die neuen Pixelwerte werden mit den Algorithmen NearestNeighbor (englische oder amerikanische Schreibweise), Bilinear, Cubic, CubicSpline, Lanczos, Max oder Min resampling berechnet. Die Voreinstellung ist NearestNeighbor.

- ST_Reskew - Verfügbarkeit: 2.0.0 benötigt GDAL 1.6.1+ Skaliert einen Raster, indem lediglich der Versatz (oder Rotationsparameter) angepasst wird. Neue Pixelwerte werden über NearestNeighbor, bilinear, kubisch, CubicSpline oder mit dem Lanczos-Filter errechnet. Die Standardeinstellung ist NearestNeighbor.
- ST_SameAlignment - Verfügbarkeit: 2.0.0 Gibt TRUE zurück, wenn die Raster die selbe Rotation, Skalierung, Koordinatenreferenzsystem und Versatz (Pixel können auf dasselbe Gitter gelegt werden, ohne dass die Gitterlinien durch die Pixel schneiden) aufweisen. Wenn nicht, wird FALSE und eine Bechreibung des Problems ausgegeben.
- ST_SetBandIsNoData - Verfügbarkeit: 2.0.0 Setzt die Flag "isnodata" für das Band auf TRUE.
- ST_SharedPaths - Verfügbarkeit: 2.0.0 Gibt eine Sammelgeometrie zurück, welche die gemeinsamen Strecken der beiden eingegebenen LineStrings/MultiLinestrings enthält.
- ST_Slope - Verfügbarkeit: 2.0.0 Gibt die Neigung (standardmäßig in Grad) eines Höhenrasterbandes zurück. Nützlich für Terrain-Analysen.
- ST_Snap - Verfügbarkeit: 2.0.0 Fängt die Segmente und Knoten einer Eingabegeometrie an den Knoten einer Referenzgeometrie.
- ST_SnapToGrid - Verfügbarkeit: 2.0.0 benötigt GDAL 1.6.1+ Skaliert einen Raster durch Fangen an einem Führungsgitter. Neue Pixelwerte werden über NearestNeighbor, bilinear, kubisch, CubicSpline oder mit dem Lanczos-Filter errechnet. Die Standardeinstellung ist NearestNeighbor.
- ST_Split - Verfügbarkeit: 2.0.0 erfordert GEOS Gibt eine Sammlung von Geometrien zurück, die durch Aufteilung einer Geometrie durch eine andere Geometrie entstanden sind.
- ST_StdDev4ma - Verfügbarkeit: 2.0.0 Funktion zur Rasterdatenverarbeitung, welche die Standardabweichung der Zellwerte in der Nachbarschaft von Pixel errechnet.
- ST_Sum4ma - Verfügbarkeit: 2.0.0 Funktion zur Rasterdatenverarbeitung, die die Summe aller Zellwerte in der Nachbarschaft von Pixel errechnet.
- ST_SummaryStats - Verfügbarkeit: 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 if no band is specified.
- ST_Transform - Verfügbarkeit: 2.0.0 benötigt GDAL 1.6.1+ Projiziert einen Raster von einem bekannten Koordinatenreferenzsystem in ein anderes bekanntes Koordinatenreferenzsystem um. Die Optionen für die Skalierung sind NearestNeighbor, Bilinear, Cubisch, CubicSpline und der Lanczos-Filter, die Standardeinstellung ist NearestNeighbor.
- ST_UnaryUnion - Verfügbarkeit: 2.0.0 Berechnet die Vereinigung der Komponenten einer einzelnen Geometrie.
- ST_Union - Verfügbarkeit: 2.0.0 Gibt die Vereinigung mehrerer Rasterkacheln in einem einzelnen Raster mit mehreren Bändern zurück.
- ST_ValueCount - Verfügbarkeit: 2.0.0 Gibt Datensätze aus, die den Zellwert und die Anzahl der Pixel eines Rasterbandes (oder Rastercoveragebandes) für gegebene Werte enthalten. Wenn kein Band angegeben ist, wird Band 1 angenommen. Pixel mit dem Wert NODATA werden standardmäßig nicht gezählt; alle anderen Pixelwerte des Bandes werden ausgegeben und auf die nächste Ganzzahl gerundet.
- TopoElementArray_Agg - Verfügbarkeit: 2.0.0 Gibt für eine Menge an element_id, type Feldern (topoelements) ein topoelementarray zurück.
- TopoGeo_AddLineString - Verfügbarkeit: 2.0.0 Fügt einen Linienzug unter Berücksichtigung eines Toleranzwerts zu einer Topologie hinzu , wobei Kanten/Maschen dabei aufgetrennt werden können.
- TopoGeo_AddPoint - Verfügbarkeit: 2.0.0 Fügt einen Punkt, unter Berücksichtigung einer Toleranz, an eine bestehende Topologie an. Existierende Kanten werden eventuell aufgetrennt.
- TopoGeo_AddPolygon - Verfügbarkeit: 2.0.0 Fügt ein Polygon unter Berücksichtigung eines Toleranzwertes zu einer bestehenden Topologie hinzu, wobei bestehende Kanten/Maschen aufgetrennt werden können. Gibt Maschenidentifikatoren zurück.
- TopologySummary - Verfügbarkeit: 2.0.0 Nimmt den Namen einer Topologie und liefert eine Zusammenfassung der Gesamtsummen der Typen und Objekte in der Topologie.

- toTopoGeom - Verfügbarkeit: 2.0 Wandelt eine einfache Geometrie in eine TopoGeometry um.
- ~ - Verfügbarkeit: 2.0.0 Gibt TRUE zurück, wenn das umschreibende Rechteck von A jenes von B enthält. Das umschreibende Rechteck ist in Double Precision.
- ~= - Verfügbarkeit: 2.0.0 Gibt TRUE zurück, wenn die bounding box von A ident mit jener von B ist.

Erweiterte Funktionen in PostGIS 2.0

- && - Erweiterung: Mit 2.0.0 wurde die Unterstützung für polyedrische Oberflächen eingeführt. Gibt TRUE zurück, wenn die 2D Bounding Box von A die 2D Bounding Box von B schneidet.
- AddGeometryColumn - Verbesserung: 2.0.0 use_typmod Argument eingeführt. Standardmäßig wird eine typmod Geometrie anstelle einer Constraint-basierten Geometrie erzeugt. Entfernt eine Geometriespalte aus einer räumlichen Tabelle.
- Box2D - Erweiterung: Mit 2.0.0 wurde die Unterstützung für polyedrische Oberflächen, Dreiecke und TIN eingeführt. Gibt ein BOX2D zurück, das die 2D-Ausdehnung einer Geometrie darstellt.
- Box3D - Erweiterung: Mit 2.0.0 wurde die Unterstützung für polyedrische Oberflächen, Dreiecke und TIN eingeführt. Gibt ein BOX3D zurück, das die 3D-Ausdehnung einer Geometrie darstellt.
- CreateTopology - Enhanced: 2.0 added support for hasZ. Erstellt ein neues Topologie-Schema und trägt es in die Tabelle topology.topology ein.
- GeometryType - Erweiterung: Mit 2.0.0 wurde die Unterstützung für polyedrische Oberflächen, Dreiecke und TIN eingeführt. Gibt den Geometrietyp des ST_Geometry Wertes zurück.
- Populate_Geometry_Columns - Erweiterung: 2.0.0 Der optionale Übergabewert use_typmod wurde eingeführt, um bestimmen zu können, ob die Spalten mit Typmodifikatoren oder mit Check-Constraints erstellt werden sollen. Stellt sicher, dass Geometriespalten mit Typmodifikatoren definiert sind oder geeignete räumliche Beschränkungen haben.
- ST_3DExtent - Erweiterung: Mit 2.0.0 wurde die Unterstützung für polyedrische Oberflächen, Dreiecke und TIN eingeführt. Aggregatfunktion, die den 3D-Begrenzungsrahmen von Geometrien zurückgibt.
- ST_Affine - Erweiterung: Mit 2.0.0 wurde die Unterstützung für polyedrische Oberflächen, Dreiecke und TIN eingeführt. Wenden Sie eine affine 3D-Transformation auf eine Geometrie an.
- ST_Area - Erweiterung: Mit 2.0.0 wurde 2D-Unterstützung für polyedrische Oberflächen eingeführt. Gibt den geometrischen Schwerpunkt einer Geometrie zurück.
- ST_AsBinary - Erweiterung: Mit 2.0.0 wurde die Unterstützung für polyedrische Oberflächen, Dreiecke und TIN eingeführt. Rückgabe der OGC/ISO Well-Known Binary (WKB)-Darstellung der Geometrie/Geografie ohne SRID-Metadaten.
- ST_AsBinary - Erweiterung: 2.0.0 - Unterstützung für höherdimensionale Koordinatensysteme eingeführt. Rückgabe der OGC/ISO Well-Known Binary (WKB)-Darstellung der Geometrie/Geografie ohne SRID-Metadaten.
- ST_AsBinary - Erweiterung: 2.0.0 Unterstützung zum Festlegen des Endian beim geographischen Datentyp eingeführt. Rückgabe der OGC/ISO Well-Known Binary (WKB)-Darstellung der Geometrie/Geografie ohne SRID-Metadaten.
- ST_AsEWKB - Erweiterung: Mit 2.0.0 wurde die Unterstützung für polyedrische Oberflächen, Dreiecke und TIN eingeführt. Rückgabe der Extended Well-Known Binary (EWKB) Darstellung der Geometrie mit SRID-Metadaten.
- ST_AsEWKT - Erweiterung: Mit 2.0.0 wurde die Unterstützung für den geographischen Datentyp, polyedrische Oberflächen, Dreiecke und TIN eingeführt. Gibt die Well-known-Text(WKT) Darstellung der Geometrie mit den SRID-Metadaten zurück.
- ST_AsGML - Erweiterung: 2.0.0 Unterstützung durch Präfix eingeführt. Für GML3 wurde die Option 4 eingeführt, um die Verwendung von LineString anstatt von Kurven für Linien zu erlauben. Ebenfalls wurde die GML3 Unterstützung für polyedrische Oberflächen und TINS eingeführt, sowie die Option 32 zur Ausgabe der BBox. Gibt die Geometrie als GML-Element - Version 2 oder 3 - zurück.
- ST_AsKML - Erweiterung: 2.0.0 - Präfix Namensraum hinzugefügt. Standardmäßig kein Präfix Gibt die Geometrie als GML-Element - Version 2 oder 3 - zurück.

- ST_Azimuth - Erweiterung: mit 2.0.0 wurde die Unterstützung des geographischen Datentyps eingeführt. Gibt die 2-dimenionale kürzeste Strecke zwischen zwei Geometrien als Linie zurück
- ST_Dimension - Erweiterung: 2.0.0 - Unterstützung für polyedrische Oberflächen und TIN eingeführt. Gibt die Dimension der Koordinaten für den Wert von ST_Geometry zurück.
- ST_Dump - Erweiterung: Mit 2.0.0 wurde die Unterstützung für polyedrische Oberflächen, Dreiecke und TIN eingeführt. Gibt einen Satz von geometry_dump Zeilen für die Komponenten einer Geometrie zurück.
- ST_DumpPoints - Erweiterung: Mit 2.0.0 wurde die Unterstützung für polyedrische Oberflächen, Dreiecke und TIN eingeführt. Gibt eine Zusammenfassung des Inhalts einer Geometrie wieder.
- ST_Expand - Erweiterung: Mit 2.0.0 wurde die Unterstützung für polyedrische Oberflächen, Dreiecke und TIN eingeführt. Gibt einen Begrenzungsrahmen zurück, der aus einem anderen Begrenzungsrahmen oder einer Geometrie erweitert wurde.
- ST_Extent - Erweiterung: Mit 2.0.0 wurde die Unterstützung für polyedrische Oberflächen, Dreiecke und TIN eingeführt. Aggregatfunktion, die den Begrenzungsrahmen von Geometrien zurückgibt.
- ST_Force2D - Erweiterung: Mit 2.0.0 wurde die Unterstützung für polyedrische Oberflächen eingeführt. Die Geometrien in einen "2-dimensionalen Modus" zwingen.
- ST_Force3D - Erweiterung: Mit 2.0.0 wurde die Unterstützung für polyedrische Oberflächen eingeführt. Zwingt die Geometrien in einen XYZ Modus. Dies ist ein Alias für ST_Force3DZ.
- ST_Force3DZ - Erweiterung: Mit 2.0.0 wurde die Unterstützung für polyedrische Oberflächen eingeführt. Zwingt die Geometrien in einen XYZ Modus.
- ST_ForceCollection - Erweiterung: Mit 2.0.0 wurde die Unterstützung für polyedrische Oberflächen eingeführt. Wandelt eine Geometrie in eine GEOMETRYCOLLECTION um.
- ST_ForceRHR - Erweiterung: Mit 2.0.0 wurde die Unterstützung für polyedrische Oberflächen eingeführt. Orientiert die Knoten in einem Polygon so, dass sie der Drei-Finger-Regel folgen.
- ST_GMLToSQL - Erweiterung: Mit 2.0.0 wurde die Unterstützung für polyedrische Oberflächen und TIN eingeführt. Returns a geometry from GML. Alias for ST_GeomFromGML.
- ST_GMLToSQL - Erweiterung: 2.0.0 Standardwert für den optionalen Parameter SRID eingefügt. Returns a geometry from GML. Alias for ST_GeomFromGML.
- ST_GeomFromEWKB - Erweiterung: Mit 2.0.0 wurde die Unterstützung für polyedrische Oberflächen und TIN eingeführt. Gibt einen geometrischen Datentyp (ST_Geometry) aus einer Well-known-Binary (WKB) Darstellung zurück.
- ST_GeomFromEWKT - Erweiterung: Mit 2.0.0 wurde die Unterstützung für polyedrische Oberflächen und TIN eingeführt. Gibt einen spezifizierten ST_Geometry-Wert von einer erweiterten Well-known-Text Darstellung (EWKT) zurück.
- ST_GeomFromGML - Erweiterung: Mit 2.0.0 wurde die Unterstützung für polyedrische Oberflächen und TIN eingeführt. Nimmt als Eingabe eine GML-Darstellung der Geometrie und gibt ein geometrisches PostGIS-Objekt aus.
- ST_GeomFromGML - Erweiterung: 2.0.0 Standardwert für den optionalen Parameter SRID eingefügt. Nimmt als Eingabe eine GML-Darstellung der Geometrie und gibt ein geometrisches PostGIS-Objekt aus.
- ST_GeometryN - Erweiterung: Mit 2.0.0 wurde die Unterstützung für polyedrische Oberflächen, Dreiecke und TIN eingeführt. Gibt den Geometrietyp des ST_Geometry Wertes zurück.
- ST_GeometryType - Erweiterung: Mit 2.0.0 wurde die Unterstützung für polyedrische Oberflächen eingeführt. Gibt den Geometrietyp des ST_Geometry Wertes zurück.
- ST_IsClosed - Erweiterung: Mit 2.0.0 wurde die Unterstützung für polyedrische Oberflächen eingeführt. Gibt den Wert TRUE zurück, wenn die Anfangs- und Endpunkte des LINESTRING's zusammenfallen. Bei polyedrischen Oberflächen, wenn sie geschlossen (volumetrisch) sind.
- ST_MakeEnvelope - Erweiterung: 2.0: es wurde die Möglichkeit eingeführt, eine Einhüllende/Envelope festzulegen, ohne dass die SRID spezifiziert ist. Erzeugt ein rechteckiges Polygon aus den gegebenen Minimum- und Maximumwerten. Die Eingabewerte müssen in dem Koordinatenreferenzsystem sein, welches durch die SRID angegeben wird.

- ST_MakeValid - Verbessert: 2.0.1, Geschwindigkeitsverbesserungen Versucht, eine ungültige Geometrie gültig zu machen, ohne dass Scheitelpunkte verloren gehen.
- ST_NPoints - Erweiterung: Mit 2.0.0 wurde die Unterstützung für polyedrische Oberflächen eingeführt. Gibt die Anzahl der Punkte (Knoten) einer Geometrie zurück.
- ST_NumGeometries - Erweiterung: Mit 2.0.0 wurde die Unterstützung für polyedrische Oberflächen, Dreiecke und TIN eingeführt. Gibt die Anzahl der Punkte einer Geometrie zurück. Funktioniert für alle Geometrien.
- ST_Relate - Verbessert: 2.0.0 - Unterstützung für die Angabe von Grenzknotenregeln hinzugefügt. Prüft, ob zwei Geometrien eine topologische Beziehung haben, die einem Schnittpunktmatrixmuster entspricht, oder berechnet ihre Schnittpunktmatrix
- ST_Rotate - Erweiterung: Mit 2.0.0 wurde die Unterstützung für polyedrische Oberflächen, Dreiecke und TIN eingeführt. Dreht eine Geometrie um einen Ursprungspunkt.
- ST_Rotate - Verbessert: In Version 2.0.0 wurden zusätzliche Parameter zur Angabe des Ursprungs der Drehung hinzugefügt. Dreht eine Geometrie um einen Ursprungspunkt.
- ST_RotateX - Erweiterung: Mit 2.0.0 wurde die Unterstützung für polyedrische Oberflächen, Dreiecke und TIN eingeführt. Dreht eine Geometrie um die X-Achse.
- ST_RotateY - Erweiterung: Mit 2.0.0 wurde die Unterstützung für polyedrische Oberflächen, Dreiecke und TIN eingeführt. Dreht eine Geometrie um die Y-Achse.
- ST_RotateZ - Erweiterung: Mit 2.0.0 wurde die Unterstützung für polyedrische Oberflächen, Dreiecke und TIN eingeführt. Dreht eine Geometrie um die Z-Achse.
- ST_Scale - Erweiterung: Mit 2.0.0 wurde die Unterstützung für polyedrische Oberflächen, Dreiecke und TIN eingeführt. Skaliert eine Geometrie um bestimmte Faktoren.
- ST_ShiftLongitude - Erweiterung: Mit 2.0.0 wurde die Unterstützung für polyedrische Oberflächen und TIN eingeführt. Verschiebt die Längenkoordinaten einer Geometrie zwischen -180..180 und 0..360.
- ST_Summary - Erweiterung: 2.0.0 Unterstützung für geographische Koordinaten hinzugefügt Gibt eine Zusammenfassung des Inhalts einer Geometrie wieder.
- ST_Transform - Erweiterung: Mit 2.0.0 wurde die Unterstützung für polyedrische Oberflächen eingeführt. Rückgabe einer neuen Geometrie mit in ein anderes räumliches Bezugssystem transformierten Koordinaten.
- ST_Value - Erweiterung: 2.0.0 Der optionale Übergabewert "exclude_nodata_value" wurde hinzugefügt. Gibt den Zellwert eines Pixels aus, das über columnx und rowy oder durch einen bestimmten geometrischen Punkt angegeben wird. Die Bandnummern beginnen mit 1 und wenn keine Bandnummer angegeben ist, dann wird Band 1 angenommen. Wenn exclude_nodata_value auf FALSE gesetzt ist, werden auch die Pixel mit einem nodata Wert mit einbezogen. Wenn exclude_nodata_value nicht übergeben wird, dann wird er über die Metadaten des Rasters ausgelesen.
- ValidateTopology - Erweiterung: 2.0.0 effizientere Ermittlung sich überkreuzender Kanten. Falsch positive Fehlmeldungen von früheren Versionen fixiert. Liefert eine Menge validatetopology_returntype Objekte, die Probleme mit der Topologie beschreiben.

Geänderte Funktionen in PostGIS 2.0

- AddGeometryColumn - Änderung: 2.0.0 Diese Funktion aktualisiert die geometry_columns Tabelle nicht mehr, da geometry_columns jetzt ein View ist, welcher den Systemkatalog ausliest. Standardmäßig werden auch keine Bedingungen/constraints erzeugt, sondern es wird der in PostgreSQL integrierte Typmodifikaor verwendet. So entspricht zum Beispiel die Erzeugung einer wgs84 POINT Spalte mit dieser Funktion: ALTER TABLE some_table ADD COLUMN geom geometry(Point,4326); Entfernt eine Geometriespalte aus einer räumlichen Tabelle.
- AddGeometryColumn - Änderung: 2.0.0 Falls Sie das alte Verhalten mit Constraints wünschen, setzen Sie bitte use_typmod vom standardmäßigen true auf false. Entfernt eine Geometriespalte aus einer räumlichen Tabelle.

- AddGeometryColumn - Änderung: 2.0.0 Views können nicht mehr händisch in "geometry_columns" registriert werden. Views auf eine Geometrie in Typmod-Tabellen, bei denen keine Adapterfunktion verwendet wird, registrieren sich selbst auf korrekte Weise, da sie die Typmod-Verhaltensweise von der Spalte der Stammtabelle erben. Views die ein geometrische Funktion ausführen die eine andere Geometrie ausgibt, benötigen die Umwandlung in eine Typmod-Geometrie, damit die Geometrie des Views korrekt in "geometry_columns" registriert wird. Siehe . Entfernt eine Geometriespalte aus einer räumlichen Tabelle.
- Box3D - Änderung: 2.0.0 In Versionen vor 2.0 war dies üblicherweise Box2D anstelle von Box3D. Da Box2D überholt ist, wurde dies zu Box3D geändert. Stellt das umschreibende Rechteck eines Raster als Box3D dar.
- DropGeometryColumn - Changed: 2.0.0. Retained for backward compatibility after geometry_columns became a view of the system catalogs. Entfernt eine Geometriespalte aus einer räumlichen Tabelle.
- DropGeometryTable - Changed: 2.0.0. Retained for backward compatibility after geometry_columns became a view of the system catalogs. Löscht eine Tabelle und alle Referenzen in dem geometry_columns View.
- Populate_Geometry_Columns - Änderung: 2.0.0 Standardmäßig werden nun Typmodifikatoren anstelle von Check-Constraints für die Beschränkung des Geometrietyps verwendet. Sie können nach wie vor stattdessen die Verhaltensweise mit Check-Constraints verwenden, indem Sie die neu eingeführte Variable use_typmod auf FALSE setzen. Stellt sicher, dass Geometriespalten mit Typmodifikatoren definiert sind oder geeignete räumliche Beschränkungen haben.
- ST_3DExtent - Geändert: 2.0.0 In früheren Versionen wurde dies als ST_Extent3D bezeichnet. Aggregatfunktion, die den 3D-Begrenzungsrahmen von Geometrien zurückgibt.
- ST_3DLength - Änderung: 2.0.0 In Vorgängerversionen als ST_Length3D bezeichet. Gibt den geometrischen Schwerpunkt einer Geometrie zurück.
- ST_3DMakeBox - Geändert: 2.0.0 In früheren Versionen hieß diese Funktion ST_MakeBox3D Erzeugt einen BOX3D, der durch zwei 3D-Punktgeometrien definiert ist.
- ST_3DPerimeter - Änderung: 2.0.0 In Vorgängerversionen als ST_Perimeter3D bezeichet. Gibt den geometrischen Schwerpunkt einer Geometrie zurück.
- ST_AsBinary - Changed: 2.0.0 Inputs to this function can not be unknown -- must be geometry. Constructs such as ST_AsBinary('POIN 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. Rückgabe der OGC/ISO Well-Known Binary (WKB)-Darstellung der Geometrie/Geografie ohne SRID-Metadaten.
- ST_AsGML - Änderung: 2.0.0 verwendet standardmäßig benannte Argumente. Gibt die Geometrie als GML-Element - Version 2 oder 3 - zurück.
- ST_AsGeoJSON - Änderung: 2.0.0 Unterstützung für Standardargumente und benannte Argumente. Rückgabe einer Geometrie oder eines Merkmals im GeoJSON-Format.
- ST_AsSVG - Änderung: 2.0.0 verwendet Standardargumente und unterstützt benannte Argumente. Gibt eine Geometrie als SVG-Pfad aus.
- ST_EndPoint - Änderung: 2.0.0 unterstützt die Verarbeitung von MultiLinestring's die nur aus einer einzelnen Geometrie bestehen, nicht mehr. In früheren Versionen von PostGIS gab die Funktion bei einem aus einer einzelnen Linie bestehender MultiLinestring den Anfangspunkt zurück. Ab 2.0.0 gibt sie nur NULL zurück, so wie bei jedem anderen MultiLinestring. Die alte Verhaltensweise war undokumentiert, aber Anwender, die annahmen, dass Sie Ihre Daten als LINESTRING vorliegen haben, könnten in 2.0 dieses zurückgegebene NULL bemerken. Returns the last point of a LineString, CircularLineString, or NURBSCurve.
- ST_GDALDrivers - Änderung: 2.0.6, 2.1.3 - standardmäßig ist kein Treiber aktiviert, solange die GUC oder die Umgebungsvariable "gdal_enabled_drivers" nicht gesetzt sind. Gibt eine Liste der Rasterformate aus, die von PostGIS über die Bibliothek GDAL unterstützt werden. Nur die Formate mit can_write=True können von ST_AsGDALRaster verwendet werden.
- ST_GeomFromText - Änderung: 2.0.0 - In Vorgängerversionen von PostGIS war ST_GeomFromText('GEOMETRYCOLLECTION(E erlaubt. Um eine bessere Übereinstimmung mit der SQL/MM Norm zu erreichen, ist dies in PostGIS 2.0.0 nun nicht mehr gestattet. Hier sollte nun ST_GeomFromText('GEOMETRYCOLLECTION EMPTY') geschrieben werden. Gibt einen spezifizierten ST_Geometry Wert aus einer Well-known-Text Darstellung (WKT) zurück.

- ST_GeometryN - Änderung: 2.0.0 Vorangegangene Versionen geben bei Einzelgeometrien NULL zurück. Dies wurde geändert um die Geometrie für den ST_GeometrieN(..,1) Fall zurückzugeben. Gibt den Geometrietyp des ST_Geometry Wertes zurück.
- ST_IsEmpty - Änderung: 2.0.0 - In Vorgängerversionen von PostGIS war ST_GeomFromText('GEOMETRYCOLLECTION(EMPTY erlaubt. Um eine bessere Übereinstimmung mit der SQL/MM Norm zu erreichen, ist dies nun nicht mehr gestattet. Prüft, ob eine Geometrie leer ist.
- ST_Length - Änderung: 2.0.0 Wesentliche Änderung -- In früheren Versionen ergab die Anwendung auf ein MULTI/POLYGON vom geographischen Datentyp den Umfang des POLYGON/MULTIPOLYGON. In 2.0.0 wurde dies geändert und es wird jetzt 0 zurückgegeben, damit es mit der Verhaltensweise beim geometrischen Datentyp übereinstimmt. Verwenden Sie bitte ST_Perimeter, wenn Sie den Umfang eines Polygons wissen wollen Gibt den geometrischen Schwerpunkt einer Geometrie zurück.
- ST_LocateAlong - Änderung: 2.0.0 In Vorgängerversionen als ST_Locate_Along_Measure bezeichet. Der alte Name ist überholt und wird in der Zukunft entfernt ist aber noch verfügbar. Gibt die Punkte auf einer Geometrie zurück, die einem Messwert entsprechen.
- ST_LocateBetween - Änderung: 2.0.0 In Vorgängerversionen als ST_Locate_Along_Measure bezeichet. Der alte Name ist überholt und wird in der Zukunft entfernt ist aber noch verfügbar. Gibt die Teile einer Geometrie zurück, die einem Messbereich entsprechen.
- ST_ModEdgeSplit - Änderung: 2.0 - In Vorgängerversionen fälschlicherweise als ST_ModEdgesSplit bezeichnet Trennt eine Kante auf, indem ein neuer Knoten entlang einer bestehenden Kante erstellt wird. Ändert die ursprüngliche Kante und fügt eine neue Kante hinzu.
- ST_NumGeometries - Änderung: 2.0.0 Bei früheren Versionen wurde NULL zurückgegeben, wenn die Geometrie nicht vom Typ GEOMETRYCOLLECTION/MULTI war. 2.0.0+ gibt nun 1 für Einzelgeometrien, wie POLYGON, LINESTRING, POINT zurück. Gibt die Anzahl der Punkte einer Geometrie zurück. Funktioniert für alle Geometrien.
- ST_NumInteriorRings - Änderung: 2.0.0 - In füheren Versionen war ein MULTIPOLYGON als Eingabe erlaubt, wobei die Anzahl der inneren Ringe des ersten Polygons ausgegeben wurde. Gibt die Anzahl der inneren Ringe einer Polygongeometrie aus.
- ST_PointN - Änderung: 2.0.0 arbeitet nicht mehr mit MultiLinestring's, die nur eine einzelne Geometrie enthalten. In früheren Versionen von PostGIS gab die Funktion bei einem, aus einer einzelnen Linie bestehender MultiLinestring, den Anfangspunkt zurück. Ab 2.0.0 wird, so wie bei jedem anderen MultiLinestring auch, NULL zurückgegeben. Gibt die Anzahl der Stützpunkte eines ST_LineString oder eines ST_CircularString zurück.
- ST_ScaleX - Änderung: 2.0.0 In WKTRaster Versionen wurde dies als ST_PixelSizeX bezeichnet. Gibt die X-Komponente der Pixelbreite in den Einheiten des Koordinatenreferenzsystems aus.
- ST_ScaleY - Änderung: 2.0.0. Versionen von WKTRaster haben dies als "ST_PixelSizeY" bezeichnet. Gibt die Y-Komponente der Pixelhöhe in den Einheiten des Koordinatenreferenzsystems aus.
- ST_SetScale - Änderung: 2.0.0. Versionen von WKTRaster haben dies als "ST_SetPixelSizeY" bezeichnet. Dies wurde mit 2.0.0 geändert. Setzt die X- und Y-Größe der Pixel in den Einheiten des Koordinatenreferenzsystems. Entweder eine Zahl pro Pixel oder Breite und Höhe.
- ST_StartPoint - Änderung: 2.0.0 unterstützt die Verarbeitung von MultiLinestring's die nur aus einer einzelnen Geometrie bestehen, nicht mehr. In früheren Versionen von PostGIS gab die Funktion bei einem aus einer einzelnen Linie bestehender MultiLinestring den Anfangspunkt zurück. Ab 2.0.0 gibt sie nur NULL zurück, so wie bei jedem anderen MultiLinestring. Die alte Verhaltensweise war undokumentiert, aber Anwender, die annahmen, dass Sie Ihre Daten als LINESTRING vorliegen haben, könnten in 2.0 dieses zurückgegebene NULL bemerken. Returns the first point of a LineString, CircularLineString, or NURBSCurve.

13.12.15 PostGIS-Funktionen neu oder erweitert in 1.5

Die unten aufgeführten Funktionen sind PostGIS-Funktionen, die hinzugefügt oder erweitert wurden.

Neue Funktionen in PostGIS 1.5

- && - Verfügbarkeit: Mit 1.5.0 wurde die Unterstützung von geograpischen Koordinaten eingeführt Gibt TRUE zurück, wenn die 2D Bounding Box von A die 2D Bounding Box von B schneidet.
- PostGIS_LibXML_Version - Verfügbarkeit: 1.5 Gibt die Versionsnummer der libxml2-Bibliothek zurück.
- ST_AddMeasure - Verfügbarkeit: 1.5.0 Interpoliert Maße entlang einer linearen Geometrie.
- ST_AsBinary - Verfügbarkeit: 1.5.0 Unterstützung von geograpischen Koordinaten. Rückgabe der OGC/ISO Well-Known Binary (WKB)-Darstellung der Geometrie/Geografie ohne SRID-Metadaten.
- ST_AsGML - Verfügbarkeit: 1.5.0 Unterstützung von geograpischen Koordinaten. Gibt die Geometrie als GML-Element - Version 2 oder 3 - zurück.
- ST_AsGeoJSON - Verfügbarkeit: 1.5.0 Unterstützung von geograpischen Koordinaten. Rückgabe einer Geometrie oder eines Merkmals im GeoJSON-Format.
- ST_AsText - Verfügbarkeit: 1.5 - Unterstützung von geograpischen Koordinaten. Gibt die Well-known-Text(WKT) Darstellung der Geometrie/Geographie ohne die SRID Metadaten zurück.
- ST_Buffer - Verfügbarkeit: 1.5 - ST_Buffer wurde um die Unterstützung von Abschlusstücken/endcaps und Join-Typen erweitert. Diese können zum Beispiel dazu verwendet werden, um Linienzüge von Straßen in Straßenpolygone mit flachen oder rechtwinkeligen Abschlüssen anstatt mit runden Enden umzuwandeln. Ein schlanker Adapter für den geographischen Datentyp wurde hinzugefügt. Berechnet eine Geometrie, die alle Punkte innerhalb eines bestimmten Abstands zu einer Geometrie umfasst.
- ST_ClosestPoint - Verfügbarkeit: 1.5.0 Gibt den 2D-Punkt auf g1 zurück, der g2 am nächsten ist. Dies ist der erste Punkt der kürzesten Linie von einer Geometrie zur anderen.
- ST_CollectionExtract - Verfügbarkeit: 1.5.0 Gibt bei einer Geometriesammlung eine Multi-Geometrie zurück, die nur Elemente eines bestimmten Typs enthält.
- ST_Covers - Verfügbarkeit: 1.5 - Unterstützung von geograpischen Koordinaten. Prüft, ob jeder Punkt von B in A liegt
- ST_DFullyWithin - Verfügbarkeit: 1.5.0 Tests if a geometry is entirely inside a distance of another
- ST_DWithin - Verfügbarkeit: Mit Version 1.5.0 wurde die Unterstützung für Geografie eingeführt. Prüft, ob zwei Geometrien innerhalb eines bestimmten Abstands liegen
- ST_Distance - Verfügbarkeit: 1.5.0 die Unterstützung des geograpischen Datentyps wurde eingeführt. Geschwindigkeitsverbesserungen bei einer umfangreichen Geometrie und bei einer Geometrie mit vielen Knoten Gibt die größte 3-dimensionale Distanz zwischen zwei geometrischen Objekten als Linie zurück
- ST_DistanceSphere - Verfügbarkeit: 1.5 die Unterstützung für weitere geometrische Datentypen neben Punkten eingeführt. Bei Vorgängerversionen wurden nur Punkte unterstützt. Gibt die kürzeste Distanz zwischen zwei geometrischen Objekten zurück, die über Länge, Breite und ein bestimmtes Referenzellipsoid gegeben sind. Vorgängerversionen von PostGIS 1.5 unterstützten nur Punkte.
- ST_DistanceSpheroid - Verfügbarkeit: 1.5 die Unterstützung für weitere geometrische Datentypen neben Punkten eingeführt. Bei Vorgängerversionen wurden nur Punkte unterstützt. Gibt die kürzeste Distanz zwischen zwei geometrischen Objekten zurück, die über Länge, Breite und ein bestimmtes Referenzellipsoid gegeben sind. Vorgängerversionen von PostGIS 1.5 unterstützten nur Punkte.
- ST_DumpPoints - Verfügbarkeit: 1.5.0 Gibt eine Zusammenfassung des Inhalts einer Geometrie wieder.
- ST_Envelope - Verfügbarkeit: 1.5.0 Änderung der Verhaltensweise insofern, das die Ausgabe in Double Precision anstelle von Float4 erfolgt Gibt eine Geometrie in doppelter Genauigkeit (float8) zurück, welche das Umgebungsrechteck der beigestellten Geometrie darstellt.
- ST_Expand - Verfügbarkeit: 1.5.0 Verhalten geändert, um double precision statt float4 Koordinaten auszugeben. Gibt einen Begrenzungsrahmen zurück, der aus einem anderen Begrenzungsrahmen oder einer Geometrie erweitert wurde.
- ST_GMLToSQL - Verfügbarkeit: 1.5, benötigt libxml2 1.6+ Returns a geometry from GML. Alias for ST_GeomFromGML.

- ST_GeomFromGML - Verfügbarkeit: 1.5, benötigt libxml2 1.6+ Nimmt als Eingabe eine GML-Darstellung der Geometrie und gibt ein geometrisches PostGIS-Objekt aus.
- ST_GeomFromKML - Verfügbarkeit: 1.5, benötigt libxml2 2.6+ Nimmt als Eingabe eine KML-Darstellung der Geometrie und gibt ein geometrisches PostGIS-Objekt aus.
- ST_HausdorffDistance - Verfügbarkeit: 1.5.0 Gibt den kürzesten 3-dimensionalen Abstand zwischen zwei geometrischen Objekten als Linie zurück
- ST_Intersection - Verfügbarkeit: Mit Version 1.5 wurde die Unterstützung für den Datentyp Geographie eingeführt. Berechnet eine Geometrie, die den gemeinsamen Teil der Geometrien A und B darstellt.
- ST_Intersects - Verfügbarkeit: Mit Version 1.5 wurde die Unterstützung für Geografie eingeführt. Prüft, ob sich zwei Geometrien schneiden (sie haben mindestens einen Punkt gemeinsam)
- ST_Length - Verfügbarkeit: 1.5.0 Unterstützung von geograpischen Koordinaten. Gibt den geometrischen Schwerpunkt einer Geometrie zurück.
- ST_LongestLine - Verfügbarkeit: 1.5.0 Gibt die größte 3-dimensionale Distanz zwischen zwei geometrischen Objekten als Linie zurück
- ST_MakeEnvelope - Verfügbarkeit: 1.5 Erzeugt ein rechteckiges Polygon aus den gegebenen Minimum- und Maximumwerten. Die Eingabewerte müssen in dem Koordinatenreferenzsystem sein, welches durch die SRID angegeben wird.
- ST_MaxDistance - Verfügbarkeit: 1.5.0 Gibt die größte 2-dimensionale Distanz zwischen zwei geometrischen Objekten in projizierten Einheiten zurück.
- ST_ShortestLine - Verfügbarkeit: 1.5.0 Gibt die 2-dimenionale kürzeste Strecke zwischen zwei Geometrien als Linie zurück
- ~= - Verfügbarkeit: 1.5.0 "Verhaltensänderung" Gibt TRUE zurück, wenn die bounding box von A ident mit jener von B ist.

13.12.16 PostGIS-Funktionen neu oder erweitert in 1.4

Die unten aufgeführten Funktionen sind PostGIS-Funktionen, die hinzugefügt oder erweitert wurden.

Neue Funktionen in PostGIS 1.4

- Populate_Geometry_Columns - Verfügbarkeit: 1.4.0 Stellt sicher, dass Geometriespalten mit Typmodifikatoren definiert sind oder geeignete räumliche Beschränkungen haben.
- ST_Collect - Verfügbarkeit: 1.4.0 - ST_MakeLine(geomarray) wurde eingeführt. ST_MakeLine Aggregatfunktion wurde verbessert, um mehr Punkte schneller handhaben zu können. Erzeugt eine GeometryCollection oder Multi*-Geometrie aus einer Reihe von Geometrien.
- ST_ContainsProperly - Verfügbarkeit: 1.4.0 Prüft, ob jeder Punkt von B im Inneren von A liegt
- ST_GeoHash - Verfügbarkeit: 1.4.0 Gibt die Geometrie in der GeoHash Darstellung aus.
- ST_IsValidReason - Verfügbarkeit: 1.4 Gibt einen Text zurück, der angibt, ob eine Geometrie gültig ist, oder einen Grund für die Ungültigkeit.
- ST_LineCrossingDirection - Verfügbarkeit: 1.4 Gibt eine Zahl zurück, die das Kreuzungsverhalten von zwei LineStrings angibt
- ST_LocateBetweenElevations - Verfügbarkeit: 1.4.0 Gibt die Teile einer Geometrie zurück, die in einem Höhenbereich (Z) liegen.
- ST_MakeLine - Verfügbarkeit: 1.4.0 - ST_MakeLine(geomarray) wurde eingeführt. ST_MakeLine Aggregatfunktion wurde verbessert, um mehr Punkte schneller handhaben zu können. Erzeugt einen Linienzug aus einer Punkt-, Mehrfachpunkt- oder Liniengeometrie.
- ST_MinimumBoundingCircle - Verfügbarkeit: 1.4.0 Gibt das kleinste Kreispolygon zurück, das eine Geometrie enthält.
- ST_Union - Verfügbarkeit: 1.4.0 - ST_Union wurde verbessert. ST_Union(geomarray) wurde eingeführt und auch schnellere Aggregat-Sammlung in PostgreSQL. Berechnet eine Geometrie, die die Punktmengenvereinigung der Eingabegeometrien darstellt.

13.12.17 PostGIS-Funktionen neu oder erweitert in 1.3

Die unten aufgeführten Funktionen sind PostGIS-Funktionen, die hinzugefügt oder erweitert wurden.

Neue Funktionen in PostGIS 1.3

- ST_AsGML - Verfügbarkeit: 1.3.2 Gibt die Geometrie als GML-Element - Version 2 oder 3 - zurück.
- ST_AsGeoJSON - Verfügbarkeit: 1.3.4 Rückgabe einer Geometrie oder eines Merkmals im GeoJSON-Format.
- ST_CurveToLine - Verfügbarkeit: 1.3.0 Konvertiert eine Geometrie mit Kurven in eine lineare Geometrie.
- ST_LineToCurve - Verfügbarkeit: 1.3.0 Konvertiert eine lineare Geometrie in eine gekrümmte Geometrie.
- ST_SimplifyPreserveTopology - Verfügbarkeit: 1.3.3 Gibt eine vereinfachte und gültige Darstellung einer Geometrie zurück, die den Douglas-Peucker-Algorithmus verwendet.

Kapitel 14

Meldung von Problemen

14.1 Software Bugs melden

Effektive Fehlerberichte sind ein wesentlicher Beitrag zur Weiterentwicklung von PostGIS. Am wirksamsten ist ein Fehlerbericht dann, wenn er von den PostGIS-Entwicklern reproduziert werden kann. Idealerweise enthält er ein Skript das den Fehler auslöst und eine vollständige Beschreibung der Umgebung in der er aufgetreten ist. Ausreichend gute Information liefert `SELECT postgis_full_version()` [für PostGIS] und `SELECT version()` [für PostgreSQL].

Falls Sie nicht die aktuelle Version verwenden, sollten Sie zuerst unter release changelog nachsehen, ob Ihr Bug nicht bereits bereinigt wurde.

Die Verwendung des PostGIS bug tracker stellt sicher dass Ihre Berichte nicht verworfen werden, und dass Sie über die Prozessabwicklung am Laufenden gehalten werden. Bevor Sie einen neuen Fehler melden fragen Sie bitte die Datenbank ab ob der Fehler schon bekannt ist. Wenn es ein bekannter Fehler ist, so fügen Sie bitte jegliche neue Information die Sie herausgefunden haben hinzu.

Vielleicht möchten Sie zuvor Simon Tatham's Artikel über How to Report Bugs Effectively lesen, bevor Sie einen Fehlerbericht senden.

14.2 Probleme mit der Dokumentation melden

Die Dokumentation sollte die Eigenschaften und das Verhalten der Software exakt wiederspiegeln. Wenn das nicht der Fall ist, so kann entweder ein Softwarebug oder eine fehlerhafte bzw. unzulängliche Dokumentation daran Schuld sein.

Probleme in der Dokumentation können unter PostGIS bug tracker gemeldet werden.

Wenn die Überarbeitung trivial ist, können Sie diese in einem neuen Bug Tracker Issue beschreiben. Geben Sie bitte die exakte Stelle in der Dokumentation an.

Wenn es sich um umfangreichere Änderungen handelt, ist ein Subversion Patch zweifellos die bessere Wahl. Dabei handelt es sich um einen vierstufigen Vorgang unter Unix (angenommen, Sie haben Subversion bereits installiert):

Before editing or building the manual, read the repository's documentation development guide. It covers the executable example, generated figure, validation, HTML, and PDF workflows used by the manual.

1. Eine Kopie des PostGIS Subversion Trunks auschecken. Eingabe unter Unix:

   git clone https://git.osgeo.org/gitea/postgis/postgis.git

   Dies wird im Verzeichnis ./trunk gespeichert

2. Erledigen Sie die Änderungen an der Dokumentation mit Ihrem Lieblingstexteditor. Auf Unix, tippen Sie (zum Beispiel):

   vim doc/postgis.xml

   Bedenken Sie bitte, dass die Dokumentation in DocBook XML und nicht in HTML geschrieben ist. Falls Sie damit nicht vertraut sind, so folgen Sie bitte dem Beispiel in der restlichen Dokumentation.

3. Erzeugung einer Patchdatei, welche die Unterschiede zur Master-Kopie der Dokumentation enthält. Unter Unix tippen Sie bitte:

   git diff doc/postgis.xml > doc.patch

4. Fügen Sie den Patch einem neuen Thema/Issue im Bug Tracker bei.

Kapitel 15

Credits and Acknowledgments

PostGIS is built and sustained by a worldwide community. This chapter records the people, organizations, funders, and upstream projects that have contributed code, documentation, infrastructure, testing, support, and funding.

15.1 Project Steering Committee

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

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

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

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

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

15.2 Core Contributors Present

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

Loïc Bartoletti SFCGAL enhancements and maintenance and ci support

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

Martin Davis GEOS enhancements and documentation

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

Aliaksandr Kalenik Geometry Processing, PostgreSQL gist, general bug fixing

Laurențiu Nicola Bug fixes, maintenance, stability, and performance

15.3 Core Contributors Past

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

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

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

Jorge Arévalo Raster development, GDAL driver support, loader

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

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

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

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

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

Jeff Lounsbury Original development of the Shapefile loader/dumper.

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

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

David Zwarg Raster development (mostly map algebra analytic functions)

15.4 PostGIS Raster Origins

PostGIS raster discussions date to 2002 and included early PGCHIP experiments. Xing Lin developed the **geotiff2pgraster** PGRaster prototype during Google Summer of Code 2007. The WKT Raster project began in 2008, Jorge Arévalo developed the first GDAL driver prototype during Google Summer of Code 2009, and the 2010 FOSS4G code sprint agreed to make WKT Raster the PostGIS raster type. Raster support shipped as part of PostGIS 2.0 in 2012.

The early implementation team included Pierre Racine, Mateusz Loskot, Sandro Santilli, Jorge Arévalo, Bborie Park, David Zwarg, and Regina Obe. Their work established the raster type, loader, GDAL integration, raster/vector operations, map algebra, documentation, and testing that became the current raster subsystem.

Other recorded contributions include Jeff Adams's early sprint work, Qing Liu's raster distance analysis, Maurício Carvalho Mathias de Paulo's QGIS raster plugin and related paper with Lubia Vinhas, and Ignacio Brodin's gvSIG raster plugin and presentation with Jorge Sanz. Bryce L. Nordgren created early raster coverage and seamless-architecture diagrams. Pierre Racine wrote the original raster/vector tutorial and created its walkthrough and early API diagrams.

The historical project planning record credits Steve Cumming and Université Laval with funding Pierre Racine's work from June 2008 through July 2012; Martin Daly and Cadcorp with funding about six months of Mateusz Loskot's work through June 2009; Tyler Erickson and Michigan Tech Research Institute with a $1,200 USD contribution; Miguel Lizondo and Deimos Space with support for about two years of Jorge Arévalo's work through June 2011; and the Center for Vectorborne Diseases at UC Davis with funding Bborie Park's work from March 2010 through July 2012.

15.5 Other Contributors

Individual Contributors

@amwps290 (GitHub user)
dovkoy (GitHub user)
@kannes (GitHub user: Hannes)
@sunhm89 (GitHub user)
@wewindy (GitHub user: Chen)
Adrien Berchet
Alex Bodnaru
Alex Mayrhofer
Andrea Peri
Andreas Forø Tollefsen
Andreas Neumann
Andreas Schild
Andrew Gierth
Andrew Harvey
Andrew Kroh
Anne Ghisla
Antoine Bajolet
Ariel Mashraki
Arno Renevier
Arthur Bazin
Arthur Lesuisse
Artur Zakirov
Ayo Adesugba
Barbara Phillipot
Baris Ergun
Bas Couwenberg
Ben Jubb
Benjamin Morel
Bernd Reiß
Bernhard Reiter
Bilal Kahraman
Bill Mill
Björn Esser
Brian Hamlin
Bruce Rindahl
Bruno Wolff III
Bryce L. Nordgren
Carl Anderson
Carlos López Quintanilla
Charles Karney
Charlie Savage
Chris Mayo
Christian Schroeder
Christoph Berg
Christoph Moench-Tegeder
Dan Faudemer
Dan Levy
Dane Springmeyer
Daniel Nylander
Dapeng Wang
Daryl Herzmann
Dave Fuhry
David G. Johnston
David Garnier
David Skea
David Techer
Dennis Tighe
Denys Kovshun
Devrim Gündüz
Dian M Fay
Dmitry Vasilyev
Edouard Choinière
Eduin Carrillo
Egor Ivkov
Fabrice Fontaine
Florentin Drogeanu
Florian Weimer
François Bonzon
Frank Warmerdam
George Silva
Gerald Fenoy
Gino Lucrezi
Giovanni Zezza
Grant Hutchins
Greg Troxel
Guillaume Lelarge
Giuseppe Broccolo
Han Wang
Hannes Janetzek
Hans Lemuet
Haribabu Kommi
Harrissou Sant-anna
Havard Tveite
Ignacio Brodin
IIDA Tetsushi
Ingvild Nystuen
Jackie Leng
James Addison
James Marca
Jan Katins
Jan Tojnar
Jason Smith
Jean Felder
Jeff Adams
Jelte Fennema
Jim Jones
Joe Conway
John Carmack
Jonne Savolainen
Jorge Gustavo Rocha
Jorge Sanz
Jose Carlos Martinez Llari
Jostein Leira
Jörg Habenicht
Juan Sebastian Jimenez Aldana
Juergen E. Fischer
Julien Monticolo
Julien Rouhaud
Kashif Rasul
Klaus Foerster
Kris Jurka
Kristian Thy
Lars Aksel Opsahl
Laurențiu Nicola
Laurenz Albe
Lars Roessiger
Lauri Kajan
Leo Hsu
Loic Dachary
Louis Descoteaux
Luca Delucchi
Luca S. Percich
Lucas C. Villa Real
Ludovic Hirlimann
Lubia Vinhas
Maksim Korotkov
Manuel Morillo
Maria Arias de Reyna
Matt Bretl
Matthias Bay
Maurício Carvalho Mathias de Paulo
Maxim Kochetkov
Maxim Korotkov
Maxime Guillaud
Maxime van Noppen
Maxime Schoemans
Megan Ma
Mehmet Ince
Michael Fuhr
Michal Páleník
Mike Toews
Mitchell Henke
Nathan Wagner
Nathaniel Clay
Nicolas Roelandt
Nikita Shulga
Nikolai Berkoff
Norman Barker
Norman Vine
Patricia Tozer
Paul Norman
Paulo Cesar Coronado
Pavel Seleznev
Pawel Ostrowski
Peter Nguyen
Pierre Cardinal
Qing Liu
Rafal Magda
Ralph Mason
Rémi Cura
Richard Greenwood
Robert Coup
Rodrigo Smarzaro
Roger Crew
Ron Mayer
Sam Peters
Sarath Kumar
Sergei Shoulbakov
Sergey Fedoseev
Shinichi Sugiyama
Shoaib Burq
Silvio Grosso
Simon Greener
Stefan Corneliu Petrea
Steffen Macke
Stepan Kuzmin
Stephen Frost
Stephen Woodbridge
Steven Ottens
Svyatoslav Bogdanov
Talha Rizwan
Temuri Doghonadze
Teramoto Ikuhiro
Thomas Muguet
Thomas Petazzoni
Tobias Bussmann
Tom Glancy
Tom van Tilburg
Victor Collod
Vincent Bre
Vincent Mora
Vincent Picavet

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

- Aiven
- Arrival 3D
- Associazione Italiana per l'Informazione Geografica Libera (GFOSS.it)
- AusVet
- Avencia
- Azavea
- Boundless
- Cadcorp
- Camptocamp
- Carto
- Crunchy Data
- City of Boston (DND)
- City of Helsinki
- Clever Elephant Solutions
- Cooperativa Alveo
- Deimos Space
- Faunalia
- Geographic Data BC
- Google
- HighGo
- Hunter Systems Group
- IITM Pravartak Security Team
- INIA-CSIC
- ISciences, LLC
- Jirotech (formerly LISAsoft)
- Koordinates
- Kontur
- Lidwala Consulting Engineers
- Logical Tracking & Tracing International AG
- Maponics
- Maumap
- Michigan Tech Research Institute
- Natural Resources Canada
- Norwegian Forest and Landscape Institute
- Norwegian Institute of Bioeconomy Research (NIBIO)
- OSGeo
- Oslandia
- Palantir Technologies
- Paragon Corporation
- PlanetScale
- Postgres Pro

- R3 GIS
- Refractions Research
- Regione Toscana - SITA
- Safe Software
- Sirius Corporation plc
- Stadt Uster
- UC Davis Center for Vectorborne Diseases
- Université Laval
- U.S. Census Bureau
- U.S. Department of State (HIU)
- Zonar Systems

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

PostGIS has experimented with community fundraising for features in the past. We used 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.

External Libraries and Platforms PostgreSQL, the extensible database system on which PostGIS is built. PostgreSQL provides the SQL engine, transactions, query planner, extension framework, and index access methods used by PostGIS.

The GEOS geometry operations library.

The PROJ coordinate transformation and geodesic calculation library.

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

LibXML2 for XML-based spatial formats, and JSON-C for GeoJSON input.

The GNU Multiple Precision Arithmetic Library (GMP) for precision-sensitive topology operations.

protobuf-c for Mapbox Vector Tile and Geobuf output.

The iconv API, supplied by the C library or GNU libiconv, for character encoding conversion in the shapefile loader.

SFCGAL for advanced 2D and 3D functions, together with the CGAL algorithms that SFCGAL exposes.

Bundled Libraries and Source Code PostGIS ships selected source from FlatGeobuf and FlatBuffers to support FlatGeobuf input and output.

Wagyu and Mapbox Geometry provide polygon clipping and validation for Mapbox Vector Tile output.

Ryu provides round-trip-safe floating-point formatting, and uthash provides in-process hash tables.

The shapefile loader and dumper include code derived from ShapeLib, originally by Frank Warmerdam and Even Rouault. They also include public-domain `getopt` code from AT&T.

`liblwgeom/lookup3.c` contains the public-domain `lookup3` hash function by Bob Jenkins.

Anhang A

Anmerkungen zur Release

A.1 PostGIS 3.7.0

This version requires GEOS 3.10+, PostgreSQL 14-19beta3, Proj 6.1+, and libgmp. To take advantage of all postgis extension features, GEOS 3.15+ is needed. To take advantage of all postgis_sfcgal extension features SFCGAL 2.3+ is needed. Python 3 is a new requirement to build the documentation.

Visual examples in the docs are off by default. To build docs with them, configure with `--with-visual-examples` and make sure you have a running PostgreSQL instance; they are then generated automatically when building the docs (or explicitly with `make -C regress visual-examples`).

A.1.1 Wechselnde Änderungen

#5688, [topology] topology building functions now interpret tolerance 0 (old default) to really mean 0 and want -1 (new default) to mean "use topology precision" (Sandro Santilli)

#6024, Remove support for PostgreSQL 12 and 13 (Regina Obe)

#6040, ST_GeoHash round trip through ST_GeomFromGeoHash (Paul Ramsey)

#6053, [address_standardizer] Extension removed and moved to https://github.com/postgis/address_standardizer (Paul Ramsey)

#6052, [tiger_geocoder] Extension removed and moved to https://git.osgeo.org/postgis/postgis_tiger_geocoder (Regina Obe)

#6079, Remove support for GEOS < 3.10 (Sandro Santilli)

#1793, Remove ST_MemCollect aggregate (Darafei Praliaskouski)

GH-848, CG_AlphaShape now returns a MultiPolygon (Jean Felder)

[docs] Python 3 and an SVG rasterizer (rsvg-convert preferred, GraphicsMagick or ImageMagick supported) now required for doc building (Darafei Praliaskouski)

A.1.2 New Features

#1124, #2935, #3033, #4658, #4659, [loader] Rework loader arguments: shp2pgsql can create UNLOGGED tables and choose the feature id column name, shp2pgsql --drop-table can emit DROP TABLE before prepare output, raster2pgsql/shp2pgsql expose loader actions as long options, both loaders accept long aliases for existing options, and

-if-not-exists makes creation actions idempotent (Darafei Praliaskouski)

#4208, Add single-geometry variants of ST_MaxDistance and ST_LongestLine (Darafei Praliaskouski)

[topology] FindVertexSegmentPairsBelowDistance function (Sandro Santilli)

ST_CoverageEdges, returns MultiLinestring of distinct shared edges in polygonal coverage (Paul Ramsey)

ST_MinimumSpanningTree, window function to calculate MST (Paul Ramsey)

#5993, [topology] Add max_edges parameter to TopoGeo_AddLinestring (Sandro Santilli)

#6001, support MultiLineString in ST_MakeLine (Paul Ramsey)

#2858, ST_MMin and ST_MMax (Paul Ramsey)

#5941, [raster] Support GDT_Float16 pixel type (Darafei Praliaskouski)

#5992, Optimize GiST index for repeated edge subdivision in topology (Darafei Praliaskouski)

#5702, Allow the compiler to detect the parallelism -flto=auto (Darafei Praliaskouski)

#4798, ST_AsGeoJSON warns about duplicate property keys (Darafei Praliaskouski)

#4222, [raster] ST_DumpAsPolygons honours PostgreSQL interrupts (Darafei Praliaskouski)

#5889, [topology] Include representative locations in topology build errors (Darafei Praliaskouski)

#2614, Use GEOSPreparedDistance and caching to accelerate ST_DWithin (Paul Ramsey)

GH-839, ST_Multi support for TIN and surfaces (Loïc Bartoletti)

#4398, Add ST_CatmullSmoothing smoothes but retains original vertices (Paul Ramsey)

#4560, ST_3DInterpolatePoint for M interpolation from XYZ inputs (Paul Ramsey)

#1291, ST_MakePolygon support for curved rings (Darafei Praliaskouski)

GT-270, Add NURBSCurve (Loïc Bartoletti)

#2863, Add ST_XSize, ST_YSize, ST_ZSize, and ST_MSize dimension helpers (Darafei Praliaskouski)

GT-806, #6124 --with-visual-examples opt-in configure switch to build dynamic images (Regina Obe)

A.1.3 Erweiterungen

#2045, pgsql2shp query dumps no longer require temporary table privileges (Darafei Praliaskouski)

#1577, [loader] Report unsupported .zip archive input before trying to open shapefile sidecar files (Darafei Praliaskouski)

#3158, Move geometry and geography typmod bit helpers out of the public liblwgeom header (Darafei Praliaskouski)

#3776, [raster] Keep distinct 32-bit integer pixel values separate in ST_DumpAsPolygons (Darafei Praliaskouski)

#4678, [raster] Move st_sum4ma neighborhood callback to C (Darafei Praliaskouski)

#2116, [raster] Add ST_Value nearest-neighbor boundary options (Darafei Praliaskouski)

#2804, #4315, [raster] Support two-argument ST_MapAlgebra callbacks and pass callback call data as actual arguments (Darafei Praliaskouski)

#6062, [topology] Stop using recursive snapping, for improved robustness (Sandro Santilli)

#6065, Improved winding order computation robustness for rings having collapsed elements (Sandro Santilli)

#2137, [raster] Reuse matching source raster overviews in raster2pgsql -l (Darafei Praliaskouski)

#4749, Use point-in-polygon predicate fast paths for point-only GeometryCollections (Darafei Praliaskouski)

#2807, [raster] Preserve missing NODATA values in ST_MapAlgebra outputs (Darafei Praliaskouski)

#2832, [raster] Avoid padding single-tile raster2pgsql overviews when padding is not requested (Darafei Praliaskouski)

#2623, Generate a pgsql2shp GID field when exporting geometry-only shapefiles (Darafei Praliaskouski)

#2850, Preserve pgsql2shp numeric precision and scale in DBF metadata (Darafei Praliaskouski)

Build PostgreSQL extension modules with `-fno-semantic-interposition` when available so LTO can optimize same-DSO calls to exported PostGIS entry points directly instead of routing through the module PLT; in a local PostgreSQL 18 `pgbench` microbenchmark over 100k four-point groups this reduced `ST_Collect` aggregate latency by about 4% (Darafei Praliaskouski)

#5973, Install PostGIS utilities under `pg_config --bindir` by default and document `pg_config`-derived install paths more accurately (Darafei Praliaskouski)

#2808, Clarify postgis_full_version() output when installed component scripts are newer than core scripts (Darafei Praliaskouski)

#2838, Emit valid X3D Coordinate nodes for 2D polygon output (Darafei Praliaskouski)

GH-899, [raster] Honor PostgreSQL interrupts in long-running GDAL progress callbacks (Darafei Praliaskouski)

#6073, #6074, Stabilize geography LRS endpoint interpolation and pole centroid regressions on 32-bit systems (Darafei Praliaskouski)

#2583, Preserve Z/ZM dimensional qualifiers in ST_AsEWKT output for TIN (Darafei Praliaskouski)

#4616, Detect pgsql2shp query keywords case-insensitively (Darafei Praliaskouski)

#1986, ST_GeomFromGML support for GML ArcString and curved polygon rings (Darafei Praliaskouski)

#4749, Use point-in-polygon predicate fast paths for point-only GeometryCollections, including ST_Disjoint (Darafei Praliaskouski)

#3676, Mark geometry and geography btree comparison support functions as leakproof (Darafei Praliaskouski)

GH-875, Allow ST_OrderingEquals joins to use hash and merge join plans (Darafei Praliaskouski)

#1168, [raster] Support curved geometry inputs in ST_AsRaster (Darafei Praliaskouski)

#1416, ST_AsX3D outputs NURBSCURVE as X3D NurbsCurve and linearizes other curve geometries (Darafei Praliaskouski)

#3705, ST_AsX3D outputs valid coordinate nodes for 2D multi-geometries and GeometryCollections (Darafei Praliaskouski)

GT-484, Use column-level SELECT privileges for metadata views (Arthur Bazin)

#1408, #2386, #2557, #2805, #2898, #3743, #4323, #4332, #4385, #5109, #5532, #5593, #5638, #5645, #5950, #6022, #6082, GH-1161, GT-584, Numerous doc enhancements (Darafei Praliaskouski)

A.1.4 Bug Fixes

GH-1175, Guard missing operands before reading node tags in spatial selectivity estimation (Maksim Korotkov)

#6118, Reject empty geometry type modifiers without an out-of-bounds read (Egor Ivkov)

Fix WKB and TWKB parser resource exhaustion on malformed input (Darafei Praliaskouski)

#6038, Avoid stale relation lookups in geometry_columns after topology objects are dropped (Darafei Praliaskouski)

#5487, Improve error detail for repeated upgrades after an incomplete deprecated-function cleanup (Darafei Praliaskouski)

#5975, Initialize skipped union-find cluster ids deterministically (Darafei Praliaskouski)

#5904, [topology] Avoid MinGW format warnings when logging topology ids (Darafei Praliaskouski)

GH-888, [sfcgal] Avoid stale detoasted geometry access in CG_Visibility empty-input handling for SFCGAL < 2.2 (Darafei Praliaskouski)

#3179, Mark truncated libpgcommon PostgreSQL messages instead of silently dropping the tail (Darafei Praliaskouski)

#6003, Avoid size_t formats in liblwgeom logging/error wrappers for MinGW builds (Darafei Praliaskouski)

GH-886, ST_ClusterRelateWin validates NULL relate matrices before detoasting them (Darafei Praliaskouski)

GH-889, ST_Relate pattern inversion avoids out-of-bounds access on short DE-9IM patterns (Darafei Praliaskouski)

GH-890, Avoid GIDX BRIN summary merges across mismatched dimensions (Darafei Praliaskouski)

#5959, Prevent histogram target overflow when analysing massive tables (Darafei Praliaskouski)

#6080, Avoid out-of-bounds hex WKB lookup for high-bit input bytes (Darafei Praliaskouski)

#4828, geometry_columns handles NOT VALID SRID checks without errors (Darafei Praliaskouski)

#6048, [raster] ST_Clip no longer crashes when clipping sparse band selections (Darafei Praliaskouski)

GH-894, [raster] Fix Float16 import, mixed nodata warping, band replacement cleanup, GDAL credential redaction, and Python GDAL compatibility (Darafei Praliaskouski)